CS598 Capstone Project Task 1 – rcook4

Extracted and cleaned the data

I reviewed the three question groups and found that all results could be accurately and comprehensively obtained using the Airline On-Time Performance Data (https://www.transtats.bts.gov/DatabaseInfo.asp?DB_ID=120).

The data was part of the ESB Snapshot as zip files under read-only folders, one file per year-month. The data extraction was performed as four batches for cost and performance optimization. Each batch ran on a single free-tier EC2 instance. The process used bash commands to mount the snapshot then unzip the year-month files into a folder named *raw*. It was discovered that two files were unusable (HTML not CSV) and I confirmed on Piazza (@19) that neither those files nor the data within them were required for this project.

To ensure accuracy, I performed spot checks of the unzipped files inspecting the schema for consistency. There existed two fields (*OriginCityName*, *DestCityName*) requiring quoted identifiers. There also existed two different record schemas. The record schema before 1991-Oct differed from the later record schema by having twenty additional NULLable fields but as such they were still compatible schemas.

Data cleaning was limited to the elimination of quoted identifiers which was done as a performance optimization. The process was performed in parallel across four EC2 instances. It used bash commands of sed to first find and replace, comma space, with hyphen, then to find and replace, double quotes, with nothing. Also for performance optimization, the cleaned files were GZIP compressed and uploaded in parallel to a single S3 bucket under a folder named *scrubbed*.

In Athena, I created external table *scrubbed* over the folder using *org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe*. Had the quoted identifiers not been removed I would have had to use a slower deserializer (*OpenCSVSerDe*).

I decided to use a single base table on which all queries would be performed against. A single data source ensured the results would be consistent and intuitive. To ensure accuracy, I removed data for any cancelled or diverted flights.

> What is the departure delay of a canceled flight?
>
> What is the arrival delay of a diverted flight?
>
> Will diverted/cancelled flights be counted as flights to/from an airport?

The *completedflights* table was derived from *scrubbed*. It was partitioned by year as a performance optimization. It was trimmed to only eleven columns also as a performance optimization. The actual and Computer Reservation System (CRS) arrival/departure columns were converted to timestamps to make the SQL more intuitive.

The *missions* table was derived from *completedflights*. It was partitioned by day of the month as a performance optimization. It served as a set of all valid combinations, X to Y to Z airports occurring across a two-day window in 2018.

Integrated each system

Project code can be publicly viewed here: https://github.com/rcook4/cs598project

The pipeline was comprised of the following systems in order:

> **EBS Snapshot => EC2 (bash) => S3 => Athena (SQL) => S3 => EC2 (bash) => DMS (*only g3q2*) => DynamoDB**

I found there to be some ambiguity around which technologies could be used so I confirmed in Piazza (@26) that my pipeline met the project requirements.

ESB Snapshot files were transformed on the **EC2 instances** and placed into **S3** folder scrubbed in less than 20 minutes. Then **S3** folder scrubbed was used as the data backing **Athena** table *scrubbed*. Create Table As Select (CTAS) commands were used to generate new data GZIP CSV files back into **S3** for both derived folders and answer folders. The derived tables (*completedflights*, *missions*) combined took less than 5 minutes. The *g3q2* (answer) table took less than 3 minutes and all other answer tables took less than 10 seconds. Excluding *g3q2*, the **S3** answer folders were downloaded by the **EC2 instances** then transform it into JSON and finally batch loaded into **DynamoDB** answer tables within 10 minutes.

Originally, I believed DynamoDB *g3q2* table would be just the six results so the EC2 instances initially included it. After learning *g3q2* would be over fifty million records, I decided to change my approach for that only that table.

For *g3q2*, the EC2 instances downloaded the GZIP CSV files from S3 to GUNZIP and uploaded them back into S3 under separate dayofthemonth folders as a performance optimization. Database Migration Services (DMS) allowed for easy scaling, scheduling, monitoring and logging which would be important for this large load process. The load process used a DynamoDB destination endpoint and for each dayofthemonth, a S3 source endpoint and a migration. The first batch parallelism included five tasks on a r4.8xlarge instance and one task on each of five t2.medium instances. All ten tasks loaded about 2 million records in 4 hours. The second batch included one task on each of ten t2.medium instances. Again, all tasks loaded about 2 million records in 4 hours. The third and final batch included eleven tasks on a r4.8xlarge instance. Tasks 21st to 30th loaded about 2 million records in 4 hours but task 31st loaded half the records in half the time.

After uncovering that performance scaled linear with record size, my next step was to cut the data in 100 instead of 31. Before that I received a bill from AWS for $100. It was discovered that the UIUC AWS credits did ***not*** cover use of DMS. I reported this on Piazza (@27) for guidance and was effectively told to explain what tuning optimizations I would have made because no DMS credits would be made available to me.

I predict that using parallelism of 100 instead of 31 would have cut the load time for each piece down to a third (1 hour, 20 minutes). There was no sign of parallelism degrading speed so running all 100 tasks at the same time was my plan.

Results of each question

The first three questions in group two state to compute the top ten answers ordered by “on-time performance”. The given example answers for those questions used **mean-minutes-of-delay metric** but it is only specifically required for the last question. I computed answers to last question and the results matched within 10% of the example answer. For that slight variation, I believe it was due to my choice to throw out delayed and cancelled flights, which I stated in the above section, *Extracted and cleaned the data*. I mention that matching results because I wanted to be clear that it was possible for me to match the listed results but I intentionally chose not to. Instead of mean delay I decided to use percentage of flight delayed longer than a specific threshold. My reasoning was that I consider impactful delays, such as one causing a missed connection, to more detrimental to “on-time performance” than just being a single data point within an average. I was unsure what the threshold should be until I noticed the dataset itself has an indicator column of whether or not the flight had more than a 15-minute delay. It seems the Bureau of Transportation Statistics created this column as a measure to hold flights accountable to, and 15 minutes sounded like a reasonable buffer. My answers to group two questions use this **failed-to-meet-expectations-percentage metric** for computing “on-time performance”.

Which carrier looks better to you?

```
origin uniquecarrier avg_mins neg_delay short_delay long_delay depdel15 pct_long

BWI     PA (1)        4.8      20        64          21         21       20

BWI     EA            8.6      48        5145        895        902      14.8
```

The SQL query algorithm used for group 1 questions entails:

1. Use a FROM subquery to AGGREGATE the metric, sort it then limit to ten
2. SELECT the computed RANK using the AGGREGATE

G1Q1 RANK	AIRPORT	TRIPS_TOTAL
1	ORD	11,501,147
2	ATL	10,901,013
3	DFW	10,086,241
4	LAX	7,298,022
5	PHX	6,248,811
6	DEN	5,878,489
7	DTW	5,258,478
8	IAH	5,218,212
9	MSP	4,854,545
10	SFO	4,838,743

G2Q2 RANK	AIRLINE	PERFORMANCE
1	HA	6.21
2	AQ	9.42
3	PS	10.96
4	ML (1)	15.41
5	WN	17.38
6	OO	17.76
7	EA	18.40
8	9E	19.17
9	NW	19.40
10	F9	19.57

The SQL query algorithm used for group 2 questions entailed:

1. Use a FROM subquery to AGGREGATE the “on-time performance” using the *completedflights* table
2. Use another FROM subquery to calculate RANK using the AGGREGATE
3. SELECT top ten results using the RANK

G2Q1 RANK	*CMI*	BWI	*MIA*	LAX	*IAH*	SFO
1	*US 3.86*	F9 7.21	*9E 0.00*	PS 8.31	*PI 8.16*	TZ 11.09
2	*TW 7.42*	CO 10.76	*EV 8.81*	HA 9.21	*NW 8.91*	PS 12.29
3	*OH 9.34*	AA 11.36	*PA (1) 9.85*	MQ 10.34	*PA (1) 10.45*	PA (1) 12.72
4	*PI 9.49*	NW 12.42	*XE 10.01*	TZ 11.32	*AA 10.55*	HA 12.84
5	*EV 13.76*	DL 14.37	*NW 10.93*	OO 11.84	*WN 11.64*	DL 12.91
6	*MQ 16.58*	US 14.61	*TZ 11.64*	NW 12.01	*US 12.30*	NW 13.50
7	*DH 16.63*	EA 14.80	*UA 12.69*	ML (1) 13.50	*TW 12.61*	DH 13.95
8		9E 14.84	*US 13.44*	CO 13.77	*DL 13.39*	AA 14.42
9		YV 15.01	*ML (1) 14.47*	AA 14.20	*OO 13.61*	CO 14.78
10		US 15.37	*PI 15.65*	FL 14.28	*XE 14.36*	MQ 15.86

G2Q2 RANK	*CMI*	BWI	*MIA*	LAX	*IAH*	SFO
1	*PIT 3.65*	MLB 3.40	*BUF 0.00*	PMD 0.00	*MLI 0.00*	FAR 0.00
2	*DAY 6.11*	IAD 4.23	*SAN 3.89*	LAX 0.00	*MSN 0.00*	SDF 0.00
3	*STL 7.27*	DAB 5.42	*HOU 5.69*	BZN 0.00	*HOU 4.68*	MSO 0.00
4	*PIA 7.92*	CHO 7.45	*ISP 6.43*	LGB 0.00	*AGS 4.97*	SCK 0.00
5	*DFW 10.49*	UCA 7.88	*SLC 9.13*	SDF 0.00	*EFD 7.44*	LGA 3.03
6	*CVG 11.06*	SRQ 7.89	*MEM 9.29*	VIS 3.90	*JAC 8.93*	PIE 4.05
7	*ATL 13.76*	SJU 8.36	*GNV 9.61*	MEM 6.98	*RNO 9.22*	BNA 5.36
8	*ORD 17.10*	OAJ 8.68	*TLH 9.64*	IYK 7.81	*MDW 9.30*	OAK 5.68
9		BGM 9.17	*EGE 11.29*	HDN 8.57	*VCT 10.01*	MKE 9.07
10		GSP 9.48	*TPA 12.17*	SNA 8.63	*CLL 10.32*	MEM 9.99

** there is only one flight from CMI to ABI which was diverted so it does not exist for any calculations*

G2Q3 RANK	CMI => ORD	IND => CMH	DFW => IAH	LAX => SFO	JFK => LAX	ATL => PHX
1	*MQ 22.50*	AA 0.00	*PA (1) 10.53*	TZ 9.52	*UA 21.14*	FL 23.37
2		CO 7.09	*CO 14.72*	F9 12.74	*AA 25.90*	US 24.63
3		HP 14.38	*OO 17.44*	PS 12.82	*HP 26.81*	HP 25.88
4		US 14.66	*UA 17.75*	EV 20.26	*DL 29.31*	EA 26.33
5		NW 17.23	*XE 18.04*	AA 21.71	*TW 33.09*	DL 29.08
6		DL 20.67	*EV 18.07*	MQ 22.18	*PA (1) 35.16*	
7		EA 22.90	*DL 18.87*	US 22.63		
8			*AA 20.40*	CO 22.96		
9			*MQ 25.86*	WN 23.93		
10				UA 24.70		

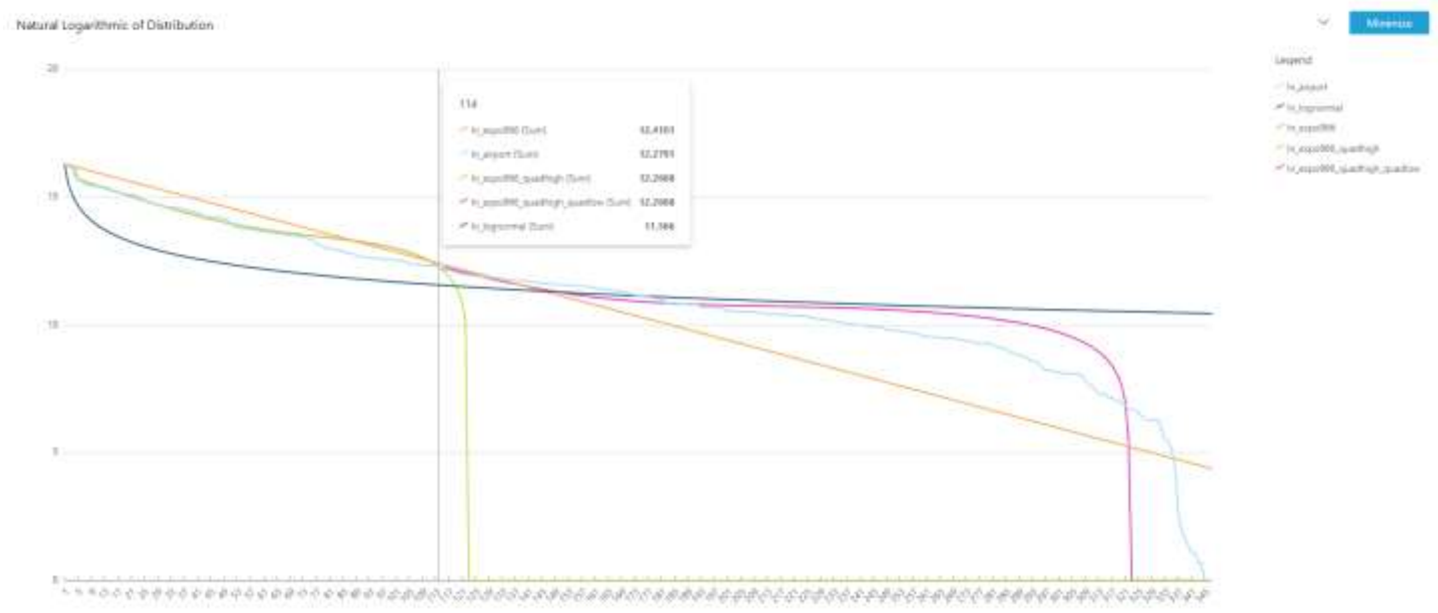
The SQL query algorithm used for group 3 question 2 entailed:

1. Build a *missions* persisted, partitioned table from all valid combinations using the *completedflights* table
2. Use a FROM subquery to FIND launch and land flights joining *completedflights* with the *missions* table
3. Use another FROM subquery to calculate RANK using the FIND
4. SELECT best mission option using the RANK

G3Q2 DELAY	MISSION	LAUNCH	LAUNCH_START	LAND	LAND_START
-38	CMI => ORD => LAX (04/03/2008)	MQ 4278	07:10 04/03/2008	AA 607	19:50 06/03/2008
-32	LAX => SFO => PHX (12/07/2008)	WN 3534	06:50 12/07/2008	US 412	19:25 14/07/2008
-31	DFW => ORD => DFW (10/06/2008)	UA 1104	07:00 10/06/2008	AA 2341	16:45 12/06/2008
-6	JAX => DFW => CRP (09/09/2008)	AA 845	07:25 09/09/2008	MQ 3627	16:45 11/09/2008
-6	LAX => ORD => JFK (01/01/2008)	UA 944	07:05 01/01/2008	B6 918	19:00 03/01/2008
18	SLC => BFL => LAX (01/04/2008)	OO 3755	11:00 01/04/2008	OO 5429	14:55 03/04/2008

In the GitHub project there exists a SQL file named by Group, Question, for each query.

The airport popularity looks similar to a Log Normal distribution until zooming at the tail. There it becomes obvious that it is actually a Zipf (discrete Pareto) distribution were each subsequent rank is approximately 96.6% of the previous rank.



This QuickSight graph uses a logarithmic scale so big and small values can be seen simultaneously. Actual values are displayed in light blue, log normal in dark blue, Zipf in orange, Zipf with a Quadratic high value offset in green, Zipf with a Quadratic high value offset and a Quadratic low value offset in pink (*Green and Pink only added as an academic interest*).

System-level or application-level optimizations

I performed system-level performance optimization by batching and parallelizing data processing across multiple instances. I also performed system-level cost optimizations. I used serverless query compute (Athena). EBS Snapshots are unreachable for serverless processing (Lambda) so I used free-tier EC instances (t2.micro), except for DMS on g3q2.

There were several application-level performance optimizations. I used data compression and data partitioning. I reformat the data to enable usage by a high performance deserializer (*OpenCSVSerDe*). I used multiple SQL tuning methods to include: ordinal column referencing, limited projections, subqueries and join optimization. I created a sort key for each DynamoDB table.

Opinion about the results

It was interesting to learn about these online publicly available transportation datasets. I am not a frequent flyer so I will have to assume that the results are sensible. The specific dataset used was more than a decade old the findings may no longer be current. On a personal level, I tend to be more price concerned than quality concerned so it is unlikely that the results will change my air travel behaviors.

Video Demonstration Link

https://mediaspace.illinois.edu/media/t/0_yexqi416

I used the tag "CS598CCC-SUMMER-2019" and the upload is marked private.