Report – A5 – Airline Connection

Hu,Ruinan
hu.ru@husky.neu.edu

Aswathanarayana,Naveen
aswathanarayana.n@husky.neu.edu

1 Introduction

Analyze the airline data and find connections and missed connections based on the two criteria. A connection is any pair of flight F and G of the same carrier such as F.Destination = G.Origin and the scheduled departure of G is <= 6 hours and >= 30 minutes after the scheduled arrival of F. A connection is missed when the actual arrival of F < 30 minutes before the actual departure of G. This program will output the number of connections and missed connections per airline, per year. Program will output the result file upon which a cleanup java program will aggregate the result set to provide the details.

2 Data structure

CustomWritable and CustomWritableComparable:

Text cityName => Name of the City
int flightType => To/From intermediate city
long flightTime => Arrival/Departure Schedule time
long actualTime => Arrival/Departure Actual time
Note: CustomWritableComparable does sort the list of CustomWritable based on Schedule time in Reducer.

Mapper:

(Key,Value)
=> (Text,CustomWritable)
=>
(<YEAR,CARRIER,INTERMEDIATE_CITY>,<DESTINATION/ORIGIN_CITY,FLAG,SCHEDULED-DEPARTURE/ARRIVAL-TIME,ACTUAL-DEPARTURE/ARRIVAL-TIME>)

Reducer:

(Key,Value)
=> (Text, Text)
=> (<YEAR,CARRIER>,<CONNECTIONS,MISSEDCONNECTIONS>)

3 Design and Implementation

Mapper generates key value pair as described in the data-structure section. Each record in the dataset is divided into two key values pair. One contains the information of origin and departure details and the other contains the information of destination and arrival details. We distinguish the two data from each record by the flag to indicate it is either arrival or destination details. Convert the time into milliseconds and do not worry about the roll over details on time. We are not handling the year roll over time. We have CustomWritable class which handles the row details from the mapper. Reducer operates on these

data. We sort the iterable from the mapper in ascending order of time. Since they are ordered ascending order of time. We compare each of the data to the next available withing the range of six hours.

Mapper:

Mapper generates two key value pair for each record from the dataset.
We consider the two cities from each record as intermediate city and put these in key.

Example for each record:
<AA,2015,Boston,ScheduledDepartureTime,ActualDepartureTime,Chicago,ScheduledArrivalTime,ActualArrivalTime>
1) <2015,AA,Boston><Chicago,0,ScheduledDepartureTime,ActualDepartureTime>
2) <2015,AA,Chicago><Boston,1,ScheduledArrivalTime,ActualArrivalTime>

To over come the rollover of the day,month, we are using Calendar object to generate milliseconds for the time-values.

Reducer:

Alogrithm

Connections
1. Sort the values for the key based on Scheduled-time
2. for each flight in the arraylist of flights
    if the flight is departureFlight(0) then ignore.
    if the flight is arrival(1) then you can make a connection
        while the difference between the scheduled arrival and departure time falls within
            conditions for connection(<=360)
            if(the arrival and departure time falls within conditions for connection(>=30))
            increment the connectionCount

MissedConnections
1. for each flight in the arraylist of flights
    if the flight is departureFlight(0) then ignore.
    if the flight is arrival(1) then you can make a connection
        while the difference between the actual arrival and departure time falls within conditions
            for missed connection(<=30)
            increment the connectionCount

This algorithm considers only the arrival flights which can make connections with the departure flight within the time frame of 30 and 360 minutes. Also we sort the array list based on schedule time it minimize the dataset we need to compare the time.

Reducer output.

(<YEAR,CARRIER>,<CONNECTIONS,MISSEDCONNECTIONS>) for each intermediate city.

4 Conclusion and Analysis

Volume of data passed through MR job => n dataset -> mapper -> 2n dataset -> reducer

1) Comparision time
c1 * 2n = arrival flights to intermediate city
c2 * 2n = departure flights from intermediate city
c1 + c2 = 1
where c is some constant.
c1 and c2 are ratio of arrival to departure flights for intermediate city.
c3 is highest density of flights per minutes
for one arriving flight we compare (density of flight X 360 minutes)
so the total comparasion time < 2 * c1 * 360 * c3 * n
2) Sorting Time for each reducer
For each intermediate city, the total number of flight of each year and carrier is not related to the size of dataset, we can assume is c4. So the sorting time for intermediate city is c4log(c4).
3) Number of reducers
We have 33 carriers, k*(n/c6) years, c6 is the lowest density of flights, and c5 citys
Running Time < k*(n/c6) * c5 * 33 * c4log(c4) + 2 * c1 * 360 * c3 * n = O(n)

Execution Time

Machine Configuration	Dataset	Time
AWS 1-Master, 9-Slave, m3.xlarge	337	6 minutes
AWS 1-Master, 5-Slave, m3.xlarge	25	2 minutes
Pseudo - Processor – i5 RAM – 8 GB HDD – 5200 RPM	25	4 minutes