CS 6240 : Assignment 3

Name: *Mahesh Bhandarkar*
Class: *Tuesday 1:35 PM*
Homework: *HW 3*

PLAIN : Pseudo Code

```
Global Counters
 SumOfTotalDelays
 NumberOfDelays

map(String key, String value):

// Parse incoming data, with comma(,) as delimiter
// Check for the Valid Leg-1 and Leg-2

// Consider only flights of ::
// Origin==’ORD’ AND Dest!=’JFK’ OR Origin!=’ORD’ AND Dest==’JFK’

// Check if FlightDate is between May2007-June2008

// key: <Origin_OR_Dest>-<FlightDate>
 if (Origin == ‘ORD’)
    key: <Dest>-<FlightDate>
 else
    key: <Origin>-<FlightDate>

// value: <Origin>-<DepTime>-<ArrTime>-<Delay>
          emit(key, value)

reduce(String key, Iterator values):
 list_ORD
 list_Others

 for (val in Values)
{
   if (key[0]==’ORD’)
       list_ORD.add(val)
   else
       list_Others.add(val)
}
```

```
for each x_Entries in list_ORD:
   for each y_Entries in list_ORD:
      // Check ArrivalTime < DepartureTime at Terminal X
      // If true: Extract Delay from value from
      //          ..both (c_Entries and y_Entries) and sum them
      //           third element in ‘Entries’ delimited by ‘-’ is the delay
      result = x_Entries[3] + y_Entries[3]
      SumOfTotalDelays.increment(result);
      NumberOfDelays.increment(1);

Main:
 Double sum = getGlobalCounter.SumOfTotalDelays;
 Double total = getGlobalCounter.NumberOfDelays;
 Double avg = sum/total;
 System.out.println(“Average Flight Delay = ” + avg);
```

PLAIN : Source Code

```java
package Assignment3;


import java.io.IOException;
import java.lang.reflect.Array;
import java.text.SimpleDateFormat;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;


public class FlightDataCompChallenge {

        public enum GlobalCounters
        {
                SUM_TOTAL_OF_DELAYS, // Stores the 'Summation' of all Flight Delays
                NUMBER_OF_DELAYS     // Stores the 'Total number' of Flight Delays
        }

        public static class FlightDataMapper extends Mapper<Object, Text, Text, Text>
        {
            public void map(Object key, Text value, Context context) throws IOException, InterruptedException
```

```
{
        // Split input by comma & store in array
        String[] anEntry = value.toString().split(",");

        // Extract required values from anEntry
        String
            flightYear = anEntry[0],
            flightMonth = anEntry[2],
            flightDay = anEntry[3],
                flightDate = anEntry[5],
                originCityCode = anEntry[11].replace("\"", ""),
                destCityCode = anEntry[18].replace("\"", ""),
                departureTime = anEntry[26],
                arrivalTime = anEntry[37],
                delay = anEntry[39],
                cancelled = anEntry[43],
                diverted = anEntry[45];

        boolean isWithinDateRange = false;
        try
        {
                String sArr[]=flightDate.split("-");
                Date newDate=new Date(sArr[0] + "/" + sArr[1] + "/" + sArr[2]);
                /*
                 *  Checking if the 'flightDate' lies
                 *  between end of may'07 and start of june'08.
                */
                isWithinDateRange = newDate.after(new Date("2007/05/31"))
                                        && newDate.before(new Date("2008/06/01"));
        }
        catch (Exception e)
        {
                e.printStackTrace();
        }

        /* LEG-1
         * Checking whether flight Leaves ORD, but does NOT land at JFK
         * */
        Boolean isValidLEG_1 = (originCityCode.equals("ORD")
                                        && !destCityCode.equals("JFK"));

        /* LEG-2
         * Checking whether flight lands at JFK, but NOT originated from ORD
         * */
        Boolean isValidLEG_2 = (!originCityCode.equals("ORD")
                                        && destCityCode.equals("JFK"));

        /*
         * Checking whether flight is NOT Cancelled and NOT Delayed
         * */
        if ((isValidLEG_1 || isValidLEG_2)
                        && isWithinDateRange
                        && cancelled.equals("0.00")
                        && diverted.equals("0.00"))
        {
                /*
                 * aKey would be one of :
                 *
                 *    -- 'destCityCode - flightDate'
                 *    -- 'originCityCode - flightDate'
                 *
                 * .. with the character '-' being a delimiter
                 *
                 * */
                Text aKey;

                if (originCityCode.equals("ORD"))
                        // -- 'destCityCode - flightDate'
                        aKey = new Text(destCityCode + "-" + flightDate);
                        else
                                // -- 'originCityCode | flightDate'
                                aKey = new Text(originCityCode + "-" + flightDate);

                        // Value ::  originCityCode - departureTime - arrivalTime - delay
                        //  .. with the character '-' being a delimiter
                        String valString = originCityCode + "-" + departureTime
                                                                 + "-" + arrivalTime
                                                                 + "-" + delay;
                        Text val = new Text(valString);
                        context.write(aKey, val);
                }
        }
}
```

```
public static class DelayCalcReducer extends Reducer<Text, Text, Text, FloatWritable>
{
        private float fResult;

        public void reduce(Text key, Iterable<Text> values, Context context)
                                                      throws IOException, InterruptedException
        {
                // list_ORD    : stores flights originating from ORD to terminal-X
                ArrayList<String> list_ORD = new ArrayList<String>();

                // list_Others : stores all others
                ArrayList<String> list_Others = new ArrayList<String>();

                for (Text aVal : values)
                {
                        String strVal = aVal.toString();
                        String[] arrValTokens = strVal.split("-");

                        if (arrValTokens[0].equals("ORD"))
                        {
                                list_ORD.add(strVal);
                        }
                        else
                        {
                                list_Others.add(strVal);
                        }
                }

                /* Check for flight arrival time to terminal-X
                 * to be lesser than departure time to JFK
                 * */
                String[] entry_ORD, entry_Others;

                for (int i = 0; i < list_ORD.size(); i++)
                {
                        entry_ORD = list_ORD.get(i).split("-");

                        for (int j = 0; j < list_Others.size(); j++)
                        {
                                entry_Others = list_Others.get(j).split("-");
                                // Check if time is valid
                                if (isValidTime(entry_ORD, entry_Others))
                                {
                                        fResult = Float.parseFloat(entry_ORD[3])
                                                        + Float.parseFloat(entry_Others[3]);
                                        // Increment Global counters
                                    context.getCounter(GlobalCounters.SUM_TOTAL_OF_DELAYS)
                                                            .increment((long) fResult);
                                        context.getCounter(GlobalCounters.NUMBER_OF_DELAYS)
                                                                    .increment(1);
                                }
                        }
                }
        }

        private boolean isValidTime(String[] aEntry_ORD, String[] aEntry_Others)
        {

                boolean result = !(aEntry_ORD[2].trim().length() == 0 // arrival/departure Time not Empty
                                            || aEntry_Others[1].trim().length() == 0)
                                            /* Arrival Time, less than Departure Time*/
                                && (Integer.parseInt(aEntry_ORD[2].replace("\"", ""))
                                    < Integer.parseInt(aEntry_Others[1].replace("\"", "")));
                return result;
        }
    }


    public static void main(String[] args) throws Exception
    {
        Configuration conf = new Configuration();
        String[] otherArgs = new GenericOptionsParser(conf, args)
                        .getRemainingArgs();
        if (otherArgs.length != 2)
        {
                System.err.println("Usage: FlightDataCompChallenge <input> <output>");
                System.exit(2);
        }
        Job job = new Job(conf, "Average Flight Delay Calculator");
        job.setJarByClass(FlightDataCompChallenge.class);
        job.setMapperClass(FlightDataMapper.class);
        // job.setCombinerClass(DelayCalcReducer.class);
        job.setReducerClass(DelayCalcReducer.class);
        /* 10 Reduce Tasks*/
        job.setNumReduceTasks(10);
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(Text.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(FloatWritable.class);
        FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
        FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
        if (job.waitForCompletion(true))
        {
                // Get all the Global counters
                Counters globalCounters = job.getCounters();

                double SumTotalOfDelays = (double)globalCounters
                                                .findCounter(GlobalCounters.SUM_TOTAL_OF_DELAYS)
                                                .getValue();
                double NumberOfDelays = (double)globalCounters
                                                .findCounter(GlobalCounters.NUMBER_OF_DELAYS)
                                                .getValue();
                // Console out the result
                System.out.println("Average Flight Delay = " + SumTotalOfDelays/NumberOfDelays);
                System.exit(0);
        }
        System.exit(1);
    }
}
```

JOIN-FIRST.v1: Source Code

```
REGISTER file:/home/hadoop/lib/pig/piggybank.jar;
 DEFINE CSVLoader org.apache.pig.piggybank.storage.CSVLoader;

 -- Set 10 Reducers
 SET default_parallel 10;

 -- Define the source of Data
 -- 'local source' : /Users/mahesh/Documents/workspace/data/mapreduce/3/actual/data.csv
 -- 'AWS source' : s3://hw3flightchallenge/data/data.csv/
 Flights1 = LOAD 's3://hw3flightchallenge/data/data.csv/' USING CSVLoader;
 Flights2 = LOAD 's3://hw3flightchallenge/data/data.csv/' USING CSVLoader;



 -- Get Required Fields from the CSV Data
 Flights1 = FOREACH Flights1 GENERATE (chararray)$5 as flightdate,
                                      (chararray)$11 as origin,
                                      (chararray)$17 as dest,
                                      (int) $24 as depTime,
                                      (int) $35 as arrTime,
                                      (float)$37 as arrDelayMins,
                                      (int) $41 as cancelled,
                                      (int) $43 as diverted;

 Flights2 = FOREACH Flights2 GENERATE (chararray)$5 as flightdate,
                                      (chararray)$11 as origin,
                                      (chararray)$17 as dest,
                                      (int) $24 as depTime,
                                      (int) $35 as arrTime,
                                      (float)$37 as arrDelayMins,
                                      (int) $41 as cancelled,
                                      (int) $43 as diverted;

 -- Filter out the Data that that contains cancelled/diverted flight data
 Flights1 = FILTER Flights1 BY (cancelled != 1) AND (diverted!=1);
 Flights2 = FILTER Flights2 BY (cancelled != 1) AND (diverted!=1);


 -- We have 2 Legs in the Problem Statement
 -- Flights1 = Flight data Originating from ORD but Landing on Airport terminal-X
 -- Flights1 = Flight data Originating from Airport terminal-X but Landing on JFK
 Flights1 = FILTER Flights1 BY (origin == 'ORD' AND dest!='JFK');
 Flights2 = FILTER Flights2 BY (origin != 'ORD' AND dest=='JFK');



 -- ------------------------------- JOIN -----------------------------------
 -- We Join Flights1(LEG1) and Flights2(LEG2) on FlightDate
 joinOnDate = JOIN Flights1 BY (dest, flightdate), Flights2 BY (origin, flightdate);

 -- We check whether the arrival Time is lesser than the Departure time, on Terminal-X
 -- Eg: consider 1st tuple in arrLessThanDp will be ::
 -- (2007-01-01,ORD,ABE,1628,1918,0.0,0,0,2007-01-01,ABE,JFK,2150,2300,45.0,0,0)
 -- arrTime < depTime :: $4 < $11 :: 1918 < 2150 ===> true
 arrLessThanDep = FILTER joinOnDate BY $4 < $11;
  -- ------------------------------- --- -----------------------------------
-- ------------------------------- FILTER -----------------------------------

-- Now we just filter out records having dates between (end of may'07 and start of june'08)
-- $0 is the position for the Flights1 Flight's date, for arrival to terminal-X
-- $8 is the position for the Flights2 Flight's date, for departure from terminal-X
lastJoin = FILTER arrLessThanDep BY ToDate($0,'yyyy-MM-dd') > ToDate('2007-05-31', 'yyyy-MM-dd')
                                                AND ToDate($0, 'yyyy-MM-dd') < ToDate('2008-06-01', 'yyyy-MM-dd')
                                                AND ToDate($8, 'yyyy-MM-dd') > ToDate('2007-05-31', 'yyyy-MM-dd')
                                                AND ToDate($8, 'yyyy-MM-dd') < ToDate('2008-06-01', 'yyyy-MM-dd');

result = FOREACH lastJoin GENERATE (float)($5 + $13) as sumOfDelays;

 -- ------------------------------ --- -----------------------------------

-- Group the final 'result' Relation
gResult = GROUP result ALL;

-- Iterate through the grouped result and Run Aggregate function AVG on it
average_delay = FOREACH gResult GENERATE AVG(result.sumOfDelays);

--dump average_delay;
STORE average_delay INTO 's3://hw3flightchallenge/outputs/1/' USING PigStorage();
```

JOIN-FIRST.v1 : Source Code

```
REGISTER file:/home/hadoop/lib/pig/piggybank.jar;
 DEFINE CSVLoader org.apache.pig.piggybank.storage.CSVLoader;

 -- Set 10 Reducers
 SET default_parallel 10;

 -- Define the source of Data
 -- 'local source' : /Users/mahesh/Documents/workspace/data/mapreduce/3/actual/data.csv
 -- 'AWS source' :
 Flights1 = LOAD 's3://hw3flightchallenge/data/data.csv/' USING CSVLoader;
 Flights2 = LOAD 's3://hw3flightchallenge/data/data.csv/' USING CSVLoader;



 -- Get Required Fields from the CSV Data
 Flights1 = FOREACH Flights1 GENERATE (chararray)$5 as flightdate,
                                          (chararray)$11 as origin,
                                          (chararray)$17 as dest,
                                          (int) $24 as depTime,
                                          (int) $35 as arrTime,
                                          (float)$37 as arrDelayMins,
                                          (int) $41 as cancelled,
                                          (int) $43 as diverted;

 Flights2 = FOREACH Flights2 GENERATE (chararray)$5 as flightdate,
                                          (chararray)$11 as origin,
                                          (chararray)$17 as dest,
                                          (int) $24 as depTime,
                                          (int) $35 as arrTime,
                                          (float)$37 as arrDelayMins,
                                          (int) $41 as cancelled,
                                          (int) $43 as diverted;

 -- Filter out the Data that that contains cancelled/diverted flight data
 Flights1 = FILTER Flights1 BY (cancelled != 1) AND (diverted!=1);
 Flights2 = FILTER Flights2 BY (cancelled != 1) AND (diverted!=1);


 -- We have 2 Legs in the Problem Statement
 -- Flights1 = Flight data Originating from ORD but Landing on Airport terminal-X
 -- Flights1 = Flight data Originating from Airport terminal-X but Landing on JFK
 Flights1 = FILTER Flights1 BY (origin == 'ORD' AND dest!='JFK');
 Flights2 = FILTER Flights2 BY (origin != 'ORD' AND dest=='JFK');



 -- ------------------------------- JOIN -----------------------------------
 -- We Join Flights1(LEG1) and Flights2(LEG2) on FlightDate
 joinOnDate = JOIN Flights1 BY (dest, flightdate), Flights2 BY (origin, flightdate);

 -- We check whether the arrival Time is lesser than the Departure time, on Terminal-X
 arrLessThanDep = FILTER joinOnDate BY $4 < $11;
  -- ------------------------------- --- -----------------------------------



 -- ------------------------------- FILTER -----------------------------------

 -- Now we just filter out records having dates between (end of may'07 and start of june'08)
 -- $0 is the position for the Flights1 Flight's date, for arrival to terminal-X
 -- $8 is the position for the Flights2 Flight's date, for departure from terminal-X (*Removed*)
 lastJoin = FILTER arrLessThanDep BY ToDate($0,'yyyy-MM-dd') > ToDate('2007-05-31', 'yyyy-MM-dd')
                                                                           AND ToDate($0, 'yyyy-MM-dd') < ToDate('2008-06-
01', 'yyyy-MM-dd');
```

```
result = FOREACH lastJoin GENERATE (float)($5 + $13) as sumOfDelays;

 -- ------------------------------- --- -----------------------------------

-- Group the final 'result' Relation
gResult = GROUP result ALL;

-- Iterate through the grouped result and Run Aggregate function AVG on it
average_delay = FOREACH gResult GENERATE AVG(result.sumOfDelays);

--dump average_delay;
STORE average_delay INTO 's3://hw3flightchallenge/outputs/2s3://hw3flightchallenge/data/data.csv//' USING PigStorage();
```

FILTER-FIRST : Source Code

```
REGISTER file:/home/hadoop/lib/pig/piggybank.jar;
 DEFINE CSVLoader org.apache.pig.piggybank.storage.CSVLoader;

 -- Set 10 Reducers
 SET default_parallel 10;

 -- Define the source of Data
 -- 'local source' : /Users/mahesh/Documents/workspace/data/mapreduce/3/actual/data.csv
 -- 'AWS source' :
 Flights1 = LOAD 's3://hw3flightchallenge/data/data.csv/' USING CSVLoader;
 Flights2 = LOAD 's3://hw3flightchallenge/data/data.csv/' USING CSVLoader;



 -- Get Required Fields from the CSV Data
 Flights1 = FOREACH Flights1 GENERATE (chararray)$5 as flightdate,
                                      (chararray)$11 as origin,
                                      (chararray)$17 as dest,
                                      (int) $24 as depTime,
                                      (int) $35 as arrTime,
                                      (float)$37 as arrDelayMins,
                                      (int) $41 as cancelled,
                                      (int) $43 as diverted;

 Flights2 = FOREACH Flights2 GENERATE (chararray)$5 as flightdate,
                                      (chararray)$11 as origin,
                                      (chararray)$17 as dest,
                                      (int) $24 as depTime,
                                      (int) $35 as arrTime,
                                      (float)$37 as arrDelayMins,
                                      (int) $41 as cancelled,
                                      (int) $43 as diverted;

 -- Filter out the Data that that contains cancelled/diverted flight data
 Flights1 = FILTER Flights1 BY (cancelled != 1) AND (diverted!=1);
 Flights2 = FILTER Flights2 BY (cancelled != 1) AND (diverted!=1);




  -- ------------------------------- FILTER -----------------------------------
 -- We have 2 Legs in the Problem Statement
 -- Flights1 = Flight data Originating from ORD but Landing on Airport terminal-X
 -- Flights2 = Flight data Originating from Airport terminal-X but Landing on JFK
 Flights1 = FILTER Flights1 BY (origin == 'ORD' AND dest!='JFK')
                        AND ToDate(flightdate, 'yyyy-MM-dd') > ToDate('2007-05-31', 'yyyy-MM-dd')
                        AND ToDate(flightdate, 'yyyy-MM-dd') < ToDate('2008-06-01', 'yyyy-MM-dd');

 Flights2 = FILTER Flights2 BY (origin != 'ORD' AND dest=='JFK')
                        AND ToDate(flightdate, 'yyyy-MM-dd') > ToDate('2007-05-31', 'yyyy-MM-dd')
                        AND ToDate(flightdate, 'yyyy-MM-dd') < ToDate('2008-06-01', 'yyyy-MM-dd');
  -- ------------------------------- FILTER -----------------------------------




 -- ------------------------------- JOIN -----------------------------------
 -- We Join Flights1(LEG1) and Flights2(LEG2) on FlightDate
 joinOnDate = JOIN Flights1 BY (dest, flightdate), Flights2 BY (origin, flightdate);

 -- We check whether the arrival Time is lesser than the Departure time, on Terminal-X
 arrLessThanDep = FILTER joinOnDate BY $4 < $11;
  -- ------------------------------- JOIN -----------------------------------
 -- Now we just filter out records having dates between (end of may'07 and start of june'08)

 result = FOREACH arrLessThanDep GENERATE (float)($5 + $13) as sumOfDelays;

 -- Group the final 'result' Relation
 gResult = GROUP result ALL;

 -- Iterate through the grouped result and Run Aggregate function AVG on it
 average_delay = FOREACH gResult GENERATE AVG(result.sumOfDelays);

 --dump average_delay;
 STORE average_delay INTO 's3://hw3flightchallenge/outputs/3/' USING PigStorage();
```

Performance Comparison

Jobs	Plain	JoinFirst v1	JoinFirst v2	FilterFirst
Run Time (seconds)	241	583	568	550

Did your PLAIN program beat Pig?

YES!

Analyzing the performance of each program, on the EMR, apparently PLAIN-version of the program beats all the others. We see that it is more than twice as fast than the PIG-implementations.

How did the differences in the Pig programs affect runtime? Can you explain why these runtime results happened?

Apparently, there is not a significant difference in the run-times of the PIG programs.

Analyzing the run times, we see that JoinFirst.v1 is slowest of all the PIG implementations, JoinFirst.v2 takes the second place, and finally FilterFirst, is the fastest of the implementations.

FilterFirst, I believe is the fastest of them because it has the least records to process, as compared to the other programs.

Some times Joining Tables/Relations, results in a increased set of records in the resulting joined output.

I believe, JoinFirst.v2 was faster than v1. Because the Output-Relation, from this join had lesser rows to be processed, as compared to the JoinFirst.v1 implementation of join.

Average Delay Calculated by each program:

Jobs	**Plain**	**JoinFirst v1**	**JoinFirst v2**	**FilterFirst**
Delay (seconds)	50.67124150519758	50.67124150519758	50.67124150519758	50.67124150519758