1. *Byzantine Failure/Fault:*

   - *Definition:* A Byzantine failure or fault refers to a type of failure in a distributed system where a component (node or process) behaves arbitrarily, providing incorrect or malicious responses to other components.

2. *Byzantine Process:*

   - *Definition:* A Byzantine process is a component (node or process) in a distributed system that may exhibit arbitrary and malicious behavior, such as sending incorrect information or intentionally trying to disrupt the normal functioning of the system.

Consensus:

- *Meaning:* Consensus in distributed systems refers to the agreement among a group of processes or nodes in the system to reach a common decision or value, even in the presence of faults or failures.

If an algorithm solves consensus for f failed (crashing) processors we say it is: an f-resilient consensus algorithm

1. *Rollback Recovery Complexity:*

   - *Reason:* Rollback recovery in distributed systems is complicated due to the challenges of coordinating and ensuring consistent states across multiple processes, dealing with varying execution speeds, and handling the potential for message losses or failures.

2. *Livelock and Domino Effect:*

   - *Livelock:* In checkpointing and rollback recovery, livelock occurs when processes are repeatedly rolling back and re-executing the same set of actions without making progress.

   - *Domino Effect:* Domino effect refers to the situation where the rollback of one process triggers rollbacks in other processes, creating a cascading effect and potentially prolonging the recovery process.

WFG (Wait-for-Graph):

- *Purpose:* The Wait-for-Graph (WFG) is used in distributed systems and concurrency control to represent and analyze the dependencies and relationships between different processes or transactions waiting for resources. It helps in detecting and resolving deadlocks in a system.

Distributed Systems:

- *Definition:* A distributed system is a collection of independent computers or nodes that work together as a unified system, sharing resources and coordinating tasks across a network.

Feature	Synchronous Checkpoint Algorithm	Asynchronous Checkpoint Algorithm
Timing of Checkpoints	**Synchronous:** Coordinated checkpoints are taken at the same time across all processes.	**Asynchronous:** Checkpoints are taken independently by each process without coordination.
Communication Overhead	**Synchronous:** Requires communication and coordination among processes for simultaneous checkpoints.	**Asynchronous:** Involves less communication overhead as checkpoints are taken independently.
System Overhead	**Synchronous:** May introduce system-wide synchronization, potentially causing increased overhead.	**Asynchronous:** Typically results in lower system-wide synchronization overhead.
Implementation Complexity	**Synchronous:** Coordination makes the implementation more complex.	**Asynchronous:** Simpler to implement due to independence in checkpoint initiation.
Checkpoint Rollback Efficiency	**Synchronous:** Provides better efficiency in terms of rollback since all processes have consistent checkpoints.	**Asynchronous:** Rollback efficiency may vary as checkpoints are not necessarily consistent across processes.

1. *Peer-to-Peer Computing:*

- *Definition:* Peer-to-peer computing is a decentralized network architecture where individual nodes (peers) collaboratively share resources, services, or information directly with each other without relying on a central server.

2. *Overlay Graphs:*

- *Definition:* Overlay graphs in the context of peer-to-peer networks refer to the logical connections established between nodes, often to enhance communication efficiency or support specific functionalities without directly mirroring the physical network topology.

3. *Chord:*

- *Definition:* Chord is a distributed hash table (DHT) protocol used in peer-to-peer systems. It provides a scalable and efficient way to locate data in a decentralized manner by maintaining a circular ring of nodes, each responsible for a specific range of keys.

4. *Content Addressable Networks (CAN):*

- *Definition:* Content Addressable Networks are peer-to-peer systems that use multidimensional coordinate spaces to represent and locate content. Nodes in the network are responsible for a specific region in the coordinate space, and content is addressed based on its coordinates.

5. *Tapestry:*

- *Definition:* Tapestry is a peer-to-peer overlay network designed for efficient and fault-tolerant routing. It employs a distributed hash table (DHT) approach and uses routing tables to ensure reliable and scalable content location and retrieval in a decentralized environment.

1. *Napster:*

- *Definition:* Napster was an early peer-to-peer (P2P) file-sharing system that gained popularity in the late 1990s. It allowed users to share and download music files directly from each other's computers. Napster used a centralized server to maintain an index of available files.

2. *Gnutella:*

- *Definition:* Gnutella is a decentralized peer-to-peer network protocol used for file sharing. Unlike Napster, Gnutella does not rely on a central server. Instead, users connect directly to each other, creating a distributed network where files can be shared and searched across multiple computers.

1. *Causally Ordered Events:*

- *Definition:* Causally ordered events in distributed systems refer to a sequencing of events based on their causal relationships. If event A causally precedes event B, A is considered to have happened before B in the system.

2. *Consistent vs. Inconsistent Global States:*

- *Consistent Global State:* A global state is consistent if it could have been reached by some valid sequence of events in the system.

- *Inconsistent Global State:* A global state is inconsistent if it cannot be reached by any valid sequence of events, possibly due to message delays or concurrency issues.

3. *Transit Messages in Recording Global States:*

- *Reason:* Allowing transit messages during global state recording captures the potential causal relationships between events, ensuring accurate representation of the system's progress.

4. *Additional Vector in Causal Ordering for Multi-cast:*

- *Reason:* In a multi-cast environment, additional vectors are used to track the causal relationships between events across different multicast groups, ensuring accurate causal ordering within each group.

5. *Phantom Deadlocks:*

- *Reason:* Phantom deadlocks occur due to variations in the timing of resource allocation and deallocation messages. They are not real deadlocks but can lead to unnecessary system interventions.

6. *Idle Token:*

- *Definition:* An idle token is a token in distributed systems that signifies that a process is currently not in need of a particular resource or permission, indicating its idle state.

7. *Orphaned and Lost Messages:*

- *Orphaned Messages:* Messages that are sent but never received or processed by any recipient.

- *Lost Messages:* Messages that are sent but do not reach their intended destination due to network issues or other failures.

8. *Log-Based Checkpointing:*

- *Definition:* Log-based checkpointing involves periodically recording the system's state by storing key information in a log. In case of a failure, the system can be restored to a consistent state using the logged information.

9. *P2P vs. Client-Server Model:*

- *P2P Model:* Peer-to-peer model involves decentralized communication, where nodes communicate directly with each other.

- *Client-Server Model:* In a client-server model, clients request services or resources from a centralized server.

10. *Distributed Shared Memory:*

- *Definition:* Distributed Shared Memory (DSM) is a model in distributed systems where physically separate memory locations are treated as a single shared memory space, allowing processes to access and modify shared data.

Strict Consistency in Memory Consistency Models:

- *Definition:* Strict consistency is a memory consistency model that imposes the most stringent requirements on the order of memory operations in a parallel or distributed system. In a system adhering to strict consistency:

  1. *Global Time Order:* All memory operations appear to be instantaneously applied at a global, single, and consistent point in time.

  2. *Consistent View:* Every processor observes the same order of operations. This means that if one processor performs write A before write B, all other processors must also observe A before B.

  3. *Instantaneous Visibility:* Changes made by one processor become instantaneously visible to all other processors.

- *Synchronization Cost:*

  - Achieving strict consistency often requires extensive synchronization and coordination between processors, leading to increased communication overhead and potentially impacting system performance.

- *Implementation Challenges:*

  - Implementing strict consistency can be challenging in distributed systems due to issues like network delays, varying clock speeds, and the need for precise synchronization.

- *Use Cases:*

  - Strict consistency is seldom used in practical systems due to its high synchronization cost. It is more suitable for scenarios where the need for perfect consistency outweighs performance considerations, such as certain critical systems in avionics or nuclear power plants.

- *Alternative Models:*

  - Many distributed systems opt for weaker consistency models, such as sequential consistency or eventual consistency, to strike a balance between performance and the need for consistency in practice. These models relax some of the strict requirements to improve overall system efficiency.

Algorithmic Challenges in Distributed Computing:

1. *Synchronization:*

   - *Challenge:* Coordinating processes to achieve synchronization in a distributed environment where clocks may drift and communication delays can vary significantly.

   - *Example:* Implementing mutual exclusion or distributed locking algorithms.

2. *Consistency:*

   - *Challenge:* Ensuring consistent views of shared data across multiple nodes despite concurrency, delays, and faults.

   - *Example:* Designing distributed algorithms for maintaining data consistency in a replicated database.

3. *Concurrency Control:*

   - *Challenge:* Managing access to shared resources to prevent conflicts and ensure data integrity in the presence of concurrent operations.

   - *Example:* Developing distributed transaction protocols to maintain atomicity and isolation.

4. *Fault Tolerance:*

   - *Challenge:* Designing algorithms to detect, isolate, and recover from node failures or network partitions in a way that ensures system reliability.

   - *Example:* Implementing distributed consensus algorithms like Paxos or Raft.

5. *Load Balancing:*

   - *Challenge:* Distributing computational load evenly across nodes to optimize resource utilization and prevent bottlenecks.

   - *Example:* Load balancing algorithms in distributed systems to allocate tasks efficiently.

6. *Distributed Coordination:*

   - *Challenge:* Coordinating the activities of distributed processes to achieve a common goal, especially in the absence of a centralized controller.

   - *Example:* Consensus algorithms for reaching an agreement among distributed nodes.

7. *Message Ordering:*

   - *Challenge:* Establishing a reliable order of messages exchanged between nodes to ensure correct execution of distributed algorithms.

- *Example:* Implementing causal ordering or total ordering of messages.

8. *Scalability:*

- *Challenge:* Ensuring that the system can handle an increasing number of nodes without significant degradation in performance or efficiency.

- *Example:* Designing scalable distributed architectures for large-scale systems.

9. *Network Partitioning:*

- *Challenge:* Addressing issues arising from network partitions that can lead to disconnected components within the distributed system.

- *Example:* Developing algorithms for handling partitioned networks and maintaining consistency.

10. *Adaptability:*

- *Challenge:* Designing algorithms that can adapt to dynamic changes in the system, such as node additions, removals, or varying workloads.

- *Example:* Self-stabilizing algorithms for systems that need to recover from transient faults.

Overlay in Tapestry P2P System:

The overlay in the Tapestry peer-to-peer (P2P) system refers to the logical network structure created on top of the physical network. In Tapestry, this overlay is formed using a distributed hash table (DHT) approach. Each node in the system is assigned a unique identifier, and the overlay organizes these nodes into a structured network based on their identifiers.

- *Distributed Hash Table (DHT):* Tapestry employs a DHT to map keys (identifiers) to their corresponding values (data or nodes). This mapping facilitates efficient lookup and retrieval of data in a decentralized manner.

Routing Scheme in Tapestry P2P System:

Tapestry uses a routing scheme that enables efficient and fault-tolerant routing between nodes in the overlay network. The routing scheme involves finding a route or path from the source node to the destination node based on their unique identifiers.

- *Routing Algorithm:* Tapestry uses a routing algorithm that utilizes the prefix matching of node identifiers. Nodes maintain routing tables that help in forwarding messages to nodes whose identifiers share a common prefix with the destination identifier.

- *Routing Tables:* Each node in Tapestry maintains a routing table that consists of multiple levels. The levels represent the digits of the node identifiers. For example, if node identifiers are represented in hexadecimal, each level corresponds to a digit (0-9, A-F).

- *Routing Process:*

  1. When a node wants to route a message to a destination node, it consults its routing table.

  2. The node forwards the message to the node with the identifier that matches the longest common prefix with the destination identifier.

  3. This process is repeated iteratively until the message reaches the destination node.

- *Fault Tolerance:* Tapestry incorporates fault tolerance by ensuring that multiple nodes in the routing path share a common prefix with the destination. If one node fails, the routing algorithm can route around the failed node by selecting an alternative path.

In summary, Tapestry's overlay is structured using a DHT approach, and its routing scheme efficiently directs messages through the overlay network based on the unique identifiers of the nodes. The use of prefix matching and distributed routing tables contributes to the effectiveness and fault tolerance of the Tapestry P2P system.

---

1. *Two Important Key Issues of Distributed Systems:*

   - *Synchronization:* Coordinating events and processes in a distributed system.

   - *Consistency:* Ensuring consistent views of shared data across nodes.

2. *Concurrent Events:*

   - *Definition:* Events occurring simultaneously in a distributed system, potentially leading to conflicts or dependencies.

3. *Vector Clock and Timestamp (3,4,2) at System S2:*

   - *Explanation:* In a vector clock, (3,4,2) at S2 means that S2 has seen 3 events in its own process, 4 events in the second process, and 2 events in the third process.

4. *Cut:*

   - *Definition:* A cut in a distributed computation represents a partition between the events that have occurred and those that haven't, providing a snapshot of the system.

5. *Reason for Allowing Transit Messages in Recording Global States:*

   - *Explanation:* Allowing transit messages captures potential causal relationships, ensuring accurate representation of the system's progress during global state recording.

6. *Two Performance Parameters of Distributed Mutex Exclusion:*

   - *Latency:* Time taken for a process to enter a critical section.

   - *Throughput:* Number of critical sections executed per unit time.

7. *Idle Token:*

   - *Definition:* An idle token signifies that a process is not currently in need of a particular resource or permission, indicating its idle state.

8. *Message Complexity of Two Non-Token-Based D-MUTEX Algorithms:*

   - *Example:* Lamport's Mutex Algorithm and Ricart-Agrawala Algorithm.

   - *Complexity:* Lamport's Algorithm requires $2n - 2$ messages, while Ricart-Agrawala Algorithm requires $n - 1$ messages.

9. *Difference Between Starvation and Deadlocks:*

   - *Starvation:* Persistent blocking of a process due to unavailability of resources.

   - *Deadlock:* Mutual blocking of two or more processes, each waiting for the other to release a resource.

10. *Maximum Number of Malicious Processes in Byzantine Agreement:*

   - *Formula:* $\lceil (n - 1) / 3 \rceil$, where n is the total number of processes.

11. *Two Applications of Byzantine Consensus:*

   - *Cryptocurrencies:* Achieving consensus in blockchain networks.

   - *Distributed Databases:* Ensuring consistency and agreement on data across distributed nodes.

12. *Impact of Lost Messages on Consistency and Performance:*

   - *Consistency:* Lost messages may lead to inconsistencies in replicated data.

- *Performance:* Increased latency and potential for retransmissions impact system performance.

13. *Disadvantages of Global State Recording Protocol for Checkpointing:*

- *Overhead:* Incurs high overhead due to the need for coordinated checkpoints.
- *Complexity:* Requires synchronization among processes, leading to increased complexity.

14. *Comparison Between Stable Log and Volatile Log:*

- *Stable Log:* Persists across failures, ensuring durability.
- *Volatile Log:* May be lost during failures, sacrificing durability for performance.

Aspect	Peer-to-Peer (P2P) Model	Client-Server Model
Architecture	Decentralized, peers (nodes) communicate directly with each other.	Centralized, clients communicate with a central server.
Communication	Peers both consume and provide resources/services.	Clients request services/resources from the server.
Resource Management	Shared responsibility among peers; each peer is a resource provider.	Centralized management by the server; clients are resource consumers.
Dependency on Server	No single point of failure; peers can function independently.	Single point of failure; server availability crucial for operation.
Scalability	Potentially more scalable as each peer contributes to the network.	Scalability may be limited, dependent on server capacity.
Content Discovery	May use distributed mechanisms for content discovery (e.g., distributed hash tables).	Centralized directory or indexing for content discovery.
Data Storage	Distributed and may involve peer-to-peer replication.	Centralized, server-centric data storage.
Examples	BitTorrent, Gnutella, blockchain	Web applications, file servers,

Peer-to-Peer (P2P) Systems:

Definition:

- *Peer-to-Peer (P2P)* refers to a decentralized network architecture where individual nodes, or peers, communicate directly with each other without relying on a central server. In P2P systems, each peer has the capability to both request and provide resources or services.

Key Characteristics:

1. *Decentralization:*
   - P2P systems distribute the workload and resources across the network, eliminating the need for a central point of control. Each peer is considered equal in terms of functionality.

2. *Resource Sharing:*
   - Peers in a P2P system can act both as consumers and providers of resources. They can share files, processing power, bandwidth, or other resources directly with other peers.

3. *Autonomy:*
   - Each peer operates independently and is responsible for its own actions. There is no central authority dictating the behavior of peers.

4. *Scalability:*
   - P2P systems often exhibit good scalability as more peers can be added to the network without relying on a central server. The network's capacity increases with the addition of more peers.

5. *Resilience:*
   - P2P systems can be more resilient to failures since there is no single point of failure. If one peer goes offline, others can continue to function.

6. *Content Discovery:*
   - P2P systems employ various mechanisms for content discovery, such as distributed hash tables (DHTs), indexing, or searching, to locate resources within the network.

7. *Examples:*
   - *File Sharing Networks:* BitTorrent, Gnutella.
   - *Blockchain and Cryptocurrencies:* Bitcoin, Ethereum.
   - *Collaborative Computing:* SETI@Home, Folding@Home.

Types of P2P Systems:

1. *Pure P2P:*

   - All nodes have equal status and functionality. There is no distinction between client and server roles.

2. *Hybrid P2P:*

   - Combines P2P with client-server architecture. Some nodes may have more specialized roles, and there might be certain centralization for specific functions.

Challenges:

1. *Security Concerns:*

   - Ensuring secure communication and preventing malicious activities can be challenging in decentralized environments.

2. *Content Availability:*

   - Content availability relies on the willingness of peers to share resources. If few peers have a specific resource, availability might be limited.

3. *Dynamic Network Topology:*

   - P2P networks often face challenges in maintaining stability and efficient routing as nodes join or leave the network.

Overall, Peer-to-Peer systems offer a flexible and decentralized approach to distributed computing, enabling resource sharing and collaboration without heavy reliance on central servers.