THE UNIVERSITY OF DODOMA

COLLEGE OF INFORMATICS AND VIRTUAL EDUCATION

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (CSE)

GROUP ASSIGNMENT

COURSE NAME: DISTRIBUTED COMPUTER

COURSE CODE: CP 314

PROGRAM: Bsc.CNISE

GROUP PARTICIPANTS

NAMES	REGISTRATION NUMBER
GABRIEL MBETI	T22-03-02516
HENRY SELEMAN	T22-03-00295
PUDENCIANA PETER	T22-03-11584
AMANI MLIMANDOLA	T22-03-05155
IBRAHIMU BONIFACE	T22-03-05565
JIMSON MAHENGE	T22-03-08372
PETRO MOHERE	T22-03-02116
GEORGE KALAMU	T22-03-08400
NASSIBU ALLY	T22-03-01214
EMMANUEL WOLFGANG	T22-03-09178

QUESTION ONE

a. Numerical Deviation Bound

Scenario: A product with an initial stock of 100 units experiences a reduction of 15 units at the primary server.

Replica A: Reflects 85 units immediately.

Replica B: Still shows 100 units (no update received).

Replica C: Shows 95 units (partial update).

i. Identifying Violations:

Replica A: 85 units → Deviation = |85 - 85| = 0, (No violation).

Replica B: 100 units → Deviation = |100 - 85| = 15, (Violation, exceeds given limit of ±10).

Replica C: 95 units → Deviation = |95 - 85| = 10, (No violation, Since it is within the limit of ±10).

Therefore, replica B violate the numerical deviation bound.

ii. Strategy to Ensure Compliance:

a. Synchronous Updates for Critical Data: Implement partial synchronization for high-impact transactions.

b. Adaptive Replication Frequency: Increase update frequency for critical replicas based on deviation trends.

c. Conflict Resolution Algorithms: Employ reconciliation methods to automatically detect and correct deviations.

b. Staleness Bound

Scenario: A price update changes from $20 to $25 at T=0.

Replica A: Receives at T=2 seconds.

Replica B: Receives at T=6 seconds.

Replica C: Receives at T=8 seconds.

i. Identifying Violations:

Replica A: T=2 → Within 5 seconds (No violation)

Replica B: T=6 → Exceeds 5 seconds (Violation since received 1second later)

Replica C: T=8 → Exceeds 5 seconds (Violation since received 3 seconds later)

Therefore Replica B and Replica C violate the staleness bound.

ii. Impact of Staleness:

a. Customer Misinformation: Users may see outdated prices, leading to dissatisfaction.

b. Revenue Loss: Transactions based on old prices can result in financial discrepancies.

c. Operational Inconsistencies: Challenges in synchronizing promotions, discounts, and stock levels.

c. Order Deviation

Scenario: Updates occur as follows:

T1: Price from $20 to $25

T2: Stock from 100 to 90 units

T3: Price from $25 to $30

Replica A: Processes T1, T2, T3 (Correct order).

Replica B: Processes T2, T1, T3.

Replica C: Processes T1, T3, T2.

i. Order Deviation Calculation:

- **Replica A: 0 deviations (Correct order)**
- **Replica B: T2 before T1 = 1 deviation (Within limit)**
- **Replica C: T3 before T2 = 1 deviation (Within limit)**

ii. Mechanism to Minimize Deviations:

a. Vector Clocks: Track causality between events.

b. Total Order Broadcast Protocols: Ensure updates are processed in a consistent sequence.

c. Version Control Mechanisms: Implementing strong versioning to track and correct out-of-order updates.

QUESTION TWO

a. Eventual consistency is a relaxed data consistency model used in distributed systems. It guarantees that if no new updates are made to a replicated dataset, all replicas will eventually converge to the same state. This model supports high availability and fault tolerance by allowing temporary inconsistencies during periods of network latency or partitioning.

b. Convergence Mechanisms in Distributed Systems

The system ensures eventual convergence through:

i. Asynchronous Updates: Updates are propagated in the background without waiting for immediate acknowledgment from all replicas, improving responsiveness.

ii. Conflict Resolution Strategies: Techniques such as Last-Write-Wins (LWW), version vectors, and application-specific logic resolve concurrent updates.

Iii. Propagation Protocols:

Gossip Protocols: Replicas periodically exchange state information.

Anti-Entropy Protocols: Nodes compare and synchronize their states to eliminate discrepancies.

Pub-Sub Systems: Updates are broadcasted to subscribers to ensure consistent changes.

c. Real-World Impact of Temporary Inconsistencies and Mitigation Strategies

i. Impact: Temporary inconsistencies can negatively affect user experience. For example:

a) A user may see a comment before the original photo is loaded.

b) Like counts might differ temporarily across servers.

ii. Mitigation Strategies:

a) Read Repair: During read operations, discrepancies are detected and corrected.

b) Quorum-Based Replication: Requires a majority of replicas to agree on data values before responding to requests.

c) Client-Side Caching with Expiry: Reduces stale data by enforcing data refresh policies.

c) UI Design Adjustments: Designing user interfaces to gracefully handle temporary inconsistencies (e.g., showing "loading" indicators).

d. Trade-offs of Eventual Consistency

i. Performance: High performance due to reduced synchronization overhead and low latency.

ii. Availability: High availability, as operations are not blocked by network delays or replica failures.

iii. Data Accuracy: Potential for temporary inconsistencies, requiring conflict resolution mechanisms.

iv. Scalability: Suitable for large, distributed systems as it minimizes the need for global coordination.

e. Other Data-Centric Consistency Models

i. Causal Consistency: Ensures operations with cause-effect relationships are observed in the same order by all replicas.

Example: A photo appears before its comment.

Strength: Maintains causality without strict ordering.

Limitation: Requires dependency tracking.

ii. Grouping Consistency: Guarantees consistent updates within logically grouped data sets, often enforced through exclusive locks.

Example: Order and payment in an e-commerce system appear together.

Strength: Maintains logical data integrity.

Limitation: Defining dynamic group boundaries can be challenging.

iii. Sequence Consistency: Ensures all replicas apply updates in the same global order as initiated.

Example: A user's actions (like, comment, share) appear in the same sequence.

Strength: Predictable and intuitive behavior.

Limitation: Requires global coordination, leading to high latency.

QUESTION THREE

a. The Bully Algorithm and Its Role in Synchronization

The Bully Algorithm is a well-known leader election algorithm designed for distributed systems. It ensures that the process with the highest priority (or ID) becomes the leader, thus coordinating tasks and maintaining synchronization among processes.

How the Bully Algorithm Works:

i. Initiation: When a process detects that the current leader has failed (usually through timeout mechanisms), it initiates an election.

ii. Election Messages: The initiating process sends an "election" message to all processes with higher IDs.

iii. Responses: If a higher-ID process responds, it takes over the election, sending its own election messages to even higher-ID processes.

iv. Leader Declaration: If no higher-ID process responds, the initiating process declares itself the leader and broadcasts a "coordinator" message to all processes.

Role in Synchronization:

i. Centralized Control: The leader coordinates resource access, preventing conflicts and ensuring ordered task execution.

ii. Fault Tolerance: The algorithm quickly detects leader failures and elects a new leader to restore synchronization.

iii. Efficiency: The process ensures only one leader exists at a time, reducing overhead and confusion in resource management.

b. Failure Recovery Mechanisms for Maintaining Synchronization

Failure recovery is critical to maintain system synchronization when a process or leader fails. The following mechanisms are commonly used:

i. Failure Detection:

Heartbeat Messages: Processes periodically send heartbeat signals to confirm they are active.

Timeout Mechanisms: If a heartbeat is not received within a specified time, the process is considered failed.

ii. Re-election of Leader: When a leader fails, the Bully Algorithm or similar leader election protocols are triggered to elect a new leader.

iii. State Synchronization:

Checkpointing : Processes periodically save their state. After recovery, they can resume from the last checkpoint to minimize data loss.

Log-Based Recovery: Processes maintain logs of operations. Upon recovery, logs are replayed to restore the system state.

iv. Redundancy and Replication:

Backup Leaders: A secondary leader (backup) can take over immediately if the primary leader fails.

Data Replication: Critical data is replicated across multiple nodes to ensure availability during failures.

v. Consensus Protocols

Paxos or Raft: These protocols ensure that a majority of nodes agree on system decisions, even in the presence of failures.

QUESTION FOUR

a. Is Environmental Monitoring with WSNs an Example of Distributed Computing?
Yes, environmental monitoring using Wireless Sensor Networks can be considered an example of distributed computing. Here are the reasons:

i. Decentralized Data Processing: Each sensor node independently collects environmental data, processes it locally, and transmits the relevant

information to a central base station. This decentralized approach is a core characteristic of distributed systems.

ii. Resource Sharing: Sensor nodes share network resources such as communication channels, energy (battery life), and processing power to achieve a common goal—monitoring environmental conditions.

iii. Fault Tolerance: Distributed computing principles enhance system reliability. Even if some sensor nodes fail, the system continues functioning using data from operational nodes.

iv. Scalability: The system can easily expand by adding more sensor nodes without significant changes to the existing infrastructure, a hallmark of distributed systems.

b. Reference Broadcast Synchronization (RBS) Mechanism: Reference Broadcast Synchronization (RBS) is a widely used protocol in WSNs for maintaining time synchronization among sensor nodes. Unlike traditional time synchronization methods that rely on timestamping the sending and receiving times, RBS focuses on the reception time of broadcast messages.

How RBS Works:

i. **Reference Node Broadcast:** A reference node (or base station) periodically sends out a reference broadcast message without attaching a timestamp.

ii. **Reception and Timestamping:** All sensor nodes within the broadcast range receive the message nearly simultaneously. Upon receiving the broadcast, each node records the local timestamp of when the message was received.

iii. **Exchange of Timestamps:** The receiving nodes then exchange their recorded timestamps with each other.

iv. **Offset Calculation:** Each node calculates the clock offset relative to the other nodes based on the differences in their reception timestamps. This process helps the nodes synchronize their clocks.