Cloud Computing Capstone project

Johan Kielbaey - August 12th 2017

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. A set of questions had to be solved using the techniques and frameworks presented in the courses.

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.
The video demonstration is available on https://youtu.be/7sQnNYBh8-Y

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 BTS[1], 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, DayofMonth, DayOfWeek, Origin, Dest, UniqueCarrier, FlightNum, CRSArrTime, ArrTime, ArrDelayMinutes, ArrDelay, CRSDepTime, DepTime, DepDelayMinutes, DepDelay, Cancelled.

To process these files and extract the data, I used the new serverless paradigm, called AWS Lambda[2]. 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 '*handle_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 '*get_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. Each file would take between 40 and 50 seconds to be processed, however as all 242 files are processed in parallel, the total time was less than 60 seconds. If the total number of files to be processed would increase drastically, the total processing time would remain in the same order of time.

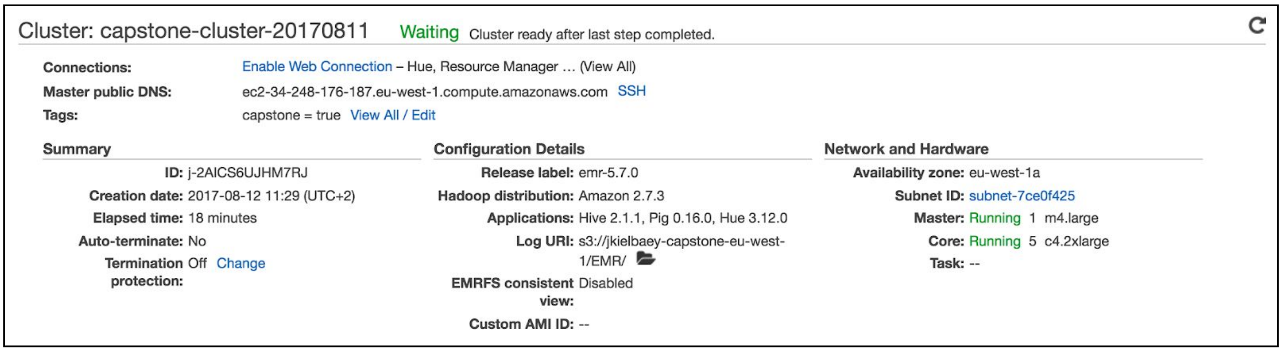
Note: The processing of the files, revealed there were 2 invalid files included in the data set. The contents of these files indicate the files actually did not exist on the servers of BTS. I excluded these from further processing.

- *2008/On_Time_On_Time_Performance_2008_11.zip*
- *2008/On_Time_On_Time_Performance_2008_12.zip*

1
https://www.transtats.bts.gov/Tables.asp?DB_ID=120&DB_Name=Airline%20On-Time%20Performance%20Data&DB_Short_Name=On-Time
2 https://aws.amazon.com/lambda/

System

To answer the questions in this project, I decided to use Hive on an EMR cluster (with 1 master node and 5 core nodes). Two steps were added to the EMR cluster to (i) automatically load the data from S3 into HDFS (using s3-dist-cp) and (ii) create an external table using the files in HDFS.



The screenshot above illustrates the EMR cluster used for this task .

I choose to use Hive as this would allow me to be more productive and focus on the actual questions, instead of having to write Map/Reduce functions in Java. The EMR service is used to analyze TBs of data in large enterprises, so it was the obvious choice as platform for Map/Reduce. For the questions in group 2 I decided to use DynamoDB to store the results. Hive on EMR comes with a storage adapter for DynamoDB built-in. DynamoDB is a fully managed NoSQL database.

Optimizations

By default EMR on a 3 node cluster has a replication factor 2. A first optimization was to override this replication factor to 3. This meant that during the initial load of the data into HDFS there was more network traffic. However, during the actual query execution, the chance of having the data available on the local worker node had increased, which reduced network traffic between nodes.

While cleaning the data, I re-arranged the original data into separate files per day, which were grouped in 2 levels of directories (per year and per month). This allowed loading the data as separate partitions per month into Hive. When querying on partitions, Hive does not have to read data from unused partitions, which can dramatically reduce the amount of data Hive must process.

I experimented with modifying various Hive settings. AWS already did an excellent job in providing an optimized configuration of Hive. Most of the frequently suggested recommendations[3] were already applied out-of-the-box. Not all of the remaining recommended settings resulted in shorter run times.
The baseline performance (out-of-the-box EMR cluster) for question 1.1 was 87.6 sec on a 3 node cluster. Through Hive setting and query parameter tuning this was reduced to 60.2 sec. This 31% improvement was achieved using following changes:

- Enable parallel execution of independent MapReduce stages.
- Compressing intermediate output.
- Enable CBO based on stats and calculate table statistics.

[3] See Sources for the list of Hive tuning resources I used.

A final optimization was to add more CPU cores to the cluster by increasing the number of core nodes from 3 to 5 and rebalancing the data over all nodes. This cluster modification increased the number of cores from 12 to 20 and thus allowed Hive to execute more map and reduce jobs in parallel. The same query completed in 48.3 sec[4].

Results

This section presents the solutions to each of the different questions.

Group 1

The queries used to calculate these results were rather simple. For the first query I used 2 sub queries to count the number of incoming and outgoing flights and add both numbers to calculate the total per airport. The queries for 1.2 and 1.3 only differ from each other in the GROUP BY field.

Question 1.1

Airport	Total number of flights
ORD	12.449.354
ATL	11.540.422
DFW	10.799.303
LAX	7.723.596
PHX	6.585.534
DEN	6.273.787
DTW	5.636.622
IAH	5.480.734
MSP	5.199.213
SFO	5.171.023

Question 1.2

Airlines	Average delay (minutes)
HA	-1.01
AQ	1.16
PS	1.45
ML (1)	4.75
PA (1)	5.32
F9	5.47
WN	5.56
NW	5.56
OO	5.74
9E	5.87

Question 1.3

Weekday	Average delay (minutes)
6 (Saturday)	4.3
2 (Tuesday)	5.99
7 (Sunday)	6.61
1 (Monday)	6.72
3 (Wednesday)	7.2
4 (Thursday)	9.09
5 (Friday)	9.72

Group 2

Question 2.1

The query for this question has multiple nested queries. The innermost query calculates the average delay for each airline at every airport. This information is then ordered and foreseen of a ranking number. The outer query filters to only retain the top 10 airlines per airport and insert this data into DynamoDB. Via the DynamoDB Storage handler an external table in Hive is directly linked with a DynamoDB Table. The only catch with DynamoDB is that the write capacity units need to be set sufficiently high to allow Hive to insert the data fast enough into DynamoDB.

[4] In the video report the execution time for question 1.1 was 32 sec. The reason is that I created a new EMR cluster for the video using c4.2xlarge core nodes which contain double amount of CPU and memory resources compared to the c4.xlarge instances.

Airport	Airline	Average delay (minutes)
CMI	OH	0.61
	US	2.03
	TW	4.12
	PI	4.46
	DH	6.03
	EV	6.67
	MQ	8.02
BWI	F9	0.76
	PA (1)	4.76
	CO	5.18
	YV	5.5
	NW	5.71
	AA	6.0
	9E	7.24
	US	7.49
	DL	7.68
	UA	7.74

Airport	Airline	Average delay (minutes)
MIA	9E	-3.0
	EV	1.2
	TZ	1.78
	XE	1.87
	PA (1)	4.2
	NW	4.5
	US	6.09
	UA	6.87
	ML (1)	7.5
	FL	8.57
LAX	MQ	2.41
	OO	4.22
	FL	4.73
	TZ	4.76
	PS	4.86
	NW	5.12
	F9	5.73
	HA	5.81
	YV	6.02
	US	6.75

Airport	Airline	Average delay (minutes)
IAH	NW	3.56
	PA (1)	3.98
	PI	3.99
	US	5.06
	F9	5.55
	AA	5.7
	TW	6.05
	WN	6.23
	OO	6.59
	MQ	6.71
SFO	TZ	3.95
	MQ	4.85
	F9	5.16
	PA (1)	5.29
	NW	5.76
	PS	6.3
	DL	6.56
	CO	7.08
	US	7.53
	TW	7.79

Question 2.2

The query for this question only differed from question 2.1 in the field used to group the data.

Airport	Destination airport	Average delay (minutes)
CMI	ABI	-7.0
	PIT	1.1
	CVG	1.89
	DAY	3.12
	STL	3.98
	PIA	4.59
	DFW	5.94
	ATL	6.67
	ORD	8.19
BWI	SAV	-7.0
	MLB	1.16
	DAB	1.47
	SRQ	1.59
	IAD	1.79
	UCA	3.65
	CHO	3.74
	GSP	4.2
	SJU	4.44
	OAJ	4.47

Airport	Destination airport	Average delay (minutes)
MIA	SHV	0.0
	BUF	1.0
	SAN	1.71
	SLC	2.54
	HOU	2.91
	ISP	3.65
	MEM	3.75
	PSE	3.98
	TLH	4.26
	MCI	4.61
LAX	SDF	-16.0
	IDA	-7.0
	DRO	-6.0
	RSW	-3.0
	LAX	-2.0
	BZN	-0.73
	MAF	0.0
	PIH	0.0
	IYK	1.27
	MFE	1.38

Airport	Destination airport	Average delay (minutes)
IAH	MSN	-2.0
	AGS	-0.62
	MLI	-0.5
	EFD	1.89
	HOU	2.17
	JAC	2.57
	MTJ	2.95
	RNO	3.22
	BPT	3.6
	VCT	3.61
SFO	SDF	-10.0
	MSO	-4.0
	PIH	-3.0
	LGA	-1.76
	PIE	-1.34
	OAK	-0.81
	FAR	0.0
	BNA	2.43
	MEM	3.3
	SCK	4.0

Question 2.3

The query for this question was again very similar to the queries for 2.1 and 2.2. In this case there were more fields to group the data by. Storing the results into DynamoDB required combining the source and destination airport into a single column ('flight') due to the way DynamoDB handles partition and sort keys.

X - Y	Airline	Average delay (minutes)	X - Y	Airline	Average delay (minutes)	X - Y	Airline	Average delay (minutes)
CMI-ORD	MQ	10.14	DFW-IAH	PA (1)	-1.6	JFK-LAX	UA	3.31
				EV	5.09		HP	6.68
				UA	5.41		AA	6.9
				CO	6.49		DL	7.93
LAX-SFO	TZ	-7.62	ATL-PHX	OO	7.56	IND-CMH	PA (1)	11.02
	PS	-2.15		XE	8.09		TW	11.7
	F9	-2.03		AA	8.38		CO	-2.55
	EV	6.96		DL	8.6		AA	5.5
	AA	7.39		MQ	9.1		HP	5.7
	MQ	7.81		FL	4.55		NW	5.76
	US	7.96		US	6.29		US	6.88
	WN	8.79		HP	8.48		DL	10.69
	CO	9.35		EA	8.95		EA	10.81
	NW	9.85		DL	9.81			

Question 2.4

The query for this question didn't require any nested queries. A simple select that grouped the data by origin and destination airport was sufficient. Again to store this into DynamoDB, both had to be combined into a single column.

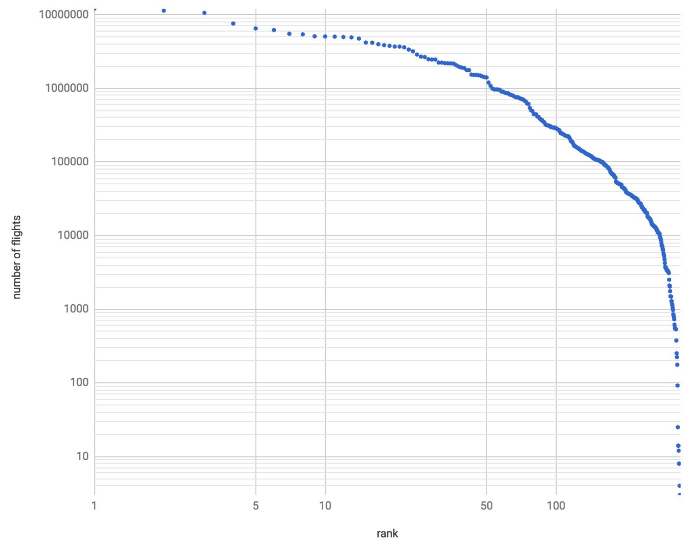
X - Y	Average delay (minutes)
CMI-ORD	10.14
IND-CMH	2.9
DFW-IAH	7.65
LAX-SFO	9.59
JFK-LAX	6.64
ATL-PHX	9.02

Group 3

Question 3.1

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.

Airport	Rank	Number of flights	Expected
ORD	1	12.051.796	12.051.796
ATL	2	11.323.515	6.025.898
DFW	3	10.591.818	4.017.265
LAX	4	7.586.304	3.012.949
PHX	5	6.505.078	2.410.359
DEN	6	6.183.518	2.008.633
DTW	7	5.504.120	1.721.685
IAH	8	5.416.653	1.506.475
MSP	9	5.087.036	1.339.088
SFO	10	5.062.339	1.205.180
STL	11	5.031.131	1.095.618
EWR	12	4.979.913	1.004.316
LAS	13	4.917.971	927.061
CLT	14	4.735.669	860.843
LGA	15	4.179.969	803.453



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

The query for this question combines 2 sub queries. For each flight (X → Y and Y → Z) I used a different sub query to meet the requirements of the particular flights (departure before/after 12:00PM). In both sub queries a ranking on the arrival delay of the flight on any given date was calculated. Both sub queries were combined in such a way that they met the requirement of having 2 days between both flights, while at the same time only retaining the best ranking (the least amount of delay) flight between each combination of origin and destination. The combination of both flights was stored in a separate Hive table.

	Leg 1			Leg 2			
Date	X - Y	Flight number	Delay	Y - Z	Flight number	Delay	Total delay
04-03-2008	CMI - ORD	MQ 4278	-14.0	ORD - LAX	AA 607	-24.0	-38.0
09-09-2008	JAX - DFW	AA 845	1.0	DFW - CRP	MQ 3627	-7.0	-6.0
01-04-2008	SLC - BFL	OO 3755	12.0	BFL - LAX	OO 5429	6.0	18.0
12-07-2008	LAX - SFO	WN 3534	-13.0	SFO - PHX	US 412	-19.0	-32.0
10-06-2008	DFW - ORD	UA 1104	-21.0	ORD - DFW	AA 2341	-10.0	-31.0
01-01-2008	LAX - ORD	UA 944	1.0	ORD - JFK	B6 918	-7.0	-6.0

Conclusion

Airports work with arrival and departure slots (a reservation to land or take off at a given time). A flight that misses its slots must wait in line for another slot, which can lead to financial impact (loss of revenue, increased operational costs, etc) for the airline. An airline that is consistently missing its slot, might even lose its slot all together and thus incur on a higher financial impact. For airlines, keeping to a timely schedule is important. In that sense these results are as expected. I was surprised to see negative averages, which meant that some flights are structurally leaving or arriving early.
This type of information can be very useful for companies and organizations linked to aviation. A travel agent can benefit from this information to provide better recommendations to customers when booking flights. Airlines could use it to analyze why they have delays, at which airport or at what time of the day these delays occur most frequently and decide to alter their flight schedules to mitigate such delays in order to incur in less financial impact, therefore achieving better reputation and improved customer satisfaction.

Sources

Hive tuning

- 10 Best Practices for Apache Hive, https://www.qubole.com/blog/hive-best-practices/
- 5 ways to make your Hive queries run faster, https://hortonworks.com/blog/5-ways-make-hive-queries-run-faster/
- Hive Performance Tuning, http://hadooptutorial.info/hive-performance-tuning/
- Enable Compression in Hive, http://hadooptutorial.info/enable-compression-in-hive

Hive & DynamoDB

- Hive & DynamoDB Pitfalls, http://arjon.es/2014/01/29/hive-dynamodb-pitfalls/

Zipf distribution

- Zipf's law, https://en.wikipedia.org/wiki/Zipf%27s_law
- The mystery of Zipf, https://plus.maths.org/content/mystery-zipf