



INICIO GRABACIÓN



SANJOSÉ
FUNDACIÓN DE EDUCACIÓN SUPERIOR

The background features a photograph of two hands shaking in a firm grip, symbolizing agreement or partnership. This image is overlaid with a large, semi-transparent dark blue circle on the right side and a smaller, semi-transparent light blue circle on the left side, which contains the text.

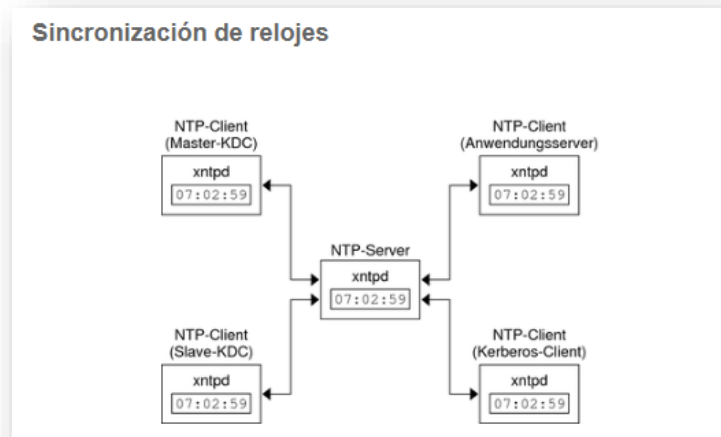
SINCRONIZACION EN SISTEMAS DISTRIBUIDOS

SINCRONIZACION EN RELOJES



Un sistema distribuido debe permitir el apropiado uso de los recursos, debe encargarse de un buen desempeño y de la consistencia de los datos, además de mantener seguras todas estas operaciones. La sincronización de procesos en los sistemas distribuidos resulta más compleja que en los centralizados, debido a que la información y el procesamiento se mantienen en diferentes nodos.

Un sistema distribuido debe mantener vistas parciales y consistentes de todos los procesos cooperativos y de cómputo. Tales vistas pueden ser provistas por los mecanismos de sincronización. El término sincronización se define como la forma de forzar un orden parcial o total en cualquier conjunto de eventos, y es usado para hacer referencia a tres problemas distintos pero relacionados entre sí:



SINCRONIZACION EN RELOJES



1. La sincronización entre el emisor y el receptor.
2. La especificación y control de la actividad común entre procesos cooperativos.
3. La serialización de accesos concurrentes a objetos compartidos por múltiples procesos. Haciendo referencia a los métodos utilizados en un sistema centralizado, el cual hace uso de semáforos y monitores; en un sistema distribuido se utilizan algoritmos distribuidos para sincronizar el trabajo común entre los procesos y estos algoritmos

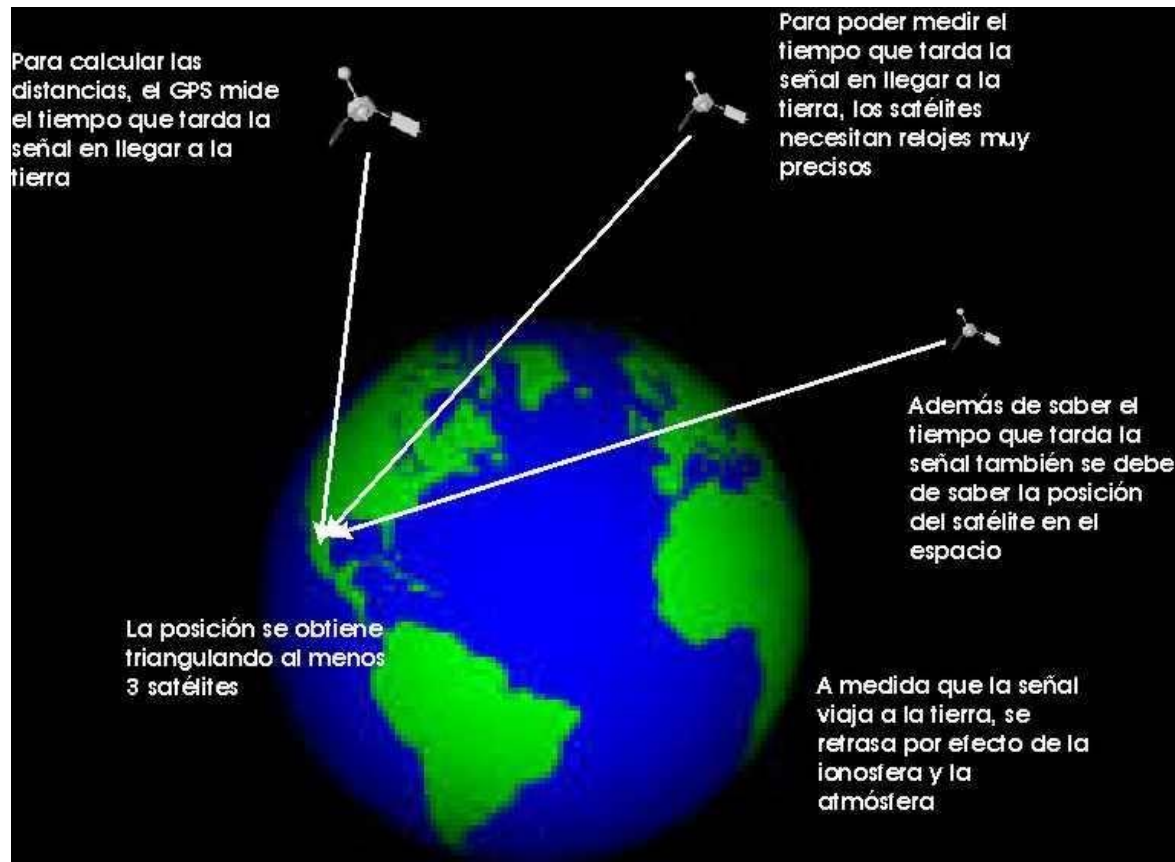
Casi todas las computadoras tienen un circuito para dar seguimiento al tiempo. A pesar del amplio uso de la palabra “reloj” para hacer referencia a estos dispositivos, en realidad no son relojes en el sentido usual. Tal vez cronómetro sea una mejor palabra con la cual designarlos.



Sistema de posicionamiento global



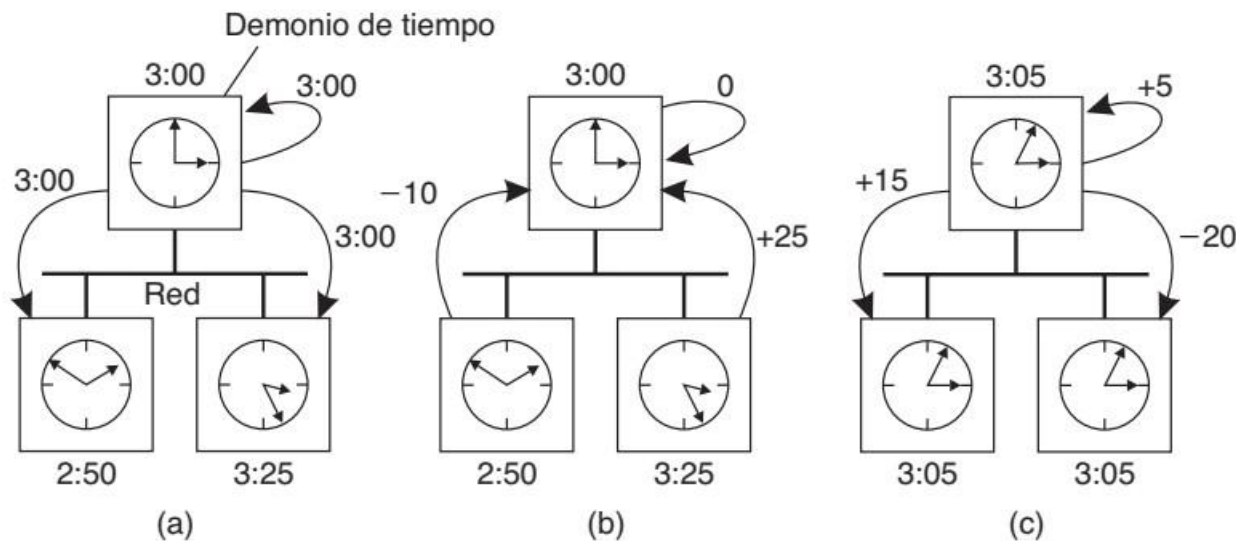
El posicionamiento se establece a través de un sistema distribuido altamente específico y dedicado llamado GPS (por sus siglas en inglés), y que significa sistema de posicionamiento global.




Algoritmos de sincronización de relojes




Todos los algoritmos tienen como base el mismo modelo del sistema. Se supone que cada máquina tiene un cronómetro que ocasiona una interrupción H veces por segundo. Cuando este cronómetro se apaga, el manipulador de interrupciones agrega 1 al reloj de software, el cual da seguimiento al número de marcas (interrupciones) a partir de algún momento pasado acordado.



(a) El demonio de tiempo pregunta a las otras máquinas los valores de sus relojes. (b) Las máquinas responden. (c) El demonio de tiempo les indica cómo ajustar sus relojes.



Algoritmos de sincronización de relojes



La sincronización de relojes en un sistema distribuido consiste en garantizar que los procesos se ejecuten en forma cronológica y a la misma vez respetar el orden de los eventos dentro del sistema. Para lograr esto existen varios métodos o algoritmos que se programan dentro del sistema operativo, entre los cuales tenemos:

Algoritmo de Cristian

Algoritmo de Berkeley






ALGORITMO DE CRISTIAN




Este algoritmo está basado en el uso del tiempo coordinado universal (siglas en inglés, UTC), el cual es recibido por un equipo dentro del sistema distribuido. Este equipo, denominado receptor de UTC, recibe a su vez solicitudes periódicas del tiempo del resto de máquinas del sistema a cada uno de los cuales les envía una respuesta en el menor plazo posible informando el tiempo UTC solicitado, con lo cual todas las máquinas del sistema actualicen su hora y se mantenga así sincronizado todo el sistema. El receptor de UTC recibe el tiempo a través de diversos medios disponibles, entre los cuales se menciona las ondas de radio, Internet, entre otros. Un gran problema en este algoritmo es que el tiempo no puede correr hacia atrás:

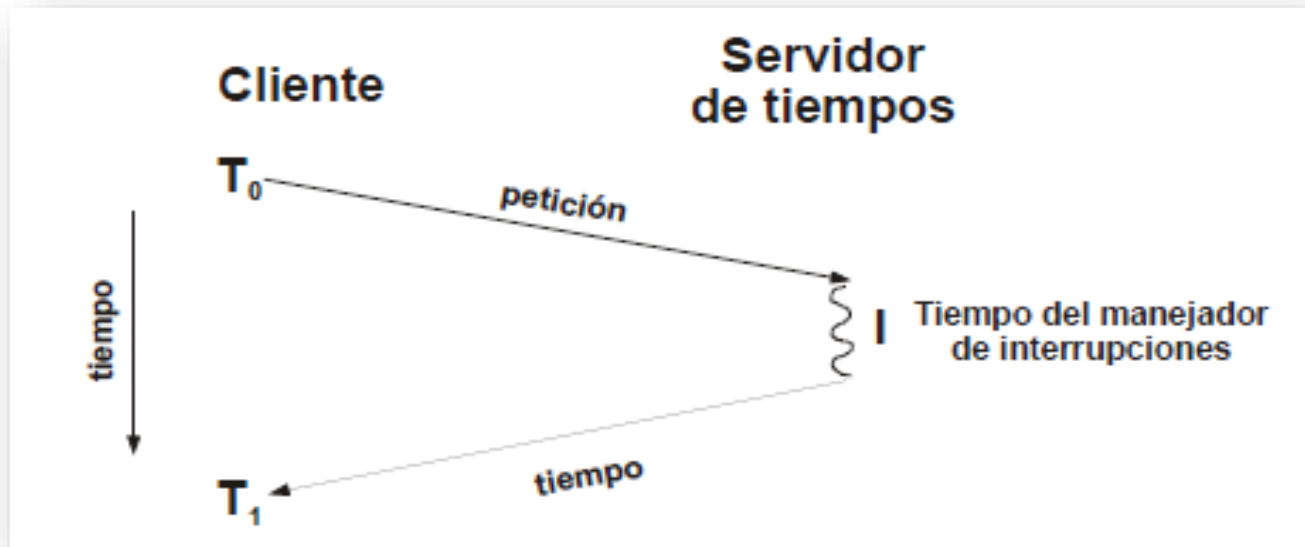
- El tiempo del receptor UTC no puede ser menor que el tiempo de la máquina que le solicitó el tiempo.
 - El servidor de UTC debe procesar las solicitudes de tiempo con el concepto de interrupciones, lo cual incide en el tiempo de atención.
 - El intervalo de transmisión de la solicitud y su respuesta debe ser tomado en cuenta para la sincronización. El tiempo de propagación se suma al tiempo del servidor para sincronizar al emisor cuando éste recibe la respuesta.
- 



ALGORITMO DE CRISTIAN



- El tiempo del receptor UTC no puede ser menor que el tiempo de la máquina que le solicitó el tiempo.
 - El servidor de UTC debe procesar las solicitudes de tiempo con el concepto de interrupciones, lo cual incide en el tiempo de atención.
 - El intervalo de transmisión de la solicitud y su respuesta debe ser tomado en cuenta para la sincronización. El tiempo de propagación se suma al tiempo del servidor para sincronizar al emisor cuando éste recibe la respuesta.
 - El tiempo de propagación se suma al tiempo del servidor para sincronizar al emisor cuando éste recibe la respuesta.
 - Los relojes no deben retroceder para evitar errores
- 



$$\frac{T_{fc} - T_{ic}}{2} + T_{serv} = T_c$$

Donde:

T_{fc} = Tiempo final cliente

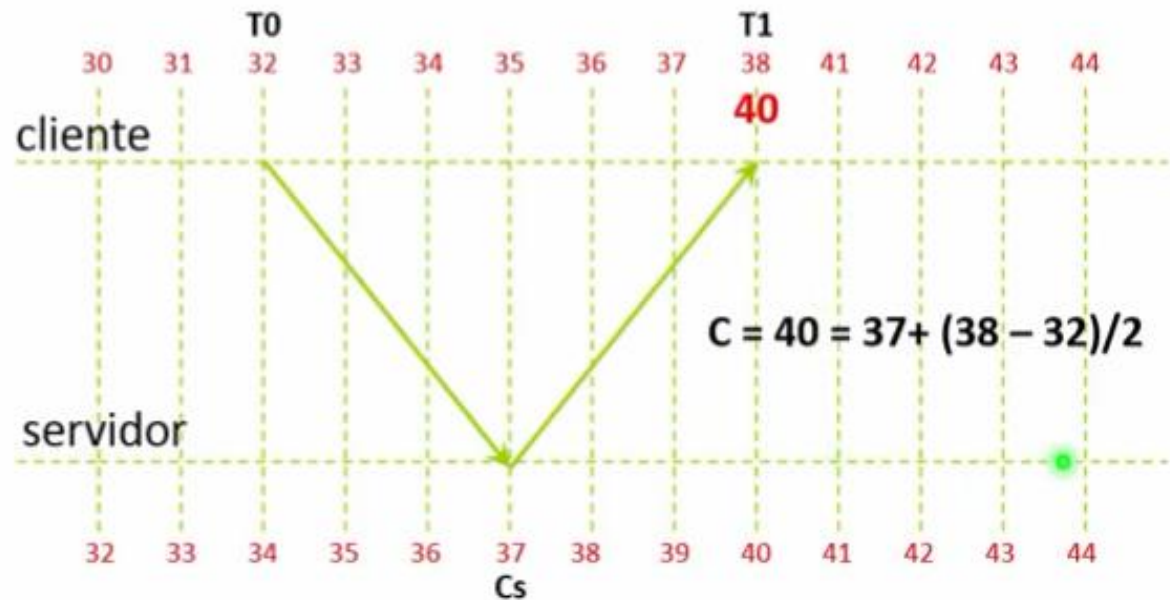
T_{ic} = Tiempo inicial cliente

T_{serv} = Tiempo servidor

T_c : Tiempo que debe tener el cliente

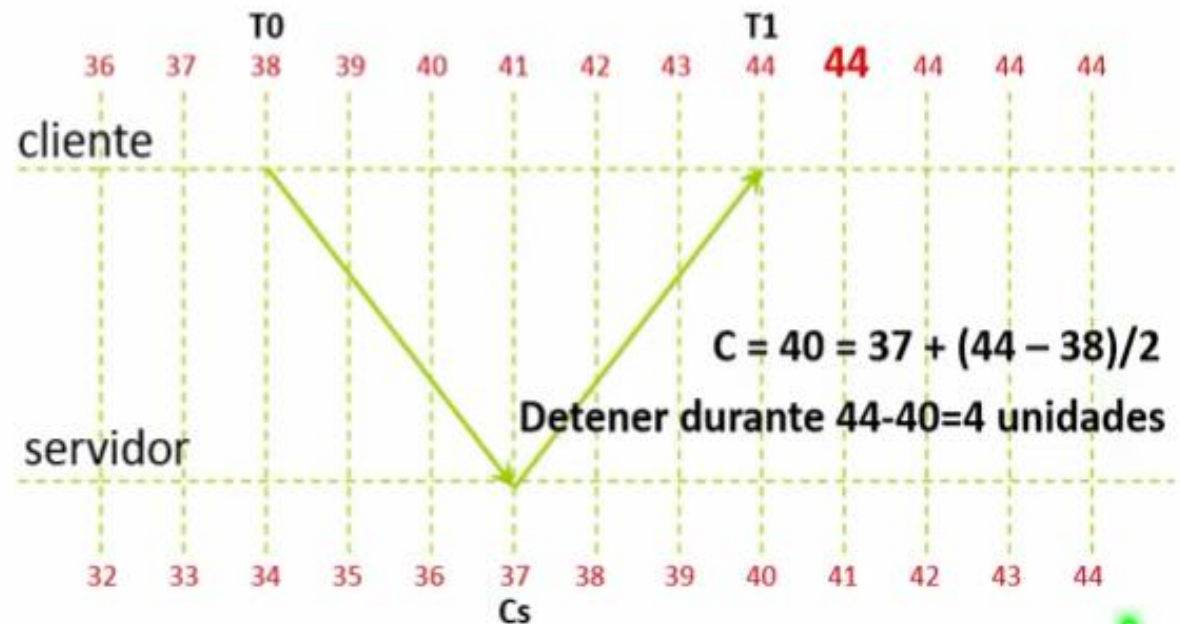
Descripción

- ▶ El cliente pide el valor del reloj al servidor en **T0** (según **Cc**)
- ▶ El servidor contesta con el valor de su reloj **Cs**
- ▶ La respuesta llega al cliente en **T1** (según **Cc**)
- ▶ $C = Cs + (T1 - T0)/2$
- ▶ Si $C > Cc$, $Cc = C$
- ▶ Si $C < Cc$, se detiene Cc las siguientes $Cc - C$ unidades de tiempo



Descripción

- ▶ El cliente pide el valor del reloj al servidor en **T0** (según **Cc**)
- ▶ El servidor contesta con el valor de su reloj **Cs**
- ▶ La respuesta llega al cliente en **T1** (según **Cc**)
- ▶ **$C = C_s + (T1 - T0)/2$**
- ▶ Si **$C > C_c$** , **$C_c = C$**
- ▶ Si **$C < C_c$** , se detiene **C_c** las siguientes **$C_c - C$** unidades de tiempo






INCONVENIENTES ALGORITMO CRISTIAN



El problema que se presenta es la posibilidad de fallo debido a la existencia de un único servidor. Cristian sugiere múltiples servidores de tiempo sincronizados que suministren el tiempo. El cliente envía un mensaje de petición a todos los servidores y toma la primera respuesta recibida.

El algoritmo no contempla problemas de malfuncionamiento o fraude por parte del servidor.

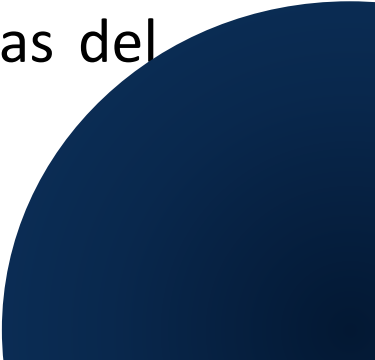
La capacidad de cada nodo para leer el valor del reloj de otro nodo puede generar errores debido al retraso en la comunicación de mensajes entre nodos. La demora se puede calcular tomando en cuenta el tiempo necesario para preparar, transmitir y recibir un mensaje vacío en ausencia de errores de transmisión y carga del sistema.





ALGORITMO BERKELEY



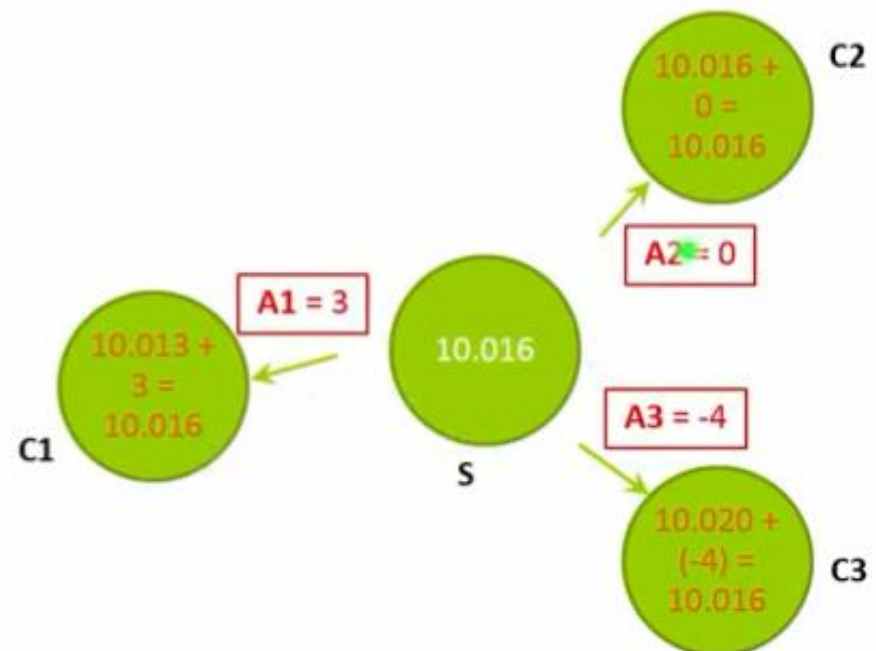
- Un sistema distribuido basado en el algoritmo de Berkeley no dispone del tiempo coordinado universal (UTC); en lugar de ello, el sistema maneja su propia hora. Para realizar la sincronización del tiempo en el sistema, también existe un servidor de tiempo que, a diferencia del algoritmo de Cristian, se comporta de manera activa. Este servidor realiza un muestreo periódico del tiempo que poseen algunas de las máquinas del sistema, con lo cual calcula un tiempo promedio, el cual es enviado a todas las máquinas del sistema a fin de sincronizarlo.
- 



Descripción

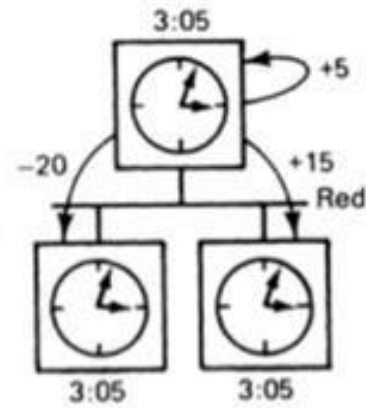
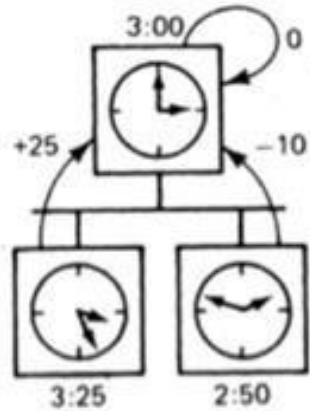
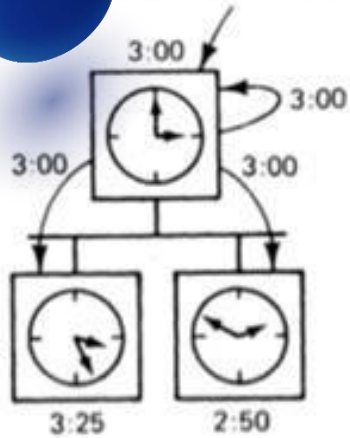
- Cada cliente ajusta su reloj incrementándolo en **A_i**
(si el ajuste es negativo, se programa un ajuste gradual)

T0	10.000
D1	10.003 - 10.000 = 3
D2	10.006 - 10.000 = 6
D3	10.010 - 10.000 = 10
T1i	10.010
D1'	3 - (10010 - 10000)/2 = -2
D2'	6 - (10010 - 10000)/2 = 1
D3'	10 - (10010 - 10000)/2 = 5
D	(-2 + 1 + 5 + 0)/4 = 1
A1	1 - (-2) = 3
A2	1 - 1 = 0
A3	1 - 5 = -4

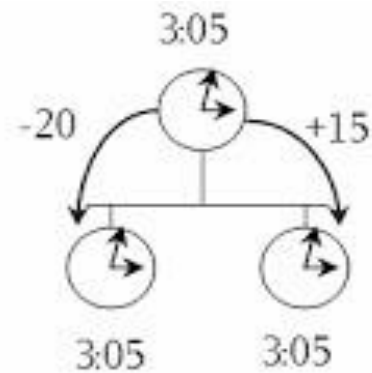
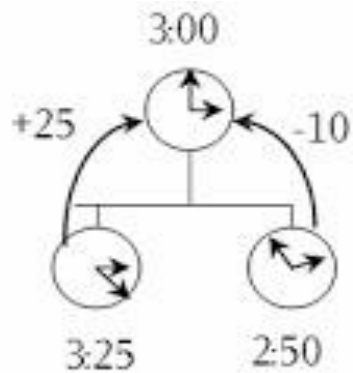
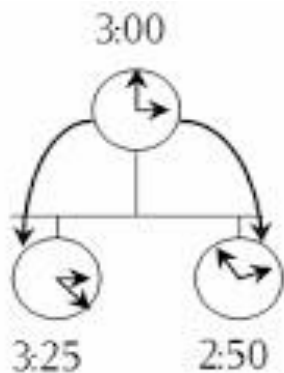






Demonio para el tiempo




Hora	V resta	*(-1)	+ 5	+ hora	=
3:00	0	0	$0+5=5$	$5+3:00$	3:05
3:25	+25	-25	$-25+5=-20$	$-20+3:25$	3:05
2:50	-10	10	$10+5=15$	$15+2:50$	3:05





```
1.1. //Número de esclavos que participan en la sincronización x
2.2. Nesclavos=x
3.3. suma=0
4.4. //Obtenemos el tiempo de los esclavos y calculamos la diferencia con el
tiempo del 5.maestro
6.5. tiempo_Esclavos[]=PedirTiempoEsclavos();
7.6. for(i=0; i<Nesclavos;i++){
8.7. diferencia_tiempos[i]= - tiempo_Esclavo[i] -TiempoMasterActual();
9.8. }
10.9. //Calculamos la diferencia media .
11.10. for(i=0;i<Nesclavos;i++){
12.11. suma+=diferencia_tiempos[i];
13.12. }
14.//Debemos tener también en cuenta el servidor a la hora de hacer la media de ahí que sumemos +1
15.13. diferencia_media=suma/(Nesclavos+1);
16.14. //Calculamos tiempo de sincronización del maestro sumándole la deriva
media.
17.15. tiempo_sicronizacionMaestro= TiempoMasterActual()+ diferencia_media
18.16. // Enviamos deriva de cada reloj para que ajuste su tiempo y se
sincronicen.
19.17. for(i=0;i<Nesclavos;i++){
20.18. enviar_Deriva(diferencia_media, i )
21.19. }
```





INCONVENIENTES ALGORITMO BERKELEY

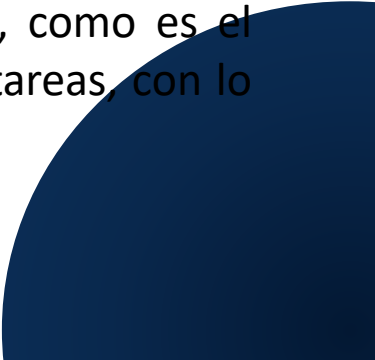


En caso de que el maestro falle, dejará de preguntar la hora y, en consecuencia, se perderá la sincronización.

En el paso de mensajes entre maestro y esclavos y viceversa se ha de tener en cuenta el retraso que genera el propio envío de los mensajes por lo que tendremos errores a la hora de sincronizar.

Otro de los problemas de ser un algoritmo centralizado será el de tener que elegir un nuevo maestro debido a que se puedan producir fallos en el tratamiento de los datos para llevar a cabo la sincronización, buscando un maestro que cometa menor cantidad de fallos y el sistema se mantenga en correcto funcionamiento.

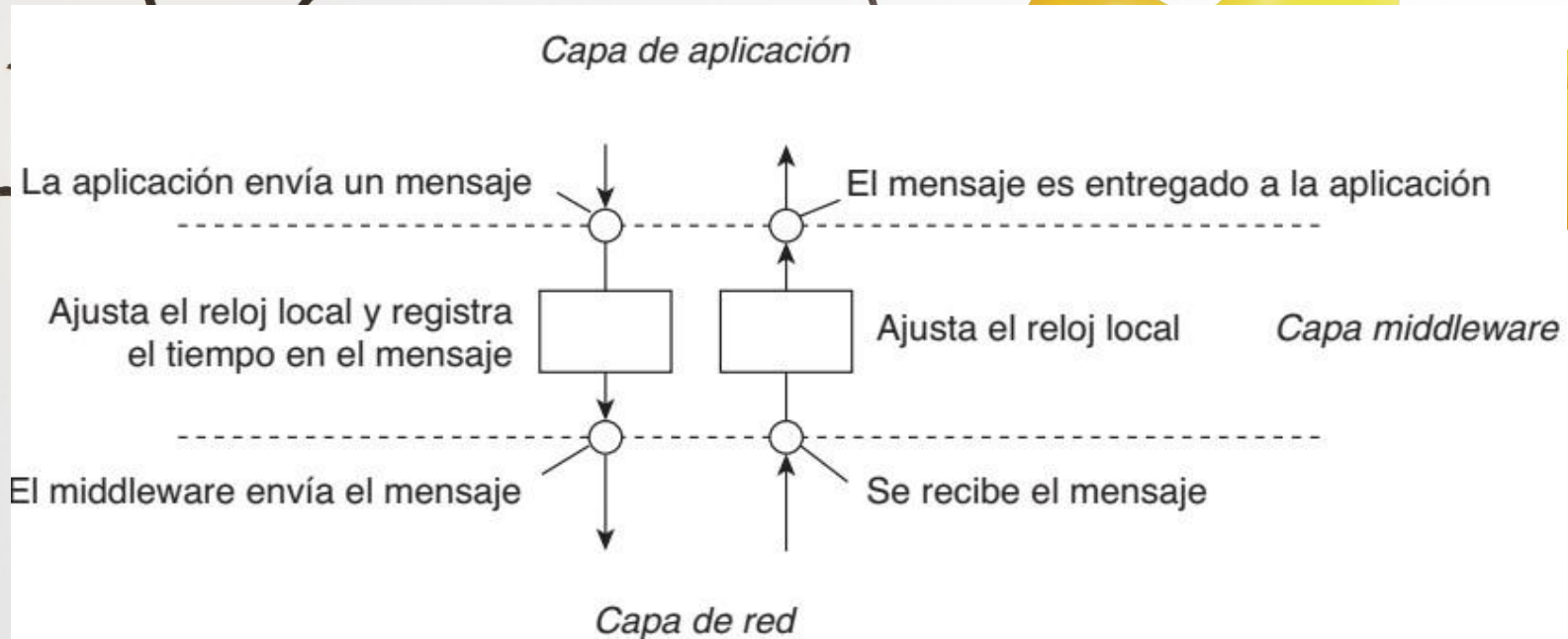
. La demora introducida por los equipos de conmutación, se agrava en los casos en los que el equipo cumple otras funciones además de conmutar paquetes, como es el caso de un PC que, además de actuar como conmutador, atiende otras tareas, con lo cual, la misma tarea de conmutar puede insumir diferentes tiempos.





Relojes lógicos de Lamport

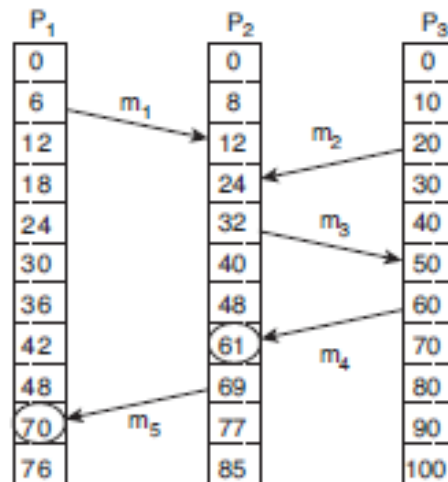
Indican el orden en que suceden ciertos eventos, no el instante real en que suceden ya que no dependen de relojes reales, para sincronizar los relojes lógicos, Lamport definió una relación llamada *ocurrencia anterior*. La expresión $a \rightarrow b$ se lee como “a ocurre antes que b”, y significa que todos los procesos coinciden en que ocurre el primer evento a y, después de eso, ocurre el evento b.



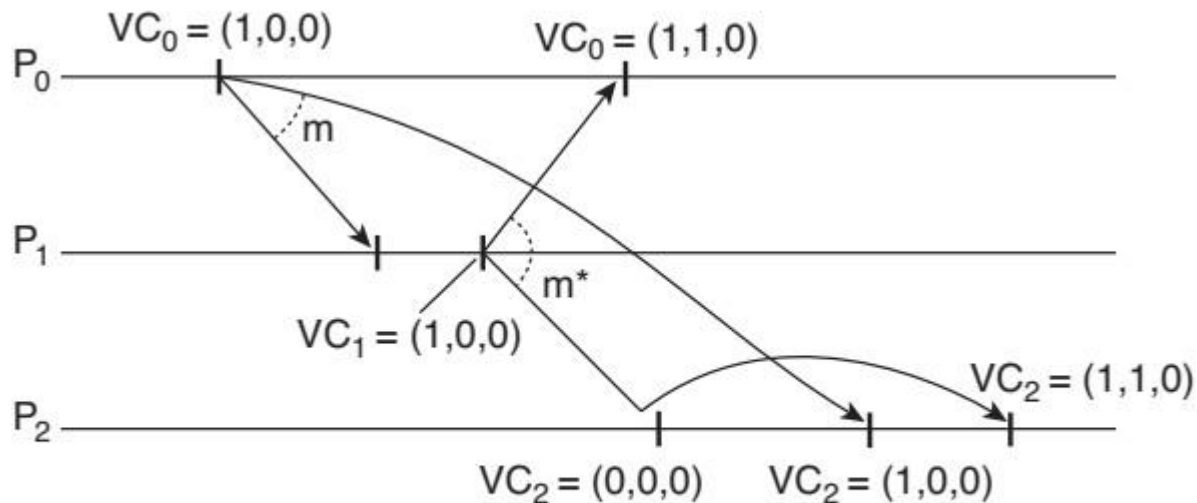
Relojes vectoriales



consideremos los mensajes enviados por los tres procesos que muestra la figura. Denotamos como $T_{env}(m_i)$ al tiempo lógico en que se envió el mensaje m_i , y de igual modo, como $T_{rec}(m_i)$ al tiempo de recepción. Por construcción, sabemos que para cada mensaje $T_{env}(m_i) < T_{rec}(m_j)$. ¿Pero qué podemos concluir en general a partir de $T_{env}(m_i) < T_{rec}(m_j)$?



Asocian un valor vectorial a cada evento, $VC(a)$, asignado a un evento a , tiene la propiedad de que si $VC(a) < VC(b)$ para algún evento b , entonces se sabe que el evento a precede en causalidad al evento b .



Imposición de la comunicación causal.



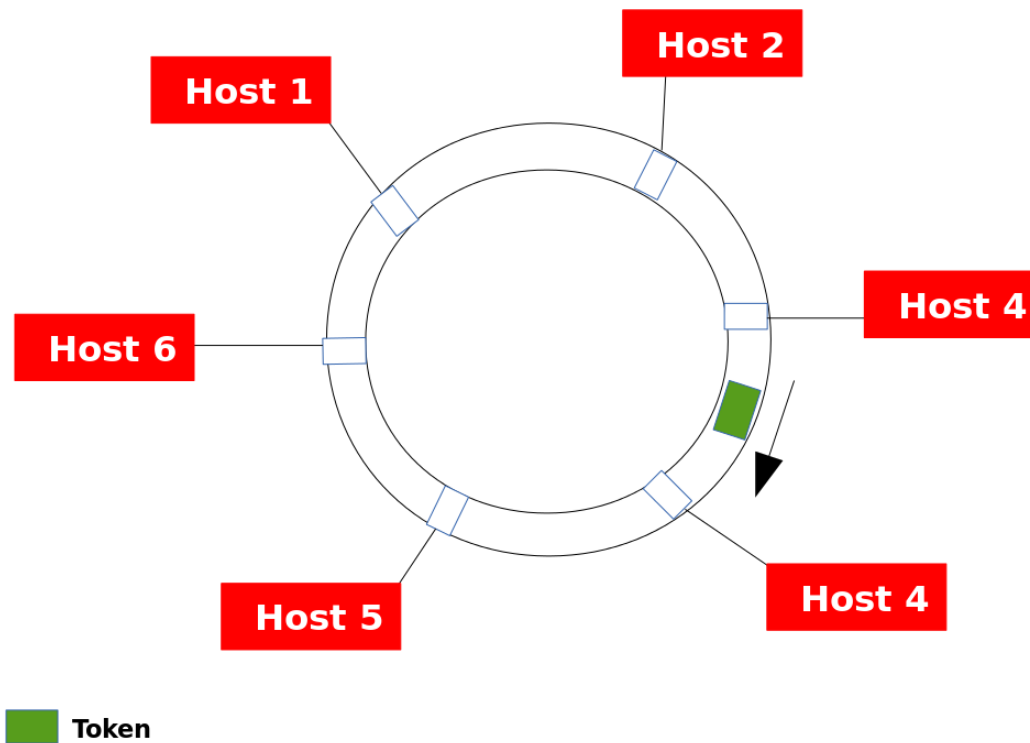
EXCLUSIÓN MUTUA

Visión general



Los algoritmos distribuidos de exclusión mutua pueden clasificarse en dos diferentes categorías.

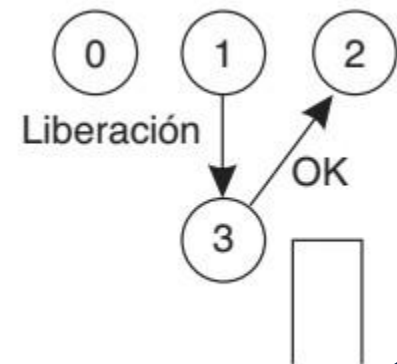
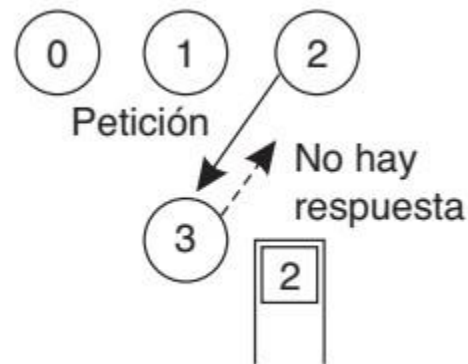
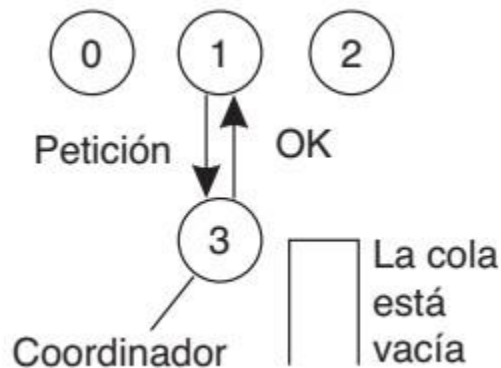
En las soluciones basadas en token, la exclusión mutua se logra pasando entre los procesos un mensaje especial conocido como token. Sólo hay un token disponible, y quien lo tenga puede acceder al recurso compartido. Cuando termina, el token pasa al siguiente proceso. Si un proceso tiene el token pero no está interesado en acceder al recurso, simplemente lo pasa.



Un algoritmo centralizado



En un sistema distribuido, la manera más directa de lograr la exclusión mutua es simular lo que se hace en un sistema de un procesador. Se elige un proceso como coordinador. Siempre que un proceso desea acceder a un recurso compartido, envía un mensaje de petición al coordinador mencionando el recurso al que desea acceder, y solicita permiso. Si ningún otro proceso está accediendo al recurso en ese momento, el coordinador devuelve una respuesta en la que otorga el permiso.

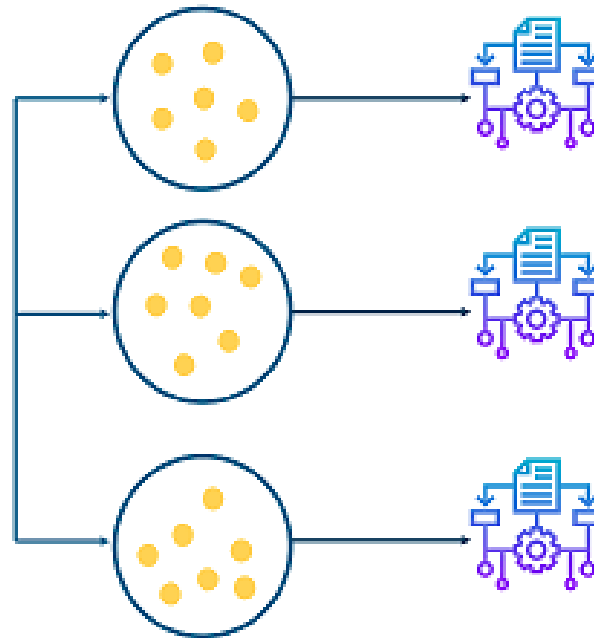


Un algoritmo descentralizado



Se supone que cada recurso se replica n veces. Cada réplica tiene su propio coordinador para controlar el acceso de procesos concurrentes.

Sin embargo, siempre que un proceso desee acceder al recurso, éste simplemente tendrá que lograr una votación mayoritaria a partir de $n/2$ coordinadores. A diferencia del esquema centralizado, asumimos que cuando un coordinador no otorga el permiso para acceder a un recurso (lo que hará cuando haya otorgado el permiso a otro proceso), se lo informa al solicitante.

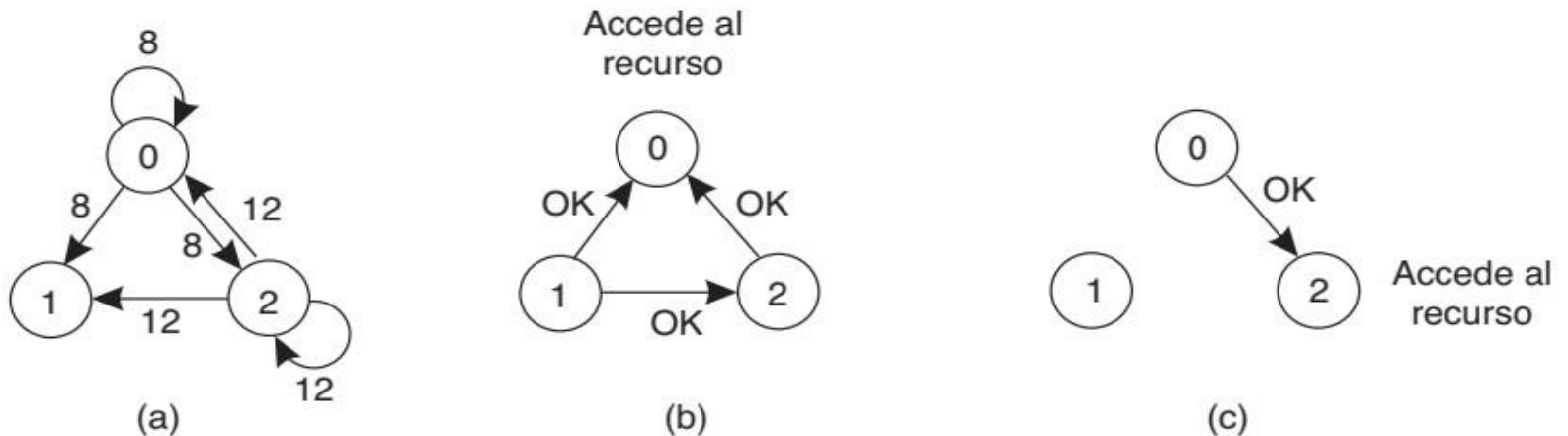


Un algoritmo distribuido



Cuando un proceso desea acceder a un recurso compartido, elabora un mensaje que contiene el nombre del recurso, su número de proceso, y el tiempo actual (lógico). Entonces envía el mensaje a todos los demás procesos, incluyéndose de manera conceptual. Se supone que el envío de los mensajes es confiable; es decir, no se pierde mensaje alguno.

Cuando un proceso recibe un mensaje de petición de otro proceso, la acción que tome dependerá de su propio estado con respecto al recurso mencionado en el mensaje.

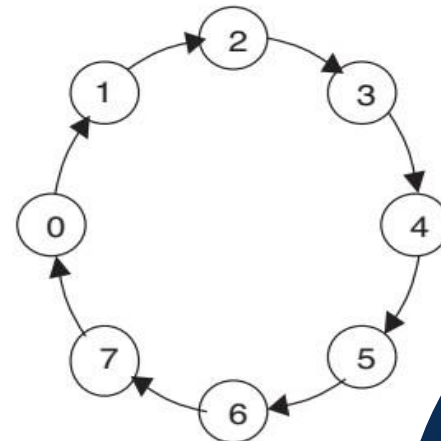
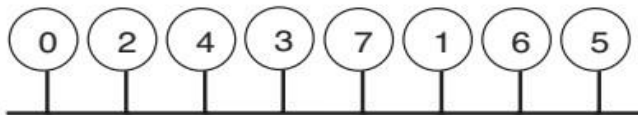


Un algoritmo de anillo de token



En software, un anillo lógico se construye con cada proceso asignado a una posición en el anillo. En el anillo, las posiciones se pueden localizar con el orden numérico de las direcciones de red o por otros medios. No importa cuál es el orden. Todo lo que importa es que cada proceso sabe cuál es el siguiente después de él.

Cuando se inicia el anillo, al proceso 0 se le asigna un token. El token circula alrededor del anillo. Se pasa desde el proceso k al proceso $k + 1$ (modula el tamaño del anillo) en los mensajes punto a punto. Cuando un proceso adquiere el token de su vecino, verifica si necesita acceder al recurso compartido. Si es así, el proceso sigue adelante, hace el trabajo que requiere hacer, y libera los recursos.



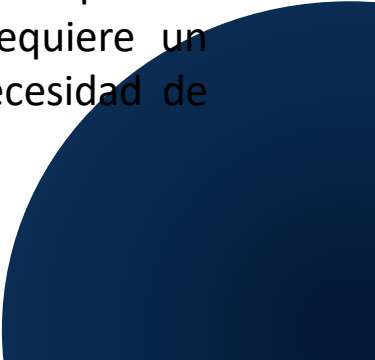


POSICIONAMIENTO GLOBAL DE LOS NODOS

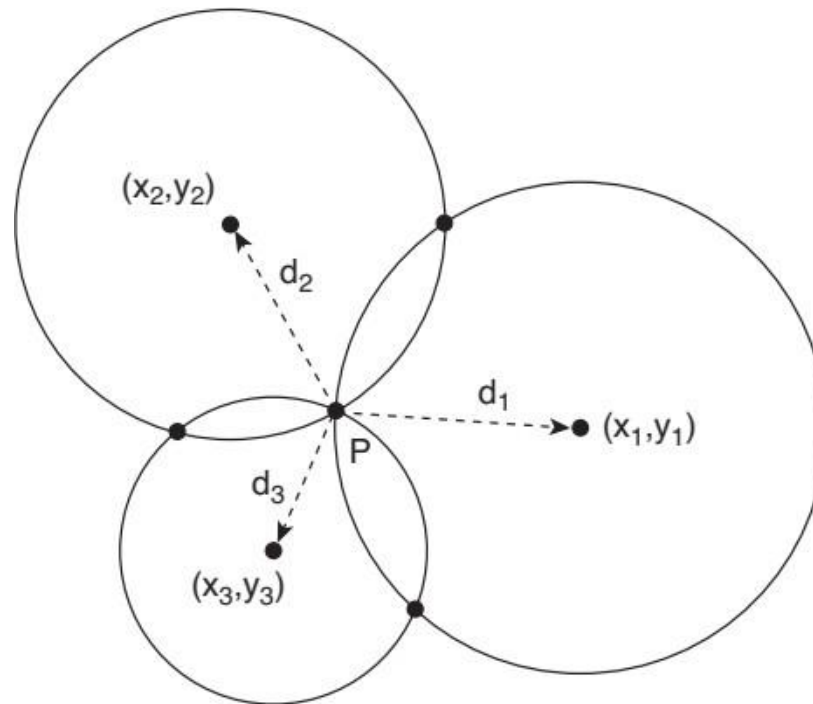


En redes geométricas sobrepuestas, a cada nodo se le asigna una posición dentro de un espacio geométrico m -dimensional, tal que la distancia entre dos nodos en dicho espacio refleja una métrica de rendimiento en el mundo real. El ejemplo más simple, y más aplicado, es en donde la distancia se corresponde con la latencia internodal. En otras palabras, dados dos nodos P y Q , entonces la distancia $d(P,Q)$ refleja el tiempo que le toma a un mensaje viajar desde P hacia Q y viceversa.

Existen muchas aplicaciones para las redes geométricas. Considere una situación en donde un sitio web en el servidor O es replicado en múltiples servidores S_1, \dots, S_k en internet. Cuando un cliente C solicita una página desde O , este último pudiera decidir redireccionar una petición hacia el servidor más cercano a C , esto es, aquel que da el mejor tiempo de respuesta. Si conocemos la ubicación geométrica de C , así como la de cada réplica del servidor, entonces O puede simplemente elegir al servidor S_i para el cual $d(C, S_i)$ es mínima. Observemos que dicha selección solamente requiere un procesamiento local en O . En otras palabras, no existe, por ejemplo, necesidad de mostrar todas las latencias entre C y cada uno de los servidores replicados.



POSICIONAMIENTO GLOBAL DE LOS NODOS





FUNDACIÓN DE EDUCACIÓN SUPERIOR

SAN JOSÉ

INSTITUCIÓN TECNOLÓGICA

FIN DE
GRABACIÓN