

# Flight Data Analysis

## CS644 Project Report

**Harini Reddy Gade**  
**Harsha Vardhan Kancharla**  
**Nandini Reddy Palwayi**  
**Laxmiharshika Vegesna**

A project report in PDF that includes:

- a. A diagram that shows the structure of your Oozie workflow
- b. A detailed description of the algorithm you designed to solve each of the problems
- c. A performance measurement plot that compares the workflow execution time in response to an increasing number of VMs used for processing the entire data set (22 years) and an in-depth discussion on the observed performance comparison results
- d. A performance measurement plot that compares the workflow execution time in response to an increasing data size (from 1 year to 22 years) and an in-depth discussion on the observed performance comparison results

## Introduction

This project is about analyzing the On-time Performance data set (flight data set) from the period of October 1987 to April 2008. We will be extracting the useful data from the csv files obtained and applying our mapper and reducer function using oozie and hadoop to find the highest and lowest probability, respectively, for being on schedule, 3 airports with the longest and shortest average taxi time per flight (both in and out), respectively and the most common reason for flight cancellations. We will be using Java Its to run our project and an AWS VM to run hadoop and the oozie.

## Background

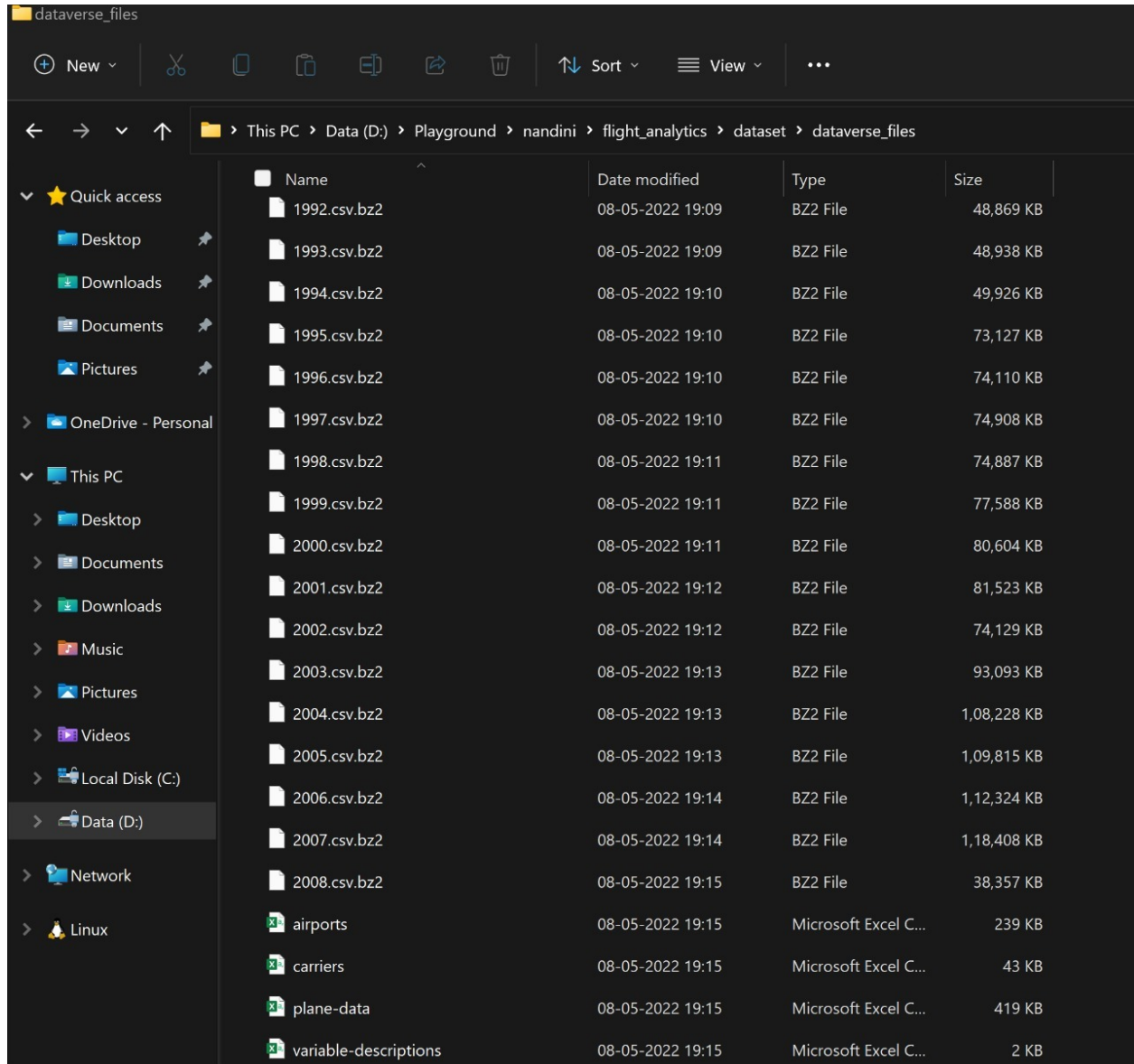
In this project, we will be focusing on finding some useful data from the dataset given and also measuring how the vertical scaling affects the performance of the system. We will also look into the time taken for the execution of our algorithm with the increase in the size of the dataset. Big data Hadoop is known for distributed computation, so therefore this project gives us very good insights into the crucial features of the Hadoop system.

**Vertical Scaling:** *Vertical scaling can essentially resize your server with no change to your code. It is the ability to increase the capacity of existing hardware or software by adding resources. Vertical scaling is limited by the fact that you can only get as big as the size of the server.*

## Dataset

We will be downloading the dataset from a Harvard website,

<https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/HG7NV7> . This is about 1.47Gb in size in compressed format. After extraction, each file would become around 200-500MB, and the data set is in CSV format.



The screenshot shows a Windows File Explorer window titled 'dataverse\_files'. The address bar indicates the path: 'This PC > Data (D:) > Playground > nandini > flight\_analytics > dataset > dataverse\_files'. The left sidebar shows the 'Data (D:)' drive selected. The main pane displays a list of files with columns for Name, Date modified, Type, and Size.

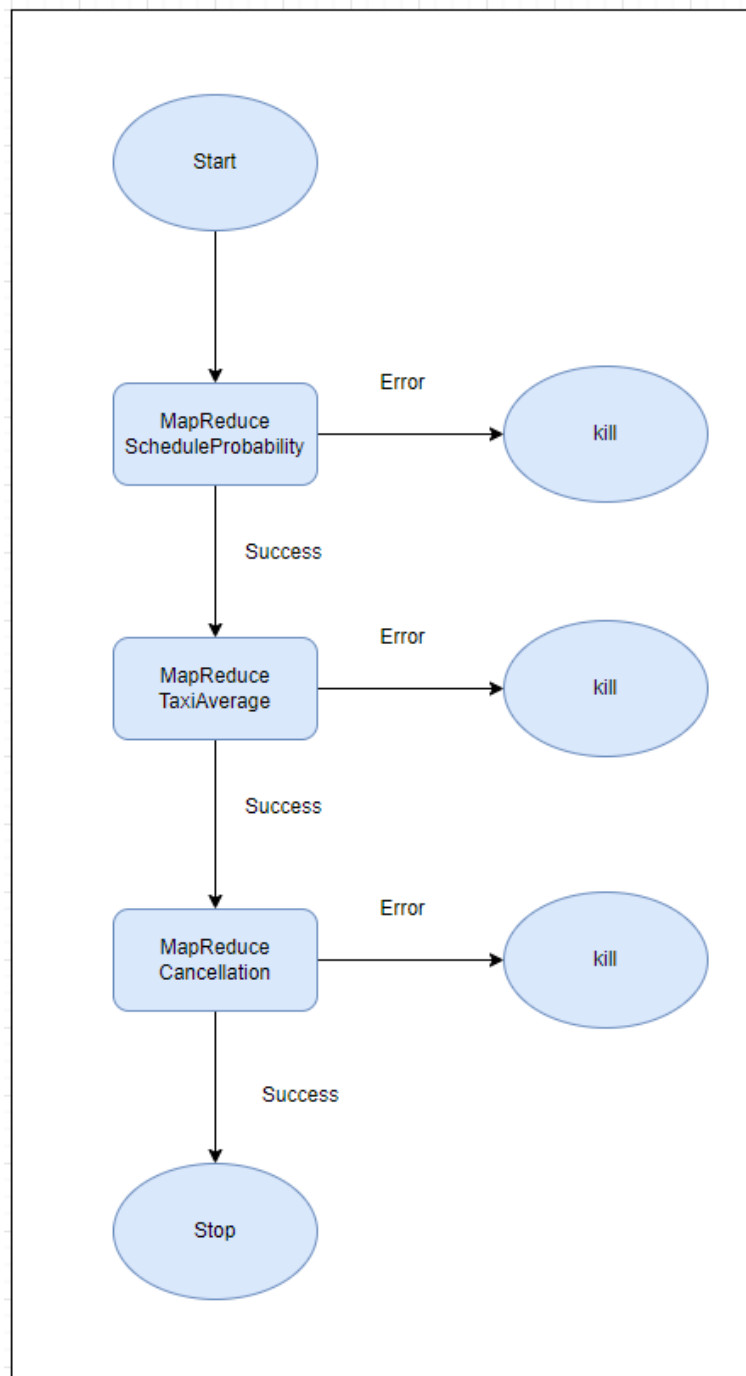
Name	Date modified	Type	Size
1992.csv.bz2	08-05-2022 19:09	BZ2 File	48,869 KB
1993.csv.bz2	08-05-2022 19:09	BZ2 File	48,938 KB
1994.csv.bz2	08-05-2022 19:10	BZ2 File	49,926 KB
1995.csv.bz2	08-05-2022 19:10	BZ2 File	73,127 KB
1996.csv.bz2	08-05-2022 19:10	BZ2 File	74,110 KB
1997.csv.bz2	08-05-2022 19:10	BZ2 File	74,908 KB
1998.csv.bz2	08-05-2022 19:11	BZ2 File	74,887 KB
1999.csv.bz2	08-05-2022 19:11	BZ2 File	77,588 KB
2000.csv.bz2	08-05-2022 19:11	BZ2 File	80,604 KB
2001.csv.bz2	08-05-2022 19:12	BZ2 File	81,523 KB
2002.csv.bz2	08-05-2022 19:12	BZ2 File	74,129 KB
2003.csv.bz2	08-05-2022 19:13	BZ2 File	93,093 KB
2004.csv.bz2	08-05-2022 19:13	BZ2 File	1,08,228 KB
2005.csv.bz2	08-05-2022 19:13	BZ2 File	1,09,815 KB
2006.csv.bz2	08-05-2022 19:14	BZ2 File	1,12,324 KB
2007.csv.bz2	08-05-2022 19:14	BZ2 File	1,18,408 KB
2008.csv.bz2	08-05-2022 19:15	BZ2 File	38,357 KB
airports	08-05-2022 19:15	Microsoft Excel C...	239 KB
carriers	08-05-2022 19:15	Microsoft Excel C...	43 KB
plane-data	08-05-2022 19:15	Microsoft Excel C...	419 KB
variable-descriptions	08-05-2022 19:15	Microsoft Excel C...	2 KB

We have about 29 columns in our dataset, from which we would be using around 5-7 columns only to obtain our results.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	
1	Year	Month	DayofMon	DayOfWee	DepTime	CRSDepTin	ArrTime	CRSArrTim	UniqueCar	FlightNum	TailNum	ActualElap	CRSElapset	AirTime	ArrDelay	DepDelay	Origin	Dest	Distance	TaxiIn	TaxiOut	Cancelled	C
2	2008	1	3	4	1343	1325	1451	1435	WN	588	N240WN	68	70	55	16	18	HOU	LIT	393	4	9	0	
3	2008	1	3	4	1125	1120	1247	1245	WN	1343	N523SW	82	85	71	2	5	HOU	MAF	441	3	8	0	
4	2008	1	3	4	2009	2015	2136	2140	WN	3841	N280WN	87	85	71	-4	-6	HOU	MAF	441	2	14	0	
5	2008	1	3	4	903	855	1203	1205	WN	3	N308SA	120	130	108	-2	8	HOU	MCO	848	5	7	0	
6	2008	1	3	4	1423	1400	1726	1710	WN	25	N462WN	123	130	107	16	23	HOU	MCO	848	6	10	0	
7	2008	1	3	4	2024	2020	2325	2325	WN	51	N483WN	121	125	101	0	4	HOU	MCO	848	13	7	0	
8	2008	1	3	4	1753	1745	2053	2050	WN	940	N493WN	120	125	107	3	8	HOU	MCO	848	6	7	0	
9	2008	1	3	4	622	620	935	930	WN	2621	N266WN	133	130	107	5	2	HOU	MCO	848	7	19	0	
10	2008	1	3	4	1944	1945	2210	2215	WN	389	N266WN	146	150	124	-5	-1	HOU	MDW	937	7	15	0	
11	2008	1	3	4	1453	1425	1716	1650	WN	519	N514SW	143	145	124	26	28	HOU	MDW	937	6	13	0	
12	2008	1	3	4	2030	2015	2251	2245	WN	894	N716SW	141	150	122	6	15	HOU	MDW	937	11	8	0	
13	2008	1	3	4	708	615	936	840	WN	969	N215WN	148	145	128	56	53	HOU	MDW	937	10	10	0	

Solution:

### Oozie Workflow Architecture



*A diagram that shows the structure of your Oozie workflow*

## Algorithms used to solve each problem

### **The three airlines with the best and lowest chances of being on time, respectively**

#### **Mapper Phase:**

1. Go through the input files line by line.
2. Because it's a comma-separated file (CSV), we split it into fields and put them all in an array.
3. Retrieve the values for the fields airlines(8), arrival delay(14), and departure delay(15) (15)
4. Now we find all the airlines (all airlines) and count the airlines (on-time airlines) with a delay of fewer than 10 minutes (departure and arrival delay).
5. 5. Write to context<airlines all, 1> and context<airlines on time, 1>

#### **Reducer Phase:**

1. Count the number of times the current key suffix is all (airlines all). This tells you the overall number of flights, which includes both delayed and on-time flights. We also check if this key is the current element, and if it isn't, we change it.
2. When the airline suffix isn't "all." We get airlines on time, for example. To calculate the chance of an airline being on time, we count them and divide "airlines on time" by "airlines all."
3. These probability values are entered into a tree map, and we use a modified comparator to sort the data by likelihood.
4. For the highest likelihood and the lowest probability, two tree maps are used. When the sorted output size exceeds 3, we extract the last and first values.
5. Add the output to the context.

### **Calculation of the longest and shortest average taxi times for three airports (In and Out)**

#### **Mapper Phase:**

1. Read input files line by line in the mapper phase.
2. Because it's a comma-separated file (CSV), we split it into fields and put them all in an array.
3. Retrieve the values for the variables origin(16), destination(17), taxiIn(19), and taxiOut(20) (20)
4. We verify that the taxiIn and taxiOut times are integers, and then we write to contextairportorigin, taxiIn> and contextairportdestination, taxiOut>, respectively.

#### **The phase of reduction:**

1. Understand the context. It will be given all of the values that correspond to a certain key.
2. We iterate over the context values, which are a list of airport taxiIn and taxiOut values.
3. Add together all of the values to get the total number of values.
4. Determine the average taxi by multiplying the sum of all values by the entire number of values.
5. These average values are entered into a tree map, and we utilize a modified comparator to sort the data by probability.
6. Create a class OutPair and implement an interface with airportname and average taxi time as instance variables. Comparable
7. In the compareTo method, sort by average cab time.

8..Two tree sets are used for airports with the longest taxi lines and another for airports with the shortest taxi lines

.9.Write the output in the context

### **Discover the most common reasons for flight cancellation.**

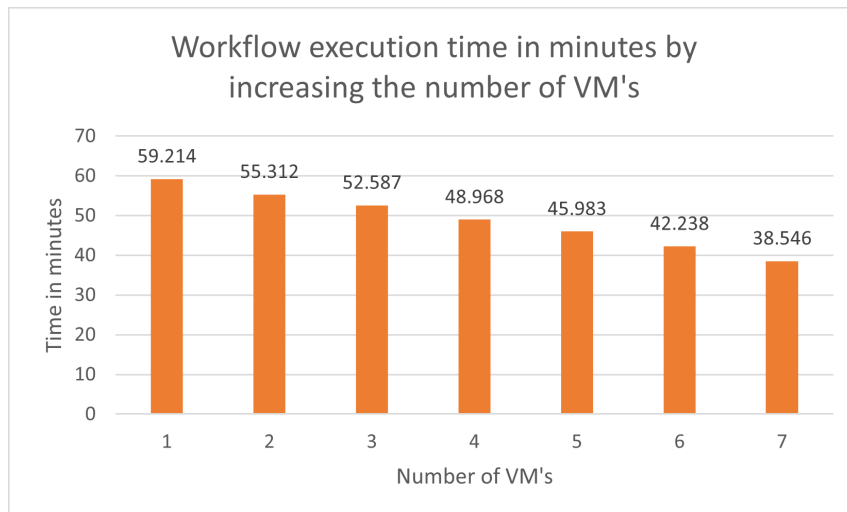
#### **Mapper Phase:**

1. Go through the input files line by line.
2. Because it's a comma-separated file (CSV), we split it into fields and put them all in an array.
3. Retrieve the values for the fields associated with the cancellationCode column (22)
4. If cancellationCode doesn't correspond to " " and "CancellationCode" and "NA" we select tsuch cancellation codes( i.e A, B, C, D)
5. Write to context<cancellationCode, 1>

#### **Reducer Phase:**

1. Understand the context. Each Combiner will be given all of the values that correlate to a certain key.
2. Calculate the sum of all the values by iterating through the context values.
3. These values and the key are entered into a tree map, and we use a customized comparator to sort the data depending on the highest number of values.
4. When the sorted output size is more than 1, we extract the last value.
5. Add the output to the context.

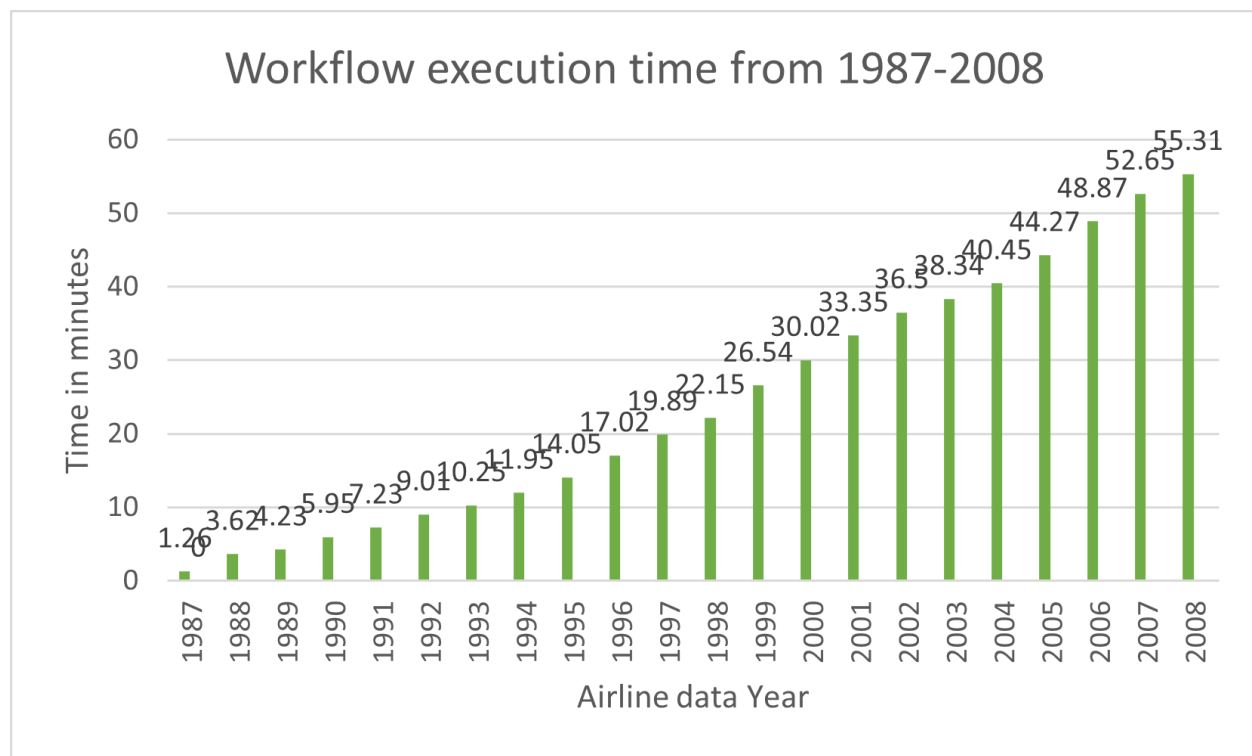
**C. A performance measurement plot that compares the workflow execution time in response to an increasing number of VMs used for processing the entire data set (22 years) and an in-depth discussion on the observed performance comparison results**



Here we are doing an analysis on execution of work processes having mapreduce occupations by shifting the quantity of assets utilized.

We are saving information consistent for every one of the runs for example flight information for every one of the 22 years. We are beginning by utilizing Hadoop on 1 Virtual machine. The absolute execution time taken is 59.214 mins. Presently we expand each VM in turn. We notice that as we increment the quantity of VMs there is a huge drop in time taken for execution. Consequently, we infer that presentation and number of assets utilized for handling enormous information are straightforwardly relative.

**D. A performance measurement plot that compares the workflow execution time in response to an increasing data size (from 1 year to 22 years) and an in-depth discussion on the observed performance comparison results**



In this analysis, we need to figure out execution regarding shifting info information.

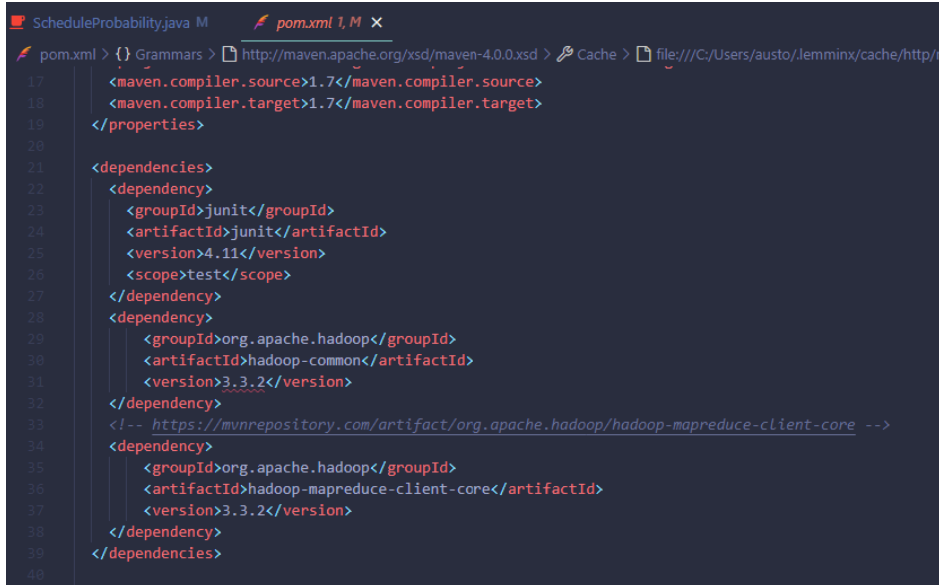
We are utilizing 2 Virtual machines all through this trial. In the first place, we execute the work process on just a single information record( 1987.csv). We see that execution finishes in truly irrelevant time.

Presently, we increment information by one year for each run and record the execution time. We see that the execution time continuously increments as the info information increments.

Subsequently, we can presume that presentation and information size are conversely corresponding.



## Libraries Being Used:



```
17 <maven.compiler.source>1.7</maven.compiler.source>
18 <maven.compiler.target>1.7</maven.compiler.target>
19 </properties>
20
21 <dependencies>
22 <dependency>
23 <groupId>junit</groupId>
24 <artifactId>junit</artifactId>
25 <version>4.11</version>
26 <scope>test</scope>
27 </dependency>
28 <dependency>
29 <groupId>org.apache.hadoop</groupId>
30 <artifactId>hadoop-common</artifactId>
31 <version>3.3.2</version>
32 </dependency>
33 <!-- https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-mapreduce-client-core -->
34 <dependency>
35 <groupId>org.apache.hadoop</groupId>
36 <artifactId>hadoop-mapreduce-client-core</artifactId>
37 <version>3.3.2</version>
38 </dependency>
39 </dependencies>
40
```

We use maven as our package manager, it allows us to install the dependencies with just a single command and also keep track of all the dependencies being used in this project in just a single file. This also allows us to pin down our libraries to specific versions which help us get the ambiguity of using unknown library versions out of the picture.

## Mapper Code for the Schedule Probability:

```
public static class Map extends
    Mapper<LongWritable, Text, Text, LongWritable> {
    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {

        String[] elements = value.toString().split(regex: ",");
        String airlines = elements[8].trim();
        String arrivalDelay = elements[14].trim();
        String departureDelay = elements[15].trim();
        if (!arrivalDelay.equalsIgnoreCase(anotherString: "ArrDelay")
            && !arrivalDelay.equalsIgnoreCase(anotherString: "NA")
            && !departureDelay.equalsIgnoreCase(anotherString: "DepDelay")
            && !departureDelay.equalsIgnoreCase(anotherString: "NA")) {
            if (Integer.parseInt(arrivalDelay) <= 10
                && Integer.parseInt(departureDelay) <= 10) {
                // System.out.println(Integer.parseInt(ad));
                context.write(new Text(airlines + " " + "ontime"),
                    new LongWritable(1));
            }
            context.write(new Text(airlines + " " + "all"),
                new LongWritable(1));
        }
    }
}
```

## Reducer Code for the Schedule Probability:

```
public static class Reduce extends Reducer<Text, LongWritable, Text, Text> {

    private DoubleWritable sumCount = new DoubleWritable();
    private DoubleWritable relativeCount = new DoubleWritable();
    private Text currentElement = new Text(string: "B-L-A-N-K / E-M-P-T-Y");

    public void reduce(Text key, Iterable<LongWritable> values,
        Context context) throws IOException, InterruptedException {
        String[] keyComp = key.toString().split(regex: " ");
        if (keyComp[1].equals(anObject: "all")) {
            if (keyComp[0].equals(currentElement.toString())) {
                sumCount.set(sumCount.get() + fetchSumCount(values));
            } else {
                currentElement.set(keyComp[0]);
                sumCount.set(value: 0);
                sumCount.set(fetchSumCount(values));
            }
        } else {
            // on schedule count is count
            // total airlines count is sumCount
            double count = fetchSumCount(values);

            relativeCount.set((double) count / sumCount.get());
            Double relativeCountD = relativeCount.get();
            sortedOutput.add(new OutputPair(relativeCountD, count, key.toString(), currentElement.toString()));
            sortedOutput2.add(new OutputPair(relativeCountD, count, key.toString(), currentElement.toString()));
            System.out.println(sortedOutput.size()+ " sorted output size check ");
            if (sortedOutput.size() > 3) {
                sortedOutput.pollLast();
            }
            if (sortedOutput2.size() > 3) {
                sortedOutput2.pollFirst();
            }
        }
        /*
        * HighestProbabilitytop3.add(new
        * OutputPair(relativeCountD, currentElement.toString()));
        * LowestProbabilitytop3.add(new
        * OutputPair(relativeCountD, currentElement.toString()));
        *
        * if(HighestProbabilitytop3.size()>3){
        * HighestProbabilitytop3.pollLast();
        */
    }
}
```

Mapper Code for the 3 Airports with shortest and longest taxi time:

```
public static class Map extends
Mapper<LongWritable, Text, Text, LongWritable> {
    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {

        String[] elements = value.toString().split(regex: ",");
        String airportorigin = elements[16].trim();
        String airportdep = elements[17].trim();
        String In = elements[19].trim();
        String Out = elements[20].trim();

        if(isInteger(In)){
            int taxiIn = Integer.parseInt(In);
            context.write(new Text(airportorigin), new LongWritable(taxiIn));
        }
        if(isInteger(Out)){
            int taxiOut = Integer.parseInt(Out);
            context.write(new Text(airportdep), new LongWritable(taxiOut));
        }
    }
}
```

Reducer Code for the 3 Airports with shortest and longest taxi time:

```
public static class Reduce extends Reducer<Text, LongWritable, Text, Text> {

    private DoubleWritable sumCount = new DoubleWritable();
    private DoubleWritable relativeCount = new DoubleWritable();

    public void reduce(Text key, Iterable<LongWritable> values,
        Context context) throws IOException, InterruptedException {
        int count =0;
        long TotalValue = 0;
        double average = 0.0;
        for (LongWritable value : values) {
            TotalValue = TotalValue+value.get();
            count++;
        }
        average = TotalValue/count;
        sortedOutput.add(new OutputPair(average, key.toString()));
        sortedOutput2.add(new OutputPair(average, key.toString()));
        System.out.println(sortedOutput.size()+ " sorted output size check ");
        if (sortedOutput.size() > 3) {
            sortedOutput.pollLast();
        }
        if (sortedOutput2.size() > 3) {
            sortedOutput2.pollFirst();
        }
        context.write(key,new Text(Double.toString(average)));
    }
}
```

## Mapper Code for the Common Cancellation Reason:

```
public static class Map extends Mapper<LongWritable, Text, Text, LongWritable> {
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

        String[] elements = value.toString().split(regex: ",");

        String canc = elements[22].trim();
        if ((!canc.equalsIgnoreCase(anotherString: "")) && !canc.equalsIgnoreCase(anotherString: "CancellationCode") && !canc.equalsIgnoreCase(anotherString: "NA")) {

            // int taxiIn = Integer.parseInt(In);
            context.write(new Text(canc), new LongWritable(value: 1));
        }
    }
}
```

## Reducer Code for the Common Cancellation Reason:

```
public static class Reduce extends Reducer<Text, LongWritable, Text, LongWritable> {

    private DoubleWritable sumCount = new DoubleWritable();
    private DoubleWritable relativeCount = new DoubleWritable();
    private Text currentElement = new Text(string: "B-L-A-N-K / E-M-P-T-Y");
    private LongWritable result = new LongWritable();

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

        long result1;
        long sum = 0;
        for (LongWritable val : values) {
            sum += val.get();
        }
        result1 = sum;
        result.set(sum);
        sortedOutput.add(new OutputPair(result1, key.toString()));
        context.write(key, result);
        if (sortedOutput.size() > 1) {
            sortedOutput.pollLast();
        }
    }
}
```

## Conclusion

In this project, we have implemented three different MapReduce algorithms to find the three airlines to find out the best and lowest chances of being on time, calculating the longest and shortest average taxi time for three airports, and discovering the most common reasons for flight cancellations.

We have tested the main feature of the Hadoop system, Distributed Computing. We have tested the time taken for the processing when we increase the number of machines to do the same work and even the time is taken when we increase the input data.