Cloud computing capstone Task 1

Part 1

Setting up the environment

To perform the capstone tasks I choose Amazon Elastic Compute Cloud services, which enables us to use scalable and reliable infrastructure.

My architecture in Amazon Cloud consists of a cluster with three c3.xlarge instances, which are optimized for a computing purposes. As a server operation system, I choose Red Hat Linux Amazon Machine Images. After Amazon instances were up and running, I connected them into a network to create a cluster.

To manage Hadoop cluster and integrate it into Amazon environment I choose Ambari Hortonworks framework.

The process of creating a cluster and setupping Hadoop and Ambari Hortonworks server shown as series of screenshots on the video.

Extract data

The data for processing is provided as EBS Volume and I choose to mount this volume to my infrastructure as read-only volume. The are a lot of data on provided volume and we need to choose with what specific data we will work. I choose «airline_ontime» data set as it is providing us with all the data we needed. Also all data are zipped so I wrote a bash-script which recursively unzip all data sets for all years.

Clean data

To clean up the data I choose PIG, which is a platform for analyzing large data sets. As PIG is used on the top of Hadoop, I copied all the extracted data to Hadoop file system.

I prepared a PIG-script, which queries all the data sets and extract only the necessary data.

In screenshot in the video it is shown how PIG-script can be lunched from Ambari Hortonworks interface.

Part 2

Group 1

Rank the top 10 most popular airports by numbers of flights to/from the airport.

To perform this task I used MapReduce approach with two map-reduce jobs.

The first map prepares a list of flights from origin and destination airports. For each record in the dataset I write to context `Map(Origin,1), Map(Destination,1)`. Reduce job counts all flights from the airports.

`Map(Origin,1), Map(Destination,1) -> list(Airport,1)`

`Reduce(Airport, list (Airport,1)) → list(Airport, Count)`

The second map receives output from the first job and reduce produces a list of top 10 airports.

Results:

"ORD" 12446097
"ATL" 11537401
"DFW" 10795494
"LAX" 7721141
"PHX" 6582467
"DEN" 6270420
"DTW" 5635421
"IAH" 5478257
"MSP" 5197649
"SFO" 5168898

Rank the top 10 airlines by on-time arrival performance.

To perform this task I used MapReduce approach with two map-reduce jobs.

The first map prepares a list of airlines flights and its arrival delays in minutes. For each record in the dataset I write to context `Map(Airline,DelayInMinutes)`. Reduce job counts average arrival delay for all airline flights.

`Map(Airline, DelayInMinutes) -> list(Airline, DelayInMinutes)`

`Reduce(Airline, list (Airline, DelayInMinutes)) → list(Airline, AverageDelayInMinutes)`

The second map receives output from the first job and second reduce produces a list of top 10 airports.

Results:

"Hawaiian Airlines Inc.: HA"(19690) -1.0118043
"Aloha Airlines Inc.: AQ"(19678) 1.1569234
"Pacific Southwest Airlines: PS (1)"(19391) 1.4506385
"Midway Airlines Inc. (1): ML (1)"(20295) 4.747609
"Pan American World Airways (1): PA (1)"(20384) 5.3503227
"Frontier Airlines Inc.: F9"(20436) 5.4658813
"Northwest Airlines Inc.: NW"(19386) 5.5603867
"Southwest Airlines Co.: WN"(19393) 5.5804043
"SkyWest Airlines Inc.: OO"(20304) 5.7363167
"Endeavor Air Inc.: 9E"(20363) 5.8671846

Group 2

For each airport X, rank the top-10 carriers in decreasing order of on-time departure performance from X.

To perform this task I used MapReduce approach with two map-reduce jobs.

The first map prepares a list of composite keys of a departure airport and a carrier `{airport; carrier}` and its departure delays in minutes. For each record in the dataset, I write to the context `Map(DepartureAirport_Carrier,DepartureDelayInMinutes)`. The first reduce job counts average departure delay for this composite key and for each departure airport produces composite value of carrier and its average delay in minutes `{carrier; avg_delay}`.

`Map(DepartureAirport_Carrier, DepartureDelayInMinutes) -> list(DepartureAirport_Carrier, DepartureDelayInMinutes)`

`Reduce(DepartureAirport, list (DepartureAirport, Carrier_ AverageDelayInMinutes)) → list(DepartureAirport, Carrier_AverageDelayInMinutes)`

The second map receives output from the first job, splits the composite value and for each departure airport prepares the list of the carriers and its departure delays in minutes. The second reduce job for each departure airport produces top 10 carriers and its departure delay

`Map(DepartureAirport, Carrier_AverageDelayInMinutes) -> list(DepartureAirport, Carrier_AverageDelayInMinutes)`

`Reduce(DepartureAirport, list (DepartureAirport, Carrier_AverageDelayInMinutes)) → list(DepartureAirport, {Carrier, AverageDelayInMinutes})`

Results for individual queries:

"CMI" "US"(2.12min) "EV"(2.5min) "DH"(3.5min) "TW"(6.55min) "PI"(7.43min) "MQ"(9.83min)
"BWI" "PA"(2.3min) "DH"(2.59min) "NW"(4.71min) "AA"(4.81min) "CO"(5.1min) "WN"(5.83min) "HP"(6.16min) "US"(6.26min) "UA"(6.49min) "TW"(6.94min)
"MIA" "RU"(0.55min) "NW"(2.96min) "TZ"(4.43min) "PA"(4.53min) "XE"(4.87min) "US"(4.89min) "UA"(5.21min) "AS"(5.83min) "ML"(7.05min) "TW"(7.58min)
"LAX" "MQ"(1.45min) "EA"(2.21min) "HA"(2.32min) "NW"(2.57min) "OO"(2.83min) "FL"(3.27min) "TZ"(4.14min) "YV"(4.35min) "AA"(5.02min) "F9"(5.31min)
"IAH" "PA"(0.1min) "NW"(2.07min) "US"(3.51min) "OO"(3.94min) "F9"(4.7min) "RU"(4.8min) "WN"(5.34min) "9E"(5.73min) "MQ"(5.88min) "TW"(6.1min)
"SFO" "TZ"(2.56min) "NW"(3.11min) "PS"(3.4min) "EA"(3.51min) "MQ"(4.2min) "FL"(4.41min) "F9"(4.44min) "PA"(4.55min) "OH"(5.17min) "TW"(5.28min)

For each airport X, rank the top-10 airports in decreasing order of on-time departure performance from X.

To perform this task I used MapReduce approach with two map-reduce jobs.

The first map prepares a list of composite keys of a departure airport and a destination airport `{dep_airport; dest_airport}` and its departure delays in minutes. For each record in the dataset, I write to the context `Map(DepartureAirport_DestinationAirport,DepartureDelayInMinutes)`. The first reduce job counts average departure delay for this composite key and for each departure airport produces composite value of destination airport and its average delay in minutes `{dest_airport; avg_delay}`.

`Map(DepartureAirport_ DestinationAirport, DepartureDelayInMinutes) -> list(DepartureAirport_ DestinationAirport, DepartureDelayInMinutes)`

`Reduce(DepartureAirport, list (DepartureAirport, DestinationAirport _ AverageDelayInMinutes)) → list(DepartureAirport, DestinationAirport _AverageDelayInMinutes)`

The second map receives output from the first job, splits the composite value and for each departure airport prepares the list of the destination airports and its departure delays in minutes. The second reduce job for each departure airport produces top 10 destination airports and its departure delay

`Map(DepartureAirport, DestinationAirport _AverageDelayInMinutes) -> list(DepartureAirport, DestinationAirport _AverageDelayInMinutes)`

`Reduce(DepartureAirport, list (DepartureAirport, DestinationAirport _AverageDelayInMinutes)) → list(DepartureAirport, {DestinationAirport, AverageDelayInMinutes})`

Results for individual queries:

"CMI" "CVG"(0.26) "ATL"(2.5) "DAY"(3.07) "PIA"(4.35) "DFW"(6.13) "STL"(7.2) "ORD"(10.14) "PIT"(10.55)
"BWI" "ORD"(8.91) "MHT"(9.06) "AVL"(9.13) "ALB"(9.18) "BOS"(9.32) "OKC"(10.67) "MKE"(11.15) "EWR"(13.59) "JFK"(14.96) "LYH"(18.01)
"MIA" "CMH"(10.32) "DAY"(10.85) "CHS"(11.45) "LGA"(12.02) "ORD"(12.23) "SRQ"(12.37) "JFK"(13.92) "EWR"(14.31) "BOS"(14.66) "BUF"(28)
"LAX" "HNL"(8.46) "XNA"(9.57) "MFR"(9.59) "SFO"(9.59) "SDF"(15) "LGB"(22.33) "MAF"(23) "EGE"(23.56) "RDM"(29.56) "BUR"(97.33)
"IAH" "ATL"(8.56) "AMA"(8.76) "EGE"(9.01) "ANC"(9.37) "LAX"(9.75) "OGG"(9.85) "EWR"(11.4) "LGA"(11.42) "SFO"(11.56) "ORD"(11.91)
"SFO" "PSP"(12) "BDL"(12.4) "CEC"(14.23) "HDN"(14.51) "COS"(14.87) "LMT"(15.39) "SCK"(16.67) "BZN"(20.95) "OTH"(24.43) "EGE"(39.31)

For each source-destination pair X-Y, determine the mean arrival delay (in minutes) for a flight from X to Y.

To perform this task I used MapReduce approach.

The map prepares a list of composite key of a departure airport and a destination airport `{dep_airport; dest_airport}` and its departure delays in minutes. For each record in the dataset, I write to the context `Map(DepartureAirport_DestinatioAirport,DepartureDelayInMinutes)`. The reduce job counts average departure delay for this composite key and for each departure-destination key counts its average delay in minutes.

`Map(DepartureAirport_DestinatioAirport, DepartureDelayInMinutes) -> list(DepartureAirport_DestinatioAirport, DelayInMinutes)`

`Reduce(DepartureAirport_DestinatioAirport, list (DepartureAirport_DestinatioAirport)) → list(DepartureAirport_DestinatioAirport, AverageDelayInMinutes)`

Results for individual queries:

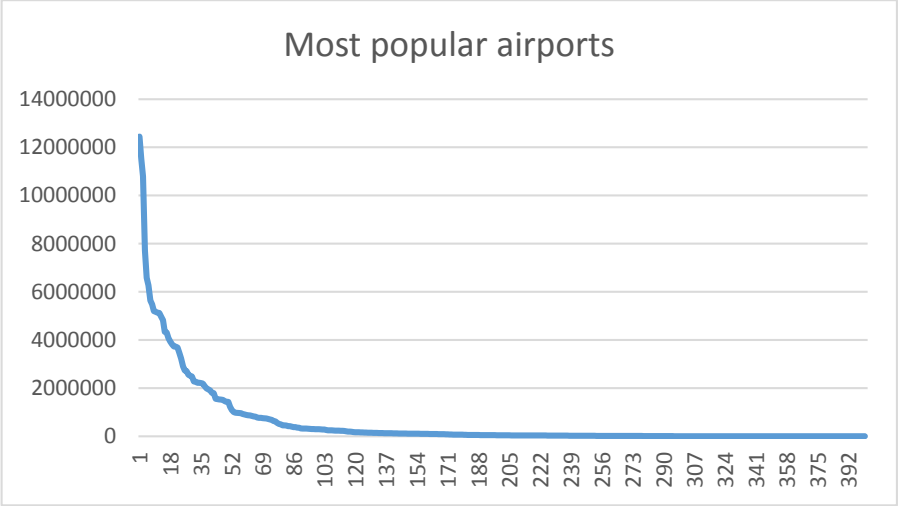
"ATL"-"PHX" 9.021342
"CMI"-"ORD" 10.143662
"DFW"-"IAH" 7.617333
"IND"-"CMH" 2.8911366
"JFK"-"LAX" 6.635119
"LAX"-"SFO" 9.589283

Group 3

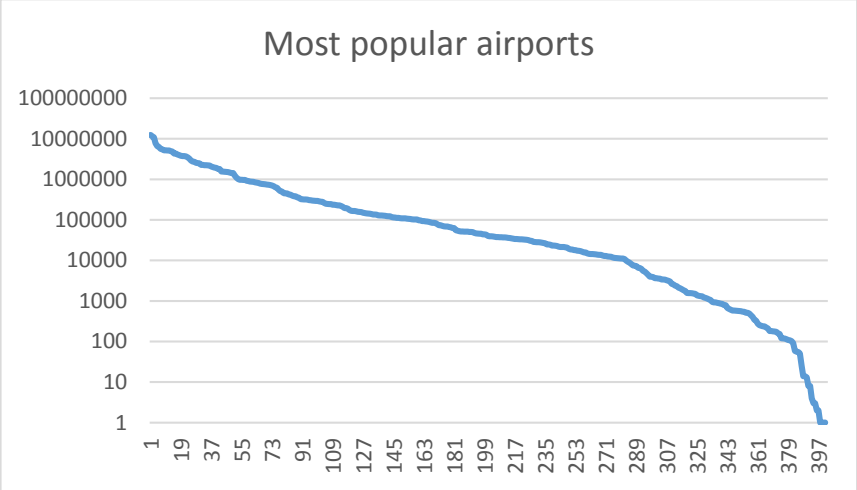
Does the popularity distribution of airports follow a Zipf distribution? If not, what distribution does it follow?

To perform this task I used MapReduce approach.

I take all data (not top 10) from task 1 in group 1 (most popular airports by numbers of flights to/from the airport). And I built a line graph with this data:



Then I changed the linear scale to logarithmic and I've got almost Zipf distribution graph, except for the low end:



Tom wants to travel from airport X to airport Z. However, Tom also wants to stop at airport Y for some sightseeing on the way.

My idea how to complete this task:

- Prepare two data sets. The first with first leg flights, which are filtered by departure time between 00:00 and 12:00. The second with second leg flights, which are filtered by departure time greater than 12:00. Both of this datasets are with the flights which were in 2008 year. Prepare this datasets with the help of PIG script.

- Join those two datasets with PIG. The key to join in first dataset is destination airport and the key to join in the second dataset is departure airport. The schema of resulted dataset:

```
LEG_ONE::FlightDate,
LEG_ONE::AirlineID,
LEG_ONE::Carrier,
LEG_ONE::Origin,
LEG_ONE::Dest,
LEG_ONE::DepTime,
LEG_TWO::FlightDate,
LEG_TWO::AirlineID,
LEG_TWO::Carrier,
LEG_TWO::Origin,
LEG_TWO::Dest,
LEG_TWO::DepTime;
```

- Filter resulted dataset with PIG - `LEG_ TWO::FlightDate` must be two days after `LEG_ONE::FlightDate`

- Final filtered dataset must be processed with MapReduce job to calculate best on-time performance.

Integration

We can integrate Ambari Hortonworks with various others systems, for example Apache Cassandra. We can easily download additional Cassandra Ambari service and install it from the Ambari dashboard with the help of Service Wizard.

Optimization

As we have a cluster of several servers, there are several possibilities for optimization in described above infrastructure to optimize expenses needed to perform a calculation job:

- There is possibility to add/remove server to the cluster.

- There is possibility to change RAM and CPU of each server.

- There is possibility to add additional volume storages to each server.