Anjali Singh (2023JCS2565) & Sharath Kumar Reddy Gandham(2023JCS2542) & Prince Gupta (2023JCS2561) & Akash dabaniya (2021CS10111)

1. Give the Client Code.

Solution:

```
import socket
import time
import pickle
from collections import OrderedDict

# vayu_server
server1_ip = "10.237.26.109"
server1_port = 9801

# master server
server2_ip = '10.194.39.228'
server2_port = 2831
server1_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server2_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
buffer = set()
conn = 0
while conn<10:
        try:
            server1_socket.connect((server1_ip, server1_port))
            break
        except ConnectionRefusedError:
            print("Server is not available. Retrying...")
            conn+=1
while conn<10:
        try:
            server2_socket.connect((server2_ip, server2_port))
            break
        except ConnectionRefusedError:
            print("Server is not available. Retrying...")
            conn+=1
start_time = time.time()
dic = OrderedDict()
while True:
    # Send data to the server
    message = "SENDLINE\n"
    server1_socket.send(message.encode('utf-8'))

    # Receive data from the server
    count = 0
    code = ""
    while count != 2:
        row_data = server1_socket.recv(1024)
        code += row_data.decode('utf-8')
        count = code.count('\n')
    print(code)

    lines = code.split('\n')
    li = lines[0]
```

```
    line = lines[1]
    if li != '-1' and li not in buffer:
        buffer.add(li)
        server2_socket.send(code.encode('utf-8'))
        response = server2_socket.recv(1024).decode('utf-8')
        print(response)
        if int(response) == 0:
            # d1 = server2_socket.recv(1024)
            c = 0
            file = b''
            while True:
                row_data = server2_socket.recv(1024)
                if not row_data:
                    break
                file += row_data
            dic = pickle.loads(file)
            print(dic)
            print(len(dic))
            msg = 'SUBMIT\n'
            server1_socket.send(msg.encode('utf-8'))
            idk = '2023JCS2561@asap\n'
            server1_socket.send(idk.encode('utf-8'))
            total_data = '1000\n'
            server1_socket.send(total_data.encode('utf-8'))
            size = len(dic)
            if size == 1000:
                dic['1000'] = "this is dammy line"

            for key in dic.keys():
                k = key+'\n'
                v = dic[key]+'\n'
                c += 1
                if c > 1000:
                    resp = server1_socket.recv(1024).decode('utf-8')
                    print(resp)
                    break
                server1_socket.send(k.encode('utf-8'))
                server1_socket.send(v.encode('utf-8'))
            server1_socket.close()
            break

end_time = time.time()
elapsed_time = end_time - start_time

print(elapsed_time)
```

2. Give the Master Code.

Solution:

```
import socket
import threading
import time
```

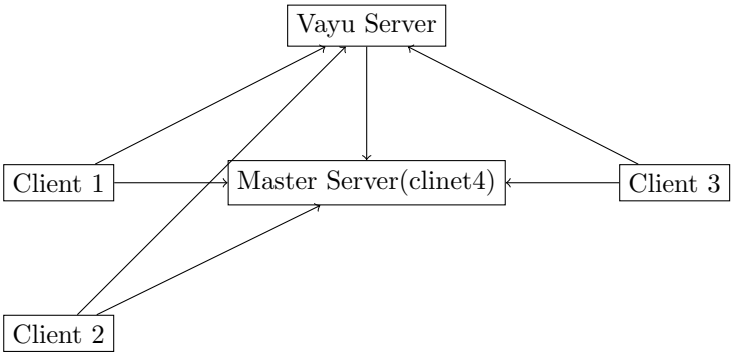
```
import pickle
import matplotlib.pyplot as plt
from collections import OrderedDict
lock = threading.Lock()

# Define the server address and port
server_ip = '10.194.39.228'
server_port = 2831

# Create a socket object
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Bind the socket to the server address and port
server_socket.bind((server_ip, server_port))

# Listen for incoming connections (up to 5 clients)
server_socket.listen(3)
print(f"Server listening on {server_ip}:{server_port}")

buffer = set()
dic = OrderedDict()

vayu_ip = "10.237.26.109"
vayu_port = 9801

start_time = time.time()

x = []
y = []


def receive_from_vayu():
    j = 0
    vayu_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    while j < 10:
        try:
            vayu_socket.connect((vayu_ip, vayu_port))
            break
        except:
            print("vayu server is connected")
            j += 1
    while True:
        # Send data to the server
        message = "SENDLINE\n"
        vayu_socket.send(message.encode('utf-8'))
        # Receive data from the server
        count = 0
        deCode = ""
        while count != 2:
            row_dat = vayu_socket.recv(1024)
            deCode += row_dat.decode('utf-8')
            count = deCode.count('\n')

        lines = deCode.split('\n')
        l = lines[0].split()[-1]

        if l != '-1' and l not in buffer:
```

```
            with lock:

                dic[lines[0]] = lines[1]
                buffer.add(l)
        if len(buffer) >= 1000:
            c = 0
            msg = 'SUBMIT\n'
            vayu_socket.send(msg.encode('utf-8'))
            idk = '2023JCS2542@asap\n'
            vayu_socket.send(idk.encode('utf-8'))
            total_data = '1000\n'
            vayu_socket.send(total_data.encode('utf-8'))
            size = len(dic)
            if size == 1000:
                dic['1000'] = "this is dammy line"
            for key in dic.keys():
                k = key + '\n'
                v = dic[key] + '\n'
                c += 1
                if c > 1000:
                    resp = vayu_socket.recv(1024).decode('utf-8')
                    print(resp)
                    break
                vayu_socket.send(k.encode('utf-8'))
                vayu_socket.send(v.encode('utf-8'))

            vayu_socket.close()
            break


# Function to handle client connections
def handle_client(conn, addr):
    print(f"[NEW CONNECTION] {addr} connected.")

    while True:
        with lock:
            count = 0
            data = ""
            while count != 2:
                row_data = conn.recv(1024)
                data += row_data.decode('utf-8')
                count = data.count('\n')
            x.append(time.time()-start_time)
            salines = data.split('\n')

            if salines[0] not in buffer:
                dic[salines[0]] = salines[1]
            buffer.add(salines[0])
            length = len(buffer)
            y.append(len(buffer))
        if length >= 1000:
            response_message = b'0'
            conn.send(response_message)
            pickle_data = pickle.dumps(dic)
            conn.send(pickle_data)
            print(f"[DISCONNECTED] {addr} disconnected")
```

```
            conn.close()
            break

        else:
            response_message = b'1'  # encoding 1 as a byte
            conn.send(response_message)


server = threading.Thread(target=receive_from_vayu, args=())
server.start()

# Accept a client connection
conn1, addr1 = server_socket.accept()
print(f"Accepted connection from {addr1}")

# Start a new thread to handle the client
client_handler1 = threading.Thread(target=handle_client, args=(conn1, addr1)
    )
client_handler1.start()

# Accept a client connection
conn2, addr2 = server_socket.accept()
print(f"Accepted connection from {addr2}")

# Start a new thread to handle the client
client_handler2 = threading.Thread(target=handle_client, args=(conn2, addr2)
    )
client_handler2.start()

conn3, addr3 = server_socket.accept()
print(f"Accepted connection from {addr3}")

# Start a new thread to handle the client
client_handler3 = threading.Thread(target=handle_client, args=(conn3, addr3)
    )
client_handler3.start()

client_handler1.join()
client_handler2.join()
client_handler3.join()
server.join()

end_time = time.time()
elapsed_time = end_time - start_time
print(elapsed_time)
plt.plot(y, x)
plt.title("4 clients")
plt.show()
```

3. Explain the distributed algorithm you have implemented.

Solution:



- **Vayu Server**
  - The central server, known as the Vayu server, is responsible for handling incoming connections from multiple clients.
  - The Vayu server listens for incoming connections and allows clients to connect to it.
- **Client Behavior**
  - Four clients connect to the Vayu server. These clients could be individual devices or computers running software that communicates with the server.
- **Master Server (One of the Clients)**
  - One of the four clients assumes a dual role as both a regular client and a master server.
  - This master server is responsible for collecting data from all clients connected to it, including itself.
- **Data Transfer**
  - All clients, including the master server, fetch data from the Vayu server. They can request and receive data from the Vayu server as needed.
- **Data Compilation**
  - The master server collects data from itself and the other clients, presumably waiting until it has received a total of 1000 lines of data or until some other condition is met.
- **File Completion**
  - Once the master server has compiled the required amount of data (e.g., 1000 lines), it processes this data as needed. It could be saving it to a file, performing calculations, or any other relevant operation.
- **Closing Client Connections(Master Server)**
  - After the master server has completed its task and the file is ready, it initiates the process of closing the connections to all clients, including itself. This ensures that no further data is sent or received.
- **Client Response (All Clients)**

- All clients, including the master server, respond to the request to close the connection. They release any resources associated with the connection.

- **Closing Vayu Server Connections (All Clients)**
  - Once all clients have closed their connections to the master server, they can proceed to close their connections to the Vayu server.

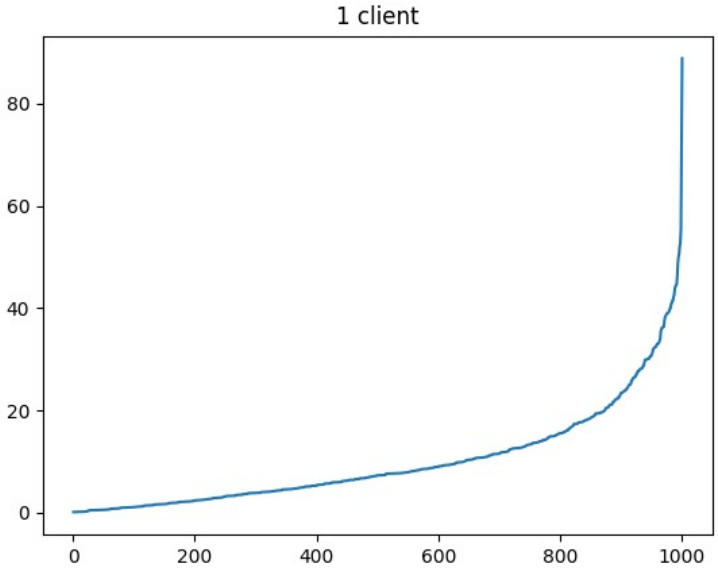
- **Finalization**
  - At this point, all connections are closed, and the communication between clients and the Vayu server has ended.

We have one Vayu server and four clients. One client is acting both as a master server and a client. All clients are connected to the Vayu server, fetching data from it, and sending it to the master server. Master server keeps on compiling the data received from all clients, and when the file is complete,i.e, of 1000 lines the master server closes connection of all clients to it. And then all the clients close their connection to vayu. We use try-and-expect error handling for connecting to both the master server and Vayu server if the server is not available. Attempting 10 times to connect with the server. after that will terminate.

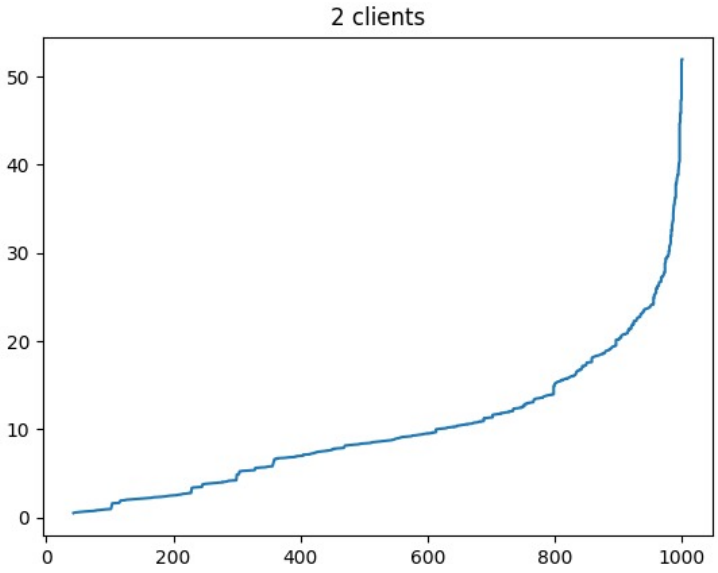
4. Show graphs with one client downloading the file by itself, with two clients working in cooperation, then three clients, and finally, four clients. Does the download time reduce linearly as you add more and more clients?

Solution: In all the graphs the horizontal axis represent the number of lines received and the vertical axis represent the time taken in seconds.

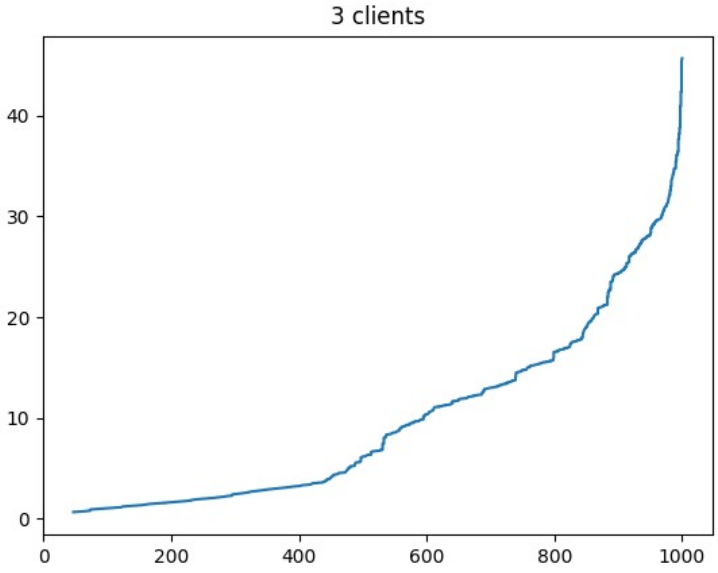
(a) With one Client



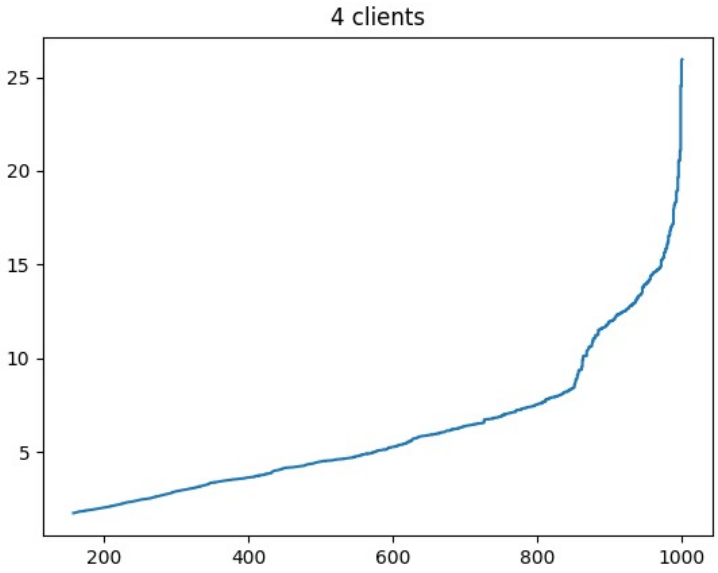
(b) With two Clients



(c) With three Clients



(d) With four clients



As we can see from the above graphs for one client we are having **83.06** sec time, for two clients we are having **52.07** sec time, for three clients we are having **44.08** sec time and for four clients we are taking **26.07 sec**.
So, the time to download the whole file is decreasing with increasing number of clients.