Practical Work 3: MPI File Transfer

Dang Trung Nguyen – 22BA13239

December 6, 2025

1 Introduction

In this practical, we extend our previous TCP-based file transfer system by upgrading it to an MPI-based solution. Instead of using sockets and a client/server model, the system is now implemented using two MPI ranks that communicate through message passing. The goal is to demonstrate how distributed communication can be implemented using MPI primitives.

This report explains the MPI implementation choice, the design of the MPI service, the organization of the system, and the code used to implement file transfer.

2 MPI Implementation Choice

For this practical, the following stack is used:

- **MPICH / OpenMPI** – underlying MPI framework
- **mpi4py** – Python bindings for MPI

Reasons for choosing mpi4py:

- Simple integration with Python, matching our TCP implementation
- High performance, using real MPI communication under the hood
- Clear, readable code with familiar Send/Recv functions
- Easy to use under Linux/Kali and compatible with MPICH or OpenMPI

3 MPI Service Design

The system consists of only two processes:

- **Rank 0: Sender**
- **Rank 1: Receiver**

Rank 0 reads the local file, sends metadata first (filename and size), and then streams the file in multiple chunks. Rank 1 receives these chunks and reconstructs the file.

Design Diagram (Bidirectional)

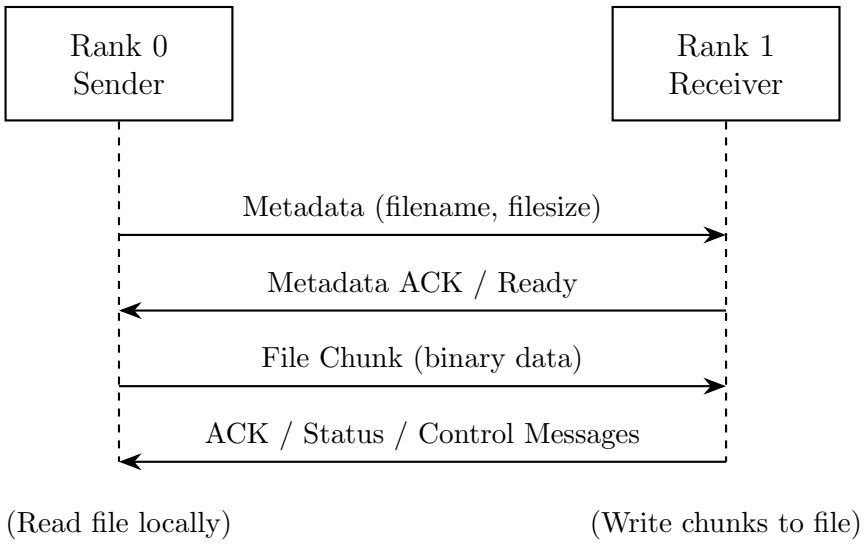

Figure 1: Design Diagram

4 System Organization

The system uses three main components:

- Local file system of Rank 0
- MPI communication layer
- Output file system of Rank 1

Overall Architecture Diagram

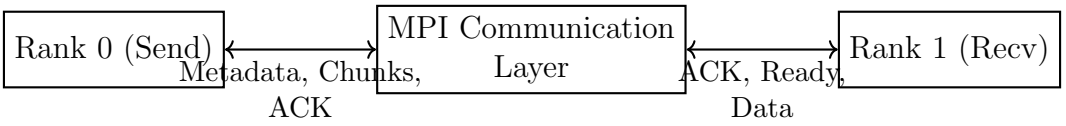

Figure 2: Overall System Architecture (External Diagram)

5 Implementation

Below are the core parts of the MPI-based file transfer system.

Sender (Rank 0)

```
filename = os.path.basename(file_path)
filesize = os.path.getsize(file_path)

comm.send(filename, dest=1, tag=0)
comm.send(filesize, dest=1, tag=1)

with open(file_path, "rb") as f:
    while True:
        chunk = f.read(CHUNK_SIZE)
        if not chunk:
            break
        comm.send(chunk, dest=1, tag=2)
```

Receiver (Rank 1)

```
filename = comm.recv(source=0, tag=0)
filesize = comm.recv(source=0, tag=1)

with open("received_" + filename, "wb") as f:
    received = 0
    while received < filesize:
        chunk = comm.recv(source=0, tag=2)
        f.write(chunk)
        received += len(chunk)
```

6 Execution

The program is executed using exactly two MPI ranks:

```
mpiexec -n 2 python3 mpi_file_transfer.py <file_path>
```

Rank 0 sends the file and Rank 1 writes the output.

7 Conclusion

The MPI version successfully replaces the TCP approach and demonstrates how distributed systems can exchange data efficiently using message passing. mpi4py provides a clean and effective abstraction for building parallel and distributed applications.