FLIGHT DATA ANALYSIS REPORT

Semester: Spring 2024

TEAM:
NAGA DATHA SAIKIRAN BATTULA (nb547)
GADE HARSHITH (hg355)
SAI NIKHIL DUNUKA (sd2279)
YASHWANTH REDDY BUSHIREDDY (byr3)

Contents

Introduction:

In this project, we implemented an Oozie workflow which can be used to execute three map-reduce programs to analyze the flight data. The dataset is available from Harvard university.

Dataset:https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/HG7NV7

Overview:

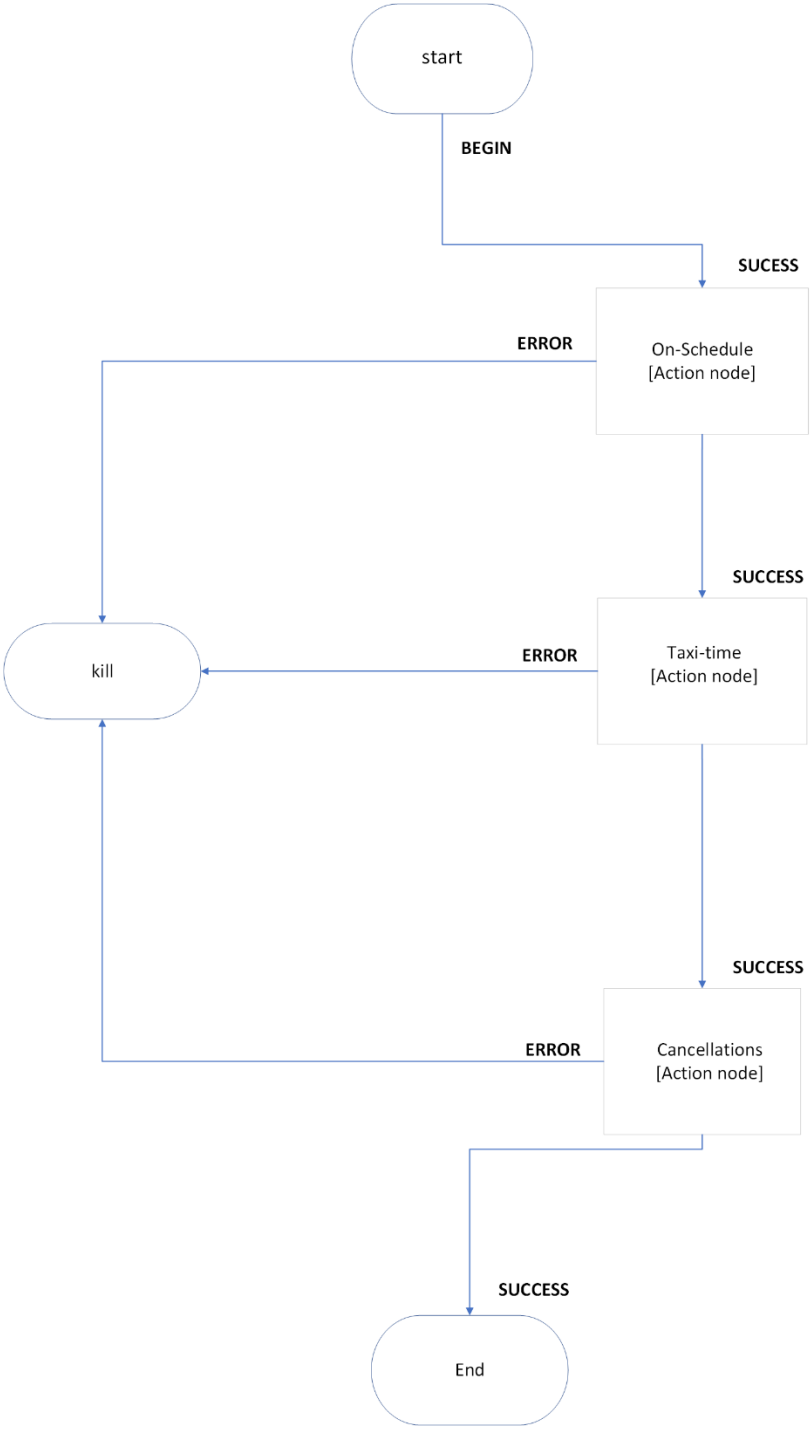
The project revolves around the execution of three distinct tasks, each designed to address specific facets of flight data analysis. These tasks encompass:

1. **On-Schedule Performance Analysis**: This task involves identifying airlines with superior on-schedule performance and contrasting them with those exhibiting suboptimal punctuality. By scrutinizing flight schedules, departure/arrival times, and delays, we aim to discern trends and patterns that shed light on operational efficiency and service quality.

2. **Airport Taxi Time Analysis**: Delving into the intricacies of ground operations, this task focuses on analyzing taxi times at airports. By examining the duration of taxiing for departing and arriving flights, we endeavor to identify airports with efficient taxiing procedures and pinpoint areas for improvement to streamline ground operations.

3. **Flight Cancellation Analysis**: Understanding the underlying causes of flight cancellations is paramount for mitigating disruptions and enhancing passenger experience. This task involves dissecting flight cancellation data to identify common reasons for cancellations, ranging from weather-related factors to operational constraints, enabling airlines to proactively address issues and minimize disruptions.

Structure of Oozie Workflow:



PseudoCode and Algorithm Description:

a. Flights Schedule (On-time):

Pseudocode:

```
Mapper: AirlineScheduleMapper

map(key, value, context):
      information = split value by ","
      if information[0] is not equal to "Year":
          scheduleStatus = "0"
          if information[14] is not equal to "NA":
             delayMinutes = parse information[14] to integer
             if delayMinutes is less than or equal to 10:
                 set scheduleStatus to "1"
             emit (information[8], scheduleStatus) as output
Reducer: AirlineScheduleReducer


      reduce(key, values, context):
          onScheduleCount = 0.0
          totalCount = 0
          for value in values:
              onScheduleCount += parse value to integer
              increment totalCount
          result = onScheduleCount / totalCount
          store result in a map with key as airline code

      cleanup(context):
          sort the map by values in descending order
          emit the top 3 airlines with highest on-time performance
          emit the bottom 3 airlines with lowest on-time performance
          if no data available, emit a message indicating so
```

Algorithm Description:

Mapper:

- The AirlineScheduleMapper class extends the Mapper class provided by Hadoop, and it overrides the map method.
- In the map method, each line of input data is split into fields using a comma (,) delimiter.
- If the first field is not equal to "Year" (i.e., it's not a header row), the mapper checks if the delay in minutes (information[14]) is available and not "NA".
- If the delay is less than or equal to 10 minutes, it sets the schedule status to "1" (indicating on-time) and emits the airline code (information[8]) and the schedule status as output.

Reducer:

- The AirlineScheduleReducer class extends the Reducer class provided by Hadoop, and it overrides the reduce and cleanup methods.
- In the reduce method, it iterates through the values for each key and calculates the average on-time performance (i.e., the ratio of flights that are on schedule) for each airline.
- The results are stored in a TreeMap with the airline code as the key and the average on-time performance as the value.
- In the cleanup method, the TreeMap is sorted by values in descending order.
- The top 3 airlines with the highest on-time performance and the bottom 3 airlines with the lowest on-time performance are emitted as output.
- If there is no data available, a message indicating so is emitted.

b. Taxi time:

PseudoCode:

```
Mapper: AirlineTaxiTimeMapper

map(key, value, context):
    data = split value by ","
    if data[0] is not equal to "Year":
        if data[20] is not equal to "NA":
            emit (data[16], data[20]) as output
        if data[19] is not equal to "NA":
            emit (data[17], data[19]) as output
Reducer: AirlineTaxiTimeReducer


reduce(key, values, context):
    normalMap = new TreeMap
    exceptionMap = new TreeMap
    for value in values:
        temp = parse value to integer
        increment normalCount by temp
        increment totalCount by 1
    result = normalCount / totalCount
    if result equals 0.0:
        add (key, result) to exceptionMap
    else:
        add (key, result) to normalMap

cleanup(context):
    if normalMap is not empty:
        sort normalMap by values in descending order
        emit the top 3 airlines with highest taxi time
        emit the bottom 3 airlines with lowest taxi time
```

```
            emit airlines with zero data (if any)
        else:
            emit "No data available for analysis"
```

Algorithm Description:

Mapper:

- The AirlineTaxiTimeMapper class extends the Mapper class provided by Hadoop, and it overrides the map method.
- In the map method, each line of input data is split into fields using a comma (,) delimiter.
- If the first field is not equal to "Year" (i.e., it's not a header row), the mapper checks if the taxi-in and taxi-out times are available and not "NA".
- If the taxi-in time (data[20]) or taxi-out time (data[19]) is available, it emits key-value pairs where:
- The key is the airline code (data[16] or data[17] depending on the field).
- The value is the taxi time.

Reducer:

- The AirlineTaxiTimeReducer class extends the Reducer class provided by Hadoop, and it overrides the reduce and cleanup methods.
- In the reduce method, it iterates through the values for each key and calculates the average taxi time for each airline.
- The results are stored in two TreeMap objects: normalMap for airlines with non-zero taxi times and exceptionMap for airlines with zero taxi times.
- In the cleanup method, if normalMap is not empty, it sorts normalMap by values in descending order and emits the top and bottom 3 airlines with the highest and lowest taxi times, respectively. It also emits airlines with zero data (if any) from exceptionMap.
- If normalMap is empty, it emits "No data available for analysis".

c. Cancellation:

PseudoCode:

```
Mapper: AirlineCancellationsMapper

map(key, value, context):
    information = split value by ","
    if information[0] is not equal to "Year":
        if information[21] is "1" (indicating a cancellation) and
information[22] is not equal to "NA" and has a non-empty value:
            emit (information[22], 1) as output

Reducer: AirlineCancellationReducer


reduce(key, values, context):
    for value in values:
```

```
        increment total by value
    store total in cancellationReasons map with key as cancellation
reason

cleanup(context):
    if cancellationReasons is not empty:
        sort cancellationReasons by values in descending order
        find the most common cancellation reason
        emit the most common cancellation reason with its count
        if no most common reason found, emit a message indicating so
    else:
        emit a message indicating no data available
```

Algorithm Description:

Mapper:

- The AirlineCancellationsMapper class extends the Mapper class provided by Hadoop, and it overrides the map method.
- In the map method, each line of input data is split into fields using a comma (,) delimiter.
- If the first field is not equal to "Year" (i.e., it's not a header row), the mapper checks if the flight was canceled (information[21] is "1") and if the cancellation reason (information[22]) is available and not "NA" and has a non-empty value.
- If the conditions are met, it emits key-value pairs where:
  - The key is the cancellation reason.
  - The value is 1 (indicating one occurrence of cancellation reason).

Reducer:

- The AirlineCancellationReducer class extends the Reducer class provided by Hadoop, and it overrides the reduce and cleanup methods.
- In the reduce method, it iterates through the values for each key and calculates the total count of cancellations for each cancellation reason.
- The results are stored in a TreeMap called cancellationReasons with the cancellation reason as the key and the total count as the value.
- In the cleanup method, if cancellationReasons is not empty, it sorts cancellationReasons by values in descending order and finds the most common cancellation reason.
- It emits the most common cancellation reason along with its count.
- If no most common reason is found, it emits a message indicating so.
- If cancellationReasons is empty, it emits a message indicating that no data is available.

Performance Measurement plot (VM increase):

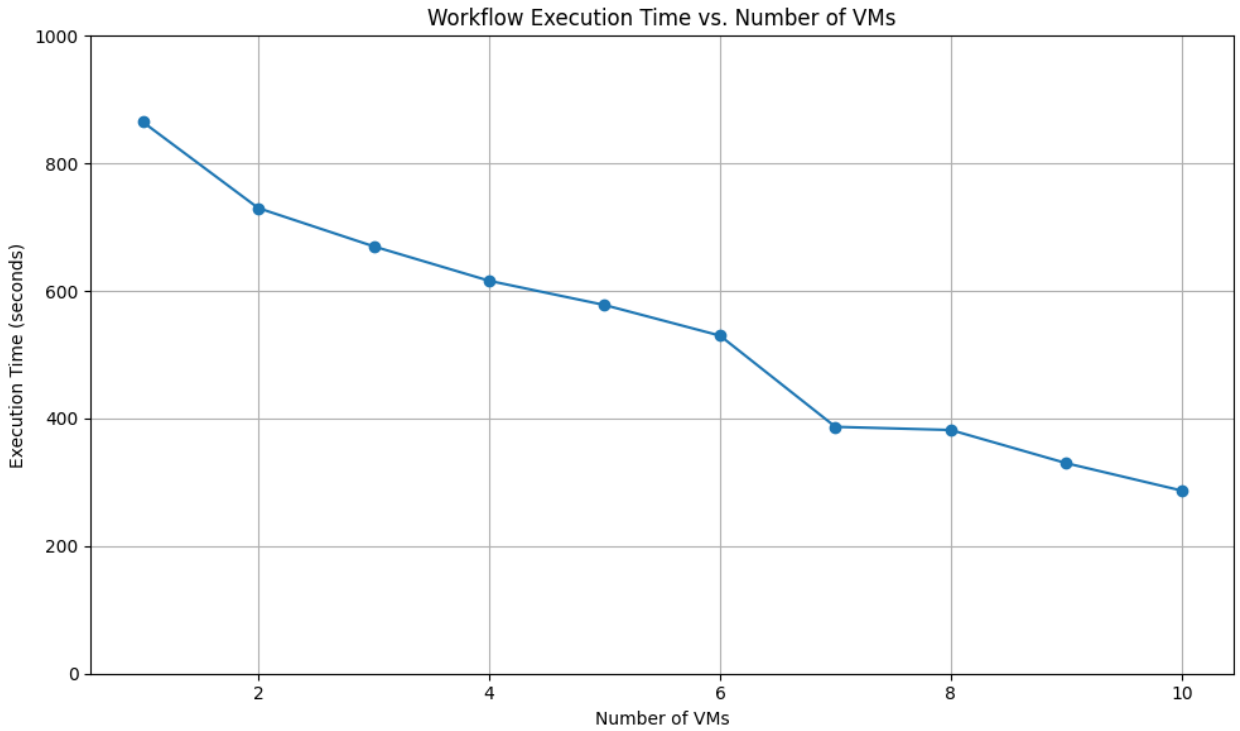


Figure.1

Figure 1 illustrates the interaction between the duration required to execute a workflow and the employment of a variable number of Virtual Machines (VMs) in analyzing a dataset spanning 22 years. It is clearly observed from the graph that there is a significant reduction in the time needed to complete the workflow as the number of VMs increases from one to eight.

This trend suggests an initial substantial improvement in performance with an increase in VMs. However, there is a point beyond which the advantages begin to diminish, indicating that further addition of VMs may not lead to proportional enhancements in performance.

Firstly, the additional overhead in configuring and managing a higher number of VMs may impede the workflow's ability to scale. Additionally, potential conflicts over resources and bottlenecks in the architecture of the Hadoop cluster may limit the scalability of individual MapReduce tasks.

The structure of the dataset, including how the data is distributed, also plays a critical role in the efficiency of parallel processing. An uneven distribution of workload across the VMs can result in underutilization of resources, leading to insufficient performance gains.

In conclusion, though an initial boost in the number of VMs can decrease the workflow completion time, it's important to consider the scalability challenges and characteristics of the workload to optimize performance in large-scale distributed computing environments effectively.

Performance Measurement plot (Dataset increase):

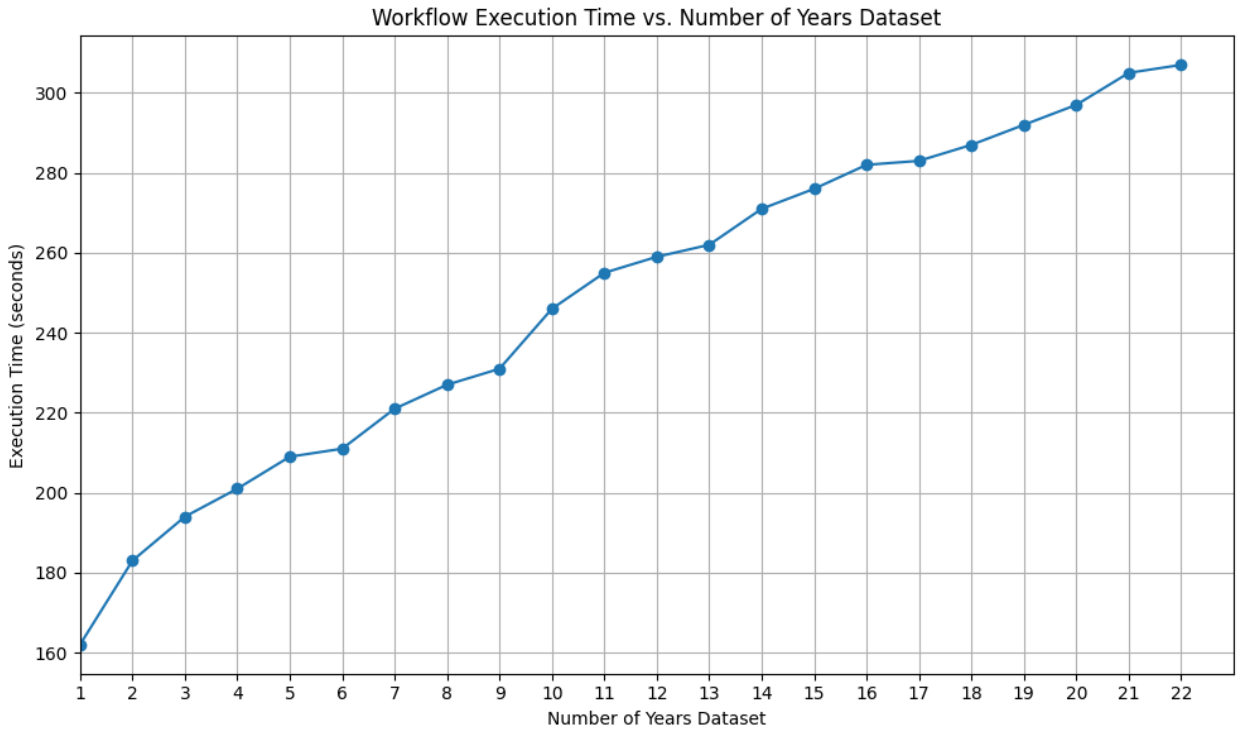


Figure 2

Figure 2 illustrates the link between the duration required to perform workflow tasks and the increasing volume of data over a period of 1 to 22 years. The graph presents a straightforward correlation: as data volume increases, so does the time required for workflow execution.

This pattern suggests that managing larger datasets demands additional computational efforts, ultimately leading to longer execution times. As data volume grows, the workflow must handle a higher quantity of records, resulting in increased processing time and resource consumption.

Several factors contribute to the observed performance patterns. Initially, processing and analyzing larger data volumes requires more time, which stretches the time needed to accomplish workflow tasks. Additionally, the overall performance could be affected by resource contention and scalability constraints, especially in environments that utilize distributed computing.

Furthermore, the complexity of data processing tasks, such as data merging, aggregating, and altering, influences workflow performance. As datasets grow, these tasks may require more computational power, potentially decelerating execution times.

In conclusion, while the enlargement of data is essential for comprehensive analysis, recognizing the associated performance costs is crucial. Understanding the relationship between data volume and workflow execution time is key to optimizing workflow configurations and resource allocation in scenarios involving large-scale data processing.

Output Workflow: