Cloud Computing Capstone: Task 1 Report by Panagiotis Salteris

Data extraction and cleaning

I performed the following steps in order to extract and clean the data:

- By taking a look at the queries and the dataset profile, it was apparent that the only
- relevant table is *airline_ontime*
- Fetched the table data and running the *script move-to-hdfs.sh* from the snapshot, unzipped the csv files to HDFS keeping only the columns which were needed to answer the questions
- As a result only the following columns were taken in the new csv files in HDFS:
    - *Year, Month, Quarter, DayOfWeek, DayofMonth, FlightDate, UniqueCarrier, FlightNum, Origin, Dest, CRSDepTime, DepTime, DepDelay, DepDelayMinutes, CRSArrTime, ArrTime, ArrDelay, ArrDelayMinutes, Cancelled, Diverted.*

Systems

The following systems were used to complete the task:

- an ec2 t2.medium instance was used to store the data and as a master node for the Hadoop system and as resource manager for the Yarn system
- three ec2 t2.large instances were used as data nodes for the Hadoop system and as node managers for the Yarn system
- hadoop-streaming-2.10.0.jar was used to run all the map reduce tasks in order to find all results from the questions
- DynamoDB was used to store the results of part 2 and 3.2

Algorithms and approaches

1.1 Calculating top 10 most popular airports by numbers of flights: A map-reduce job collects all the *Origin* and *Dest* airport field from *airline_ontime* data and counts each airport. This is very similar to the well-known Word Count example in Hadoop documentation.
<..., line> -> **map()** -> <airport_id(*Origin and Dest*), 1> -> **reduce()** -> <airport_id, occurrence>
As result we get one file with airports in alphabetical order. And with the following command the result is sorted recursively by the number of occurrences:
~$ hdfs dfs -cat /task1-group1-1-hadoop/part-00000 | sort -k2rn | head -10

“ORD” 12449354
“ATL” 11540422
“DFW” 10799303
“LAX” 7723596
“PHX” 6585534
“DEN” 6273787
“DTW” 5636622
“IAH” 5480734
“MSP” 5199213
“SFO” 5171023

1.2 Calculating top 10 airlines by on-time arrival performance: The approach was to map the data by *UniqueCarrier* and *ArrDelay* where the rows with missing *ArrDelay* value were ignored. Then a reduce task was used to find the average arrival delay for each result *UniqueCarrier.*
<..., line> -> **map()** -> <unique_carrier, arr_delay> -> **reduce()** -> < unique_carrier, ave_arr_delay>
With the following command the result is sorted recursively by the number of occurrences:
~$ hdfs dfs -cat /task1-group1-2-hadoop/part-00000 | sort -k2n | head -10

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

2.1 For each airport compute the top 10 carriers in decreasing order of on-time departure:
Firstly, the data are mapped by <(*Origin, UniqueCarrier*), *DepDelay*> where rows with missing *DepDelay* were ignored and with a reduce task average value of *DepDelay* is computed for each *Origin*, *UniqueCarrier* pair.
<..., line> -> **map()** -> <(origin, unique_carrier), dep_delay> -> **reduce()** -> <(origin, unique_carrier), ave_dep_delay>
Then the results are stored into DynamoDB table. Finally with a python script and the use of boto3 module the data for specific keys were retrieved and sorted.

The query for CMI returned the following items:

OH	0.61
US	2.03
TW	4.12
PI	4.46
DH	6.03
EV	6.67
MQ	8.02

The query for BWI returned the following items:

F9	0.76
PA (1)	4.76
CO	5.18
YV	5.50
NW	5.71
AA	6.00
9E	7.24
US	7.49
DL	7.68
UA	7.74

The query for MIA returned the following items:

9E	-3.0
EV	1.20
TZ	1.78
XE	1.87
PA (1)	4.20
NW	4.50
US	6.09
UA	6.87
ML (1)	7.50

The query for LAX returned the following items:

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

FL	8.57	US	6.75

The query for IAH returned the following items:		The query for SFO returned the following items:	
NW	3.56	TZ	3.95
PA (1)	3.98	MQ	4.85
PI	3.99	F9	5.16
US	5.06	PA (1)	5.29
F9	5.55	NW	5.76
AA	5.70	PS	6.30
TW	6.05	DL	6.56
WN	6.23	CO	7.08
OO	6.59	US	7.53
MQ	6.71	TW	7.79

2.2 For each airport Compute the top 10 destinations in decreasing order of on-time departure:
Firstly, the data are mapped by <(*Origin*, *Dest*), *DepDelay*> where rows with missing *DepDelay* were ignored and with a reduce task average value of *DepDelay* is computed for each *Origin*, *Dest* pair.
<..., line> -> **map()** -> <(origin, dest), dep_delay> -> **reduce()** -> <(origin, dest), ave_dep_delay>
Then the results are stored into DynamoDB table. Finally with a python script and the use of boto3 module the data for specific keys were retrieved and sorted.

The query for CMI returned the following items:		The query for BWI returned the following items:	
ABI	-7.0	SAV,	7.0
PIT	1.10	MLB	1.16
CVG	1.89	DAB	1.47
DAY	3.12	SRQ	1.59
STL	3.98	IAD	1.79
PIA,	4.59	UCA	3.65
DFW	5.94	CHO	3.74
ATL	6.67	GSP	4.20
ORD	8.19	SJU	4.44
		OAJ	4.47

The query for MIA returned the following items:		The query for LAX returned the following items:	
SHV	0.0	SDF	-16.0
BUF	1.0	IDA	-7.0
SAN	1.71	DRO	-6.0
SLC	2.5	RSW	-3.0
HOU	2.91	LAX	-2.0
ISP	3.65	BZN	-0.73
MEM	3.75	MAF	0.0
PSE	3.98	PIH	0.0
TLH	4.26	IYK	1.27
MCI	4.61	MFE	1.38

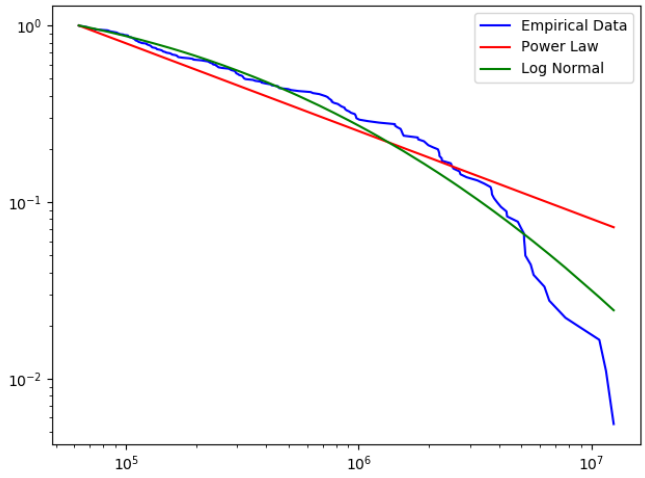
The query for IAH returned the following items:		The query for SFO returned the following items:	
MSN	-2.0	SDF	-10.0
AGS	-0.62	MSO	-4.0

MLI	-0.5	PIH	-3.0
EFD	1.89	LGA	-1.76
HOU	2.17	PIE	-1.34
JAC	2.57	OAK	-0.81
MTJ	2.95	FAR	0.0
RNO	3.22	BNA	2.43
BPT	3.60	MEM	3.30
VCT	3.61	SCK	4.0

2.4 For each origin destination pair, compute mean arrival delay: This is very similar to solution to Question 2.2. Here the compound value consists arrival delay instead of departure delay. The data were mapped by *Origin*, *Dest* and average *ArrDelay* was computed from the reduce task. Rows with missing *ArrDelay* were ignored.
<..., line> -> **map()** -> <(origin, dest), arr_delay> -> **reduce()** -> <(origin, dest), ave_arr_delay>
The output was stored in dynamodb and with another python script the following results were retrieved:

The query for CMI returned the following items:
"CMI" -> "ORD": 10.14
The query for IND returned the following items:
"IND" -> "CMH": 2.9
The query for DFW returned the following items:
"DFW" -> "IAH": 7.65
The query for LAX returned the following items:
"LAX" -> "SFO": 9.59
The query for JFK returned the following items:
"JFK" -> "LAX": 6.64
The query for ATL returned the following items:
"ATL" -> "PHX": 9.02

3.1 Total frequency (to and from) of flights from each airport was calculated and ranked in descending order. This gave the frequency count as well as the rank needed to plot the curve. The CCDF of the popularity for airports looks like the following:



The CCDF of power-law distributions should be a straight line. Also the lognormal distribution fits better to the empirical data. So, the popularity of the airports definitely doesn't follow Zipf distribution.
Log likelihood ration tests gives us the following results, when comparing the fitted power-law and lognormal distributions:
R -3.485522, p 0.000491
As R is negative, the empirical data more likely follows a log normal distribution.

3.2 In order to save the data in dynamodb with unique keys an additional column needs to be added. Apart from this the queries are also easier in order to retrieve the first or second leg. Keys will be constructed from (origin, dest, flight_date, am_or_pm). Values will be created from (unique_carrier, flight_num, dep_time, arr_delay). Reduce jobs will get the minimum arrival delay for each flight with specific origin, destination, flight number and whether it is pm or am.
<..., line> -> map() -> <(origin, dest, flight_date, am_or_pm), (unique_carrier, flight_num, dep_time, arr_delay)> -> reduce() -> <(origin, dest, flight_date, am_or_pm), (unique_carrier, flight_num, dep_time, minimum(arr_delay))>
The output was stored in dynamodb and with another python script the following results were retrieved:

Tom wants to fly from "CMI" to "ORD" at 2008-03-04 and from "ORD" to "LAX" at 2008-03-06:
{'CMI -> ORD', '4374', 'MQ', '-14.0', '2008-03-04', '605 AM'}
{'ORD -> LAX', '455', 'AA', '-24.0', '2008-03-06', '1510 PM'}
Total arrival delay: -38.0

Tom wants to fly from "JAX" to "DFW" at 2008-09-09 and from "DFW" to "CRP" at 2008-09-11:
{'JAX -> DFW', '845', 'AA', '1.0', '2008-09-09 ', '725 AM '}
{'DFW -> CRP', '3419', 'MQ', '-7.0', '2008-09-11', '1450 PM'}
Total arrival delay: -6.0

Tom wants to fly from "SLC" to "BFL" at 2008-04-01 and from "BFL" to "LAX" at 2008-04-03:
{'SLC -> BFL', '3755', 'OO', '12.0', '2008-09-09 ', '1100 AM '}
{'BFL -> LAX', '5429', 'OO', '6.0', '2008-09-11', '1455 PM'}
Total arrival delay: 18.0

Tom wants to fly from "LAX" to "SFO" at 2008-07-12 and from "SFO" to "PHX" at 2008-07-14:
{'LAX -> SFO', '889', 'UA', '-13.0', ' 2008-07-12', '759 AM'}
{'SFO -> PHX', '1619', 'WN', '-19.0', '2008-07-14', '1605 PM'}
Total arrival delay: -32.0

Tom wants to fly from "DFW" to "ORD" at 2008-06-10 and from "ORD" to "DFW" at 2008-06-12:
{'DFW -> ORD', '2332', 'AA', '-21.0', '2008-06-10', '915 AM'}
{'ORD -> DFW', '2361', 'AA', '-10.0', '2008-06-12', '2150 PM'}
Total arrival delay: -31.0

Tom wants to fly from "LAX" to "ORD" at 2008-01-01 and from "ORD" to "JFK" at 2008-01-03:
{'LAX -> ORD', '2276', 'AA', '1.0', '2008-01-01', '630 AM'}
{'ORD -> JFK', '5366', 'OH', '-7.0', '2008-01-03', '1730 PM'}
Total arrival delay: -6.0