Task 1: Data Extraction, Batch Processing with Hadoop

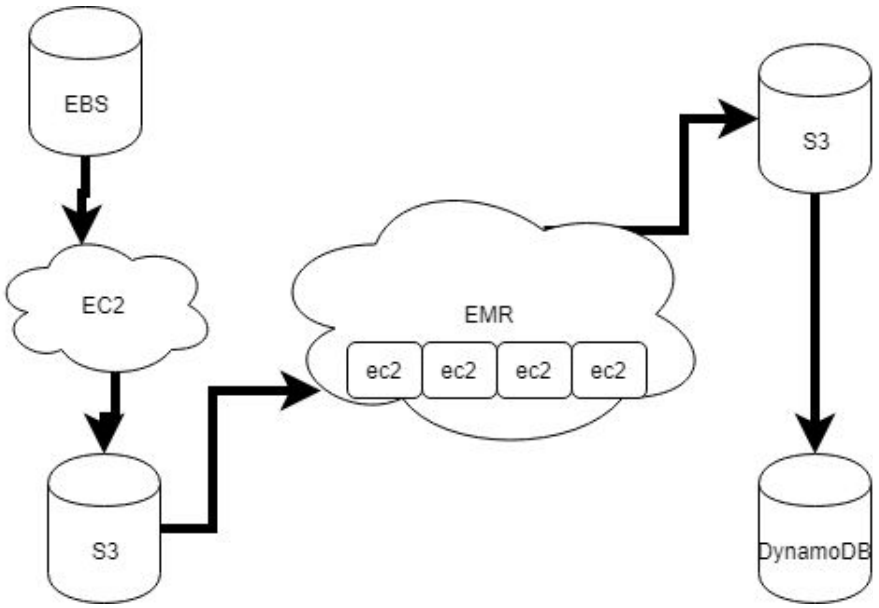
Data Extraction and Cleaning

1. Created new EBS volume from provided transportation dataset web link
    a. https://aws.amazon.com/datasets/transportation-databases/
2. Attached EBS volume to EC2 instance in order to access data
3. Reviewed data structure and determined the only folder I needed to answer the questions was “aviation/airline_ontime”
4. Unzip all csv files from yearly subfolders
    a. Year 2002 only has files for the first 4 months of the year
    b. Year 2008 zip files for months 11 and 12 are corrupted
5. Identify cleaning steps
    a. Information is separated by commas but some string entries contain commas
        i. Origin City Name and Destination City Name
    b. Only need a subset of 55 information columns in each line to answer the questions
        i. Year (0), Month (2), DayOfMonth (3), UniqueCarrier (6), FlightNum (10), Origin (11), Dest (17), CRSDepTime (23), DepDelay (25), ArrDelay (36)
    c. Remove cancelled and diverted flights
        i. Cancelled (41), Diverted (43)
6. Use python script to extract only needed rows and write new csv files
7. Upload cleaned csv files to amazon S3

Integrated System

Based on the already being tied to use AWS I decided to use its EMS (Elastic Map Reduce) service in order to minimize the amount of configuration and boiler plate work I needed to do, along with minimizing the cost of running instances. EMS will automatically handle setting up HDFS and MapReduce. It also allows me to easily input and output data with S3. Scaling is also easy to accomplish to further improve performance. Results are finally imported into DynamoDB.

Diagram of the system:



Questions and Results

Results are based on data that does not include cancelled or diverted flights. Negative delay times are factored in.

Format: Year (0), Month (1), DayOfMonth (2), UniqueCarrier (3), FlightNum (4), Origin (5), Dest (6), CRSDepTime (7), DepDelay (8), ArrDelay (9)

Full solutions with code can be found here:
https://github.com/zimmeren/CloudComputing/tree/master/CloudCapstone/ProjectPart1/cloudcapstone/src/main/java/zimmeren/cloudcapstone

NOTE: Some entries are different than answer sheet because of cleaning data to remove cancelled and diverted flights

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

Data Columns: Origin (5), Dest (6)

Solved using two rounds of map reduce. First round mapped Origin and Destination each with count of 1 then reduce summed these counts for each Airport name. Second round did an inverse map so reduce could sort all into the correct order. By limiting the reducer of the 2nd round to 1 I ensured all results were sorted in a single file

Rank	Airport	Number of In or Out Flights
1	ORD	12020931
2	ATL	11301229
3	DFW	10562404
4	LAX	7574328

5	PHX	6494512
6	DEN	6169795
7	DTW	5491596
8	IAH	5400340
9	MSP	5073589
10	SFO	5050872

1.2 Rank the top 10 airlines by on-time arrival performance.
Data Columns: UniqueCarrier (3), ArrDelay (9)
Solved using two rounds of map reduce. First round mapped Airline each with arrival delay then reduce averaged these counts for each Airline name. Second round did an inverse map so reduce could sort all into the correct order. By limiting the reducer of the 2nd round to 1 I ensured all results were sorted in a single file

Rank	**Airline**	**Average Arrival Delay in Minutes**
1	HA	-1.0118043
2	AQ	1.1569234
3	PS	1.4506385
4	ML (1)	4.747609
5	PA (1)	5.322431
6	F9	5.4658813
7	NW	5.557816
8	WN	5.582633
9	OO	5.7363024
10	9E	5.8671846

2.1 For each airport X, rank the top-10 carriers in decreasing order of on-time departure performance from X.
Data Columns: UniqueCarrier (3), Origin (5), DepDelay (8)
Solved using one round of map reduce. Mapped the combination of origin and carrier as a paired key and sum and count as value. Had a Combiner that took that and output the same. Then a Reducer that output Origin and Carrier as key and average departure delay as value.

Airport	Airlines in order of average departure delay (delay in minutes)
CMI	OH(0.612), US(1.824), TW(3.744), PI(4.369), DH(6.080), EV(6.665), MQ(7.992)
BWI	F9(0.746), PA1(4.761), CO(5.113), YV(5.504), NW(5.655), AA(5.913), 9E(7.22), US(7.461), DL(7.641), UA(7.671)
MIA	9E(-3.666), EV(1.202), TZ(1.782), XE(1.862), PA1(4.026), NW(4.423), US(6.018), UA(6.808), ML1(7.512), PI(7.950)
LAX	MQ(2.399), OO(4.214), FL(4.687), TZ(4.758), PS(4.805), NW(5.092), F9(5.721), HA(5.817), YV(6.036), US(6.729)
IAH	NW(3.515), PA1(3.901), PI(3.952), US(5.029), F9(5.549), AA(5.616), TW(6.027), WN(6.213), OO(6.583), MQ(6.709)
SFO	TZ(3.935), MQ(4.782), F9(5.155), PA1(5.284), NW(5.717), PS(6.256), DL(6.512), CO(7.047), US(7.479), TW(7.763)

2.2 For each airport X, rank the top-10 airports in decreasing order of on-time departure performance from X.
Data Columns: Origin (5), Dest (6), DepDelay (8)
Solved using one round of map reduce. Mapped the combination of origin and destination as a paired key and sum and count as value. Had a Combiner that took that and output the same. Then a Reducer that output Origin and Carrier as key and average departure delay as value.

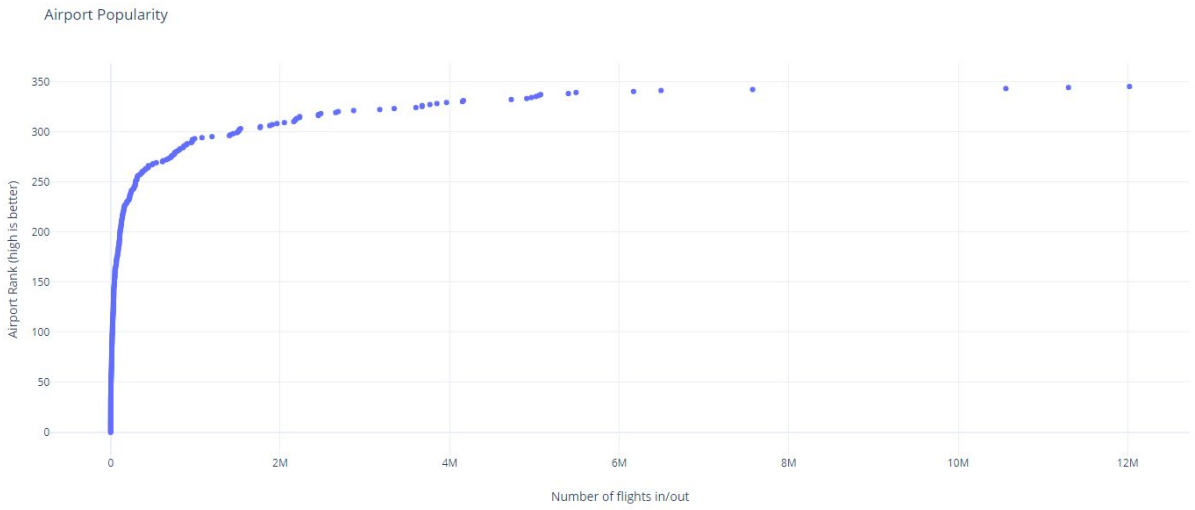
Airport	Destination airports in order of average departure delay (delay in minutes)
CMI	PIT(1.107), CVG(1.902), DAY(2.92), PIA(3.413), STL(3.8417), DFW(5.978), ATL(6.665), ORD(8.163)
BWI	MLB(1.161), DAB(1.508), SRQ(1.55), IAD(1.79), UCA(3.32), CHO(3.74), BGM(3.79), DCA(3.97), GSP(4.192), SJU(4.2)
MIA	SAN(1.71), BUF(2.0), SLC(2.41), HOU(2.888), ISP(3.649), MEM(3.686), GNV(3.96), PSE(3.97), TLH(4.234), MCI(4.612)
LAX	SDF(-16), LAX(-2), BZN(-0.727), MAF(0), MFE(1.77), IYK(1.26), MEM(1.867), PIE(1.926), SNA(2.09), SBA(2.27)
IAH	MSN(-2) ,MLI(-1), AGS(-0.61), EFD(1.887), HOU(2.172), JAC(2.422), MTJ(2.902), GUC(3.537), BPT(3.599), CLL(4.214)
SFO	SDF(-10), MSO(-4), LGA(-1.75), PIE(-1.34), OAK(-0.813), FAR(0), BNA(2.379), MEM(3.26), SJC(3.985), MKE(3.988)

2.4 For each source-destination pair X-Y, determine the mean arrival delay (in minutes) for a flight from X to Y.
Data Columns: Origin (5), Dest (6), ArrDelay (9)
Solved using one round of map reduce. Mapped the combination of origin and destination as a paired key and sum and count as value. Had a Combiner that took that and output the same. Then a Reducer that output Origin and Carrier as key and average arrival delay as value.

Origin	Destination	Average Arrival Delay in Minutes
CMI	ORD	10.143662
IND	CMH	2.8999038
DFW	IAH	7.6544423
LAX	SFO	9.589283

JFK	LAX	6.635119
ATL	PHX	9.021342

3.1 Does the popularity distribution of airports follow a Zipf distribution? If not, what distribution does it follow?
Data Columns: Origin (5), Dest (6)



Above is a plot of the airport ranking vs the number of flights. Most of the data points are accumulated around 0 and then only some of the most popular get to right. Because of the heavy accumulation of data and lack of spread this does not follow a ZIPF distribution but more likely a logarithmic normal distribution.

3.2 Airport X → Y → Z best performance

Requirements:

- The second leg of the journey (flight Y-Z) must depart two days after the first leg (flight X-Y). For example, if X-Y departs on January 5, 2008, Y-Z must depart on January 7, 2008.
- Tom wants his flights scheduled to depart airport X before 12:00 PM local time and to depart airport Y after 12:00 PM local time.
- Tom wants to arrive at each destination with as little delay as possible. You can assume you know the actual delay of each flight.

Data Columns: Year (0), Month (1), DayOfMonth (2), UniqueCarrier (3), FlightNum (4), Origin (5), Dest (6), CRSDepTime (7) ArrDelay (9)
Solved using one round of map reduce. Only used 2008 data. The mapper wrote two outputs per input line. They were both keyed on Airport and Date. Each output contained a special writable that had all the flight information. The first output was given a date of +2 days and marked as “searching”. This meant it was the first leg and was looking for second legs in the map phase. The second output was just its regular date and was not marked “searching”. In the

reducer phase all keys came together for a Y airport on a date. Created output that was a combination of every “searching” and not “searching” key pairs. This created an output key of X->Y->Z Date. Output value was the information for leg1 and leg2 flights.

Route	Date	Total Delay	Leg 1 Info	Leg 2 Info
CMI->ORD->LAX	4/3/2008	-38	MQ 4278 7:10	AA 607 19:50
JAX->DFW->CRP	9/9/2008	-6	AA 845 7:25	MQ 3627 16:45
SLC->BFL->LAX	1/4/2008	18	OO 3755 11:00	OO 5429 14:55
LAX->SFO->PHX	12/7/2008	-32	WN 3534 6:50	US 412 19:25
DFW->ORD->DFW	10/6/2008	-31	UA 1104 7:00	AA 2341 16:45
LAX->ORD->JFK	1/1/2008	-6	UA 944 7:05	B6 918 19:00

Optimizations

1. Removed unnecessary data during the cleaning phase in order to limit the amount of data needed to be transferred and each step, and read line by line during the compute steps
2. Increased cluster size in order to more quickly process data and compute results
3. Use MapReduce Combiners whenever possible so I can split some reduce work across nodes
4. Avoid the need for merge multiple output files by setting sort jobs to have a single reducer task that way all data is output in a single file. This works well when the data being output is a few thousand lines or less. Don’t do this in large data cases
5. For question 3.2 only use data from 2008
6. Increase write capacity of DynamoDB to increase the speed of data uploads
7. Add search index to DynamoDB tables

Analysis

The group 1 questions are useful for macro level questions. Like best performing airline, or most popular airport. Basically things that look good for marketing. The group 2 questions are useful for lower level questions. Like best airline to take at a given airport. The group 3 question, is the most micro level of them. It is useful to analyze past flights to see if you got the best one in hindsight. It can also be used to spot trends for the future. .

Video Demonstration

https://youtu.be/xTi5_jYWT5I