India Institute of Technology, Jodhpur

Name: Amish Kar
Roll Number: G24AI2086
Subject: FDS

Assignment 1: Vector Clocks and Causal Ordering

Objective : Distributed systems require mechanisms to maintain consistency across multiple nodes. This project aims to implement a distributed key-value store that ensures causal consistency using vector clocks.

Challenges:

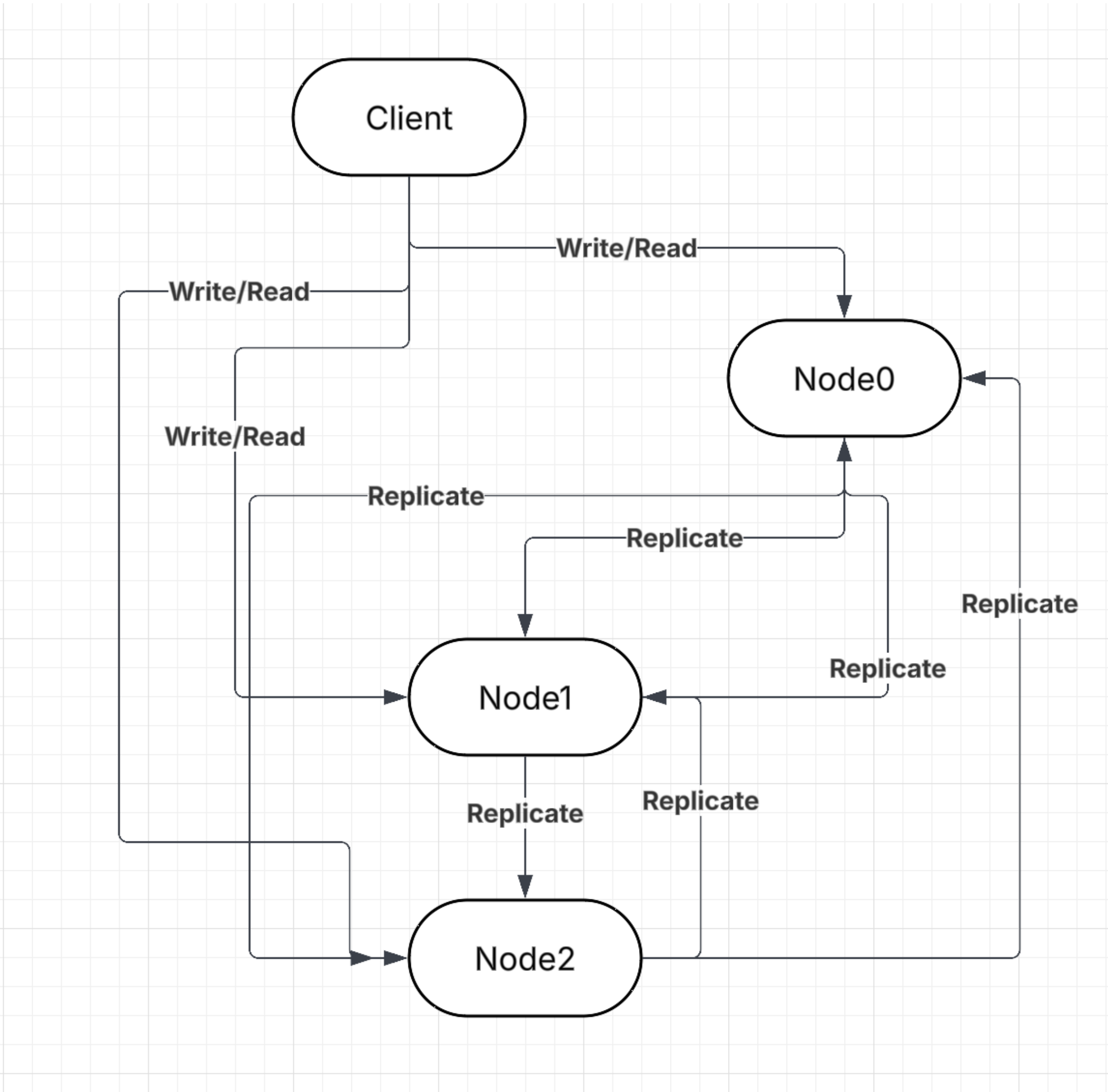
- **Causal Ordering**: Ensuring that if Event B depends on Event A, all nodes process A before B.
- **Network Partitions**: Handling delayed or lost messages while maintaining consistency.
- **Concurrency Control**: Managing concurrent writes across distributed nodes.

System Architecture:

The system consists of:

1. **Multiple Nodes**: Each node stores key-value pairs and maintains a **vector clock**.
2. **Client**: Tests causal consistency by performing writes and reads.
3. **Docker Containers**: Each node runs in an isolated container.

Component Diagram:



Data Flow:

1. **Write Request**: A client sends a write to a node.
2. **Vector Clock Update**: The node increments its own clock and propagates the update.
3. **Replication**: The node replicates the write to other nodes.
4. **Causal Delivery**: Nodes buffer writes until dependencies are satisfied.

Implementation Details:

Vector Clock Logic

Each node maintains a vector clock to track causality class VectorClock: will implement the logic for Vector clock.

```
class VectorClock:
    def __init__(self, node_id, node_count):
        self.node_id = node_id
        self.clock = {str(i): 0 for i in range(node_count)}
        self.lock = Lock()

    def increment(self):
        with self.lock:
            self.clock[str(self.node_id)] += 1
            return self.clock.copy()

    def update(self, received_clock):
        with self.lock:
            for node, time in received_clock.items():
                node_str = str(node)
                if time > self.clock.get(node_str, 0):
                    self.clock[node_str] = time
```

Causal Consistency Rules

Before applying a write, a node checks:
is_causally_ready

```python
def is_causally_ready(self, received_clock):
    for node, time in received_clock.items():
        node_str = str(node)
        if node_str != str(self.node_id) and time > self.clock.get(node_str, 0):
            return False
    return True
```

Key APIs

POST /write	Write a key-value pair (triggers replication)
POST /replicate	Internal replication endpoint
GET /read/<key>	Read a value with its vector clock
GET /debug	View node state (data, pending writes, clock)

Performance Analysis

Test Scenario

1. **Write to Node0**: POST /write {"key": "x", "value": 1}
2. **Read from Node1**: GET /read/x → Returns {"value": 1, "clock": {"0":1, "1":0, "2":0}}
3. **Write to Node1**: POST /write {"key": "y", "value": 2} (depends on x=1)
4. **Verify at Node2**: GET /read/y → Must return {"value": 2} (proves causality).

Observations:

- **Causal Consistency**: Node2 only processes y=2 after seeing x=1.
- **Network Delays**: If Node2 receives y=2 before x=1, it buffers the write.

Screenshots:

```
Waiting for nodes to be ready...

=== Test 1: Writing initial value to node0 ===
Write successful. Clock: {'0': 1, '1': 0, '2': 0}

=== Test 2: Reading from node1 and writing dependent value ===
Read response: {'clock': {'0': 1, '1': 0, '2': 0}, 'status': 'success', 'value': 1}
Updating value from 1 to 2
Write successful. Clock: {'0': 2, '1': 0, '2': 0}

=== Test 3: Verifying at node2 ===
Causal consistency verified: y=2
Vector clock: {'0': 2, '1': 0, '2': 0}

=== System State ===

Node 0 (http://node0:5000):
Data: {'x': {'clock': {'0': 1, '1': 0, '2': 0}, 'value': 1}}
Pending writes: 0
Vector clock: {'0': 1, '1': 0, '2': 0}

Node 1 (http://node1:5000):
Data: {'x': {'clock': {'0': 1, '1': 0, '2': 0}, 'value': 1}, 'y': {'clock': {'0': 2, '1': 0, '2': 0}, 'value': 2}}
Pending writes: 0
Vector clock: {'0': 2, '1': 0, '2': 0}

Node 2 (http://node2:5000):
Data: {'x': {'clock': {'0': 1, '1': 0, '2': 0}, 'value': 1}, 'y': {'clock': {'0': 2, '1': 0, '2': 0}, 'value': 2}}
Pending writes: 0
Vector clock: {'0': 2, '1': 0, '2': 0}
 * Terminal will be reused by tasks, press any key to close it.
```

Conclusion:

Vector clocks are essential for causal consistency but require careful implementation