MPI File Transfer System Design and Implementation

Vu Hai Thien Long (22BI13268)
Luu Linh Ly (22BI13269)
Nguyen Duc Duy (22BI13120)
Le Viet Hoang Lam (22BI13235)
Nguyen Ngoc Nhi (22BI13351)

11 December 2024

1 Introduction

This report details the design and implementation of an MPI-based file transfer system. Building upon a TCP-based system, the MPI version leverages parallel processing to enhance performance and scalability.

2 Choice of MPI Implementation

We chose **mpi4py** as our MPI implementation for the following reasons:

- **Seamless Integration with Python**: Allows us to utilize existing Python code with minimal modifications.
- **Comprehensive Documentation**: Facilitates easier development and troubleshooting.
- **Active Community Support**: Ensure access to a wealth of resources and assistance.
- **Performance Efficiency**: Provides robust performance suitable for high-throughput file transfers.

3 Design of the MPI Service

3.1 System Architecture

The system employs a Master-Worker architecture where the master process coordinates file transfer tasks, and worker processes handle the actual file transmission.

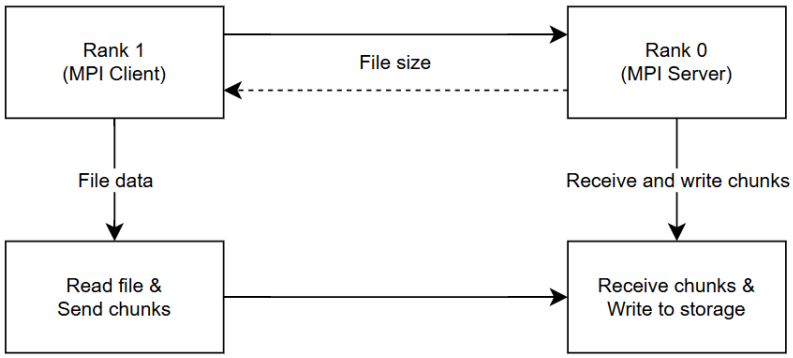

Figure 1: MPI Service Design

3.2 Components and Workflow

1. **Client Process**:
   - Initiates a connection to the server.
   - Sends the file in chunks using MPI communication.
2. **Server Process**:
   - Listens for incoming file transfer requests.
   - Receives and assembles the file from chunks.

4 System Organization

The system is organized into distinct modules:

- **Client Module**: Handles file reading and sending operations.
- **Server Module**: Manages file reception and storage.
- **Communication Module**: Facilitates MPI-based data transfer between the client and the server.

5 Implementation of File Transfer

5.1 Client Code Snippet

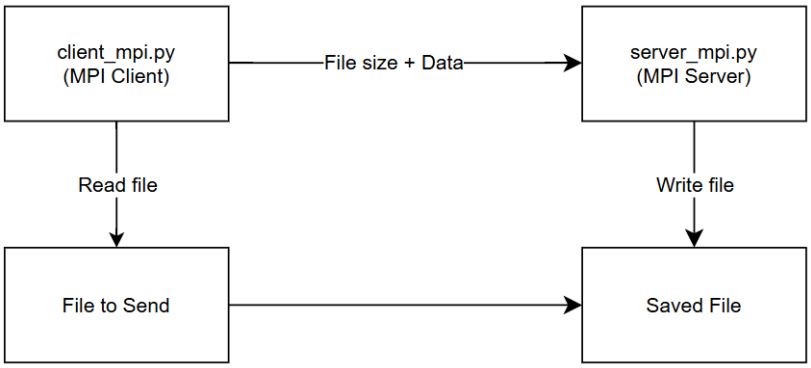

Figure 2: System Organization Diagram

```python
from mpi4py import MPI
import sys

def send_file(file_path, server_rank=0):
    comm = MPI.COMM_WORLD
    rank = comm.Get_rank()

    if rank == server_rank:
        print("Master process does not send files.")
        return

    try:
        with open(file_path, "rb") as f:
            data = f.read()
    except FileNotFoundError:
        print(f"File {file_path} not found.")
        sys.exit(1)

    file_size = len(data)
    comm.send(file_size, dest=server_rank, tag=11)
    comm.Send([bytearray(data), MPI.BYTE], dest=server_rank,
        tag=22)
    print(f"File '{file_path}' sent to server.")

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: mpirun -n <number_of_processes> python
            client_mpi.py <file_path>")
        sys.exit(1)

    file_path = sys.argv[1]
    send_file(file_path)
```

Listing 1: MPI Client Implementation

5.2 Server Code Snippet

```python
from mpi4py import MPI
import sys

def receive_file(output_path="received_file_mpi.txt",
    client_rank=1):
    comm = MPI.COMM_WORLD
    rank = comm.Get_rank()

    if rank != client_rank:
        print("Only the designated server process receives files.")
        return

    file_size = comm.recv(source=MPI.ANY_SOURCE, tag=11)
    buffer = bytearray(file_size)
    comm.Recv([buffer, MPI.BYTE], source=MPI.ANY_SOURCE, tag=22)

    with open(output_path, "wb") as f:
        f.write(buffer)
    print(f"File received and saved as '{output_path}'.")

if __name__ == "__main__":
    receive_file()
```

Listing 2: MPI Server Implementation

6 Roles and Responsibilities

- **Vu Hai Thien Long**: Implemented the MPI client and handled file reading operations.
- **Luu Linh Ly**: Developed the MPI server and managed file reception.
- **Nguyen Duc Duy**: Created system diagrams and contributed to the LaTeX report.
- **Le Viet Hoang Lam**: Integrated client and server modules and ensured seamless communication.
- **Nguyen Ngoc Nhi**: Coordinated group activities and managed version control.

7 Conclusion

The MPI-based file transfer system successfully enhances the original TCP-based system by leveraging parallel processing capabilities. The use of ‘mpi4py‘ facilitated efficient implementation and scalability, making the system robust for handling multiple file transfers concurrently.