Table of content:

Project Overview

Project:
Flight Data Analysis

Description:
In this project, you will develop cloud-based Big Data workflows to process and analyze a large volume of flight data.

Instructions:
1. Install Hadoop on your AWS VMs.
2. Download the Airline On-time Performance data set (flight data set) from the period of October 1987 to April 2008 on the Statistical Computing website: http://stat-computing.org/dataexpo/2009/the-data.html

   I am using 3 years of data (2006 to 2008) for this project.

3. Design, implement, and run MapReduce jobs to find out
   - the 3 airlines with the highest and lowest probability, respectively, for being on schedule;
   - the 3 airports with the longest and shortest average taxi time per flight (both in and out), respectively; and
   - the most common reason for flight cancellations.

Requirements:
1. Your workflow must contain at least three MapReduce jobs.
2. Run your workflow to analyze the entire data set (total 22 years from 1987 to 2008) at one time and measure the execution time.
3. Run your workflow to analyze the data in a progressive manner with an increment of 1 year, i.e. the first year (1987), the first 2 years (1987-1988), the first 3 years (1987-1989), …, and the total 22 years (1987-2008), and measure each corresponding execution time.

Submitted components and description

1. commands.txt:
   This document has all the used commands from data file movement to result file creation.

2. Output.txt:
   This file has results of all the executions.

3. Source code:
   I have used Python to write Mappers and Reducers. Source_Code folder has following components-

   For airlines schedule delay problem:
   Mapper_prob.py
   Reducer_prob.py

   For taxi time problem:
   Mapper_taxi.py
   Reducer_taxi.py

   For cancellation reason problem:
   Mapper_cancel.py
   Reducer_cancel.py

4. MapReduce_Output.txt:
   This document has brief statistics of all executed jobs. Stats include File System Counters details, Map-Reduce Framework details, shuffle details, File Input Format Counters, File Output Format Counters and execution time (real, user and sys).

5. Project Report.pdf:
   This document.

Technical specification

I have used Python as programming language to code MapReduce jobs. Following are the technical details:

Sr. No.	Item	Detail	Version
1	Cloud Solution	AWS EC2	
2	Linux	Amazon Linux AMI	2018.03.0

3	Big Data library	Hadoop	3.1.3
4	Programming Language	Python	3.6
5	Machine Java version		Java 1.8

Algorithm and Design

I have written my Mapper and Reducer scripts in Python which can be run on Hadoop using Hadoop Streaming API which uses STDIN/STDOUT

PROBABILITY to be ON TIME by AIRLINES

Mapper -

- Reads line by line from the input file
- Header row is ignored
- Each row of the Input file is split into fields using "," as the delimiter
- Only 2 split values are read to be Mapper Output - Unique Carrier is the KEY and ArrDelay is the VALUE in minutes
- Since this field was present in the File, we need not worry about getting Scheduled Arrival Time and Actual Arrival Time and worry about time zone, difference etc.

Reducer -

- The input to Reducer is the Sorted Output from Mapper in the form of UniqueCarrier<tab>ArrDelay
- Each line is read and split by Tab
- ArrDelay that have NA are ignored
- For each Key (UniqueCarrier), the ArrDelay (that is >0) is added up to get a Total ArrDelay
- A counter for rows being read is maintained too, this includes rows with ArrDelay =0, <0 or >0 (please note that we have already ignored rows that have NA)
- The Probability of each Key (UniqueCarrier) being ON TIME is then calculated by (1 - Total ArrDelay/Total Rows)
- The Key and Probability Values are then written into a Python List - 'finallist'
- Then once we have all the Keys (UniqueCarrier) and their respective Probabilities in 'finallist', I have a function Sort that is then called to get Top3 and Bottom3 Probabilities
- The Final Output as can been seen in the Results file is the Unique Carrier - Prob to be ON TIME

TAXI IN and TAXI OUT by AIRPORT

Mapper -

- Reads line by line from the input file

- Header row is ignored
- Each row of the Input file is split into fields using "," as the delimiter
- Each row has 4 fields of interest for us, Origin, Dest, TaxiIn, TaxiOut
- Origin Airport will have flights taking off hence it will correspond to TaxiOut time
- Dest Airport will have flights landing hence it will correspond to TaxiIn time
- So, for each row, my Mapper generates 2 Output Key-Value pairs - Origin-taxiOut, Dest-taxiIN
- Also, I am ignoring taxi times that are 0
- Please note that I am adding Suffix -IN and -OUT to the Airports so as my Reducer knows exactly what its input is, whether for TaxiIN or TaxiOUT

Reducer -

- The input to Reducer is the Sorted Output from Mapper in the form of Airportcode-IN/OUT<tab>TaxiOUT/IN
- Each line is read and split by Tab
- Taxi times that have NA are ignored
- For each Key (AirportCode-IN), the Taxi time is added up to get a Total Taxi time
- A counter for rows being read is maintained too (please note that we have already ignored rows that have NA)
- The Average Taxi time of each Key (Airportcode-IN) is then calculated by (Total Taxi Time/Total Rows)
- The Key and Average Taxi time are then written into a Python List - 'finallist_IN'
- The above 4 steps are being done simultaneously for Key (AirportCode-OUT) too
- For each Key (AirportCode-OUT), the Taxi time is added up to get a Total Taxi time
- A counter for rows being read is maintained too (please note that we have already ignored rows that have NA)
- The Average Taxi time of each Key (Airportcode-OUT) is then calculated by (Total Taxi Time/Total Rows)
- The Key and Average Taxi time are then written into a Python List - 'finallist_OUT'
- So now we have 2 Python Lists with AirportCode and Average Taxi time, one list is for IN and other for OUT
- I have a function Sort that is then called to get Top3 and Bottom3 Average Times for each of the above 2 Lists
- The Final Output as can been seen in the Results file is the Airport - Average taxi time

CANCELATION Code

Mapper -

- Reads line by line from the input file
- Header row is ignored
- Each row of the Input file is split into fields using "," as the delimiter
- Only 1 value from the row is needed for the Mapper Output Key- CancellationCode is the KEY and 1 is the VALUE (1 since we just need a counter for occurrence of cancellation code)
- There is a field called Cancelled which has values 1/0, first I thought it is needed to check whether a flight was cancelled and then get the cancellation code but realized it is not needed at all for the task at hand

Reducer -

- The input to Reducer is the Sorted Output from Mapper in the form of CancellationCode<tab>1
- Each line is read and split by Tab
- CancellationCode that have NA are ignored
- For each Key (CancellationCode), I could see in the Data Dictionary that values are A, B, C and D that are mapped to Carrier, Weather, NAS and Security
- So, my Reducer checks the InputKey (A, B, C, D) and generates a new Output Key corresponding to these codes
- A counter for rows being read is maintained too
- The New Output Key and Count Values are then written into a Python List - 'finallist'
- Then once we have all the Keys (CancellationCode) and their respective Counts in 'finallist', I have a function Sort that is then called to Sort the list in Descending Order
- The Final Output as can been seen in the Results file is the CancellationCode - Count

Performance measurement plot

As only one VM was used for this project so resources were stable, and performance of job was dependent on the amount of data

Following “Data vs Time” performance plot shows linear increase in processing time for each MapReduce job with increase in data. We can say jobs performed in quite optimal way and processing time can be decreased with use of multiple VMs and that would a real example of Big Data processing.