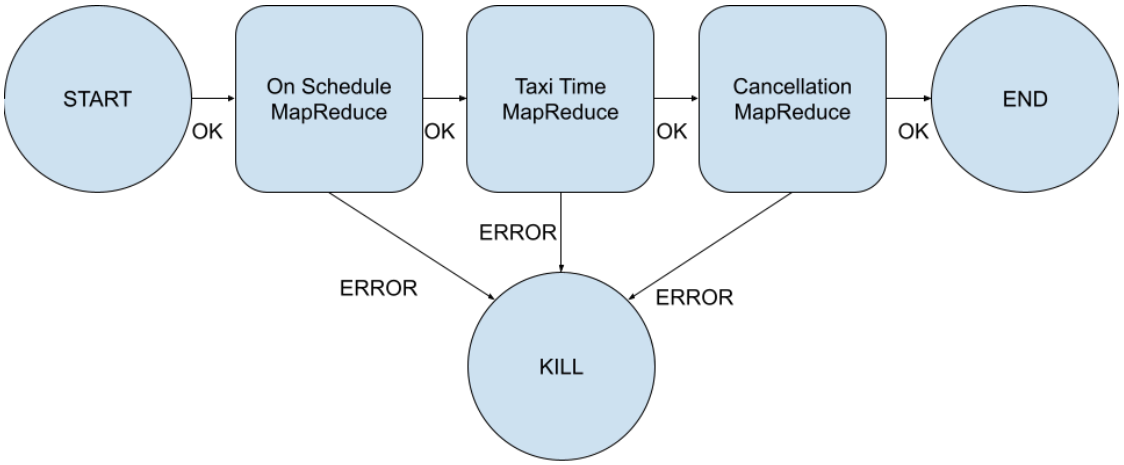
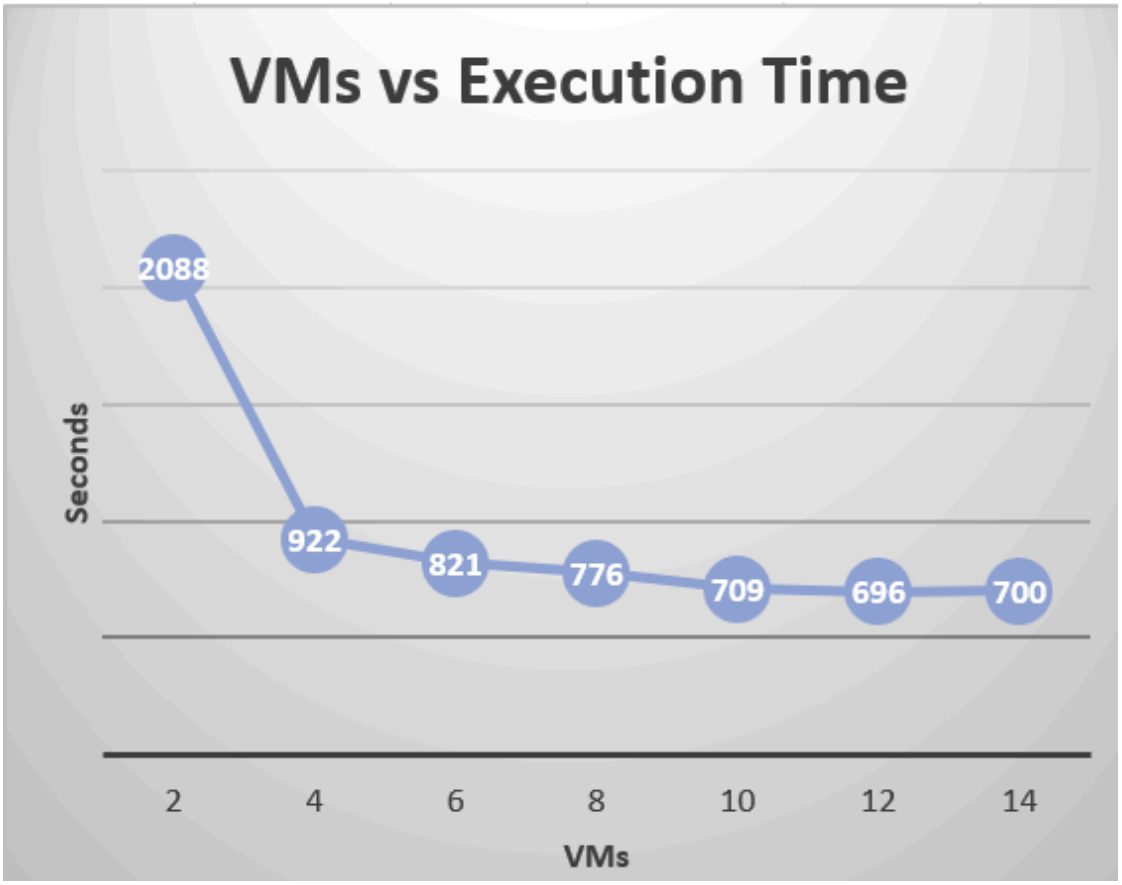


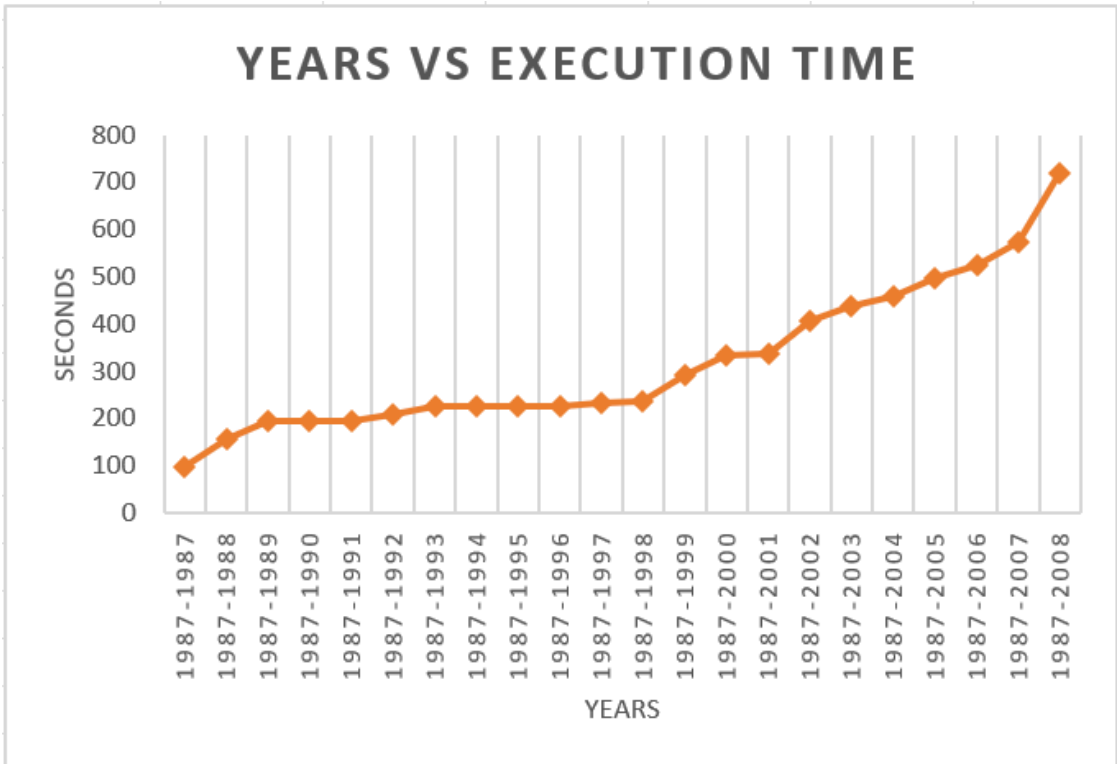
The `CancellationsMapper` reads input records, splits each line into fields, and filters out the header row. It then checks if a particular flight was canceled (indicated by a specific field, e.g., `"1"` in the 22nd column) and whether the cancellation reason is valid (not `"NA"` and non-empty). When these conditions are met, it emits the cancellation reason as the key with a count of `1`. The corresponding `CancellationsReducer` aggregates these counts for each cancellation reason, storing the results in a sorted map. During cleanup, it identifies the most common cancellation reason and maps it to a descriptive label (e.g., carrier, weather, NAS, security). If no cancellations are recorded, it outputs an appropriate message indicating the absence of data.

The `OnScheduleMapper` processes airline schedule data, filtering out header records. For each airline, it examines the scheduled delay (found in a specific column) and assigns a binary indicator: `"1"` if the delay is less than or equal to 10 minutes, and `"0"` otherwise. These indicators are emitted with airline identifiers as keys. The `OnScheduleReducer` then computes the average on-schedule performance for each airline by summing the indicators and dividing by the total number of records. After processing all data, it sorts the airlines based on their average punctuality, displaying the top three and bottom three performers, along with messages indicating the absence of data if applicable.

The `TaxiTimeMapper` analyzes taxi time data, emitting pairs of location identifiers and taxi times, provided the data is valid (not `"NA"`). The `TaxiTimeReducer` calculates the average taxi time for each location, handling exceptions where data might be missing or zero. It maintains separate maps for valid averages and exceptional cases. During cleanup, it sorts and outputs the top three locations with the highest taxi times and the bottom three with the lowest times, along with messages for cases where data is insufficient. It also reports any locations with zero or missing data, ensuring comprehensive insights into taxi operation efficiency.



Increasing the number of VMs typically leads to a reduction in workflow execution time. This is because adding more VMs enhances the processing capacity of the Hadoop cluster, allowing data to be processed in parallel across multiple data nodes. As a result, each MapReduce job completes faster, which in turn shortens the overall duration of the Oozie workflow. However, this trend does not continue indefinitely; for a fixed data volume, further increasing the number of VMs may no longer yield shorter execution times. Once the execution time reaches a certain threshold, adding more VMs will have little to no impact on reducing it further. This is because a higher number of VMs increases the communication overhead between data nodes in the Hadoop cluster, leading to longer information exchange times that can offset the benefits of additional processing power.



As the volume of data grows, the execution time of the Oozie workflow correspondingly tends to increase. Initially, this increase in processing time is gradual as the data volume expands, mainly because the data growth during the first few years is modest. However, after 1998, the rate at which processing time increases accelerates significantly, resulting in a much steeper slope compared to earlier years. This is largely due to the rapid rise in flight data between 1998 and 2008, which surpasses previous growth rates. Additionally, more people have started choosing air travel during this period.