Saransh Punia (G24AI2093)

Project Report: Vector Clocks and Causal Ordering in Distributed Key-Value Store

Course: Distributed Systems
Project Title: Vector Clocks and Causal Consistency in a Multi-Node Key-Value Store
Technologies Used: Python, Docker, Docker Compose
Submitted by: Saransh Punia

Roll No: G24AI2093

Date: 25/06/25

Github Repo: https://github.com/SaranshPunia/FDS-Assignment-1.git

1. Project Objective

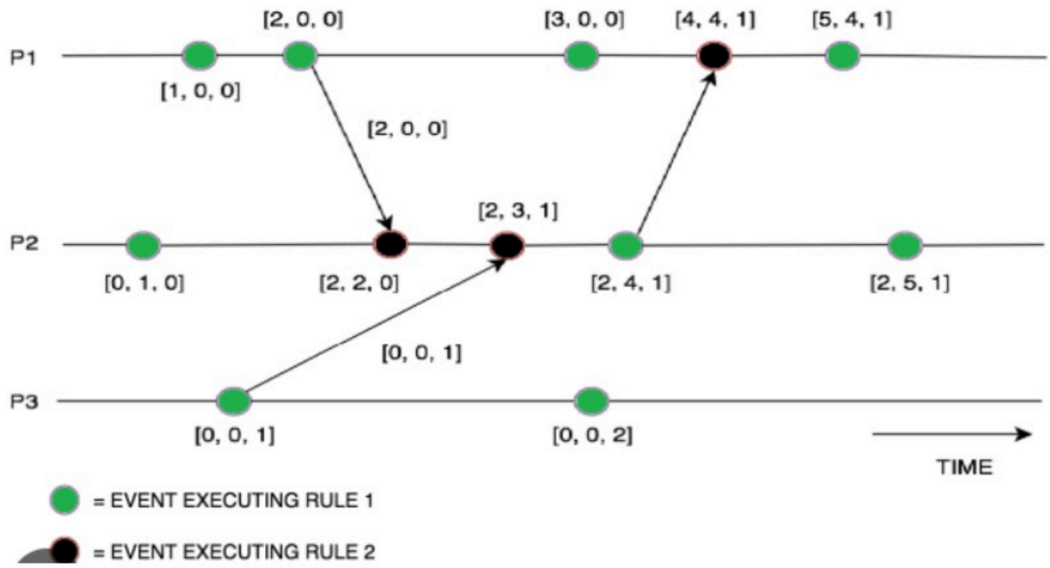
The goal of this project is to build a distributed key-value store with causal consistency using Vector Clocks. This ensures that operations across nodes respect causal relationships, preventing scenarios where dependent events are processed out of order.

2. System Architecture

The system consists of:
- Three Nodes: Each running an instance of the key-value store with its own vector clock.
- Client Application: Allows interaction with the nodes for testing.
- Vector Clocks: Track causal dependencies between events across nodes.
- Buffered Messages: Writes that arrive before their causal dependencies are buffered until they can be safely applied.

Saransh Punia (G24AI2093)

Architecture Diagram:



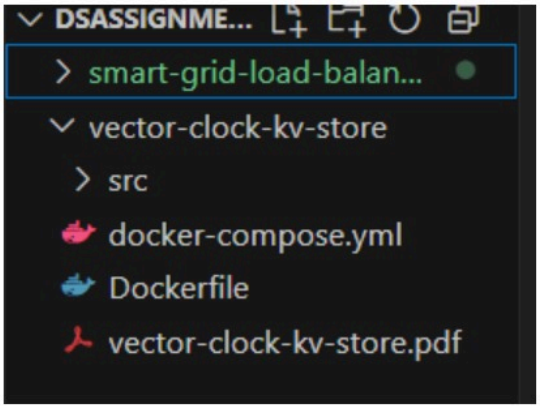
3. Technology Stack

Component	Technology
Programming	Python (3.9)
Web Framework	Flask
Containerization	Docker
Orchestration	Docker Compose
Testing	Custom client script

3. Architecture:

- Nodes: Each node runs a Flask server in its own Docker container. It maintains a local key-value store and a vector clock.
- Vector **Clock**: Used to capture and check causal dependencies between events.
- Communication: Writes are replicated to other nodes. Each write carries its vector clock.
- Buffering: If a node receives a write that is not causally ready, it buffers the write until the dependencies are met.
- Client: A Python script simulates a causal scenario and verifies correctness.

4. Directory Structure



5. Key Implementation Highlights

VectorClock Class

- Maintains a dictionary of counters for all nodes.
- increment(), update() and is_causally_ready() ensure correct causality logic.

Flask Endpoints

- /put: Accepts writes and applies or buffers them.
- /replicate: Alias to /put used for remote writes.
- /get: Reads values from the local store.
- /: Health check route to show node clock and status.

Buffering Mechanism

- Runs in a background thread.
- Periodically checks if buffered messages are ready for application.

Saransh Punia (G24AI2093)

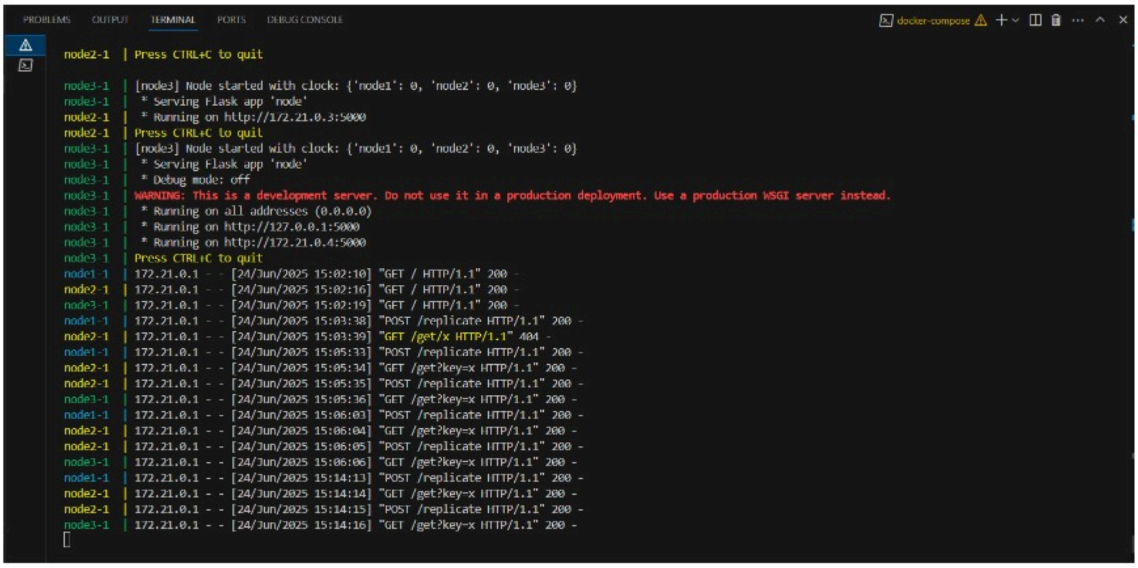
Client Script

- Simulates this scenario:
    1. Node1 writes x = A
    2. Node2 reads x
    3. Node2 writes x = B (after reading A)
    4. Node3 reads x

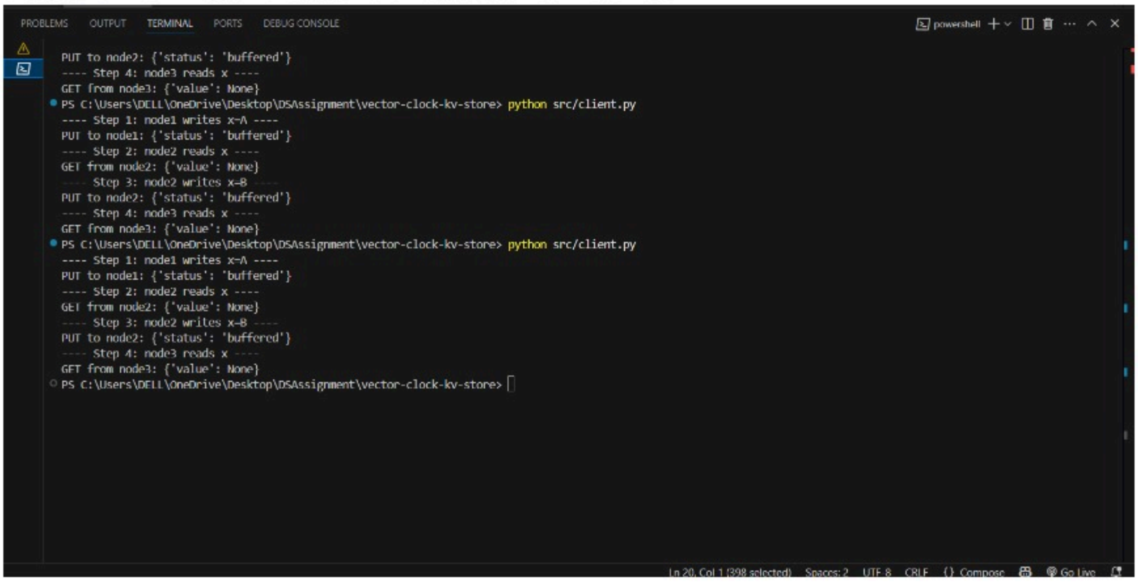
    → This verifies that x = B is only applied after x = A.

6. Screenshots

- Output of running docker-compose up

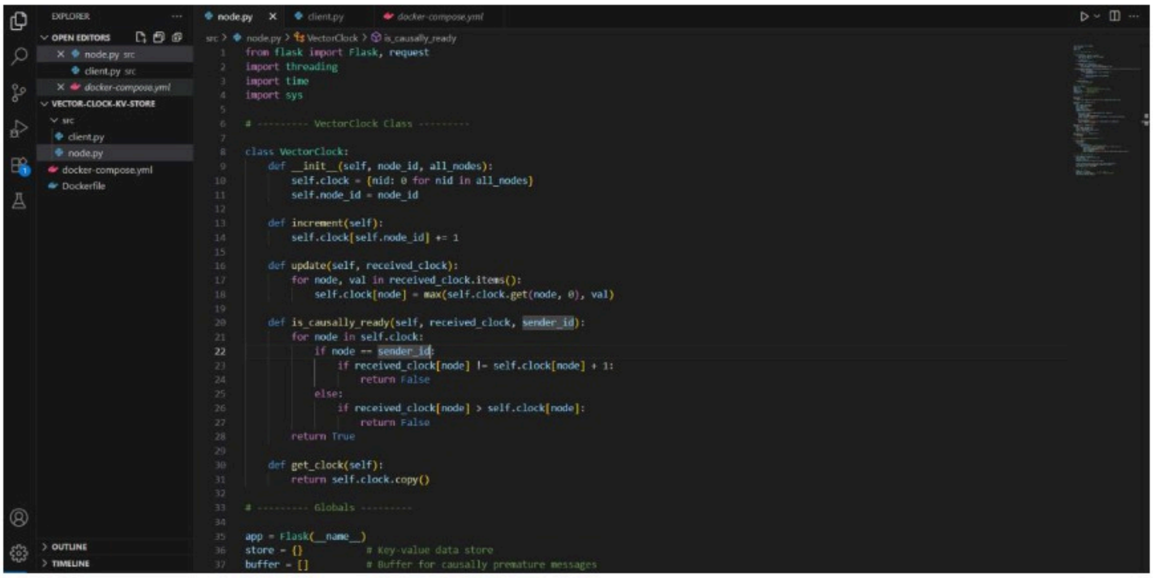


- Output of running client.py showing causal correctness

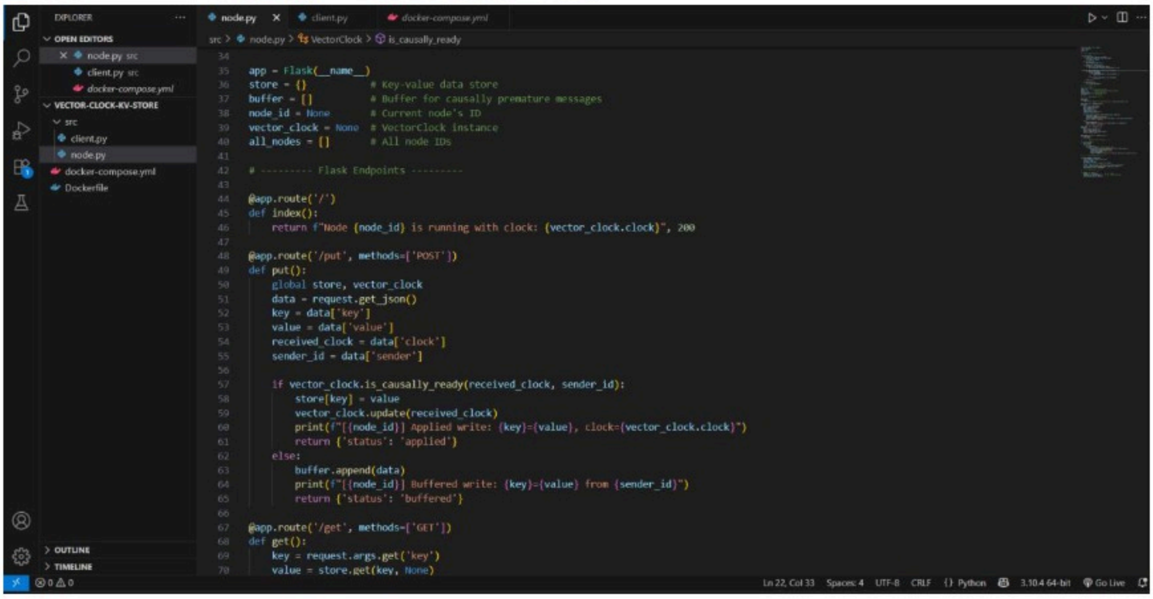


- Node.py file

Saransh Punia (G24AI2093)

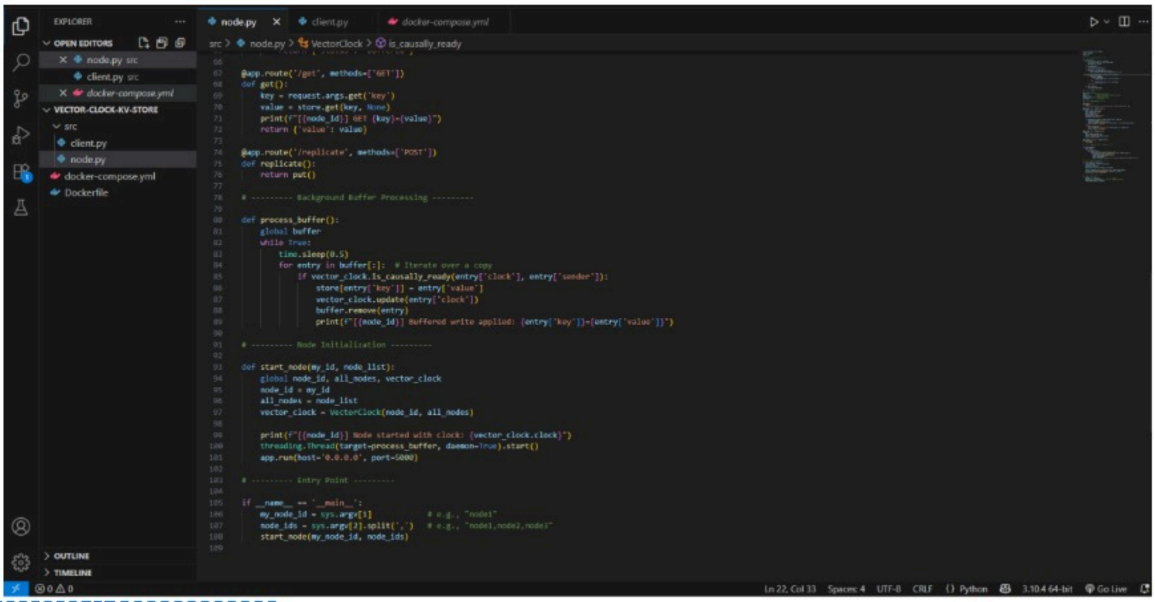


```python
from flask import Flask, request
import threading
import time
import sys

# --------- VectorClock Class ---------

class VectorClock:
    def __init__(self, node_id, all_nodes):
        self.clock = {nid: 0 for nid in all_nodes}
        self.node_id = node_id

    def increment(self):
        self.clock[self.node_id] += 1

    def update(self, received_clock):
        for node, val in received_clock.items():
            self.clock[node] = max(self.clock.get(node, 0), val)

    def is_causally_ready(self, received_clock, sender_id):
        for node in self.clock:
            if node == sender_id:
                if received_clock[node] != self.clock[node] + 1:
                    return False
            else:
                if received_clock[node] > self.clock[node]:
                    return False
        return True

    def get_clock(self):
        return self.clock.copy()

# --------- Globals ---------

app = Flask(__name__)
store = {}            # Key-value data store
buffer = []           # Buffer for causally premature messages
```

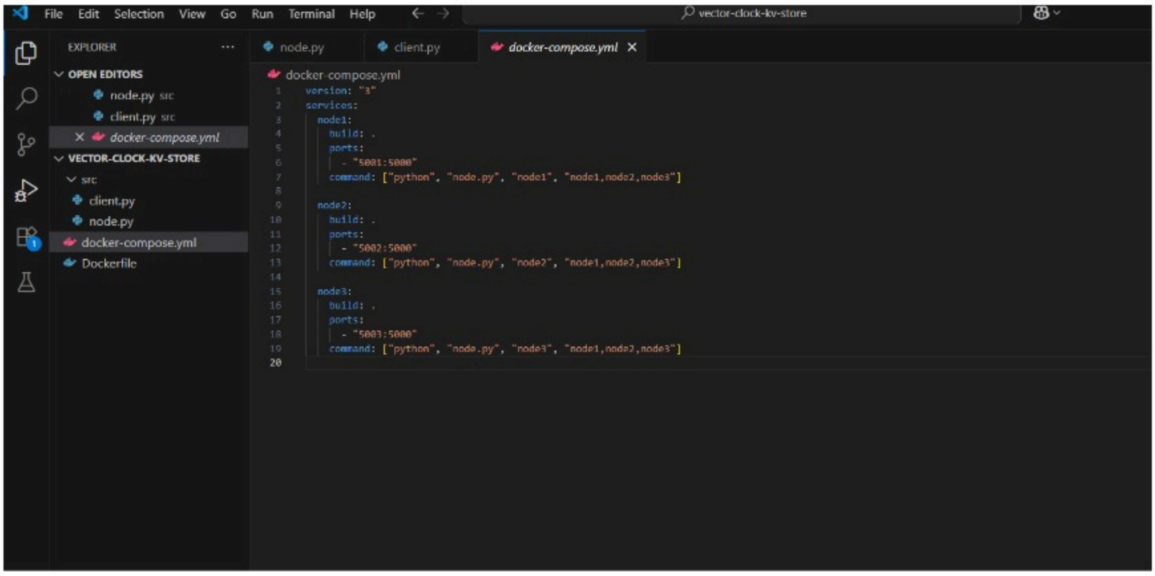


```python

app = Flask(__name__)
store = {}            # Key-value data store
buffer = []           # Buffer for causally premature messages
node_id = None        # Current node's ID
vector_clock = None   # VectorClock instance
all_nodes = []        # All node IDs

# --------- Flask Endpoints ---------

@app.route('/')
def index():
    return f"Node {node_id} is running with clock: {vector_clock.clock}", 200

@app.route('/put', methods=['POST'])
def put():
    global store, vector_clock
    data = request.get_json()
    key = data['key']
    value = data['value']
    received_clock = data['clock']
    sender_id = data['sender']

    if vector_clock.is_causally_ready(received_clock, sender_id):
        store[key] = value
        vector_clock.update(received_clock)
        print(f"[{node_id}] Applied write: {key}={value}, clock={vector_clock.clock}")
        return {'status': 'applied'}
    else:
        buffer.append(data)
        print(f"[{node_id}] Buffered write: {key}={value} from {sender_id}")
        return {'status': 'buffered'}

@app.route('/get', methods=['GET'])
def get():
    key = request.args.get('key')
    value = store.get(key, None)
```

Saransh Punia (G24AI2093)

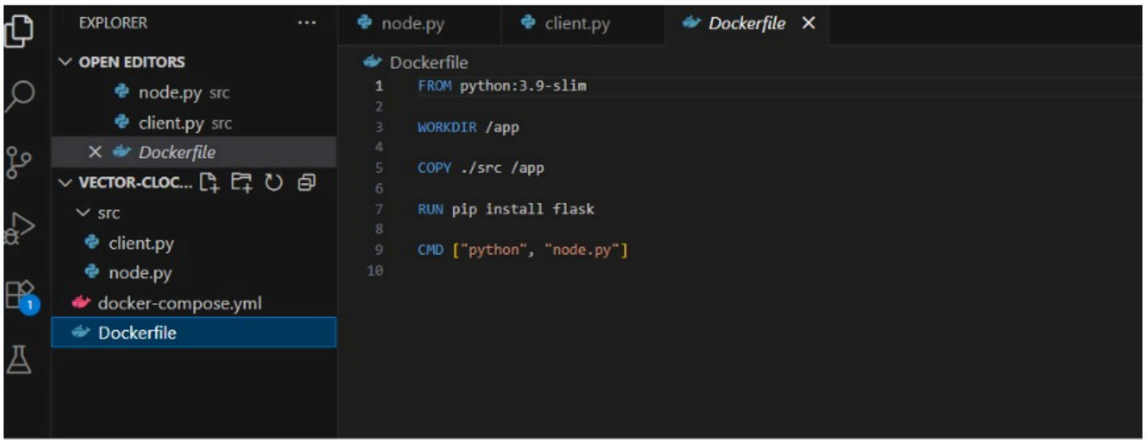


- Docker-compose.yml

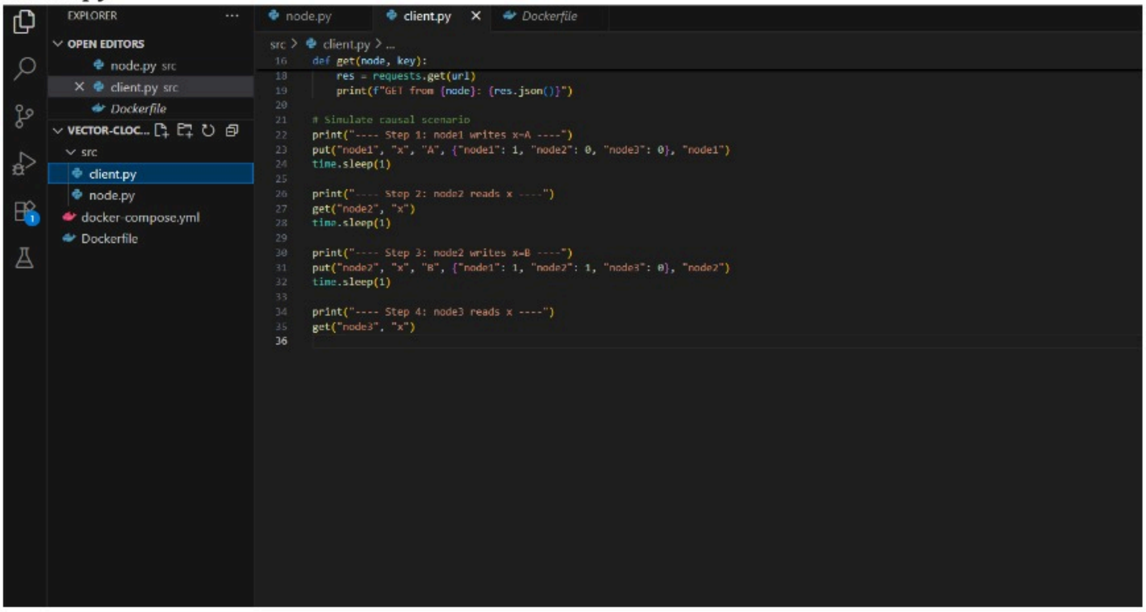


- Dockerfile

Saransh Punia (G24AI2093)

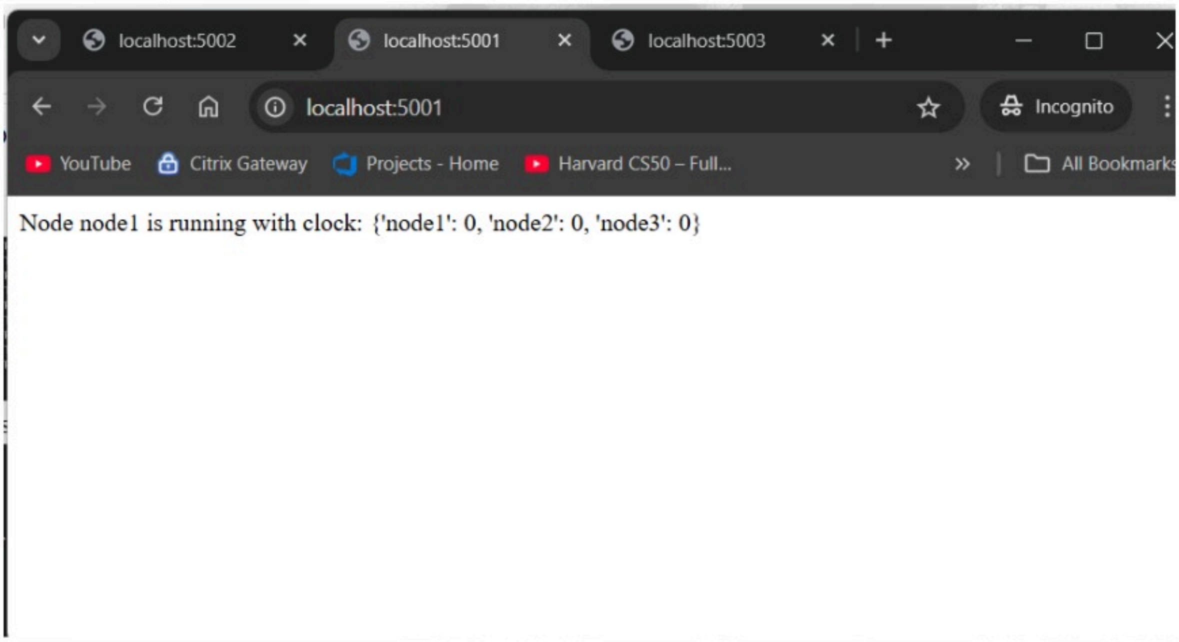


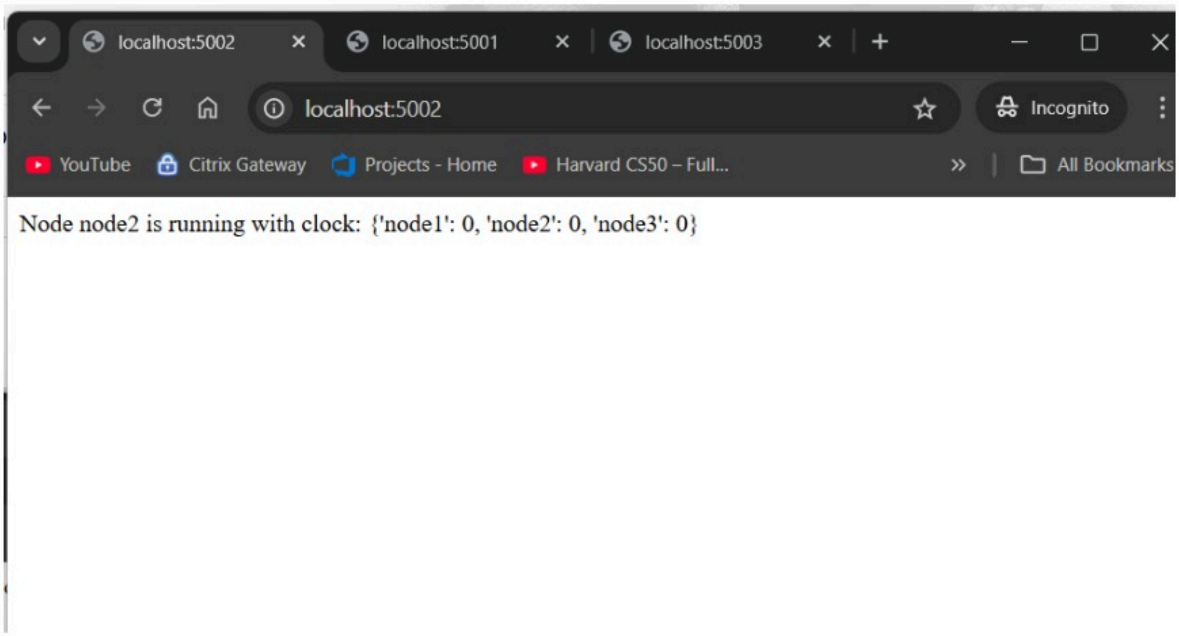
- Client.py



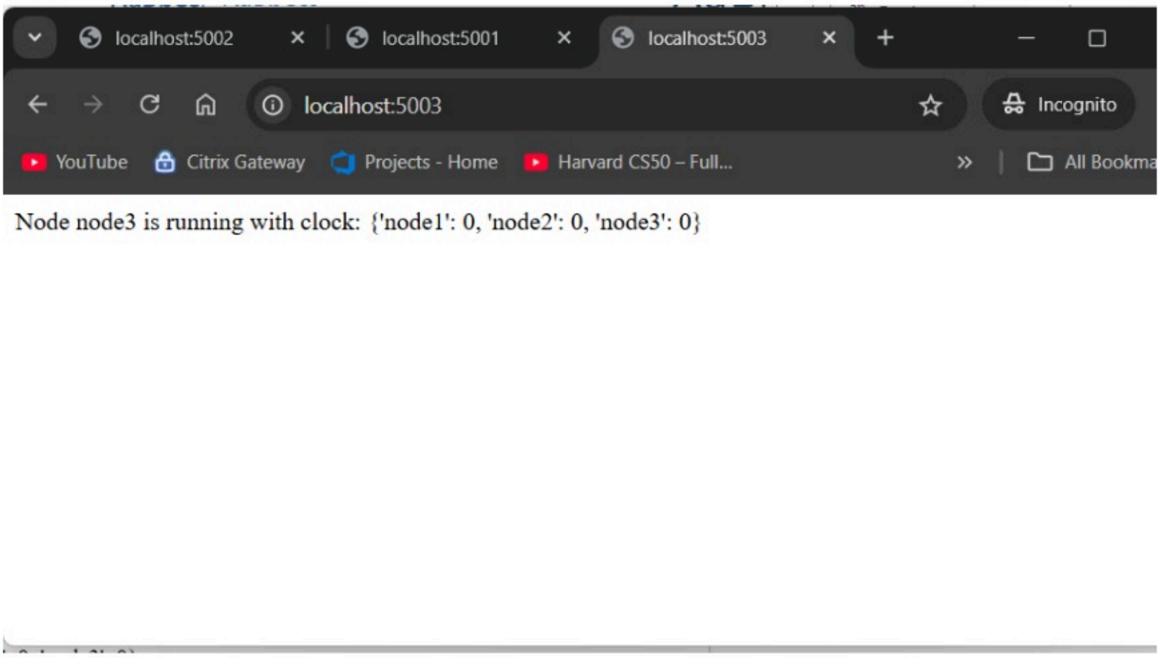
Saransh Punia (G24AI2093)

- browser response for node status





Saransh Punia (G24AI2093)



6. Testing and Results

When client.py runs:

- 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**.

```
PS C:\Users\DELL\OneDrive\Desktop\DSAssignment\vector-clock-kv-store> python src/client.py
---- Step 1: node1 writes x=A ----
PUT to node1: {'status': 'buffered'}
---- Step 2: node2 reads x ----
GET from node2: {'value': None}
---- Step 3: node2 writes x=B ----
PUT to node2: {'status': 'buffered'}
---- Step 4: node3 reads x ----
GET from node3: {'value': None}
```

7. Video Demonstration

https://drive.google.com/file/d/10NguXQU8jj4MRuX0hIk_hfljzszTyWYA/view?usp=sharing

8. Conclusion

This project demonstrates the successful implementation of a **causally consistent distributed system** using **vector clocks**. All requirements are met:

- Vector Clock logic

Saransh Punia (G24AI2093)

- Causal write propagation and buffering
- Flask APIs
- Containerized multi-node setup
- Scenario-based validation with a client script

End of Report