**Synchronization in Distributed Systems**

[Distributed System](https://www.geeksforgeeks.org/design-issues-of-distributed-system/) is a collection of computers connected via the high speed communication network. In the distributed system, the hardware and software components communicate and coordinate their actions by message passing. Each node in distributed systems can share their resources with other nodes. So, there is need of proper allocation of resources to preserve the state of resources and help coordinate between the several processes. To resolve such conflicts, synchronization is used. Synchronization in distributed systems is achieved via clocks.

The physical clocks are used to adjust the time of nodes. Each node in the system can share its local time with other nodes in the system. The time is set based on UTC (Universal Time Coordination). UTC is used as a reference time clock for the nodes in the system.

The clock synchronization can be achieved by 2 ways: External and Internal Clock Synchronization.

1. **External clock synchronization** is the one in which an external reference clock is present. It is used as a reference and the nodes in the system can set and adjust their time accordingly.
2. **Internal clock synchronization** is the one in which each node shares its time with other nodes and all the nodes set and adjust their times accordingly.

There are 2 types of clock synchronization algorithms: Centralized and Distributed.

1. **Centralized** is the one in which a time server is used as a reference. The single time server propagates it’s time to the nodes and all the nodes adjust the time accordingly. It is dependent on single time server so if that node fails, the whole system will lose synchronization. Examples of centralized are- Berkeley Algorithm, Passive Time Server, Active Time Server etc.
2. **Distributed** is the one in which there is no centralized time server present. Instead the nodes adjust their time by using their local time and then, taking the average of the differences of time with other nodes. Distributed algorithms overcome the issue of centralized algorithms like the scalability and single point failure. Examples of Distributed algorithms are – Global Averaging Algorithm, Localized Averaging Algorithm, NTP (Network time protocol) etc.

**Logical Clock in Distributed System**

**Logical Clocks**refer to implementing a protocol on all machines within your distributed system, so that the machines are able to maintain consistent ordering of events within some virtual timespan. A logical clock is a mechanism for capturing chronological and causal relationships in a distributed system. Distributed systems may have no physically synchronous global clock, so a logical clock allows global ordering on events from different processes in such systems.

**Example:**   
If we go outside then we have made a full plan that at which place we have to go first, second and so on. We don’t go to second place at first and then the first place. We always maintain the procedure or an organization that is planned before. In a similar way, we should do the operations on our PCs one by one in an organized way.

Suppose, we have more than 10 PCs in a distributed system and every PC is doing it’s own work but then how we make them work together. There comes a solution to this i.e. LOGICAL CLOCK.

**Method-1:**  
To order events across process, try to sync clocks in one approach.

This means that if one PC has a time 2:00 pm then every PC should have the same time which is quite not possible. Not every clock can sync at one time. Then we can’t follow this method.

**Method-2:**  
Another approach is to assign Timestamps to events.

Taking the example into consideration, this means if we assign the first place as 1, second place as 2, and third place as 3 and so on. Then we always know that the first place will always come first and then so on. Similarly, If we give each PC their individual number than it will be organized in a way that 1st PC will complete its process first and then second and so on.

BUT, Timestamps will only work as long as they obey causality.

**What is causality?**Causality is fully based on HAPPEN BEFORE RELATIONSHIP.

* Taking single PC only if 2 events A and B are occurring one by one then TS(A) < TS(B). If A has timestamp of 1, then B should have timestamp more than 1, then only happen before relationship occurs.
* Taking 2 PCs and event A in P1 (PC.1) and event B in P2 (PC.2) then also the condition will be TS(A) < TS(B). Taking example- suppose you are sending message to someone at 2:00:00 pm, and the other person is receiving it at 2:00:02 pm.Then it’s obvious that TS (sender) < TS (receiver).

**Properties Derived from Happen Before Relationship –**

* **Transitive Relation –**  
  If, TS(A) <TS(B) and TS(B) <TS(C), then TS(A) < TS(C)

**Causally Ordered Relation** –  
a->b, this means that a is occurring before b and if there is any changes in a it will surely reflect on b.

**Concurrent Event** –  
This means that not every process occurs one by one, some processes are made to happen simultaneously i.e., A || B.

**Lamport’s Algorithm for Mutual Exclusion in Distributed System**

**Lamport’s Distributed Mutual Exclusion Algorithm** is a permission based algorithm proposed by Lamport as an illustration of his synchronization scheme for distributed systems.  
In permission based timestamp is used to order critical section requests and to resolve any conflict between requests.

In Lamport’s Algorithm critical section requests are executed in the increasing order of timestamps i.e. a request with smaller timestamp will be given permission to execute critical section first than a request with larger timestamp.

In this algorithm:

* Three type of messages (**REQUEST**, **REPLY** and **RELEASE**) are used and communication channels are assumed to follow FIFO order.
* A site send a **REQUEST** message to all other site to get their permission to enter critical section.
* A site send a **REPLY** message to requesting site to give its permission to enter the critical section.
* A site send a **RELEASE** message to all other site upon exiting the critical section.
* Every site Si, keeps a queue to store critical section requests ordered by their timestamps.  
  **request\_queuei** denotes the queue of site Si
* A timestamp is given to each critical section request using Lamport’s logical clock.
* Timestamp is used to determine priority of critical section requests. Smaller timestamp gets high priority over larger timestamp. The execution of critical section request is always in the order of their timestamp.

**Algorithm:**

* **To enter Critical section:**
  + When a site Si wants to enter the critical section, it sends a request message **Request(tsi, i)** to all other sites and places the request on **request\_queuei**. Here, Tsi denotes the timestamp of Site Si
  + When a site Sj receives the request message **REQUEST(tsi, i)** from site Si, it returns a timestamped REPLY message to site Si and places the request of site Si on **request\_queuej**

.

* **To execute the critical section:**
  + A site Si can enter the critical section if it has received the message with timestamp larger than **(tsi, i)** from all other sites and its own request is at the top of **request\_queuei**
* **To release the critical section:**
  + When a site Si exits the critical section, it removes its own request from the top of its request queue and sends a time stamped **RELEASE** message to all other sites
  + When a site Sj receives the timestamped **RELEASE** message from site Si, it removes the request of Si from its request queue

**Message Complexity:**   
Lamport’s Algorithm requires invocation of 3(N – 1) messages per critical section execution. These 3(N – 1) messages involves

* (N – 1) request messages
* (N – 1) reply messages
* (N – 1) release messages

**Drawbacks of Lamport’s Algorithm:**

* **Unreliable approach:** failure of any one of the processes will halt the progress of entire system.
* **High message complexity:** Algorithm requires 3(N-1) messages per critical section invocation.

**Performance:**

* Synchronization delay is equal to maximum message transmission time
* It requires 3(N – 1) messages per CS execution.
* Algorithm can be optimized to 2(N – 1) messages by omitting the **REPLY** message in some situations.

**Ricart–Agrawala Algorithm in Mutual Exclusion in Distributed System**

**Ricart–Agrawala algorithm** is an algorithm to for mutual exclusion in a distributed system proposed by Glenn Ricart and Ashok Agrawala. This algorithm is an extension and optimization of Lamport’s Distributed Mutual Exclusion Algorithm. Like Lamport’s Algorithm, it also follows permission based approach to ensure mutual exclusion.

In this algorithm:

* Two type of messages (**REQUEST** and **REPLY**) are used and communication channels are assumed to follow FIFO order.
* A site send a **REQUEST** message to all other site to get their permission to enter critical section.
* A site send a **REPLY** message to other site to give its permission to enter the critical section.
* A timestamp is given to each critical section request using Lamport’s logical clock.
* Timestamp is used to determine priority of critical section requests. Smaller timestamp gets high priority over larger timestamp. The execution of critical section request is always in the order of their timestamp.

**Algorithm:**

* **To enter Critical section:**
  + When a site Si wants to enter the critical section, it send a time stamped **REQUEST** message to all other sites.
  + When a site Sj receives a **REQUEST** message from site Si, It sends a **REPLY** message to site Si if and only if
    - Site Sj is neither requesting nor currently executing the critical section.
    - In case Site Sj is requesting, the timestamp of Site Si‘s request is smaller than its own request.

Otherwise the request is deferred by site Sj.

* **To execute the critical section:**
  + Site Si enters the critical section if it has received the **REPLY** message from all other sites.
* **To release the critical section:**
  + Upon exiting site Si sends **REPLY** message to all the deferred requests.

**Message Complexity:**Ricart–Agrawala algorithm requires invocation of 2(N – 1) messages per critical section execution. These 2(N – 1) messages involves

* (N – 1) request messages
* (N – 1) reply messages

**Drawbacks of Ricart–Agrawala algorithm:**

* **Unreliable approach:** failure of any one of node in the system can halt the progress of the system. In this situation, the process will starve forever.  
  The problem of failure of node can be solved by detecting failure after some timeout.

**Performance:**

* Synchronization delay is equal to maximum message transmission time
* It requires 2(N – 1) messages per Critical section execution

**Deadlock detection in Distributed systems**

In a distributed system deadlock can neither be prevented nor avoided as the system is so vast that it is impossible to do so. Therefore, only deadlock detection can be implemented. The techniques of deadlock detection in the distributed system require the following: 

* **Progress –**   
  The method should be able to detect all the deadlocks in the system.
* **Safety –**   
  The method should not detect false or phantom deadlocks.

There are three approaches to detect deadlocks in distributed systems. They are as follows: 

1. **Centralized approach –**   
   In the centralized approach, there is only one responsible resource to detect deadlock. The advantage of this approach is that it is simple and easy to implement, while the drawbacks include excessive workload at one node, single-point failure (that is the whole system is dependent on one node if that node fails the whole system crashes) which in turns makes the system less reliable.
2. **Distributed approach –**   
   In the distributed approach different nodes work together to detect deadlocks. No single point failure (that is the whole system is dependent on one node if that node fails the whole system crashes) as the workload is equally divided among all nodes. The speed of deadlock detection also increases.
3. **Hierarchical approach –**   
   This approach is the most advantageous. It is the combination of both centralized and distributed approaches of deadlock detection in a distributed system. In this approach, some selected nodes or clusters of nodes are responsible for deadlock detection and these selected nodes are controlled by a single node.