BTS Aviation Dataset - Big Data Stream Processing

We use stream processing in Apache Spark to process and extract relevant information from an aviation dataset provided by the Bureau of Transportation Statistics. The dataset contains records of flights in the United States from 1988 to 2008, including flight origin and destination and departure and arrival delays. There are 116,754,606 observations in the aviation selection.

Data Extraction and Cleaning

The dataset is stored in an Elastic Block Store (EBS) snapshot (snap-e1608d88) on AWS in the us-east-1 availability zone. We extract the data by creating an EBS volume based on the snapshot. Using an EC2 instance and the AWS Command Line Interface we copy the snapshot to a S3 bucket. We then download the "airline_ontime" folder, which contains the relevant dataset, using the AWS CLI on the local machine.

The dataset consists of 240 tables, each representing a month from 1988 to 2008. We observe that 2002 is missing the data from May to December. We also observe that the files for October 2008 to December 2008 are corrupted, which are excluded.

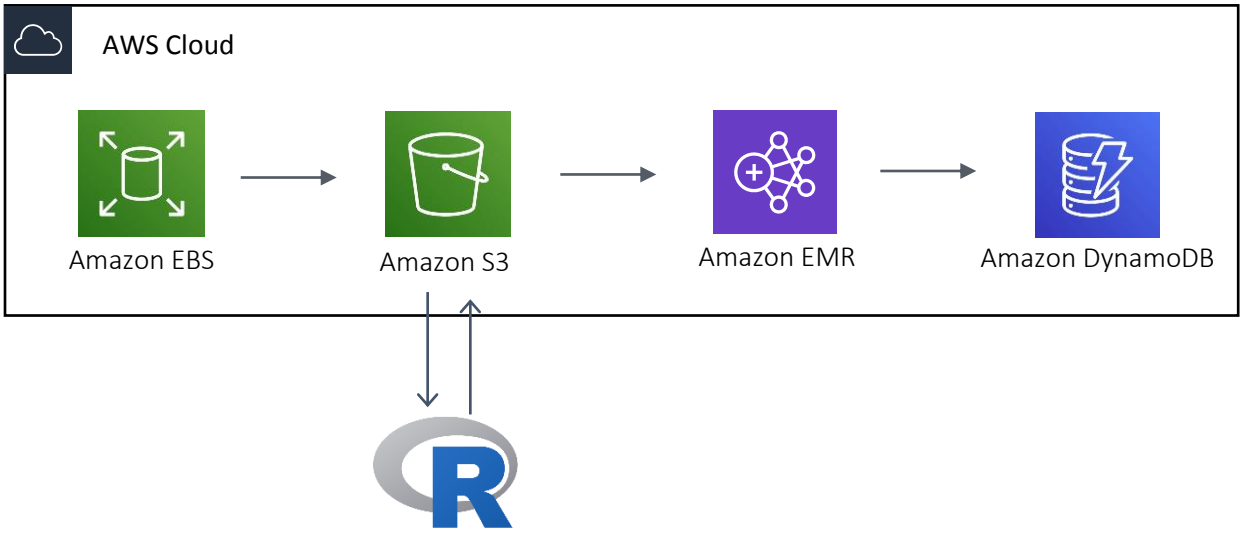
We load the tables in R and clean the dataset by replacing blanks with "NULL" and extracting only 7 columns from the tables. The extracted columns are:

```
"DayOfWeek","FlightDate","UniqueCarrier","Origin","Dest","DepDelay","ArrDelay"
```

The R script (included) loads each table, performs the two operations and writes a new table.

System Integration

The new tables are uploaded to S3, which is used as storage as it is well integrated within AWS. The EMR cluster uses a m5.xlarge instance type, which has 4 vCPUs, 16 GB of memory and up to 10 Gbps of dedicated bandwidth. The cluster uses 1 master and 2 core nodes. Finally, the cluster is accessed through a Jupyter notebook and will process the data queries using PySpark. Selected queries will be stored in AWS DynamoDB, a NoSQL database. The data pipeline architecture can be seen below:



Approaches and Algorithms

We load the data from S3 into a Spark DataFrame and create a new temporary view using a SparkDataFrame in the Spark Session:

```
df = spark.read.option("header", "true").csv("s3n://aviation-
database2/aviation-data/data/")
df.createOrReplaceTempView("aviation")
```

This will integrate using PySpark SQL, to query the dataset and store the results of some of our queries in DynamoDB. Tables are created beforehand in DynamoDB in preparation for the storage of our selected queries. We then create data frames in Spark for our queries and map them to the DynamoDB tables. Finally, DynamoDB tables can also be queried using PySpark SQL, streamlining our data frame creations and relevant queries. We also install and import pandas and matplotlib to assist in visualizing the data distribution.

Results

Using Spark SQL, we extract relevant information about the characteristics of commercial aviation in the United States. We display below the results, indicating performance in seconds, and produce the queries used in the appendix.

Top 10 most popular airports by numbers of flights to/from the airport

origin	flights
ORD	12449354
ATL	11540422
DFW	10799303
LAX	7723596
PHX	6585534
DEN	6273787
DTW	5636622
IAH	5480734
MSP	5199213
SFO	5171023

Days of the week by on-time arrival performance

DayOfWeek	AverageDelay
6	4.3
2	5.99
7	6.61
1	6.72
3	7.2
4	9.09
5	9.72

For each airport X: top-10 carriers in decreasing order of on-time departure performance from X

origin	uniquecarrier	avgdelay	airline_rank
CMH	DH	3.49	1
CMH	AA	3.51	2
CMH	NW	4.04	3
CMH	ML (1)	4.37	4
CMH	DL	4.71	5
CMH	PI	5.2	6
CMH	EA	5.94	7
CMH	US	5.99	8
CMH	TW	6.16	9
CMH	YV	7.96	10
BOS	TZ	3.06	1
BOS	PA (1)	4.45	2
BOS	ML (1)	5.73	3
BOS	EV	7.21	4
BOS	NW	7.25	5
BOS	DL	7.45	6
BOS	XE	8.1	7
BOS	US	8.69	8
BOS	AA	8.73	9
BOS	EA	8.89	10
JFK	UA	5.97	1
JFK	XE	8.11	2
JFK	CO	8.2	3
JFK	DH	8.74	4
JFK	AA	10.08	5
JFK	B6	11.13	6
JFK	PA (1)	11.52	7
JFK	NW	11.64	8
JFK	DL	11.99	9
JFK	TW	12.64	10
SRQ	TZ	-0.38	1
SRQ	XE	1.49	2
SRQ	YV	3.4	3
SRQ	AA	3.63	4
SRQ	UA	3.95	5
SRQ	US	3.97	6
SRQ	TW	4.3	7
SRQ	NW	4.86	8
SRQ	DL	4.87	9
SRQ	MQ	5.35	10
SEA	OO	2.71	1
SEA	PS	4.72	2
SEA	YV	5.12	3
SEA	TZ	6.35	4
SEA	US	6.41	5
SEA	NW	6.5	6
SEA	DL	6.54	7
SEA	HA	6.86	8
SEA	AA	6.94	9
SEA	CO	7.1	10

For each source-destination pair X-Y: top-10 carriers in decreasing order of on-time arrival performance at Y from X

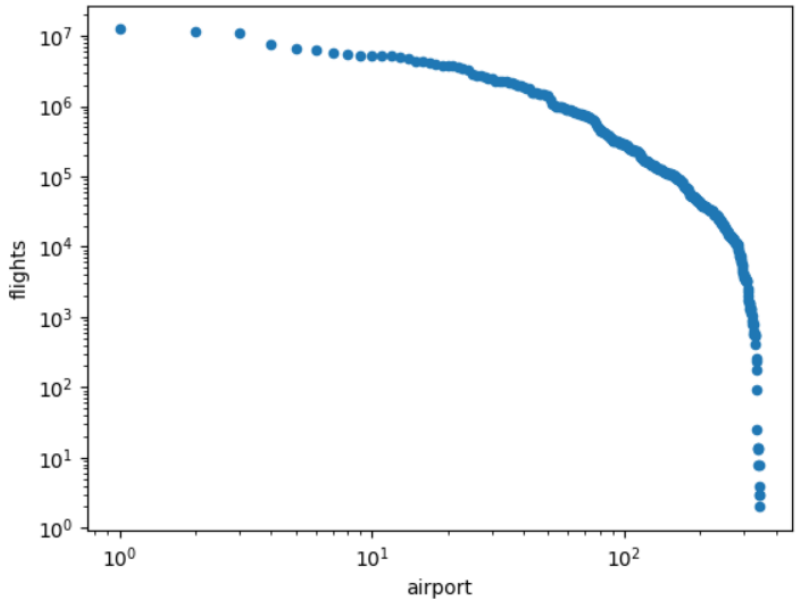
route	uniquecarrier	avgdelay	airline_rank
OKC-DFW	TW	0.1	1
OKC-DFW	EV	1.36	2
OKC-DFW	AA	4.57	3
OKC-DFW	MQ	4.68	4
OKC-DFW	DL	6.73	5
OKC-DFW	OO	12.84	6
OKC-DFW	OH	47.5	7
BOS-LGA	TW	-11.0	1
BOS-LGA	US	1.1	2
BOS-LGA	DL	2.02	3
BOS-LGA	PA (1)	6.07	4
BOS-LGA	EA	9.48	5
BOS-LGA	MQ	12.64	6
BOS-LGA	NW	15.23	7
BOS-LGA	AA	28.0	8
BOS-LGA	OH	30.45	9
BOS-LGA	TZ	133.0	10
MSP-ATL	9E	null	1
MSP-ATL	EA	4.2	2
MSP-ATL	OO	4.77	3
MSP-ATL	FL	6.29	4
MSP-ATL	DL	6.34	5
MSP-ATL	NW	7.02	6
MSP-ATL	OH	8.3	7
MSP-ATL	EV	10.12	8
LGA-BOS	TW	-3.0	1
LGA-BOS	US	-2.9	2
LGA-BOS	PA (1)	-0.42	3
LGA-BOS	DL	1.75	4
LGA-BOS	EA	4.82	5
LGA-BOS	MQ	9.87	6
LGA-BOS	NW	14.44	7
LGA-BOS	OH	27.98	8
LGA-BOS	AA	28.5	9

For each source-destination pair X-Y: mean arrival delay (in minutes) for a flight from X to Y

route	avgdelay
OKC-DFW	4.97
BOS-LGA	3.78
MSP-ATL	6.74
LGA-BOS	1.48

Flight Distribution

Plotting the airports' "popularity" on a log-log axis table, we do not see a Zipf distribution as hypothesized. A Zipf distribution would show a decreasing straight line on a log-log plot, which is not demonstrated here. In fact, we see a log-normal distribution, which fits in well with our analysis, where a few airports are extremely popular with a subsequent exponential decrease in popularity.



Appendix

PySpark Queries:

Group 1

Rank the top 10 most popular airports by numbers of flights to/from the airport.

```
spark.sql("SELECT origin, popular1+popular2 AS flights FROM (SELECT
origin, count(*) AS popular1 FROM aviation GROUP BY origin) AS ori
INNER JOIN (SELECT dest, count(*) AS popular2 FROM aviation GROUP BY
dest) AS des ON ori.origin = des.dest ORDER BY flights DESC Limit
10").show()
```

Rank the days of the week by on-time arrival performance.

```
spark.sql("SELECT DayOfWeek, round(AVG(ArrDelay),2) AS AverageDelay
FROM aviation WHERE DayOfWeek IS NOT NULL GROUP BY DayOfWeek  ORDER BY
AverageDelay").show()
```

Group 2

For each airport X, rank the top-10 carriers in decreasing order of on-time departure performance from X.

```
carriers_departures = spark.sql("SELECT * FROM (SELECT origin,
uniquecarrier, avgdelay, rank() over(PARTITION BY origin ORDER BY
avgdelay, origin) AS airline_rank FROM (SELECT origin, uniquecarrier,
round(AVG(depdelay),2) AS avgdelay FROM aviation GROUP BY origin,
uniquecarrier) AS no_rank) AS rank WHERE rank.airline_rank <= 10 AND
(origin LIKE ('%SRQ%') OR origin LIKE ('%CMH%') OR origin LIKE
('%JFK%') OR origin LIKE ('%SEA%') OR origin LIKE ('%BOS%'))")

carriers_departures.show(carriers_departures.count())
```

For each source-destination pair X-Y, rank the top-10 carriers in decreasing order of on-time arrival performance at Y from X.

```
carriers_arrival_all = spark.sql("SELECT rank.origin || '-' ||
rank.dest AS route, rank.uniquecarrier,rank.avgdelay,rank.airline_rank
FROM (SELECT origin, dest, uniquecarrier, avgdelay, rank()
over(PARTITION BY origin, dest ORDER BY avgdelay, uniquecarrier) AS
airline_rank FROM (SELECT origin, dest, uniquecarrier,
round(AVG(arrdelay),2) AS avgdelay FROM aviation WHERE arrdelay IS NOT
NULL GROUP BY origin, dest, uniquecarrier) AS no_rank) AS rank WHERE
rank.airline_rank <= 10")

carriers_arrival_all[carriers_arrival_all.route.isin("LGA-BOS","BOS-
LGA","OKC-DFW","MSP-ATL")].show(carriers_arrival_all.count())
```

For each source-destination pair X-Y, determine the mean arrival delay (in minutes) for a flight from X to Y.

```
mean_arrival_all = spark.sql("SELECT origin || '-' || dest AS route,
round(AVG(ArrDelay),2) AS avgdelay FROM aviation WHERE arrdelay IS NOT
NULL GROUP BY Origin, Dest")

mean_arrival_all[mean_arrival_all.route.isin("LGA-BOS","BOS-LGA","OKC-
DFW","MSP-ATL")].collect()
```

Group 3

Does the popularity distribution of airports follow a Zipf distribution? If not, what distribution does it follow?

```
dist = spark.sql("SELECT origin, popular1+popular2 AS flights FROM
(SELECT origin, count(*) AS popular1 FROM aviation GROUP BY origin) AS
ori INNER JOIN (SELECT dest, count(*) AS popular2 FROM aviation GROUP
BY dest) AS des ON ori.origin = des.dest ORDER BY flights DESC")

dist_pd = dist.toPandas()
dist_pd["airport"] = range(1,len(dist_pd)+1)

fig, ax = plt.subplots(figsize=(8, 4))
dist_pd.plot(x = 'airport', y = 'flights', loglog=True)
%matplot plt
```