DISTRIBUTED SYSTEMS

MODULE 1

Distributed Systems in Computer Science

Distributed systems form a cornerstone of modern computing, where multiple independent machines collaborate to perform tasks as a unified entity. These systems are designed to share resources, process data collectively, and coordinate activities across multiple locations, offering advantages such as enhanced scalability, performance, and robustness against failures.

Relation to Parallel Multiprocessor/Multicomputer Systems

Parallel multiprocessor and multicomputer systems are designed to address computational problems by distributing tasks across multiple processing units. These units can either be within a single machine (multiprocessor) or across multiple interconnected machines (multicomputer). The main objective of both systems is to enhance computational performance and efficiency through parallelism.

- **Multiprocessor Systems**: These systems involve multiple processors within a single physical machine. They share a common memory space, enabling quick communication between processors. However, they face challenges in terms of memory access, synchronization, and contention for shared resources.
- **Multicomputer Systems**: These systems consist of independent computers connected via a network. Each computer has its own local memory, and communication occurs over the network. The primary challenges here include managing network latency, reliability, and message passing between the nodes.

Message-Passing Systems vs. Shared Memory Systems

Distributed computing systems can be categorized into **message-passing systems** and **shared memory systems**, each with its own set of characteristics and communication methods.

- **Message-Passing Systems**: These systems depend on exchanging messages between independent processes, where each process operates on its own memory. Communication happens through sending and receiving messages, typically used in

multicomputer systems where tasks are distributed across different nodes. The main challenge here is managing network communication efficiently to minimize latency and optimize throughput.

- **Shared Memory Systems**: In contrast, shared memory systems allow multiple processes to access a common memory space. This is typical in multiprocessor systems, where all processors share and access the same memory. The key benefit is that communication between processes is faster, as they directly access shared variables. However, the major challenge lies in handling synchronization issues, such as managing concurrent access to resources, which could lead to race conditions and deadlocks if not addressed properly.

Synchronous vs. Asynchronous Executions

Execution models in distributed systems are divided into **synchronous** and **asynchronous** approaches, which determine how processes synchronize with one another.

- **Synchronous Execution**: In synchronous systems, processes operate in lockstep, meaning each process waits for others to reach certain synchronization points before it proceeds. While this ensures that all processes are aligned, it can lead to inefficiencies due to idle waiting time, especially when tasks have uneven workloads. Synchronous execution simplifies coordination but can become a bottleneck if not efficiently managed.
- **Asynchronous Execution**: In asynchronous systems, processes run independently, without waiting for each other. This allows greater flexibility and efficiency, as processes can continue without waiting for synchronization signals. However, the lack of coordination can lead to conflicts or inconsistencies, especially when managing concurrent tasks and ensuring that no race conditions occur.

Design Issues and Challenges

Designing distributed systems that involve parallel processing or message-passing introduces several challenges. Some of the key design issues include:

- **Concurrency Management**: Coordinating multiple concurrent processes, ensuring proper synchronization, and preventing race conditions are crucial aspects of design.
- **Fault Tolerance**: Systems must be able to recover gracefully from failures, such as network partitions or node crashes.
- **Scalability**: The system must efficiently scale to accommodate an increasing number of processes or nodes without significant performance degradation.
- **Latency**: Minimizing communication delays, especially in message-passing systems where data transfer over the network can introduce significant latency.
- **Consistency**: Maintaining a consistent system state across nodes, especially when facing network failures or concurrency issues, is critical.

A Model of Distributed Executions

A **model of distributed executions** offers a framework for understanding how tasks are assigned to different nodes, how processes communicate, and how events are ordered in a distributed system. A key aspect of this model is the **happens-before** relationship, which defines the order of events. If event A influences event B, event A is said to happen before event B. This relationship helps in maintaining causality and is crucial for ensuring consistency in distributed systems.

Models of Communication in Distributed Systems

Communication models define how processes interact and exchange data in distributed systems. Several models of communication are commonly used:

1. **Direct Communication (Message-Passing)**: In this model, processes send messages directly to each other. It is most commonly used in multicomputer systems, where processes are distributed across different machines. This model provides flexibility but requires efficient management of network latency and reliability.
2. **Indirect Communication**: Processes communicate via an intermediary, such as a message queue or shared memory. This model decouples processes, offering greater flexibility but possibly introducing delays due to the additional layer of communication.

3. **Synchronous Communication**: Processes must wait for acknowledgment or a response from the receiver before continuing. While this simplifies coordination, it can cause delays if not properly optimized.
4. **Asynchronous Communication**: In this model, processes send messages without waiting for a response, allowing the sender to continue execution immediately. This approach is more efficient but introduces challenges related to message timing and ordering.

Networks and Past/Future Cones of Events

In distributed systems, understanding the **past and future cones of an event** is essential for determining causal relationships between events. The **past cone** refers to all the events that must have occurred before a specific event, while the **future cone** represents events that will occur after it. This concept is crucial for maintaining consistency and causality, ensuring that events across distributed nodes are ordered correctly.

Models of Process Communication

Finally, **models of process communication** define how processes share data and synchronize their actions in a distributed system. These models vary depending on whether processes use synchronous or asynchronous communication, shared memory, or message-passing systems. By modeling these interactions, developers can design systems that are efficient, reliable, and scalable.