Mukul Sharma **CS 6240 - 02**
MapReduce : HomeWork 3

PLAIN PSEUDO-CODE

```
Map(Object key, Text value){

        // parse the incoming data using CSV parser

        // Store each required field from the parsed line

        //Applying Filters

        isValidFlight (){

          1. Filter all flights whose origin is "ORD" then their destination is not "JFK" and if destination is
          JFK then origin  is not ORD

          2. The Flight lies between the given dates.

          3. The flight is not cancelled or diverted

        }

        //Create key according to Origin/Destination of the flight and Date

        If(Origin = = 'ORD')

          Key = destination-floghtdate

          Else

          Key = origin-flightdate

        //club other fields together in value by string concatenation

          Emit(key, value);

}

Reduce(String key, array  iterable values){

        //create two array lists, one storing flights form origin == ORD and second one for all other
        flights.

        Iterate over the two lists to find flight pairs with arrival time of flight 1 < departure of flight 2

        Store delays in global counters and the count of flights}

Main(){

// Calculate and report average}
```

PLAIN SOURCE-CODE

```
import com.opencsv.CSVParser;

public class Plain {

	public enum DelayCounter {

		Total_Delay,  //Global Counter to Store delay sum

		Entities	//Counter to keep track of total flight pairs

	}

	public static class flightDelayPlainMapper extends Mapper<Object, Text, Text, Text>

	{	@Override

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

		{	if(((LongWritable) key).get()>0){

			//get data from CSV file line by line

			CSVParser parser = new CSVParser();

			String[] lineEntries = parser.parseLine(value.toString());

			//get all values for processing

			String flightDate = lineEntries[5];

			String origin = lineEntries[11];

			String destination = lineEntries[17];

			String arrivalTime = lineEntries[35];

			String departureTime = lineEntries[24];

			String ArrDelayMinutes = lineEntries[37];

			String cancelled = lineEntries[41];

			String diverted = lineEntries[43];

			Text nkey;

	//Check if the flight is a valid flight

	if(isValidFlight(origin, destination, flightDate, cancelled, diverted)){

	//Store all required attributes of valid flight
```

```
        String val =  origin + "," + destination + "," + flightDate + "," + arrivalTime + "," +
                        departureTime + "," + ArrDelayMinutes;
        //Assign keys according to flightDate and origin/destination
        if(origin.equals("ORD"))
                nkey = new Text(destination + "-" + flightDate);
        else
                nkey = new Text(origin + "-" + flightDate);
                Text nVal = new Text(val);
                context.write(nkey, nVal);
                }
        }
}


        /*Applying Filters
         * 1. Filter all flights whose origin is "ORD" then their desti nation is not "JFK" and if
        destination is JFK then origin is not ORD
         * 2. The Flight lies between the given dates.
         * 3. The flight is not cancelled or diverted  * */
        boolean isValidFlight(String origin,String destination,
                        String flightDate,String cancelled,String diverted){
                if(hasValidOriginAndDestination(origin, destination)
                                && hasValidDate(flightDate)
                                && notCanceledOrDiverted(cancelled, diverted)){
                                return true;
                        }
                        else
                                return false;
                }
```

```
/*Filter all flights whose origin is "ORD" then their desti
 * nation is not "JFK" and if destination is JFK then origin
 * is not ORD
 * */
boolean hasValidOriginAndDestination(String origin, String destination){
	if(origin.equals("ORD") && !(destination.equals("JFK"))
				|| !(origin.equals("ORD")) && (destination.equals("JFK"))){
		return true;
	}
	else
		return false;
}

/* 2. The Flight lies between the given dates.*/
boolean hasValidDate(String flightDate){
	SimpleDateFormat ft = new SimpleDateFormat ("yyyy-MM-dd");
	Date min = null,max = null,d = null;
	try {
		min = ft.parse("2007-05-31");
		max = ft.parse("2008-06-01");
		d = ft.parse(flightDate);
	} catch (ParseException e) {
		e.printStackTrace();
	}
	return (d.after(min) && d.before(max));
}
```

```
		/* 3. The flight is not cancelled or diverted */
		boolean notCanceledOrDiverted(String cancelled, String diverted){
			if( cancelled.equals("0.00") && diverted.equals("0.00"))
				return true;
			else
				return false;
		}
	}

	/*Reducer Class*/
public static class flightDelayReducer extends Reducer<Text,Text,Text,FloatWritable> {
	float delays = 0;
	@Override
	public void reduce(Text key, Iterable<Text> values, Context context)
			throws IOException, InterruptedException {
		/* For Storing different list elements */
		String[] elementsF1, elementsF2;
		String itemF1=null, itemF2=null;

		/*List for storing flights originating for ORD*/
		ArrayList<String> f1Data = new ArrayList<String>();
		/*List for all other flights*/
		ArrayList<String> f2Data = new ArrayList<String>();
		/*Separate the potential two legs in different lists*/
		for(Text vItem : values){
			String vItemToString = vItem.toString();
```

```
                String[] itemBreak = vItemToString.split(",");
                if(itemBreak[0].equals("ORD"))
                        f1Data.add(vItemToString);
                else
                        f2Data.add(vItemToString);
        }

        /* Iterate over both of the above created lists to determine
         * correct pair of flights where arrival time of leg 1 is before
         *  the departure time of leg 2  */
        for(Iterator<String> i = f1Data.iterator(); i.hasNext(); ) {
                itemF1 = i.next();
                elementsF1 = itemF1.split(",");
                for(Iterator<String> j = f2Data.iterator(); j.hasNext(); ) {
                        itemF2 = j.next();
                        elementsF2 = itemF2.split(",");
                        if(legalFlightTime(elementsF1[3],elementsF2[4])){
                                delays = Float.parseFloat(elementsF1[5])+
                                        Float.parseFloat(elementsF2[5]);

context.getCounter(DelayCounter.Total_Delay).increment((long) delays);

context.getCounter(DelayCounter.Entities).increment(1);
                        }
                }
        }
}
```

```
		/*To check if arrival time of leg 1 is before the departure time of leg 2 */
		boolean legalFlightTime(String arrF1,String depF2){
			return (Integer.parseInt(depF2) > Integer.parseInt(arrF1));
		}
	}

	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: Plain <input> <output>");
	    System.exit(2);
	   }
		Job job = Job.getInstance(conf, "Average Flight Delay");
		job.setJarByClass(Plain.class);
		job.setMapperClass(flightDelayPlainMapper.class);
		job.setReducerClass(flightDelayReducer.class);
		job.setNumReduceTasks(10);
		job.setMapOutputKeyClass(Text.class);
		job.setMapOutputValueClass(Text.class);
		job.setOutputKeyClass(Text.class);
		job.setOutputValueClass(Text.class);
		FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
		FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));

		if(job.waitForCompletion(true)){
			Counters counters = job.getCounters();
```

```
                        float total = counters.findCounter(DelayCounter.Total_Delay).getValue();
                        float numbers = counters.findCounter(DelayCounter.Entities).getValue();
                        float average = total/numbers;
                        System.out.println("The total average is = "+average);
                        System.exit(0);
                }
                else
                        System.exit(1);
        }
}
```

PIG PROGRAMS

FILTER-FIRST : SOURCE-CODE

```
--PLAIN--

REGISTER file:/home/hadoop/lib/pig/piggybank.jar;

DEFINE CSVLoader org.apache.pig.piggybank.storage.CSVLoader;


-- Setting 10 Reducers for the job --

SET default_parallel 10;


-- Loading flight data for leg one and leg 2 using CSV loader --

FlightData1 = load 's3://mukulbucket/input/data.csv/' USING CSVLoader;

FlightData2 = load 's3://mukulbucket/input/data.csv/' USING CSVLoader;


-- Extracting the required fields from the flight data for both legs --

Flights1 = FOREACH FlightData1 GENERATE (chararray)$11 AS origin, (chararray)$17 AS dest,
                                        (int)$35 AS arrTime, (int)$24 AS depTime,
                                        (float)$37 AS arrDelayMinutes, (chararray)$5 AS flightDate,
                                        (int)$41 AS canceled, (int)$43 AS diverted;


Flights2 = FOREACH FlightData2 GENERATE (chararray)$11 AS origin, (chararray)$17 AS dest,
                                        (int)$35 AS arrTime, (int)$24 AS depTime,
                                        (float)$37 AS arrDelayMinutes, (chararray)$5 AS flightDate,
                                        (int)$41 AS canceled, (int)$43 AS diverted;

--------- ALL FILTERS ----------

-- Filtering flights 1 and 2 for correct destination/origin, as given the  flight starting from ORD cannot end
-- at JFK and a flight ending at JFK  cannot start from ORD --

Flights1 = FILTER Flights1 BY (origin == 'ORD' AND dest != 'JFK');

Flights2 = FILTER Flights2 BY (origin != 'ORD' AND dest == 'JFK');
```

```
-- Filtering flights on the basis of cancelled or delayed
Flights1 = FILTER Flights1 BY (canceled != 1 AND diverted != 1);
Flights2 = FILTER Flights2 BY (canceled != 1 AND diverted != 1);

-- Filter for date range, extracting the flights that lie in the given date range --
Flights1 = FILTER Flights1 BY ToDate($5, 'yyyy-MM-dd') < ToDate('2008-06-01','yyyy-MM-dd')
                              AND ToDate($5, 'yyyy-MM-dd') > ToDate('2007-05-31','yyyy-MM-dd');

Flights2 = FILTER Flights2 BY ToDate($5, 'yyyy-MM-dd') < ToDate('2008-06-01','yyyy-MM-dd')
                              AND ToDate($5, 'yyyy-MM-dd') > ToDate('2007-05-31','yyyy-MM-dd');

---------- JOIN ----------
-- JOIN according to date for flights1 and flights 2 and destination of flight1 matches the origin of flight 2
afterJoinOndateAndDest = JOIN Flights1 BY (dest, flightDate), Flights2 BY (origin, flightDate);

---------- LAST FILTER ---------
-- Filter for arrival time of flight1 < departure of flight 2, that is leg 1 arrives before leg 2 leaves
legalTime = FILTER afterJoinOndateAndDest BY $2 < $11;

-- Calculating Average --
sum = FOREACH legalTime GENERATE (float)($4+$12) AS delay;
groupedSum = GROUP sum ALL;
average = FOREACH groupedSum GENERATE AVG(sum.delay) AS average;

-- Store the result into firstfilter folder --
STORE average INTO 's3://mukulbucket/hw3pig/firstfilter/' USING PigStorage();
```

JOIN-FIRST-VERSION -1 : SOURCE-CODE

```
-- JOIN FIRST Version 1 --

REGISTER file:/home/hadoop/lib/pig/piggybank.jar;

DEFINE CSVLoader org.apache.pig.piggybank.storage.CSVLoader;


-- Setting 10 Reducers for the job --

SET default_parallel 10;


-- Loading flight data for leg one and leg 2 using CSV loader --

FlightData1 = load 's3://mukulbucket/input/' USING CSVLoader;

FlightData2 = load 's3://mukulbucket/input/' USING CSVLoader;


-- Extracting the required fields from the flight data for both legs --

Flights1 = FOREACH FlightData1 GENERATE (chararray)$11 AS origin, (chararray)$17 AS dest,
                                        (int)$35 AS arrTime, (int)$24 AS depTime,
                                        (float)$37 AS arrDelayMinutes, (chararray)$5 AS flightDate,
                                        (int)$41 AS canceled, (int)$43 AS diverted;


Flights2 = FOREACH FlightData2 GENERATE (chararray)$11 AS origin, (chararray)$17 AS dest,
                                        (int)$35 AS arrTime, (int)$24 AS depTime,
                                        (float)$37 AS arrDelayMinutes, (chararray)$5 AS flightDate,
                                        (int)$41 AS canceled, (int)$43 AS diverted;


--------- ALL FILTERS -----------

-- Filtering flights 1 and 2 for correct destination/origin, as given the  flight starting from ORD ---
-- cannot end at JFK and a flight ending at JFK --
```

```
-- cannot start from ORD --

Flights1 = FILTER Flights1 BY (origin == 'ORD' AND dest != 'JFK');

Flights2 = FILTER Flights2 BY (origin != 'ORD' AND dest == 'JFK');


-- Filtering flights on the basis of cancelled or delayed

Flights1 = FILTER Flights1 BY (canceled != 1 AND diverted != 1);

Flights2 = FILTER Flights2 BY (canceled != 1 AND diverted != 1);


---------- JOIN ----------

-- JOIN according to date for flights1 and flights 2 and destination of flight1 matches the origin --
-- of flight 2--

afterJoinOndateAndDest = JOIN Flights1 BY (dest, flightDate), Flights2 BY (origin, flightDate);


-- Filter for arrival time of flight1 < departure of flight 2, that is leg 1 arrives before leg 2 leaves

legalTime = FILTER afterJoinOndateAndDest BY $2 < $11;


--------Version 1 Specific requirement ---------

-- Filter for date range, extracting the flights that lie in the given date range for both flights 1
and 2. --

legalDateRange = FILTER legalTime BY

        ToDate($5, 'yyyy-MM-dd') < ToDate('2008-06-01','yyyy-MM-dd')

        AND ToDate($5, 'yyyy-MM-dd') > ToDate('2007-05-31','yyyy-MM-dd')

        AND ToDate($13, 'yyyy-MM-dd') < ToDate('2008-06-01','yyyy-MM-dd')

        AND ToDate($13, 'yyyy-MM-dd') > ToDate('2007-05-31','yyyy-MM-dd');


-- Calculate Average --

sum = FOREACH legalDateRange GENERATE (float)($4+$12) AS delay;
```

```
groupedSum = GROUP sum ALL;

average = FOREACH groupedSum GENERATE AVG(sum.delay);


-- Store the result into joinfv1 folder --

STORE average INTO 's3://mukulbucket/hw3pig/joinfv1/' USING PigStorage();
```

JOIN-FIRST VERSION-2 :SOURCE CODE

```
-- JOIN FIRST Version 2 --

REGISTER file:/home/hadoop/lib/pig/piggybank.jar;

DEFINE CSVLoader org.apache.pig.piggybank.storage.CSVLoader;

-- Setting 10 Reducers for the job --

SET default_parallel 10;

-- Loading flight data for leg one and leg 2 using CSV loader --

FlightData1 = load 's3://mukulbucket/input/' USING CSVLoader;

FlightData2 = load 's3://mukulbucket/input/' USING CSVLoader;

-- Extracting the required fields from the flight data for both legs --

Flights1 = FOREACH FlightData1 GENERATE (chararray)$11 AS origin, (chararray)$17 AS dest,
                                        (int)$35 AS arrTime, (int)$24 AS depTime,
                                        (float)$37 AS arrDelayMinutes, (chararray)$5 AS flightDate,
                                        (int)$41 AS canceled, (int)$43 AS diverted;

Flights2 = FOREACH FlightData2 GENERATE (chararray)$11 AS origin, (chararray)$17 AS dest,
                                        (int)$35 AS arrTime, (int)$24 AS depTime,
                                        (float)$37 AS arrDelayMinutes, (chararray)$5 AS flightDate,
                                        (int)$41 AS canceled, (int)$43 AS diverted;
```

--------- ALL FILTERS ----------

```
-- Filtering flights 1 and 2 for correct destination/origin, as given the  flight starting from ORD  --
-- cannot end at JFK and a flight ending at JFK  cannot start from ORD --

Flights1 = FILTER Flights1 BY (origin == 'ORD' AND dest != 'JFK');

Flights2 = FILTER Flights2 BY (origin != 'ORD' AND dest == 'JFK');


-- Filtering flights on the basis of cancelled or delayed

Flights1 = FILTER Flights1 BY (canceled != 1 AND diverted != 1);

Flights2 = FILTER Flights2 BY (canceled != 1 AND diverted != 1);
```

---------- JOIN ----------

```
-- JOIN according to date for flights1 and flights 2 and destination of flight1 matches the origin --
-- of flight 2

afterJoinOndateAndDest = JOIN Flights1 BY (dest, flightDate), Flights2 BY (origin, flightDate);


-- Filter for arrival time of flight1 < departure of flight 2, that is leg 1 arrives before leg 2 leaves

legalTime = FILTER afterJoinOndateAndDest BY $2 < $11;
```

--------Version 2 Specific requirement ---------

```
-- Filter for date range, extracting the flights that lie in the given date range for flights 1. --

legalDateRange = FILTER legalTime BY
                        ToDate($5, 'yyyy-MM-dd') < ToDate('2008-06-01','yyyy-MM-dd')
                        AND ToDate($5, 'yyyy-MM-dd') > ToDate('2007-05-31','yyyy-MM-dd');


-- Calculate average

sum = FOREACH legalDateRange GENERATE (float)($4+$12) AS delay;

groupedSum = GROUP sum ALL;
```

```
average = FOREACH groupedSum GENERATE AVG(sum.delay) AS average;


--store the result into joinfv2 folder

STORE average INTO 's3://mukulbucket/hw3pig/joinfv2/' USING PigStorage();
```

Mukul Sharma **CS 6240 - 02**
MapReduce : HomeWork 3

PERFORMANCE COMPARISON

Running time

	Plain	FilterFirst	JoinFirst v1	JoinFirst v2
Running time	271	524	512	496

Q. Did your PLAIN program beat Pig?

ANS: Yes, my PLAIN program did beat PIG, more over there is a big difference in running time of the plain program and all the other 3 PIG programs.

I believe that the reason for fast execution of plain program is that it is directly in JAVA where as in case of the PIG programs; they are first converted to their JAVA equivalent and then run. This creates a big difference in the running time.

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

ANS: There is not a big difference in the run times of the pig programs however when we closely look at the three programs, we find that join-first version 2 is the fastest of all the 3 because it had to do less number of processing for filtering out flights according to the dates as it did this work for only leg-1 flights.

Average flight delay times :

	Plain	FilterFirst	JoinFirst v1	JoinFirst v2
Average Delay	50.67124	50.67124150519758	50.67124150519758	50.67124150519758