Practical Work 3 - MPI File Transfer

Tran Hai Nam

December 2025

1 Introduction

This practical work implements a 1-to-1 file transfer using MPI (message passing) instead of sockets or RPC. The goal is to exercise explicit point-to-point communication (`MPI_Send`/`MPI_Recv`) by sending file metadata and payload between two MPI processes in a clear, minimal design.

2 Objectives

- Build a 1-to-1 file transfer with MPI, using rank 0 as sender and rank 1 as receiver.
- Keep the application-level protocol simple: send metadata (filename, size) then stream the file in fixed-size chunks.
- Provide a CLI to specify input file (sender) and optional output path (receiver).

3 System Design

3.1 Architecture Overview

Two MPI ranks participate:

- **Rank 0 (Sender)**: reads a local file, sends metadata and file content.
- **Rank 1 (Receiver)**: receives metadata and content, writes to disk.

Ranks greater than one exit immediately.

3.2 Protocol and Data Flow

Application-level fields:

- `name_len` (INT): length of filename in bytes (UTF-8).
- `file_size` (LONG): size of the file in bytes.
- `filename` (BYTE array): UTF-8 encoded name.
- Payload: file content split into 64 KB chunks (BYTE).

Figure 1 illustrates the message flow.

3.3 Error Handling

If fewer than two ranks: rank 0 logs and exits. If input file is missing: sender aborts MPI. If output path cannot be created: receiver aborts. Any rank >1 exits immediately to keep the lab 2-process only.

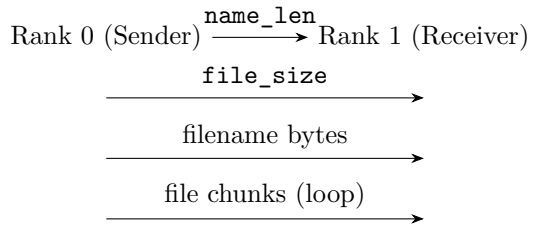

Figure 1: MPI point-to-point flow: metadata followed by file chunks.

4 Implementation

4.1 Language, Libraries, Environment

Java with MPI-for-Java (MPJ-style API). Requires `mpj.jar` on the classpath; run with `mpjrun`. Chunk size is fixed at 64 KB for streaming.

4.2 Code Structure

- `mpi/FileTransferMPI.java`:
  - `main`: initialize MPI, branch by rank, finalize.
  - `runSender`: validate input, derive remote name, send metadata then chunks.
  - `runReceiver`: receive metadata, choose output path, ensure parent directory exists, receive chunks until complete.

4.3 Algorithms

Sender

1. Read `input_file`, compute `name_len`, `file_size`, encode filename (UTF-8).
2. Send `name_len` (INT), `file_size` (LONG), filename bytes (BYTE).
3. Loop: read up to 64 KB from file, `MPI_Send` that chunk (BYTE) to rank 1 until all bytes sent.

Receiver

1. Receive `name_len`, `file_size`, filename bytes; reconstruct filename.
2. Decide `output_path` (argument or filename), create parent directory if needed.
3. Loop: `toRecv = min(64 KB, remaining)`; `MPI_Recv` chunk, write to file, update remaining until done.

5 Build and Run

5.1 Compilation

```
javac -cp ".;path\to\mpj.jar" mpi/FileTransferMPI.java
```

5.2 Execution

```
mpjrun.bat -np 2 -cp ".;path\to\mpj.jar" mpi.FileTransferMPI ^
  "D:\Repo\He_phan_tan\mpi\input_mpi.bin" "D:\Repo\He_phan_tan\mpi\
     output_mpi.bin"
```

If the output path is omitted, the receiver uses the same filename in the current directory. Ranks greater than one exit immediately.

6 Evaluation

6.1 Strengths

- Minimal MPI example focusing on `Send`/`Recv` with clear metadata/payload separation.
- Streaming avoids loading the entire file into memory.
- Small, dependency-light code (only MPI runtime).

6.2 Limitations

- Exactly two ranks; no broadcast or multi-receiver support.
- No checksum, authentication, encryption, or retry/resume.
- Depends on MPI runtime and correct classpath for `mpj.jar`.

6.3 Comparison

- Lab 1 (TCP): manual socket/header but flexible streaming; no MPI runtime needed.
- Lab 2 (RPC): higher abstraction, less wire control; current implementation buffered whole file.
- Lab 3 (MPI): explicit message control, but requires MPI setup; point-to-point only.

6.4 Test Result

Test run (multicore, MPJ Express 0.44) with input `input_mpi.bin` (183 bytes) and output `output_mpi.bin`:

- Sender log: “Sending ’...\input_mpi.bin’ (183 bytes) to rank 1”; “Done. Total sent: 183 bytes”.
- Receiver log: “Receiving ’...\input_mpi.bin’ (183 bytes) -> ...\output_mpi.bin”; “Done. Stored ...\output_mpi.bin (183 bytes)”.

No errors reported.

7 Possible Extensions

- Integrity/security: add checksum (e.g., MD5/SHA-256) and secure channels if needed.
- Scalability: support broadcast or scatter/gather for multiple receivers.
- Performance: tune chunk size; explore non-blocking `Isend`/`Irecv` and pipelining.

8 Conclusion

This practical demonstrates a straightforward MPI-based file transfer with two ranks, reinforcing message-passing fundamentals and contrasting with socket- and RPC-based designs from earlier labs.