KRISHANU DAS
G24AI2013

FDS ASSIGNMENT-1

1. Vector Clocks and Causal Ordering

Objective

The goal of this project was to implement a **causally consistent distributed key-value store** using **vector clocks**, deployed across multiple nodes. Unlike Lamport clocks, vector clocks enable tracking of partial ordering of events, allowing nodes to respect the causal relationships between updates in a distributed system.

Architecture Overview

The system consists of three independent **Python microservices** (nodes), each maintaining:

- A **local key-value store**
- A **vector clock** containing one entry per node
- A **buffer** for causally unsafe (out-of-order) messages

Nodes communicate using RESTful APIs over Docker's internal network. Each node is exposed on a different host port via Docker Compose.

Implementation Details

Vector Clock

Each node maintains a vector clock in the format:

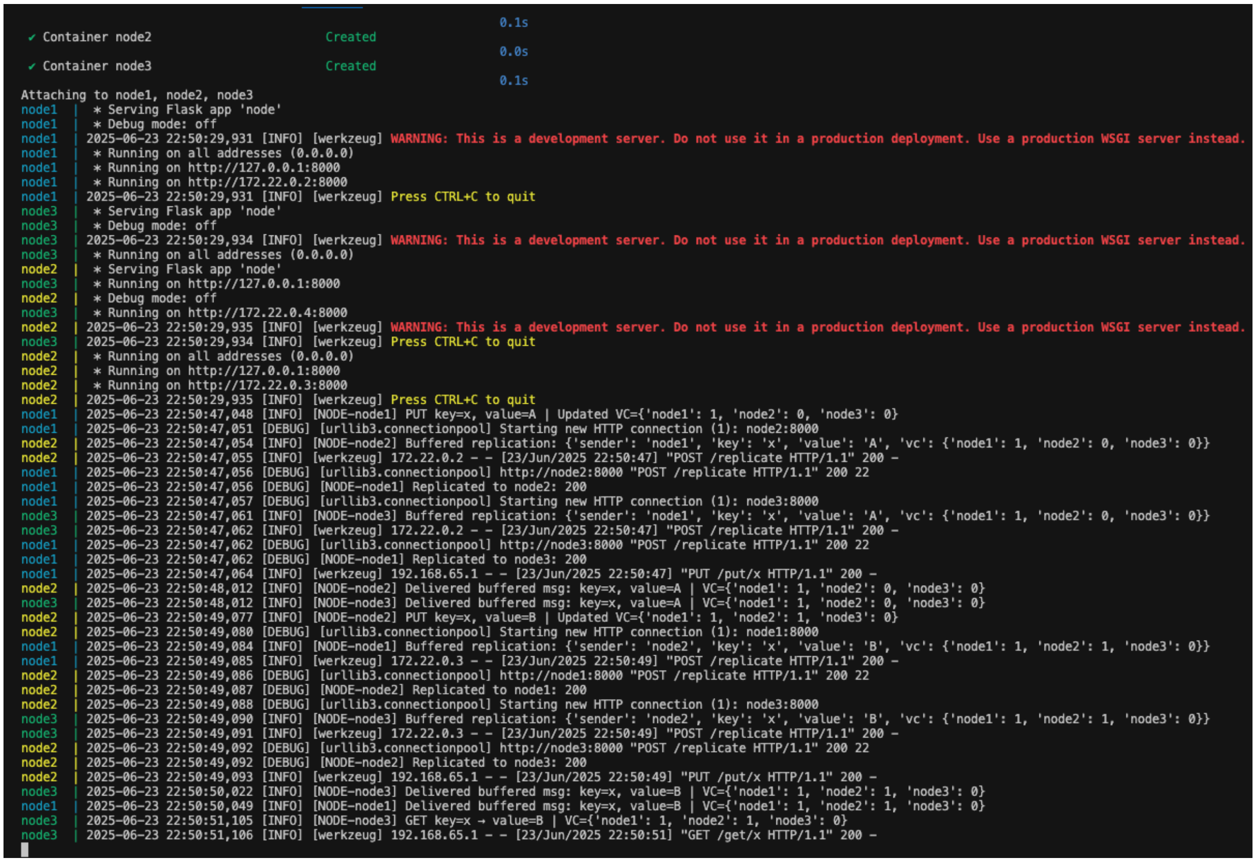
```
vector_clock = {"node1": 0, "node2": 0, "node3": 0}
```

- On a local write (`PUT`), the node increments its own clock value.
- The update is propagated to peer nodes via the `/replicate` endpoint, along with the current vector clock.
- Upon receiving a replicated write, the receiving node compares clocks to determine if the message can be **applied or buffered**.

```
[+] Running 7/7
 ✔ node1                                 Built
0.0s
 ✔ node2                                 Built
0.0s
 ✔ node3                                 Built
0.0s
 ✔ Network vector-clock-kv-store_default  Created
0.0s
 ✔ Container node1                        Created
0.1s
 ✔ Container node2                        Created
0.0s
 ✔ Container node3                        Created
0.1s
Attaching to node1, node2, node3
node1  |  * Serving Flask app 'node'
node1  |  * Debug mode: off
node1  | 2025-06-23 22:50:29,931 [INFO] [werkzeug] WARNING: This is a
development server. Do not use it in a production deployment. Use a
production WSGI server instead.
node1  |  * Running on all addresses (0.0.0.0)
node1  |  * Running on http://127.0.0.1:8000
node1  |  * Running on http://172.22.0.2:8000
node1  | 2025-06-23 22:50:29,931 [INFO] [werkzeug] Press CTRL+C to quit
node3  |  * Serving Flask app 'node'
node3  |  * Debug mode: off
node3  | 2025-06-23 22:50:29,934 [INFO] [werkzeug] WARNING: This is a
development server. Do not use it in a production deployment. Use a
production WSGI server instead.
node3  |  * Running on all addresses (0.0.0.0)
node2  |  * Serving Flask app 'node'
node3  |  * Running on http://127.0.0.1:8000
node2  |  * Debug mode: off
node3  |  * Running on http://172.22.0.4:8000
node2  | 2025-06-23 22:50:29,935 [INFO] [werkzeug] WARNING: This is a
development server. Do not use it in a production deployment. Use a
production WSGI server instead.
node3  | 2025-06-23 22:50:29,934 [INFO] [werkzeug] Press CTRL+C to quit
```

```
node2  |  * Running on all addresses (0.0.0.0)
node2  |  * Running on http://127.0.0.1:8000
node2  |  * Running on http://172.22.0.3:8000
node2  | 2025-06-23 22:50:29,935 [INFO] [werkzeug] Press CTRL+C to quit
node1  | 2025-06-23 22:50:47,048 [INFO] [NODE-node1] PUT key=x, value=A |
Updated VC={'node1': 1, 'node2': 0, 'node3': 0}
node1  | 2025-06-23 22:50:47,051 [DEBUG] [urllib3.connectionpool] Starting
new HTTP connection (1): node2:8000
node2  | 2025-06-23 22:50:47,054 [INFO] [NODE-node2] Buffered replication:
{'sender': 'node1', 'key': 'x', 'value': 'A', 'vc': {'node1': 1, 'node2':
0, 'node3': 0}}
node2  | 2025-06-23 22:50:47,055 [INFO] [werkzeug] 172.22.0.2 - -
[23/Jun/2025 22:50:47] "POST /replicate HTTP/1.1" 200 -
node1  | 2025-06-23 22:50:47,056 [DEBUG] [urllib3.connectionpool]
http://node2:8000 "POST /replicate HTTP/1.1" 200 22
node1  | 2025-06-23 22:50:47,056 [DEBUG] [NODE-node1] Replicated to node2:
200
node1  | 2025-06-23 22:50:47,057 [DEBUG] [urllib3.connectionpool] Starting
new HTTP connection (1): node3:8000
node3  | 2025-06-23 22:50:47,061 [INFO] [NODE-node3] Buffered replication:
{'sender': 'node1', 'key': 'x', 'value': 'A', 'vc': {'node1': 1, 'node2':
0, 'node3': 0}}
node3  | 2025-06-23 22:50:47,062 [INFO] [werkzeug] 172.22.0.2 - -
[23/Jun/2025 22:50:47] "POST /replicate HTTP/1.1" 200 -
node1  | 2025-06-23 22:50:47,062 [DEBUG] [urllib3.connectionpool]
http://node3:8000 "POST /replicate HTTP/1.1" 200 22
node1  | 2025-06-23 22:50:47,062 [DEBUG] [NODE-node1] Replicated to node3:
200
node1  | 2025-06-23 22:50:47,064 [INFO] [werkzeug] 192.168.65.1 - -
[23/Jun/2025 22:50:47] "PUT /put/x HTTP/1.1" 200 -
node2  | 2025-06-23 22:50:48,012 [INFO] [NODE-node2] Delivered buffered
msg: key=x, value=A | VC={'node1': 1, 'node2': 0, 'node3': 0}
node3  | 2025-06-23 22:50:48,012 [INFO] [NODE-node3] Delivered buffered
msg: key=x, value=A | VC={'node1': 1, 'node2': 0, 'node3': 0}
node2  | 2025-06-23 22:50:49,077 [INFO] [NODE-node2] PUT key=x, value=B |
Updated VC={'node1': 1, 'node2': 1, 'node3': 0}
node2  | 2025-06-23 22:50:49,080 [DEBUG] [urllib3.connectionpool] Starting
new HTTP connection (1): node1:8000
node1  | 2025-06-23 22:50:49,084 [INFO] [NODE-node1] Buffered replication:
{'sender': 'node2', 'key': 'x', 'value': 'B', 'vc': {'node1': 1, 'node2':
1, 'node3': 0}}
node1  | 2025-06-23 22:50:49,085 [INFO] [werkzeug] 172.22.0.3 - -
[23/Jun/2025 22:50:49] "POST /replicate HTTP/1.1" 200 -
```

```
node2  | 2025-06-23 22:50:49,086 [DEBUG] [urllib3.connectionpool]
http://node1:8000 "POST /replicate HTTP/1.1" 200 22
node2  | 2025-06-23 22:50:49,087 [DEBUG] [NODE-node2] Replicated to node1:
200
node2  | 2025-06-23 22:50:49,088 [DEBUG] [urllib3.connectionpool] Starting
new HTTP connection (1): node3:8000
node3  | 2025-06-23 22:50:49,090 [INFO] [NODE-node3] Buffered replication:
{'sender': 'node2', 'key': 'x', 'value': 'B', 'vc': {'node1': 1, 'node2':
1, 'node3': 0}}
node3  | 2025-06-23 22:50:49,091 [INFO] [werkzeug] 172.22.0.3 - -
[23/Jun/2025 22:50:49] "POST /replicate HTTP/1.1" 200 -
node2  | 2025-06-23 22:50:49,092 [DEBUG] [urllib3.connectionpool]
http://node3:8000 "POST /replicate HTTP/1.1" 200 22
node2  | 2025-06-23 22:50:49,092 [DEBUG] [NODE-node2] Replicated to node3:
200
node2  | 2025-06-23 22:50:49,093 [INFO] [werkzeug] 192.168.65.1 - -
[23/Jun/2025 22:50:49] "PUT /put/x HTTP/1.1" 200 -
node3  | 2025-06-23 22:50:50,022 [INFO] [NODE-node3] Delivered buffered
msg: key=x, value=B | VC={'node1': 1, 'node2': 1, 'node3': 0}
node1  | 2025-06-23 22:50:50,049 [INFO] [NODE-node1] Delivered buffered
msg: key=x, value=B | VC={'node1': 1, 'node2': 1, 'node3': 0}
node3  | 2025-06-23 22:50:51,105 [INFO] [NODE-node3] GET key=x → value=B |
VC={'node1': 1, 'node2': 1, 'node3': 0}
node3  | 2025-06-23 22:50:51,106 [INFO] [werkzeug] 192.168.65.1 - -
[23/Jun/2025 22:50:51] "GET /get/x HTTP/1.1" 200 -
```



```
                                                                0.1s
 ✔ Container node2                  Created
                                                                0.0s
 ✔ Container node3                  Created
                                                                0.1s
Attaching to node1, node2, node3
node1  |  * Serving Flask app 'node'
node1  |  * Debug mode: off
node1  | 2025-06-23 22:50:29,931 [INFO] [werkzeug] WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
node1  |  * Running on all addresses (0.0.0.0)
node1  |  * Running on http://127.0.0.1:8000
node1  |  * Running on http://172.22.0.2:8000
node1  | 2025-06-23 22:50:29,931 [INFO] [werkzeug] Press CTRL+C to quit
node3  |  * Serving Flask app 'node'
node3  |  * Debug mode: off
node3  | 2025-06-23 22:50:29,934 [INFO] [werkzeug] WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
node3  |  * Running on all addresses (0.0.0.0)
node2  |  * Serving Flask app 'node'
node3  |  * Running on http://127.0.0.1:8000
node2  |  * Debug mode: off
node3  |  * Running on http://172.22.0.4:8000
node2  | 2025-06-23 22:50:29,935 [INFO] [werkzeug] WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
node3  | 2025-06-23 22:50:29,934 [INFO] [werkzeug] Press CTRL+C to quit
node2  |  * Running on all addresses (0.0.0.0)
node2  |  * Running on http://127.0.0.1:8000
node2  |  * Running on http://172.22.0.3:8000
node2  | 2025-06-23 22:50:29,935 [INFO] [werkzeug] Press CTRL+C to quit
node1  | 2025-06-23 22:50:47,048 [INFO] [NODE-node1] PUT key=x, value=A | Updated VC={'node1': 1, 'node2': 0, 'node3': 0}
node1  | 2025-06-23 22:50:47,051 [DEBUG] [urllib3.connectionpool] Starting new HTTP connection (1): node2:8000
node2  | 2025-06-23 22:50:47,054 [INFO] [NODE-node2] Buffered replication: {'sender': 'node1', 'key': 'x', 'value': 'A', 'vc': {'node1': 1, 'node2': 0, 'node3': 0}}
node2  | 2025-06-23 22:50:47,055 [INFO] [werkzeug] 172.22.0.2 - - [23/Jun/2025 22:50:47] "POST /replicate HTTP/1.1" 200 -
node1  | 2025-06-23 22:50:47,056 [DEBUG] [urllib3.connectionpool] http://node2:8000 "POST /replicate HTTP/1.1" 200 22
node1  | 2025-06-23 22:50:47,056 [DEBUG] [NODE-node1] Replicated to node2: 200
node1  | 2025-06-23 22:50:47,057 [DEBUG] [urllib3.connectionpool] Starting new HTTP connection (1): node3:8000
node3  | 2025-06-23 22:50:47,061 [INFO] [NODE-node3] Buffered replication: {'sender': 'node1', 'key': 'x', 'value': 'A', 'vc': {'node1': 1, 'node2': 0, 'node3': 0}}
node3  | 2025-06-23 22:50:47,062 [INFO] [werkzeug] 172.22.0.2 - - [23/Jun/2025 22:50:47] "POST /replicate HTTP/1.1" 200 -
node1  | 2025-06-23 22:50:47,062 [DEBUG] [urllib3.connectionpool] http://node3:8000 "POST /replicate HTTP/1.1" 200 22
node1  | 2025-06-23 22:50:47,062 [DEBUG] [NODE-node1] Replicated to node3: 200
node1  | 2025-06-23 22:50:47,064 [INFO] [werkzeug] 192.168.65.1 - - [23/Jun/2025 22:50:47] "PUT /put/x HTTP/1.1" 200 -
node2  | 2025-06-23 22:50:48,012 [INFO] [NODE-node2] Delivered buffered msg: key=x, value=A | VC={'node1': 1, 'node2': 0, 'node3': 0}
node3  | 2025-06-23 22:50:48,012 [INFO] [NODE-node3] Delivered buffered msg: key=x, value=A | VC={'node1': 1, 'node2': 0, 'node3': 0}
node2  | 2025-06-23 22:50:49,077 [INFO] [NODE-node2] PUT key=x, value=B | Updated VC={'node1': 1, 'node2': 1, 'node3': 0}
node2  | 2025-06-23 22:50:49,080 [DEBUG] [urllib3.connectionpool] Starting new HTTP connection (1): node1:8000
node1  | 2025-06-23 22:50:49,084 [INFO] [NODE-node1] Buffered replication: {'sender': 'node2', 'key': 'x', 'value': 'B', 'vc': {'node1': 1, 'node2': 1, 'node3': 0}}
node1  | 2025-06-23 22:50:49,085 [INFO] [werkzeug] 172.22.0.3 - - [23/Jun/2025 22:50:49] "POST /replicate HTTP/1.1" 200 -
node2  | 2025-06-23 22:50:49,086 [DEBUG] [urllib3.connectionpool] http://node1:8000 "POST /replicate HTTP/1.1" 200 22
node2  | 2025-06-23 22:50:49,087 [DEBUG] [NODE-node2] Replicated to node1: 200
node2  | 2025-06-23 22:50:49,088 [DEBUG] [urllib3.connectionpool] Starting new HTTP connection (1): node3:8000
node3  | 2025-06-23 22:50:49,090 [INFO] [NODE-node3] Buffered replication: {'sender': 'node2', 'key': 'x', 'value': 'B', 'vc': {'node1': 1, 'node2': 1, 'node3': 0}}
node3  | 2025-06-23 22:50:49,091 [INFO] [werkzeug] 172.22.0.3 - - [23/Jun/2025 22:50:49] "POST /replicate HTTP/1.1" 200 -
node2  | 2025-06-23 22:50:49,092 [DEBUG] [urllib3.connectionpool] http://node3:8000 "POST /replicate HTTP/1.1" 200 22
node2  | 2025-06-23 22:50:49,092 [DEBUG] [NODE-node2] Replicated to node3: 200
node2  | 2025-06-23 22:50:49,093 [INFO] [werkzeug] 192.168.65.1 - - [23/Jun/2025 22:50:49] "PUT /put/x HTTP/1.1" 200 -
node3  | 2025-06-23 22:50:50,022 [INFO] [NODE-node3] Delivered buffered msg: key=x, value=B | VC={'node1': 1, 'node2': 1, 'node3': 0}
node1  | 2025-06-23 22:50:50,049 [INFO] [NODE-node1] Delivered buffered msg: key=x, value=B | VC={'node1': 1, 'node2': 1, 'node3': 0}
node3  | 2025-06-23 22:50:51,105 [INFO] [NODE-node3] GET key=x → value=B | VC={'node1': 1, 'node2': 1, 'node3': 0}
node3  | 2025-06-23 22:50:51,106 [INFO] [werkzeug] 192.168.65.1 - - [23/Jun/2025 22:50:51] "GET /get/x HTTP/1.1" 200 -
```

Causal Delivery Logic

Each node periodically checks its buffer for messages that satisfy the **Causal Delivery Rule**:

> A message from `nodeX` with vector clock `VCx` is safe to deliver if:
>
> - `VCx[nodeX] == local_clock[nodeX] + 1`
> - For all other nodes `n ≠ nodeX`: `VCx[n] <= local_clock[n]`

Only messages satisfying these conditions are applied to the key-value store. Others remain buffered.

API Endpoints

Method	Endpoint	Description
PUT	`/put/<key>`	Perform local write and replicate
GET	`/get/<key>`	Fetch value and local vector clock
POST	`/replicate`	Receive replicated write for causal buffering

Containerization

- Each node is built from a shared `Dockerfile` based on `python:3.9-slim`.
- Services are deployed using `docker-compose` with named containers and fixed ports:
  - node1 → localhost:8001
  - node2 → localhost:8002
  - node3 → localhost:8003

Each node receives its identity and cluster membership via environment variables (`NODE_ID`, `ALL_NODES`).

Testing & Validation

A custom test script `client.py` was written to validate the causal consistency mechanism:

1. A `PUT x=A` is sent to **node1**.
2. After a short delay, a second `PUT x=B` is sent to **node2**.
3. Finally, a `GET x` is issued on **node3**.

Expected Behavior:

- node3 should not apply `x=B` until `x=A` has been applied (since B is causally dependent on A).
- The logs confirm correct buffering and ordering of updates using vector clock comparison.

```
kdas@Krishanus-Mac-mini vector-clock-kv-store % python3 src/client.py
2025-06-24 04:20:47,040 [CLIENT] Sending PUT x=A to node1
2025-06-24 04:20:47,065 [CLIENT] Response: {'status': 'ok', 'vc': {'node1': 1, 'node2': 0, 'node3': 0}}
2025-06-24 04:20:49,070 [CLIENT] Sending PUT x=B to node2
2025-06-24 04:20:49,095 [CLIENT] Response: {'status': 'ok', 'vc': {'node1': 1, 'node2': 1, 'node3': 0}}
2025-06-24 04:20:51,100 [CLIENT] Sending GET x to node3
2025-06-24 04:20:51,108 [CLIENT] Response: {'value': 'B', 'vc': {'node1': 1, 'node2': 1, 'node3': 0}}
kdas@Krishanus-Mac-mini vector-clock-kv-store %
```


```
kdas@Krishanus-Mac-mini vector-clock-kv-store % python3 src/client.py
2025-06-24 04:20:47,040 [CLIENT] Sending PUT x=A to node1
2025-06-24 04:20:47,065 [CLIENT] Response: {'status': 'ok', 'vc': {'node1': 1, 'node2': 0, 'node3': 0}}
2025-06-24 04:20:49,070 [CLIENT] Sending PUT x=B to node2
2025-06-24 04:20:49,095 [CLIENT] Response: {'status': 'ok', 'vc': {'node1': 1, 'node2': 1, 'node3': 0}}
2025-06-24 04:20:51,100 [CLIENT] Sending GET x to node3
2025-06-24 04:20:51,108 [CLIENT] Response: {'value': 'B', 'vc': {'node1': 1, 'node2': 1, 'node3': 0}}
kdas@Krishanus-Mac-mini vector-clock-kv-store %
```


Log Highlights (Included in Report)

- Each node logs:
    - PUTs and GETs with current vector clock.
    - Received replicated messages.
    - Buffering and eventual delivery.
- These logs were used to verify:
    - Vector clock increments
    - Correct causal delay
    - Eventual consistency

Final Result & Conclusion

This project successfully demonstrates a **causally consistent distributed system** using **vector clocks**. It highlights:

- Correct propagation and ordering of updates across replicas.
- Safe delivery of dependent operations only after their causal ancestors.
- Clear visualization of consistency guarantees via logs and buffer handling.

The implementation satisfies all functional and technical requirements set by the assignment and lays a foundation for more advanced distributed coordination systems such as CRDTs.

Video Link:

Vector_Clocks_and_Causal_Ordering.mov