CourseNumber: CS6240 Paralleled Data Processing
HomeworkNumber: HW3
Name: Yihan Tian

Source Code

Plain Java Program

- Design: filter and join data in Mapper and aggregate to compute average delay in Reducer. Used a helper class, RecordValidator to process data.
- DataModel: Record, represents a cleaned, valid line of record. Contains valid flight data information of date, time, middleCity, isFirstFlight, and delay.
- RecordValidator: in this class, read from original data file, and process each line to create a record. First validate if a line of record is from origin “ORD” or to destination “JFK”, and if the flight date is between June 2007 and May 2008, and if the flight is cancelled or diverted. If all these requirements meet, then create a Record correspondingly. Set Record date to date, set delay to arrDelayMins, if record’s origin is “ORD”, set isFirstFlight to true, time to arrTime, middleCity to destCity; if record’s destination is “JFK”, set isFirstFlight to false, time to depTime, middleCity to originCity.
- Mapper: Use setup for the whole task to setup a HashMap for join. The HashMap uses middleCity(String) as key, and list of Records with corresponding middleCity(ArrayList<Record>) as value. When getting a new record, first look into the map to check if there are some records also has a same middleCity, if so then check if there exists a record that can connect with the current record with suitable time. If a valid record is found in map, then combine it with incoming record, and emit this combined result to reducer, and removed the found record in map; else put the incoming record in map.
- Reducer: compute the average delay.

Pseudo Code

```
        Input: <Object, Text>
        Output: <Text, Text>
        Map {
          setup {
            findMap = new HashMap<String:middleCity, ArrayList<Record>>();
          }
          Mapper {
            inRecord = Validator.prepareRecord();
            If (findMap.hasKey(inRecord.getMiddleCity())) {
              If (findMap.get(inRecord.getMiddleCity()) has a record correspond with
inRecord) {
                delay = foundRecord.getDelay() + inRecord.getDelay();
                emit(“FoundRecofd”, delay);
              }
            } else {
                findMap.put(inRecord.getMiddleCity(), new ArrayList() including inRecord);
            }
        }
    }

    Input: <Text, DoubleWritable>
    Output: <Text, DoubleWritable>
    Reducer {
        Reduce {
            delay = sum of all input delays;
            count = total num of all input records;
            emit("AverageDelay",  delay/count);
        }
    }

    Validator {
        prepareRecord(line) {
            if (line.origin != "ORD" and lind.dest != "JFK") return null;
            if (line.date not between 2007.June and 2008.May) return null;
            record = new Record();
            if (line.origin == "ORD") {
                record.setDate(line.date);
                record.setTime(line.arrTime);
                record.setDelay(line.delay);
                record.setIsFirstFlight(true);
                record.setMiddleCity(line.dest);
            }
            if (line.dest == "JFK") {
                record.setDate(line.date);
                record.setTime(line.depTime);
                record.setDelay(line.delay);
                record.setIsFirstFlight(false);
                record.setMiddleCity(line.origin);
            }
        }
    }
```

Source Code

Validator Class

```java
/**
 * The type Data validator. Read, validate, and clean original data.
 */
public class DataValidator {

  private StringReader stringReader;
  private CSVReader csvReader;
  private static DataValidator instance = null;
  private DataValidator() throws IOException {

  }

  /**
   * Gets instance.
   *
   * @return the instance
   * @throws IOException the io exception
   */
  public static DataValidator getInstance() throws IOException {
    if (instance == null) {
      instance = new DataValidator();
    }
    return instance;
  }

  /**
   * Prepare record record. Read a line of record, check if its valid, if so parse the
data to Record.
   *
   * @param line the line
   * @return the record
   * @throws Exception the exception
   */
  public Record prepareRecord(String line)
      throws Exception {
    stringReader = new StringReader(line);
    csvReader = new CSVReader(stringReader);
    String[] value = csvReader.readNext();
    String depCity = value[Constants.DEP_IND].trim();
    String arrCity = value[Constants.ARR_IND].trim();
    String cancelled = value[Constants.CANCELLED_IND].trim();
    String diverted = value[Constants.DIVERTED_IND].trim();
    Date date = new SimpleDateFormat("yyyy-MM-
dd").parse(value[Constants.DATE_IND].trim());
    String depTime = value[Constants.DEP_TIME_IND].trim();
    String arrTime = value[Constants.ARR_TIME_IND].trim();
    String delay = value[Constants.DELAY_IND].trim();

    // Validate if the line of data is valid. If it is valid, get it's middle city.
    String middle = validateRecord(depCity, arrCity, date, cancelled, diverted);

    // If the data is valid, parse data into a Record.
    if (middle != null) {
      Record record = new Record();
      record.setDate(date);
      // Check if in this line of data there is a delay, if not just set delay to 0
      record.setDelay(validateString(delay) ? Double.valueOf(delay) : 0);
      record.setMiddleCity(middle);
      // Find out this record is the first flight or second flight in the two-legged
flight, and set
      // true if it is first, false for second.
      record.setFirstFlight(middle.equals(arrCity));
      // If the flight is first flight, set record time to flight's arrive time, else
set to departure time.
      record.setTime(middle.equals(arrCity) ? Integer.valueOf(arrTime) :
Integer.valueOf(depTime));
      return record;
    } else {
      return null;
    }
  }

  // Check if the line of data is valid.
  private String validateRecord(String dep, String arr, Date date, String cancelled,
      String diverted) {
    if((dep.equals(Constants.VALID_DEP) || arr.equals(Constants.VALID_ARR)) &&
cancelled
        .equals(Constants.VALID_STATUS) && diverted.equals(Constants.VALID_STATUS) &&
date
        .after(Constants.VALID_DATE_LOWER_BOUND) &&
date.before(Constants.VALID_DATE_UPPER_BOUND)) {
      return dep.equals(Constants.VALID_DEP) ? arr : dep;
    }
    return null;
  }

  // Check if a string is valid.
  private Boolean validateString(String str) {
    return (!str.equals("") && str.length() != 0 && !str.equals(null));
  }

  /**
   * Clean up.
   *
   * @throws IOException the io exception
   */
  public void cleanUp() throws IOException {
    this.stringReader.close();
    this.csvReader.close();
  }

}
```

MapReduce, Main class

```
/**
 * The main task that finds out valid records and compute average delay for them.
 */
public class AverageDelay {

  /**
   * The type Delay mapper.
   */
  public static class DelayMapper extends Mapper<Object, Text, Text, DoubleWritable> {

    // Use a HashMap for data filtering and combining. Use first flight's
    // dest city code/second flight's dep city code as key, and cleaned record, Record
as value.
    // For a new read in record, with a middle city X, if the findMap already has some
valid records
    // which dest/dep city is X, then search in the corresponding entry if there is
another record
    // that can pair up with this current read in record, if so emit the pair, else
put the new
    // record in map.
    private HashMap<String, ArrayList<Record>> findMap;
    private DataValidator validator;

    @Override
```

```java
    protected void setup(Context context) throws IOException, InterruptedException {
      super.setup(context);
      findMap = new HashMap<String, ArrayList<Record>>();
      validator = DataValidator.getInstance();
    }

    @Override
    protected void map(Object key, Text value, Context context)
        throws IOException, InterruptedException {
      try {
        Record record = validator.prepareRecord(value.toString());
        if (record != null) {
          String middleCity = record.getMiddleCity();
          Boolean isFirstFlight = record.getFirstFlight();
          if (findMap.containsKey(middleCity)) {
            // If there are already some records that has same middleCity, check if
that is valid
            // to be combined as a two-legged record.
            for (Record rec : findMap.get(middleCity)) {
              // Check if the found data and income data are first and second, and if
first time
              // before second time.
              if (!rec.getFirstFlight().equals(isFirstFlight) && rec.getDate()
                  .equals(record.getDate()) && ((isFirstFlight && record.getTime() <
rec.getTime())
                      || !isFirstFlight && record.getTime() > rec.getTime())) {
                // If there is a two-legged flight, calculate its delay and emit to
reducer.
                Double delay = rec.getDelay() + record.getDelay();
                findMap.get(middleCity).remove(rec);
                context.write(new Text("valid"), new DoubleWritable(delay));
              }
            }
          } else {
            // If there is not such a data in findMap that can create a two-legged
flight, put the
            // income record in map
            ArrayList<Record> recList = new ArrayList<Record>();
            recList.add(record);
            findMap.put(middleCity, recList);
          }
        }
      } catch (Exception e) {
        e.printStackTrace();
      }
    }

    @Override
    protected void cleanup(Context context) throws IOException, InterruptedException {
      super.cleanup(context);
      findMap.clear();
      validator.cleanUp();
    }
  }

  /**
   * The type Delay reducer. Aggregate data and compute average delay.
   */
  public static class DelayReducer extends Reducer<Text, DoubleWritable, Text, Text> {

    @Override
    protected void reduce(Text key, Iterable<DoubleWritable> values, Context context)
        throws IOException, InterruptedException {
      double totalDelay = 0.0;
      int count = 0;
      for (DoubleWritable val : values) {
        totalDelay += val.get();
        count++;
      }
      double avgDelay = (count == 0) ? 0 : (totalDelay / count);
      context.write(new Text("Average Delay: "), new Text(String.valueOf(avgDelay)));
    }
  }

  /**
   * The entry point of application.
   *
   * @param args the input arguments
   * @throws IOException the io exception
   * @throws ClassNotFoundException the class not found exception
   * @throws InterruptedException the interrupted exception
   */
  public static void main(String[] args)
      throws IOException, ClassNotFoundException, InterruptedException {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "AverageDelay");
    job.setJarByClass(AverageDelay.class);
    job.setMapperClass(DelayMapper.class);
    job.setReducerClass(DelayReducer.class);
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(DoubleWritable.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }

}
```

Record Model

```
/**
 * The type Record. Represent a valid, cleaned record of flight data.
 */
public class Record {
  private String middleCity;
  private Date date;
  private Integer time;
  private Double delay;
  private Boolean isFirstFlight;

  /**
   * Instantiates a new Record.
   */
  public Record() {

  }

  /**
   * Instantiates a new Record.
   *
   * @param middleCity the middle city
   * @param date the date
   * @param time the time
   * @param delay the delay
   * @param isFirstFlight the is first flight
   */
  public Record(String middleCity, Date date, Integer time, Double delay,
      Boolean isFirstFlight) {
    this.middleCity = middleCity;
    this.date = date;
    this.time = time;
    this.delay = delay;
    this.isFirstFlight = isFirstFlight;
  }

  /**
   * Gets middle city.
   *
   * @return the middle city
   */
  public String getMiddleCity() {
    return middleCity;
  }

  /**
   * Sets middle city.
   *
   * @param middleCity the middle city
   */
  public void setMiddleCity(String middleCity) {
    this.middleCity = middleCity;
  }

  /**
   * Gets date.
   *
   * @return the date
   */
  public Date getDate() {
    return date;
  }

  /**
   * Sets date.
   *
   * @param date the date
   */
  public void setDate(Date date) {
    this.date = date;
  }

  /**
   * Gets time.
   *
   * @return the time
   */
  public Integer getTime() {
    return time;
  }

  /**
   * Sets time.
   *
   * @param time the time
   */
  public void setTime(Integer time) {
    this.time = time;
  }

  /**
   * Gets delay.
   *
   * @return the delay
   */
  public Double getDelay() {
    return delay;
  }

  /**
   * Sets delay.
   *
   * @param delay the delay
   */
  public void setDelay(Double delay) {
    this.delay = delay;
  }

  /**
   * Gets first flight.
   *
   * @return the first flight
   */
  public Boolean getFirstFlight() {
    return isFirstFlight;
  }

  /**
   * Sets first flight.
   *
   * @param firstFlight the first flight
   */
  public void setFirstFlight(Boolean firstFlight) {
    isFirstFlight = firstFlight;
  }
}
```

Constants

```
/**
 * The type Constants.
 */
public class Constants {

  /**
   * The constant VALID_STATUS.
   */
  public static final String VALID_STATUS = "0.00";
  /**
   * The constant VALID_DEP.
   */
  public static final String VALID_DEP = "ORD";
  /**
   * The constant VALID_ARR.
   */
```

```java
  public static final String VALID_ARR = "JFK";
  /**
   * The constant VALID_DATE_LOWER_BOUND.
   */
  public static final Date VALID_DATE_LOWER_BOUND = new Date(107, Calendar.JUNE,1);
  /**
   * The constant VALID_DATE_UPPER_BOUND.
   */
  public static final Date VALID_DATE_UPPER_BOUND = new Date(108, Calendar.MAY,31);

  /**
   * The constant DATE_IND.
   */
  public static final int DATE_IND = 5;
  /**
   * The constant DEP_TIME_IND.
   */
  public static final int DEP_TIME_IND = 24;
  /**
   * The constant ARR_TIME_IND.
   */
  public static final int ARR_TIME_IND = 35;
  /**
   * The constant DEP_IND.
   */
  public static final int DEP_IND = 11;
  /**
   * The constant ARR_IND.
   */
  public static final int ARR_IND =17;
  /**
   * The constant DELAY_IND.
   */
  public static final int DELAY_IND = 37;
  /**
   * The constant CANCELLED_IND.
   */
  public static final int CANCELLED_IND = 41;
  /**
   * The constant DIVERTED_IND.
   */
  public static final int DIVERTED_IND = 43;
}
```

Pig Latin Program

Source Code

Filter First

```
register file:/usr/lib/pig/lib/piggybank.jar
define CSVLoader org.apache.pig.piggybank.storage.CSVLoader;

-- Load file to Flights1 andd Flights2
Flights1 = load '$INPUT' using CSVLoader as
(Year:int,Quarter,Month:int,DayofMonth,DayOfWeek,FlightDate,UniqueCarrier,AirlineID,Ca
rrier,TailNum,FlightNum,Origin,OriginCityName,OriginState,OriginStateFips,
OriginStateNam,OriginWac,Dest,DestCityName,DestState,DestStateFips,DestStateName,DestW
ac,CRSDepTime,DepTime,DepDelay,DepDelayMinutes,DepDel15,

DepartureDelayGroups,DepTimeBlk,TaxiOut,WheelsOff,WheelsOn,TaxiIn,CRSArrTime,ArrTime:i
nt,ArrDelay,ArrDelayMinutes:int,ArrDel15,ArrivalDelayGroups,ArrTimeBlk,Cancelled,

CancellationCode,Diverted,CRSElapsedTime,ActualElapsedTime,AirTime,Flights,Distance,Di
stanceGroup,CarrierDelay,WeatherDelay,NASDelay,SecurityDelay,LateAircraftDelay);

Flights2 = load '$INPUT' using CSVLoader as
(Year:int,Quarter,Month:int,DayofMonth,DayOfWeek,FlightDate,UniqueCarrier,AirlineID,Ca
rrier,TailNum,FlightNum,Origin,OriginCityName,OriginState,OriginStateFips,

OriginStateNam,OriginWac,Dest,DestCityName,DestState,DestStateFips,DestStateName,DestW
ac,CRSDepTime,DepTime:int,DepDelay,DepDelayMinutes,DepDel15,

DepartureDelayGroups,DepTimeBlk,TaxiOut,WheelsOff,WheelsOn,TaxiIn,CRSArrTime,ArrTime,A
rrDelay,ArrDelayMinutes:int,ArrDel15,ArrivalDelayGroups,ArrTimeBlk,Cancelled,

CancellationCode,Diverted,CRSElapsedTime,ActualElapsedTime,AirTime,Flights,Distance,Di
stanceGroup,CarrierDelay,WeatherDelay,NASDelay,SecurityDelay,LateAircraftDelay);

-- Remove columns that are not used
First =  foreach Flights1 generate Year, Month, FlightDate, Origin, Dest, ArrTime,
DepTime, ArrDelayMinutes, Cancelled, Diverted;
Second = foreach Flights2 generate Year, Month, FlightDate, Origin, Dest, ArrTime,
DepTime, ArrDelayMinutes, Cancelled, Diverted;

-- Filter out flight with conditions
FirstFlight = filter First by (((Year == 2007 and Month >= 6) or (Year == 2008 and
Month <= 5)) and Origin == 'ORD' and Dest != 'JFK' and Cancelled == '0.00' and
Diverted == '0.00');
SecondFlight = filter Second by ((Year == 2007 and Month >= 6) or (Year == 2008 and
Month <= 5) and Origin != 'ORD' and Dest == 'JFK' and Cancelled == '0.00' and Diverted
== '0.00');

-- Join data, and remove those flights which time doesn't match
JoinFlight = join FirstFlight by (FlightDate, Dest), SecondFlight by (FlightDate,
Origin);
Filtered = filter JoinFlight by (FirstFlight::ArrTime < SecondFlight::DepTime);

-- Calculate average delay
Delay = foreach Filtered generate (FirstFlight::ArrDelayMinutes +
SecondFlight::ArrDelayMinutes) as delay;
Grouped = group Delay all;
AvgDelay = foreach Grouped generate AVG(Delay);
```

```
store AvgDelay into '$OUTPUT';
```

Join First Version1

```
register file:/usr/lib/pig/lib/piggybank.jar
define CSVLoader org.apache.pig.piggybank.storage.CSVLoader;

Flights1 = load '$INPUT' using CSVLoader as
(Year:int,Quarter,Month:int,DayofMonth,DayOfWeek,FlightDate,UniqueCarrier,AirlineID,Ca
rrier,TailNum,FlightNum,Origin,OriginCityName,OriginState,OriginStateFips,

OriginStateNam,OriginWac,Dest,DestCityName,DestState,DestStateFips,DestStateName,DestW
ac,CRSDepTime,DepTime,DepDelay,DepDelayMinutes,DepDel15,

DepartureDelayGroups,DepTimeBlk,TaxiOut,WheelsOff,WheelsOn,TaxiIn,CRSArrTime,ArrTime:i
nt,ArrDelay,ArrDelayMinutes:int,ArrDel15,ArrivalDelayGroups,ArrTimeBlk,Cancelled,

CancellationCode,Diverted,CRSElapsedTime,ActualElapsedTime,AirTime,Flights,Distance,Di
stanceGroup,CarrierDelay,WeatherDelay,NASDelay,SecurityDelay,LateAircraftDelay);

Flights2 = load '$INPUT' using CSVLoader as
(Year:int,Quarter,Month:int,DayofMonth,DayOfWeek,FlightDate,UniqueCarrier,AirlineID,Ca
rrier,TailNum,FlightNum,Origin,OriginCityName,OriginState,OriginStateFips,

OriginStateNam,OriginWac,Dest,DestCityName,DestState,DestStateFips,DestStateName,DestW
ac,CRSDepTime,DepTime:int,DepDelay,DepDelayMinutes,DepDel15,

DepartureDelayGroups,DepTimeBlk,TaxiOut,WheelsOff,WheelsOn,TaxiIn,CRSArrTime,ArrTime,A
rrDelay,ArrDelayMinutes:int,ArrDel15,ArrivalDelayGroups,ArrTimeBlk,Cancelled,

CancellationCode,Diverted,CRSElapsedTime,ActualElapsedTime,AirTime,Flights,Distance,Di
stanceGroup,CarrierDelay,WeatherDelay,NASDelay,SecurityDelay,LateAircraftDelay);

FirstCleaned1 = foreach Flights1 generate Year, Month, FlightDate, Origin, Dest,
ArrDelayMinutes, ArrTime, DepTime, Cancelled, Diverted;
FirstCleaned2 = foreach Flights2 generate Year, Month, FlightDate, Origin, Dest,
ArrDelayMinutes, ArrTime, DepTime, Cancelled, Diverted;

FirstFiltered1 = filter FirstCleaned1 by (Origin == 'ORD' and Dest != 'JFK' and
Cancelled == '0.00' and Diverted == '0.00');
FirstFiltered2 = filter FirstCleaned2 by (Origin != 'ORD' and Dest == 'JFK' and
Cancelled == '0.00' and Diverted == '0.00');

JointData = join FirstFiltered1 by (Dest, FlightDate), FirstFiltered2 by (Origin,
FlightDate);

FilteredJoint = filter JointData by FirstFiltered1::ArrTime < FirstFiltered2::DepTime;
```

```
FinalFiltered = filter FilteredJoint by ((FirstFiltered1::Year == 2007 and
FirstFiltered1::Month >= 6) or (FirstFiltered1::Year == 2008 and FirstFiltered1::Month
<= 5)) and ((FirstFiltered2::Year == 2007 and FirstFiltered2::Month >= 6) or
(FirstFiltered2::Year == 2008 and FirstFiltered2::Month <= 5));

TotalDelay = foreach FinalFiltered generate (FirstFiltered1::ArrDelayMinutes +
FirstFiltered2::ArrDelayMinutes) as total;
Grouped = group TotalDelay all;
AvgDelay = foreach Grouped generate AVG(TotalDelay);

store AvgDelay into '$OUTPUT';
```

Join First Version2

```
register file:/usr/lib/pig/lib/piggybank.jar
define CSVLoader org.apache.pig.piggybank.storage.CSVLoader;

Flights1 = load '$INPUT' using CSVLoader as
(Year:int,Quarter,Month:int,DayofMonth,DayOfWeek,FlightDate,UniqueCarrier,AirlineID,Ca
rrier,TailNum,FlightNum,Origin,OriginCityName,OriginState,OriginStateFips,

OriginStateNam,OriginWac,Dest,DestCityName,DestState,DestStateFips,DestStateName,DestW
ac,CRSDepTime,DepTime,DepDelay,DepDelayMinutes,DepDel15,

DepartureDelayGroups,DepTimeBlk,TaxiOut,WheelsOff,WheelsOn,TaxiIn,CRSArrTime,ArrTime:i
nt,ArrDelay,ArrDelayMinutes:int,ArrDel15,ArrivalDelayGroups,ArrTimeBlk,Cancelled,

CancellationCode,Diverted,CRSElapsedTime,ActualElapsedTime,AirTime,Flights,Distance,Di
stanceGroup,CarrierDelay,WeatherDelay,NASDelay,SecurityDelay,LateAircraftDelay);

Flights2 = load '$INPUT' using CSVLoader as
(Year:int,Quarter,Month:int,DayofMonth,DayOfWeek,FlightDate,UniqueCarrier,AirlineID,Ca
rrier,TailNum,FlightNum,Origin,OriginCityName,OriginState,OriginStateFips,

OriginStateNam,OriginWac,Dest,DestCityName,DestState,DestStateFips,DestStateName,DestW
ac,CRSDepTime,DepTime:int,DepDelay,DepDelayMinutes,DepDel15,

DepartureDelayGroups,DepTimeBlk,TaxiOut,WheelsOff,WheelsOn,TaxiIn,CRSArrTime,ArrTime,A
rrDelay,ArrDelayMinutes:int,ArrDel15,ArrivalDelayGroups,ArrTimeBlk,Cancelled,

CancellationCode,Diverted,CRSElapsedTime,ActualElapsedTime,AirTime,Flights,Distance,Di
stanceGroup,CarrierDelay,WeatherDelay,NASDelay,SecurityDelay,LateAircraftDelay);
```

```
FirstCleaned1 = foreach Flights1 generate Year, Month, FlightDate, Origin, Dest,
ArrDelayMinutes, ArrTime, DepTime, Cancelled, Diverted;
FirstCleaned2 = foreach Flights2 generate Year, Month, FlightDate, Origin, Dest,
ArrDelayMinutes, ArrTime, DepTime, Cancelled, Diverted;

FirstFiltered1 = filter FirstCleaned1 by (Origin == 'ORD' and Dest != 'JFK' and
Cancelled == '0.00' and Diverted == '0.00');
FirstFiltered2 = filter FirstCleaned2 by (Origin != 'ORD' and Dest == 'JFK' and
Cancelled == '0.00' and Diverted == '0.00');

JointData = join FirstFiltered1 by (Dest, FlightDate), FirstFiltered2 by (Origin,
FlightDate);

FilteredJoint = filter JointData by FirstFiltered1::ArrTime < FirstFiltered2::DepTime;

FinalFiltered = filter FilteredJoint by ((FirstFiltered1::Year == 2007 and
FirstFiltered1::Month >= 6) or (FirstFiltered1::Year == 2008 and FirstFiltered1::Month
<= 5)));
TotalDelay = foreach FinalFiltered generate (FirstFiltered1::ArrDelayMinutes +
FirstFiltered2::ArrDelayMinutes) as total;
Grouped = group TotalDelay all;
AvgDelay = foreach Grouped generate AVG(TotalDelay);

store AvgDelay into '$OUTPUT';
```

Performance

Runtime Report:

Plain Java	72s
Filter-First	82s
Join-First-Version1	54s
Join-First-Version2	54s

Runtime Analysis

Among these 4 programs, Join-First output the best performance, but version1 and virsion2 didn't show much difference in runtime. Plain Java did little better than Filter-First pig program, but Join-First is still much better.

Before running, I expected Java perform better than Pig, because pig programs has to load data and generate new data and it might take many time; in pig filter first performs better than join

first, and join-first version 2 performs better than join-first version 1, because I expected that with filtered data, and less record, further computation could be faster.
So in the comparison, the performances showed in table above, and reasons could be:

- Java-Pig: Virtual machines for java program need more time to start up
- Join-first 2 versions: though records and conditions are less in version 2, 2 programs still need to load same times of data, that takes more portion of the runtime
- Filter/join first: After the join, 2 datasets were combined to 1 at very begining, and in further process, no need to load twice of the data, which saves time and resources.

Average Flight Delay:

Plain Java	50.47260274
Filter-First	50.65014143
Join-First-Version1	50.65014143
Join-First-Version2	50.65014143