CS 6240

HW3

Rushil Patel

1) PLAIN [Source Code]

Pseudo Code :

```
Map1(Object key, Text value)
{
    // Parse input data
    Flight f = new Flight (value);

    //Check Data
    if( f ! Diverted && f! Cancelled && f.Origin ='ORD'
      && f.Destination !='JFK' && f.isFlightDateValid)
      {
          f.setleg =1;
          Emit([date,Destination],f);
      }
}

Map2(Object key, Text value)
{
    // Parse input data
    Flight f = new Flight (value);

    //Check Data
    if( f ! Diverted && f! Cancelled && f.Origin  !='ORD'
      && f.Destination =='JFK' && f.isFlightDateValid)
      {
          f.setleg = 2
          Emit([date,Destination],f);
      }
}
```

```
Redeuce(Text Key, Text Value[])
{
        Flight f1 [] = all flights with leg=1 in Value[] from map1
        Flight f2[]= all flights  with leg=2 in Value[] from map2

        For(Flight x in f1)
        {
             For(Flight y in f2)
        {
             if(x.arrivaltime < f2.destination time)
{
        Increment  total delay counter by ( x.delaymins + y.delaymins;

        Increment total flight counter by 1; )
}
        }
        }
 }
```

PROGRAM :

```
/**
 *  Licensed under the Apache License, Version 2.0 (the "License");
 *  you may not use this file except in compliance with the License.
 *  You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 *  Unless required by applicable law or agreed to in writing, software
 *  distributed under the License is distributed on an "AS IS" BASIS,
 *  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 *  See the License for the specific language governing permissions and
 *  limitations under the License.
 */

import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Counters;
```

```
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.MultipleInputs;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;
import au.com.bytecode.opencsv.CSVParser;

// Reference class to convert each link of input
// into object for easy access
class Flight1 {

	public int getYear() {
		return Year;
	}

	public void setYear(int year) {
		Year = year;
	}

	public int getMonth() {
		return Month;
	}

	public void setMonth(int month) {
		Month = month;
	}

	public String getFlightDate() {
		return FlightDate;
	}

	public void setFlightDate(String flightDate) {
		FlightDate = flightDate;
	}

	public String getOrigin() {
		return Origin;
	}

	public void setOrigin(String origin) {
		Origin = origin;
	}

	public String getDesination() {
		return Desination;
	}

	public void setDesination(String desination) {
```

```
		Desination = desination;
}

public int getDepartTime() {
		return DepartTime;
}

public void setDepartTime(int departTime) {
		DepartTime = departTime;
}

public int getArrivalTime() {
		return ArrivalTime;
}

public void setArrivalTime(int arrivalTime) {
		ArrivalTime = arrivalTime;
}

public float getArrivalDelayMins() {
		return ArrivalDelayMins;
}

public void setArrivalDelayMins(float arrivalDelayMins) {
		ArrivalDelayMins = arrivalDelayMins;
}

public float getCancelled() {
		return Cancelled;
}

public void setCancelled(float cancelled) {
		Cancelled = cancelled;
}

public int getLeg() {
		return leg;
}

public void setLeg(int leg) {
		this.leg = leg;
}

public float getDiverted() {
		return Diverted;
}

public void setDiverted(float diverted) {
		Diverted = diverted;
}
```

```
public boolean isComplete() {
	return isComplete;
}

// Check if any of the fields are empty
public void setComplete(boolean isComplete) {
	this.isComplete = isComplete;
}

int Year;
int Month;
String FlightDate;
String Origin;
String Desination;
int DepartTime;
int ArrivalTime;
float ArrivalDelayMins;
float Cancelled;
int leg;
float Diverted;
boolean isComplete = true;

// Base Constructor used for Mapper
public Flight1(int year, int month, String flightDate, String origin,
		String desination, int departTime, int arrivalTime,
		float arrivalDelayMins, float cancelled, int leg, float diverted,
		boolean isComplete) {
	super();
	Year = year;
	Month = month;
	FlightDate = flightDate;
	Origin = origin;
	Desination = desination;
	DepartTime = departTime;
	ArrivalTime = arrivalTime;
	ArrivalDelayMins = arrivalDelayMins;
	Cancelled = cancelled;
	this.leg = leg;
	Diverted = diverted;
	this.isComplete = isComplete;
}

// Constructor for Reduce
public Flight1(String[] x) {
	DepartTime = Integer.parseInt(x[0]);
	ArrivalTime = Integer.parseInt(x[1]);
	ArrivalDelayMins = Float.parseFloat(x[2]);
	leg = Integer.parseInt(x[3]);
}
```

```
	// Converts the required fields from mapper to reducer
	public String toString() {
		return DepartTime + "," + ArrivalTime + ","
				+ Float.toString(ArrivalDelayMins) + ","
				+ Integer.toString(leg);
	}
}

// Start Class
public class FlightAvg {

	public enum MyCounters {
		Sum, Total
	}

	// Mappe for First Lef
	public static class FirstLegMapper extends Mapper<Object, Text, Text, Text> {

		// Takes Input of object as Key and Text as value.
		public void map(Object key, Text value, Context context)
				throws IOException, InterruptedException {

			// Parser for parsing CSV File
			CSVParser parser = new CSVParser();

			// Date formatter for comparing and parsing Dates
			SimpleDateFormat dt = new SimpleDateFormat("yyyy-MM-dd");

			// Parse Input Line
			String[] line = parser.parseLine(value.toString());

			// Create String Builder
			StringBuilder b = new StringBuilder();

			// Extract date and append it to the ouput key
			b.append(line[5]);

			// Append , to ouput key
			b.append(",");

			// Create new Object for Flight1 and pass details of Flight from
			// Parsed row
			Flight1 f = new Flight1(line[0].equals("") ? 0
					: Integer.parseInt(line[0]), line[2].equals("") ? 0
					: Integer.parseInt(line[2]), line[5], line[11], line[17],
					line[24].equals("") ? 0 : Integer.parseInt(line[24]),
					line[35].equals("") ? 0 : Integer.parseInt(line[35]),
					line[37].equals("") ? 0 : Float.parseFloat(line[37]),
					line[41].equals("") ? 0 : Float.parseFloat(line[41]), 0,
```

```
                                line[43].equals("") ? 0 : Float.parseFloat(line[43]), true);

                // Check for valid Flights
                if (f.getDiverted() != 1.0 && f.getCancelled() != 1.0) {
                        try {
                                if ((dt.parse("2007-06-01").compareTo(
                                                dt.parse(f.getFlightDate())) <= 0)
                                                && (dt.parse("2008-05-31").compareTo(
                                                                dt.parse(f.getFlightDate())) >= 0)) {
                                        if (f.getOrigin().equals("ORD")
                                                        && !f.getDesination().equals("JFK")) {

                                                // If flight is valid append Destination of current
                                                // flight to the key
                                                b.append(f.getDesination());
                                                f.setLeg(1);

                                                // Emit key [date,destination] and value [details of
                                                // flight]
                                                context.write(new Text(b.toString()),
                                                                new Text(f.toString()));
                                        }
                                }
                        } catch (Exception e) {

                        }
                }
        }

}

// Mapper for second leg
public static class SecondLegMapper extends
                Mapper<Object, Text, Text, Text> {

        public void map(Object key, Text value, Context context)
                        throws IOException, InterruptedException {

                // Parser for parsing flight detials
                CSVParser parser = new CSVParser();

                // Date formatter for parsing and comparing dates
                SimpleDateFormat dt = new SimpleDateFormat("yyyy-MM-dd");

                // Parse input line
                String[] line = parser.parseLine(value.toString());

                // String builder to create new key
                StringBuilder b = new StringBuilder();
```

```
                // Append Date
                b.append(line[5]);

                // Append , to key
                b.append(",");

                // Create new Flight object using parsed data
                Flight1 f = new Flight1(line[0].equals("") ? 0
                                : Integer.parseInt(line[0]), line[2].equals("") ? 0
                                : Integer.parseInt(line[2]), line[5], line[11], line[17],
                                line[24].equals("") ? 0 : Integer.parseInt(line[24]),
                                line[35].equals("") ? 0 : Integer.parseInt(line[35]),
                                line[37].equals("") ? 0 : Float.parseFloat(line[37]),
                                line[41].equals("") ? 0 : Float.parseFloat(line[41]), 0,
                                line[43].equals("") ? 0 : Float.parseFloat(line[43]), true);

                // Check for Flight validity for second leg flights
                if (f.getDiverted() != 1.0 && f.getCancelled() != 1.0) {
                        try {
                                if ((dt.parse("2007-06-01").compareTo(
                                                dt.parse(f.getFlightDate())) <= 0)
                                                && (dt.parse("2008-05-31").compareTo(
                                                                dt.parse(f.getFlightDate())) >= 0)) {
                                        if (!f.getOrigin().equals("ORD")
                                                        && f.getDesination().equals("JFK")) {

                                                // If flight is valid second leg flight
                                                b.append(f.getOrigin());

                                                // Append Origin
                                                f.setLeg(2);

                                                // Emite key [date,origin] and value [flight
                                                // details]
                                                context.write(new Text(b.toString()),
                                                                new Text(f.toString()));
                                        }
                                }
                        } catch (Exception e) {

                        }

                }

        }

}

// Reducer takes key [date,destination/origin] and value as flight details
public static class IntSumReducer extends
```

```
        Reducer<Text, Text, Text, IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<Text> values, Context context)
            throws IOException, InterruptedException {

        // Set sum = 0
        int sum = 0;

        // List to hold all Left Leg Flights
        ArrayList<Flight1> l1 = new ArrayList<Flight1>();

        // List to hold all right Leg Flights
        ArrayList<Flight1> l2 = new ArrayList<Flight1>();

        // Iterate through all the values and seperate left leg and right
        // leg
        for (Text x : values) {
            Flight1 f = new Flight1((x.toString()).split(","));
            if (f.leg == 1) {
                l1.add(f);
            } else {
                if (f.leg == 2) {
                    l2.add(f);
                }
            }

        }

        // Check condition for depart time of second leg flight > arrival
        // time of first leg flight
        for (Flight1 FirstLeg : l1) {
            for (Flight1 SecondLeg : l2) {
                if (SecondLeg.getDepartTime() > FirstLeg.getArrivalTime()) {

                    // Add Delyas
                    float delay = FirstLeg.getArrivalDelayMins()
                            + SecondLeg.getArrivalDelayMins();
                    // Increment total delay counter
                    context.getCounter(MyCounters.Total).increment(
                            (long) delay);

                    // Increment total flight count counter
                    context.getCounter(MyCounters.Sum).increment(1);
                }
            }
        }
        context.write(new Text(key), new IntWritable(sum));
    }
}
```

```
// Main Method
public static void main(String[] args) throws Exception {
	Configuration conf = new Configuration();
	String[] otherArgs = new GenericOptionsParser(conf, args)
				.getRemainingArgs();
	if (otherArgs.length != 3) {
		System.err.println("Usage: wordcount <in> <in> <out>");
		System.exit(2);
	}

	// Set job
	Job job = new Job(conf, "Flight count");

	// Set number of reduce tasks 10
	job.setNumReduceTasks(10);

	// Set jar by class
	job.setJarByClass(FlightAvg.class);

	// Set reducer class
	job.setReducerClass(IntSumReducer.class);

	// Set Map Output Key
	job.setMapOutputKeyClass(Text.class);

	// Set Map output Value
	job.setMapOutputValueClass(Text.class);

	// Set Output Key
	job.setOutputKeyClass(Text.class);

	// Set Outpur Value
	job.setOutputValueClass(IntWritable.class);

	// Set Mapper Class
	MultipleInputs.addInputPath(job, new Path(otherArgs[0]),
				TextInputFormat.class, FirstLegMapper.class);

	// Set Reducer Class
	MultipleInputs.addInputPath(job, new Path(otherArgs[1]),
				TextInputFormat.class, SecondLegMapper.class);

	FileOutputFormat.setOutputPath(job, new Path(otherArgs[2]));

	// Wait for completing job
	boolean jobCompleteFlag = job.waitForCompletion(true);

	// Get counter values
	Counters counters = job.getCounters();
```

```
            // Get sum
            double sum = counters.findCounter(MyCounters.Sum).getValue();
            System.out.println("Counter Sum:" + sum);

            // Get total delay
            double totalDelay = counters.findCounter(MyCounters.Total).getValue();

            // Prink Solution
            System.out.println("Counter TotalDelay:" + totalDelay);
            System.out.println("Avg Delay:" + totalDelay / sum);
            System.exit(job.waitForCompletion(true) ? 0 : 1);
      }

}
```

2) JOINFIRST Version 1 [Source Code]

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

-- Set defaul reduce tasts to 10
SET default_parallel 10;

-- Get all data for Flight 1 [ Left Leg ]
F1 = LOAD '$INPUT' USING CSVLoader(',');
F1 = FOREACH F1 GENERATE        (int)$0 AS (year1),
                                        (int)$2 AS (month1),
                                        (chararray)$5 AS (flightDate1),
                                        (chararray)$11 AS (origin1),
                                        (chararray)$17 AS (destination1),
                                        (int)$24 AS (departureTime1),
                                        (int)$35 AS (arrivalTime1),
                                        (int)$37 AS (arrivalDelayMins1),
                                        (int)$41 AS (cancelled1),
                                        (int)$43 AS (diverted1);

-- Get all data for Flight 2 [ Right Leg ]
F2 = LOAD '$INPUT' USING CSVLoader(',');
F2 = FOREACH F2 GENERATE        (int)$0 AS (year2),
                                        (int)$2 AS (month2),
                                        (chararray)$5 AS (flightDate2),
```

```
                                        (chararray)$11 AS (origin2),
                                        (chararray)$17 AS (destination2),
                                        (int)$24 AS (departureTime2),
                                        (int)$35 AS (arrivalTime2),
                                        (int)$37 AS (arrivalDelayMins2),
                                        (int)$41 AS (cancelled2),
                                        (int)$43 AS (diverted2);

-- Fliter Flights 1 and 2 according to destination, origin, cancelled and diverted values.
F1 = FILTER F1 BY (cancelled1 !=1) AND (diverted1 !=1) AND (origin1 == 'ORD') AND (destination1 != 'JFK');
F2 = FILTER F2 BY (cancelled2 !=1) AND (diverted2 !=1) AND (origin2 != 'ORD') AND (destination2 == 'JFK');

-- Join F1 and F2 Flights based on flightdate and origin of F2 should be same as origin of F1
Same_Date_OriDes = JOIN F1 BY (destination1,flightDate1) , F2 BY (origin2,flightDate2);
Same_Date_OriDes_Filtered = FILTER Same_Date_OriDes BY (departureTime2 > arrivalTime1);

-- Filter all flights where flights are out of Required range of Dates
Same_Date_OriDes_Filtered_Range = FILTER Same_Date_OriDes_Filtered BY  ((year1 == 2007 AND month1 >= 6) OR
(year1 == 2008 AND month1 <= 5)) AND ((year2 == 2007 AND month2 >= 6) OR (year2 == 2008 AND month2 <= 5));


-- Calculate Delay of all valid flights
delay = FOREACH Same_Date_OriDes_Filtered_Range GENERATE (arrivalDelayMins1 + arrivalDelayMins2) as
total_delay;


-- Group Flights so that they can be used for AVG command
final = group delay all;


-- Find AVG of total delay
avg = foreach final generate AVG(delay.total_delay);

-- Store the Avg Value
STORE avg INTO '$OUTPUT';
```

3) JOINFIRST Version 2 [Source Code]

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

-- Set defaul reduce tasts to 10
SET default_parallel 10;

-- Get all data for Flight 1 [ Left Leg ]
F1 = LOAD '$INPUT' USING CSVLoader(',');
```

```
F1 = FOREACH F1 GENERATE        (int)$0 AS (year1),
                                        (int)$2 AS (month1),
                                        (chararray)$5 AS (flightDate1),
                                        (chararray)$11 AS (origin1),
                                        (chararray)$17 AS (destination1),
                                        (int)$24 AS (departureTime1),
                                        (int)$35 AS (arrivalTime1),
                                        (int)$37 AS (arrivalDelayMins1),
                                        (int)$41 AS (cancelled1),
                                        (int)$43 AS (diverted1);

-- Get all data for Flight 2 [ Right Leg ]
F2 = LOAD '$INPUT' USING CSVLoader(',');
F2 = FOREACH F2 GENERATE        (int)$0 AS (year2),
                                        (int)$2 AS (month2),
                                        (chararray)$5 AS (flightDate2),
                                        (chararray)$11 AS (origin2),
                                        (chararray)$17 AS (destination2),
                                        (int)$24 AS (departureTime2),
                                        (int)$35 AS (arrivalTime2),
                                        (int)$37 AS (arrivalDelayMins2),
                                        (int)$41 AS (cancelled2),
                                        (int)$43 AS (diverted2);


-- Fliter Flights 1 and 2 according to destination, origin, cancelled and diverted values.
F1 = FILTER F1 BY (cancelled1 !=1) AND (diverted1 !=1) AND (origin1 == 'ORD') AND (destination1 != 'JFK');
F2 = FILTER F2 BY (cancelled2 !=1) AND (diverted2 !=1) AND (origin2 != 'ORD') AND (destination2 == 'JFK');



-- Join F1 and F2 Flights based on flightdate and origin of F2 should be same as origin of F1
Same_Date_OriDes = JOIN F1 BY (destination1,flightDate1) , F2 BY (origin2,flightDate2);
Same_Date_OriDes_Filtered = FILTER Same_Date_OriDes BY (departureTime2 > arrivalTime1);

-- Filter all flights where flights are out of Required range of Dates
Same_Date_OriDes_Filtered_Range = FILTER Same_Date_OriDes_Filtered BY  ((year1 == 2007 AND month1 >= 6) OR
(year1 == 2008 AND month1 <= 5));

-- Calculate Delay of all valid flights
delay = FOREACH Same_Date_OriDes_Filtered_Range GENERATE (arrivalDelayMins1 + arrivalDelayMins2) as
total_delay;

-- Group Flights so that they can be used for AVG command
final = group delay all;

-- Find AVG of total delay
avg = foreach final generate AVG(delay.total_delay);

-- Store the Avg Value
```

```
STORE avg INTO '$OUTPUT';
```

4) FILTERFIRST [Source Code]

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


-- Set defaul reduce tasts to 10
SET default_parallel 10;

-- Get all data for Flight 1 [ Left Leg ]
F1 = LOAD '$INPUT' USING CSVLoader(',');
F1 = FOREACH F1 GENERATE        (int)$0 AS (year1),
                                           (int)$2 AS (month1),
                                           (chararray)$5 AS (flightDate1),
                                           (chararray)$11 AS (origin1),
                                           (chararray)$17 AS (destination1),
                                           (int)$24 AS (departureTime1),
                                           (int)$35 AS (arrivalTime1),
                                           (int)$37 AS (arrivalDelayMins1),
                                           (int)$41 AS (cancelled1),
                                           (int)$43 AS (diverted1);

-- Get all data for Flight 2 [ Right Leg ]
F2 = LOAD '$INPUT' USING CSVLoader(',');
F2 = FOREACH F2 GENERATE        (int)$0 AS (year2),
                                           (int)$2 AS (month2),
                                           (chararray)$5 AS (flightDate2),
                                           (chararray)$11 AS (origin2),
                                           (chararray)$17 AS (destination2),
                                           (int)$24 AS (departureTime2),
                                           (int)$35 AS (arrivalTime2),
                                           (int)$37 AS (arrivalDelayMins2),
                                           (int)$41 AS (cancelled2),
                                           (int)$43 AS (diverted2);

-- Fliter Flights 1 and 2 according to destination, origin, cancelled and diverted values including checks for valid
flightdates.
F1 = FILTER F1 BY (cancelled1 !=1) AND (diverted1 !=1) AND (origin1 == 'ORD') AND (destination1 != 'JFK') AND
      ((year1 == 2007 AND month1 >= 6) OR (year1 == 2008 AND month1 <= 5));
F2 = FILTER F2 BY (cancelled2 !=1) AND (diverted2 !=1) AND (origin2 != 'ORD') AND (destination2 == 'JFK') AND
((year2 == 2007 AND month2 >= 6) OR (year2 == 2008 AND month2 <= 5));


-- Join F1 and F2 Flights based on flightdate and origin of F2 should be same as origin of F1
```

```
Same_Date_OriDes = JOIN F1 BY (destination1,flightDate1) , F2 BY (origin2,flightDate2);
Same_Date_OriDes_Filtered = FILTER Same_Date_OriDes BY (departureTime2 > arrivalTime1);


-- Calculate Delay of all valid flights
delay = FOREACH Same_Date_OriDes_Filtered GENERATE (arrivalDelayMins1 + arrivalDelayMins2) as total_delay;

-- Find AVG of total delay
final = group delay all;

-- Find AVG of total delay
avg = foreach final generate AVG(delay.total_delay);

-- Store the Avg Value
STORE avg INTO '$OUTPUT';
```

Performance Difference for Pig Latin Program for taking Input once only:

I did not find any difference in taking input once or twice. Both performed equally.

Performance Comparisons

Program	Total Run Time (in seconds)
Plain Line	**399**
Join First Version 1	472
Join First Version 2	**462**
Filter First	**468**

Evaluation:

From the above evaluation we can see that Plain Line command beats all 3 Pig Latin Programs.

Did your PLAIN program beat Pig?

Yes the PLAIN Program beats all the Pig Latin programs by an average of 1 min.

How did the differences in the Pig programs affect runtime?

The major difference that I see in all the Pig Latin programs is that the number of filters and Joins are different based on the strategies used.

Can you explain why these runtime results happened?

In the Join First Version 1 program we are first joining all the tuples and then Filters all the unwanted data.
The Version 2 there is a decrease in run time because only one check is made on the date that reduces the number of operations thereby reduces the run time by some seconds.
In the Filter First Program we first filter all the irrelevant data and then perform joins on it therefore we do spend time processing not required data.

Average Flight Delay :

Program	AVG Delay
Plain	50.67124150519758
Join First Version 1	50.67124150519758
Join First Version 2	50.67124150519758
Filter First	50.67124150519758