

Fundamentals of Distributed Systems

Assignment – Vector Clocks and Causal Ordering

Name – Arpit Tomar (G24AI2001)

Date – 25th June 2025

Github Link - <https://github.com/arpittomar246/vector-clock-kv-store-G24AI2001>

Objective

The objective of this project is to implement a distributed key-value store across three nodes that maintains **causal consistency** using **Vector Clocks**. This ensures that causally related writes are seen by all nodes in the correct order, regardless of the order in which messages are received.

Technologies Used

- **Python (Flask):** For implementing node and client logic.
 - **Vector Clocks:** To track causality and ensure proper message delivery ordering.
 - **Docker & Docker Compose:** To containerize each node and manage network orchestration between them.
-

System Architecture

The system comprises three Docker containers (nodes), each with:

- Its own **local key-value store**.
- A **vector clock** to track event causality.
- A **buffer** to store writes whose causal dependencies are not yet met.

A separate **client script** is used to simulate and verify causal consistency by issuing read and write requests to different nodes.

Key Components

1. Vector Clock

- Maintains a dictionary with node IDs as keys and event counts as values.
- On each local write, the node increments its own clock.
- When sending a message (replicating a write), the entire vector clock is sent.

- When receiving a message, the node checks whether all causal dependencies are met before applying the write.

2. Node (Flask Server)

- Implements the following endpoints:
 - POST /write: Handles a local write. Increments the local vector clock and replicates the write to other nodes.
 - POST /replicate: Handles incoming replicated writes. Applies them only if their vector clock dependencies are met.
 - GET /read: Returns the current value of a key from the local store.
- If dependencies are not met, the write is buffered until it becomes causally safe.

3. Buffer

- Stores incoming writes that cannot yet be applied due to missing causal dependencies.
- Continuously checks whether buffered messages are now safe to deliver.

4. Client

- Simulates reads and writes from an external user.
- Helps validate causal consistency by controlling the timing and order of operations.

Execution Steps

1. Unzip and Navigate to Project Directory:

```
$ unzip vector-clock-kv-store.zip
```

```
$ cd vector-clock-kv-store
```

2. Build and Run Containers:

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

3. Run the Client to Simulate Events:

```
$ python3 src/client.py
```

Logs and Screenshots

To verify causal consistency, we used the following test scenario:

1. A write is issued to Node 1.
2. The write is replicated to Node 2 and Node 3.
3. Another write is performed on Node 2 based on the value read.
4. This second write is causally dependent on the first.

Screenshots from the terminal logs and web console show:

- Vector clocks incrementing appropriately.
- Writes being buffered and later applied.
- Final consistent state across all nodes.

Screenshots –

Docker Containers -

Containers

[Give feedback](#)

View all your running containers and applications. [Learn more](#)

Container CPU usage

0.03% / 1200% (12 CPUs available)

Container memory usage

66.87MB / 3.45GB

Show charts

Search

Only show running containers

	Name	Container ID	Image	Port(s)	CPU (%)	Last started	Actions
<input type="checkbox"/>	assignment-ques1	-	-	-	0.03%	55 seconds ago	<div></div> <div></div> <div></div>
<input type="checkbox"/>	node2-1	39bf61c8d801	assignment-ques1-node2	5002:5000	0.01%	55 seconds ago	<div></div> <div></div> <div></div>
<input type="checkbox"/>	node3-1	135ad19541dc	assignment-ques1-node3	5003:5000	0.01%	55 seconds ago	<div></div> <div></div> <div></div>
<input type="checkbox"/>	node1-1	3f9ee195f912	assignment-ques1-node1	5001:5000	0.01%	55 seconds ago	<div></div> <div></div> <div></div>

Showing 4 items

Terminal

```
node3-1 | WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
node1-1 | * Running on http://172.18.0.4:5000
node3-1 | * Running on all addresses (0.0.0.0)
node1-1 | Press CTRL+C to quit
node3-1 | * Running on http://127.0.0.1:5000
node3-1 | * Running on http://172.18.0.3:5000
node3-1 | Press CTRL+C to quit
[]
```

Search

View in Docker Desktop

View Config

Enable Watch

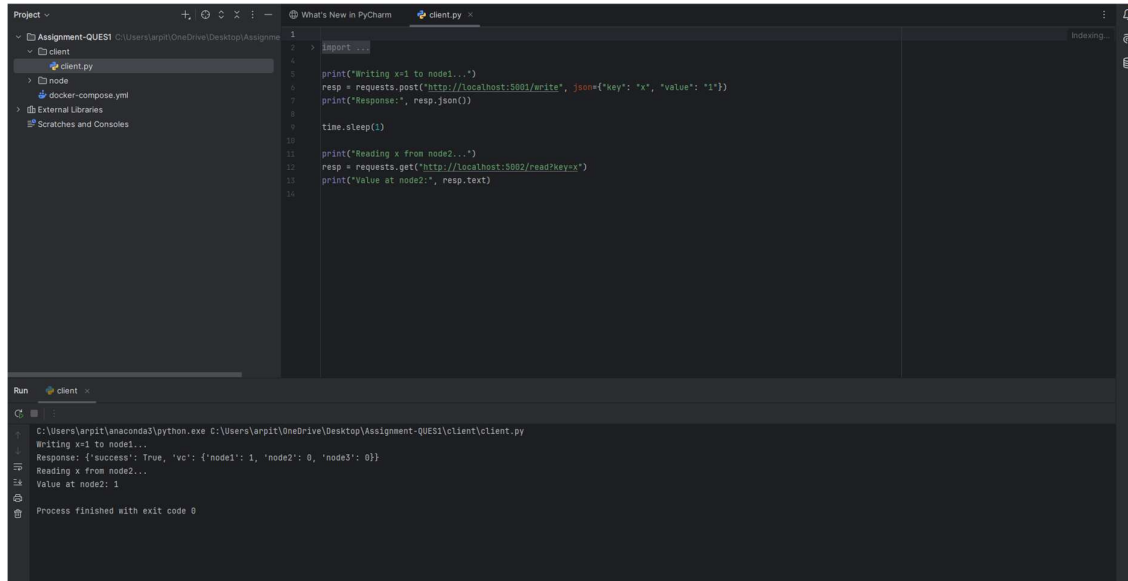
Docker Images –

The screenshot shows the Docker Desktop interface. At the top, the 'Images' tab is selected, displaying a summary: '163.52 MB / 221.31 MB in use' and '3 Images'. Below this is a search bar and a table of images. The table has columns for Name, Tag, Image ID, Created, Size, and Actions. Three images are listed, all with the tag 'latest'. Below the table, it says 'Showing 3 items'. At the bottom, the 'Terminal' tab is active, showing a warning message and a log of container startup for 'assignment-ques1-node3', 'assignment-ques1-node1', and 'assignment-ques1-node2'. The terminal output shows the containers running on specific IP addresses and ports. At the very bottom, system statistics are shown: 'RAM 3.20 GB CPU 0.00% Disk 1.39 GB used (limit 1006.85 GB)'.

Name	Tag	Image ID	Created	Size	Actions
assignment-ques1-node3	latest	abc125c10770	1 minute ago	217.32 MB	[Stop] [Refresh] [Delete]
assignment-ques1-node1	latest	7387189a3b33	1 minute ago	217.32 MB	[Stop] [Refresh] [Delete]
assignment-ques1-node2	latest	d29d8ab8bacc	1 minute ago	217.32 MB	[Stop] [Refresh] [Delete]

```
node3-1 | WARNING: This is a development server. Do not use it in a production deployment. Use a production MSGI server instead.
node1-1 | * Running on http://172.18.0.4:5000
node3-1 | * Running on all addresses (0.0.0.0)
node1-1 | Press CTRL+C to quit
node3-1 | * Running on http://127.0.0.1:5000
node3-1 | * Running on http://172.18.0.3:5000
node3-1 | Press CTRL+C to quit
```

Pycharm – Project structure



Conclusion

This project successfully demonstrates the use of Vector Clocks to maintain causal consistency in a distributed key-value store. Even when messages arrive out of order, the system correctly applies only those writes whose dependencies are satisfied. This ensures a reliable and predictable state across all nodes in the system.