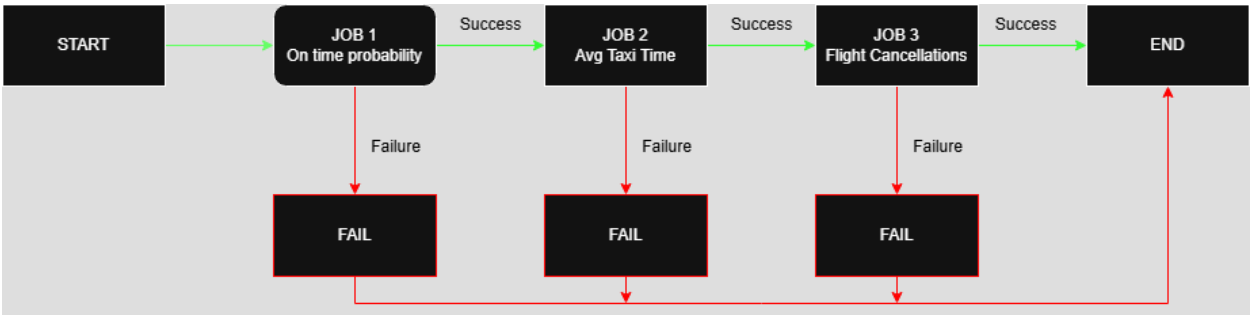
Project – Airline Data Analysis

DS644 – Introduction to Big Data

Dylan Da Costa (dd567), Omkar Naik (on36), Vaishnavi Gandhi (vg489)

1. Oozie Workflow Diagram



2. Explanation of Code Logic for the 3 Jobs

- On time Probability

Mapper: The mapper function analyzes flight records to assess the on-time performance of airlines. It reads input line by line from standard input and skips the header row by checking if the first field is "Year." For each data row, it extracts the airline carrier code (from column index 8) and the arrival delay (from index 14). If both values are valid and the delay is not missing ("NA", "null"), the script attempts to convert the delay value into a float. If successful, it classifies the flight as on-time if the delay is less than or equal to zero, meaning the flight arrived early or exactly on time. It then emits a line in the format carrier\t1, on_time, where 1 represents one flight, and on_time is either 1 (on time) or 0 (delayed). This allows the reducer to compute on-time performance probabilities later.

Reducer: The reducer function aggregates the total number of flights and the number of on-time arrivals for each airline carrier. It reads the mapper output and uses a defaultdict to maintain a running total of flights and on-time flights for each carrier. For each line of input, it splits the carrier and the values, then updates the flight totals accordingly. After all lines are processed, the reducer calculates the on-time probability for each carrier by dividing the number of on-time flights by the total number of flights. These probabilities are stored in a list of tuples and then sorted in ascending order. Finally, the reducer prints the three carriers with the lowest on-time performance, followed by the three with the highest, offering insight into which airlines are most and least reliable in terms of punctuality.

- Avg Taxi Time

Mapper: The mapper function for calculating average taxi time reads flight records from standard input using Python's built-in csv module. This is helpful because it safely handles fields that may contain commas enclosed in quotes. It skips the header row, then processes each subsequent line by extracting the origin airport code (from column index 16), the taxi-in time (index 19), and the taxi-out time (index 20). These values are converted to floats, with missing entries defaulting to 0.0. The mapper calculates the total taxi time for each flight by summing the taxi-in and taxi-out times. If the origin airport is present, it emits a line in the format origin\t total_taxi_time\t 1. Here, the taxi time is recorded alongside a count of 1, allowing the reducer to compute an average later.

Reducer: The reducer function aggregates the taxi time data emitted by the mapper. It uses a dictionary (defaultdict) to keep a running total of taxi times and counts for each airport. For every line of input, it splits the values and adds the taxi time to the airport's total and increments the count. After all lines are processed, it computes the average taxi time for each airport by dividing the total taxi time by the number of records. It then creates a list of tuples containing each airport and its corresponding average taxi time. This list is sorted in ascending order based on average taxi time. The reducer finally prints out two reports: the three airports with the shortest average taxi times and the three with the longest. These insights can help identify operational efficiency across different airports.

- Flight Cancellations

Mapper: The mapper function reads flight data line by line from standard input, typically a CSV file. It first skips the header row by checking if the first field is "Year." For each remaining line, it splits the row by commas to access individual fields, then extracts the cancellation reason code from the 23rd column (index 22). If the cancellation code is valid — meaning it's not empty, "NA", or "null", the mapper outputs a key-value pair in the format cancel_code\t1, where \t represents a tab character. This output indicates that one flight was cancelled for the given reason and is sent to the reducer for aggregation

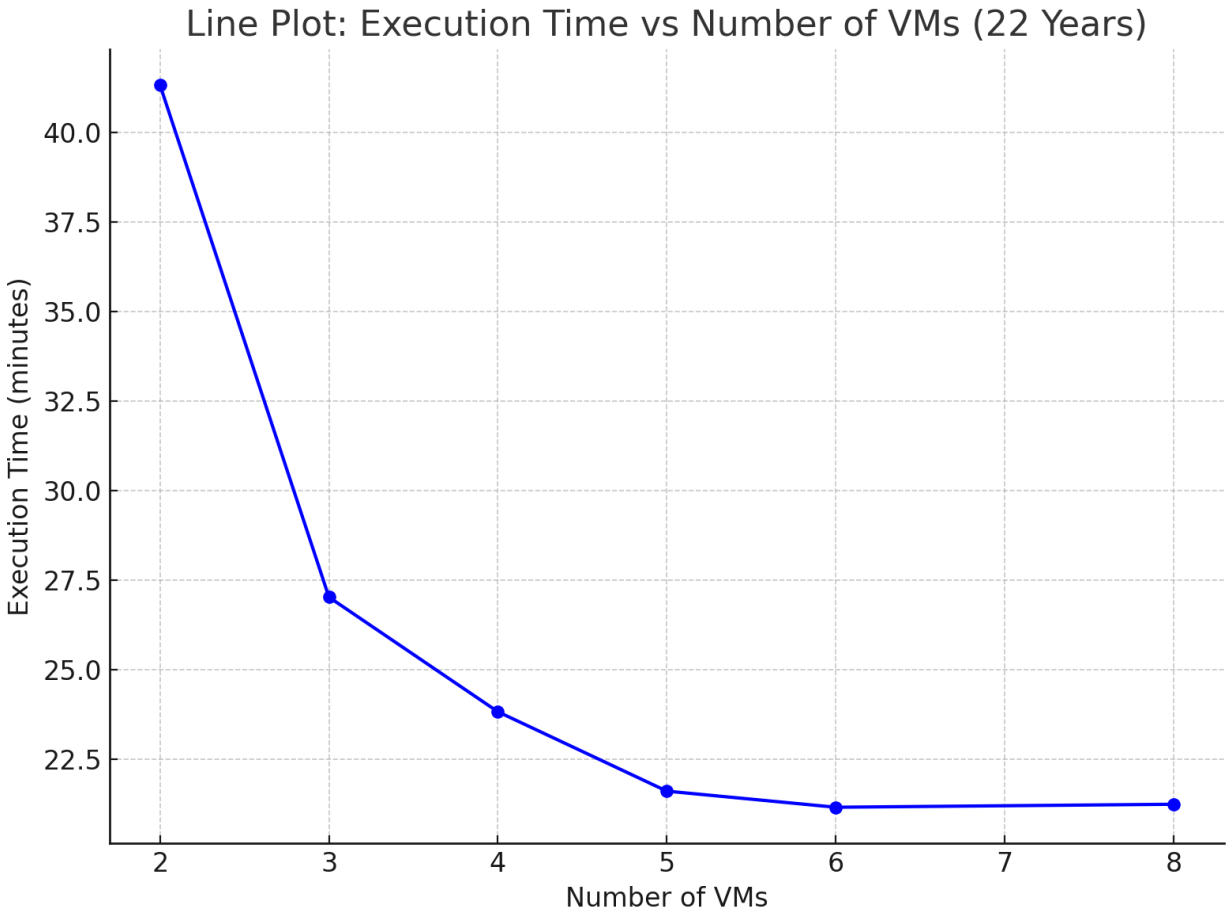
Reducer: The reducer function takes these key-value pairs as input, which are grouped and sorted by cancellation reason. It keeps track of the current cancellation code its processing and accumulates the total count for that reason. Each time it encounters a new reason; it compares the accumulated count of the previous reason with the highest count seen so far. If it's greater, it updates the maximum. After processing all lines, the reducer prints only the cancellation reason with the highest total count. The final output displays the most common cancellation code and the number of times it occurred, helping identify the leading cause of flight cancellations

3. Performance Measurement Plots

- Increasing VMs vs Execution Time (Entire Dataset)

This plot shows how the total execution time changes when the number of virtual machines (VMs) is increased, while processing the constant data size of 22 years.

The plot indicates that execution time decreases significantly when increasing from 2 to 4 VMs, highlighting the benefits of parallel processing. However, beyond 4 or 5 VMs, the reduction in execution time becomes marginal. This pattern reflects diminishing returns, where the overhead of managing more VMs limits further performance improvements. It suggests there is an optimal range for VM usage, beyond which additional resources do not yield proportional gains.



- Varying Dataset Size vs Execution Time

This plot illustrates how execution time increases as the amount of data, measured in years, increases while using a fixed number of VMs.

Initially, from 1 to around 10 years, the increase in execution time is gradual, suggesting efficient scaling for smaller datasets. However, as the number of years continues to grow, especially beyond 15, the execution time begins to rise more steeply. This indicates that the system begins to face performance bottlenecks due to resource constraints such as CPU, memory, or I/O limits. The steeper slope at the higher data volumes reflects the increasing complexity and load on the system.