DESIGN ISSUES AND CHALLENGES IN DISTRIBUTED SYSTEMS

The important design issues and challenges is categorized as

- related to systems design and operating systems design, or
- component related to algorithm design, or
- emerging from recent technology

Issues related to system and operating systems design

The following functions must be addressed when designing and building a distributed system:

Communication: This task involves designing suitable communication mechanisms among the various processes in the networks.

Examples: RPC, RMI

Processes: The main challenges involved are: Process and thread management at both client and server environments, code migration, design of software and mobile agents.

Naming: Devising easy to use and robust schemes for names, identifiers,and addresses is essential for locating resources and processes in a transparent and scalable manner.

Synchronization: Mutual exclusion, leader election, synchronizing physical clocks are some synchronization mechanisms.

Data storage and access Schemes: Schemes for data storage for accessing the data in a fast and scalable manner across the network are important for efficiency.

Consistency and replication:

- To avoid bottlenecks, to provide fast access to data, and to provide scalability, replication of data objects is highly desirable.
- This leads to issues of managing the replicas, and dealing with consistency among the replicas/caches in a distributed setting.

Fault tolerance:

- To maintaining correct and efficient operation in spite of any failures of links, nodes, and processes.
- Process resilience, reliable communication, distributed commit, and check pointing and recovery are some of the fault-tolerance mechanisms.

Security:

- Distributed systems security involves various aspects of cryptography, secure channels, access control, authorization, and secure group management.

Applications Programming Interface (API) and transparency: The API for communication and other services for the ease of use and wider adoption of distributed systems services by non-technical users.

Transparency deals with hiding the implementation policies from user, and is classified as follows

Access transparency: hides differences in data representation on different systems and provides uniform operations to access system resources.

Location transparency: makes the locations of resources transparent to the users.
Migration transparency: allows relocating resources without changing names.
Concurrency transparency deals with masking concurrent use of shared resources for user.
Failure transparency: refers to the system being reliable and fault-tolerant.

Algorithmic challenges in distributed computing

Designing useful execution models and frameworks

The interleaving model, partial order model, input/output automata model and the Temporal Logic of Actions (TLA) are some examples of models that provide different degrees of infrastructure.

Dynamic distributed graph algorithms and distributed routing algorithms

- The distributed system is generally modeled as a distributed graph.
- Hence graph algorithms are the base for large number of higher level communication, data dissemination, object location, and object search functions.
- These algorithms must have the capacity to deal with highly dynamic graph characteristics. They are expected to function like routing algorithms.

Time and global state in a distributed system

Synchronization is made on based on logical time. Logical time is relative time and eliminates the overheads of providing physical time for applications

Logical time can

(i) Capture the logic and inter-process dependencies

(ii) Track the relative progress at each process

Maintaining the global state of the system across space involves the role of time dimension for consistency.

Synchronization/coordination mechanisms

- Synchronization is essential for the distributed processes to facilitate concurrent execution without affecting other processes.
- The synchronization mechanisms also involve resource management and concurrency management mechanisms.

Some techniques for providing synchronization are:

- ✓ **Physical clock synchronization**
- ✓ **Leader election**
- ✓ **Mutual exclusion:** Access to the critical resource(s) has to be coordinated.

- ✓ **Deadlock detection and resolution**
- ✓ **Termination detection**
- ✓ **Garbage collection:** Detecting garbage requires coordination among the processes.

Group communication, multicast, and ordered message delivery

- A group is a collection of processes on an application domain.
- Group management protocols are needed for group communication wherein processes can join and leave groups dynamically, or fail.
- The concurrent execution of remote processes may sometimes violate the semantics and order of the distributed program. Hence, a formal specification of the semantics of ordered delivery need to be formulated.

Monitoring distributed events and predicates

- Predicates defined on program variables that are local to different processes are used for specifying conditions on the global system state, and are useful for applications like debugging, sensing the environment, and in industrial process control hence for monitoring such predicates are important.
- An important paradigm for monitoring distributed events is that of *event streaming*.

Distributed program design and verification tools

Debugging distributed programs

Data replication, consistency models, and caching

Distributed shared memory abstraction

- A shared memory is easier to implement since it does not involve managing the communication tasks.
- The communication is done by the middleware by message passing.
- The overhead of shared memory is to be dealt by the middleware technology.

Some of the methodologies that does the task of communication in shared memory distributed systems are:

Wait-free algorithms:

Mutual exclusion:

Register constructions:

Reliable and fault-tolerant distributed systems

The following are some of the fault tolerant strategies:

- ✓ **Consensus algorithms:**
- ✓ **Replication and replica management:**

Voting and quorum systems: Providing redundancy in the active or passive components in the system and then performing voting based on some quorum criterion is a classical way of dealing with fault-tolerance. Designing efficient algorithms for this purpose is the challenge.

Distributed databases and distributed commit: The distributed databases should also follow atomicity, consistency, isolation and durability (ACID) properties.

Self-stabilizing systems:
A *self-stabilizing* algorithm is any algorithm that is guaranteed to eventually take the system to a good state even if a bad state reached due to some error.

Check pointing and recovery algorithms:
Check pointing involves periodically recording the current state on secondary storage so that, in case of a failure, the entire computation is not lost but can be recovered from one of the recently taken checkpoints.

Failure detectors

- In asynchronous distributed systems there is no bound on time for message transmission.
- Hence, it is impossible to distinguish a sent-but-not-yet-arrived message from a message that was never sent ie., alive or failed.
- Failure detectors represent a class of algorithms that probabilistically suspect another process as having failed.

Load balancing

The objective of load balancing is to gain higher throughput, and reduce the user perceived latency.

The following are some forms of load balancing:

- **Data migration**
- **Computation migration**
- **Distributed scheduling**

Real-time scheduling

- Real-time scheduling is important for mission-critical applications, to accomplish the task execution on schedule.
- The problem becomes more challenging in a distributed system where a global view of the system state is absent.

Performance

- Although high throughput is not the primary goal of using a distributed system, achieving good performance is important.
- In large distributed systems, network latency and access to shared resources can lead to large delays which must be minimized.

Applications of distributed computing and newer challenges

Mobile systems
Mobile systems typically use wireless communication which is based on electromagnetic waves and utilizes a shared broadcast medium.

Characteristics and issues are

– *communication* : range and power of transmission,

– *engineering* : battery power conservation, interfacing with wired Internet, signal processing and interference.

– *computer science Perspective*: routing, location management, channel allocation, localization and position estimation, and the overall management of mobility.

There are two architectures for a mobile network.

1. *base-station* approach are *cellular approach*,

– where a *cell* is the geographical region within range of a static and powerful base transmission station is associated with base station.

– All mobile processes in that cell communicate via the base station.

2. *ad-hoc network* approach

– where there is no base station

– All responsibility for communication is distributed among the mobile nodes,

Sensor networks

- A sensor is a processor with an electro-mechanical interface that is capable of sensing physical parameters.
- They are low cost equipment with limited computational power and battery life. They are designed to handle streaming data and route it to external computer network and processes.
- They are susceptible to faults and have to reconfigure themselves.
- These features introduces a whole newset of challenges, such as position estimation and time estimation when designing a distributed system .

➢ **Ubiquitous or pervasive computing**

- In Ubiquitous systems the processors are embedded in the environment to performapplication functions in the background.
- **Examples:** Intelligent devices, smart homes etc.
- They are distributed systems with recent advancements operating in wireless environments through actuator mechanisms.
- They can be self-organizing and network-centricwith limited resources.

➢ **Peer-to-peer computing**

- Peer-to-peer (P2P) computing is computing over an application layernetwork where all interactions among the processors are at a same level.
- This is a form of symmetric computation against the client sever paradigm.
- They are self-organizing with or without regular structure to the network.
- Some of the key challenges include: object storage mechanisms, efficient object lookup, and retrieval in a scalable manner; dynamic reconfiguration with nodes as well as objects joining and leaving the networkrandomly; replication strategies to expedite object search; tradeoffs betweenobject size latency and table sizes; anonymity, privacy, and security.

➢ **Publish-subscribe, content distribution, and multimedia**

- The users in present day require only the information of interest.
- In a dynamic environment where the informationconstantly fluctuates there is great demand for
- **Publish:**an efficient mechanism for distributing this information
- **Subscribe:** an efficient mechanism to allow end users to indicate interest in receiving specific kinds of information
- An efficient mechanism foraggregating large volumes of published information and filtering it as per theuser's subscription filter.

- Content distribution refers to a mechanism that categorizes the information based on parameters.
- The publish subscribe and content distribution overlap each other.
- Multimedia data introduces special issue because of its large size.

➢ **Distributed agents**

- Agents are software processes or sometimes robots that move around the systemto do specific tasks for which they are programmed.
- Agents collect and process information and can exchangesuch information with other agents.
- Challenges in distributed agent systems include coordination mechanisms among the agents, controlling the mobility of the agents,their software design and interfaces.

➢ **Distributed data mining**

- Data mining algorithms process large amount of data to detect patternsand trends in the data, to mine or extract useful information.
- Themining can be done by applying database and artificial intelligence techniquesto a data repository.

➢ **Grid computing**

- Grid computing is deployed to manage resources. For instance, idle CPU cycles of machines connected to the network will be available to others.
- The challenges includes:scheduling jobs, framework for implementing quality of service, real-time guarantees,security.

➢ **Security in distributed systems**

The challenges of security in a distributed setting include: confidentiality, authentication and availability. This can be addressed usingefficient and scalable solutions.

1.9 A MODEL OF DISTRIBUTED COMPUTATIONS: DISTRIBUTED PROGRAM

- A distributed program is composed of a set of asynchronous processes that communicate by message passing over the communication network. Each process may run on different processor.
- The processes do not share a global memory and communicate solely by passing messages. These processes do not share a global clock that is instantaneously accessible to these processes.
- Process execution and message transfer are asynchronous – a process may execute an action spontaneously and a process sending a message does not wait for the delivery of the message to be complete.
- The global state of a distributed computation is composed of the states of the processes and the communication channels. The state of a process is characterized by the state of its local memory and depends upon the context.
- The state of a channel is characterized by the set of messages in transit in the channel.

1.9.1 A MODEL OF DISTRIBUTED EXECUTIONS

- The execution of a process consists of a sequential execution of its actions.
- The actions are atomic and the actions of a process are modeled as three types of events: internal events, message send events, and message receive events.
- The occurrence of events changes the states of respective processes and channels, thus causing transitions in the global system state.
- An internal event changes the state of the process at which it occurs.

- A send event changes the state of the process that sends the message and the state of the channel on which the message is sent.
- The execution of process pi produces a sequence of events e1, e2, e3, …, and it is denoted by H_i: $H_i = (h_i \rightarrow_i)$. Here $h_i$are states produced by p_i and →are the casual dependencies among events p_i.
- $\rightarrow_{msg}$indicates the dependency that exists due to message passing between two events.
- *se*

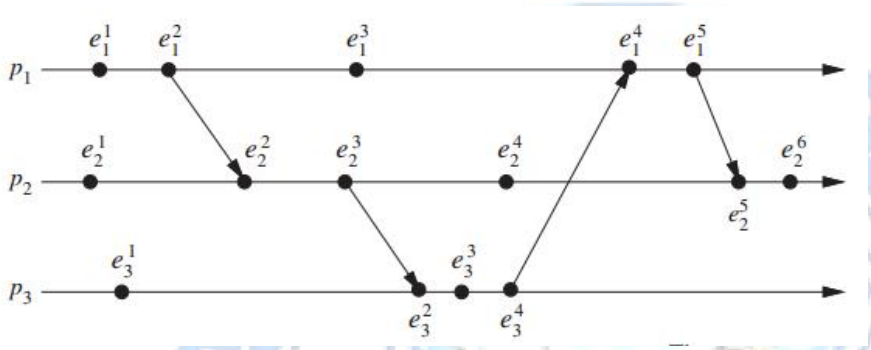

Fig 1.16: Space time distribution of distributed systems

- An internal event changes the state of the process at which it occurs.A send event changes the state of the process that sends the message andthe state of the channel on which the message is sent.
- A receive event changes the state of the process that receives the messageand the state of the channel on which the message is received.

1.9.2 Casual Precedence Relations

Causal message ordering is a partial ordering of messages in a distributed computing environment. It is the delivery of messages to a process in the order in which they were transmitted to that process.

> ***It places a restriction on communication between processes by requiring that if the transmission of message m_i to process p_k necessarily preceded the transmission of message m_j to the same process, then the delivery of these messages to that process must be ordered such that m_i is delivered before m_j.***

Happen Before Relation

The partial ordering obtained by generalizing the relationship between two process is called as ***happened-before relation or causal ordering or potential causal ordering.*** This term was coined by Lamport. Happens-before defines a partial order of events in a distributed system. Some events can't be placed in the order. If say A →B if A happens before B. A→B is defined using the following rules:

- ✓ **Local ordering:**A and B occur on same process and A occurs before B.
- ✓ **Messages:** send(m) → receive(m) for any message m
- ✓ **Transitivity:** e → e'' if e → e' and e' → e''
- Ordering can be based on two situations:

1. If two events occur in same process then they occurred in the order observed.

2. During message passing, the event of sending message occurred before the event of receiving it.

Lamports ordering is happen before relation denoted by →

- a→b, if a and b are events in the same process and a occurred before b.
- a→b, if a is the vent of sending a message m in a process and b is the event of the same message m being received by another process.

- If a→b and b→c, then a→c. Lamports law follow transitivity property.

When all the above conditions are satisfied, then it can be concluded that a→b is casually related. Consider two events c and d; c→d and d→c is false (i.e) they are not casually related, then c and d are said to be concurrent events denoted as c||d.

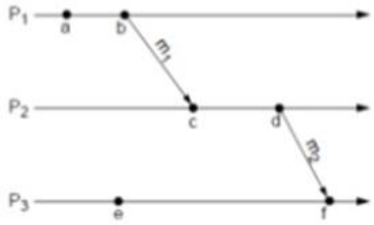

Fig 1.17: Communication between processes

Fig 1.22 shows the communication of messages m1 and m2 between three processes p1, p2 and p3. a, b, c, d, e and f are events. It can be inferred from the diagram that, a→b; c→d; e→f; b->c; d→f; a→d; a→f; b→d; b→f. Also a||e and c||e.

1.9.3 Logical vs physical concurrency

Physical as well as logical concurrency is two events that creates confusion in distributed systems.

Physical concurrency: Several program units from the same program that execute simultaneously.

Logical concurrency: Multiple processors providing actual concurrency. The actual execution of programs is taking place in interleaved fashion on a single processor.

Differences between logical and physical concurrency

Logical concurrency	Physical concurrency
Several units of the same program execute simultaneously on same processor, giving an illusion to the programmer that they are executing on multiple processors.	Several program units of the same program execute at the same time on different processors.
They are implemented through interleaving.	They are implemented as uni-processor with I/O channels, multiple CPUs, network of uni or multi CPU machines.

1.10 MODELS OF COMMUNICATION NETWORK

The three main types of communication models in distributed systems are:

FIFO (first-in, first-out): each channel acts as a FIFO message queue.

Non-FIFO (N-FIFO): a channel acts like a set in which a sender process adds messages and receiver removes messages in random order.

Causal Ordering (CO): It follows Lamport's law.

- The relation between the three models is given by CO ⊂ FIFO ⊂ N-FIFO.

A system that supports the causal ordering model satisfies the following property:

CO: For any two messages m_{ij} and m_{kj}, if $send(m_{ij}) \longrightarrow send(m_{kj})$, then $rec(m_{ij}) \longrightarrow rec(m_{kj})$.

1.11 GLOBAL STATE

Distributed Snapshot represents a state in which the distributed system might have been in. A snapshot of the system is a single configuration of the system.

- The global state of a distributed system is a collection of the local states of its components, namely, the processes and the communication channels.
- The state of a process at any time is defined by the contents of processor registers, stacks, local memory, etc. and depends on the local context of the distributed application.
- The state of a channel is given by the set of messages in transit in the channel.

The state of a channel is difficult to state formally because a channel is a distributed entity and its state depends upon the states of the processes it connects. Let $S_{i,j}^{x,y}$ denote the state of a channel C_{ij} defined as follows:

$$SC_{ij}^{x,y} = \{m_{ij} \mid send(m_{ij}) \preceq LS_i^x \wedge rec(m_{ij}) \npreceq LS_j^y\}.$$

A distributed snapshot should reflect a consistent state. A global state is consistent if it could have been observed by an external observer. For a successful Global State, all states must be consistent:

- If we have recorded that a process P has received a message from a process Q, then we should have also recorded that process Q had actually send that message.
- Otherwise, a snapshot will contain the recording of messages that have been received but never sent.
- The reverse condition (Q has sent a message that P has not received) is allowed.

The notion of a global state can be graphically represented by what is called a **cut**. A cut represents the last event that has been recorded for each process.
The history of each process if given by:

$$history(p_i) = h_i = <e_i^0, e_i^1, e_i^2, \ldots>$$

Each event either is an internal action of the process. We denote by s_i^k the state of process p_i immediately before the k^{th} event occurs. The state s_i in the global state S corresponding to the cut C is that of p_i immediately after the last event processed by p_i in the cut – ei^{ci} . The set of events ei^{ci} is called the frontier of the cut.

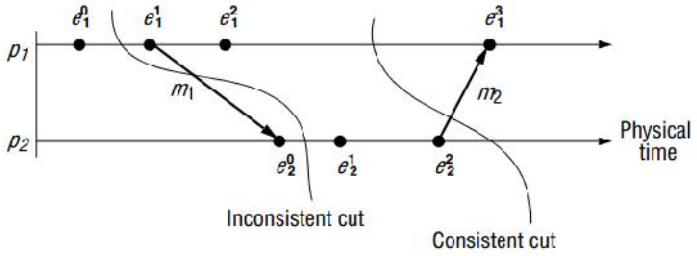

Fig 1.18: Types of cuts

Consistent states: The states should not violate causality. Such states are called consistent global states and are meaningful global states.

Inconsistent global states: They are not meaningful in the sense that a distributed system can never be in an inconsistent state.

1.12 CUTS OF A DISTRIBUTED COMPUTATION

A cut is a set of cut events, one per node, each of which captures the state of the node on which it occurs.

Cut is pictorially a line 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 theright of the cut. For a cut C, let PAST(C) and FUTURE(C) denote the set ofevents in the PAST and FUTURE of C, respectively.

Consistent cut: A consistent global state corresponds to a cut in which every message received in the PAST of the cut was sent in the PAST of that cut.

Inconsistent cut: A cut is inconsistent if a message crosses the cut from the FUTURE to the PAST.

1.13 PAST AND FUTURE CONES OF AN EVENT

In a distributed computation, an event e_j could have been affected only by all events e_i, such that $e_i \rightarrow e_j$ and all the information available at e_i could be made accessible at e_j. In other word e_i and e_j should have a causal relationship. Let Past(e_i) denote all events in the past of e_j in any computation.

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

The term max(past(e_i)) denotes the latest event of process p_i that has affected e_j. This will always be a message sent event.

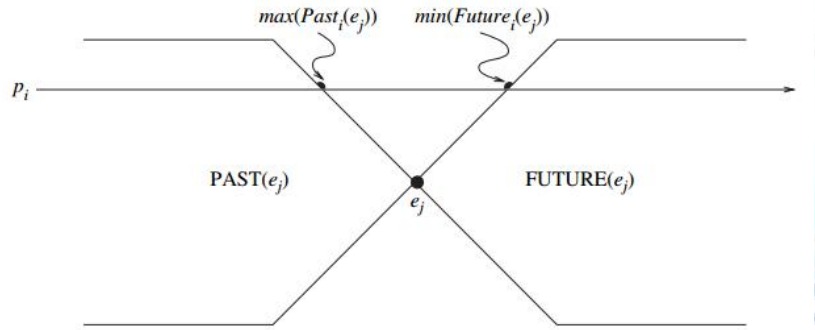

Fig 1.19: Past and future cones of event

A cut in a space-time diagram is a line joining an arbitrary point on each process line that slices the space-time diagram into a PAST and a FUTURE. A consistent global state corresponds to a cut in which every message received in the PAST of the cut was sent in the PAST of that cut.

The future of an event $e_i$denoted by Future(e_j) contains all the events e_i that are casually affected by e_j.

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

Future$_i$(e_i) is the set of those events of Future (e_j) are the process p_i and min(Future$_i$(e_j)) as the first event on process p_i that is affected by e_j. All events at a process p_i that occurred afterMax(Past(e_j)) but before min(Future$_i$(e_j)) are concurrent with e_j.

1.14 MODELS OF PROCESS COMMUNICATIONS

There are two basic models of process communications

Synchronous: The sender process blocks until the message has been received by the receiver process. The sender process resumes executiononly after it learns that the receiver process has accepted the message. The sender and the receiver processes must synchronize to exchange a message.

Asynchronous: It is non- blocking communication where the sender and the receiver do not synchronize to exchange a message. The sender process does not wait for the message to be delivered to the receiver

process. The message is buffered by the system and is delivered to the receiver process when it is ready to accept the message. A buffer overflow may occur if a process sends a large number of messages in a burst to another process, thus causing a message burst.

Asynchronous communication achieves high degree of parallelism and non- determinism at the cost of implementation complexity with buffers. On the other hand, synchronization is simpler with low performance. The occurrence of deadlocks and frequent blocking of events prevents it from reaching higher performance levels.