# Cloud Computing Capstone Task 2 Report

Roney Duílio Stein

Video: https://www.youtube.com/watch?v=OGHewSs_6vY

Source code: https://github.com/roneystein/aviation-spark-processing

## General Considerations

- The datasets used to answer the questions were:
  - **Airline Origin and Destination Survey**: used only for question 1.1 (it's a bigger dataset and only origin and destination fields are kept).
  - **Airline On-Time Performance Data**: remaining questions.
- On-time **performance** is defined as the percentage of flights arrived with **15 or more minutes** of delay, as defined by fields ArrDel15 and DepDel15.
- Questions selected: 1.1, 1.3, 2.2, 2.3, 2.4, 3.1, 3.2.
- Video doesn't cover loading all the scenarios because of the time constraint of 5 minutes.

## Data extraction and cleaning

During Task 1: used a shell script to verify the archive's ZIP files integrity, the existence and content of CSV files inside each. Two ZIP files had no valid content. Extraction was automated using two shell scripts that, when run in the directory containing the data files/directories, processed the content using pipes, not temporary files, as follows:

1. Extracted the CSV files from ZIP files;
2. Processed through the Perl script (one for Origin Destination and another for On-time tables) filtering the desired data and only required columns;
3. Compressed using BZip2;
4. Piped to proper HDFS location.

In the end, for task 1, we had two pre-processed file sets, one for Origin-Destination (Q1.1) and another for On-time statistics.

These task 1 input file sets were transferred for one of the Kafka nodes.

## System construction and integration

The Spark cluster was setup using 5 EC2 m4.xlarge nodes. Cassandra cluster was built using the same cluster nodes. Kafka cluster consisting of 3 nodes using EC2 c4.large instances.

Having the data in one the the Kafka nodes the data was input into the topics using "`bz2cat <files> | kafka-console-producer`", one file at a time.

The Spark programs stored the data to be presented in Cassandra using the Datastax connector.

## Optimizations

Some optimizations made in the process or system:

- Partitioning the Kafka topics for better computing parallelization;
- Used Kafka's Direct Stream for better and easier parallelization.

## Algorithms

The algorithms were coded using Scala.

### **Question 1.1:** Top10Airports.scala

A stateful word counting:

1. Reads the Origin-Destination messages from Kafka into a stream;
2. For each line tokenizes words (separate origin from destination);
3. Groups by airport and count the grouped lines;
4. The airport and count is stored in the state using mapWithState, summing with the old values;
5. The entire state is sorted by number of lines (arrivals and departures) in descending order, cropped the top 10 and displayed.

### **Question 1.3:** RankWeekday.scala

Calculate stateful average value on a group of words:

1. Reads the On-time messages from Kafka into a stream;
2. Emits only the day of week, arrival delay in minutes (only positive) and "1" to count total of records;
3. Reduces by day of week summing delay field and total of records;

4. Stores total of records and total delay per week day into the state summing with the old values;
5. For the entire state calculates the average delay in minutes, sort by delay in ascending order and prints the result.

**Question 2.2:** eachTop10Air.scala
Calculate stateful average for a group then reorder and crop the results:

1. Reads the On-time messages from Kafka into a stream;
2. Emits origin, destination, departure delayed flag (set when >15 min. delayed) and "1" for record count;
3. Reduces by origin-destination pairs summing the delayed flights and number of flights;
4. Stores the values into the state summing with the old values;
5. Gets the state entries and calculates the average delayed departure as a percentage;
6. Groups the averages by origin and:
    a. Order by percentage of delayed departures in ascending order;
    b. Crop the top 10;
    c. Adds a rank numbering;
7. Saves to Cassandra: origin, destination, percentage of delayed flights and rank.

**Question 2.3:** xyTop10Carriers.scala
Calculate average for a group, sort the carriers and list the top.

1. Reads the On-time messages from Kafka into a stream;
2. Emits origin, destination, carrier, arrival delayed flag (set when >15 min. delayed) and "1" for record count;
3. Reduces by origin-destination-carrier triples summing the delayed flights and number of flights;
4. Stores the values into the state summing with the old values;
5. Gets the state entries and calculates the average delayed arrivals as a percentage;
6. Groups the averages by origin-destination pairs and:
    a. Order by percentage of delayed arrivals in ascending order;
    b. Crop the top 10;
    c. Adds a rank numbering;
7. Saves to Cassandra: origin, destination, carrier, percentage of delayed flights and rank.

**Question 2.4:** xyMeanArrivalDelay.scala
Calculate average value for a group.

1. Reads the On-time messages from Kafka into a stream;
2. Emits origin, destination, arrival delay minutes (positive values) and "1" for record count;
3. Reduces by origin-destination pairs summing the delay minutes and number of flights;
4. Stores the values into the state summing with the old values;
5. Gets the state entries and calculates the average of delay in minutes;
6. Saves to Cassandra: origin, destination, mean delay.

**Question 3.2:** xyz.scala
Each record is identified as AM or PM if the flight departures before or after 12:00pm. Follows a selection of best flight for each origin-destination pair, being saved into the same database table. No cross-product or join is made due to the time difference of the events.

1. Reads the On-time messages from Kafka into a stream;
2. Emits the date of flight, origin, destination, day period (AM or PM string), flight (carrier + flight number), departure time and arrival delay in minutes;
3. Reduces by (date, origin, destination, period) and gets only the least delayed flight;
4. Stores the flights into the state keeping only the least delayed flight to avoid querying the database;
5. Saves the state changes to Cassandra.

## Results

Between production and consumption of messages to/from Kafka I observed that a small portion of messages were discarded, dropped or could not be handled by Kafka, no error messages were observable during these operations. This doesn't concern Spark, only Kafka. Due to this some records will be missing for processing.

**Question 1.1:** Top 10 most popular airports.

**(Airport, Popularity)**

```
(ATL,58172803)
(ORD,49587804)
(DFW,44350601)
(DEN,31218149)
(LAX,25450055)
(MSP,24667738)
(CLT,24254589)
(DTW,22926686)
(PHX,21863253)
(IAH,21500862)
```

**Question 1.3:** Rank of the days of weed by on-time arrival performance.

**(Day of Week, % delayed arrivals)**

```
(Saturday,   17.169464)
(Tuesday,    18.838078)
(Monday,     19.60201)
(Sunday,     19.78856)
(Wednesday,  20.43235)
(Thursday,   22.983517)
(Friday,     23.879276)
```

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

| origin | rank | destination | percentage_delayed |
|---|---|---|---|
| SRQ | 1 | EYW | 0 |
| SRQ | 2 | FLL | 0 |
| SRQ | 3 | TPA | 4 |
| SRQ | 4 | MEM | 4 |
| SRQ | 5 | MCO | 4 |
| SRQ | 6 | BNA | 5 |
| SRQ | 7 | RDU | 5 |
| SRQ | 8 | IAH | 6 |
| SRQ | 9 | MSP | 8 |
| SRQ | 10 | RSW | 8 |
| CMH | 1 | AUS | 0 |
| CMH | 2 | SYR | 0 |
| CMH | 3 | ALB | 0 |
| CMH | 4 | OMA | 0 |
| CMH | 5 | CLE | 6 |
| CMH | 6 | SDF | 7 |
| CMH | 7 | IND | 8 |
| CMH | 8 | DAY | 9 |
| CMH | 9 | MEM | 9 |
| CMH | 10 | MSP | 9 |
| JFK | 1 | SWF | 0 |
| JFK | 2 | ANC | 0 |
| JFK | 3 | ISP | 0 |
| JFK | 4 | AGS | 0 |
| JFK | 5 | ABQ | 0 |
| JFK | 6 | MYR | 0 |
| JFK | 7 | UCA | 3 |
| JFK | 8 | STX | 5 |
| JFK | 9 | HPN | 8 |
| JFK | 10 | STT | 8 |
| SEA | 1 | EUG | 0 |
| SEA | 2 | PSC | 7 |
| SEA | 3 | CVG | 8 |
| SEA | 4 | MEM | 10 |
| SEA | 5 | DTW | 11 |
| SEA | 6 | IND | 11 |
| SEA | 7 | IAH | 12 |
| SEA | 8 | LIH | 12 |
| SEA | 9 | DFW | 12 |
| SEA | 10 | CLE | 12 |
| BOS | 1 | SWF | 0 |
| BOS | 2 | ONT | 0 |
| BOS | 3 | LGA | 7 |
| BOS | 4 | AUS | 8 |
| BOS | 5 | BDL | 10 |
| BOS | 6 | MSY | 10 |
| BOS | 7 | LGB | 12 |
| BOS | 8 | OAK | 12 |
| BOS | 9 | RSW | 13 |
| BOS | 10 | CVG | 13 |

**Question 2.3:** For each source-destination rank the top 10 carriers in decreasing order of on-time arrival performance.

| origin | destination | rank | carrier | percentage_delayed |
|---|---|---|---|---|
| LGA | BOS | 1 | TW | 0 |
| LGA | BOS | 2 | DL | 13 |
| LGA | BOS | 3 | US | 13 |
| LGA | BOS | 4 | EA | 16 |
| LGA | BOS | 5 | MQ | 28 |
| LGA | BOS | 6 | NW | 30 |
| LGA | BOS | 7 | OH | 39 |
| LGA | BOS | 8 | AA | 100 |

| | | | | |
|---|---|---|---|---|
| BOS | LGA | 1 | TW | 0 |
| BOS | LGA | 2 | DL | 11 |
| BOS | LGA | 3 | US | 15 |
| BOS | LGA | 4 | EA | 18 |
| BOS | LGA | 5 | MQ | 30 |
| BOS | LGA | 6 | NW | 33 |
| BOS | LGA | 7 | AA | 50 |
| BOS | LGA | 8 | OH | 62 |
| BOS | LGA | 9 | TZ | 100 |
| OKC | DFW | 1 | TW | 5 |
| OKC | DFW | 2 | EV | 12 |
| OKC | DFW | 3 | AA | 12 |
| OKC | DFW | 4 | OO | 14 |
| OKC | DFW | 5 | DL | 14 |
| OKC | DFW | 6 | MQ | 18 |
| OKC | DFW | 7 | OH | 100 |

| | | | | |
|---|---|---|---|---|
| MSP | ATL | 1 | EA | 13 |
| MSP | ATL | 2 | OO | 18 |
| MSP | ATL | 3 | FL | 20 |
| MSP | ATL | 4 | DL | 23 |
| MSP | ATL | 5 | NW | 24 |
| MSP | ATL | 6 | OH | 25 |
| MSP | ATL | 7 | EV | 27 |

**Question 2.4:** For each source-destination determine the mean arrival delay in minutes.

| origin | destination | mean_delay |
|---|---|---|
| LGA | BOS | 7.95104 |
| BOS | LGA | 8.42176 |
| OKC | DFW | 8.37676 |
| MSP | ATL | 12.07444 |

**Question 3.2:**

Results are queried from database using two queries, one for the first leg specifying "AM" in the period field and another for the second leg using "PM" as period. Although this output is not the result of a direct computation using cross-product or joins, in a real life use case this makes a much more flexible solution and uses much less database resources.

| departure_date | period | origin | destination | delay | departure_time | flight |
|---|---|---|---|---|---|---|
| 03/04/08 | AM | BOS | ATL | 7 | 600 | FL 270 |
| 05/04/08 | PM | ATL | LAX | 0 | 1705 | DL 75 |
| 07/09/08 | AM | PHX | JFK | 0 | 1130 | B6 178 |
| 09/09/08 | PM | JFK | MSP | 0 | 1750 | NW 609 |
| 24/01/08 | AM | DFW | STL | 0 | 945 | AA 1030 |
| 26/01/08 | PM | STL | ORD | 0 | 1335 | AA 1835 |
| 16/05/08 | AM | LAX | MIA | 10 | 820 | AA 280 |
| 18/05/08 | PM | MIA | LAX | 0 | 1930 | AA 456 |

## Conclusion

The results generated could be used for some kind of recommendation system, improvements in the airport or airline infrastructure, improvements of airline process, and others.

Spark compared to Hadoop seems easier to use and faster. Building a streaming computation, in the other hand, have some additional challenges compared to a batch processing system. One of the challenges is managing state when doing stateful computations and another is managing the resources so the cluster can cope with the speed of the incoming data. Overall, the algorithms employed are a little different due to the stateful computation and the incoming stream.