University Of Science and Technology of Hanoi
ICT Department

Distributed System Report

Lecturer: Le Nhu Chu Hiep

Practical Work 3: MPI File transfer

Student: Nguyễn Thị Vàng Anh - 23BI14032

1. Choice of MPI Implementation

Description: Clients submit jobs to the MPI Service (master, Rank 0). The master dispatches tasks to worker ranks, orchestrates file transfers (direct MPI_Send/ Recv or by shared storage), and verifies checksums. Shared storage (NFS/ object store) is used where appropriate for large files or persistence.

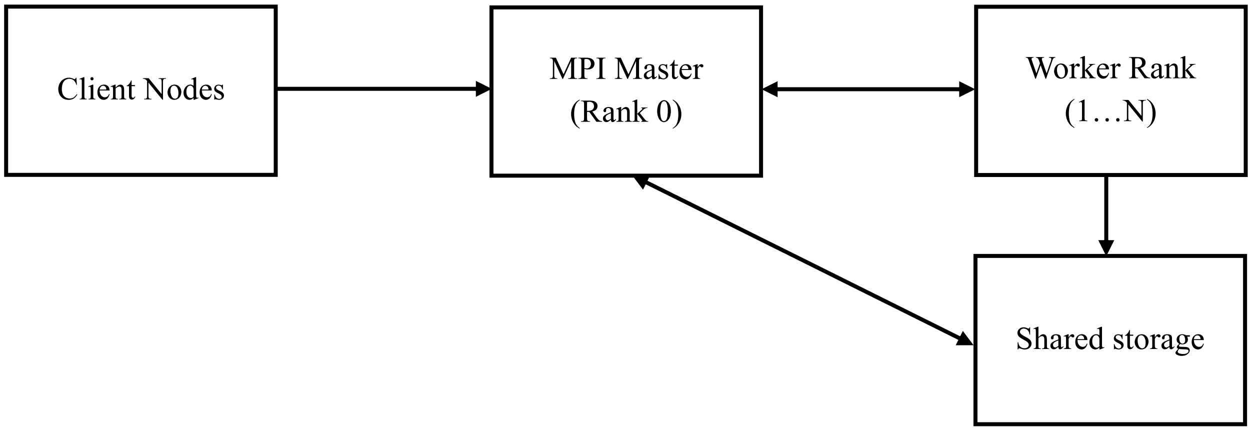
For this project, I choose MPI using mpi4py (Message Passing Interface for Python). The reasons are:

- Portability: MPI is available on almost all HPC clusters and works consistently across platforms.
- Parallel communication model: MPI provides efficient point-to-point and collective communication, ideal for distributed file transfer.
- Easy to use with Python: mpi4py allows full access to all MPI communication functions without requiring C/C++ code.
- Overall, mpi4py provides the best balance between performance, simplicity, and platform compatibility.

2. Design of the MPI Service

The system uses a two-rank architecture:

- Rank 0: Client
    - o Handles user input and sends requests (UPLOAD, DOWNLOAD, EXIT).
- Rank 1: Server
    - o Receives commands, stores uploaded files, and returns requested files



3. System Organization

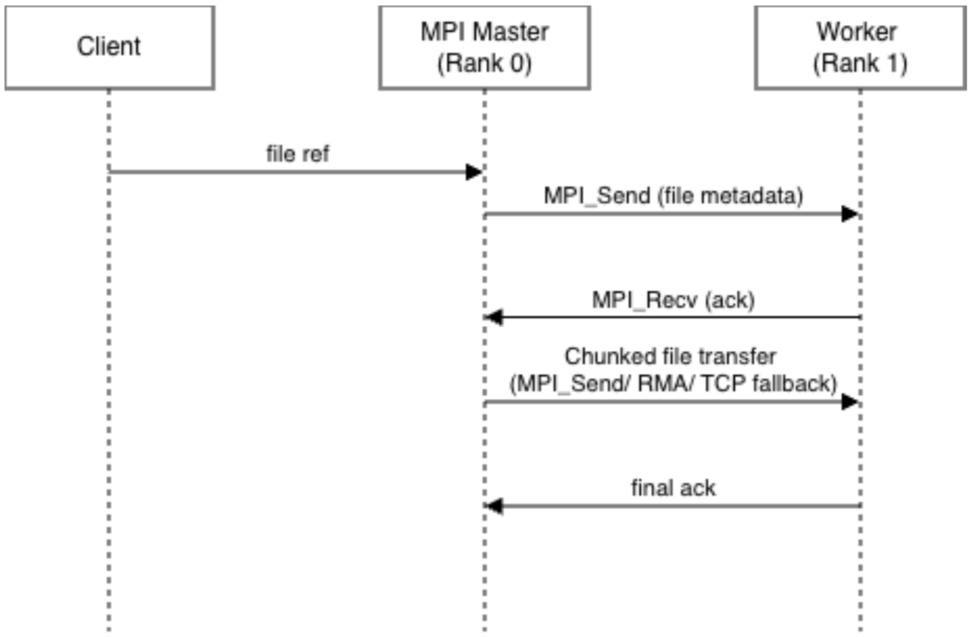
Client layer (CLI/ UI) that packages job metadata and references to files. Uses the MPI client to send requests to Rank 0.

MPI Service (Master/ Rank 0): job coordinator. Responsibilities:

- Authenticate/ authorize client
- Accept file uploads or references

- Decide transfer method (direct MPI vs shared storage)
- Dispatch tasks and track statuses
- Aggregate results and provide final output to client

Worker nodes (Ranks 1..N): execute work, request required files from master or read from shared storage, send back results.
Shared storage: ephemeral staging area for chunked transfer or persistent object store for long-term data.



4. File Transfer Implementation

Server:

```
def server():
    os.makedirs(SERVER_DIR, exist_ok=True)
    print(f"[SERVER {RANK}] Ready, storing files in '{SERVER_DIR}'")

    while True:
        cmd = COMM.recv(source=CLIENT_RANK, tag=0)
        if cmd == "EXIT":
            print("[SERVER] EXIT received, shutting down.")
            break
        elif cmd == "UPLOAD":
            handle_upload()

        elif cmd == "DOWNLOAD":
            handle_download()
        else:
            print(f"[SERVER] Unknown command: {cmd!r}")
```

```python
def handle_upload():
    filename = COMM.recv(source=CLIENT_RANK, tag=1)
    filedata = COMM.recv(source=CLIENT_RANK, tag=2)

    path = os.path.join(SERVER_DIR, filename)
    with open(path, "wb") as f:
        f.write(filedata)

    print(f"[SERVER] Stored file '{filename}' ({len(filedata)} bytes)")
    COMM.send("OK", dest=CLIENT_RANK, tag=3)

def handle_download():
    filename = COMM.recv(source=CLIENT_RANK, tag=1)
    path = os.path.join(SERVER_DIR, filename)

    if os.path.exists(path):
        with open(path, "rb") as f:
            data = f.read()
        COMM.send(True, dest=CLIENT_RANK, tag=3)
        COMM.send(data, dest=CLIENT_RANK, tag=2)
        print(f"[SERVER] Sent file '{filename}' ({len(data)} bytes)")
    else:
        COMM.send(False, dest=CLIENT_RANK, tag=3)
        print(f"[SERVER] File not found: '{filename}'")
```

Client:

```python
def client_menu():
    os.makedirs(CLIENT_DIR, exist_ok=True)
    print(f"[CLIENT {RANK}] Ready, local folder '{CLIENT_DIR}'")

    while True:
        print("\n=== MPI FILE TRANSFER MENU ===")
        print("1. Upload file to server")
        print("2. Download file from server")
        print("3. Exit")
        choice = input("Choose: ").strip()

        if choice == "1":
            upload_file()
        elif choice == "2":
            download_file()
        elif choice == "3":
            COMM.send("EXIT", dest=SERVER_RANK, tag=0)
            print("[CLIENT] Exit sent. Bye!")
            break
        else:
            print("Invalid choice, try again.")

def upload_file():
    path = input("Path to local file: ").strip()
    if not os.path.exists(path):
        print("File does not exist.")
        return

    filename = os.path.basename(path)
    with open(path, "rb") as f:
        data = f.read()

    COMM.send("UPLOAD", dest=SERVER_RANK, tag=0)
    COMM.send(filename, dest=SERVER_RANK, tag=1)
    COMM.send(data, dest=SERVER_RANK, tag=2)

    status = COMM.recv(source=SERVER_RANK, tag=3)
    if status == "OK":
        print(f"[CLIENT] Uploaded '{filename}' ({len(data)} bytes)")
    else:
        print("[CLIENT] Upload failed.")

def download_file():
    filename = input("Filename to download from server: ").strip()

    COMM.send("DOWNLOAD", dest=SERVER_RANK, tag=0)
    COMM.send(filename, dest=SERVER_RANK, tag=1)

    exists = COMM.recv(source=SERVER_RANK, tag=3)
    if not exists:
        print("[CLIENT] File does not exist on server.")
        return

    data = COMM.recv(source=SERVER_RANK, tag=2)
    path = os.path.join(CLIENT_DIR, filename)
    with open(path, "wb") as f:
        f.write(data)

    print(f"[CLIENT] Saved file to '{path}' ({len(data)} bytes)")
```

5. Conclusion

The MPI file transfer system provides a simple but robust way to demonstrate distributed communication. Using mpi4py makes it easy to structure the service with a clear server-client model, define deterministic communication flows, and efficiently transfer files using MPI messages.