SYNCHRONIZATION

- Synchronization in Distributed Systems is much more difficult compared to synchronization in uniprocessor or multiprocessor systems.

CLOCK SYNCHRONIZATION:-

- In a centralized system time is unambiguous. When a process wants to know the time, it makes a system call & kernel tells it.
- In distributed systems there should be global agreement on time.

Physical Clocks:-

- A Computer clock / timer is usually a precisely machined quartz crystal. Associated with each crystal are two registers, A counter & a holding register.

Clock Synchronization Algorithms:-

- Cristian's Algorithm
- Berkeley Algorithm
- Averaging Algorithms

- All the algorithms have the same underlying model of the system
- Each machine is assumed to have a timer that causes an interrupt H times a second.

- When this timer goes off, the interrupt handler adds 1 to a s/w clock that keeps track of the number of ticks (interrupts) since some agreed upon time in the past.
- Let us call this value of clock C.
- More specifically when the UTC (Universal Coordinated Time) time is t, the value of the clock on machine p is $C_p(t)$.

& in a perfect world, we would have

$$C_p(t) = t \text{ for all } p \text{ \& all } t.$$

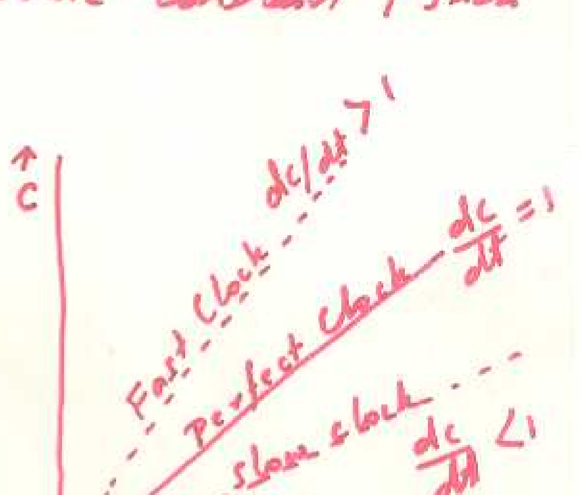
or, $\frac{dc}{dt}$ ideally should be 1.

- Real timers do not exactly interrupt H times a sec. The relative error obtainable with modern timer chips is about 10^{-5}.
- More precisely, if there exists some constant ρ such that

$$1-\rho \le \frac{dc}{dt} \le 1+\rho,$$

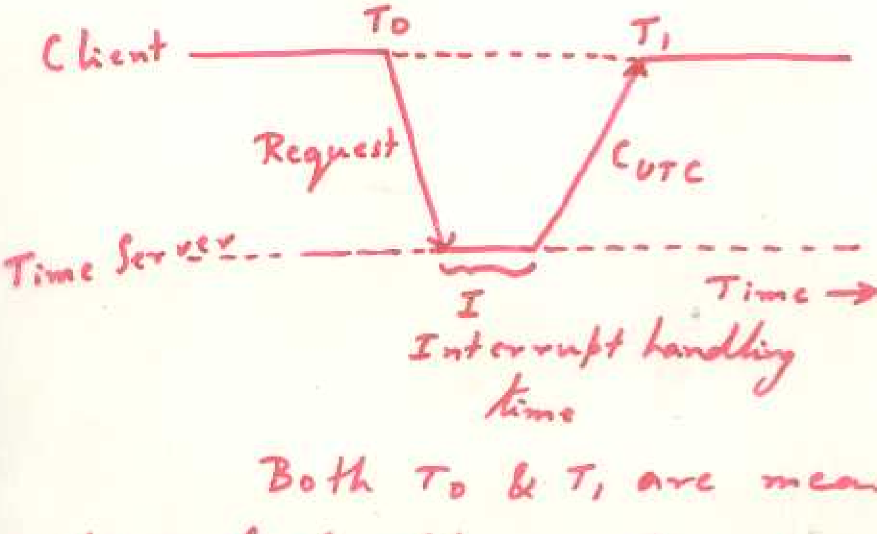
the timer can be said to be working within it's specification.

$\rho \rightarrow$ Max^m. drift rate.



Cristian's Algorithm :-

- It is well suited to systems in which one machine has a WWV (WWV broadcasts a short pulse at the start of each UTC second, a shortwave radio station in USA) receiver and the goal is to have all the other machines stay synchronized with it.
- The algo. is based on the works of Cristian (1989).
- The machine with WWV receiver is known as "Time Server"
- Periodically, certainly no more than every $\delta/2\rho$ seconds, each machine sends a message to the time server asking for the current time.
- The machine responds as fast as it can, with a message containing it's current time C_{UTC}.



Both T_0 & T_1 are measured with same clock.

- As a first approximation when the sender gets the reply, it can just sets it's clock to C_{UTC}.

Problems with the algorithm :-

① If the sender's clock is fast, C_{UTC} will be smaller than the sender's current value of C.

Soln: clock change must be introduced gradually, per interrupt basis.

② Delay in time server's reply due to network load.

The Best estimate of message propogation time is

$$\underbrace{(T_1 - T_0)/2}_{} \longleftarrow \text{(one way propogation)}.$$

When the reply comes in, the value in the message can be increased by this amount to give an estimate of the server's current time.

Considering the interrupt handling time I, the estimate will be

$$(T_1 - T_0 - I)/2 .$$

To improve the accuracy, Cristian suggested making not one measurement but a series of them. Any measurement in which $T_1 - T_0$ exceeds some threshold value are discarded.

The Berkeley Algorithm :-

- In Cristian's algorithm, the time server is passive. Other machines periodically asks it for time.
- In Berkeley UNIX, exactly the opposite approach is taken. (Gusella & Zatti, 1989).
- Here, the time server (a time daemon) is active, polling every machine from time to time to ask what time it is there.
- Based on the answers, it computes an average time and tells all the other machines to adjust their clocks.
- Time daemon's time must be set manually by the operator periodically.

Averaging Algorithms :-

- Decentralized algorithms.
- Works by dividing time into fixed length resynchronization intervals.
- The ith interval starts at $T_0 + iR$ and runs until $T_0 + (i+1)R$, where T_0 is an agreed upon moment in the past, and R is a system parameter.
- At the begining of each interval, every machine broadcasts the current time according to it's clock.
- After a machine broadcasts it's time, it starts a local timer to collect all other broadcasts that arrive during some interval S.

- When all the broadcasts arrive, an algorithm is run to compute a new time from them.
- The simplest algorithm is just to average the values from all the other machines.
- A slight variation on this theme is first to discard the m highest and m lowest values, and average the rest.

** Mostly used algorithm in the Internet is Network Time Protocol (NTP), described in (Mills, 1992). It has accuracy in the range of 1-50 msec.

Multiple External Time Sources:-

- For systems in which extremely accurate synchronization with UTC is required, it is possible to equip the system with multiple receivers for WWV, GEOS or other UTC Sources.
- Due to inherent accuracy in the time source & fluctuations in the signal path, The O.S generally establish a range in which UTC falls.
- As the various time sources produce different ranges, it is required that the machines attached to them come to a general agreement.
- To reach this agreement, each processor with a UTC Source can broadcast its range periodically, for instance at the start of each UTC minute.

Use of Synchronized Clocks:

- New algorithms are based on the use of synchronized clocks
- Liskov, 1993 proposed an algorithm which concerns how to enforce at-most-once message delivery to a server, even in the face of crashes.
- The traditional approach is for each message to bear a unique message number, and have each server store all the numbers of the messages it has seen, so that it can detect new messages from retransmissions.
- Problem with this algorithm is that
  - If a server crashes & reboots, it loses its table of message numbers.
  - For how long should message numbers be stored?
- The algorithm can be modified as follows:
  - Every message carries a connection identifier (chosen by the sender) and a timestamp.
  - For each connection, the server records in a table the most recent timestamp it has seen.
  - If any incoming message is lower than the timestamp stored, the message is rejected as a duplicate
  - To remove old timestamps a global variable is used.

  G = CurrentTime − MaxLifetime − MaxClockskew

Where,

MaxLifetime → Maxm. time a message can live

MaxClockskew → How far from UTC the clock might be at worst.

- Every ΔT, the current time is written to disk.
- When a server crashes & reboots, it reloads G from the time stored on disk and increments it by the update period ΔT.

LOGICAL CLOCKS:-

- For a certain class of algorithms, it is internal consistency of the clocks that matters, not whether they are close to a real time. For these algo's it is conventional to speak of the clocks as Logical Clocks.
- For Logical clock synchronization following algo's are used:-
  - Lamport's Algorithm (1978) / Time stamp
  - Vector Timstamp - Extension of Lamport's approach. (Lamport, 1990).
- If two processes do not interact then there is no need for synchronization of their logical clocks.

Lamport Time stamps:-

- To synchronize logical clocks, Lamport defined a clock called "happens before".

  eg: $a \to b$. a happens before b.

- The happens before relation can be observed directly in two situations.

  1. If a and b are events in the same process and a occurs before b, then $a \to b$ is true.
  2. If a is the event of a message being sent by one process, & b is the event of the message being received by another process, then $a \to b$ is also true.

- Happens before is a transitive relation.

  i.e if $a \to b$ & $b \to c$ then $a \to c$.

- If two events x & y, happen in different processes that do not exchange messages, then $x \to y$ is not true but neither is $y \to x$. These events are said to be concurrent.

- According to the algorithm. [$C(a)$ & $C(b)$ are time assigned]
  - If a happens before b in the same process $C(a) < C(b)$.
  - If a and b represent the sending and receiving of a message respectively, $C(a) < C(b)$.
  - For all distinctive events a & b, $C(a) \neq C(b)$.

Vector Timestamps:-

- Lamport timestamp lead to a situation that all events in a distributed system are totally ordered with the property that if event a happens before b, then a will also be positioned before b in ordering. i.e $C(a) < C(b)$.
- But if $C(a) < C(b)$, then this does not necessarily imply that a indeed happened before b.
- The problem is that Lamport's timestamp do not capture casuality.
- Casuality can be captured by means of Vector Timestamps.
- A Vector Timestamp $VT[a)$ assigned to an event a has the property that if $VT(a) < VT(b)$ for some event b, then event a is known to casually precede event b.
- Vector timestamps are constructed by letting each process P_i maintain a vector V_i with the following two properties:
  1. $V_i[i]$ is the number of events that have occured so far at P_i.
  2. If $V_i[j] = k$, then P_i knows that k events have occured at P_j.
- The first property is maintained by incrementing $V_i[i]$ at the occurance of each new event that happens at process P_i.
- The second property is maintained by piggybacking vectors along with messages that are sent.

GLOBAL STATE :-

- The Global State of a distributed system consists of the local state of each process, together with the messages that are currently in transit, i.e, that have been sent but not delivered.

ELECTION ALGORITHMS :-

- Algorithms for electing a coordinator process.

The Bully Algorithm :-

- Devised by Garcia-Molina (1982)

- When any process notices that the coordinator is no longer responding to requests, it initiates an election. A process P, holds an election as follows:

1. P sends an ELECTION message to all processes with higher numbers. (process numbers.)
2. If no one responds, P wins the election and becomes coordinator
3. If one of the higher-ups answers, it takes over, P's job is done.

The Ring Algorithm :-

- Based on the use of a Ring. Unlike some ring algorithms, this one does not use a token.
- We assume that the processes are physically or logically ordered, so that each process knows who its successor is.

- When any process notices that the coordinator is not functioning, it builds an ELECTION message containing it's own process number and sends the message to its successor.
- If the successor is down, the sender skips over the successor and goes to the next member along the ring. or one after that until a running process is located.
- At each step, the sender add its own process number to the list in the message effectively making itself a candidate to be elected as coordinator.
- Eventually, the message gets back to the process that started it all.
- That process recognizes this event when it receives an incoming message containing its own process number.
- At that point, the message type is changed to COORDINATOR and circulated once again, this time to inform everyone else who the coordinator is (the list member with highest process number) and who the members of the new ring are.
- When this message is circulated once, it is removed and everyone goes back to work.

MUTUAL EXCLUSION:-

- In single processor system, critical regions are protected using semaphores, monitors & similar constructs.
- For distributed systems critical regions & mutual exclusion can be implemented using following algorithms:

A Centralized Algorithm :.

- The most straightforward way to achieve mutual exclusion in a distributed system is to simulate how it is done in a one processor system.
- One process is elected as the coordinator (eg: the one running on the machine with the highest network address).
- Whenever a process wants to enter a critical region it sends a request message to the coordinator, asking for permission.
- If no other process is currently in that critical region, the coordinator sends back a reply granting permission.
- When the reply arrives, the requesting process enters the Critical Region.
- When the process exits the critical region, it sends a message to the coordinator releasing its exclusive access
- Now the access is given to next process in the queue.

A Distributed Algorithm

- Ricart & Agrawala (1981) devised this algorithm on the basis of Lamport's timestamp.
- The algorithm requires that there be a total ordering of all events in the system.
- The algorithm works as follows:
- When a process wants to enter a critical region, it builds a message containing the name of the critical region, it's process number & the current time.
- It then sends the message to all other processes.
- When a process receives a request message from another process, the action it takes depends on its state w.r.to the critical region named.
  - Three cases have to be distinguished
  - i) If the receiver is not in the critical region and does not want to enter it, back an OK message to the sender.
  - ii) If the receiver is already in the critical region it does not reply, instead it queues the request.
  - iii) If the receiver wants to enter the critical region, it compares the timestamp of incoming message with it's own. The lowest one wins.

- After sending out requests, a process sits back and waits until everyone else has given permission.
- As soon as all the permissions are in, it may enter the critical region.
- When it exits the critical region, it sends OK messages to all the processes on it's queue & deletes them all from the queue.

A Token Ring Algorithm :-

- In s/w a logical ring is constructed in which each process is assigned a position in the ring. The ring positions may be allocated in numerical order of network addresses.
- When the ring is initialized, Process 0 is given a token
- It is passed from process K to process K+1 in point to point messages.
- When a process acquires the token, it checks to see if it is attempting to enter the Critical region.
- If so, the process enters the region & exists after doing it's work.
- After exit, it passes the token along the ring.
- It is not permitted to enter the critical region using the same token.
- If a process is not interested to enter the critical region, it just passes the token to its next neighbour.

Distributed Deadlock Handling :-

* Deadlock Prevention :-

- The deadlock prevention algorithms available for Centralized systems can be used in Distributed Systems after modifications.
- A Resource - Ordering prevention technique can be used by defining a global ordering among the system resources. That is all resources in the entire system are assigned unique numbers and a process may request a resource at any processor with unique number 'i' only if it is not holding a resource with a unique number greater than 'i'.
- Banker's algorithm can also be used in a distributed system by designating one of the processes in the system (the banker) as the process that maintains the information necessary to carry out the banker's algorithm. Every resource request must be channeled through banker.
- Deadlock prevention scheme can also be developed using timestamp - ordering with resource preemption.

- To control the preemption, a unique priority number is assigned to each process. These numbers are used whether a process P_i should wait for a process P_j.
- For eg. P_i can wait for P_j if P_i has a priority higher than that of P_j, otherwise P_i is rolled back.
- This scheme prevents deadlocks because for every edge $P_i \rightarrow P_j$ in the wait-for graph, P_i has a higher priority than P_j. Thus a cycle cannot exist.
- But starvation of lower priority processes may occur with this scheme.
- This difficulty can be avoided using timestamps.
- * The Wait-Die Scheme :- This approach is based on a nonpreemptive technique. When process P_i requests a resource currently held by P_j, P_i is allowed to wait only if it has a smaller timestamp than does P_j (i.e P_i is older than P_j). Otherwise P_i is rolled back (dies).
- * The Wound-Wait Scheme :- This approach is based on a preemptive technique. When process P_i requests a resource held by P_j, P_i is allowed to wait only if it has a larger timestamp than does P_j (i.e

P_i is younger than P_j). Otherwise P_j is rolled back (P_j is wound by P_i).

* **Deadlock Detection :-**

- Deadlock detection algorithm can be used to avoid unnecessary preemptions that occurs in a deadlock-prevention algorithm.
- A Wait-for graph is constructed for the system.
- A Global wait-for graph is constructed using several local wait-for graphs.

$$\begin{tikzcd}
P_1 & & P_2 \arrow[ll] \arrow[d] \\
P_5 \arrow[u] & & P_3
\end{tikzcd}$$

Site A

$$\begin{tikzcd}
P_2 & & P_4 \arrow[ll] \\
P_3 \arrow[urr] & &
\end{tikzcd}$$

Site B

$$\begin{tikzcd}
P_1 & P_2 \arrow[l] \arrow[d] & P_4 \arrow[l] \\
P_5 \arrow[u] & P_3 \arrow[ur] &
\end{tikzcd}$$

Global Wait for graph

This graph contains a cycle, $\Rightarrow$ The system is in a Deadlocked State.

Methods for organizing the Wait-for graph in a Distributed System:-

* Centralized Approach :-

- A global wait-for graph is constructed as the union of all the local wait-for graphs.
- It is maintained in a single process : The deadlock detection coordinator.
- Since, there is communication delay in the system, two types of wait-for graphs are used.
  * The Real graph describes the real but unknown state of the system at any instance in time.
  * The Constructed graph is an approximation generated by the coordinator during the execution of it's algorithm.
- The algorithm is as follows:-
  1. The controller sends an initiating message to each site in the system.
  2. On receiving this message, a site sends it's local wait-for graph to the coordinator.
  3. When the controller has received a reply from each site, it constructs a graph as follows:-

a. The constructed graph contains a vertex for every process in the system.

b. The graph has an edge $P_i \rightarrow P_j$ if and only if-

i) There is an edge $P_i \rightarrow P_j$ in one of the wait-for graphs. or

ii) An edge $P_i \rightarrow P_j$ with some label 'TS' appears in more than one wait-for graph.

- If there is no cycle in the constructed graph, then the system is not in Deadlocked State.

* Fully Distributed Approach :-

- All controllers share equally the responsibility for detecting deadlock.

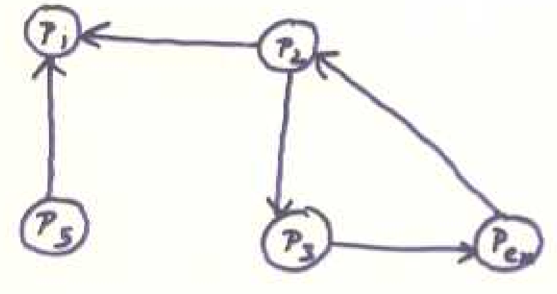
- In this scheme, every site constructs a wait-for graph that represents a part of the total graph, depending on the dynamic behaviour of the system.

- The idea is that, if a deadlock exists, a cycle will appear in at least one of the partial graphs.

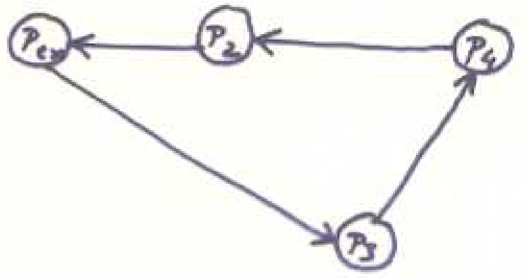
- Each site maintains its local wait for graph. These graphs have one extra node P_{ex}.

- An arc $P_i \rightarrow P_{ex}$ exists in the graph if P_i is waiting for a data item in another site being held by any process

- An arc $P_{ex} \rightarrow P_j$ exists, if there exists a process in another site that is waiting to acquire a resource currently being held by P_j in this local site.



Site A



Site B

- If a local wait for graph contains a cycle that does not involve node P_{ex}, then the system is in a deadlocked state.
- If, there exists a cycle involving P_{ex}, then this implies that there is a possibility of deadlock. A distributed deadlock detection algorithm can be used to verify it.