S6240 – PARALLEL DATA PROCESSING IN MAP-REDUCE

Class: CS6240-02
Homework Number: 3
Name: Arpit Mehta

Pseudo-Code for PLAIN Java Program:

```
// Mapper
FlightdataMapper:
    // Input: <byte offset, line>
    // Output: <Text<layoverCityCode+date>, Text<depTime/arrTime+delay>>
    map(Object key, Text value, Context context):
        Parse the input line to create FlightData object
      1. Check if flight information is valid for first leg:
            a. is cancelled
            b. is diverted
            c. is year & month of flight in range (between 6-2007 to 5-2008)
            d. origin = ORD OR destination = JFK but not both.
      2. If conditions are true:
            a. Create a Text key object = (layoverCityCode + flightDate)
            b. Create a Text value object:
                  i.   ("F" + ":" + arrTime + ":" + arrDelay) → First Flight
                  ii.  ("S" + ":" + depTime + ":" + arrDelay) → Second Flight

// Partitioner
FlightDataPartitioner:
    int getPartition(Text key, Text value, int numPartitions):
        // Partitioning is done on the basis of the month of the flight
        get month from the key by splitting the string with regex "-"
        return (month-1)

// Reducer
FlightDataReducer:
    // define lstFirstFlights, that will hold the list of all first leg flights.
    ArrayList<Text> lstFirstFlights = new ArrayList<Text>();

    // define lstSecondFlights, that will hold the list of all second leg flights.
    ArrayList<Text> lstSecondFlights = new ArrayList<Text>();

    for all values:
        if First flight
            lstFirstFlights.add(value)
```

```
                else if second flight:
                        lstSecondFlights.add(value)
        // Reducer-side joining
        for all lstFirstFlights:
                for all lstSecondFlights:
                        if(firstFlight.arrTime < secondFlight.depTime)
                                counter[DELAY_SUM] += (firstFlight.arrDelay + secondFlight.arrDelay)
                                counter[FREQUENCY] += 1

// Calculation of average delay between ORD to JFK
Once the job completes, in the main/driver code the average delay is calculated by:
if job is successful:
        total_delay = get values in counter[DELAY_SUM];
        frequency = get values in counter[FREQUENCY];
        average_delay = (float) (total_delay) / (float)(frequency)

output average_delay
```

Source Codes:

PLAIN:

FlightConstants.java: Defines constants used in average flight delay calculation

```java
public class FlightConstants {
	/**
	 * Number of reduce tasks
	 */
	public static final int NUM_REDUCE_TASKS = 12;

	/**
	 * From year
	 */
	 public static final int FROM_YEAR = 2007;

	/**
	 * To year
	 */
	public static final int TO_YEAR = 2008;

	/**
	 * From month
	 */
 public static final int FROM_MONTH = 6;

/**
 * To month
 */
public static final int TO_MONTH = 5;

/**
 * Origin airport code
 */
public static final String ORIGIN_CODE = "ORD";

/**
 * Destination airport code
 */
 public static final String DESTINATION_CODE = "JFK";

/**
 * The delimiter used in Mappers
 */
public static final String DELIMITER = ":";

/**
 * Fixed fields in CSV flight data
 */
public static final int INDEX_FLIGHT_YEAR = 0;
public static final int INDEX_FLIGHT_MONTH = 2;
public static final int INDEX_FLIGHT_DATE = 5;
public static final int INDEX_ORIGIN = 11;
public static final int INDEX_DESTINATION = 17;
public static final int INDEX_DEP_TIME = 24;
public static final int INDEX_ARR_TIME = 35;
public static final int INDEX_ARR_DELAY_MINUTES = 37;
public static final int INDEX_CANCELLED = 41;
public static final int INDEX_DIVERTED = 43;

/**
 * Enum for global counters to calculate average flight delay
 */
public enum AverageFlightDelayCounters {
      DELAY_SUM,
      FREQUENCY
};
```

```
}
```

FlightData.java: Generic class that holds the parsed flight data from the csv

```
public class FlightData {
      /**
       * The flight date
       */
      private String flightDate;

      /**
       * The flight month
       */
      private int flightMonth;

      /**
       * The flight year
       */
      private int flightYear;

      /**
       * The origin code
       */
      private String origin;

      /**
       * The destination code
       */
      private String destination;

      /**
       * Is flight cancelled
       */
      private boolean isCancelled;

      /**
       * s flight delayed
       */
      private boolean isDiverted;

      /**
       * Departure Time
       */
```

```
private String depTime;

/**
 * Arrival Time
 */
private String arrTime;

/**
 * Arrival delay in minutes
 */
private String arrDelay;

/**
 * Default constructor
 */
public FlightData() {
      setFlightDate(new String());
      setFlightMonth(0);
      setFlightYear(0);
      setCancelled(false);
      setDiverted(false);
      setOrigin(new String());
      setDestination(new String());
      setDepTime(new String());
      setArrTime(new String());
      setArrDelay(new String());
}

@Override
public String toString() {
      String str = null;

      str = "Flight Data: [" + "flightYear = " + flightYear
                  + ", flightMonth = " + flightMonth + ", flightDate: "
                  + flightDate + ", origin = " + origin + ", destination = "
                  + destination + ", departureTime = " + depTime
                  + ", arrivalTime = " + arrTime + ", arrDelayMinutes = "
                  + arrDelay + ", isCancelled = "
                  + ((isCancelled == false) ? "no" : "yes") + ", isDiverted = "
                  + ((isDiverted == false) ? "no" : "yes") + "]";

      return str;
}
```

```
/*
 * Getters & Setters
 */
public String getFlightDate() {
      return flightDate;
}

public void setFlightDate(String flightDate) {
      this.flightDate = flightDate;
}

public int getFlightMonth() {
      return flightMonth;
}

public void setFlightMonth(int flightMonth) {
      this.flightMonth = flightMonth;
}

public int getFlightYear() {
      return flightYear;
}

public void setFlightYear(int flightYear) {
      this.flightYear = flightYear;
}

public String getOrigin() {
      return origin;
}

public void setOrigin(String origin) {
      this.origin = origin;
}

public String getDestination() {
      return destination;
}

public void setDestination(String destination) {
      this.destination = destination;
}
```

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

	public void setCancelled(boolean isCancelled) {
		this.isCancelled = isCancelled;
	}

	public boolean isDiverted() {
		return isDiverted;
	}

	public void setDiverted(boolean isDiverted) {
		this.isDiverted = isDiverted;
	}

	public String getDepTime() {
		return depTime;
	}

	public void setDepTime(String depTime) {
		this.depTime = depTime;
	}

	public String getArrTime() {
		return arrTime;
	}

	public void setArrTime(String arrTime) {
		this.arrTime = arrTime;
	}

	public String getArrDelay() {
		return arrDelay;
	}

	public void setArrDelay(String arrDelay) {
		this.arrDelay = arrDelay;
	}

}
```

FlightDataParser.java: This class provides utility functions to parse csv data and create FlightData objects.

```
public class FlightDataParser {
      /**
       * The instance of FlightDataParser class
       */
      private static FlightDataParser instance = null;

      /**
       * The CSV Reader object
       */
      private static CSVReader csvReader;

      /**
       * The string reader object
       */
      private static StringReader strReader;

      /**
       * Constructor
       */
      private FlightDataParser() {
            // A private Constructor prevents any other class from instantiating.
      }

      /**
       * getInstance
       *
       * @return object FlightDataParser
       */
      public static FlightDataParser getInstance() {
            if (instance == null) {
                  instance = new FlightDataParser();
            }

            return instance;
      }

      /**
       * getFlightData: Function parses the CSV flight data and returns a
       * FlightData object.
       *
       * @param String
```

```
 *            line
 *
 * @return object FlightData
 * @throws IOException
 */
public FlightData getFlightData(String line) throws IOException {
      FlightData fData = new FlightData();

      strReader = new StringReader(line);
      csvReader = new CSVReader(strReader);

      String[] values = csvReader.readNext();

      fData.setFlightYear(Integer
                  .parseInt(values[FlightConstants.INDEX_FLIGHT_YEAR].trim()));
      fData.setFlightMonth(Integer
                  .parseInt(values[FlightConstants.INDEX_FLIGHT_MONTH].trim()));
      fData.setFlightDate(values[FlightConstants.INDEX_FLIGHT_DATE].trim());
      fData.setOrigin(values[FlightConstants.INDEX_ORIGIN].trim());
      fData.setDestination(values[FlightConstants.INDEX_DESTINATION].trim());
      fData.setDepTime(values[FlightConstants.INDEX_DEP_TIME].trim());
      fData.setArrTime(values[FlightConstants.INDEX_ARR_TIME].trim());
      fData.setArrDelay(values[FlightConstants.INDEX_ARR_DELAY_MINUTES]
                  .trim());
      fData.setCancelled((values[FlightConstants.INDEX_CANCELLED].trim()
                  .equals("0.00")) ? false : true);
      fData.setDiverted((values[FlightConstants.INDEX_DIVERTED].trim()
                  .equals("0.00")) ? false : true);

      return fData;
}

/*
 * Get & Set methods
 */
public CSVReader getCsvReader() {
      return csvReader;
}

public void setCsvReader(CSVReader csvReader) {
      FlightDataParser.csvReader = csvReader;
}
```

```
	public StringReader getStrReader() {
		return strReader;
	}

	public void setStrReader(StringReader strReader) {
		FlightDataParser.strReader = strReader;
	}
}
```

FlightUtils.java: Provides other utility functions for checking if the flight is valid, i.e, if the origin is ORD or destination is JFK.

```
public class FlightUtils {

	/**
	 * Function checks if the flight is of interest.
	 *
	 * @param fData
	 *            FlightData object
	 *
	 * @return boolean isValid
	 */
	public static boolean isValidFlight(FlightData fData) {
		boolean isValid = false;

		isValid = (isMonthAndYearValid(fData)
				&& isOriginOrDestinationValid(fData) && !(fData.isCancelled() &&
fData
				.isDiverted()));

		return isValid;
	}

	/**
	 * Function checks if the flight matches the criteria for the first flight.
	 *
	 * @param fData
	 * @return isValid
	 */
	public static boolean isFirstFlight(FlightData fData) {
		boolean isValid = false;
		String origin = fData.getOrigin().trim();
		String destination = fData.getDestination().trim();
```

```
            isValid = (origin.equalsIgnoreCase(FlightConstants.ORIGIN_CODE) &&
!destination
                        .equalsIgnoreCase(FlightConstants.DESTINATION_CODE));

            return isValid;
      }

      /**
       * Function checks if the flight matches the criteria for the second flight.
       *
       * @param fData
       * @return isValid
       */
      public static boolean isSecondFlight(FlightData fData) {
            boolean isValid = false;
            String origin = fData.getOrigin().trim();
            String destination = fData.getDestination().trim();

            isValid = (!origin.equalsIgnoreCase(FlightConstants.ORIGIN_CODE) &&
destination
                        .equalsIgnoreCase(FlightConstants.DESTINATION_CODE));

            return isValid;
      }

      /**
       * isOriginOrDestinationValid: Function checks if the flight's origin or
       * destination is valid
       *
       * @param fData
       *
       * @return
       */
      private static boolean isOriginOrDestinationValid(FlightData fData) {
            boolean isValid = false;

            isValid = (isFirstFlight(fData) || isSecondFlight(fData));

            return isValid;
      }

      /**
```

```
         * Function checks if the month & the year of the flight is within range.
         *
         * @param fData
         *
         * @return boolean isValid
         */
        private static boolean isMonthAndYearValid(FlightData fData) {
              boolean isValid = false;
              int year = fData.getFlightYear();
              int month = fData.getFlightMonth();

              // Checks
              if (year == FlightConstants.FROM_YEAR) {
                    if ((month > FlightConstants.FROM_MONTH) && (month <= 12)) {
                          isValid = true;
                    }
              } else if (year == FlightConstants.TO_YEAR) {
                    if ((month < FlightConstants.TO_MONTH) && (month >= 1)) {
                          isValid = true;
                    }
              } else if ((year > FlightConstants.FROM_YEAR)
                          && (year < FlightConstants.TO_YEAR)) {
                    isValid = true;
              } else {
                    isValid = false;
              }

              return isValid;
        }
}
```

AverageFlightDelay.java: Calculates the average flight delay between ORD → JFK

```
public class AverageFlightDelay {

      public static class FlightDataMapper extends
                  Mapper<Object, Text, Text, Text> {
            // FlightDataParser object
            private FlightDataParser dataParser;

            @Override
            protected void setup(Context context) throws IOException,
                        InterruptedException {
```

```
		super.setup(context);

		setParser(FlightDataParser.getInstance());
	}

	public void map(Object key, Text value, Context context)
			throws IOException, InterruptedException {
		// Get Flight Data
		FlightData fData = dataParser.getFlightData(value.toString());

		// Verify if flight is valid
		if (FlightUtils.isValidFlight(fData)) {
			// Define the key
			// FlightDataMapperKey outKey = null;
			Text outKey = null;

			// Define Value
			Text outValue = null;

			if (FlightUtils.isFirstFlight(fData)) {
				outKey = createKey(fData.getDestination(),
						fData.getFlightDate());

				outValue = createValue("F1", fData.getArrTime(),
						fData.getArrDelay());
			} else if (FlightUtils.isSecondFlight(fData)) {
				outKey = createKey(fData.getOrigin(),
fData.getFlightDate());

				outValue = createValue("F2", fData.getDepTime(),
						fData.getArrDelay());
			}

			// TODO For testing
			// System.out.println(outKey + " ---> " + outValue);

			// Emit
			if ((outKey != null) && (outValue != null)) {
				context.write(outKey, outValue);
			}
		}
	}
```

```
/**
 * Function returns the out value for the mapper
 *
 * @param f1orf2
 * @param arrOrDepTime
 * @param arrDelay
 *
 * @return Text
 */
private Text createValue(String f1orf2, String arrOrDepTime,
            String arrDelay) {
      Text returnValue = null;

      if (!isNullString(arrOrDepTime) && !isNullString(arrDelay)) {
            returnValue = new Text(f1orf2 + FlightConstants.DELIMITER
                        + arrOrDepTime.trim() + FlightConstants.DELIMITER
                        + arrDelay.trim());
      }

      return returnValue;
}

/**
 * Function return an output value for the mapper
 *
 * @param dest
 * @param date
 * @return
 */
private Text createKey(String dest, String date) {
      Text returnKey = null;
      if (!isNullString(dest) && !isNullString(date)) {
            returnKey = new Text(dest.trim() + date.trim());
      }
      return returnKey;
}

/**
 * Function checks if a string is null
 *
 * @param str
 * @return boolean
 */
	private boolean isNullString(String str) {
		if ((str == null) || (str.trim().length() == 0)) {
			return true;
		} else {
			return false;
		}
	}

	@Override
	protected void cleanup(Context context) throws IOException,
			InterruptedException, NullPointerException {
		// Close the parser's string reader
		dataParser.getStrReader().close();

		// Close the parser's CSV reader
		dataParser.getCsvReader().close();

		super.cleanup(context);
	}

	/*
	 * Get & set methods
	 */
	public FlightDataParser getParser() {
		return dataParser;
	}

	public void setParser(FlightDataParser parser) {
		this.dataParser = parser;
	}
}

/**
 * FlightDataPartitioner Class
 * <p>
 * Partitions the FlightDataMapperKey based on their HashCode. The
 * getPartition function returns partition number between 0 and
 * numPartitions.
 * </p>
 *
 * @author arpitm
 *
 */
	public static class FlightDataPartitioner extends Partitioner<Text, Text> {

		@Override
		public int getPartition(Text key, Text value, int numPartitions) {

			String[] keyParts = key.toString().split("-");
			int month = Integer.parseInt(keyParts[1]);
			return (month - 1);

//			return ((key.toString().hashCode() * 127) % numPartitions);
		}
	}

	/**
	 * FlightDataReducer Class
	 * <p>
	 * This reducer performs the join operation on the FlightDataMapperKey
	 * object and outputs the delay for the joint flights.
	 * </p>
	 *
	 * @author arpitm
	 *
	 */
	public static class FlightDataReducer extends
			Reducer<Text, Text, Text, Text> {
		// List to store all flights in first leg
		private ArrayList<Text> lstFirstFlights = null;

		// List to store all flights in the second leg
		private ArrayList<Text> lstSecondFlights = null;

		@Override
		protected void setup(Context context) throws IOException,
				InterruptedException {
			super.setup(context);
			lstFirstFlights = new ArrayList<Text>();
			lstSecondFlights = new ArrayList<Text>();
		}

		public void reduce(Text key, Iterable<Text> values, Context context)
				throws IOException, InterruptedException,
				IndexOutOfBoundsException {
			// Clear lists
```

```
            lstFirstFlights.clear();
            lstSecondFlights.clear();

            boolean isFirstFlight = false;
            boolean isSecondFlight = false;

            // Join
            for (Text value : values) {
                  String[] valueParts = value.toString().split(
                              FlightConstants.DELIMITER);

                  // TODO For testing
                  // System.out.println(value.toString());

                  isFirstFlight = (valueParts[0].equalsIgnoreCase("F1") ? true
                              : false);
                  isSecondFlight = (valueParts[0].equalsIgnoreCase("F2") ? true
                              : false);

                  if (isFirstFlight) {
                        String arrTime = valueParts[1];
                        String arrDelay = valueParts[2];
                        lstFirstFlights.add(new Text(arrTime
                                    + FlightConstants.DELIMITER + arrDelay));
                  } else if (isSecondFlight) {
                        String depTime = valueParts[1];
                        String arrDelay = valueParts[2];
                        lstSecondFlights.add(new Text(depTime
                                    + FlightConstants.DELIMITER + arrDelay));
                  }
            }

            // Execute our join logic now that the lists are filled
            executeJoinLogic(context);
      }

      @Override
      protected void cleanup(Context context) throws IOException,
                  InterruptedException {
            super.cleanup(context);

            lstFirstFlights = null;
            lstSecondFlights = null;
```

```java
}

/**
 * Executes the reducer-side join logic based on the condition
 * list1.flight.arrTime < list2.flight.depTime
 *
 * @param context
 * @throws IOException
 * @throws InterruptedException
 * @throws NumberFormatException
 */
private void executeJoinLogic(Context context) throws IOException,
            InterruptedException, NumberFormatException {
      int firstFlightArrtime = 0;
      int secondFlightDepTime = 0;
      float firstFlightArrDelay = (float) 0.0;
      float secondFlightArrDelay = (float) 0.0;
      long totalDelay = (long) 0;

      if (!lstFirstFlights.isEmpty() && !lstSecondFlights.isEmpty()) {
            for (Text firstFlight : lstFirstFlights) {
                  String[] firstFlightParts = firstFlight.toString().split(
                              FlightConstants.DELIMITER);

                  firstFlightArrtime = Integer.parseInt(firstFlightParts[0]);
                  firstFlightArrDelay = Float.parseFloat(firstFlightParts[1]);

                  for (Text secondFlight : lstSecondFlights) {

                        String[] secondFlightParts = secondFlight.toString()
                                    .split(FlightConstants.DELIMITER);

                        secondFlightDepTime = Integer
                                    .parseInt(secondFlightParts[0]);
                        secondFlightArrDelay = Float
                                    .parseFloat(secondFlightParts[1]);

                        if ((secondFlightDepTime - firstFlightArrtime) > 0) {

                              totalDelay = (long) firstFlightArrDelay
                                          + (long) secondFlightArrDelay;

                              // Increment Counters
```

```
                                        context.getCounter(

FlightConstants.AverageFlightDelayCounters.DELAY_SUM)
                                                    .increment(totalDelay);
                                        context.getCounter(

FlightConstants.AverageFlightDelayCounters.FREQUENCY)
                                                    .increment(1);
                                   }
                             }
                       }
                 }
           }
     }

     /**
      * main: Driver function
      *
      * @param args
      *
      * @throws IOException
      * @throws ClassNotFoundException
      * @throws InterruptedException
      */
     public static void main(String[] args) throws IOException,
                 InterruptedException, ClassNotFoundException {
           Configuration conf = new Configuration();

           String[] otherArgs = new GenericOptionsParser(conf, args)
                       .getRemainingArgs();

           // Arguments length check
           if (otherArgs.length != 2) {
                 System.err
                             .println("Usage: AverageFlightDelay <data-file-path>
<output-path>");
                 System.exit(1);
           }

           // Job: Average Flight Delay Calculation.
           Job job = new Job(conf, "Average Flight Delay Calculation.");
           // job.getConfiguration().set("join.type", joinType);
           job.setJarByClass(AverageFlightDelay.class);
```

```
            job.setMapperClass(FlightDataMapper.class);
            job.setPartitionerClass(FlightDataPartitioner.class);
            job.setReducerClass(FlightDataReducer.class);
            job.setMapOutputKeyClass(Text.class);
            job.setMapOutputValueClass(Text.class);
            job.setOutputKeyClass(Text.class);
            job.setOutputValueClass(Text.class);
            job.setNumReduceTasks(FlightConstants.NUM_REDUCE_TASKS);
            FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
            FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
            job.waitForCompletion(true);

            // Calculate average flight delay once the job is successful
            if (job.isSuccessful()) {
                  long delaySum = job
                              .getCounters()
                              .findCounter(

FlightConstants.AverageFlightDelayCounters.DELAY_SUM)
                              .getValue();
                  long freq = job
                              .getCounters()
                              .findCounter(

FlightConstants.AverageFlightDelayCounters.FREQUENCY)
                              .getValue();

                  float avgDelay = ((float) delaySum / (float) freq);

                  System.out.println("Average flight delay from ORD -> JFK is: "
                              + avgDelay);
                  System.exit(0);
            } else {
                  System.out.println("Job " + job.getJobName() + " Failed!");
                  System.exit(1);
            }
      }
}
```

JOIN-FIRST (VERSION 1):

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

-- CSVLoader for reading data from input file
DEFINE CSVLoader org.apache.pig.piggybank.storage.CSVLoader;

-- Set default parallel as 20
SET default_parallel 20;

-- Load the flights file as Flights1.
Flights1 = LOAD '$INPUT' USING CSVLoader() AS
(Year:int,Quarter,Month:int,DayofMonth,DayOfWeek,FlightDate,UniqueCarrier,AirlineID,Carrie
r,TailNum,FlightNum,Origin,OriginCityName,OriginState,OriginStateFips,OriginStateName,Orig
inWac,Dest,DestCityName,DestState,DestStateFips,DestStateName,DestWac,CRSDepTime,DepTime:i
nt,DepDelay,DepDelayMinutes,DepDel15,DepartureDelayGroups,DepTimeBlk,TaxiOut,WheelsOff,Whe
elsOn,TaxiIn,CRSArrTime,ArrTime:int,ArrDelay,ArrDelayMinutes:int,ArrDel15,ArrivalDelayGrou
ps,ArrTimeBlk,Cancelled,CancellationCode,Diverted,CRSElapsedTime,ActualElapsedTime,AirTime
,Flights,Distance,DistanceGroup,CarrierDelay,WeatherDelay,NASDelay,SecurityDelay,LateAircr
aftDelay);

-- Load     the   flights     file as     Flights2.
Flights2 = LOAD '$INPUT' USING CSVLoader() AS
(Year:int,Quarter,Month:int,DayofMonth,DayOfWeek,FlightDate,UniqueCarrier,AirlineID,Carrie
r,TailNum,FlightNum,Origin,OriginCityName,OriginState,OriginStateFips,OriginStateName,Orig
inWac,Dest,DestCityName,DestState,DestStateFips,DestStateName,DestWac,CRSDepTime,DepTime:i
nt,DepDelay,DepDelayMinutes,DepDel15,DepartureDelayGroups,DepTimeBlk,TaxiOut,WheelsOff,Whe
elsOn,TaxiIn,CRSArrTime,ArrTime:int,ArrDelay,ArrDelayMinutes:int,ArrDel15,ArrivalDelayGrou
ps,ArrTimeBlk,Cancelled,CancellationCode,Diverted,CRSElapsedTime,ActualElapsedTime,AirTime
,Flights,Distance,DistanceGroup,CarrierDelay,WeatherDelay,NASDelay,SecurityDelay,LateAircr
aftDelay);

-- Remove unwanted records from Flights1
Flight1Data = FOREACH Flights1 GENERATE Year, Month, FlightDate, Origin, Dest, DepTime,
ArrTime, ArrDelayMinutes, Cancelled, Diverted;

-- Remove unwanted records from Flights2
Flight2Data = FOREACH Flights2 GENERATE Year, Month, FlightDate, Origin, Dest, DepTime,
ArrTime, ArrDelayMinutes, Cancelled, Diverted;

-- Join     Flights1 and Flights2, using the condition thatthe   destination airport
in Flights1 matches     the   origin in  Flights2 and that both have   the   same flight
date.
```

```
firstFlights = FILTER Flight1Data BY (Origin == 'ORD' AND Dest != 'JFK' AND Cancelled ==
'0.00' AND Diverted == '0.00');
secondFlights = FILTER Flight2Data BY (Origin != 'ORD' AND Dest == 'JFK' AND Cancelled ==
'0.00' AND Diverted == '0.00');
jointFlights = JOIN firstFlights BY (Dest, FlightDate), secondFlights BY (Origin,
FlightDate);

-- Filter out those    join tuples where the departure    time in    Flights2 is not
after the arrival time in Flights1.
filteredJointFlights = FILTER jointFlights BY (firstFlights::ArrTime <
secondFlights::DepTime);

-- Filter out those tuples whose flight date is not between June 2007 & May 2008: Check
that both flight date of Flight 1 & Flig2 are in range
inRangeJointFlights = FILTER filteredJointFlights BY (((firstFlights::Year == 2007 AND
firstFlights::Month >= 6) OR (firstFlights::Year == 2008 AND firstFlights::Month <= 5))
AND ((secondFlights::Year == 2007 AND secondFlights::Month >= 6) OR (secondFlights::Year
== 2008 AND secondFlights::Month <= 5)));

-- Average calculation
arrDelays = FOREACH inRangeJointFlights GENERATE (firstFlights::ArrDelayMinutes +
secondFlights::ArrDelayMinutes) AS delay;
groupedArrDelays = GROUP arrDelays ALL;
avgDelay = FOREACH groupedArrDelays GENERATE AVG(arrDelays);

-- Store Delay
STORE avgDelay INTO '$OUTPUT';

--DUMP avgDelay;
```

JOIN-FIRST (VERSION 2):

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

-- CSVLoader for reading data from input file
DEFINE CSVLoader org.apache.pig.piggybank.storage.CSVLoader;

-- Set default parallel as 20
SET default_parallel 20;

-- Load    the   flights    file as    Flights1.
```

```
Flights1 = LOAD '$INPUT' USING CSVLoader() AS
(Year:int,Quarter,Month:int,DayofMonth,DayOfWeek,FlightDate,UniqueCarrier,AirlineID,Carrie
r,TailNum,FlightNum,Origin,OriginCityName,OriginState,OriginStateFips,OriginStateName,Orig
inWac,Dest,DestCityName,DestState,DestStateFips,DestStateName,DestWac,CRSDepTime,DepTime:i
nt,DepDelay,DepDelayMinutes,DepDel15,DepartureDelayGroups,DepTimeBlk,TaxiOut,WheelsOff,Whe
elsOn,TaxiIn,CRSArrTime,ArrTime:int,ArrDelay,ArrDelayMinutes:int,ArrDel15,ArrivalDelayGrou
ps,ArrTimeBlk,Cancelled,CancellationCode,Diverted,CRSElapsedTime,ActualElapsedTime,AirTime
,Flights,Distance,DistanceGroup,CarrierDelay,WeatherDelay,NASDelay,SecurityDelay,LateAircr
aftDelay);

-- Load     the   flights     file as     Flights2.
Flights2 = LOAD '$INPUT' USING CSVLoader() AS
(Year:int,Quarter,Month:int,DayofMonth,DayOfWeek,FlightDate,UniqueCarrier,AirlineID,Carrie
r,TailNum,FlightNum,Origin,OriginCityName,OriginState,OriginStateFips,OriginStateName,Orig
inWac,Dest,DestCityName,DestState,DestStateFips,DestStateName,DestWac,CRSDepTime,DepTime:i
nt,DepDelay,DepDelayMinutes,DepDel15,DepartureDelayGroups,DepTimeBlk,TaxiOut,WheelsOff,Whe
elsOn,TaxiIn,CRSArrTime,ArrTime:int,ArrDelay,ArrDelayMinutes:int,ArrDel15,ArrivalDelayGrou
ps,ArrTimeBlk,Cancelled,CancellationCode,Diverted,CRSElapsedTime,ActualElapsedTime,AirTime
,Flights,Distance,DistanceGroup,CarrierDelay,WeatherDelay,NASDelay,SecurityDelay,LateAircr
aftDelay);

-- Remove unwanted records from Flights1
Flight1Data = FOREACH Flights1 GENERATE Year, Month, FlightDate, Origin, Dest, DepTime,
ArrTime, ArrDelayMinutes, Cancelled, Diverted;

-- Remove unwanted records from Flights2
Flight2Data = FOREACH Flights2 GENERATE Year, Month, FlightDate, Origin, Dest, DepTime,
ArrTime, ArrDelayMinutes, Cancelled, Diverted;

-- Join     Flights1 and Flights2, using the condition thatthe   destination airport
in Flights1 matches     the   origin in  Flights2 and that both have   the   same flight
date.
firstFlights = FILTER Flight1Data BY (Origin == 'ORD' AND Dest != 'JFK' AND Cancelled ==
'0.00' AND Diverted == '0.00');
secondFlights = FILTER Flight2Data BY (Origin != 'ORD' AND Dest == 'JFK' AND Cancelled ==
'0.00' AND Diverted == '0.00');
jointFlights = JOIN firstFlights BY (Dest, FlightDate), secondFlights BY (Origin,
FlightDate);

-- Filter out those     join tuples where the departure     time in     Flights2 is not
after the arrival time in Flights1.
filteredJointFlights = FILTER jointFlights BY (firstFlights::ArrTime <
secondFlights::DepTime);
```

```
-- Filter out those tuples whose flight date is not between June 2007 & May 2008: Check
that both flight date of Flight 1 & Flig2 are in range
inRangeJointFlights = FILTER filteredJointFlights BY ((firstFlights::Year == 2007 AND
firstFlights::Month >= 6) OR (firstFlights::Year == 2008 AND firstFlights::Month <= 5));

-- Average calculation
arrDelays = FOREACH inRangeJointFlights GENERATE (firstFlights::ArrDelayMinutes +
secondFlights::ArrDelayMinutes) AS delay;
groupedArrDelays = GROUP arrDelays ALL;
avgDelay = FOREACH groupedArrDelays GENERATE AVG(arrDelays);

-- Store Delay
STORE avgDelay INTO '$OUTPUT';

--DUMP avgDelay;
```

FILTER-FIRST (VERSION 1):

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

-- CSVLoader for reading data from input file
DEFINE CSVLoader org.apache.pig.piggybank.storage.CSVLoader;

-- Set default parallel as 20
SET default_parallel 20;

-- Load      the flights file as Flights1.
```

```
Flights1 = LOAD '$INPUT' USING CSVLoader() AS
(Year:int,Quarter,Month:int,DayofMonth,DayOfWeek,FlightDate,UniqueCarrier,
AirlineID,Carrier,TailNum,FlightNum,Origin,OriginCityName,OriginState,OriginStateFips,
OriginStateName,OriginWac,Dest,DestCityName,DestState,DestStateFips,DestStateName,DestWac,
CRSDepTime,DepTime:int,DepDelay,DepDelayMinutes,DepDel15,DepartureDelayGroups,DepTimeBlk,
TaxiOut,WheelsOff,WheelsOn,TaxiIn,CRSArrTime,ArrTime:int,ArrDelay,ArrDelayMinutes:int,
ArrDel15,ArrivalDelayGroups,ArrTimeBlk,Cancelled,CancellationCode,Diverted,CRSElapsedTime,
ActualElapsedTime,AirTime,Flights,Distance,DistanceGroup,CarrierDelay,WeatherDelay,NASDela
y,
SecurityDelay,LateAircraftDelay);

-- Load      the flights file as Flights2.
Flights2 = LOAD '$INPUT' USING CSVLoader() AS
(Year:int,Quarter,Month:int,DayofMonth,DayOfWeek,FlightDate,UniqueCarrier,
AirlineID,Carrier,TailNum,FlightNum,Origin,OriginCityName,OriginState,OriginStateFips,
OriginStateName,OriginWac,Dest,DestCityName,DestState,DestStateFips,DestStateName,DestWac,
CRSDepTime,DepTime:int,DepDelay,DepDelayMinutes,DepDel15,DepartureDelayGroups,DepTimeBlk,
TaxiOut,WheelsOff,WheelsOn,TaxiIn,CRSArrTime,ArrTime:int,ArrDelay,ArrDelayMinutes:int,
ArrDel15,ArrivalDelayGroups,ArrTimeBlk,Cancelled,CancellationCode,Diverted,CRSElapsedTime,
ActualElapsedTime,AirTime,Flights,Distance,DistanceGroup,CarrierDelay,WeatherDelay,NASDela
y,
SecurityDelay,LateAircraftDelay);

-- Remove unwanted records from Flights1
Flight1Data = FOREACH Flights1 GENERATE Year, Month, FlightDate, Origin, Dest, DepTime,
ArrTime, ArrDelayMinutes, Cancelled, Diverted;

-- Remove unwanted records from Flights2
Flight2Data = FOREACH Flights2 GENERATE Year, Month, FlightDate, Origin, Dest, DepTime,
ArrTime, ArrDelayMinutes, Cancelled, Diverted;

-- Remove as many records and attributes from Flights1 & Flights2 as possible, with the
condition flight date is valid
firstFlights = FILTER Flight1Data BY ((Origin == 'ORD' AND Dest != 'JFK') AND
                                        Cancelled == '0.00' AND Diverted == '0.00' AND
                                        ((Year == 2007 and Month >= 6) OR
                                         (Year == 2008 and Month <= 5)));
secondFlights = FILTER Flight2Data BY ((Origin != 'ORD' AND Dest == 'JFK') AND
                                        Cancelled == '0.00' AND Diverted == '0.00' AND
                                         ((Year == 2007 and Month >= 6) OR
                                          (Year == 2008 and Month <= 5)));
jointFlights = JOIN firstFlights BY (Dest, FlightDate), secondFlights BY (Origin,
FlightDate);
```

```
-- Filter out those    join tuples where the departure    time in    Flights2 is not
after the arrival time in Flights1.
filteredJointFlights = FILTER jointFlights BY (firstFlights::ArrTime <
secondFlights::DepTime);

-- Average calculation
arrDelays = FOREACH filteredJointFlights GENERATE (firstFlights::ArrDelayMinutes +
secondFlights::ArrDelayMinutes) AS delay;
groupedArrDelays = GROUP arrDelays ALL;
avgDelay = FOREACH groupedArrDelays GENERATE AVG(arrDelays);

-- Store Delay
STORE avgDelay INTO '$OUTPUT';

--DUMP avgDelay;
```

<u>FILTER-FIRST (VERSION 2):</u>

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

-- CSVLoader for reading data from input file
DEFINE CSVLoader org.apache.pig.piggybank.storage.CSVLoader;

-- Set default parallel as 20
SET default_parallel 20;

-- Load    the   flights    file as    Flights1.
Flights1 = LOAD '$INPUT' USING CSVLoader() AS
(Year:int,Quarter,Month:int,DayofMonth,DayOfWeek,FlightDate,UniqueCarrier,
AirlineID,Carrier,TailNum,FlightNum,Origin,OriginCityName,OriginState,OriginStateFips,
OriginStateName,OriginWac,Dest,DestCityName,DestState,DestStateFips,DestStateName,DestWac,
CRSDepTime,DepTime:int,DepDelay,DepDelayMinutes,DepDel15,DepartureDelayGroups,DepTimeBlk,
TaxiOut,WheelsOff,WheelsOn,TaxiIn,CRSArrTime,ArrTime:int,ArrDelay,ArrDelayMinutes:int,
ArrDel15,ArrivalDelayGroups,ArrTimeBlk,Cancelled,CancellationCode,Diverted,CRSElapsedTime,
ActualElapsedTime,AirTime,Flights,Distance,DistanceGroup,CarrierDelay,WeatherDelay,NASDela
y,
SecurityDelay,LateAircraftDelay);

-- Load the flights file as Flights2.
Flights2 = FOREACH Flights1 GENERATE *;

-- Remove unwanted records from Flights1
Flight1Data = FOREACH Flights1 GENERATE Year, Month, FlightDate, Origin, Dest, DepTime,
ArrTime, ArrDelayMinutes, Cancelled, Diverted;

-- Remove unwanted records from Flights2
Flight2Data = FOREACH Flights2 GENERATE Year, Month, FlightDate, Origin, Dest, DepTime,
ArrTime, ArrDelayMinutes, Cancelled, Diverted;

-- Remove as many records and attributes from Flights1 & Flights2 as possible, with the
condition flight date is valid
firstFlights = FILTER Flight1Data BY ((Origin == 'ORD' AND Dest != 'JFK') AND
                                        Cancelled == '0.00' AND Diverted == '0.00' AND
                                        ((Year == 2007 and Month >= 6) OR
                                         (Year == 2008 and Month <= 5)));
secondFlights = FILTER Flight2Data BY ((Origin != 'ORD' AND Dest == 'JFK') AND
                                        Cancelled == '0.00' AND Diverted == '0.00' AND
                                         ((Year == 2007 and Month >= 6) OR
                                           (Year == 2008 and Month <= 5)));
jointFlights = JOIN firstFlights BY (Dest, FlightDate), secondFlights BY (Origin,
FlightDate);

-- Filter out those    join tuples where the departure    time in    Flights2 is not
after the arrival time in Flights1.
filteredJointFlights = FILTER jointFlights BY (firstFlights::ArrTime <
secondFlights::DepTime);

-- Average calculation
arrDelays = FOREACH filteredJointFlights GENERATE (firstFlights::ArrDelayMinutes +
secondFlights::ArrDelayMinutes) AS delay;
groupedArrDelays = GROUP arrDelays ALL;
avgDelay = FOREACH groupedArrDelays GENERATE AVG(arrDelays);

-- Store Delay
STORE avgDelay INTO '$OUTPUT';
```

```
--DUMP avgDelay;
```

PERFORMACE COMPARISON:

Program-Type	Run-Time
Plain (JAVA)	4 min 47 sec
Join-First (Version 1)	6 min 40 sec
Join-First (Version 2)	6 min 31 sec
Filter-First (Version 1)	6 min 27 sec
Filter-First (Version 2)	7 min 26 sec

From the above run-times, it is clear that the Java program performed the best. This difference in run-times could be because pig programs are eventually converted into Map-reduce tasks, therefore, there's an overhead in converting Pig programs to Mapreduce tasks. Also any tune-ups made in the Java program (grouping comparison, secondary sort and enhancing the join logic) are also missing in Pig. And all the pig programs load the data twice and apply filter to the whole data twice, which could also be time consuming.

Among the 4 pig programs, the Filter-First version 1 has the least run-time although Join-first (v1 & 2) and Filter-first (v1) are very close. The version 2 for filter-first has slightly higher run-time than others.

1) Join-first v1 vs v2: I had expected v2 to perform slightly better than v1, because in both cases, after join operation, both the flights have the same date. Hence performing additional check of the validity of date of flight 2 is unnecessary and adds to more processing time in v1. As expected in v1, the extra check adds a little more execution time.
2) Join-first vs Filter-first: I was expecting Fliter-first to perform almost same when compared to Join-first. Filter-first filters out unnecessary data before the join, so that should have been better. But Filter-filter performs almost the same as Join-first.
3) Filter-first v1 vs v2: I have modified the Filter-first version to load the data once and use it for Flight1 and Flight2. Although I expected both performs almost same, v2 has higher execution time than v1.

From the above observation of the Pig programs, we can conclude that the Pig programs, even with their minor variations, should and have perform almost same. The reason being Pig has it's own query plan optimization and it optimizes across an entire scripts. So the result becomes the same.

RESULT:

The table below reports the average delay for each of the programs:

Program-Type	**Average Delay**
Plain (JAVA)	50.671241
Join-First (Version 1)	50.67124150519758
Join-First (Version 2)	50.67124150519758
Filter-First (Version 1)	50.67124150519758
Filter-First (Version 2)	50.67124150519758

From the table above, the average delay calculated by the Java & Pig programs are the same.