Contents

Setting Up a Cluster for Parallel Computing

Created by: Manas Bavaskar | GitHub | LinkedIn | Email

Introduction

This is a guide for creating a 4-node Ubuntu cluster to run a parallel Bitonic Sort using MPI.

So, I will be explaining this through an example:

In our Example: - **Master Node**: Manages job scheduling, stores the MPI code, and shares it with worker nodes. - **Worker Nodes (3 nodes)**: Run the parallel sorting tasks.

For simplicity, we assume the following IP addresses:

- **Master Node**: `192.168.1.1`
- **Worker Node 1**: `192.168.1.2`
- **Worker Node 2**: `192.168.1.3`
- **Worker Node 3**: `192.168.1.4`

Before moving further into the cluster setup, it's important to know how to find your IP address, username and verify network connectivity.

Initial Setup: Determining IP Addresses and Verifying Connectivity

Finding Your IP Address

On each Ubuntu machine, open a terminal and run one of these commands:

- **Method 1: Using `hostname -I`**

  ```
  hostname -I
  ```

  This prints one or more IP addresses. The address on your primary network interface (often the first one) is what you need.

- **Method 2: Using `ip addr show`**

  ```
  ip addr show
  ```

  Look for the section labeled `inet` under your network interface (e.g., `eth0` for wired or `wlan0` for wireless).

Finding Your System Username

On the Ubuntu machine, to find out your username

Open a terminal and run the command:

```
whoami
```

This will print out your username which we will later use during our SSH process

Testing Network Connectivity

To ensure that nodes can communicate:

1. **Ping another node:**

   ```
   ping 192.168.1.2
   ```

   If you see replies, then the node is reachable.

2. **Repeat on all nodes** to verify each machine can connect to the others.

Make sure to note the IP address of each machine for later configuration.

Step 1: Setting Up the Cluster

A cluster is a group of machines working together as one system.

In our example:

- **Master Node** handles tasks and stores code.
- **Worker Nodes** execute the parallel computations.

1.1 Install Required Packages

On every node (master and workers), open a terminal and install the following:

```
sudo apt update
sudo apt install mpich nfs-kernel-server nfs-common openssh-server -y
```

Why These Packages?

- **MPICH**: Provides the MPI implementation needed for parallel computing.
- **NFS-Kernel-Server**: Installed on the master node to create a shared directory.
- **NFS-Common**: Installed on worker nodes to mount the NFS share from the master.
- **OpenSSH-Server**: Enables secure SSH connections between nodes.

After installation, confirm that SSH is running:

```
sudo systemctl status ssh
```

If SSH isn't running, you can start it with:

```
sudo systemctl start ssh
```

Step 2: Setting Up SSH for Passwordless Login

Setting up SSH key-based authentication allows the master node to run commands on worker nodes without repeatedly entering passwords.

2.1 Generate SSH Keys on the Master Node

On the master node, run:

```
ssh-keygen -t rsa
```

- When prompted for a file location, press **Enter** to use the default (`~/.ssh/id_rsa`).
- Leave the passphrase empty by pressing **Enter** twice.

2.2 Copy SSH Keys to Worker Nodes

For each worker node, copy the public key from the master:

```
ssh-copy-id username@192.168.1.X
```

Replace X with the worker node's IP (e.g., 2, 3, or 4).
This command appends the master's public key to the worker's `~/.ssh/authorized_keys` file.

2.3 Test SSH Access

From the master node, test logging in without a password:

```
ssh username@192.168.1.2
```

You should be logged in immediately without a prompt for a password. This confirms that SSH key-based authentication is set up correctly.

Step 3: Creating a Shared Network Storage with NFS

A shared directory lets you store code, data, and binaries in one location accessible by every node.

3.1 Configure the NFS Server on the Master Node

1. **Create a Shared Directory:**

   ```
   sudo mkdir -p /home/shared
   sudo chmod 777 /home/shared
   ```

   - `mkdir -p`: Creates the directory; the `-p` flag ensures no errors if it already exists.
   - `chmod 777`: Grants full read, write, and execute permissions.

2. **Edit the NFS Exports File:** Open `/etc/exports` in your favorite editor:

   ```
   sudo nano /etc/exports
   ```

   Add the following line:

   ```
   /home/shared 192.168.1.0/24(rw,sync,no_root_squash,no_subtree_check)
   ```

   This line tells the server to allow any machine on the 192.168.1.x network to read and write to `/home/shared`.

3. **Restart the NFS Service:**

   ```
   sudo systemctl restart nfs-kernel-server
   ```

Why Use NFS?

- **Shared Access**: Ensures every node uses the same set of files.
- **Consistency**: Any changes made on the master are immediately visible to all workers.
- **Simplified Management**: Only one copy of the code or data is maintained.

3.2 Mount the NFS Share on Worker Nodes

On each worker node:

1. **Create a Mount Point:**

   ```
   sudo mkdir -p /home/shared
   ```

2. **Mount the Shared Directory:**

   ```
   sudo mount 192.168.1.1:/home/shared /home/shared
   ```

   This command tells the worker node to connect to the shared directory on the master node.

3. **Make the Mount Permanent (Optional):** Edit the `/etc/fstab` file to auto-mount at boot:

```
sudo nano /etc/fstab
```

Add:

```
192.168.1.1:/home/shared /home/shared nfs defaults 0 0
```

4. **Verify the Mount:**

```
df -h
```

Look for `/home/shared` in the output to confirm it is mounted correctly.

Step 4: Running Parallel Bitonic Sort

With the cluster set up and shared storage in place, we can now store, compile, and execute our MPI code.

4.1 Store MPI Code in the Shared Directory

On the master node, create the MPI code file:

```
nano /home/shared/bitonic_sort.cpp
```

Paste your Bitonic Sort MPI code into this file. Here's a simplified version for reference:

```cpp
#include <mpi.h>
#include <iostream>
#include <vector>
#include <algorithm>
#include <cstdlib>
#include <ctime>

#define ARRAY_SIZE 1000000

using namespace std;

void bitonic_merge(vector<int>& arr, int low, int cnt, bool ascending) {
    if (cnt > 1) {
        int mid = cnt / 2;
        for (int i = low; i < low + mid; i++) {
            if ((arr[i] > arr[i + mid]) == ascending)
                swap(arr[i], arr[i + mid]);
        }
        bitonic_merge(arr, low, mid, ascending);
        bitonic_merge(arr, low + mid, mid, ascending);
    }
}

void bitonic_sort(vector<int>& arr, int low, int cnt, bool ascending) {
    if (cnt > 1) {
```

```
        int mid = cnt / 2;
        bitonic_sort(arr, low, mid, true);
        bitonic_sort(arr, low + mid, mid, false);
        bitonic_merge(arr, low, cnt, ascending);
    }
}

vector<int> generate_data(int size) {
    vector<int> data(size);
    for (int i = 0; i < size; i++)
        data[i] = rand() % 100000;
    return data;
}

int main(int argc, char** argv) {
    int rank, size;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    int local_size = ARRAY_SIZE / size;
    vector<int> local_data(local_size);

    if (rank == 0) {
        srand(time(NULL));
        vector<int> data = generate_data(ARRAY_SIZE);
        for (int i = 0; i < size; i++) {
            MPI_Send(&data[i * local_size], local_size, MPI_INT, i, 0, MPI_COMM_WOR
        }
    }
    MPI_Recv(local_data.data(), local_size, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STAT

    double start_time = MPI_Wtime();
    bitonic_sort(local_data, 0, local_size, true);
    double end_time = MPI_Wtime();

    vector<int> sorted_data;
    if (rank == 0) {
        sorted_data.resize(ARRAY_SIZE);
    }
    MPI_Gather(local_data.data(), local_size, MPI_INT, sorted_data.data(), local_si

    if (rank == 0) {
        cout << "Nodes: " << size << " | Execution Time: " << (end_time - start_tim
    }

    MPI_Finalize();
    return 0;
```

```
}
```

Compile the code from the shared directory:

```
mpic++ /home/shared/bitonic_sort.cpp -o /home/shared/bitonic_sort
```

This makes the executable available to all nodes via the shared NFS directory.

4.2 Prepare the MPI Hosts File

On the master node, create a file listing the IP addresses of all nodes:

```
nano ~/mpi_hosts
```

Enter:

```
192.168.1.1  slots=1
192.168.1.2  slots=1
192.168.1.3  slots=1
192.168.1.4  slots=1
```

This file tells MPI which nodes to use for parallel processing.

Alternatively, you can specify the host IPs directly in the `mpirun` command without using a hosts file:

```
mpirun -np 4 --hosts 192.168.1.1,192.168.1.2,192.168.1.3,192.168.1.4 /home/shared/b
```

This method allows for quick modifications without editing a separate hosts file.

4.3 Run the MPI Code

Run the MPI program using different numbers of processes (nodes):

```
mpirun -np 1 --hostfile ~/mpi_hosts /home/shared/bitonic_sort
mpirun -np 2 --hostfile ~/mpi_hosts /home/shared/bitonic_sort
mpirun -np 3 --hostfile ~/mpi_hosts /home/shared/bitonic_sort
mpirun -np 4 --hostfile ~/mpi_hosts /home/shared/bitonic_sort
```

You should see output that indicates the number of nodes and the execution time for each run.

4.4 Automate the Experiment

To simplify running multiple tests, create a script:

```
nano run_experiments.sh
```

Insert the following:

```
#!/bin/bash
for i in {1..4}
do
    echo "Running on $i nodes..."
    mpirun -np $i --hostfile ~/mpi_hosts /home/shared/bitonic_sort
```

```
    echo "--------------------------------"
done
```

Make the script executable:

```
chmod +x run_experiments.sh
```

Run the script:

```
./run_experiments.sh
```

Step 5: Observing Performance Gains

After running the experiments, you might see output similar to:

```
Nodes: 1 | Execution Time: 12.3 seconds
--------------------------------
Nodes: 2 | Execution Time: 6.8 seconds
--------------------------------
Nodes: 3 | Execution Time: 4.9 seconds
--------------------------------
Nodes: 4 | Execution Time: 3.2 seconds
```

These results show that as more nodes participate in the computation, the overall execution time decreases—demonstrating the benefits of parallel processing.

Contact Information

GitHub LinkedIn Email

For any issues or questions, please email at: mcb76593@gmail.com