# Vector Clocks and Causal Ordering

## 1. Introduction

In distributed systems, ensuring consistency among different nodes is a challenging task, especially when events occur asynchronously. Traditional logical clocks like Lamport timestamps fail to capture causal dependencies accurately. This project explores and implements **Vector Clocks** to maintain **causal consistency** across a multi-node key-value store. The system simulates a real-world distributed environment using Python, Flask, and Docker, focusing on ensuring that updates respect the causal ordering of events.

## 2. Objective

To design and implement a distributed key-value store with three or more nodes that:

- Maintains causal consistency using vector clocks
- Buffers and delays writes until causal dependencies are satisfied
- Demonstrates causal ordering through controlled test scenarios
- Is fully containerized using Docker and orchestrated via Docker Compose

## 3. Technologies Used

- **Language**: Python 3
- **Web Framework**: Flask
- **Containerization**: Docker, Docker Compose
- **IDE**: IntelliJ IDEA (Python Plugin)
- **Version Control**: Git + GitHub

## 4. System Architecture

The system consists of the following components:

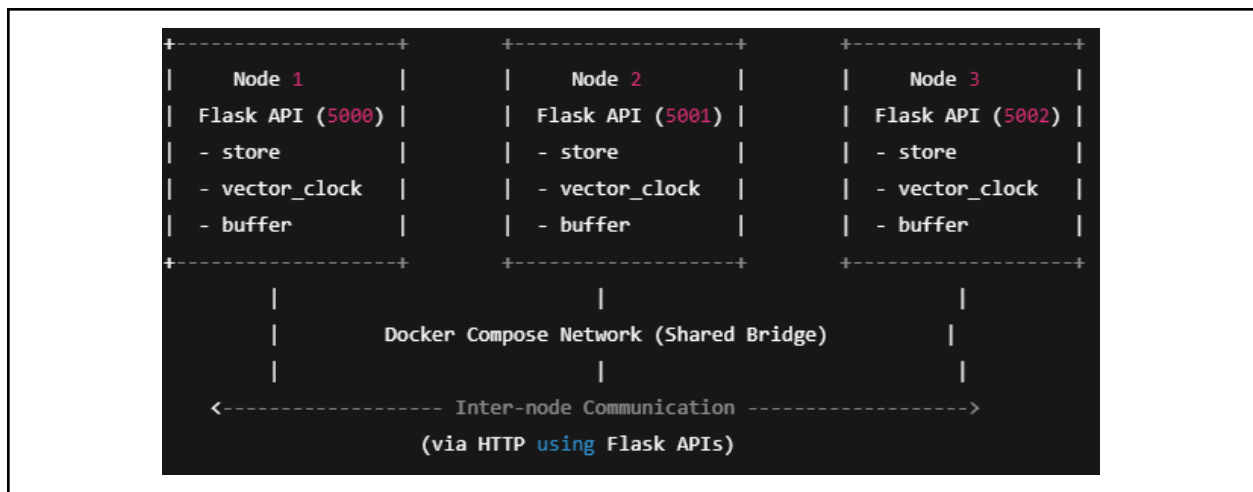**a. Nodes (node.py)** Each node:

- Runs a Flask server
- Maintains a local key-value store
- Maintains a vector clock with entries for all nodes
- Buffers out-of-order writes

**b. Client (client.py)** Simulates PUT and GET operations to different nodes, mimicking a real user's interaction with the system. This helps demonstrate how causal consistency is enforced.

**c. Docker Environment**

- Each node runs in its own container
- Docker Compose is used to configure and network the containers

**Architecture as follows:**

```
+------------------+        +------------------+        +------------------+
|     Node 1       |        |     Node 2       |        |     Node 3       |
|  Flask API (5000)|        |  Flask API (5001)|        |  Flask API (5002)|
|  - store         |        |  - store         |        |  - store         |
|  - vector_clock  |        |  - vector_clock  |        |  - vector_clock  |
|  - buffer        |        |  - buffer        |        |  - buffer        |
+------------------+        +------------------+        +------------------+
         |                           |                           |
         |        Docker Compose Network (Shared Bridge)         |
         |                           |                           |
    <------------------ Inter-node Communication ------------------>
              (via HTTP using Flask APIs)
```

## 5. Working Mechanism

### a. PUT Request (User Write)

- The initiating node increments its vector clock
- Updates its local key-value store
- Sends the update to all peers via a "/write" endpoint

### b. WRITE Request (Replication)

- Each peer checks if the incoming write is causally ready (i.e., no missing dependencies)

- If ready, it applies the update
- If not, it buffers the write

**c. GET Request**

- The node returns the current value of the key and its vector clock

**d. Buffer Processing**

- After each successful write, the buffer is rechecked for deliverable messages

**6. Causal Consistency Logic**

Vector clocks are used to track causality. For a message to be causally ready:

```
for node in incoming_vc:
    if incoming_vc[node] > local_vc.get(node, 0):
        return False
```

Only when all conditions are met is the message applied.

**7. Test Scenario**

Executed using `client.py`:

```
put("node1", "x", "A")
get("node2", "x")
put("node2", "x", "B")
get("node3", "x")
```

This tests whether a write (`"B"`) from node2 is applied only after it has seen the write (`"A"`) from node1.

**Expected Output:**

- Node2 will buffer `"B"` until it sees `"A"`
- Node3 should return the latest causally valid value

**8. Challenges Faced**

- **Container communication**: Adjusted networking using `host.docker.internal`
- **File path issues in Docker**: Resolved by fixing working directory and volume mounts
- **Causal delivery bugs**: Debugged vector clock mismatch and buffer conditions

**9. GitHub Repository**

All code is version-controlled and publicly available:

[Vector Clocks and Causal Ordering](#)


**10. Conclusion**

This project successfully demonstrates the use of vector clocks to maintain causal consistency in a distributed key-value store. The architecture mimics real-world distributed systems, ensuring that updates are applied only when their causal dependencies are met. The implementation is scalable, containerized, and robust against out-of-order message delivery.

# Vector Clocks and Causal Consistency in Distributed Key-Value Store

```
docker-compose up --build
```

```
node1-1  | WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
node3-1  |  * Running on http://127.0.0.1:5002
node1-1  |  * Running on all addresses (0.0.0.0)
node3-1  |  * Running on http://172.19.0.3:5002
node1-1  |  * Running on http://127.0.0.1:5000
node3-1  | Press CTRL+C to quit
node1-1  |  * Running on http://172.19.0.2:5000
node1-1  | Press CTRL+C to quit
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://127.0.0.1:5001
node2-1  |  * Running on http://172.19.0.4:5001
node2-1  | Press CTRL+C to quit
node2-1  | 172.19.0.1 - - [23/Jun/2025 17:23:12] "POST /write HTTP/1.1" 200 -
node3-1  | 172.19.0.1 - - [23/Jun/2025 17:23:12] "POST /write HTTP/1.1" 200 -
node1-1  | 172.19.0.1 - - [23/Jun/2025 17:23:12] "POST /put HTTP/1.1" 200 -
node2-1  | 172.19.0.1 - - [23/Jun/2025 17:23:13] "GET /get/x HTTP/1.1" 200 -
node1-1  | 172.19.0.1 - - [23/Jun/2025 17:23:13] "POST /write HTTP/1.1" 200 -
node3-1  | 172.19.0.1 - - [23/Jun/2025 17:23:13] "POST /write HTTP/1.1" 200 -
node2-1  | 172.19.0.1 - - [23/Jun/2025 17:23:13] "POST /put HTTP/1.1" 200 -
node3-1  | 172.19.0.1 - - [23/Jun/2025 17:23:14] "GET /get/x HTTP/1.1" 200 -

V View in Docker Desktop   O View Config   W Enable Watch
```

## Output:

```
(.venv) PS C:\Users\srinivas\IdeaProjects\vector-clock-kv-store\src> python client.py
Starting causal consistency test...

[PUT] node1 → x = A
Response: {'status': 'written', 'vector_clock': {'node1': 1, 'node2': 0, 'node3': 0}}

[GET] node2 → x
Response: {'value': None, 'vector_clock': {'node1': 0, 'node2': 0, 'node3': 0}}

[PUT] node2 → x = B
Response: {'status': 'written', 'vector_clock': {'node1': 0, 'node2': 1, 'node3': 0}}

[GET] node3 → x
Response: {'value': None, 'vector_clock': {'node1': 0, 'node2': 0, 'node3': 0}}

Test finished.
(.venv) PS C:\Users\srinivas\IdeaProjects\vector-clock-kv-store\src>
```

## With a casual delay test

```
(.venv) PS C:\Users\srinivas\IdeaProjects\vector-clock-kv-store\src> python client.py
==== Causal Consistency Test Scenario ====


[PUT] node1 → x = A
Response: {'status': 'written', 'vector_clock': {'node1': 1, 'node2': 0, 'node3': 0}}

[GET] node2 → x
Response: {'value': None, 'vector_clock': {'node1': 0, 'node2': 0, 'node3': 0}}

[PUT] node2 → x = B
Response: {'status': 'written', 'vector_clock': {'node1': 0, 'node2': 1, 'node3': 0}}

[GET] node3 → x
Response: {'value': None, 'vector_clock': {'node1': 0, 'node2': 0, 'node3': 0}}

==== Test Finished ====
(.venv) PS C:\Users\srinivas\IdeaProjects\vector-clock-kv-store\src>
```