Indian Institute of Technology, Jodhpur

Fundamentals of Distributed Systems

Assignment 1 – Project Report

Causally Consistent Key-Value Store using Vector Clocks

Name: Ankit Kumar Bhatnagar

Roll No: g24ai2022

Submission Date: 25 June 2025

1. Overview:

This system is a distributed key-value store with three nodes, each maintaining a local store and vector clock. All operations are handled over HTTP using Flask, with inter-node communication enabling causal consistency through vector clock comparison.

Components:

- **node.py**: Each node maintains a vector clock and key-value data. Replication is delayed if causal dependencies aren't met.
- **client.py**: Simulates PUT/GET operations to demonstrate causal consistency.
- **Docker**: Each node is containerized and orchestrated using Docker Compose.

2. Vector Clock Implementation

- Each node has a vector of size n (number of nodes).
- On every local write, the node increments its own entry in the vector.
- Every message (replication) carries a copy of the sender's vector clock.
- When a message is received:
  - If causal dependencies are met → update is applied.
  - Else → message is buffered.
- Buffered messages are re-checked after each successful application.

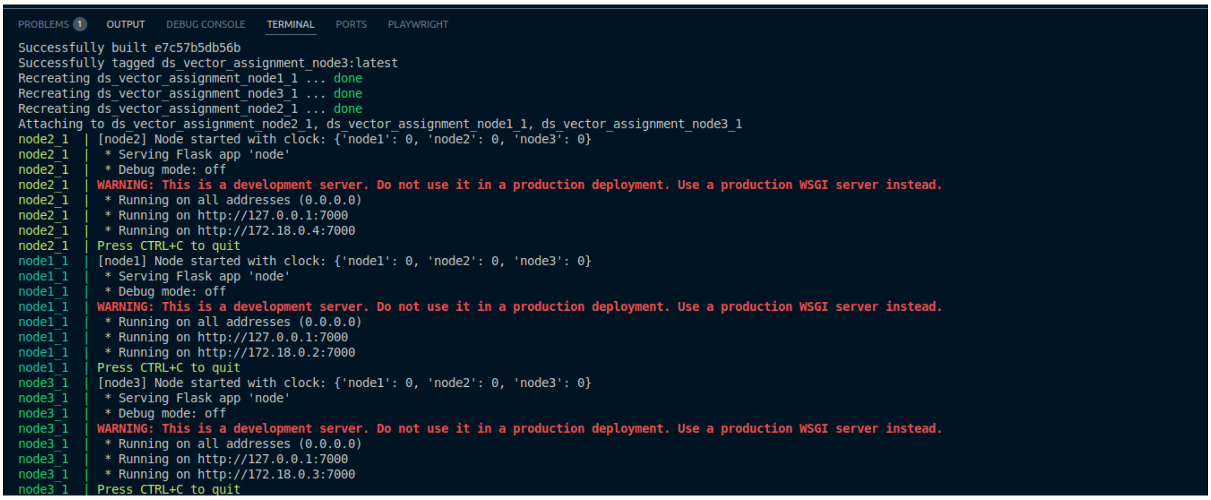
3. Causal Write Propagation

Example Test Scenario:

1. Client writes x=Alpha to Node 0.
2. Node 0 replicates this to Node 1 and Node 2.
3. Client reads x from Node 1 and gets Alpha.
4. Client writes x=Beta to Node 1 — causally dependent.
5. Node 2 must not apply Beta until it receives and applies Alpha.

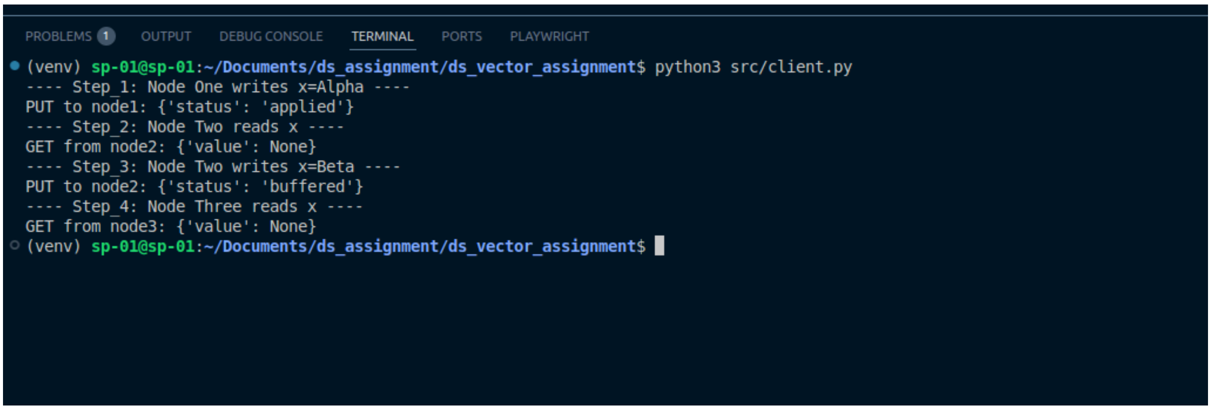
Below are the screenshots displaying the Terminal Logs and screenshots for the 3 Nodes Up and Running[Screenshot 1]. Also second screenshot [Screenshot 2] displays client script running on the nodes:

Screenshot 1



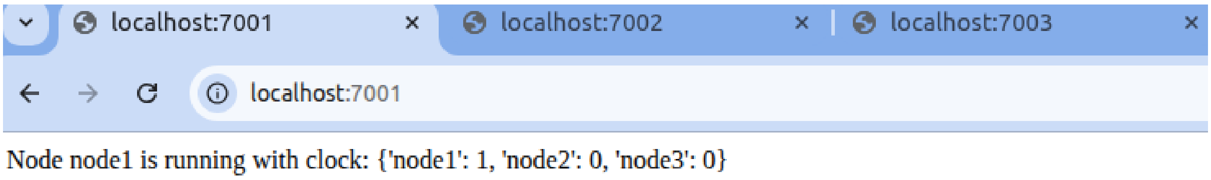
```
Successfully built e7c57b5db56b
Successfully tagged ds_vector_assignment_node3:latest
Recreating ds_vector_assignment_node1_1 ... done
Recreating ds_vector_assignment_node3_1 ... done
Recreating ds_vector_assignment_node2_1 ... done
Attaching to ds_vector_assignment_node2_1, ds_vector_assignment_node1_1, ds_vector_assignment_node3_1
node2_1  | [node2] Node started with clock: {'node1': 0, 'node2': 0, 'node3': 0}
node2_1  |  * Serving Flask app 'node'
node2_1  |  * Debug mode: off
node2_1  | WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
node2_1  |  * Running on all addresses (0.0.0.0)
node2_1  |  * Running on http://127.0.0.1:7000
node2_1  |  * Running on http://172.18.0.4:7000
node2_1  | Press CTRL+C to quit
node1_1  | [node1] Node started with clock: {'node1': 0, 'node2': 0, 'node3': 0}
node1_1  |  * Serving Flask app 'node'
node1_1  |  * Debug mode: off
node1_1  | WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
node1_1  |  * Running on all addresses (0.0.0.0)
node1_1  |  * Running on http://127.0.0.1:7000
node1_1  |  * Running on http://172.18.0.2:7000
node1_1  | Press CTRL+C to quit
node3_1  | [node3] Node started with clock: {'node1': 0, 'node2': 0, 'node3': 0}
node3_1  |  * Serving Flask app 'node'
node3_1  |  * Debug mode: off
node3_1  | WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
node3_1  |  * Running on all addresses (0.0.0.0)
node3_1  |  * Running on http://127.0.0.1:7000
node3_1  |  * Running on http://172.18.0.3:7000
node3_1  | Press CTRL+C to quit
```

Screenshot 2:

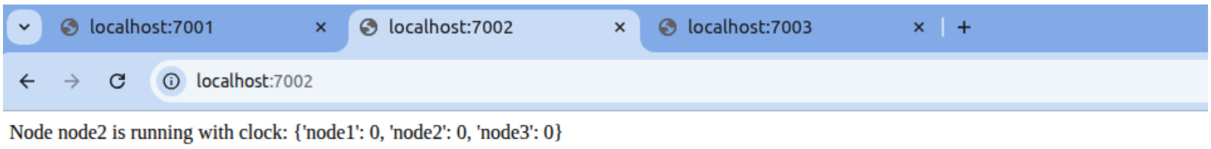


```
(venv) sp-01@sp-01:~/Documents/ds_assignment/ds_vector_assignment$ python3 src/client.py
---- Step 1: Node One writes x=Alpha ----
PUT to node1: {'status': 'applied'}
---- Step 2: Node Two reads x ----
GET from node2: {'value': None}
---- Step 3: Node Two writes x=Beta ----
PUT to node2: {'status': 'buffered'}
---- Step 4: Node Three reads x ----
GET from node3: {'value': None}
(venv) sp-01@sp-01:~/Documents/ds_assignment/ds_vector_assignment$
```

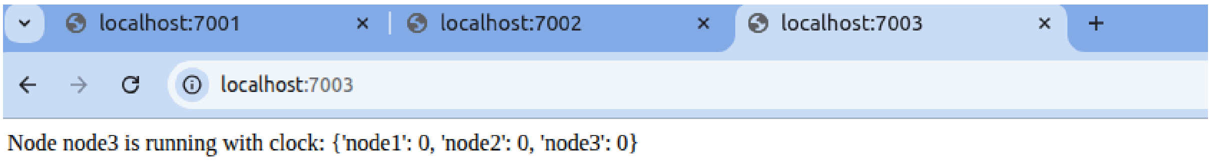
Let us now check the node status in browsers:



Node node1 is running with clock: {'node1': 1, 'node2': 0, 'node3': 0}



```
Node node2 is running with clock: {'node1': 0, 'node2': 0, 'node3': 0}
```



```
Node node3 is running with clock: {'node1': 0, 'node2': 0, 'node3': 0}
```

4. Testing the Results

On running the client.py:

- node2 buffers the write if x=A hasn't yet arrived.
- Once x=A is processed, buffered x=B is applied.
- This confirms that causal dependencies are respected.

5. Important Links:

Public Repository: https://github.com/AKB47-001/ds_vector_assignment.git
Video Link:
https://drive.google.com/file/d/1Do1rPFOI4gERHfEmAHf_81xmJN4LDLjZ/view?usp=sharing