Assignment 2 - COL334

Team - bratva
Pratham, Priyanshu, Kushagra, Utkarsh

September 8, 2023

§1 Our algorithm

We have implemented a `master-slave` type algorithm. For n computers with $n \leq 4$, there is one *master* and $n-1$ *slaves*.

At any given moment, there are n computers taking the data from the server (vayu.iitd.ac.in) independently, and the master client co-ordinates between the other slave clients, by sending and receiving lines from them.

Therefore, the master client has $2*(n-1)+1 = 2n-1$ parallel threads running on the system, $2*(n-1)$ for receiving and sending to the every slave client and 1 thread for receiving input lines from the server. Note that each slave client has 3 parallel threads (2 for coordinating to the master client) and 1 thread for receiving input lines from the server.

§2 Adding more computers

§2.1 Inferences

We notice that the time for downloading the file for all computers changes from **70s** to **37s** to **27s** to **20s** for one, two, three and four systems respectively. We observe that this is **not a linear decrease.** This could primarily be due to the trend that we observe in the analysis section.

§2.2 Analysis

Further, we can estimate the number of total no. of iterations (where 1 iteration is time for one line to be fetched from Vayu) to get 1000 unique lines.

For one system, when we have read $1000-k$ unique lines, $\mathcal{P}(\text{next line is new}) = \frac{k}{1000}$. Since this is a binomial distribution, the expected number of iterations before we reach a new line $= \frac{1000}{l}$ where $l =$ number of lines yet to be read. Thus, expected number of iterations $= \sum_{n=1}^{1000} \frac{1000}{n} \approx 1000 \log 1000$.

For two systems, when the master and slave client have read $1000-k$ unique lines together (taking union - in the shared dictionary, which are being constantly updated), $\mathcal{P}(\text{next line is new}) = 1 - \mathcal{P}(\text{next line is old}) = 1 - (\frac{1000-k}{1000})^2 = \frac{2k}{1000} - (\frac{k}{1000})^2$. We note that this probability is less than $\frac{2k}{1000}$. And, hence the expected number of iterations before we reach a new line = $\frac{1000}{2l-\frac{l^2}{1000}}$ where l = number of lines yet to be read. Thus, expected number of iterations = $\sum_{n=1}^{1000} \frac{1000}{2l-\frac{l^2}{1000}} > 500 \log 1000$ ($\approx 500 \log 1000$). Therefore, the time reduces by a factor around 2, but less than 2.

Similarly, we also have similar arguments for three and four systems, with $\mathcal{P}$(next line is new) $= 1 - (\frac{1000-k}{1000})^3 = \frac{3k}{1000}$ - $(\frac{3000k^2-k^3}{1000^3})$ & $1 - (\frac{1000-k}{1000})^4 = \frac{4k}{1000} - (\frac{6*10^6k^2-4000k^2+k^4}{1000^4})$ respectively.

We note that this has some approximations such as assuming perfectly shared dictionaries and synchronized updations, which is not the case in real life, but it captures the idea why the time is around halved (but less than half) for two systems and so on. Therefore the above analysis also shows that for more than one computer, the number of lines one computer has to read such that their union reaches 1000 unique lines would not be linearly decreasing.

§2.3 Plots

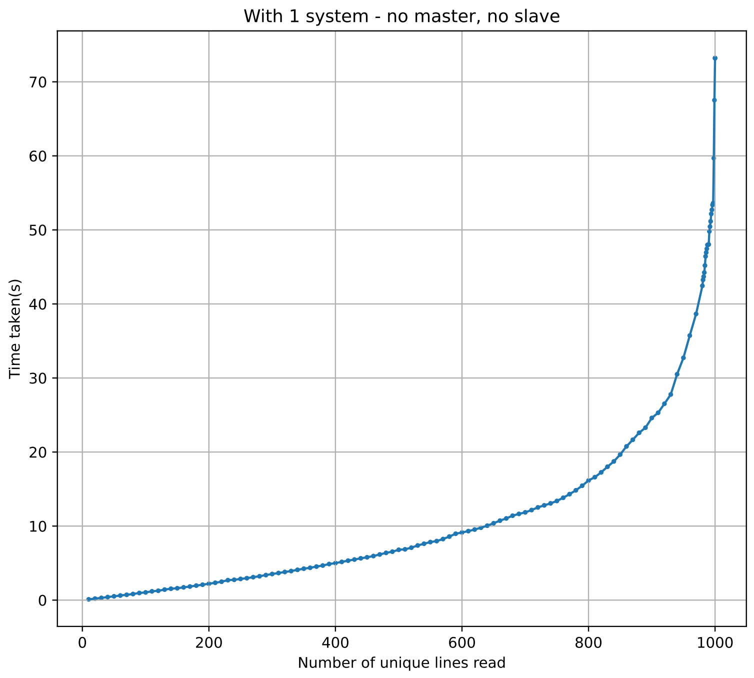


Figure 1: A single client reads 1000 unique lines in 72s

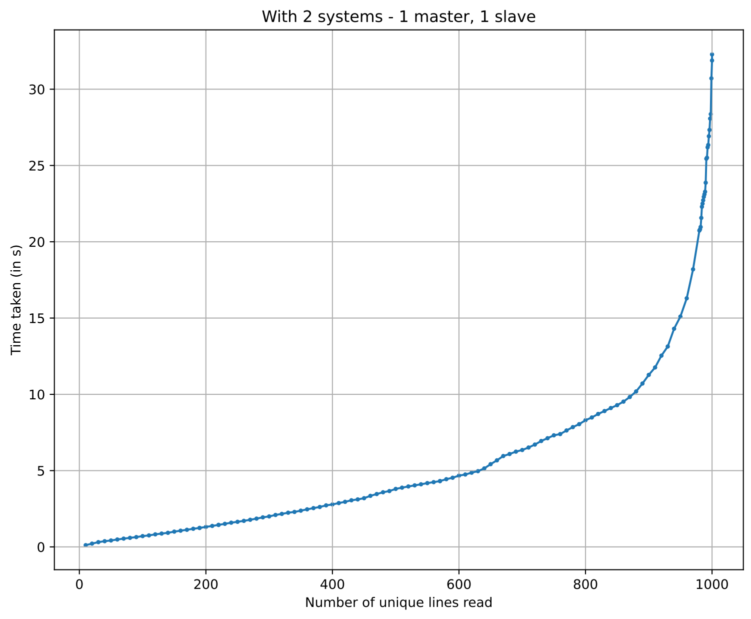


Figure 2: 2 clients reads 1000 unique lines in 37s

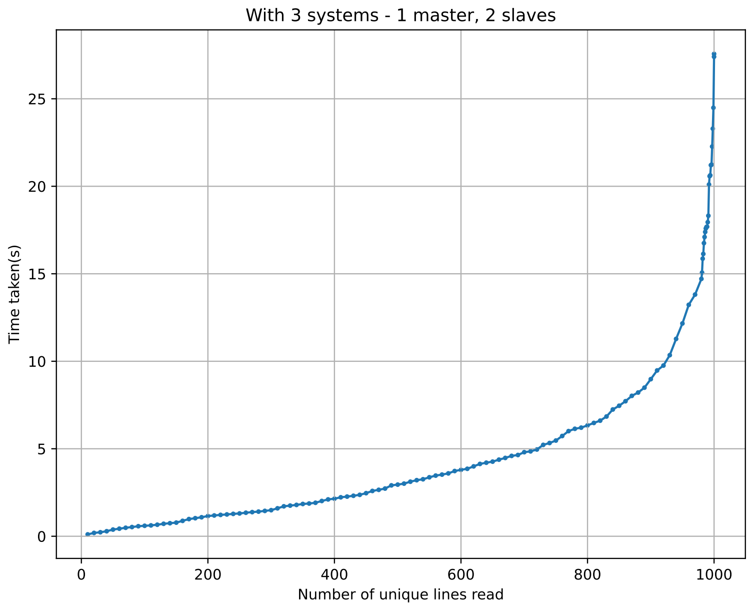


Figure 3: 3 clients reads 1000 unique lines in 27s

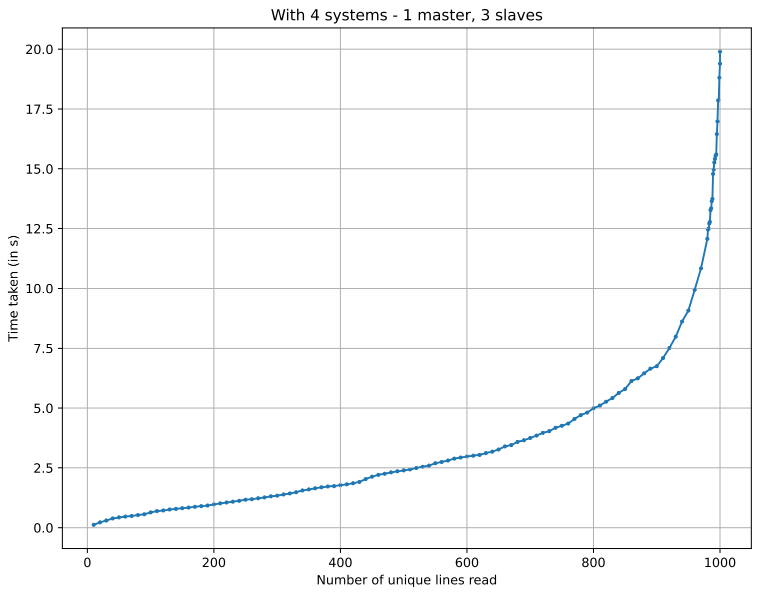


Figure 4: 4 clients reads 1000 unique lines in 20s

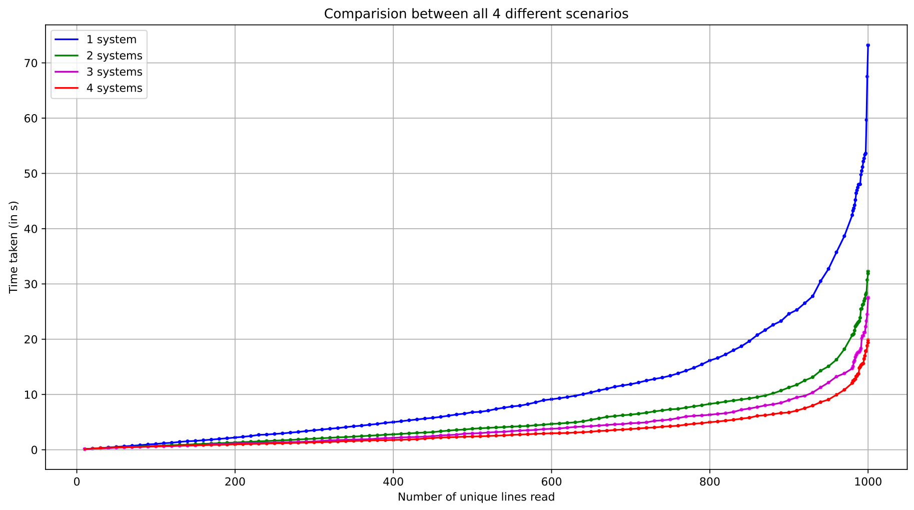


Figure 5: Comparison of all 4 scenarios

§3 Exception Handling

§3.1 Failed Connections

`Connection reset by peer` error appears when we get an IP address which had been given to somebody else just a little while ago, and their connection has not been closed properly.
We handle this and similar failed connections by excepting the error until the connection is successfully established by retrying 5 times at max.

```
MAX_RECONNECT_ATTEMPTS = 5
reconnect_attempts = 0
while reconnect_attempts < MAX_RECONNECT_ATTEMPTS:
    try:
        client_socket.send(message.encode())
        received_message = client_socket.recv(1024)
        break  # Connection successful, exit the loop
    except (BrokenPipeError, ConnectionRefusedError, TimeoutError) as e:
        reconnect_attempts += 1
        if reconnect_attempts >= MAX_RECONNECT_ATTEMPTS:
            print("Maximum reconnection attempts reached. Exiting.")
            return
    print(f"Reconnecting (Attempt {reconnect_attempts})...")
    client_socket.connect(server_address)
```

§3.2 Broken Pipes

In case there is a slight synchronization error delay, and the connection is closed before the input is sent, a broken pipe exception is shown on the system which tries to send data through the network.

In such cases, we restart the network by attempting to connect the two systems, and begin the threads as soon as the connection is established and we start receiving input lines from the server.

§3.3 Reconnection

We have handled exceptions for the following reconnection cases:

- If a client node in our master-slave network suddenly dies and wants to reconnect.
- If a client-server connection **breaks** and the client **reconnects**.
- If the client program stops, (i.e. client.py stops running and is rerun again), the master client waits for 5 seconds (after which the connection is closed), and then master client sends the 1000 lines to the client once again after reconnection.

```
try:
    conn.send(line_details)
    except (BrokenPipeError, ConnectionRefusedError, TimeoutError) as e:
        break
    try:
        received_message= conn.recv(1024).decode()
    except OSError as e:
        break
    if (received_message == "RECEIVED"):
        index1+=1

client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.settimeout(5)
try:
    if (total_received==1000):
        client_socket.close()
        return
    client_socket.bind((host, port))
    client_socket.listen()
    conn, address = client_socket.accept()
except socket.timeout:
    print("Timeout: No connection received from slave 1 within 5 seconds.")
    client_socket.close()
    return
```

§4 Implementation

§4.1 Parsing

We ensured that since the data being received from the server in packets, each line could comprise of multiple packets, and so the packets are taken until it ends in a newline `"\n"` character, and appended to form the whole line.

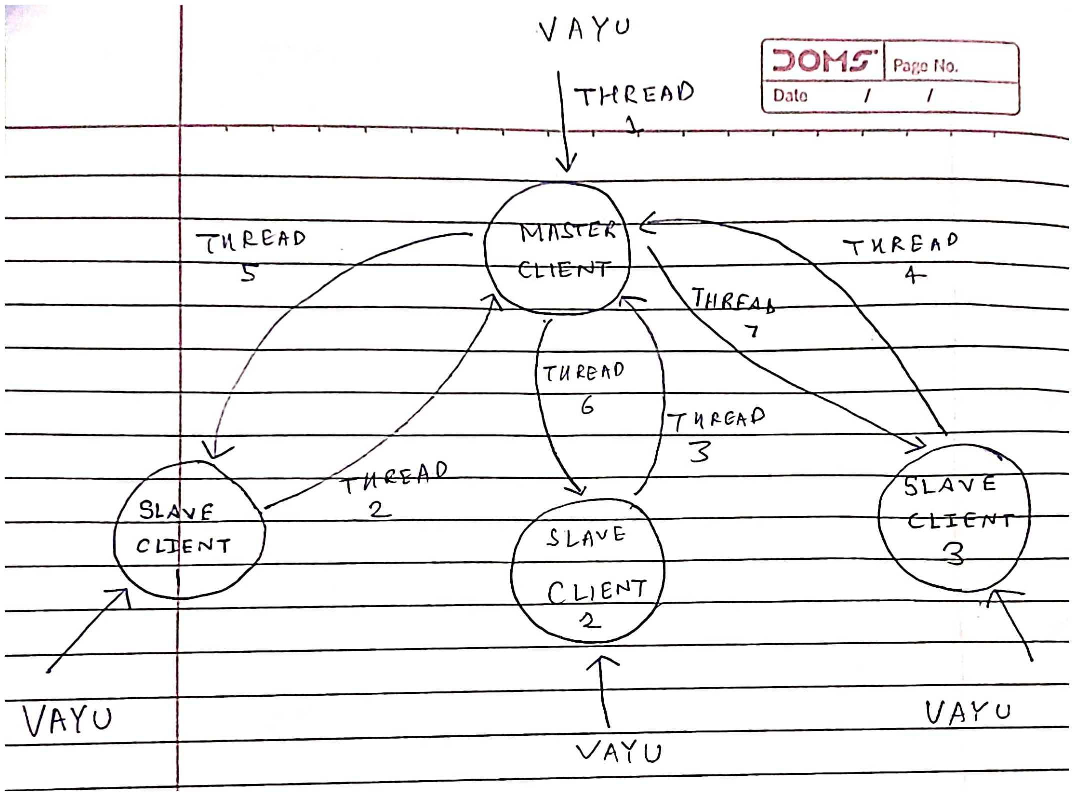
On the contrary, while sending lines to the other clients, even one packet could have multiple newline `"\n"` characters, so we scanned through the line to break the packet, corresponding to different lines.

§4.2 Client and Master roles

`master.py`: The master server's code contains 1 thread for communicating with Vayu, i.e. receiving lines from it. There are 6 other threads for sending and receiving lines from three other clients (slaves). The 3 receiving threads depend on `client.py`. The working of the three sending threads is as follows -
There is a an array of distinct line numbers that the master has read. There are three indices where index_i shows that master has send client_i lines numbered from 0 to index_i. Thus whenever $\text{index}_i <$ size of the array, the sending thread for client i sends that required line to the client. This ensures that at any given time the master will try to send all of the distinct lines it just received to all the clients, if it hasn't send them before in an efficient way.

`client.py`: Whereas master stores a dictionary of line number and lines that also represents the union of all distinct lines that all servers have read, the clients keep sending all new lines that they receive to master at any given moment. This is done in parallel with a separate thread to receive all the lines that master sends to the client i as described above. Does a client works with two threads for sending and receiving with the master, and has a separate one for receiving content from Vayu.

§4.3 FSM representing our implementation



Note that all the systems take input lines from Vayu server and exchange them between each other using different threads. (7 for master and 3 for slave) (The threads in the figure are marked for the master for clarity)