PROJECT TITLE: FLIGHT DELAY ANALYSIS USING BIG DATA PROCESSING WITH MAPREDUCE

NAME: Tanushri Vijayakumar
COURSE NO: DS644
SECTION NO: 852
UCID: 31698861
EMAIL: tv233@njit.edu

DATASET DESCRIPTION

Dataset Name: Flight Delay and Cancellation Dataset (2019-2023)
Dataset Source:
https://www.kaggle.com/datasets/patrickzel/flight-delay-and-cancellation-dataset-2019-2023?resource=download&select=flights_sample_3m.csv

1. MapReduce Code

1.1. Code Explanation

1.1.1. OnTimePerformance.java

- Mapper class: Parses flight data, extracts AIRLINE_CODE and computes total delays from carrier, weather, NAS, security, and late aircraft delays. Emits (Airline Code, On-Time Flag 1/0) based on whether total delay less than or equal to 5 minutes.
- Reducer Class: Aggregates the total number of flights and the number of on-time flights for each airline. Computes the on-time probability rate.
- Driver Class: Configures the job setup, specifies input/output formats, mapper, reducer, and runs the MapReduce job.

1.1.2. TaxiTimeAnalysis.java

- Mapper Class: Parses flight data, extracts ORIGIN airport with TAXI_OUT time and DEST airport with TAXI_IN time, emitting (Airport, Taxi Time) pairs.
- Reducer Class: Aggregates the taxi times and number of flights per airport, computes average taxi time.
- Driver Class: Sets up the configuration, mapper, reducer, and executes the TaxiTime Analysis job.

1.2. Full Java Code

1.2.1. OnTimePerformance.java

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
```

```
// Main class to perform on-time flight performance analysis
public class OnTimePerformance {

    // Mapper class: processes each line and emits (airline_code, on_time_flag)
    public static class OnTimeMapper extends Mapper<LongWritable, Text, Text,
IntWritable> {

        private boolean headerSkipped = false; // To make sure we skip the first
header line
        private Text airlineCode = new Text();  // Key: airline code
        private IntWritable onTimeFlag = new IntWritable(); // Value: 1 if on-time,
0 otherwise

        @Override
        public void map(LongWritable key, Text value, Context context)
                throws IOException, InterruptedException {
            String line = value.toString();

            // Skip the first line which is the header
            if (!headerSkipped) {
                headerSkipped = true;
                return;
            }

            // Split the CSV line into fields
            String[] fields = line.split(",", -1); // -1 to include empty fields also

            // Ignore lines with insufficient columns
            if (fields.length < 13) return;

            try {
                // Extract airline code (assuming it's in column 2)
                String carrier = fields[1].trim();
                if (carrier.isEmpty()) return; // Skip if carrier code is missing

                // Read delay columns safely (with fallback if missing or invalid)
                float delayCarrier = parseFloatSafe(fields[7]);
                float delayWeather = parseFloatSafe(fields[8]);
                float delayNAS = parseFloatSafe(fields[9]);
                float delaySecurity = parseFloatSafe(fields[10]);
                float delayLateAircraft = parseFloatSafe(fields[11]);

                // Calculate total delay
                float totalDelay = delayCarrier + delayWeather + delayNAS +
delaySecurity + delayLateAircraft;

                // Rule: Flight is "on-time" if total delay is 5 minutes or less
                int onTime = (totalDelay <= 5.0f) ? 1 : 0;

                // Emit (airlineCode, onTimeFlag)
                airlineCode.set(carrier);
                onTimeFlag.set(onTime);

                context.write(airlineCode, onTimeFlag);

            } catch (Exception e) {
                // If parsing fails, skip that record
            }
        }

        // Helper function to safely parse floats
        private float parseFloatSafe(String s) {
            if (s == null || s.trim().isEmpty()) return 0.0f;
            try {
                return Float.parseFloat(s.trim());
            } catch (NumberFormatException e) {
                return 0.0f;
            }
        }
    }

    // Reducer class: aggregates results for each airline
    public static class OnTimeReducer extends Reducer<Text, IntWritable, Text,
Text> {

        @Override
        public void reduce(Text key, Iterable<IntWritable> values, Context context)
                throws IOException, InterruptedException {

            int totalFlights = 0;   // Total number of flights for this airline
            int onTimeFlights = 0;  // Number of on-time flights

            // Sum up total flights and on-time flights
            for (IntWritable v : values) {
                totalFlights++;
                onTimeFlights += v.get(); // Add 1 if on-time, 0 if delayed
            }
            // Calculate the on-time rate in percentage
            float onTimeRate = (totalFlights == 0) ? 0 : (onTimeFlights * 100.0f) /
totalFlights;

            // Format the output nicely
            String result = String.format("TotalFlights=%d, OnTimeFlights=%d,
OnTimeRate=%.2f%%",
                                          totalFlights, onTimeFlights, onTimeRate);

            // Emit (airlineCode, formatted result)
            context.write(key, new Text(result));
        }
    }

    // Main function to configure and run the job
    public static void main(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.println("Usage: OnTimePerformance <input path> <output
path>");
            System.exit(-1); // Exit if input and output paths are not provided
        }

        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "On-Time Performance Analysis"); // Job
name

        job.setJarByClass(OnTimePerformance.class);
        job.setMapperClass(OnTimeMapper.class);
        job.setReducerClass(OnTimeReducer.class);

        // Set output key and value types
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        // Set input and output file paths
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        // Wait for the job to finish and exit appropriately
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

1.2.2. TaxiTimeAnalysis.java

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

// Main class to perform average taxi time analysis at airports
public class TaxiTimeAnalysis {

  // Mapper class: emits (airport_code, taxi_time) for both taxi out and taxi in
times
  public static class TaxiMapper extends Mapper<LongWritable, Text, Text,
FloatWritable> {

    private boolean headerSkipped = false; // To ensure we skip the header line
    private Text airport = new Text();      // Key: airport code (origin or
destination)
    private FloatWritable taxiTime = new FloatWritable(); // Value: taxi time

    @Override
    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {
      String line = value.toString();

      // Skip the header line
      if (!headerSkipped) {
        headerSkipped = true;
        return;
      }

      // Split the CSV line into fields
      String[] fields = line.split(",", -1); // -1 to include empty fields as empty
strings

      // Ignore invalid lines with fewer than 13 columns
      if (fields.length < 13) return;

      try {
        // Extract important fields: origin, destination, taxi out, and taxi in
        String originAirport = fields[2].trim();  // ORIGIN column
        String destAirport = fields[3].trim();    // DEST column
```

```
                String taxiOutStr = fields[4].trim();     // TAXI_OUT column
                String taxiInStr = fields[5].trim();      // TAXI_IN column

                // Emit (origin airport, taxi out time) if data is valid
                if (!originAirport.isEmpty() && !taxiOutStr.isEmpty()) {
                    float taxiOut = Float.parseFloat(taxiOutStr);
                    airport.set(originAirport);
                    taxiTime.set(taxiOut);
                    context.write(airport, taxiTime);
                }

                // Emit (destination airport, taxi in time) if data is valid
                if (!destAirport.isEmpty() && !taxiInStr.isEmpty()) {
                    float taxiIn = Float.parseFloat(taxiInStr);
                    airport.set(destAirport);
                    taxiTime.set(taxiIn);
                    context.write(airport, taxiTime);
                }

            } catch (Exception e) {
                // If any parsing error happens, skip the current row
            }
        }
    }

    // Reducer class: calculates the average taxi time for each airport
    public static class TaxiReducer extends Reducer<Text, FloatWritable, Text,
FloatWritable> {

        @Override
        public void reduce(Text key, Iterable<FloatWritable> values, Context
context)
                throws IOException, InterruptedException {

            float sum = 0;   // Sum of all taxi times
            int count = 0;   // Count of taxi records

            // Aggregate sum and count
            for (FloatWritable v : values) {
                sum += v.get();
                count++;
            }

            // Only write output if there were valid records
            if (count > 0) {
                float avgTaxiTime = sum / count; // Compute average
                context.write(key, new FloatWritable(avgTaxiTime)); // Emit
(airport_code, avgTaxiTime)
            }
        }
    }

    // Main method to configure and start the Hadoop job
    public static void main(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.println("Usage: TaxiTimeAnalysis <input path> <output
path>");
            System.exit(-1); // Exit if incorrect number of arguments provided
        }

        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Taxi Time Analysis"); // Set job name

        job.setJarByClass(TaxiTimeAnalysis.class);
        job.setMapperClass(TaxiMapper.class);
        job.setReducerClass(TaxiReducer.class);

        // Define output types
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(FloatWritable.class);

        // Set input and output file paths
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        // Submit the job and exit based on success/failure
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

1.3. Compilation and execution commands

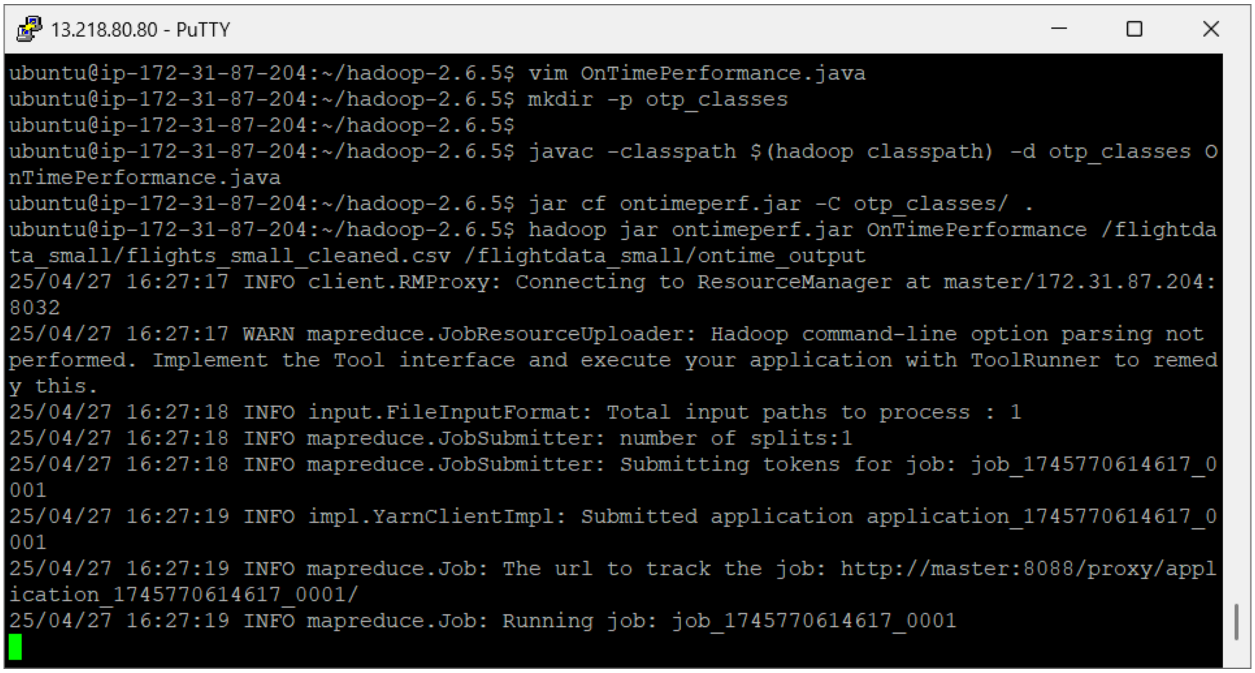
```
#COMPILATION:
mkdir -p otp_classes
javac -classpath $(hadoop classpath) -d otp_classes OnTimePerformance.java
jar cf ontimeperf.jar -C otp_classes/ .
```

```
mkdir -p taxi_classes
javac -classpath $(hadoop classpath) -d taxi_classes TaxiTimeAnalysis.java
jar cf taxitime.jar -C taxi_classes/ .

#EXECUTION:
hadoop jar ontimeperf.jar OnTimePerformance
/flightdata_small/flights_small_cleaned.csv /flightdata_small/ontime_output

hadoop jar taxitime.jar TaxiTimeAnalysis /flightdata_small/flights_small_cleaned.csv
/flightdata_small/taxitime_output
```

2. Code Execution and Output Interpretation

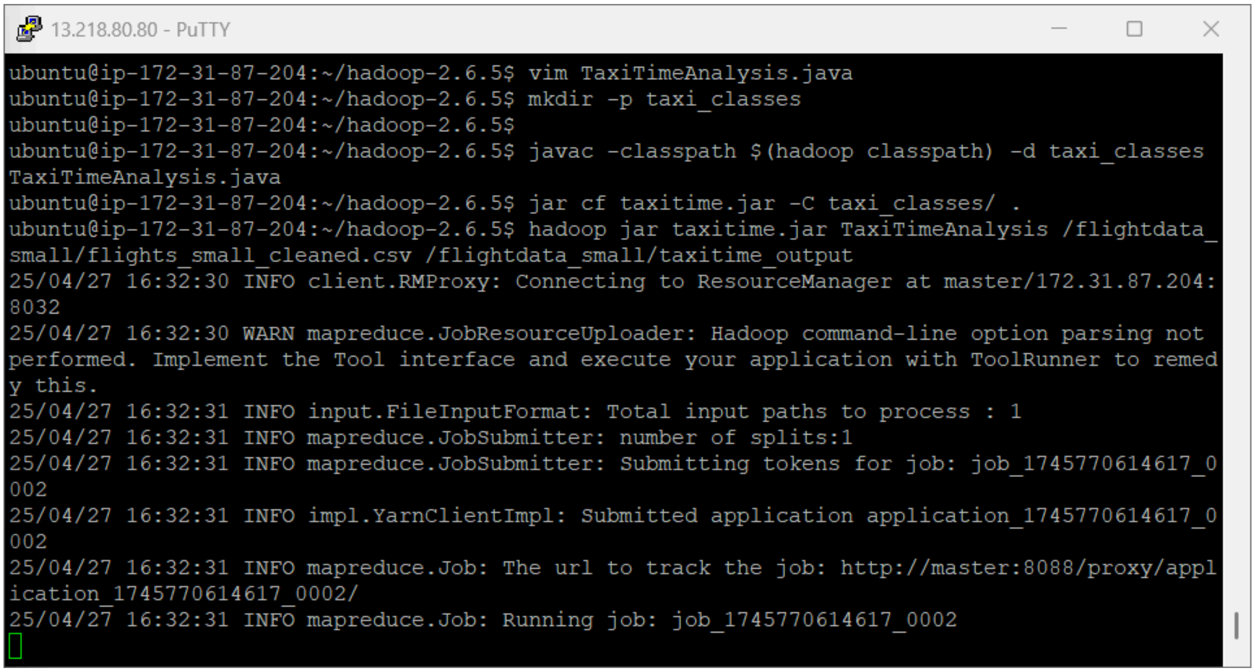
2.1. Screen of running your code command and its result

Execution for MapReduce program1:



```
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$ vim OnTimePerformance.java
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$ mkdir -p otp_classes
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$ javac -classpath $(hadoop classpath) -d otp_classes O
nTimePerformance.java
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$ jar cf ontimeperf.jar -C otp_classes/ .
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$ hadoop jar ontimeperf.jar OnTimePerformance /flightda
ta_small/flights_small_cleaned.csv /flightdata_small/ontime_output
25/04/27 16:27:17 INFO client.RMProxy: Connecting to ResourceManager at master/172.31.87.204:
8032
25/04/27 16:27:17 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not
performed. Implement the Tool interface and execute your application with ToolRunner to remed
y this.
25/04/27 16:27:18 INFO input.FileInputFormat: Total input paths to process : 1
25/04/27 16:27:18 INFO mapreduce.JobSubmitter: number of splits:1
25/04/27 16:27:18 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1745770614617_0
001
25/04/27 16:27:19 INFO impl.YarnClientImpl: Submitted application application_1745770614617_0
001
25/04/27 16:27:19 INFO mapreduce.Job: The url to track the job: http://master:8088/proxy/appl
ication_1745770614617_0001/
25/04/27 16:27:19 INFO mapreduce.Job: Running job: job_1745770614617_0001
```

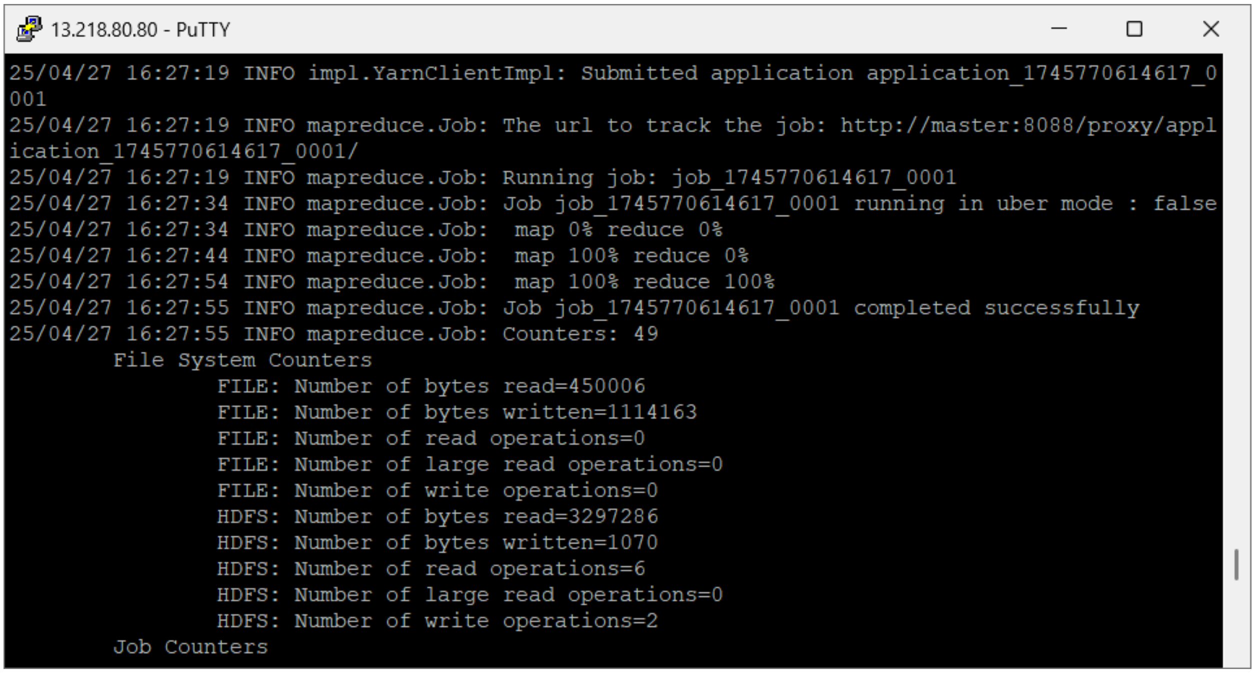
Execution for MapReduce program2:



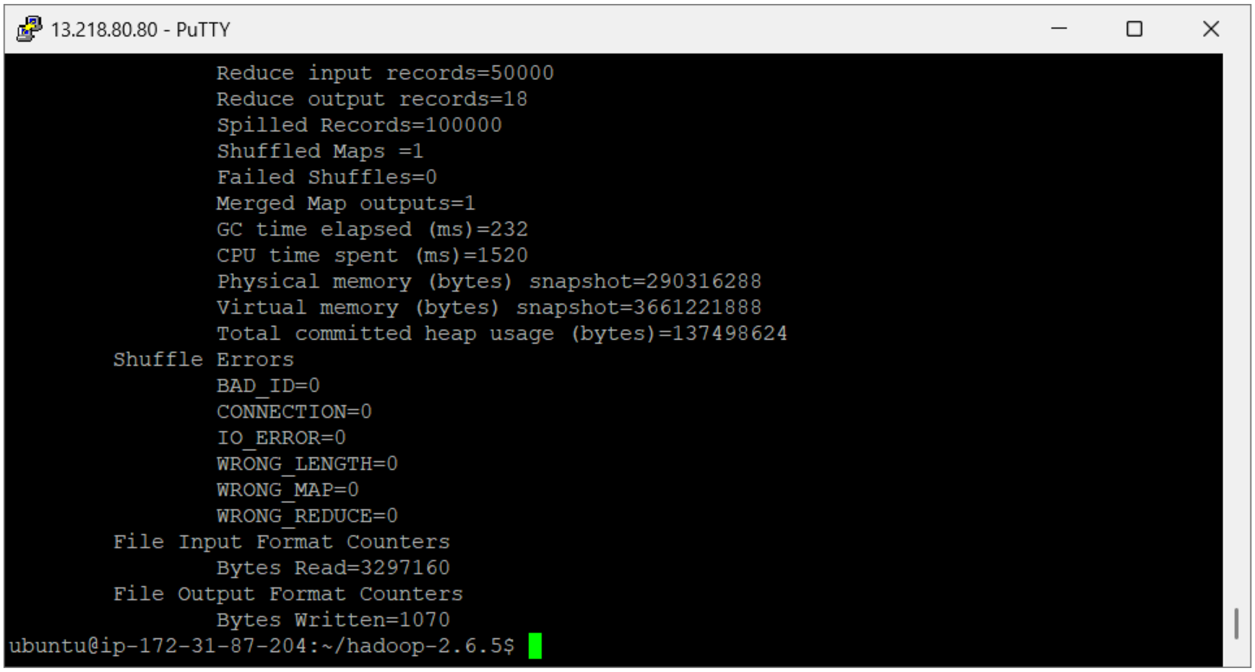
```
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$ vim TaxiTimeAnalysis.java
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$ mkdir -p taxi_classes
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$ javac -classpath $(hadoop classpath) -d taxi_classes
TaxiTimeAnalysis.java
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$ jar cf taxitime.jar -C taxi_classes/ .
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$ hadoop jar taxitime.jar TaxiTimeAnalysis /flightdata_
small/flights_small_cleaned.csv /flightdata_small/taxitime_output
25/04/27 16:32:30 INFO client.RMProxy: Connecting to ResourceManager at master/172.31.87.204:
8032
25/04/27 16:32:30 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not
performed. Implement the Tool interface and execute your application with ToolRunner to remed
y this.
25/04/27 16:32:31 INFO input.FileInputFormat: Total input paths to process : 1
25/04/27 16:32:31 INFO mapreduce.JobSubmitter: number of splits:1
25/04/27 16:32:31 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1745770614617_0
002
25/04/27 16:32:31 INFO impl.YarnClientImpl: Submitted application application_1745770614617_0
002
25/04/27 16:32:31 INFO mapreduce.Job: The url to track the job: http://master:8088/proxy/appl
ication_1745770614617_0002/
25/04/27 16:32:31 INFO mapreduce.Job: Running job: job_1745770614617_0002
```

2.2. Process of MapReduce without any error until the end

MapReduce running for program1:

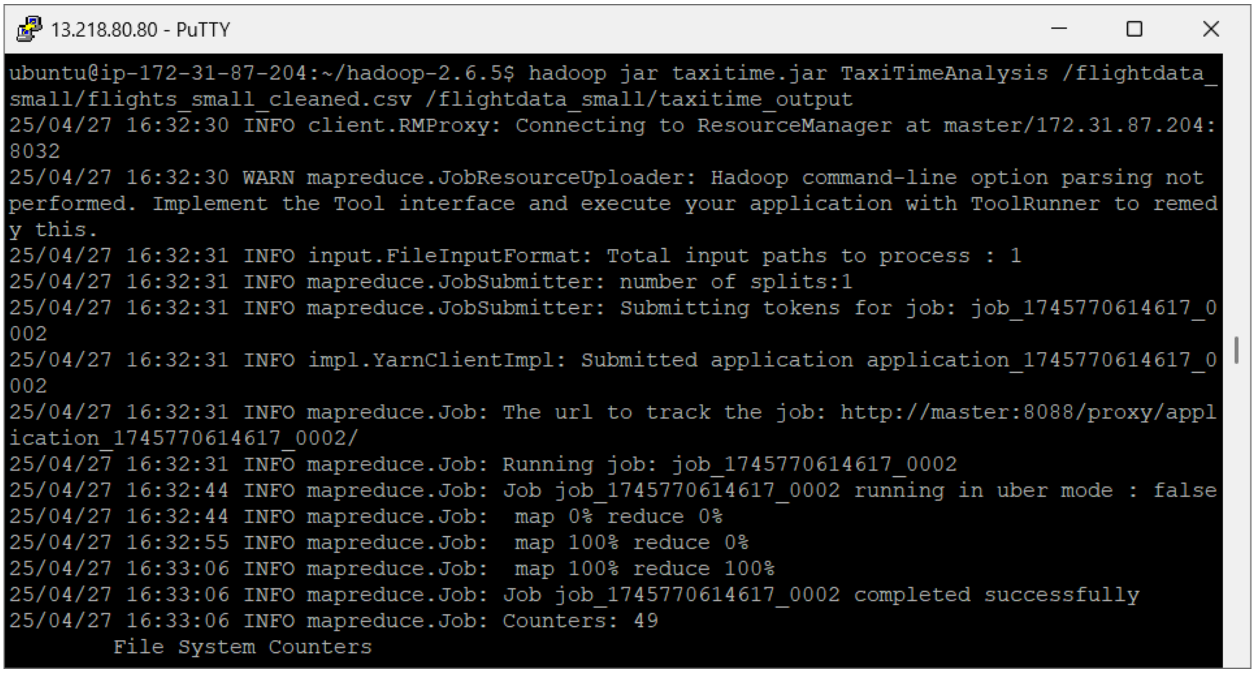


```
25/04/27 16:27:19 INFO impl.YarnClientImpl: Submitted application application_1745770614617_0
001
25/04/27 16:27:19 INFO mapreduce.Job: The url to track the job: http://master:8088/proxy/appl
ication_1745770614617_0001/
25/04/27 16:27:19 INFO mapreduce.Job: Running job: job_1745770614617_0001
25/04/27 16:27:34 INFO mapreduce.Job: Job job_1745770614617_0001 running in uber mode : false
25/04/27 16:27:34 INFO mapreduce.Job:  map 0% reduce 0%
25/04/27 16:27:44 INFO mapreduce.Job:  map 100% reduce 0%
25/04/27 16:27:54 INFO mapreduce.Job:  map 100% reduce 100%
25/04/27 16:27:55 INFO mapreduce.Job: Job job_1745770614617_0001 completed successfully
25/04/27 16:27:55 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=450006
                FILE: Number of bytes written=1114163
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=3297286
                HDFS: Number of bytes written=1070
                HDFS: Number of read operations=6
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
```

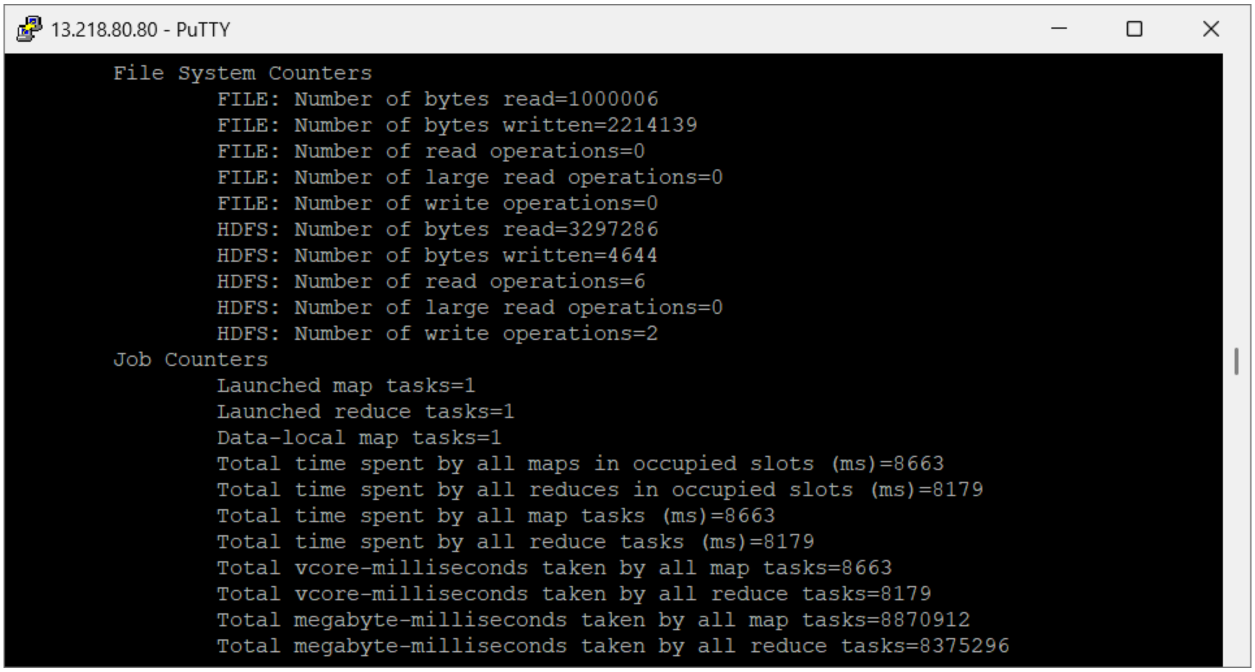


```
                Reduce input records=50000
                Reduce output records=18
                Spilled Records=100000
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=232
                CPU time spent (ms)=1520
                Physical memory (bytes) snapshot=290316288
                Virtual memory (bytes) snapshot=3661221888
                Total committed heap usage (bytes)=137498624
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=3297160
        File Output Format Counters
                Bytes Written=1070
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$
```

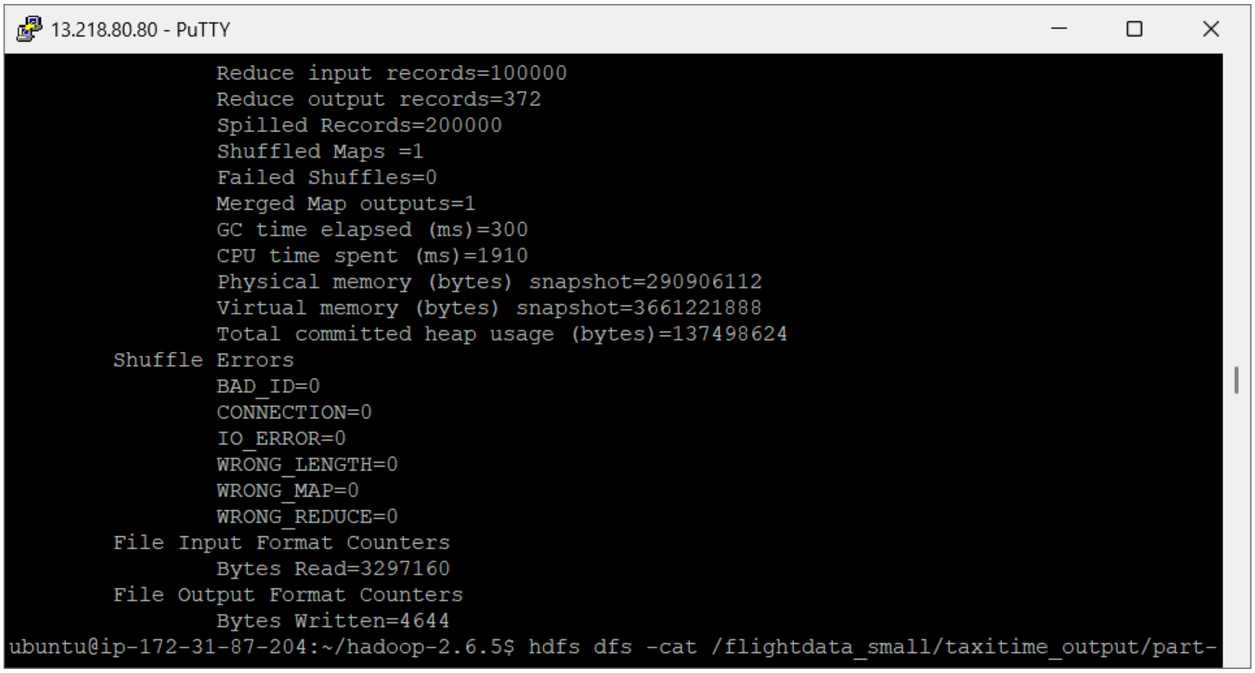
MapReduce running for program2:



```
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$ hadoop jar taxitime.jar TaxiTimeAnalysis /flightdata_
small/flights_small_cleaned.csv /flightdata_small/taxitime_output
25/04/27 16:32:30 INFO client.RMProxy: Connecting to ResourceManager at master/172.31.87.204:
8032
25/04/27 16:32:30 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not
performed. Implement the Tool interface and execute your application with ToolRunner to remed
y this.
25/04/27 16:32:31 INFO input.FileInputFormat: Total input paths to process : 1
25/04/27 16:32:31 INFO mapreduce.JobSubmitter: number of splits:1
25/04/27 16:32:31 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1745770614617_0
002
25/04/27 16:32:31 INFO impl.YarnClientImpl: Submitted application application_1745770614617_0
002
25/04/27 16:32:31 INFO mapreduce.Job: The url to track the job: http://master:8088/proxy/appl
ication_1745770614617_0002/
25/04/27 16:32:31 INFO mapreduce.Job: Running job: job_1745770614617_0002
25/04/27 16:32:44 INFO mapreduce.Job: Job job_1745770614617_0002 running in uber mode : false
25/04/27 16:32:44 INFO mapreduce.Job:  map 0% reduce 0%
25/04/27 16:32:55 INFO mapreduce.Job:  map 100% reduce 0%
25/04/27 16:33:06 INFO mapreduce.Job:  map 100% reduce 100%
25/04/27 16:33:06 INFO mapreduce.Job: Job job_1745770614617_0002 completed successfully
25/04/27 16:33:06 INFO mapreduce.Job: Counters: 49
        File System Counters
```



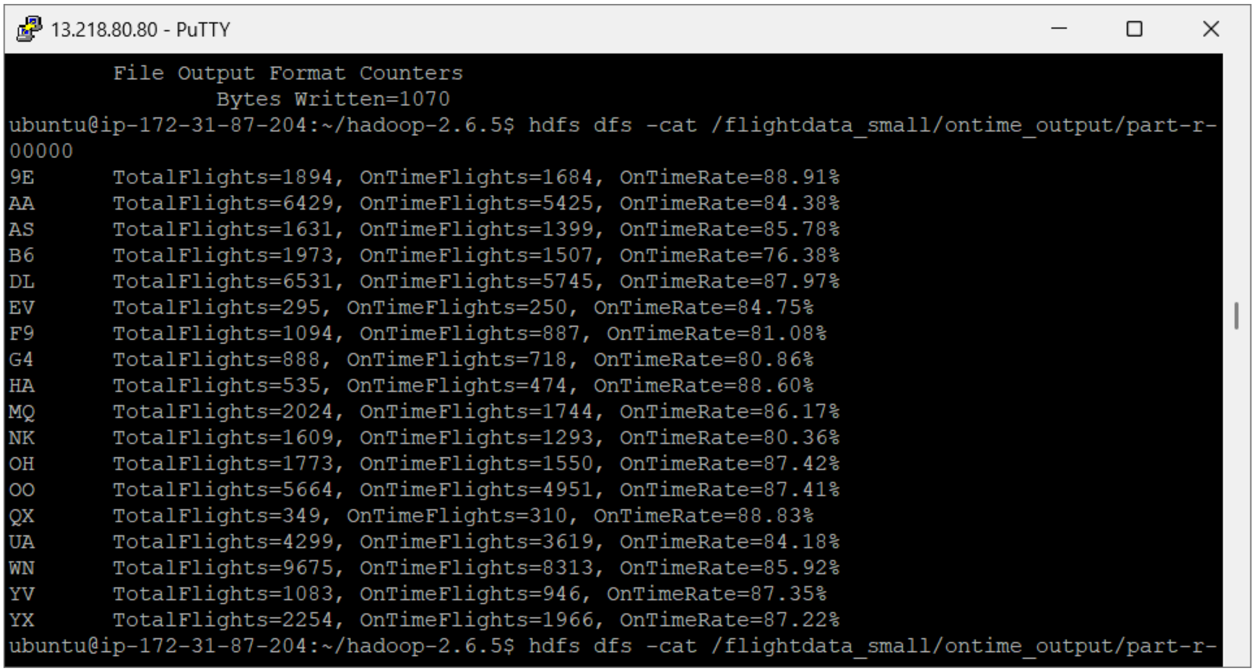
```
        File System Counters
                FILE: Number of bytes read=1000006
                FILE: Number of bytes written=2214139
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=3297286
                HDFS: Number of bytes written=4644
                HDFS: Number of read operations=6
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=1
                Launched reduce tasks=1
                Data-local map tasks=1
                Total time spent by all maps in occupied slots (ms)=8663
                Total time spent by all reduces in occupied slots (ms)=8179
                Total time spent by all map tasks (ms)=8663
                Total time spent by all reduce tasks (ms)=8179
                Total vcore-milliseconds taken by all map tasks=8663
                Total vcore-milliseconds taken by all reduce tasks=8179
                Total megabyte-milliseconds taken by all map tasks=8870912
                Total megabyte-milliseconds taken by all reduce tasks=8375296
```



```
                Reduce input records=100000
                Reduce output records=372
                Spilled Records=200000
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=300
                CPU time spent (ms)=1910
                Physical memory (bytes) snapshot=290906112
                Virtual memory (bytes) snapshot=3661221888
                Total committed heap usage (bytes)=137498624
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=3297160
        File Output Format Counters
                Bytes Written=4644
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$ hdfs dfs -cat /flightdata_small/taxitime_output/part-
```

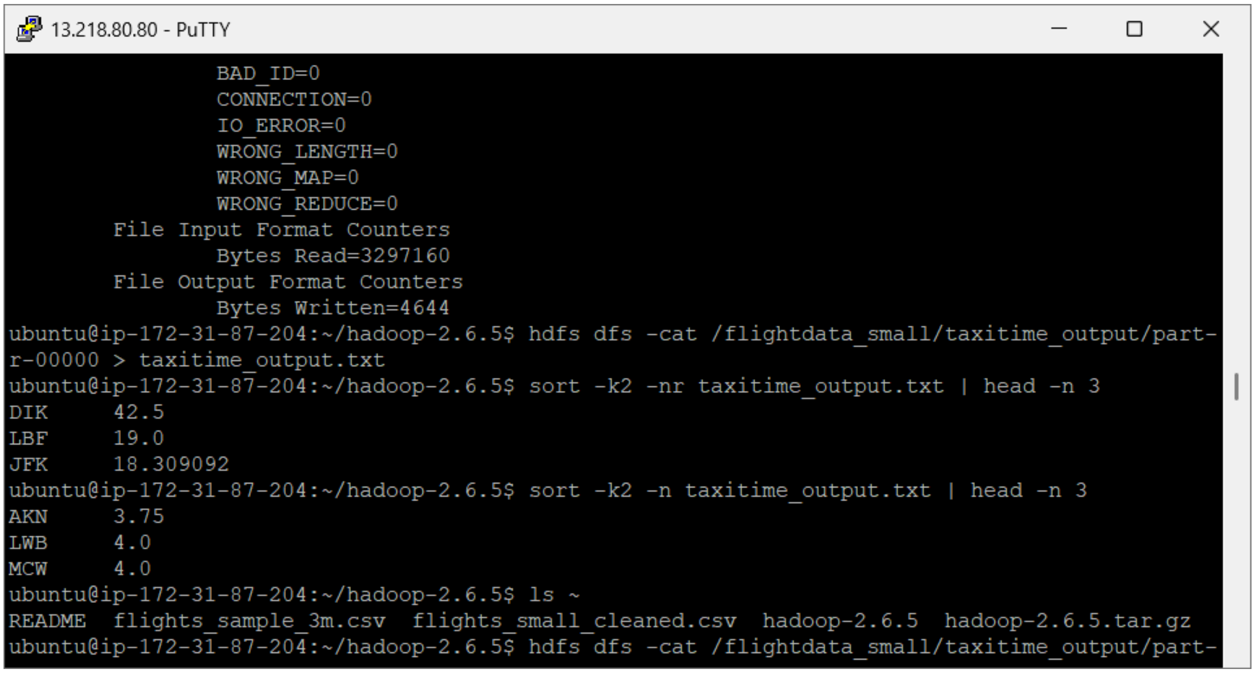
2.3. Screen of getting the output command and its result

OUTPUT FOR PROGRAM 1:

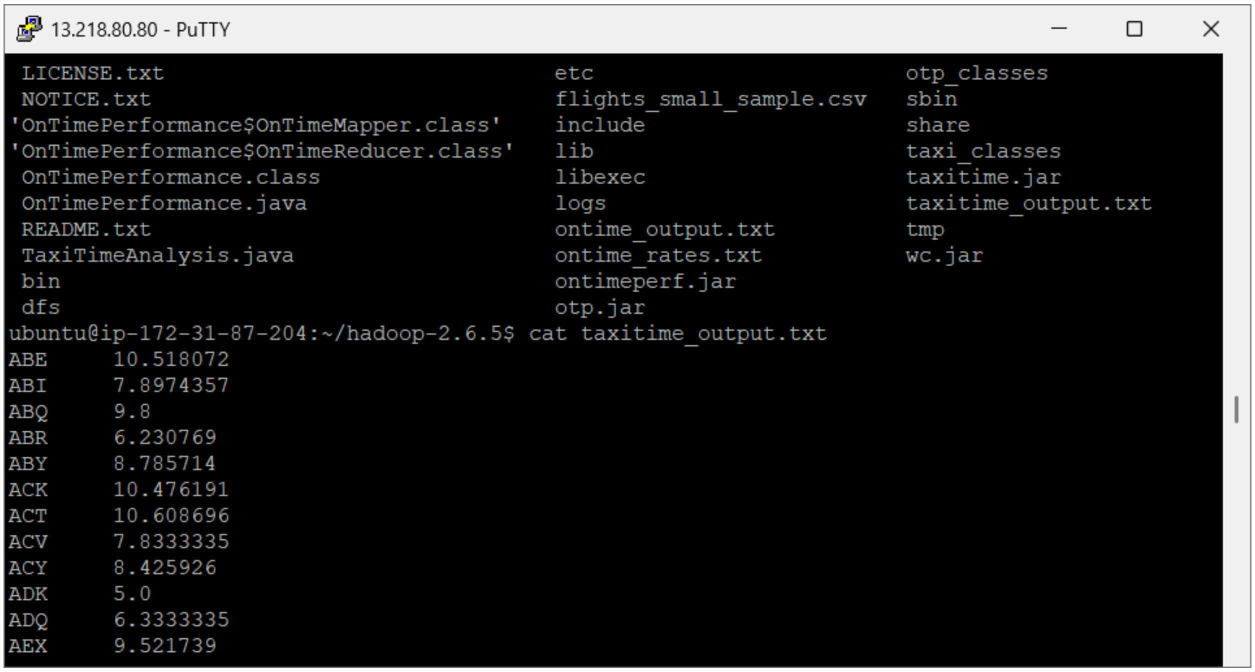


```
        File Output Format Counters
                Bytes Written=1070
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$ hdfs dfs -cat /flightdata_small/ontime_output/part-r-
00000
9E      TotalFlights=1894, OnTimeFlights=1684, OnTimeRate=88.91%
AA      TotalFlights=6429, OnTimeFlights=5425, OnTimeRate=84.38%
AS      TotalFlights=1631, OnTimeFlights=1399, OnTimeRate=85.78%
B6      TotalFlights=1973, OnTimeFlights=1507, OnTimeRate=76.38%
DL      TotalFlights=6531, OnTimeFlights=5745, OnTimeRate=87.97%
EV      TotalFlights=295, OnTimeFlights=250, OnTimeRate=84.75%
F9      TotalFlights=1094, OnTimeFlights=887, OnTimeRate=81.08%
G4      TotalFlights=888, OnTimeFlights=718, OnTimeRate=80.86%
HA      TotalFlights=535, OnTimeFlights=474, OnTimeRate=88.60%
MQ      TotalFlights=2024, OnTimeFlights=1744, OnTimeRate=86.17%
NK      TotalFlights=1609, OnTimeFlights=1293, OnTimeRate=80.36%
OH      TotalFlights=1773, OnTimeFlights=1550, OnTimeRate=87.42%
OO      TotalFlights=5664, OnTimeFlights=4951, OnTimeRate=87.41%
QX      TotalFlights=349, OnTimeFlights=310, OnTimeRate=88.83%
UA      TotalFlights=4299, OnTimeFlights=3619, OnTimeRate=84.18%
WN      TotalFlights=9675, OnTimeFlights=8313, OnTimeRate=85.92%
YV      TotalFlights=1083, OnTimeFlights=946, OnTimeRate=87.35%
YX      TotalFlights=2254, OnTimeFlights=1966, OnTimeRate=87.22%
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$ hdfs dfs -cat /flightdata_small/ontime_output/part-r-
```

OUTPUT FOR PROGRAM 2:



```
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=3297160
        File Output Format Counters
                Bytes Written=4644
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$ hdfs dfs -cat /flightdata_small/taxitime_output/part-
r-00000 > taxitime_output.txt
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$ sort -k2 -nr taxitime_output.txt | head -n 3
DIK     42.5
LBF     19.0
JFK     18.309092
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$ sort -k2 -n taxitime_output.txt | head -n 3
AKN     3.75
LWB     4.0
MCW     4.0
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$ ls ~
README  flights_sample_3m.csv  flights_small_cleaned.csv  hadoop-2.6.5  hadoop-2.6.5.tar.gz
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$ hdfs dfs -cat /flightdata_small/taxitime_output/part-
```



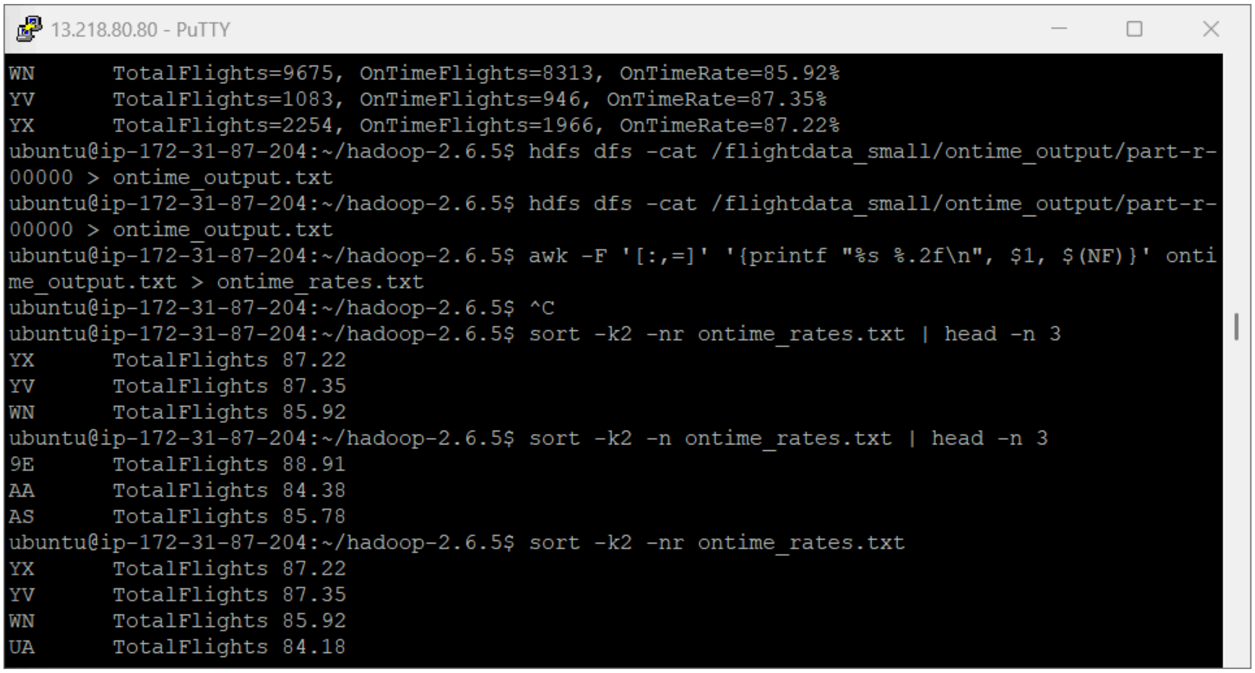
```
 LICENSE.txt                               etc                        otp_classes
 NOTICE.txt                                flights_small_sample.csv   sbin
'OnTimePerformance$OnTimeMapper.class'     include                    share
'OnTimePerformance$OnTimeReducer.class'    lib                        taxi_classes
 OnTimePerformance.class                   libexec                    taxitime.jar
 OnTimePerformance.java                    logs                       taxitime_output.txt
 README.txt                                ontime_output.txt          tmp
 TaxiTimeAnalysis.java                     ontime_rates.txt           wc.jar
 bin                                       ontimeperf.jar
 dfs                                       otp.jar
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$ cat taxitime_output.txt
ABE     10.518072
ABI     7.8974357
ABQ     9.8
ABR     6.230769
ABY     8.785714
ACK     10.476191
ACT     10.608696
ACV     7.8333335
ACY     8.425926
ADK     5.0
ADQ     6.3333335
AEX     9.521739
```

2.4. Output Interpretation

2.4.1. OnTimePerformance Output Interpretation:

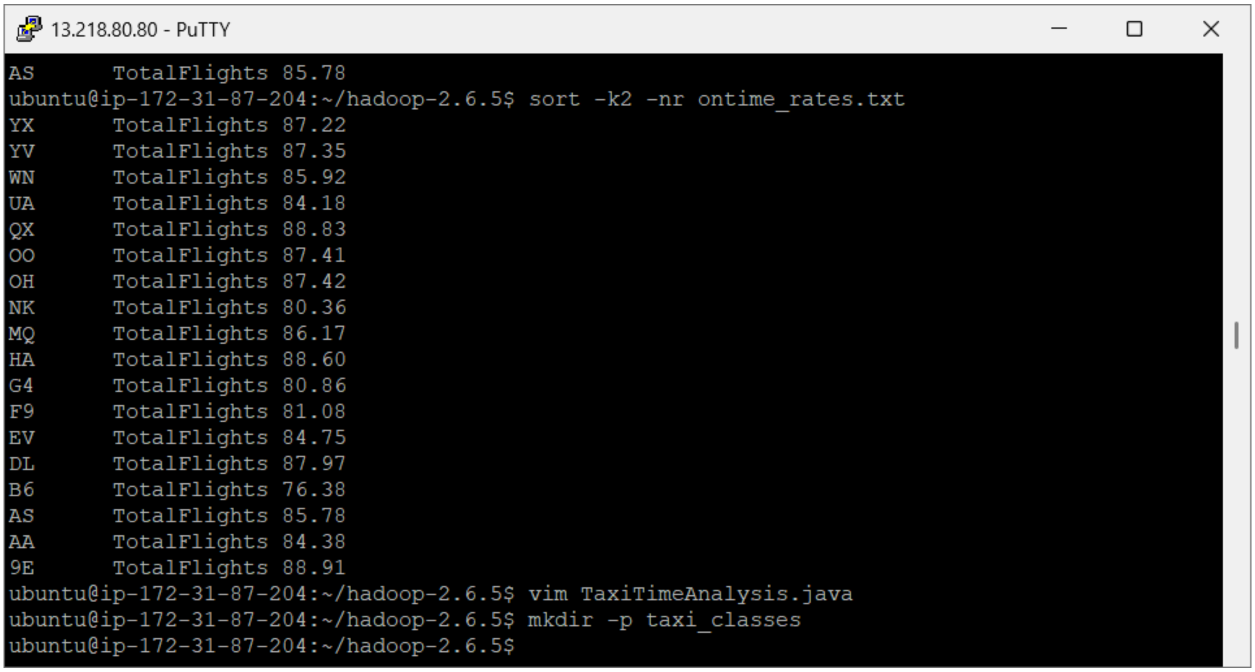
Airline codes with their total flights, number of on-time flights, and on-time probability (%) are displayed.

Top 3 airlines with highest On-Time Rate and bottom 3 airlines with lowest On-Time Rate:



```
WN      TotalFlights=9675, OnTimeFlights=8313, OnTimeRate=85.92%
YV      TotalFlights=1083, OnTimeFlights=946, OnTimeRate=87.35%
YX      TotalFlights=2254, OnTimeFlights=1966, OnTimeRate=87.22%
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$ hdfs dfs -cat /flightdata_small/ontime_output/part-r-
00000 > ontime_output.txt
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$ hdfs dfs -cat /flightdata_small/ontime_output/part-r-
00000 > ontime_output.txt
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$ awk -F '[:,=]' '{printf "%s %.2f\n", $1, $(NF)}' onti
me_output.txt > ontime_rates.txt
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$ ^C
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$ sort -k2 -nr ontime_rates.txt | head -n 3
YX      TotalFlights 87.22
YV      TotalFlights 87.35
WN      TotalFlights 85.92
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$ sort -k2 -n ontime_rates.txt | head -n 3
9E      TotalFlights 88.91
AA      TotalFlights 84.38
AS      TotalFlights 85.78
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$ sort -k2 -nr ontime_rates.txt
YX      TotalFlights 87.22
YV      TotalFlights 87.35
WN      TotalFlights 85.92
UA      TotalFlights 84.18
```

Full sorted list (not just Top 3):

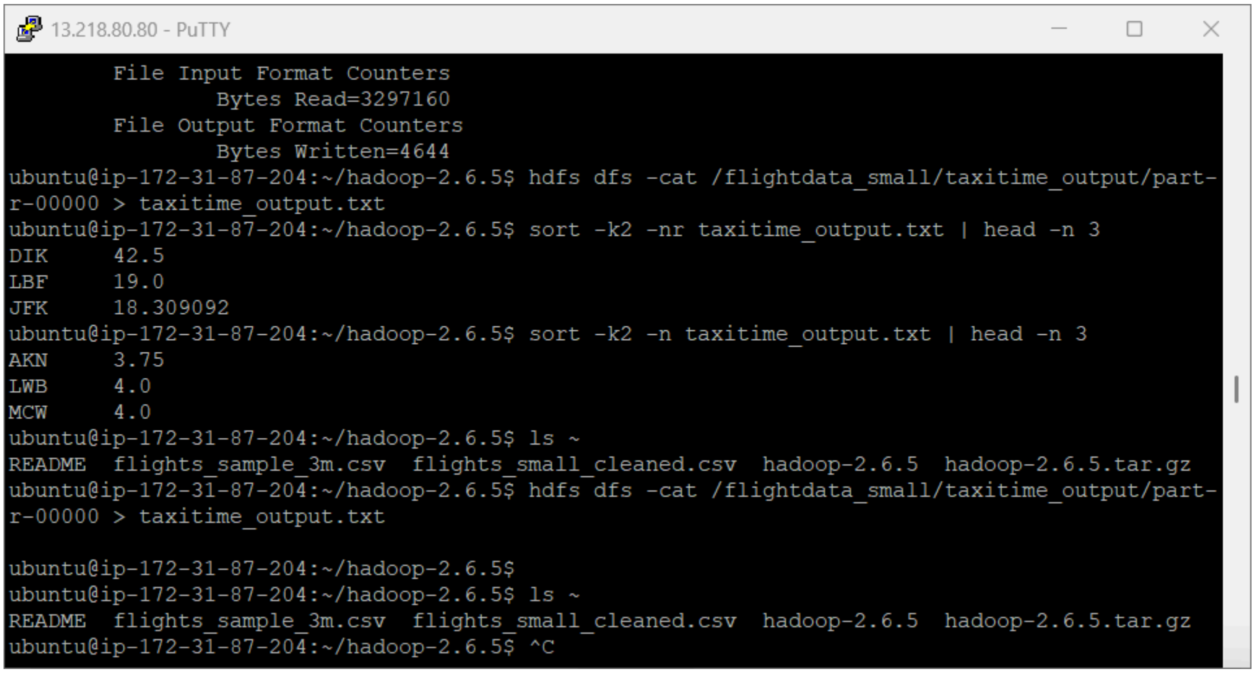


```
AS      TotalFlights 85.78
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$ sort -k2 -nr ontime_rates.txt
YX      TotalFlights 87.22
YV      TotalFlights 87.35
WN      TotalFlights 85.92
UA      TotalFlights 84.18
QX      TotalFlights 88.83
OO      TotalFlights 87.41
OH      TotalFlights 87.42
NK      TotalFlights 80.36
MQ      TotalFlights 86.17
HA      TotalFlights 88.60
G4      TotalFlights 80.86
F9      TotalFlights 81.08
EV      TotalFlights 84.75
DL      TotalFlights 87.97
B6      TotalFlights 76.38
AS      TotalFlights 85.78
AA      TotalFlights 84.38
9E      TotalFlights 88.91
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$ vim TaxiTimeAnalysis.java
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$ mkdir -p taxi_classes
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$
```

2.4.2. TaxiTimeAnalysis Output Interpretation:

Airports with average taxi times are displayed.

Top 3 airports with longest average taxi time and top 3 airports with shortest average taxi time:



```
        File Input Format Counters
                Bytes Read=3297160
        File Output Format Counters
                Bytes Written=4644
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$ hdfs dfs -cat /flightdata_small/taxitime_output/part-
r-00000 > taxitime_output.txt
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$ sort -k2 -nr taxitime_output.txt | head -n 3
DIK     42.5
LBF     19.0
JFK     18.309092
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$ sort -k2 -n taxitime_output.txt | head -n 3
AKN     3.75
LWB     4.0
MCW     4.0
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$ ls ~
README  flights_sample_3m.csv  flights_small_cleaned.csv  hadoop-2.6.5  hadoop-2.6.5.tar.gz
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$ hdfs dfs -cat /flightdata_small/taxitime_output/part-
r-00000 > taxitime_output.txt

ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$ ls ~
README  flights_sample_3m.csv  flights_small_cleaned.csv  hadoop-2.6.5  hadoop-2.6.5.tar.gz
ubuntu@ip-172-31-87-204:~/hadoop-2.6.5$ ^C
```

3. Challenges and troubleshooting

3.1. Challenges Faced

- When working with a large dataset, such as the 3 million flight records available in the original dataset, it is not always practical to process the full data using a small Hadoop cluster. Large datasets require significant memory, storage, and computing power, which small EC2 instances cannot handle easily.
- Many important columns - such as delay fields (DELAY_DUE_CARRIER, DELAY_DUE_WEATHER, etc.) and cancellation fields (CANCELLATION_CODE) - had missing values, also known as nulls. These missing values can create problems during MapReduce processing, such as errors or incorrect calculations.
- Before loading the dataset into the Hadoop Distributed File System (HDFS), it was essential to clean the data carefully. Without proper cleaning, dirty or incomplete data could cause Hadoop jobs to fail or produce wrong results.

3.2. Troubleshooting Steps

- Sampling was performed - means selecting a small random portion of the full dataset - in my case, about 50,000 rows - so that the data becomes much smaller and easier to process. This way, we could run our MapReduce jobs successfully without overloading the cluster or causing performance issues.
- The missing delay fields were filled with 0, assuming that a missing delay means no delay occurred. For cancellation fields, only considered rows where flights were actually cancelled. Handling missing values correctly ensured that the data was clean and ready for reliable analysis.
- Ensured Hadoop cluster was properly configured (namenode, datanode, YARN started). Also did data cleaning process like:
    - Keeping only the important columns needed for the project (like airline code, taxi times, delay fields).
    - Filling missing delay fields with 0 to avoid processing errors.
    - Removing rows where critical fields (like taxi times) were completely missing.
    - Checking for and removing any invalid or corrupted entries.

3.3. Performance observations

- After reducing the original large dataset to a small random sample of approximately 50,000 rows, the MapReduce jobs executed much faster and without overloading the Hadoop cluster.
- However, minor slowness was observed when attempting to rerun jobs without clearing the previous output directories.
- This was because Hadoop does not allow writing into an already existing output folder. The issue was easily resolved by manually deleting the existing output directory from HDFS before starting a new job run.
- Overall, performance was smooth and efficient after proper data preparation and careful handling of outputs.

4. Summary and Key learnings

4.1. Project Reflection

- Learned setting-up Hadoop cluster on AWS EC2.

- Gained real-world experience handling large messy datasets.
- Understood MapReduce flow - Mapper, Shuffle, Reducer.
- Understood data preparation is very critical before Hadoop processing.

4.2. Real-world Application

- Airline performance analysis is directly useful for airport operations optimization.
- Similar MapReduce models can be applied for: Logistics performance, E-commerce order delivery analysis etc.

4.3. Future Improvements

- I would like to analyze a full 3M flight dataset with bigger AWS instances.
- I would also use Spark (faster in-memory computation) instead of Hadoop MapReduce.