server.py

```
from functools import reduce
from dateutil import parser
import threading
import datetime
import socket
import time

# Data structure to store client address and clock data
client_data = {}

''' Nested thread function to receive clock time from a connected client '''
def startReceivingClockTime(connector, address):
    while True:
        try:
            # Receive clock time
            clock_time_string = connector.recv(1024).decode()
            clock_time = parser.parse(clock_time_string)
            clock_time_diff = datetime.datetime.now() - clock_time

            client_data[address] = {
                "clock_time": clock_time,
                "time_difference": clock_time_diff,
                "connector": connector
            }
            print("Client Data updated with: " + str(address), end="\n\n")

        except Exception as e:
            print(f"Error receiving time from {address}: {e}")
            break

        time.sleep(5)

''' Master thread function to accept clients over given port '''
def startConnecting(master_server):
    while True:
        # Accept a client connection
```

```
        master_slave_connector, addr = master_server.accept()
        slave_address = f"{addr[0]}:{addr[1]}"
        print(f"{slave_address} connected successfully.")

        current_thread = threading.Thread(
            target=startReceivingClockTime,
            args=(master_slave_connector, slave_address,)
        )
        current_thread.start()

# Function to calculate average clock difference
def getAverageClockDiff():
    if not client_data:
        return datetime.timedelta(0, 0)  # No clients, return zero difference

    time_difference_list = [client["time_difference"] for client in client_data.values()]
    sum_of_clock_difference = sum(time_difference_list, datetime.timedelta(0, 0))
    average_clock_difference = sum_of_clock_difference / len(client_data)

    return average_clock_difference

''' Master sync thread function to synchronize all clocks '''
def synchronizeAllClocks():
    while True:
        print("\nNew synchronization cycle started.")
        print(f"Number of clients to be synchronized: {len(client_data)}")

        if client_data:
            average_clock_difference = getAverageClockDiff()
            for client_addr, client in client_data.items():
                try:
                    synchronized_time = datetime.datetime.now() + average_clock_difference
                    client["connector"].send(str(synchronized_time).encode())
                    print(f"Sent synchronized time to {client_addr}")

                except Exception as e:
                    print(f"Error sending synchronized time to {client_addr}: {e}")
```

```
        else:
            print("No client data. Synchronization not applicable.")

        print("\n")
        time.sleep(5)

# Function to start the Clock Server / Master Node
def initiateClockServer(port=8080):
    master_server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    master_server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    print("Socket at master node created successfully\n")

    master_server.bind(('', port))
    master_server.listen(10)
    print("Clock server started...\n")

    # Start accepting client connections
    print("Starting to make connections...\n")
    master_thread = threading.Thread(target=startConnecting, args=(master_server,))
    master_thread.start()

    # Start synchronization
    print("Starting synchronization parallelly...\n")
    sync_thread = threading.Thread(target=synchronizeAllClocks, args=())
    sync_thread.start()

# Driver function
if __name__ == "__main__":
    initiateClockServer(port=8080)
```

client.py

```
from dateutil import parser
import threading
import datetime
import socket
import time

# Client thread function to send the current system time to the server
def startSendingTime(slave_client):
    while True:
        try:
            slave_client.send(str(datetime.datetime.now()).encode())
            print("Recent time sent successfully", end="\n\n")
        except Exception as e:
            print(f"Error sending time to server: {e}")
            break

        time.sleep(5)

# Client thread function to receive synchronized time from the server
def startReceivingTime(slave_client):
    while True:
        try:
            synchronized_time = parser.parse(slave_client.recv(1024).decode())
            print(f"Synchronized time at the client: {synchronized_time}", end="\n\n")
        except Exception as e:
            print(f"Error receiving synchronized time: {e}")
            break

# Function to start the Slave Client
def initiateSlaveClient(port=8080):
    slave_client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    try:
        # Connect to the Clock Server (Master Node)
        slave_client.connect(("127.0.0.1", port))
```

```
    except Exception as e:
        print(f"Error connecting to server: {e}")
        return

    # Start sending time to server
    print("Starting to send time to server...\n")
    send_time_thread = threading.Thread(target=startSendingTime, args=(slave_client,))
    send_time_thread.start()

    # Start receiving synchronized time from server
    print("Starting to receive synchronized time from server...\n")
    receive_time_thread = threading.Thread(target=startReceivingTime, args=(slave_client,))
    receive_time_thread.start()

# Driver function
if __name__ == "__main__":
    initiateSlaveClient(port=8080)
```