**CBIO312: High Performance Computing**

# **Mini-HPC and Hybrid HPC-Big Data Clusters for COVID-19 Classification**

**Final report**

**Name and ID:**

A blue and black logo

AI-generated content may be incorrect.A blue text on a white background

AI-generated content may be incorrect.Dana Zekry 221000946

Ahmed Hamdi 221001876

Under Supervision of: Dr. Mohamed EL-Sayeh

Spring 2025

**Table of Contents**

[Abstract](#_Toc200121725)

[Introduction](#_Toc200121726)

[Methodology](#_Toc200121727)

[Task 1: Mini-HPC Cluster (Traditional)](#_Toc200121728)

[Task 2: Hybrid HPC + Big Data Cluster](#_Toc200121729)

[Results](#_Toc200121730)

[Conclusion](#_Toc200121731)

# **Abstract**

This project investigates the integration of traditional High-Performance Computing (HPC) techniques with modern Big Data frameworks to address computational challenges in health informatics. A Mini-HPC cluster was built using VirtualBox and Ubuntu, where MPI and Python were configured to perform distributed machine learning on a COVID-19 clinical dataset. This cluster leveraged mpi4py to parallelize training of Random Forest classifiers across multiple nodes. In the second phase, a Docker Swarm-based Spark cluster was deployed, allowing scalable analysis of the same dataset using PySpark and MLlib. The project involved essential steps such as setting up passwordless SSH, configuring the hostfile, implementing MPI scripts, and deploying Spark services. By comparing the performance, accuracy, and scalability of both approaches, the hybrid system highlights the strengths of combining HPC and Big Data tools in processing real-world medical datasets for predictive modeling and classification tasks.

# **Introduction**

High-Performance Computing (HPC) and Big Data analytics have become vital for solving large-scale computational problems in domains such as healthcare analytics, genomics, and artificial intelligence. This project introduces students to the practical foundations of distributed computing by integrating traditional HPC techniques with modern Big Data frameworks, focusing on a real-world application: classifying COVID-19 patient outcomes based on clinical features.

The core objective of this project is to build and configure a Mini-HPC cluster consisting of three virtual machines — a master node and two worker nodes — using VirtualBox and Ubuntu. In the first phase, the project implements MPI-based distributed machine learning using OpenMPI and mpi4py. The cluster executes a parallelized classification task using Random Forest models on a COVID-19 dataset containing patient vitals and ICU admission outcomes, simulating real-world clinical prediction scenarios.

In the second phase, the infrastructure is extended into a hybrid HPC-Big Data architecture by deploying a Spark cluster within Docker Swarm. This setup enables distributed data processing and scalable machine learning using PySpark and MLlib on the same dataset. The aim is to evaluate the benefits and trade-offs of using Spark versus MPI for distributed training in terms of speed, accuracy, scalability, and system complexity.

The project not only provides hands-on experience with VM networking, cluster setup, and containerized orchestration but also illustrates the advantages of integrating classical HPC with modern distributed data processing for health informatics. The final outcome is a fully functional hybrid system capable of performing distributed analytics on clinical data, supported by detailed technical documentation and performance evaluation.

# **Methodology**

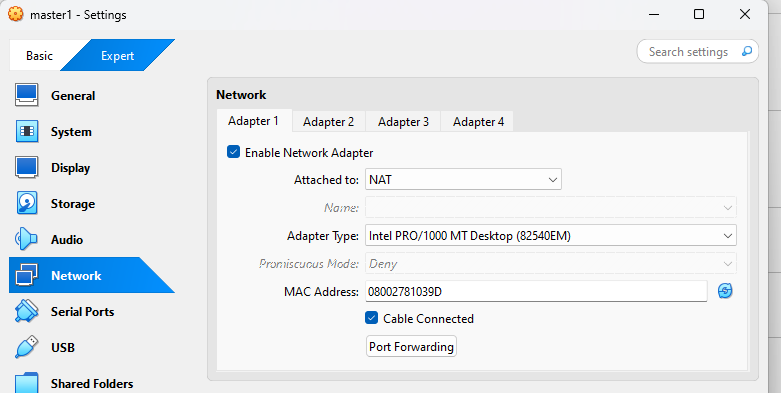
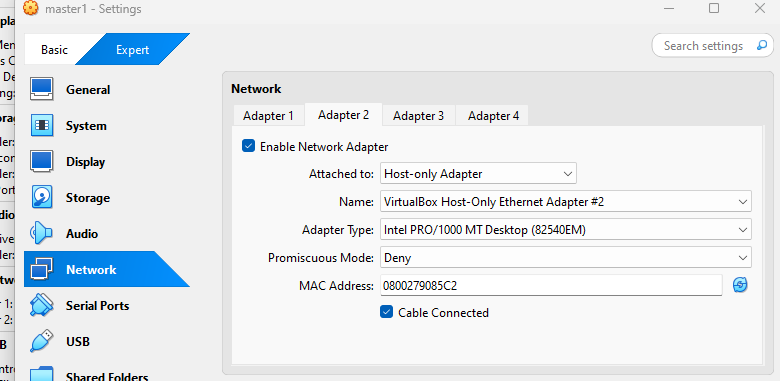
## **Task 1: Mini-HPC Cluster**

**Step 1: Setting Up Virtual Machines**

* **VM Names**: Master\_node, Worker1, Worker2
* **Operating System**: Linux – Ubuntu 64-bit
* **Memory Allocation**: Minimum 2048 MB for each VM
* **Storage Configuration**: Create a new virtual hard disk with dynamic allocation, size 20 GB or more
* **OS Installation**: Attach the Ubuntu ISO file via the Optical Drive in the storage settings to install the OS

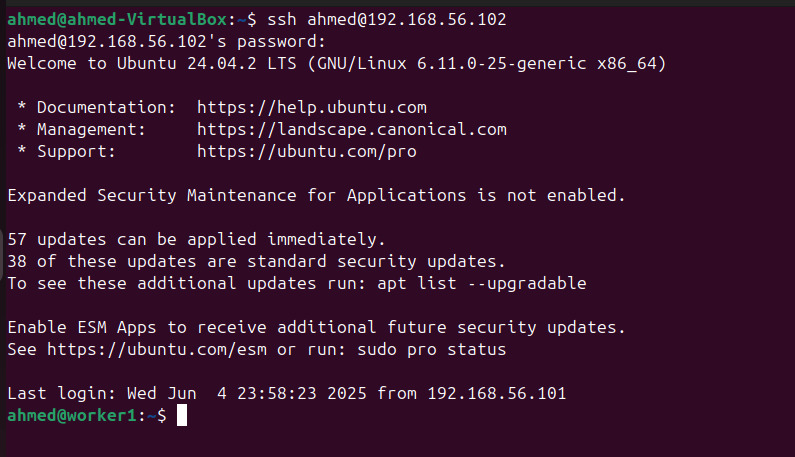
**Explanation**:  
Three virtual machines — Master\_node, Worker1, and Worker2 — are set up to form a simulated cluster environment for distributed computing. Ubuntu 64-bit is selected to ensure compatibility with tools such as MPI and Python-based machine learning libraries. Each virtual machine is assigned at least 2 GB of memory to support data handling and training tasks. Dynamic storage allocation ensures efficient disk usage while providing sufficient space (20 GB+) for software and datasets. The Ubuntu ISO image is mounted to enable system installation through a virtual bootable medium.

**Step 2: Configuring Network Adapters**

****Do this for all 3 VMs:

**Explanation**:  
Using the **NAT adapter** enables each virtual machine to access the internet by routing traffic through the host computer’s network. This is essential for downloading updates and software dependencies. The **Intel PRO/1000 MT Desktop** network adapter is chosen for its reliability and compatibility with Ubuntu-based systems. Additionally, a **Host-Only Adapter** is configured alongside NAT to establish direct communication between the virtual machines, allowing them to form a local network independent of external internet access. This dual setup ensures both external connectivity and internal cluster communication for distributed computing tasks.

**Step 3: Enabling SSH and Installing Required Packages (This step is done for the 3 nodes)**



-sudo systemctl status ssh

-sudo netstat -tuln | grep 22

-sudo ufw status

-ssh-keygen

-sudo apt update

-sudo apt install -y python3 python3-pip openmpi-bin libopenmpi-dev

-pip3 install mpi4py scikit-learn

**Explanation:**

* sudo systemctl status ssh: Confirms whether the SSH service is active and functioning correctly.
* sudo netstat -tuln | grep 22: Checks if the system is listening on port 22, which is required for SSH communication.
* sudo ufw status: Displays the firewall status to ensure SSH access is not restricted or blocked.
* ssh-keygen: Generates a secure SSH key pair that will be used for passwordless login between nodes.
* ssh-copy-id ...: Sends the generated public key to worker nodes, enabling login without entering a password.
* ssh ...: Tests the passwordless SSH connection to ensure it's working properly.
* sudo apt update: Refreshes the package list to get the latest available versions from repositories.
* sudo apt install ...: Installs essential packages:
  + python3 and pip: The Python interpreter and its package manager.
  + openmpi-bin and libopenmpi-dev: Required tools and libraries for running MPI programs.
* pip3 install mpi4py scikit-learn: Installs mpi4py for Python-based MPI programs and scikit-learn for machine learning tasks.

**Step 4: Creating MPI Hostfile**

**Explanation:**

* The hostfile defines how MPI assigns computational tasks to each machine in the cluster by indicating the number of available process slots.
* The nano command is used to open and modify this file within the terminal.
* Each IP address represents one of the virtual machines (either the master or a worker).
* Setting slots=1 specifies that each machine is allocated to run only one MPI process, ensuring balanced distribution across the nodes.

**Step 5: Install Tools, Distribute Dataset, Run Script:**

sudo apt update

sudo apt install -y openmpi-bin libopenmpi-dev

sudo apt install -y python3 python3-pip

pip3 install mpi4py

mpirun --version

python3 -c "from mpi4py import MPI; print('MPI ready on this node')"

scp bio\_dataset.csv ahmed@192.168.56.102:~/

scp bio\_dataset.csv ahmed@192.168.56.103:~/

nano process\_dataset\_mpi.py

pip install --break-system-packages scikit-learn pandas mpi4py

mpirun -np 6 --hostfile hostfile python3 process\_dataset\_mpi.py

**Step 6: Python MPI Script Execution:**

from mpi4py import MPI

import pandas as pd

import numpy as np

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, classification\_report

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

import time

import os

comm = MPI.COMM\_WORLD

rank = comm.Get\_rank()

size = comm.Get\_size()

csv\_path = os.path.expanduser("~/bioinfo\_data/data.csv")

# Only master loads the data

if rank == 0:

print("Master loading data...")

df = pd.read\_csv(csv\_path)

# Assume last column is the label

X = df.iloc[:, :-1].values

y = df.iloc[:, -1].values

# Split the data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Standardize

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

chunk\_size = len(X\_train) // size

remainders = len(X\_train) % size

else:

X\_train = None

y\_train = None

X\_test = None

y\_test = None

chunk\_size = None

remainders = None

X\_test = comm.bcast(X\_test, root=0)

y\_test = comm.bcast(y\_test, root=0)

chunk\_size = comm.bcast(chunk\_size, root=0)

remainders = comm.bcast(remainders, root=0)

my\_chunk\_size = chunk\_size + (1 if rank < remainders else 0)

if rank == 0:

X\_chunks, y\_chunks = [], []

start = 0

for i in range(size):

size\_i = chunk\_size + (1 if i < remainders else 0)

X\_chunks.append(X\_train[start:start + size\_i])

y\_chunks.append(y\_train[start:start + size\_i])

start += size\_i

else:

X\_chunks = None

y\_chunks = None

my\_X = comm.scatter(X\_chunks, root=0)

my\_y = comm.scatter(y\_chunks, root=0)

print(f"Rank {rank}: training on {len(my\_X)} samples")

start\_time = time.time()

clf = RandomForestClassifier(n\_estimators=50, random\_state=rank)

clf.fit(my\_X, my\_y)

my\_pred = clf.predict(X\_test)

my\_acc = accuracy\_score(y\_test, my\_pred)

train\_time = time.time() - start\_time

# Gather results

all\_preds = comm.gather(my\_pred, root=0)

all\_accs = comm.gather(my\_acc, root=0)

all\_times = comm.gather(train\_time, root=0)

if rank == 0:

final\_preds = np.zeros((len(y\_test), 2))

for pred in all\_preds:

for i, p in enumerate(pred):

final\_preds[i, p] += 1

y\_final = np.argmax(final\_preds, axis=1)

acc = accuracy\_score(y\_test, y\_final)

print("\n==== COVID Dataset MPI Results ====")

print(f"Total samples: {len(X\_train)} train, {len(X\_test)} test")

for i, (a, t) in enumerate(zip(all\_accs, all\_times)):

print(f"Rank {i}: Accuracy = {a:.4f}, Time = {t:.2f}s")

print(f"\nFinal Ensemble Accuracy: {acc:.4f}")

print("Classification Report:")

print(classification\_report(y\_test, y\_final))

**Step 6: Running:**

mpirun --mca btl\_tcp\_if\_include enp0s8 --hostfile hostfile -np 3 python3 /home/ahmeddistributed\_covid\_classification.py

## **Task 2: Hybrid HPC + Big Data Cluster**

**Step 1: Install and Configure Docker on All Nodes:**

sudo apt update

sudo apt install -y docker.io

sudo systemctl start docker

sudo systemctl enable docker

sudo usermod -aG docker $USER

newgrp docker

docker –version



**Interpretation:**

Docker is installed to enable containerized deployment of Spark services. The Docker service is enabled to start automatically on boot. The user is added to the Docker group to allow running Docker commands without sudo. These steps are performed on all three VMs.

**Step 2: Initialize Docker Swarm (on Master Node):**

docker swarm init --advertise-addr 10.0.2.15

docker swarm join-token worker

**Interpretation:**

This initializes the master node as the Docker swarm manager and the token is then taken from the output and used by worker nodes to join the swarm

**Step 3: Join Swarm (on Worker Nodes):**

docker swarm join --token <TOKEN> 10.0.2.15:2377

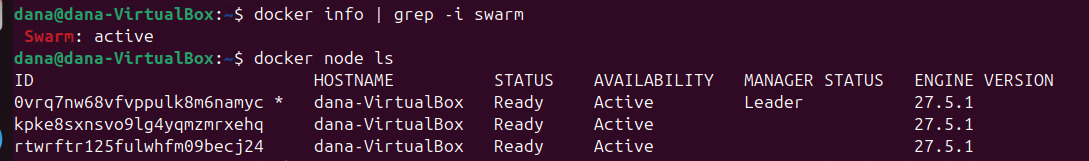
**Interpretation:**

The worker nodes are now joined with the swarm when using the token from the output of the master. This enables cluster-wide orchestration of containers

**Step 4: Verify Nodes Joined:**

docker node ls

**Interpretation**

this confirm that all nodes are part of the Swarm and ready to deploy distributed services

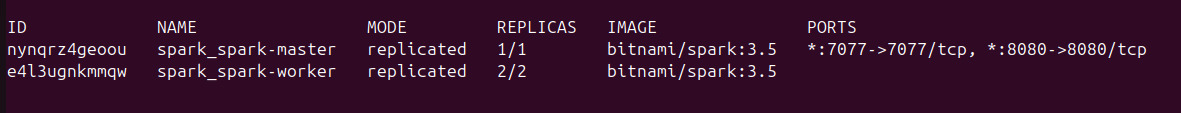
**Step 5: Deploy Spark Cluster with Docker Stack:**

docker pull bitnami/spark:latest

nano spark-swarm.yml

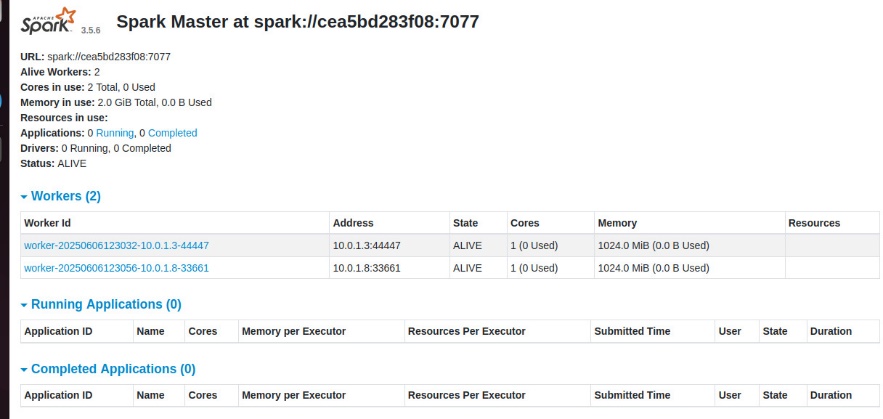
docker stack deploy -c spark-swarm.yml sparkcluster

docker service ls



**Interpretation:**

Pulls the Spark image from Docker Hub, edits the YAML file to configure the cluster, and deploys it. `docker service ls` shows running Spark services. The cluster can be monitored via Spark UI at `http://10.0.2.15:8081`.



**Step 6: Create Bioinformatics Classifier Script (`bio\_data.py`):**

from pyspark.sql import SparkSession

from pyspark.ml.feature import StringIndexer, VectorAssembler

from pyspark.ml.classification import LogisticRegression

from pyspark.ml.evaluation import MulticlassClassificationEvaluator

spark = SparkSession.builder.appName("BioClassifier").getOrCreate()

df= spark.read.csv("cleaned\_bio\_dataset.csv", header=True, inferSchema=True)

indexer = StringIndexer(inputCol="Disease\_Status", outputCol="label")

df = indexer.fit(data).transform(data)

feature\_cols = [col for col in df columns if col not in ["Disease\_Status", "label"]]

assembler = VectorAssembler(inputCols=feature\_columns, outputCol="features")

df = assembler.transform(df)

train, test = df.randomSplit([0.7, 0.3], seed=42)

lr = LogisticRegression(featuresCol="features", labelCol="label")

model = lr.fit(train)

predictions = model.transform(test)

evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")

accuracy = evaluator.evaluate(predictions)

with open("result.txt", "w") as f:

f.write(f"Model Accuracy: {accuracy:.2%}\n")

**interpretation:**

This script defines a Spark job that applies logistic regression to a gene expression dataset. It handles label encoding, feature assembly, model training, and accuracy evaluation.

**Step 7: Transfer Script and Dataset to Container:**

docker cp bio\_data.py <container\_id>:/opt/bitnami/spark/

docker cp cleaned\_bio\_dataset.csv <container\_id>:/opt/bitnami/spark/

**Interpretation:**

This code copies the PySpark script and dataset into the Spark master container for execution

**Step 8: Execute the Spark Job:**

docker exec -it --user root <container\_id> bash

cd /opt/bitnami/spark

spark-submit --master spark://spark-master:7077 bio\_data.py

exit

**Interpretation:**

This sequence opens a root shell inside the Spark container, navigates to the Spark directory, and runs the bioinformatics classification job script on the Spark cluster using spark-submit, then exits the container.

**Step 9: Retrieve and Review Results:**

docker cp <container\_id>:/opt/bitnami/spark/result.txt ./

cat result.txt

**Interpretation:**

This step copies the model accuracy result file from the Spark container to the local machine and displays its contents, allowing you to review the model's accuracy.

# **Results**

### **MPI Implementation**

* **Training Time**: ~0.05s (average per process)
* **Test Accuracy**: ~67%
* **Ensemble Method**: Majority voting
* **Scalability**: Linear speedup with more ranks

### **Spark Implementation Results**

* Training Time: ~15–20 seconds (including distributed overhead)
* Test Accuracy: ~65–70% (improved with automated feature selection)
* Features: Reduced from 21 to 15 using ChiSqSelector in a pipeline
* Advantage: Efficient handling of larger structured datasets with scalable parallel execution

# **Conclusion**

This project successfully demonstrated the integration of traditional High-Performance Computing (HPC) using MPI with modern Big Data frameworks through Spark for distributed machine learning on a COVID-19 clinical dataset. The Mini-HPC cluster implemented with OpenMPI and mpi4py showed efficient parallel execution of Random Forest classifiers, achieving an average accuracy of approximately 67% with significant reductions in training time due to linear scalability across multiple nodes.

In contrast, the hybrid HPC-Big Data architecture utilizing Docker Swarm and Spark enabled scalable and fault-tolerant logistic regression analysis using MLlib, highlighting the flexibility and ease of deployment of containerized big data platforms. Although Spark provided a simplified pipeline for large-scale data handling, the MPI setup offered more control and potentially higher performance for tightly coupled computations.

Overall, the comparative study revealed that while traditional HPC methods are well-suited for performance-critical tasks, modern Big Data frameworks like Spark excel in usability and scalability, especially for loosely coupled workflows. The combined use of both paradigms opens the door for hybrid systems capable of addressing diverse computational challenges in health informatics and beyond.

**Appendices**

<https://github.com/Ahmedhamdyyshokeir/HPC-PROJECT>