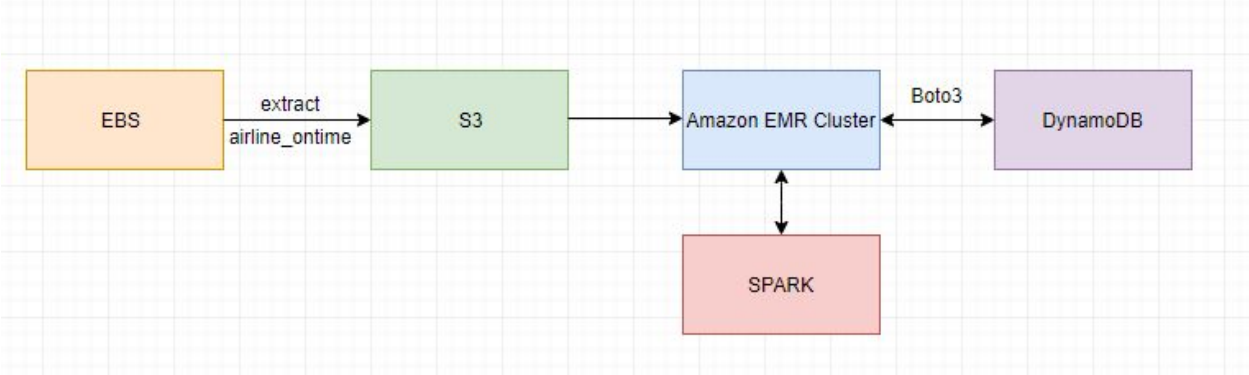
Overall Design



- As explained in the Project Overview, we have an option of choosing Apache hadoop or Spark in Task 1 for batch processing and I have gone with Spark owing to my familiarity with Python and Spark being more popular these days. I plan to solve Task 2 using kafka + spark streaming.

> You will be answering a common set of questions in different stacks/systems – in two batch processing systems (Apache Hadoop and Spark), and in a stream processing system (Apache Storm). After completing Task 1, you will be able to use the results you obtained to verify the results from Task 2.

- Secondly, Spark batch processing is faster than hadoop map-reduce because Spark can do in memory processing while Hadoop map-reduce has to read from and write to disk

Data Extraction

EBS Volume is mounted on EC2 and then only the required folder (airline_ontime) is unzipped to S3.

```
./ebs_to_s3.sh $EBS_MOUNTED_FOLDER $S3_BUCKET
```

Source
Extraction Script

Data Filtering and Exploration

All the data filtering is done in map-reduce using Spark filtering and relevant information moved to DynamoDB for querying later.

Problem 1.1

Strategy:

1. map() each airport_id to (airport_id,1)
2. reduce() count for each airport_id to get (airport_id,count)
3. takeOrdered() to filter out top 10 airports with highest counts

```
spark-submit --executor-memory 5g g1_1.py
```

Source:
g1_1.py

Results:

Format: (AirportID, Count)

```
(u'"ORD"', 12446097)
(u'"ATL"', 11537401)
(u'"DFW"', 10795494)
(u'"LAX"', 7721141)
(u'"PHX"', 6582467)
(u'"DEN"', 6270420)
(u'"DTW"', 5635421)
(u'"IAH"', 5478257)
(u'"MSP"', 5197649)
(u'"SFO"', 5168898)
```

Problem 1.2

Strategy

1) Filter for all non cancelled flights and map (flightID, (ArrivalDelay,1))
2) Get (flightID, (totalDelay,totalCount))
3) Get (flightID, totalDelay/totalCount)
4) Filter top 10 flights with best Arrival delay

Source:
g1_2.py

Results:

Format: FlightID [totalDepDelay, totalCount, AvgArrivalDelay]

```
"HA" [-264258.0, 261175, -1.01180434574519]
"AQ" [175282.0, 151507, 1.1569234424812056]
"PS" [60319.0, 41581, 1.4506385127822803]
```

"ML (1)" [328150.0, 69119, 4.747609195734892]
"PA (1)" [1570649.0, 293971, 5.342870555258852]
"F9" [1751640.0, 320468, 5.465881148819851]
"NW" [52626818.0, 9468157, 5.558295875321882]
"WN" [82237109.0, 14794127, 5.558767272986098]
"OO" [16842083.0, 2936047, 5.736312463662878]
"9E" [2713127.0, 462424, 5.8671846616957595]

Problem 1.3

Strategy

is same as Problem 1.2 except that key here is dayofWeek instead of the airportD.

Source:

g1_3.py

Results:

Format: Day [totalDepDelay, totalCount, AvgArrivalDelay]

Saturday [63333842.0, 14730888, 4.299390640944388]
Tuesday [99675613.0, 16633852, 5.9923349684727265]
Sunday [105166747.0, 15890557, 6.6181913572947755]
Monday [112546509.0, 16744616, 6.721355031372473]
Wednesday [119959179.0, 16672977, 7.194826634739555]
Thursday [151497938.0, 16672609, 9.086636530611376]
Friday [162539053.0, 16731254, 9.714696399923161]

Problem 2.1

(For each origin airport X, rank top 10 carriers in decreasing order of ontime departure)

Strategy:

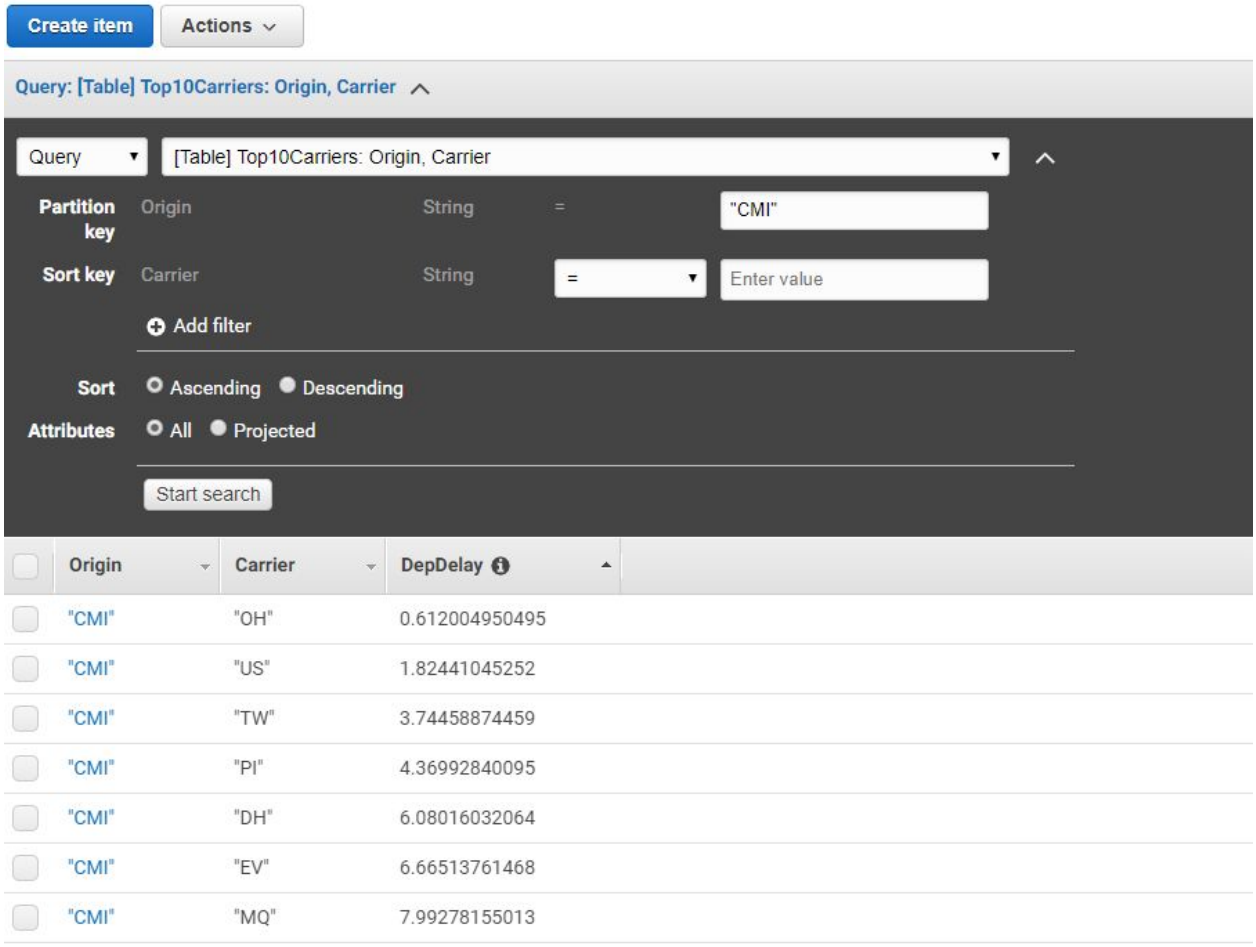
1) Filter for all non cancelled flights and map ((OriginAirport,carrier), (DepDelay,1))
2) Get ((OriginAirpor,carrier), (totalDepDelay, totalCount))
3) Get ((OriginAirport,carrier), AvgDepDelay)
4) For each OriginAirport, filter 10 best carriers in terms of Dep delay
5) Store results in dynamoDB
    - Main DB has OriginAirport as Partition key and Carrier as Sort key
    - Also created a Secondary index mapped to main DB with Partition key as OriginAirport and sort key as DepDelay. Hence querying the Secondary index returns results sorted in increasing order of DepDelay.

Source:

g2_1.py (filters top 10 carriers for each airports and puts them in DynamoDB
g2_1_query.py (To query the DynamoDB for the airports listed in Task1 Queries)

Results:

Sample snippet from dynamoDB for Airport CMI.



Sample Output obtained from executing the g2_1_query.py :

Format: {Origin: <>, Carrier: <> , Depdelay: <>}
Top 10 carriers from Airport CMI with best departure delays are:
{u'Origin': u'"CMI"', u'Carrier': u'"OH"', u'DepDelay': Decimal('0.612004950495')}
{u'Origin': u'"CMI"', u'Carrier': u'"US"', u'DepDelay': Decimal('1.82441045252')}
{u'Origin': u'"CMI"', u'Carrier': u'"TW"', u'DepDelay': Decimal('3.74458874459')}
{u'Origin': u'"CMI"', u'Carrier': u'"PI"', u'DepDelay': Decimal('4.36992840095')}
{u'Origin': u'"CMI"', u'Carrier': u'"DH"', u'DepDelay': Decimal('6.08016032064')}

{u'Origin': u'"CMI"', u'Carrier': u'"EV"', u'DepDelay': Decimal('6.66513761468')}
{u'Origin': u'"CMI"', u'Carrier': u'"MQ"', u'DepDelay': Decimal('7.99278155013')}

Results for all the airports in Task1 Queries section is in
g2_1_output

Problem 2.2
Strategy:
Same as problem 2.1 except replace carrier with DestAirport

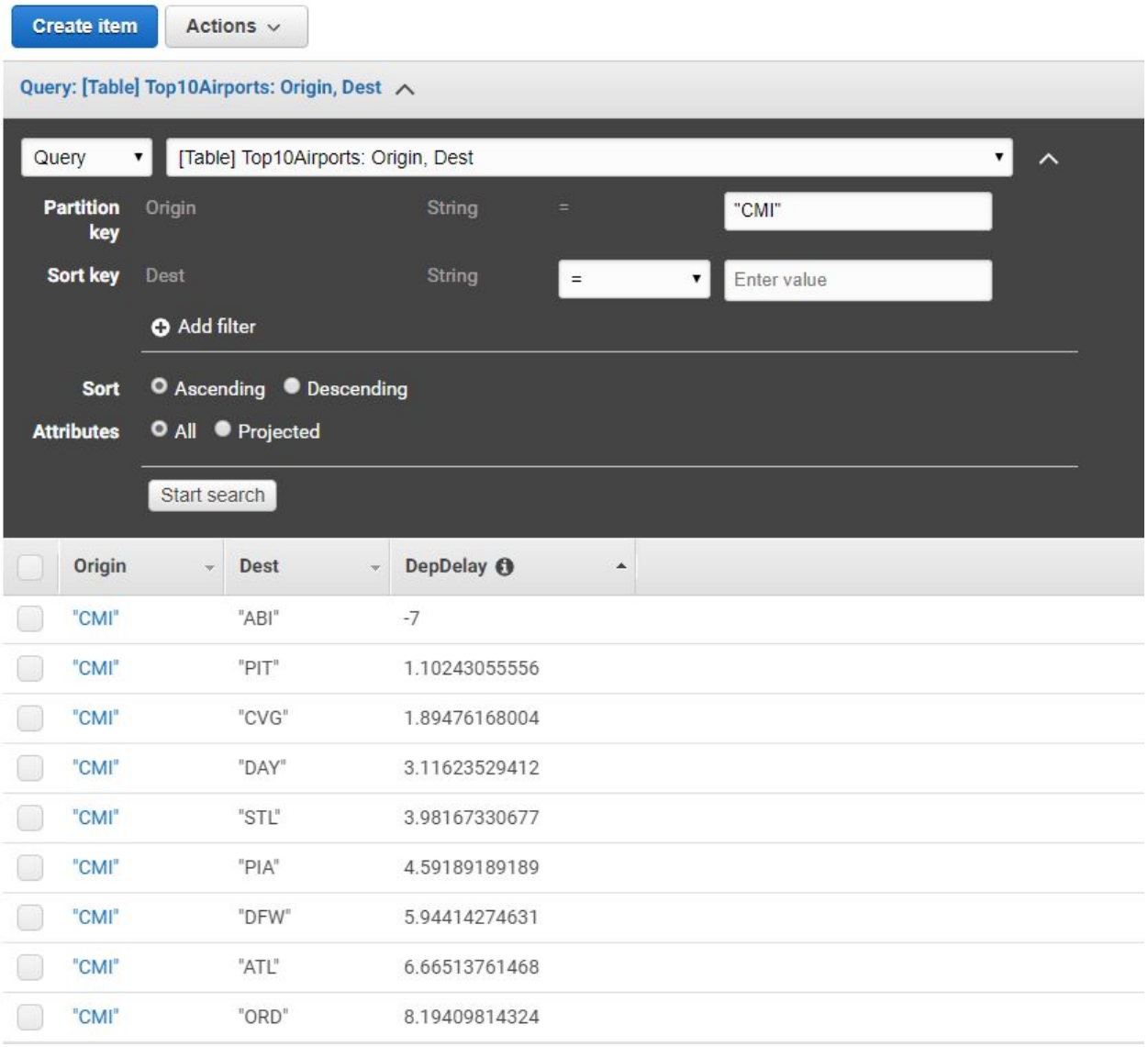
Source:
g2_2.py
g2_2_query.py

Result:

Snippet of output from query passed to dynamoDB through g2_2_query.py

Top 10 airports from Airport CMI with best departure delays are:
{u'Dest': u'"ABI"', u'Origin': u'"CMI"', u'DepDelay': Decimal('-7')}
{u'Dest': u'"PIT"', u'Origin': u'"CMI"', u'DepDelay': Decimal('1.10243055556')}
{u'Dest': u'"CVG"', u'Origin': u'"CMI"', u'DepDelay': Decimal('1.89476168004')}
{u'Dest': u'"DAY"', u'Origin': u'"CMI"', u'DepDelay': Decimal('3.11623529412')}
{u'Dest': u'"STL"', u'Origin': u'"CMI"', u'DepDelay': Decimal('3.98167330677')}
{u'Dest': u'"PIA"', u'Origin': u'"CMI"', u'DepDelay': Decimal('4.59189189189')}
{u'Dest': u'"DFW"', u'Origin': u'"CMI"', u'DepDelay': Decimal('5.94414274631')}
{u'Dest': u'"ATL"', u'Origin': u'"CMI"', u'DepDelay': Decimal('6.66513761468')}
{u'Dest': u'"ORD"', u'Origin': u'"CMI"', u'DepDelay': Decimal('8.19409814324')}

Sample snippet of query for OriginAirport “CMI” directly in dynamoDB GUI



Problem 2.4

(determine the mean arrival delay (in minutes) for a flight from X to Y)

Strategy:

1. Filter for all non cancelled flights and map (origin,dest) -> (ArrDelay,1)
2. Get (origin, dest) -> (TotalArrDelay, TotalCount)
3. Get (origin,dest) -> AvgArrDelay
4. Save to DB

Source:

g2_4.py

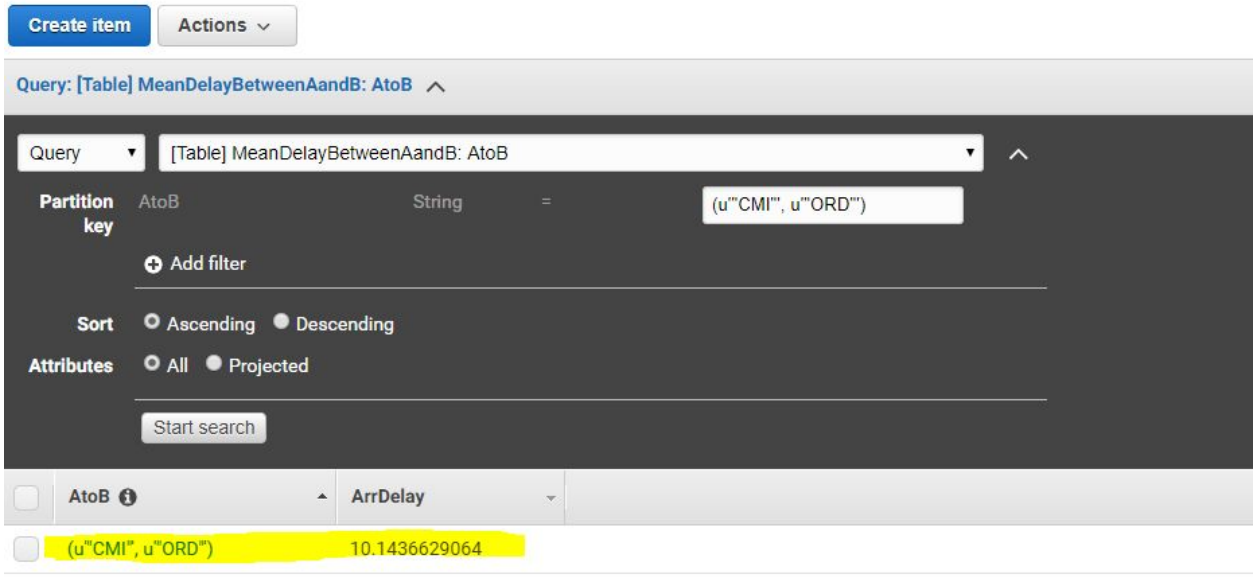
g2_4_query.py

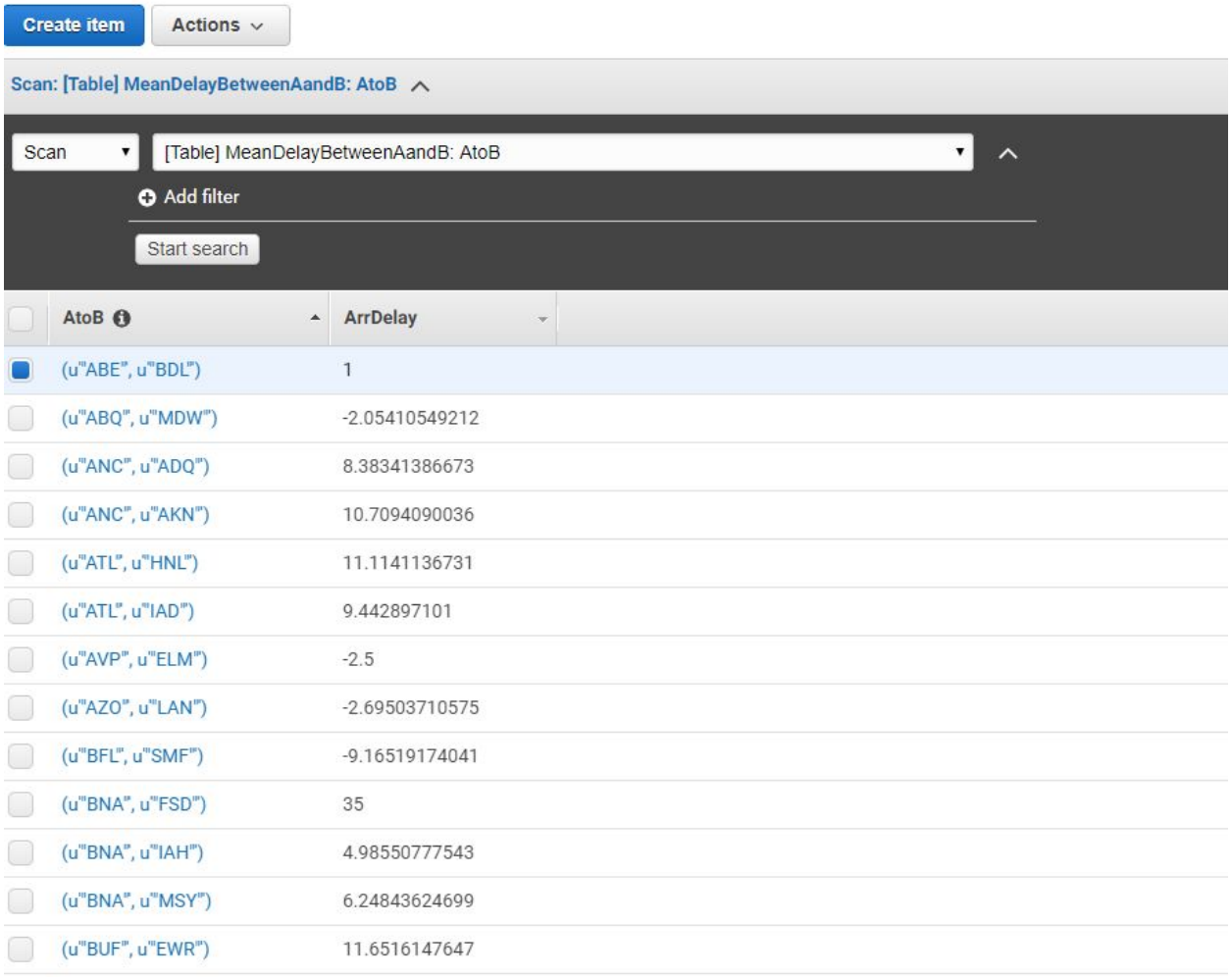
Result:

Format: { ‘AtoB’: (<origin>, <dest>) , ‘ArrDelay’ : <delay value> }

Mean average delay for source-destination (u'"CMI"', u'"ORD"') is :
{u'AtoB': u'(u\'"CMI"\', u\'"ORD"\')', u'ArrDelay': Decimal('10.1436629064')}
Mean average delay for source-destination (u'"IND"', u'"CMH"') is :
{u'AtoB': u'(u\'"IND"\', u\'"CMH"\')', u'ArrDelay': Decimal('2.89113670506')}
Mean average delay for source-destination (u'"DFW"', u'"IAH"') is :
{u'AtoB': u'(u\'"DFW"\', u\'"IAH"\')', u'ArrDelay': Decimal('7.61733279859')}
Mean average delay for source-destination (u'"LAX"', u'"SFO"') is :
{u'AtoB': u'(u\'"LAX"\', u\'"SFO"\')', u'ArrDelay': Decimal('9.58928273111')}
Mean average delay for source-destination (u'"JFK"', u'"LAX"') is :
{u'AtoB': u'(u\'"JFK"\', u\'"LAX"\')', u'ArrDelay': Decimal('6.63511915527')}
Mean average delay for source-destination (u'"ATL"', u'"PHX"') is :
{u'AtoB': u'(u\'"ATL"\', u\'"PHX"\')', u'ArrDelay': Decimal('9.02134188151')}

Arrival delay of some origin-dest pairs from dynamoDB





AtoB	ArrDelay
(u'"ABE", u'"BDL")	1
(u'"ABQ", u'"MDW")	-2.05410549212
(u'"ANC", u'"ADQ")	8.38341386673
(u'"ANC", u'"AKN")	10.7094090036
(u'"ATL", u'"HNL")	11.1141136731
(u'"ATL", u'"IAD")	9.442897101
(u'"AVP", u'"ELM")	-2.5
(u'"AZO", u'"LAN")	-2.69503710575
(u'"BFL", u'"SMF")	-9.16519174041
(u'"BNA", u'"FSD")	35
(u'"BNA", u'"IAH")	4.98550777543
(u'"BNA", u'"MSY")	6.24843624699
(u'"BUF", u'"EWR")	11.6516147647

Problem 3.1

Strategy:

1) Collected ranking of all airports into a file using Spark
2) Used powerlaw library to plot the CCDF of the power law distribution and the lognormal distribution of the data collected in 1)

Source:

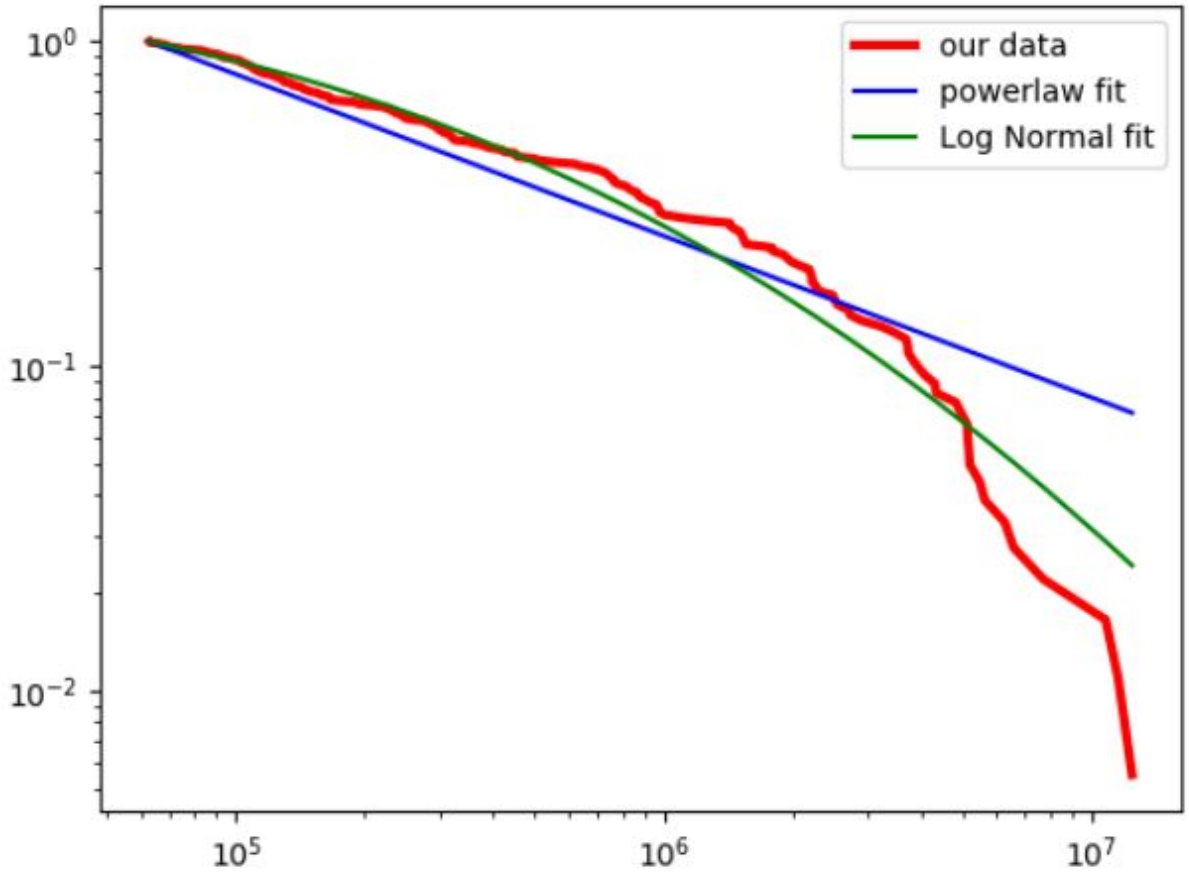
g3_1.py

pythonplot.py

Result:

g3_1.log (Ranking of airports with their count)

Our data resembles the log Normal fit better than the powerlaw fit. We can conclude that the popularity of airports doesn't follow Zipf distribution



Problem 3.2

Strategy:

1. Filter non cancelled flights
2. Create RDD for all flights which fly before noon and dest as Y. key is (date, Y) -> (flight info)
3. Create RDD for all flights which fly after noon with origin as Y and departure date subtracted by 2 days
4. Join is done on (date,Y) as key. This gives all flights landing in Y before noon and all flights departing from Y two days later
5. For X->Y->Z route, filter the flight combo with minimum arrival delay
6. Write data to DynamoDB. Since a huge set of data was to be written, increased the write capacity of DynamoDB to 1500 to transfer the data faster.

Source:

g3_2_final.py (generates results with Spark and moves it dynamoDB)

g3_2_query.py (To query dynamoDB)

Result:

The results for all task1 queries are in

Snippet of the result for query ['"CMI"-"ORD"-"LAX"', '2008-03-04'] is as follows

['"CMI"-"ORD"-"LAX"', '2008-03-04']
Shortest Arrival Delay is -38
Journey first leg in below order
origin, dest, flight, flight number, deptime, Arrival delay
((u'CMI', u'ORD', u'MQ', u'4278', u'0710', -14.0))
Journey Second leg in below order
origin, dest, flight, flight number, deptime, Arrival delay
((u'ORD', u'LAX', u'AA', u'607', u'1950', -24.0))

Snippet from query in dynamoDB

Format: XYZ | StartDate | TotalArrDelay | Info
info[0] is first leg of the journey and info[1] is second list of the journey.
info[0] and info[1] are in the format: origin, dest, flight, flight number, deptime, Arrival delay for that leg of the journey

-