UCS2701 Distributed Systems

Unit 1

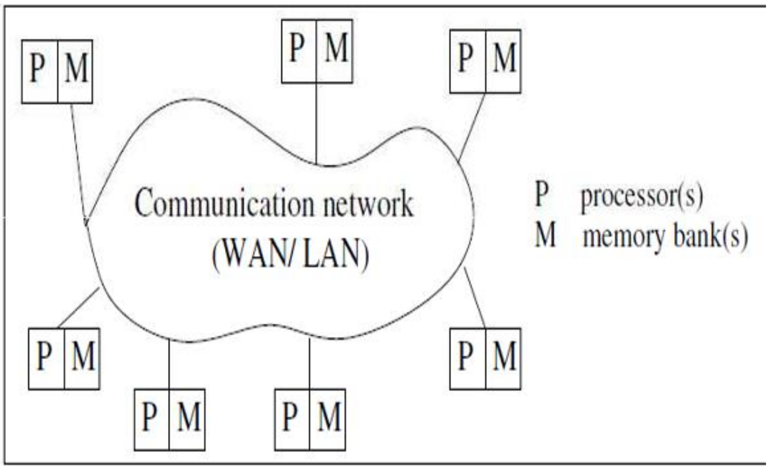
Introduction: Definition-Relation to computer system components – Motivation – Relation to parallel multiprocessor/multicomputer systems – Message-passing systems versus shared memory systems – Primitives for distributed communication – Synchronous versus asynchronous executions – Design issues and challenges; A model of distributed computations: A distributed program – A model of distributed executions – Models of communication net-works – Global state of a distributed system – Cuts of a distributed computation – Past and future cones of an event – Models of process communications.

Introduction to Distributed Systems

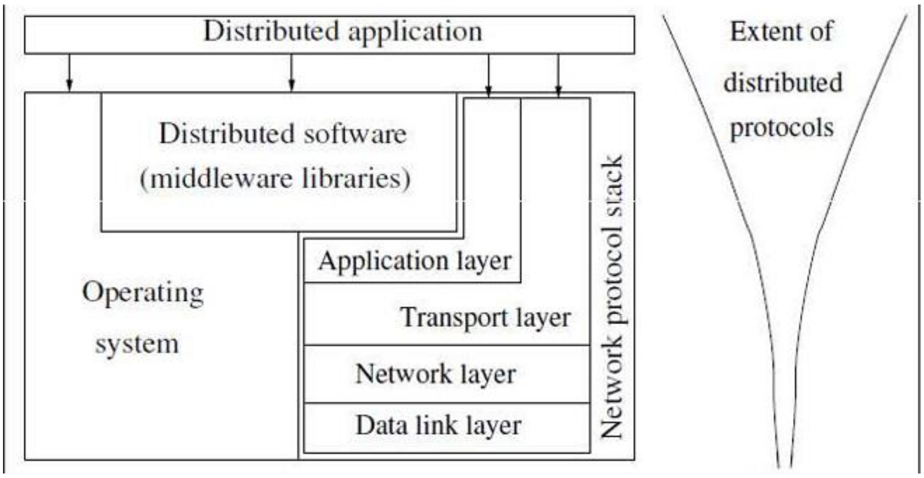
- A distributed system is a collection of independent entities that cooperate to solve a problem that cannot be individually solved
- Distributed system is one in which hardware or software components located at networked computers communicate and coordinate networked computers communicate and coordinate their actions only by passing messages.
- These Computers are semi-autonomous and are loosely coupled while they cooperate to address a problem collectively.

Characteristics of Distributed Systems:

- No common physical clock.
- No shared memory.
- Communication occurs via message passing over a communication network.
- Each computer:
  - Has its own memory.
  - Runs its own operating system.
- Geographical separation between systems.
- Autonomy and heterogeneity.
- Independent failure is natural in distributed systems:
  - Network faults can isolate computers.
  - Computer failures are not immediately known to others in the network.
- Concurrent program execution.



Relation between Software Components



Motivation for Distributed Systems

- **Inherently Distributed Computation**:
  - Example: Money transfer in banking.
- **Resource Sharing**:
  - Data in databases, special libraries, and files cannot always be replicated.
  - Example: Distributed databases.
- **Access to Geographically Remote Data & Resources**:
  - Replication of data is not always possible (e.g., data is too large or sensitive).
  - Example: Payroll data is too large and sensitive to replicate to every branch.
- **Increased Performance/Cost Ratio**:

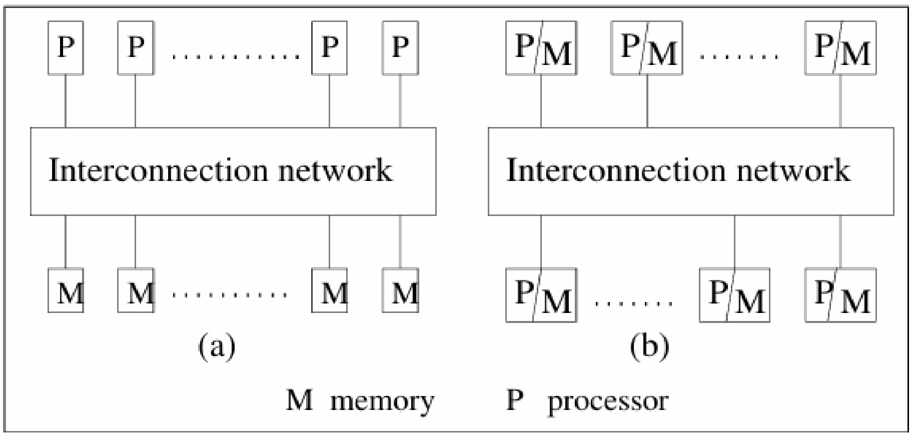
    - Achieved through resource sharing and partitioning tasks across multiple computers.
- **Reliability**:
    - Ensures availability, integrity, and fault tolerance.
- **Scalability**:
    - Adding more processors to a wide-area network (WAN) is straightforward.
- **Modularity and Incremental Expandability**:
    - Heterogeneous processors can be added easily without affecting performance.
    - Existing processors can be replaced by others with ease.

Parallel Systems

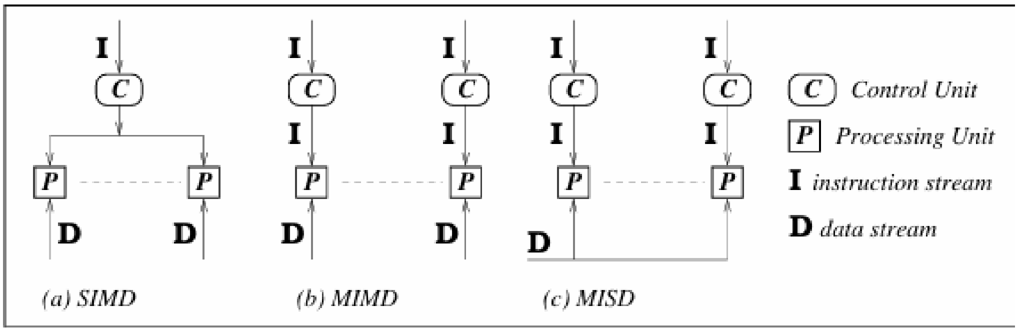
These include:

1. Multiprocessor Systems- direct access to shared memory, UMA model
2. Multicomputer parallel systems- no direct access to shared memory- NUMA model
3. Array processors- collocated, tightly coupled with a common system clock. Used in niche markets, like in DSP applications.

UMA vs NUMA



Flynn’s Taxonomy



- **SISD**: Single Instruction Stream, Single Data Stream
  - Traditional architecture.
- **SIMD**: Single Instruction Stream, Multiple Data Stream
  - Applications:
    - Scientific applications.
    - Applications on large arrays.
  - Examples:
    - Vector processors.
    - Systolic arrays.
    - Pentium/SSE.
    - DSP chips.
- **MISD**: Multiple Instruction Stream, Single Data Stream
  - Example:
    - Visualization.
- **MIMD**: Multiple Instruction Stream, Multiple Data Stream
  - Applications:
    - Distributed systems.
    - Vast majority of parallel systems.

Blocking vs Non Blocking Systems

Blocking Systems

- In a **blocking system**, a process halts its execution until the required operation (e.g., communication, resource access) is completed.

- The process cannot perform any other tasks while waiting, which might lead to inefficiencies or even deadlock in distributed systems.

Characteristics:

1. The process is paused until the operation succeeds or fails.
2. Commonly used for synchronous communication or operations.
3. Easier to design but may lead to lower system throughput due to waiting.

Advantages:

- Simpler programming model as the process's flow is sequential and predictable.
- Ensures that resources are available before proceeding.

Disadvantages:

- Inefficient use of system resources, as processes spend time waiting.
- Prone to deadlocks in poorly designed systems.

Non-Blocking Systems

- In a **non-blocking system**, a process initiates an operation and continues executing other tasks without waiting for the operation to complete.
- The process periodically checks (polls) or is notified when the operation is finished.

Characteristics:

1. The process continues execution after initiating the operation.
2. Commonly used for asynchronous communication or operations.
3. More complex to program, as handling the operation's completion requires additional mechanisms (e.g., callbacks, polling).

Advantages:

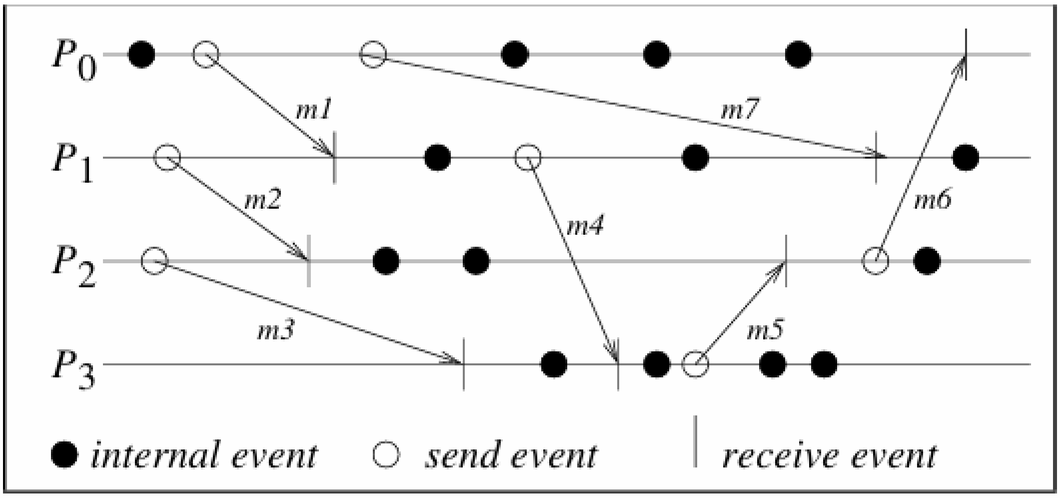
- Efficient use of resources since processes can perform other tasks while waiting for an operation to complete.
- Reduces the chances of deadlocks by avoiding unnecessary waiting.

Disadvantages:

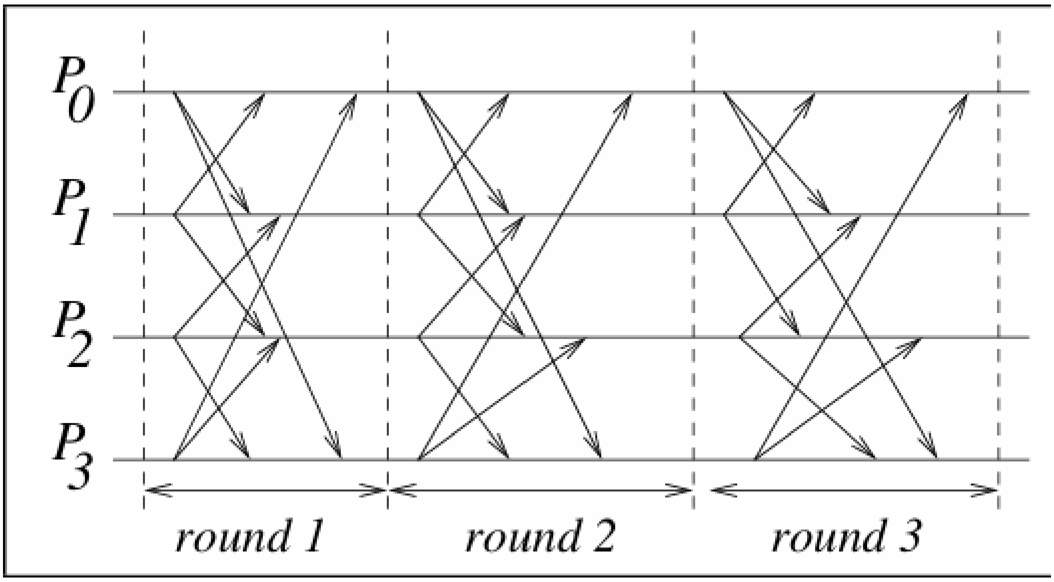
- Programming complexity increases due to the need for handling incomplete operations.
- Harder to debug and manage since the program flow is less predictable.

Asynchronous vs Synchronous Message Passing Systems

Asynchronous Execution



Synchronous Execution



a. **Asynchronous Execution**:

- Processors don’t run in sync, and there’s no limit on how much their clocks can differ.
- Messages between processors might take any amount of time to arrive (finite but no guaranteed speed).
- There’s no fixed time for a processor to complete a task.

b. **Synchronous Execution**:

- Processors work in sync with each other, and their clocks don’t drift much.
- Messages are delivered quickly and consistently within a single logical step.
- There’s a known maximum time for a processor to complete a task.

Challenges in Distributed Systems- System Perspective

1. **Communication Mechanisms**: Design efficient communication (e.g., RPC, message-oriented).
2. **Processes**: Manage code migration, threads, and software scalability.
3. **Naming**: Simplify locating resources and processes.
4. **Synchronization**: Ensure reliable process coordination.
5. **Data Storage**:

   - Fast and scalable storage, search, and access.
   - Address bottlenecks in file system design.

6. **Consistency and Replication**:

   - Fast access and scalability.
   - Ensure consistency among replicas.

7. **Fault Tolerance**: Operate correctly despite failures.
8. **Security**: Implement secure communication and access control.
9. **Scalability**: Add resources easily, support modular systems.

Challenges in Distributed Systems- Algorithmic/Design Perspective

1. **Execution Models**: Support models for interleaving, partial order, and logical time.
2. **Graph Algorithms**: Handle dynamic topologies and routing.
3. **Time and Global State**:

   - Ensure logical consistency in distributed systems.
   - Address concurrency and communication speeds.

4. **Synchronization**:

   - Clock synchronization and leader election.
   - Deadlock and termination detection.

5. **Group Communication**: Enable multicast and ordered message delivery.
6. **Data Replication**:

   - Optimize replica placement and coordination.
   - Cache and improve access speeds.

7. **Load Balancing**:

   - Dynamically redistribute tasks and data.
   - Enhance throughput and reduce latency.

8. **Fault-Tolerant Systems**:
   - Consensus algorithms, ACID properties, and recovery mechanisms.
9. **Performance Analysis**:
   - Model and reduce network latency.

Applications and Challenges of Distributed Systems

1. **Mobile Systems**:
   - Wireless communication: Power management, routing, and mobility management.
   - Models:
     - Base station model (e.g., cellular).
     - Ad-hoc network model (uses distributed graph theory).
2. **Sensor Networks**:
   - Processors with electro-mechanical interfaces for sensing.
3. **Ubiquitous Computing**:
   - Embedded processors in the environment (e.g., intelligent homes, smart workplaces).
   - Wireless, self-organizing, and resource-constrained systems.
4. **Peer-to-Peer Computing**:
   - Decentralized, symmetric, and scalable systems.
   - Efficient storage, lookup, and dynamic reconfiguration.
5. **Content Distribution**:
   - Publish/subscribe models for filtering and delivering relevant content.
6. **Distributed Agents**:
   - Processes that cooperate for specific tasks (e.g., mobility and interface design).
7. **Distributed Data Mining**:
   - Extract patterns and trends from distributed datasets.
8. **Grid Computing**:
   - Shared computing resources using idle CPU cycles.
   - Challenges: Scheduling, security, and quality of service (QoS).
9. **Security**:

- Managing confidentiality, authentication, and availability in distributed environments.
- Issues: Lack of trust, resource constraints, and unstructured networks.

Models of communication networks

There are several models of the service provided by communication networks, namely, FIFO, Non-FIFO, and causal ordering.

In the FIFO model, each channel acts as a first-in first-out message queue and thus, message ordering is preserved by a channel.

In the non-FIFO model, a channel acts like a set in which the sender process adds messages and the receiver process removes messages from it in a random order.

The "causal ordering" model is based on Lamport's "happens before" relation. A system that supports the causal ordering model satisfies the following property:

CO: For any two messages mij and mkj, if send(mij) −→ send(mkj), then rec(mij) −→ rec(mkj).

Causally ordered delivery of messages implies FIFO message delivery. Causal ordering model considerably simplifies the design of distributed algorithms because it provides a built-in synchronization.

Global state of a distributed system- A collection of local states, can talk about consistency, messages that are sent, in transit and received.

Cuts of a distributed computation

In the space-time diagram of a distributed computation, a cut is a zigzag line joining one arbitrary point on each process line.

A cut slices the space-time diagram, and thus the set of events in the distributed computation, into a PAST and a FUTURE. The PAST contains all the events to the left of the cut and the FUTURE contains all the events to the right of the cut.

Past and future cones of an event

1. Past Cone of an Event

The past cone of an event E includes all events that could have causally influenced E. These are the events that occurred before E and contributed to its occurrence.

Characteristics:

- Events in the past cone of E happened earlier in logical time.
- They are causally related to E, meaning their outcomes influenced E.
- Events that are not causally connected to E but occurred earlier are not part of the past cone.

Example:

If event E1 sends a message to event E2, E1 is in the past cone of E2 because E2 depends on E1.

2. Future Cone of an Event

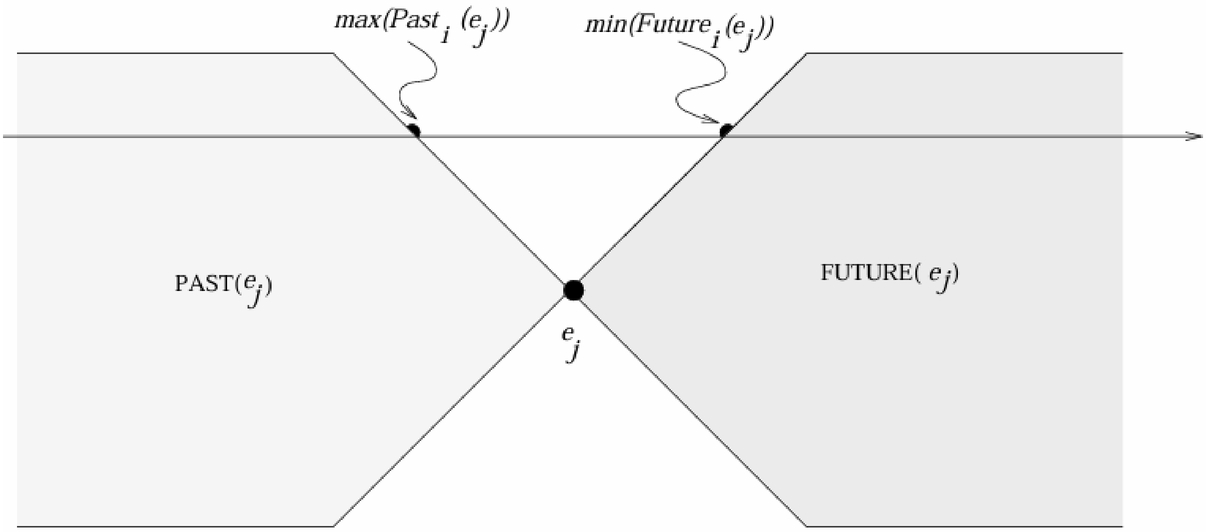
The future cone of an event E includes all events that are causally influenced by E. These are events that happen after E and are dependent on it.

Characteristics:

- Events in the future cone of E happen later in logical time.
- They represent the consequences or effects of E.
- Events concurrent with E (no causal relationship) are not part of its future cone.

Example:

If event E2 receives a message from event E1, E2 is in the future cone of E1.



$$Past(e_j) = \{e_i | \forall e_i \in H,\ e_i \rightarrow e_j\ \}.$$

$$Future(e_j) = \{e_i | \forall e_i \in H,\ e_j\ \rightarrow e_i\}.$$

Models of process communications- write about asynchronous and synchronous execution as well as blocking and non-blocking processes