Bhuvana Kanakam
SE21UCSE035
CS4101 Lab 02
08.30.2024

Implementing Lamport Logical Clocks in a Distributed System

Vector Clocks

The implementation of vector clocks allows each process in the distributed system to maintain a vector of logical clocks, one for each process. This vector helps to establish causal relationships between events.

Event Ordering Analysis

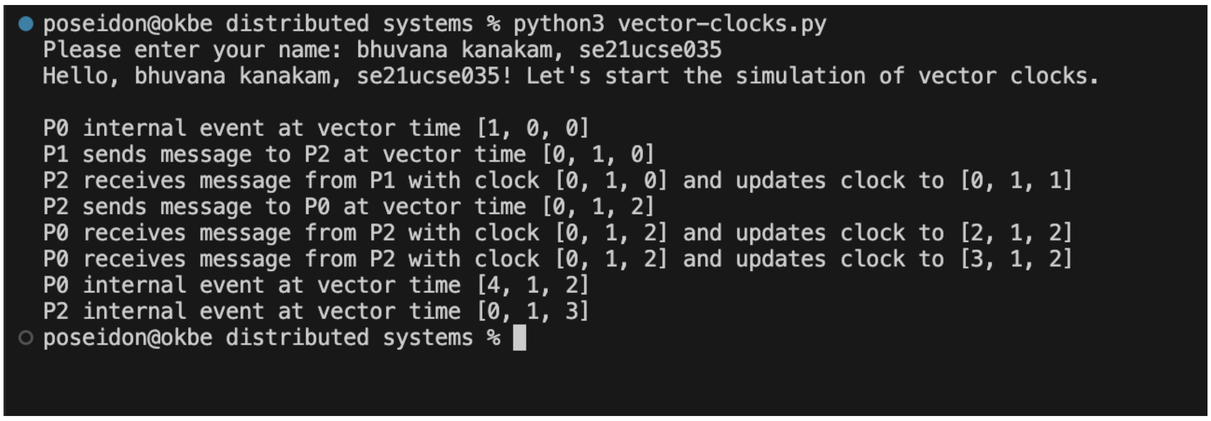
- **Internal Event:** Each process increments its own clock when performing an internal event. For example, when Process 0 performs an internal event, its clock vector updates from $[0, 0, 0]$ to $[1, 0, 0]$.
- **Send and Receive Messages:** When a process sends a message, it increments its own clock and attaches its vector clock to the message. The receiving process updates its clock by taking the element-wise maximum of its own vector and the sender's vector. For example, Process 2 sends a message to Process 3, updating its vector clock to $[0, 1, 0]$. When Process 3 receives this message, it updates its clock to $[0, 2, 1]$.
- **Causal Relationship Detection:** Vector clocks effectively capture causal relationships between events. An event $e1$ is causally before $e2$ if $e1$'s vector clock is strictly less than $e2$'s in at least one position and not greater in any other. This property ensures that messages are only delivered when the causal dependencies are met.

Implementation Code

```python
class VectorProcess:
    def __init__(self, process_id, total_processes):
        self.process_id = process_id
        self.clock = [0] * total_processes

    def increment_clock(self):
        self.clock[self.process_id] += 1

    def internal_event(self):
        self.increment_clock()
        print(f"Process {self.process_id} performs an internal "
              f"event and updates clock to {self.clock}")

    def send_message(self, receiver):
        self.increment_clock()
        print(f"Process {self.process_id} sends a message to Process "
              f"{receiver.process_id} with clock {self.clock}")
        receiver.receive_message(self.clock, self.process_id)

    def receive_message(self, sender_clock, sender_id):
        self.increment_clock()
        self.clock = [max(self.clock[i], sender_clock[i]) for i in
                      range(len(self.clock))]
        print(f"Process {self.process_id} receives a message from Process
            {sender_id} "
              f"with clock {sender_clock} and updates clock to
                {self.clock}")
```

```
process1 = VectorProcess(0, 3)
process2 = VectorProcess(1, 3)
process3 = VectorProcess(2, 3)

process1.internal_event()
process2.send_message(process3)
process3.send_message(process1)
process1.receive_message(process3.clock, process3.process_id)
process1.internal_event()
process3.internal_event()
```

Output



```
poseidon@okbe distributed systems % python3 vector-clocks.py
Please enter your name: bhuvana kanakam, se21ucse035
Hello, bhuvana kanakam, se21ucse035! Let's start the simulation of vector clocks.

P0 internal event at vector time [1, 0, 0]
P1 sends message to P2 at vector time [0, 1, 0]
P2 receives message from P1 with clock [0, 1, 0] and updates clock to [0, 1, 1]
P2 sends message to P0 at vector time [0, 1, 2]
P0 receives message from P2 with clock [0, 1, 2] and updates clock to [2, 1, 2]
P0 receives message from P2 with clock [0, 1, 2] and updates clock to [3, 1, 2]
P0 internal event at vector time [4, 1, 2]
P2 internal event at vector time [0, 1, 3]
poseidon@okbe distributed systems %
```

Concurrency Detection

The concurrency detection program identifies when two events are concurrent, meaning they do not causally depend on each other. This is detected by comparing the vector clocks of the events:

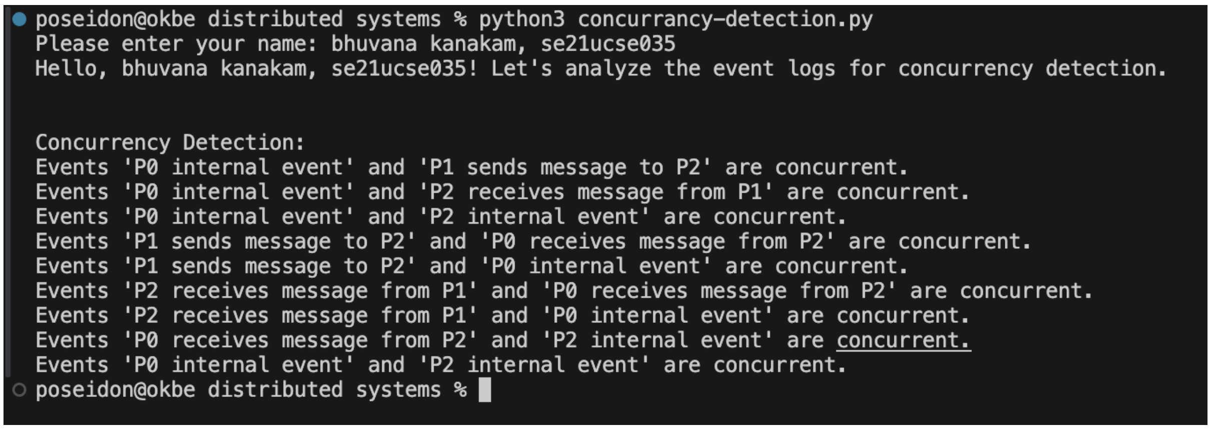
Concurrency Detection Analysis

- **Concurrent Events:** Events are considered concurrent if neither event's vector clock is strictly less than the other. For example, two internal events happening in separate processes, like $"P0 internal event", [1, 0, 0]$ and $"P2 internal event", [0, 2, 3]$, are concurrent since neither process is aware of the other's event.
- **Causal vs. Concurrent:** If the vector clock of one event is not strictly less than or greater than the other, the events are independent. This property is essential for detecting parallel executions in distributed systems, which can be crucial for debugging or optimizing performance.

Implementation Code

```
def is_concurrent(clock1, clock2):
    less_than = all(c1 <= c2 for c1, c2 in zip(clock1, clock2))
    greater_than = all(c1 >= c2 for c1, c2 in zip(clock1, clock2))

def detect_concurrency(event_logs):
    print("\nConcurrency Detection:")
    for i in range(len(event_logs)):
        for j in range(i + 1, len(event_logs)):
            event1, clock1 = event_logs[i]
            event2, clock2 = event_logs[j]
            if is_concurrent(clock1, clock2):
                print(f"Events '{event1}' and '{event2}' are concurrent.")

sample_event_logs = [
    ("P0 internal event", [1, 0, 0]),
    ("P1 sends message to P2", [0, 1, 0]),
    ("P2 receives message from P1", [0, 2, 1]),
    ("P0 receives message from P2", [2, 0, 2]),
    ("P0 internal event", [3, 0, 2]),
    ("P2 internal event", [0, 2, 3])
]

detect_concurrency(sample_event_logs)
```

Output



```
poseidon@okbe distributed systems % python3 concurrancy-detection.py
Please enter your name: bhuvana kanakam, se21ucse035
Hello, bhuvana kanakam, se21ucse035! Let's analyze the event logs for concurrency detection.


Concurrency Detection:
Events 'P0 internal event' and 'P1 sends message to P2' are concurrent.
Events 'P0 internal event' and 'P2 receives message from P1' are concurrent.
Events 'P0 internal event' and 'P2 internal event' are concurrent.
Events 'P1 sends message to P2' and 'P0 receives message from P2' are concurrent.
Events 'P1 sends message to P2' and 'P0 internal event' are concurrent.
Events 'P2 receives message from P1' and 'P0 receives message from P2' are concurrent.
Events 'P2 receives message from P1' and 'P0 internal event' are concurrent.
Events 'P0 receives message from P2' and 'P2 internal event' are concurrent.
Events 'P0 internal event' and 'P2 internal event' are concurrent.
poseidon@okbe distributed systems %
```

Comparison with Lamport Clocks

- **Lamport Clocks:** These only capture the "happens-before" relationship, meaning they can determine if one event causally precedes another but cannot detect concurrency accurately. Vector clocks, on the other hand, provide a more precise ordering by keeping track of the state of each process individually.
- **Concurrency Detection:** Unlike Lamport clocks, vector clocks can determine if events are concurrent, providing richer information about the state of the distributed system.