Sincronización de Relojes en Ambientes Distribuidos

Autor: Fernando L. Romero
Director: Fernando G. Tinetti

Instituto de Investigación en Informática LIDI (IIILIDI)
Facultad de Informática – UNLP

Tesis presentada para la aprobación del
Master en Redes de Datos
Febrero de 2009

Índice

Capítulo 1: Alcances y objetivos de la tesis

Resumen

Las mediciones de tiempo utilizando una computadora han llevado a estudiar las diferentes posibles herramientas presentes en los equipos actuales. Estas herramientas presentan características, no solo debido al hardware, sino al uso que el software hace del mismo. Con el advenimiento de los sistemas distribuidos, aparece la necesidad de mediciones en equipos remotos conectados, que lleva a tener que sincronizarlos entre sí en hora. Dicho sincronismo no es trivial, sobre todo si se quieren valores precisos. El tema adquiere complejidad por incluir no solo los aspectos relativos a los diferentes relojes empleados, sino también los problemas por comunicar las referencias de tiempo a través de la red de comunicaciones entre computadoras, tema necesario en caso de requerir sincronización. Si se desea medir las comunicaciones mismas, los tiempos de envío de mensajes entre computadora en redes locales requieren resoluciones de tiempo del orden de microsegundos y además que la sincronización de relojes remotos no difiera de ese orden. En este capítulo se introduce el tema que se abordará en el resto de la tesis, los objetivos y la organización del trabajo para arribar a los mismos.

1.1 Introducción

En la presente tesis se aborda el problema de la sincronización de relojes en ambientes distribuidos. Si bien se estudian muchos de los aspectos del mismo, se profundizan aquellos a los que están orientados los objetivos. Uno de dichos objetivos se orienta a solucionar el problema de las mediciones de rendimiento.

El problema de la evaluación de rendimiento se estudia tanto para mejorar los aspectos del procesamiento y de comunicaciones. A causa de las crecientes necesidades del área de sistemas distribuidos en cuanto a velocidad, se ensayan en la actualidad propuestas diversas tendientes a mejorar la velocidad de procesamiento y comunicaciones en aplicaciones distribuidas. Dichas propuestas requieren mediciones de tiempo que los sistemas actuales no pueden resolver de manera satisfactoria en los requerimientos de resolución y exactitud. Se debe tener en cuenta que en un sistema distribuido la exactitud es un atributo que dependerá de cuán sincronizados se encuentren los relojes del sistema, se trata por así decirlo, de un atributo del "reloj distribuido" del sistema.

Disponer en una computadora de una referencia de tiempo ajustada a una escala conocida (hora) ha sido, además de una necesidad, una realidad desde los primeros equipos. Casi todas ellas tienen implementado algún tipo de dispositivo para cumplir esta función (reloj). Con el advenimiento de las comunicaciones entre computadoras, aparecen los sistemas distribuidos. En estos sistemas cada computadora tiene su propio reloj, con lo que comienza el problema de mantener dichos relojes en hora (sincronizados), dentro de una banda de error conocida (offset, sesgo), en el caso de las computadoras que interactúan entre sí.

El presente trabajo intenta presentar las distintas facetas del problema de sincronizar los relojes en un sistema distribuido, como así también las diferentes soluciones propuestas hasta el momento. A partir de un análisis de dichas soluciones existentes, se propone una variante de alguna de ellas a fin de solucionar un aspecto no necesariamente explorado como es el de la sincronización, que tiene como objetivo un reloj del sistema para la estimación de rendimiento.

Referido al tema de sincronizar un sistema se parte de los siguientes puntos [51]:

- En un sistema centralizado el tiempo no es ambiguo.
- En un sistema distribuido no es trivial poner de acuerdo a todas las máquinas en una hora.
- Se requiere un acuerdo global en el tiempo, pues la falta de sincronización en los relojes puede ser drástica en procesos dependientes del tiempo.

Se trata de hallar una solución al problema sin utilizar hardware especializado, mas allá del ya existente en el sistema para su función principal, presuponiendo que:

- las máquinas integrantes del sistema tienen algún reloj de hardware
- se encuentran conectadas entre sí
- y su sistema operativo tiene implementados los protocolos TCP/IP

En los casos de haber herramientas de uso libre disponibles se utilizarán, evitando el desarrollo de software del que existan soluciones aptas para resolver el problema dentro de los requerimientos necesarios, detallados en los objetivos.

1.2 Requerimiento de la Biblioteca

Se desarrollaron una biblioteca para consulta y actualización de la hora, llamada *timings* que funciona en forma local en cada nodo, y otra de sincronización de la hora, *st*, que corre en forma distribuida.

Con respecto a *timings*, la consulta de la hora debe ser económica tanto en lo referente a tiempo como en ciclos de CPU. Para ello se evita el uso de llamadas al sistema, interrupciones u otros métodos que signifiquen cambios de contexto o demoras que puedan alterar el funcionamiento de la aplicación cuyo rendimiento se quiera estimar.

Con respecto a *st*, se utiliza para el necesario intercambio de los mensajes que llevan las referencias de tiempo para sincronizar los nodos, la red de comunicaciones preexistente entre computadoras, y las facilidades de comunicación proporcionadas por el protocolo TCP/IP. La sincronización de las máquinas se realizará fuera del tiempo de medición, por lo que no se hará tanto hincapié en la carga de comunicaciones como en alcanzar el objetivo de sincronización en forma y tiempo, con la correspondiente caracterización de error de sincronización.

1.3 Objetivos del Trabajo

El objetivo inicial es la sincronización de relojes en ambientes distribuidos para evaluar el rendimiento de aplicaciones distribuidas/paralelas. Dichas aplicaciones requieren para su funcionamiento intercambiar mensajes. Si no sincronizan el momento que una envía con el que la otra lo debe recibir, surgen demoras en la ejecución de dichos programas que deben ser solucionadas para su optimización [23]. Para medir dichas demoras, es imprescindible contar con relojes sincronizados al momento del envío en ambas máquinas, dentro de una banda de error compatible con la demora a medir.

La sincronización de relojes está principalmente dirigida a ambientes de cluster, con redes LAN, de uso exclusivo o controlado. Se analiza su extensión a redes de uso público tales como Internet. Dado que para sincronizar relojes es imprescindible intercambiar referencias a través de la red de comunicaciones existente en el sistema distribuido, uno de los problemas que se presenta es la variación de tiempos de comunicación de mensajes.

Este problema está más acotado en redes LAN. En Internet se agrava por el hecho de que el camino que siguen los mensajes entre dos máquinas no es fijo a causa de las condiciones cambiantes de ruteo.

Debido a que se usará en evaluación de rendimiento, es necesario minimizar la intrusión de las mediciones respecto del funcionamiento en tiempo y forma de las aplicaciones a medir. Esto conlleva dos requerimientos:

- **Desacople** entre sincronización y uso de la aplicación misma
- Minimizar y evaluar la **sobrecarga** que introduce en **procesamiento** y **comunicaciones** la medición de tiempos cuando esté en funcionamiento la aplicación bajo evaluación.

El **desacople** se refiere a la concentración fuera del tiempo de prueba de las siguientes tareas:

a) tareas de inicialización de comunicaciones,
b) cómputos de tiempos para calibrar el sistema
c) demás tareas que sean previas a la medición misma.

El desacople tiene por fin alterar lo menos posible el funcionamiento de la aplicación a medir por intrusión. Dicha intrusión, obviamente existirá por la necesidad de instrumentar el código pero será reducida a su mínima expresión, dejando, en lo posible, la mayor cantidad de tareas, para ser ejecutadas en tiempo previo o posterior a la prueba.

En el aspecto de **sobrecarga** en **procesamiento** cabe acotar que las mediciones de tiempo utilizadas en la actualidad comúnmente implican llamadas al sistema operativo, con los consiguientes cambios de contexto, desalojos de memoria, uso de memoria virtual, etc. Debido a ello, cualquier medición de tiempo implicará una **sobrecarga sobre el procesamiento** de la aplicación bajo prueba que se debe disminuir lo más posible. Evidentemente, se deben evitar las llamadas al sistema, que en general es lo que utilizan los lenguajes de programación.

En referencia a la **sobrecarga en comunicaciones**, la comunicación de las referencias imprescindibles para la sincronización debiera ser mínima al menos durante el tiempo de funcionamiento de la aplicación bajo prueba. Toda comunicación implica operaciones de entrada y salida, con lo que además de congestionar la red de comunicaciones, altera el tiempo de ejecución de las aplicaciones distribuidas a medir.

En resumen, para minimizar la intrusion, se deberá desacoplar la comunicación de las referencias realizándolas fuera del tiempo de ejecución de las antedichas aplicaciones, como asimismo se deberá ampliar el periodo en el cual se deba volver a sincronizar los relojes.

Con respecto a la **resolución** de los tiempos a medir, deberá ser al menos del orden de los microsegundos, ya que el avance en hardware de comunicaciones ha posibilitado tiempos de ese orden para el paso de mensajes entre máquinas (60 microsegundos para una LAN), tiempos que son los que se desea medir. Se debe tener en cuenta que no es trivial lograr resoluciones de microsegundo en un reloj distribuido, si bien actualmente es relativamente sencillo alcanzarlas a nivel local.

Se llevará a cabo una evaluación de las herramientas disponibles, en función de los requerimientos dados, limitando el desarrollo de nuevo software para aquellos aspectos que no puedan ser cubiertos por las existentes. Muchas de estas herramientas fueron desarrolladas con otros objetivos en mente, diferentes a los de este trabajo, con lo cual, en algunos casos, puede haber problemas en su uso por no haberse contemplado en su desarrollo objetivos tales como minimizar la intrusión o alcanzar una determinada resolución.

1.4 Trabajos Relacionados

Se han realizado investigaciones de las que han surgido diferentes algoritmos e implementaciones [10][14]. Una de las primeras fue la relacionada con el algoritmo de sincronización de computadoras de Lamport.

El algoritmo de Lamport [28] tiene como objetivo alcanzar un ordenamiento de los sucesos en un sistema distribuido. Cuestiones tales como hora real y resolución, son dejadas de lado. En aplicaciones distribuidas en las cuales el orden de eventos puede llevar a diferentes estados finales, este algoritmo representa una posible solución; como por ejemplo en el caso de las transacciones bancarias, en las cuales un depósito y una extracción en una cuenta en diferente orden generan o un crédito o un débito de intereses.

La cuestión de la hora real en computadoras es un tema heredado de los problemas con respecto a la posición de navegación marítima y más tarde, aérea. Para determinar la latitud no hay implicaciones de hora pero respecto de la longitud ecuatoriana surge la necesidad de saber la hora real para determinar la posición. Este problema encuentra solución con la aparición de las transmisiones de radio y la consecuente sincronización, que se consigue enviando, por este medio, referencias con destino a los relojes a sincronizar en las naves.

En las transmisiones de referencias de hora a través de radio, dado que las transmisiones requerían de un cierto tiempo en arribar a destino, se recurría a medir el tiempo de ida y vuelta mediante un mensaje de respuesta, enviado desde el destino en forma inmediata después del mensaje recibido. Una vez recibida esta respuesta, el emisor primario puede calcular el tiempo de ida como la mitad del tiempo de ida y vuelta. Gracias a este dato se puede determinar la verdadera hora en base a dicha referencia de hora de radio. Esta referencia de hora así transmitida puede ser capturada por una computadora conectada a un receptor de radio. Este tipo de hardware presenta el inconveniente de su precio, además de requerir que la computadora ocupe gran parte del tiempo en manejar la comunicación con la radio.

Gracias a la disponibilidad de este tipo de dispositivos y a la aparición de redes de computadora, se analiza la posibilidad de distribuir la referencia de hora radial o de cualquier tipo a través de una red, en un modelo cliente-servidor [12]. Se hace hincapié en que el servidor proporcione a cada cliente una referencia con el menor error y, en lo posible, caracterizado. Como toda arquitectura cliente-servidor, tiene un cuello de botella en el servidor, lo cual complica escalar el algoritmo. Además, ante una falla del servidor, deja al sistema sin referencia de hora.

En función de la robustez del sistema, el Algoritmo de Berkeley [24] proporciona una manera de obtener, sin una referencia externa, una hora similar entre las máquinas de una red local. Se transmiten diferencias de horas desde el servidor a los clientes para dar la referencia de corrección, con lo que no influyen los tiempos de transmisión de los paquetes. El algoritmo se ve enriquecido en cuanto a robustez por el agregado de algoritmos de elección de un nuevo servidor en caso de caída o fallo del mismo.

En las últimas décadas se ha trabajado en la elaboración de diferentes versiones del protocolo NTP (Network Time Protocol) [36][37][38], en el cual se encuentran

implementadas también cuestiones como la de obtener una hora mas exacta entre un arreglo de relojes computadoras del que tendrían cada una por separado [1] (varianza de Allan). En el protocolo NTP se evita el cuello de botella de los sistemas cliente-servidor a través de un sistema jerárquico de estratos. En cuanto a la robustez se utilizan algoritmos de elección, permitiéndose en la configuración especificar varios servidores.

El problema de NTP es que al cumplir con un conjunto de requerimientos muy amplio, no está optimizado en ningún aspecto en particular:

- Para operaciones sobre la hora local, utiliza llamadas al sistema operativo, con la consiguiente sobrecarga de procesamiento.
- El servidor puede sufrir saturación de clientes, con lo que se degrada su nivel de servicio de referencias, de manera tal que un servidor de un estrato inferior, pero menos saturado de requerimientos de clientes, puede suministrar una mejor referencia.

En cuanto al uso de relojes distribuidos en medidas de rendimiento, los sistemas de procesamiento paralelo como MPI y PVM utilizan el reloj local de cada máquina, desentendiéndose de la sincronización. Con respecto a las aplicaciones de instrumentación existentes, en general realizan mediciones de tiempo a nivel local.
En la actualidad, la mayoría de las aplicaciones que necesitan hora sincronizada utilizan NTP para ello [18], aún con las limitaciones que tiene, sobre todo en lo referente a estimaciones de rendimiento. Dichas limitaciones de NTP serán analizadas en mayor profundidad en otros capítulos.

1.5 Organización de los Capítulos

En el capítulo 2 se presenta una breve reseña histórica sobre el problema de la medición del paso del tiempo, junto con una descripción de los tipos de relojes de mayor precisión actuales, los de cuarzo y los atómicos. También se presenta el problema de posición en longitud ecuatoriana y la hora, que culmina con la creación del sistema GPS, el cual también se describe.

En el capítulo 3 se describen los dispositivos asociados a medición y registro del tiempo presentes en las computadoras actuales. Además se detallan los relojes de software y se muestran resultados de experimentos sobre los diferentes dispositivos. También se detalla la biblioteca *timings*, uno de los productos desarrollados en este trabajo de tesis.

En el capítulo 4 se presentan los algoritmos básicos de sincronización de relojes de computadoras, analizando sus ventajas e inconvenientes.

En el capítulo 5 se analizan las redes de comunicaciones en uso en los sistemas de cómputo, ya que será a través de estas redes que se enviarán los mensajes con las referencias de tiempo a fin de lograr sincronizar los relojes. Se analizan los distintos modos de enviar mensajes en este tipo de redes, y las pruebas realizadas en ellas para determinar qué modo conviene para cada necesidad del presente trabajo. También se analizan las comunicaciones con dispositivos externos a la red para adquisición de referencias de tiempo a partir de receptores de radio o relojes atómicos.

En el capítulo 6 se dan detalles de las implementaciones existentes, que emplean uno o una mezcla de los algoritmos existentes. También se muestran resultados de las pruebas llevadas a cabo sobre las implementaciones para caracterizarlas en base a los requerimientos del problema de mediciones de rendimiento en ambientes distribuidos, ya que fueron diseñadas con diferentes requerimientos, en general más amplios y menos exigentes en algunos aspectos.

En el capítulo 7 se exponen lo detalles de la herramienta de sincronización implementada.

En el capítulo 8 se exponen los experimentos llevados a cabo con dicha herramienta de sincronización.

En el capítulo 9 hay un caso de uso de la herramienta sobre un programa de multiplicación de matrices

En el capítulo 10 se exponen las conclusiones y posibles líneas de trabajo futuras para continuar profundizando en el tema.

Por último figuran las referencias bibliográficas.

Capítulo 2: Medición del Tiempo: del Sol al GPS.

Resumen

En este capítulo se presenta desde un enfoque histórico el problema de la medición del tiempo para arribar a una descripción de los últimos adelantos en la materia como lo son los relojes de cuarzo, atómicos y el sistema GPS.

2.1 Introducción

La medición del tiempo ha sido desde siempre un desafío. El significado de "el paso del tiempo" ya de por sí relaciona al tiempo con un suceso físico como sería el caminar de una persona. A partir de esto, pareciera que solo se puede hablar de tiempo cuando hay fenómenos físicos cambiantes, observables y en particular si el cambio es periódico, o sea que se repite, se pueden asociar unidades de tiempo al cambio. De la medición de frecuencia de repetición viene la noción de tiempo. En función de estos cambios periódicos se puede cuantificar un tiempo entre otros dos sucesos. También solo por comparación con otros fenómenos periódicos podemos decir qué tan estable es una frecuencia. De hecho, la percepción mental del paso del tiempo sin recurrir a la observación de un fenómeno físico es completamente subjetiva. Se puede estimar con bastante exactitud espacio, colores, pero lapsos de tiempo, no. Por ello, la historia de la medición del tiempo es la búsqueda de patrones de frecuencia estable en fenómenos físicos. A partir de que se deben percibir dichos fenómenos, lo que altere dicha percepción es también parte del proceso. Se llamará sincronización al proceso de ponerse de acuerdo en dichos patrones de frecuencia (sincronización de frecuencia) y a tomar alguna marca como valor inicial para iniciar el conteo de los ciclos de repetición. Dicha sincronización implica la cuestión de comunicar referencias. Comunicaciones, frecuencia y tiempo, son tópicos de este tema; estabilidad y error, son atributos de los tópicos que se estudian. En este capítulo se muestran los diferentes enfoques de los elementos antes mencionados en forma evolutiva hasta alcanzar los métodos actuales.

2.2 Reseña Histórica

La medición del paso del tiempo, en una cierta escala consensuada internacionalmente, tal como está disponible en la actualidad, ha llevado un largo desarrollo. Dicho desarrollo [1][51] pasa por los crecientes requerimientos de exactitud del conocimiento del transcurso del tiempo de la humanidad.

En los casos en que solo se requeria saber la disponibilidad de luz solar, la posición del sol en el cielo era suficiente. Este "reloj" primario cubría las necesidades para las tareas relacionadas con la agricultura y los viajes terrestres.

Con los primeros avances de la astronomía se logra ajustar el comienzo y el fin de las estaciones. En diversas civilizaciones, en algunas más temprano que otras, combinaciones de mediciones astronómicas bastaron durante siglos para organizar sus actividades diarias.

Éstas constaban de la aparente posición del sol en lo más alto del cielo, con lo que al repetirse al día siguiente, se tomaba la duración de un día. La medición del tiempo por mediciones astronómicas sobre la posición de las estrellas, fue una corrección sobre el tiempo solar, debido a que las estrellas, al estar más lejos que el sol, introducen menos error (el llamado error de paralaje) que la rotación de la tierra sobre su eje y la posición del sol. Este error de paralaje es debido a que la tierra tiene un movimiento de traslación alrededor del sol, que se superpone al de rotación.

Una primera necesidad de mejorar en los requerimientos de exactitud en el

conocimiento de la hora nace al comenzar la navegación fuera de la visibilidad de las costas, o sea, sin referencias terrestres. Con respecto a la latitud, las determinaciones astronómicas no presentaban problemas. Pero con respecto a la longitud, debido al giro de la tierra para determinarla, uno de los métodos requería además de un buen sextante, saber qué hora era en el momento de la medición, o lo que era equivalente, en qué posición de giro respecto de su eje se encontraba.

Por lo tanto se requería contar con un reloj con una hora suficientemente precisa. Dicha precisión en la hora no la tenían los relojes usados en los años del 1700 [42].

Por un error en el cálculo de la longitud, en el año 1707 se registró un naufragio en el que hubo cerca de 2000 víctimas, lo que llevo en el año 1714 a que el parlamento de Inglaterra ofreciera 10.000 libras para quien determinara la longitud con un grado de error, 15.000 libras en 40 minutos de grado y 20.000 libras (alrededor de 2.000.000 de dólares de hoy) en 30 minutos.

Para solucionar este problema, se diseñaron relojes más confiables, tanto en tierra, como en las naves, las cuales una vez tomada la hora de tierra, debían mantenerla sincronizada en un rango de entre 8 y 3 segundos por día.

Es de destacar que para todos los usos del reloj, el error es un componente. Si bien se va acotando con el avance tecnológico, siempre habrá un error en la medición de la hora.

Con una desviación de 3 a 8 segundos por día en la hora con la cual se mide la longitud, en un viaje de América a Europa, se garantizaba un máximo de error en la estimación de la longitud de treinta millas náuticas.

A partir del uso de este método nace una asociación entre posición y hora, que desemboca en la actualidad en el sistema GPS (posicionamiento satelital global), utilizado en la actualidad a fin de obtener la posición de cualquier móvil u objeto estacionario sobre la superficie terrestre con un mínimo error. En este sistema la posición de los astros se reemplaza con satélites en órbita geoestacionaria y los relojes utilizados son de tipo atómico.

En cuanto al concepto de segundo como unidad de tiempo, su definición ha ido cambiando a través del tiempo.

En un principio se consideró suficiente exactitud la proporcionada por el suceso físico del giro de la tierra sobre su eje, apreciada por la posición en el cielo del sol y las estrellas. A través de observaciones astronómicas se determinaba la duración del día, a partir de ahí se definía el segundo como 1/86400 parte del día.

El comienzo a partir del cual se consideraba el tiempo cero, o sea el comienzo del primer día del primer año (calendario), fue motivo de diversos acuerdos, hasta llegar a ser más o menos internacional.

A partir de la aparición de relojes mecánicos de precisión se observó que el período de una rotación era variable a lo largo del tiempo, con lo que se empezó a tomar como referencia el tiempo solar medio.

En 1884, en la Conferencia Internacional del Meridiano, el observatorio de Greenwich fue aceptado como fuente de la hora estándar. GMT (Greenwich Meridian Time) fue el estándar por casi un siglo. A partir de la hora cero definida en este meridiano, se computaban cada 15º un aumento de una hora. Cada franja así definida daba una zona horaria. A partir del año 1926 la hora según GMT pasó a llamarse Tiempo Universal (Universal Time, UT) y contaba con diversas versiones:

- UT0: Hora obtenida por observaciones astronómicas de la posición de las estrellas.
- UT1: La hora UT0 introduciéndole una corrección por el movimiento de los polos terrestres.
- UT2: UT1 corregido por las variaciones en el tiempo de rotación de la tierra. Dicho tiempo va aumentando con el paso de los años. La tierra rota cada vez mas lento, con lo que el año tenía "más días" en épocas remotas.

Con la aparición de los relojes atómicos, surgió el TAI (Internacional Atomic Time, aunque la sigla sugiere el orden francés de Temps Atomique International).

Debido a que el tiempo de rotación de la tierra aumenta a través de los años y en consecuencia el día solar medio con que se calcula UT2, surgieron desviaciones con la hora TAI, con lo que cada vez que esta diferencia supera 0,9 segundos se le agrega un segundo a TAI. Estos segundos son denominados "leap seconds".

La hora así obtenida se la denomina Tiempo Coordinado Universal. Su sigla es UTC, aunque en inglés debería ser CUT (Coordinated Universal Time) y en francés, TUC (Temps Universel Coordonné). Es decir, que la sigla que sirve de hora al sistema GPS no es ni del inglés ni del francés ni del español.

Actualmente, la medida más exacta conocida por la humanidad es la duración de un segundo.

La actual referencia máxima de tiempo y frecuencia exactos es UTC.

La exactitud en la hora hasta el momento contiene un error en un segundo de ± 0,3 nanosegundos (10exp-9) por día, lo que significa ±1 segundo en 10 millones de años.

A continuación se exponen los relojes de mayor precisión usados en la actualidad y en el capítulo 3 se realiza una descripción detallada de los dispositivos utilizados como relojes en las computadoras.

2.3 Reloj de Cuarzo

En los sistemas de cómputo se encuentran uno o varios dispositivos de tipo reloj. Son construidos en base a osciladores electrónicos, por lo general estabilizados en frecuencia por un cristal de cuarzo, con un acumulador que va contando las oscilaciones. Son relojes de cuarzo [20] [48] adaptados a la computadora.

Por lo general, los caracteriza la alta precisión y bajo costo de fabricación. Vulgarmente conocidos como electrónicos, se caracterizan por poseer una pieza de cuarzo que sirve

para generar, en resonancia, los impulsos necesarios a intervalos regulares que permitirán la medición del tiempo y cumplen una función similar a la del péndulo o el balancín en un reloj mecánico.

El cuarzo se talla en forma de lámina y se introduce en un contenedor metálico para su protección. El cristal de cuarzo para vibrar debe ser alimentado por un campo eléctrico oscilante generado por un circuito electrónico (oscilador). El cristal sometido a un estímulo eléctrico puede continuar vibrando a una cierta frecuencia (dependiente de la propia naturaleza del cristal) hasta perder ese impulso inicial. Si se mantiene el estímulo de manera periódica y sincronizada, tendremos una señal a una frecuencia extraordinariamente precisa.

Este tipo de relojes suelen utilizarse en sistemas electrónicos como relojes, computadoras, etc.

La frecuencia de oscilación dependerá de tres factores:

1) La forma en que se corte el cristal.
2) El tipo de cristal.
3) La magnitud de la tensión.

Asociado al cristal se encuentra un registro contador. Este registro incrementa su valor en uno por cada ciclo del oscilador.

El cuarzo hace el papel de regulador y estabilizador de la frecuencia del oscilador electrónico, que debe generar una señal eléctrica de la misma frecuencia. La onda eléctrica generada a su vez por el cristal excitado por el oscilador, se realimenta en forma positiva al mismo y se corrigen las desviaciones de frecuencia que pudieran producirse respecto a su valor nominal.

Este tipo de relojes puede sufrir desviaciones en su frecuencia de oscilación debido a cambios en la temperatura y presión ambiente y la presencia de campos magnéticos.

2.3.1 Experimentos

Se realizaron pruebas tendientes a verificar la estabilidad de frecuencia de los relojes de cuarzo en el tiempo similares a las reportadas en [14]. El experimento requeriría de una referencia de tiempo perfecta o al menos con la menor deriva de frecuencia posible, tal como un reloj atómico o la hora UTC. Debido a que no se cuenta con estos elementos, se toma como referencia el reloj de cuarzo del *timer* de una PC, y con el se miden las variaciones del reloj TSC (*Time Stamp Counter*). Si bien no se puede medir el error en términos absolutos, se puede determinar cuánto varía la diferencia de deriva.

Dado que en la línea de investigación que se sigue no se requiere de referencias externas, sino solo que las referencias estén ajustadas a la escala de tiempos que determine el servidor, es válido medir esta variación de frecuencia y pensar que el error será proporcional a dicha diferencia. Los resultados obtenidos son similares a los de [8], de menos de 10 ppm (partes por millón) en un período de varias horas.

2.4 Reloj Atómico

Son los utilizados para obtener el Tiempo Atómico Internacional (TAI) y el Tiempo Universal Coordinado (UTC)[38] [29]. Se denomina reloj atómico a un reloj cuyo funcionamiento se basa en la frecuencia de una vibración atómica para alimentar su contador. Se basan en un fenómeno extremadamente regular, la resonancia magnética molecular y atómica. Los primeros prototipos utilizaban como molécula de resonancia el amoníaco pero su precisión no era muy superior a los estándares de la época basados en osciladores de cuarzo. Los primeros utilizados eran masers con equipamiento incorporado y alcanzaban una exactitud de 10^{-9} segundos por día, con una precisión igual a la frecuencia del transmisor de radio que bombea el maser.

El maser, acrónimo de Microwave Amplification by Stimulated Emission of Radiation (amplificador de microondas por la emisión estimulada de radiación), es un amplificador similar al láser, pero opera en la región de microondas del espectro electromagnético y sirve para recibir señales muy débiles.

Actualmente se basan en las propiedades físicas que tienen las fuentes de emisión de cesio.

Para realizar la medición a través de estas partículas es necesario crear un campo electromagnético que no existe de forma natural en el Universo. El proceso se realiza dentro de una "trampa magneto-óptica", una esfera del tamaño de un melón en la cual se inyectan átomos de cesio y se propagan, encerrados en un campo magnético, seis rayos de luz láser.

De la misma forma que una persona disminuye su paso ante una ráfaga de viento, los átomos reducen su velocidad al ser bombardeados por los rayos láser emitidos en todas direcciones. Con este método, los átomos pueden reducir su velocidad hasta hacerla 10 mil veces más lenta de lo normal.

Cuando los átomos y los rayos láser chocan, se forma una nube de átomos muy lentos o ultra fríos.

En este tipo de reloj, los átomos de cesio emiten fotones, parecidos a una onda, que oscilan como el péndulo de un reloj antiguo.

En el año 1967 la Oficina Internacional de Pesas y Medidas eligió la frecuencia de vibración atómica como nuevo patrón base para la definición de la unidad de tiempo físico. Según este patrón, un segundo se corresponde con 9.192.631.770 ciclos de la radiación asociada a la transición hiperfina [29] desde el estado de reposo del isótopo de cesio-133. El sistema electrónico del reloj marca un segundo cuando ha ocurrido esta cantidad de ciclos de oscilación del campo eléctrico. De contar ese número de oscilaciones viene la exactitud del reloj atómico. La precisión alcanzada con este tipo de reloj atómico es tan elevada que admite únicamente un error de un segundo en 30.000 años. El reloj más preciso del mundo se diseña en el Observatorio de París, donde los actuales relojes atómicos tardan 52 millones de años para desfasarse un segundo. El nuevo objetivo de la investigación francesa es aumentar ese plazo a 32 mil millones de años.

El estándar actual de los relojes atómicos en activo permite el atraso de un segundo cada 300 mil años.

En agosto de 2004 el NIST presentó un reloj atómico del tamaño de un circuito integrado, cien veces más pequeño que cualquier otro construido hasta esa fecha, con un consumo de 0,075 watts.

2.5 GPS y UTC

GPS ha pasado a ser el mejor proveedor de hora exacta. Este sistema es usado como fuente de hora y para comunicar este dato a diferentes locaciones a través del planeta. GPS suministra tres tipos de hora:

- Hora GPS
- UTC tal como la estima y produce el Observatorio Naval de Estados Unidos
- La hora de cada reloj atómico que funciona libremente en cada satélite de GPS

La estación maestra de control (The Master Control Station (MCS)) en la base de la Fuerza Aérea de Falcon cerca de Colorado Springs, estado de Colorado, reúne los datos de los satélites de GPS en 5 estaciones de monitoreo alrededor del planeta.

Un programa de software implementa el filtro de Kalman, para estimar el error en la hora, error en la frecuencia, deriva de la frecuencia, y los parámetros de la órbita Kepleriana para cada uno de los satélites y el funcionamiento de sus relojes.

Dicha información es enviada a cada satélite para que la difundan en tiempo real. Este proceso provee hora GPS a través de la constelación de satélites con un error de nanosegundos y un error en la posición de los satélites mismos de pocos metros.

Dado este proceso, GPS no tolera la introducción de “leap seconds”. Cuando se inicia en 1980, el sistema GPS toma la hora de UTC (USNO MC). Por ese entonces TAI adelantaba 19 segundos a UTC. Desde entonces, UTC fue atrasado algunos segundos mientras que GPS no.

Por lo tanto, el tiempo del GPS todavía está muy cerca de TAI menos 19 segundos. La especificación de la hora en GPS establece que debe ser mantenida dentro de una diferencia de un microsegundo de la hora UTC (USNO MC) módulo un segundo.

2.6 Términos y Definiciones

Con respecto a los términos que tienen usos ambiguos se aclara con qué significado se usan en este trabajo[7][33][38]. Se considerará que un reloj es estable si puede mantener constante su frecuencia y que es exacto si está bien comparado con alguna hora estándar establecida con una cierta autoridad. O sea que la exactitud es la proximidad entre el resultado de una medida y el valor verdadero.

El término precisión se refiere a qué unidad de tiempo puede ser resuelta en un sistema de tiempo.

La confiabilidad (reliability) está referida al tiempo en que un reloj está disponible, operando correctamente y con estabilidad, o sea, puede suministrar la hora con una exactitud especificada.

Aunque el error sistemático de un reloj sea conocido, el reloj no será perfecto, dado que su frecuencia varía en el tiempo influenciado por factores ambientales.

Observaciones de largo plazo pueden mostrar variaciones en la frecuencia de un reloj (con su frecuencia inicial, no solo la de otro reloj). Dicha diferencia es denominada deriva (drift, aunque también se usa wander).

Puede haber relojes con estabilidad a corto plazo pobre, pero con buena estabilidad a largo plazo, y viceversa.

Lecturas repetidas del tiempo dan diferencias aleatorias. La diferencia de estas diferencias se denomina jitter.

Es pertinente aclarar que la palabra inglesa *time* tiene el significado de tiempo y también el de hora. Es así que se pregunta *¿qué hora es?* como *what time is it?*

Capítulo 3: Formas de Registro de Tiempo en Computadoras

Resumen

En este capítulo se analizan los dispositivos capaces de proveer una referencia o marca de tiempo en las computadoras actuales.

3.1 Introducción

Se puede hacer una distinción entre relojes de hardware y de software. En general, los relojes de hardware constan de un oscilador electrónico, por lo general estabilizado con un cristal de cuarzo, y un contador de ciclos. El fenómeno físico cuya frecuencia se observa sería la frecuencia natural de resonancia del cristal; el medio de observación sería la electrónica asociada. Los relojes de software son derivados a partir de los de hardware, a través de operaciones matemáticas sobre los datos suministrados por los mismos. Son dispositivos de software que tomando como referencia uno o más señales provistas por los relojes de hardware, elaboran una hora con diferentes características de las que usan de referencia. Debido a que es más fácil modificar parámetros de software que de hardware, los relojes de software son los que se sincronizan.

En una computadora suelen encontrarse tres diferentes tipos de dispositivos de hardware destinados a la medición de tiempos [25][5][7]:

- Reloj de tiempo real
- Timer
- TSC

3.2 Reloj de Tiempo Real

Por lo general está alimentado a batería, de tal manera de mantenerlo en funcionamiento cuando la máquina está apagada. Es lo más similar a un reloj de uso común: cuenta en horas, minutos y segundos, puede ser puesto en hora y consultado a través de puertos de comunicación, utilizando llamadas a funciones del BIOS (Basic Input Output System, software embebido en las computadoras tipo pc (personal computer)).

3.3 Timer

Consta de los siguientes elementos:

- Oscilador de frecuencia estabilizada a cristal de cuarzo, con una frecuencia, en el caso de la arquitectura Intel ia32, de 1.193.000 hz (ciclos por segundo). Este valor de frecuencia es independiente de la velocidad de la CPU.
- Registro acumulador que lleva cuenta del número de oscilaciones desde 0, si lo hace en forma ascendente o desde un valor prefijado hasta cero, llamado counter register.
- El registro mantenedor o latch register: mantiene el valor prefijado desde el cual inicia su cuenta el registro acumulador. El usuario tiene acceso a este registro para modificarlo.
- Registro de configuración, también accesible por el usuario para modificarlo, que permite diferentes modos del timer:
    a. un solo recorrido de cuenta
    b. realizar el conteo en forma ascendente o descendente
    c. al terminar un conteo vuelve a empezar.

Por lo general, al concluir una cuenta se emite una señal por una salida que puede ser conectada a dispositivos de interrupción.

En el caso de las arquitecturas x86, este dispositivo es el PIC (controlador de interrupciones programable), y activa la interrupción número 8. Esta activación es la conocida como pit En respuesta a esta interrupción, se actualiza el *time of day*, un registro de 4 bytes, en el que se almacena la cantidad de interrupciones producidas desde que se encendió la máquina.

El valor del *time of day* convenientemente formateado se adiciona al valor del reloj de tiempo real, convertido al formato adecuado, el cual se lee una única vez al encenderse la máquina. Al terminar la actualización de *time of day* se llama a la interrupción de software 1C, que normalmente apunta a un manejador que ejecuta solo un IRET. Este manejador se cambia por el programado por el usuario, y así utilizamos la interrupción por tiempo. Se debe cuidar de reestablecer el manejador original al no utilizar más el del usuario.

Se debe tener en cuenta que:

- La cantidad de veces por segundo que se activa la interrupción depende del valor almacenado en el registro mantenedor, y depende de los SO.
- En DOS y Windows se almacena el mayor valor que soporta el registro mantenedor, 65535, provocando 18,2 interrupciones por segundo (1193000/65535).
- En Linux, este valor se puede cambiar al compilar el kernel y es por defecto 100 veces por segundo.
- En UNIX es de 1000 veces por segundo, pero recompilando el kernel con la opción MICRO_TIME se obtiene resolución de microsegundo[56].
- Alterar estos valores cambiará el funcionamiento de los dispositivos que utilizan el *time of day* para su funcionamiento (Ej.: el time-stamp de los archivos).

Es común en aplicaciones de tiempo real utilizar la metodología de alterar los valores de los registros del timer. Si uno deseara aumentar la resolución, podría disminuir el valor del registro mantenedor, con la consecuente alteración de los mecanismos dependientes del *time of day*, pero el principal problema es que al generarse más cantidad de interrupciones por segundo, al llegar a cierto valor prácticamente todo el tiempo de la CPU se gastaría realizando cambios de contexto. Éste reloj es el que normalmente utilizan los Sistema Operativos en operaciones relacionadas con tiempo, como delay.

3.4 TSC

Los procesadores modernos tienen un registro en el que llevan cuenta de los ciclos del oscilador de la CPU desde su arranque. Este dispositivo es conocido como TSC (Time Stamp Counter) [40] [47] [59]. Se cuenta con instrucciones de lenguaje ensamblador que permiten saber su estado, RDTSC en el caso de la familia Intel (ReaD TSC). Este dispositivo no tiene posibilidad de generar interrupciones, por lo que para utilizarlo se lo debe consultar mediante encuesta (polling). El tamaño del registro acumulador es crucial, en combinación con la frecuencia del oscilador, para saber cuánto es el tiempo máximo a partir del cual rebasará el registro y comenzará la cuenta de 0. En algunos procesadores es de 32 bits, en Intel es de 64 bits, lo que le permite a un procesador con

frecuencia de oscilador de CPU del orden de Ghz contabilizar tiempos del orden de años sin rebasar. Se puede ver un ejemplo de consulta de tiempo mediante RDTSC en una máquina con procesador Intel en [26].

El uso de TSC como fuente de referencia de tiempo tiene varias ventajas:

- No se hace uso de la entrada salida, con la sobrecarga de tiempo que implica una operación de este tipo sobre cada lectura de tiempo.
- La operación de lectura se lleva a cabo mediante una instrucción de código de máquina, con lo que el procesamiento requerido ante cada lectura es mínimo.
- Tiene la resolución de la frecuencia de reloj del procesador, con lo que en la actualidad, con procesadores con relojes del orden del Gigahertz se obtienen resoluciones del orden del nanosegundo (10 e -9 seg).

Es imposible disponer en una computadora de un reloj de mayor resolución cuya lectura se realice en una sola instrucción de código de máquina.

3.5 Relojes de Software

Los relojes de software son simplemente dispositivos de software que, tomando señales de los diferentes dispositivos de hardware, elaboran algún tipo de contabilidad de dichas señales. Por ejemplo, la llamada a gettimeofday() de C da como resolución 1 microsegundo, haciendo una interpolación entre la consulta *a time of day* y TSC [8] [9]. Esta resolución puede comprobarse ejecutando el siguiente código (seudocódigo) de la figura 3.1 [55]:

```
gettimeofday(t1);
Repetir
        gettimeofday(t2);
Mientras(t1 = t2)
Imprimir (t2-t1)
```

Figura 3.1: Seudocódigo de la medida de resolución de gettimeofday()

Estos relojes, al poder cambiar su frecuencia de actualización y diferencia horaria inicial mediante variables de software, son ideales para ser sincronizados en ambos aspectos, ya que no requieren operaciones de entrada y salida, sino solo operaciones en memoria. En otro capítulo se detallan los métodos de sincronización. Su limitación radica en que nunca se tendrá mejor resolución que la del reloj de hardware del que derivan.

3.6 Análisis de Sobrecarga de Lecturas de Tiempo

Se ha visto que los métodos provistos por el sistema operativo no son apropiados [47][8] en función de su resolución en algunos casos y debido a la sobrecarga que implican las llamadas al sistema operativo durante el procesamiento. Por otro lado, los métodos y/o herramientas provistas por los lenguajes también dependen del sistema operativo y, por lo tanto, resultan inadecuados.

Se han realizado experimentos respecto de la sobrecarga en procesamiento al realizar una consulta al reloj local, utilizando gettimeofday() y consultas a TSC mediante la instrucción RDTSC a través de la rutina RDTSC.

La tabla 3.1 muestra los valores obtenidos en ciclos de CPU en las computadoras con las características que se dan en la tabla 3.2. También a partir de los datos de la tabla 3.1 (ciclos de CPU, específicamente) y los de la tabla 3.2 (MHz de reloj, específicamente) se comprueba que la precisión de la rutina RDTSC, del orden de los microsegundos, es acorde a los requerimientos del presente trabajo, especificados en el capítulo correspondiente. Esta precisión no tiene que ver con la resolución de TSC que en todos los casos es la inversa de la frecuencia del reloj de la CPU por segundo, sino con la velocidad que permite realizar una consulta tras otra.

PC	Gettimeofday()(ciclos)	Rut. RDTSC (ciclos)	Prec.Rut.RDTSC (μs)
P42400	7876	2828	≅1.18

Tabla 3.1: Ciclos de CPU de gettimeofday() y RDTSC.

PC	CPU	Frecuencia (MHz)	Sistema Operativo
P42400	Intel Pentium 4	2400	Linux 2.4.18-14

Tabla 3.2: Detalles de la PC utilizada en el experimento.

Se debe notar que la sobrecarga (tiempo extra de ejecución que no pertenece a la aplicación de usuario) de esta rutina de medición desarrollada es igual a la precisión de la tabla 3.1, dado que no hay cambios de contexto al incluir la biblioteca en el binario de la aplicación. En el experimento se provocó el desalojo de gettimeofday() y de la rutina RDTSC de memoria caché tanto de instrucciones como de datos, que es la condición real en la que serán usados en instrumentación. La rutina RDTS incluye además de la lectura del registro TSC las necesarias operaciones aritméticas para normalizar la unidad de tiempo. En realidad, de los 2828 ciclos de máquina sólo se consumen 25 para la ejecución de la macro que lee el registro TSC y lo devuelve en una variable.

A partir de estas pruebas de las diversas formas de adquirir en una máquina servicios de tiempo, tanto en lo referente a hora absoluta como a medición de intervalos y del análisis de factores como la sobrecarga y resolución, las desviaciones sistemáticas y su corrección, se obtuvo como producto la biblioteca *timings*, que se describe en la sección siguiente.

3.7 Biblioteca *timings*

Esta biblioteca utiliza como reloj de hardware el registro TSC. La consulta a dicho registro se realiza utilizando la instrucción de lenguaje ensamblador RDTSC. El lenguaje utilizado en el desarrollo de la biblioteca es C y se compiló utilizando GCC. Para integrar el lenguaje ensamblador, se utilizó una macro de *assembler inline* en vez de una función, para evitar una llamada a subrutina, minimizando de este modo la intrusión. TSC inicia su conteo a partir del valor cero al comenzar a funcionar la CPU, actualizando en uno su valor por cada ciclo del oscilador del reloj de la CPU.

Debido a que los osciladores presentan frecuencias diferentes para cada máquina, se hace necesario normalizar dichos valores mediante una constante a fin de tener una

unidad uniforme en todas las máquinas que intervengan en la posterior sincronización, por ejemplo microsegundos. Para ello se debe saber cuántos millones de ciclos ocurren por segundo (MHz, mega hertz). Los sistemas operativos, en su arranque realizan esta medición utilizando el timer. Los experimentos llevados a cabo indican que el valor calculado al inicio, en el arranque, varía una vez que el procesador entra en régimen estable de temperatura. Se han realizado intentos de mejorar este problema a través de recalcular el mismo [8].

La biblioteca *timings* hace su propia medición de los MHz del TSC, que no difiere mucho de la del sistema operativo, pero no nunca es idéntica, debido a que se realiza cuando el sistema está estable térmicamente, mientras que el sistema operativo realiza dicha medida en el momento del arranque. Esta parametrización del valor de MHz tiene otra utilidad. Se puede utilizar ecualización de las frecuencias de los relojes de las diferentes máquinas, utilizando solo el timer del servidor para proporcionar las referencias. Así se uniforma la unidad con que se actualizan los relojes de software que se sincronizan, a través del cálculo de los MHz en diferentes máquinas.

La biblioteca *timings*, en realidad tiene solamente dos funciones, que son sencillas pero muy necesarias para resolver el problema de evaluar una referencia de tiempo en función de MHz y hacer consultas rápidas (con intrusión mínima) del tiempo local. El archivo de definición de las funciones de la biblioteca, timings.h, que se muestra en la figura 3.2, refleja justamente esto.

```
                    /* timings.h */
                    /* --------- */

/* Function/s to profile code by time difference                    */
/* Basically oriented to have usec resolution with at least 1% error  */
/* It's expected to be used stand alone and from other lib/fctns      */
/* to have a synchronized ("uniform") time-reference in a cluster     */


 /*****************************************************************/
 /*                      Functions                                */
 /*****************************************************************/

 /* Initialize timing measurements with at least 1% error...        */
 /*     remMHz: just indicate if MHz is locally computed or assigned */
 /*             = 0.0 compute MHz locally (with local TSC)           */
 /*             != 0.0 just assign MHz                               */
 /*     sleepsecs: number of seconds to sleep for MHz computing...   */
 /*     Returns: MHz value                                           */
 double       timing_init(double remMHz, int sleepsecs);

 /* Return the local time since startup in us */
 long long int ulocal_time();
```

Figura 3.2: Archivo de definición de funciones de la biblioteca *timings*.

Como se puede ver en el código del *timings.h* (**figura 3.2**), la función init(…) puede recalcular la constante MHz o fijarla a partir del valor que se recibe como parámetro. Esta opción de asignación directa de MHz (sin calcularlo localmente) corresponde a lo que se explicó antes respecto de la ECUALIZACION de las unidades con que se actualizan los relojes, cuando se calcula el valor de MHz con una sola referencia de tiempo provista por el servidor en todas las máquinas a través de una función de la biblioteca st explicada más adelante.

Para eliminar los errores por la resolución del reloj de referencia, el timer se mide durante una cantidad mayor de segundos, por defecto, de 20 segundos. Para la lectura del timer se utiliza la función gettimeofday(), que ya se comprobó que tiene resolución de microsegundo. La biblioteca puede llevar a cabo este cálculo de los MHz tomando como referencia el gettimeofday() sobre el timer local (referencia local), o recibir dicha referencia a través de la red de una máquina única (referencia única) que la suministra a las demás en esta etapa de calibración de los MHz. Esto se realiza tomando en cuenta que los relojes varían de una computadora a otra, por lo que conviene calibrar todos en base a los valores del timer de una sola máquina.

Una vez calibrado el valor de los MHz, las mediciones de tiempo posteriores implican la ejecución de la macro para obtener el estado del registro TSC y una división para normalizarlo a microsegundos. Esto lo realiza la función ulocal_time(). Existe la posibilidad de solo ejecutar la macro durante la instrumentación y normalizar los valores medidos fuera del tiempo de ejecución de la aplicación a medir, guardando los valores y utilizando ciclos como unidad.

Capítulo 4: Sincronización de Relojes de Computadoras

Resumen

En este capítulo se presenta el problema de la sincronización de relojes y los diferentes algoritmos existentes. Ya que se trata de un problema no trivial, la mayoría de las soluciones están pensadas de tal manera que optimizan un aspecto de la solución en desmedro de otros. No existe al momento un algoritmo que solucione en forma integral el problema.

4.1 Introducción

La sincronización de relojes en ambientes distribuidos es un tema que presenta variadas soluciones en función de los requerimientos de los algoritmos distribuidos. Partimos de la caracterización de dichos algoritmos distribuidos [51][50]:

- La información relevante se distribuye entre varias máquinas.
- Los procesos toman las decisiones solo con base en la información disponible en forma local.
- Debe evitarse un único punto de fallo en el sistema.
- No existe un reloj común o alguna otra fuente precisa del tiempo global.

Esto indica que los algoritmos correrán en forma asincrónica, lo cual traerá aparejado demoras en la medida que estos procesos requieran interactuar, lo que lleva a la necesidad de obtener una referencia común al sistema distribuido.

La sincronización de relojes de computadoras puede entenderse desde dos puntos de vista:

- Sincronización interna: es el resultado de obtener en diferentes máquinas las mismas referencias de tiempo para un instante dado. De esta manera, al alcanzar la sincronización interna con un error acotado y conocido, se pueden medir eventos como el tiempo de transmisión de un mensaje, en donde las dos marcas de hora serán producto de dos relojes ubicados en máquinas distintas pero sincronizadas.
- Sincronización externa: si se desea saber en una máquina en particular a qué hora del día sucedió un evento, es necesario sincronizar la hora de esa máquina con algún reloj o fuente de hora autorizada. A esto se le llama sincronización externa.

En este aspecto se ve involucrado el concepto de época, o sea una referencia de tiempo que define el origen de una escala de tiempo [56]. En el caso de la sincronización interna, solo basta con definir en todos los nodos un tiempo cero cualquiera. En el caso de la externa, se debe definir en base a un acuerdo internacional, como la hora UTC. Lo mismo sucede con la unidad de tiempo con la que se actualizan los relojes. En el caso de sincronización interna, basta con que los nodos participantes tengan una unidad de medida común. En el caso de la externa, dicha medida surge de acuerdos internacionales, como la definición de segundo.

Con estas definiciones en mente, vemos que la sincronización será adecuada o no de acuerdo al uso que se les dé a las referencias de tiempo.

4.2 Errores Asociados a la Sincronización de un Reloj

Se puede definir un reloj como un dispositivo capaz de medir el tiempo. Está constituido por tres componentes:

1) Un sistema físico con oscilación periódico, tal como las oscilaciones de un péndulo o de un oscilador electrónico, con un

periodo T = unidad de tiempo/ frecuencia de oscilación (oscilaciones en esa unidad de tiempo)

2) Un acumulador que cuenta las oscilaciones, que se pueda consultar por algún método.
3) Un sistema de consulta de estado del acumulador para saber la hora.

La hora proporcionada por el reloj podría ser modelada como una función del tiempo. A partir de poner el reloj en hora (valor inicial del acumulador) a un valor inicial T0 va aumentando su valor con el paso del tiempo en una cierta cantidad *m* de unidades por unidad de tiempo, dada por la frecuencia del oscilador. La hora es una referencia de otro reloj (natural o artificial), ya que sólo podemos leer relojes y no percibir el paso del tiempo por nosotros mismos. La hora puede ser el suministrada por un sistema tal como GPS a través de un receptor, o cualquiera de los vistos en el capítulo correspondiente. A este tiempo se lo llama tiempo verdadero [3]: es el tiempo u hora definida precisamente por algún estándar. En otros casos puede ser el suministrado por otra computadora en base a su reloj local. Cada vez que en este capítulo se hace referencia a t, se lo hace con este significado.

Gráficamente y considerando un oscilador sería:

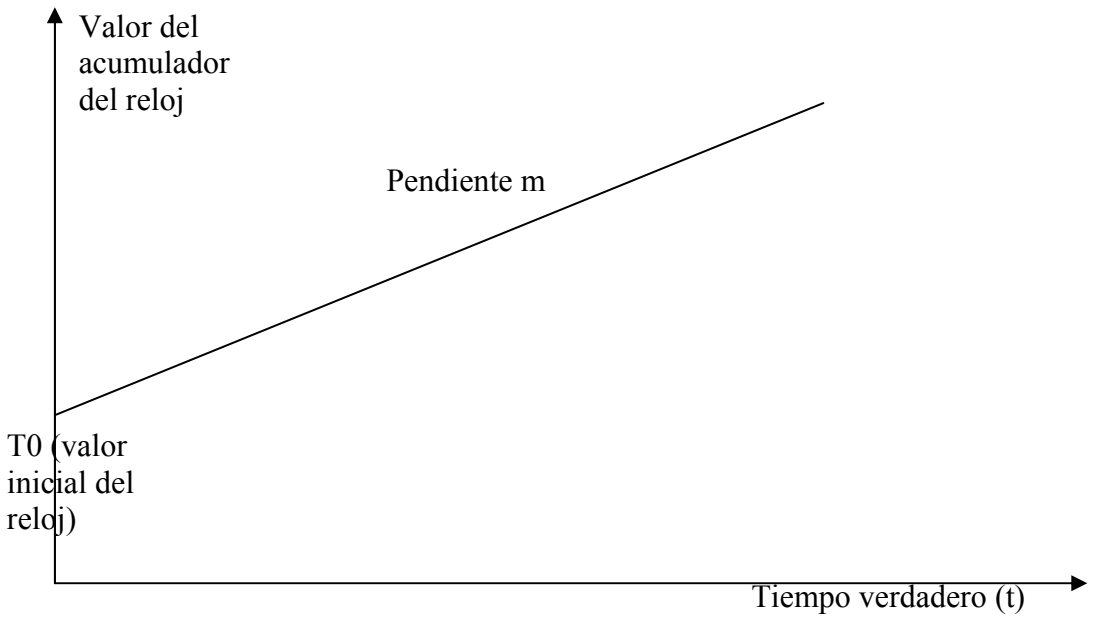


Figura 4.1: Representación gráfica del registro de tiempo en un reloj

El valor de la pendiente *m* sería 1 en caso de que la frecuencia sea de uno para cada unidad de tiempo (1 ciclo por segundo o un ciclo por microsegundo). Este no es el caso de la mayoría de los osciladores utilizados en los relojes de computadora, que actualmente tienen frecuencias del orden de los miles de millones de ciclos (GHz), con lo cual la normalización para obtener una marca de tiempo en una unidad de tiempo tal como segundo o microsegundo implica una operación de producto/cociente a partir de la consulta del acumulador. En este caso, para saber la hora teniendo en cuenta el estado

del acumulador, debiéramos realizar una normalización de dicha hora respecto de su valor inicial y frecuencia.

El tiempo medido por el reloj, *t (clk)*, sería

$$T(clk) = T0 + m * t$$

Con lo que observando T(clk) y conociendo T0 y m se puede saber la hora *t* de referencia, lo que sería hora verdadera si está sincronizado el reloj de referencia, a partir de:

$$t = (T(clk) - T0)/m$$

Por supuesto este nuevo reloj es un reloj de software. Su estado puede ser leído en una posición de memoria que será actualizada por un proceso.

La consulta del estado de un reloj (leer la hora por así decirlo), es una parte importante del problema. Cuando se quiere sincronizar un reloj con uno de referencia (sería el que proporciona la referencia a *t*, el tiempo/hora verdadero), se observa para un determinado *t* cuál es el valor de T(clk) y se toma como T0. Para el caso de la pendiente, se toman dos referencias a T(clk) y se dividen por el intervalo medido en *t* en los que fueron tomados. En el caso en que el reloj de referencia y el que se busca sincronizar estén en diferentes computadoras, se agrega un problema de comunicaciones, debido a variaciones en los tiempos de transmisión, tema analizado en la siguiente sección.

En el caso de las diferentes frecuencias de los relojes de dos computadoras, las frecuencias se normalizan para contar con una unidad común. En el caso de querer utilizar segundo o microsegundo como unidad, si la frecuencia de un oscilador es 2GHz y la frecuencia del otro reloj 3GHz, dividiremos en un caso por 2 y en el otro por 3, para obteber microsegundos. Pero a pesar de ello, puede ser que los "microsegundos" que mide cada máquina sean diferentes. Ello es debido a que para normalizar utilizamos el valor de frecuencia del oscilador medido por el sistema operativo en base a un par de referencias de tiempo suministradas por el timer local de cada máquina. Si bien estos relojes miden tiempo real pueden no estar sincronizados en frecuencia y medir "segundos" diferentes, con lo que los ciclos por segundo de las dos máquinas están medidos en unidades diferentes. Esto provoca que además del error inicial, los relojes de las dos máquinas vayan variando su diferencia en hora. En la sección de experimentos con referencia local y única se trata con más detalle la solución de este problema. En forma gráfica sería:

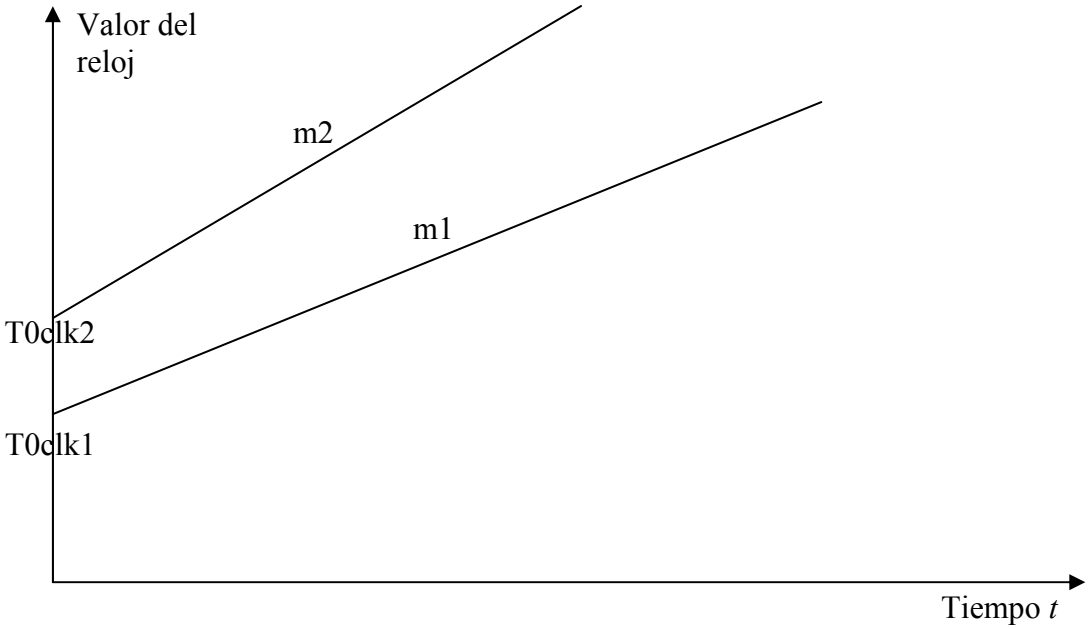


Figura 4.2: Representación gráfica de las diferencias entre dos relojes

En la figura 4.2 vemos cómo los valores de hora de las dos máquinas se van alejando con el tiempo a partir de una diferencia inicial dada por los diferentes T0. Esta diferencia de hora/tiempo se la llama *offset de tiempo*. Si la pendiente es constante (aproximadamente lo es en el caso de los relojes de computadoras, como se verá), se podrá corregir las horas de ambas máquinas con una resta entre el valor actual del reloj al momento de la sincronización y el valor que debiera tener (offset inicial) y un producto/cociente calculado en base al cociente de las frecuencias de ambos osciladores (cociente que estará relacionado con el offset de frecuencia, o sea la diferencia de frecuencias), o lo que es equivalente, el cociente de las pendientes m1 y m2.

Dicha corrección se ve degradada con un error producto de:

1) La variación en las pendientes, o sea m no será un valor constante sino que puede variar con el tiempo (debido al drift de frecuencia de cada oscilador; por lo general ambos valores son diferentes).
2) El tiempo transcurrido desde la consulta del reloj del valor T0 de referencia hasta su actualización en el reloj a sincronizar.

En el caso de tratarse de una red de computadoras, las referencias de tiempo las proporcionará otra computadora, tanto para el valor inicial como para la determinación de la pendiente (sincronización interna). La computadora que proporciona la referencia puede estar conectada con una fuente externa de hora ajustada a algún estándar como UTC, con lo que entonces se logra sincronización externa.

En el caso de osciladores/relojes reales, recordemos que debe tenerse en cuenta:

1) Que la frecuencia varía de su valor inicial a través del tiempo (drift de frecuencia) por condiciones ambientales y en algunos casos variaciones de tensión de alimentación, con lo que la gráfica no sería una recta (m no constante).
2) Esta variación de la frecuencia se puede apreciar a corto o largo plazo.

En el corto plazo, el valor de la hora se ve afectado por los errores de la consulta de estado del reloj (ver experimento realizado en esta sección). Además, puede variar con el paso del tiempo debido a las diferencias en frecuencias que aparecen a largo plazo por variar respecto de su valor inicial *m* en los osciladores de los relojes locales de cada máquina (drift de frecuencia), lo cual lleva necesariamente a volver a medir las diferencias de frecuencia o volver a sincronizar para evitar que se vayan separando los valores de hora de los dos relojes.

A los errores de offset en valor inicial y los causados por la frecuencia se los puede agrupar como la desviación interna del reloj [1]. Por otro lado, hay perturbaciones externas al reloj que pueden afectar las medidas y superponerse a las desviaciones internas, cambiando incluso el signo de las mismas. Son los problemas de consulta de estado. Existe un ruido de medición para la consulta del estado del reloj, tanto al momento de obtener una referencia para sincronización desde una máquina remota como localmente al solicitarla (consulta de estado local y remota). En este aspecto es que las comunicaciones entre computadoras y los tiempos implicados deben ser analizados para determinar el error introducido: También deben analizarse los sistemas operativos, sobre todo en el caso de los que no son de tiempo real.

En resumen, los valores de hora en dos computadoras pueden presentar a partir de ser sincronizadas en un valor, con un cierto error inicial (*offset* o diferencia inicial de hora):

- Un error de *deriva* (aumento del offset de hora, producto del *offset o diferencia de frecuencia, skew* en inglés)
- Este error irá creciendo en el tiempo dado por las diferencias en sus frecuencias.
- A este error se lo podría considerar una primera derivada de la hora respecto del tiempo, ya que indica la velocidad de cambio de las horas en ambas máquinas en el tiempo
- Además, hay otro error aleatorio dado por el cambio en el valor de frecuencia de cada una de las frecuencias de ambos relojes respecto de su propio valor inicial
- Este último error es causado por el *drift* de frecuencia, representado en la bibliografía [30][3] por el símbolo ρ
- Una definición de *drift*: es el cambio de la frecuencia de un mismo reloj desde su frecuencia inicial, por lo general debido a factores de temperatura y tensión de alimentación

4.3 Experimento sobre Deriva de Relojes en una Computadora

En línea con el tema de deriva en relojes, se realizó un experimento, utilizando un sistema operativo de tiempo real, RTAI [30][31]. El mismo consta de ejecutar una tarea de tiempo real como módulo del kernel, con una frecuencia de periodicidad de 4 segundos. La tarea lee el valor de TSC. Para fijar esta periodicidad, el planificador de tareas de RTAI utiliza el timer. Por ello disponemos de una medida de cuánto avanza TSC en 4 segundos medidos con el timer. Si los relojes de RTAI para el timer del planificador y TSC fueran perfectos, el valor de diferencia de TSC entre mediciones debiera ser constante. Las diferencias en estas lecturas mostrarían el error de medición, determinado por la deriva entre estos dos relojes, el timer y TSC. La figura 4.3 muestra

los resultados del experimento y en la tabla 4.4 figuran los valores que se muestran en el gráfico.

Como puntos aclaratorios y destacables del gráfico, señalamos que:

- El eje horizontal esta graduado en la cantidad de partes por millón (ppm) en que difieren las medidas
- En el eje vertical la frecuencia con que aparece esa medida
- La frecuencia acumulada, en línea continua, muestra cómo la mayoría de los valores están bajo 1,6 partes por millón (ver detalle en la tabla 4.1), lo que concuerda con experimentos realizados con otro hardware por otros investigadores [6][3].
- Los valores mayores a 1,6 ppm son escasos y atribuibles a ruido del experimento, teniendo en cuenta que, si bien se trata de un sistema operativo de tiempo real, al estar involucradas interrupciones, puede haber latencias no deseadas.

Si bien la deriva introducida por las diferentes frecuencias parece lineal, una observación detallada demuestra que no. En estas variaciones es que se observa el drift de cada oscilador o, mejor dicho, las diferencias de drift entre ambas frecuencias, las de TSC y timer. Las variaciones surgen a partir de que en las medidas, realizadas cada 4 segundos medidos por el timer, además de ser diferentes los incrementos de TSC, estos incrementos no crecen linealmente, lo cual daría un histograma plano en alturas de frecuencias. En un oscilador de cuarzo como los usados en computadoras para el reloj se pueden observar variaciones en la frecuencia del mismo de 1 ppm por ºC de cambio de la temperatura [6][3][3].

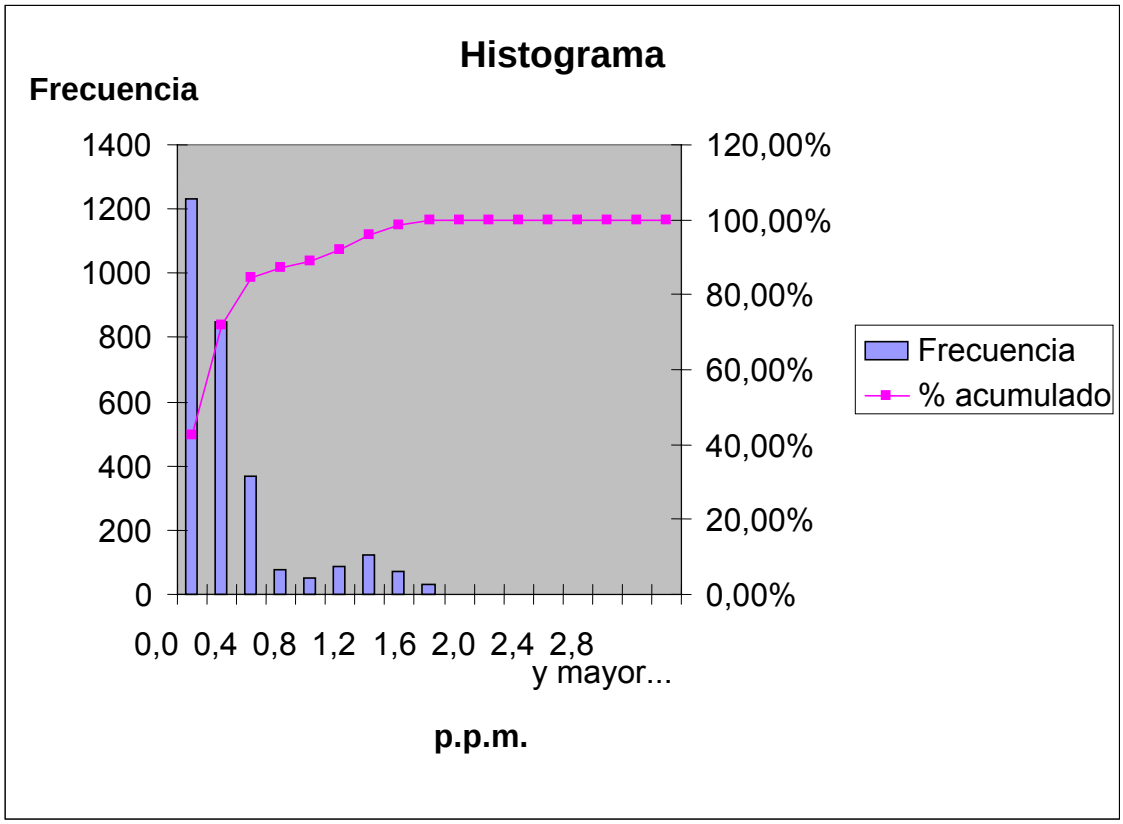


Figura 4.3: Deriva de tiempo de TSC respecto del valor del timer del scheduler

	Clase	*Frecuencia*	*% acumulado*
	0,0	1233	42,61%
	0,2	847	71,87%
	0,4	369	84,62%
	0,6	75	87,21%
	0,8	49	88,91%
	1,0	87	91,91%
	1,2	122	96,13%
	1,4	74	98,69%
*****	**1,6**	**29**	**99,69%**
	1,8	2	99,76%
	2,0	2	99,83%
	2,2	2	99,90%
	2,4	0	99,90%
	2,6	1	99,93%
	2,8	1	99,97%
	3,0	0	99,97%
y mayor...		1	100,00%

Tabla 4.1: Frecuencias de los valores de deriva

Por otro lado, se han realizado experimentos en los cuales de observan cambios en los valores de frecuencia del oscilador medidos con el *timer* en el arranque del sistema y cuando se produce la estabilización térmica unos minutos después. Varia alrededor de 11 ppm [8]. Esto lleva a que las mediciones en el momento de arranque de dicha frecuencia, llevadas a cabo por el sistema operativo sean diferentes a las que se obtienen un tiempo después del mismo. Por ello, lo mejor es corregirlas lo más cerca de la medicion de tiempo posible, por ejemplo antes de una evaluación de rendimiento.

Se destaca que:

- Los relojes del experimento no están sincronizados por ningún mecanismo.
- Se realizaron 2900 mediciones, o sea que el experimento tardó 2900 x 4 = 11600 segundos.
- Se verificaron a corto y largo plazo los errores en ppm, ya que tomando cualquier subgrupo de valores nos dio resultados de ppm similares, o sea que la estabilidad a largo plazo es buena o al menos del orden de lo que es a corto plazo.

En este experimento hay implicados dos relojes y un sistema operativo de tiempo real. No es lo mismo que ocurre en una LAN al querer comparar los relojes de dos máquinas.

4.4 Errores Debidos a la Comunicación de la Referencia

Otro aspecto del problema es que la sincronización de dos relojes en una LAN implica una consulta de estado remota. Esta consulta de estado remota estará afectada por el tiempo de transmisión entre las máquinas, que introducirá un error en las referencias

iniciales *T0* y las necesarias para determinar *m*. Si dicho tiempo fuese constante, se podría medir y eliminar este error adicionando la constante necesaria. Este tiempo, como ambos relojes están desincronizados, debe ser medido por un solo reloj, con lo cual lo que se mide es el tiempo de ida y vuelta de un paquete en la red de comunicaciones (round trip time). Se estima el tiempo de transmisión como la mitad del tiempo de ida y vuelta, pero la medición experimental muestra que esto no es necesariamente cierto. Además, el tiempo no es constante, variará en función de varios factores, tales como congestión en la red, diversos caminos que toman los paquetes, etc. No es el caso de este trabajo, ya que se trata de una red local, y se tiene control sobre el tráfico. Pero sí influyen factores tales como la ubicación en caché o memoria de datos e instrucciones implicados, las diferentes demoras de placas de red y la gestión de interrupciones.

Es necesario, aunque no se pueda eliminar el error, tener una estimación o caracterización del mismo [44]. Para ello se realizaron experimentos tendientes a medir la moda, valor mínimo y máximo en un ambiente como el que se especifica en los requerimientos, o sea, con acceso exclusivo o al menos controlado de la red de comunicaciones entre computadoras y de las computadoras, dentro de una red de área local, tal como el caso de los clusters de computadoras. Para acotar la parte variable del error, se midió el mínimo tiempo que tarda el sistema en acceder a los buffers. Como una forma de amortiguar el efecto de tiempos muy dispares en las comunicaciones de referencias, solo se consideran válidas las trasmisiones si fueron realizadas en un tiempo mitad del tiempo de ida y vuelta igual a la moda [3].

Ademas existe el problema de asimetria entre ida y vuelta. Si trasmito solo al valor de la moda, dicha asimetria estará dada por la diferencia entre la moda y el mínimo, ya que en el caso en que dicha moda esté compuesta por un tiempo de ida y vuelta asimétrico, nunca será menor al valor rtt mínimo en el cual la ida y vuelta son mínimas (aunque no podemos saber si simétricas) y un máximo que será el complemento a la moda del rtt mínimo, tal como se puede ver en la figura 4.4. A esto habría que sumarle un valor dado por la asimetria probable del tiempo mínimo de ida y el de vuelta, que es lo que compone el rtt mínimo. Lamentablemente, este valor no se pudo medir, aunque es de esperar que la diferencia entre tiempo de ida mínimo y de vuelta, sea casi despreciable en el caso de una red local frente a los demas tiempos implicados en el experimento. Esta aproximación no se indica en los gráficos por razones de claridad.

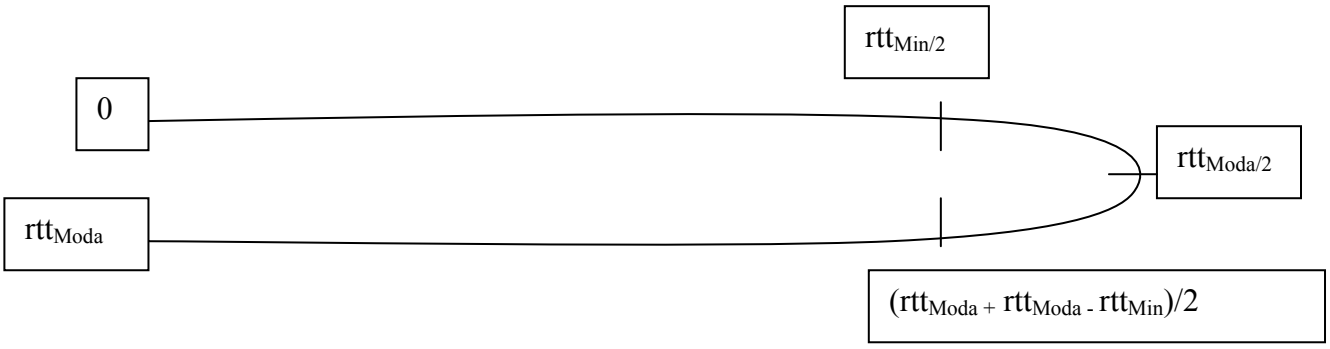


Figura 4.4: Round trip time

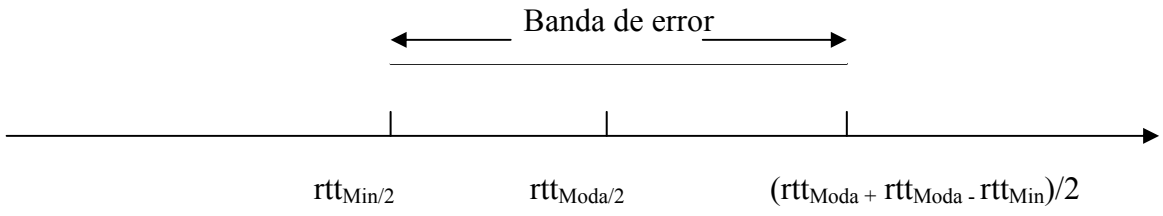


Figura 4.5: Banda de error

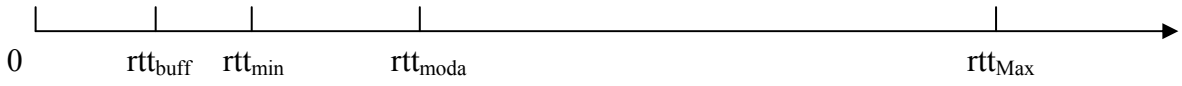


Figura 4.6: Valores relativos de tiempos de round trip time

La banda de error estaría acotada entre los valores de la moda y mínimo, tal como muestra la figura 4.5. Esta banda de error debe poder estar especificada, pues afecta a todas las mediciones.

En la figura 4.6 se muestran los valores relativos de round trip time a partir de los que se deducen los utilizados en los otras figuras.

4.5 Algoritmos de Sincronización

En los últimos años se ha abordado el problema de la sincronización de relojes de computadoras desde diferentes puntos de vista. En esta sección se exponen los principales algoritmos que resuelven el problema. Hay, además de los presentados, otros algoritmos derivados de los expuestos. En todos los casos, para la comunicación de las referencias de tiempo, se utiliza el sistema de comunicaciones ya existente entre las computadoras que se desea sincronizar. Un análisis más profundo de la cuestión nos lleva a abordar el tema de comunicaciones entre computadoras en forma extensa en otro capítulo.

4.5.1 Algoritmo de Lamport

El algoritmo de Lamport es uno de los primeros propuestos para la sincronización de sistemas distribuidos [11] y se basa en la relación “sucede antes” más la utilización de los mensajes entre las computadoras como indicadores precisos de esta relación. Más específicamente, un mensaje no puede ser recibido antes de ser enviado y, por lo tanto, si se tienen marcas de tiempo de los envíos de los mensajes se puede verificar si el tiempo actual es coherente con la definición de la relación “antes de”. Para ello se definen relojes lógicos. Un resumen de lo que realiza el algoritmo sería:

- Se analiza qué sucede antes

- a→b significa a antes de b
- Si en un proceso el evento b sucede después de a, entonces a→b es verdadero
- Si a identifica el evento de enviar un mensaje y b el de recibirlo, a→b es verdadero (no puede ser recibido antes de ser enviado)
- Si asigno valores en el tiempo C(a) y C(b), C(a) < C(b)
- Si no se cumple, adiciono el valor necesario a C(b). Nunca resto, ya que el tiempo debe ser siempre creciente
- En un mismo proceso, para dos eventos a y b C(a) debe ser diferente de C(b) (exigencia de resolución de reloj)

Las tareas a llevar a cabo en cada computadora son relativamente sencillas, aunque afectan, en cierta manera, la forma en que se procesan los mensajes:

1. Se tiene un reloj local.
2. Cada vez que se envía un mensaje, se le *agrega* al mismo una marca de tiempo (*timestamp*) con el tiempo local del que envía.
3. Cada vez que llega un mensaje, se analiza la marca de tiempo del que envió, y
    a. si la marca de tiempo es menor que el tiempo local, se asume que las computadoras están sincronizadas
    b. si la marca de tiempo es mayor o igual que el tiempo local, se cambia el tiempo local con la marca de tiempo del mensaje que se recibió más 1 (asumiendo, por ejemplo, que la transmisión necesita 1 unidad de tiempo)

En la figura 4.7 se observa esta mecánica en acción sobre tres hosts identificados como "0","1" y "2". A la izquierda se observa el intercambio de mensajes y a la derecha los cambios realizados una vez que se intercambiaron los mensajes:

- El mensaje A va de host0 a host1. Como el reloj es mayor que su timestamp, no lo cambia.
- Lo mismo el mensaje B de host1 a host2.
- El mensaje B tiene un timestamp de 60 saliendo de host2 y llega a host1 que tiene 56, lo que implica que tenga que ajustar su reloj local a 61, o sea 60 + 1.
- El D lleva desde host1 un timestamp de 69, al llegar a host0 provoca un ajuste del reloj local a 70, o sea 69 + 1.

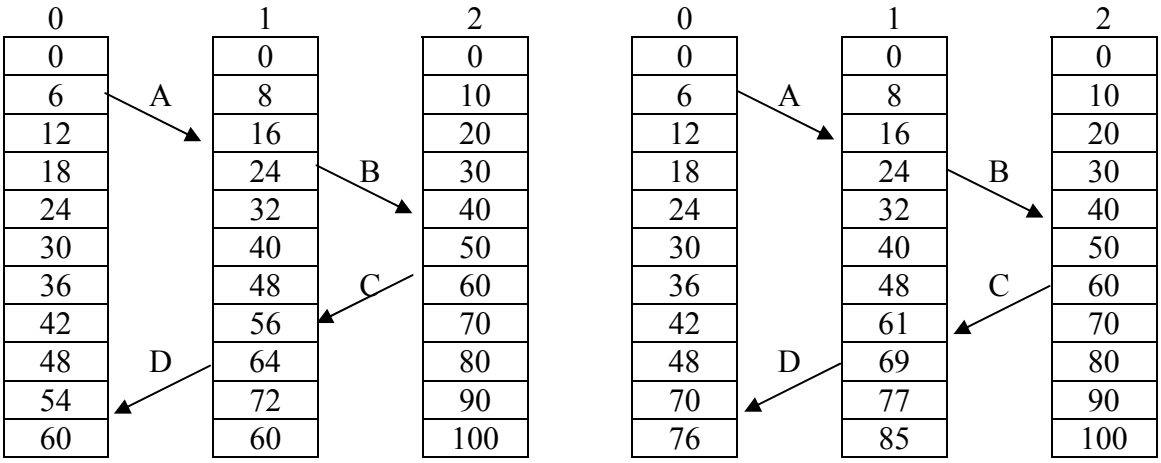


Figura 4.7: Intercambio de mensajes con timestamps en el algoritmo de Lamport

La figura 4.8 muestra más en detalle el avance del tiempo en dos computadoras, C1 y C2, y la forma en que la computadora C2 cambia su tiempo local a partir de la llegada de un mensaje con una marca de tiempo mayor que su tiempo local.

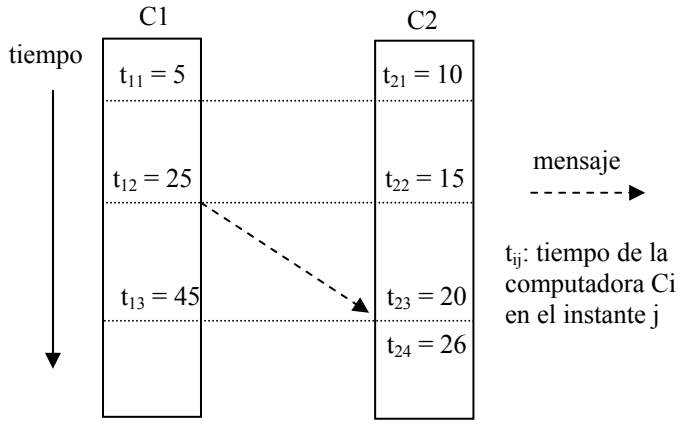


Figura 4.8: Evolución de eventos en el algoritmo de Lamport

Este algoritmo soluciona el problema de la escalabilidad, ya que solo exige un mensaje entrante y uno saliente para sincronizar cada máquina, pero no tiene un orden total de los eventos que suceden entre diferentes computadoras, excepto los relacionados directamente con envío y recepción de mensajes. Por ejemplo, en el caso de la figura 4.8, no es posible determinar la relación de tiempos entre los eventos que suceden en la computadora C1 entre los instantes de tiempo t_{12} y t_{13} con los que suceden en la computadora C2 entre los instantes de tiempo t_{21} y t_{23}.

4.5.2 Algoritmo de Cristian

Cristian [4] [53] [12] [13] [14] definió los conceptos de sincronización interna y externa, ya vistos. Recordamos que sincronización interna se refiere a mantener un grupo de relojes sincronizados, no importa qué hora tengan pero que en el grupo sea la misma, o con un margen de diferencias acotado a algún valor conocido. La sincronización externa podría definirse como mantener sincronización interna con alguna fuente fiable de hora tipo UTC.

En este marco de definición, Cristian propone sincronizar un conjunto de relojes de máquinas a partir de una que esté sincronizada externamente (tiempo u hora real), a través de la red de comunicaciones de datos entre computadoras.

El algoritmo de Cristian es probabilístico ya que no garantiza que un procesador pueda leer un reloj remoto con una precisión específica. Sin embargo, intentando una cantidad de veces suficiente, la hora recuperada puede ser leída con una precisión dada, con una probabilidad cerca de uno, según lo deseado.

Por otro lado, no se sabe cuánto es el máximo tiempo de envío de un mensaje entre dos computadoras. Algunos algoritmos suponen un tiempo máximo de envío. Son algoritmos determinísticos en el sentido que suponen que siempre llega el mensaje con la referencia en un tiempo menor al tiempo máximo. No pueden ser usados para sincronizar sistemas asincrónicos, ya que es imposible determinar el tiempo máximo. Estos algoritmos suponen un intercambio de mensajes, para n procesos a sincronizar, de n^2. Esto dificulta la escalabilidad en redes con gran número de nodos.

Cristian asume que:

1) El tiempo mínimo de demora de un mensaje $t_{\min}$ es conocido.
2) La función de distribución de la demora en los mensajes es conocida.

Esto lo asume en función de haberlo probado experimentalmente [3] en forma estadística: se envían sucesivos paquetes hacia una máquina que los contesta en la forma mas inmediata posible. Se registra la hora de salida del paquete y la hora de recepción de la respuesta, computándose por diferencia el tiempo de ida y vuelta (round trip time, rtt). Se supone que los valores de moda, mínimo y máximo encontrados serán constantes para tiempos posteriores.

Si un proceso *q* recibe un mensaje *m* desde un proceso *p*, evidentemente enviado por *p*, m sufre una demora, arbitraria y aleatoria.

Definimos:

- ρ como la tasa de deriva de la frecuencia del reloj local de *p*, este valor se mide en partes por millón (ppm) y por lo general lo caracteriza el fabricante.
- $H_p(t)$ es el valor que se lee en el contador del reloj de hardware de *p* en el tiempo real *t*
- El tiempo real *t* es definido de igual manera que el tiempo verdadero, como el más cercano a algún estándar como UTC.
- $H_p(s)$, el medido en el tiempo *s*.
- Se asume una máxima demora σ en la cual un proceso es despertado por el sistema operativo para darle el uso de la CPU, o sea, una demora del planificador máxima.

De acuerdo a este algoritmo y definiciones anteriores, el tiempo medido por un reloj será correcto respecto del tiempo real *t*, si está dentro del intervalo:

$$(1-\rho)(t-s) \le H_p(t) - H_p(s) \le (1+\rho)(t-s)$$

Esta fórmula marca los límites de la deriva o drift de un reloj. Por otro lado, si un proceso en tiempo *t* adquiere una medida del timer de *W* unidades de tiempo, el sistema operativo despertará al proceso en el intervalo de tiempo

$$[t+(1-\rho)W, t+(1+\rho)W+\sigma]$$

Pueden ocurrir tres tipos de fallas en un proceso que trata de sincronizar:

- Por rotura: cuando ante la pérdida de un evento no puede recuperarse
- Por performance: cuando reacciona ante el evento muy lentamente, posiblemente por culpa del sistema operativo y excede la demora σ.
- Arbitraria: Son las demás fallas. Por ejemplo un proceso que reacciona demasiado rápido o que envía información errónea.

Un reloj correcto será consecuencia de un proceso correcto, o sea, sin fallas.

Tomando en cuenta estos intervalos, se puede describir la idea básica del algoritmo como [fig.5]:

a) Un proceso *p* (cliente) envía a un proceso *q* (servidor) una solicitud de hora en un mensaje m_1 en su tiempo local t_0.

b) El proceso q consulta su reloj local y contesta con un mensaje m_2 con su tiempo local t_1.

c) Al llegar el mensaje m_2 al cliente *p*, este toma su tiempo local t_2.

d) Estima la demora en llegar el mensaje con la hora remota en $D/2 = (t_2 - t_0)/2$

e) Si el error en la lectura está dentro de un error constante *A*, el algoritmo toma el valor como correcto; caso contrario, lo descarta. El algoritmo supone que se retransmitirá varias veces el pedido de consulta de referencia remota hasta que dé con error aceptable, y si transcurren una cantidad de tiempo excesiva sin que ello ocurra, se considera que hay una falla en la lectura remota de la referencia.

f) Con éste valor corrige su hora, no en el reloj de hardware local sino en un reloj virtual o reloj de software, que no es sino una posición de memoria asociada al proceso que corrige la hora.

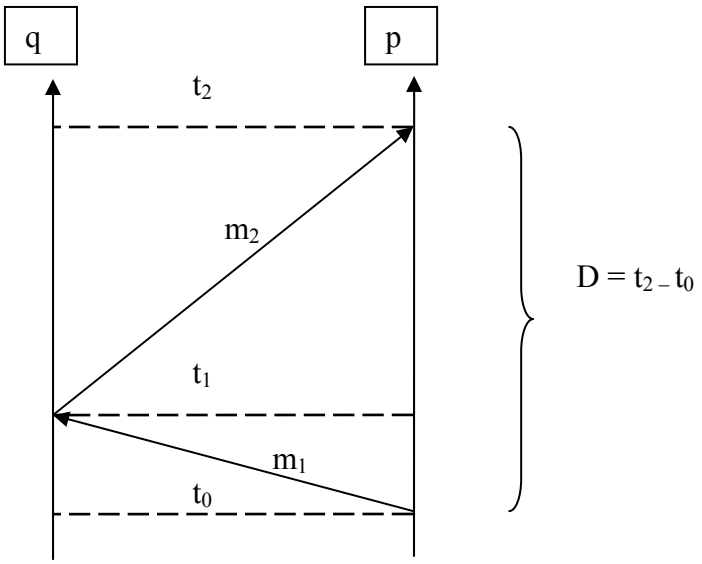


Figura 4.9: Intercambio de mensajes en algoritmo de Cristian

Por ello este método es estadístico, ya que estima que con probabilidad cercana a 1 conseguirá un valor dentro de la banda de error *A*, la cual definiremos a continuación. Para definir esta banda debemos considerar que el proceso *p* puede derivar su hora real en proporción al parámetro ρ, que mide las partes por millón en que varía la frecuencia de un oscilador a cristal. De esta manera, el tiempo de ida y vuelta (límite superior de *A*) no puede ser mayor a

$(t_2\text{-}t_0)(1+\rho)$

Este límite puede ser usado para determinar un tiempo máximo para la demora de la transmisión del mensaje en un sentido del mensaje m_2. El tiempo min surge de estadísticas sobre mensajes, o sea, es la mitad del mínimo *rtt*, estimado por el reloj local del cliente. En las siguientes ecuaciones, se destaca que los tiempos serían los del reloj local del cliente:

$$\max(t(m_2)) \equiv (t_2 - t_0)(1+\rho) - \min$$

- La condición $t(m_2) \le \max(m_2)$ se cumple ya que $(t_2 - t_0)(1+\rho) \ge t(m_1) + t(m_2)$ y $\min \le t(m_1)$, por la definición de tiempo de transmisión mínimo min.
- Estos valores min y max pueden ser usados para aproximar el valor del reloj del proceso *q* en el tiempo t_2.
- El valor del reloj de q se incrementa por $min(1\text{-}\rho)$ al menos y por $max(m_2)(1\text{-}\rho)$ cuanto mucho durante la transmisión de m_2.

Por lo tanto, A estará limitado por estos valores de [min, max].

$$min(1\text{-}\rho) < A < max(m_2)(1\text{-}\rho)$$

O sea que ρ del reloj local del cliente afecta a ambas medidas. Para minimizar el peor caso de error que *p* tiene en la estimación, el reloj de *q* en la hora t_2, simbolizado como $C_q(t_2, p)$, estará definido como el punto medio del intervalo

$$[t_1 + \min(1-\rho), t_1 + \max(m_2)(1+\rho)]$$

Por lo tanto, el punto medio quedaría:

$$C_q(t_2, p) \equiv t_1 + \frac{\max(m_2)(1+\rho) + \min(1-\rho)}{2}$$

El límite del peor caso de error $\mathrm{E}_q(t_2, p)$ que *p* puede cometer en la aproximación de la hora del reloj de *q* a la hora t_2, en la estimación de la mitad del intervalo será:

$$\mathrm{E}_q(t_2, p) \equiv \frac{\max(m_2)(1+\rho) + \min(1-\rho)}{2}$$

El proceso p espera un cierto tiempo antes y después de enviar un mensaje de referencia para enviar otro.

Si A es el máximo error aceptable, T_0 la hora en que *p* empieza a estimar la hora de *q*, t_0 un punto en el tiempo real tal que $C_p(t_0) = T_0$, D el máximo tiempo que *p* se toma para leer con un cierto error acotado la hora de q, S una constante positiva, T una hora entre $T0 + D$ y $T0 + D + S$ en la cual p necesita leer la hora de q con un cierto error menor que A. t es un punto en el tiempo real en el cual el reloj virtual de p muestra $T = C_p(t)$. Para que una lectura probabilística requerida para acceder a *p* y estimar $C_q(T, p)$ del

reloj de q con una función de error $E_q(T, p)$ sea correcta, deben satisfacerse las siguientes condiciones:

- Puntualidad: la lectura del reloj remoto toma a lo sumo D unidades de tiempo del reloj p.
- Límite de error: Si p y q no sufren fallas arbitrarias entre los tiempos T0 y T de la lectura remota, la diferencia entre el valor actual de q y el estimado por p deben ser:

$$\left| C_q(t) - C_q(T, p) \right| \leq E_q(T, p)$$

- Manejo del error: si p es correcto pero q falla durante el intervalo [t_0, t] en que dura la lectura, el error es infinito.
- Si ninguno de los dos falla durante el intervalo de tiempo [t_0, t], la probabilidad de que el error sea menor o igual que A es estrictamente positiva:

El algoritmo de Cristian presenta ciertos inconvenientes [21]:

- Al contar con un servidor de hora, si este falla, queda sin referencia. Cristian propone que los clientes hagan los requerimientos a varios servidores y se queden con la referencia que llegue primero.
- La segunda cuestión es con relación al ajuste del reloj. Si la hora de referencia recibida desde el máster es menor a la que se desea ajustar, dicho ajuste implica un atraso, con lo cual dicho reloj pasa dos veces por la misma hora. Esto traería aparejados múltiples inconvenientes; entre otros, uno bastante grave: la hora de actualización de archivos. En algunos aparecerían en el futuro modificaciones del pasado, con los inconvenientes que ello presenta, por ejemplo, para utilidades como *make*. En las implementaciones de este algoritmo, para atrasar, se baja la frecuencia del reloj durante el tiempo necesario para atrasarlo sin escalones.
- Tiene los problemas de escalabilidad de toda arquitectura cliente-servidor.

4.5.3 Algoritmo de Berkeley

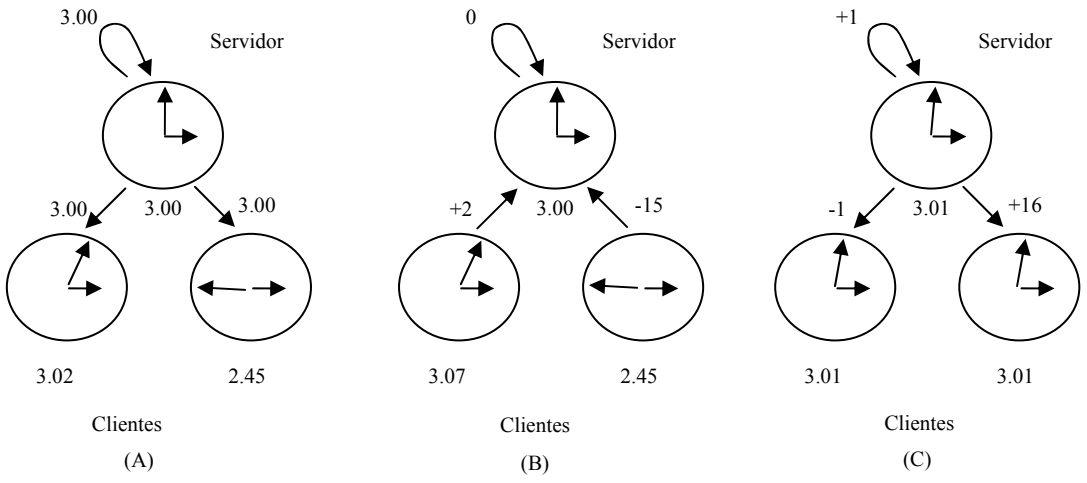


Figura 4.10: Fases del algoritmo de Berkeley

El algoritmo pasa por tres fases [22][24]:

1) El servidor (máster) es activo, pregunta a cada cliente (slave) por su hora (figura 4.10a)
2) Calcula el promedio, descontando los que están lejos del mismo (figura 4.10b)
3) Informa a cada cliente cómo debe cambiar la hora (figura 4.10c)

Estos intercambios de información se realizan a través de paquetes de datos que circulan por la red de interconexión entre las computadoras que se desea sincronizar, con las demoras correspondientes. Estas demoras deberán tenerse en cuenta a la hora de evaluar la corrección para sincronizar los relojes.

Supongamos una función derivable y continua C tal que aplicada al reloj de una computadora A nos de un valor $C_A(t)$, siendo t el tiempo Galileano Universal. La situación de intercambio de una referencia entre el servidor A y el cliente B se describe en la figura 4.11

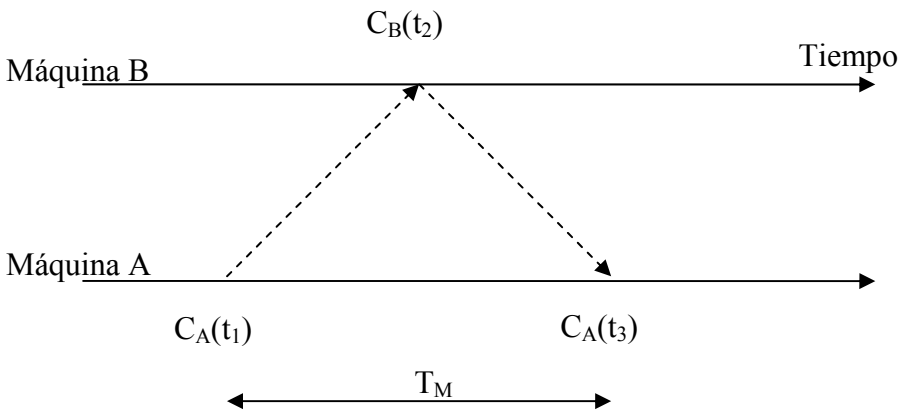


Figura 4.11: Intercambio de mensajes de referencias de hora entre dos máquinas, donde se puede apreciar el tiempo de ida y vuelta del mensaje

La diferencia entre los relojes será:

$$\frac{C_A(t_1) + C_A(t_3)}{2} - C_B(t_2)$$

Ejemplo de implementación: timed de Unix (Linux)

- Se puede correr varios máster, eligen entre todos uno, pero si se caen, vuelven a elegir a través de un algoritmo de elección.
- No toma en cuenta el rtt, pero como trabaja con diferencias, no le afecta.

4.5.4 Trabajos de Mills

Un aporte importante del trabajo de Mills, entre otros, es la definición de términos utilizados para la caracterización del problema de la sincronización de relojes.

Una fuente de hora en una red estará caracterizada por el jitter, deriva(wander), y la confiabilidad (reliability). Jitter es el valor cuadrático medio de una serie de diferencias de horas de los relojes, mientras deriva es el valor cuadrático medio de una serie de diferencias de frecuencia de los relojes.

La confiabilidad de un sistema de sincronización de hora es la fracción de tiempo en la que está conectado y operando correctamente en cuanto a tolerancias de estabilidad y precisión.

Resolución es el grado en el cual una lectura del reloj puede ser distinguida de otra, normalmente es la recíproca de la frecuencia de oscilación del reloj. La precisión es el grado en el cual una aplicación puede distinguir una lectura de reloj de otra, definida como la latencia en leer el sistema de reloj, y es una propiedad del hardware y del sistema operativo.

Precisión es el grado en el cual una lectura del reloj difiere del tiempo real o verdadero tal como lo distribuyen los estándares nacionales.

Mills ha recopilado los principios de los algoritmos ya vistos, y ha realizado una implementación, NTP (Network Time Protocol). Si bien debiera considerarse solo como implementación, debido a la gran cantidad de trabajos y aportes al tema por este autor, se incluye en este capítulo y en el de implementaciones.

Mills comienza estableciendo un modelo matemático del tiempo T(t) mostrado por un reloj en la época t [29][30]:

$$T(t) = T(t_0) + R(t_0)[t - t_0] + 1/2D(t_0)(t - t_0)^2 + x(t)$$

Donde $T(t_0)$ es el tiempo en alguna época previa t_0, $R(t_0)$ es la frecuencia y $D(t_0)$ es la deriva (derivada primera de la frecuencia) por unidad de tiempo. $T(t_0)$ y $R(t_0)$ son estimados por algún proceso. El término de segundo orden es ignorado. La naturaleza aleatoria del reloj estará caracterizada por x, en términos de frecuencia, fase o medidas de varianza.

- Mills define *offset de tiempo/hora* como la diferencia de hora entre dos relojes en cualquier instante.
- Como *offset de frecuencia* a las diferentes frecuencias de ambos relojes.

La diferencia de frecuencias, es un error sistemático. Se puede medir, y calculando la constante de proporcionalidad f1/f2 se corrigen los relojes que pudieran construirse con dichos osciladores. Este tipo de ajuste se conoce como calibración de un reloj cuando se lo realiza con un patrón tal como un reloj atómico controlado.

A este error sistemático se le agrega la deriva de frecuencia (drift en inglés). La frecuencia del reloj puede no ser estable, debido a cambios ambientales tales como la temperatura, tensión de alimentación del oscilador, etc. En este caso, aún llevado a cabo la calibración, al cambiar la frecuencia, cambia la constante de calibración. Este drift esta especificado en partes por millón.

El primer término de la ecuación de Mills es offset de tiempo (inicial, el valor a partir del cual se actualizan ambos relojes), el segundo (a) es offset de frecuencia, el tercero (b) (1/2 D(...)) sería el debido a la deriva en las frecuencias (drift) y por último, (c) es el término aleatorio x(t) que no mide, se desprecia.

Se debe comparar el tiempo y las frecuencias a través de servidores de tiempo/hora, para proveer sincronización en la subred.

Esta comparación se logra intercambiando mensajes entre máquinas que deseen sincronizar, cada uno con 3 referencias de tiempo, tal como se ve en la figura 4.12:

- T1 hora de envío de A hacia B
- T2 hora de arribo a B
- T3 hora de envío de B hacia A
- La referencia de tiempo T4 se toma al momento de llegar a la computadora A de vuelta, o sea no forma parte del mensaje.

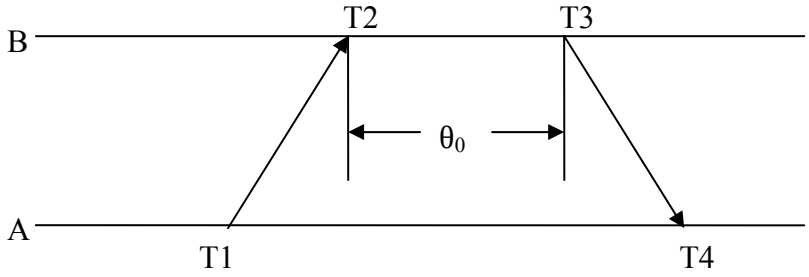


Figura 4.12: Intercambio de mensajes entre dos máquinas según Mills

Se asume *T3 > T2*. Y que las demoras para enviar un mensaje desde A hasta B son iguales a enviarlo desde B hacia A (tiempos simétricos). O lo que es lo mismo, se considera pequeña la diferencia de demora introducida por la red (demora diferencial). Si $a = T2 - T1$ y $b = T3 - T4$, el offset del reloj θ y el tiempo ida y vuelta (round trip delay) δ de B relativo a A en el tiempo T4 será:

$$\theta = \frac{a+b}{2}$$

$$\delta = a - b$$

Cada máquina puede calcular su offset o diferencia de hora y demora o delay. El cálculo del offset y el delay es continuo, en cada paquete. Este método es utilizado en telefonía. Tiene por ventaja que alguna pérdida de paquetes no afecta al método. El ajuste de la hora se hace para lograr un reloj monotónico, o sea que siempre avance, con lo cual en caso de estar adelantado, se baja su frecuencia hasta lograr gradualmente un ajuste.

Para solucionar los problemas de escalado que presenta el modelo cliente-servidor que utiliza NTP, recurre a una arquitectura de estratos, en la cual la hora suministrada por los relojes de referencia (estrato 0) es repartida entre los de estrato 1, que a su vez los distribuye entre los de estrato inferior, y así sucesivamente hasta alcanzar el estrato 16, en una arquitectura tipo árbol como muestra la figura 4.13. En las últimas versiones se prevé la utilización de 100 estratos.

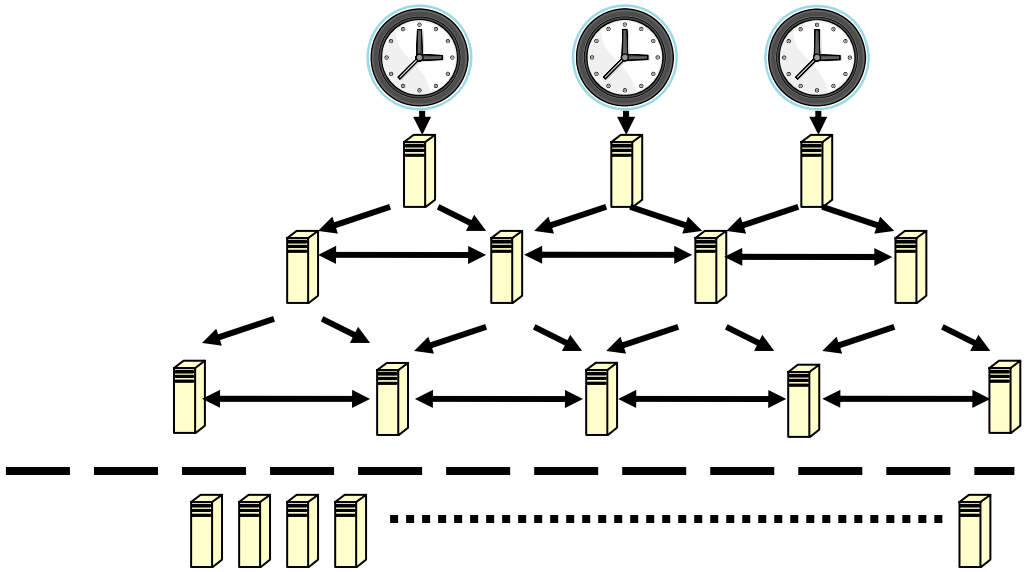


Figura 4.13: Estratos de servidores de hora de NTP

En la figura 4.14 Se puede ver el esquema del modelo de NTP. Ya hemos visto que para sincronizar relojes no basta con tomar un valor único de referencia de hora, ya que además del offset de hora existe el offset de frecuencia y el drift. Para mejorar el error en estos parámetros, en NTP se recurre a un oscilador disciplinado, implementado a través de un lazo enganchado en fase (phase-locked loop, PLL).

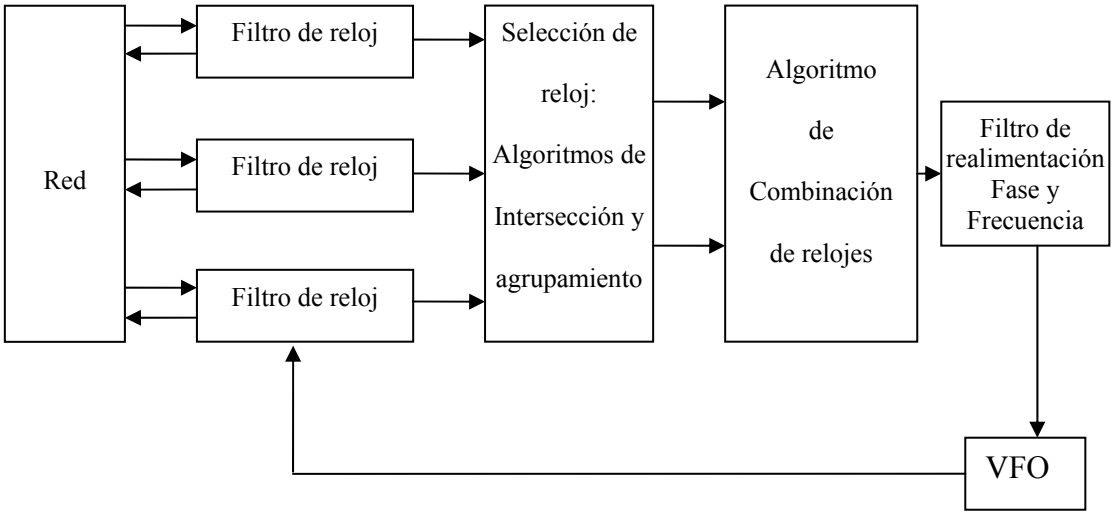


Figura 4.14: Arquitectura de NTP

El PLL provee una forma de ajustar el reloj local en hora y frecuencia en respuesta a las correcciones del protocolo de sincronización.

Los mensajes con la hora son intercambiados entre el servidor y otros pares ubicados posiblemente en otra subred. Son utilizados para determinar las demoras individuales de tiempo de ida y vuelta y el offset de hora, estimando un límite de error.

- Las demoras y offsets de cada par son procesados por un algoritmo de filtrado para reducir ruidos aleatorios accidentales. Este algoritmo selecciona entre los últimos ocho mensajes los que presentan una demora mínima y presenta como salida el correspondiente offset. El algoritmo de filtro de relojes opera en una ventana móvil de ejemplos para producir tres estimadores estadísticos de cada reloj: demora, offset y dispersión (θ, δ, y ε respectivamente).
- El algoritmo de intersección y agrupamiento selecciona el mejor subconjunto de la población de referencias de hora de los mensajes. Primero, los ordena por estrato y luego, por distancia de sincronización λ. El algoritmo de combinación computa los promedios de θ y ε del ensamble de relojes.

El algoritmo de intersección es el que se encarga de tomar de los ejemplos los verdaderos, descartando los falsos de los m mensajes con que se cuente. La condición de verdadero o falso está determinada por estar dentro de intervalos de confianza (como el Algoritmo de Marzullo y Owicki)[34], de manera similar al utilizado por DTS.

El reloj de NTP local se va corrigiendo (disciplinando) continuamente en hora y frecuencia. Un oscilador del tipo a los usados en los relojes locales cambia su frecuencia con la variación de temperatura. Esta variación está caracterizada por la Varianza de Allan [2].

El **error máximo** en el cálculo se define como *distancia de sincronización λ.* Indica el tiempo total de ida y vuelta (roundtrip delay) a la fuente de referencia primaria de hora, por ejemplo, al reloj de GPS. [41]. El **error esperado** nominal se llama *dispersión ε*.

Los componentes de ε incluyen:

1) El error máximo en la lectura del reloj local de cada máquina., que depende de la resolución del reloj y el método de ajuste.
2) El error máximo de frecuencia a partir del último mensaje recibido.
3) La dispersión entre pares (peers), a partir de la contribución de cada par dada su propia latencia en su SO y demora en la red a la fuente primaria de hora.
4) La dispersión seleccionada, que se obtiene a partir de combinar los pares para disciplinar el reloj local.

De todos los errores, el que mas pesa es la demora de la red. Desgraciadamente, este valor varía en forma casi aleatoria en función de la carga de la red y no se puede calcular de una vez para siempre.

Por lo tanto, la distancia de sincronización queda:

$$\lambda \equiv \frac{\delta}{2} + \varepsilon$$

En consecuencia, el offset con la fuente de hora original θ_0 estará en el llamado intervalo de confianza:

$$\theta - \lambda \leq \theta_0 \leq \theta + \lambda$$

ε y λ se usan como métrica en los algoritmos de selección de pares por intersección y de clustering. Fuera del intervalo de confianza se descartará una fuente de hora de un par.

Si las condiciones de seguridad lo requieren, los paquetes de NTP pueden tener procedimientos de autentificación criptográficos.

Este protocolo se destaca por contemplar gran cantidad de requerimientos en cuanto a seguridad, escalamiento, sobrecarga, tolerancia a fallos y demás y es el más utilizado actualmente. Pero justamente por cumplir con variados objetivos, no está optimizado para ninguno en especial. Se toma el concepto de calibración de frecuencia, para evitar el error sistemático que introducen las diferentes frecuencias de los relojes de computadora.

Capítulo 5: Redes de Comunicación de Datos

Resumen

En este capítulo se repasan los diferentes tópicos relacionados con las redes de comunicación: arquitecturas, protocolos, para finalmente enfocarse en los problemas de comunicaciones que se deben soluciona, teniendo en cuenta los objetivos de la tesis.

5.1 Introducción

Teniendo en cuenta que la información reside en computadoras, y que habitualmente se necesita compartir dicha información, es natural asociar computadoras con medios de comunicación. Una segunda necesidad de comunicar computadoras es la de compartir recursos, tales como impresoras, scaners y modems. Por estos motivos es que nacen las redes locales de comunicación de computadoras, que comunican computadoras dentro de un área pequeña, digamos restringida a una o varias oficinas dentro de un mismo edificio. Por otro lado, para lograr transferir información a distancias mayores, comenzó a ser frecuente el uso de la red de telefonía. Con el tiempo, nacieron sistemas de comunicación y redes especialmente diseñados para la trasmisión de datos entre computadoras a diferentes distancias.

De esta manera, la comunicación entre computadoras ha ido evolucionando a partir de redes locales de diferente tipo hasta llegar a Internet [52]. En Internet encontramos subredes de diferentes tipos, en común tienen todas la implementación del conjunto de protocolos que domina Internet, TCP/IP, condición necesaria para poder interactuar en esta red de redes. Dicho conjunto de protocolos al día de hoy se implementa en casi todos los sistemas operativos que se utilizan sobre diferentes arquitecturas de hardware. En particular, los sistemas basados en el SO Unix son predominantes, a partir del desarrollo de Unix Berkeley, encargado por el departamento de defensa de EEUU para poder conectar los laboratorios de investigación a la red DARPA.

Por otro lado, la necesidad de sincronización de hora entre computadoras se profundiza a partir del incremento de las comunicaciones entre ellas. Sobre todo a partir de la aparición de sistemas distribuidos, en donde una transacción puede involucrar datos y procesos en varias máquinas, con lo cual deben coincidir las horas de las modificaciones y las acciones en todas. Para realizar la sincronización de relojes seria deseable utilizar la misma red de comunicaciones. Este es el supuesto de los algoritmos ya vistos.

Para caracterizar las redes de interconexión de computadoras, empezaremos comentando que, debido a la gran cantidad de computadoras a comunicar y el costo de lograr una interconexión entre todas ellas, se ha impuesto la modalidad de multiplexar las comunicaciones, de tal manera que dos nodos que necesitan comunicarse solo disponen de interconexión real el tiempo en el cual intercambian datos y, el resto del tiempo, se utilizan los circuitos implicados en la interconexión para comunicaciones de otros nodos. Esta multiplexación ha sido posible gracias a la evolución de los sistemas de conmutación de comunicaciones. Para simplificar en lo posible la conmutación, se eligen tamaños estándares de datos para trasmitir. Dicha técnica es conocida como conmutación de paquetes (packet switching) para diferenciarla de conmutacion de circuitos, en los que un circuito conecta dos nodos durante todo el tiempo que dura la comunicación, ya sea que se trasmita o no. Las técnicas de conmutación de paquetes suponen la existencia de medios de trasmisión y de equipos que realizan la conmutación. Ello acarrea dos tipos de demora sobre las comunicaciones:

1) La que introducen los medios de trasmisión, que a pesar de realizarse a velocidades de la luz (en líneas fibras ópticas) o cercanas a ellas (en líneas de cobre), las grandes distancias introducen demoras, las cuales tienen como agravante el ser recorridos de longitud variable de acuerdo al camino que eligen

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

5.2 Comunicación entre Computadoras

Como vimos en la sección anterior, las computadoras habitualmente utilizan alguna red de comunicaciones como canal para comunicarse. Para conectar dos puntos en una red se puede:

1) Conmutar circuitos, de tal manera de generar un canal lógico con todas las conexiones, por el cual se trasmite sin retardos, y lo más rápido posible para liberar el canal para otra comunicación, como en telefonía.

2) Conmutar paquetes, en el cual la información a trasmitir se fracciona en "paquetes" de bits, que se envían junto con otros paquetes de otras comunicaciones, y en cada nodo se decide el camino o circuito a seguir por cada paquete (ruteo), con cual se optimiza el uso del canal, pero a costa de introducir retardos. Este método de transmisión de mensajes, similar al servicio de correos, es el mayormente usado por las redes de comunicaciones utilizadas por computadores.

Dentro de los diferentes tipos de redes de comunicaciones, podemos encontrar canales de dos tipos:

1) Canales punto a punto: Los canales de comunicación van entre dos nodos. Si dos nodos no tienen un canal directo, se comunican a través del reenvío de los paquetes de información entre nodos que provean un camino indirecto.

2) Canales de broadcast: en este tipo, el canal es común a varios nodos. Un paquete de información enviado por un nodo es recibido por todos los nodos. Cada uno verifica la dirección de destino del paquete. Si es la suya, lo recibe; si no, lo ignora.

Por otro lado, una red de comunicaciones ofrece servicios de diferente tipo. Dentro de ellos, podemos distinguir:

Servicios Orientados a Conexión
Es el modelo de la telefonía, en el cual se establece una comunicación, se habla y se corta la comunicación. En esta analogía, de un extremo se introducen paquetes de información como si fuese en un tubo, que sale en el destinatario. Es decir que se preserva el orden.

Servicios no Orientados a Conexión
Sigue el modelo del correo. Los paquetes poseen la dirección de destino. Al viajar por diferentes caminos, puede no preservarse el orden de partida al llegar a destino, con lo cual un paquete enviado primero puede llegar después de uno enviado segundo.

Con respecto a calidad de servicio, todos los servicios pueden ser confiables, en el sentido de que no habrá pérdidas de paquetes. Esto se implementa haciendo que el destinatario envíe una confirmación de la recepción, lo cual conlleva una sobrecarga de trabajo y de transferencia extra de datos.

Un Servicio especifica qué funciones debe cumplir una capa o componente de comunicaciones.

El Protocolo especifica cómo realiza dicha función, o sea, es la implementación del servicio.

Modelo de Capas

Para reducir la complejidad de diseño de una red de computadoras, se la considera como un modelo de capas (estrategia "divide y vencerás"), una sobre otra. Cada capa proporciona servicios de comunicación a la superior y obtiene a su vez servicios de la inferior. De manera que si dos procesos en dos máquinas diferentes pero de la misma capa (procesos pares) quieren comunicarse, se entienden entre si, pero haciendo uso de toda la pila de capas inferiores, salvo la capa uno (el medio físico) que es la única que solo ofrece servicios. Entre cada par de capas existen interfaces, con operaciones primitivas y servicios que la capa inferior ofrece a la superior. Una vez definidas las capas y las interfaces, la implementación de cada una de ellas puede variar, pero al mantener su interfase, la superior e inferior no pueden notarlo. El conjunto de capas y protocolos es llamado arquitectura de red.

El problema es que, al ser un tema que evoluciona y cambia año a año, la forma en que parece ser razonable dividir en un momento, no lo es tanto un tiempo después. El primer modelo de comunicaciones con esta estrategia es el modelo de capas [51], en el cual no solo se define la división, sino la forma en que operarán en conjunto. Es decir, puede variar el número de capas, el nombre, contenido y función de cada una, pero cada capa tiene como propósito ofrecer servicios de comunicación a las capas superiores, ocultando cómo están implementados dichos servicios. La capa n de una máquina puede establecer una "conversación" con la capa n de otra máquina. Las reglas y convenciones usadas en la conversación son llamadas en conjunto **protocolo.** Las entidades que realizan esta "conversación" son llamadas **procesos pares.** En la figura 5.1 se puede ver el modelo de una red de siete capas:

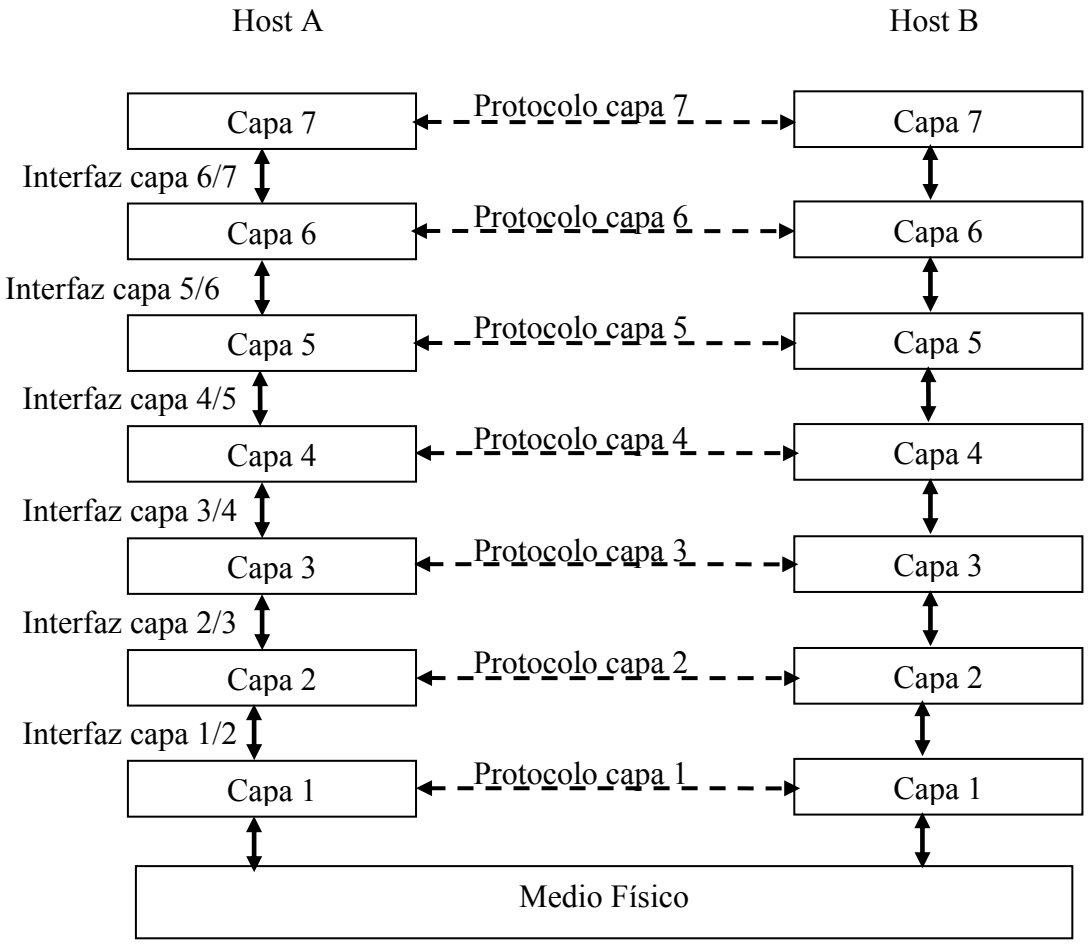


Figura 5.1: Modelo de capas de una red

A pesar de que los procesos pares se comunican entre ellos, la comunicación es virtual (en la figura 5.1, flechas de guiones). En la realidad, no lo hacen en forma directa. Los mensajes de datos y control van pasando en una máquina hacia las capas inferiores, hasta alcanzar el medio físico, para luego trepar por las diferentes capas en la otra.

Entre un par de capas adyacentes existe una interfaz. En la misma se definen qué operaciones primitivas y servicios ofrece una capa a la superior. El conjunto de capas y protocolos es llamado arquitectura de red.

5.3 El Modelo de Referencia OSI

Este modelo está basado en una propuesta de la International Standards Organization (ISO) como primer paso para lograr la estandarización de varios protocolos. También es llamado modelo ISO-OSI (Open Systems Interconnection). En la figura 5.2 [52] se puede observar su arquitectura de red.

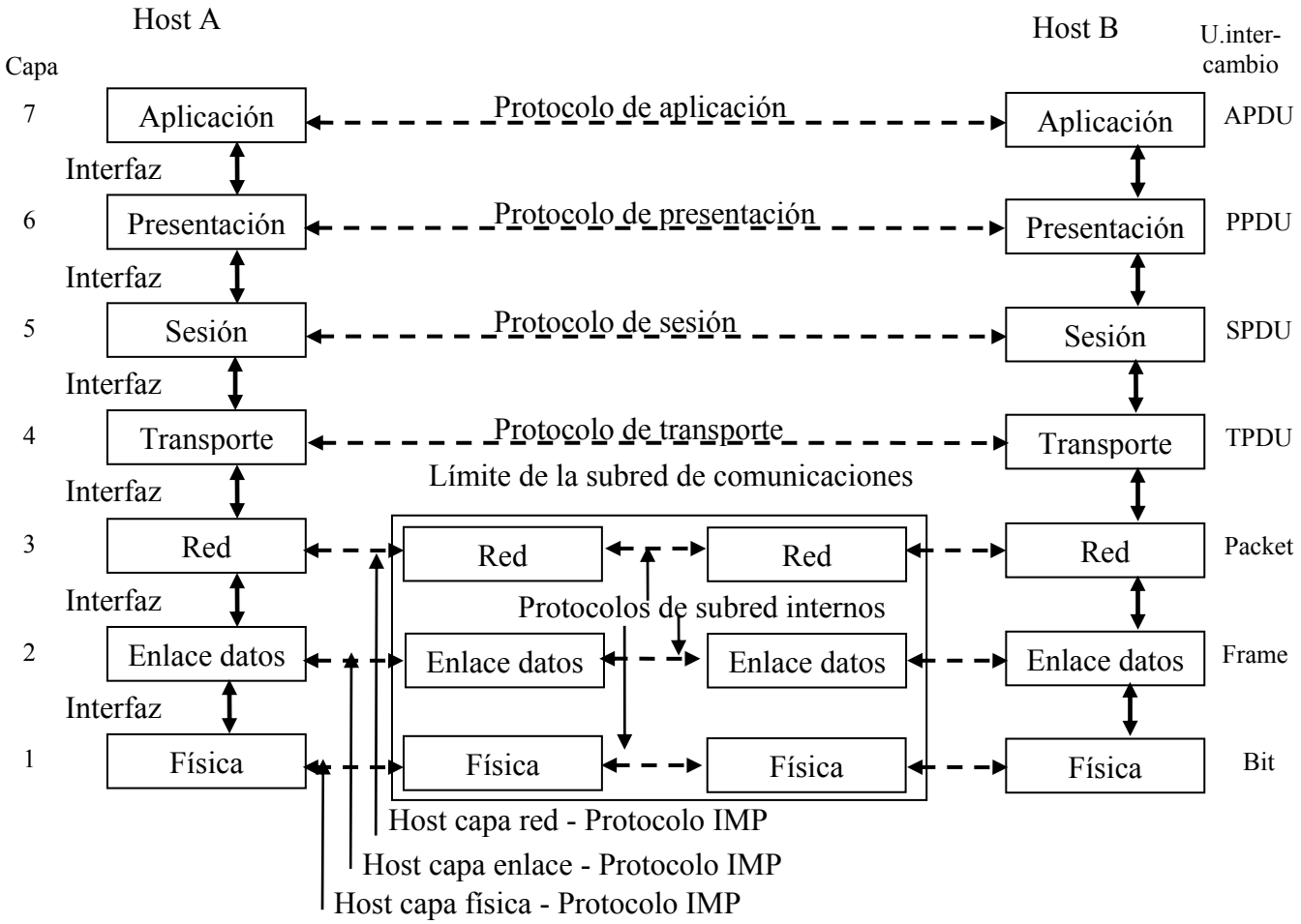


Figura 5.2: Modelo de capas OSI

Para cada capa se especifica su unidad de intercambio. El significado de las siglas es el siguiente:
APDU: Aplication Protocol Data Unit (unidad de datos del protocolo de aplicación)
PPDU: Presentation Protocol Data Unit (unidad de datos del protocolo de presentación)
SPDU: Session Protocol Data Unit (unidad de datos del protocolo de sesión)
TPDU: Transport Protocol Data Unit (unidad de datos del protocolo de transporte)
Packet: Paquete
Frame: Trama
Bit: unidad que representa un digito binario de valor cero o uno.

Capa 1: Física

Es la que se encarga de trasmitir bits "en crudo" sobre el canal de comunicaciones. De manera que si se envia un uno, al receptor le llegue un 1, no un bit 0.

En este nivel se definen cuestiones como:

- caracteristicas eléctricas como el voltage que representarán al 1 y al 0.
- Detalles de trasmision/ recepción simultanea o separada
- Cómo se inicia y finaliza una transmisión
- Pines del conector de red

Capa 2: Enlace de Datos

Se encarga de controlar y eliminar los errores en los bits que ofrece a la capa de red. Divide los datos de entrada en marcos (frames) de algunas centenas de bits, transmitiendo los marcos en forma secuencial y procesando los marcos de reconocimiento (acknowledgement frames) hacia el emisor de los recibidos. En virtud de que la capa fisica trasmite bits sin tener en cuenta límites, esta tarea está a cargo de esta capa y agrega patrones de bits al comienzo y fin del marco.

- En el caso de pérdida de un marco, se encarga de retransmitirlo.
- En caso de fallo en el marco de reconocimiento, puede ser que se envíen marcos duplicados. De estos problemas se encargará la capa de red superior. Ofrece a dicha capa servicios de diferentes calidades y costos.
- También se encarga de regular el flujo de transmisión en función del tamaño de los buffers del receptor y su velocidad de recepción.

Capa 3: Red

Funciones:

- Controla la operación de la subred.
- Determina cómo son encaminados (ruteados) los paquetes entre la fuente y el destino de transmisión. Estas rutas pueden ser estáticas y con casi ningún cambio, pueden mantenerse fijas entre el comienzo y finalización de una transmisión o pueden ser dinámicas y cambiar para cada paquete. En las redes broadcast no existe este problema.
- Controla la congestión de paquetes.
- Lleva la contabilidad del tráfico y quien lo provoca con el fin de aportar datos para la facturación de servicios.
- Soluciona los problemas devenidos de cambiar de una red a otra, ya que puede cambiar el modo de direccionamiento, el tamaño de paquete o el protocolo.

Capa 4: Transporte

- En envío recibe los datos de la capa de sesión y los divide en pequeños paquetes para pasarlos a la capa de red.
- En recepción, se asegura que lleguen todos los paquetes y en el orden correcto.
- Puede crear múltiples conexiones con la capa de red para aumentar la cantidad de datos por segundo (througthput) o multiplexar en una única conexión (para bajar costos) varios requerimientos de transporte hechos por la capa superior.
- Determina el tipo de servicio, siempre en el momento de establecer la conexión. El mas común es libre de errores, punto a punto, y en el orden que fue enviado, pero puede ser broadcast a multiples destinatarios o mensajes aislados sin resguardar el orden de envío.

Los niveles de 4 a 7 son end to end, o sea, verdaderamente entre fuente y destino, mientras que los niveles 1 a 3 se comunican con su nivel par en destino. Esto se aprecia en la figura 5.2.

Como en cada host puede haber varias conexiones en virtud de la multiprogramación,

esta capa se ocupa de ver qué conexión pertenece a qué proceso.

Capa 5: Sesión

Permite a usuarios en diferentes máquinas establecer sesiones entre ellos. Además de permitir transportar datos, esta capa provee servicios como el acceso a un sistema de tiempo compartido remoto y la realización de transferencias de archivos entre dos máquinas.

- Maneja el sentido del tráfico en ambas direcciones o solo en una.
- Por un mecanismo de tokens evita superposición entre usuarios en secciones críticas: solo ejecuta la operación el poseedor del token en ese momento.
- Puede insertar checkpoints, para recuperar operaciones entre dos máquinas ante fallas a mitad del proceso, como por ejemplo una transferencia de archivos, que de esta manera puede ser reanudada a partir del punto de falla, evitando retransferir la totalidad del archivo.

Capa 6: Presentación

Es la que se encarga de la semántica y sintaxis de los datos, para convertirlos, de ser necesario, al formato usado por la red para que todos lo usuarios puedan entender los datos. Ejemplo: cadenas de caracteres ASCII y EBCDIC.

Capa7: Aplicación

Maneja una terminal virtual, que adapta la terminal real a un tipo de comandos comunes. También se encarga de la transferencia de archivos, para presentárselo a los usuarios de una manera uniforme a ambos lados de la comunicación, mas allá de detalles tales como largo de linea de texto, etc.

5.4 ARPANET: TCP/IP

Arpanet [49] fue creado 10 años antes que el modelo ISO/OSI. Tiene una mezcla de capa 2 y 3. En capa 3 tiene un elaborado mecanismo de ruteo. Esto le permite verificar la correcta recepción en el destino de cada paquete enviado por la fuente. Vendría a ser otra capa de protocolo que no existía en el modelo OSI. Se lo trata como componente de la capa 3 por estar más cerca de ella que de otra. El Protocolo de Internet (Internet Protocol, IP), trabaja sin conexión. El Protocolo de Control de Transmisión, (Transmission Control Protocol, TCP) es orientado a conexión. No tiene capa de sesión ni de presentación. Sus servicios incluyen transferencia de archivos (FTP), correo electrónico (SMTP), y login remoto (TELNET).

En su definición TCP/IP tiene cinco capas relativamente independientes entre sí:

- Capa física
- Capa de acceso a la red
- Capa internet
- Capa extremo-a-extremo o de transporte
- Capa de aplicación

Capa Física

Equivalente a capa física del modelo OSI, se define tanto la interfaz entre el dispositivo de transmisión de datos (computadora) y el medio de transmisión (red) como las características eléctricas de éste en lo referido a señales, velocidad, etc.

Capa de Acceso a la Red

Es la responsable del intercambio de datos entre el sistema final (servidor o estación de trabajo) y la red. El emisor proporciona a la red la dirección de destino, para que ésta pueda encaminar los datos hasta el destino. Es decir que en esta capa se hace el ruteo y el acceso a la red. Aquí se definen servicios solicitados a la red como la prioridad. Hay diversos estándares desarrollados para conmutación de circuitos, de paquetes (Frame Relay, retransmisión de tramas) y de redes de área local como Ethernet. Las capas superiores debieran funcionar independientemente del tipo de red.

Capa Internet

Si dos dispositivos se encuentran en redes diferentes, los datos deben poder atravesar los dos tipos de red. De ello se ocupa esta capa. Esta capa implementa el protocolo IP (Internet Protocol) tanto en los sistemas finales como en los encaminadores intermedios que interconectan dos redes retransmitiendo los datos por el camino adecuado para alcanzar el destino.

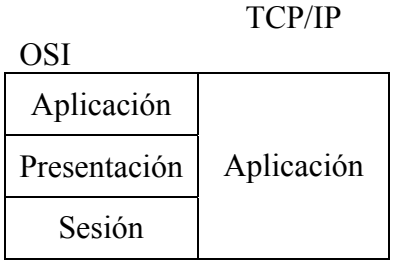
Capa de Extremo-a-Extremo o de Transporte

Proporciona los mecanismos para que los datos lleguen en forma fiable a destino, y en el orden correcto. En capa está el protocolo TCP (Transmission Control Protocol).

Capa de Aplicación

Contiene la lógica para posibilitar el funcionamiento de las aplicaciones de usuario, un módulo para cada tipo de aplicación.

En la figura 5.3 podemos ver la correspondencia entre los modelos OSI y TCP/IP



<table>
<tr><td></td><td rowspan="2">Transporte (origen - destino)</td></tr>
<tr><td>Transporte</td></tr>
<tr><td rowspan="2">Red</td><td>Internet</td></tr>
<tr><td rowspan="2">Acceso a la red</td></tr>
<tr><td>Enlace de datos</td></tr>
<tr><td>Fisica</td><td>Física</td></tr>
</table>

Figura 5.3: Modelo OSI vs TCP/IP

5.4.1 TCP, UDP, IP

En TCP/IP podemos distinguir dos tipos de puertos:

- TCP (Transport Control Protocol)
- UDP (User Datagram Protocol)

TCP proporciona conexión fiable lo cual implica que hay una entidad que controlará que los paquetes (segmentos TCP) lleguen a destino sin errores y en el orden que partieron del origen. En caso contrario, se reenviaran. Esto implica mensajes de confirmación de arribo desde el destino al origen. Este método hace que los mensajes lleguen como si hubieran recorrido un canal exclusivo para el envío aunque en rigor esto es falso, incluso muchos recorren caminos diferentes en la red. Por este efecto es que es llamado orientado a conexión.

En cambio, UDP no garantiza la entrega, ni el orden ni la existencia de paquetes duplicados. Como contrapartida, al no haber mensajes de confirmación, genera menor tráfico. En rigor, solo agrega a IP la capacidad de identificar puertos. Se dice que es no orientado a conexión.

Por último, se puede utilizar directamente IP, sin ningún protocolo de transporte.

5.4.2 Ventajas y Desventajas de los Diferentes Protocolos

Se observa que los diferentes protocolos ofrecen diferentes servicios, la cuestión es el costo implicado en el suministro de dichos servicios. Este costo se traducirá en tiempo de procesamiento de los paquetes por parte de la pila de protocolos, tamaño final de los paquetes respecto a los datos originales, y otros aspectos que influyen en la latencia y el ancho de banda efectivo al momento de una transmisión de datos. Es para echar luz sobre este aspecto, que se realizó un experimento utilizando paquetes de dos tipos de los vistos (UDP e IP), a fin de caracterizar latencia y ancho de banda para cada tipo de protocolo. Independientemente del protocolo, podríamos analizar los diferentes tiempos que involucran una transmisión de un paquete entre dos máquinas.

5.5 Tiempo de Envío de un Paquete entre dos Máquinas

Más allá del protocolo usado, y características de una transmisión, el envío de un paquete de una máquina a otra lleva un determinado tiempo. Una descomposición [6] del mismo sería:

Figura 5.4: Descomposición de tiempo de envío de un mensaje entre computadoras

Tiempo de envío: Es el gastado en la construcción del paquete en las distintas capas hasta alcanzar la capa de acceso al medio (MAC). Es variable en función del estado del Sistema Operativo (context switch, scheduling, etc.).

Tiempo de acceso al medio: Depende del protocolo usado. En el protocolo Ethernet dependerá del tráfico en la red. Es aleatorio.

Tiempo de transmisión: Depende de la velocidad de la placa de red y del largo del mensaje. Es determinístico.

Tiempo de propagación: en una red local será despreciable, ya que es lo que tarda la señal en recorrer el cable entre placas de red - switch - placa de red más el retardo que introduce el switch.

Tiempo de recepción: Es el tiempo que tarda el mensaje en atravesar hasta la MAC layer. Es determinístico.

Tiempo de recibo: Es el gastado en desarmar el paquete en las distintas capas hasta ser entregado a la aplicación. Depende del SO, al igual que el tiempode envío.

Del análisis de estos tiempos surge la posibilidad de evitar atravesar todas las capas del protocolo para optimizar comunicaciones y evitar las posibles demoras introducidas por el sistema operativo. En línea con esta posibilidad, se llevó a cabo un experimento comparando el ancho de banda y la latencia de una transmisión utilizando el protocolo UDP y paquetes Ethernet a través de sockets de tipo RAW. Como resultado de este experimento, se obtuvieron los siguientes valores:

Tamaño	tiempo IP	tiempo UDP
50	82	402
100	114	423
150	140	439
200	166	455
250	195	472
300	222	487
350	247	504
400	278	521

450	303	536
500	333	555
550	361	571
600	385	588
650	414	606
700	439	619
750	468	637
800	497	652
850	522	670
900	551	691
950	576	701
1000	603	718
1050	630	733
1100	656	750
1150	684	768
1200	713	781
1250	739	800
1300	768	818
1350	793	833
400	821	850
1450	849	865
1500	874	885

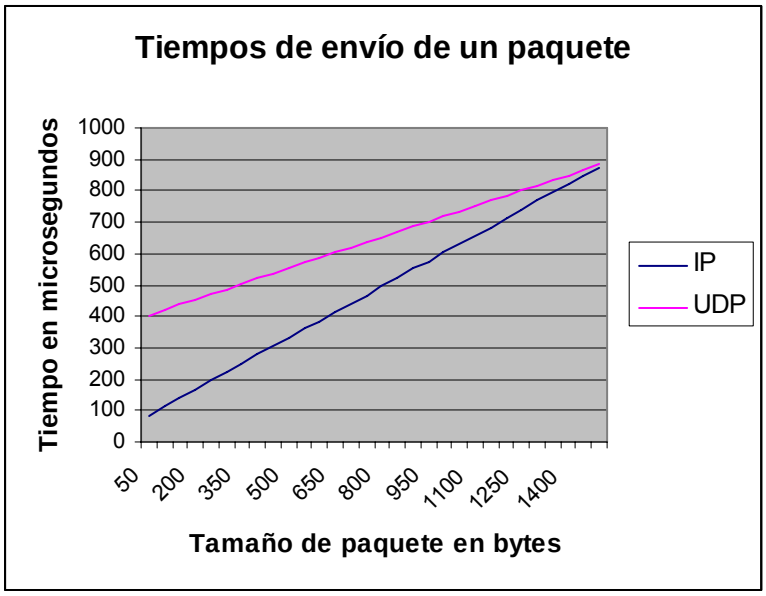


Tabla 5.1:UDP vs IP **Figura 5.5: UDP vs IP**

Nota: Los tiempos de la tabla y el gráfico tienen unidad de tiempo microsegundo y tamaño de paquete byte.

En la tabla 5.1 y en el gráfico de la figura 5.5 se pueden apreciar que los paquetes IP tienen un valor de latencia menor a los UDP (paquetes pequeños). Los valores deben ser divididos por 2 ya que son ida y vuelta (round trip time, rtt). Tenemos 402 microsegundos contra 82. Para los paquetes más grandes, la diferencia desaparece. La desventaja con IP es que los clientes y servidores con sockets de tipo RAW exigen derechos de ROOT. Posteriores experimentos detallados en el capítulo 7, realizados con otro hardware, dan para TCP latencias de 140 microsegundos.

De estos valores se deduce que:

- Para tamaños grandes de paquete, no importa demasiado el protocolo utilizado.
- Por otra parte, considerando el aspecto de fiabilidad, TCP suministra servicio fiable, no así UDP ni IP.
- Respecto al tráfico generado en la red, TCP genera el doble por los paquetes de reconocimiento (ack, acknowledge).

Para el diseño de la biblioteca de sincronización que se expone en el capítulo 7, se debe tomar en cuenta que la sincronización se realiza fuera del tiempo en que debe correr la aplicación a medir, con lo cual, el mayor tráfico generado, no tiene importancia. Pero sí la fiabilidad, que influye en el tiempo total de sincronización y el error de sincronización (un paquete que llegue fuera de orden o duplicado, puede alterar la estadística para el cálculo de la moda).

5.6 Intercomunicación con Referencias Externas de Tiempo

Las referencias de tiempo externas a una red suponen algún dispositivo a través de otro lugar de la entrada-salida diferente a la red. Este es el caso de los relojes atómicos y los receptores de GPS (Geo Posicionamiento Satelital).

En general, los relojes atómicos suelen estar en laboratorios de envergadura, conectados a hardware especializado, aunque se están desarrollando relojes más simples para reducción de costos.

Más comunes son los relojes de receptores GPS o relojes GPS que suministran la hora UTC. Este tipo de relojes suele utilizar el puerto rs232 o el usb para su conexión, proporcionando un pulso por segundo o un tren de pulsos de 1 Khz con tramas de diferentes formatos a través de estos puertos, para sincronización del reloj de cuarzo de la computadora.

De la sincronización con este pulso, se encarga el software. En particular, NTP puede ser sincronizado de esta manera, configurando así un servidor de hora del estrato 1, que puede distribuir su hora en una red.

Como ejemplos de los formatos de paquetes de hora UTC [22] damos 7 formatos diferentes de transmisión y recepción de paquetes de datos a través del puerto serial.

- Formato Garmin. Formato propietario que permite al usuario intercambiar "waypoints", rutas y puntos entre una computadora y el receptor GPS con el software adecuado
- Garmin DGPS (Diferencial GPS). Gestiona la entrada de señales DGPS provenientes de una baliza con receptor DGPS, utilizando para la comunicación el formato RTCM SC-104. Si la radiobaliza es de la marca Garmin, el receptor tomará las funciones de terminal de la radiobaliza. DGPS produce correcciones en los datos GPS.
- Salida NMEA. Este formato soporta el estándar NMEA 0183 versión 3.0. El protocolo NMEA 0183 es un medio a través del cual los instrumentos marítimos y también la mayoría de los receptores GPS pueden comunicarse entre sí. Ha sido definido y está controlado por la organización estadounidense National Marine Electronics Association
- Salida de texto. Transmisión de salida formato ASCII con datos tales como fecha, hora y localización. No permite ninguna entrada.
- Entrada RTCM. Soporte para la entrada de datos en formato RTMC SC-104 sin necesidad del uso de una radiobaliza. La transmisión de correcciones diferenciales entre el receptor de referencia (BASE) y los receptores remotos está estandarizada según propuesta de la Radio Technical Commission for Maritime Services, Special Committee 104, RTCM-SC104
- RTMC/NMEA. Permite entrada RTCM SC-104 con una salida de datos en formato NMEA 0183 versión 3.0.
- RTCM/texto. Soporte para la entrada de datos en formato RTMC SC-104 y salida en formato de texto.

Capitulo 6: Implementaciones Existentes

Resumen

En este capítulo se describen y caracterizan las implementaciones existentes: DTS, Timed y NTP. En el caso de las dos últimas, se realizaron sendos experimentos a fin de comprobar el grado de sincronización que era posible alcanzar dentro de una red local.

6.1 Introducción

Las implementaciones existentes utilizan mayormente una combinación de los algoritmos vistos, y cada una está pensada para diferentes conjuntos de requerimientos. Es importante destacar que la solución al problema de la sincronización en ambientes distribuidos implica un compromiso de enfoques, en donde la optimización de un aspecto deteriora otro. La solución ideal no existe ni tampoco la implementación definitiva, con lo cual surgirán nuevas implementaciones cada vez que se tome un subconjunto de requerimientos diferentes.

Los primeros intentos en implementaciones fueron:

- Daytime Protocol: especificado en el RFC 867 del año 1983, envía paquetes ASCII con la fecha y hora en segundos a través del puerto 13 en UDP o TCP.
- Time Protocol: especificado en el rfc868 del año 1983 para las redes ARPA. Los servidores de hora envían al cliente el tiempo en segundos a partir de la medianoche del 1º de enero de 1900. Esta es recibida en el puerto 37 tanto utilizando TCP o UDP, como un binario de 32 bits.
- También se usaron paquetes ICMP (Internet Control Message Protocol) para obtener la hora, que se usa en Timed de UNIX.
- IBM desarrolló DTS (Distributed Time Service).
- En la actualidad, NTP es el protocolo más usado.

6.2 DTS

DTS (Distributed Time Service)[15] es una implementación usada hace un tiempo por IBM para sincronizar relojes en sistemas DCE (Distributed Computing Enviroment). Actualmente, DCE utiliza NTP para esta tarea.

DTS provee sincronización en la hora de las máquinas de un sistema distribuido y se encarga de que dicha hora sea lo más cercana posible a la hora correcta, que en el caso de DTS, se considera la hora UTC.

Hay varios tipos de componentes de un sistema DTS:

1. Cliente (clerk) de hora
2. Servidores de hora
    1. Servidor de hora local
    2. Servidor de hora global
    3. Servidor de hora de correo
    4. Servidor de correo de respaldo
3. DTS API

4. Interfaz de proveedor de hora (Time-Provider Interface, TPI)

5. Formato de hora que incluye inexactitud

Los componentes activos son los clientes y servidores. Hay dos interfaces: la interfaz de programación DTS API (Application programmer interface) y la externa, TPI (Time Provider Interface), a un proveedor externo de hora. DTS define un formato para expresar la hora.

El cliente (clerk) corre en una máquina cliente y mantiene sincronizada la hora consultando al servidor y ajustando la hora local. Está configurado conociendo los límites del hardware del reloj local del sistema. Cuando ha pasado el tiempo suficiente para que la hora presente una inexactitud fuera de la especificada, el cliente realiza una sincronización consultando varios servidores y calculando en base a esos datos la probable hora correcta, con la cual actualiza el reloj local. La actualización puede ser gradual o abrupta. En la forma abrupta, se actualiza de una vez el valor del registro de la hora. En la gradual, la frecuencia de actualización del tick del reloj local es cambiada a fin de aproximar el reloj al valor correcto.

La figura 6.1 muestra una LAN con dos clientes y tres servidores. Cada cliente interroga dos servidores para sincronizar. Los servidores se interrogan todos entre si.

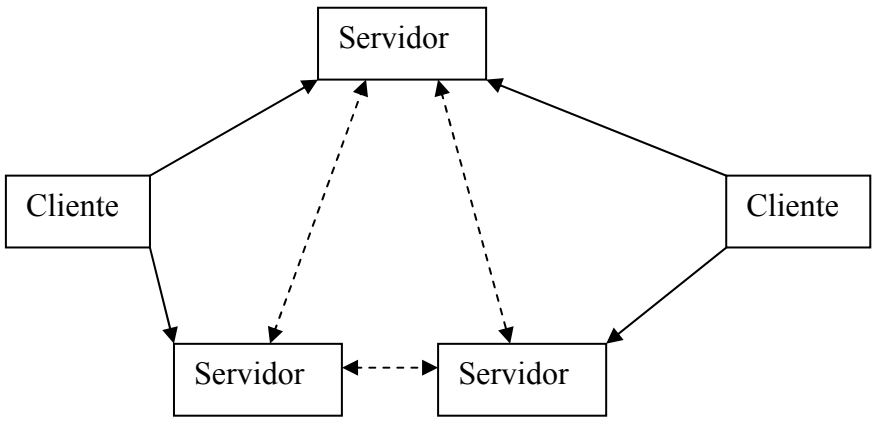


Figura 6.1: Clientes y servidores DTS en una red local.

Un servidor de hora está diseñado para responder requerimientos de hora en todo momento. En general hay tres servidores en una red local y uno o más conectados a un proveedor externo de hora, tal como UTC. Esto distingue entre servidores de hora local y global. Un cliente sabe a cuál está accediendo, y será función de su requerimiento a cuál interroga.

6.2.1 DTS API

DTS provee una biblioteca API que permite a los programadores manipular timestamps,

tanto para representar la hora en diferentes formatos como para compararlos.

Los servidores de NTP pueden proveer servicios de hora a DTS y los servidores de DTS pueden suministrar servicios de hora a un sistema NTP

6.2.2 Formato de hora

DTS provee un intervalo en vez de una hora. Por ejemplo, en vez de reportar las 9:52:0, reporta (exageradamente) "entre 9:51:0 y 9:53:0", dando idea del error que maneja.

DTS está escrito en C y provee una API con las siguientes funciones básicas:

- Recuperación de información de hora en timestamps
- Conversión entre timestamps binarias que usan diferentes estructuras de hora
- Conversión entre timestamps binarias y representación ASCII
- Conversión entre hora UTC y hora local
- Manipulación de timestamps binarios
- Comparación de dos valores de hora binarios
- Cálculo de valores binarios de hora
- Obtención de información de zona horaria

Un listado y descripción de cada función puede hallarse en [17]. Hora absoluta: DTS usa en sus procesos internos timestamps binarios que representan UTC pero que no pueden ser directamente leídos. Incluye la hora, la inexactitud de la misma, junto con otro tipo de información. Cuando uno necesita saber la hora, DTS la muestra con el siguiente formato:

1990-11-21-13:30:25.785-04:00I000.082

Este formato cumple la norma ISO 8601[61]. Dicha norma no especifica la inexactitud, por lo cual, en este formato, no se muestra. La figura 6.2 muestra el significado de cada campo.

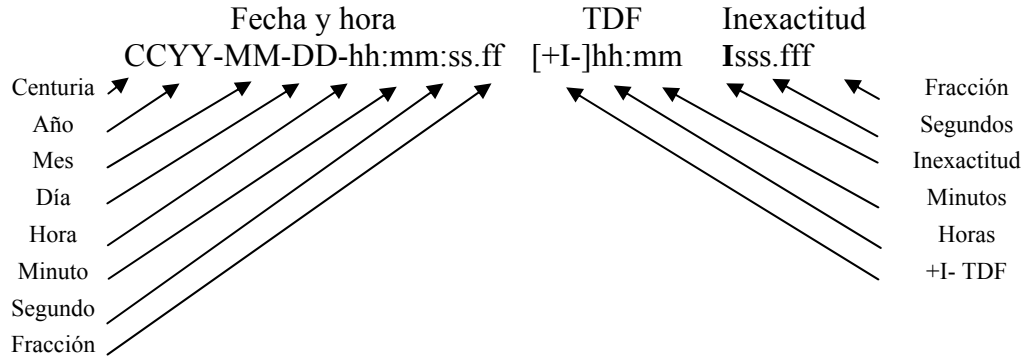


Figura 6.2: Sintaxis de un paquete DTS de fecha, hora e inexactitud

El signo +/- indica el offset respecto de la hora UTC. La presencia de este time differential factor (TDF) indica que, tanto la fecha como la hora, son locales del sistema y no UTC. La hora local es la UTC más la TDF. La inexactitud (inaccuracy (**I**)) indica el comienzo de la información de inexactitud.

Hora relativa: es un intervalo de tiempo discreto que usualmente se suma o resta a otro tiempo. Una TDF asociada con un tiempo absoluto es un buen ejemplo. En la figura 6.3 se muestra la sintaxis:

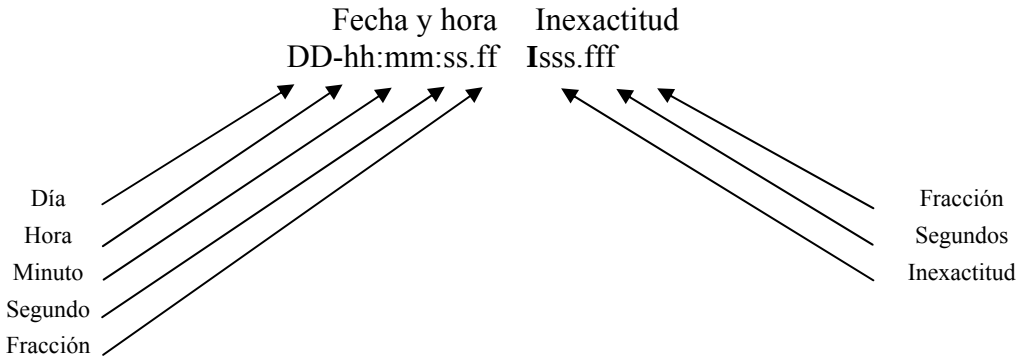


Figura 6.3: Sintaxis de un paquete DTS para una referencia de tiempo absoluto

El siguiente ejemplo muestra una hora relativa de 21 días, 8 horas, 30 minutos y 25 segundos con una inexactitud de 0,300 segundos:

```
21-08:30:25.000I00.300
```

La duración de un periodo de tiempo usa la siguiente sintaxis:

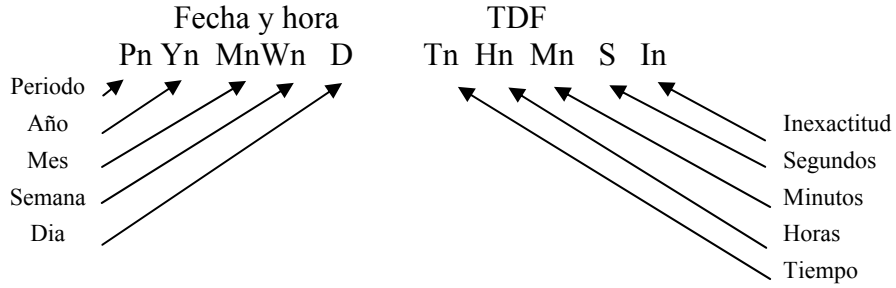


Figura 6.4: Sintaxis de un paquete DTS para un periodo de tiempo

TDF es la sigla de time differential factor, factor diferencial de hora con el que se corrige la hora de acuerdo a la zona horaria.

El ejemplo siguiente representa un periodo de un año, 6 meses, 15 días, 11 horas, 30 minutos, 30 segundos y no especifica inexactitud.

P1Y6M15DT11H30M30S

En cuanto a su arquitectura, DTS usa cliente-servidor a través de RPC, donde el daemon DTS es el cliente y el proceso proveedor de tiempo es el servidor, ambos corriendo en el mismo sistema. Detalles de dicho RPC se pueden ver en [16]. Actualmente no es posible conseguir una implementación de DTS, por lo tanto, no se realizaron pruebas con el mismo.

DTS utiliza las diversas referencias de los diferentes servidores para definir un intervalo de confianza en la hora. De esta manera, las referencias muy alejadas a este intervalo, quedan descartadas. Y con las referencias dentro del intervalo, se promedia la hora de referencia con la que se ajusta el reloj. Este esquema, similar al del algoritmo de Berkeley aunque más elaborado, también se utiliza en NTP.

6.3 Timed

Esta implementado sobre la base del Time Synchronization Protocol (TSP), un protocolo diseñado para proveer sincronización en una red de área local para el sistema operativo UNIX 4.3 BSD. El Timed está construido para cumplir con el protocolo DARPA UDP, y está basado en un esquema maestro-esclavo. Sirve para dos propósitos:

- Soporta mensajes de sincronización de los relojes de las máquinas de una red de área local.
- Soporta mensajes para la elección entre los esclavos en caso de desaparecer el maestro.

Cada host tiene un proceso time-daemon (timed), uno corriéndolo como maestro y los otros como esclavos. Los esclavos envían su hora al maestro, éste calcula un promedio en el cual se incluye, descartando los valores de los relojes muy alejados del mismo, y luego envía a los esclavos la diferencia con la cual deben ajustar. Asumiendo que el tiempo de ida y vuelta del mensaje es simétrico, al transmitir las diferencias, no hace falta tomar en cuenta dicho tiempo de transmisión.

Muchos de los mensajes requieren confirmación y en caso de no recibirla, deben ser retransmitidos. Esto genera un alto tráfico de mensajes en la red. Para medir las diferencias, utiliza ICMP Time Stamp Requests. Este proceso se repite periódicamente. Además, un proceso timed en el nodo que sirve de puerta de enlace permite hacer broadcast a otras redes de los paquetes de sincronización.

Para asegurar un servicio continuo y confiable, es necesario implementar un algoritmo de elección para elegir un nuevo máster en el caso de que desaparezca el proceso máster porque se rompe la máquina, por ejemplo, o porque se particiona la red por un problema de cableado o conmutación de paquetes. En este algoritmo, los esclavos eligen entre ellos a un nuevo maestro. Se debe destacar que la falla del maestro provoca solo una gradual divergencia en las horas de los esclavos, por lo que el proceso de elección no necesariamente debe ser inmediato.

Algunos mensajes requieren confiabilidad, algo que el protocolo ICMP no provee. Por ello, se utilizan mensajes de confirmación, número de secuencia de paquetes y retrasmisión cuando se pierde alguno. Cuando un mensaje que requiere confirmación no se recibe tras múltiples envíos, el daemon que lo envió asume que dicha dirección desapareció.

Relativo al timed de Linux, se realizaron pruebas, pero la herramienta suministrada por Linux, para medir diferencias de tiempos una vez que las máquinas se sincronizan, el tdiff, tiene una resolución de 1ms., reportando, en el mejor de los casos, entre 0 y 1, como se puede observar en la tabla 6.1. Timed utiliza el algoritmo de Berkeley. Debe

prestarse atención a los 4 minutos entre 16:15 y 16:19, que es el lapso que tarda en sincronizar, pasando del valor -36044 de diferencia a 0. Dado que la resolución de tdiff es de 1 milisegundo, todo error menor a 1 se reporta como cero.

Hora	Error(ms)
16:15:00	-36044
16:16:00	-36044
16:17:00	-36044
16:18:00	-36044
16:19:00	0
16:20:00	0
16:21:00	0
16:22:00	0
16:23:00	0
16:24:00	0
16:25:00	-1
16:26:00	-1
16:27:00	-1
16:28:00	-1
16:29:00	-1
16:30:00	-1
16:31:00	-1
16:32:00	-1

Tabla 6.1: Mediciones de sincronización con Timed

Timed [57] es un proceso servidor de hora de Unix/Linux Se invoca en el arranque desde el archivo rc. Sincroniza los clientes tomando en cuenta la hora de otras máquinas, aumentando o disminuyendo la velocidad de sus relojes para acercarlos a la hora promedio de las máquinas de la red. Dicho promedio se computa a partir de las diferencias de hora usando mensajes ICMP de requerimiento de hora. Está basado en el modelo cliente-servidor. Cuando el proceso timed arranca en una máquina, pregunta al máster de la red por la hora de ésta y a partir de allí, el master le informa periódicamente las diferencias en más o en menos a las que debe ajustar su hora a través de la llamada adjtime. En caso de caída del máster, se elige un nuevo máster entre los host que están configurados como posibles másters (a través del flag –F activado).

6.4 NTP

NTP (Network Time Protocol) [32][33][34][35][36][37] es la implementación de uso más generalizado para sincronizar computadoras. Si bien inicialmente se basa en el algoritmo de Cristian, a medida que aparecieron nuevos requerimientos, se fue haciendo más complejo. Así es que al requerimiento inicial de sincronizar relojes de computadora con una hora de referencia tal como UTC, surgieron temas como seguridad, que llevó a encriptar los paquetes de referencias de hora, y tolerancia a fallas. Los detalles de los algoritmos utilizados en esta implementación fueron presentados en el capitulo de algoritmos. Acá solo se resumen detalles, se analizan ventajas/desventajas y se muestran los resultados de los experimentos llevados a cabo con esta implementación. Su arquitectura básica está basada en el modelo cliente-servidor, que se muestra en la figura 6.5:

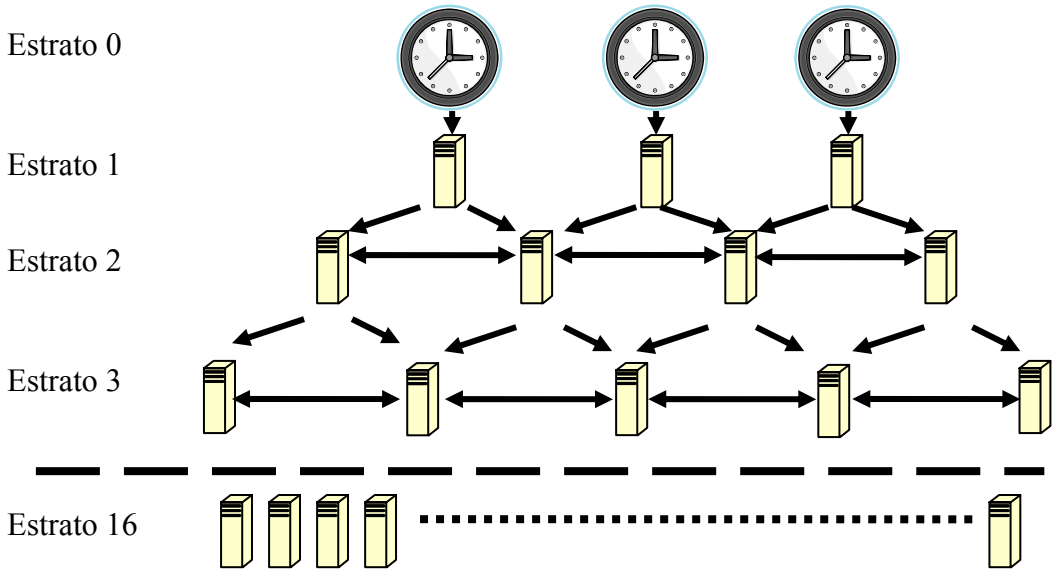


Figura 6.5: Arquitectura de estratos NTP

Los diferentes niveles o Estratos son una solución al cuello de botella del modelo cliente-servidor, representado por un servidor que no puede atender todos los requerimientos. Esto hace que los servidores de estrato 0, compuesto por relojes atómicos o relojes sincronizados por GPS, suministren la referencia horaria a los de estrato 1, que serán los encargados de distribuirla a través de toda la jerarquía. Para evitar el cuello de botella antedicho, solo se permiten consultas por parte de servidores previamente autorizados, autorización resultante de evaluar la carga actual del servidor. Con ello, todo servidor que esté conectado a un servidor estrato 1 será automáticamente estrato 2 y así sucesivamente.

Además del modo de conexión como cliente, un servidor NTP permite la conexión como peer, en cuyo caso servidores de estrato 1 en adelante pueden configurarse como un arreglo de servidores para mejorarse mutuamente la exactitud de la hora que suministrarán. El estrato más bajo es el 16 para la implementación en uso de NTP3, aunque en la nueva versión en desarrollo (NTP5) hay 256 estratos. Cuando un cliente se conecta a la red, lo hace a través de un servidor de estrato más alto que el propio y lo hará como estrato 16. Con el tiempo, alcanzará un estrato menos del servidor al que se conecta. Por ello, sería deseable conectarse con el de estrato más alto posible. Pero estos pueden estar congestionados o tener un vínculo de comunicaciones que produce demoras variables, por lo que tal vez sea mejor un servidor de un estrato más bajo pero con una comunicación mejor. La solución a este problema son los pools de servidores, a los que un cliente se conecta y le suministra el mejor servidor posible en ese momento, tanto por estrato como por congestión y comunicación.

Como parte de este trabajo se analizaron y cuantificaron las características de NTP. Esta cuantificación se refiere principalmente a:

- Precisión y resolución posible del reloj sincronizado. Aunque NTP es paramétrico, esto no significa que se puede obtener una resolución y una precisión arbitraria cualquiera. Se obtiene como producto del proceso.

- Sobrecarga en la red de comunicaciones.
- Error alcanzado en la sincronización.
- Tiempo en el cual se alcanza una sincronización aceptable.
- Cambios en dicha sincronización a través del tiempo.

Todo este análisis es básicamente experimental [35] [38] [39], dado que el funcionamiento de NTP se define por un lado, paramétricamente y por el otro, con el comportamiento o rendimiento de la red de interconexión.

NTP corrige dos tipos de diferencias (*offsets*) para sincronizar dos o más computadoras:

- **Offset de hora:** Va realizando ajustes de acuerdo a si la diferencia respecto de la referencia es grande o pequeña. Si la diferencia es mayor a 128 μs, se da en un escalón (*stepping*) y si es menor a 128ms, en forma gradual (*slewing*). En todos los casos, si el reloj local esta adelantado, va haciendo ajustes graduales de tal manera de no producir discontinuidades en la hora (hora posterior < hora actual).
- **Offset de frecuencia:** a partir de analizar la primera derivada de la diferencia de horas, establece si la frecuencia es mayor o menor a la de la referencia, cambiando el valor local de frecuencia al correcto.

En la tabla 6.2 se pueden observar algunos valores relacionados con los tiempos necesarios para alcanzar la sincronización de una máquina con respecto a un servidor en la misma red local. Las computadoras son iguales, con las características especificadas en la tabla 6.3 que es repetición de la tabla 3.2, y la red de interconexión es Ethernet de 100 Mb/s dedicada (no hay otro tráfico en la red) con *switch* de capa 2. Mientras no posee suficientes datos como para asegurar sincronización, NTP informa que no está sincronizado, y el valor de hora dependerá del reloj local.

	Hora	Offset	Dist. Sync.	synch?	Estrato(ref:12)
1	15:46:14	0,000018	0,00375	Not	16
2	15:47:00	0,000019	0,00444		
3	15:48:00	0,000022	0,00534		
4	15:49:00	0,000016	0,45090		
5	15:50:00	0,000016	0,45180		
6	15:51:00	0,000018	0,45271		
7	15:52:00	-0,000369	0,01141	Yes	12
8	15:53:00	-0,000431	0,01134		
9	15:54:00	-0,000442	0,01126		
10	15:55:00	-0,000446	0,01120		
11	15:56:00	-0,000451	0,01111		
12	15:57:00	-0,000459	0,01105		
13	15:58:00	-0,000457	0,01097		
14	15:59:01	-0,000461	0,01187		
15	16:00:00	-0,000455	0,01178		

Tabla 6.2: Valores de offset y distancia de sincronización

PC	CPU	Frecuencia (MHz)	Sistema Operativo
P42400	Intel Pentium 4	2400	Linux 2.4.18-14

Tabla 6.3: Detalles de la PCs utilizadas en el experimento.

6.4.1 Tiempo de Sincronización y Sobrecarga

En la tabla 6.2 se observa el proceso de sincronizar dos máquinas utilizando NTP. Las primeras seis filas muestran que la máquina no está sincronizada (columna “sinch?” en not). A partir de allí, se alcanza un valor de sincronización (columna “sinch?” en yes). Podemos observar que alcanzar la sincronización llevó un tiempo de 6 minutos (fila 7 de la tabla 6.2). El valor de la distancia de sincronización es de 0,01141 segundos con un offset de 0,000369 segundos. A partir de ahí, no mejoró demasiado la sincronización. A partir de alcanzar una sincronización mínima a los 6 minutos de haber comenzado (fila 7 de la tabla 6.2), NTP ya proporciona los valores corregidos. A partir de este primer instante de sincronización, los valores relacionados no mejoran, aún después de varias horas de funcionamiento de NTP, valores que fueron recortados de la tabla.

En la figura 6.7 se ve cómo los valores de offset comienzan a cobrar un valor verdadero a partir del momento de la sincronización a la hora 15:52, momento coincidente con la fila 7 de la tabla 6.2. Los valores anteriores no tienen sentido alguno, ya que el reloj no está sincronizado. Dichos valores, si bien no aseguran nada respecto del error, nos dan el valor de offset medido para el cual se calculó la distancia de sincronización.

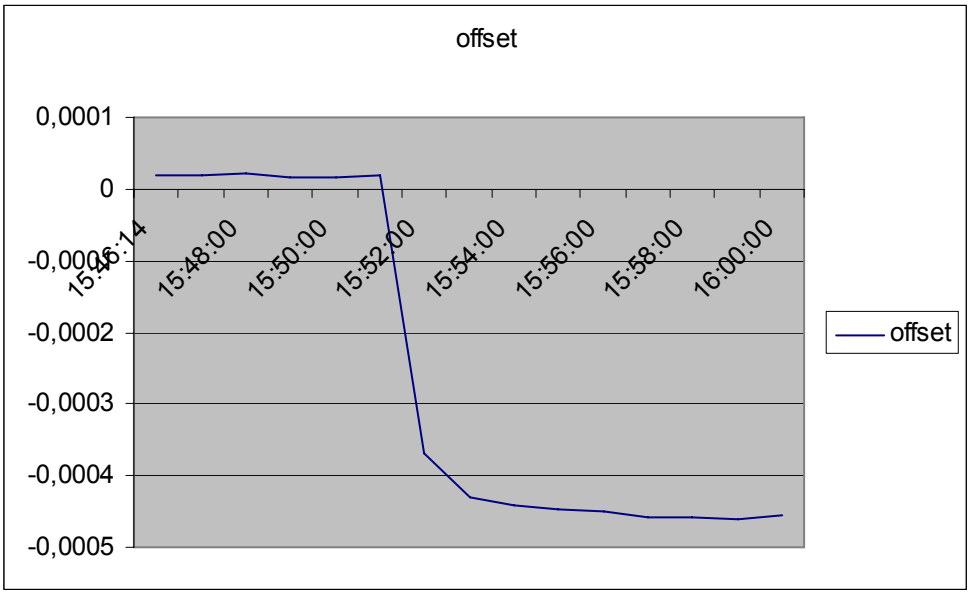


Figura 6.7: Gráfico de tiempos de corrección de offset

En la figura 6.8 se observa cómo disminuye la distancia de sincronización al momento de la sincronización, en la hora 15:52. También se aprecia el valor relativo con el valor de offset.

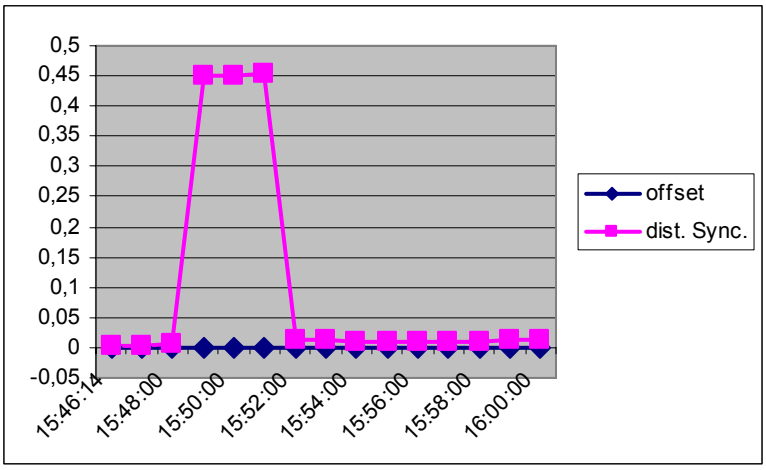


Figura 6.8: Offset y distancia de sincronización

Con respecto a la ***sobrecarga en comunicaciones***, se puede configurar la frecuencia a la que se interrogará al servidor. En caso de querer eliminar toda intrusión de NTP en las medidas de rendimiento que se deban realizar, sería necesario detener los procesos de NTP, con lo que los valores de error aumentarán, y lo harán en forma imprevisible, pues el reloj empezará a tomar como referencia el reloj local, con las correcciones hechas por NTP en el período en que estuvo funcionando. De otra manera, habría que admitir el "ruido" de los paquetes de sincronización, normalmente en sucesiones (o *ráfagas*) de 4 paquetes UDP, que también implican el costo de procesamiento por parte de los de NTP, con las posibles llamadas al sistema para ajustar los valores de la hora.

Otro inconveniente es que la consulta y actualización del reloj se realizan a través de llamadas al sistema operativo, lo cual trae aparejado una ***sobrecarga en procesamiento*** debido al los cambios de contexto requeridos. Esto se explicó con mayor nivel de detalle y cuantificación en el capítulo 3.

6.4.2 Sincronización de Varios Clientes: Error a Través del Tiempo

El valor que proporciona NTP para estimar la magnitud del error de sincronización en un instante dado, es la distancia de sincronización λ (Dist.Sync. en la tabla 6.2). Este parámetro se calcula en base a una estadística en los tiempos de comunicación con el servidor de referencia, la distancia de sincronización con que el servidor a su vez recibe los valores de referencia, y los valores registrados de *offset de frecuencia*. También figura en la tabla los valores de *offset de hora* θ debidos a la demora en las comunicaciones. Los detalles del cálculo de la distancia de sincronización y del offset de hora están en el capítulo 4, sección 4.5.4 sobre el final de la sección. Las variaciones de frecuencia (o posible derivada segunda de la hora) se registran e incorporan al cálculo de la distancia de sincronización. El seudo código del cálculo de la distancia de sincronización puede verse en el RFC-1305 [37].

La dificultad de contar con la distancia de sincronización como estimación de error, consiste en que no se puede saber con certeza el error de sincronización sino un valor relacionado con las cotas, que normalmente es del orden de los milisegundos. En la

tabla 6.2 se puede verificar que se sincroniza a valores de offset de alrededor de 0,000369 segundos con una distancia de sincronización de 0,01141 segundos.

La distancia de sincronización λ y el offset θ (explicados en el capítulo 4) dan una idea del error en que se sincronizaron los relojes. El comando nptime proporciona otro indicador. Con el fin de obtener valores de este indicador, se realizó la prueba de sincronizar con un servidor un cluster de 16 máquinas. El servidor está sincronizado en forma externa con varios de los servidores de un pool. Para evaluar el error alcanzado en la sincronización, se utilizó el comando ntptime, que realiza una llamada a ntp_gettime(). El experimento se realizó durante 2400 minutos. En la tabla 6.4 se muestran los valores máximos y estimados de error, que se obtuvieron a partir de dicho comando en el experimento. Como los errores se miden a partir de datos estadísticos, aparecen estos dos tipos de error, el estimado y el máximo.

	Maximun	Estimated
Min	239673	31
Max	989368	2434

Tabla 6.4: Valores de error devueltos por ntp_gettime en microsegundos

En el gráfico de la figura 6.10 puede apreciarse la variabilidad del error estimado, entre 31 y 2500 microsegundos. En la figura 6.11 se observan los valores de error máximo a lo largo del mismo experimento, que también muestran una variabilidad importante, entre 200.000 y 1.000.000 de microsegundos. Aún en los casos de valores mínimos del valor estimado (31 us), se trata de decenas de microsegundos, que son tiempos comparables a los de demora de un paquete en dicha red, demora que en los diferentes experimentos realizados en esta tesis fue del orden de 54 microsegundos. También se aprecia que el error no disminuye con el tiempo, ya que sigue variando de igual manera que al principio entre los valores mínimos y máximos. En consecuencia, el uso de NTP para sincronizar hora en forma interna, con fines de aplicar instrumentación para mejora de rendimiento, queda descartado, ya que el probable error es del tamaño de lo que se pretende medir.

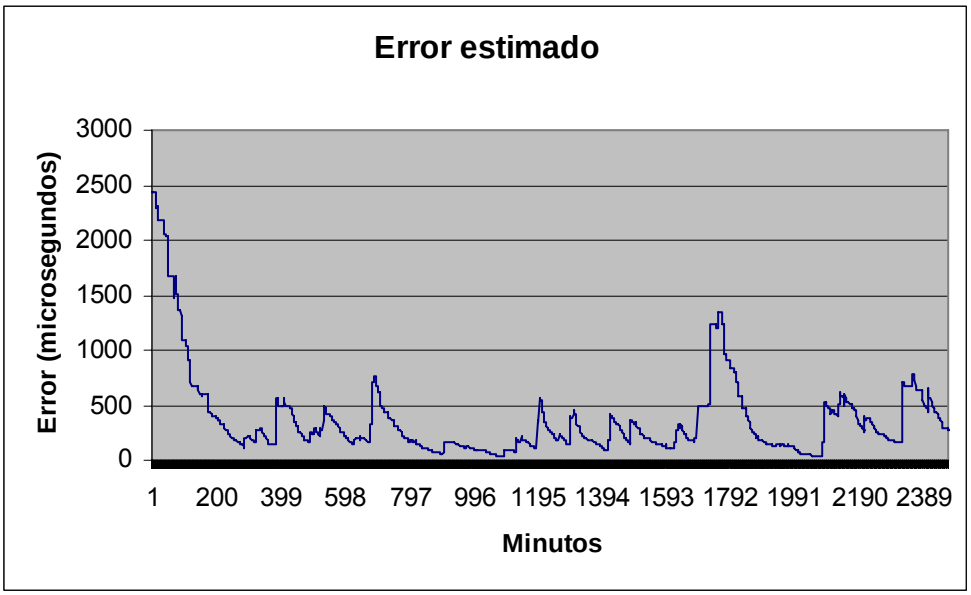


Figura 6.10: Gráfico de pruebas de valores de error valor estimado

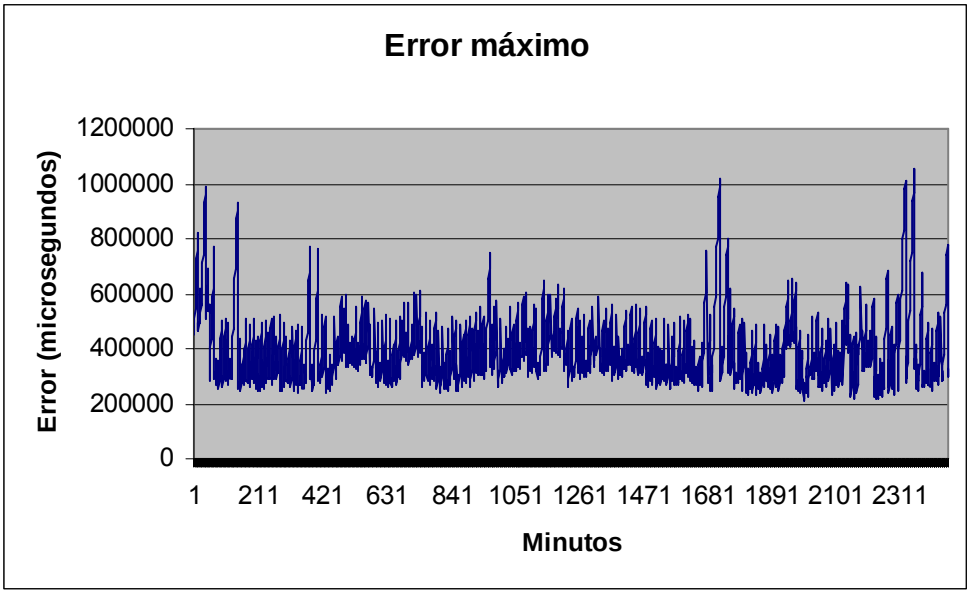


Figura 6.11: Gráfico de pruebas de valores de error valor máximo

6.4.3 Precisión y Resolución Posible del Reloj Sincronizado

En NTP la precisión es determinada en forma automática y medida como potencia de 2. Podemos verificarla con el comando ntpq -c rl, si nos da precisión = -16, la precisión es aproximadamente 15 microsegundos (2^{-16} s). Para pruebas realizadas en una máquina de las especificadas en la tabla 6.3 da -17, o sea:

2e-17 = 7,6 us.

Una de las primeras consecuencias de utilizar NTP está relacionada con la precisión del reloj obtenido, independientemente de la sincronización. Si bien la precisión del reloj de NTP es de 2e-17s = 7,6 μs., en estas mismas computadoras, la resolución del reloj sin NTP es de 1 μs, con lo que en cierta forma se pierde algo de precisión *a priori*. Se debe tomar en cuenta que la precisión es el rango de incertidumbre de un valor medido, expresado como una desviación estándar o un múltiplo de ella [43].

Es importante observar que para utilizar los servidores de estrato 1 y 2, actualmente se necesita solicitar que levanten las restricciones del firewall, por lo que hay que contar con dicho permiso.

Capítulo 7: Descripción y Pruebas de la Biblioteca Desarrollada

Resumen

En el presente capítulo se describen los requerimientos que llevaron a implementar una biblioteca de sincronización, y también, los detalles de cada una de las funciones de la misma.

7.1 Introducción

Una vez presentados los distintos aspectos del problema de sincronizar computadoras en hora, se deben analizar las peculiaridades del enfoque de la presente tesis, partiendo de un subconjunto de los requerimientos generales de sincronización, para lograr una solución acorde a los objetivos planteados. Esto nos lleva a implementar una biblioteca que los cumpla. A fin de validar el cumplimiento de los requerimientos se realizaron diversos experimentos.

7.2 Análisis de Requerimientos

El problema de la sincronización de relojes en computadoras es sumamente complejo y presenta los siguientes tópicos:

- Resolución
- Precisión
- Exactitud
- Fiabilidad
- Tolerancia a fallas
- Interoperabilidad
- Sobrecarga del sistema
- Seguridad

La tarea de sincronizar relojes implica un compromiso entre estas variables, lo cual lleva a que por optimizar un aspecto de la solución, se degrade otro. Por ejemplo, en función de mejorar la seguridad se recurre a encriptar los paquetes de comunicación de referencias horarias, lo cual degrada el aspecto de la sobrecarga, tanto de comunicaciones como el procesamiento de los mismos. Por ello, se plantea la sincronización de relojes pero con restricciones en los aspectos a resolver.

Básicamente, se desea contar con una herramienta de instrumentación para programas paralelos que:

- Pueda ser usada inicialmente en un cluster de PC's, con la posibilidad de ser extendido a clusters en general y luego en plataformas distribuidas aún más generales.
- Sea de alta resolución, es decir, que se pueda utilizar para medir tiempos cortos, del orden de microsegundos.
- No altere el funcionamiento de la aplicación bajo prueba, o que la alteración sea mínima y conocida por la aplicación.
- Utilice en forma predecible la red de interconexión. Más específicamente, se puedan determinar, desde la aplicación, los intervalos de tiempo en los cuales se utilizará la red. De esta forma, se puede *desacoplar* el uso de la red de interconexión, ya que habrá intervalos de tiempo usados para la sincronización e intervalos de tiempo utilizados para la ejecución de programas paralelos.

En el capítulo 3 se analizaron los requerimientos que debe cumplir el registro de tiempos. Con ellos se desarrolló la biblioteca *timings*, la cual provee la resolución adecuada, con el menor uso de recursos, sin hacer uso de la entrada/salida ni de llamadas al sistema, y con el menor uso posible de ciclos de CPU.

Queda el problema de los requerimientos de sincronización. En el caso de procesamiento distribuido, con un programa que ejecuta procesos en diferentes computadoras o en los que el tiempo de las comunicaciones es importante, la tarea de medir intervalos de tiempo implica sincronizar los relojes de las diferentes computadoras que se utilizan [14][37][38]. Sería deseable que esta tarea de sincronización:

- Se lleve a cabo fuera del tiempo en que se ejecuta el programa que se está monitorizando,
- Se conozca el tipo (o al menos magnitud) de error con que se sincroniza.
- No haya necesidad de incluir hardware adicional al del sistema. Esto implica que todo lo referente a las comunicaciones deberá utilizar la red de interconexión entre computadoras que ya existe y el sistema de medición existente en cada sistema de cómputo.

7.3 Biblioteca de Sincronización

La biblioteca de sincronización debe cumplir como funcionalidad principal, dos aspectos:

- Proporcionar en todos los relojes de las computadoras sincronizadas un mismo valor inicial de hora.
- Igualar la frecuencia de actualización de dichos relojes, lo que implica que presenten igual duración las unidades de un reloj con las de los demás (o sea que el "segundo" de una máquina dure lo mismo que el de la otra).

Se utilizará el modelo cliente-servidor, con lo cual habrá un servidor que tendrá por funciones:

- Proporcionar las referencias de hora necesarias.
- Concentrar la información de errores para reportes posteriores.

En función de los requerimientos vistos, se decide desarrollar la biblioteca de sincronización partiendo de modificar el algoritmo de Cristian. Recapitulando, el algoritmo de Cristian presenta los siguientes pasos:

- El cliente envía un requerimiento de sincronización al servidor, tomando nota de la hora local de envío.
- El servidor contesta con un mensaje que contenga su hora local.
- El servidor evalúa el tiempo de ida y vuelta (rtt: *round trip time*) del mensaje al recibir la contestación desde el cliente, toma una segunda medición de su reloj local y resta la primera lectura realizada al envío de la referencia. Si el valor de rtt es igual a la moda, calculada en una etapa del proceso anterior, se toma como hora correcta la del servidor más la mitad del rtt y se le envía al cliente un mensaje de parada.

En base a las pruebas del capítulo 5, sección 5, se decide utilizar el protocolo TCP para el desarrollo de la biblioteca de sincronización. Si bien TCP supone un gasto mayor en comunicaciones, se debe tener en cuenta que el algoritmo de Cristian requiere un intercambio fiable de paquetes. No se pueden perder paquetes ni llegar mensajes duplicados sin perder precisión en la sincronización y sin que haya un aumento en el

error de la misma. Los experimentos de la sección 5.5 del capítulo 5 dan cuenta de la sobrecarga al usar los diferentes tipos de paquetes y protocolos de comunicación. Además, el intercambio de paquetes con referencia, se lleva a cabo fuera del tiempo en el cual se ejecuta la aplicación a instrumentar, con lo cual la sobrecarga de comunicaciones no afecta el error del experimento. Por otro lado, la mayor sobrecarga en TCP se da cuando hay reenvíos, que no es el caso en una red local como las usadas en los clusters.

Todo lo relativo a intercambio de referencias de tiempo, se realiza en un *loop* que da por válidas las referencias que se entregaron en un tiempo de transmisión igual a la moda, previamente calculada estadísticamente. Caso contrario, se vuelve a transmitir la referencia. Este recaudo tiene por fin que el cálculo de error sea lo más riguroso posible. Uno de los mayores problemas de error de sincronización tiene relación directa con las demoras producidas por las diferencias entre envío y recepción de los procesos. Estas demoras se producen por razones como: desalojo de páginas de *cache*, tiempos de acceso a los *buffers* del sistema operativo, disponibilidad de otros recursos como CPU y los utilizados por la pila de protocolos.

Una vez resuelto el problema del *offset inicial*, se deben tener en cuenta las diferencias de frecuencias con las cuales se actualiza la hora en cada computadora. Los sistemas operativos como Linux usualmente tienen calculado un valor de frecuencia del oscilador El sistema operativo utiliza este valor para transformar los valores de TSC en unidades de tiempo a fin de interpolar los valores proporcionados por el timer. Este valor, generalmente dado en MHz (millones de ciclos por segundo), no es necesariamente correcto para el registro de la hora con precisión de microsegundos. Por esta razón, en la biblioteca *timings*, se rehace el cálculo de MHz del oscilador de cada computadora, de manera tal que el cálculo de la hora posterior utilizando este valor tenga precisión del orden de microsegundos o mejor. Como se aclaró en el caso de NTP, tener un reloj local (o los relojes locales de todas las computadoras) con precisión del orden de microsegundos no necesariamente implica que el error de sincronización sea de esta magnitud.

Para compensar el error de frecuencia debido a las diferentes frecuencias de los relojes, se obtiene una constante de calibración en el arranque. Dicha constante se calcula en base a los MHz de los relojes a calibrar. Obtenida esta calibración, dicho valor puede cambiar por la deriva o *drift* del valor de frecuencia del oscilador debido a la temperatura, la tensión de alimentación u otros factores. Dicho cambio no es lineal. Es decir, tiene derivada (incremento de frec)/(inc de tiempo o por unidad de tiempo) distinta de cero. También podría calcularse y corregirse este error en caso de que la derivada sea constante, pero no es el caso: los experimentos muestran la existencia de una segunda derivada.

El cálculo de MHz reales de una computadora es relativamente sencillo a partir de la lectura de los valores de TSC y tomando como referencia los valores del reloj de *hardware* (*timer*) del sistema. Aunque la precisión del reloj del sistema no sea muy apropiada para el registro de tiempo del orden de microsegundos, la frecuencia del mismo es suficientemente estable como para ser considerada constante en un intervalo de tiempo suficiente como para el cómputo inicial de MHz. Los sistemas operativos suelen proveer llamadas al sistema que mejoran la resolución interpolando los valores del *timer* con los de TSC, como el caso de la llamada *gettimeofday()* de Linux. En este

caso, si bien se aumenta la resolución, estará presente un error debido a la falta de calibración del valor de MHz. Recuérdese que este valor se calcula en el arranque de la máquina por parte del SO, y los experimentos llevados a cabo demuestran que una vez que la máquina entra en régimen, dicho valor debería recalcularse [8].

Asumiendo que el oscilador al que se accede con RDTSC es de frecuencia constante, el cálculo de MHz puede ser realizado en un único punto en el tiempo, con lo cual este tiempo inicial puede ser relativamente grande (segundos o quizás minutos). Sin embargo, el cálculo no necesariamente es tan sencillo en diferentes computadoras que luego se deben sincronizar. La razón fundamental está relacionada con el reloj de hardware de referencia con el cual se realiza el cálculo de MHz de cada computadora, que puede presentar diferencias entre una y otra.

La alternativa es, justamente, que una sola de las computadoras, usualmente la que funciona como servidor de hora [45], proporcione las referencias de hora con las cuales se hace el cálculo de MHz en cada una de las demás computadoras. Este esquema es sencillo y también se pueden evitar errores haciendo mayor el tiempo durante el cual se contabilizan las oscilaciones en base a las cuales se lleva a cabo el cálculo de MHz. En cierta forma, tanto las referencias de tiempo del reloj de hardware de una PC como las referencias de tiempo que se reciben desde otra computadora tienen un margen de error, que se puede disminuir proporcionalmente con el propio tiempo que se contabiliza. En este caso, también se utiliza la red de interconexión y se toman los tiempos de mensajes como significativos para el control del error en el cálculo de MHz.. Más específicamente:

- En las computadoras utilizadas de 2.4 GHz se tienen aproximadamente 2.4×10^9 oscilaciones por segundo.
- Un paquete TCP utiliza aproximadamente 120 μs como tiempo de ida y vuelta, con lo cual se puede asumir que se necesitan aproximadamente 60 μs para un envío-recepción de un paquete TCP individual (o *one-way transmission time*).
- Durante un tiempo de 10 segundos (10^7 μs) en el servidor, el tiempo que se contabiliza en los clientes a partir de los mensajes recibidos podría tener el error de tiempo de transmisión dos mensajes, es decir 120 μs, con lo cual el posible error está acotado por:

$$120/10^7$$

Es decir que el error en la contabilización de oscilaciones es de $120/10^7$, o de otra manera, si k es la cantidad de oscilaciones contabilizadas, el valor real puede variar en:

$$\pm120/10^7\times k.$$

Es de acotar que los 10 segundos mencionados pueden ampliarse en caso de requerirse mayor precisión.

En el caso de la biblioteca desarrollada, llamada *st*, se provee la siguiente funcionalidad (dada, en cierta forma, en el orden en que pueden/deben ser utilizadas las funciones):

- Inicio (Init): Es la primera función que se ejecuta. Crea las conexiones TCPs y determina el valor de MHz de cada máquina a fin de lograr minimizar el offset de frecuencia entre los relojes de las diferentes máquinas que intervienen en la sincronización. Como la frecuencia de los relojes utilizados está dada por la frecuencia del reloj de la CPU, el TSC, se debe medir la frecuencia, por lo

general, en MHz (Megahertz). Tanto para esta función como para determinar el valor inicial de la hora, se deben intercambiar mensajes entre las computadoras. Esto se realiza utilizando paquetes TCP, por lo que deberán abrirse los consiguientes sockets. Así, la biblioteca comienza su funcionamiento con este módulo de inicialización en todas las máquinas, en el cual se crean los sockets necesarios, se inicializa el valor de MHz medido con referencia al reloj local o al de una única máquina (referencia única), dejando todo preparado para el proceso de sincronización. El valor de MHz se utiliza además para normalizar a microsegundos u otra unidad de tiempo los ciclos de máquina que se obtienen a partir de la lectura del TSC. También se realiza una estadística de tiempos de intercambio de mensajes con cada cliente, para determinar el valor de la moda en la transmisión de mensajes.

Los valores de hora en dos computadoras [44][45][46] pueden diferir a partir de un instante de tiempo, por un cierto error inicial (*offset*), que irá cambiando en el tiempo dado por las diferencias en sus frecuencias (primera derivada) más un error aleatorio dado por la deriva en el valor de frecuencia de cada una de las frecuencias, producto de cambios ambientales [33] [32]. Al menos la diferencia de dos relojes en un instante de tiempo puede ser estimada en una computadora a partir conocer el tiempo de transmisión entre dos máquinas y el valor de dichos relojes. Si dicho tiempo de transmisión fuese constante, se podría medir y eliminar este error utilizando la constante necesaria. Como ambos relojes no están aún sincronizados, el tiempo debe ser medido por un solo reloj. Por lo tanto, se mide el tiempo de ida y vuelta de un paquete en la red de comunicaciones (*round trip time*). Se estima el tiempo de transmisión como la mitad del tiempo de ida y vuelta, lo cual no necesariamente es correcto. A partir de la descomposición de este tiempo [39] y para acotar lo más posible el error, se decide medir el tiempo que tarda el SO en colocar un mensaje en los buffers, listo para ser transmitido. Dicha medición se realiza en forma local en todas las computadoras involucradas. En todos los casos, se miden los tiempos de ida y vuelta, se estiman simétricos y se obtiene el tiempo dividiéndolo por dos. Si bien esta estimación puede no ser del todo cierta, el error cometido sería pequeño, dado que la tarea llevada a cabo es simétrica y las diferencias de ida y vuelta son causadas por planificación de CPU, interrupciones de hardware y demás interferencias producidas por el propio sistema operativo. Se podrían obtener mejoras al respecto utilizando un Sistema Operativo de Tiempo Real, lo cual sería poco práctico en un cluster dedicado a correr aplicaciones paralelas. De todos modos, no se descarta implementar solo el servidor de hora con un Sistema Operativo de Tiempo Real.

- Sincronización (Synchro): Determina los “deltas” o diferencias de hora (offset de hora) y este valor queda almacenado en la memoria de los clientes a fin de poder utilizar la función que da la hora sincronizada. Los clientes solicitarán sincronizarse al servidor, proporcionando, en caso de requerirlo el servidor, su valor de hora para realizar las comprobaciones de error. En el proceso de sincronización, se suceden los siguientes pasos:
    - los clientes solicitan al máster una referencia,
    - éste se la envía en base a su propio reloj local,
    - el cliente confirma la recepción y guarda el valor de diferencia (delta de hora).

- En el momento de arribo del mensaje de recepción al servidor, éste verifica que la mitad del tiempo de ida y vuelta sea el de la moda que se midió en la estadística previa.
- En caso positivo, el cliente queda sincronizado y da por válido el delta calculado. De no ser así, se repite el envío de referencia, la medición del tiempo de ida y vuelta del mensaje y el cálculo de delta.

Se observa que en esta función se implementa el Algoritmo de Cristian pero con una modificación. Si bien esta modificación supone un intercambio de dos mensajes más, se logra centralizar en el servidor la información relativa tanto a los resultados de la sincronización como al error con el que se logró. De esta forma, el servidor puede controlar tanto el proceso de sincronización como los errores.

Como vimos en capítulos anteriores, para sincronizar dos relojes debemos hacerlo en un valor inicial, con el menor error posible. Dicho error tendrá origen principalmente, además de en otros factores, en el tiempo de transmisión de la referencia de dicho valor inicial. Para ello, se realizaron experimentos tendientes a medir respecto del tiempo de transmisión, la moda y los valores mínimo y máximo, en un ambiente como el que se especifica en los requerimientos. Para acotar aun más la parte variable, se midió el tiempo mínimo que el sistema tarda en acceder a los buffers de comunicación.

A partir de los experimentos llevados a cabo, se resuelve que las trasmisiones que se lleven a cabo en las funciones de la biblioteca de sincronización se tomen como válidas sólo si fueron realizadas en un tiempo rtt (round-trip time, tiempo de ida y vuelta de un paquete) igual a la moda. Con ello se tiende a acotar los errores debido a las comunicaciones tanto en la determinación del offset inicial entre relojes como de la referencia desde el servidor a fin de calcular las diferentes frecuencias (MHz).

Un problema que se presentó es el de la asimetría entre los tiempos de ida y vuelta. Si se toman como válidas sólo las referencias de tiempo trasmitidas en tiempo rtt igual al valor de la moda, es muy probable que dicha asimetría pueda estar dada por la diferencia entre la moda y el valor mínimo. Para realizar mediciones que demuestren esto, se debería contar con un reloj sincronizado, lo cual implica que dichas mediciones tendrían error, ya que se sincroniza con dicho error.

Actualmente, se está estudiando si es posible estimar este error nulo, que surge como resultado de asumir que los tiempos de ida y vuelta de un mensaje son iguales.

- Hora local sincronizada (Ulocaltime): se forma en base a una lectura del reloj TSC local, valor al que se le suma el delta determinado por la sincronización.
- Testeo de error en la hora (Test): proporciona la diferencia de horas entre los relojes de las diferentes máquinas. Se complementa con la información proporcionada por la función que determina la variabilidad de los tiempos de envío-recepción de mensajes. Se puede utilizar en alguna parte o al final del experimento para corroborar el grado de sincronización entre relojes y realizar

las correcciones pertinentes en los valores adquiridos de hora durante el mismo experimento.

- Finalización de utilización de las funciones de la biblioteca (Close): se eliminan las conexiones TCP y asignaciones de memoria al final de la prueba de instrumentación.

El conjunto de las funciones de la biblioteca ST se puede apreciar en el detalle de st.h dado en la Figura 7.1, donde se muestran las funciones que se corresponden con la descripción de funcionalidad dada anteriormente. A continuación realizamos un breve repaso de la utilidad de cada función:

- ***st_init_ip():*** es la función encargada de crear todas las conexiones TCP, inicializando los sockets correspondientes. Deja todo listo para realizar la sincronización. Se ejecuta sólo una vez al comienzo del experimento de instrumentación completo.
- ***st_synchro():*** tiene como tarea intercambiar las referencias de hora necesarias entre el servidor y los clientes a fin de garantizar una hora inicial uniforme en todas las máquinas. Se llama al menos una vez al comienzo del experimento, salvo que se quiera resincronizar porque el tiempo de ejecución de la aplicación bajo prueba lo justifique.
- ***st_usynclocal_time():*** proporciona una referencia de la hora sincronizada. Será la verdadera instrumentación del código a medir. Está implementada teniendo en cuenta que la intrusión sea mínima para una alteración mínima de los valores de tiempos de ejecución y comunicación que se quieran medir, excluyendo las llamadas al sistema.
- ***st_test():*** proporciona las diferencias de hora entre las diferentes computadoras respecto del "tiempo uniforme esperado". Se lo llama esperado, ya que cada computadora tendrá una pequeña diferencia respecto de la hora global sincronizada. Proporciona las correcciones que debieran hacerse en las respectivas referencias obtenidas a través de ***st_usynclocal_time()***.
- ***st_close():*** se utiliza al final del experimento con la utilización de la biblioteca de sincronización. Básicamente cierra todas las conexiones TCP abiertas durante el ***st_init_ip()***.
- ***st_equal():*** envía a todos los clientes una referencia de tiempo a partir de la cual cada uno guarda el valor de su TSC. A partir de allí, se produce una demora durante la cual el TSC se irá incrementando. Cabe aclarar que cuanto mayor es esta demora, menor será el error con el que se calculará la variable MHz utilizada por ***st_usynclocal_time()*** para transformar los valores de ciclos proporcionados por la llamada a la instrucción de assembler RDTS, a microsegundos. Recibe como parámetro una cantidad de segundos, que servirán para la demora de separación entre referencias.
- ***st_close():*** Cierra todos los sockets creados en init.

```
/*  st.h: Synchronized time library header file                */
/* This library is expected to be used along with the timings library */
/* Since the timing library is for local timing measurements and this */
/* one is the corresponding one for a cluster, i.e. maintains a       */
/* synchronized, "uniform", global and measurable time along the      */
/* computers in a cluster (LAN)                                       */

  /* Structure for reporting statistics and some value/s for error    */
  /* computing/guessing/bounding, every measure is half rtt in fact   */
  typedef struct
  {
… VALORES ESTADISTICOS
  } rttimes;

  /* Init client/server connections and local timing computing (no sync) */
  /* (equal == 0)  ==>  MHz with local RDTSC                          */
  /* (equal == 1)  ==>  MHz with remote/server RDTSC via TCP sockets     */
  /* sleepsecs: number of seconds to sleep for MHz computing...          */
  int st_init_ip(char *hostips[], int n, int equal, int sleepsecs);

  /* Close client/server connections */
  void st_close();

  /* Init synchronization, should be as close to instr. time as possible */
  long long int st_synchro();

  /* Return synchronized local_time in micro seconds */
  long long int st_usynclocal_time();

  /* Set the synchronized timing difference since synchro() call */
  void st_test(long long int errs[]);

  /* Return the filled in rtt array */
  /* m: number of synchronized clients. m = n-1 */
  void st_get_rtt(rttimes rep[], int m);
```

Figura 7.1: st.h, archive header de la biblioteca *st*.

Para un entendimiento más acabado de cómo realizan las funciones su labor, a continuación está un seudocódigo de las mismas:

st_init_ip():
En el comienzo de la ejecución de la biblioteca de sincronización se debe disponer de un archivo con las direcciones IP de las máquinas que se desean sincronizar. Dicho

archivo será leído por el módulo de inicio. El conjunto de tareas que realiza este módulo se puede apreciar en el siguiente seudo código:

```
Proceso INICIO
        Obtener IP
        Obtener socket TCP
        Iniciar biblioteca tiempos timing (necesario para medir tiempo en forma local)
        Realizar Estadística para medir moda, valor mínimo y máximo
        If (se desea emplear referencia única)
                Calcular MHz con referencia única
        Else
                Calcular MHz con referencia local
        Reinicializar bibliotecas de tiempo timing
        Medir tiempo acceso a buffers
```

A partir de haberse concluido el inicio, puede utilizarse toda la funcionalidad de la biblioteca. La siguiente función a ejecutarse será:

st_synchro():
En caso de necesitar sincronizar, la función ejecuta lo siguiente:

```
Proceso SINCRONIZAR
        /* servidor */
        Para cada cliente
                Recibir petición
                Mientras (tiempo ida y vuelta/2) distinto de tiempo.moda con ese cliente
                        Enviar hora local + tiempo.moda con ese cliente
                        Recibir respuesta de cliente
                        Calcular (tiempo de ida y vuelta/2)
                Enviar confirmación a cliente sincronizado
        /*cliente*/
        Enviar pedido de sincronizar
        Mientras (no recibe confirmación de sincronizado)
                Recibir tiempo del servidor
                Calcular constante de corrección del tiempo local
                Enviar tiempo corregido
                Recibir confirmación
```

st_test():
Esta función sirve para evaluar la diferencia de hora entre los clientes y el servidor, lo cual es útil tanto para saber el grado de sincronización obtenido si se ejecuta a continuación de *st_synchro()*, como para apreciar el grado de sincronización al final de los experimentos con instrumentación. En la página siguiente podemos ver el seudo código de esta función:

Proceso TEST

/ servidor */*

Para cada cliente

Recibir petición

Mientras (tiempo ida y vuelta/2) distinto de tiempo.moda con ese cliente

Enviar hora local + tiempo.moda con ese cliente

Recibir y guardar error de cliente

Calcular (tiempo de ida y vuelta/2)

Enviar confirmación a cliente testeado

*/*cliente*/*

Enviar pedido de test

Mientras (no recibe confirmación testeado)

Recibir tiempo del servidor

Calcular error con tiempo local

Enviar error

Recibir confirmación

st_usynclocal_time():

Es la función con la cual se consulta el reloj sincronizado local. Como se puede ver, no se utilizan llamadas al sistema, siendo especialmente cuidado el aspecto de baja intrusión en su diseño.

*/*adquisición de hora local sincronizada*/*

{

Leer TSC

Pasar a microsegundos

Sumar Delta calculado en synchro

}

st_equal():

Calculo de MHz a partir de referencias de tiempo proporcionadas por el servidor a fin de equilibrar los valores de actualización de la hora, o sea, reducir el offset de frecuencia.

{

Cliente pide ecualizar

Server proporciona primera referencia, con control de rtt= moda, sinó repite

Cliente toma valor TSC local

Server proporciona segunda referencia, con control de rtt= moda, sinó repite

Cliente toma valor TSC local

Calculo MHz=diferencia de TSC/diferencia de referencias

}

st_close():

En caso de concluir la prueba de medición de tiempos, el *Proceso CLOSE* cerrará los puertos abiertos en el *Proceso INICIO*

{

Cerrar Sockets

Liberar memoria

}

Además de las funciones antedichas, dentro de la biblioteca *st* están las siguientes funciones, que no son accesibles, y son utilizadas desde adentro de las funciones anteriores:

autoping():
Devuelve el tiempo que tarda en llegar un mensaje a los buffers de entrada/salida. Trabaja en conjunto con el programa **serverping()**, que proporciona la respuesta del mensaje enviado

{
Armar comando para ejecutar serverping()
Ejecutar comando en segundo plano
Ejecutar n veces
    {
    Tomar referencia de hora local
    Enviar mensaje a serverping() (ping)
    Recibir mensaje de serverping() (pong)
    Tomar segunda referencia de hora local
    }
Guardar timepo ida y vuelta (ping-pong)
Cerrar serverping()
}

st_init_rtt(char *localip):
Encargada de realizar una estadística de una cantidad n de tiempos de envío de mensajes, proporciona los correspondientes valores de la moda, valor máximo, valor mínimo, con los cuales se calcularán los errores con que se adquieren los valores de tiempo proporcionados por ***st_usynclocal_time()***

{
Servidor:
n veces hacer
    {
    Enviar n mensajes a clientes, midiendo hora de envio local
    Recibir respuesta de cliente
    Segunda medicion de hora local
    Calcular rtt
    Contabilizar valor con frecuencia de repetición (histograma de tiempos rtt)
    }
Llenar el arreglo rtt para todos los clientes para n mensajes ida y vuelta
Cliente:
    {
    Responder mensaje a servidor hasta que el servidor envía un 0
    }
}.

7.3 Detalles de la Implementación de Cada Función

La lectura de esta sección es para aquellos que requieran documentación a fin de aclarar los detalles de más bajo nivel en la implementación de las funciones.

```
st_init_ip(...)
{
   Establecer la cantidad n de máquinas a sincronizar
   Leer dirección IP local y de broadcast a través de st_ip_get()
   Inicializar la capa de sockets a través de get_st_tcp_socket()
   Realizar las conexiones TCP (utilización de biblioteca sockets)
   Determinar la identificación de la máquina local
   Ejecución de st_init_rtt() para adquirir los valores de rtt
   If equal == 1...

Ejecución de st_equal  si el parámetro equal es 1
Sino en los clientes ejecutar timing_init(0.0, sleepsecs)(Cada
computadora con referencia de tiempo local)

   autoping():pingpongs locales. El servidor recibe los valores
}


/*Hora sincronizada local:*/
st_usynclocal_time()
{
  us  = ulocal_time()
  us  = us + delta; /* delta assignada en synchro() */
  return us;
}
st_synchro()
En el server
Aceptar cada requerimiento de cliente
Sincronizar cada cliente en un orden fijo
    for (i = 0; i < (cantidad de máquinas - 1); i++)
    {
      do {Mientras tmin es diferente de rtt[i].rttmoda
        Adquisición de hora local
        Definir hora remota como hora local mas rtt[i].rttmoda;
        Envío de hora de referencia al cliente


        Recibo de hora del cliente ajustada
        Nueva adquisición de hora local
        Calculo de tiempo de transmisión

      } mientras (rtt != rtt[i].rttmoda)

      Envío 0 para detener el loop de synchro en client
      }
  }

  En el client
  {
   Envío requerimiento de sincronización al server
   Recibo referencia de hora del server
   }

mientras (referencia != 0) (El server avisa cuando termina)
     {
     Adquisición de hora local
     Diferencia entre hora local y la referencia
     Envío de corrección de hora local
     Recibo hora de referencia de Server
     }
  }
```

```
}


st_test()
{

En el Server
Para cada cliente
    {
    Recibo requerimiento de los clientes
    Adquisición de hora local
    Hora remota es local mas rtt[i].rttmoda;
    Envío algo al cliente para medir rtt
    Recibo del cliente error de hora
    Adquisición de hora local
    tmin = (utrecv - utsend) / 2
      } Mientras(tmin != rtt[i].rttmoda)

    Envío hora 0 para parar el loop en el cliente

 En el cliente
  {
    Envío requerimiento al server de test

    do {
       Recibo de referencia de hora del server
       Si referencia != 0
        {
        Ajuste local de la adquisición de hora
        Calculo y envío de diferencia entre hora de referencia y local
        }
       } while (referencia != 0);
  }

st_equal();Equalización de relojes, retorna MHz

{
  Lectura de TSC, usado como macro para evitar retorno de llamada de
función

  En el server
    Ecualiza cada cliente en orden fijo con referencia de hora
    Recibe requerimiento del cliente
    Envío de primera referencia de hora a cliente para estimar MHz
    controlando round-trip time via este loop
      do{
         Adquisición de hora local */
         Envío de hora a cliente y medir round-trip time
         Recibe algo del cliente
         Adquisición de hora local
         delay  = (hora al recibo – hora al envío) / 2;

      } mientras (delay != rtt[i].rttmoda)

      Envío 0 al cliente, control de loop
     }

    Espero un tiempo suficiente
    Ecualizo cada cliente en orden fijo: segunda referencia de hora
          Recibo requerimiento del cliente
```

```
          Envío segunda referencia de hora a cliente para estimación
de  MHz
          Controlando round-trip time via loop
       do{
          Adquisición de hora local
          Envío al cliente y mido round-trip time
          Recibo algo del cliente
          Adquisición de hora local
          delay   = (hora recibo – hora envío)/ 2

       } mientras (delay != rtt[i].rttmoda);

       Envío 0 al cliente, control de loop
      }
  }

  En el client

  {
    Envío requerimiento de ecualización
    Recibo primera referencia de hora del servidor, controlando round-
trip time via loop
    do{
       Recibo referencia de hora del server */
       Si referencia != 0
      {
        Leer TSC local
        Calculo y envío diferencia entre referencia y hora local
        }
    } mientras(referencia != 0)


    Recibo segunda referencia de hora del server, controlando round-
trip time via loop
    Respuesta a requerimiento de ecualización
    do{
       Recibo referencia de hora del servidor
       Si referencia != 0
      {
        Leer TSC local
        Envío de diferencia entre hora local y referencia
      }
    } mientras referencia != 0

    Calculo MHz como Diferencia de TSC/diferencia de referencias
    Ejecuta timing_init(MHz, sleepsecs)
  }
}

Estimación de error en demora como Maxima cantidad de diferencias en
microsegundos del round trip time estimado 2000 veces
Tamaño máximo de array:   300

st_init_rtt()
{
  /*     server        */
  {
    Tiempo de demora con clientes en orden fijo
    Para todas las máquinas
    {
```

```
      Inicio experimento en forma individual para una cantidad MAXTIME
de veces
      for(j = 0; j < MAXTIME; j++)
      Inicio sumario de valores por cliente/
      Recibo requerimiento del cliente
      for(j = 0; j < RTTNUMBER; j++)
      {
       Recibo primera referencia de hora: hora de ping
       Envío hora de referencia al cliente
       Recibo la hora del cliente
       Nueva adquisición de hora: hora de pong
       Mido tiempo de transmission en un sentido
       como = (tiempo de recibo – tiemp ode envío) / 2
        Cuenta los diferentes rtt y contabiliza sus frecuencias    para
        cálculo de la moda
        rttfreq[delay]++
          if(frecmoda < rttfreq[delay])
          {
            moda     = delay
            frecmoda = rttfreq[delay]
          }

          if (delaymin > delay)
            delaymin = delay

          if (delaymax < delay)
            delaymax = delay
        }
        else ++outofrange
      }
      rtt[i].rttmax  = delaymax
      rtt[i].rttmin  = delaymin
      rtt[i].rttmoda = moda
      rtt[i].rttout  = (outofrange * 100) / RTTNUMBER

      Envío 0 para detener loop en el cliente

    }
  }
  Lado del cliente cliente
  {
   Envío requerimiento al servidor
   Recibo referencia de hora del server
   while (tsend != 0) /* Server dice cuando parar*/
    {
     Echo al server
     Recibo del server
     }
  }
  return 0;
}


/*“Cliente" para ping a buffers (también arranca el servidor
correspondiente serverping)*/
#define ITERS   1000       /* Cantidad de ping-pongs a medir */
autoping()
{
Línea de commando para ejecutar serverping2:
Conexión TCP con sí mismo: Hago el comando
Ejecuto server en foreground
```

```
  Medidas de envío-recibo
  Para ITERS veces
  {
   Obtengo hora local
   Envío la hora
   Recibo respuesta de serverping
   Obtengo hora local
   Calculo ida-vuelta
  }

  Detengo servidor enviando un 0
  Elimino el servidor
  Cierro conexiones TCP
}


/*Retorna el rtt array lleno desde todos lo clientes, al servidor */
st_get_rtt(...)
{
  Adquisición de valores estadísticos de tiempos de ida y vuelta
(rtt)sobre un arreglo que se devuelve
}

st_close()
{
  Libero memoria de conecciones TCP
  Cierro capa TCP
}
```

El funcionamiento de la biblioteca supondría, en todos los casos, la información de moda, los errores y demás datos, información que se concentra en el máster para permitir informar al usuario los valores medidos. Además, el servidor tiene control no sólo de los valores de tiempo medidos, sino de los errores con los que está trabajando. En el cálculo del error se incluye la varianza en los tiempos de comunicación, tiempo mínimo, máximo y moda.

Debe hacerse hincapié en que todo el proceso descrito se lleva a cabo fuera del tiempo de ejecución de la aplicación que se quiere medir. Dentro de la ejecución de la aplicación bajo prueba se utilizan sólo las adquisiciones de referencias de tiempo a través de las funcionalidades de la biblioteca *timings*, diseñadas para alterar sólo lo imprescindible los tiempos de ejecución a medir. En caso de desear una resincronización, se debe realizar en forma explícita, y siempre es posible medir el tiempo que se demora para tomarlo en cuenta en la medición que se esté llevando a cabo.

Capítulo 8: Evaluación de la Biblioteca Desarrollada

Resumen

En el presente capítulo se describen las pruebas y experimentos llevados a cabo sobre la biblioteca de sincronización a fin de comprobar el ajuste a los requerimientos y objetivos iniciales bajo los cuales se desarrolló.

8.1 Introducción

Como parte del desarrollo de esta tesis se realizaron una serie de experimentos con el fin de validar los requerimientos de la biblioteca de sincronización ya enunciados. Estos experimentos incluyen los aspectos de:

- Comunicaciones de referencias (tiempo de latencia de mensajes)
- Calibración de frecuencias de relojes
- Escalabilidad
- Sobrecarga de procesamiento y comunicaciones

8.2 Experimentos con Referencia Local o con Referencia Única

Inicialmente, se diseñaron experimentos para comprobar cómo mejora la sincronización en frecuencia al utilizar un único reloj para la calibración. Los relojes de computadoras adolecen de *offset* de frecuencia, con lo que al calibrar los MHz, este *offset* se hereda. La idea para mejorar este problema es ecualizar las referencias a partir de que sólo el servidor proporcione las mismas en base a su reloj local.

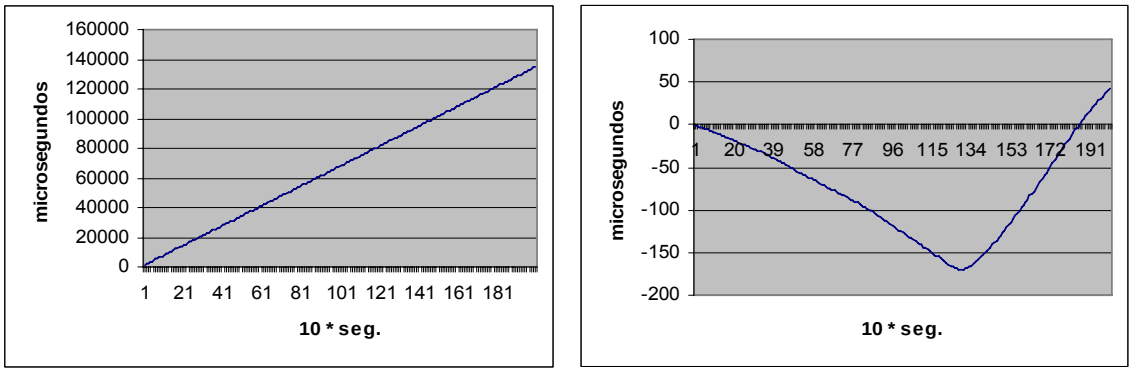
La tabla 8.1 muestra los valores que se obtienen al sincronizar relojes con cálculo de MHz totalmente independientes (con los relojes locales) y con cálculo de MHz con referencia única, es decir donde una sola computadora proporciona las referencias de tiempo para el cálculo de MHz. El experimento se llevó a cabo sincronizando desde 2 hasta 17 máquinas, y luego se midió la diferencia 1 segundo después. En todos los casos, son valores mínimos y máximos luego de 50 corridas.

Cant. Máq.	**Ref. única**		**Ref. local**	
	Mín.	Máx.	Mín.	Máx.
	Us	us	Us	us
2	-1	3	42	103
3	-3	3	56	116
4	-3	3	55	98
5	-2	3	55	114
6	-2	3	57	244
7	-3	3	56	1533
8	-3	4	56	325
9	-3	3	56	210
10	-3	4	-4	1308
11	-4	4	41	1395
12	-4	5	58	1802
13	-4	5	57	1277
14	-3	3	43	3808
15	-3	4	43	1853
16	-4	3	43	1109
17	-14	6	50	2128

Tabla 8.1: Errores de Sincronización por cálculo de MHz.

La tabla 8.1 muestra los errores mínimo y máximo de sincronización. Debe recordarse que el error de sincronización se define como la diferencia en los tiempos locales de las computadoras y estos tiempos, a su vez, tienen una resolución de aproximadamente 1.18 μs [47].

En la figura 8.2 se puede ver la diferencia de hora a partir de un ajuste inicial entre dos máquinas cuya frecuencia de reloj ha sido corregida tomando referencia local y referencia única. Se debe notar la diferencia en cuanto a los valores de las diferencias: para la figura 8.2a) varía entre 0 y poco menos de 150000 μs, y para la figura 8.2b) varía entre aproximadamente -150 y 50 μs. Obsérvese que con referencia local, figura 8.2a), la diferencia aumenta proporcionalmente a medida que transcurre el tiempo y en aproximadamente media hora sin resincronizar, se ha apartado del valor de la hora de la máquina de referencia alrededor de 0,14 segundos.

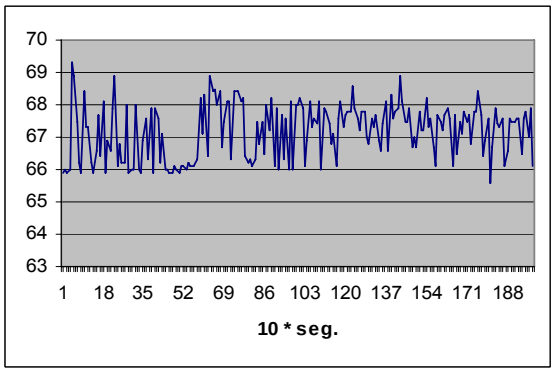


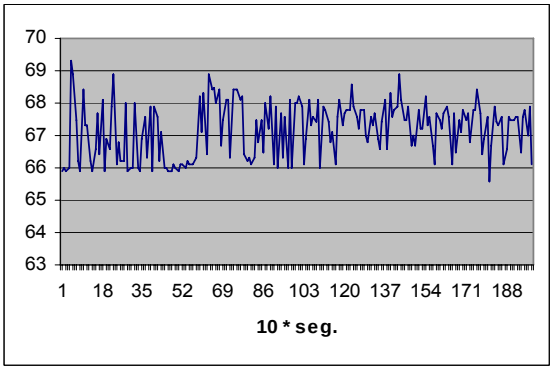
a) Referencia Local b) Referencia Unica.

Figura 8.2: Diferencias de tiempo según el cálculo de MHz.

En el caso de referencia única, figura 8.2b), al transcurrir el mismo tiempo, la diferencia máxima fue 0,000150 segundos (150 μs.). Es importante notar que en vez de aumentar constantemente, vuelve a converger al valor de la hora de referencia. Estas variaciones corresponden a variaciones tipicas de los cristales, al drift de los osciladores, identificado como ρ. Por lo general, esto ocurre debido a variaciones ambientales o a la tensión de alimentación, como podrá verificarse en la primera y segunda derivada. Como no se mide con un reloj calibrado, lo que se miden son diferencias entre relojes de dos computadoras que, en ambos casos, se ven afectados de drift.

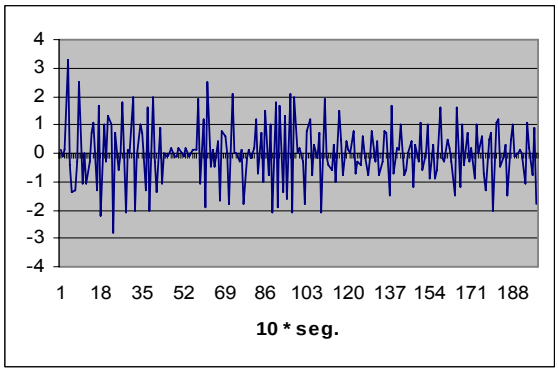
La figura 8.3 muestra los valores relacionados con la primera derivada de los valores de hora. En el caso de referencia local, figura 8.3a), este valor es relativamente grande y siempre positivo, lo cual implica el constante aumento de las diferencias de tiempo. En el caso de referencia única, el cambio en el signo hace converger las diferencias de hora hacia valores menores. Como dato adicional, se grafican en la figura 8.4 los valores de la segunda derivada, donde se observa que tanto para referencia local como única son similares (0 a 0,3). Esta derivada es la que muestra los cambios debidos a factores ambientales, que normalmente son difíciles de controlar más alla de recomendaciones como tener las máquinas en el mismo cuarto o en condiciones climáticas similares, con las medidas de refrigeración acordes al hardware, etc.

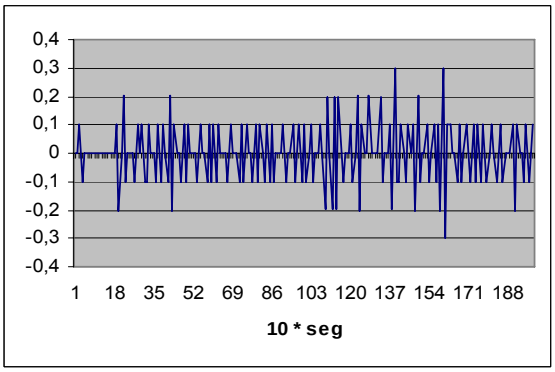




a) Referencia Local b) Referencia Única.

Figura 8.3: Primera Derivada de las diferencias de tiempo según el cálculo de MHz.



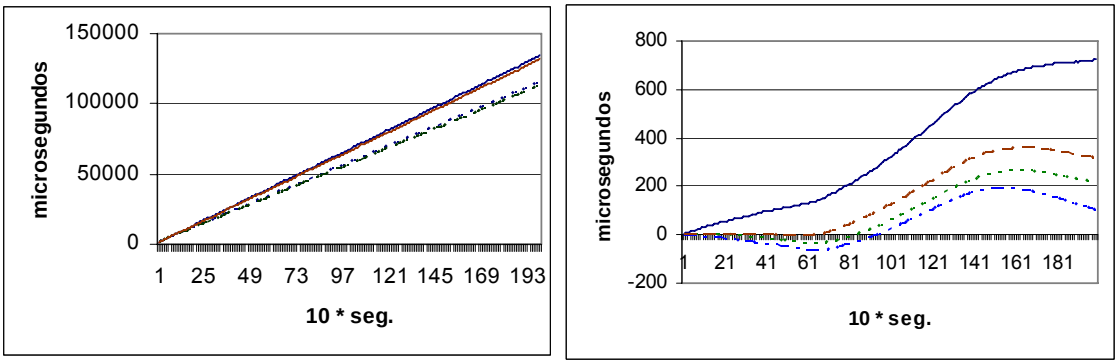


a) Referencia Local b) Referencia Única.

Figura 8.4: Segunda Derivada de las diferencias de tiempo según el cálculo de MHz.

En todos los experimentos se tienen en cuenta las diferencias entre relojes, ya que para ver solamente las diferencias del cliente, el servidor debería proporcionar un reloj perfecto. Una posible mejora sería disponer de una máquina con sistema operativo de tiempo real para suministrar la referencia de tiempo. Otra forma adicional de mejorar los valores a medir sería tener un reloj de mejor referencia. Como la idea es no recurrir a ningun tipo de hardware adicional, una de las posibilidades es armar un ensamble de relojes. Se demuestra que un arreglo de relojes puede dar una hora más estable [60][9] y también puede tomar la idea del algoritmo de Berkeley [11].

En la figura 8.5 se pueden ver las diferencias de hora de 4 clientes con respecto al servidor. En el caso de referencia local, figura 8.5a), luego de 2000 segundos sin resincronizar, el desfasaje máximo supera los 30000 microsegundos. Con referencia única, figura 8.5b) dicho valor es de 750 microsegundos, o sea unas 40 veces más pequeño. Esto demuestra que, más allá del error inicial que puede ser aproximadamente igual en ambos casos, se obtiene un error en frecuencia sensiblemente menor. Esto llevará a una menor pérdida de sincronización a medida que transcurre el tiempo y evitará una resincronización de relojes para mediciones de tiempos relativamente largos. El hecho de que un gráfico presenta rectas y el otro curvas está relacionado con la escala del eje y, en realidad, la recta no es tal, sino su derivada segunda sería 0.



a) Referencia Local b) Referencia Única.

Figura 8.5:Diferencias de Tiempo de 4 clientes con respecto al Servidor de Hora.

8.4 Experimentos Relacionados con la Escalabilidad de la Biblioteca

Se realizaron experimentos para estimar los problemas de escalabilidad del algoritmo implementado en la biblioteca *st*. A priori se sabe que el modelo cliente/servidor implementado en *st* presentará un problema de escalabilidad. Así que se trata de determinar cuantitativamente el grado de incidencia del aumento de la cantidad de clientes a sincronizar. En los experimentos se midió:

- El tiempo para sincronizar de 2 a 16 máquinas entre sí.
- El tiempo de inicio del sistema, lo cual implica:
  - Creación de interfaces de comunicaciones
  - Mediciones a nivel local y de la red para la puesta en marcha de la biblioteca de referencias de tiempo a nivel local (*timings*)
- Estos tiempos de inicio van de un mínimo de 1290 milisegundos para 2 máquinas hasta 6558 milisegundos para 16 máquinas.

El experimento sobre la biblioteca se implementó en un cluster de 16 máquinas con las características dadas en la tabla 8.2.

PC	CPU	Frecuencia (MHz)	Sistema Operativo
P42400	Intel Pentium 4	2400	Linux 2.4.18-14

Tabla 8.2: Detalles de las PCs de la figura 8.7

La tabla 8.3 muestra los valores obtenidos en un experimento realizado midiendo los tiempos de comunicación, ida y vuelta dividido dos, entre 2 hasta 16 computadoras, y llevando la estadística durante 20 mediciones en cada caso. Se reportan los valores máximo y mínimo, expresados en microsegundos. Para el cálculo del error se considera que los mensajes con referencias sólo se consideran válidos en caso de realizarse en un tiempo igual a la moda, y como dicho tiempo se computa en ida y vuelta, el error nunca puede ser mayor que la posible asimetría del valor moda. El valor máximo de error, que es poco probable, presenta una posible asimetría, que se calcula a partir de restar los valores de moda menos 22. Se debe tener en cuenta que el tiempo local hasta los buffers es constante e igual a 22 en todos los experimentos. Dichos valores máximos de error son casi imposibles pues suponen una asimetría máxima, o sea, todo el tiempo computado a la ida y nada a la vuelta o viceversa:

Moda	Máximo	Mínimo	Local	Error máximo
54-55	116-296	51-53	22-22 (cero variación)	32-33

Tabla 8.3: Valores de tiempos de comunicación

Por otro lado, en un caso más probable, la asimetría no debiera ser, en ningún caso, un tiempo menor al mínimo, con lo que el valor de error final estaría acotado por el valor:

$$Error = Moda - Mínimo$$

que da un rango 1 a 4 microsegundos. Es importante notar la poca variabilidad de los tiempos medidos, esperable en un ambiente de red local, y teniendo en cuenta que se desacopla y controla el uso de la red de comunicaciones en todo momento. Esta situación es la misma que se logra en el uso de la herramienta. Se pueden comparar estos valores con los obtenidos en NTP en el capitulo 6, sección 6.4.2. La tabla 8.4 muestra dicha comparación. Los datos evitan cualquier comentario sobre la superioridad de la biblioteca *st* respecto de NTP para sincronizar la hora en un cluster, sobre todo, teniendo en cuenta que la experiencia se realizó sobre el mismo hardware:

	NTP		Biblioteca *st*	
	Máximo	Estimado	Máximo	Estimado
Min	239673	31	32	1
Max	989368	2434	33	4

Tabla 8.4: Comparación de error en sincronización de NTP contra la Bilblioteca *st*.

Otro experimento fue llevado a cabo con el fin de medir el tiempo que lleva sincronizar máquinas a fin de contar con una medida de la escalabilidad del algoritmo de sincronización. Para ello, se sincronizaron los relojes de 2 a 16 computadoras y se obtuvieron los valores que se muestran en la tabla 8.5, expresados en microsegundos. Algunos valores se han omitido por comodidad de la lectura de la tabla, ya que son lineales y pueden ser calculados por interpolación con los otros:

	2	4	6	8	10	12	14	16
Mínimo	37972	39902	40819	38596	41910	37522	38525	41774
Máximo	43362	46933	52908	48233	59397	53436	56029	52791
Promedio	40475	42444	45314	43585	47353	45433	47645	48312

Tabla 8.5: Tiempos de ejecución para 20 corridas de Synchro en 2 a 16 máquinas.

En forma gráfica se puede apreciar la evolución de estos tiempos en la Fig. 8.9, donde, además, se muestran los valores intermedios que se omitierono en la tabla 8.5.

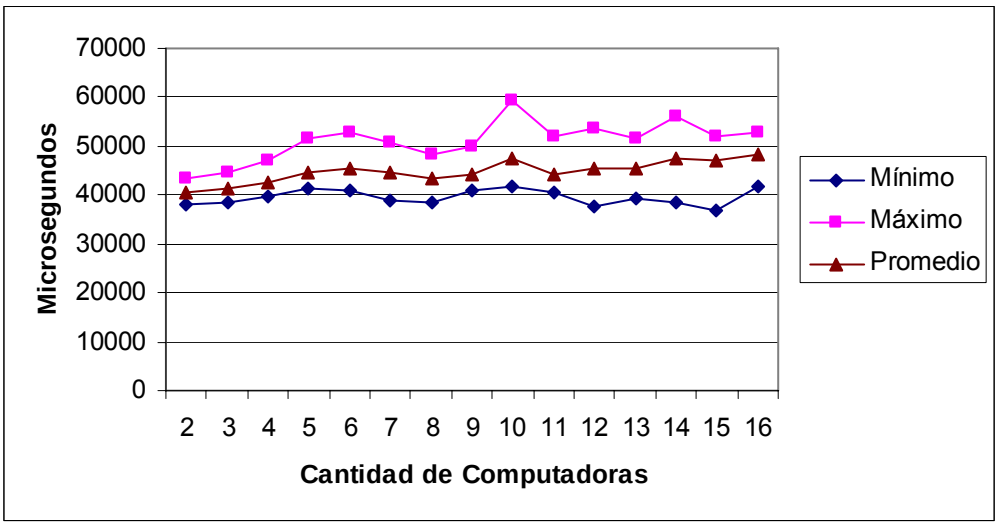


Figura 8.9: Tiempos de ejecución para 20 corridas de Synchro en 2 a 16 máquinas.

Estos resultados permiten concluir que la escalabilidad de este algoritmo, a pesar de basarse en un modelo cliente-servidor, no representa mayores incovenientes en los tamaños típicos de clusters. La sincronización se lleva a cabo entre el servidor y cada cliente, en forma secuencial. Como se desprende de restar al tiempo de sincronizar 16 el de sincronizar 2, y dividir dicho tiempo por 14 máquinas, corresponderían 560 microsegundos por máquina agregada a la sincronización. Los tiempos máximos medidos son menores a 60 milisegundos en total. Teniendo en cuenta la cantidad de computadoras de un cluster, son valores despreciables, sobre todo si se piensa que son tiempos previos al proceso de medición, y absolutamente aislados del mismo.

Es destacable la uniformidad y linealidad de los tiempos, lo que permite estimar extrapolaciones de valores. Por ejemplo, sería posible estimar en 560 x 100 = 56000, o sea, 56 milisegundos el tiempo agregado a los 40 milisegundos iniciales para sincronizar un cluster de 100 computadoras, llevando el total del tiempo de sincronización a menos de una décima de segundo para un cluster de 100 computadoras. En experimentos posteriores, se comprobó que los aproximadamente 40 milisegundos iniciales se deben a falta de sincronización entre clientes y el servidor. Al sincronizarse los clientes con un *delay*, desaparece dicho tiempo inicial, ya que solicitan sincronizarse recién cuando el servidor ya terminó de inicializarse.

Con respecto a la sobrecarga de los tiempos de medición durante la ejecución de un programa paralelo, se debe recordar que cada consulta de tiempo incluida en una medición implica un tiempo de entre 230 y 340 ciclos de reloj de CPU [37], con lo cual, en una computadora con velocidad de reloj 2Ghz, estariamos en el orden de 1 a 2 décimas de microsegundo por cada medición. Cabe recordar que en [37] se reporta un tiempo entre 402 y 615 ciclos para gettimeofday(), que es una llamada al SO, opcional al método utilizado por la biblioteca *timings*,.

Capítulo 9: Pruebas de Utilización de la Biblioteca

Resumen

En el presente capítulo se describen las pruebas llevadas a cabo sobre la utilización de la biblioteca de sincronización en la instrumentación de una multiplicación de matrices en un cluster.

9.1 Algoritmo a Evaluar

Dentro de las pruebas realizadas [44] [45] [46] a la herramienta de sincronización se encuentra la medición de un programa que implementa la multiplicación de matrices basado en mensajes *broadcast*, dado en [54]. La Fig. 8.1 muestra el esquema de dicha multiplicación de matrices para cuatro computadoras: la Fig. 8.1 muestra la distribución de las matrices y la Fig. 8.2 muestra la secuencia de pasos a realizar para completar el cómputo. El primer bloque de columnas de la matriz B, que está asignado a $Comp_1$, se denomina B_1; el segundo bloque, que está asignado a $Comp_2$, se denomina B_2 y así siguiendo hasta el bloque de columnas B_4, asignado a $Comp_4$. De manera similar, se denotan los bloques de filas de A y C, como A_1,…, A_4, y C_1,…, C_4, respectivamente. Las comunicaciones son necesarias específicamente por los datos de la matriz B, que se necesitan en todas las computadoras y es por eso que se elige la comunicación broadcast.

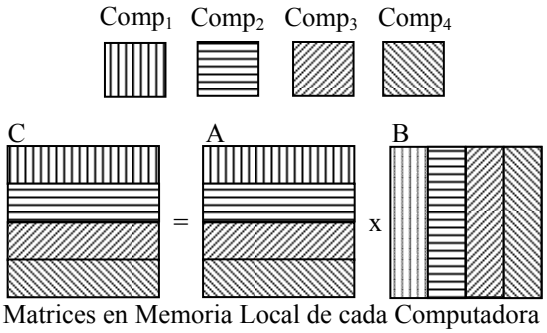


Figura 8.1: Distribución de Datos en cada $Comp_i$

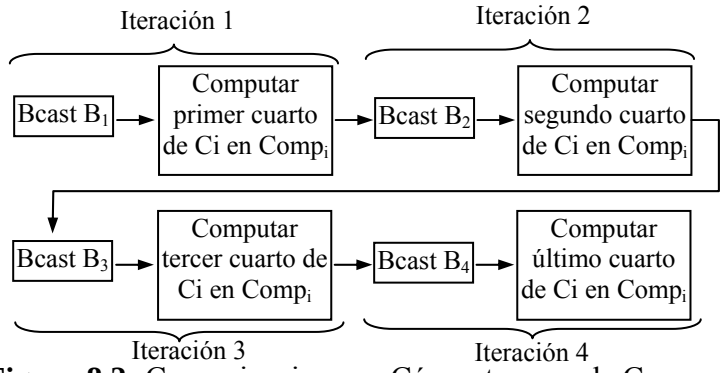


Figura 8.2: Comunicaciones y Cómputo en cada $Comp_i$

El proceso completo de multiplicación es iterativo, tal como se muestra en la Fig. 8.1.b). Cada una de las iteraciones implica multiplicar Ai x Bj, y da por resultado una porción de Ci; específicamente, una cuarta parte en la Fig. 8.1a) de la porción de Ci que debe calcular la $Comp_i$.

9.2 Resultados de la Evaluación en un Cluster

La implementación se llevó a cabo en el cluster que se describe anteriormente utilizando la biblioteca de pasaje de mensajes LAM/MPI. La tabla 8.1 muestra la sucesión de tiempos sincronizados medidos en cada computadora en cada iteración para los pasos de broadcast y cómputo local, para una multiplicación de matrices con tres computadoras,

donde se muestran los datos por iteraciones en cada computadora. Los datos correspondientes a los mensajes broadcast ("T pre broadcast" y "T post broadcast") así como los datos correspondientes a cómputo local ("T pre multiplic." y "T post multiplic."), se dan en microsegundos relativos al comienzo del programa, tal como se dan en cualquier traza o sucesión de eventos marcados en el tiempo. La columna "T medido" indica el tiempo transcurrido entre dos eventos, también en microsegundos. La fila identificada con **(*1)** muestra que la computadora 0 completa antes que las demás la multiplicación local de la segunda iteración y, por lo tanto, como se muestra en la fila identificada con **(*3)**, la ejecución del mensaje broadcast de la iteración siguiente comienza antes (hace la llamada a la rutina broadcast de MPI). Dado que la computadora 2 utiliza más tiempo del esperado para la multiplicación local de la segunda iteración, tal como lo indica la fila identificada con **(*2)**, esta computadora retrasa el tiempo total del broadcast de la computadora 0. Los tiempos de los eventos de las filas marcadas con **(*1)**, **(*2)**, y **(*3)** muestran claramente que el problema no es el mensaje broadcast sino la falta de sincronismo en la llegada a la ejecución del mismo, en particular entre las computadoras 0 y 2.

Iteración 1			
Computadora	T pre broadcast	T post broadcast	T medido
0	1308	45349479	45348171
1	0	45265849	45265849
2	1358	45354136	45352778
	T pre multiplic.	T post multiplic.	
0	45349660	106538154	61188494
1	45266079	111471259	66205180
2	45354379	111751098	66396719
Iteración 2			
Computadora	T pre broadcast	T post broadcast	T medido
0	111751346	157630336	45878990
1	106538384	155790790	49252406
2	111471465	157624078	46152613
	T pre multiplic.	T post multiplic.	
(*1) 0	157630495	220519014	62888519
1	155791002	225184906	69393904
(*2) 2	157624396	241815043	84190647
Iteración 3			
Computadora	T pre broadcast	T post broadcast	T medido
(*3) 0	220519252	280237878	59718626
1	225185128	260482253	35297125
2	241815389	280211015	38395626
	T pre multiplic.	T post multiplic.	
0	260482431	324973188	64490757
1	280238147	371666747	91428600
2	280211238	382929145	102717907

Tabla 8.1: Datos de Instrumentación de la Multiplicación de Matrices con Tres Computadoras

Es importante remarcar que los valores que se muestran en la tabla 8.1 se obtienen utilizando llamadas a la biblioteca de sincronización de relojes y, además, esta utilización no implica más de 20 líneas de código agregado al programa *original* de multiplicación de matrices.

Capítulo 10: Conclusiones y Trabajos Futuros

Resumen

El problema de la sincronización de relojes presenta una variedad de soluciones para el estado actual de las arquitecturas, tanto de red como de computadoras. El advenimiento de la computación móvil, presenta aspectos nuevos que deben ser estudiados, de igual manera que los sistemas de múltiples procesadores.

10.1 Conclusiones

La elección del algoritmo de Cristian parece la más lógica para el problema que se intenta solucionar. El algoritmo de Lamport sólo se refiere a un orden de eventos y en los lapsos en los que dichos eventos no producen pasajes de mensajes entre procesos, no se puede asegurar el orden entre un evento en un proceso y otro evento en otro, en el caso de que dichos procesos corran en máquinas diferentes.

El problema de la sincronización de relojes para su uso en instrumentación fue investigado a partir de los problemas que surgieron al utilizar las herramientas existentes. En algunos casos, como el *timed* [BERKELEY], el inconveniente proviene de su baja resolución, ya que no se pueden medir errores de sincronización menores al milisegundo. En el caso de NTP, los errores eran demasiado grandes para ser aceptables. La tabla 8.4 del capítulo 8, que se repite aquí para comodidad en la lectura, muestra la mejora obtenida en error máximo y estimado por la biblioteca desarrollada durante esta tesis.

	NTP		Biblioteca *st*	
	Máximo	Estimado	Máximo	Estimado
Min	239673	31	32	1
Max	989368	2434	33	4

Tabla 9.1: Comparación de error en sincronización de NTP contra la Bilblioteca *st*.

En cuanto a la intrusión, al evitar las llamadas al sistema y el uso de funciones, se mejora notablemente la fiabilidad de los tiempos obtenidos en la instrumentación. Tal como se mostró en el capítulo 3, sección 3.6 el uso de TSC para tener referencias de tiempo demostró tener buena resolución, y bajo nivel de sobrecarga de procesamiento, gettimeofday() cuesta 7876 ciclos de reloj, mientras que ulocal_time() cuesta aproximadamente 2900 ciclos de reloj.

La sobrecarga en comunicaciones, para el enfoque de la instrumentación, implica poder aislar las comunicaciones provenientes del proceso de sincronización de la ejecución de la aplicación que se quiere instrumentar. Este objetivo lo cumple la biblioteca, no así NTP o Timed. En estas aplicaciones, el usuario no tiene un control sobre su ejecución, más allá de bajar el servicio, con la correspondiente pérdida de sincronización.

Cabe destacar la sencillez del uso y utilidad para detectar problemas de la biblioteca, lo cual se puede ver en el capítulo 9, en la multiplicación de matrices, donde se detecta la verdadera razón del problema en base a la instrumentación de tiempos.

10.2 Trabajos Futuros

Las nuevas líneas de investigación referidas al tema de sincronización de computadoras vienen de la mano de los cambios que surgen tanto en la arquitectura de redes como en la de las computadoras mismas. Con respecto a las redes de computadoras [27], el advenimiento de la computación móvil, Internet II y las necesidades de saber la ubicación de los dispositivos móviles traen aparejados nuevos requerimientos. En esta línea se debe trabajar sobre el protocolo PTP [58], el que aparentemente desplazaría a NTP. En este contexto, se debería probar la implementación de PTPd, el proceso de

PTP y comprobar, teniendo acceso a multicast, los verdaderos valores alcanzables de sincronización. Una inmediata línea de investigación en este tema está relacionada con ampliar la sincronización, con fines de hacer medidas de rendimiento, a ambientes interclusters. Las posibles variantes para lograr la sincronización entre clusters conectados mediante Internet pasan por utilizar PTP, NTP o utilizar la red de UTC mediante el uso de receptores conectados a una de las máquinas en cada cluster.

Con respecto a los cambios en la arquitectura misma de las computadoras, esto impacta en tres casos: las computadoras con procesadores múltiples, los procesadores con múltiples núcleos y los sistemas de ahorro de energía que incluyen variaciones de velocidad del procesador.

Otra de las líneas de desarrollo para continuar está relacionada con la mejora en la escalabilidad de la sincronización. Al respecto, se pueden seguir dos enfoques:

- Que cada máquina provea sincronización a la siguiente: En este caso el camino que se debe seguir es crítico. Cada máquina intermediaria introducirá un error debido a la variabilidad del tiempo de transmisión del mensaje con la siguiente, con lo cual, cada nuevo salto o pasaje de mensajes entre un nodo y otro (hop) introduce un error a la sumatoria de errores. En este caso, se prioriza la distribución. En el caso extremo, para n máquinas a sincronizar habrá n-1 hops en una arquitectura tipo anillo, con lo cual, si suponemos que los errores debidos a la varianza en los tiempos de comunicación son homogéneos e iguales a Δt, el error total será del orden de $(n-1) \times \Delta t$.
- En este enfoque, lo mejor sería centralizar con un único servidor de hora. Por supuesto, esto es de difícil escalabilidad en Internet. En una red local se puede recurrir a mensajes broadcast.

Como se puede apreciar, la solución será una combinación de los dos enfoques vistos. La arquitectura adoptada por NTP es de estratos en los que se adopta un compromiso entre la cantidad de hops y la sobrecarga de los servidores a través de un árbol de jerarquías.

Otra posible arquitectura para investigar sería un servidor en cada LAN conectado vía radio a la referencia GPS, con un SO de tiempo real, que suministre por broadcast la referencia [19]. También se deberá abordar el problema de la sincronización en máquinas con varios procesadores (multicore), ya que en un futuro, los clusters estarán compuestos por máquinas de este tipo.

Bibliografía

[1]The Science of Timekeeping.pdf (Allan) en from/mas/tin

[2] Allan.Varianza de Allan

[3] Aggelos Bletsas, “Evaluation of Kalman Filtering for Network Time Keeping”, ieee transactions on ultrasonics, ferroelectrics, and frequency control, vol. 52, no. 9, september 2005

[4] Probabilistic clock synchronization Authors: Flaviu Cristian.Distributed Computing Springer verlag 1989.

[5] Building Fault-Tolerant Hardware Clocks from COTS Components-Christof Fetzer and Flaviu Cristian

[6] Fine-Grained Network Time Synchronization using Reference Broadcasts .Jeremy Elson, Lewis Girod and Deborah Estrin Department of Computer Science, University of California, Los Angeles fjelson,girod,destring@cs.ucla.edu

[7] Transp: http://www.cis.uoguelph.ca/~xli/courses/cis4400/c6.pdf

[8] TSC-I2: A Lightweight Implementation for Precision-Augmented Timekeeping Xun Luo, University of Illinois at Chicago, X. Luo, “TSC-I2: A Lightweight Implementation for Precision-Augmented Timekeeping”, reporte técnico de University of Illinois at Chicago, http://tsc-xluo.sourceforge. net/TSC-I2_Tech_Report.pdf

[9] http://fasttime.sourceforge.net/doc/internal.html

[10] Coulouris G., Dollimore J., Kinberg T., “Sistemas Distribuidos. Conceptos y Diseño”, 3ª edición. Pearson Educación, 2001. ISBN: 8478290494.

[11] Cristian F. "Synchronous and Asynchronous Group Communication", Comm. ACM, Vol.39, No. 4, April 1996, pp.8897.

[12] F. Cristian. "Probabilistic Clock Synchronization". Distributed Computing, 3: 146–158, 1989.

[13] Cristian F., C. Fetzer. "The Timed Asynchronous Distributed System Model" IEEE Transactions on Parallel and Distributed systems, June 1999, pp. 603618.

[14] Cristian F., Fetzer C., "The Time Asynchronous Distributed System Model", IEEE Transactions on Parallel Systems, June 1999, pp. 603618.

[15] http://www-306.ibm.com/software/network/dce/library/publications/dceaix_ 22/ a3u2j/ A3U2JM02.HTM# ToC
http://www.univie.ac.at/dcedoc/A3U2S/A3U2SM02.HTM#ToC_94

[16] DTS: Remote Procedure Calls.http://www-306.ibm.com/software/network/dce/library/publications/dceaix_22/a3u2j/A3U2J160.HTM

[17] DTS: Funciones http://www-306.ibm.com/software/network/dce/library/publications/dceaix 22/ a3u2j/A3U2J158.HTM

[18] Elson K. J., Romer K., "Wireless Sensor Networks: A New Regime for Time Synchronization in Distributed Systems", Proceedings of the First Wirkshop on Hot Topics In Networks (HotNets1), Princeton, New Jersey, October 2002.

[19] Elson K. J., Girod L., Estrin D., "FineGrained Network Time Synchronization using Reference Broadcasts", Proceedings of fifth symposium on Operating System Design and Implementation. December 2002.

[20] Essen, L., "Frequency and Time Standards," *Proceedings of the IRE* , vol.50, no.5, pp.1158-1164, May 1962
URL: http://ieeexplore.ieee.org/iel5/10933/4066672/04066832.pdf?isnumber=4066672&prod=JNL&arnumber=4066832&arnumber=4066832&arSt=1158&ared=1164&arAuthor=Essen%2C+L.

[21] Fetzer C., Christian F., "Integrating External and Internal Clock Synchronization", June 1996. Real-Time Systems. Springer Netherlands. ISSN 0922-6443 (Print) 1573-1383 (Online). Vol. 12, Number 2 / marzo de 1997.DOI 10.1023/A:1007905917490. Páginas 123-171. Subject Collection Informática

[22]http://www.elo.utfsm.cl/~mineducagv/docs/ListaDetalladadeModulos/tutorial_ps.pdf

[23] Grove D. A.. "Performance Modelling of messagepassing parallel programs", Thesis (Ph.D.), Computer Science, University of Adelaide, Mayo 2003

[24]Riccardo Gusella , Stefano Zatti, The Berkeley UNIX 4.3BSD Time Synchronization Protocol, University of California at Berkeley, Berkeley, CA, 1985

[25]Robert Jourdain, “Programmer’s Problem Solver”, Brady Publishing,ISBN 0-13-720194-X,1992

[26](http://www-unix.mcs.anl.gov/~kazutomo/rdtsc.html)

[27] Kim K. H., Im C., Athreya P, "Realization of a Distributed OS Component for Internal Clock Synchronization in a LAN Environment".Proc. ISORC 2002, IEEE 5th Int'l Symp on Objetoriented Realtime distributed Computing, Washington, D.C., April 2002, pp. 263270.

[28] Leslie Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," Communications of the ACM, July 1978, 21(7):558-565.

[29] J. Levine. Introduction to time and frequency metrology.
Review of Scientific Instruments, 70(6), June 1996.

[30] Paolo Mantegazza, “DIAPM-RTAI Programming Guide 1.0.” https://www.rtai.org. September 2000

[31] P. Cloutier, P. Mantegazza, S. Papacharalambous, I. Soanes, S. Hughes, and K. Yaghmour. Diapm-rtai position paper. Real Time Operating Systems Workshop, November 2000.

[32] Mills D. L. "Measured performance of the Network Time Protocol in the Internet System". ACM Computer Communication Review 20, Jan. 1990. pp. 6575.

[33] Mills D. L., "Modelling and analysis of computer network clocks", Electrical Engineering Department Report 9252, University of Delaware, May 1992.

[34] Mills, D. L. "Improved algorithms for synchronizing computer network clocks", IEEE/ACM Transactions on Networks June 1995.

[35] Mills D. L. "Measured Performance of the Network Time Protocol in the Internet System", ACM Computer Review 20, 1,
January 1990, pp.6575.

[36] Mills D.L., "Internet time Synchronization: the Network Time Protocol", IEEE trans. Communications COM39, October 1991, pp. 14821493.

[37] Mills D.L., "Network Time Protocol (Version 3) specification, implementation and analysis", DARPA Networking Group Report RFC1305, University od Delaware, March 1992.

[38] Mills D. L., "A Brief History of NTP Time: Confessions of an Internet Timekeeper". ACM Computer Communications Review 33, 2 (April 2003), pp 922.

[39] Mills, D.L. "Unix kernel modifications for precision time synchronization". Electrical Engineering Department Report 94101, University of Delaware, October 1994.

[40] Mills, D.L, Kamp P.H., "The Nanokernel", Proc. Precision Time and Time Interval (PTTI) Applications and Planning Meeting (Reston VA, November 2000).

[41] http://www.cis.udel.edu/~mills/database/papers/trans.pdf y en disco D

[42](ref:Museo maritimo nacional de Greenwich,
http://www.nmm.ac.uk/server/show/conWebDoc.355/viewPage/1).

[43] http://www.ntp.org/ntpfaq/NTP-s-sw-clocks-quality.htm

[44] Fernando L. Romero, Walter Aróztegui, Fernando G. Tinetti,"Sincronización de Relojes en Ambientes Distribuidos" XII Congreso Argentino de Ciencias de la Computación (XII CACIC) Octubre 2006

[45] Fernando L. Romero, Fernando G. Tinetti. “Problemas de la Sincronización de Relojes en Clusters”. XIII Congreso Argentino de Ciencias de la Computación (XIII CACIC) Octubre 2007

[46]CACIC 2008 Fernando L. Romero, Armando E. De Giusti, Fernando G. Tinetti. “Sincronización de Relojes para Evaluación de Rendimiento: Experiencias en un Cluster Utilizado para Cómputo Numérico”. XIV Congreso Argentino de Ciencias de la Computación (XIV CACIC) Octubre 2008

[47] Rubini A., Corbet J., "Linux Device Drivers 2nd Edition" ISBN 0596000081. June 2001.

[48]Smith, W. L., Precision Oscillators. In: Precision Frequency Control, Vol. 2 (E. A. Gerber and A. Ballato, eds.), Academic Press, New York, pp. 45-98, 1985.

[49] Stallings, William, “Comunicaciones y redes de computadores”, 7ª ed., Prentice Hall,Pearson Educación, S.A., Madrid, 2007. ISB : 84-205-4110-9

[50]Falta referencia [figura free runing]

[51] A. S. Tanenbaum. Sistemas Operativos Distribuidos. Prentice Hall Hispanoamericana, S.A., México, 1996

[52] Tanembaum Andrew. “Computer Networks” Second Edition. ISBN 0-13-162959-X. Prentice Halls 1995.

[53]Gregory D. Troxel “Time surveying: Clock Synchronization Over Packet Network” Tesis Doctorado en Filosofía en Electrical Engeniering and Computer Science. MTI. 1994. en escritorio/sinc ext y inter

[54]F. G. Tinetti, Cómputo Paralelo en Redes Locales de Computadoras, Tesis Doctoral (Doctorado en Informática), Universidad Autónoma de Barcelona, Marzo de 2004. Disponible en http://ftinetti.googlepages.com/tesisdoctoral

[55]http://answers.google.com/answers/threadview?id=203397

[56] Digital UNIX Frequently Asked Questions (FAQ). http://www.hmi.de/it/software/tru64-faq.html

[57]Manual de Unix. Timed.

[58] A. Vallat, D. Schneuwly, ”Clock Synchronization in Telecommunications via PTP (IEEE 1588)” , Frequency Control Symposium, 2007 Joint with the 21st European Frequency and Time Forum. IEEE International. May 29 2007-June 1 2007.pp. 334-341. ISSN: 1075-6787. ISBN: 978-1-4244-0647-0

[59] Work P., Nguyen K., "Measure Code Sections Using The Enhanced Timer", http://www.intel.com.ar, October 2005.

[60] Zhao Y., Zhou W., Huang J, Yu S., "SelfAdaptive Clock Synchronization for Computational Grid", Journal of Computer Science and Technology, 2003 Volume: 18 Issue: 4 pp. 434 – 441.

[61]http://isotc.iso.org/livelink/livelink/4021199/ISO_8601_2004_E.zip?func=doc.Fetch&nodeid=4021199

Agradecimientos

A mi familia que me apoya en todos mis proyectos, especialmente a mi esposa Carmen y a mi hija Mariela, que me corrigió la redacción en varios lugares.

A la gente del IIILIDI y de la Facultad de Informática, que me aceptó a pesar de que ingresé ya grande para becario de un Master, brindandome siempre su apoyo cada vez que lo solicité, especialmente a:

Mi Director Dr. Fernando Tinetti, que tuvo que hacer horas extras para atenderme, enseñandome hasta a programar y estilos de escribir.

El Director del IIILIDI, Ing. Armando Degiusti, que fue el que me invito en primer lugar a trabajar en el instituto y a realizar estos estudios.

Al Lic. Franco Chichizola y al Lic. Andrés Barbieri, que mas de una vez me ayudaron en cuestiones técnicas, además de brindarme su amistad.