Kartik Ugemuge

CMPSC 497

Lab – 2

Detailed Description of the System and Protocol Specifications

This lab simulates a distributed data store with multiple data centers which use causal consistency. Each data center is capable of handling client read and write requests while maintaining a global Lamport clock to ensure consistent ordering of operations across all centers. The system uses socket programming similar to lab 1 for communication and a dependency tracking mechanism to check causal relationships.

Protocol Specifications:

- Communication Model: The system uses TCP/IP for reliable communication between clients and data centers. Data centers also communicate with each other to ensure causal ordering.
- Client Requests: Clients can issue read and write commands in the format: port number "command key value". For example, 8000 “write x lost ring".
- Causal Consistency: Achieved using Lamport clocks and dependency tracking. Writes are delayed until all dependencies are met, to simulate real life instances.

Structure of the Program

Global Variable:

- version_counter: A global counter used as Lamport Clock.

Classes and Functions:

- DependencyTracker: Stores a dictionary and uses functions to manage this dependency list.
- handle_client(): Handles client requests for reading, writing, and replicate writting data. It updates the global Lamport clock and performs dependency tracking.
- replicate_write(): Sends replicated write operations to other data centers, uses sleep() for network delays
- handle_replication(): Checks if all dependencies are met and commits write action.
- start_data_center(): Initializes the data center and handles incoming client and replication connections.
- client_behavior(): Simulates client/peer behavior, sending read or write requests to the data center.

Main Execution:

- Starts multiple data centers on different ports using different terminal windows and simulates client actions.

Description of What Works and What Doesn't

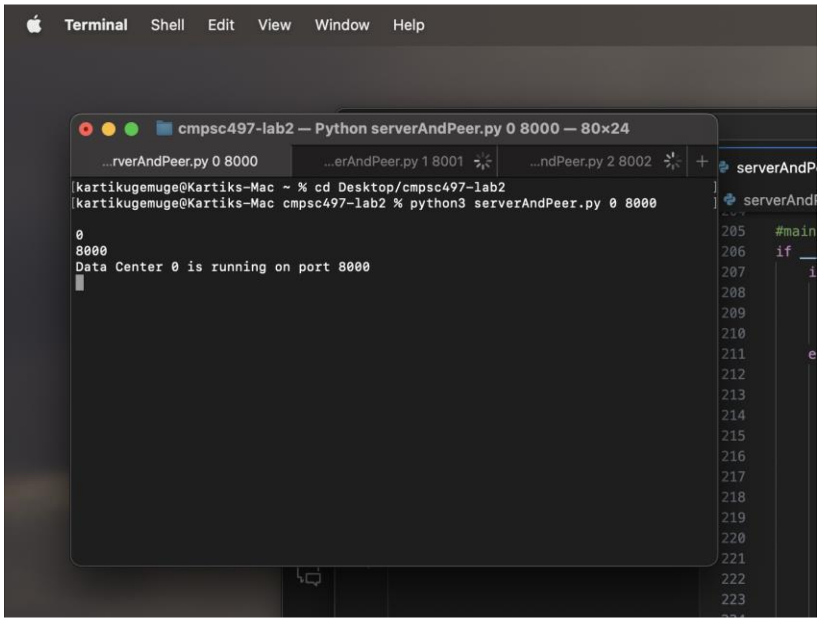
What works:

- Everything mentioned in the lab description works, the example mentioned in the lab description can be simulated.

What does not work:

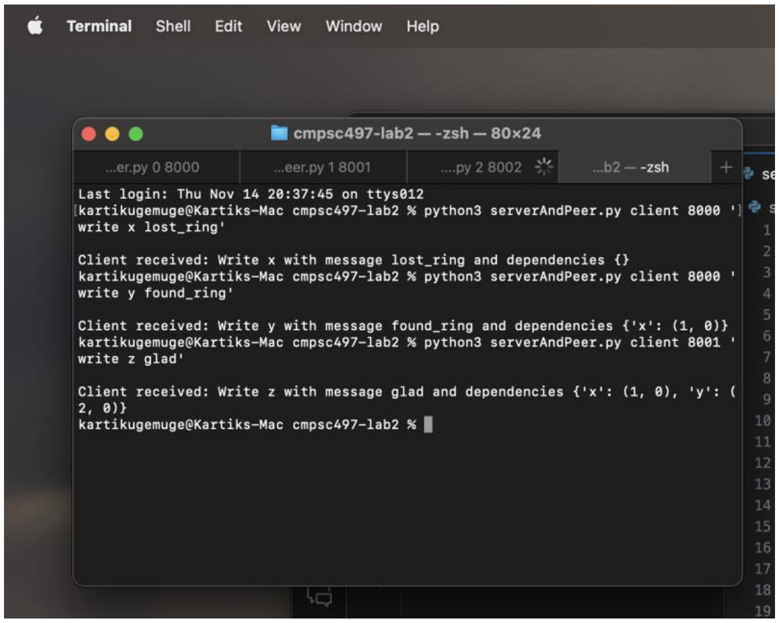
- As mentioned in the lab description, this lab does not contain write-conflicts and garbage collection as it was an optional requirement. So, the last-write-wins policy is not used.
- The implementation is limited to 3 data centers. So, it will not scale for additional data centers without changing the code.

Sample Output of the Program



```
kartikugemuge@Kartiks-Mac ~ % cd Desktop/cmpsc497-lab2
kartikugemuge@Kartiks-Mac cmpsc497-lab2 % python3 serverAndPeer.py 0 8000

0
8000
Data Center 0 is running on port 8000
```

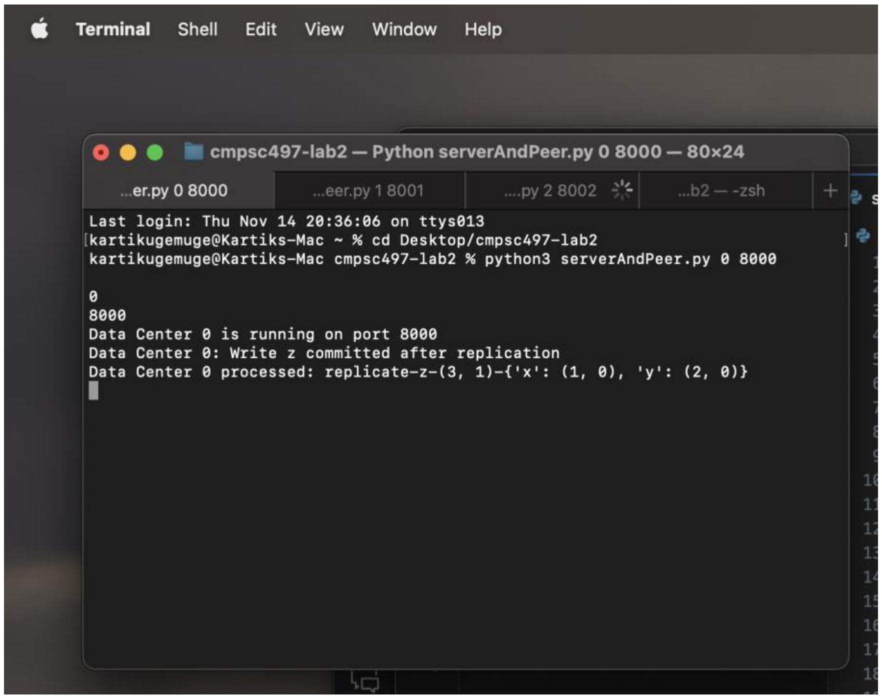
This image shows Data Center 0 being setup on port number 8000



```
Last login: Thu Nov 14 20:37:45 on ttys012
[kartikugemuge@Kartiks-Mac cmpsc497-lab2 % python3 serverAndPeer.py client 8000 ']
write x lost_ring'

Client received: Write x with message lost_ring and dependencies {}
kartikugemuge@Kartiks-Mac cmpsc497-lab2 % python3 serverAndPeer.py client 8000 '
write y found_ring'

Client received: Write y with message found_ring and dependencies {'x': (1, 0)}
kartikugemuge@Kartiks-Mac cmpsc497-lab2 % python3 serverAndPeer.py client 8001 '
write z glad'

Client received: Write z with message glad and dependencies {'x': (1, 0), 'y': (
2, 0)}
kartikugemuge@Kartiks-Mac cmpsc497-lab2 %
```

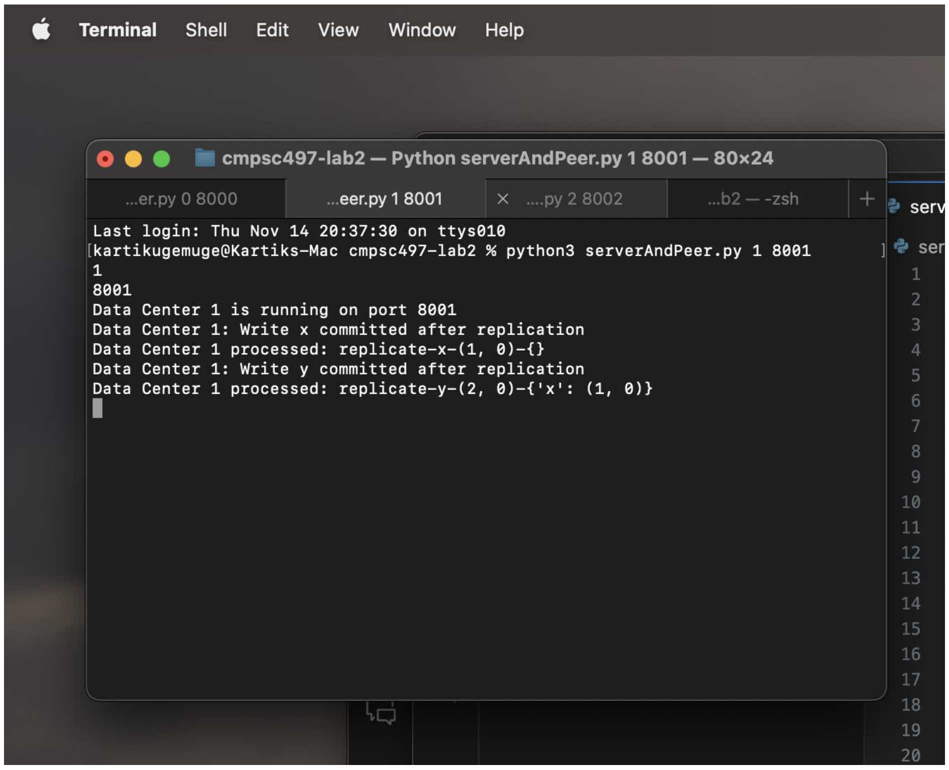
Here the clients for Data Center 0 and 1 are being simulated, each receive the confirmation of message written instantly.

Note: The delay between Data Centers 0 and 1 is much smaller than the Delay from Data center 0 and 2.


Terminal Shell Edit View Window Help
cmpsc497-lab2 — Python serverAndPeer.py 0 8000 — 80×24
...er.py 0 8000
...eer.py 1 8001
....py 2 8002
...b2 — -zsh

```
Last login: Thu Nov 14 20:36:06 on ttys013
kartikugemuge@Kartiks-Mac ~ % cd Desktop/cmpsc497-lab2
kartikugemuge@Kartiks-Mac cmpsc497-lab2 % python3 serverAndPeer.py 0 8000

0
8000
Data Center 0 is running on port 8000
Data Center 0: Write z committed after replication
Data Center 0 processed: replicate-z-(3, 1)-{'x': (1, 0), 'y': (2, 0)}
```

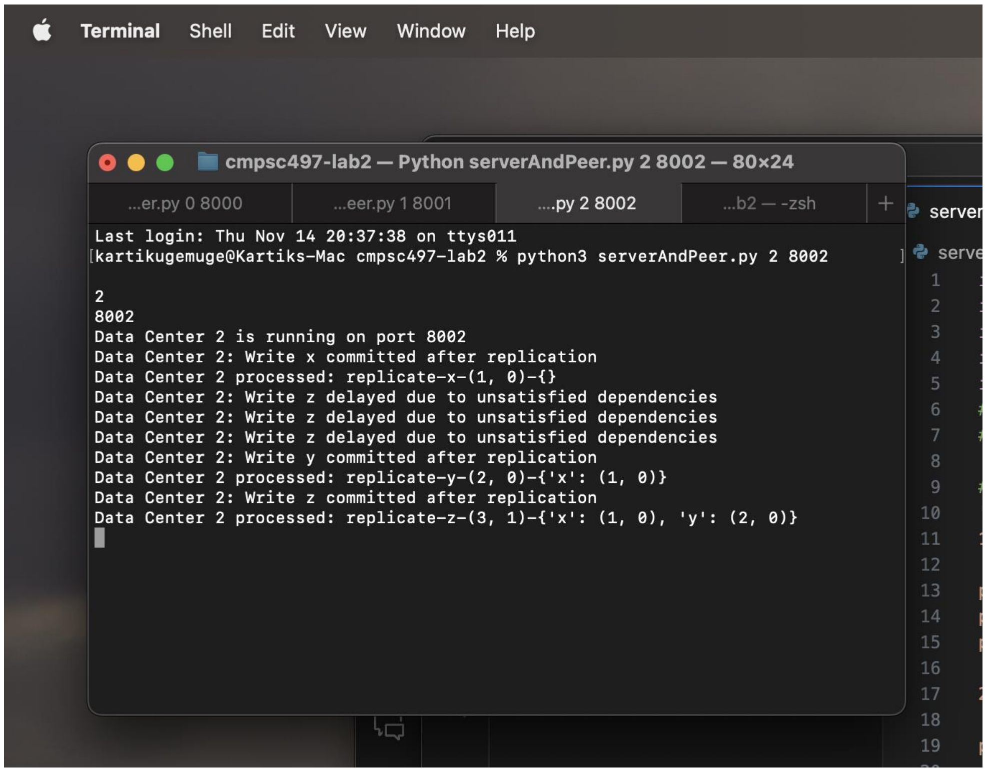
The above display is for Data Center 0, it wrote 2 messages as seen on the client screen and replicated message z.


Terminal Shell Edit View Window Help
cmpsc497-lab2 — Python serverAndPeer.py 1 8001 — 80×24
...er.py 0 8000
...eer.py 1 8001
....py 2 8002
...b2 — -zsh

```
Last login: Thu Nov 14 20:37:30 on ttys010
kartikugemuge@Kartiks-Mac cmpsc497-lab2 % python3 serverAndPeer.py 1 8001
1
8001
Data Center 1 is running on port 8001
Data Center 1: Write x committed after replication
Data Center 1 processed: replicate-x-(1, 0)-{}
Data Center 1: Write y committed after replication
Data Center 1 processed: replicate-y-(2, 0)-{'x': (1, 0)}
```

This above display is for Data Center 1, It received 2 replicate writes after a small delay.

And wrote message z.



```
Last login: Thu Nov 14 20:37:38 on ttys011
kartikugemuge@Kartiks-Mac cmpsc497-lab2 % python3 serverAndPeer.py 2 8002

2
8002
Data Center 2 is running on port 8002
Data Center 2: Write x committed after replication
Data Center 2 processed: replicate-x-(1, 0)-{}
Data Center 2: Write z delayed due to unsatisfied dependencies
Data Center 2: Write z delayed due to unsatisfied dependencies
Data Center 2: Write z delayed due to unsatisfied dependencies
Data Center 2: Write y committed after replication
Data Center 2 processed: replicate-y-(2, 0)-{'x': (1, 0)}
Data Center 2: Write z committed after replication
Data Center 2 processed: replicate-z-(3, 1)-{'x': (1, 0), 'y': (2, 0)}
```

This display is for Data Center 2, Delay from data center 0 to 2 is big and delay from data center 1 to 2 is smaller, so it received, messages in the order: x, z, y, and delayed z until it received y.

The source code has been commented and it includes the commands I used above.