# Cloud Computing Project 3 Report

## PROJECT DETAILS

Student Name: Tara Acheson
Student ID : 17376856
Project Title : Hadoop/MapReduce implementation for a medical application
CC Project: 2
Due Date : 06/12/2024
Submitted Date :

## Application overview.

### Objectives:

1) Identify the medical dataset that you would like to analyse.

The medical dataset chosen for this project was “UCI Heart Disease Data” from kaggle.com (“UCI Heart Disease Data,” n.d.).

2) Clearly define the analysis you want to perform on the medical dataset (e.g.,automatic diagnosis, symptom-disorder matching, medical report classification, etc.).

I chose to build a predictive model that identifies whether a patient within a certain age-range has a higher risk of heart disease (binary outcome).

3) Choose the algorithm that you will use to implement your use case.

For this predictive model, I chose to implement a bucket-based clustering approach for feature grouping, followed by MapReduce aggregation for analysis. The goal is to process and categorise medical data to predict heart disease risk based on patient demographics and clinical measurements. This method focuses on grouping data based on predefined ranges (buckets) for age and cholesterol levels, and then using a distributed data processing framework (MapReduce) to compute summary statistics and identify high-risk groups.

4) Implement the selected algorithm in Hadoop/MapReduce.

This was done using Docker to run a local Hadoop environment for testing and development. The Docker image I used was a pre-configured Hadoop container, and I ran it locally to execute the MapReduce jobs.

5) Build a Dashboard to run your analysis and display the results.

I built a web-based dashboard using Flask and HTML to display the results of the MapReduce analysis.

## Problem Context and Dataset Collection

### Problem:

Heart disease is a leading cause of mortality worldwide, and early prediction can significantly improve outcomes by enabling timely medical interventions. The problem addressed in this project is predicting the likelihood of heart disease in patients based on various clinical and demographic factors. This predictive model serves as a valuable tool for identifying high-risk individuals, guiding clinicians, and improving preventive healthcare strategies.

The objective is to predict whether a patient is at risk of heart disease using a binary classification approach. The predictive model is trained on features such as age, sex, cholesterol levels, and other clinical measurements, which are known to correlate with heart disease.

### Dataset:

The dataset used for this project is the "UCI Heart Disease Data," sourced from Kaggle.com. It contains clinical and demographic data from patients that can be used to predict heart disease.

This dataset is highly relevant for medical analytics due to its rich set of clinically significant features. It allows for analysing patient data patterns, understanding risk factors, and building predictive models to support clinical decision-making. Heart disease prediction models can help:

- Improve diagnostics: Assist physicians in identifying high-risk patients.
- Enhance preventive care: Enable timely interventions for individuals at risk.
- Support resource allocation: Prioritize healthcare resources for individuals most in need.

## Methodology and Approach to Implementation

The methodology for this project is centered around efficiently processing large-scale medical data to predict heart disease risk based on key patient features. The process involves three primary steps: data preprocessing, bucket-based clustering for feature grouping, and MapReduce-based analysis.

### Data Preprocessing and Feature Grouping

I preprocessed the data to ensure that the dataset was cleaned, consistent, and well-structured for the Mapper function. This allowed the MapReduce algorithm to focus on analysing the distribution of heart disease cases across age and cholesterol buckets without handling raw data inconsistencies.
I created two bucket definitions for grouping data:

- Age Buckets (age_bucket): Categorise patients into predefined age ranges (e.g., 20-30, 31-40, etc.).
- Cholesterol Buckets (chol_bucket): Group patients based on cholesterol levels into medically relevant ranges (e.g., 100-199, 200-239, etc.).

I then extracted only the necessary columns:

- id: Patient identifier.
- age: Original age for reference.

- age_bucket: Grouped age ranges.
- chol: Original cholesterol levels.
- chol_bucket: Grouped cholesterol ranges.
- num: Indicates heart disease status (numeric).

Lastly I transformed the target feature (num) by mapping to a binary indicator (heart_disease) which displayed 1 if heart disease was present (num >0) and 0 otherwise. The preprocessed dataset was then saved as preprocessed_heart_disease_data.csv for input into the Hadoop MapReduce framework

### MapReduce Framework Implementation

The implementation consists of two components: the Mapper and the Reducer.

The Mapper processes input data to emit key-value pairs where the key is a combination of age_bucket and chol_bucket, and the value is the binary diagnosis(<age_bucket,chol_bucket>\t<heart_disease>).

The Reducer aggregates results for each key (age-cholesterol bucket) to compute summary statistics. The output statistics for each key include the total count, heart disease count, and percentage.

### Modifications for Hadoop

- Data preprocessing: Data was formatted as key-value pairs for MapReduce processing. Features like age and cholesterol were bucketed into predefined ranges for easier grouping.
- Mapper and Reducer adaptations: The Mapper emitted the buckets as keys, while the Reducer computed the aggregate statistics for each bucket. Since this approach does not require iterative training models (like logistic regression), the focus was solely on data aggregation and statistical analysis.
- Docker-based setup: Hadoop was run within a Docker container for portability and ease of local testing.

## Why Hadoop/MapReduce is Suitable for This Project

MapReduce is suitable for processing large medical datasets, such as this heart disease prediction dataset, due to its ability to handle large-scale data efficiently. It divides the processing task into smaller chunks that can be executed in parallel across multiple nodes in a distributed environment. This parallelisation makes it possible to process vast amounts of patient data in a fraction of the time compared to traditional methods. Additionally, MapReduce's scalability ensures that as the dataset grows, more computational resources can be added to maintain performance.

Another key advantage of MapReduce is its simplicity and fault tolerance. The model is easy to understand, with the Mapper handling data transformation and the Reducer performing aggregation, making it straightforward to implement for complex medical data analysis tasks. In a distributed system, MapReduce automatically handles node failures by reassigning tasks to other nodes, ensuring that the processing continues without interruptions - an essential feature when working with critical medical data.

Finally, MapReduce is well-suited for batch processing tasks, such as this project where I aggregate and analyse patient data based on specific features like age and cholesterol levels. Its ability to process heterogeneous data types and optimize for data locality makes it ideal for processing structured medical data efficiently.

## Software Features and Functionalities

This software system is designed to process medical data (specifically predicting heart disease risk based on patient features)
The system integrates multiple components, including data processing using Hadoop, analysis with MapReduce, and visualization through a web-based dashboard. Below are the key features of the software:

- Data Processing in Hadoop: The software uses Hadoop to handle large-scale medical datasets efficiently. Data is processed in a distributed environment, where the dataset can be divided into smaller chunks and processed in parallel across multiple nodes. This allows for the efficient handling of vast amounts of patient data, enabling faster computations. The data preprocessing step involves bucketing key features (age and cholesterol) into predefined ranges for easier group-based analysis. This pre-processing is handled by the Mapper function, which emits key-value pairs based on these buckets, while the Reducer aggregates the results to calculate summary statistics.
- Analysis Using MapReduce: The software's analysis is built on the MapReduce framework. The Mapper processes each data point, extracting relevant features and creating key-value pairs. The Reducer aggregates these key-value pairs, calculating the total number of patients in each group and the number diagnosed with heart disease. The reducer then computes the percentage of heart disease cases within each group, which helps in identifying high-risk populations. This analysis is fully distributed, allowing for efficient processing of large datasets.
- Visualization in the Dashboard: The results of the MapReduce analysis are displayed through an interactive dashboard built with Flask and HTML. The dashboard showcases key metrics, such as the total number of patients in each group, the number diagnosed with heart disease, and the percentage of heart disease cases within each group.

## Worked Example of MapReduce Job Execution

The entire job has been designed to run on one script, navigate to the root directory and run:
- ./run_hadoop_streaming.sh

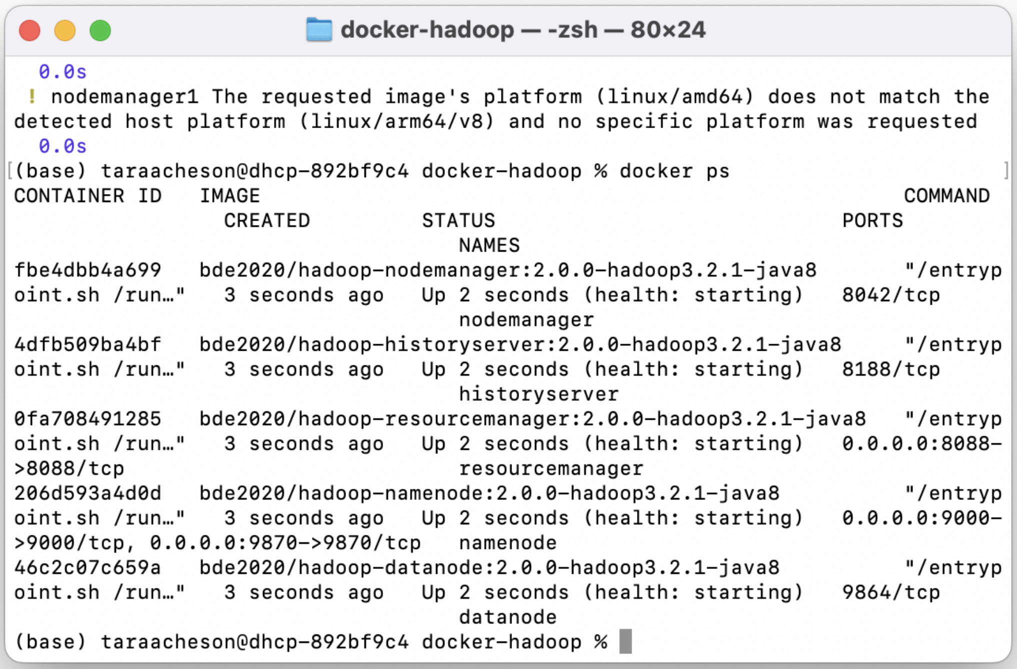
This may fail if the data files are not in the correct directory, the following instructions can also be followed to run the jobs manually.

### Start the Hadoop cluster

Navigate to the “docker-hadoop” directory within the project folder.
Run “docker-compose up -d” to start the Hadoop cluster using Docker Compose. This will bring up the following services:
- Namenode: Manages the HDFS filesystem.
- Datanode: Stores the data in HDFS.
- ResourceManager: Manages the YARN cluster resources.
- NodeManager: Manages the execution of MapReduce jobs.
- HistoryServer: Provides historical information about MapReduce jobs.

Check the containers are running using “docker ps”. This step might take a few minutes as the containers start up. You can also access the Hadoop Web Interfaces and check the status of your cluster, YARN resource usage, and Hadoop file system from here:

- Namenode UI: http://localhost:9870
- ResourceManager UI: http://localhost:8088


docker-hadoop — -zsh — 80×24

```
  0.0s
 ! nodemanager1 The requested image's platform (linux/amd64) does not match the
detected host platform (linux/arm64/v8) and no specific platform was requested
  0.0s
(base) taraacheson@dhcp-892bf9c4 docker-hadoop % docker ps
CONTAINER ID   IMAGE                                                    COMMAND
                 CREATED          STATUS                                PORTS
                                    NAMES
fbe4dbb4a699   bde2020/hadoop-nodemanager:2.0.0-hadoop3.2.1-java8       "/entryp
oint.sh /run…"   3 seconds ago    Up 2 seconds (health: starting)   8042/tcp
                                    nodemanager
4dfb509ba4bf   bde2020/hadoop-historyserver:2.0.0-hadoop3.2.1-java8     "/entryp
oint.sh /run…"   3 seconds ago    Up 2 seconds (health: starting)   8188/tcp
                                    historyserver
0fa708491285   bde2020/hadoop-resourcemanager:2.0.0-hadoop3.2.1-java8   "/entryp
oint.sh /run…"   3 seconds ago    Up 2 seconds (health: starting)   0.0.0.0:8088-
>8088/tcp                           resourcemanager
206d593a4d0d   bde2020/hadoop-namenode:2.0.0-hadoop3.2.1-java8          "/entryp
oint.sh /run…"   3 seconds ago    Up 2 seconds (health: starting)   0.0.0.0:9000-
>9000/tcp, 0.0.0.0:9870->9870/tcp   namenode
46c2c07c659a   bde2020/hadoop-datanode:2.0.0-hadoop3.2.1-java8          "/entryp
oint.sh /run…"   3 seconds ago    Up 2 seconds (health: starting)   9864/tcp
                                    datanode
(base) taraacheson@dhcp-892bf9c4 docker-hadoop %
```

## Submit the MapReduce Job to Hadoop

If the files are not already present in the Hadoop File System, they can be copied using the following commands:

- docker cp preprocessed_heart_disease_data.csv namenode:/
- docker exec -it namenode /bin/bash
- hdfs dfs -mkdir /heart_data
- hdfs dfs -put preprocessed_heart_disease_data.csv /heart_data/
- hdfs dfs -ls /heart_data

You should see the file listed in /heart_data.
The mapper.py and reducer.py scripts can be copied to the container using:

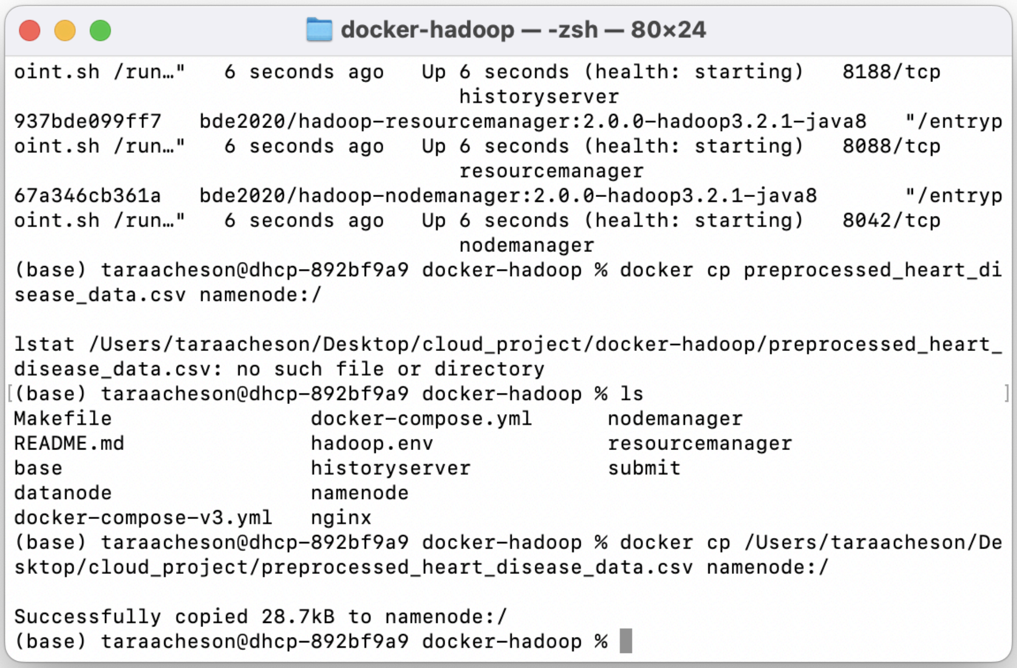
- docker cp mapper.py namenode:/
- docker cp reducer.py namenode:/

Verify the files are inside the NameNode container using:

    - docker exec -it namenode /bin/bash
    - ls /

Ensure the scripts have execution permission by running:

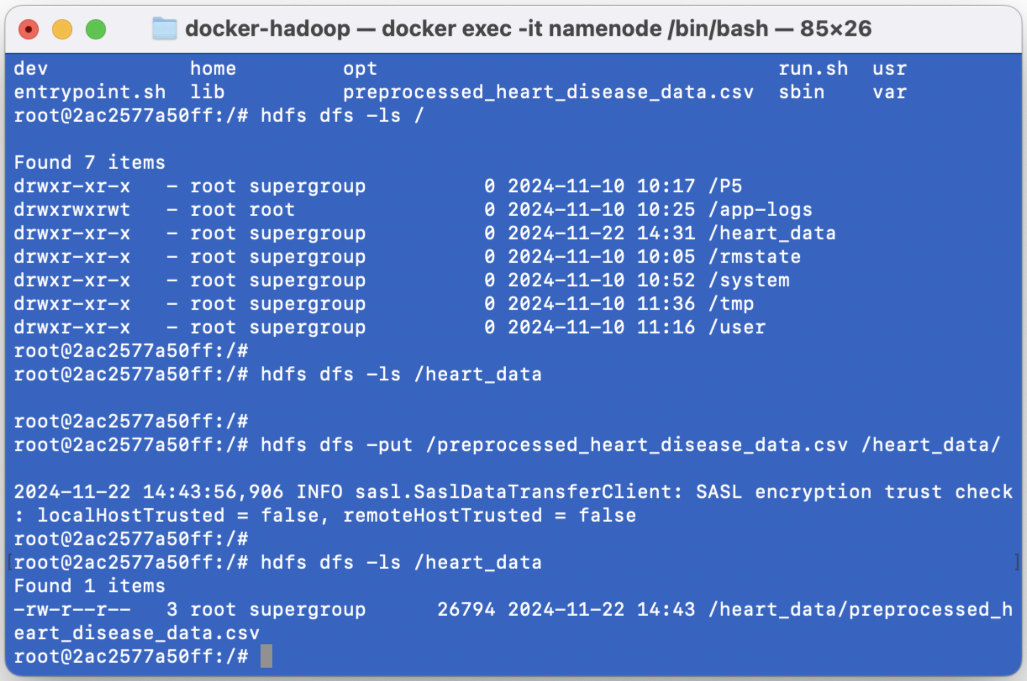
- chmod +x mapper.py reducer.py



```
oint.sh /run…"   6 seconds ago   Up 6 seconds (health: starting)   8188/tcp
                                    historyserver
937bde099ff7   bde2020/hadoop-resourcemanager:2.0.0-hadoop3.2.1-java8   "/entryp
oint.sh /run…"   6 seconds ago   Up 6 seconds (health: starting)   8088/tcp
                                    resourcemanager
67a346cb361a   bde2020/hadoop-nodemanager:2.0.0-hadoop3.2.1-java8       "/entryp
oint.sh /run…"   6 seconds ago   Up 6 seconds (health: starting)   8042/tcp
                                    nodemanager
(base) taraacheson@dhcp-892bf9a9 docker-hadoop % docker cp preprocessed_heart_di
sease_data.csv namenode:/

lstat /Users/taraacheson/Desktop/cloud_project/docker-hadoop/preprocessed_heart_
disease_data.csv: no such file or directory
(base) taraacheson@dhcp-892bf9a9 docker-hadoop % ls
Makefile                docker-compose.yml      nodemanager
README.md               hadoop.env              resourcemanager
base                    historyserver           submit
datanode                namenode
docker-compose-v3.yml   nginx
(base) taraacheson@dhcp-892bf9a9 docker-hadoop % docker cp /Users/taraacheson/De
sktop/cloud_project/preprocessed_heart_disease_data.csv namenode:/

Successfully copied 28.7kB to namenode:/
(base) taraacheson@dhcp-892bf9a9 docker-hadoop %
```



```
dev              home         opt                                   run.sh  usr
entrypoint.sh    lib          preprocessed_heart_disease_data.csv   sbin    var
root@2ac2577a50ff:/# hdfs dfs -ls /

Found 7 items
drwxr-xr-x   - root supergroup          0 2024-11-10 10:17 /P5
drwxrwxrwt   - root root                0 2024-11-10 10:25 /app-logs
drwxr-xr-x   - root supergroup          0 2024-11-22 14:31 /heart_data
drwxr-xr-x   - root supergroup          0 2024-11-10 10:05 /rmstate
drwxr-xr-x   - root supergroup          0 2024-11-10 10:52 /system
drwxr-xr-x   - root supergroup          0 2024-11-10 11:36 /tmp
drwxr-xr-x   - root supergroup          0 2024-11-10 11:16 /user
root@2ac2577a50ff:/#
root@2ac2577a50ff:/# hdfs dfs -ls /heart_data

root@2ac2577a50ff:/#
root@2ac2577a50ff:/# hdfs dfs -put /preprocessed_heart_disease_data.csv /heart_data/

2024-11-22 14:43:56,906 INFO sasl.SaslDataTransferClient: SASL encryption trust check
: localHostTrusted = false, remoteHostTrusted = false
root@2ac2577a50ff:/#
root@2ac2577a50ff:/# hdfs dfs -ls /heart_data
Found 1 items
-rw-r--r--   3 root supergroup      26794 2024-11-22 14:43 /heart_data/preprocessed_h
eart_disease_data.csv
root@2ac2577a50ff:/#
```

## Run the MapReduce job using Hadoop Streaming:

To ensure compatibility with the older Debian-based Hadoop cluster images this project was built on, you need to update the package repositories inside the container. This is required because the images were built on an outdated version of Debian.

Start by running the following command inside the container to update the sources list:

- echo -e "deb http://archive.debian.org/debian/ stretch main\n\
- deb-src http://archive.debian.org/debian/ stretch main\n\
- deb http://archive.debian.org/debian-security/ stretch/updates main\n\
- deb-src http://archive.debian.org/debian-security/ stretch/updates main" > /etc/apt/sources.list

After updating the sources, run the following commands to update the package lists and install Python 3:

- apt-get update
- apt-get install python3

Inside the NameNode container, run:

“

```
hadoop jar /opt/hadoop-3.2.1/share/hadoop/tools/lib/hadoop-streaming-3.2.1.jar \
  -input /heart_data/preprocessed_heart_disease_data.csv \
  -output /heart_data_output \
  -mapper "python3 mapper.py" \
  -reducer "python3 reducer.py" \
  -file mapper.py \
  -file reducer.py
```

“

Delete the output directory if it exists

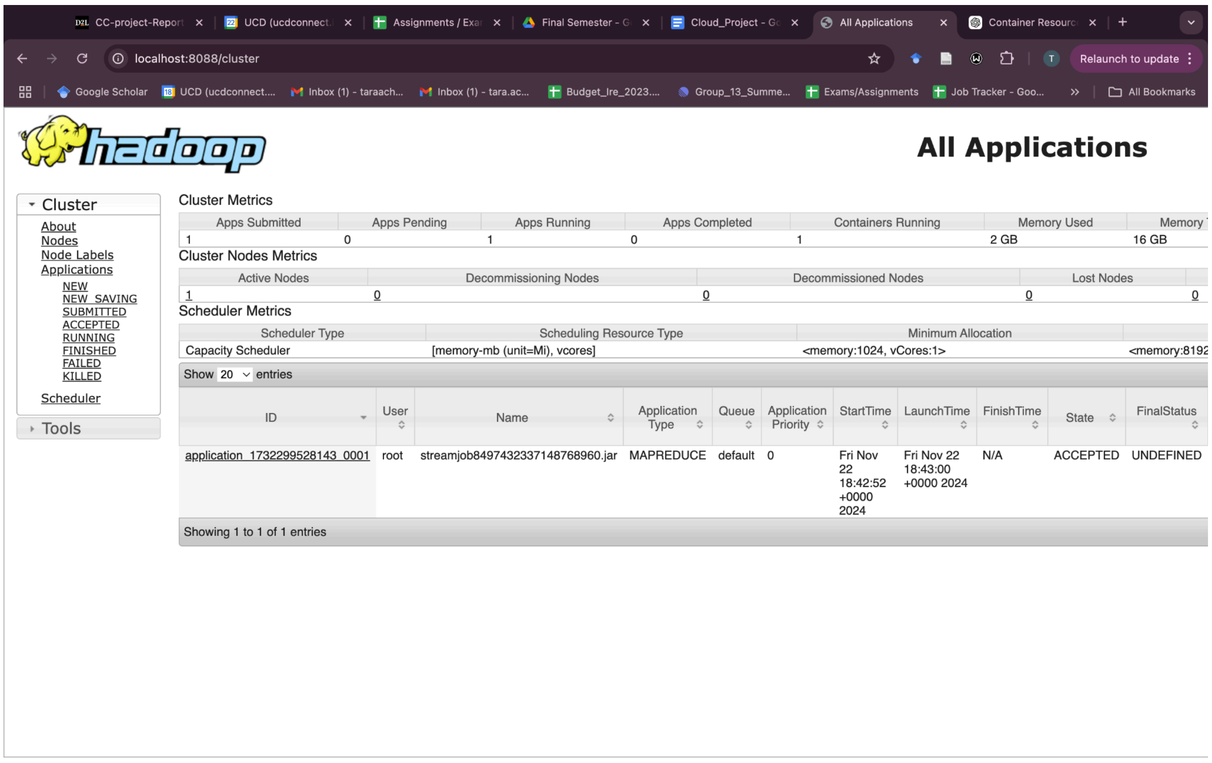
- hadoop fs -rm -r /heart_data_output

Verify the output once the job is completed by running:

- hdfs dfs -ls /heart_data/output

View the results:

- hdfs dfs -cat /heart_data/output/part-00000



**All Applications**

Cluster Metrics

| Apps Submitted | Apps Pending | Apps Running | Apps Completed | Containers Running | Memory Used | Memory |
|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 1 | 2 GB | 16 GB |

Cluster Nodes Metrics

| Active Nodes | Decommissioning Nodes | Decommissioned Nodes | Lost Nodes |
|---|---|---|---|
| 1 | 0 | 0 | 0 |

Scheduler Metrics

| Scheduler Type | Scheduling Resource Type | Minimum Allocation | |
|---|---|---|---|
| Capacity Scheduler | [memory-mb (unit=Mi), vcores] | <memory:1024, vCores:1> | <memory:8192 |

Show 20 entries

| ID | User | Name | Application Type | Queue | Application Priority | StartTime | LaunchTime | FinishTime | State | FinalStatus |
|---|---|---|---|---|---|---|---|---|---|---|
| application_1732299528143_0001 | root | streamjob8497432337148768960.jar | MAPREDUCE | default | 0 | Fri Nov 22 18:42:52 +0000 2024 | Fri Nov 22 18:43:00 +0000 2024 | N/A | ACCEPTED | UNDEFINED |

Showing 1 to 1 of 1 entries

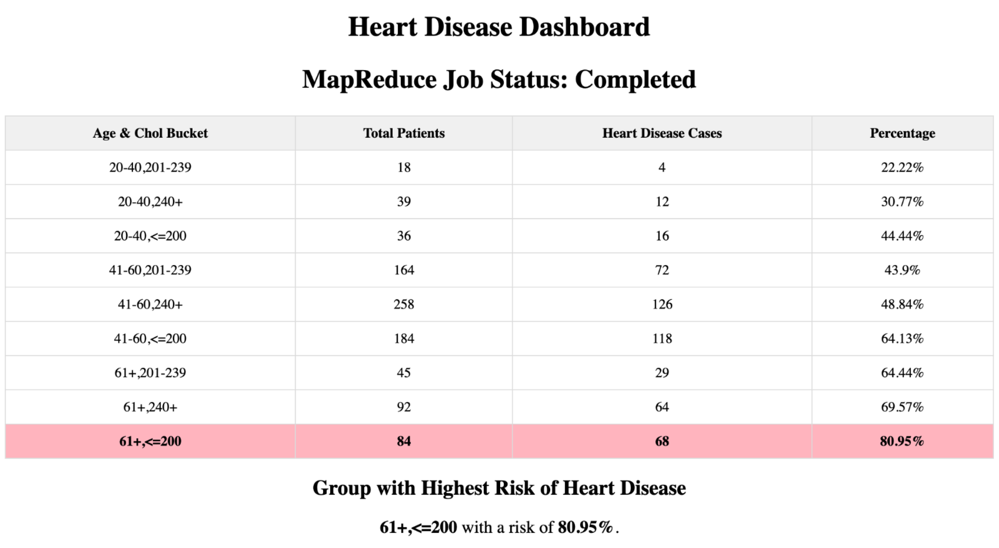
## Displaying Results for Dashboard

The Flask app queries the YARN ResourceManager to retrieve the status of the MapReduce job. Ensure that the ResourceManager is running and accessible at http://localhost:8088.

In the app directory, run the Flask app:

- python app.py

Open a web browser and go to http://localhost:5001. This will show the job status on the dashboard and show the resulting buckets once completed.



**Heart Disease Dashboard**

**MapReduce Job Status: Completed**

| Age & Chol Bucket | Total Patients | Heart Disease Cases | Percentage |
|---|---|---|---|
| 20-40,201-239 | 18 | 4 | 22.22% |
| 20-40,240+ | 39 | 12 | 30.77% |
| 20-40,<=200 | 36 | 16 | 44.44% |
| 41-60,201-239 | 164 | 72 | 43.9% |
| 41-60,240+ | 258 | 126 | 48.84% |
| 41-60,<=200 | 184 | 118 | 64.13% |
| 61+,201-239 | 45 | 29 | 64.44% |
| 61+,240+ | 92 | 64 | 69.57% |
| **61+,<=200** | **84** | **68** | **80.95%** |

**Group with Highest Risk of Heart Disease**

**61+,<=200** with a risk of **80.95%**.

## Conclusion

By categorizing patients into age and cholesterol buckets, the analysis provided valuable insights into how different demographic groups are affected by heart disease. For example, the highest risk groups, such as individuals aged 61+ with cholesterol levels over 240, showed significantly higher percentages of heart disease cases (up to 80.95%).
The software enables a group-based risk assessment, allowing healthcare professionals to easily identify high-risk populations and focus preventive measures on them, improving healthcare decision-making and resource allocation.
The MapReduce framework, combined with Hadoop proved highly effective for handling large-scale medical datasets. By leveraging the system's ability to process and aggregate data across multiple nodes, it delivers faster results compared to single-machine processing. The distributed nature of Hadoop also ensures scalability, allowing the system to accommodate increasing volumes of medical data, making it well-suited for future growth as healthcare data continues to expand.

### Potential for Future Improvements:

- Real-Time Data Processing: Currently, the system processes static data. Future improvements could focus on enabling real-time data processing, where new patient records are automatically added to the analysis as they become available.
- More Detailed Risk Factors: The current model uses only age and cholesterol levels for risk assessment. Incorporating additional factors, such as lifestyle data (e.g., smoking, physical activity, diet), medical history, and genetic predisposition, could lead to more accurate and personalized heart disease risk predictions.
- Advanced Machine Learning Algorithms: Moving from rule-based categorization (age and cholesterol buckets) to machine learning models (such as logistic regression or decision trees) could improve the accuracy of predictions and help in identifying complex, nonlinear patterns that might not be captured through traditional bucketing.
- User Customization: Allowing users to customize the risk buckets based on their own criteria could make the tool more flexible and useful to a wider range of healthcare settings and research applications.

The software created for this project effectively integrates distributed data processing with visual analytics to provide valuable insights into heart disease risk among different demographic groups. The use of Hadoop's MapReduce framework has shown strong performance in processing large datasets, and the results are actionable for healthcare professionals to focus on high-risk groups.

## References

UCI Heart Disease Data [WWW Document], n.d. URL https://www.kaggle.com/datasets/redwankarimsony/heart-disease-data (accessed 11.27.24).