UNIT-I: Introduction to Distributed Systems

1. Introduction to Distributed Systems:

1. **Definition**: A system with multiple independent computers working together.
2. **Goal**: Share resources, increase speed, reliability, and scalability.
3. **Coordination**: Each computer communicates to perform tasks.
4. **Transparency**: Users see it as one system, not separate computers.
5. **Types**: Includes distributed computing, databases, and file systems.
6. **Challenges**: Security, fault tolerance, and maintaining consistency.
7. **Decentralization**: No single central control, all systems work independently.
8. **Efficiency**: Faster processing by dividing tasks among multiple machines.
9. **Examples**: Internet, cloud computing, and large-scale corporate networks.
10. **Applications**: Useful in banking, stock trading, e-commerce, etc.

2. Characteristics of Distributed Systems:

1. **Scalability**: Can easily add more machines to grow the system.
2. **Fault Tolerance**: Can handle machine failures without losing data.
3. **Concurrency**: Many tasks happen at the same time.
4. **Transparency**: Users see the system as a whole, not individual computers.
5. **Resource Sharing**: Systems can share files, databases, and processing power.
6. **Heterogeneity**: Different types of machines can work together.
7. **Openness**: Easy to add new machines and resources.
8. **Latency**: Delays in data transfer between computers.
9. **Security**: Ensures data protection across machines.
10. **Coordination**: Machines must work together smoothly.

3. Examples of Distributed Systems:

1. **Client-Server**: Users (clients) request services from central servers.
2. **Peer-to-Peer (P2P)**: All computers (peers) are equal and share resources.
3. **Grid Computing**: Connects multiple computers to solve large problems.
4. **Cloud Computing**: Access services over the internet, like storage and software.
5. **Cluster Computing**: Groups of computers working together as one.
6. **Internet**: Biggest example, connects millions of systems globally.
7. **File Sharing**: P2P systems like BitTorrent for sharing files.
8. **Online Gaming**: Distributed servers handling many players in real time.
9. **E-commerce**: Platforms like Amazon using distributed systems for transactions.

10. **Social Media**: Systems like Facebook use distributed servers for managing users.

4. Advantages of Distributed Systems:

1. **Increased Reliability**: If one machine fails, others continue working.
2. **Scalability**: Easily add more machines to handle more work.
3. **Efficiency**: Tasks get completed faster by sharing work among machines.
4. **Resource Sharing**: Systems can share data and computing power.
5. **Cost Effective**: Use regular computers instead of expensive supercomputers.
6. **Geographical Spread**: Machines can be in different locations, but work together.
7. **Availability**: System works 24/7 without downtime.
8. **Flexibility**: Can adapt to new tasks and resources easily.
9. **Security**: Distributed systems can be more secure by splitting data.
10. **Data Backup**: Keeps multiple copies of data for safety.

5. System Models:

1. **Architectural Models**: Describes how components of the system are arranged (e.g., client-server).
2. **Fundamental Models**: Explains basic system behaviors (e.g., security, reliability).
3. **Network Models**: Describes how data is transferred between machines.
4. **Interaction Models**: Describes communication between components.
5. **Security Models**: Describes ways to protect the system from attacks.
6. **Performance Models**: Measures how efficiently the system works.
7. **Failure Models**: Describes what happens if a machine fails.
8. **Transparency Models**: Makes the system look like a single entity to users.
9. **Consistency Models**: Ensures data stays the same across all machines.
10. **Concurrency Models**: Describes how tasks are done at the same time.

6. Networking and Internetworking:

1. **Definition**: Connecting multiple systems to share data.
2. **Types of Networks**: LAN (Local), WAN (Wide Area), MAN (Metropolitan).
3. **Internet**: The largest network connecting distributed systems globally.
4. **Protocols**: Rules for data transfer (e.g., TCP/IP, HTTP).
5. **Routers**: Direct data between different networks.
6. **IP Address**: Identifies each machine in a network.
7. **Firewalls**: Protect networks from unauthorized access.
8. **Network Latency**: Delays in data travel between machines.
9. **Bandwidth**: Amount of data that can be transferred.

10. **Interconnection**: Connecting different networks to form bigger networks.

7. Interprocess Communication (IPC):

1. **Message Passing**: Processes send and receive messages to share data.
2. **Shared Memory**: Processes share a common memory space to communicate.
3. **Synchronous vs Asynchronous**: Synchronous waits for a reply, asynchronous does not.
4. **Remote Procedure Call (RPC)**: Allows processes to run code on other systems.
5. **Pipes and Sockets**: Tools for sending messages between processes.
6. **Security**: Ensuring data is safely transferred between systems.
7. **Efficiency**: Reducing delays and resource usage during communication.
8. **Buffering**: Storing messages temporarily during transfer.
9. **Deadlock**: When processes get stuck waiting for each other.
10. **Semaphores**: Mechanisms to control access to shared memory.

8. Distributed Objects and Remote Method Invocation (RMI):

1. **Definition**: Objects located on different systems communicate with each other.
2. **RMI**: Allows Java objects to call methods on remote systems.
3. **Serialization**: Converts data into a format suitable for sending over a network.
4. **Stubs and Skeletons**: Intermediate programs that help in communication.
5. **JNDI**: A service that helps in finding and using remote objects.
6. **RMI Registry**: Stores information about remote objects.
7. **Security**: Protects data when calling remote methods.
8. **Garbage Collection**: Automatically cleans up unused objects.
9. **Performance**: Managing the speed and efficiency of remote calls.
10. **Applications**: Used in building complex web-based systems.

9. RPC (Remote Procedure Call):

1. **Definition**: A system that lets programs run code on other systems.
2. **Stub Programs**: The local part that interacts with the remote program.
3. **Security**: Protecting the data sent between machines.
4. **Message Formatting**: Converting messages into a format both systems understand.
5. **Latency**: Reducing delays when calling remote functions.
6. **Asynchronous Calls**: The system can keep working while waiting for a reply.
7. **Failures**: Handling errors when communication fails.
8. **ID Management**: Identifying and matching requests with replies.

9. **Performance**: Optimizing the speed of remote calls.
10. **Applications**: Used in network file systems, web services, etc.

10. Case Study – Java RMI:

1. **Definition**: Allows Java apps to communicate with remote objects.
2. **Components**: Stubs, skeletons, and an RMI registry to locate objects.
3. **Use**: Suitable for building networked applications.
4. **Serialization**: Automatically sends data between systems.
5. **Security**: Provides security features like encryption.
6. **Exception Handling**: Handles remote failures easily.
7. **Naming Service**: Helps in finding remote objects using names.
8. **Performance**: Can be optimized by reducing unnecessary remote calls.
9. **Setup**: Requires setting up port numbers and security policies.
10. **Limits**: Network delays can affect performance.

UNIT-II: Synchronization

1. Time and Global States:

1. **Global State**: Represents the overall condition of the system.
2. **Snapshot**: A way to capture the current state of all machines without stopping them.
3. **Causality**: How events affect each other in distributed systems.
4. **Logical Time**: Time based on events, not the actual clock.
5. **Physical Time**: Real-world time used to sync events.
6. **NTP**: A protocol for syncing clocks over the internet.
7. **Logical Clocks**: A way to keep track of events without using physical clocks.
8. **Global State Debugging**: Helps find problems in distributed systems.
9. **Consistency**: Ensuring all systems have the same state.
10. **Timestamps**: Used to record the time of events.

2. Logical Clocks:

1. **Definition**: A way to order events in a system without real clocks.
2. **Lamport Timestamps**: A method to give each event a unique number.
3. **Vector Clocks**: Track relationships between multiple events.
4. **Synchronizing Events**: Ensuring all events happen in the correct order.
5. **Concurrency**: Detecting events that happen at the same time.
6. **Implementation**: Steps to add logical clocks to a system.
7. **Use**: Applied in systems like databases and cloud apps.
8. **Limit

ations**: Cannot detect all causal relationships. 9.** Complexity**: Adds extra steps to event handling. 10.** Applications**: Used in debugging and

performance tracking.

3. Distributed Mutual Exclusion:

1. **Goal**: Ensure only one process uses a resource at a time.
2. **Token-based Algorithms**: A token is passed around to grant access.
3. **Permission-based Algorithms**: Processes request access from others.
4. **Ricart-Agrawala Algorithm**: A simple permission-based mutual exclusion protocol.
5. **Token Ring**: A token circulates between machines, and the one holding it can access the resource.
6. **Failures**: Handling cases where processes or messages fail.
7. **Fairness**: Ensuring all processes get a chance to access the resource.
8. **Efficiency**: Reducing the number of messages needed to coordinate access.
9. **Deadlock**: Avoiding situations where processes are stuck waiting for each other.
10. **Starvation**: Preventing processes from being denied access forever.

4. Elections:

1. **Definition**: Choosing a leader or coordinator among distributed processes.
2. **Bully Algorithm**: The process with the highest ID becomes the leader.
3. **Ring Algorithm**: Processes form a circle, and the highest ID wins.
4. **Failures**: Handling failures during the election process.
5. **Fairness**: Ensuring that all processes have a chance to participate.
6. **Efficiency**: Reducing the number of messages needed to elect a leader.
7. **Termination**: Ensuring the election process completes in a reasonable time.
8. **Backup Leaders**: Keeping backup leaders in case the main leader fails.
9. **Use Cases**: Used in databases, cloud computing, and distributed applications.
10. **Applications**: Helps manage resources, coordinate tasks, and prevent conflicts.

5. Multicast Communication:

1. **Definition**: Sending a message to multiple receivers at once.
2. **Group Communication**: Processes are organized into groups for communication.
3. **Reliable Multicast**: Ensuring all members get the message, even if some fail.
4. **Total Ordering**: Ensuring all processes receive messages in the same order.
5. **Efficiency**: Reducing the number of messages needed to communicate.
6. **Fault Tolerance**: Handling failures of receivers or the sender.

7. **Applications**: Used in video streaming, online gaming, and cloud services.
8. **Tree-based Protocols**: Organizing processes in a tree to send messages efficiently.
9. **Acknowledgment**: Receivers send back acknowledgments to confirm receipt.
10. **Security**: Encrypting messages to protect against unauthorized access.

6. Consensus Protocols:

1. **Definition**: Agreement among distributed processes on a single value.
2. **Paxos Algorithm**: A popular protocol for reaching consensus.
3. **Raft Algorithm**: A simpler and easier-to-understand alternative to Paxos.
4. **Quorum**: A majority of processes must agree before making a decision.
5. **Fault Tolerance**: Handling failures without disrupting the consensus.
6. **Message Complexity**: Reducing the number of messages required for consensus.
7. **Applications**: Used in cloud systems, databases, and financial systems.
8. **Performance**: Balancing speed and reliability.
9. **Leader-based Consensus**: A leader coordinates the agreement process.
10. **Use Cases**: Essential in distributed databases, cloud computing, and blockchain systems.

This should help simplify and cover the core points for each topic!