Name-PRIYA MALIK

Enrollment No.-08501032021

IT2

Assignment 2
Distributed Systems

Q a) : Given a distributed system with three nodes (A, B, and C), illustrate how Lamport timestamps can be used to order the following events:

Node A sends a message to Node B.

Node B sends a message to Node C.

Node C sends a message back to Node A.

Lamport Timestamps: Overview and Explanation

Lamport timestamps are used in distributed systems to provide a logical ordering of events based on the causal relationships between them. This mechanism ensures that events that happen causally earlier are assigned smaller timestamps than those that occur later. The "happened-before" relation (->) is used to capture causality. For example, if event a happens before event b, then a -> b, and the timestamp of a will be smaller than that of b.

Algorithm for Lamport Timestamps:

1. **Initialize Clock:** Each process (node) starts with a logical clock initialized to zero. **2. Event Processing:** When an event occurs within a node, its timestamp is incremented by a fixed drift time (d), generally set to 1.
3. **Message Sending:** When a process sends a message, it includes its current logical clock value in the message.
4. **Message Receiving:** Upon receiving a message, the receiving process updates its clock by taking the maximum of its current clock value and the timestamp in the message, then adds 1.

This ensures the following:

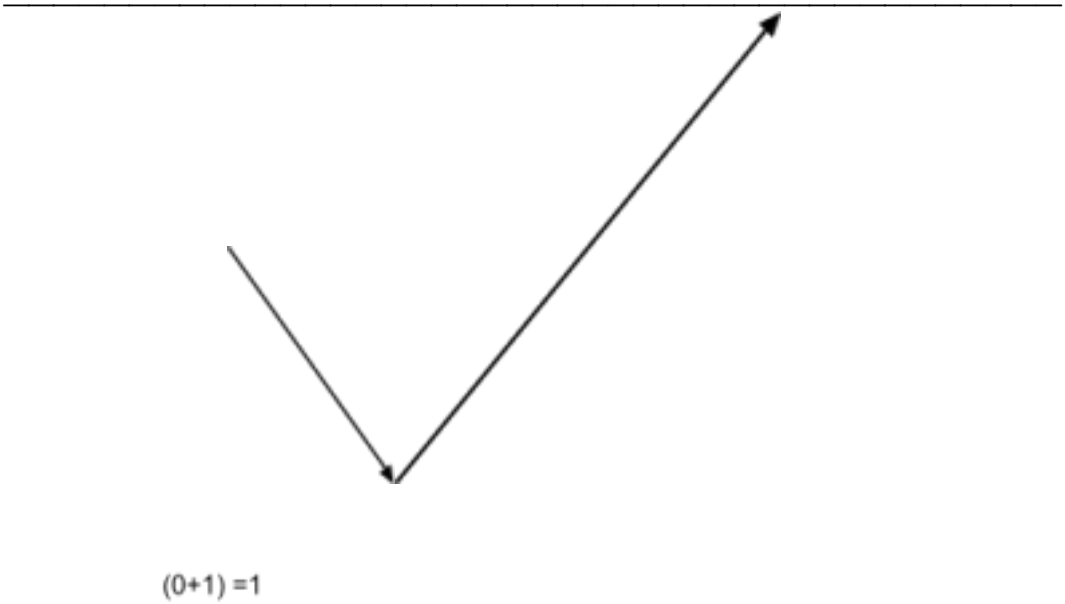
- [C1] If a -> b in the same process, then Ci(a) < Ci(b).
- [C2] If a happens in one process, and its effect is seen as b in another process, then Ci(a) < Cj(b).

Implementation Details:

- For a process Pi with events Eij (jth event in process i), its clock is represented as Ci. ● When an event occurs:
  - Rule 1: If an event a -> b happens in the same process, the logical clock of b is incremented from a as Ci(b) = Ci(a) + 1.
  - Rule 2: When a message is received, the receiving process updates its clock using Cj = max(Cj, tm) + 1, where tm is the timestamp from the received message.

Initial Time Stamp

NodeA 0



NodeB 0

(2+1) =3 Max(0,1) + 1 = 2

NodeC0_____

_____ (4+1) = 5 Max(0,3) +1 = 4

Timestamp Assignment Based on the Diagram:

1. **Node A:** Starts with a timestamp of 0. When it sends a message to Node B, its clock increments by 1. Hence, the timestamp is (0 + 1) = 1.
2. **Node B:** Receives the message from Node A with timestamp 1. Its clock updates to max(0, 1) + 1 = 2. When Node B sends a message to Node C, it increments its clock by 1 to 2 + 1 = 3.
3. **Node C:** Receives the message from Node B with timestamp 3. It updates its clock to max(0, 3) + 1 = 4. When Node C sends a message back to Node A, its clock increments by 1 to 4 + 1 = 5.
4. **Node A (again):** Receives the message from Node C with timestamp 5. It updates its clock to max(1, 5) + 1 = 6.

Causal Relation in the System:

- The causal relationship ensures that every event's timestamp reflects the logical ordering of events:
    - ○ Node A's sending to Node B is the first event.
    - ○ Node B's processing and sending to Node C depend on the message from Node A. ○ Node C's processing and sending back to Node A depend on the message from Node B.

This logical timestamping ensures that the events maintain a strict causal order, as reflected in their timestamps.

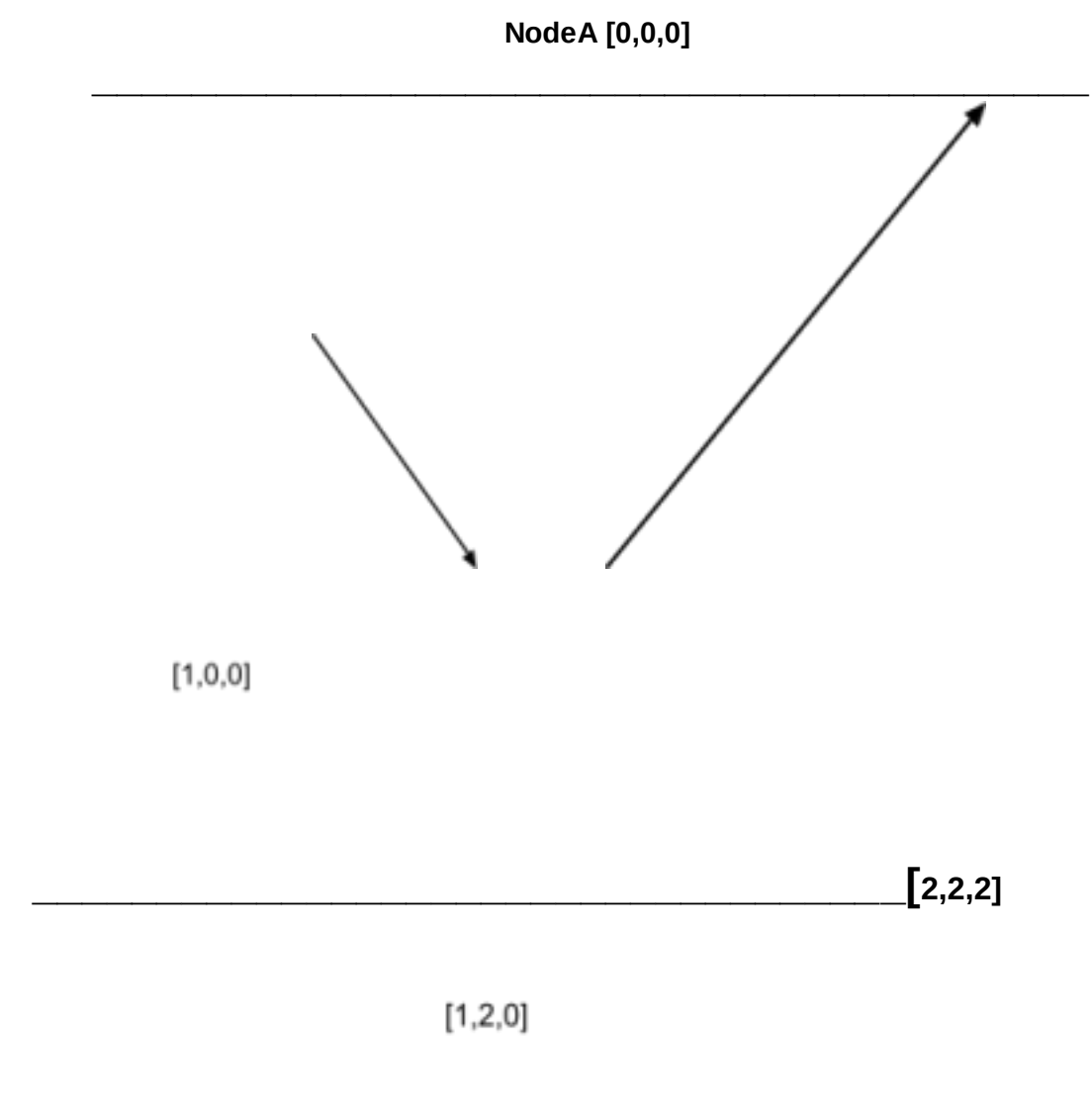
Q b): Identify any ambiguities in event ordering using Lamport timestamps and explain how vector clocks can resolve them.

Lamport timestamps provide a logical ordering of events in a distributed system based on the causal relationship (happened-before relation). However, Lamport

timestamps can fail to capture concurrent events, leading to ambiguities in event ordering.

Ambiguities in Lamport Timestamps

1. **Causal Ambiguity:** Lamport timestamps only ensure that if one event causally affects another, the former will have a smaller timestamp. However, if two events are independent (i.e., neither event causally influences the other), Lamport timestamps cannot distinguish between them. Both events may have timestamps that suggest one occurred before the other, even though they are concurrent.
2. **Concurrency Misinterpretation:** For example, if two nodes (e.g., Node B and Node C) perform independent actions that do not depend on each other, their Lamport timestamps may give the false impression of causality. This ambiguity arises because Lamport timestamps lack the capability to represent concurrent events explicitly.



NodeC [0,0,0]

Example from the Diagram (Using Vector Clocks): Let's consider the distributed system with three nodes: Node A, Node B, and Node C. Using vector clocks, we resolve ambiguities by maintaining a vector of timestamps, one for each node, in the following manner:

1. **Node A Sends to Node B:**
   - ○ Node A starts with [0,0,0] and increments its own clock to [1,0,0] before sending a message.
   - ○ Node B receives the message. It updates its clock to max([0,0,0], [1,0,0]) + 1 = [1,1,0].
2. **Node B Sends to Node C:**
   - ○ Node B increments its clock to [1,2,0] before sending a message to Node C. ○ Node C receives the message and updates its clock to max([0,0,0], [1,2,0]) + 1 = [1,2,1].
3. **Node C Sends Back to Node A:**
   - ○ Node C increments its clock to [1,2,2] and sends a message to Node A. ○ Node A updates its clock to max([1,0,0], [1,2,2]) + 1 = [2,2,2].

Resolving Ambiguities with Vector Clocks

Vector clocks resolve ambiguities by providing a partial ordering of events. The key improvement is that they capture both causality and concurrency.

- **Concurrency Detection:** Using vector clocks, two events e1 and e2 are concurrent if their vector timestamps cannot be ordered, i.e., neither VC(e1) < VC(e2) nor VC(e2) < VC(e1) holds. This allows the system to explicitly recognize independent events.

- **Example of Concurrent Events:** If Node B and Node C had performed independent actions (without communication), their vector clocks might look like [1,1,0] and [0,0,1]. These timestamps are incomparable, indicating the events are concurrent.
- **Causal Relationships:** If an event e1 with vector clock [1,2,0] causally affects an event e2 with vector clock [1,2,1], the relationship VC(e1) < VC(e2) will hold. This explicitly shows that e1 happened before e2.

Advantages of Vector Clocks

1. Precise Ordering: They clearly identify whether events are causally related or concurrent.

2. Concurrent Event Representation: By recognizing incomparable vector timestamps, vector clocks explicitly represent concurrency, which Lamport timestamps cannot. **3. Better Event Tracking:** Each process maintains a history of its interactions with other processes, enabling more accurate tracking of dependencies.