Final Project: Flight Data Analysis

Christopher Chan

DS644 – Introduction to Big Data

December 22nd 2023

ProfessorYijie Zhang

1. Objective

In this project, a cloud-based Big Data workflow (Hadoop, Oozie, and AWS EC2) was developed for processing and analyzing the flight data from October 1987 to April 2008. The three airlines with the highest and lowest probabilities of being on schedule, the three airports with the longest and shortest average taxi times per flight, and the most common reason for flight cancellations was determined by analyzing the flight data. The workflow must encompass at least three MapReduce jobs in fully distributed mode. Performance evaluation of the workflow was evaluated on 2 VMs, gradually scaling up to 7 VMs in increments, and measuring execution time. Additionally, the workflow is run progressively on the entire dataset in one-year increments.

2. Algorithms Description

Part 1: Find the 3 airlines with the highest and lowest probability, respectively, for being on schedule.

MapperOne

The mapper function parses the “ArrDelay” column from the flight data. If the value is within the range of -5 to 5 minutes, it is classified as on-time; otherwise, it is classified as late. The map function emits key-value pairs with the concatenated airline and airport code as the key and a float value of 0.0 or 1.0 for late and on-time, respectively.

ReducerOne

The reduce function calculates the on-time percentage for each unique key by dividing the total value by the number of occurrences. The key and the percentages are stored in a TreeMap called onTimeMap. In the cleanup phase, onTimeMap is sorted by values and stored as sortedOnTimeMap. Iterating through sortedOnTimeMap, the top and bottom three entries are written to the output to identify the three airlines with the highest and lowest probabilities of being on schedule.

Part 2: Find the 3 airports with the longest and shortest average taxi time per flight (both in and out), respectively.

MapperTwo

The map function parses input file line by line and is initialized as a String called line and parses the “Taxi In Time”, “Taxi Out Time”, and “Location” column from the flight data. The map function checks if the String has a length greater than 19, if the “Taxi In Time” and “Taxi Out Time” is a number, and if “Taxi In Time” and “Taxi Out Time” is not empty. If the conditions are met, the map function emits key-value pairs with the concatenated “Taxi In” or “Taxi Out” and the airport code as the key and a double of the taxi time. By configuring the key, the map function distinguishes between "Taxi In" and "Taxi Out" values, allowing for the identification of airports with the longest and shortest average taxi times for both incoming and outgoing flights.

ReducerTwo

In the reducer class, two TreeMap, "inAverageMap" and "outAverageMap" store the computed averages taxi time. In the reduce function, the code iterates through the input values, accumulating the sum and counting the number of values. The average taxi is calculated, the result is stored in "inAverageMap” and "outAverageMap” depending on if the key contains “Taxi In” or “Taxi Out”. The cleanup sorts both maps using the sortByValue function and utilizes a helper method, "writeTopAndBottom," to write the top and bottom three entries for both incoming and outgoing averages to the output context. The resulting output includes information about the airport, direction (either "Taxi In" or "Taxi Out"), and the average taxi time with the longest and shortest average taxi times for each direction.

Part 3: Find the most common reason for flight cancellations.

MapperThree

The map function parses input file line by line and is initialized as a String called line. The line is parsed to extract the “Cancellation code” column from the flight data. The map function checks if the cancellation code is empty and NA before emitting a key-value pair for the cancellation code as the key and a long of 1.

ReducerThree

The reducer iterates through the values associated with a unique key and sums up their individual counts. The value is stored in a TreeMap called top5Map. In the cleanup method, the top5Map is sorted based on the values using a helper method called sortByValues and stored in a new TreeMap called sortedTop5Map. The code iterates through sortedTop5Map and writes the outputs of the cancellation code and total count. The max value in the treeMap can be used to determine the most common reason for flight cancellation.

Workflow

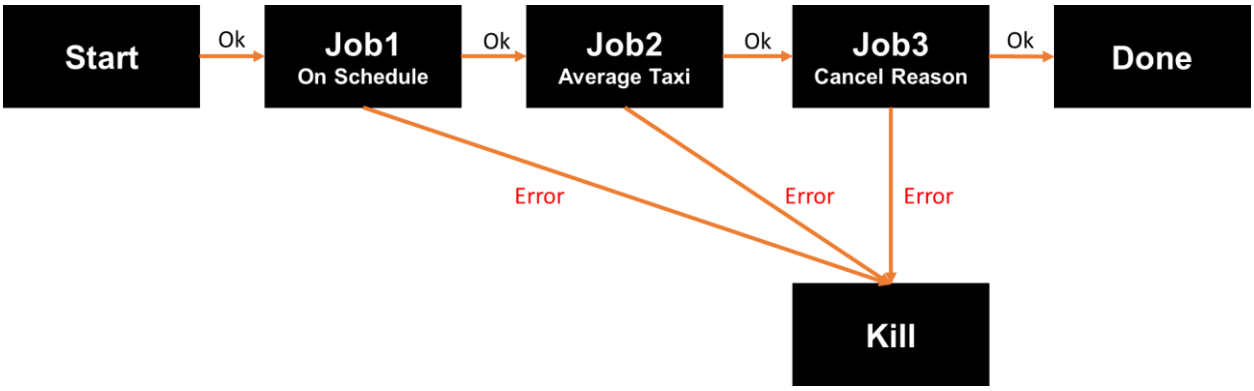

Figure 1. Oozie Workflow

The script containing the three mapreduce functions were packaged into a JAR file and Oozie workflow was established to execute the functions subquentially. In the event of an error within any function, the workflow will terminate. If no errors occur, the workflow will proceed uninterrupted.

3. Performance Measurements

After executing the flight analysis script on various numbers of data node configurations, the data reveals a downward trend in the execution time as the number of DataNodes increases. The graphical representation below shows a noticeable reduction in execution time from 1369 seconds to 938 seconds as the number of data nodes increase from 2 to 7. This relationship indicates that distributing the workload across a greater number of data nodes leads to a shorter execution time of the flight analysis task. The data reveals the importance of parallel processing as there is remarkable performance improvements achieved through the data node scaling.

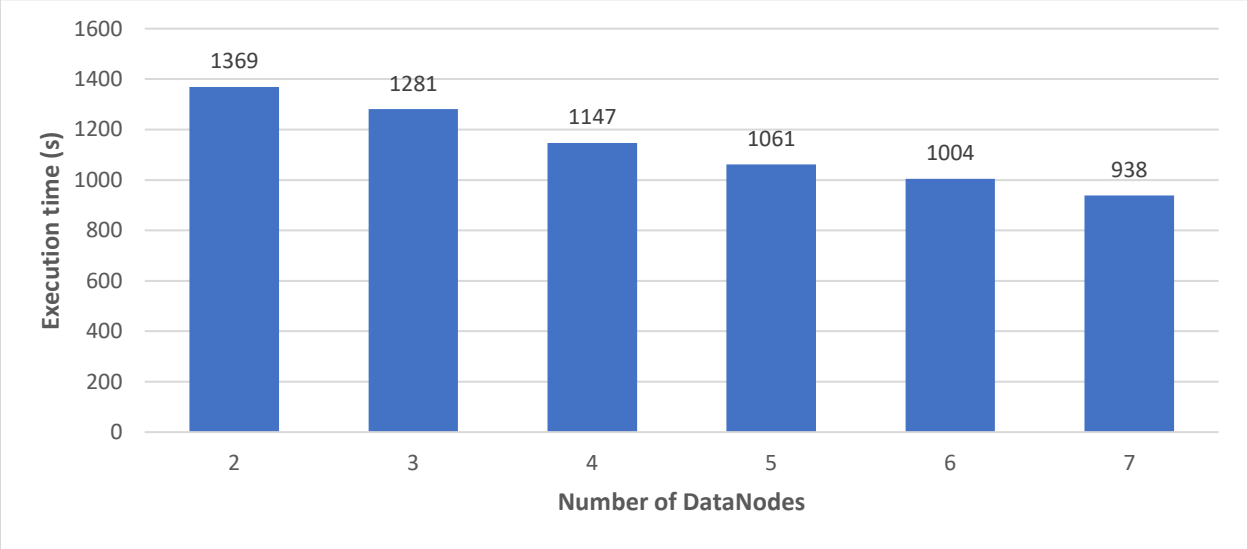

Figure 2. Oozie workflow execution time based on various number of datanodes

NUMBER OF DATANODE	EXECUTION TIME (S)
2	1369
3	1281
4	1147
5	1061
6	1004
7	938

Table 1. Number of data node versus task execution time

After executing the flight analysis script on various numbers of data node configurations, the data reveals an upward trend in the execution time as the input file size increases. The graphical representation below shows a noticeable increase in execution time from 96 seconds to 938 seconds as the number files increase from 1 (only 1987) to 22 (1987 – 2008). This relationship was expected as a larger input file would take longer to execute given that the computing power remains relatively constant.

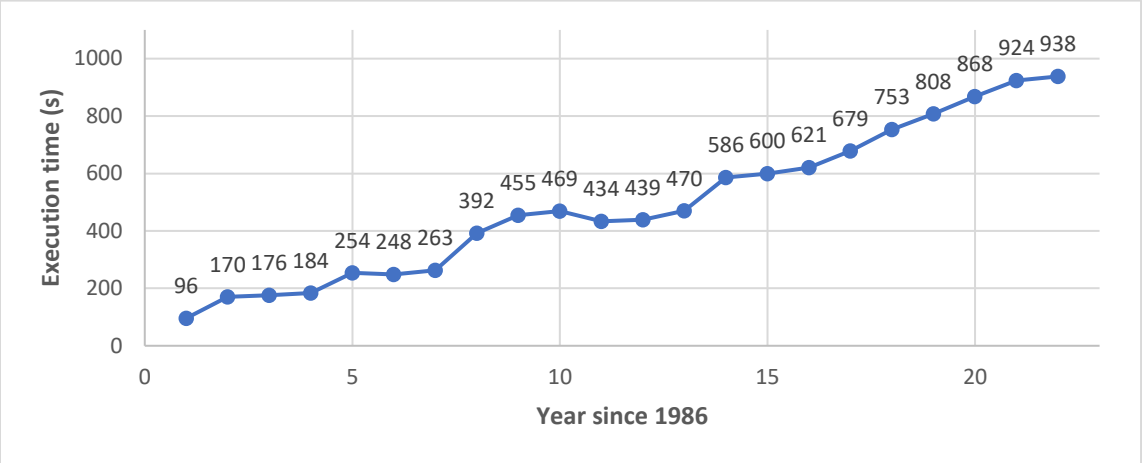

Figure 3. Oozie workflow execution time on 7 data nodes based on input file size

Years	Actual Year	Execution Time (s)
1	1987	96
2	1988	170
3	1989	176
4	1990	184
5	1991	254
6	1992	248
7	1993	263
8	1994	392
9	1995	455
10	1996	469
11	1997	434
12	1998	439
13	1999	470
14	2000	586
15	2001	600
16	2002	621
17	2003	679
18	2004	753
19	2005	808
20	2006	868
21	2007	924
22	2008	938

Table 2. Input size versus task execution time

Table 2.