Distributed File Transfer

Different ways of file Transferring

- **Single-host communication**
  - **Socket Communication :-** The code establishes communication between a client and a server using sockets. The client and server communicate over a common network, i.e, IITD network. The client connects to the remote server specified by the server IP and server port. The connection was established to be a TCP connection in Python.
  - **Data Reception :-** The main thread of the client sends "SENDLINE" messages to the server and expects lines from it in response. It continuously receives and processes lines from the server. It reconstructs complete messages from the received lines and maintains a count of received lines. The client assembles complete messages by sequentially appending received lines. It tracks the order of lines using a numbering system, ensuring that lines are correctly ordered before submitting them to the server.
  - **Data Handling :-** The lines received from the server may get fragmented and are hence properly parsed by maintaining a variable to track when the next non -1 digit is received. The lines received are in order due to the connection being over TCP and hence are able to be properly parsed with this logic.
  - **Error Handling :-** We have put an exception code whenever we are connecting or receiving lines from the server as it might be possible that the Server-Client connection breaks off suddenly so that we can reconnect to the server again. We close the previous socket and start a fresh connection by redefining and reconnecting to the server over the same socket.
- **Distributed Algorithm for Peer to Peer exchange**:-

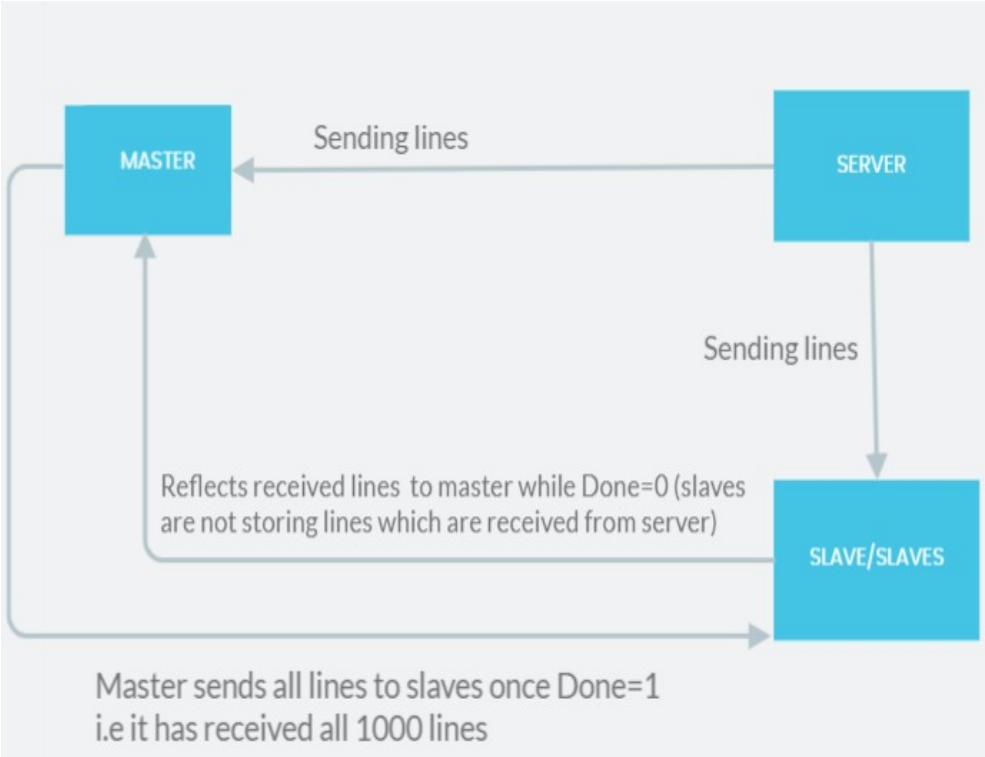

Figure 1: **Pictorial Representation of Algorithm**

In this assignment, we worked with multiple clients (1 to 4) to download a file from a server. One of these clients was in charge (we called it the "master"), while the others were helpers (we called them "slaves"). The server sent lines of text representing the data we have to assemble along with line numbers to specify the data ordering.

Steps followed During the implementation:-
1 . **Getting Started :** We established a TCP connection from each helper (slave) client to the master client and connected to the server from all clients.
2 . **Sharing with the Master:** When the server sends lines, the helper(slave) clients shared/reflect them with the master without storing them.
3 . **Master Keeping Track:** The master received n(number of clients) streams of lines, from each helper client and one through its own server connection. The n streams were processed in n parallel threads and kept count of using global variables to store the line count and a list to store whether a line with some line number has been received or not. Another list stores the line with the line number.
4. **File Completion Check:** Once the master had received 1000 lines, it sent an acknowledgment message to all the helper clients. All the helpers have a thread running in parallel which checks if it has received the acknowledgement message. After receiving acknowledgment, the helpers start receiving lines from the master client.
5. **Sharing the Full File:** The master then sends 1000 lines to each helper client which takes less than 1 second because each line only needs to be sent once. The parsing of fragmented data from the TCP connection is handled using the same logic of data handling as written above.

Methodology of Helper(slave) clients:- Helpers received lines from the server and sent them to the master. They handled line reception in the main thread and simultaneously sent lines to the master client in the main thread. A separate thread was used to listen to the master for an acknowledgment message, signaling that the master had received all lines and that it was ready to share.
For Master:- The master received lines from the server in the main thread. Lines from different slaves were received in separate threads, allowing for efficient handling. It sent lines to different slaves in the main thread, once it received all 1000 lines.

- **Question:-** Does the download time reduce linearly as you add more and more clients?
  Answer:-
  - **Observation** While it's evident that adding more clients reduces the time taken, the relationship doesn't seem strictly linear based on these data points. A linear relationship would mean that adding each client reduces the time by a constant amount. Here, the reduction appears to be more significant when going from 1 to 2 clients compared to going from 3 to 4 clients.
  - **Reason** No, adding more Clients to the time doesn't decrease linearly because increasing the number of clients causes the number of lines processed by the master to increase linearly at the same time. The time should decrease inversely with the number of clients, but inconsistencies are observed due to variations in the number of lines required to achieve the same count of unique lines.

Number of Clients	Total Time Taken(ms)
1	78.15919
2	44.88218
3	28.47864
4	18.87127

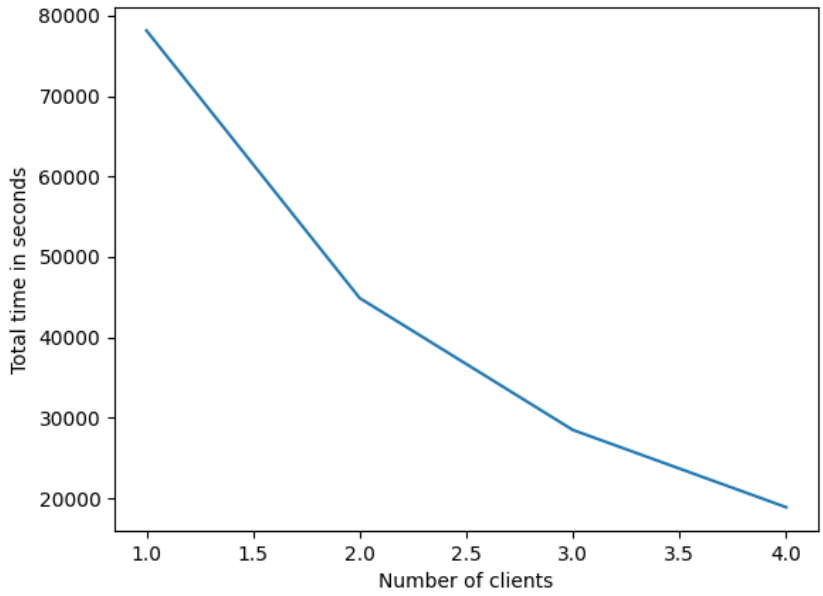

Figure 2: **Total Time Taken vs Number of Clients**

- **Exception Scenarios**

  1. **Client - Client Disconnect:-** A client node suddenly disconnects maybe due to network issues.
  Handling:- The global variable connect is made in each helper client to keep track if the connection with the master breaks off. The function connection is run again to establish the connection with the master. While the Master maintains a list of such connect variables for each helper client and starts accepting connections if the connection is lost with any helper client.

  2. **Client-Server Connection Break :-** The connection between a client and server unexpectedly breaks during the data transfer.
  Handling:- The exception is handled by closing the previous connection and starting a fresh connection by redefining the same socket and reconnecting to the server over it. The exception is tracked by just looking for an exception other than socket. timeout while interacting with the server.

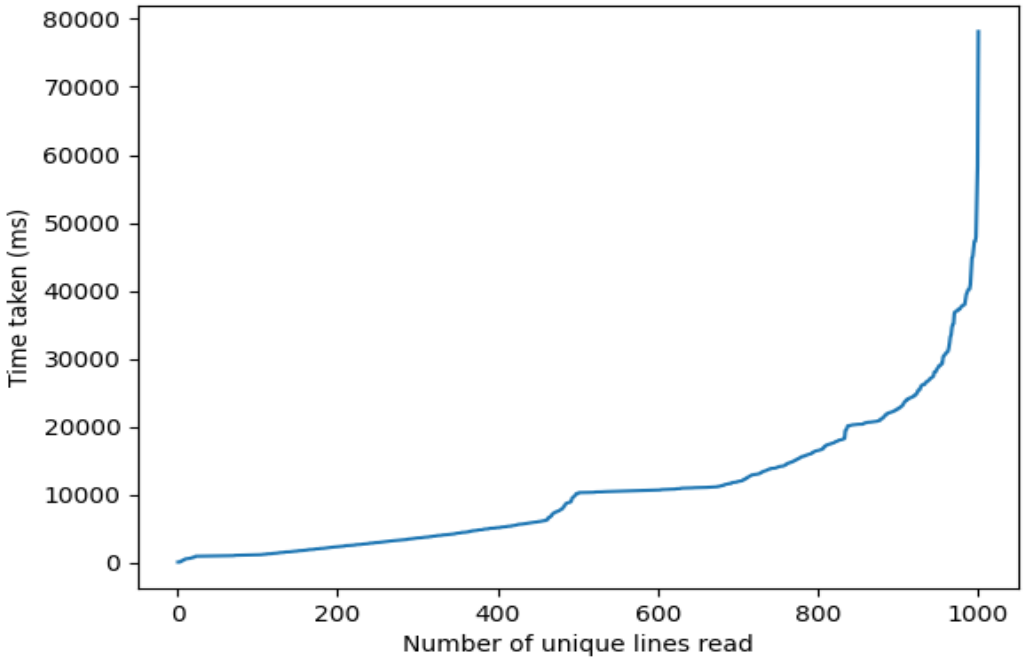

Figure 3: **One Client**

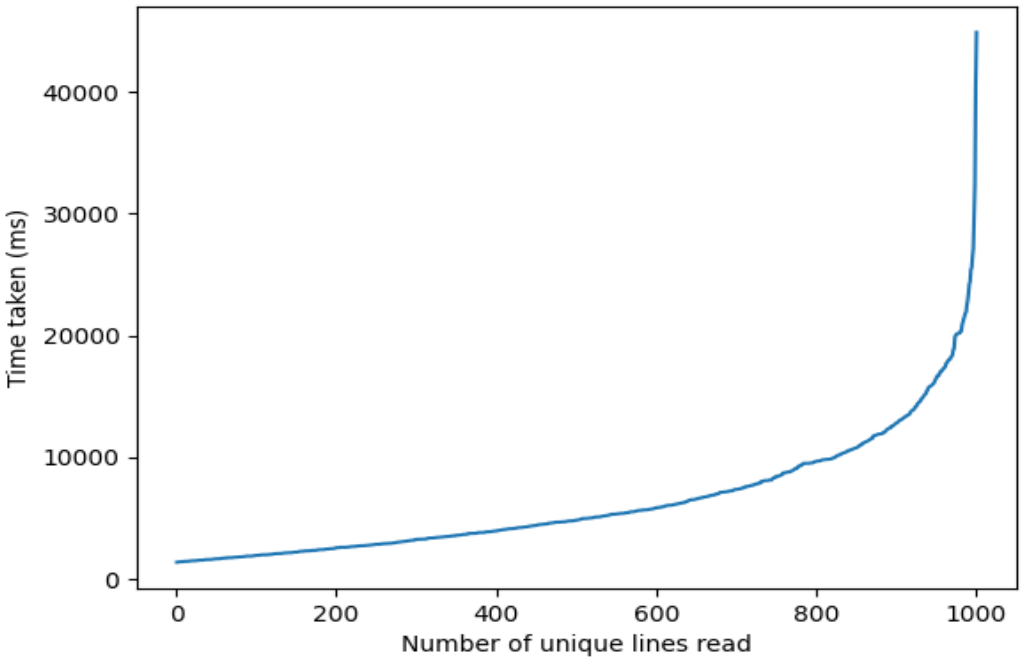

Figure 4: **Two Clients**

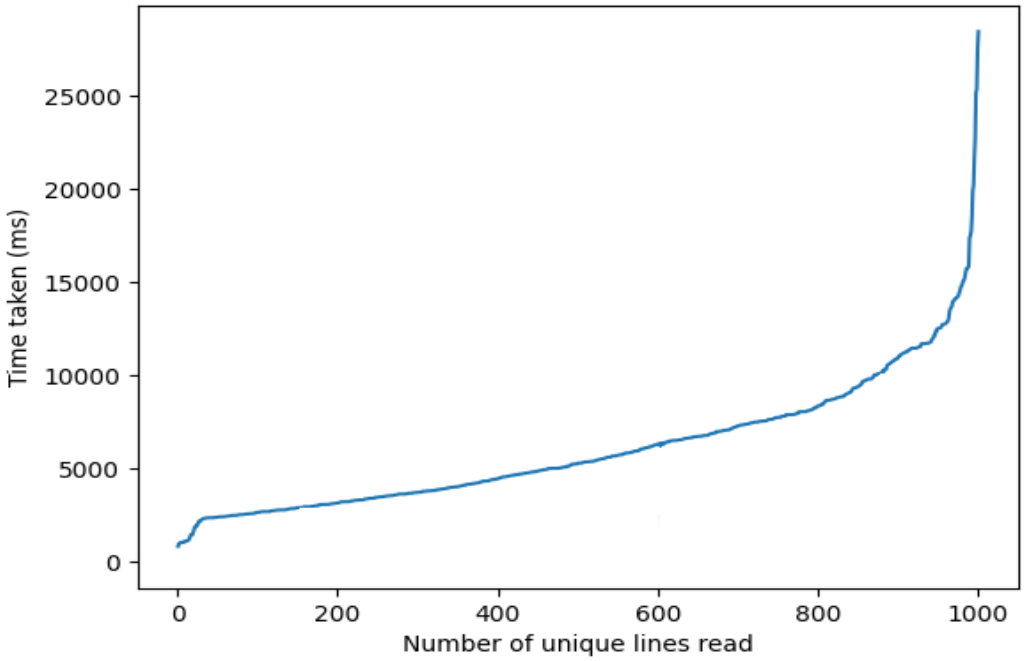

Figure 5: **Three Clients**

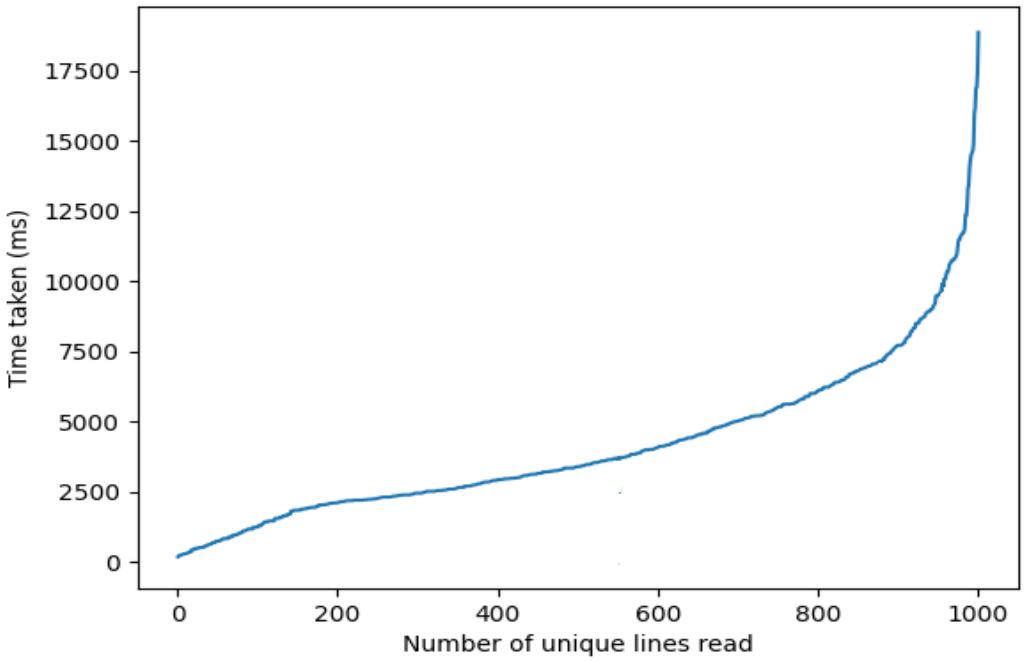

Figure 6: **Four Clients**