Data Extraction, Batch Processing with Spark Architecture for HW1

Wyatt Melin: wmelin2@illinois.ede

I. Abstract

Big Data is rapidly growing and cloud computing model is the only go to fit to process all of it in a timely mannder. Cloud computing service provides the perfect framework for traditional for main stream open source Big Data software like Spark, Hadoop and HDFS to operate. Since the EBS transportation snapshot is restricted to the us-east-1 (N. Virgina) region, cleaning the data and running spark streaming jobs were developed accordingly in the same region for performance and cost. Cleaning the data with custom python software pandas ahead of time in an EC2 instance will reduce the amount of steps in N time spark will take for analysis.

II. Cleaning the Data

To get started with analyzing the data, first step is to attach the EBS volume to an EC2 instance. In the best interest of performance, the EC2 instance should be launched in the same region as the volume.

III. Explore the data

Once the EC2 is running and attached, a second writable storage media is required to explore the aviation transport data. In my case, I decided to create a python to loop through the aviation directory and upload only csv files (excluding the noise from other file formats) to HDFS. Once in HDFS, Spark would take over and pull the files to the EMR cluster. It was to my surprise that unzipping the data was about 50GB! Because of this, I made sure I added 250 GB writeable storage to my EMR cluster.

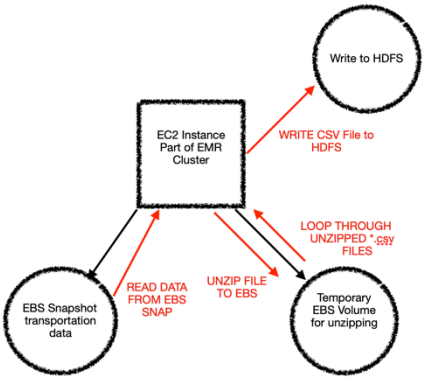


Figure 1 Demonstrates how files were both read and cleaned and finally forwarded onto HDFS to setup download for further processing by Spark and HDFS.

IV. HDFS+Spark

Using Spark is known for its in memory performance. Because of this software implementation, it offers lower application performances. However, not everything can fit into one machine. Especially when it comes to loading, storing and processing this large amount of data. There's a very limited amount of space in the L1, L2 cache of processors, not all data can fit inside this cache. Most instructions will be fed out to RAM once read from disk. At worst case scenario, the RAM will run out of space and revert to page memory/swap space. This can significant implications for overall read/write performance.

V. AWS architecture and desgin

To avoid the single machine solution, EMR is one solution AWS offers that is an easy end to end solution for

Big data workloads. It's very easy to setup a Spark cluster, configure master nodes and worker nodes for scaling. In the advance configuration, AWS gives the ability to configure the worker and master nodes on spot instances. In order to harness multiple worker nodes using the spot instances at once, I had to file a support ticket with AWS customer support to increase my concurrent spot instance machines quota. They increased my spot instance quota to 250 machines. [1]

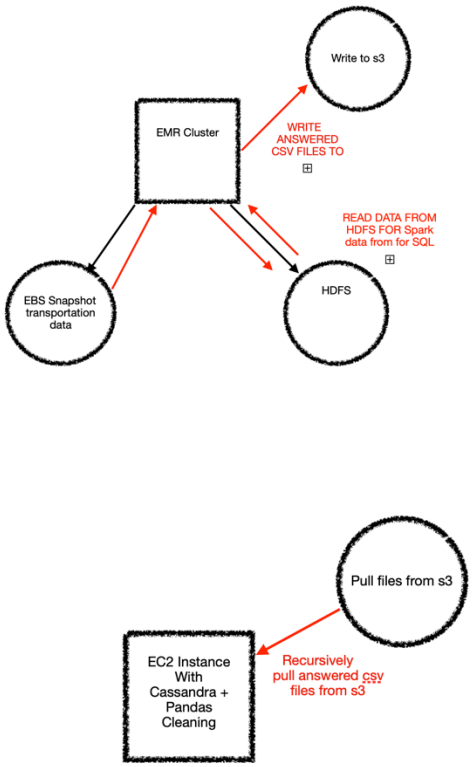


Figure 2 Demonstrates the bulk query analysis in AWS EMR and Casandra data ingestion from results posted from s3

Once processing is done on EMR cluster, I then normalized the results into one CSV per each question answered. In other words, once I have answered all the questions I then upload the results to s3. Once in s3, I then spined up an EC2 instance with Cassandra installed. Cassandra's csql has a feature that can import CSV files into the db. However, there's some caveat to this approach. Importing via csql infers that the data is already formatted with a primary key. Therefore, there's some pre-processing required in pandas to insert a unique identifier in each record before it can be handed over to csql for file import.

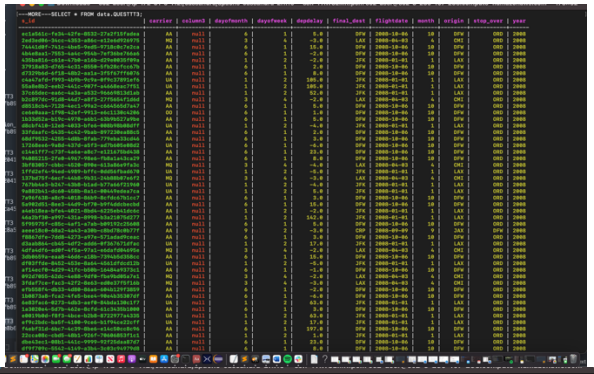
VI. HOME WORK QUESTIONS – DATA RETRVAL SPARK SQL TASK 1

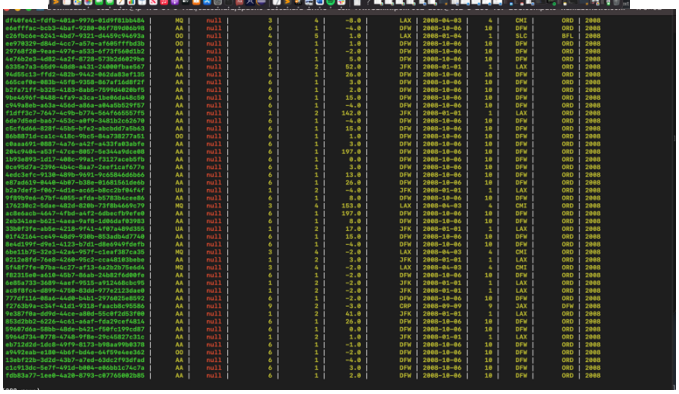
- 1.2 - Rank the top 10 airline by on-time arrival performance
    - Query: ***sqlContext.sql("SELECT COUNT(Carrier) as count,Carrier,ArrDelay FROM airports WHERE Cancelled != 1.00 AND ArrDelay IS NOT NULL GROUP BY Carrier,ArrDelay ORDER BY ArrDelay,COUNT(Carrier) DESC LIMIT 10")***
- 1.3 - Rank the days of the week by on-time arrival performance.
    - Query:***SqlContext.sql("SELECT COUNT(DayOfWeek) as count,DayOfWeek,ArrDelay FROM airports WHERE Cancelled != 1.00 AND ArrDelay IS NOT NULL GROUP BY DayOfWeek,ArrDelay ORDER BY ArrDelay,COUNT(DayOfWeek) DESC")***
- 2.1 – For each airport X, rank the top-10 carriers in decreasing order of on-time departure performance from X.
    - ***sqlContext.sql("SELECT COUNT(Carrier) as count,DepDelay,ORIGIN FROM airports WHERE Cancelled != 1.00 AND DepDelay IS NOT NULL AND ORIGIN IN ('CMI','BWI','MIA','LAX','IAH','SFO') GROUP BY Carrier,DepDelay,ORIGIN ORDER BY DepDelay ASC LIMIT 10")***
- 2.2 – For each source airport X, rank the top-10 destination airports in decreasing order of on-time departure performance from X.
    - ***sqlContext.sql("SELECT COUNT(Dest) as count,DepDelay,ORIGIN FROM airports WHERE Cancelled != 1.00 AND DepDelay IS NOT NULL AND ORIGIN IN ('CMI','BWI','MIA','LAX','IAH','SFO') GROUP BY Dest,DepDelay,ORIGIN ORDER BY DepDelay ASC LIMIT 10")***
- 2.3 – For each source-destination pair X-Y, rank the top-10 carries in decreasing order of on-time arrival performance at Y from X
    - ***teenagers = sqlContext.sql("WITH CTE_L AS (SELECT Carrier,DepDelay FROM airports WHERE Origin='CMI' AND Dest='ORD' AND Cancelled != 1.00 AND DepDelay IS NOT NULL GROUP BY Carrier,DepDelay UNION ALL SELECT Carrier,DepDelay FROM airports WHERE Origin='IND' AND Dest='CMH' AND Cancelled != 1.00 AND DepDelay IS NOT NULL GROUP BY Carrier,DepDelay UNION ALL SELECT Carrier,DepDelay FROM airports WHERE Origin='DFW' AND Dest='IAH' AND Cancelled != 1.00 AND DepDelay IS NOT NULL GROUP BY Carrier,DepDelay UNION ALL SELECT Carrier,DepDelay FROM airports WHERE Origin='LAX' AND Dest='SFO' AND Cancelled != 1.00 AND DepDelay IS NOT NULL GROUP BY Carrier,DepDelay UNION ALL SELECT***

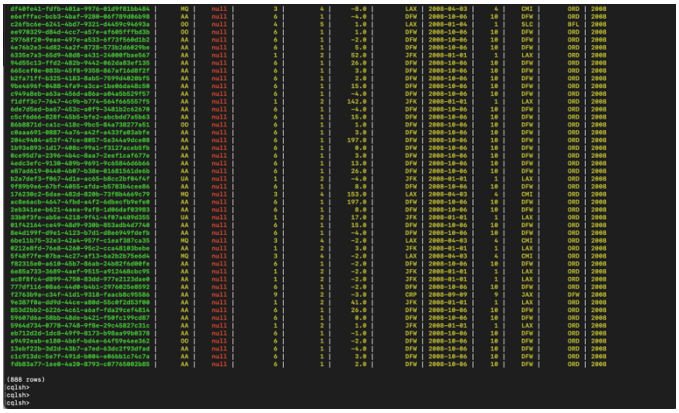
Carrier,DepDelay FROM airports WHERE Origin='JFK' AND Dest='LAX' AND Cancelled != 1.00 AND DepDelay IS NOT NULL GROUP BY Carrier,DepDelay UNION ALL SELECT Carrier,DepDelay FROM airports WHERE Origin='ATL' AND Dest='PHX' AND Cancelled != 1.00 AND DepDelay IS NOT NULL GROUP BY Carrier,DepDelay) SELECT * FROM CTE_L ORDER BY CTE_L.DepDelay LIMIT 10")

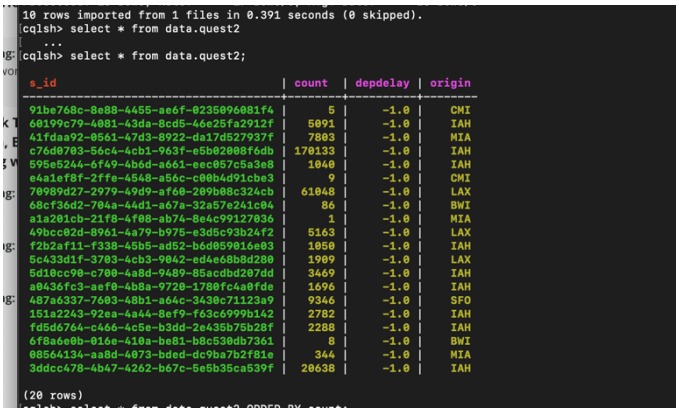
- Question 3.2
  - CMI → ORD → LAX, 04/03/2008
    - ***sqlContext.sql("WITH CTE_L AS (SELECT CMI.Carrier AS Carrier,CMI.DepDelay AS DepDelay,CMI.FlightDate AS FlightDate,CMI.DayOfWeek AS DayOfWeek,CMI.DayofMonth AS DayofMonth,CMI.Month AS Month,CMI.Year AS Year,CMI.Dest as STOP_OVER,CMI.Origin AS Origin,ORD.Dest AS ORIGIN FROM airports AS ORD JOIN (SELECT Carrier,DepDelay,FlightDate,DayOfWeek, DayofMonth,Month,Year,Dest,Origin FROM airports WHERE Origin='CMI' AND Cancelled != 1.00 AND DepDelay IS NOT NULL AND FlightDate = '2008-04-03') AS CMI ON CMI.Dest = ORD.Origin WHERE ORD.Origin='ORD' AND ORD.Dest='LAX' AND ORD.Cancelled != 1.00 AND ORD.DepDelay IS NOT NULL AND ORD.FlightDate = '2008-04-03') SELECT * FROM CTE_L")***
  - JAX → DFW → CRP, 09/09/2008
    - ***sqlContext.sql("WITH CTE_L AS (SELECT JAX.Carrier,JAX.DepDelay,JAX.FlightDate,JAX.DayOfWeek,JAX.DayofMonth,JAX.Month,JAX.Year,JAX.Dest as STOP_OVER,JAX.Origin,DFW.Dest as FINAL_DEST FROM airports AS DFW JOIN (SELECT Carrier,DepDelay,FlightDate,DayOfWeek, DayofMonth,Month,Year,Dest,Origin FROM airports WHERE Origin='JAX' AND Cancelled != 1.00 AND DepDelay IS NOT NULL AND FlightDate = '2008-09-09') AS JAX ON JAX.Dest = DFW.Origin WHERE DFW.Origin='DFW' AND DFW.Dest='CRP' AND DFW.Cancelled != 1.00 AND DFW.DepDelay IS NOT NULL AND DFW.FlightDate = '2008-09-09') SELECT * FROM CTE_L")***
  - SLC → BFL → LAX, 01/04/2008
    - ***sqlContext.sql("WITH CTE_L AS (SELECT SLC.Carrier,SLC.DepDelay,SLC.FlightDate,SLC.DayOfWeek,SLC.DayofMonth,SLC.Month,SLC.Year,SLC.Dest as STOP_OVER,SLC.Origin,BFL.Dest as FINAL_DEST FROM airports AS BFL JOIN (SELECT Carrier,DepDelay,FlightDate,DayOfWeek, DayofMonth,Month,Year,Dest,Origin FROM airports WHERE Origin='SLC' AND Cancelled != 1.00 AND DepDelay IS NOT NULL AND FlightDate = '2008-01-04') AS SLC ON SLC.Dest = BFL.Origin WHERE BFL.Origin='BFL' AND BFL.Dest='LAX' AND BFL.Cancelled != 1.00 AND BFL.DepDelay IS NOT NULL AND BFL.FlightDate = '2008-01-04') SELECT * FROM CTE_L")***
  - LAX → SFO → PHX, 12/07/2008
    - ***sqlContext.sql("WITH CTE_L AS (SELECT LAX.Carrier,LAX.DepDelay,LAX.FlightDate,LAX.DayOfWeek,LAX.DayofMonth,LAX.Month,LAX.Year,LAX.Dest as STOP_OVER,LAX.Origin,SFO.Dest as FINAL_DEST FROM airports AS SFO JOIN (SELECT Carrier,DepDelay,FlightDate,DayOfWeek, DayofMonth,Month,Year,Dest,Origin FROM airports WHERE Origin='LAX' AND Cancelled != 1.00 AND DepDelay IS NOT NULL AND FlightDate = '2008-12-08') AS LAX ON LAX.Dest = SFO.Origin WHERE SFO.Origin='SFO' AND SFO.Dest='PHX' AND SFO.Cancelled != 1.00 AND SFO.DepDelay IS NOT NULL AND SFO.FlightDate = '2008-12-08') SELECT * FROM CTE_L")***
  - DFW → ORD → DFW, 10/06/2008
    - ***sqlContext.sql("WITH CTE_L AS (SELECT DFW.Carrier,DFW.DepDelay,DFW.FlightDate,DFW.DayOfWeek,DFW.DayofMonth, DFW.Month,DFW.Year,DFW.Dest as STOP_OVER,DFW.Origin,ORD.Dest as FINAL_DEST FROM airports AS ORD JOIN (SELECT Carrier,DepDelay,FlightDate,DayOfWeek, DayofMonth,Month,Year,Dest,Origin FROM airports WHERE Origin='DFW' AND Cancelled != 1.00 AND DepDelay IS NOT NULL AND FlightDate = '2008-10-06') AS DFW ON DFW.Dest = ORD.Origin WHERE ORD.Origin='ORD' AND ORD.Dest='DFW' AND ORD.Cancelled != 1.00 AND ORD.DepDelay IS NOT NULL AND ORD.FlightDate = '2008-10-06') SELECT * FROM CTE_L")***
  - LAX → ORD → JFK, 01/01/2008
    - ***sqlContext.sql("WITH CTE_L AS (SELECT LAX.Carrier,LAX.DepDelay,LAX.FlightDate,LAX.DayOfWeek,LAX.DayofMonth,LAX.Month,LAX.Year,LAX.Dest as STOP_OVER,LAX.Origin,ORD.Dest as FINAL_DEST FROM airports AS ORD JOIN (SELECT Carrier,DepDelay,FlightDate,DayOfWeek, DayofMonth,Month,Year,Dest,Origin FROM airports WHERE Origin='LAX' AND Cancelled != 1.00 AND DepDelay IS NOT NULL AND FlightDate = '2008-01-01') AS LAX ON LAX.Dest = ORD.Origin WHERE ORD.Origin='ORD' AND ORD.Dest='JFK' AND ORD.Cancelled != 1.00 AND ORD.DepDelay IS NOT NULL AND ORD.FlightDate = '2008-01-01') SELECT * FROM CTE_L")***

VII. HOME WORK QUESTIONS – CASSANDRA, TASK 1 QUESTIONS 3.2,2.1,2.2,2.3









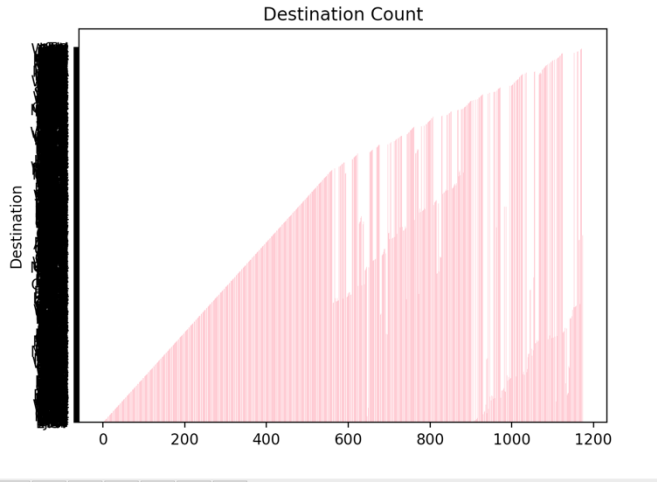
```
Starting copy of data.quest21 with columns ['carrier', 'depdelay', 's_id'].
Processed: 10 rows; Rate:       14 rows/s; Avg. rate:       22 rows/s
10 rows imported from 1 files in 0.451 seconds (0 skipped).
cqlsh> select * from data.QUEST21;

 s_id                                 | carrier | depdelay
--------------------------------------+---------+----------
 4e2fd91a-0333-4e44-a5ba-7a19b53ad8cb |      HP |     -1.0
 c7572cde-da40-48e6-b836-e9085aacf4d3 |      WN |     -1.0
 282501ee-2d35-42a2-941f-8e887d73bff6 |      DL |     -1.0
 2f73df02-aaa0-4744-91d2-db44e2019520 |      DL |     -1.0
 604cad2d-09af-4134-93f4-bd9c4a8ad5e0 |      MQ |     -1.0
 aea438b1-c4e6-4051-8045-037da2950ef7 |      OO |     -1.0
 0f84ac0e-3fac-4200-943c-9ab2be8a3269 |      CO |     -1.0
 c7e91157-3485-42dd-974f-e9947999ba4a |      MQ |     -1.0
 1c5628a6-6732-462e-a350-edd29d304242 |      F9 |     -1.0
 7e8d7b2a-7db5-40be-a4d0-73b5792418c4 |      NW |     -1.0

(10 rows)
```

VIII. HOME WORK QUESTIONS – TASK 1 QUESTIONS 1.2, 1.3

1.2 Results – Top 10		
Count	*Carrier*	*ArrDelay*
536220	DL	-1
464211	WN	-1
445166	US	-1
420117	AA	-1
396823	UA	-1
282005	NW	-1
221618	CO	-1
123695	HP	-1
110624	TW	-1
91606	OO	-1

1.3 Results – Top 14		
Count	*DayOfWeek*	*ArrDelay*
538950	1	-1
537849	2	-1
529760	3	-1
515826	4	-1
509155	5	-1
493765	7	-1
461557	6	-1
471416	2	-10
464469	1	-10
449941	6	-10
449026	3	-10
444732	7	-10
409823	4	-10
399515	5	-10

As per my analysis of distribution, I ran a spark sql query to group by destination and count the number of times a particular destination was selected across the whole data set. Since there were so many different destinations, I had a tough time labeling in matplotlib on the y axis. Please find below results of the distribution of popular destinations:



REFERENCES

[1] https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-spot-limits.html