Cloud Computing Capstone Project Report

by

Karthik Balaji Gandhi

AWS setup:

AWS Instances	4
AWS instance type	M4.2xlarge (us-west-1b)
Vcpu	8
Memory	32 GB
Storage	200 GB (EBS)
Hadoop Distribution	Hortonworks Hadoop Platform
Hadoop Version	hadoop-common-2.7.1
Cassandra Distribution	Apache Cassandra – 3.2.1
PIG	Apache PIG version 0.15

I tried Cloudera's Hadoop distribution as well. But Hortonworks is by far the cleanest and easiest to use.

System integration:

Hortonworks provided me with a well-integrated platform that had all the frameworks I needed. Initially I underestimated the memory, cpu and storage needs. So I had to iterate it for 3 times. The above configuration worked really well for my needs. Cassandra was not provided by Hortonworks. So downloaded and installed Apache Cassandra on top of the Hortonworks platform. I had to adjust the JMX settings for Cassandra. Oftentimes I had to run the same command on all 4 servers. So I used saltstack to automate that.

How data were extracted and cleaned:

Using Python 2.6, I cleaned, extracted and organized the data by year. (Ex) 2008.csv, 2007.csv, etc.

I ran into issues were a couple of files were missing or in the wrong format. Precisely, 9 months' worth of data from 1993 was missing and a couple of months of data from 2008 were corrupt. My script pulled the missing data from the BTS website. This may vary my results slightly as I have more data points in my input.

I combined redundant columns together and removed the redundant variables. For ex, the reasons for cancellation of a flight are not as relevant as the cancellation itself. The final set of columns are,

Year, Month, DayofMonth, DayOfWeek, Origin, Destination, DepTime, CRSDepTime, ArrTime, CRSArrTime, CarrierCode, FlightNum, ActualElapsedTime, CRSElapsedTime, ArrDelay, DepDelay, Origin, Destination ,Cancelled

The data was cleaned using by extensively using Python the Pandas library. I was able to bring down 37 GB of raw uncompressed data to about 10 GB. The data were then pushed out to HDFS through simple Hadoop commands. The replication factor for the HDFS is set to 3.

Approaches and results for individual problems:

1.1 Top Airports:

I reduced it to a simple two stage map-reduce(MR) problem. First stage aggregates the counts of airports that appear in “Origin” and “Destination” fields of my input data. The second stage does a sort and emits the top 10 airports.

AIRPORT	FREQUENCY
SFO	5278456
MSP	5357663
IAH	5665046
DTW	5783380
DEN	6475024
PHX	6844153
LAX	7943304
DFW	11138977
ATL	11839964
ORD	12688656

1.2 Top Airlines by On-time Arrival Performance:

Effectively used the two-stage MR, one to count and another to sort the results. Initially, I was using the absolute sum of the arrival delays for each airline. The results didn’t make sense. So I decided to use the average arrival delay as my metric.

AIRLINE	PERFORMANCE
9E	6.108228
OO	5.877036
F9	5.693196
WN	5.5184155
NW	5.4308276
PA (1)	5.322431
ML (1)	4.747609
PS	1.4506385
AQ	1.1569234
HA	-0.77522093

2.1 *Top airlines by airport based on departure performance:*

Used MR to calculate (Airport, Airline_<AVG_Delay>) and store it to Cassandra. Then used Cassandra’s Java API to run queries and get the data. The SORT and LIMIT are done with Cassandra primitives. You can see my code through the provided GitHub link at the bottom of this document. Just to make sure that the results are correct, I ran a second MR job to calculate the result. The Cassandra output matched the MR output.

AIRPORT	Top Airlines								
CMI	OH	US	TW	PI	DH	EV	MQ		
BWI	F9	PA (1)	CO	YV	NW	AA	9E	US	UA	FL
MIA	EV	TZ	XE	PA (1)	NW	US	UA	ML	MQ	FL
LAX	MQ	OO	TZ	PS	FL	NW	YV	F9	HA	US
IAH	NW	PA	PI	US	AA	F9	TW	WN	MQ	OO
SFO	TZ	MQ	PA	F9	NW	PS	DL	CO	US	AA

2.2 *Top Destinations by Airport based on departure performance:*

Same approach as 2.1. Used MR to calculate (OriginAirport, Destination_Airport<AVG_Delay>) pair and loaded that data to Cassandra. Results from Cassandra output were verified to match the results from MR.

AIRPORT	Top Destination									
CMI	ABI	PIT	CVG	DAY	STL	PIA	DFW	ATL	ORD	SPI
BWI	SAV	MLB	DAB	SRQ	IAD	UCA	CHO	GSP	OAJ	SJU
MIA	SHV	BUF	SAN	SLC	HOU	MEM	ISP	PSE	TLH	MCI
LAX	SDF	IDA	DRO	RSW	LAX	BZN	MAF	PIH	IYK	MFE
IAH	MSN	AGS	MLI	EFD	HOU	JAC	MTJ	RNO	BPT	VCT
SFO	SDF	MSO	PIH	LGA	PIE	OAK	FAR	BNA	MEM	SCK

2.3 *Top Airlines for a given X->Y based on departure performance:*

Same approach as 2.1 and 2.2. Used MR to calculate (<Origin->Destination>, <Airline, AVG_Delay> Pair. My Cassandra driver processed the data and loaded it into Cassandra and then queried it for the required result.

Source to destination	Top Airlines								
CMI->ORD	MQ								
IND->CMH	CO	HP	AA	NW	US	DL	EA		
DFW->IAH	PA (1)	EV	UA	CO	OO	XE	AA	DL	MQ	
LAX->SFO	TZ	PS	F9	EV	AA	MQ	US	CO	NW	UA
JFK->LAX	UA	AA	HP	DL	PA (1)	TW				
ATL->PHX	FL	US	HP	EA	DL					

3.1 *Popularity Distribution of the Airports:*

The intermediate result for question 1.1 forms the input for this question. The popularity distribution follows the ZipF distribution. I studied the real number distributions and integer distributions. Since we don’t have negative numbers or decimal points, I eliminated the real number distributions or a 4 quadrant graph.

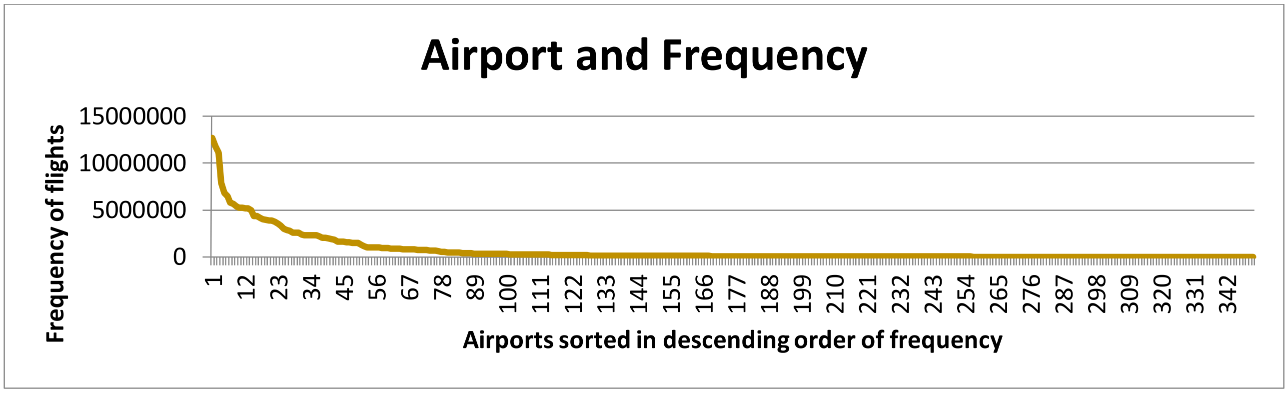

Figure 1 – Frequency plot

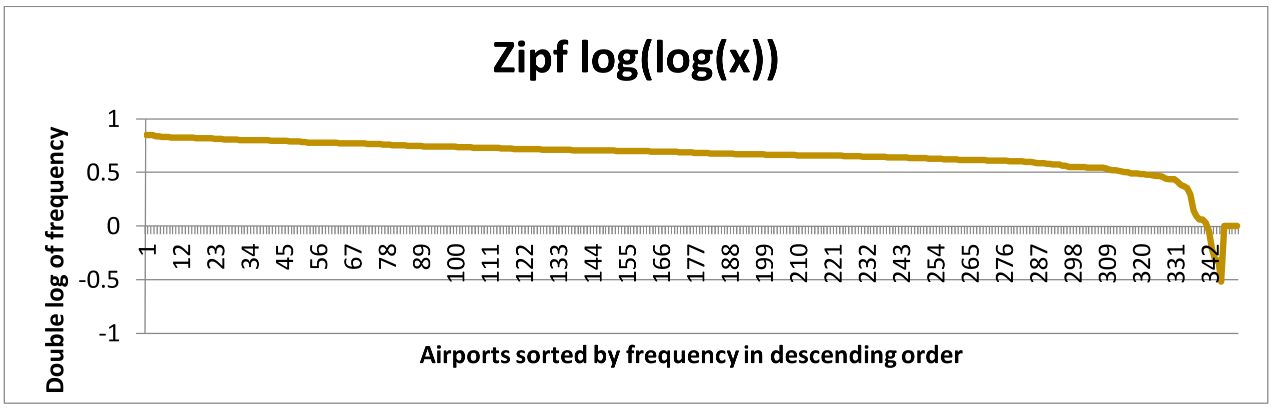

Figure 2 – Double log plot

As seen from Figure 1, most airports fall in the tail of the distribution and a very few airports have higher frequencies. As seen from Figure 2, a double logarithmic of the plotted co-ordinates almost form a straight line. These observations clearly match the properties of a ZipF distribution.

3.2 *Tom's Flight Options*

Since I have been using MR for the rest, I decided to try out PIG for this script. I generated two data sets of data, one that satisfies all the conditions of a first trip and the other that satisfies all the conditions of a second trip. Then I joined the data points based on the "Destination" of the first trip and the "Origin" of the second trip. Since we only want the trip with minimum delay for each (X->Y-Z) triplet, I sorted the result and took the top result. The final dataset was loaded into Cassandra and then was queried for the task1 input.

04/03/2008	CMI->ORD	MQ_4401	ORD->LAX	AA_1345
09/09/2008	JAX->DFW	AA_845	DFW->CRP	MQ_3627
01/04/2008	SLC->BFL	OO_3755	BFL_LAX	OO_5429
12/07/2008	LAX->SFO	WN_3534	SFO->PHX	US_412
10/06/2008	DFW->ORD	UA_1104	ORD->DFW	OO_6119
01/01/2008	LAX->ORD	UA_944	ORD->JFK	B6_918

Optimizations and efficiency:

1. Using saltstack to automate system management really saved a lot of time.
2. The processed data was about the right size without compromising accuracy of the expected output.
3. The decision to optimize data based on the year is to optimize HDFS reads and writes. HDFS is built for large block sizes, in the order of 64MB. Dealing with small files will make the reads very inefficient.

Summary:

I verified the results based on my intuition and data. For all the questions from Group 2, I had the output of MR to compare against Cassandra. I struggled a bit with PIG scripts as the JOIN operation took a lot of time. But sorting the columns in question made it slightly better.

Video Link – https://www.youtube.com/watch?v=bx9GQ__XPW0

GitHub Link – https://github.com/karthikBG/AviationAnalytics