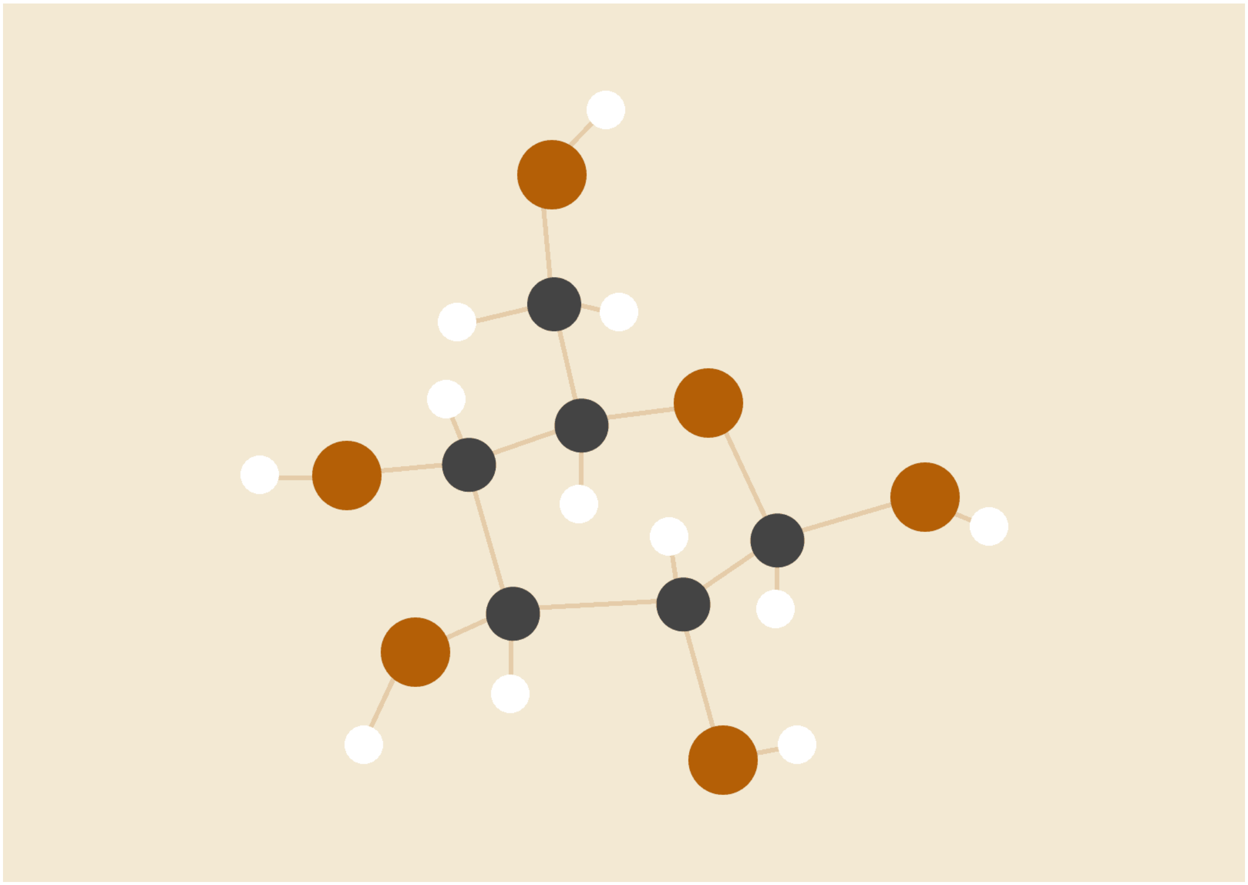
Flight Data Analysis

Project report

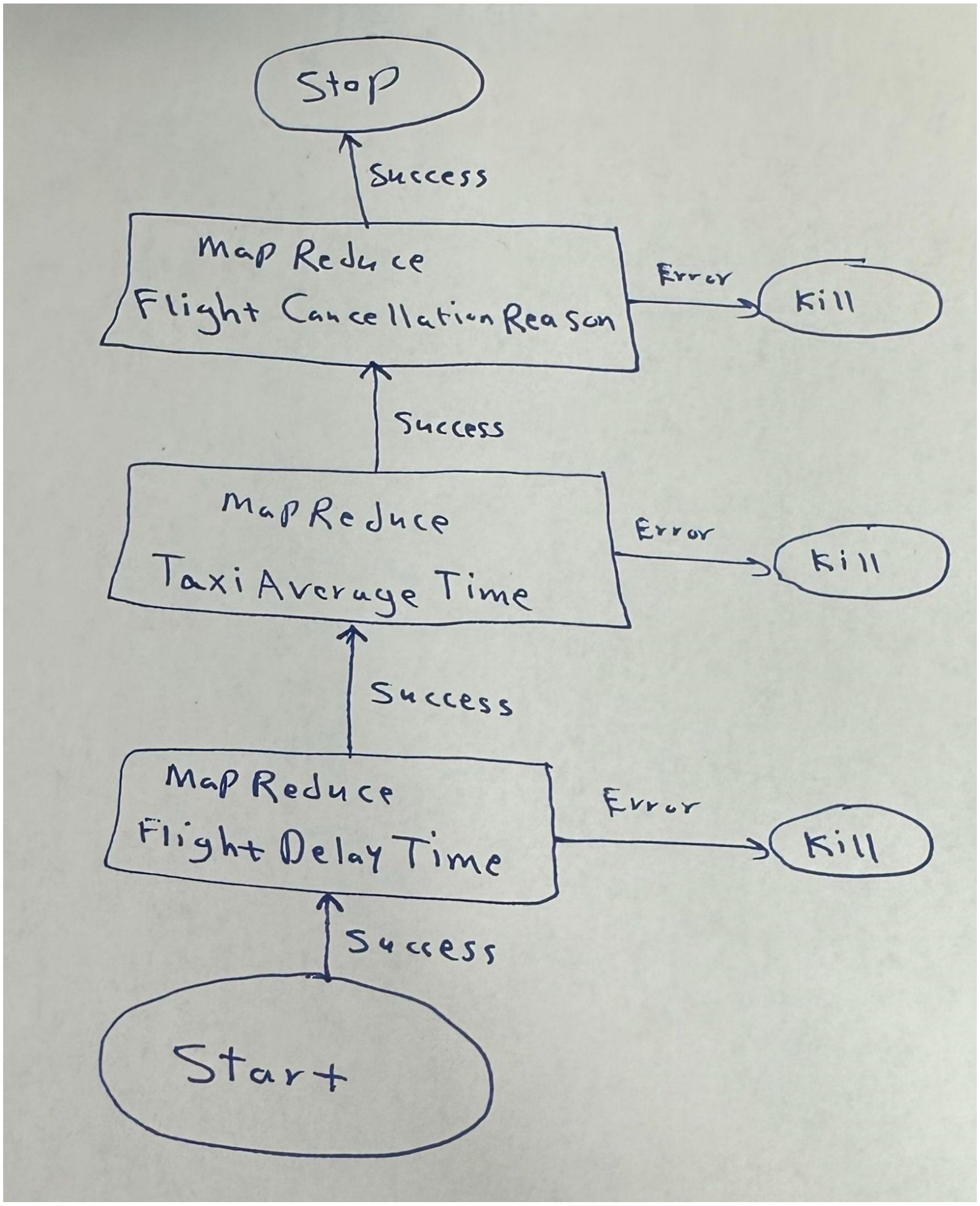


Spencer Namazi Nia

12.06.2023

CS644 Project report

Oozie Diagram Workflow



Algorithm - Flight Delay

For the flight delay task, I am supposed to find out the top three airlines with the highest probability of on scheduled flights, and the top three airlines with the lowest probability of on time flights.

Mapper function: since each entry or flight record is in a csv format, the values of each attribute are separated by commas. Therefore, I know how to split the record and fetch the desired attribute values that are the airline name (column number 8), arrival delay (column number 14), and departure delay (column number 15). Then, I ginore the first column because it is simply just the header names. By taking a look at the database, we can figure out that if a flight has delay lower than a threshold like 10 minutes for both arrival and departure, it can be considered as an on scheduled flight.Therefore, in mapper I write two types of (key, value) pairs. The first type is for those rows which their arrival and departure delay is less than 10 minutes. Let's say the name of the airline for such a flight is X. then I write it as: (X:ontime, 1). The second type is simply all the flight records that are written in the context as: (X:all, 1)

Reducer function: First I create a few necessary variables like a Text object that keeps track of the current airline in the reducer. Also, a total count and a relative count that are going to store the total number of flights and total number of on scheduled flights for that specific airline respectively. Then, I split the key in the reducer by colon because I had used ":" to write in the mapper file. After splitting it, we will have a tempList in the format: [X, "ontime"] or [X, "all"]. So, if the second element is "all", we check the first element which is X (the airline name). If currentAirline is already equal to X, then we just get the sum of the values of this key, and add this sum to the total current variable. Else, we set the currentAirline to this specific airline and assign the sum of the values for this key to the total current variable. This is all the reducer does for key value pairs belonging to "all". Regarding "on time" airlines, we compute the sum of the values for that specific key/airline. Then, we divide it by total count for that airline to get the relative count. Now, we have the probability of being on time for this airline. I have defined a data structure called ComparableOutput which is basically the object that I return as output. I have defined a comparator for this object because I want to compare the on-time probabilities between different airlines. Also, I check if the size of any of the two result sets exceeds 3, I remove the last element to keep their size as 3. In this way, we will finally have the top 3 airlines with highest and lowest probabilities of being on scheduled. These two sets are of type TreeSet with my defined comparable Object type, therefore they are alway sorted. Finally, in the main file I just write the results in the output files.

Algorithm - Taxi Average Time

In this part, it is supposed to find out the average taxi time for all airports and finally print the top 3 highest and lowest ones. My algorithm is pretty similar to the previous part.

Mapper function: since each entry or flight record is in a csv format, the values of each attribute are separated by commas. Therefore, I know how to split the record and fetch the desired attribute values that are the original airport (column number 16), departure airport (column number 17), inTime (Co. 19), and outTime (Co. 20). InTime and outTime might be NA and not integer, so I firstly check if they are not integers I simply ignore them. Else, for both origin and departure airport, i write them in the context in this tuple format: (key=airport_name, value=inTime) and (key=airport_name, value=outTime)

Reducer Function: In reduce function, for each airport as a key, now we have access to its corresponding times. So the count of them is the number of values, and total time is sum of them. Therefore these two parameters can be calculated in the reducer function easily. Then, I divide them to get the average taxi time for that specific airport. For keeping track of the highest and lowest average taxi times, I have two Treesets (one for highest and one for lowest) that their order are based on a comparable object class that I have defined, and it has two attributes: the airport, and its average taxi time. In the reducer, each time I add the corresponding object for an airport to these two lists, and then if the size of each list exceeds 3, I will simply remove the last element. Finally, I will write the airport and its average taxi time in the context. In the main file, I just simply print the output in text files.

Algorithm - Flight Cancellation

In this part, it is supposed to find the most common reason for flight cancellation among all the existing records.

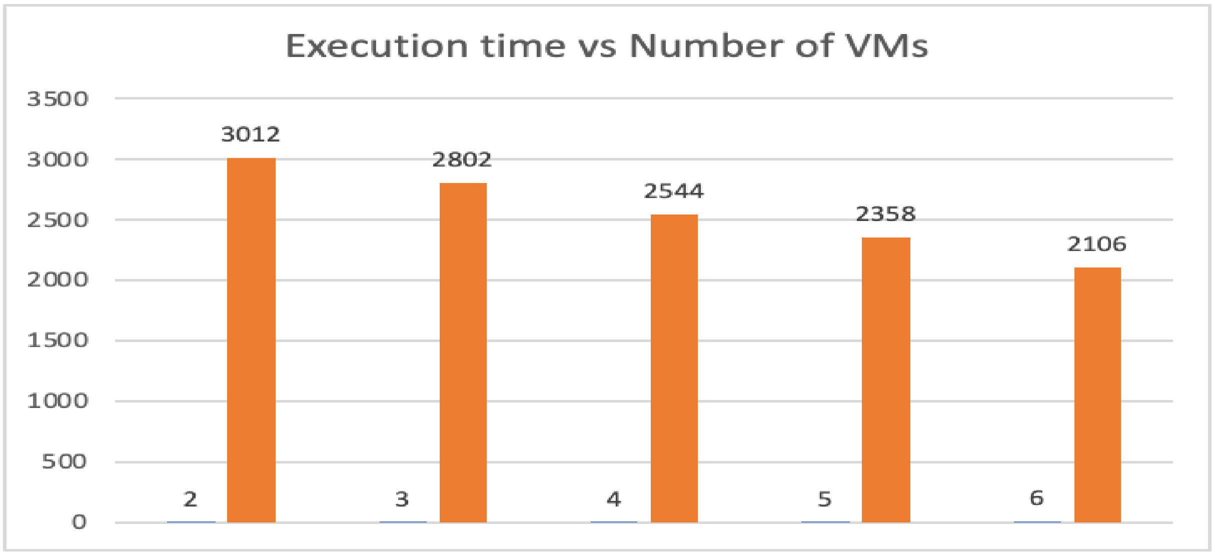
Mapper function: since each entry or flight record is in a csv format, the values of each attribute are separated by commas. Therefore, I know how to split the record and fetch the desired attribute values, and the only desired attribute is the cancellation_reason (Column index = 22). Then, we simply add write this pair to the concept: (cancellation_code, 1)

Reducer function: I have defined a comparableOutput object class for this part which consists of a cancellation code and its corresponding count. I also have a TreeSet which is is a set of my comparable objects, and is going to only keep track of one element at a time, which belongs to the most common reason of flight cancellation. In reduce function, for each key, we sum over all its values to get the total number of occurrences for that specific reason. Then we add it to the TreeSet, and check if TreeSet size exceeds 1, we simply remove from the end of that to only keep track of the MOST common reason. Finally, we write the most common reason with its total count in the context. In the main file, we check the code for the most common reason, and based on what it is (A, B, C, etc), we print the most common reason for the flight cancellation with its number of occurrences.

Performance Measurement Plot (No. VMs)

X axis: number of virtual machines

Y axis: execution time (seconds)

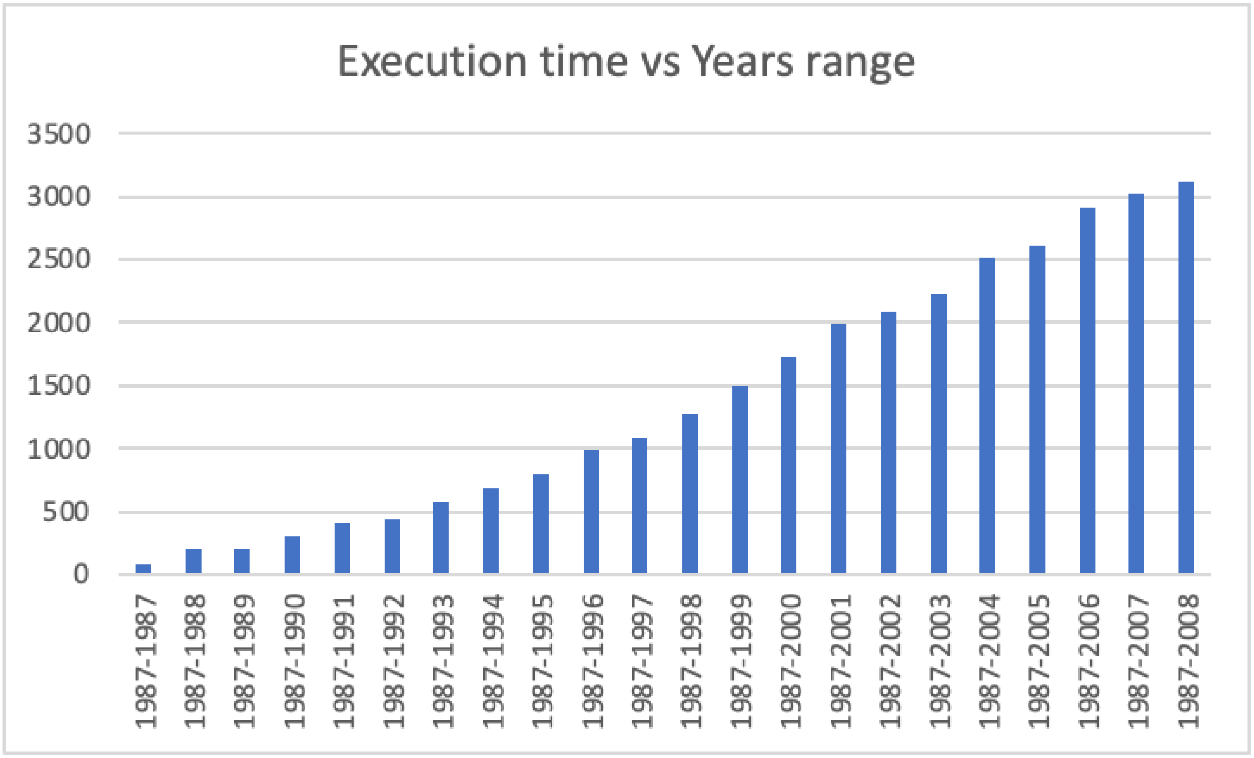


In this experiment, the data that has been used for each experiment instance is fixed as the whole dataset for all the 22 years. The varying parameter is the number of used virtual machines or nodes to accomplish the experiment, and I have tested this count from 2 to 6. For each text instance, the total execution time has been measured in seconds. What can be inferred from this chart is that as we increase the number of virtual machines, it drastically reduces the total time taken to carry out the experiment. Therefore, we can conclude that for such a large amount of data, it is recommended to use a relatively higher number of virtual machines.

Performance Measurement Plot (No. Years)

X axis: years to be considered

Y axis: execution time in seconds



In this experiment, the number of virtual machines is fixed as 2, and the varying parameter is the number of years that have been considered in the experiment that is basically the size of the dataset. We have 22 different years, so the x-axis can have 22 different values, and for each, we compute the total time taken in seconds, and we can see their result comparison in the above chart. As the number of years to be considered increases, the size of the data increases, and therefore, total time needed to complete the experiment instance increases as well. Thus, we can conclude that the larger the dataset, typically the longer it takes for the machine or machines to complete the task on the dataset(s).

Conclusion

I have provided all the results in an attached file called “output.txt” for different years and different tasks. Also, all the “.java”, “.class”, “.xml”, and “.jar” files are stored.

“Thanks!”