CSE 5306:
DISTRIBUTED SYSTEMS
SUMMER 2023

PROJECT-2

NAME: JAI VENKAT SHANMUKH GADIRAJU
UTA ID: 1002069147

NAME: MAYUKHA THUMIKI
UTA ID: 1002055616

I have neither given nor received unauthorized assistance on this work

Sign: **Date:** 08/06/2023

Jai Venkat Shanmukh Gadiraju
Mayukha Thumiki

1 Project Description

In this project, we have developed a fault-tolerant 2-Phase distributed commit (2PC) protocol. We introduced controlled and random failures to observe how the 2PC protocol handles node crashes. The system comprises one transaction coordinator (TC) and two participant nodes. We used multiple processes to emulate multiple nodes, and each node, including the TC and the participants, implements a time-out mechanism. If no response is received within the time-out period, the node transitions to either the abort or commit state. We evaluated the following possibilities of failures:

1. The **TC** fails before sending the "**prepare**" message.
2. The **TC** fails to receive "**yes**" from a node.
3. The **TC** fails after sending one "**commit**" message to the nodes.
4. A **node** fails after replying "**yes**" to the **TC**.

2 Implementation

The implementation of the 2PC protocol has been written in **Python**. Sockets and threading module was utilized to create and handle connections with multiple nodes concurrently.

We used the Python programming language to implement the 2PC protocol. To simulate numerous nodes as multiple processes, we used the multiprocessing module. Participants were implemented as independent processes, and the TC was implemented separately. We utilized sockets and threading, much as project 1.

The 2PC protocol consists of two steps – prepare and commit. The TC sends a "prepare" message to all the participant nodes. If TC receives a "yes" as a response from all the nodes, it sends out a "commit" message and the transaction is executed. If TC receives a "no" response from any of the participants, it aborts the transaction.

i. Part 1

We evaluated a case of failure where the Transaction Coordinator fails before sending the "prepare" message. The nodes will not receive the "prepare" message until the time-out and will abort. When the TC comes back up and sends the "prepare" message, the nodes will send a "no" message as response.

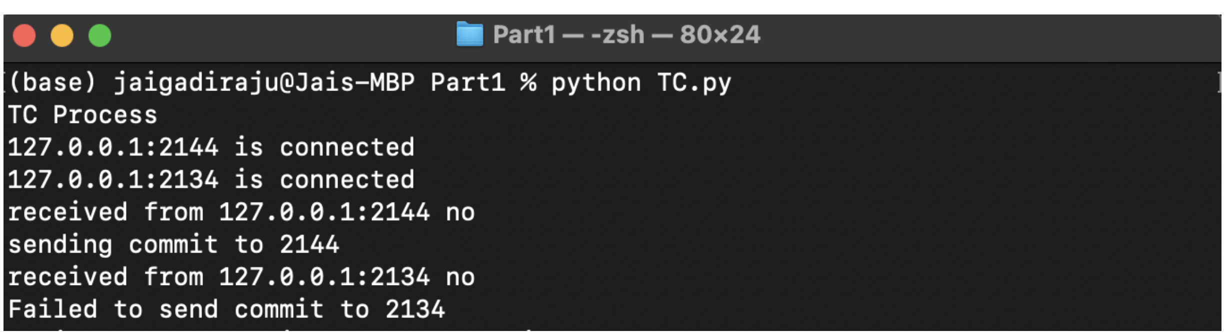

Figure 1: TC

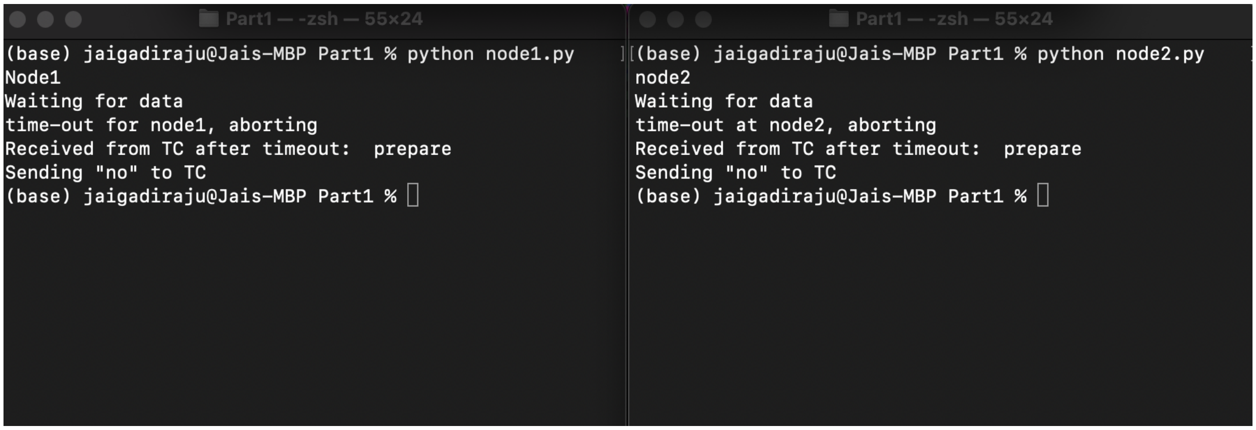

Figure 2: Participant nodes

ii. Part 2

The second case of failure we evaluated was of the TC not receiving "yes" from a node resulting in the transaction being aborted.

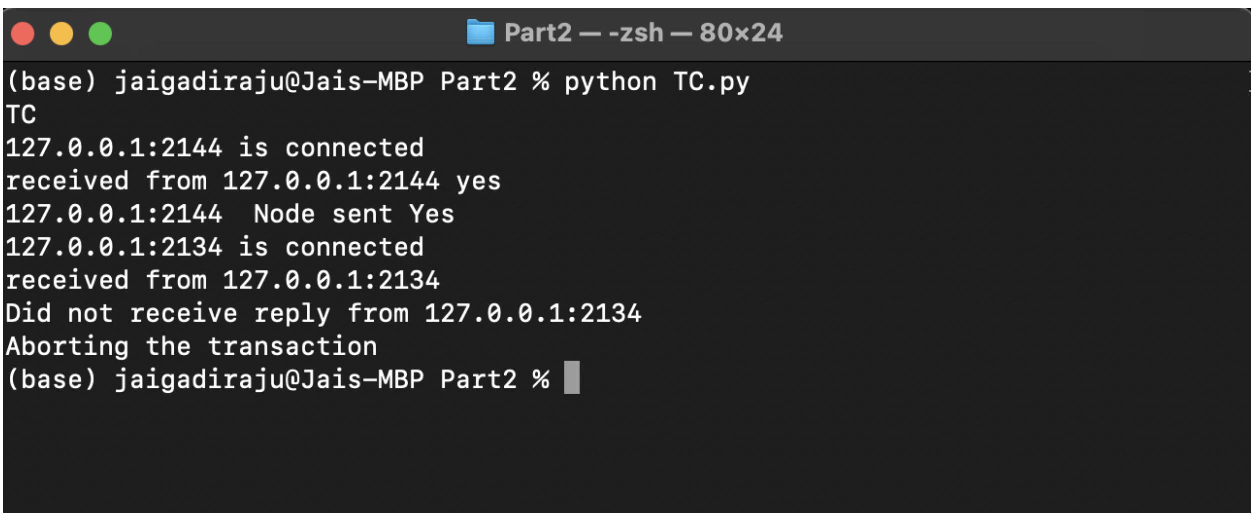

Figure 3: TC

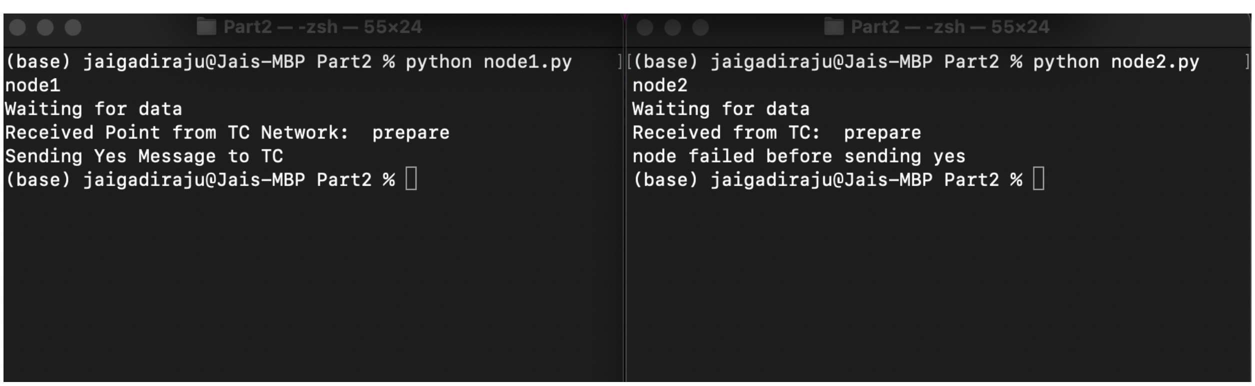

Figure 4: Participant nodes

iii. Part 3

The third case of failure consisted of the TC failing after sending a "commit" message to one but not all nodes. Therefore, it cannot abort, and must send the remaining "commit" messages after coming back up. To restore its state after coming back up, the TC needs to store the transaction information on disk before sending the "commit" message to the nodes.

```
Part3 — -zsh — 80×24
(base) jaigadiraju@Jais-MBP Part3 % python TC.py
TC Process
127.0.0.1:2144 is connected
received from 127.0.0.1:2144 yes
127.0.0.1:2144  Node sent Yes
sending commit to 2144
127.0.0.1:2134 is connected
received from 127.0.0.1:2134 yes
127.0.0.1:2134  Node sent Yes
Failed to send commit to 2134
TC is up and retrying to send commit
sending commit to 2134
>>>>>>>>>>>>>>>>>>>>>>>
transaction states has been going at TC
{2144: ['sent prepare', 'yes', 'sent commit'], 2134: ['sent prepare', 'yes', 'fa
iled to send commit', 'retrying to send commit', 'sent commit']}
(base) jaigadiraju@Jais-MBP Part3 %
```

Figure 5: TC

```
Part3 — -zsh — 55×24
(base) jaigadiraju@Jais-MBP Part3 % python node1.py
Node1
Waiting for data
Received Messagefrom TC:  prepare
Sending yes Message to TC
waiting until get commit from TC
Received from TC:  commit
(base) jaigadiraju@Jais-MBP Part3 %
```

```
Part3 — -zsh — 55×24
(base) jaigadiraju@Jais-MBP Part3 % python node2.py
node2
Waiting for data
Received Message from TC :  prepare
Sending yes From Node 2 to TC Side
waiting until gets from commit from TC
Received commit from TC:  commit
(base) jaigadiraju@Jais-MBP Part3 %
```

Figure 6: Participant nodes

iv. Part 4

Finally, we evaluated the case where a node fails after sending a "yes" response but before receiving the commit information for the transaction from the TC. Similar to TC, node must also store the transaction information before responding with "yes" to be able to store its state in case of failure. After the node is back up again, it fetches the commit information from the TC.

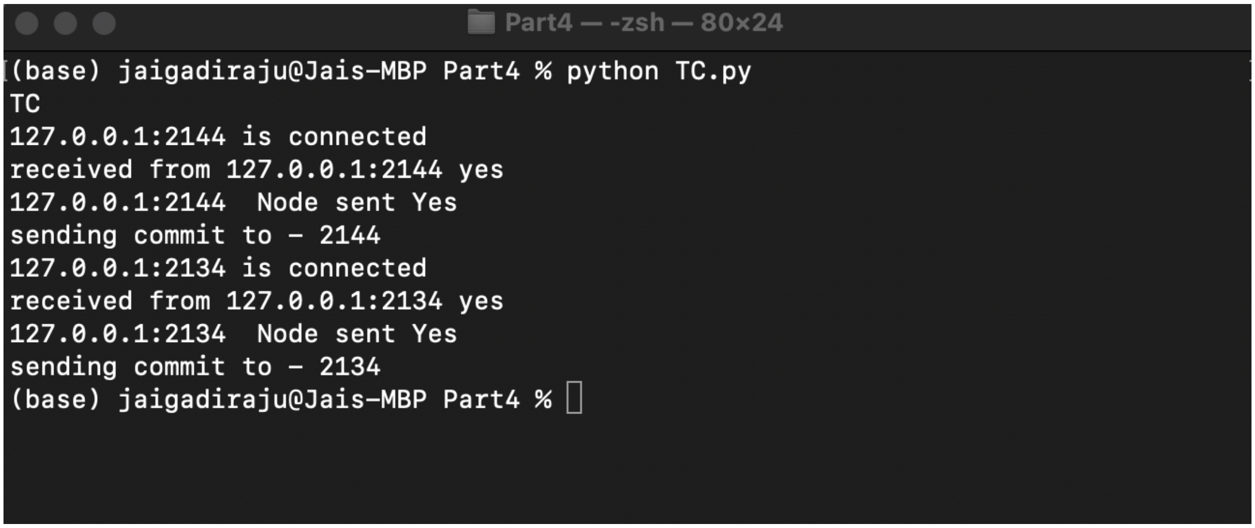

Figure 7: TC

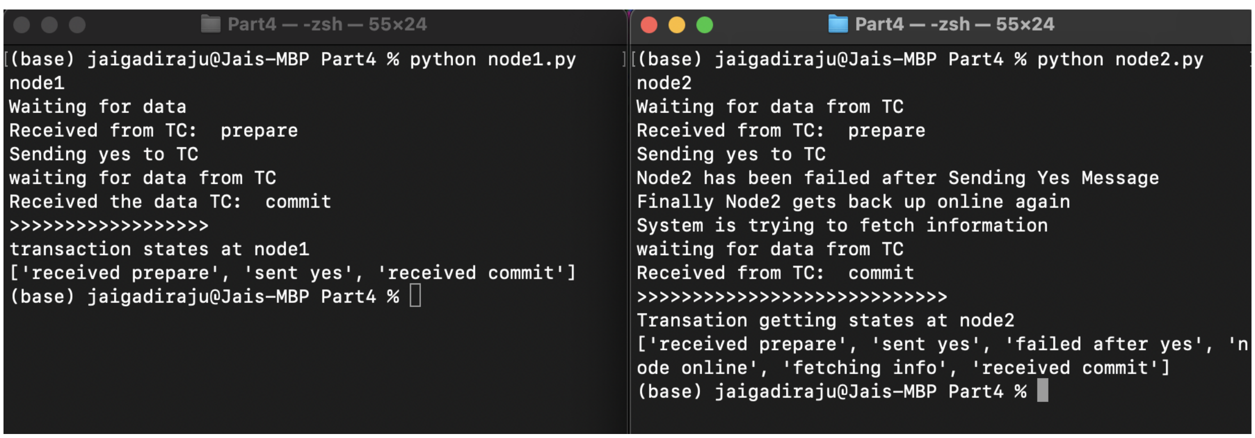

Figure 8: Participant nodes

3 What we learned

During this project, we gained valuable insights into the intricacies of building distributed systems using the 2-Phase Commit (2PC) protocol. We explored essential aspects like synchronization, fault tolerance, scalability, performance, and security within distributed systems. Additionally, we placed a strong emphasis on developing a reliable recovery mechanism to address various failure scenarios. Overall, this hands-on experience deepened our understanding of distributed system design and implementation, serving as a solid foundation for future endeavors in this domain.

4 Issues encountered

The implementation of the 2PC protocol presented various challenges:

i. **Fault tolerance**: ensuring fault tolerance was the major challenge.
ii. **Recovery**: implementing a reliable state recovery system in case of failures needed considerable attention.
iii. **Synchronization**: To avoid inconsistencies, loss of data and delays in transaction processing, synchronization was essential.