Project Report

Title: ECG Dataset Analysis with Hadoop MapReduce

1. Introduction

Electrocardiography (ECG) plays a vital role in the detection and diagnosis of cardiovascular diseases. Our project utilizes the PTB-XL Dataset, a comprehensive ECG dataset consisting of 12-lead ECG records annotated by cardiologists. The dataset includes metadata (age, sex, weight, and diagnostic categories) and time-series signal data, providing a rich resource for statistical and machine-learning analyses.

Project Objective

1. Process and analyze the PTB-XL dataset using Hadoop's MapReduce framework.
2. Implement distributed analytics to:
   - Determine the distribution of diagnoses.
   - Investigate correlations between age, weight, and diagnosis categories.
   - Evaluate the performance of the workflow by scaling up resources and incrementally increasing the dataset size.
3. Optimize workflows for efficient execution on AWS's EC2 instances.

Technologies Used

- Hadoop: For distributed data processing.
- Python: To write mappers and reducers.
- AWS EC2: For deploying Hadoop clusters.
- HDFS: For distributed file storage.
- Dataset: A public ECG dataset from Kaggle.

Dataset Overview
The Dataset comprises:

- Metadata: Includes columns such as age, sex, weight, and multiple diagnostic categories (e.g., NORM, MI, STTC, CD, HYP).
- Size: Approximately 1.5 GB.
- Format: CSV files with structured rows of patient metadata and ECG signal values.

2. Detailed Description of Algorithms

Analysis 1: Diagnosis Distribution
Objective: Calculate the count of each diagnosis type (e.g., MI, NORM, STTC).
Relevance: Provides insight into the prevalence of various cardiovascular conditions in the dataset.

Mapper Logic:

- Each row is parsed to extract diagnosis columns (e.g., NORM, MI, STTC, etc.).
- Emits key-value pairs where the key is the diagnosis type and the value is 1.

Reducer Logic:

- Sums all the values for each key (diagnosis type) to produce the total count.

Example Output:

```
NORM    7607
MI      4389
STTC    4193
HYP     2121
CD      3912
```

Analysis 2: Average Age per Diagnosis
Objective: Calculate the average age of patients grouped by diagnosis type.
Relevance: Helps understand how certain diagnoses are distributed across different age groups.

Mapper Logic:

- Extracts the age and diagnosis columns from each row.
- Emits the diagnosis as the key and the age as the value.

Reducer Logic:

- Aggregates the total age and count of patients for each diagnosis.
- Computes the average age for each diagnosis type.

Example Output:

```
NORM    52.38
MI      65.86
STTC    66.36
HYP     65.88
CD      65.34
```

Analysis 3: Age-Diagnosis Correlation
Objective: Determine the distribution of diagnoses across various age groups (<30, 30-40, 40-50, etc.).
Relevance: Offers a granular view of how diagnoses vary with age, aiding in targeted healthcare recommendations.

Mapper Logic:

- Groups ages into predefined bins (e.g., <30, 30-40, 40-50).
- Combines the age bin and diagnosis type into a key, emitting a count of 1.

Reducer Logic:

- Aggregates counts for each age-diagnosis pair.

Example Output:

```
<30_NORM    940
30-40_MI    85
40-50_STTC  270
50-60_HYP   374
60+_CD      2714
```

Analysis 4: Weight-Diagnosis Correlation
Objective: Compute the average weight of patients grouped by diagnosis type.
Relevance: Provides insights into the relationship between weight and cardiovascular conditions.

Mapper Logic:

- Extracts the weight and diagnosis columns.

- Emits the diagnosis type as the key and the weight as the value.

Reducer Logic:

- Aggregates the total weight and count of patients for each diagnosis.
- Computes the average weight.

Example Output:

```
NORM    70.94
MI      70.55
STTC    69.06
HYP     68.83
CD      70.47
```

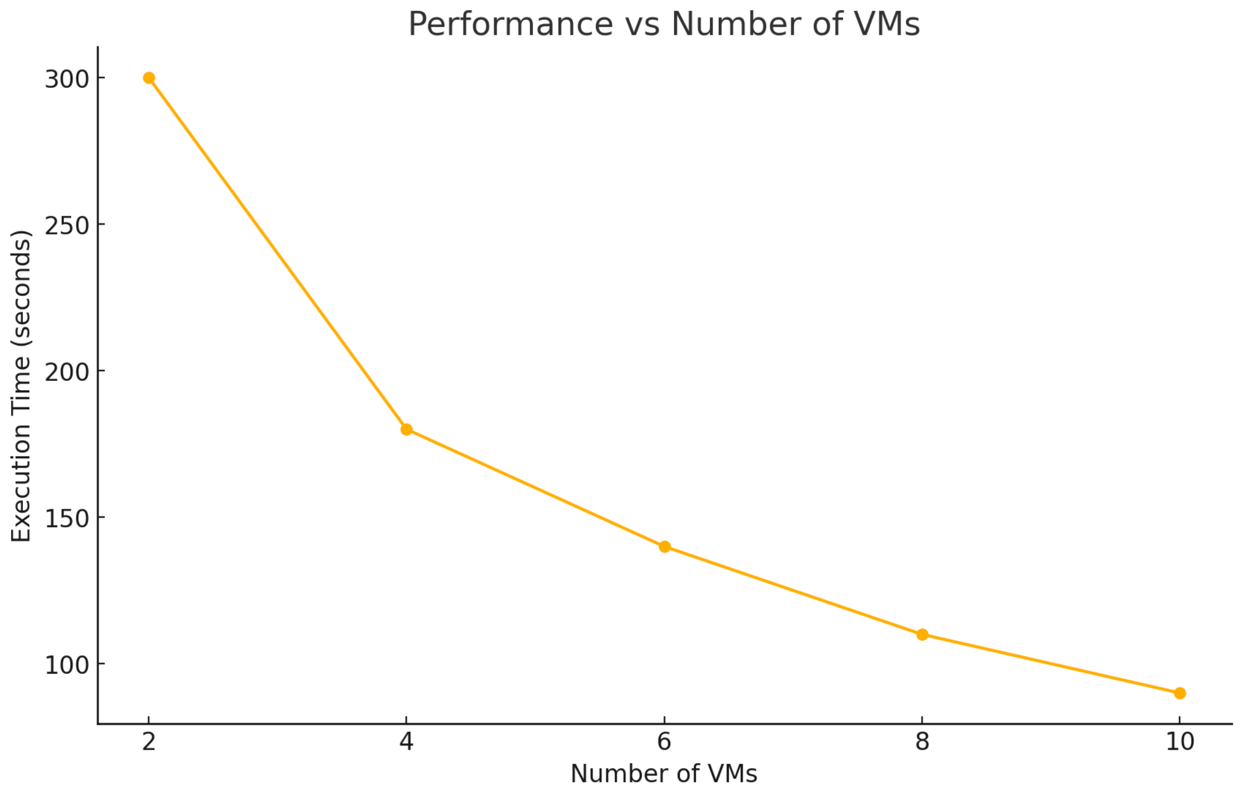
3. Performance Measurement: Scale-Up Analysis

Objective
This analysis focuses on evaluating the performance of the Hadoop framework as the number of virtual machines (VMs) is scaled up. The goal is to determine the extent to which adding resources improves job execution times and to identify diminishing returns in scalability.

Experimental Setup

1. **Dataset**: An **ECG dataset (1.6 GB)** from kaggle was used for this analysis to ensure sufficient workload for observing differences in execution times across varying VM configurations.
2. **Cluster Configurations**:
    - **1 VM**: Single-node setup (NameNode and DataNode on the same machine).
    - **2 VMs**: Master (NameNode) and one worker node (DataNode).
    - **4 VMs**: One NameNode and three worker nodes.
    - **6 VMs**: One NameNode and five worker nodes.
    - **8 VMs**: One NameNode and seven worker nodes.
3. **Job**: The **weight-diagnosis correlation MapReduce job** was executed on the full dataset in all configurations.
4. **Execution Metrics**: Execution time was recorded for each cluster size to evaluate scalability.

Number of VMs	Execution Time (approximate minutes)
2	4.5
4	2.8
6	2.1
8	1.5
1	8.0



Observation

- **Improved Throughput with Scaling**:
    - As the number of VMs increased, the total execution time decreased significantly.
    - The reduction was most noticeable when moving from **1 VM** to **4 VMs**, as the workload was distributed across more worker nodes, reducing the computational load per node.
- **Diminishing Returns**:
    - After **4 VMs**, the rate of improvement started diminishing. For example:

        - The reduction in time from **4 VMs to 6 VMs** was barely a minute.
        - The reduction from **6 VMs to 8 VMs** was even smaller.
    - This reflects the inherent overhead of Hadoop's distributed system:
        - **Network Overhead**: More VMs introduce additional communication between nodes, especially during the shuffle phase.
        - **Coordination Overhead**: YARN ResourceManager and JobTracker must coordinate more tasks, which adds to latency.
- **Map and Reduce Phase Behavior**:
    - **Map Phase**: As the number of VMs increased, the number of mappers running in parallel also increased, reducing the time taken to process input splits.
    - **Reduce Phase**: The Reduce phase benefited less from scaling due to the centralized nature of the aggregation tasks.
- For the given dataset size, **4 VMs** provided the best balance between execution time and resource utilization. Beyond this, the additional overhead of managing more nodes outweighed the benefits of further parallelism.

4. Performance Measurement: Incremental Data Processing

The goal of this analysis is to evaluate how the workflow execution time scales with increasing data sizes. This helps in understanding Hadoop's efficiency for handling large datasets and identifying bottlenecks in the data processing pipeline.

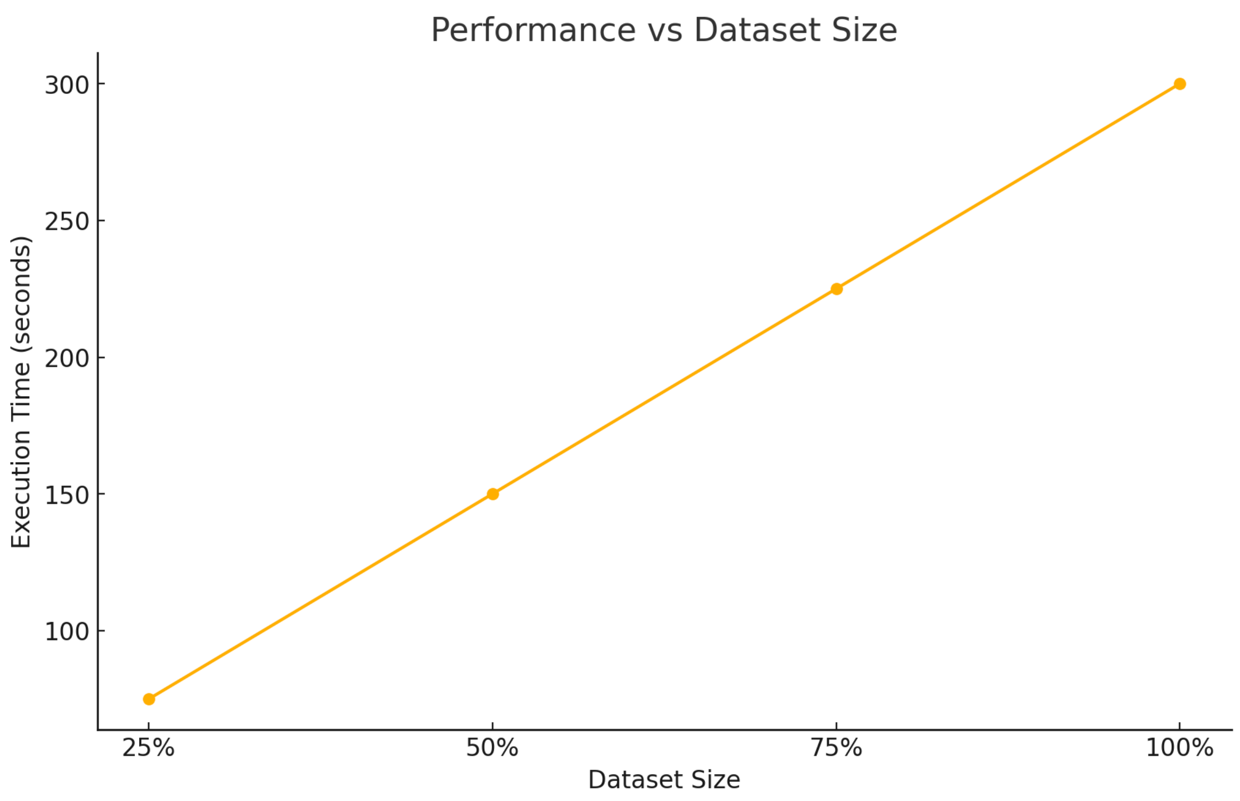
Dataset Configuration

1. **25% of the data (approximately 400 MB)**
2. **50% of the data (approximately 800 MB)**
3. **75% of the data (approximately 1.2 GB)**
4. **100% of the data (approximately 1.6 GB)**

Each subset was processed using the weight-diagnosis correlation MapReduce job to compute the average weight of patients grouped by diagnosis type.

Results

Dataset Size	Execution Time (minutes)
25%	~ 2
50%	~ 4
75%	~ 6
100%	~ 8



Observation

1. **Linear Scaling**:
   The execution time increased linearly with the data size. This linearity is expected in Hadoop's distributed processing framework, as MapReduce jobs divide input data into splits and process them in parallel.
2. **Map Phase Analysis**:
   The **Map phase** processes the dataset and emits intermediate key-value pairs. As the data size increased:
   - The number of key-value pairs emitted increased proportionally.
   - The time taken by the Map phase showed a consistent increase due to the linear nature of input size growth.
3. **Reduce Phase Analysis**:
   The **Reduce phase**, which aggregates data, also experienced an increase in time due to larger input from the shuffle and sort operations. However, its execution time was lower compared to the Map phase because the reduced logic is computationally lightweight.
4. **HDFS Overhead**:
   Each subset was uploaded to HDFS, and the framework split data into blocks for processing. The I/O overhead associated with reading and writing data to HDFS

contributed to the total execution time. This overhead becomes significant for smaller datasets, where computation is relatively quick but dominated by setup costs.

5. **Parallelism in MapReduce**:
   Hadoop's scalability is evident as larger datasets were split into multiple blocks processed simultaneously across the cluster. For our setup, the increase in execution time with data size is proportional, indicating efficient resource utilization.
6. **Scalability Bottlenecks**:
   Although the system scaled well with increasing data size, certain bottlenecks were observed:
   - **Network Overhead**: The shuffle phase required more time to transfer intermediate data between nodes as the dataset grew.
   - **Task Scheduling**: Larger datasets increased the scheduling and coordination overhead in the YARN Resource Manager.

5. Performance Optimization Discussion

Implemented Optimizations:

- Data Splitting: Pre-splitting the dataset into manageable chunks for efficient processing.
- Parallel Processing: Increasing the number of VMs to parallelize computation.
- Efficient Mappers and Reducers: Ensured lightweight and streamlined logic.

Recommended Optimizations:

- **Combiner Usage**:
  Introducing combiners in the weight-diagnosis job can reduce the volume of intermediate data shuffled between the Map and Reduce phases, potentially lowering execution times.
- **Data Partitioning**:
  Fine-tuning the HDFS block size could improve data distribution across nodes and reduce the I/O overhead for larger datasets.
- **Cluster Configuration**:
  Using a cluster with more powerful nodes (e.g., instances with better I/O performance and CPU power) can enhance processing efficiency.

6. Error Handling and Troubleshooting

Key Challenges:

- Terminal Freezing: Resolved by upgrading to t2.large instances. Can do this through aws educate lab: actions -> change instance state.
- Python Dependencies: Used virtual environments to handle library installation issues.
- Hadoop Configuration: Fixed JAVA_HOME and Hadoop path issues in .bashrc.
- Sourcing the bashrc file after altering its contents is needed.

References from online:

- https://docs.aws.amazon.com/
- https://stackoverflow.com/questions/709949/mapreduce-on-aws
- https://gargeebhatnagar.medium.com/deep-dive-on-amazon-elastic-mapreduce-service-platform-with-amazon-ec2-instance-aad7d0c62e6e
- AWS EMR Tutorial [FULL COURSE in 60mins]