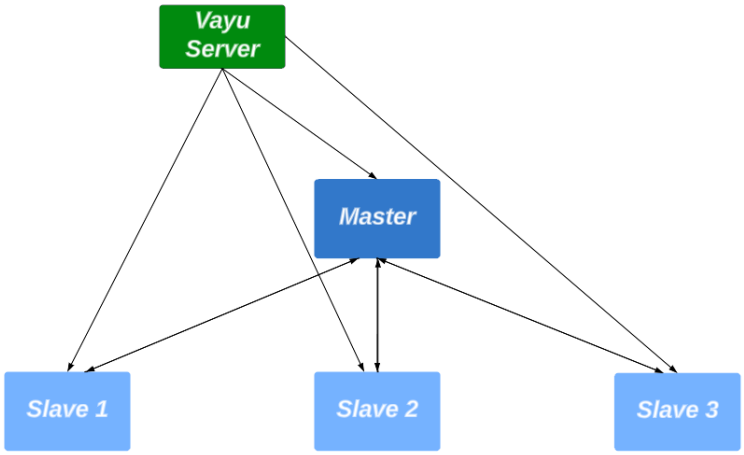
COL334 Assignment-2 Report

Amaiya Singhal 2021CS50598
Om Dehlan 2021CS10076
Akshit Goel 2021CS10071
Jaivardhan Singh 2021CS10074

Our Distribution Algorithm

Architecture

- We have implemented a **Master-Slave** P2P distribution algorithm consisting of 1 Master and 3 slaves.
- The Master and all the 3 Slaves connect to the Vayu server, and the master connects to each of the 3 slaves.
- The given flowchart pictorially explains how our devices are connected among themselves and with the Vayu server.
  - The arrows in the flowchart above show the direction in which the data is being transferred
  - All the Slaves and the Master send `SENDLINE` requests to the Vayu server and received the lines.
  - The slaves share the lines received by them from Vayu with the Master.
  - The Master shares the lines it receives from Vayu with all the 3 slaves.
  - The Master also shares the lines it receives from the Slaves with the other Slaves.
  - Since the transfer rate between our devices are much faster as compared to the connection between the Vayu server and any client (due to the rate limit from Vayu), any line received by one of the clients, the master or the slaves, is quickly passed on to all others via the master.
  - This ensures that the 4 devices send the submission to the Vayu server almost at the same time.



Implementation

Overview of the code

- We have used `python` to implement the Master and the Slave. The Master file is run on 1 system and the Slave file is run on 3 other systems.
- The `socket` library was used for establishing connections amoung the Master and the slaves as well as for the connection with the Vayu server.
- We used the `threading` library to implement multithreading. This is done so that we could simultaneously run multiple functions and receive/send from multiple clients.
  - The master has 1 thread for connection to the Vayu server and 1 thread for connection to each slave, making it a total of 4 threads.
  - Each slave has 2 threads, 1 thread for connection to the Vayu server and 1 thread for connecting with the Master.
- Each thread is associated with a function that keeps receiving the lines from its corresponding socket and sends the lines to others if required.
- A dictionary is maintained by everyone i.e the Master and the Slaves with line numbers as the key and the line contents as the value containing all the lines received by that client.
- When a new line is received by a slave from Vayu, it adds the line to the dictionary and sends it to the Master. The master then adds it to its own dictionary sends it to all the other slaves (if it had not received that line before).
- When a new line is received by the Master from Vayu, it adds it to its dictionary and shares it with all the slaves. The slave on receiving a new line from master adds it to its own dictionary.
- The lines are exchanged between the Master and the Slaves only if it was a new line for the sender, this is done to reduce the number of redundant exchanges.
- A client submits when its dictionary size is equal to 1000 i.e the total number of lines.

Ensuring Simultaneous Start of Master and Slaves

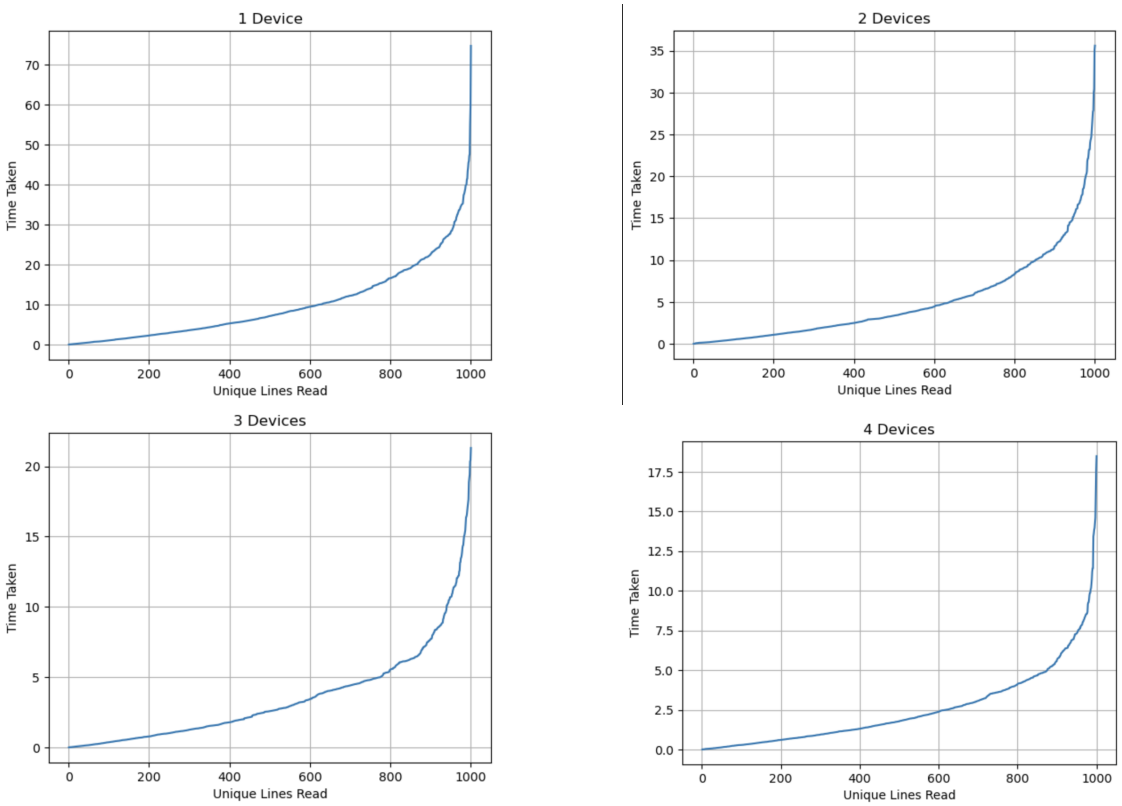
1. Master starts listening for connections from the slaves.
2. Upon connection with each slave, master sends acknowledgement message "ACK" and waits for a response message from all slaves.
3. Upon recieving the message "ACK", the slaves start the master thread and are ready to receive from the master and send an acknowledgement message "SYN-ACK" to the master.
4. When master receives this acknowledgement from all the 3 slaves it starts each of the threads responsible for receiving lines from the slaves and sends a final message "FIN" to each of them.
5. At this point, it is ensured that the Master and all the 3 slaves are ready to accept lines from one another and there **cannot be any loss of lines between them** if any one of them starts receiving lines from Vayu.
6. Immediately after this, the master also starts its thread for receiving lines from the Vayu server.
7. The slaves on receiving the message "FIN" also start their thread for connection to the Vayu server.
8. Now the network is fully connected and running.

Exception Handling

We handled two types of exceptions:

- **Client-Server disconnect:**
  - If any of our 4 client servers disconnect from the Vayu host during execution, the client server will stop sending SENDLINE requests to the Vayu server and instead create a new TCP socket with Vayu.
  - Once a new connection has been established the client process goes back to sending `SENDLINE` requests.
  - During this time the client server continues to communicate and exchange lines with the 3 slave servers (if the client is the master server) or the master server (if the client is a slave server).
- **Master-Slave disconnect:**
  - If the connection between a master and slave process breaks then these processes will stop using the old sockets.
  - Instead, a new connection will be created to exchange data.
  - While the two processes are disconnected they continue to download new lines from Vayu which they cannot share with each other, the other client is expected to download this line from Vayu (or other slaves in case of the master) directly as each process keeps track of the lines it has received.

Observations



Number of unique lines read vs Time Taken

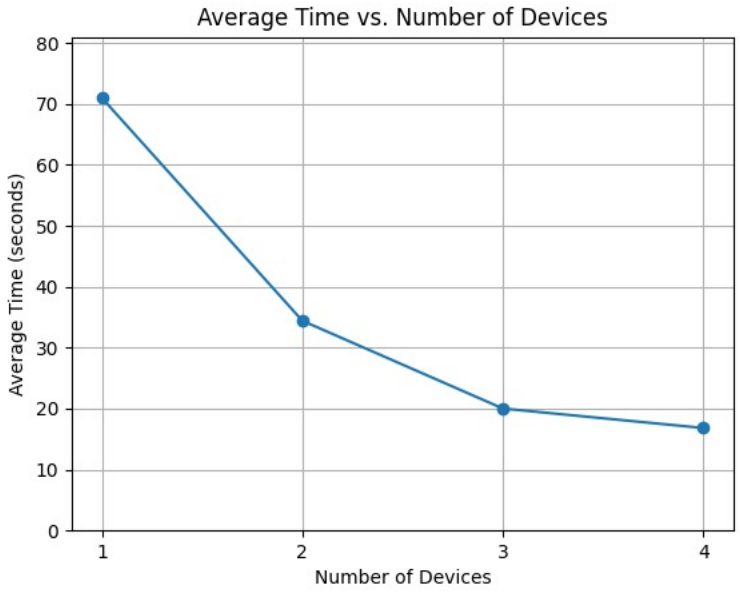
Understanding the Plots

- If we look at the single client case, once k lines have been read the probability of receiving a new line is (1000 - k)/1000. This is because we may receive any line with equal probability.
- The expected number of SENDLINE requests required to receive the (k + 1)th line is 1000/(1000 - k).
- From this expression it is clear that **the time taken to receive a new line increases** as more lines have been received.
- This is why the slope of time vs lines received graph keeps increasing.

Expected Time of Completion

- The expected number of requests to receive the 1st + 2nd +.....1000th line is 1000/1000 + 1000/999 +..... 1000/1 which is equal to 7484.47.
- Since the server is rate limited to 100 responses every second the time it would take to send these requests is 74.84 seconds.

Variation with Number of Devices



- The expected number of lines we need to receive 1000 unique lines stays constant as we add more client servers.
- However, the rate at which we can request lines increases linearly with the number of clients we use.
- Therefore, two clients can receive all the lines in half the time (or at double the rate), three clients can receive in a third of the time and four clients take a fourth of the time.
- So, the time taken to receive all the lines **does not decrease linearly** with the number of clients, it's inversely proportional to the number of clients.
- That is, as we add more clients the time difference we observe keeps decreasing instead of staying constant.

NOTE

When calculating the time required to run we ignored the time required for network latency or overhead computations required by our processes and only focused on the time it would take to receive all 1000 lines. This is because we are using multiple threads so some of these computations will be running while we are sending and receiving messages from Vayu. The time cost associated with these is negligible compared to the rate limit of one line every 10ms.

Table 1: Execution Times for Different Numbers of Devices

Number of Devices	**Run 1 (s)**	**Run 2 (s)**	**Run 3 (s)**
4 Devices (Expected 18.71s)	15.162	15.811	18.488
3 Devices (Expected 24.95s)	18.344	21.323	22.386
2 Devices (Expected 37.42s)	34.665	32.991	35.615
1 Device (Expected 74.84s)	74.717	68.018	72.327

- The runtimes we observed with different numbers of client servers support these calculations.
- As the Vayu server responds with a random line each time the slight difference in run-time can be explained by variance.