MPI File Transfer – Practical Work 3

Nguyen Hoang Long – Distributed Systems

December 18, 2025

1 Introduction

This practical work extends the TCP file transfer system from Practical Work 1 to a solution based on MPI (Message Passing Interface). Instead of using a client–server model over TCP sockets, we use an MPI communicator in which different ranks cooperate to transfer a file. In our design, rank 0 acts as the sender and rank 1 as the receiver.

2 Choice of MPI Implementation

We chose the `mpi4py` library on top of an existing MPI distribution (such as Open MPI or MPICH) for the following reasons:

- It allows us to keep using Python, reusing code and structure from the TCP and RPC practical works.
- `mpi4py` provides a high-level interface to standard MPI primitives (`send`, `recv`, collective operations), while still mapping directly to the underlying C MPI calls.
- It is portable: the same code can be executed on laptops, clusters or teaching labs as long as an MPI implementation and Python are available.

3 MPI Service Design

3.1 Communication Model

We use a simple point-to-point communication pattern between two ranks:

- Rank 0: reads a local file, then sends the file name, file size and binary data to rank 1 via MPI.
- Rank 1: receives the metadata and data from rank 0 and writes the file to disk.

The data is sent in three messages so that the receiver can allocate and handle the file in a controlled way.

3.2 MPI Call Flow

Figure 1 shows the sequence of MPI calls used to transfer one file.

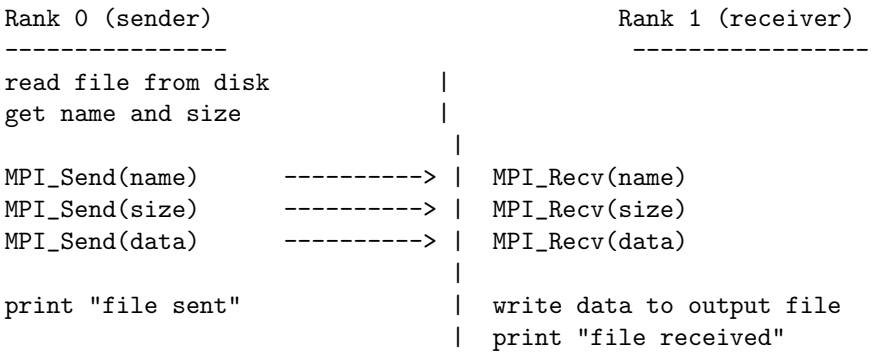

Figure 1: MPI request–response flow for file transfer between rank 0 and rank 1.

4 System Organization

The system is organized as a single MPI program (SPMD model) launched with two processes. Both execute the same script but follow different code paths depending on their rank. Communication happens in the default communicator `MPI.COMM_WORLD`.
Figure 2 shows the high-level architecture.

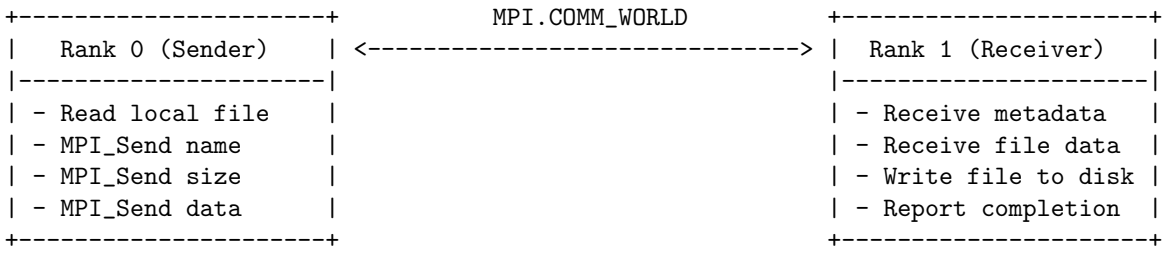

Figure 2: System organization of the MPI file transfer system.

5 Implementation

5.1 MPI File Transfer Code Snippet

The following Python code (using `mpi4py`) shows the core logic of our MPI file transfer. The script is executed with two processes, for example:

```
mpirun -np 2 python mpi_file_transfer.py
```

```
from mpi4py import MPI
import os

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

FILENAME = "test.txt"

if size < 2:
    if rank == 0:
        print("Please run with at least 2 MPI processes.")
    raise SystemExit

if rank == 0:
    # Sender process
    filesize = os.path.getsize(FILENAME)
    with open(FILENAME, "rb") as f:
        data = f.read()

    # Send metadata and data to rank 1
    comm.send(FILENAME, dest=1, tag=0)
    comm.send(filesize,  dest=1, tag=1)
    comm.send(data,      dest=1, tag=2)

    print(f"Rank 0: sent {FILENAME} ({filesize} bytes) to rank 1.")

elif rank == 1:
    # Receiver process
    name = comm.recv(source=0, tag=0)
    filesize = comm.recv(source=0, tag=1)
    data = comm.recv(source=0, tag=2)

    out_name = "received_" + name
    with open(out_name, "wb") as f:
        f.write(data)

    print(f"Rank 1: received {out_name} "
          f"({len(data)} bytes, expected {filesize}).")
```

This implementation illustrates how MPI can be used to build a simple file transfer mechanism without explicit sockets. The MPI runtime takes care of process creation, message routing and low-level communication.

6 Conclusion

We implemented a basic MPI-based file transfer system by extending the previous TCP design to use `mpi4py`. By relying on MPI primitives (`send`/`recv`) and ranks, we can easily run the same code on different machines or cores. The practical demonstrates how MPI can be used for simple point-to-point services such as file transfer and provides a basis for more advanced distributed applications.