

**CS6240**  
**Parallel Data Processing**  
**in MapReduce**

**Fall 2013**  
**Project Report**

**By**

**Nishant Agarwal**  
**Tushar Bhandari**  
**Purushottam Jha**  
**Rushil Patel**

## **INTRODUCTION:**

Our topic for Final Project is Analysis of Flights and Airlines. Our goals for this project is to learn implementations of HBase, Hive, PigLatin as well as solving more complex problems in plain MapReduce on local as well as on Amazon EMR and perform detailed representation of our analysis on charts.

## **TASK LIST :**

<b>Task 1</b>	HBase	a) On Time Arrival Performance of Airlines b) Average delay at each airport.
<b>Task 2</b>	Hive and Pig Latin	Three legged round trip flight from Boston.
<b>Task 3</b>	Map Reduce	Calculate Page Rank of each Airport.
<b>Task 4</b>	Map Reduce	Hubs and Spokes

## **DATA:**

Name: Airline On-Time Performance and Causes of Flight Delays

Size: 4.34GB with 4336074861 records.

Fields: 55 Columns (such as FlightNo, Date, DepDelayMins, etc.)

View of sample dataset: (with few columns)

flightdate	flightnum	origin	origincityname	dest	destcityname	deptime	arrtime	arrdelay minutes
2007-10-02	415	IND	Indianapolis, IN	MDW	Chicago, IL	1102	1100	30.00
2007-10-02	917	IND	Indianapolis, IN	MDW	Chicago, IL	0728	0712	0.00
2007-10-02	1022	IND	Indianapolis, IN	MDW	Chicago, IL	1855	1842	0.00
2007-10-02	2081	IND	Indianapolis, IN	MDW	Chicago, IL	1530	1522	0.00
2007-10-02	1033	IND	Indianapolis, IN	PHX	Phoenix, AZ	1500	1544	0.00

Here is the link for the dataset:

<https://explore.data.gov/Transportation/Airline-On-Time-Performance-and-Causes-of-Flight-D/ar4r-an9z>

## **TECHNICAL DISCUSSION:**

### **TASK 1 : HBASE**

#### **a) On-time arrival performance of airlines**

**Purpose:**

Calculating and analyzing performance of airlines based on their arrival delay.

**Approach:**

We have two mapreduce jobs. The first job computes the average arrival delay of each airline by reading the data from the csv file and storing this intermediate result in reverse order (key: averageDelay, value: airlineID) into a HBase table. Since HBase has the property of sorting by rowkey it sorts the data according to average delay. Here, to take advantage of the HBase property we had to store the averageDelay in bytes array of float values rather than Strings. (which we were doing before). The second mapreduce job is a map-only job that just reads this from the HBase table and writes output to the file. This ensures that the sorting is done using HBase rowkey property.

**Pseudo-code:**

Job 1:

```
map(key, value) { // key = line offset , value = each record as String
    airlineId = value.getAirlineID();
    arrDelayMins = value.getArrDelayMinutes();
    emit(airlineId, arrDelayMins);
}

reduce(key, List[values]) {
    sum, total = 0,
    for delayValue in values:
        total ++;
        sum += delayValue;

    // Finally compute average and emit in reducer
    averageDelayMins =(sum / total);
    // HBaseConnection is a utility class that we have additionally created to talk to HBase.
    HBaseConnection.addRecord(tablename, sum/total , "airlineID", "", key.toString());
}
```

**Output of Job1 (on 4.34 GB data):** *initial few records only*

Key : Airline ID	Value: Average Delay
19386	14.574111
19393	10.166066
19678	3.915491
19690	4.53628
19704	15.95415
19790	12.485989
19805	18.206627
19930	12.79604
19977	17.726479
20304	12.365397

Job 2:

```
map( key, value) { // key is averageDelay, value is airlineID
    averageDelay = row.get() //convert average delay from Bytes to DoubleWritable
    byte[] result = value.getValue(CF, ATTR);
```

```

        Text airlineID = new Text(Bytes.toString(result));
        emit(averageArrivalDelay, airlineID);
    }

```

### Output of Job2:

The output of Job2 is same as Job1 but the only difference is that all the rows are sorted according to the arrival delay.

Now to make things more elegant we have a sequential program to convert these airlineID's into actual airline name. *(Top few records only)*

Key : Average Delay	Value: Airline ID
3.9154911041259766	Aloha Air Cargo: KH
4.536280155181885	Hawaiian Airlines Inc.: HA
10.16606616973877	Southwest Airlines Co.: WN
10.564054489135742	Frontier Airlines Inc.: F9
12.040234565734863	Endeavor Air Inc.: 9E
12.365397453308105	SkyWest Airline(s Inc.: OO
12.48598861694336	Delta Air Lines Inc.: DL
12.796039581298828	Alaska Airlines Inc.: AS
13.382158279418945	US Airways Inc.: US (Merged with America West 9/05. Reporting for both starting 10/07.)
13.557802200317383	AirTran Airways Corporation: FL

**Source:** /HBase/HBase1/src

**Log:** /HBase/HBase1/logs

**Output:** /HBase/HBase1/output

### b) Average delay at each airport

#### **Purpose:**

Calculating the average arrival delay at each airport and analyse on the map of U.S.A.

#### **Approach:**

This job computes the average arrival delay of each airport by reading the data from the csv file and storing the result (key: airport (destination), value: averageDelay) into a HBase table.

#### **Pseudo-code:**

```

map(key,value) { // key = line offset , value = each record as String
    destination =value.getDest();
    arrDelayMins = value.getArrDelayMinutes();
    emit (destination, arrDelayMins);
}

reduce(key, List[values]) { // key is destination(each airport)
    // values is a list of average delay. This list contains only one value.
    sum, total = 0;
    for value in values:
        arrDelayMins = float(value);

```

```

        total ++;
        sum += arrDelayMins;
    // Finally compute average and emit in reducer
    averageDelayMins =(sum / total);
    // HBaseConnection is a utility class that we have additionally created to talk to HBase.
    HBaseConnection.addRecord(tablename, key, "delay", "", averageDelayMins);
}

```

### Output:

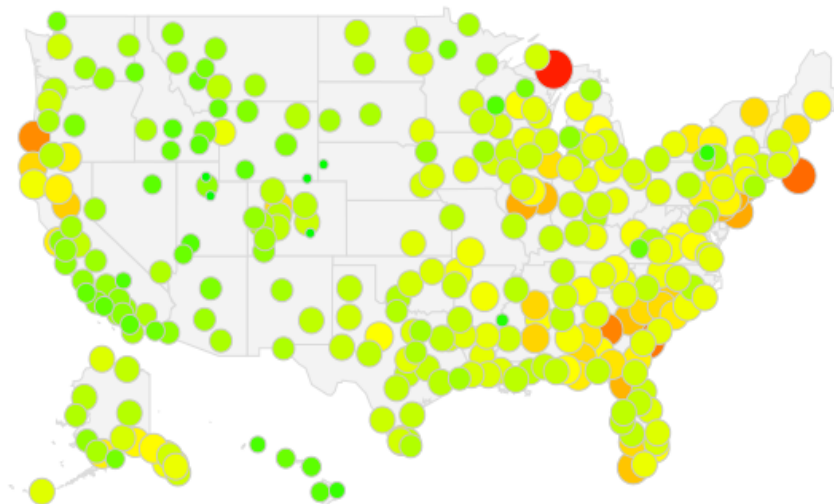
Actual output contains 316 rows - one for each airport. Here are top few records seen from HBase:

```

tushar@ubuntu: ~/Downloads/hbase-0.94.13/bin
hbase(main):008:0> scan 'average_arrival_delay_airport'
ROW COLUMN+CELL
ABE column=delay:, timestamp=1385273156230, value=15.436178
ABI column=delay:, timestamp=1385273156475, value=17.1708
ABQ column=delay:, timestamp=1385273156622, value=11.197535
ABY column=delay:, timestamp=1385273156628, value=17.424364
ACK column=delay:, timestamp=1385273156634, value=27.321066
ACT column=delay:, timestamp=1385273156642, value=11.898975
ACV column=delay:, timestamp=1385273156653, value=16.295553
ACY column=delay:, timestamp=1385273156674, value=22.995174
ADK column=delay:, timestamp=1385273156679, value=15.110526
ADQ column=delay:, timestamp=1385273156684, value=7.856813
AEX column=delay:, timestamp=1385273156691, value=15.545198
AGS column=delay:, timestamp=1385273156758, value=21.567919
AKN column=delay:, timestamp=1385273156762, value=18.128204
ALB column=delay:, timestamp=1385273156787, value=15.653317
ALO column=delay:, timestamp=1385273156792, value=10.74216
AMA column=delay:, timestamp=1385273156803, value=12.934547
ANC column=delay:, timestamp=1385273156830, value=14.513902
APF column=delay:, timestamp=1385273156834, value=19.676895
ASE column=delay:, timestamp=1385273156843, value=12.4784775
ATL column=delay:, timestamp=1385273157369, value=14.7314005
ATW column=delay:, timestamp=1385273157380, value=15.199821
AUS column=delay:, timestamp=1385273157474, value=13.127228

```

Plotting this on the United States map, it is more evident that there are more flights on the east coast compared to the west coast and also more delays on the east coast.



**Source:** /HBase/HBase2/src  
**Log:** /HBase/HBase2/logs  
**Output:** /HBase/HBase2/output

### Performance on EMR:

Program	No. of Worker Machines	Machine Type	Time Taken
HBase1 (Task 1)	2	large	8 minutes
HBase 1 (Task 1)	2	large	9 minutes
HBase2 (Task 2)	5	large	4 minutes
HBase2 (Task 2)	5	large	4 minutes

## TASK 2 : HIVE & PIG LATIN - Three legged round trip flight from Boston.

### Purpose:

We are finding shortest time taking three legged round trip flights from Boston(e.g. journeys of the form BOS-X-Y-BOS). We have done it in both HQL and PigLatin.

### Approach:

**HIVE:** We have used CSVSerde to create the flight table in hive and have stored it as TEXTFILE.

We are adding CSVSerde jar on EMR as given below:

```
add jar s3n://finalproj-puru/lib/csv-serde-1.1.2-0.11.0-all.jar;
```

The create query is given below:

```
CREATE EXTERNAL TABLE IF NOT EXISTS Flight(  
    Year FLOAT,Quarter FLOAT,Month FLOAT,DayofMonth FLOAT,DayOfWeek  
    FLOAT,...,LateAircraftDelay FLOAT  
row format serde 'com.bizo.hive.serde.csv.CSVSerde'  
STORED AS TEXTFILE  
LOCATION 's3n://finalproj-puru/table/';
```

We are loading the data in hive as follows:

```
LOAD DATA INPATH 's3n://finalproj-puru/piginput/data.csv' INTO TABLE flight;
```

We are adding up the actual elapsed times all the three flights and also the layovers at the two intermediate airports. We are also checking for the condition that the departure time of second flight should be greater than the arrival time of first flight and the departure time of third flight should be greater than the arrival time of the second flight.

### PigLatin:

We are using filter and joins to check for the required conditions as given in the pseudo code.

### Pseudo Code:

#### HIVE:

```
INSERT OVERWRITE DIRECTORY 's3n://finalproj-puru/hiveoutput'  
Select a.origin, a.dest, b.dest, c.dest, a.flightdate, a.DepTime, a.ArrTime,  
b.DepTime, b.ArrTime, c.DepTime, c.ArrTime, (a.actualelapsedtime +  
b.actualelapsedtime + c.actualelapsedtime + (b.DepTime-a.ArrTime) +  
(c.DepTime-b.ArrTime)) AS TotalTime from flight a JOIN flight b on (a.dest =  
b.origin) JOIN flight c on (b.dest = c.origin) where a.flightdate  
='2008-01-01' AND b.flightdate = '2008-01-01' AND c.flightdate = '2008-01-01'  
AND a.origin = 'BOS' AND c.dest = 'BOS'
```

```

AND a.DepTime < a.ArrTime
AND b.DepTime < b.ArrTime
AND c.DepTime < c.ArrTime
AND b.DepTime > a.ArrTime
AND c.DepTime > b.ArrTime AND b.DepTime - a.ArrTime > 100 AND c.DepTime -
b.ArrTime > 100
AND a.cancelled != 1 AND b.cancelled != 1 AND b.cancelled != 1
Order by TotalTime LIMIT 20;

```

### PigLatin:

```

Flights1_Data = FILTER Flights1_Data by (orig1 == 'BOS') AND (flightDate1 ==
'2008-01-01') AND (cancelled1 != 1);
Flights2_Data = FILTER Flights2_Data by (flightDate2 == '2008-01-01') AND
(cancelled2 != 1);
Flights3_Data = FILTER Flights3_Data by (dest3 == 'BOS') AND (flightDate3 ==
'2008-01-01') AND (cancelled3 != 1);

flf2 = JOIN Flights1_Data BY (dest1), Flights2_Data BY (orig2);
flf2 = FILTER flf2 BY depTime2 > arrTime1;

flf2f3 = JOIN flf2 BY (dest2), Flights3_Data BY (orig3);
flf2f3 = FILTER flf2f3 BY depTime3 > arrTime2;

flf2f3 = FILTER flf2f3 BY ((depTime2-arrTime1) > 100) AND ((depTime3-arrTime2)
> 100);

final = FOREACH flf2f3 GENERATE
orig1,dest1,dest2,dest3,flightDate1,depTime1,arrTime1,depTime2,arrTime2,depTim
e3,arrTime3, (actualElapsedTime1 + actualElapsedTime2 + actualElapsedTime3 +
(depTime2 - arrTime1) + (depTime3 - arrTime2)) AS totalTripTime;

final = ORDER final BY totalTripTime;
final = limit final 20;
STORE final INTO '$OUTPUT';

```

**Source:** Hive: /Hive/src

PigLatin: /PigLatin/src

**Log:** Hive: /Hive/logs

PigLatin: /PigLatin/logs

**Output:** Hive: /Hive/output

PigLatin: /PigLatin/output

Origin	Dest1	Dest2	End	Flight Date	F1: DepTime	F1: ArrTime	F2: DepTime	F2: ArrTime	F3: DepTime	F3: ArrTime	TotalTripTi me
BOS	DCA	LGA	BOS	1/1/2008	847	1028	1131	1223	1328	1435	428
BOS	DCA	LGA	BOS	1/1/2008	847	1028	1131	1223	1339	1442	435
BOS	DCA	LGA	BOS	1/1/2008	1602	1735	1852	2004	2107	2200	438
BOS	IAD	LGA	BOS	1/1/2008	1021	1211	1320	1420	1528	1625	444
BOS	DCA	LGA	BOS	1/1/2008	847	1028	1131	1223	1354	1452	445
BOS	JFK	PHL	BOS	1/1/2008	1434	1542	1645	1832	1938	2041	447
BOS	LGA	DCA	BOS	1/1/2008	1128	1229	1352	1502	1619	1738	450
BOS	IAD	LGA	BOS	1/1/2008	1021	1211	1320	1420	1530	1632	451
BOS	IAD	LGA	BOS	1/1/2008	1021	1211	1320	1420	1556	1647	466
BOS	IAD	PHL	BOS	1/1/2008	1411	1604	1715	1808	1938	2041	470

### Performance on EMR:

Program	No. of Machines	Machine Type	Time Taken
HIVE	11	small	32 Mins
HIVE	20	small	15 Mins
PigLatin	11	small	14 Mins 49 Secs
PigLatin	20	small	13 Mins 23 Secs

## TASK 3 : PLAIN MAP REDUCE - Calculate PageRank of each Airport.

**Purpose:** To identify the busyness of the airport based on the traffic and the busyness value of it's neighbors

**Approach:** This task is divided into two sub-tasks. First task is responsible for creating an intermediate input file which will consist of only required information along with the initial pagerank value i.e.  $1/\text{total number of airports}$ . This intermediate input will then be provided to the pagerank algorithm.

### FIRST SUB-TASK:

This task was performed using in-mapper aggregation. While analyzing the input it was observed that flight details of the same airport was often in the same data chunk. Hence, in-mapper aggregation was the chosen pattern. Also, to keep the source code dynamic and adaptable to any graph change i.e. new node or new vertex, we calculated the marginal value (total number of nodes) using the order inversion design pattern.

**Source:** /Pagerank/src/pagerank/InitAirportRank.java

**Log:** /Pagerank/logs/syslog - initialize pagerank using in-mapper aggregation

**Output:** /pagerank/output/initialize

### Pseudo-code:

```
setup()
    Set airports
    Map originCounter<String,Integer>

map(Key k, Value v)
    flight = parse(v)
    airports.add(flight.destination)
    airport.add(flight.origin) //to handle dangling nodes
    if(originCounter.containsKey(flight.origin))
        originCounter[flight.destination][flight.origin]++
    else
        originCounter[flight.destination].put(flight.origin,1)

cleanup()
    foreach d in airports
        emit('*',[d,1]) // to count # of airports (nodes)
```



```

foreach o in originCounter
    emit(o.flightDestination,[o.flightOrigin,o.count])
    // neighbor list of each airport along the count from each origin

reduce(Key k, Values v1,v2,v3,...)
    if(k=='*')
        for each v in v1,v2,v3...
            add v in set
        marginal = set size
    else
        for each v in v1,v2,v3
            add count of all similar input links
        emit(k,[(pagerank,1/marginal),(origin1,count1),(origin2,count2),...])

```

**Input:** snapshot of few records

Origin	Destination
ORD	LAX
ORD	LAX
ORD	LAX
BOS	JFK
JFK	LAX
BOS	ORD

**Output:**

```

BOS    PageRank, 0.14285714285714285
CLE    PageRank, 0.14285714285714285
DEN    PageRank, 0.14285714285714285; SFO, 1
DTW    PageRank, 0.14285714285714285; CLE, 1
LAX    PageRank, 0.14285714285714285; ORD, 15; BOS, 2
ORD    PageRank, 0.14285714285714285
SFO    PageRank, 0.14285714285714285

```

## SECOND SUB\_TASK:

This task follows the pagerank algorithm.

**Source:** /Pagerank/src/pagerank/PageRank.java

**Log:** /Pagerank/logs/syslog - pagerank using 10 machines

**Output:** /pagerank/output/part-r-00000 to part-r-00015

**Pseudo-code:**

```

map(nid n, node N)
    emit(nid n, N)
    foreach o in originCounter
        total+=o.count
    p = N.pageRank / total
    for all nid m in N.incomingLinks do
        emit(nid m, p)

reduce(nid m, [p1,p2,...])
    s=0; M = null

```

```

for all p in [p1,p2,...] do
    if isNode(p) then
        M=p
    else
        s += p
M.pageRank = alpha /totalNumberOfAirports + (1-alpha)*s
emit(nid m, node M)

```

#### Input:

```

BOS    PageRank,0.14285714285714285
CLE    PageRank,0.14285714285714285
DEN    PageRank,0.14285714285714285;SFO,1
DTW    PageRank,0.14285714285714285;CLE,1
LAX    PageRank,0.14285714285714285;ORD,15;BOS,2
ORD    PageRank,0.14285714285714285
SFO    PageRank,0.14285714285714285

```

#### Output:

```

BOS    PageRank,0.03509803921568627
CLE    PageRank,0.06333333333333332
DEN    PageRank,0.03333333333333333;SFO,1
DTW    PageRank,0.03333333333333333;CLE,1
LAX    PageRank,0.03333333333333333;ORD,15;BOS,2
ORD    PageRank,0.03509803921568627
SFO    PageRank,0.06333333333333332

```

#### Things done different:

1. Multiple direct connections between two nodes were taken into account because it effects the busyness value of the airport. 10 flights from BOS to JFK were represented as (JFK BOS,10)
2. Performed comparison between 'In-mapper aggregation using 10 machine' vs 'No aggregation using 20 machines'
3. Performed comparison between 'Pagerank using 10 machines' vs 'Pagerank using 20 machines (both using In-mapper aggregation)

#### Performance on EMR:

##### Initialization: In-Mapper Aggregation Vs No Aggregation

	In-Mapper Aggregation	No Aggregation
# of machines	10	20
machine type	small	small
map output records	163749	26790152
time taken (in secs)	295	297

**Interesting Observation:** Even with half the amount of workers, in-mapper was faster

##### Pagerank: 10 machines Vs 20 machines

	10 small machines	20 small machines
Iteration #1	140	170
Iteration #2	143	168
Iteration #3	141	169
Iteration #4	142	171

**Surprising observation:** With 20 machines the runtime for each iteration increased as compared to 10 machines. The possible reason being given the small size of input (only 316 keys), it takes more time and overhead for the master to distribute the job. So, more the # of machines, more is the overhead and more is the runtime.

## TASK 4 : PLAIN MAP REDUCE - Hubs and Spokes.

### Purpose:

In this Task we are identifying the Major Hubs and Major Spokes from the Airline data of flights that we have.

We have to split the task first into two separate Jobs:

**JOB 1** : Setup the Graph Structure from the flight data. The flight data provided represents edges. Now we need to convert these edges into Node structure. Where each Node represents the following information

1. AirPort
2. Hub Value
3. Spoke Value
4. In Links
5. Out Links

**JOB 2** : Now after getting these Node structures which we obtained in Job 1, We iteratively run HITS algorithm on the available Nodes there by computing the values for Hub and Spoke.

**Format of Output:** Airport HubVal SpokeVal [List of In-Links]---[List of Out-Links]

### Sample Output of Job 1

```
ACY      1.0      1.0
ATL:1.0:1.0,MCO:1.0:1.0,ATL:1.0:1.0,---BWI:1.0:1.0,JFK:1.0:1.0,LGA:1.0:1.0,ATL:1.0:1.0,BOS:1.0:1.0,MYR:1.0:1.0,MCO:1.0:1.0,ATL:1.0:1.0,
ADK      1.0      1.0      AKN:1.0:1.0,ANC:1.0:1.0,AKN:1.0:1.0,---AKN:1.0:1.0,ANC:1.0:1.0,AKN:1.0:1.0,
```

### Sample Output of Job 2

```
ACY      0.21      1.0
ATL:1.0:1.0,MCO:1.0:1.0,ATL:1.0:1.0,---BWI:1.0:1.0,JFK:1.0:1.0,LGA:1.0:1.0,ATL:1.0:1.0,BOS:1.0:1.0,MYR:1.0:1.0,MCO:1.0:1.0,ATL:1.0:1.0,
ADK      1.0      0.132      AKN:1.0:1.0,ANC:1.0:1.0,AKN:1.0:1.0,---AKN:1.0:1.0,ANC:1.0:1.0,AKN:1.0:1.0,
```

### Pseudo-code:

Job 1:

map( Key k, Value V) // Here Key : Object , Value: Text [Each line of input]

```
{
    Flight f = parse(v)
    emit(f.origin,"Out_FLIGHT"+"", "+f.destination)
    emit(f.destination,"In_FLIGHT"+"", "+f.origin)
}
```

// Here Key : Text [ Each Airport ] Values : [List of in-link airports and out-link airports]

```
reduce(Key k, [v1,v2,v3,...])
{
    Node N = new node();
    for airport in values{
        if (airport is "In_FLIGHT")
            N.addInlink(airport)
        else
            N.addOutlink(airport)
    }
    emit(N,"")
}
```

Job 2:

#### **MAPPER:**

map(Key k, Value v) // Here Key: Object Value: Text

```
{
    Node n = parse(v)
    emit(n.nid, N )
    for(Node p in n.getInList()){
        n.setIsIn("YES")
        emit(p.nid,n)
    }
    for(Nope p in n.getOutList()){
        n.setIsIn("NO")
        emit(p.nid,n)
    }
}
```

#### **REDUCER**

```
setup()
{
    map = new Map(); // initialise hashmap
}
```

reduce(Key k, Values[v1,v2,v3,.... ]) // Here Key : Text Values:[ Node n1. Node n2,.....]

```
{
    Node M = null
    HubVal = 0
    SpokeVal = 0
    for(Node n in values)
    {
        if(isNode(N))
    }
```

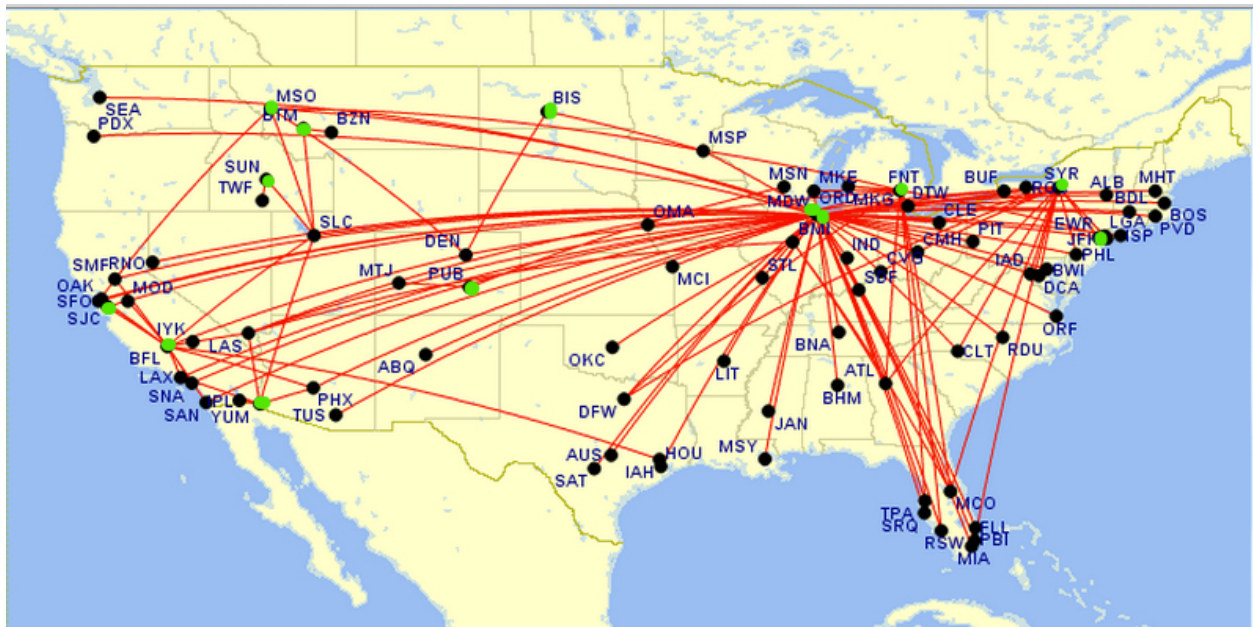
```

        M=n
    else
        if(n.isInList())
            SpokeVal +=n.getHubVal()
        else
            HubVal += n.getSpokeVal()
    }
    map.put(n)
}

cleanup()
{
    HubNorm = 0
    SpokeNorm = 0
    for(val in map)
        HubNorm += Math.pow(val.Hubval,2)
        SpokeNorm +=Math.pow(val.SpokeVal,2)
    for(val in map)
        val.HubVal = val.HubVal/HubNorm
        val.SpokeVal = val.getSpokeVal/SpokeNorm
    emit(val,"")
}

```

Top Hubs Denoted By Green Dots:



**Log:**/HubsAndSpokes/logs/

### Performance on EMR:

Setup challenges:

1. **Loading data.csv on S3 into hive table:** On our local machines we used the following command to load data.csv into our hive table:  

```
LOAD DATA LOCAL INPATH 'path_to_data.csv' INTO TABLE flight;
```

Which clearly doesn't work on aws.  
We removed `LOCAL` from the above query. Also in the create table statement we specified an additional clause `'LOCATION 's3n://finalproj-puru/table/';'` so that the table is created on S3 instead of hdfs.
2. **Adding external libraries(e.g. CSVSerDe) on EMR:** As data.csv is a csv file, hive create table needed CSVSerDe so that it parses the data properly. We added the following line to the hive code to include the external library which was stored in our S3.

```
add jar s3n://finalproj-puru/lib/csv-serde-1.1.2-0.11.0-all.jar;
```

3. Writing hive output on S3: Writing output of our hive program to S3 was also tricky. We used the following code to do it:

```
INSERT OVERWRITE DIRECTORY 's3n://finalproj-puru/hiveoutput'
```

4. **Insert into and reading from HBase:** Initially, we were storing strings as bytes into HBase table into the key column because we wanted the average delay to be sorted in increasing key order. Since that is not going to give correct results for strings, we changed that to Double. As HBase stores the records in bytes, it would return results as bytes array of Double and was not readable. Therefore, we had another map-only job that would read from HBase and write into file. The map-only job would ensure that the sorting is because of HBase row key property.
5. **HBase on EMR:** When we run HBase on EMR its persistent and we need to keep the Cluster running for the 2 jobs and deal with restoring HBase. In order to reduce these complexities, we run both the Jobs in the same Main function therefore the cluster need not be kept alive. We just needed to follow these steps and it worked:
  1. Create a jar as we do for plain MapReduce.
  2. Create a cluster on EMR and just select to add HBase. Also we can now select Auto-terminate to YES as the second mapreduce job will take the required input from HBase and create our output file in S3.
  3. The rest of the procedure remains the same as we did for plain MapReduce.

**6.** Screenshot for the successful run of hive program is given below:

Services

Edit

Purushottam Jha

Oregon

Help

Steps

Add step

Steps

Filter: All steps

Filter steps ...

3 steps (all loaded)

View all interactive jobs

View all jobs

ID	Name	Status	Start time local time (UTC-8)	Elapsed time	Log files	Actions
s-3ABM4JW6WNHVO	finalHive9	Completed	2013-12-06 20:09:56	15 minutes	<a href="#">View logs</a>	<a href="#">View jobs</a>
<p><b>JAR location:</b> s3://us-west-2.elasticmapreduce/lib/script-runner/script-runner.jar</p> <p><b>Main class:</b> None</p> <p><b>Arguments:</b> s3://us-west-2.elasticmapreduce/lib/hive/hive-script, --run-hive-script, --hive-versions, 0.11.0.1, --args, -f, s3n://finalproj-puru/hivequery.q</p> <p><b>Action on failure:</b> Terminate cluster</p>						
s-33SILY6MN351H	Setup hive	Completed	2013-12-06 20:08:30	1 minute	<a href="#">View logs</a>	<a href="#">View jobs</a>
<p><b>JAR location:</b> s3://us-west-2.elasticmapreduce/lib/script-runner/script-runner.jar</p> <p><b>Main class:</b> None</p> <p><b>Arguments:</b> s3://us-west-2.elasticmapreduce/lib/hive/hive-script, --base-path, s3://us-west-2.elasticmapreduce/lib/hive/, --install-hive, --hive-versions, 0.11.0.1</p> <p><b>Action on failure:</b> Terminate cluster</p>						
s-3240QF09DC0LV	Setup hadoop debugging	Completed	2013-12-06 20:07:53	37 seconds	<a href="#">View logs</a>	<a href="#">View jobs</a>
<p><b>JAR location:</b> s3://us-west-2.elasticmapreduce/lib/script-runner/script-runner.jar</p> <p><b>Main class:</b> None</p> <p><b>Arguments:</b> s3://us-west-2.elasticmapreduce/lib/state-pusher/0.1/fetch</p> <p><b>Action on failure:</b> Terminate cluster</p>						

7. Screenshots for successful run of HBase Task1 is given below

Services

▼

Edit

▼

Rushill

▼

Oregon

▼

Help

▼

Elastic MapReduce

▼

Cluster List

>

Cluster Details

EMR Help

Add step

Resize

Clone

Terminate

Cluster: Hbase

Status: **Running**

Running step

ID: j-23SQKU8R0ULL0

Key name: --

Auto-terminate: Yes

Subnet ID: subnet-d1d3590a

Creation date: 2013-12-09 13:46:13 (local time - UTC-5)

IAM role: --

End date: --

Visible to all users: None

Elapsed time: 6 minutes

AMI version: 2.4.2

Master public DNS name: ec2-54-201-175-115.us-west-2.compute.amazonaws.com

Hadoop distribution: Amazon 1.0.3

Log URI: s3n://hbasefinal/logs/

Termination protection: On [Change](#)

Applications: HBase 0.92.0

Tags: -- [View All / Edit](#)

Monitoring

Hardware Configuration

Steps

Add step

Steps

Filter: **All steps**

▼

Filter steps ...

3 steps (all loaded)

[View all interactive jobs](#)
[View all jobs](#)

ID	Name	Status	Start time ▼ local time (UTC-5)	Elapsed time	Log files	Actions
s-3HEYBG1K66NIR	Hbase1 2 Large	Running	2013-12-09 13:49:45	3 minutes	<a href="#">View logs</a>	<a href="#">View jobs</a>
s-2LKV80B99MTW	Start HBase	Completed	2013-12-09 13:49:38	7 seconds	<a href="#">View logs</a>	<a href="#">View jobs</a>
s-1R6G66QJHKTBE	Setup hadoop debugging	Completed	2013-12-09 13:49:24	14 seconds	<a href="#">View logs</a>	<a href="#">View jobs</a>

Bootstrap Actions

Elastic MapReduce

Cluster List

Cluster Details

EMR Help

Add step

Refresh

Clone

Terminate

Cluster: Hbase

Status: Terminating Steps completed

ID: j-239QKUBRDULL0

Key name: --

Auto-terminate: Yes

Subnet ID: subnet-d1d359ba

Creation date: 2013-12-09 13:46:13 (local time - UTC-5)

IAM role: --

End date: --

Visible to all users: None

Elapsed time: 12 minutes

AMI version: 2.4.2

Master public DNS name: ec2-54-201-175-115.us-west-2.compute.amazonaws.com

Hadoop distribution: Amazon 1.0.3

Log URI: s3n://hbasefinal/logs/

Termination protection: On

Applications: HBase 0.92.0

Tags: --

Monitoring

Hardware Configuration

Steps

Add step

Steps

Filter: All steps

Filter steps ...

3 steps (all loaded)

View all interactive jobs

View all jobs

ID	Name	Status	Start time local time (UTC-5)	Elapsed time	Log files	Actions
s-3HEYBG1K66NIR	Hbase1 2 Large	Completed	2013-12-09 13:49:45	8 minutes	<a href="#">View logs</a>	<a href="#">View jobs</a>
s-2UKV8D9G9MTW	Start HBase	Completed	2013-12-09 13:49:38	7 seconds	<a href="#">View logs</a>	<a href="#">View jobs</a>
s-1R6G66DUWKTBE	Setup hadoop debugging	Completed	2013-12-09 13:49:24	14 seconds	<a href="#">View logs</a>	<a href="#">View jobs</a>

Bootstrap Actions



## 8. Screenshots for successful run of HBase Task2 is given below:

This screenshot shows the AWS Elastic MapReduce console for a cluster in the us-west-2 region. The cluster is named 'j-3G3'. The 'Steps' section shows three steps: 'Hbase2 2 large' (Running), 'Start HBase' (Completed), and 'Setup hadoop debugging' (Completed). The 'Hbase2 2 large' step is currently running, with a start time of 2013-12-09 13:56:35 and an elapsed time of 1 minute. The console also displays monitoring, hardware configuration, and bootstrap actions sections.

Master public DNS name: --  
Log URI: s3n://flights-project/logs/  
Applications: HBase 0.92.0  
Hadoop distribution: Amazon 1.0.3  
Termination protection: On  
Tags: --

Monitoring  
Hardware Configuration  
Steps

Add step

Filter: All steps 3 steps (all loaded)

ID	Name	Status	Start time local time (UTC-5)	Elapsed time	Log files	Actions
s-314S1XMGC TK3A	Hbase2 2 large	Running	2013-12-09 13:56:35	1 minute	<a href="#">View logs</a>	<a href="#">View jobs</a>
s-CNNAD1GE 9IG5	Start HBase	Completed	2013-12-09 13:56:28	7 seconds	<a href="#">View logs</a>	<a href="#">View jobs</a>
s-3NM8UN2Z OWO3E	Setup hadoop debugging	Completed	2013-12-09 13:56:14	14 seconds	<a href="#">View logs</a>	<a href="#">View jobs</a>

Bootstrap Actions

© 2008 - 2013, Amazon Web Services, Inc. or its affiliates. All rights reserved. [Privacy Policy](#) [Terms of Use](#) [Feedback](#)

This screenshot shows the AWS Elastic MapReduce console for the same cluster, but the 'Hbase2 2 large' step is now 'Completed'. The start time remains 2013-12-09 13:56:35, but the elapsed time has increased to 9 minutes. The other two steps remain completed. The console layout is identical to the previous screenshot, showing monitoring, hardware configuration, and bootstrap actions sections.

Master public DNS name: --  
Log URI: s3n://flights-project/logs/  
Applications: HBase 0.92.0  
Hadoop distribution: Amazon 1.0.3  
Termination protection: On  
Tags: --

Monitoring  
Hardware Configuration  
Steps

Add step

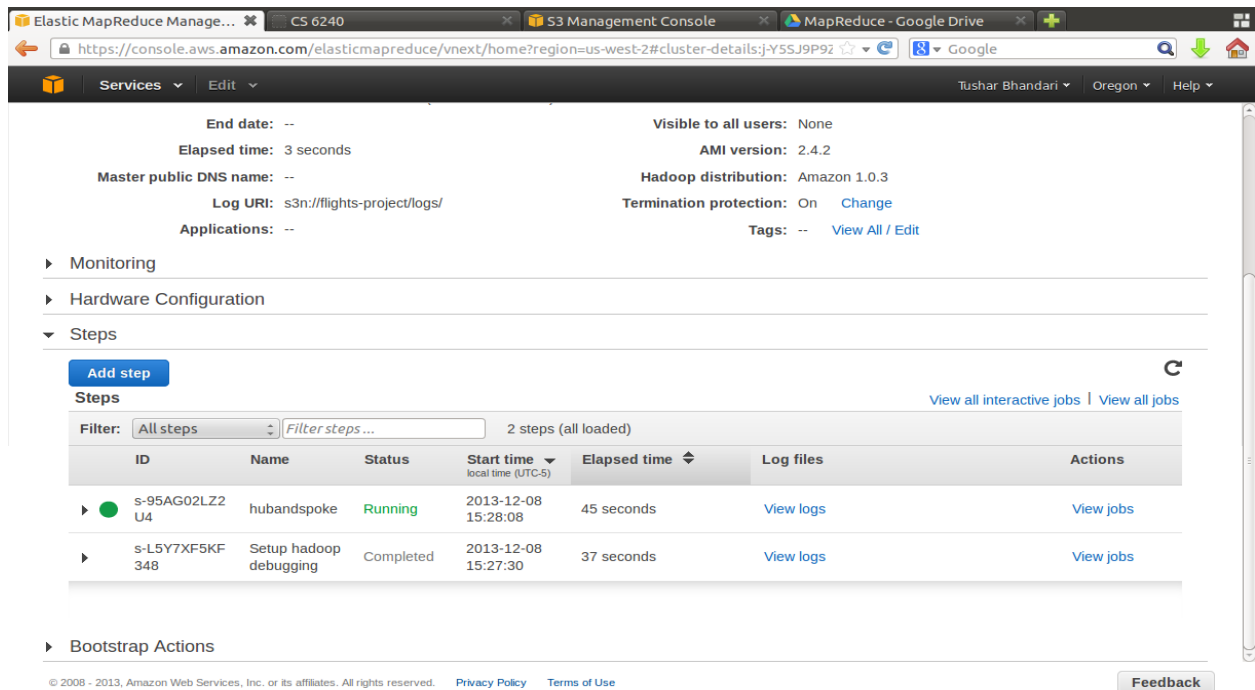
Filter: All steps 3 steps (all loaded)

ID	Name	Status	Start time local time (UTC-5)	Elapsed time	Log files	Actions
s-314S1XMGC TK3A	Hbase2 2 large	Completed	2013-12-09 13:56:35	9 minutes	<a href="#">View logs</a>	<a href="#">View jobs</a>
s-CNNAD1GE 9IG5	Start HBase	Completed	2013-12-09 13:56:28	7 seconds	<a href="#">View logs</a>	<a href="#">View jobs</a>
s-3NM8UN2Z OWO3E	Setup hadoop debugging	Completed	2013-12-09 13:56:14	14 seconds	<a href="#">View logs</a>	<a href="#">View jobs</a>

Bootstrap Actions

© 2008 - 2013, Amazon Web Services, Inc. or its affiliates. All rights reserved. [Privacy Policy](#) [Terms of Use](#) [Feedback](#)

## 9. Screenshots for successful run of Hubs and Spokes is given below:



End date: -- Visible to all users: None  
Elapsed time: 3 seconds AMI version: 2.4.2  
Master public DNS name: -- Hadoop distribution: Amazon 1.0.3  
Log URI: s3n://flights-project/logs/ Termination protection: On [Change](#)  
Applications: -- Tags: -- [View All / Edit](#)

Monitoring

Hardware Configuration

Steps

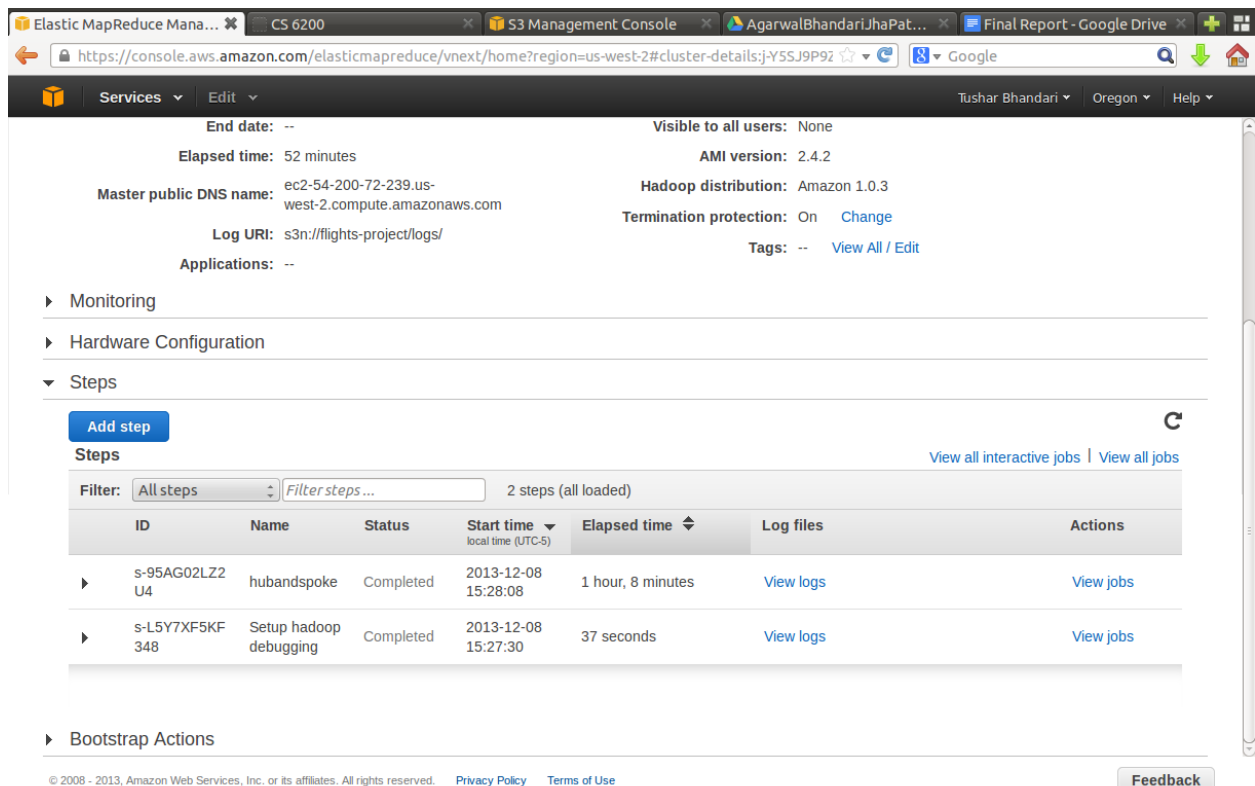
[Add step](#) [View all interactive jobs](#) | [View all jobs](#)

Filter: [All steps](#)  2 steps (all loaded)

ID	Name	Status	Start time local time (UTC-5)	Elapsed time	Log files	Actions
s-95AG02LZ2U4	hubandspoke	Running	2013-12-08 15:28:08	45 seconds	<a href="#">View logs</a>	<a href="#">View jobs</a>
s-L5Y7XF5KF348	Setup hadoop debugging	Completed	2013-12-08 15:27:30	37 seconds	<a href="#">View logs</a>	<a href="#">View jobs</a>

Bootstrap Actions

© 2008 - 2013, Amazon Web Services, Inc. or its affiliates. All rights reserved. [Privacy Policy](#) [Terms of Use](#) [Feedback](#)



End date: -- Visible to all users: None  
Elapsed time: 52 minutes AMI version: 2.4.2  
Master public DNS name: ec2-54-200-72-239.us-west-2.compute.amazonaws.com Hadoop distribution: Amazon 1.0.3  
Log URI: s3n://flights-project/logs/ Termination protection: On [Change](#)  
Applications: -- Tags: -- [View All / Edit](#)

Monitoring

Hardware Configuration

Steps

[Add step](#) [View all interactive jobs](#) | [View all jobs](#)

Filter: [All steps](#)  2 steps (all loaded)

ID	Name	Status	Start time local time (UTC-5)	Elapsed time	Log files	Actions
s-95AG02LZ2U4	hubandspoke	Completed	2013-12-08 15:28:08	1 hour, 8 minutes	<a href="#">View logs</a>	<a href="#">View jobs</a>
s-L5Y7XF5KF348	Setup hadoop debugging	Completed	2013-12-08 15:27:30	37 seconds	<a href="#">View logs</a>	<a href="#">View jobs</a>

Bootstrap Actions

© 2008 - 2013, Amazon Web Services, Inc. or its affiliates. All rights reserved. [Privacy Policy](#) [Terms of Use](#) [Feedback](#)

## Conclusion:

Here is a summary of our main results and contributions for every task:

**1. HBase:** The task focuses on calculating average arrival delay for airlines and airport. Airports on the east coast have more delays followed by airports on the west coast followed by airports in the central part of the United States. This can be because of the following reasons:

- a. Weather conditions on the east coast are more severe compared to the other parts of the United States.
- b. Also airports are busier in the eastern part compared to west and central US.

For airlines, Aloha Air Cargo has the least average delay whereas JetBlue airways has the maximum average delay.

**2. Hive and PigLatin:** This task focuses on finding three legged flights of the form BOS-X-Y-BOS. We used both Hive and PigLatin for the same task and PigLatin performed better compared to Hive. One of the reasons is PigLatin directly reads data from file and converts the job to MapReduce program. On the other hand, there is some time spent on creating a table and loading data into the tables in Hive and then finally executing the query.

### **3. Page Rank:**

Summary:

1. Number of iteration for convergence is inversely proportional to the alpha value
2. Pagerank value has a significant effect on the number of input links even if from same airport. Hence, it would be unwise to count 10 flights from BOS to JFK as only 1 link.
3. The number of workers have to be in proportion with the input. For smaller inputs, if the number of workers are high, it will degrade the performance of the overall operation.

Future Extension:

1. Analyze the performance of Initial page ranking and graph creation by using combiner.
2. Use the value of pagerank and analyze the effect of it on the delay/cancellation of flights inbound/outbound from that airport.
3. Pagerank can also be used by Airline companies to decide on the connecting route for a new flight line.

**4. Hubs and Spokes:** This task focuses on finding the hubs and spokes for the given dataset. We used HITS algorithm to determine hub and spoke values for each airport. We also plotted this on a map to make it more intuitive. The following were the results:

Airport with the best hub value is : Syracuse, NY

Airport with the best spoke value is: Colorado Springs, CO