MPI File Transfer System

Nguyen Viet Khoa

December 5, 2025

1 Why the Chosen MPI Implementation

I chose mpi4py, the Python bindings for MPI, because it integrates seamlessly with existing Python code from the original TCP system. It is easy to install and use, supports high-level operations such as send/recv, and works with standard MPI backends, including OpenMPI. This avoids rewriting in lower-level languages like C++, making the upgrade straightforward while maintaining portability across clusters.

2 Design of the MPI Service

The MPI service uses a single-program multiple-data (SPMD) model where all processes run the same script. Rank 0 acts as the server, reading the file and sending it in chunks to Rank 1 (the client), which receives and saves the data. Communication uses MPI send/recv for simplicity. File size is sent first to handle reception, and chunks prevent message overflow for large files.

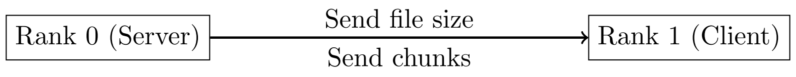

Figure 1: MPI Service Design

3 Organization of the System

The system is organized in a new directory called `MPI`, copied from the original TCP implementation. It contains:

- `mpi_file_transfer.py`: The main MPI script handling both server and client logic based on rank.
- Files to transfer (e.g., `example.txt`) placed in the directory accessible by rank 0.

No separate server/client files are needed, as MPI handles process differentiation. Run with `mpirun -np 2 python mpi_file_transfer.py <filename>`.

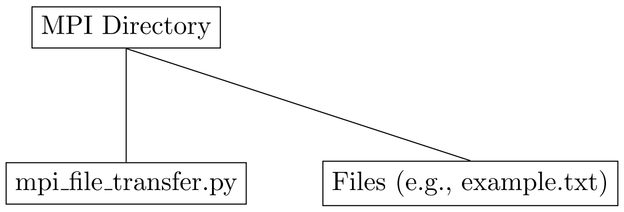

Figure 2: System Organization

4 Implementation of the File Transfer

The file transfer uses MPI's `send` and `recv` for data exchange. The server (rank 0) reads the file, sends its size, then chunks the data. The client (rank 1) receives the size, then accumulates chunks and writes to a new file. Here's a code snippet:

Listing 1: Core Implementation Snippet

```
if rank == 0:  # Server
    with open(filename, 'rb') as f:
        file_data = f.read()
    file_size = len(file_data)
    comm.send(file_size, dest=1)
    for i in range(0, file_size, CHUNK_SIZE):
        chunk = file_data[i:i + CHUNK_SIZE]
        comm.send(chunk, dest=1)

elif rank == 1:  # Client
    file_size = comm.recv(source=0)
    received_data = b''
    remaining = file_size
    while remaining > 0:
        chunk = comm.recv(source=0)
        received_data += chunk
        remaining -= len(chunk)
    with open(received_filename, 'wb') as f:
        f.write(received_data)
```

5 Who Does What

- **Rank 0 (Server)**: Checks arguments, reads the local file, sends the file size, chunks, and sends the data via MPI. Handles errors like file not found by sending a signal (-1).
- **Rank 1 (Client)**: Receives the file size, accumulates chunks until complete, and writes the data to a new local file. Exits on error signals.