SYNCHRONOUS VS ASYNCHRONOUS EXECUTIONS

The execution of process in distributed systems may be synchronous or asynchronous.

Asynchronous Execution:

A communication among processes is considered asynchronous, when every communicating process can have a different observation of the order of the messages being exchanged. In an asynchronous execution:

- there is no processor synchrony and there is no bound on the drift rate of processor clocks
- message delays are finite but unbounded
- no upper bound on the time taken by a process

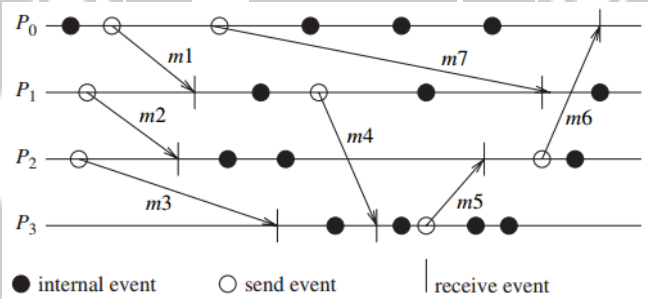

Fig : Asynchronous execution in message passing system

Synchronous Execution:

A communication among processes is considered synchronous when every process observes the same order of messages within the system. In the same manner, the execution is considered synchronous, when every individual process in the system observes the same total order of all the processes which happen within it. In an synchronous execution:

- processors are synchronized and the clock drift rate between any two processors is bounded
- message delivery times are such that they occur in one logical step or round
- upper bound on the time taken by a process to execute a step.

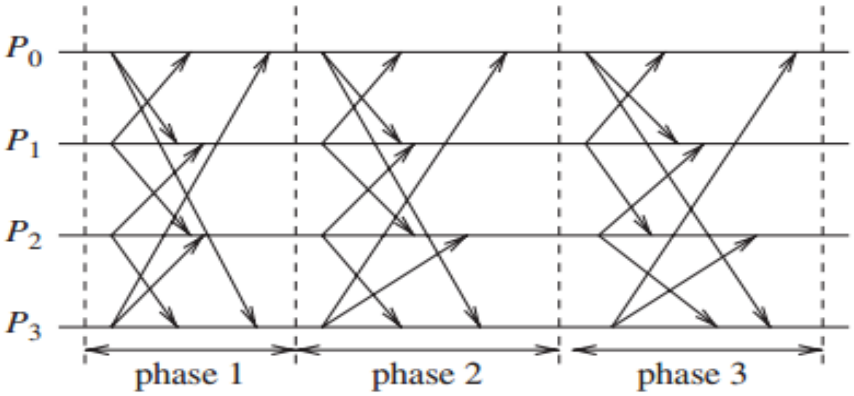

Fig: Synchronous execution

Emulating an asynchronous system by a synchronous system (A → S)

An asynchronous program can be emulated on a synchronous system fairly trivially as the synchronous system is a special case of an asynchronous system – all communication finishes within the same round in which it is initiated.

Emulating a synchronous system by an asynchronous system (S → A)

A synchronous program can be emulated on an asynchronous system using a tool called synchronizer.

Emulation for a fault free system

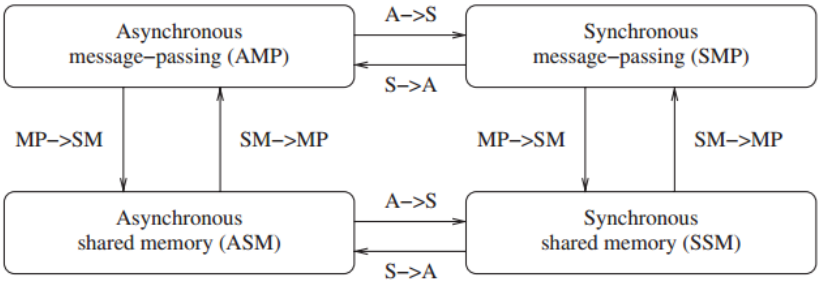

Fig : Emulations in a failure free message passing system

If system A can be emulated by system B, denoted A/B, and if a problem is not solvable in B, then it is also not solvable in A. If a problem is solvable in A, it is also solvable in B. Hence, in a sense, all four classes are equivalent in terms of computability in failure-free systems.

DESIGN ISSUES AND CHALLENGES IN DISTRIBUTED SYSTEMS

The design of distributed systems has numerous challenges. They can be categorized into:

- Issues related to system and operating systems design

- Issues related to algorithm design
- Issues arising due to emerging technologies

The above three classes are not mutually exclusive.

Issues related to system and operating systems design

The following are some of the common challenges to be addressed in designing a distributed system from system perspective:

- **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, migration of code between systems, 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. The remote and highly varied geographical locations make this task difficult.
- **Synchronization:** Mutual exclusion, leader election, deploying physical clocks, global state recording are some synchronization mechanisms.
- **Data storage and access Schemes:** Designing file systems for easy and efficient data storage with implicit accessing mechanism is very much essential for distributed operation
- **Consistency and replication:** The notion of Distributed systems goes hand in hand with replication of data, to provide high degree of scalability. The replicas should be handed with care since data consistency is prime issue.
- **Fault tolerance:** This requires maintenance of fail proof links, nodes, and processes. Some of the common fault tolerant techniques are resilience, reliable communication, distributed commit, check pointing and recovery, agreement and consensus, failure detection, and self-stabilization.
- **Security:** Cryptography, secure channels, access control, key management – generation and distribution, authorization, and secure group management are some of the security measure that is imposed on distributed systems.

- **Applications Programming Interface (API) and transparency:** The user friendliness and ease of use is very important to make the distributed services to be used by wide community. Transparency, which is hiding inner implementation policy from users, is of the following types:
  - **Access transparency:** hides differences in data representation
  - **Location transparency:** hides differences in locations y providing uniform access to data located at remote locations.
  - **Migration transparency:** allows relocating resources without changing names.
  - **Replication transparency:** Makes the user unaware whether he is working on original or replicated data.
  - **Concurrency transparency:** Masks the concurrent use of shared resources for the user.
  - **Failure transparency**: system being reliable and fault-tolerant.
- **Scalability and modularity:** The algorithms, data and services must be as distributed as possible. Various techniques such as replication, aching and cache management, and asynchronous processing help to achieve scalability.

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.
- The performance of these algorithms has direct impact on user-perceived latency, data traffic and load in the network.

- **Time and global state in a distributed system**

- The geographically remote resources demands the synchronization based on logical time.

- Logical time is relative 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. This can be done with extra effort in a coordinated manner.
- Deriving appropriate measures of concurrency also involves the time dimension, as the execution and communication speed of threads may vary a lot.

➢ **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:** Physical clocks usually diverge in the values due to hardware limitations. Keeping them synchronized is a fundamental challenge to maintain common time.

✓ **Leader election:** All the processes need to agree on which process will play the role of a distinguished process or a leader process. A leader is necessary even for many distributed algorithms because there is often some asymmetry.

✓ **Mutual exclusion:** Access to the critical resource(s) has to be coordinated.

✓ **Deadlock detection and resolution:** This is done to avoid duplicate work, and deadlock resolution should be coordinated to avoid unnecessary aborts of processes.

✓ **Termination detection**: cooperation among the processes to detect the specific global state of quiescence.

✓ **Garbage collection:** Detecting garbage requires coordination among the processes.

➢ **Group communication, multicast, and ordered message delivery**

- A group is a collection of processes that share a common context and collaborate on a common task within 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, and then implemented.

➢ **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.
- On-line algorithms for monitoring such predicates are hence important.
- An important paradigm for monitoring distributed events is that of event streaming, wherein streams of relevant events reported from different processes are examined collectively to detect predicates.
- The specification of such predicates uses physical or logical time relationships.

➢ **Distributed program design and verification tools**

Methodically designed and verifiably correct programs can greatly reduce the overhead of software design, debugging, and engineering. Designing these is a big challenge.

➢ **Debugging distributed programs**

Debugging distributed programs is much harder because of the concurrency and replications. Adequate debugging mechanisms and tools are need of the hour.

➢ **Data replication, consistency models, and caching**

- Fast access to data and other resources is important in distributed systems.
- Managing replicas and their updates faces concurrency problems.
- Placement of the replicas in the systems is also a challenge because resources usually cannot be freely replicated.

➢ **World Wide Web design – caching, searching, scheduling**

- WWW is a commonly known distributed system.
- The issues of object replication and caching, pre fetching of objects have to be done on WWW also.
- Object search and navigation on the web are important functions in the operation of the web.

➢ **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:** The ability of a process to complete its execution irrespective of the actions of other processes is wait free algorithm. They control the access to shared resources in the shared memory abstraction. They are expensive.

✓ **Mutual exclusion:** Concurrent access of processes to a shared resource or data is executed in mutually exclusive manner. Only one process is allowed to execute the critical section at any given time. In a distributed system, shared variables or a local kernel cannot be used to implement mutual exclusion. Message passing is the sole means for implementing distributed mutual exclusion.

✓ **Register constructions:** Architectures must be designed in such a way that, registers allows concurrent access without any restrictions on the concurrency permitted.

➢ **Reliable and fault-tolerant distributed systems**

The following are some of the fault tolerant strategies:

✓ **Consensus algorithms:** Consensus algorithms allow correctly functioning processes to reach agreement among themselves in spite of the existence of malicious processes. The goal of the malicious processes is to prevent the correctly functioning processes from reaching agreement. The malicious processes operate by sending messages with misleading information, to confuse the correctly functioning processes.

✓ **Replication and replica management:** The Triple Modular Redundancy (TMR) technique is used in software and hardware implementation. TMR is a fault-tolerant form of N-modular redundancy, in which three systems perform a process and that result is processed by a majority-voting system to produce a single output.

✓ **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:** All system executions have associated good(or legal) states and bad (or illegal) states; during correct functioning, the system makes transitions among the good states. A self-stabilizing algorithm guarantee to take the system to a good state even if a bad state were to arise due to some error. Self-stabilizing algorithms require some in-built redundancy to track additional variables of the state and do extra work.

✓ **Checkpointing and recovery algorithms:** Check pointing is 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. Check pointing in a distributed environment is difficult because if the checkpoints at the different processes are not coordinated, the local checkpoints may become useless because they are inconsistent with the checkpoints at other processes.

✓ **Failure detectors:** The asynchronous distributed do not have a bound on the message transmission time. This makes the message passing very difficult, since the receiver do not know the waiting time. Failure detectors probabilistically suspect another process as having failed and then converge on a determination of the up/down status of the suspected process.

➢ **Load balancing:** The objective of load balancing is to gain higher throughput, and reduce the user perceived latency. Load balancing may be necessary because of a variety off actors such as high network traffic or high request rate causing the network connection to be a bottleneck, or high computational load. The following are some forms of load balancing:

✓ **Data migration:** The ability to move data around in the system, based on the access pattern of the users

✓ **Computation migration**: The ability to relocate processes in order to perform a redistribution of the workload.

✓ **Distributed scheduling:** This achieves a better turnaround time for the users by using idle processing power in the system more efficiently.

➢ **Real-time scheduling**

Real-time scheduling becomes more challenging when a global view of the system state is absent with more frequent on-line or dynamic changes. The message propagation delays which are network-dependent are hard to control or predict. This is an hindrance to meet the QoS requirements of the network.

➢ **Performance**

User perceived latency in distributed systems must be reduced. The common issues in performance:

✓ **Metrics:** Appropriate metrics must be defined for measuring the performance of theoretical distributed algorithms and its implementation.

✓ **Measurement methods/tools:** The distributed system is a complex entity appropriate methodology and tools must be developed for measuring the performance metrics.

Applications of distributed computing and newer challenges

The deployment environment of distributed systems ranges from mobile systems to cloud storage. All the environments have their own challenges:

➢ **Mobile systems**

- Mobile systems which use wireless communication in shared broadcast medium have issues related to physical layer such as transmission range, power, battery power consumption, interfacing with wired internet, signal processing and interference.
- The issues pertaining to other higher layers include routing, location management, channel allocation, localization and position estimation, and mobility management.
- Apart from the above mentioned common challenges, the architectural differences of the mobile network demands varied treatment. The two architectures are:

✓ **Base-station approach (cellular approach):** The geographical region is divided into hexagonal physical locations called cells. The powerful base station transmits signals to all other nodes in its range

✓ **Ad-hoc network approach:** This is an infrastructure-less approach which do not have any base station to transmit signals. Instead all the responsibility is distributed among the mobile nodes.

✓ It is evident that both the approaches work in different environment with different principles of communication. Designing a distributed system to cater the varied need is a great challenge.

➢ **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 new set 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 perform application 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-centric with limited resources.

➢ **Peer-to-peer computing**

- Peer-to-peer (P2P) computing is computing over an application layer network 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 network randomly; replication strategies to expedite object search; tradeoffs between object 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 information constantly 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 for aggregating large volumes of published information and filtering it as per the user'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 system to do specific tasks for which they are programmed.
- Agents collect and process information and can exchange such 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 patterns and trends in the data, to mine or extract useful information.
- The mining can be done by applying database and artificial intelligence techniques to 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 using efficient and scalable solutions.