Installation and integration

1. Create three EC2 instances from Centos 7 image.
2. Install Hortonworks Data Platform (HDP).
3. Create HDP cluster. Include Zookeeper, HDFS, Yarn, MapReduce services.
4. Install Cassandra separately, from apache yum repository.

Steps 2 and 3 were performed as per the documentation at https://hortonworks.com/blog/deploying-hadoop-cluster-amazon-ec2-hortonworks/. HDP nicely integrates Hadoop, HDFS, Zookeeper, YARN, etc. Nothing additional is needed to make them work together. Cassandra, however, was installed separately using rpm package. The 'reduce' programs use the gocql API to talk to Cassandra using an AWS private LAN IP address.

Data Extraction

High level procedure

1. Create EBS volume based on transportation dataset snapshot id.
2. Attach the new volume to the node-1 instance.
3. A new device (/dev/xvdf in demo) is hotplugged into node-1 VM.
4. Mount the new device to a directory.
5. Use the "cleanup_data" program (described below) to extract the relevant data from dataset and create cleaned CSV files.
6. Compress the files using gzip.
7. Upload the gzip'd csv files to HDFS using "hadoop fs -put" command.

"cleanup_data" program

This is a golang program developed for this project. It unzips and parses a set of CSV files, extracts a subset of fields, and outputs a new CSV file for each input CSV file.
The program takes two inputs - the input directory, and the output directory, both as command line flags. It recursively scans the input directory looking for "*.zip" files. Each of these is converted to a new CSV file in the output directory. The fields extracted are:
Year,Month,DayofMonth,DayOfWeek,UniqueCarrier,Origin,Dest,CRSDepTime,DepDelay,ArrDelay,Cancelled,Diverted

Group 1 - Question 1

Map function

For this, and all subsequent map functions, an input record is one CSV line, having information about a single flight
Output: Two key-value pairs

1. Key: "Origin" field from CSV record. Value: 1
2. Key: "Dest" field from CSV record Value: 1

Example cleaned input: 1990,1,2,2,US,DCA,CLT,1855,26,25,0,0
Output: DCA 1
CLT 1

Reduce function
Sum all values for the key and output a single line for each key:
Key Sum-of-values
Thus, the reduce function would count the number of times an airport was either the source or the destination. The output of reduce function would be one line per airport. After running the hadoop job, the reducer output can be concatenated and sorted by “sort” command.

Result: ORD(12449354), ATL(11540422), DFW(10799303), LAX(7723596), PHX(6585534), DEN(6273787), DTW(5636622), IAH(5480734), MSP(5199213), SFO(5171023)

Group 1 Question 2

Map function
Output: One key-value pair.
Key: “UniqueCarrier” field from CSV record
Value: “ArrDelay” field from CSV record
Example cleaned input: 1990,1,2,2,US,DCA,CLT,1855,26,25,0,0
Output: US 25

Reduce function
For a given input key, it tracks the sum all the values seen for that key, and also the count of number of records seen for they key. After all records of a key are processed (they are all input to reducer before a different key is), the reducer calculates the mean value for that key, and prints a line (one per airline):
Key SumOfValues/CountOfRecords NumberOfRecords
Printing of “NumberOfRecords” is just for validation. After running the hadoop map-reduce job, the reducer output can be concatenated, and sorted by the Unix command “sort”.

Result: 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

Group 2 Question 1

Map function
Output: One key-value pair
Key: “Origin” field from CSV record
Value: <UniqueCarrier><TAB><DepDelay>. Both are fields from CSV record.
Example cleaned input: 1990,1,2,2,US,DCA,CLT,1855,26,25,0,0
Output: DCA US 26

Reduce function
Reduce function maintains a map (balanced binary tree) for every key.
Map key: string carrier_code
Value: struct { int64 num_flights, int64 sum_delays }

When the reduce function reads a new record, it checks whether it is of the same key (airport) as it was processing previously. If yes, it updates the above mentioned map based on carrier and delay time (tab separated in input).
If the new record has a different key, it updates Cassandra for the key being previously processed. Before that it copies the map elements into a list, and sorts them based on mean delay. The top 10 airlines from the sorted list are then updated into the cassandra table "airport_best_airlines" with CQL INSERT command.
Cassandra table schema: CREATE TABLE airport_best_airlines (apt ascii, rank int, airline ascii, avg_delay float, num_flights bigint, PRIMARY KEY(apt,rank));
"rank" is an integer from 1 to 10.
The best airlines for any particular airport can be later queried using **CQL query** like:
SELECT * FROM airport_best_airlines WHERE apt = 'CMI';

Result:
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 5.50, NW 5.71, AA 6.00, 9E 7.24, US 7.49, DL 7.68, UA 7.74
MIA: 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, 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
IAH: NW 3.56, PA (1) 3.98, PI 3.99, US 5.06, F9 5.55, AA 5.70, 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 6.30, DL 6.56, CO 7.08, US 7.53, TW 7.79

Group 2 Question 2

This solution has exactly the same logic as Group 2 Question 1. The map function outputs "Dest" field instead of "UniqueCarrier". And the reduce function just uses a different Cassandra table (and column name) to populate the result.
Cassandra table schema: CREATE TABLE airport_best_dest_airport (apt ascii, rank int, dest_airport ascii, avg_delay float, num_flights bigint, PRIMARY KEY(apt,rank));
Query example: SELECT * FROM airport_best_dest_airport WHERE apt = 'CMI';

Result:
CMI: ABI -7.0, PIT 1.10, 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 4.20, SJU 4.44, OAJ 4.47
MIA: SHV 0.0, BUF 1.0, SAN 1.71, SLC 2.5, 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
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 3.60, 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 3.30, SCK 4.0

Group 2 Question 4

Map function

Output: One key-value pair

Key: <Origin>_<Dest> (“Origin” and “Dest” fields with ‘_’ in between)

Value: <ArrDelay> field from CSV record.

Example cleaned input: 1990,1,2,2,US,DCA,CLT,1855,26,25,0,0

Output: DCA_CLT 25

Reduce function

The reduce function calculates the mean of all values belonging to the same key. After processing all records of the key, it updates a row in Cassandra.

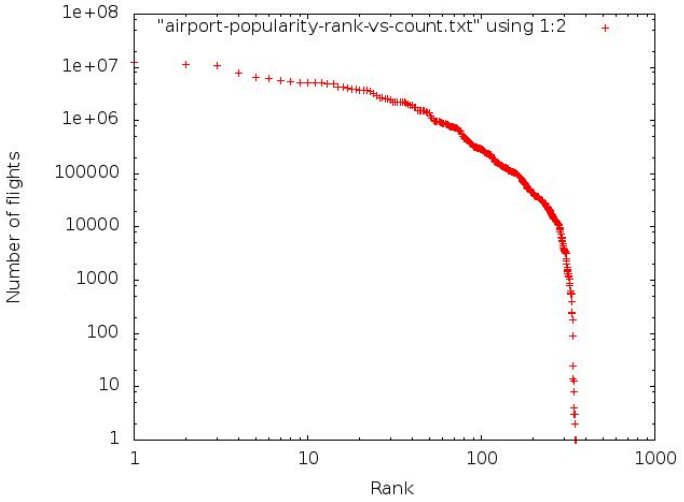
Cassandra table schema: CREATE TABLE src_dst_avg_delays (src ascii, dst ascii, avg_delay float, num_flights bigint, PRIMARY KEY(src, dst));

Example query: SELECT * FROM src_dst_avg_delays WHERE src = 'CMI' AND dst = 'ORD';

Result: CMI->ORD 10.14, IND->CMH 2.90, DFW->IAH 7.65, LAX->SFO 9.59, JFK->LAX 6.64, ATL->PHX 9.02

Group 3 Question 1

The reducer output of Group 1 question 1 can be used to answer this question. The output has the number of flights for each airport. It is sorted using the Unix “sort -n -k 2” command and saved to an intermediate file. The result is then transformed from “<airport> <num_flights>” to “<rank> <num_flights>” using an awk script, the rank simply being the line number in the sorted file. The result is then plotted using gnuplot as a log-log <rank>-<num_flights> graph.



Since the log-log graph is *not* a straight line, it is *not* a zipf distribution.

Group 3 Question 2

The key observation is that any query for X->Y->Z starting on date D can be satisfied by two independent queries:

1. What is the best flight from X->Y on date D in the morning (before noon).
2. What is the best flight from Y->Z on date (D+2) in the afternoon.

That guides the map function and the Cassandra table schema.

Map function

Output: One key-value pair

Key: <Origin>_<Dest>_<Date>_<morning> (Combination of four fields, three directly from CSV and “morning” being a boolean indicating whether it is flight before or after noon)

Value: <UniqueCarrier> <CRSDepTime> <ArrDelay> fields from CSV record, space separated.

Reduce function

For the input keys, it just picks the best flight (in terms of arrival delay) for the key. When it has processed all records for the key, it updates the best flight in the Cassandra table.

Cassandra table schema: CREATE TABLE src_dst_best_flights_of_day (src ascii, dst ascii, date timestamp, morning boolean, UniqueCarrier ascii, CRSDepTime ascii, ArrDelay int, PRIMARY KEY(src, dst, date, morning));

Example queries: SELECT * from src_dst_best_flights_of_day where src='CMI' and dst='ORD' and date='2008-03-04' and morning=true; SELECT * from src_dst_best_flights_of_day where src='ORD' and dst='LAX' and date='2008-03-06' and morning=false;

Result:

CMI -> ORD, 07:10 04/03/2008, delay: -14.0. ORD -> LAX, 19:50 06/03/2008, delay: -24.0
JAX -> DFW, 07:25 09/09/2008, delay: 1.0. DFW -> CRP, 16:45 11/09/2008, delay: -7.0
SLC -> BFL, 11:00 01/04/2008, delay: 12.0. BFL -> LAX, 14:55 03/04/2008, delay: 6.0
LAX -> SFO, 06:50 12/07/2008, delay: -13.0. SFO -> PHX, 19:25 14/07/2008, delay: -19.0
DFW -> ORD, 07:00 10/06/2008, delay: -21.0. ORD -> DFW, 16:45 12/06/2008, delay: -10.0
LAX -> ORD, 07:05 01/01/2008, delay: 1.0. ORD -> JFK, 19:00 03/01/2008, delay: -7.0

Optimizations

- Storing CSV data as gzip’d files. That saves a lot of space, and hence cost.
- Compressing the intermediate output between map and reduce tasks. That reduces the space requirement spike during the run, and also saves on network bandwidth.

Conclusion

MapReduce is a powerful, easy to use, and scalable framework for analyzing huge amounts of data. Observations derived from analysing such large data sets provides meaningful insights and reveals several previously unknown patterns, and is very useful for guiding decisions, not only in business, but also in consumer space.