# CLOUD COMPUTING CAPSTONE PROJECT PART 1

## INTRODUCTION

The goal of the Capstone project in this Cloud Computing specialization was to apply in practice the knowledge and skills gained throughout the different courses by analyzing a public transportation dataset of the US Bureau of Transportation Statistics.

This report covers the first part of the Capstone project, which focusses on using batch processing systems to solve the presented questions. It contains an overview of the data cleaning process, how the system was constructed, the different optimizations used to improve the performance of the system and queries, the actual results and a conclusion to complete the report.

## Data extraction and cleaning

The RAW data set contained far more data than what was necessary for this project. Analysis of the data directory provided by the EBS volume, indicated that only the ‘airline\_ontime’ directory was of use for this project. The tables in this dataset contained about 80 fields, of which only the following were useful and retained during the cleaning process: FlightDate, Year, Month, Quarter, DayofMonth, DayOfWeek, Origin, Dest, UniqueCarrier, FlightNum, CRSArrTime, ArrTime, ArrDelayMinutes, ArrDelay, CRSDepTime, DepTime, DepDelayMinutes, DepDelay, Cancelled.

To process these files and extract the data. I created a temporary EC2 instance which copy the contents from EBS volume to S3 in the zip format. I used the new serverless paradigm called AWS Lambda to process zip and extract relevant data parallelly. In this managed service, AWS handles the operational challenges like scaling at a very low cost. I implemented 2 functions to process the data:

1. The function ‘​*clean\_zipfile*’​ performed the ETL task for a single zip file. From a high-level perspective the function (i) read the RAW zip file from S3 storage, (ii) decompressed it in memory, (iii) extracted the useful fields and (iv) wrote new CSV files onto S3 (uncompressed).
2. The function ‘​process\_zipfiles ​’ (i) queried S3 and (ii) triggered an asynchronous execution of the first function for each zip file on S3.

The advantage is the automatic scaling this approach offered and scale it with larger set of data too. *Note: The processing of the files, revealed there were some invalid files included in the data set.*

## System

To answer the questions in this project, I decided to use Map-Reduce Hadoop program which processed file from HDFS and eventually stored the final result to DynamoDB using help of HDFS to DynamoDB connector.

YARN

MAP-REDUCE

HDFS

DynamoDB

DynamoDB Connector

I created EMR cluster on AWS to run this Map-Reduce program. EMR cluster provided s3-dist-cp which help me to move clean data from S3 to HDFS as part of initial sets of the setup.

## Solutions Approach - GRoup 1

*Question 1.1 – Rank the top 10 most popular airports by number of flights to/from the airport  
Question 1.2 Rank the top 10 airlines by on-time arrival performance*

A similar approved is being used to slows both above questions. There were two map-reduce stages ran to compute it. Second stage is more less common for both question where map get top 10 locally to the cluster and run single reduce to get global top 10 record. First stage differ little bit.   
*Question 1.1* – Map Reduce flow<.. line> -> **map1()** -> <airport, 1> (for both from and to) -> **reduce1() -> <**airport\_id, total\_count> -> **map2() ->** < NULL, Top\_10\_airport\_based\_on\_count> -> **reduce2() ->** <Top10 airport>  
Map Reduce command: *bin/hadoop jar hsc4\_capstone\_jar.jar com.cloudcomputing.PopularAirportsPlaintext ontime\_data top\_10\_ airports*

*Question 1.2* – Map Reduce flow <.. line> -> **map1()** -> <carrier\_id, arrival\_delay> -> **reduce1() -> <** carrier\_id, avg\_arrival\_delay> -> **map2() ->** < NULL, Top\_10\_carrier\_based\_on\_least\_arrival\_time> -> **reduce2() ->** <Top10 carriers>  
Map Reduce command: *bin/hadoop jar hsc4\_capstone\_jar.jar com.cloudcomputing.AverageDelays ontime\_data top\_10\_carriers*

## Solutions Approach - GRoup 2

*Question 2.1 For each airport X, rank the top-10 carriers in decreasing order of on-time departure performance from X*

*For this calculation, the map-reduce jobs will use a custom compound key, that's composed from airport ID and carrier ID.  
Map jobs will emit each \*\*<airport,carrier>\*\* key pair's on-time departure performance and reduce jobs will calculate average for each \*\*<airport,carrier>\*\* key pair.  
  
<..., line> -> \*\*map()\*\* -> <(airport\_id, carrier\_id departure\_delay> -> \*\*reduce()\*\* -> <(airport\_id, carrier\_id average\_departure\_delay>*

*Question 2.2 For each source airport X, rank the top-10 destination airports in decreasing order of on-time departure performance from X.*

*This is very similar to solution to Question 2.1. Here the compound key consists origin and destination airport.  
  
<..., line> -> \*\*map()\*\* -> <(airport\_from, airport\_to departure\_delay> -> \*\*reduce()\*\* -> <(airport\_from, airport\_to average\_departure\_delay>*

*Question 2.4 For each source-destination pair X-Y, determine the mean arrival delay (in minutes) for a flight from X to Y.*

*This is very similar to solution to Question 2.2. Here the compound value consists arrival delay instead of departure delay.  
  
<..., line> -> \*\*map()\*\* -> <(airport\_from, airport\_to arrival\_delay> -> \*\*reduce()\*\* -> <(airport\_from, airport\_to average\_arrival\_delay>*

## Solutions Approach - GRoup 3

*Question 3.1 Does the popularity distribution of airports follow a Zipf distribution? If not, what distribution does it follow?*

In a Zipf distribution there is an inverse relation between the rank of an event and the frequency of its occurrence. In context of the aviation dataset, this would mean that the most popular airport should have twice as many flights as the 2nd most popular airport and 3 times as many flights as, the 3rd most popular airport, and so on.

A screenshot of a cell phone

Description automatically generated

The table on the left presents the rank-frequency table for our dataset. The 4th column contains the expected value if the popularity distribution would be a Zipf distribution. Clearly the actual values (3rd column) don’t match with the expected values. A Zipf distribution can also easily be recognized as a straight downward line on a log-log graph. The graph on the right above displays this type of graph for the given dataset. Again this is clearly not a straight line. Both support the conclusion that the popularity distribution is not a Zipf distribution.

*Question 3.2 Tom wants to travel from airport X to airport Z. However, Tom also wants to stop at airport Y for some sightseeing on the way. More concretely, Tom has the following requirements*

We use the same approach except, that key-value pairs will be both custom Writable extensions in this case. Keys will be constructed from (airport\_from, airport\_to, flight\_date, am\_or\_pm). Values will be created from (carrier\_id, flight\_num, departure\_time, arrival\_delay). Reduce jobs will get all arrival delays from all the keys.  
The main goal here is to be able to tell average delay for all the origin-destination pair at a given date. Distinguishing morning and afternoon flights. Because the problem can be split into two independent events (getting from X to Y and from Y to Z we can answer each questions by two queries.  
  
<..., line> -> **map()**-> <(airport\_from, airport\_to, flight\_date, am\_or\_pm (carrier\_id, flight\_num, departure\_time, arrival\_delay)> -> **reduce()** -> <(airport\_from, airport\_to, flight\_date, am\_or\_pm (carrier\_id, flight\_num, departure\_time, average\_arrival\_delay)>  
  
finally, we can run the query as separate map reduce step which take inputs from user and perform query on previous step map-reduce output. Final output is stored the results to DynamoDB table.

## Optimizations

## results

*## Question 1.1*  
12446097, ORD  
11537401, ATL  
10795494, DFW  
7721141, LAX  
6582467, PHX  
6270420, DEN  
5635421, DTW  
5478257, IAH  
5197649, MSP  
5168898, SFO   
  
*## Question 1.2*  
-1.01, HA  
1.16, AQ  
1.45, PS  
4.75, ML  
5.35, PA  
5.47, F9  
5.56, NW  
5.56, WN  
5.74, OO  
5.87, 9E  
  
  
*## Question 2.1*CMI  
```  
[(0.61, u'OH', u'CMI'),  
(2.03, u'US', u'CMI'),  
(4.12, u'TW', u'CMI'),  
(4.46, u'PI', u'CMI'),  
(6.03, u'DH', u'CMI'),  
(6.67, u'EV', u'CMI'),  
(8.02, u'MQ', u'CMI')]  
```  
  
BWI  
```  
[(0.76, u'F9', u'BWI'),  
(4.76, u'PA', u'BWI'),  
(5.18, u'CO', u'BWI'),  
(5.5, u'YV', u'BWI'),  
(5.71, u'NW', u'BWI'),  
(5.75, u'AL', u'BWI'),  
(6.0, u'AA', u'BWI'),  
(7.24, u'9E', u'BWI'),  
(7.5, u'US', u'BWI'),  
(7.68, u'DL', u'BWI')]  
```  
  
MIA  
```  
[(-3.0, u'9E', u'MIA'),  
(1.2, u'EV', u'MIA'),  
(1.3, u'RU', u'MIA'),  
(1.78, u'TZ', u'MIA'),  
(2.75, u'XE', u'MIA'),  
(4.2, u'PA', u'MIA'),  
(4.5, u'NW', u'MIA'),  
(6.06, u'US', u'MIA'),  
(6.87, u'UA', u'MIA'),  
(7.5, u'ML', u'MIA')]  
```  
  
LAX  
```  
[(1.95, u'RU', u'LAX'),  
 (2.41, u'MQ', u'LAX'),  
 (4.22, u'OO', u'LAX'),  
 (4.73, u'FL', u'LAX'),  
 (4.76, u'TZ', u'LAX'),  
 (4.86, u'PS', u'LAX'),  
 (5.12, u'NW', u'LAX'),  
 (5.73, u'F9', u'LAX'),  
 (5.81, u'HA', u'LAX'),  
 (6.02, u'YV', u'LAX')]  
```  
  
IAH  
```  
[(3.56, u'NW', u'IAH'),  
 (3.98, u'PA', u'IAH'),  
 (3.99, u'PI', u'IAH'),  
 (4.8, u'RU', u'IAH'),  
 (5.06, u'US', u'IAH'),  
 (5.1, u'AL', u'IAH'),  
 (5.55, u'F9', u'IAH'),  
 (5.71, u'AA', u'IAH'),  
 (6.05, u'TW', u'IAH'),  
 (6.23, u'WN', u'IAH')]  
```  
  
SFO  
```  
[(3.95, u'TZ', u'SFO'),  
 (4.85, u'MQ', u'SFO'),  
 (5.16, u'F9', u'SFO'),  
 (5.29, u'PA', u'SFO'),  
 (5.76, u'NW', u'SFO'),  
 (6.3, u'PS', u'SFO'),  
 (6.56, u'DL', u'SFO'),  
 (7.08, u'CO', u'SFO'),  
 (7.4, u'US', u'SFO'),  
 (7.79, u'TW', u'SFO')]  
```  
  
*# Question 2.2*CMI  
```  
[(-7.0, u'ABI', u'CMI'),  
 (1.1, u'PIT', u'CMI'),  
 (1.89, u'CVG', u'CMI'),  
 (3.12, u'DAY', u'CMI'),  
 (3.98, u'STL', u'CMI'),  
 (4.59, u'PIA', u'CMI'),  
 (5.94, u'DFW', u'CMI'),  
 (6.67, u'ATL', u'CMI'),  
 (8.19, u'ORD', u'CMI')]  
```  
  
BWI  
```  
[(-7.0, u'SAV', u'BWI'),  
 (1.16, u'MLB', u'BWI'),  
 (1.47, u'DAB', u'BWI'),  
 (1.59, u'SRQ', u'BWI'),  
 (1.79, u'IAD', u'BWI'),  
 (3.65, u'UCA', u'BWI'),  
 (3.74, u'CHO', u'BWI'),  
 (4.2, u'GSP', u'BWI'),  
 (4.45, u'SJU', u'BWI'),  
 (4.47, u'OAJ', u'BWI')]  
```  
  
MIA  
```  
[(0.0, u'SHV', u'MIA'),  
 (1.0, u'BUF', u'MIA'),  
 (1.71, u'SAN', u'MIA'),  
 (2.54, u'SLC', u'MIA'),  
 (2.91, u'HOU', u'MIA'),  
 (3.65, u'ISP', u'MIA'),  
 (3.75, u'MEM', u'MIA'),  
 (3.98, u'PSE', u'MIA'),  
 (4.26, u'TLH', u'MIA'),  
 (4.61, u'MCI', u'MIA')]  
```  
  
LAX  
```  
[(-16.0, u'SDF', u'LAX'),  
 (-7.0, u'IDA', u'LAX'),  
 (-6.0, u'DRO', u'LAX'),  
 (-3.0, u'RSW', u'LAX'),  
 (-2.0, u'LAX', u'LAX'),  
 (-0.73, u'BZN', u'LAX'),  
 (0.0, u'MAF', u'LAX'),  
 (0.0, u'PIH', u'LAX'),  
 (1.27, u'IYK', u'LAX'),  
 (1.38, u'MFE', u'LAX')]  
```  
  
IAH  
```  
[(-2.0, u'MSN', u'IAH'),  
 (-0.62, u'AGS', u'IAH'),  
 (-0.5, u'MLI', u'IAH'),  
 (1.89, u'EFD', u'IAH'),  
 (2.17, u'HOU', u'IAH'),  
 (2.57, u'JAC', u'IAH'),  
 (2.95, u'MTJ', u'IAH'),  
 (3.22, u'RNO', u'IAH'),  
 (3.6, u'BPT', u'IAH'),  
 (3.61, u'VCT', u'IAH')]  
```  
  
SFO  
```  
[(-10.0, u'SDF', u'SFO'),  
 (-4.0, u'MSO', u'SFO'),  
 (-3.0, u'PIH', u'SFO'),  
 (-1.76, u'LGA', u'SFO'),  
 (-1.34, u'PIE', u'SFO'),  
 (-0.81, u'OAK', u'SFO'),  
 (0.0, u'FAR', u'SFO'),  
 (2.43, u'BNA', u'SFO'),  
 (3.3, u'MEM', u'SFO'),  
 (4.0, u'SCK', u'SFO')]  
```  
  
*## Question 2.4*CMI - ORD  
```  
[[u'CMI', u'ORD', u'10.14']]  
```  
IND - CMH  
```  
[[u'IND', u'CMH', u'2.89']]  
```  
DFW - IAH  
```  
[[u'DFW', u'IAH', u'7.62']]  
```  
LAX - SFO  
```  
[[u'LAX', u'SFO', u'9.59']]  
```  
JFK - LAX  
```  
[[u'JFK', u'LAX', u'6.64']]  
```  
ATL - PHX  
```  
[[u'ATL', u'PHX', u'9.02']]  
```  
  
*## Question 3.2*CMI → ORD → LAX, 04/03/2008  
```  
MQ 4278 at 07:10, delay: -14.00  
AA 607 at 19:50, delay: -24.00  
```  
  
JAX → DFW → CRP, 09/09/2008  
```  
AA 845 at 07:25, delay: 1.00  
MQ 3627 at 16:45, delay: -7.00  
```  
  
SLC → BFL → LAX, 01/04/2008  
```  
OO 3755 at 11:00, delay: 12.00  
OO 5429 at 14:55, delay: 6.00  
```  
  
LAX → SFO → PHX, 12/07/2008  
```  
WN 3534 at 06:50, delay: -13.00  
US 412 at 19:25, delay: -19.00  
```  
  
DFW → ORD → DFW, 10/06/2008  
```  
UA 1104 at 07:00, delay: -21.00  
AA 2341 at 16:45, delay: -10.00  
```  
  
LAX → ORD → JFK, 01/01/2008  
```  
UA 944 at 07:05, delay: 1.00  
B6 918 at 19:00, delay: -7.00  
```

## 