

Indian Institute of Technology-Jodhpur

Fundamentals of Distributed Systems

Vector Clocks and Causal Ordering
Bhagyalaxmi Pandiyan (G24AI2088)

Github - <https://github.com/BhagyalaxmiPandiyan/vector-clock-kv-store-G24AI2088->

1. Introduction

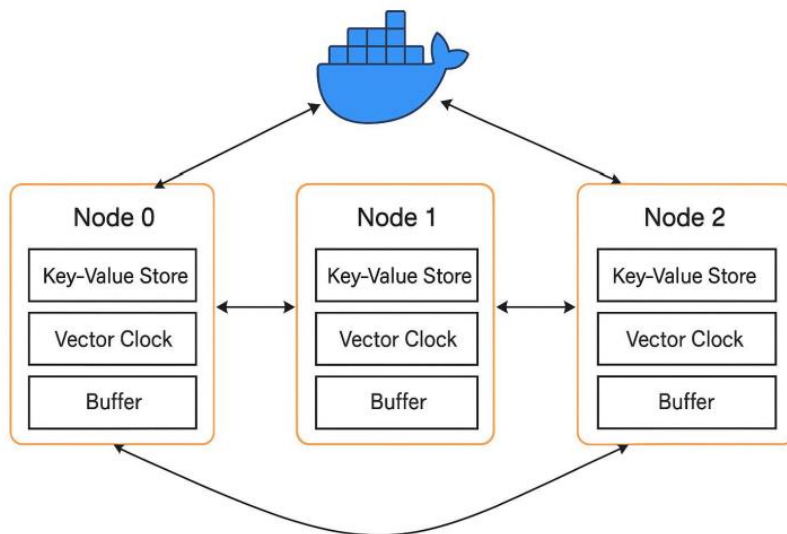
In distributed systems, ensuring that events are processed in a consistent and causally correct order is critical for correctness and predictability. Traditional Lamport timestamps help establish a partial ordering of events but are insufficient to detect causality. This project explores the use of **Vector Clocks**, which provide a mechanism to capture causality more effectively.

We implement a **multi-node key-value store** that uses Vector Clocks to ensure **causal consistency**. Each node in the system maintains its own local state and vector clock. By comparing vector clocks, nodes decide when to process or delay updates, preserving the causality of operations.

2. Objectives

- To understand and implement Vector Clocks in a distributed environment.
 - To build a Python-based distributed key-value store that preserves causal consistency.
 - To enable causal delivery of messages even in the presence of out-of-order communication.
 - To containerize the solution using Docker and Docker Compose.
 - To test and verify causal consistency through simulated scenarios.
-

3. System Architecture



Components:

1. **Nodes (3 or more):**
 - o Each node is an independent service hosting a key-value store.
 - o Each node maintains a vector clock to track causality.
2. **Client:**
 - o Sends requests to one or more nodes for testing causal relationships.
3. **Docker Environment:**
 - o Nodes and client are containerized.
 - o Docker Compose manages inter-container communication.

Workflow:

- A client sends a PUT request to update a key on a node.
 - The node increments its vector clock and replicates the update to other nodes.
 - Recipient nodes compare vector clocks.
 - o If the update is causally valid, it is applied immediately.
 - o Otherwise, the message is buffered until dependencies are resolved.
-

4. Implementation Details

Technologies Used:

- Python 3 (Flask for HTTP services)
- Docker, Docker Compose

Directory Structure:

```
vector-clock-kv-store/  
|-- src/  
|   |-- node.py           # Node server implementation  
|   |-- client.py         # Script to send operations  
|-- Dockerfile            # Docker configuration for each node  
|-- docker-compose.yml    # Multi-node orchestration
```

Key Logic in `node.py`:

- Maintains a dictionary for the key-value store.
- Maintains a vector clock as a dictionary keyed by node ID.
- Implements:
 - o PUT /put: Updates the key-value store, increments clock, broadcasts update.
 - o POST /replicate: Receives updates from other nodes, checks causality.
- Buffered updates are stored in a queue and revisited periodically.

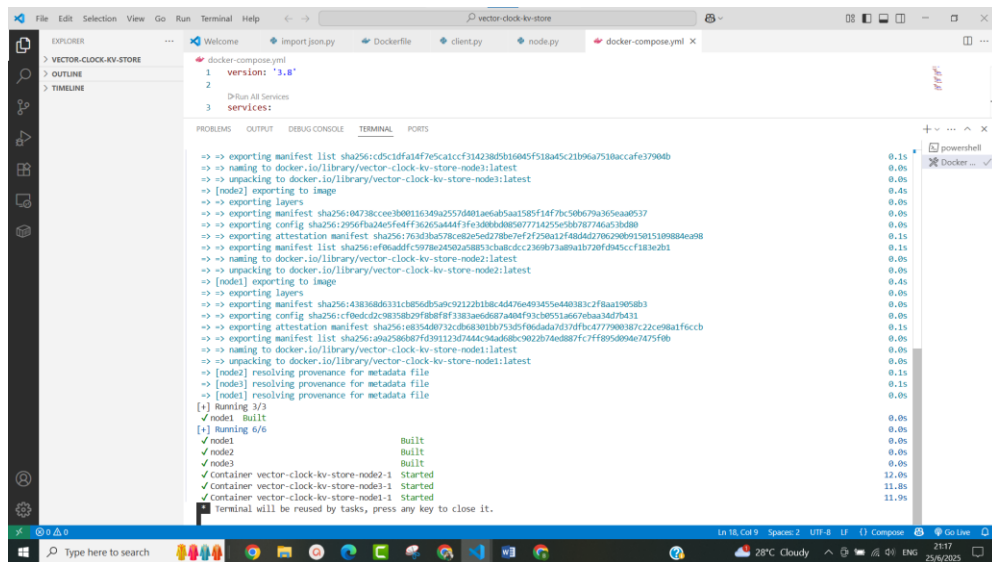
Causal Delivery Check:

```
for i in range(len(vector_clock)):  
    if i == sender_id:  
        if recv_clock[i] != local_clock[i] + 1:  
            return False  
    elif recv_clock[i] > local_clock[i]:  
        return False
```

Client Logic (*client.py*):

- Simulates scenarios such as:
 - Concurrent updates from different nodes.
 - Read-after-write behavior.
 - Out-of-order message delivery.

4. Testing and Results



```
vector-clock-kv-store
docker-compose.yml
1 version: '3.8'
2
3 DrRun All Services
services:
  node1:
    build: .
    container_name: vector-clock-kv-store-node1
    restart: always
  node2:
    build: .
    container_name: vector-clock-kv-store-node2
    restart: always
  node3:
    build: .
    container_name: vector-clock-kv-store-node3
    restart: always

[+] Running 3/3
 ✓ node1 Built
 ✓ node2 Built
 ✓ node3 Built
 ✓ container vector-clock-kv-store-node2-1 Started
 ✓ container vector-clock-kv-store-node3-1 Started
 ✓ container vector-clock-kv-store-node1-1 Started
Terminal will be reused by tasks, press any key to close it.
```

6. Video Demonstration

https://drive.google.com/file/d/1XhiPRSFWGSGyVO-62UbjcSuBgx29xh_7/view?usp=sharing

7. Conclusion

This project successfully demonstrates the use of Vector Clocks in maintaining causal consistency within a distributed key-value store. The system correctly buffers and delivers messages based on vector clock comparison. Containerization ensures easy deployment and testing of multi-node setups. The testing scenarios prove that the implementation meets all consistency goals set by the assignment