MPI-Based Parallel File Transfer System

Bui Truong An - 22BA13001

1 Choice of MPI Implementation

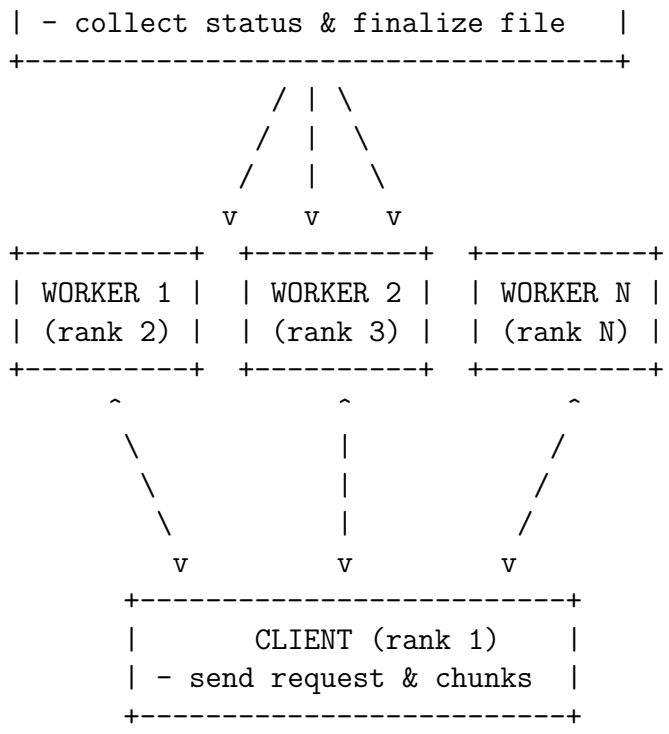
For this project, I chose **mpi4py** as the MPI implementation. The reasons include:

- **High-level Python interface**: mpi4py wraps the standard MPI API while allowing fast development in Python without sacrificing performance.
- **Full MPI-1 and MPI-2 support**: Features such as point–to–point messaging, communicators, non-blocking operations, and collective communication are supported.
- **Easy integration with file I/O**: Since the project requires file splitting, reading binary chunks, and writing temporary files, Python simplifies implementation.
- **Portability**: Works on OpenMPI, MPICH, and most HPC systems.

2 MPI Service Design

The system operates using a **master–worker architecture**. Rank 0 acts as the master server, while ranks 1..N are workers that receive assigned file chunks.

```
+-----------------------------------+
|            MASTER (rank 0)        |
| - receive transfer request        |
| - assign chunk ranges to workers  |
```

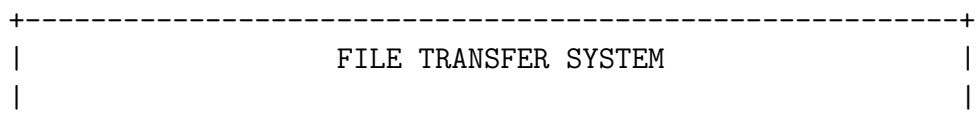


Workflow summary:

1. Client (rank 1) sends metadata to Master (rank 0).
2. Master computes chunk ranges and sends assignments to Workers.
3. Client streams chunked messages (tagged) directly to Workers.
4. Workers write partial files and notify Master when done.
5. Master acknowledges completion to Client and records saved file.

3 System Organization

The implementation separates concerns into three logical components.



```
|  [Client]   <-->   [Master / Coordinator]   <-->  [Workers]  |
|  (chunking,        (assign ranges,               (receive,
|   stream)           metadata, finalize)           write .part)|
+-----------------------------------------------------------+

Temp files: uploads/received_<timestamp>_filename.part<rank>
```

4 File Transfer Implementation

Below is a representative code snippet from the client and server showing how MPI-based file transfer is implemented.

4.1 Client: Sending Chunks

Listing 1: Client sending file chunks

```
for chunk_num, chunk_data in enumerate(chunks):
    self.comm.send({'data': chunk_data},
                   dest=MPI.ANY_SOURCE,
                   tag=100 + chunk_num)
```

4.2 Server Worker: Receiving Chunks

Listing 2: Worker receiving assigned chunks

```
with open(temp_file, 'wb') as f:
    for chunk_num in range(start_chunk, end_chunk):
        chunk_data = self.comm.recv(
            source=source_rank,
            tag=100 + chunk_num
        )
        f.write(chunk_data['data'])
```

4.3 Master: Distributing Work

Listing 3: Master assigning chunk ranges

```
work_data = {
    'source_rank': source_rank,
    'start_chunk': start_chunk,
    'end_chunk': end_chunk,
    'filepath': filepath
}
self.comm.send(work_data, dest=worker, tag=10)
```

5 Conclusion

The ascii-figure based report keeps diagrams simple, portable, and easy to read while preserving the full description of the master–worker MPI file-transfer design implemented with `mpi4py`.