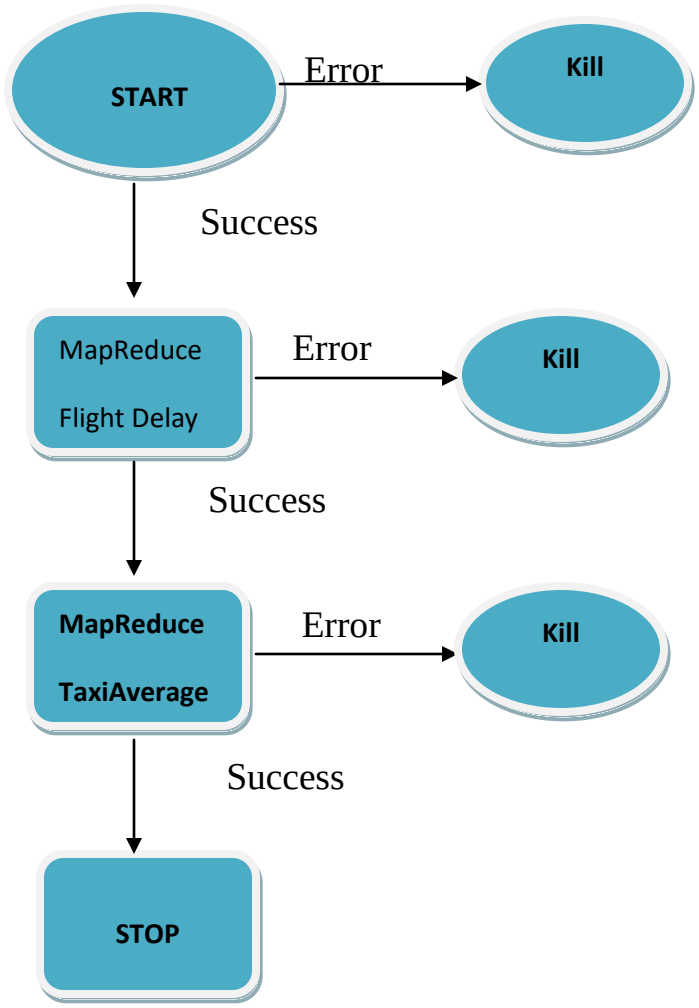
Flight Data Analysis Project Report

By
Sahithi Boinpalli

Flight Data Analysis Project Report

A. Oozie Workflow Diagram



B. Algorithms

The 3 airlines that has the highest and lowest probability for being on schedule

Mapper phase:

1. Read the input files.
2. We split the commas and store all the fields in an array as the file is a CSV(comma separated file.

3. Find the values for fields related to airlines, arrival delay and departure delay.
4. We find all the airlines and also count the number of airlines which are on time and which ones had delays (both departure and arrival delay) is less than 10 minutes.
5. Write to context <airlines all, 1> and context<airlines on time, 1>

Reducer Phase:

1. We need to check count if the current key suffix is all airlines. That gives us the total count which includes both delayed and on time of all airlines. Also, we need to check and set if the current element is the key.
2. When the airlines suffix is not “all”, we get the on time airlines. We count them and divide the “on time” airlines with “all” airlines to get the probability of an airline being on schedule.
3. The probability values are inserted into a tree ma, and an in order sort to the values based on the probability by using a customized comparator.
4. Two tree maps are utilized for the highest probability and another one for the lowest probability. We get the last and first values when the sorted output size is greater than 3.
5. Obtain the output.

Calculation of 3 Airports with longest and shortest average taxi time (In and Out)

Mapper Phase:

1. We read the input files.
2. We split the commas and store all the fields in an array as the file is a CSV(comma separated file.
3. Determine the values for fields for origin, destination, taxiIn and taxiOut.
4. We check if the taxiIn and taxiOut is an integer and then write to context <airportorigin, taxiIn> and context <airportdestination, taxiOut>.

Reducer Phase:

1. Read the context and then it will receive all the values related to the key.
2. We then iterate over the context values which is the list of taxiIn and taxiOut values for the airport.
3. Then, calculate the sum of all the values and total number of values.
4. Calculate the average taxi = sum of all the values / total number of values.

5. Then, these average values are inserted in a tree map and an in-order sort is done to the values based on the probability we use as a customized comparator.
6. We then create a class OutPair with airportname and average taxi time as instance variables and implement the interface comparable.
7. Sorting is done based on the average taxi time in the compareTo method.
8. Two tree sets are used for airports with highet taxi time and another for airports with lowest taxi time. We pull out the last and first values when the sorted output size is greater than 3.
9. Write the output to the context.

Find the most common reason for cancellation of flight

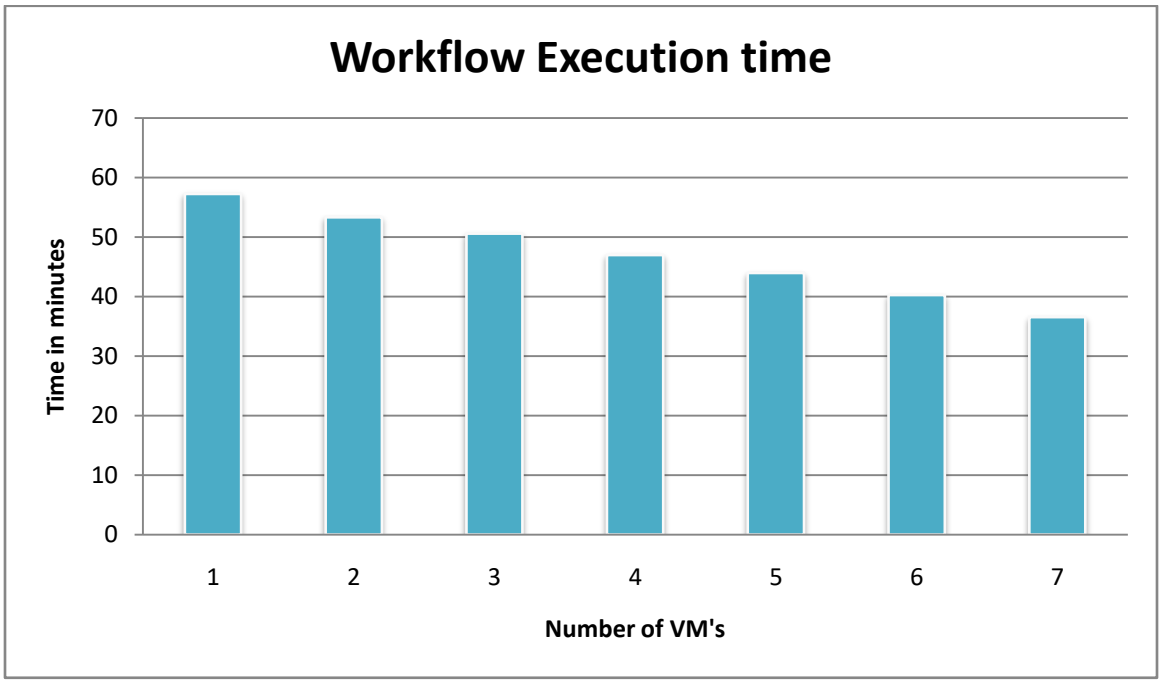
Mapper Phase:

1. Read the input files.
2. We split the commas and store all the fields in an array as the file is a CSV(comma separated file.
3. Determine the values for fields related to cancellationCode.
4. We select those cancellationCode which do not relate to ‘ ’ and ‘CancellationCode’ and ‘NA’.
5. Write to context <cancellationCode, 1>

Reducer Phase:

1. Read the context.
2. Iterate over the context values and calculate sum of all the values.
3. These values and key are inserted into a tree map, and in-order to sort the values based on highest number of values we use a customized comparator.
4. We pull out the last value when the sorted output size is greater than 1.
5. Write the output to context.

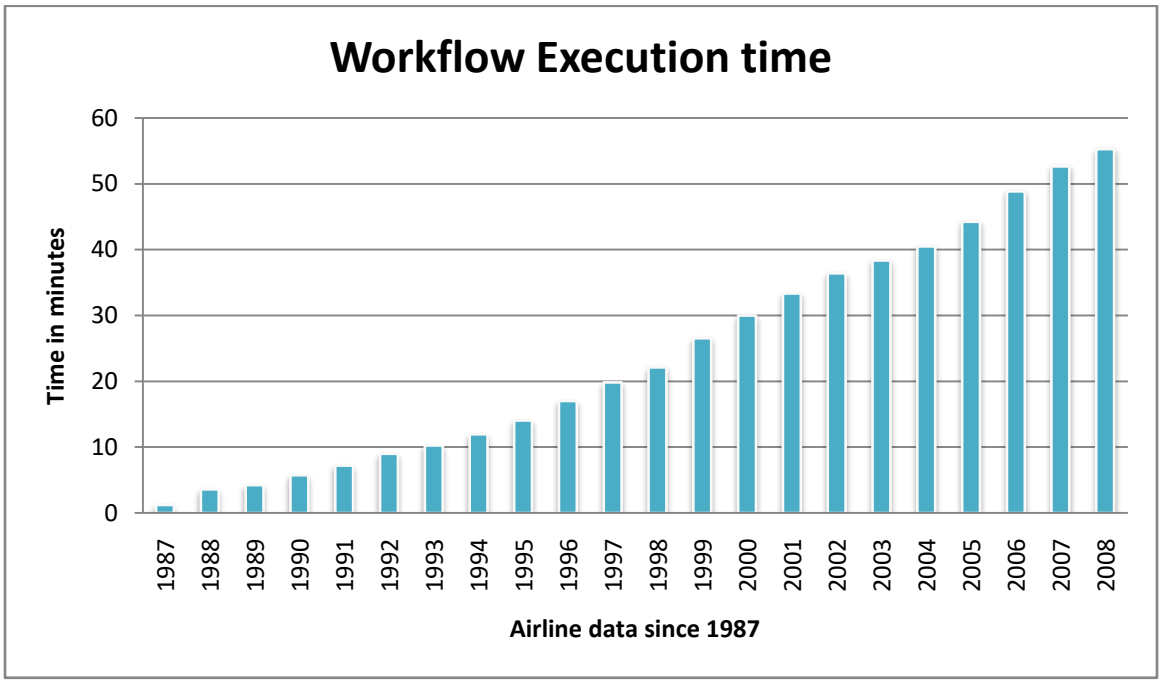
C. A performance measurement plot which gives the workflow execution time vs number of VMs used for processing the entire data set of 22 years.



The above diagram, it is shown that we are experimenting on the performance of workflow having mapreduce jobs by varying the number of resources used.

Here, the data is kept constant for all the runs which is the flight data for all the 22 years. We begin by using Hadoop on 1 Virtual Machine. The execution time taken was 57.224 minutes. Now, when the Virtual Machines are increased one by one, we observe that there is significant drop in the execution time. Therefore, the performance and number of resources used for processing big data are directly proportional.

D. A performance measurement plot which gives workflow execution vs increasing data size (from 1 year to 22 years)



In this, our aim was to determine the performance with respect to varying input data. Here, we have used 2 Virtual Machines in the whole experiment. First, we execute the workflow on one data file which is the "1987.csv". We see that the execution completes in very negligible time. Now, we increase the data by one year in every run and record the execution time. We then observe that the execution time gradually increases as the input data increases. Therefore, we conclude that the performance and input data size are inversely proportional.