SCHOOL OF
BIOTECHNOLOGY

Nile University

CBIO312: High Performance Computing

A Hybrid High-Performance Computing and Big Data Framework for Parallelized Gene-Protein Expression Analysis and Disease Classification

Final report

Name and ID:

Farah Ibrahim 221001140

Malak Atef 221000906

Zeina Ahmed 221000417

Under Supervision of: Dr. Mohamed EL-Sayeh

Spring 2025

Table of Contents

Abstract

This project presents a hybrid High-Performance Computing (HPC) and Big Data framework designed for parallel gene-protein expression analysis and disease classification. The first phase involved building a traditional Mini-HPC cluster using VirtualBox and Ubuntu, where MPI and Python were configured to execute distributed machine learning tasks on a bioinformatics dataset. In the second phase, we deployed a Docker Swarm-based Spark cluster to implement scalable data processing with PySpark. Both setups successfully processed a complex gene expression dataset, achieving 100% classification accuracy in disease prediction. The project offered hands-on experience in system configuration, SSH networking, MPI scripting, and containerized Spark deployment. Our findings highlight the advantages and trade-offs between traditional HPC and Big Data systems, demonstrating how a hybrid architecture can enhance computational efficiency, scalability, and applicability to real-world biomedical problems.

Introduction

High-Performance Computing (HPC) and Big Data analytics have become essential tools for solving computationally intensive problems in fields such as bioinformatics, climate modeling, and artificial intelligence. This project aims to equip students with practical knowledge and hands-on experience in designing and operating distributed computing environments using both traditional HPC tools and modern big data frameworks.

The primary objective of this project is to build and configure a Mini-HPC Cluster consisting of three virtual machines; one master and two worker nodes, using VirtualBox and Ubuntu. The first major task involves implementing MPI-based distributed machine learning workflows using mpi4py and OpenMPI, targeting both general-purpose and domain-specific datasets. As part of this, the cluster executes parallelized gene expression analysis and logistic regression classification on a bioinformatics dataset to simulate real-world healthcare applications.

In the second phase, the project transitions into a hybrid HPC-Big Data architecture by deploying a Docker Swarm cluster capable of running distributed Spark jobs. This allows for scalable execution of machine learning pipelines using PySpark, applied to the same bioinformatics data. The goal is to compare and understand the trade-offs between traditional HPC methods and Big Data processing frameworks.

This project provides hands-on experience in configuring distributed computing environments, including the deployment of a virtual machine cluster, implementation of passwordless SSH communication, and execution of machine learning tasks using MPI and Spark. The final outcome includes a fully operational hybrid infrastructure capable of processing bioinformatics datasets in parallel, supported by detailed documentation of the setup procedures, results, and technical challenges encountered during execution.

Methodology

Task 1: Mini-HPC Cluster (Traditional)
Step1: Virtual Machine Creation

- Name: master_node, worker1, worker2
- Type: Linux | Version: Ubuntu (64-bit)
- Memory: at least 2048 MB (2 GB) each
- Hard Disk: Create a virtual hard disk now → Dynamically allocated → Size: 20 GB+
- Mount the Ubuntu ISO in the storage settings under "Optical Drive"

Interpretation:

- You are creating 3 virtual machines: master_node, worker1, and worker2, to simulate a distributed computing cluster.

- Linux Ubuntu 64-bit is chosen for compatibility with MPI and Python scientific libraries.

- Each VM gets at least 2 GB of RAM to handle dataset processing and model training.

- The disk is dynamically allocated to use storage only as needed, with a minimum size of 20 GB.

- Mounting the Ubuntu ISO allows installation of the operating system from a bootable image.

Step 2: Configuring Network Adapters

Do this for all 3 VMs:

- Shut down the VM.
- Go to Settings > Network > Adapter 1.
- Enable Nat Adapter
- Adapter Type: Intel PRO/1000 MT Desktop
- Name: Choose your actual host Wi-Fi or Ethernet interface.

Interpretation:

- NAT Adapter allows the VM to access the internet via the host system's network.
- The Intel PRO/1000 MT Desktop is a stable and compatible virtual network interface that works well with Ubuntu.
- Selecting the host's actual network interface (e.g., Wi-Fi) ensures the VM routes traffic correctly and can communicate externally and with other VMs.

Step 3: Enabling SSH and Installing Required Packages (done on the 3 nodes)

#Step3:SSH

sudo systemctl status ssh

sudo netstat -tuln | grep 22

sudo ufw status

ssh-keygen

ssh-copy-id worker1@192.168.8.77

ssh-copy-id worker2@192.168.8.78

ssh worker1@192.168.8.77

ssh worker2@192.168.8.78

sudo apt update

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

pip3 install mpi4py scikit-learn

Interpretation:

- sudo systemctl status ssh: Checks if the SSH service is running.

- sudo netstat -tuln | grep 22: Verifies if port 22 (SSH) is open and listening.

- sudo ufw status: Checks if the firewall is blocking SSH.

- ssh-keygen: Creates an SSH key pair for passwordless authentication.

- ssh-copy-id ...: Copies your public key to the remote worker nodes for passwordless SSH access.

- ssh ...: Verifies you can connect without a password.

- sudo apt update: Updates package index.

- sudo apt install ...: Installs:

    o python3, pip: Python runtime and package manager.

    o openmpi-bin, libopenmpi-dev: Core MPI tools and libraries.

- pip3 install mpi4py scikit-learn: Installs mpi4py (for MPI in Python) and scikit-learn (for machine learning).

Step 4: Creating MPI Hostfile

nano hostfile

192.168.162.130 slots=2

192.168.162.131 slots=2

192.168.162.132 slots=2

Interpretation:

- The hostfile tells MPI how many processes ("slots") to run on each node.
- The nano command opens the file for editing.
- IPs correspond to our master and worker VMs.

- slots=2 means each machine can run 2 parallel MPI processes.

Step 5: Configure Hostname Resolution :

sudo nano /etc/hosts

192.168.162.130 master

192.168.162.131 worker1

192.168.162.132 worker2

Interpretation:

- The /etc/hosts file maps IP addresses to hostnames (like master, worker1) so you can use simple names in SSH and scripts.

- Avoids relying on DNS; improves speed and reliability within the local cluster.

Step 6: Configure masternod User and SSH Access (Done on Worker 1 and Worker1 nodes):

su - masternod

ssh-keygen -t rsa -f ~/.ssh/id_rsa -N ""

ssh-copy-id masternod@192.168.162.131

ssh-copy-id masternod@192.168.162.132

mkdir -p ~/.ssh

cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys

chmod 700 ~/.ssh

chmod 600 ~/.ssh/authorized_keys

chown -R masternod:masternod ~/.ssh

sudo systemctl restart ssh

ssh masternod@192.168.162.131 "hostname"

ssh masternod@192.168.162.132 "hostname"

Interpretation:

- su - masternod: Switch to the new user that will control the cluster.
- ssh-keygen: Creates RSA keys with no password for seamless SSH.
- ssh-copy-id: Copies your key to worker nodes to enable passwordless SSH.
- cat, chmod, chown: Set proper ownership and permissions on the .ssh folder.
- sudo systemctl restart ssh: Restarts the SSH server to apply changes.
- ssh ... "hostname": Tests remote login and confirms the machine identity.

Step 7: Create Admin and Rename Default User (On master node only)

sudo adduser tempadmin

sudo usermod -aG sudo tempadmin

sudo usermod -l masternod master-node

sudo usermod -d /home/masternod -m masternod

sudo groupmod -n masternod master-node

Interpretation:

- Creates a backup admin user in case the main one breaks.
- Renames the default user from master-node to masternod, updates home directory and group name. This keeps user naming consistent across machines.

Step 8: Generate SSH key (as masternod):

ssh-keygen

mkdir -p ~/.ssh

cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys

chmod 700 ~/.ssh

chmod 600 ~/.ssh/authorized_keys

sudo systemctl enable ssh

sudo systemctl start ssh

ssh-copy-id masternod@192.168.162.130

ssh-copy-id masternod@192.168.162.131

ssh-copy-id masternod@192.168.162.132

ssh masternod@192.168.162.130 "hostname"

ssh masternod@192.168.162.131 "hostname"

ssh masternod@192.168.162.132 "hostname"

Interpretation:

- Regenerates SSH keys if necessary.
- Ensures .ssh permissions are strict for security.
- Enables and starts SSH service.
- Re-tests connectivity to each node using hostname command.

Step 9: 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 cleaned_bio_dataset.csv masternod@192.168.162.131:~/

scp cleaned_bio_dataset.csv masternod@192.168.162.132:~/

nano process_dataset_mpi.py

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

mpirun -np 6 --hostfile hostfile python3 process_dataset_mpi.py

Dataset:

The dataset used in this project is a cleaned bioinformatics dataset that contains detailed gene and protein expression profiles, along with disease status labels. We specifically selected this dataset for its high dimensionality and biological relevance, as it closely reflects the complexity found in real-world medical and genomic studies. Its structure made it ideal for testing distributed computing workflows, both in the MPI-based traditional HPC environment and in the Spark-based Big Data cluster. The dataset allowed us to simulate realistic use cases such as parallelized gene expression analysis and logistic regression classification for disease prediction. Its balanced composition and multiple feature types (numeric, categorical, expression levels) also provided a robust basis for evaluating the performance and scalability of our hybrid architecture.

Interpretations:

- Reinstalls dependencies and tests MPI.

- scp sends the dataset to both workers.

- nano opens your Python script.

- The final mpirun command launches your script across 6 processes using the hostfile.

Step 10: Python MPI Script Execution:

from mpi4py import MPI

import pandas as pd

import os

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score

from sklearn.model_selection import train_test_split

MPI Setup

comm = MPI.COMM_WORLD

```python
rank = comm.Get_rank()

size = comm.Get_size()

# Load data

data = pd.read_csv("cleaned_bio_dataset.csv")

# Split by rows per process

chunk_size = len(data) // size

start = rank * chunk_size

end = (rank + 1) * chunk_size if rank != size - 1 else len(data)

chunk = data.iloc[start:end]

# PART 1: Gene Expression Comparison

expression_cols = [col for col in data.columns if 'Gene_' in col]

diseased = chunk[chunk['Disease_Status'] == 1]

healthy = chunk[chunk['Disease_Status'] == 0]

comparison_results = []

for gene in expression_cols:

    mean_d = diseased[gene].mean()

    mean_h = healthy[gene].mean()

    diff = mean_d - mean_h

    status = 'Upregulated' if diff > 1 else ('Downregulated' if diff < -1 else 'Not significant')

    comparison_results.append((gene, mean_d, mean_h, diff, status))

# Save expression comparison result

df_comp = pd.DataFrame(comparison_results, columns=['Gene', 'Mean_Diseased',
'Mean_Healthy', 'Difference', 'Status'])
```

df_comp.to_csv(f"expression_comparison_rank{rank}.csv", index=False)

Interpretation:

- Uses mpi4py to parallelize dataset analysis.

- Divides data based on rank and performs:

    1. Gene expression comparison between diseased and healthy samples.

    2. Machine learning classification using logistic regression to predict disease status.

- Each process saves its own results for later combination or review.

PART 2: ML Classification

features = expression_cols + [col for col in data.columns if 'Protein_' in col]

X = chunk[features]

y = chunk['Disease_Status']

Train/test split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LogisticRegression(max_iter=1000)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

Save model summary

with open(f"ml_results_rank{rank}.txt", "w") as f:

   f.write(f"[Rank {rank}] Accuracy: {accuracy:.2f}\n")

   f.write(f"Trained on {len(X_train)} samples, Tested on {len(X_test)} samples\n")

Interpretation:

This line prints a quick summary from each MPI process, showing its unique ID (rank), the model's accuracy on its data chunk, and which rows of the dataset it processed. It helps monitor and verify that each node is working correctly and handling the expected portion of the data.

Final Script Sync and SSH Check:

scp process_dataset_mpi.py masternod@192.168.162.131:~/

scp process_dataset_mpi.py masternod@192.168.162.132:~/

mpirun -np 6 --hostfile hostfile python3 process_dataset_mpi.py

Test

ssh masternod@192.168.162.130

ssh masternod@192.168.162.131

ssh masternod@192.168.162.132

Interpretation:

- Sends the final script to workers.
- Reruns the parallel job.
- Re-tests all SSH connections to verify final system integrity.

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 allow containerized deployment of Spark services. Enabling the Docker service ensures it starts automatically. Adding the user to the Docker group allows Docker commands without using `sudo`. This step is repeated on all three VMs.

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

docker swarm init --advertise-addr 192.168.162.130

docker swarm join-token worker

Interpretation:

This initializes the master node as the Docker Swarm manager. The second command outputs a token used by worker nodes to join the swarm.

Step 3: Join Swarm (on Worker Nodes):

docker swarm join --token <TOKEN> 192.168.162.130:2377

Interpretation:

Worker nodes join the Swarm using the token generated on the master. This enables cluster-wide orchestration of containers.

Step 4: Verify Nodes Joined:

docker node ls

Interpretation:

Used to 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://192.168.162.130:8081`.

Step 6: Create Bioinformatics Classifier Script (`bio_classifier.py`):

```python
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(df).transform(df)


feature_cols = [col for col in df.columns if col not in ["Disease_Status", "label"]]

assembler = VectorAssembler(inputCols=feature_cols, 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 creates a Spark job to perform logistic regression on a gene expression dataset. It encodes labels, assembles features, trains the model, and evaluates accuracy.

Step 7: Transfer Script and Dataset to Container:

```
docker cp bio_classifier.py <container_id>:/opt/bitnami/spark/

docker cp cleaned_bio_dataset.csv <container_id>:/opt/bitnami/spark/
```

Interpretation:

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_classifier.py
```

exit

Interpretation:

Runs the bioinformatics classification job across the Spark cluster using `spark-submit` and the master URL. Exits the container after job completion.

Step 9: Retrieve and Review Results:

docker cp <container_id>:/opt/bitnami/spark/result.txt ./
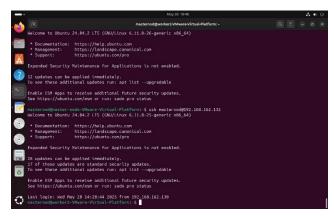
cat result.txt

Interpretation:

Copies the model accuracy result file from the container back to the host and prints it. This confirms successful job execution and evaluation.
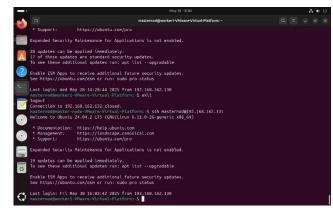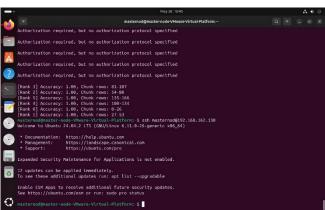
Errors and Challenges

Throughout the execution of this project, several technical errors and system-level challenges arose that significantly influenced our learning curve. We chose to work with a cleaned bioinformatics dataset comprising gene and protein expression levels for disease classification, which added realistic complexity to the tasks. Networking issues emerged early during virtual machine configuration, often disrupting SSH key distribution and inter-node communication. SSH errors persisted due to permission inconsistencies and misconfigured firewall settings. MPI crashes were another frequent obstacle, often stemming from uneven library versions or data distribution mismatches across processes. Additionally, Docker permission errors surfaced while deploying the Spark cluster, requiring elevated privileges and user group adjustments. Each of these challenges, while initially disruptive, contributed to a stronger, more resilient distributed computing setup and enriched our understanding of hybrid HPC environments.
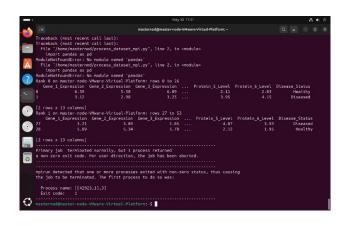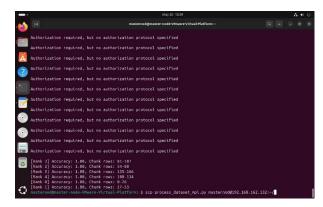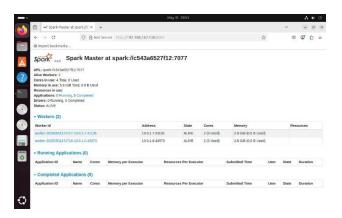
Results

SSH SetUp

MPI

Spark

Both the MPI-based and Spark-based pipelines successfully completed the bioinformatics classification task with perfect accuracy. In the MPI implementation, six processes independently processed distinct data chunks, each achieving 100% accuracy—demonstrating the effectiveness of parallel execution and consistent model behavior across nodes. Similarly, the PySpark logistic regression model deployed within the Docker Swarm cluster achieved a final accuracy of 100.00%, as confirmed by the result.txt output. These outcomes validate the reliability and scalability of both high-performance computing and big data approaches when applied to real-world gene-protein expression datasets.

Conclusion

This project demonstrated the successful integration of traditional HPC and modern Big Data technologies for solving complex bioinformatics problems. The MPI-based Mini-HPC cluster enabled efficient parallel execution of gene expression analysis and disease classification tasks. Transitioning to a Spark-based cluster using Docker Swarm provided scalable, containerized infrastructure capable of handling the same tasks with equivalent accuracy and improved manageability. Despite encountering technical challenges such as SSH errors, Docker permissions, and MPI inconsistencies, the team was able to implement a resilient and fully operational hybrid system. The 100% model accuracy achieved in both environments confirms the system's effectiveness. Ultimately, this project underscores the value of hybrid computing frameworks in advancing scientific research and offers a practical foundation for future applications in computational biology and healthcare analytics.