Fri – 1.35 to 3.15pm HW3 Sahil Khandwala

Source Code:-

Plain MapReduce -

```
package CS6240;

import java.io.IOException;
import java.util.ArrayList;

import org.apache.commons.el.OrOperator;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Partitioner;
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.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

import au.com.bytecode.opencsv.CSVParser;

public class Flight {

        // create global counters
        public static enum Global_Counters {
                count,
                averageFlightDelay
        };

        public static class FlightDataMapper extends Mapper<Object, Text, Text, Text>
        {
                public void map(Object key, Text value, Context context) throws IOException,
InterruptedException
                {
                        // using CSV parser to pass the input file
                        CSVParser parser = new CSVParser();

                   String origin = null;
                   String destination = null;
                   String diverted = null;
                   String cancelled = null;
                   String year = null;
                   String month = null;
```

```
            String flightdate = null;
            String DepartureTime = null;
            String ArrivalTime = null;
            String ArrivalDelay = null;


                String[] line = parser.parseLine(value.toString());
                origin = line[11];
                destination = line[17];
                diverted = line[43];
                cancelled = line[41];
                year = line[0];
                month = line[2];
                flightdate = line[5];
                DepartureTime = line[24];
                ArrivalTime = line[35];
                ArrivalDelay = line[37];

                // checking if origin is ORD and destination != JFK and checking if the
                // flight is not diverted or cancelled and falls within flightdate of
                // >= June 2007 and <= May 2008
                if(origin.equalsIgnoreCase("ORD") && !
destination.equalsIgnoreCase("JFK"))
                {
                    if(cancelled.equals("0.00") && diverted.equals("0.00"))
                    {
                        int yr = Integer.parseInt(year);
                        int mon = Integer.parseInt(month);

                        if((yr == 2007 && mon >= 6) || (yr == 2008 && mon <= 5))
                        {
                            // key is set a destination, flightdate
                            Text key1 = new Text(destination + "," + flightdate);
                            Text val = new Text(origin + "," + ArrivalTime + "," +
DepartureTime + "," + ArrivalDelay);
                            context.write(key1, val);
                        }
                    }
                }

                // checking if origin!= ORD and destination = JFK and checking if the
                // flight is not diverted or cancelled and falls within flightdate of
                // >= June 2007 and <= May 2008
                else if(!origin.equalsIgnoreCase("ORD") &&
destination.equalsIgnoreCase("JFK"))
                {
                        if(cancelled.equals("0.00") && diverted.equals("0.00"))
                        {
                            int yr = Integer.parseInt(year);
```

```
                                        int mon = Integer.parseInt(month);

                                        if((yr == 2007 && mon >= 6) || (yr == 2008 && mon
<= 5))
                                        {
                                                //key is set as origin, flightdate
                                                Text key1 = new Text(origin + "," + flightdate);
                                                Text val = new Text(origin + "," + ArrivalTime
+ "," + DepartureTime + "," + ArrivalDelay);
                                                context.write(key1, val);
                                        }
                                }
                        }
                }
        }

        public static class FlightDataReducer extends Reducer<Text, Text, Text, Text>
        {
                public void reduce(Text key, Iterable<Text> values, Context context) throws
IOException
                {
                        // 2 lists are made to store the data
                        ArrayList<String[]> list1 = new ArrayList<String[]>();
                        ArrayList<String[]> list2 = new ArrayList<String[]>();

                        for(Text v : values)
                        {
                                String[] record = new CSVParser().parseLine(v.toString());
                                if(record[0].equalsIgnoreCase("ORD"))
                                {       // if origin is ORD we are adding to one list
                                        list1.add(record);
                                }
                                else
                                {
                                        // if not we are adding to another list
                                        list2.add(record);
                                }
                        }

                        //String[] flight1;
                        //String[] flight2;

                        for(String[] flight1: list1)
                        {
                                for(String[] flight2: list2)
                                {
                                        // checking if arrival time of 1 flight which matches all the
conditions is < the departure time
                                        // of another flight which also matches the conditions
```

```
					if(Double.parseDouble(flight1[1]) <
Double.parseDouble(flight2[2]))
					{
						// updating global counters
						// incrementing totalflightcount by 1 and total
delay by the arrivaldelay of flight1 + arrivaldelay of flight2

	context.getCounter(Global_Counters.averageFlightDelay).increment((long)
(Float.parseFloat(flight1[3])+ Float.parseFloat(flight2[3])));

	context.getCounter(Global_Counters.count).increment(1);
					}
				}
			}
		}
	}

	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("Error");
			System.exit(2);
		}

		Job job = new Job(conf, "Flight Data Join");
		job.setJarByClass(Flight.class);
		job.setMapperClass(FlightDataMapper.class);
		job.setReducerClass(FlightDataReducer.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)) {
		long avgDelay =
(long)job.getCounters().findCounter(Global_Counters.averageFlightDelay).getValue();
		long totalFlightCount =
(long)job.getCounters().findCounter(Global_Counters.count).getValue();

		//getting total avergae flight delay by dividing avgdelay by totalflightcount

		double totalAverageFlightDelay = (float)avgDelay/(float)totalFlightCount;
		System.out.println("Total Delay " + avgDelay);
		System.out.println("TotalFlightCount "+ totalFlightCount);
		System.out.println("Average Flight Delay for 2-Legged Flight is " +
totalAverageFlightDelay);
            }
         }
}
```

JoinFirst(Version1)-

```
# Using CSV Loader instead of PigStorage which is located at this location

REGISTER file:/home/hadoop/lib/pig/piggybank.jar;
define CSVLoader org.apache.pig.piggybank.storage.CSVLoader();

# Setting the number of reducers to 10
SET default_parallel 10;

# Loading the data twice
Flight1 = LOAD 's3://homework34/data_input/data.csv' using CSVLoader() parallel 10;
Flight2 = LOAD 's3://homework34/data_input/data.csv' USING CSVLoader() parallel 10;

#Filtering the data by the different conditions
FilteredFlight1 = Filter Flight1 by $11 == 'ORD' and $17!= 'JFK' and (float)$41 == 0.00 and
(float)$43 == 0.00 parallel 10;

#Filtering the data by the different conditions
FilteredFlight2 = Filter Flight2 by $11!= 'ORD' and $17 == 'JFK' and (float)$41 == 0.00 and
(float)$43 == 0.00 parallel 10;

#Only selecting the relevant data which would be useful to us from relation Flight1
Relation1 = FOREACH FilteredFlight1 GENERATE (int)$0 as Year1, (int)$2 as Month1, $5 as
FlightDate1, $11 as Origin1, $17 as Dest1, (int)$35 as ArrivalTime, (float)$37 as ArrDelay1
parallel 10;

#Only selecting the relevant data which would be useful to us from relation Flight2
Relation2 = FOREACH FilteredFlight2 GENERATE (int)$0 as Year2, (int)$2 as Month2, $5
as FlightDate2, $11 as Origin2, $17 as Dest2, (int)$24 as DepartureTime, (float)$37 as
ArrDelay2 parallel 10;

#Joining the 2 Flights data if the destination of 1 and the origin of the 2nd flight is the same
along with having the same flight date
JoinFlights = Join Relation1 by (Dest1, FlightDate1), Relation2 by (Origin2, FlightDate2)
parallel 10;

#Filtering the Join relation to get only flights which have a arrival time of 1 flight less than
departure time of the next flight
FilterTime = Filter JoinFlights by ArrivalTime < DepartureTime;

#Filtering the data further condition being only flights between June 2007 and May 2008 for
both Flight1 and Flight2 data
FilterDate = Filter FilterTime by
```

```
((((Year1==2007) and (Month1>=6)) or ((Year1==2008) and (Month1<=5))) and
(((Year2==2007) and (Month2>=6)) or ((Year2==2008) and (Month2<=5)))) parallel 10;


#Generating the average delay for by adding arrivaldelay of all relevant flight1 + flight2 data
DelayCalculation = FOREACH FilterDate GENERATE (float)ArrDelay1 + ArrDelay2 as
TotalDelay parallel 10;

AllCounts = group DelayCalculation All parallel 10;

# Generating the average delay
AvgDelay = FOREACH AllCounts GENERATE AVG(DelayCalculation.TotalDelay) parallel 10;

#Storing the result data onto a file
STORE AvgDelay INTO 's3://homework34/output/PigLatin/Prog1/Result' ;
```

JoinFirst(Version2) – Same as version except we are filtering Flight date between June 2007 to May 2008 only for flight 1 data

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

SET default_parallel 10;

Flight1 = LOAD 's3://homework34/data_input/data.csv' using CSVLoader() parallel 10;
Flight2 = LOAD 's3://homework34/data_input/data.csv' USING CSVLoader() parallel 10;

FilteredFlight1 = Filter Flight1 by $11 == 'ORD' and $17!= 'JFK' and (float)$41 == 0.00 and
(float)$43 == 0.00 parallel 10;

FilteredFlight2 = Filter Flight2 by $11!= 'ORD' and $17 == 'JFK' and (float)$41 == 0.00 and
(float)$43 == 0.00 parallel 10;

Relation1 = FOREACH FilteredFlight1 GENERATE (int)$0 as Year1, (int)$2 as Month1, $5 as
FlightDate1, $11 as Origin1, $17 as Dest1, (int)$35 as ArrivalTime, (float)$37 as ArrDelay1
parallel 10;

Relation2 = FOREACH FilteredFlight2 GENERATE (int)$0 as Year2, (int)$2 as Month2, $5
as FlightDate2, $11 as Origin2, $17 as Dest2, (int)$24 as DepartureTime, (float)$37 as
ArrDelay2 parallel 10;

JoinFlights = Join Relation1 by (Dest1, FlightDate1), Relation2 by (Origin2, FlightDate2);

FilterTime = Filter JoinFlights by ArrivalTime < DepartureTime parallel 10;

# Checking only if data of Flight1 is within range between June 2007 to May 2008
FilterDate = Filter FilterTime by
(((Year1==2007) and (Month1>=6)) or ((Year1==2008) and (Month1<=5))) parallel 10;
```

```
DelayCalculation = FOREACH FilterDate GENERATE (float)ArrDelay1 + ArrDelay2 as
TotalDelay parallel 10;

AllCounts = group DelayCalculation All parallel 10;

AvgDelay = FOREACH AllCounts GENERATE AVG(DelayCalculation.TotalDelay) parallel 10;

STORE AvgDelay INTO 's3://homework34/output/PigLatin/Prog2/Result';
```

FilterFirst – (Same as JoinFirst except that we are filtering the data between June 2007 and May 2008 and then joining both the relation sets)

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

SET default_parallel 10;

Flight1 = LOAD 's3://homework34/data_input/data.csv' using CSVLoader() parallel 10;
Flight2 = LOAD 's3://homework34/data_input/data.csv' USING CSVLoader() parallel 10;

# Filtering the input data also by the time range of flights being between June 2007 and May
2008
FilteredFlight1 = FILTER Flight1 BY $11 == 'ORD' and $17 != 'JFK' and (float)$41 == 0.00
and (float)$43 == 0.00 and ((((int)$0==2007) and ((int)$2>=6)) or (((int)$0==2008) and
((int)$2<=5))) parallel 10;

# Filtering the input data also by the time range of flights being between June 2007 and May
2008
FilteredFlight2 = FILTER Flight2 BY $11 != 'ORD' and $17 == 'JFK' and (float)$41 == 0.00
and (float)$43 == 0.00 and ((((int)$0==2007) and ((int)$2>=6)) or (((int)$0==2008) and
((int)$2<=5))) parallel 10;

ExtractFlight1 = FOREACH FilteredFlight1 GENERATE (int)$0 as year, (int)$2 as mon, $5 as
fldate1, $17 as dest1, (int)$35 as arr, (float)$37 as arrdel1, $11 as origin1 parallel 10;

ExtractFlight2 = FOREACH FilteredFlight2 GENERATE (int)$0 as year, (int)$2 as mon, $5
as fldate2, $11 as origin2, (int)$24 as dep, (float)$37 as arrdel2, $17 as dest2 parallel 10;

JoinFlight = Join ExtractFlight1 BY (dest1, fldate1), ExtractFlight2 BY (origin2, fldate2)
parallel 10;

FilterByTime = FILTER JoinFlight BY arr < dep parallel 10;

aggregation= FOREACH FilterByTime GENERATE (float) arrdel1 + arrdel2 as totdel parallel
10;

aggregation1 = GROUP aggregation ALL parallel 10;
```

```
out = FOREACH aggregation1 GENERATE AVG(aggregation.totdel) parallel 10;

STORE out INTO 's3://homework34/output/PigLatin/Prog3/Result';
```

Pseudo Code for Plain MapReduce -

```
Mapper :-
1. Create global counters to store totalFlightCount and totalDelay
2. checking for origin = ORD and destination != JFK
checking if flight is not diverted and canceled and falls within the required flight data of June 2007 to May 2008. Key is set as (destination, flightdate)
3. checking the above same details for origin!=ORD and destination = JFK. Key is set as (origin, flightdate)

Reducer:-

1. if origin = "ORD" they are put into 1 list if not we are inderting into another list
2. then we are iterating through 2 lists and checking if arrival time of list1 is < departure time of list2
3. incrementing global counter totalFlightCount by 1 and totalDelay by (arrivaldelay of flight1 + arrivaldelay of flight2)

we are getting the average flight delay by dividing totalDelay by totalFlightCount
```

Total Running Time (Taken from Controller log file):-

Plain MapReduce – 263 seconds

JoinFirst(Version1) - 567 seconds

JoinFirst(Version2) – 558 seconds

FilterFirst – 545 seconds

Critical Analysis :-

Did your PLAIN program beat Pig?
Yes, Plain program ran much faster than Pig script. Mostly because for each join or group by operation there is a different map-reduce job thus there may eventually be multiple Map Reduce jobs created, this leads to an in-efficient approach.

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

There was not observed a great difference in times among the 3 Pig programs. This happened because even though in the first 2 program we employed the Join First technique and in the third program we did the filterfirst technique. Internally the Pig optimizer has optimized the program in its best possible way and executed it. Like for example it pushes Filter operator upstream to filter out the records as early as possible instead of unnecessarily joining ahead.

Average Flight Delay :-

Plain MapReduce – 50.67 mins

JoinFirst(Version1) - 50.67124150519758 mins

JoinFirst(Version2) - 50.67124150519758 mins

FilterFirst – 50.67124150519758 mins