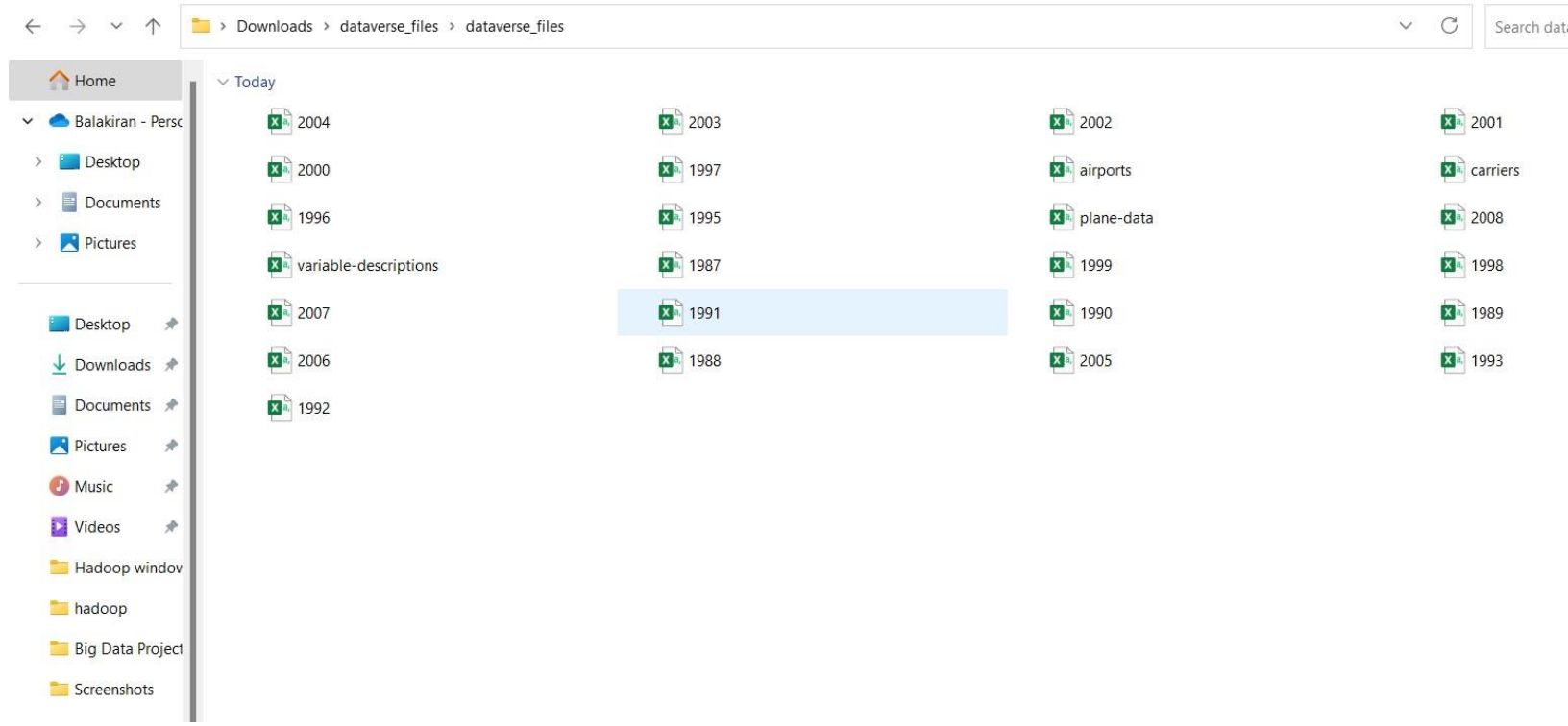
mapper and reducer functions with oozie and Hadoop to find the three airports with the longest and shortest average taxi times per flight (both in and out), respectively, as well as the most typical reason for flight cancellations. To run our project and an AWS VM to run Hadoop and the oozie, we will be utilizing Java.

Background

In this project, we will be concentrating on extracting some relevant data from the provided dataset and measuring how the vertical scaling affects the system's performance. With the expansion of the dataset, we will also examine how long it takes for our algorithm to run. huge data Given that Hadoop is recognized for distributed computation, this project provides us with excellent insights into the key components of the Hadoop system.

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.

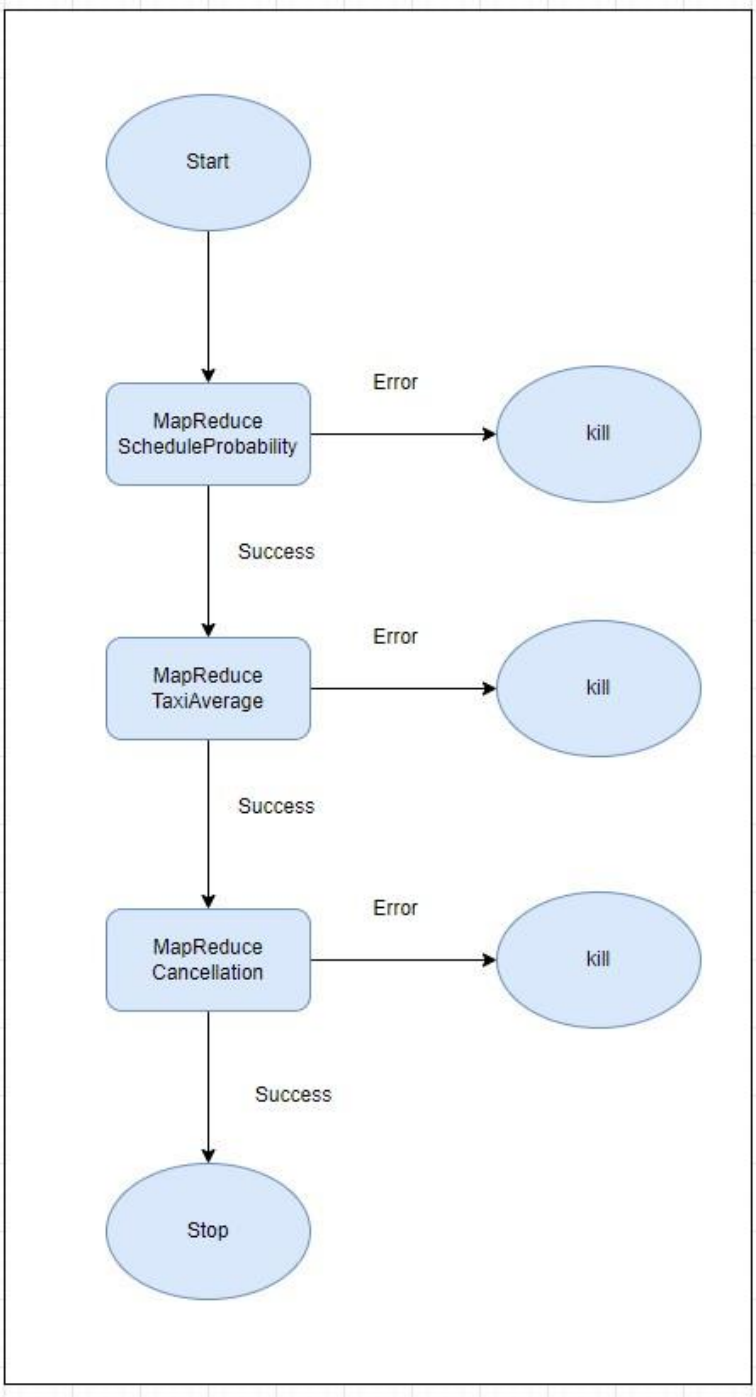


We have about 29 columns in our dataset, from which we would be using around 5-7 columns only to obtain our results.

	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	CRSElapsed	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 taxi Out 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 Out Pair and implement an interface with airport name and average taxi time as instance variables. Comparable
7. In the compare To 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.

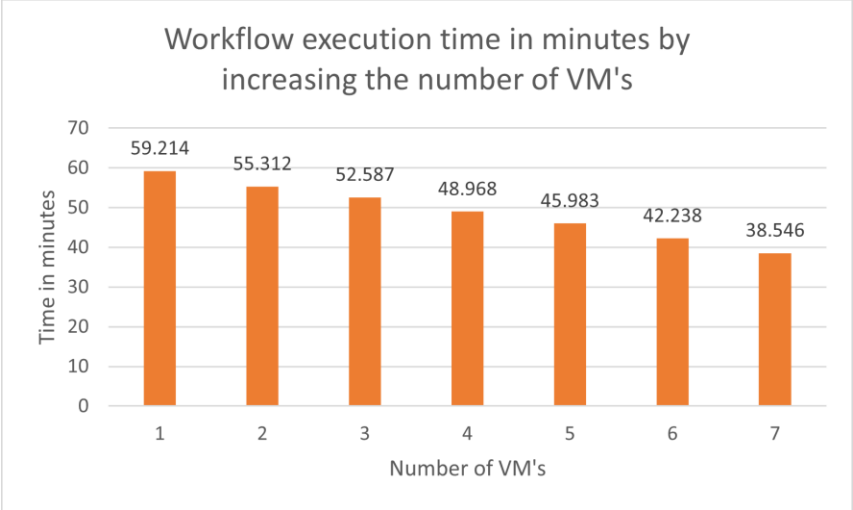
Mapper Phase:

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 cancellation Code column (22)
4. If cancellation Code doesn't correspond to " " and " Cancellation Code" and "NA" we select tsuchcancellation 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.

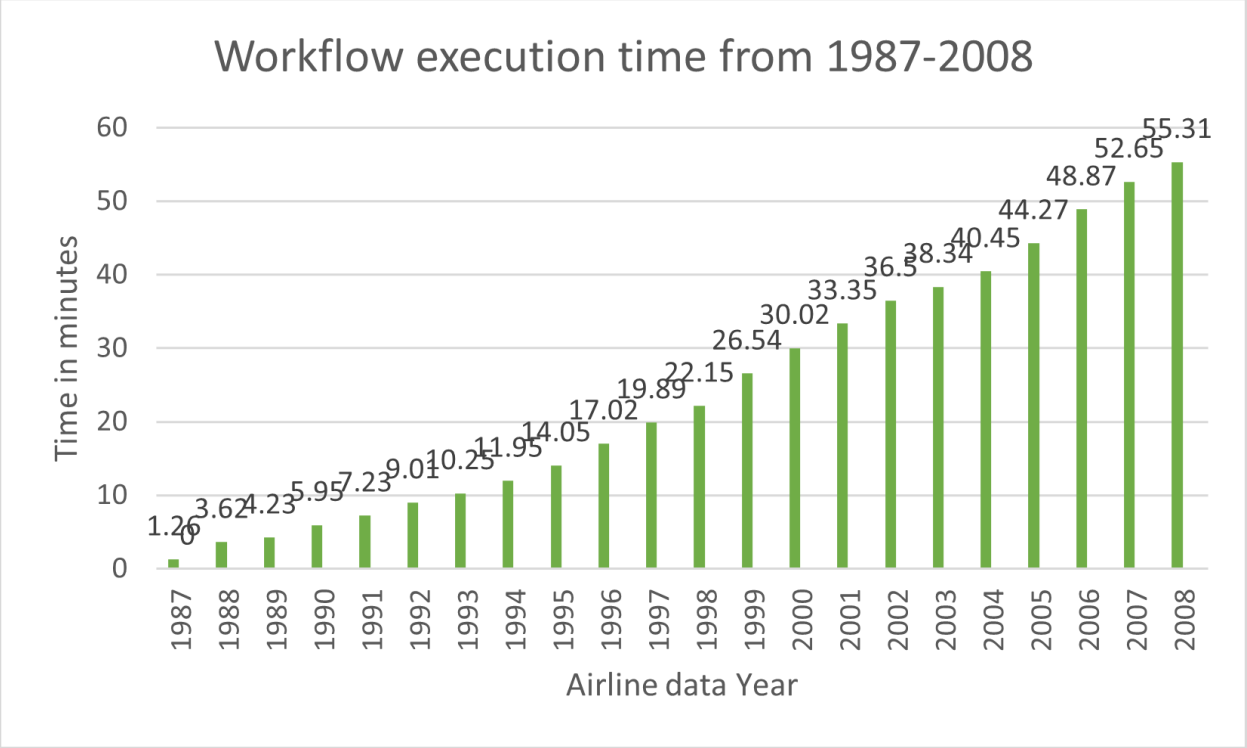
C. A performance measurement plot that compares the workflow execution time in response to an increasing number of VMs used for processing the entire data set (22 years) and an in-depth discussion on the observed performance comparison results.



Here, we are analyzing the execution of work processes that involve map-reduce jobs by changing the amount of resources used.

We are storing data that is consistent across all runs, such as flight information across all 22 years. Hadoop will be used in the beginning on a single virtual machine. The execution took 59.214 minutes in total. Right now, we grow each VM individually. We observe a significant decrease in execution time when we increase the number of VMs. As a result, we deduce that the presentation and quantity of resources used to handle vast amounts of information are simply 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 study, we must determine how to execute the movement of information.
Throughout this study, we have used 2 virtual machines. First off, we only use one information record (1987.csv) to carry out the work procedure. The conclusion of the execution is clearly completely immaterial. Now, we record the execution time for each run and increment data by one year. As the amount of information increases, we observe that the execution time also grows continually.
We can therefore assume that presentation and information size are mutually inversely correlated.

Libraries Being Used:

ScheduleProbability.java M pom.xml 1, M

pom.xml > {} Grammars > http://maven.apache.org/xsd/maven-4.0.0.xsd > Cache > file:///C:/Users/austo/.lemminx/cache/http/n

```
    <maven.compiler.source>1.7</maven.compiler.source>
    <maven.compiler.target>1.7</maven.compiler.target>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-common</artifactId>
        <version>3.3.2</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-mapreduce-client-core -->
    <dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-mapreduce-client-core</artifactId>
        <version>3.3.2</version>
    </dependency>
  </dependencies>
```

We use maven as our package manager since it enables us to handle all the dependencies required in this project in a single file and install them with a single command. This also enables us to limit our libraries to certain versions, removing the uncertainty associated with using unidentified library versions.

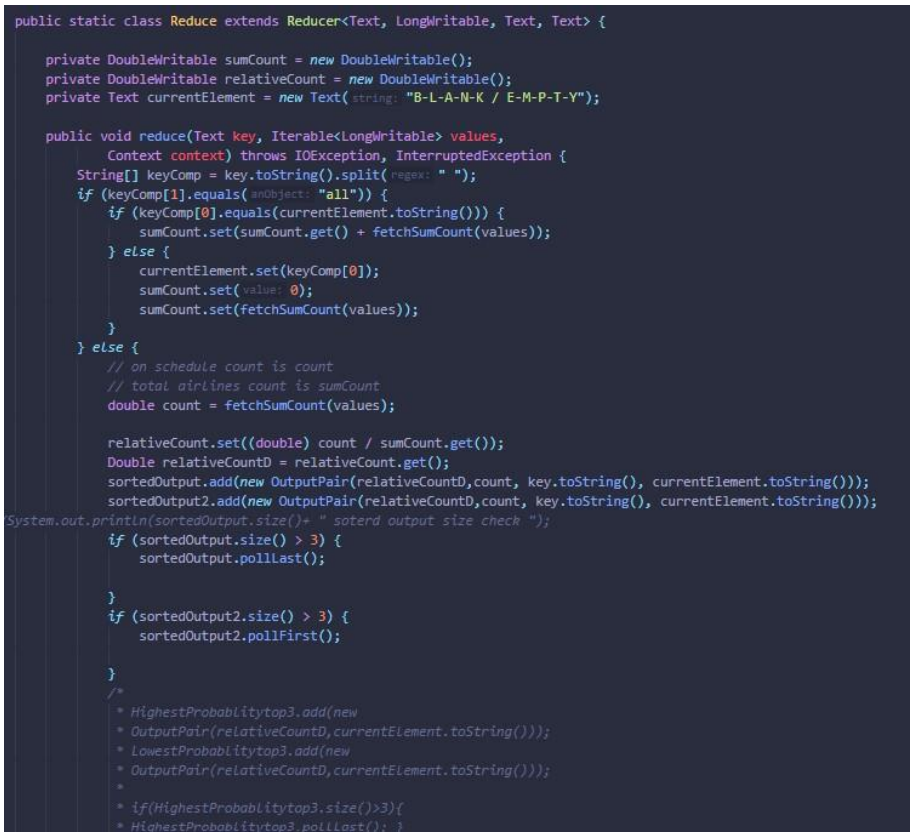
Mapper Code for the Schedule Probability:

```
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()+ " soterd output size check ");
            if (sortedOutput.size() > 3) {
                sortedOutput.pollLast();

            }
            if (sortedOutput2.size() > 3) {
                sortedOutput2.pollFirst();

            }
            /*
             * HighestProbablitytop3.add(new
             * OutputPair(relativeCountD,currentElement.toString()));
             * LowestProbablitytop3.add(new
             * OutputPair(relativeCountD,currentElement.toString()));
             *
             * if(HighestProbablitytop3.size()>3){
             * HighestProbablitytop3.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:

```java
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()));
ystem.out.println(sortedOutput.size()+ " soterd 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:

```java
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:

```java
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 designed three separate MapReduce algorithms to identify the three airlines to determine the best and lowest chances of being on time, calculate the longest and shortest average taxi times for three airports, and identify the most frequent causes of flight cancellations.

The primary component of the Hadoop system, distributed computing, has been put to the test. We have examined the processing times when using more machines to perform the same task as well as the processing times when using more input data.