**CS3551-DISTRIBUTED COMPUTING**

**PART C (15 Marks)**

**UNIT-1 INTRODUCTION**

**Q1: Explain the Model of Distributed Computations with respect to Distributed Programs and Models of Communication Networks.**

**1. Distributed Programs**

A **Distributed Program** consists of multiple processes running on different machines and communicating via a network.

**2. Model of Distributed Executions**

A **distributed execution** consists of:

1. **Processes** – Independently executing tasks.
2. **Events** – Actions occurring in each process.
3. **Causal Ordering** – Ensuring events happen in a logical sequence.

**3. Models of Communication Networks**

1. **Synchronous Network** – Message delays are bounded.
2. **Asynchronous Network** – No bound on message delays.
3. **Partially Synchronous** – Assumes some timing constraints.

**Example:**

In a **stock trading system**, multiple stock exchanges update prices simultaneously and need efficient communication.

**Q2: Describe the design issues in Distributed Computing and how they impact system performance.**

1. **Transparency**
   * **Access Transparency:** Users shouldn’t worry about whether resources are local or remote.
   * **Replication Transparency:** Multiple copies of data should appear as one.
   * **Failure Transparency:** System should recover automatically.
2. **Fault Tolerance**
   * Techniques like **checkpointing, redundancy, and message logging** help in failure recovery.
3. **Security**
   * **Data Encryption** for secure communication.
   * **Authentication** to verify users.
4. **Concurrency Control**
   * Avoids **race conditions** using **locking mechanisms**.
5. **Scalability**
   * Distributed Systems must handle increasing load using **load balancing and caching**.

**Example:**

* **Google’s Search Engine** uses a highly scalable distributed system for handling billions of queries.

**Q3: Explain the role of Primitives for Distributed Communication and their types.**

**1. Communication Primitives**

Distributed systems use **communication primitives** to exchange messages between nodes.

**2. Types of Communication Primitives**

1. **Send & Receive** – Basic message exchange.
2. **Remote Procedure Call (RPC)** – Function execution on a remote machine.
3. **Remote Method Invocation (RMI)** – Object-oriented equivalent of RPC.
4. **Message Queues** – Used for asynchronous communication.
5. **Publish-Subscribe Model** – Nodes publish messages that other nodes subscribe to.

**Example:**

* **WhatsApp** uses asynchronous message queues to ensure reliable message delivery.

**UNIT-II LOGICAL TIME AND GLOBAL CLOCK**

**Q1: Describe the framework for Logical Clocks and explain the concept of Vector Clocks with an example.**

**Framework for Logical Clocks**

Logical clocks provide **a way to order events** without relying on physical time.

**Vector Clocks (VC)**

* Each process maintains a **vector of timestamps** (one per process).
* **Rules:**
  1. **Increment local timestamp** before an event.
  2. When sending a message, attach **vector clock**.
  3. On receiving a message, update **each element as max(local, received) + 1**.

**Example:**

Assume three processes:

**Initial State:**

makefile

CopyEdit

P1: (1,0,0)

P2: (0,1,0)

P3: (0,0,1)

If P1 sends a message to P2, its timestamp updates:

yaml

CopyEdit

P1: (2,0,0) → Sends → P2 receives → P2: (2,2,0)

**Uses:**

* Detecting **causality and concurrent events**.
* Used in **distributed debugging and consistency protocols**.

**Q2: Discuss the importance of Global State in Distributed Systems. How is it achieved?**

**Importance of Global State**

* Global state represents the **entire system’s status** at a point in time.
* Used for **fault tolerance, checkpointing, and deadlock detection**.

**Challenges in Capturing Global State**

* No **global clock** → Events occur asynchronously.
* Different nodes may have **inconsistent views**.

**Techniques to Capture Global State**

1. **Chandy-Lamport Algorithm** → Works for FIFO channels.
2. **Consistent Snapshot Protocols** → Ensure no partial snapshots are taken.

**Example:**

* **Banking Transactions** must maintain a consistent **global state** to prevent fraud.

**Q3: Explain Total Order and Causal Order in Group Communication. How are they implemented?**

**1. Causal Order**

* Ensures that **causally related** messages arrive in the same order.
* Uses **Vector Clocks** to maintain ordering.

**Example:**

pgsql

CopyEdit

P1 sends M1 → P2 sends M2 → All processes see M1 before M2.

**2. Total Order**

* Ensures **all messages arrive in the same order** at all processes, even if they are unrelated.
* Implemented using **Centralized or Distributed Agreement Protocols**.

**Example:**

* **Stock Market Orders** must follow **total order** to maintain fairness.

**Implementation Techniques**

1. **Sequencer-Based Approach** → A central sequencer assigns timestamps.
2. **Distributed Consensus (Paxos, Raft)** → All nodes agree on message order.

**Comparison Table**

| **Feature** | **Causal Order** | **Total Order** |
| --- | --- | --- |
| **Ensures** | Only causally related messages are ordered. | All messages are ordered. |
| **Complexity** | Lower | Higher |
| **Implementation** | Vector clocks | Consensus algorithms |

**UNIT-III DISTRIBUTED MUTEX AND DEADLOCKS**

**Q1: Explain Ricart-Agrawala’s Algorithm for Distributed Mutual Exclusion.**

**Ricart-Agrawala Algorithm Steps:**

1. A process sends a **request message (REQ)** to all nodes.
2. Other processes reply with:
   * **ACK** if they are not in the critical section.
   * **DEFER** if they are already inside the CS.
3. When the requesting process receives **all ACKs**, it **enters the critical section**.
4. After exiting, it sends **release messages** to all deferred requests.

**Advantages:**

✔ Reduces the number of **messages exchanged** compared to Lamport’s Algorithm.

**Example:**

* Used in **distributed databases** to handle concurrent transactions.

**Q2: Describe the Chandy-Misra-Haas Algorithm for Deadlock Detection.**

**Algorithm for the AND Model:**

1. Each process sends a **probe message** along dependency edges.
2. If a process receives its own probe, a **deadlock is detected**.
3. The process with the **highest priority** terminates to break the deadlock.

**Algorithm for the OR Model:**

* Uses **wait-for graphs (WFG)** to detect **circular wait conditions**.

**Example:**

* Used in **banking transactions** to detect deadlocks in multi-user environments.

**Q3: Compare Lamport’s, Ricart-Agrawala’s, and Token-Based Mutual Exclusion Algorithms.**

| **Algorithm** | **Type** | **Messages Exchanged** | **Failure Handling** | **Fairness** |
| --- | --- | --- | --- | --- |
| **Lamport’s** | Timestamp-based | **3N–1** | Difficult | Moderate |
| **Ricart-Agrawala** | Permission-based | **2(N–1)** | Better than Lamport | Good |
| **Token-Based** | Token circulation | **1 per entry** | Token loss issue | Excellent |

**Example:**

* **Lamport’s** → Used in **banking transactions**.
* **Ricart-Agrawala’s** → Used in **distributed databases**.
* **Token-Based** → Used in **Ethernet networks**.

**UNIT-IV CONSENSUS AND RECOVERY**

**Q1: Explain the Different Types of Agreement Protocols in Synchronous Systems with Failures.**

**Agreement Protocols in Synchronous Systems with Failures**

1. **Byzantine Agreement Protocol**
   * Handles up to **(n-1)/3 faulty nodes**.
   * Requires **multiple rounds of message exchange**.
   * Used in **blockchain networks**.
2. **Fail-Stop Agreement Protocol**
   * Assumes nodes fail but do not behave maliciously.
   * Nodes **detect failures** and reconfigure.
   * Used in **real-time systems**.
3. **Consensus using Majority Voting**
   * If **majority of nodes agree**, decision is finalized.
   * Used in **distributed databases**.

**Example:**

* **Google Spanner** uses **Paxos algorithm** for distributed consensus.

**Q2: Compare Coordinated vs Asynchronous Checkpointing in Distributed Systems.**

| **Feature** | **Coordinated Checkpointing** | **Asynchronous Checkpointing** |
| --- | --- | --- |
| **Synchronization** | Requires global coordination. | No global synchronization. |
| **Overhead** | Higher due to coordination. | Lower overhead. |
| **Recovery Complexity** | Simpler as all nodes rollback together. | More complex due to rollback dependencies. |
| **Performance Impact** | Causes delays during checkpointing. | Minimal impact. |
| **Example** | Banking transactions. | Cloud applications. |

**Q3: Explain the Algorithm for Asynchronous Checkpointing and Recovery with an Example.**

**Algorithm Steps:**

1. Each process **periodically saves** its local state.
2. A process recovers from failure using **its latest checkpoint**.
3. Messages received after checkpointing are **logged separately**.
4. After recovery, the process **replays logged messages** to restore state.

**Example:**

* **Google Docs auto-save feature** → Saves document progress in the background.

**Advantages:**

✔ Less synchronization required.  
✔ Improves system efficiency.

**Disadvantages:**

❌ Higher risk of **rollback propagation**.

**UNIT-V CLOUD COMPUTING**

**Q1: Explain Load Balancing, Scalability, and Elasticity in Cloud Computing.**

**1. Load Balancing:**

* Distributes incoming traffic across multiple servers.
* Prevents **overloading** and ensures availability.
* **Example:** AWS Elastic Load Balancer (ELB).

**2. Scalability:**

* Ability to **increase/decrease** resources based on demand.
* **Horizontal scaling:** Adding more servers.
* **Vertical scaling:** Increasing capacity of existing servers.
* **Example:** Netflix scales up during peak streaming hours.

**3. Elasticity:**

* **Automatic scaling** based on workload.
* Ensures optimal performance and cost-efficiency.
* **Example:** Cloud auto-scaling in AWS, Azure.

**Q2: What is Replication in Cloud Computing? Explain Its Benefits.**

**Definition:**

Replication is the process of **creating multiple copies** of data across cloud servers to improve availability and reliability.

**Types of Replication:**

1. **Data Replication** → Copies of data are stored across locations.
2. **Database Replication** → Ensures backup and fault tolerance.
3. **Virtual Machine Replication** → Redundant VMs for disaster recovery.

**Benefits of Replication:**

✔ **High Availability** → Data is always accessible.  
✔ **Fault Tolerance** → Reduces impact of server failures.  
✔ **Improved Performance** → Users access data from the nearest location.

**Example:**

* Google Drive replicates user files across multiple data centers.

**Q3: Describe Cloud Compute Services, Storage Services, and Application Services with Examples.**

**1. Compute Services:**

* Provide virtual machines and processing power.
* **Example:**
  + **AWS EC2** → Virtual servers for hosting applications.
  + **Google Compute Engine** → Runs workloads on Google's infrastructure.

**2. Storage Services:**

* Cloud storage solutions for data management.
* **Example:**
  + **Amazon S3** → Scalable object storage.
  + **Google Cloud Storage** → Secure file storage.

**3. Application Services:**

* Cloud-hosted applications for users.
* **Example:**
  + **Google Docs** → Cloud-based document editing.
  + **Salesforce** → Customer relationship management (CRM).