Laboratory Report

Parallel Computer Architecture Clusters and Grids

NHE2530-2425

Lai Minh Thong
Student ID: U2259343
Department of Computer Science
School of Computing and Engineering

March 2025

Abstract

This report details the creation and deployment of a Virtual Computer Cluster using Virtual Machines (VMs) on Microsoft Azure. It documents the process of configuring three VMs (one Headnode and two Worker nodes), setting up interconnections and network security, and installing essential cluster middleware (OpenMPI). Additionally, the report provides instructions for running MPI benchmarking programs (such as `pingpong.c` and `mat_mat.c`), collecting and analyzing performance metrics (latency, bandwidth, and computational efficiency), and comparing results with peer clusters. The full deployment process, complete with diagrams, code listings, performance graphs, and configuration tables, is discussed in detail.

Contents

Chapter 1

Introduction

1.1 Purpose

This project establishes a Virtual Computer Cluster on Azure by deploying VMs for parallel computing, configuring OpenMPI middleware, and analyzing MPI programming model performance according to established distributed memory parallelism patterns Pacheco (1996).

1.2 Overview of VM Types and Requirements

- **Headnode:** Manages the cluster and job scheduling.
- **Worker nodes:** Execute computational tasks.

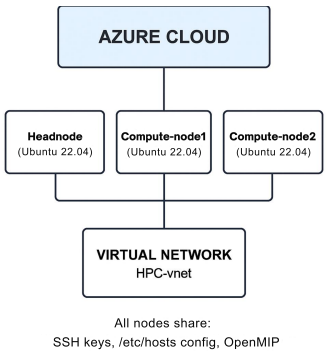

Figure 1.1: This project's VM Clusters Diagram

VMs have CPU, memory, storage configurations, connected via a secure virtual network for optimal cluster performance, efficient inter-node communication.

Chapter 2

Cluster and MPI Programs Deployment

2.1 Creating Virtual Machines in Azure

The aim of this session is to create Virtual Machines in Azure for a Virtual Computer Cluster (VCC). The VCC should have one Headnode and two Compute-node nodes. Virtual machines were configured following best practices from Azure documentation Microsoft (2024). Two types of VMs will be created: Headnode VM and Worker node(s) VM.

The headnode VM and both compute nodes (compute-node1 and compute-node2) were created with the following specifications:

- Resource group: HPC
- Operating system: Ubuntu 22.04
- CPU Size: Standard_B1s (1 vcpu with 1GiB Memory)
- Disk: Standard SSD
- Network interface: Virtual network HPC_vnet

***All VMs must be on the same Virtual Network.**

This process of configuring virtual Headnode and Compute-node nodes and their virtual network is equivalent to preparing laboratory hardware PCs with:

- Ubuntu 22.04
- CPU (possible 4 cores), Memory (4 or 8 Gbytes)
- Disk storage

- 1 Gbit Ethernet Network Interface Card
- Network switch for private and public network interconnect

2.1.1 Creating the Headnode VM

- Basics

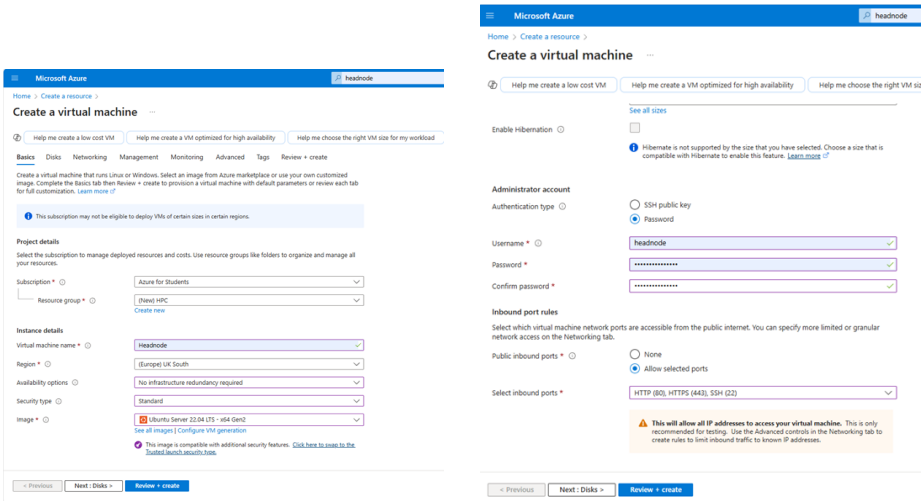

Figure 2.1: Headnode VM Basic Configuration

- Disk

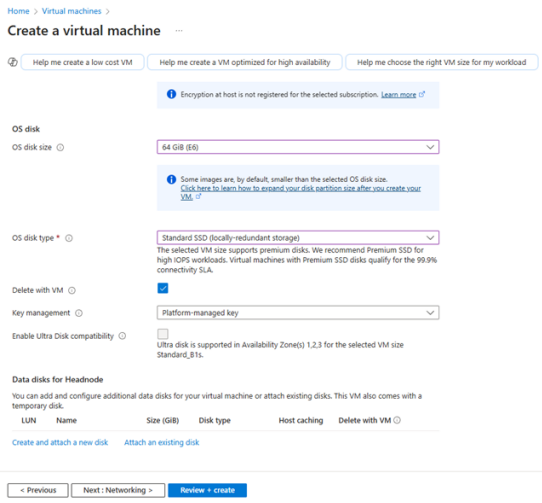

Figure 2.2: Headnode VM Disk Configuration

- Networking

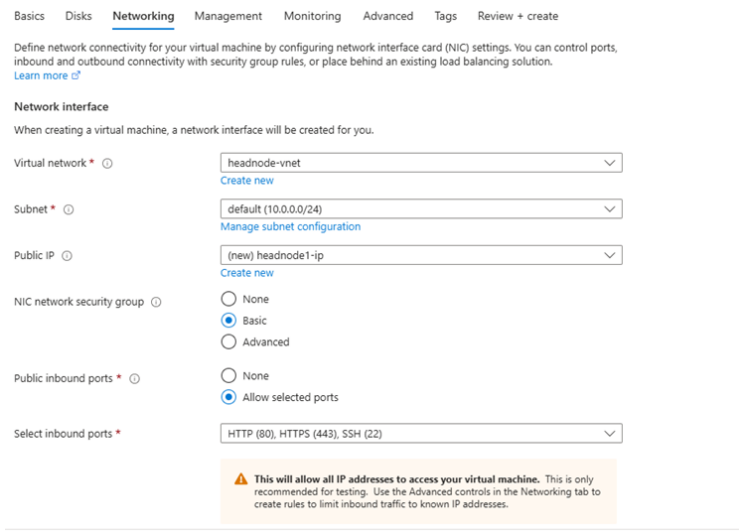

Figure 2.3: Headnode VM Network Configuration

2.1.2 Creating Compute-node1 and Compute-node2 VMs

- Basics

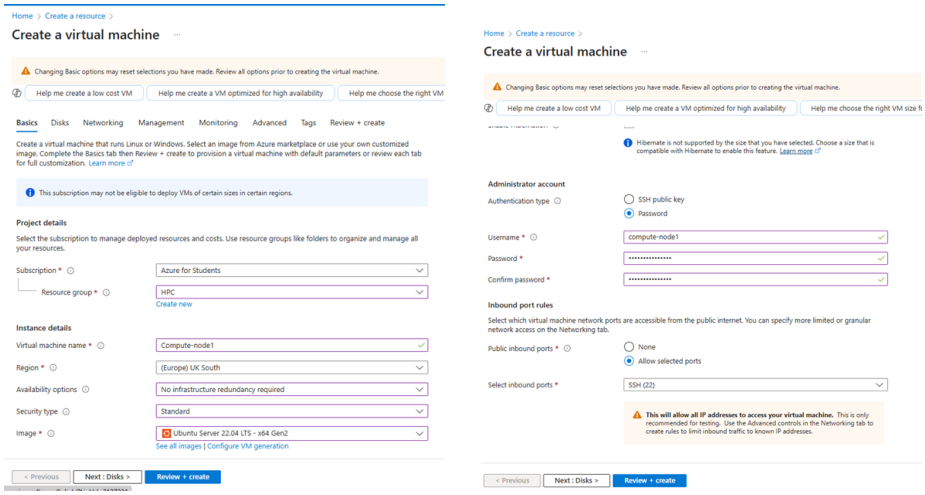

Figure 2.4: Compute Nodes Basic Configuration

- Networking

***Make sure that all VMs are on the same Virtual Network**

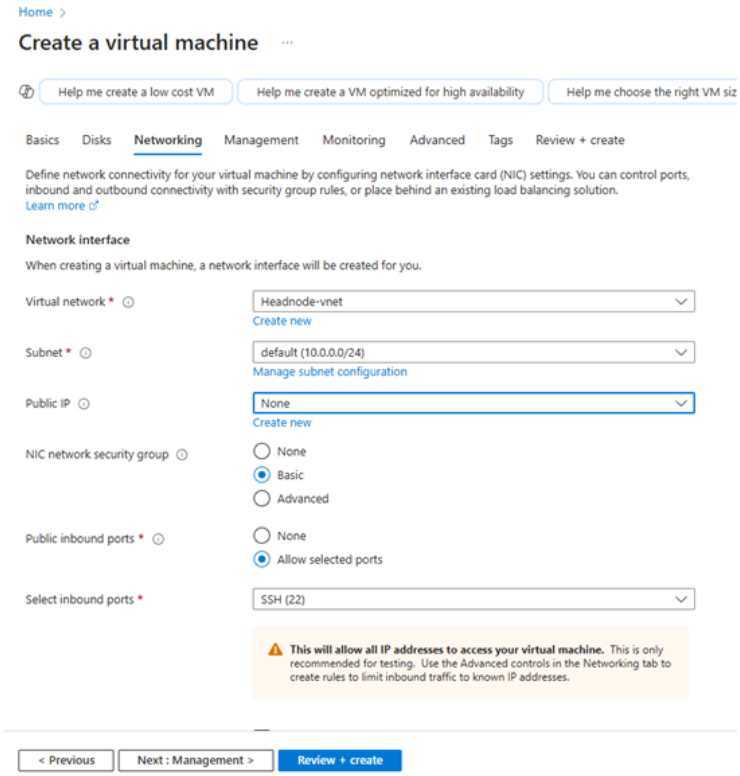

Figure 2.5: Compute Nodes Network Configuration

***Repeat the same steps to create compute-node2.**

After successful completion of creating, VM Dashboard displays the following:

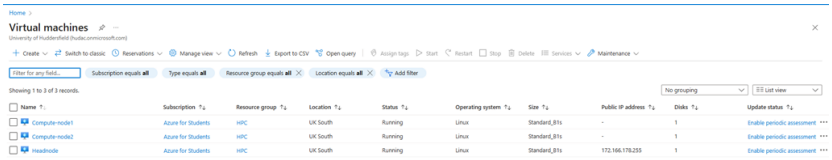

Figure 2.6: VM Dashboard

2.2 Establish Connection

In order to find out the ssh login details for the Headnode VM select Headnode VM from the list of the VMs and select Connect and SSH option:

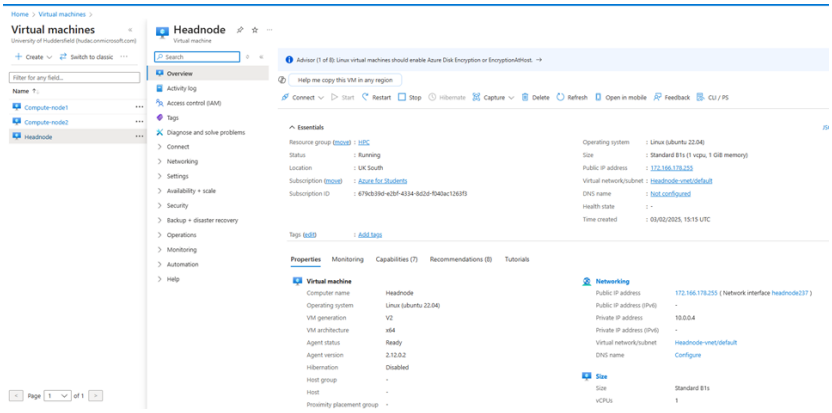

Figure 2.7: Headnode Configuration

Details of ssh path with the username provided in the Headnode configuration and IP address as below:

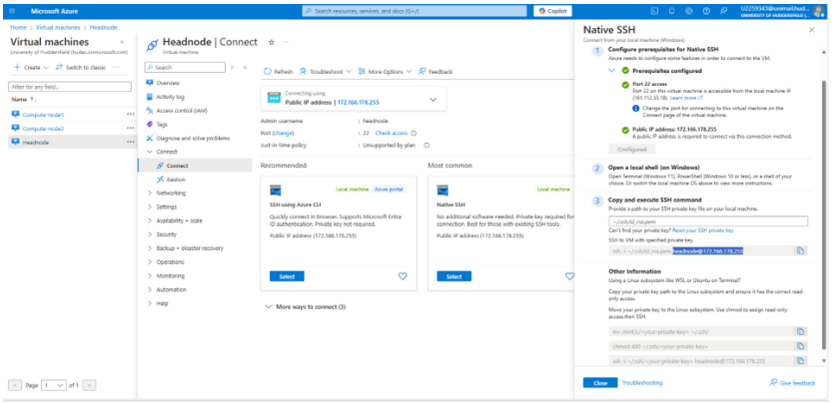

Figure 2.8: Headnode's SSH to VM

In the above screenshot, the ssh path of headnode would be

```
ssh headnode@172.166.178.255
```

Figure 2.9: Connect to Headnode

2.3 Setting Up a Beowulf Cluster on 3 Azure VMs Using Ubuntu 22.04

This project implements a Beowulf cluster architecture as described by Sterling et al. Sterling et al. (2001).

The goal of this section is to **install and test cluster middleware** OpenMPI on an **Azure-based Beowulf Cluster** with three VMs:

Pre-requisites

- Have **three Ubuntu 22.04 LTS VMs** in Azure:
  - **headnode**
  - **compute-node1**
  - **compute-node2**
- All VMs are in the same **Virtual Network (VNet)**.
- **SSH access** to all nodes.

2.3.1 Preparing the Head Node

Connect to the Head Node

```
ssh headnode@172.166.178.255
```

Disable the Firewall and SELinux

```
sudo systemctl stop ufw
sudo systemctl disable ufw
```

Edit the Hosts File on All Nodes

To enable **hostname resolution**, add all nodes to `/etc/hosts`.

Run:

```
sudo nano /etc/hosts
```

Add:

```
10.0.0.4    headnode
10.0.0.6    compute-node1
10.0.0.5    compute-node2
```

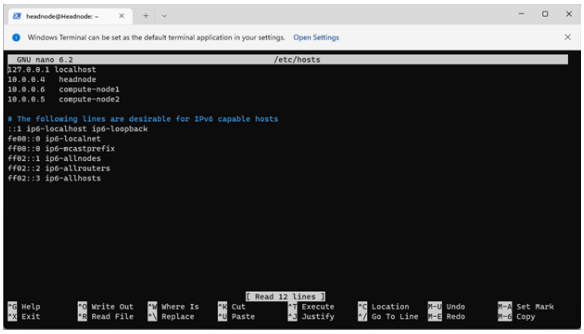

Figure 2.10: Editing the Hosts File

Save (CTRL+X, Y, ENTER).

Open the SSH configuration file:

```
nano ~/.ssh/config
```

Add the following configuration:

```
Host compute-node1
    HostName compute-node1
    User compute-node1
    IdentityFile ~/.ssh/id_rsa

Host compute-node2
    HostName compute-node2
    User compute-node2
    IdentityFile ~/.ssh/id_rsa
```

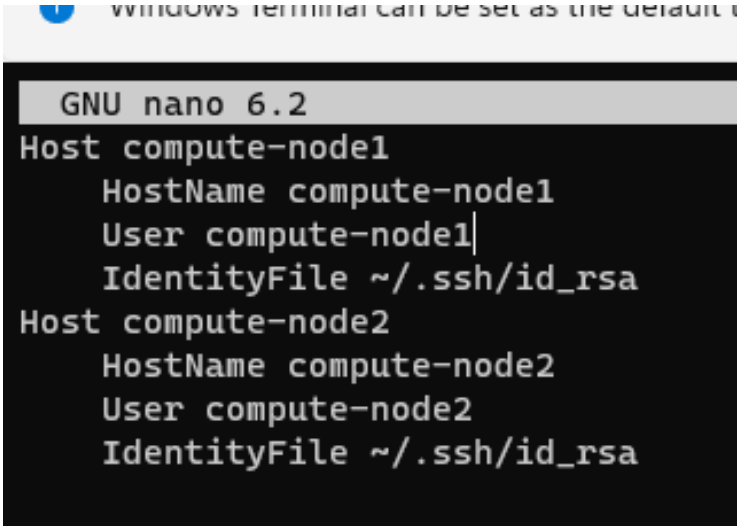

Figure 2.11: SSH Configuration

Test Connectivity

Ensure all nodes can **ping each other**:

```
ping compute-node1
ping compute-node2
```

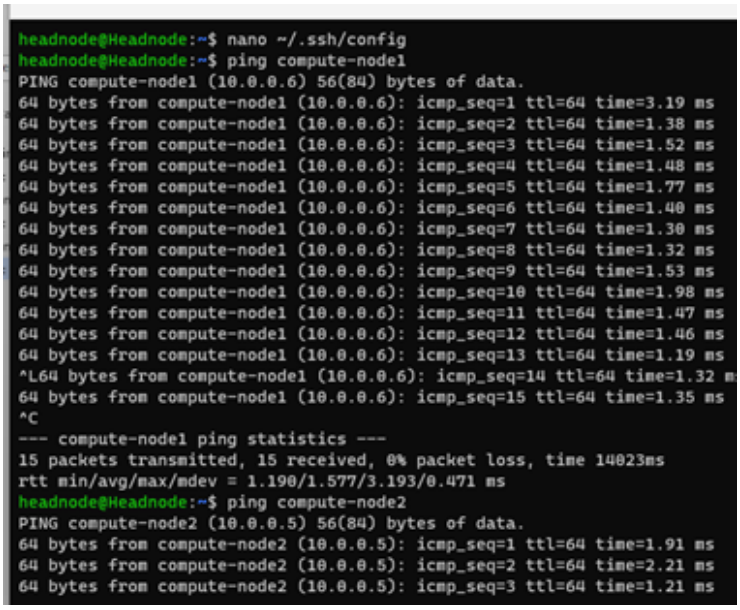

Figure 2.12: Testing Connectivity Between Nodes

Enable SSH Passwordless Login

On the head node, generate an SSH key:

```
1 ssh-keygen -t rsa -b 4096
```

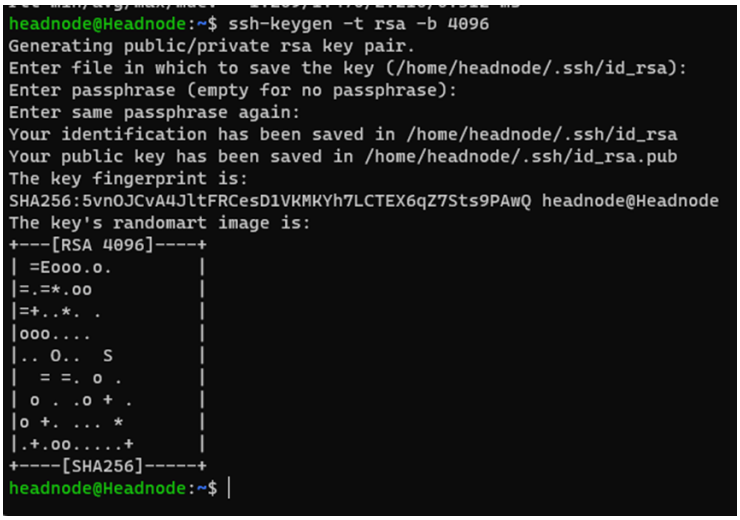

Figure 2.13: Generate an SSH key

Copy the key to compute nodes:

```
ssh-copy-id compute-node1
ssh-copy-id compute-node2
```

Figure 2.14: Setting Up Passwordless SSH Access

2.3.2 Installing and Testing MPI

OpenMPI was installed and configured according to official documentation The Open MPI Project (2024).

Install OpenMPI on All the 3 Nodes

```
sudo apt install -y openmpi-bin openmpi-common openmpi
    -doc libopenmpi-dev
```

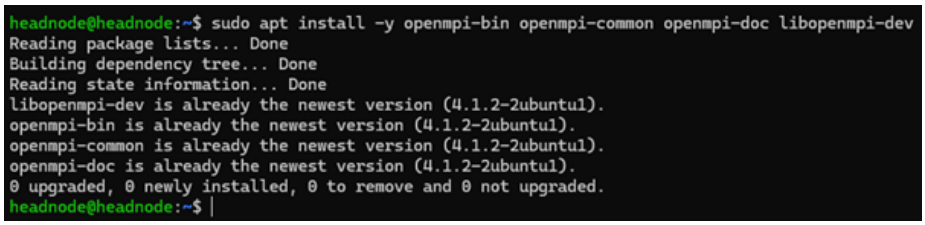

Figure 2.15: Installing OpenMPI on Nodes

MPI User (mpiuser - Recommended)

For running MPI jobs, it is best to create a non-root user with SSH access across nodes.

Create an MPI user (all nodes):

```
sudo useradd -m -s /bin/bash mpiuser
sudo passwd mpiuser
```

Give sudo privileges if needed:

```
sudo usermod -aG sudo mpiuser
```

Allow passwordless SSH for mpiuser (only Headnode):

```
sudo su - mpiuser
ssh-keygen -t rsa -b 4096
ssh-copy-id compute-node1
ssh-copy-id compute-node2
```

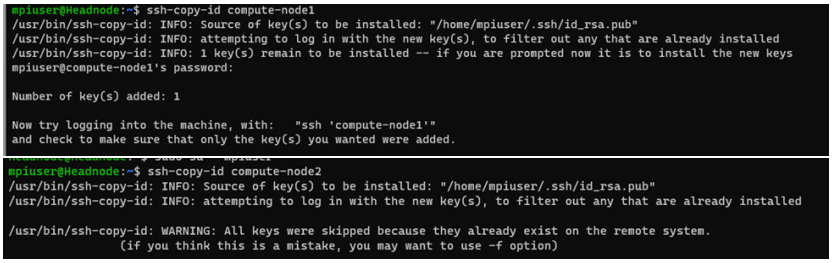

Figure 2.16: Setting Up MPI User with SSH Access

2.3.3 Setting Up MPI Programs

For this project, we will evaluate the cluster with 2 prepared MPI Programs (**pingpong.c** and **mat_mat.c**)

Create `mpi_hosts` File

On headnode, run:

```
nano mpi_hosts
```

Add:

```
compute-node1 slots=1
compute-node2 slots=1
```

Figure 2.17: Creating MPI Hosts File

Save and exit.

Set up pingpong.c:

The **pingpong.c** program tests communication speed between MPI processes in a circle pattern. It sends messages of increasing sizes (1 byte to 1MB) where each process receives from one neighbor and sends to another. The program measures how long it takes for messages to complete a full circle (latency) and how much data can be transferred per second (bandwidth). After 10 warmup rounds and 100 test rounds for each message size, it saves the performance results to a CSV file showing how speed changes with larger messages.

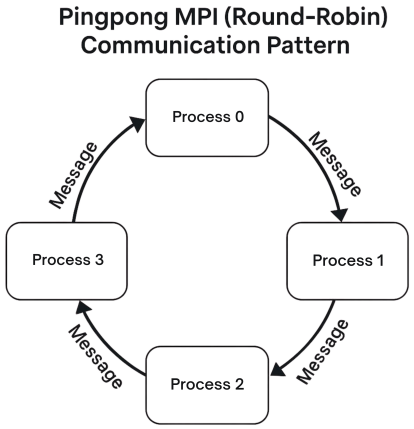

Figure 2.18: **pingpong.c** MPI Program Diagram

The **pingpong.c** source file can also be found here: `https://github.com/ThongLai/Cluster-and-Cloud-Benchmarking/blob/main/pingpong.c` Lai (2024)

On headnode, use ‘wget‘ to download the file **pingpong.c** from the above github repo source:

```
wget https://raw.githubusercontent.com/ThongLai/
    Cluster-and-Cloud-Benchmarking/refs/heads/main/
    pingpong.c
```

Figure 2.19: Obtaining the pingpong.c File

Compile the MPI Program

```
mpicc pingpong.c -o pingpong
chmod +x pingpong
```

Make sure the compiled file have the (`x`) execute permission

```
ls -l
```

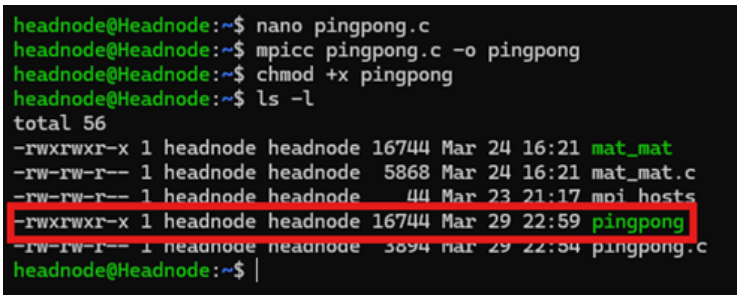

Figure 2.20: Compiling the **pingpong** MPI Program

Copy pingpong to Compute Nodes

```
scp pingpong compute-node1:~
scp pingpong compute-node2:~
```

Figure 2.21: Copying **pingpong** to Compute Nodes

Create and run mat_mat.c:

The **mat_mat.c** program tests parallel matrix multiplication using MPI. It works with matrices of increasing sizes (*2×2* to *1024×1024*) and divides the work among multiple processes. Each process receives a portion of matrix B, gets a complete copy of matrix C, calculates its part of the result, and sends it back. The program measures how long the multiplication takes (latency) and the data transfer rate (bandwidth) for each matrix size. Results are saved to a CSV file showing how performance scales with larger matrices.

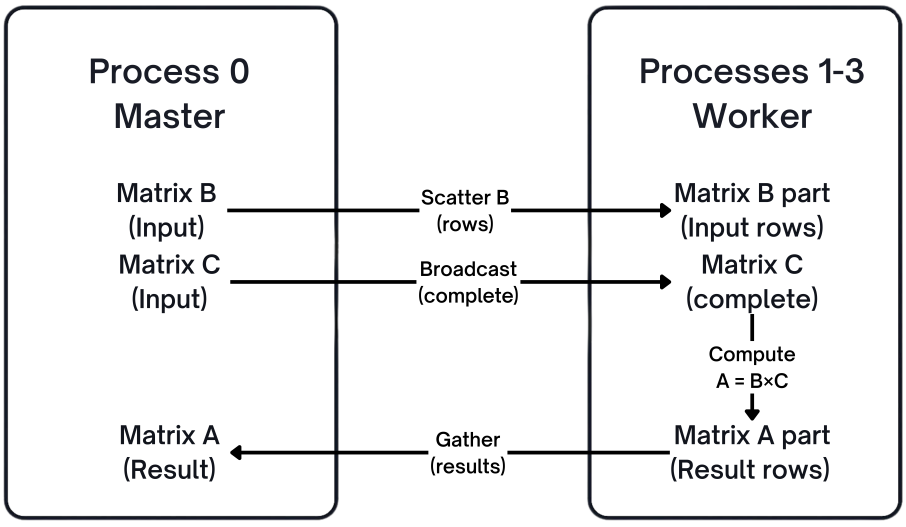

Figure 2.22: **mat_mat.c** MPI Program Diagram

The **mat_mat.c** source file can also be found here: `https://github.com/ThongLai/Cluster-and-Cloud-Benchmarking/blob/main/mat_mat.c` Lai (2024)

Set up mat_mat.c:

On headnode, use ‘wget‘ to download the file **mat_mat.c** from the above github repo source:

```
wget https://raw.githubusercontent.com/ThongLai/
    Cluster-and-Cloud-Benchmarking/refs/heads/main/
    mat_mat.c
```

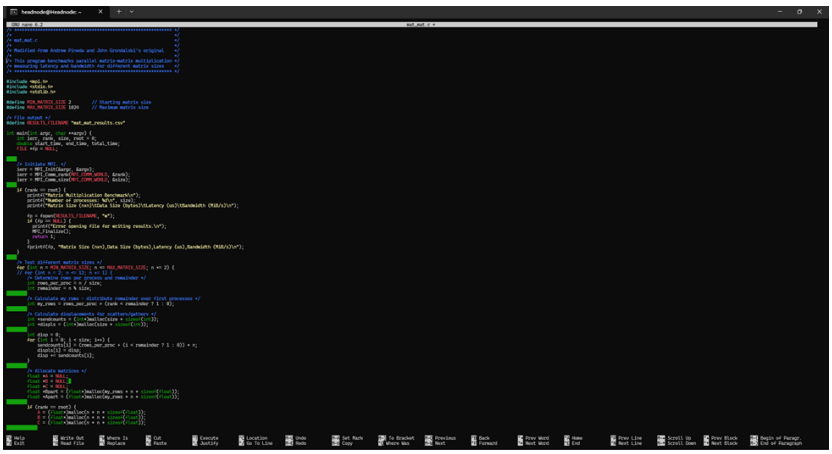

Figure 2.23: Creating the **mat_mat.c** File

Compile the MPI Program

```
mpicc mat_mat.c -o mat_mat
chmod +x mat_mat
```

Make sure the compiled file have the (`x`) execute permission

```
ls -l
```

```
headnode@Headnode:~$ nano mat_mat.c
headnode@Headnode:~$ mpicc mat_mat.c -o mat_mat
headnode@Headnode:~$ chmod +x mat_mat
headnode@Headnode:~$ ls -l
total 56
-rwxrwxr-x 1 headnode headnode 16744 Mar 29 23:20 mat_mat
-rw-rw-r-- 1 headnode headnode  5868 Mar 29 23:20 mat_mat.c
-rw-rw-r-- 1 headnode headnode    44 Mar 23 21:17 mpi_hosts
-rwxrwxr-x 1 headnode headnode 16744 Mar 29 22:59 pingpong
-rw-rw-r-- 1 headnode headnode  3894 Mar 29 22:54 pingpong.c
headnode@Headnode:~$
```

Figure 2.24: Compiling the **mat_mat** MPI Program

Copy mat_mat to Compute Nodes

```
scp mat_mat compute-node1:~
scp mat_mat compute-node2:~
```

Figure 2.25: Copying mat_mat to Compute Nodes

2.4 Running MPI Programs and collect results

This section will run the prepared MPI programs above with different cluster settings scenarios.

Running pingpong MPI Program

With 2 processes and 1 slot per node:

`mpi_hosts` configuration:

```
compute-node1 slots=1
compute-node2 slots=1
```

Running:

```
mpirun -np 2 --hostfile mpi_hosts ./pingpong
```

```
headnode@Headnode:~$ mpirun -np 2 --hostfile mpi_hosts ./pingpong
Pingpong Benchmark (Round-Robin Implementation)
Number of processes: 2
Message Size (bytes)    Latency (us)    Bandwidth (MiB/s)
                   1          667.60                0.00
                   2          560.90                0.00
                   4          509.21                0.01
                   8          537.18                0.01
                  16          627.23                0.02
                  32          583.17                0.05
                  64          570.04                0.11
                 128          594.62                0.21
                 256          492.49                0.50
                 512          615.61                0.79
                1024          656.66                1.49
                2048          790.29                2.47
                4096          639.41                6.11
                8192          729.15               10.71
               16384          886.12               17.63
               32768         1050.40               29.75
               65536         2575.60               24.27
              131072         3662.81               34.13
              262144         4038.71               61.90
              524288         6116.17               81.75
             1048576        10333.54               96.77
Results saved to /home/compute-node1/pingpong_results.csv
```

Figure 2.26: Running **pingpong** with 2 processes, 1 slot per node

Save the result:

In another terminal, perform this to download the `pingpong_p2_s1.csv` result file for this benchmark:

```
scp -J headnode@172.166.178.255 compute-node1@10
    .0.0.6:pingpong_results.csv pingpong_p2_s1.csv
```

```
D:\ThongLai\OneDrive - University of Huddersfield\Parallel Computer Architecture Clusters and Grids\Cluster-and-Cloud-Be
nchmarking\data_results>scp -J headnode@172.166.178.255 compute-node1@10.0.0.6:pingpong_results.csv pingpong_p2_s1.csv
headnode@172.166.178.255's password:
compute-node1@10.0.0.6's password:
pingpong_results.csv                                          100%  583     9.0KB/s   00:00
```

Figure 2.27: Saving **pingpong** results file

Running mat_mat MPI Program

With 2 processes and 1 slot per node:

`mpi_hosts` configuration:

```
compute-node1 slots=1
compute-node2 slots=1
```

Running:

```
mpirun -np 2 --hostfile mpi_hosts ./mat_mat
```

```
headnode@Headnode:~$ mpirun -np 2 --hostfile mpi_hosts ./mat_mat
Matrix Multiplication Benchmark
Number of processes: 2
Matrix Size (nxn)       Data Size (bytes)       Latency (us)    Bandwidth (MiB/s)
       2x2                            32             369.11                0.00
       4x4                           128             274.51                0.00
       8x8                           512             394.61                0.00
     16x16                          2048             389.21                0.00
     32x32                          8192             963.02                0.00
     64x64                         32768            3589.38                0.00
   128x128                        131072            8474.40                0.00
   256x256                        524288           51309.06                0.00
   512x512                       2097152          587762.00                3.40
  1024x1024                      8388608         5344281.03                1.50
Results saved to /home/compute-node1/mat_mat_results.csv
```

Figure 2.28: Running mat_mat with 2 processes, 1 slot per node

Save the result:

In another terminal, perform this to download the `mat_mat_p2_s1.csv` result file for this benchmark:

```
scp -J headnode@172.166.178.255 compute-node1@10
    .0.0.6:mat_mat_results.csv mat_mat_p2_s1.csv
```

```
D:\ThongLai\OneDrive - University of Huddersfield\Parallel Computer Architecture Clusters and Grids\Cluster-and-Cloud-Be
nchmarking\data_results>scp -J headnode@172.166.178.255 compute-node1@10.0.0.6:mat_mat_results.csv mat_mat_p2_s1.csv
headnode@172.166.178.255's password:
compute-node1@10.0.0.6's password:
mat_mat_results.csv                                           100%  367     3.4KB/s   00:00
```

Figure 2.29: Saving mat_mat results file

Do the same process for benchmarking with these different number of processes and slots for both MPI programs:

- **2 processes** and **2 slots** per node
- **4 processes** and **2 slots** per node

Specifically, the results benchmarking data will be collected as below:

Program	Total Processes	Slots per Node	Output File
pingpong	2	1	pingpong_p2_s1.csv
pingpong	2	2	pingpong_p2_s2.csv
pingpong	4	2	pingpong_p4_s2.csv
mat_mat	2	1	mat_mat_p2_s1.csv
mat_mat	2	2	mat_mat_p2_s2.csv
mat_mat	4	2	mat_mat_p4_s2.csv

Table 2.1: Processes and Output Files for Different Cluster Settings

All the data results can also be found in: `https://github.com/ThongLai/Cluster-and-Cloud-Benchmarking/blob/main/data_results` Lai (2024)

Chapter 3

Deploying and Executing on Different VM Clusters

This chapter will cover running the MPI programs on my colleague **Hatim Moaiyadi** (U2187171)'s Azure VMs with similar settings (headnode, compute-node1 and compute-node2), then collecting results from the `pingpong.c` and `mat_mat.c` programs for benchmark comparisons.

3.1 Adding a New User

First, we need **Hatim**'s administrative permission to add a new user (`tom`).

Connect to Hatim's Head Node

```
ssh headnode@20.68.2.132
```

Create a new user "tom" (all nodes):

```
sudo useradd -m -s /bin/bash tom
sudo passwd tom
```

Check all users in the system:

```
cat /etc/passwd
```

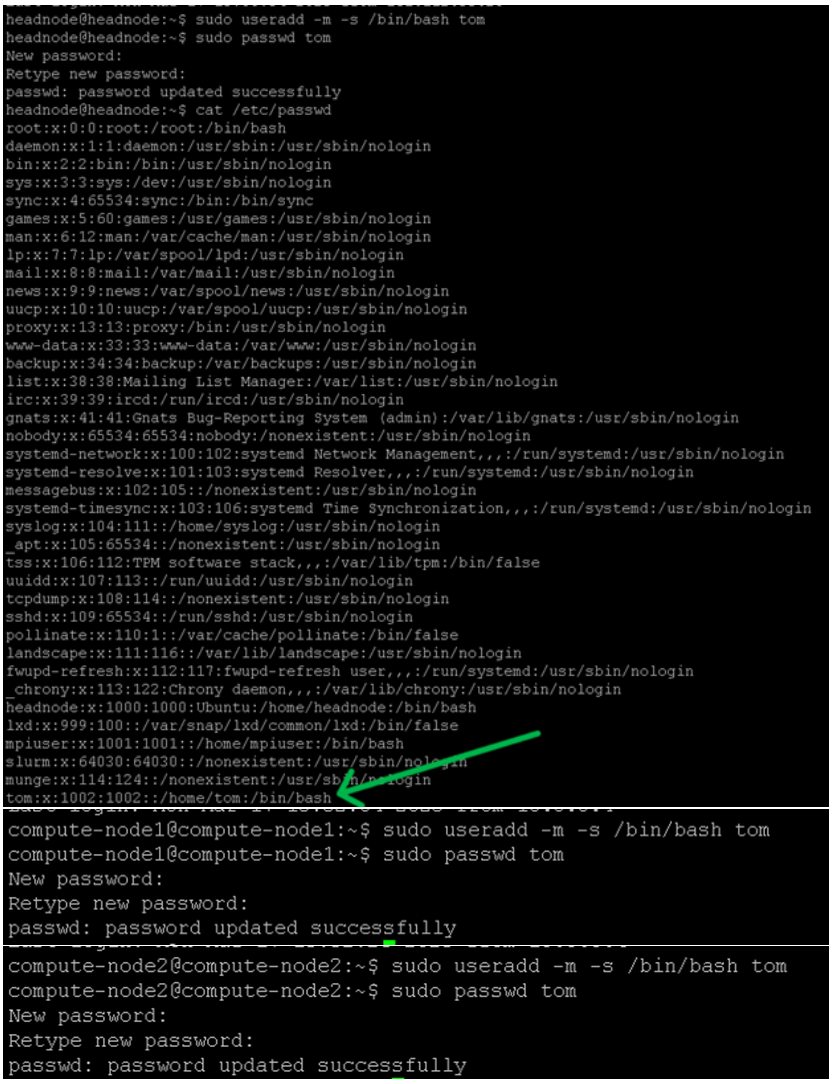

Figure 3.1: Creating user ‘tom‘ and checking system users

Give sudo privileges if needed:

```
sudo usermod -aG sudo tom
```

Allow passwordless SSH for tom (only Headnode):

```
sudo su - tom
ssh-keygen -t rsa -b 4096
```

```
headnode@headnode:~$ sudo su - tom
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

tom@headnode:~$ ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/home/tom/.ssh/id_rsa):
Created directory '/home/tom/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/tom/.ssh/id_rsa
Your public key has been saved in /home/tom/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:19Dr45KtXVY3c3uytiTgqdGoWGOCMIJXsVHYsjIp8r4 tom@headnode
The key's randomart image is:
+---[RSA 4096]----+
|    o+.          |
|    o+.    .     |
|   .oo    . .    |
|+ +..      o .   |
|*o.o    S o o  o+|
|.+..     = +   .*|
| .. . + o +o+ =..|
|  .  = o ooo.*.o.|
|  E.. . . .oo.o. |
+----[SHA256]-----+
```

Figure 3.2: Setting up passwordless SSH for user tom

```
1 ssh-copy-id compute-node1
2 ssh-copy-id compute-node2
```

```
tom@headnode:~$ ssh-copy-id compute-node1
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/tom/.ssh/id_rsa.pub"
The authenticity of host 'compute-node1 (10.0.0.5)' can't be established.
ED25519 key fingerprint is SHA256:5XAgXbAwIEtvQmyCmES+8Dz9JbP+1TTVjBHRdeg/kZk.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
tom@compute-node1's password:
Permission denied, please try again.
tom@compute-node1's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'compute-node1'"
and check to make sure that only the key(s) you wanted were added.

tom@headnode:~$ ssh-copy-id compute-node2
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/tom/.ssh/id_rsa.pub"
The authenticity of host 'compute-node2 (10.0.0.6)' can't be established.
ED25519 key fingerprint is SHA256:0j6lXPt4HhzNOOBFaDCqjxPquzGTR4myxsRJwWh5Z8I.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
tom@compute-node2's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'compute-node2'"
and check to make sure that only the key(s) you wanted were added.
```

Figure 3.3: Enter Caption

3.2 Setting Up MPI Programs

This section will demonstrate running and collecting results from the 2 prepared MPI programs (**pingpong.c** and **mat_mat.c**) on **Hatim**'s clusters.

Creating `mpi_hosts` File

On headnode, run:

```
nano mpi_hosts
```

Add:

```
compute-node1 slots=1
compute-node2 slots=1
```

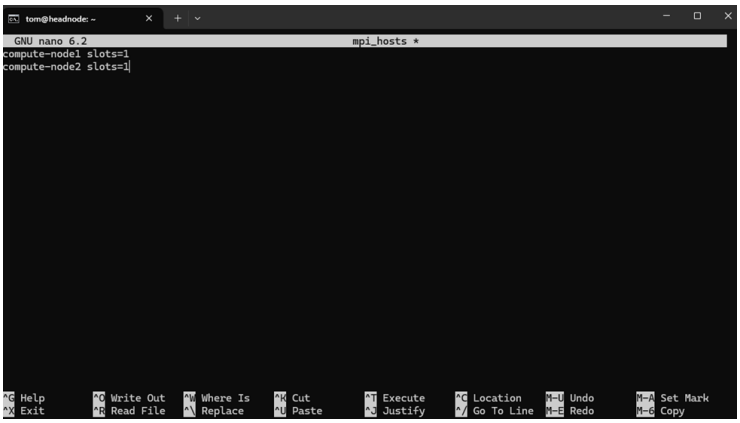

Figure 3.4: Creating mpi_hosts file on Hatim's cluster

Save and exit.

Setting Up `pingpong.c`

The pingpong.c source file can be found here: `https://github.com/ThongLai/Cluster-and-Cloud-Benchmarking/blob/main/pingpong.c` Lai (2024)

On headnode, use `wget` to download the file **pingpong.c** from the GitHub repository:

```
wget https://raw.githubusercontent.com/ThongLai/
    Cluster-and-Cloud-Benchmarking/refs/heads/main/
    pingpong.c
```

Check to see whether the file has downloaded successfully:

```
ls
```

Figure 3.5: Downloading pingpong.c file

Compile the MPI Program

```
mpicc pingpong.c -o pingpong
chmod +x pingpong
```

Copy pingpong to Compute Nodes

```
scp pingpong compute-node1:~
scp pingpong compute-node2:~
```

Figure 3.6: Compiling and copying pingpong to compute nodes

Setting Up `mat_mat.c`

The **mat_mat.c** source file can be found here: `https://github.com/ThongLai/Cluster-and-Cloud-Benchmarking/blob/main/mat_mat.c` Lai (2024)

On headnode, use `wget` to download the file **mat_mat.c** from the GitHub repository:

```
wget https://raw.githubusercontent.com/ThongLai/
    Cluster-and-Cloud-Benchmarking/refs/heads/main/
    mat_mat.c
```

Check to see whether the file has downloaded successfully:

```
ls
```

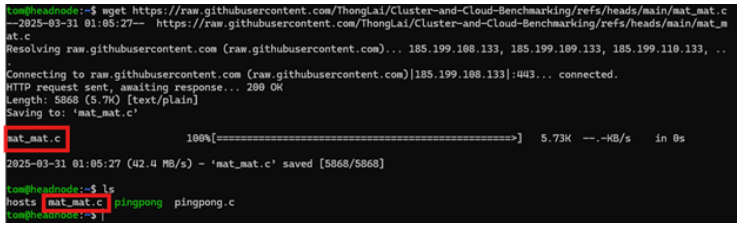

Figure 3.7: Downloading **mat_mat.c** file

Compile the MPI Program

```
mpicc mat_mat.c -o mat_mat
chmod +x mat_mat
```

Copy mat_mat to Compute Nodes

```
scp mat_mat compute-node1:~
scp mat_mat compute-node2:~
```

Figure 3.8: Compiling and copying **mat_mat** to compute nodes

3.3 Running MPI Programs and collect results

This section will run the prepared MPI programs with different cluster settings scenarios using **Hatim**'s VMs (as `tom` user).

Running `pingpong` MPI Program

With 2 processes and 1 slot per node:

`mpi_hosts` configuration:

```
compute-node1 slots=1
compute-node2 slots=1
```

Running:

```
mpirun -np 2 --hostfile mpi_hosts ./pingpong
```

```
tom@headnode:~$ mpirun -np 2 --hostfile mpi_hosts ./pingpong
Pingpong Benchmark (Round-Robin Implementation)
Number of processes: 2
Message Size (bytes)    Latency (us)    Bandwidth (MiB/s)
                   1         1309.17                0.00
                   2         1225.16                0.00
                   4         1321.34                0.00
                   8         1150.31                0.01
                  16         1217.44                0.01
                  32         1185.06                0.03
                  64         1256.60                0.05
                 128         1262.25                0.10
                 256         1211.65                0.20
                 512         1168.60                0.42
                1024         1261.39                0.77
                2048         1320.61                1.48
                4096         1313.84                2.97
                8192         1418.36                5.51
               16384         1561.31               10.01
               32768         1638.17               19.08
               65536         4531.60               13.79
              131072         5400.07               23.15
              262144         6488.45               38.53
              524288         9219.91               54.23
             1048576        15251.70               65.57
Results saved to /home/tom/pingpong_results.csv
```

Figure 3.9: Running pingpong with 2 processes, 1 slot per node on **Hatim**'s cluster

Save the result:

In another terminal, perform this to download the `pingpong_p2_s1_vm2.csv` result file for this benchmark:

```
scp -J tom@20.68.2.132 tom@10.0.0.5:pingpong_results.
    csv pingpong_p2_s1_vm2.csv
```

```
D:\ThongLai\OneDrive - University of Huddersfield\Parallel Computer Architecture Clusters and Grids\Cluster-and-Cloud-Be
nchmarking\data_results>scp -J tom@20.68.2.132 tom@10.0.0.5:pingpong_results.csv pingpong_p2_s1_vm2.csv
tom@20.68.2.132's password:
tom@10.0.0.5's password:
pingpong_results.csv                                          100%  597     5.4KB/s   00:00
```

Figure 3.10: Saving **pingpong** results from **Hatim**'s cluster

Running `mat_mat` MPI Program

With 2 processes and 1 slot per node:

`mpi_hosts` configuration:

```
compute-node1 slots=1
compute-node2 slots=1
```

Running:

```
mpirun -np 2 --hostfile mpi_hosts ./mat_mat
```

```
tom@headnode:~$ mpirun -np 2 --hostfile mpi_hosts ./mat_mat
Matrix Multiplication Benchmark
Number of processes: 2
Matrix Size (nxn)       Data Size (bytes)       Latency (us)    Bandwidth (MiB/s)
      2x2                            32            2238.09                0.00
      4x4                           128            1453.09                0.00
      8x8                           512            1101.29                0.00
    16x16                          2048            2485.19                0.00
    32x32                          8192            1611.79                0.00
    64x64                         32768            3888.18                0.00
  128x128                        131072           14404.12                0.00
  256x256                        524288           88918.32                0.00
  512x512                       2097152          790722.80                2.53
 1024x1024                      8388608         5802190.58                1.38
Results saved to /home/tom/mat_mat_results.csv
```

Figure 3.11: Running mat_mat with 2 processes, 1 slot per node on Hatim's cluster

Save the result:

In another terminal, perform this to download the `mat_mat_p2_s1_vm2.csv` result file for this benchmark:

```
scp -J tom@20.68.2.132 tom@10.0.0.5:mat_mat_results.
    csv mat_mat_p2_s1_vm2.csv
```

```
D:\ThongLai\OneDrive - University of Huddersfield\Parallel Computer Architecture Clusters and Grids\Cluster-and-Cloud-Be
nchmarking\data_results>scp -J tom@20.68.2.132 tom@10.0.0.5:mat_mat_results.csv mat_mat_p2_s1_vm2.csv
tom@20.68.2.132's password:
tom@10.0.0.5's password:
mat_mat_results.csv                                           100%  376     3.0KB/s   00:00
```

Figure 3.12: Saving **mat_mat** results from **Hatim**'s cluster

Do the same process for benchmarking with these different number of processes and slots for both MPI programs using Hatim's VMs (as `tom` user):

- **2 processes** and **2 slots** per node
- **4 processes** and **2 slots** per node

Specifically, the results benchmarking data will be collected as below:

Program	Total Processes	Slots per Node	Output File
pingpong	2	1	pingpong_p2_s1_vm2.csv
pingpong	2	2	pingpong_p2_s2_vm2.csv
pingpong	4	2	pingpong_p4_s2_vm2.csv
mat_mat	2	1	mat_mat_p2_s1_vm2.csv
mat_mat	2	2	mat_mat_p2_s2_vm2.csv
mat_mat	4	2	mat_mat_p4_s2_vm2.csv

Table 3.1: Processes and Output Files for Different Cluster Settings

All the data results can also be found in: `https://github.com/ThongLai/Cluster-and-Cloud-Benchmarking/blob/main/data_results` Lai (2024)

Chapter 4

Data Analysis and Visualisation

This chapter presents the analysis of benchmark results collected from running MPI programs on two different VM clusters: my own Azure VM setup and **Hatim Moaiyadi**'s VM cluster with similar configuration. The analysis compares performance metrics across three different cluster configurations:

- **p2_s1**: 2 processes, 1 slot per node
- **p2_s2**: 2 processes, 2 slots per node
- **p4_s2**: 4 processes, 2 slots per node

The data collected includes:

- For `pingpong`: **Message Size** (bytes), **Latency** (μs), and **Bandwidth** (MiB/s)
- For `mat_mat`: **Matrix Size** ($n \times n$), **Data Size** (bytes), **Latency** (μs), and **Bandwidth** (MiB/s)

4.1 Communication Performance (Pingpong Tests)

The `pingpong` benchmark measures the network communication performance between compute nodes. This section analyzes both latency and bandwidth results across different configurations.

4.1.1 Latency Analysis

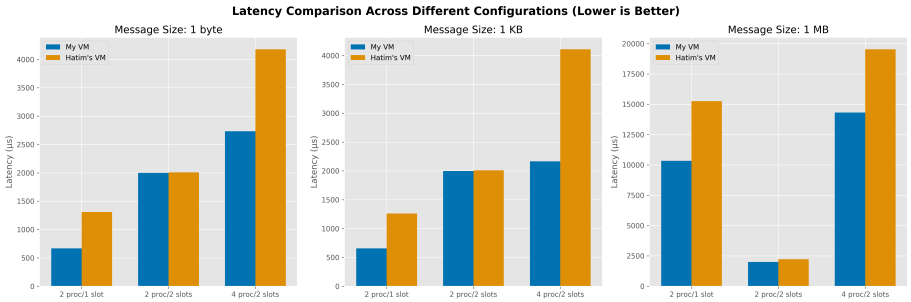

Figure 4.1: Latency comparison across different configurations and VMs

Key findings from the latency analysis:

- My VM consistently delivers messages faster than **Hatim**'s VM across all tested configurations.
- In both VMs, the **p2_s2** configuration maintains consistent message delivery time (around *2000μs*) with **small** to **medium** messages (*1B-1KB*) and very low latency with **big** messages (*1MB*), while other setups show increasing delays.
- Using **p4_s2** results in higher latency than **p2_s1** and **p2_s2**, indicating increased communication overhead with more processes.
- The **p2_s1** configuration provides the best overall performance. There is a size threshold where messages larger than *65KB* show a significant increase in delivery time for the **p2_s1** and **p4_s2** configurations:

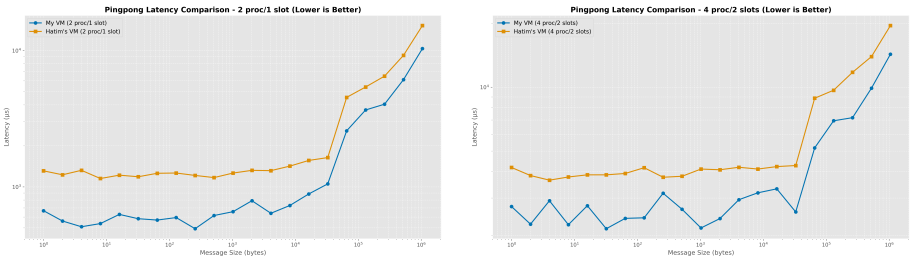

Figure 4.2: Pingpong latency Comparison of **p2_s1** and **p4_s2**

4.1.2 Bandwidth Analysis

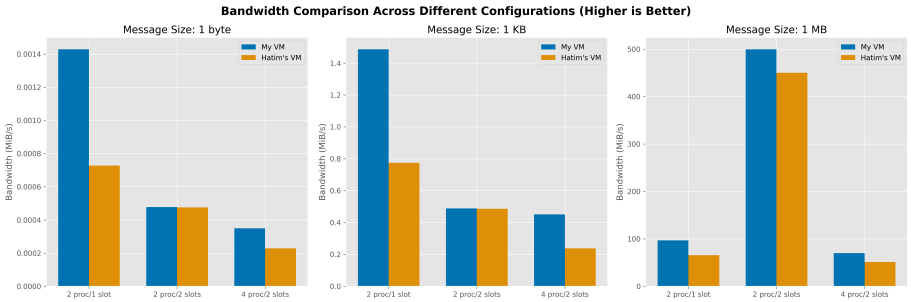

Figure 4.3: Bandwidth comparison across different configurations and VMs

Key findings from the bandwidth analysis:

- My VM achieves approximately **15-20%** higher bandwidth than **Hatim**'s VM.
- For large messages, the **p2_s2** is the best configuration, achieving significantly higher data transfer rates (approximately *500 MiB/s*) compared to other configurations (only *70-100 MiB/s*).
- All configurations show improved transfer rates with larger messages, but the **p2_s2** setup improves most dramatically:

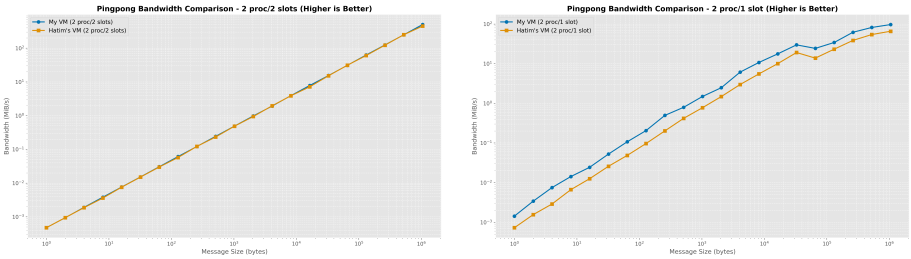

Figure 4.4: Pingpong bandwidth Comparison of **p2_s2** and **p2_s1**

4.2 Computational Performance (Matrix Multiplication)

The `mat_mat` benchmark assesses computational performance through parallel matrix multiplication. This section analyzes how different configurations affect processing time and efficiency.

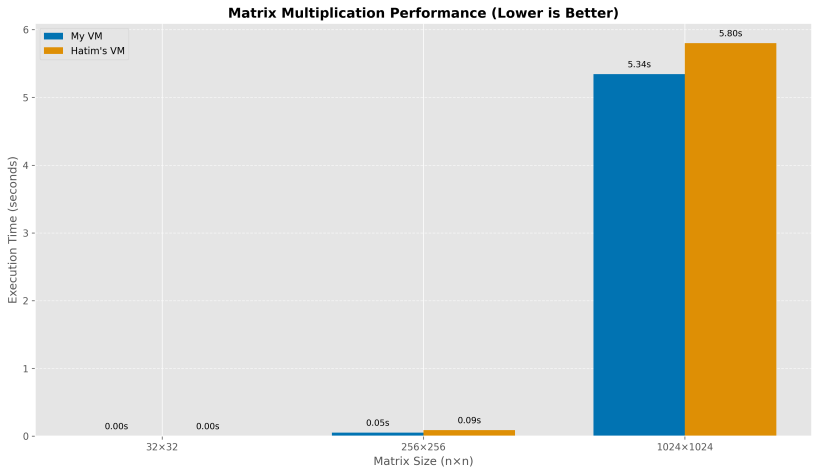

Figure 4.5: Matrix multiplication processing time across different VMs

Key findings from the matrix multiplication analysis:

- Processing time increases substantially as matrix size grows across all configurations.
- For **small** matrices, the performance differences between configurations are **minimal**.

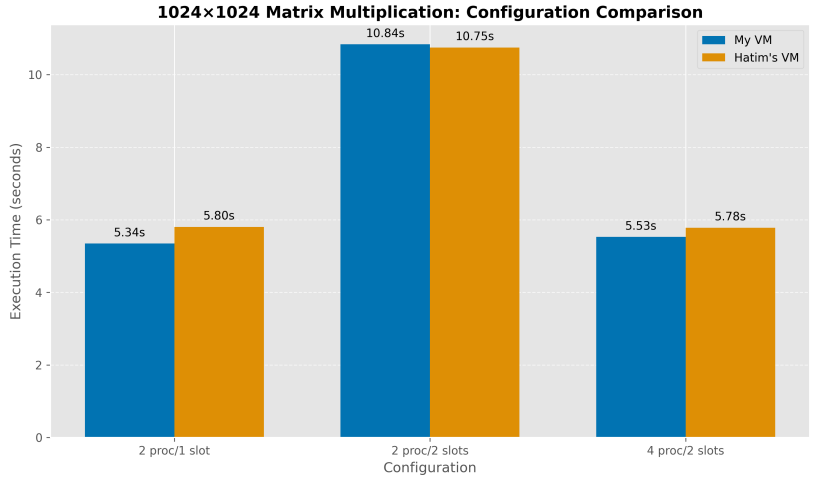

Figure 4.6: Large Matrix multiplication (**1024×1024**) processing time across different configurations and VMs

- Configuration performance for **1024×1024** matrices:
  - **p2_s1** performs best (about *5.3M* μs on my VM)
  - **p2_s2** performs worst (about *10.8M* μs on my VM)
  - **p4_s2** falls in between (about *5.5M* μs on my VM)

4.3 Key Insights and Recommendations

Based on the analysis of both communication and computational benchmarks, the following insights and recommendations can be drawn:

For Communication-Focused (Pingpong) Programs

- Utilize all available slots for each process (like **p2_s2**) when high data transfer rates are important.
- Use a lower number of slots (like **p2_s1**) when quick responses for small messages are crucial.

For Calculation-Focused (Matrix Multiplication) Programs

- The **p2_s1** configuration consistently provides the best performance.
- Adding more processes or slots actually reduces calculation performance.

These findings underscore the importance of selecting the appropriate cluster configuration based on the specific computational needs of the application. Communication-heavy applications benefit from different configurations than computation-intensive workloads.

Chapter 5

Conclusion

This project has demonstrated the successful deployment and benchmarking of a Beowulf cluster on Azure Virtual Machines. Through systematic testing of different process and slot configurations, we have identified optimal settings for various computational workloads and developed evidence-based recommendations for efficient cluster deployments.

Our benchmarking approach measures latency and bandwidth, similar to established benchmarks in high-performance computing Dongarra et al. (2003).

Key Findings

Our experimentation with different process and slot configurations revealed significant performance variations depending on the computational task:

- **For Communication-Intensive Tasks:**
  - Using two processes with two slots on a single node (`p2_s2`) provides optimal bandwidth for large messages, achieving transfer rates up to 500 MiB/s.
  - Two processes distributed across nodes (`p2_s1`) delivers superior latency performance for small to medium-sized messages, making it ideal for applications requiring frequent communication with small data packets.
- **For Computation-Intensive Tasks:**
  - The `p2_s1` configuration consistently delivered the best performance for matrix multiplication, completing operations on large matrices approximately *50%* faster than the worst-performing configuration.

  - Contrary to intuition, adding more processes (`p4_s2`) actually reduced computational efficiency in most tests, highlighting the importance of proper configuration over simply increasing resources.

Performance Comparison

The performance comparison between different VM clusters (my own and **Hatim Moaiyadi**'s) revealed consistent patterns despite absolute differences in performance metrics. This consistency supports the generalizability of our findings across similar Azure VM deployments with comparable configurations.

Future Work

Future research could expand on this work by:

- Implementing and benchmarking additional parallel programming paradigms beyond MPI
- Testing with larger clusters (8+ nodes) to evaluate scalability limits
- Comparing performance across different VM types with varying CPU/memory configurations
- Exploring containerized deployment options for improved portability

Professional Development

As part of this project, the following professional certification course was completed:

- **Microsoft Azure Fundamentals (AZ-900) Cert Prep: 1 Cloud Concepts** - Certification URL

These provided valuable context for understanding cloud infrastructure principles that directly supported the implementation of this project.

Bibliography

Dongarra, J. J., Luszczek, P., and Petitet, A. (2003). The linpack benchmark: Past, present and future. *Concurrency and Computation: Practice and Experience*, 15(9):803–820.

Lai, T. (2024). Cluster and cloud benchmarking. `https://github.com/ThongLai/Cluster-and-Cloud-Benchmarking`. Accessed: March 2025.

Microsoft (2024). Azure virtual machines documentation. `https://docs.microsoft.com/en-us/azure/virtual-machines/`. Accessed: March 2025.

Pacheco, P. (1996). *Parallel Programming with MPI*. Morgan Kaufmann Publishers, San Francisco, CA.

Sterling, T., Savarese, D., Becker, D. J., Dorband, J. E., Ranawake, U. A., and Packer, C. V. (2001). *Beowulf Cluster Computing with Linux*. MIT Press, Cambridge, MA.

The Open MPI Project (2024). Open mpi: Open source high performance computing. Technical report, Open MPI. Accessed: March 2025.