Name: Khoa Cu Dang (Cody) Cao

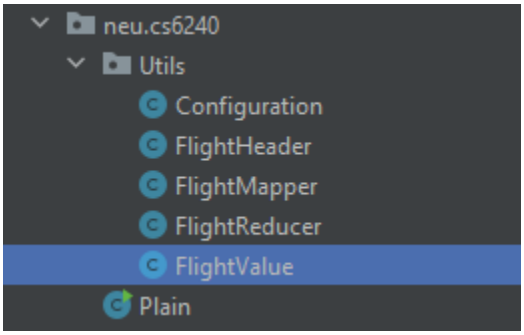
Course: CS6620 Spring 2024

Homework 3 Report

Source Code

Plain MapReduce

Like Homework 3, the Plain program for Homework 3 has its code separated into utility classes.



Configuration: This class stores all required constant values such as original airport, destination airports, date range, etc.

```
15 usages  khoacao-ccdk +1 *
public class Configuration {
    2 usages
    public static final int NUM_REDUCER = 10;
    3 usages
    public static final String ORIGIN_AIRPORT = "ORD";
    2 usages
    public static final String DEST_AIRPORT = "JFK";
    2 usages
    public static final int INTERVAL_START_YEAR = 2007;
    1 usage
    public static final int INTERVAL_START_MONTH = 6;
    2 usages
    public static final int INTERVAL_END_YEAR = 2008;
    1 usage
    public static final int INTERVAL_END_MONTH = 5;
    1 usage
    public static final String EXPECTED_CANCELLED_STATUS = "0.00";
    1 usage
    public static final String EXPECTED_DIVERTED_STATUS = "0.00";
}
```

FlightHeader: this class stores the index of the parameters within a CSV line of flight record that is needed for the purpose of this program

```java
/**
 * This class acts as a header for the column values of the data
 *
 * @author Cody Cao
 */
11 usages  khoacao-ccdk
public class FlightHeader {
    1 usage
    public static final int YEAR = 0;
    1 usage
    public static final int MONTH = 2;
    1 usage
    public static final int FLIGHT_DATE = 5;
    1 usage
    public static final int ORIGIN = 11;
    1 usage
    public static final int DEST = 17;
    1 usage
    public static final int DEP_TIME = 24;
    1 usage
    public static final int ARR_TIME = 35;
    1 usage
    public static final int ARR_DELAY_MINUTES = 37;
    1 usage
    public static final int CANCELLED = 41;
    1 usage
    public static final int DIVERTED = 43;
}
```

FlightMapper: The map() function is called for each line of the data. It then performs the necessary filters.

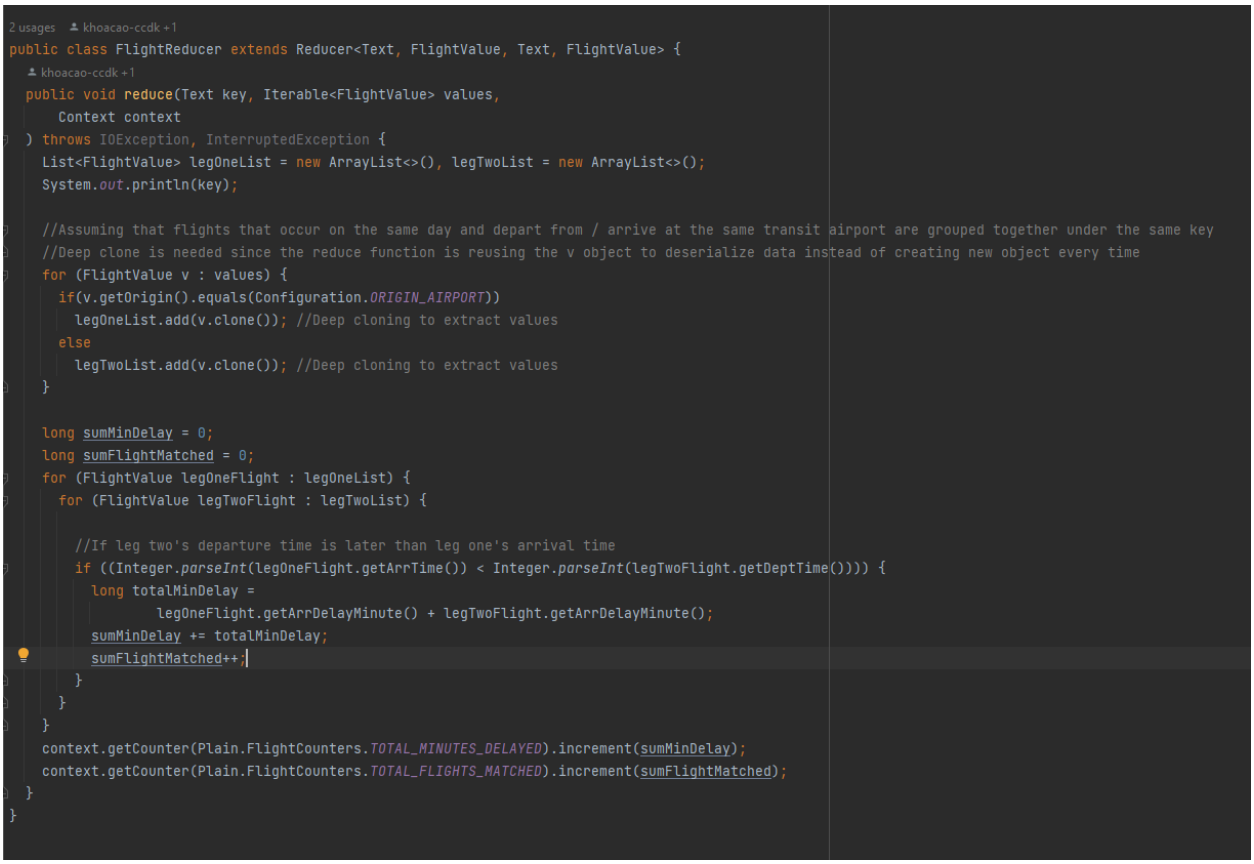
- Not cancelled/diverted.
- Within the specified date range.
- Originates from the ORD airport or arrives at JFK airport.

After filtering, the record is emitted as a key-value pair, with the key being a combination of flight date and the transit airport, while the value being a FlightValue object that contains necessary values.

```
2 usages  khoacao-ccdk +1
public class FlightMapper extends Mapper<Object, Text, Text, FlightValue> {
  khoacao-ccdk +1
  public void map(Object key, Text value, Context context)
      throws IOException, InterruptedException {
    CSVParser parser = new CSVParser();
    String[] tokens = parser.parseLine(value.toString());
    String origin = tokens[FlightHeader.ORIGIN];
    String dest = tokens[FlightHeader.DEST];
    int year = Integer.parseInt(tokens[FlightHeader.YEAR]);
    int month = Integer.parseInt(tokens[FlightHeader.MONTH]);
    String cancelled = tokens[FlightHeader.CANCELLED];
    String diverted = tokens[FlightHeader.DIVERTED];
    String flightDate = tokens[FlightHeader.FLIGHT_DATE];

    //Only consider non-cancelled and non-diverted flights
    if (!(cancelled.equals(Configuration.EXPECTED_CANCELLED_STATUS)
        && diverted.equals(Configuration.EXPECTED_DIVERTED_STATUS))) {
      return;
    }

    //Only consider flights within the defined time range
    if ((year > Configuration.INTERVAL_START_YEAR || (year == Configuration.INTERVAL_START_YEAR
        && month >= Configuration.INTERVAL_START_MONTH))
        &&
        (year < Configuration.INTERVAL_END_YEAR || (year == Configuration.INTERVAL_END_YEAR
            && month <= Configuration.INTERVAL_END_MONTH))) {
      String transitAirport;
      if (origin.equals(Configuration.ORIGIN_AIRPORT) && !dest.equals(Configuration.DEST_AIRPORT)){
        transitAirport = dest;
      }
      else if (!origin.equals(Configuration.ORIGIN_AIRPORT) && dest.equals(Configuration.DEST_AIRPORT)){
        transitAirport = origin;
      }
      else return;


      //Filtering necessary data only
      String depTime = tokens[FlightHeader.DEP_TIME];
      String arrTime = tokens[FlightHeader.ARR_TIME];
      int arrDelayMinutes = (int) Math.round(Double.parseDouble(tokens[FlightHeader.ARR_DELAY_MINUTES]));

      FlightValue fValue = new FlightValue(origin, dest, depTime, arrTime, arrDelayMinutes);
      context.write(new Text(String.format("%s,%s", flightDate, transitAirport)), fValue);
    }
  }
}
```

FlightReducer: The logics of the reducer comes with an expectation that all values with the same key are matched together, which means each reduce call should have access to every flight on the same date that has the same transit airport.

The calculated results are then used to update the global counters so that there is a centralized counter value.



```
2 usages  khoacao-ccdk +1
public class FlightReducer extends Reducer<Text, FlightValue, Text, FlightValue> {
  khoacao-ccdk +1
  public void reduce(Text key, Iterable<FlightValue> values,
      Context context
  ) throws IOException, InterruptedException {
    List<FlightValue> legOneList = new ArrayList<>(), legTwoList = new ArrayList<>();
    System.out.println(key);

    //Assuming that flights that occur on the same day and depart from / arrive at the same transit airport are grouped together under the same key
    //Deep clone is needed since the reduce function is reusing the v object to deserialize data instead of creating new object every time
    for (FlightValue v : values) {
      if(v.getOrigin().equals(Configuration.ORIGIN_AIRPORT))
        legOneList.add(v.clone()); //Deep cloning to extract values
      else
        legTwoList.add(v.clone()); //Deep cloning to extract values
    }

    long sumMinDelay = 0;
    long sumFlightMatched = 0;
    for (FlightValue legOneFlight : legOneList) {
      for (FlightValue legTwoFlight : legTwoList) {

        //If leg two's departure time is later than leg one's arrival time
        if ((Integer.parseInt(legOneFlight.getArrTime()) < Integer.parseInt(legTwoFlight.getDeptTime()))) {
          long totalMinDelay =
                  legOneFlight.getArrDelayMinute() + legTwoFlight.getArrDelayMinute();
          sumMinDelay += totalMinDelay;
          sumFlightMatched++;
        }
      }
    }
    context.getCounter(Plain.FlightCounters.TOTAL_MINUTES_DELAYED).increment(sumMinDelay);
    context.getCounter(Plain.FlightCounters.TOTAL_FLIGHTS_MATCHED).increment(sumFlightMatched);
  }
}
```

Main class: This class sets up the mapreduce program, as well as writing the result values to the output file after the program's execution.

A flaw with this approach is that I could not figure out a way to write to a file in S3. Thus, I used System.out.println(), which would then write results to the stdout file.

```java
/**
 * This class performs the plain MapReduce approach to calculate the average delay time
 *
 * @author Cody Cao
 */
khoacao-ccdk +1 *
public class Plain {

  4 usages  khoacao-ccdk
  public static enum FlightCounters {
    2 usages
    TOTAL_FLIGHTS_MATCHED,
    2 usages
    TOTAL_MINUTES_DELAYED
  }

  khoacao-ccdk +1 *
  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf,  jobName: "Average Delay - Plain");
    job.setJarByClass(Plain.class);
    job.setMapperClass(FlightMapper.class);

    //Setting number of reducer in accordance to the number of words
    job.setReducerClass(FlightReducer.class);
    job.setNumReduceTasks(NUM_REDUCER);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(FlightValue.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    Path outputPath = new Path(args[1]);
    FileOutputFormat.setOutputPath(job, outputPath);
    boolean success = job.waitForCompletion( verbose: true);

    if (success) {
      Counters counters = job.getCounters();

      long totalFlightsMatched = counters.findCounter(FlightCounters.TOTAL_FLIGHTS_MATCHED)
          .getValue();
      long totalMinutesDelayed = counters.findCounter(FlightCounters.TOTAL_MINUTES_DELAYED)
          .getValue();
      writeGlobalCounter(totalFlightsMatched, totalMinutesDelayed);
    }
    System.exit(success ? 0 : 1);
  }

  1 usage  khoacao-ccdk *
  private static void writeGlobalCounter(long totalFlightsMatched, long totalMinutesDelayed) {
    // Write the counter values to the file
    double avgDelay = 1.0d * totalMinutesDelayed / totalFlightsMatched;
    System.out.println("Total flights matched: " + totalFlightsMatched + "\n");
    System.out.println("Total minutes delayed: " + totalMinutesDelayed + "\n");
    System.out.println("Average delay minutes: " + avgDelay);
  }
}
```

Pseudo code

Mapper:

1. Read each line from input.
2. Parse the line using a CSV parser to get individual fields.
3. Extract origin, destination, year, month, cancelled status, diverted status, and flight date from the parsed tokens.
4. Check if the flight is not cancelled and not diverted. If cancelled or diverted, skip the record.
5. Check if the flight falls within the defined time range. If not, skip the record.
6. Determine the transit airport based on origin and destination.
7. Extract departure time, arrival time, and arrival delay minutes.
8. Emit key-value pairs where the key is a combination of flight date and transit airport, and the value is a FlightValue object containing origin, destination, departure time, arrival time, and arrival delay minutes.

Reducer:

1. Receive key-value pairs grouped by the transit airport and flight date.
2. Separate flights into two lists: legOneList for flights departing from ORD and legTwoList for flights arriving at JFK.
3. Iterate over each flight in legOneList and legTwoList.
4. Check if the departure time of leg two's flight is later than the arrival time of leg one's flight.
5. If the condition is met, calculate the total delay minutes as the sum of arrival delay minutes of leg one and leg two's flights.
6. Increment global counters for total minutes delayed and total flights matched.

After the program execution, writes the global counter values to the stdout file of the Step on EMR.

Pig FilterFirst

```
REGISTER s3://cs6240-hw1-cody/piggybank.jar
DEFINE CSVLoader org.apache.pig.piggybank.storage.CSVLoader;

-- Setting number of reducer tasks to 10
SET default_parallel 10;

-- Loads data from parameter
Flight = LOAD '$input' using CSVLoader();

-- Parse required columns as Flight1_data
Flight1_data = FOREACH Flight GENERATE $5 as FlightDate, $11 as Origin, $17 as Dest,
                $24 as DepTime, $35 as ArrTime, $37 as ArrDelayMinutes, $41 as Cancelled,
            $43 as Diverted;

-- Parse required columns as Flight2_data
Flight2_data = FOREACH Flight GENERATE $5 as FlightDate, $11 as Origin, $17 as Dest,
            $24 as DepTime, $35 as ArrTime, $37 as ArrDelayMinutes, $41 as Cancelled,
            $43 as Diverted;

-- Filter unnecessary records for Flight1_data & Flight2_data
Flight1_data = FILTER Flight1_data BY (Origin eq 'ORD' AND Dest neq 'JFK') AND (Cancelled neq '1' OR Diverted neq '1')
            AND (ToDate(FlightDate,'yyyy-MM-dd') < ToDate('2008-06-01','yyyy-MM-dd')) AND
            (ToDate(FlightDate,'yyyy-MM-dd') > ToDate('2007-05-31','yyyy-MM-dd'));

Flight2_data = FILTER Flight2_data BY (Origin neq 'ORD' AND Dest eq 'JFK') AND (Cancelled neq '1' OR Diverted neq '1')
            AND (ToDate(FlightDate,'yyyy-MM-dd') < ToDate('2008-06-01','yyyy-MM-dd')) AND
            (ToDate(FlightDate,'yyyy-MM-dd') > ToDate('2007-05-31','yyyy-MM-dd'));

-- Join Flight1_data with Flight2_data based on date and transit airport
Results = JOIN Flight1_data by (FlightDate, Dest), Flight2_data by (FlightDate, Origin);
Results = FILTER Results BY $4 < $11;
Results = FILTER Results BY (ToDate($0,'yyyy-MM-dd') < ToDate('2008-06-01','yyyy-MM-dd')) AND
     (ToDate($0,'yyyy-MM-dd') > ToDate('2007-05-31','yyyy-MM-dd'));

-- Calculate TotalDelay for every filtered flights
Results = FOREACH Results GENERATE (float)($5 + $13) as TotalDelay;

-- Calculate total delay and average
Grouped = GROUP Results all;
Average = FOREACH Grouped GENERATE AVG(Results.TotalDelay);

-- Store the output to S3
STORE Average into '$output' USING PigStorage();
--STORE Average into '$output';
```

Pseudocode:

1. Define a CSVLoader for loading CSV files.
2. Set the default number of reducer tasks to 10.
3. Load data from the input parameter into Flight using the CSVLoader.
4. Parse required columns from Flight into Flight1_data and Flight2_data.
5. Filter the data in Flight1_data and Flight_2 data (not cancelled/diverted, within date range, correct departure/arrival airport)
6. Join Flight1_data with Flight2_data based on date and transit airport
7. Filter Results to keep only relevant flights based on date.
8. Filter Results to keep only records that are valid (leg 2 deptTime > leg 1 arrTime)
9. Filter Results to keep records that are within the date range
10. Calculate the total delay for each filtered flight in Results.
11. Group Results to calculate the average total delay.
12. Store the average total delay to the output location specified.

Pig JoinFirst V1

```
REGISTER s3://cs6240-hw1-cody/piggybank.jar
DEFINE CSVLoader org.apache.pig.piggybank.storage.CSVLoader;

-- Set number of reducer tasks to 10
SET default_parallel 10;

-- Load the input from parameter
Flight = LOAD '$input' using CSVLoader();

-- Parse required columns as Flight1
Flight1 = FOREACH Flight GENERATE $5 as FlightDate, $11 as Origin, $17 as Dest,
          $24 as DepTime, $35 as ArrTime, $37 as ArrDelayMinutes, $41 as Cancelled,
      $43 as Diverted;
-- Parse required columns as Flight2
Flight2 = FOREACH Flight GENERATE $5 as FlightDate, $11 as Origin, $17 as Dest,
          $24 as DepTime, $35 as ArrTime, $37 as ArrDelayMinutes, $41 as Cancelled,
      $43 as Diverted;

-- Filter unnecessary records
Flight1_data = FILTER Flight1 BY (Origin eq 'ORD' AND Dest neq 'JFK') AND (Cancelled neq '1' OR Diverted neq '1');
Flight2_data = FILTER Flight2 BY (Origin neq 'ORD' AND Dest eq 'JFK') AND (Cancelled neq '1' OR Diverted neq '1');

-- Join Flight1_data with Flight2_data based on date and intermidiate airport
Results = JOIN Flight1_data by (FlightDate, Dest), Flight2_data by (FlightDate, Origin);

-- Filter data if the time & date range doesn't match
Results = FILTER Results BY $4 < $11;
Results = FILTER Results BY (ToDate($0,'yyyy-MM-dd') < ToDate('2008-06-01','yyyy-MM-dd'))
        AND (ToDate($0,'yyyy-MM-dd') > ToDate('2007-05-31','yyyy-MM-dd'))
        AND (ToDate($8,'yyyy-MM-dd') < ToDate('2008-06-01','yyyy-MM-dd'))
        AND (ToDate($8,'yyyy-MM-dd') > ToDate('2007-05-31','yyyy-MM-dd'));

-- Calculate TotalDelay for every filterred twolegged flights
Results = FOREACH Results GENERATE (float)($5 + $13) as TotalDelay;

-- Calculate total delay and average
Grouped = GROUP Results all;
Average = FOREACH Grouped GENERATE AVG(Results.TotalDelay);

-- Store the output to S3
STORE Average into '$output' USING PigStorage();
```

Pseudocode:

1. Define a CSVLoader to handle CSV file loading.
2. Set the default number of reducer tasks to 10.
3. Load the input data from the input parameter using the defined CSVLoader into Flight.
4. Parse the required columns from `Flight` into `Flight1` and `Flight2`.
5. Filter out unnecessary records from `Flight1` and `Flight2` (only keep non-cancelled and non-diverted flights).
6. Join `Flight1` with `Flight2` based on the date and transit airport.
7. Filter Results to keep only records that are valid (leg 2 deptTime > leg 1 arrTime)
8. Further filter the joined data to ensure flight date of BOTH leg 1 and leg 2 are within the desired date range.
9. Calculate the total delay for each filtered flight leg.
10. Group the results to calculate the average total delay across all flight legs.
11. Store the average total delay to the output location specified.

Pig JoinFirst V2

```
REGISTER s3://cs6240-hw1-cody/piggybank.jar
DEFINE CSVLoader org.apache.pig.piggybank.storage.CSVLoader;

-- Set number of reducer tasks to 10
SET default_parallel 10;

-- Load the input from parameter
Flight = LOAD '$input' using CSVLoader();

-- Parse only required columns as Flight1
Flight1 = FOREACH Flight GENERATE $5 as FlightDate, $11 as Origin, $17 as Dest,
          $24 as DepTime, $35 as ArrTime, $37 as ArrDelayMinutes, $41 as Cancelled,
      $43 as Diverted;
-- Parse only required columns as Flight2
Flight2 = FOREACH Flight GENERATE $5 as FlightDate, $11 as Origin, $17 as Dest,
          $24 as DepTime, $35 as ArrTime, $37 as ArrDelayMinutes, $41 as Cancelled,
      $43 as Diverted;

-- Filter unnecessary records
Flight1_data = FILTER Flight1 BY (Origin eq 'ORD' AND Dest neq 'JFK') AND (Cancelled neq '1' OR Diverted neq '1');
Flight2_data = FILTER Flight2 BY (Origin neq 'ORD' AND Dest eq 'JFK') AND (Cancelled neq '1' OR Diverted neq '1');

-- Join Flight1_data with Flight2_data based on date and intermidiate airport
Results = JOIN Flight1_data by (FlightDate, Dest), Flight2_data by (FlightDate, Origin);

-- Filter data if the time & date range doesn't match
Results = FILTER Results BY $4 < $11;
Results = FILTER Results BY (ToDate($0,'yyyy-MM-dd') < ToDate('2008-06-01','yyyy-MM-dd')) AND
      (ToDate($0,'yyyy-MM-dd') > ToDate('2007-05-31','yyyy-MM-dd'));

-- Calculate TotalDelay for every filterred twolegged flights
Results = FOREACH Results GENERATE (float)($5 + $13) as TotalDelay;

-- Calculating total delay and average
Grouped = GROUP Results all;
Average = FOREACH Grouped GENERATE AVG(Results.TotalDelay);

-- Storing the output to S3
STORE Average into '$output' USING PigStorage();
```

Pseudocode:

1. Define a CSVLoader to handle loading CSV files.
2. Set the default number of reducer tasks to 10.
3. Load the input data from input using the defined CSVLoader to `Flight`.
4. Extract only the required columns from `Flight` into `Flight1` and `Flight2`.
5. Filter out unnecessary records from `Flight1` and `Flight2` (only keep non-cancelled and non-diverted flights).
6. Join `Flight1` with `Flight2` based on the date and transit airport.
7. Further filter the joined data to ensure it falls within the desired time and date range.
8. Further filter the joined data to ensure flight date of leg 1 is within the desired date range.
9. Group the results to calculate the average total delay across all filtered flights.
10. Store the calculated average total delay to the specified output location.

Output Results

Plain

```
Total flights matched: 466756

Total minutes delayed: 23651106

Average delay minutes: 50.67124150519758
```

Filter First

```
50.67124150519758
```

JoinFirst V1

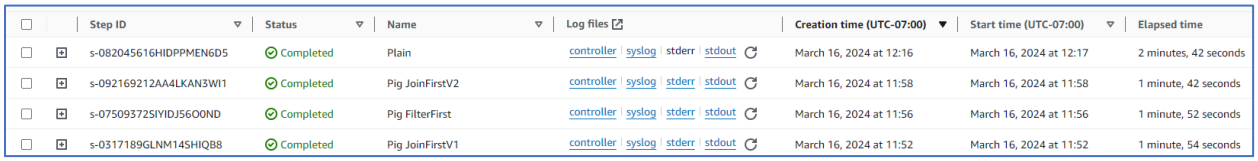
```
50.67124150519758
```

JoinFirst V2

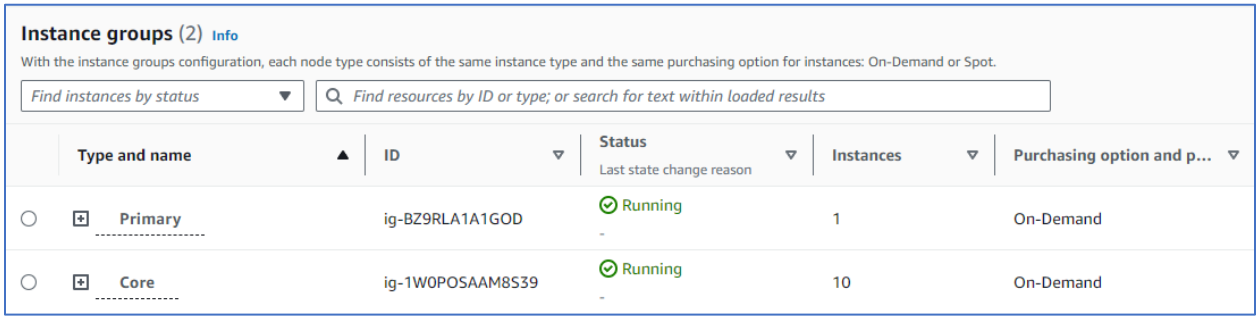
```
50.67124150519758
```

Performance Comparison

Proof of AWS EMR execution



Step ID	Status	Name	Log files	Creation time (UTC-07:00)	Start time (UTC-07:00)	Elapsed time
s-082045616HIDPPMEN6D5	Completed	Plain	controller | syslog | stderr | stdout	March 16, 2024 at 12:16	March 16, 2024 at 12:17	2 minutes, 42 seconds
s-092169212AA4LKAN3WI1	Completed	Pig JoinFirstV2	controller | syslog | stderr | stdout	March 16, 2024 at 11:58	March 16, 2024 at 11:58	1 minute, 42 seconds
s-07509372SIYIDJ56O0ND	Completed	Pig FilterFirst	controller | syslog | stderr | stdout	March 16, 2024 at 11:56	March 16, 2024 at 11:56	1 minute, 52 seconds
s-0317189GLNM14SHIQB8	Completed	Pig JoinFirstV1	controller | syslog | stderr | stdout	March 16, 2024 at 11:52	March 16, 2024 at 11:52	1 minute, 54 seconds



Instance groups (2) Info

With the instance groups configuration, each node type consists of the same instance type and the same purchasing option for instances: On-Demand or Spot.

Find instances by status | Find resources by ID or type; or search for text within loaded results

Type and name	ID	Status (Last state change reason)	Instances	Purchasing option and p...
Primary	ig-BZ9RLA1A1GOD	Running	1	On-Demand
Core	ig-1W0POSAAM8S39	Running	10	On-Demand

Runtime

Name	Start Time	End Time	Execution Time (seconds)
Plain	3/16/2024 19:17:41	3/16/2024 19:19:50	129
Filter First	3/16/2024 18:56:29	3/16/2024 18:58:00	91
Join First V1	3/16/2024 18:52:56	3/16/2024 18:54:26	90
Join First V2	3/16/2024 18:58:58	3/16/2024 19:00:24	86

Plain Java vs Pig

The actual runtime difference between Pig and Plain Java program meets my expectations due to the following reasons:

- The nature of Pig allows for more Abstraction and under the hood optimization, which helps tremendously when it comes to data processing.
- My Plain Java approach is not as optimized as Pig's approach of handling data.
- Pig has PigOptimizer which can rewrite the scripts into more optimized execution plan using safe operations.

Difference between Pig Scripts

Filter First vs Join First

The performance difference between these two approaches is dependent on the nature of the dataset. In this case, we see that the Join First approach being faster. This can potentially be explained by:

- Data produced after join is significantly smaller and easier to perform as a filter.
- The given date range and other conditions covers a large subset of data. Thus, the output of filtering still gives a large data set to be joined and further filtered.

Two Join First approaches

The difference between these two approaches seems to be more obvious, as the additional filter condition that requires checking both legs' flight date requires more execution time, thus resulting in different run time results.