Rayan Ghaffar
CMPSC 497.1
Lab 2

Description

The Python system I built consists of two programs, a server and user client. These files replicate a data server and a computer client that accesses a server. Multiple server and client objects can be instantiated to simulate multiple servers and clients interacting with each other. Clients can directly communicate with their specified server and servers can propagate and replicate updates to all other servers in. The goal of this system is to demonstrate how casual consistency, a process in which related operations happen in proper order, is maintained when messages span multiple servers and multiple clients.

I achieved this using Lamport's Algorithm in which every message is timestamped locally by each server of when it is received and if a timestamp of an incoming process is too early, a delayed time is added on to allow the clocks to not interfere. I have the first process to compare with timestamp 0, this is so the initial process will always go through. This project heavily utilizes key concepts covered in class about casual consistency. I was able make the client-server interface work through the use of socket programming and threading. This allowed me to send data between two end points while processes ran without being disturbed.

Several data structures were used to store information. Namely, I used a dictionary called "data_store" that had a key being the key of the process and a value being a nested tuple of (message value, (timestamp, serverID). I set the client and server programs to print useful information throughout the process such as when a client/server is connected, all servers connected to a specific server, information in a server was updated, what the update contained, the current data_store after a read operation, if a write maintained casually consistency, if a write must be delayed, if a replication was successful, if a user command was requested, if the request passes or failed, and the data obtained from the success of the request.

Working Parts

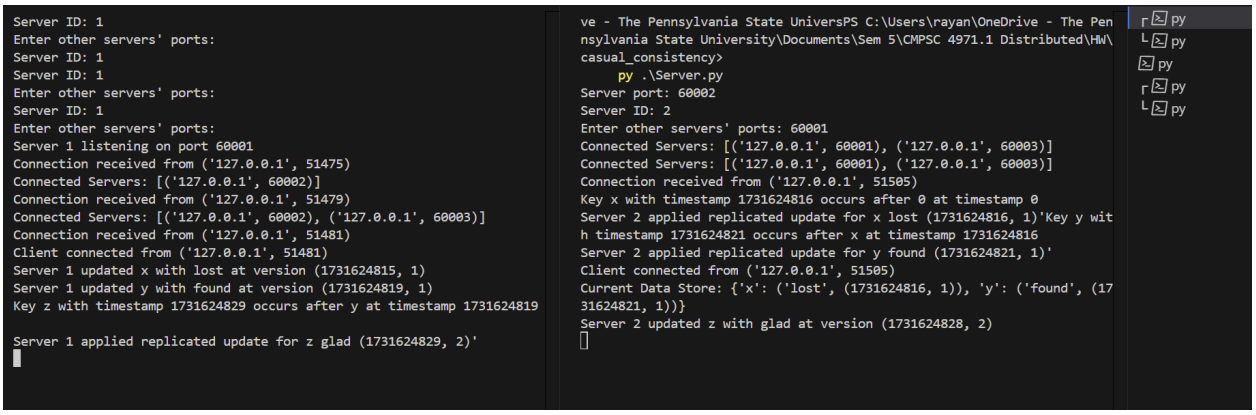
For the most part, everything assigned in the instructions of the project works successfully. My programs are able to take a client's write request, replicate to all other clients connected to the server, propagate the update to other servers, and have the those servers replicate the update to their clients. My system is able to properly implement Lamport's clock using timestamps, and in the case of an inconsistency, the message will be delayed and the server will output the details of why it would not work.

Not Working Part

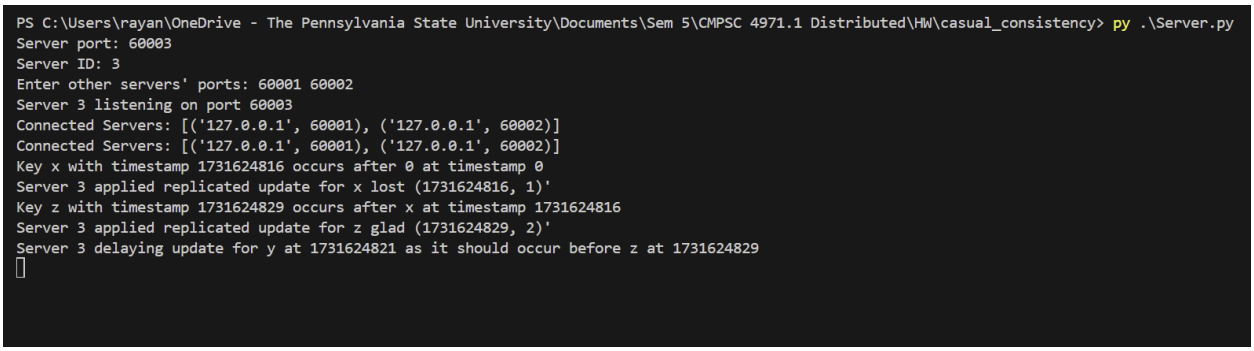
As stated earlier, my programs complete the tasks assigned. The main issue I have now is that test case from the assignment description has to have a hardcoded time.sleep() delay to trigger the processes being out of sync as I am running all the servers and clients on my personal computer in which the clock will always be consistent as it is one device. I used the sleep delay as stated in the description file and I can comment this out to show my program still works when the times are consistent, so I do not have any non-working parts.

Sample Output

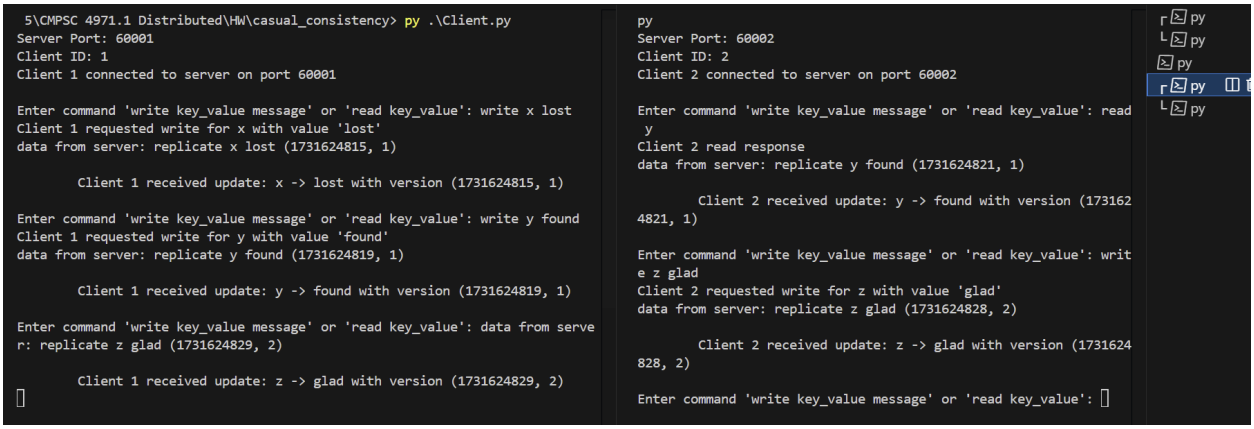
Picture 1: left side is server 1 (West Coast), right side is server 2 (Central Area). Servers are created, connected together with server 3, respective clients. Server 1 gets write requests for x and y, while server 2 compares the timestamps and replicates if casually consistent. Server 2 print data store when there is a read request from the client. Both servers are updated with write request z at the end.



Picture 2: server 3 (East Coast). This is the server where the delay will happen. Server is connected with other servers, approves write request x and z as they are consistent. Delays write request y as its timestamp should have been earlier but arrives later breaking the consistency



Picture 3: left side is client 1 (Alice on west coast), right is client 2 (Bob on central area). Both are instantiated and connected to their servers. Client 1 write request x and y. Client 2 read request returns y found as its server accepted and replicated the earlier write requests. Client 2 write request z glad which is replicated over to client 1 as it is consistent with the timestamps.



Running Program

This utilizes the terminal where each client/server is a new terminal instance. Save the Py files into any directory you choose. Open a terminal instance and set the path to be where the files are located. To create a server, run in the terminal “py Server.py”; likewise, to create a client run “py Client.py”. My system depends on all servers being created first, then the clients will be created. Connecting servers to

other servers or clients is done during the instantiation (the first server will not be connected to anything at the time of); this can be done by following the sample input/output provided. Read/write commands for the user follow a similar nature. If all servers aren't instantiated before the clients, some issues with threading, buffers, and connectivity could occur but the program is generally still usable.