File Transfer using MPI

Lab Report

Nguyen Manh Khoi
22BI13219

December 24, 2024

1 Introduction

Message Passing Interface (MPI) is a communication protocol that allows multiple processes to communicate with each other, typically in distributed computing environments. MPI is suitable for distributed systems where processes running on different machines communicate with each other.

The goal of this lab is to use MPI to develop a simple file transfer system where multiple processes (acting as client and server) communicate to transfer files. The system employs Python's `mpi4py` library to demonstrate this process.

2 Objectives

The objectives of this lab:

- Implement a file transfer system using MPI for communication.
- Demonstrate how multiple processes communicate to transfer a file from client to server.
- Implement a basic MPI client-server communication model using `mpi4py`.

3 Methodology

1. The **server** (rank 0) waits for a filename from the client (rank 1), then listens for file data sent by the client.
2. The **client** (rank 1) sends the filename to the server and then sends the file data in chunks to the server.
3. The server writes the received data into a file and saves it locally once all data has been received.
4. Communication between the server and client is carried out using MPI's `send()` and `recv()` methods.

4 System Design

The system consists of two main components: the **server** and the **client**. The server listens for incoming messages, including the filename and file data, while the client sends these messages. The design assumes two processes running on a single machine (for simplicity) but can be expanded to multiple machines in a distributed environment.

4.1 File Transfer Protocol

The file transfer protocol follows this sequence:

1. The client sends the file name to the server.
2. The server requests the file data and writes it to a new file.
3. The client sends the file data in chunks, which are received by the server.
4. The server acknowledges receipt and writes the data to the file.
5. The transfer ends when the client sends a "None" message signaling the end of data transmission.

5 Implementation

The file transfer system is implemented using Python's `mpi4py` library. Below is the code for both the server and client.

5.1 Server Code

Listing 1: Server Code using MPI

```
from mpi4py import MPI
import os

def server_program():
    # Initialize MPI
    comm = MPI.COMM_WORLD
    rank = comm.Get_rank()
    size = comm.Get_size()

    # process 0 act as the server
    if rank == 0:
        host = '172.19.219.4'  # Server IP (same as before)
        port = 12344             # Port to listen on

        print(f"Server listening on {host}:{port}...")

        # Receive file from client
        file_name = comm.recv(source=1, tag=1)  # Client (rank 1) sends fil
        print(f"Server is receiving the file: {file_name}")
```

```
        # Save data
        with open(file_name, 'wb') as f:
            data = comm.recv(source=1, tag=2)  # Receive file content from
            while data:
                f.write(data)
                data = comm.recv(source=1, tag=2)

        print(f"File {file_name} received successfully.")
    else:
        print(f"Process {rank} is idle...")

if __name__ == "__main__":
    server_program()
```

5.2 Client Code

Listing 2: Client Code using MPI

```
from mpi4py import MPI
import os

def client_program():
    # Init mpi
    comm = MPI.COMM_WORLD
    rank = comm.Get_rank()
    size = comm.Get_size()

    if rank == 1:  # Client process with rank 1
        host = '172.19.219.4'  # Server IP (same as before)
        port = 12344             # Port

        # File sent to server
        filename = 'send_file.txt'

        # Send file to server
        comm.send(filename, dest=0, tag=1)
        print(f"Sending file: {filename}")

        # Open the file to send
        with open(filename, 'rb') as f:
            data = f.read(1024)  # Read file in chunks
            while data:
                comm.send(data, dest=0, tag=2)  # Send data chunks to serve
                data = f.read(1024)

        # Show server file transfer
        comm.send(None, dest=0, tag=2)
```

```
            print(f"File {filename} sent successfully.")
        else:
            print(f"Process {rank} is idle...")

if __name__ == "__main__":
    client_program()
```

5.3 Sample Output from the Server

```
Server listening on 172.19.219.4:12344...
Server is receiving the file: send_file.txt
File send_file.txt received successfully.
```

The server waits for the file transfer to complete. Upon receiving the file name and data, it writes the data to a file on disk.

5.4 Sample Output from the Client

```
Sending file: send_file.txt
File send_file.txt sent successfully.
```

The client sends the file in chunks and informs the server once the transfer is complete. If the transfer is successful, the client displays a success message.

6 Discussion

This lab demonstrated file transfer using MPI in a parallel and distributed manner. By using multiple processes, we were able to transfer a file in chunks. This parallel communication model enhances scalability in distributed systems.

Some areas for improvement include:

- Handling large files more efficiently by implementing more advanced chunking and parallel file sending.
- Enhancing error handling, especially for network issues or file system errors.
- Scaling the system to multiple clients and servers using MPI's collective operations.

7 Conclusion

In this lab, we successfully implemented a file transfer system using MPI. Python's `mpi4py` library enabled efficient communication between client and server processes. The system allowed the client to send files to the server and handle file transfers in parallel.

Future improvements could include concurrent client handling, better error handling, and more efficient data chunking to support larger files.