**CS6240**-**Intermediate report**

**Team Members:**

Aakash Shah

Sumanth Munikoti

Rahul Pandey

Github: <https://github.com/2020-F-CS6240/project-projectsgroup-16>

**Project Overview:**

The project

Use HBase as an index for an equi-join implementation:store one input relation in HBase, then use a MapReduce or Spark job that scans through the other input relation and looks for matches in the HBase table(s). Compare the performance and scalability of this approach against hash+shuffle (Reduce-side join).

**Input Data :**

Describe the data you are working with. Include a line of input, if feasible. (Do not include binary data or lines that are too long.)

The input data represents On-time performance of flights in the US, published by the Bureau of Transportation Statistics, it can be found at :<http://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236&DB_Short_Name=On-Time>

Input data format:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| FL\_DATE | ORIGIN\_AIRPORT\_ID | ORIGIN\_CITY\_NAME | DEST\_AIRPORT\_ID | DEST\_CITY\_NAME | DEP\_TIME | ARR\_TIME | CANCELLED | DISTANCE |
| 1/1/14 | 14683 | "San Antonio, TX" | 11298 | "Dallas/Fort Worth, TX" | 1647 | 1744 | 0 | 247 |

**Overview Task 1:**

We try to create a travel itinerary which covers 3 intermediate stops before coming back to the origin airport. We implement this join using Reduce-Side join on the flight data

**Pseudo-Code:**

The task is to be completed in two Map-Reduce jobs:

Job 1:

Map(flightRecord) {

Emit(flightRecord.origin, (flightRecord, “out”)

Emit(flightRecord.destination, (flightRecord, “in”)

}

Reduce(airportId, …[(flightRecords, flag)] flights) {

inFlights = []

outFlights =[]

foreach flight in flights:

if flag of fligh == out, then add the flight to outFlights

else add the flight to inFlights

for (inflight in inFlights):

for (outFlight in outFlights):

if ( inflight.origin != outFlight.destination) {

mergedFlightPlan 🡪 origin: inflight.origin, destination: outflight.destination, arrTime: outflight.arrivalTime, departureTime: inflight.departureTime

emit(airportId, mergedFlightPlan)

}

}

Second Job:

// reads merged flight plan created by the reduce of the first job

Map(airportId, FlightPlan) {

Emit(FlightPlan.origin, (flightRecord, “out”)

Emit(FlightPlan.destination, (flightRecord, “in”)

}

Reduce(airportId, …[(flightRecords, flag)] flight) {

inFlights = []

outFlights =[]

foreach flight in flights:

if flag of fligh == out, then add the flight to outFlights

else add the flight to inFlights

for (inflight in inFlights):

for (outFlight in outFlights):

if ( inflight.origin == outFlight.destination

&& inflight.intermediateAirport != outFlight .intermediateAirport) {

mergedFlightPlan 🡪 origin: inflight.origin, destination: outflight.destination, arrTime: outflight.arrivalTime, departureTime: inflight.departureTime

emit(inflight, outFlight)

}

}

**Algorithm and Program Analysis:**

The first job yields all the flight plans that go from one city to another city while stopping at an intermediate location. This yields us a flight Plan of A🡪B🡪C (A != C )

In the first job all of the input data is read once but written twice in the map phase of the job while omitting the cancelled flights(optimizing number of records to be processed for the later jobs, by ignoring cancelled flights). In the reduce task we find out all the flight plans which are not a “return” flight, (that is of the format A🡪B🡪A), that is we do an equi-join of the input flight records data to yield the required flight plan

In the second job we have flight plans of the format A🡪B🡪C. We do a self equi-join on this data and compute all flight plans which are of the form A🡪B🡪C🡪D🡪E. (A == E)

**Experiments :**

**Speedup :**

Input Records: 5000

|  |  |  |
| --- | --- | --- |
| M4.xlarge | 3 + 1 | 6+1 |
| 5000 | 3 min 45 seconds | 2 min 40 seconds |
| logs | https://github.com/2020-F-CS6240/project-projectsgroup-16/blob/master/logs/3%2B1/logs.txt | https://github.com/2020-F-CS6240/project-projectsgroup-16/blob/master/logs/6%2B1/logs.txt |

**Scalability :**

|  |  |  |
| --- | --- | --- |
| M4.xlarge | 5000 records | 10000 records |
| 6 + 1 | 2 min 40 seconds | 36 min 34 seconds |
| logs | https://github.com/2020-F-CS6240/project-projectsgroup-16/blob/master/logs/6%2B1/logs.txt | https://github.com/2020-F-CS6240/project-projectsgroup-16/blob/master/logs/10k/logs.txt |

**Result Sample:**

We tested the join on a simple test input data for 64 rows and the output is available at: https://github.com/2020-F-CS6240/project-projectsgroup-16/tree/master/output

**Conclusions:**

**Overview Task 2:**

We try to create a travel itinerary which covers 3 intermediate stops before coming back to the origin airport. We implement this join using HBase

To Do