Cloud Computing Capstone: Task 1 Report by Arpan Agrawal

Data extraction and cleaning

I performed the following steps in order to extract and clean the data:

- By taking a look at the queries and the dataset profile, it was apparent that the only relevant table is *airline_ontime*
- Fetched the table data from the snapshot and unzipped it to get the csv files
- I intended to use hive to perform the queries. CSV files cannot be very easily loaded into hive tables (all the fields are loaded as string). So the next step was to convert CSV files to JSON
- This was done by writing a python script. The script also type casted the fields into int, float, bool wherever required (default type is string in csv)
- Instead of dropping rows with any missing values, a placeholder value was inserted which will indicate that the value is to be ignored. For instance, in place of missing numbers -1000 was inserted.
- Only the following columns were taken in the json:
    - Year, Month, Quarter, DayOfWeek, DayofMonth, FlightDate, UniqueCarrier, FlightNum, Origin, Dest, CRSDepTime, DepTime, DepDelay, DepDelayMinutes, CRSArrTime, ArrTime, ArrDelay, ArrDelayMinutes, Cancelled, Diverted.

Systems

The following systems were used to complete the task:

- S3 was used to store the cleaned data
- DynamoDB was used to store the result of part 2 and 3.2
- EMR-Hive on Tez was used to run the queries

System integration:

- Clean data was loaded into S3 from the UI
- Hive-S3
    - EMR-Hive supports creating tables on the top of data stored in S3
    - org.apache.hive.hcatalog.data.JsonSerDe serde was used to serialize/deserialize the json data
- Hive-DynamoDB
    - Table in dynamodb were created from the dynamodb UI
    - EMR-Hive supports writing data to existing DynamoDB tables
    - This was done by using org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler as storage handler

Algorithms and approaches

1.2 Calculating top 10 airlines by on-time arrival performance: The approach was to group the data by UniqueCarrier and calculate the average value of ArrDelay for each group. The result was sorted in ascending order of avg arrival delay and the top 10 rows were reported. The rows with missing ArrDelay value were ignored (by checking equality with placeholder value).

1.3 Ranking the days of the week by on -time arrival performance: The approach was very similar to the previous one. The data was grouped by `DayOfWeek` and the average value of `ArrDelay` was computed for each group. The result was sorted in ascending order of avg arrival delay and the `DayOfWeek` (integer) was converted into actually week day names using `CASE` clause. Rows with missing ArrDelay, `DayOfWeek` were ignored.

2.1 For each airport compute the top 10 carriers in decreasing order of on-time departure: Firstly, the data is grouped by (Origin, UniqueCarrier) and average value of DepDelay is computed for each group. Then, for every Origin, top 10 UniqueCarrier on the basis of ascending average DepDelay is stored into DynamoDB table. This was done using rank() clause partitioned over Origin and ordered by average DepDelay and selecting the rows with rank less than or equal to 10. Rows with missing DepDelay were ignored.

2.2 For each airport Compute the top 10 destinations in decreasing order of on-time departure: Firstly, the data is grouped by (Origin, Dest) and average value of DepDelay is computed for each group. Then, for every Origin, top 10 Dest on the basis of ascending average DepDelay is stored into DynamoDB table. This was done by using rank() clause partitioned over Origin and ordered by avg DepDelay and selecting the rows with rank less than or equal to 10. Rows with missing DepDelay were ignored.

2.4 For each origin destination pair, compute mean arrival delay: The data was grouped by Origin, Dest and average ArrDelay was computed for each group. Rows with missing ArrDelay were ignored. The output was stored in dynamodb.

3.1 Total frequency (to and fro) of flights from each airport was calculated and ranked in descending order. This gave the frequency count as well as the rank needed to plot the curve.

3.2 The approach for this problem was to inner join the table with itself. In the join, the left table referred to the first leg of the journey and the right to the second leg. The join condition was left table's destination = right table's origin. All the condition of departure time, flight date etc were put in. Then the options corresponding to each route (X-Y-Z) and trip start date, were ranked in ascending order of total arrival delay and the one with minimum total delay was stored in dynamodb. When multiple such options were available, the one occurring first was chosen.

Results

1.2 The answer is in format: <airline> <avg arrival delay>

HA	-1.01
AQ	1.16
PS	1.45

ML (1)	4.75
PA (1)	5.32
F9	5.47
NW	5.56
WN	5.56
OO	5.74
9E	5.87

1.3 The answer is in format: <weekday> <avg arrival delay>

Saturday	4.3
Tuesday	5.99
Sunday	6.61
Monday	6.72
Wednesday	7.2
Thursday	9.09
Friday	9.72

2.1 The answers are in format: <airport> <carrier> <avg departure delay>

CMI:			BWI:			MIA:		
CMI	OH	0.61	BWI	F9	0.76	MIA	9E	-3.0
CMI	US	2.03	BWI	PA (1)	4.76	MIA	EV	1.2
CMI	TW	4.12	BWI	CO	5.18	MIA	TZ	1.78
CMI	PI	4.46	BWI	YV	5.5	MIA	XE	1.87
CMI	DH	6.03	BWI	NW	5.71	MIA	PA (1)	4.2
CMI	EV	6.67	BWI	AA	6.0	MIA	NW	4.5
CMI	MQ	8.02	BWI	9E	7.24	MIA	US	6.09
			BWI	US	7.49	MIA	UA	6.87
			BWI	DL	7.68	MIA	ML (1)	7.5
			BWI	UA	7.74	MIA	FL	8.57
LAX:			IAH:			SFO:		
LAX	MQ	2.41	IAH	NW	3.56	SFO	TZ	3.95
LAX	OO	4.22	IAH	PA (1)	3.98	SFO	MQ	4.85
LAX	FL	4.73	IAH	PI	3.99	SFO	F9	5.16
LAX	TZ	4.76	IAH	US	5.06	SFO	PA (1)	5.29
LAX	PS	4.86	IAH	F9	5.55	SFO	NW	5.76
LAX	NW	5.12	IAH	AA	5.7	SFO	PS	6.3
LAX	F9	5.73	IAH	TW	6.05	SFO	DL	6.56
LAX	HA	5.81	IAH	WN	6.23	SFO	CO	7.08
LAX	YV	6.02	IAH	OO	6.59	SFO	US	7.53
LAX	US	6.75	IAH	MQ	6.71	SFO	TW	7.79

2.2 The answers are in format: <origin airport> <dest airport> <avg departure delay>

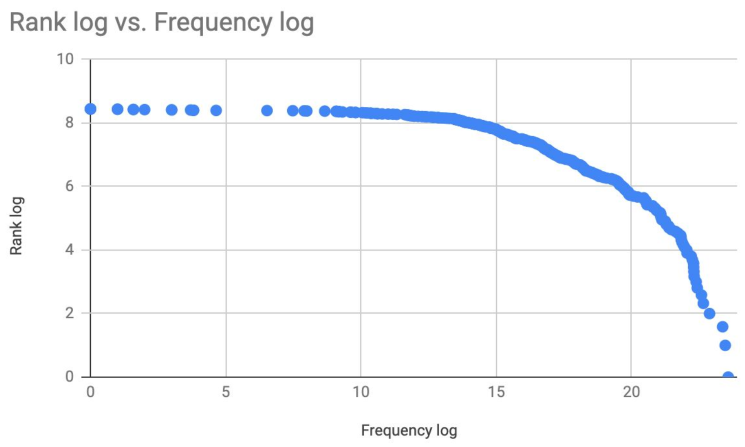
CMI: CMI ABI -7.0 CMI PIT 1.1 CMI CVG 1.89 CMI DAY 3.12 CMI STL 3.98 CMI PIA 4.59 CMI DFW 5.94 CMI ATL 6.67 CMI ORD 8.19	BWI: BWI SAV -7.0 BWI MLB 1.16 BWI DAB 1.47 BWI SRQ 1.59 BWI IAD 1.79 BWI UCA 3.65 BWI CHO 3.74 BWI GSP 4.2 BWI SJU 4.44 BWI OAJ 4.47	MIA: MIA SHV 0.0 MIA BUF 1.0 MIA SAN 1.71 MIA SLC 2.54 MIA HOU 2.91 MIA ISP 3.65 MIA MEM 3.75 MIA PSE 3.98 MIA TLH 4.26 MIA MCI 4.61
LAX: LAX SDF -16.0 LAX IDA -7.0 LAX DRO -6.0 LAX RSW -3.0 LAX LAX -2.0 LAX BZN -0.73 LAX MAF 0.0 LAX PIH 0.0 LAX IYK 1.27 LAX MFE 1.38	IAH: IAH MSN -2.0 IAH AGS -0.62 IAH MLI -0.5 IAH EFD 1.89 IAH HOU 2.17 IAH JAC 2.57 IAH MTJ 2.95 IAH RNO 3.22 IAH BPT 3.6 IAH VCT 3.61	SFO: SFO SDF -10.0 SFO MSO -4.0 SFO PIH -3.0 SFO LGA -1.76 SFO PIE -1.34 SFO OAK -0.81 SFO FAR 0.0 SFO BNA 2.43 SFO MEM 3.3 SFO SCK 4.0

2.4 The answer is in format: <origin> <destination> <avg arrival delay>

CMI	ORD	10.14
IND	CMH	2.9
DFW	IAH	7.65
LAX	SFO	9.59
JFK	LAX	6.64
ATL	PHX	9.02

3.1 Below shows the plot of log of frequency (x axis) and log of popularity rank (y axis). If it was a zipf distribution, then the curve would have followed frequency*popularity rank = constant curve which is same as y+x = constant for log graph. Clearly, this is not the case.

The plot is closer to poisson distribution.



3.2 The answer is in format:
<route> <trip start date> <total arrival delay> <first leg flight details> <first leg arrival delay> <second leg flight details> <second leg arrival delay>

- CMI-ORD-LAX 2008-03-04 -38.0 MQ 4278 710 2008-03-04 -14.0 AA 607 1950 2008-03-06 -24.0
- JAX-DFW-CRP 2008-09-09 -6.0 AA 845 725 2008-09-09 1.0 MQ 3627 1645 2008-09-11 -7.0
- SLC-BFL-LAX 2008-04-01 18.0 OO 3755 1100 2008-04-01 12.0 OO 5429 1455 2008-04-03 6.0
- LAX-SFO-PHX 2008-07-12 -32.0 WN 3534 650 2008-07-12 -13.0 US 412 1925 2008-07-14 -19.0
- DFW-ORD-DFW 2008-06-10 -31.0 UA 1104 700 2008-06-10 -21.0 AA 2341 1645 2008-06-12 -10.0
- LAX-ORD-JFK 2008-01-01 -6.0 UA 944 705 2008-01-01 1.0 B6 918 1900 2008-01-03 -7.0

Optimizations

- Unnecessary fields are dropped in data cleaning step
- Vectorization in Hive was to speed up joins, aggregates, scans and filters by executing them in batch of 1024
- Q 3.2 required join to take place and the data was skewed. To optimize it, hive.optimize.skewjoin was enabled
- Numbers of reducers were set to 300 to parallelize the processing

Usefulness of results

- All the queries were very relevant in the real world. The technique employed in Q2 and Q3.2 of computing the result offline and storing it in a key value store so that it can be queried quickly query was good and will prove to be useful. Also, quite a few optimization lessons were learnt by optimizing Q 3.2