# Sincronización de Relojes en Ambientes Distribuidos

**Fernando L. Romero, Walter Aróztegui, Fernando G. Tinetti**[1]

**Instituto de Investigación en Informática LIDI (III-LIDI)**
**Facultad de Informática – UNLP**

**Centro de Técnicas Analógico-Digitales (CeTAD)**
**Facultad de Ingeniería - UNLP**
fromero@lidi.info.unlp.edu.ar, waroz@graffiti.net, fernando@info.unlp.edu.ar

## CONTEXTO

Esta línea de Investigación forma parte de dos de los Subproyectos dentro del Proyecto "Sistemas Distribuidos y Paralelos" acreditado por la UNLP y de proyectos específicos apoyados por CyTED, CIC, Agencia e IBM.

## RESUMEN

Esta línea de investigación se orienta a resolver el problema de sincronización de tiempo en ambientes distribuidos. El objetivo inicial de la sincronización de relojes es la estimación de rendimiento a partir de experimentos con la mínima instrumentación (y su consecuente interferencia) posible.

El algoritmo básico sigue las estrategias clásicas de sincronización de tiempo en ambientes distribuidos [1] [3] [11]. Sin embargo, se adaptan (al menos en principio) al entorno de un cluster o, al menos, de una red de interconexión sobre la que se tiene acceso exclusivo (o al menos controlado) para todas las comunicaciones entre las computadoras que se sincronizan. Este ambiente es específicamente el de los entornos de cómputo paralelo en clusters.

***Keywords:*** *Sincronización de Procesos, Relojes Distribuidos, Rendimiento e Instrumentación, Sistemas Paralelos y Distribuidos, Paralelismo en Clusters e Interclusters, Sincronización Interna y Externa .*

## 1. INTRODUCCION

Siempre ha sido necesario disponer de un sistema capaz de proporcionar una referencia de tiempo en los sistemas de cómputo [19] [21]. Dicha referencia es esencial para resolver problemas tales como el ordenamiento de eventos (ej: envío y recepción de correo electrónico, eventos dentro de las transacciones, inicio de procesos en tiempo real, etc.).

También cobra importancia la medición de tiempos en la optimización del rendimiento tanto en sistemas de cómputo monoprocesador como en sistemas paralelos y distribuidos [2] [4] [7] [12] [13]. En todos los casos, las aplicaciones con fuertes requerimientos de cómputo o procesamiento son las que también requieren la optimización para el máximo aprovechamiento del hardware disponible. La relación es bastante directa: a partir de la monitorización de los tiempos de ejecución se pueden analizar los problemas de rendimiento e intentar solucionarlos [9].

Aún en el caso de las mediciones en una misma computadora (usualmente en el contexto de un sistema con un único procesador), es deseable que el registro de tiempos no influya en el tiempo de ejecución de la misma. En todos los casos, se necesitan resoluciones de reloj acordes a los tiempos que se deben medir en las aplicaciones.

[1] Investigador Asistente CICPBA

Usualmente, los métodos provistos por el sistema operativo no son apropiados [20] para todos los casos. Por otro lado, los métodos y/o herramientas provistas por los lenguajes dependen del sistema operativo y, por lo tanto, resultan inadecuados también.

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 conlleva la necesidad de sincronizar los relojes de las diferentes computadoras que se utilizan [5] [16] [17]. Sería deseable que esta tarea de sincronización se lleve a cabo fuera del tiempo en que se ejecute el programa que se está monitorizando, y conociendo el tipo (o al menos magnitud) de error con que se sincroniza.

Por otro lado, es deseable que dicha sincronización se lleve a cabo sin la 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 existente y el sistema de medición existente en cada sistema de cómputo.

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.
- Que 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.

## 2. LINEAS DE INVESTIGACION Y DESARROLLO

Inicialmente, se estudian tanto los algoritmos básicos como las implementaciones existentes. En este sentido, se cuenta con una amplia cantidad de información tanto de los algoritmos como de las implementaciones. Como requisito previo, normalmente se establece que cada computadora cuente con un oscilador físico de frecuencia más o menos constante. A partir de este oscilador físico se derivan los relojes lógicos que son los que se sincronizan [8] [10] [15]. En todos los casos, lo que se tiende a resolver son las diferencias de (Fig. 1):

1. Referencia fija en el tiempo a partir de la cual se contabiliza el tiempo en cada computadora. Aunque normalmente es constante, es relativamente difícil establecer con precisión su valor.
2. Frecuencia entre los relojes de las computadoras que se sincronizan. En algunos casos, se suelen incluir en este punto las diferencias de las variaciones de los relojes, dado que los relojes no necesariamente tienen frecuencia constante a lo largo del tiempo.

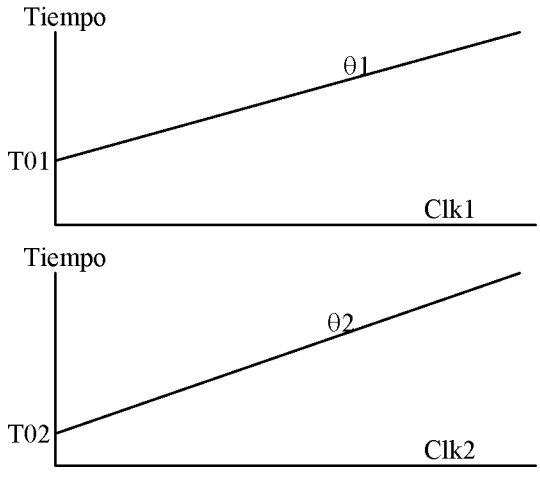


Figura 1: Relojes de Dos Computadoras.

La Fig. 1 muestra dos computadoras, cada una de ellas con su oscilador, que a su vez tiene sus propias características. El instante de tiempo T01 es la referencia al tiempo real en una computadora y T02 es el equivalente en la otra computadora. La pendiente θ1 corresponde al oscilador de una computadora y θ2 es la correspondiente pendiente de la otra.

Una de las formas más sencillas e intuitivas de sincronizar dos computadoras consiste en determinar el reloj de una de ellas en función del reloj de la otra teniendo en cuenta algún tipo de interconexión entre ellas. La Fig. 2 muestra esquemáticamente esta forma de sincronización, donde el reloj de una computadora se puede establecer a partir del reloj de la otra teniendo en cuenta el tiempo de las comunicaciones de sincronización entre las dos.

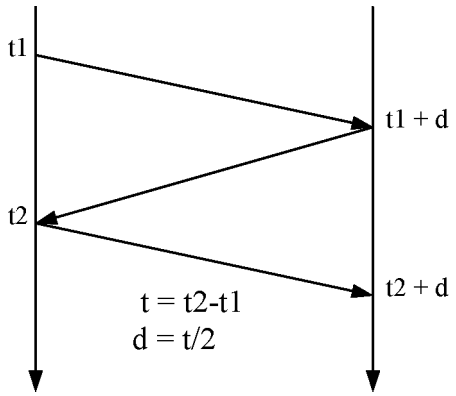


Figura 2: Sincronización Básica.

Sin embargo, como paso previo se deben investigar las formas de hacer referencias al reloj físico y/o a los relojes lógicos con la mínima sobrecarga. La idea básica para disminuir la sobrecarga normalmente se basa en evitar llamadas al sistema operativo para, al menos, evitar el consiguiente cambio de contexto. También es necesario investigar en particular hasta qué punto las referencias a los relojes/osciladores físicos son portables o al menos tienen funciones u operaciones equivalentes en diferentes plataformas de hardware.

Paralelamente a lo anterior, al menos en principio, es necesario analizar y, más específicamente cuantificar las características de NTP (Network Time Protocol). Esta cuantificación se refiere principalmente a:

- Resolución posible del reloj sincronizado. Aunque NTP es paramétrico esto no significa *a priori* que se puede obtener una resolución arbitraria cualquiera.
- Relación entre la resolución definida y la sobrecarga en la red de comunicaciones.
- Relación entre la resolución definida y la sobrecarga en la pila de protocolos TCP/UDP/IP.
- Relación entre la resolución definida y la sobrecarga en el uso de CPU y la jerarquía de memoria.

Todo este análisis es básicamente experimental [14] [17] [18], 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.

Una vez realizada la sincronización mínima entre las computadoras se debe investigar el comportamiento de la misma en términos de escalabilidad. Usualmente la sincronización se da entre dos máquinas y en el caso particular de NTP se lleva a cabo con el modelo cliente/servidor. Es claro que cualquier tipo de centralización (en el servidor, por ejemplo) tiene sus inconvenientes de escalabilidad y al menos debería ser posible su cuantificación. En este contexto específico es muy interesante la posibilidad de sincronización utilizando mensajes broadcasts con su consiguiente ahorro de comunicaciones punto a punto.

Como extensiones futuras, siempre es deseable la sincronización externa de los relojes [8]. Este paso está muy ligado también a la posibilidad de utilizar más de un cluster de computadoras para cómputo paralelo y en este contexto la sincronización de los relojes va más allá del análisis de rendimiento con el objetivo de optimizarlo. Las alternativas en este contexto están abiertas para varias posibilidades. Entre otras alternativas: una nivel jerárquico superior a to-

dos los clusters, tomar un cluster como el de mayor jerarquía, sincronizar con algún tipo de estrategia de tiempo real, etc.

## 3. RESULTADOS OBTENIDOS/ESPERADOS

Inicialmente, la idea es contar con una biblioteca mínima en cuanto a la medición de tiempos de cómputo en un mismo sistema (usualmente en una máquina monoprocesador). Ya se cuenta con una, para el sistema operativo Linux en PCs (procesadores Intel y compatibles) en las cuales se puede hacer referencia al contador de ciclos del oscilador. Entre las tareas pendientes se tienen la cuantificación de la sobrecarga y posibilidad de extensión a otros sistemas operativos y/o plataformas de hardware. En lo referente a la sobrecarga, los datos preliminares muestran resultados altamente satisfactorios, dado que la sobrecarga no excede las decenas de ciclos de reloj. En este sentido, ya es posible la instrumentación de código y el análisis de las aplicaciones que se resuelven en un mismo sistema o máquina. La precisión es satisfactoria: del orden de los microsegundos.

Se está desarrollando software (básicamente una biblioteca de una cantidad reducida de funciones) para instrumentación de programas paralelos que se ejecuten en clusters de PCs. Esta biblioteca tiene los lineamientos dados antes, con énfasis en la resolución del orden de los microsegundos, la mínima sobrecarga de procesamiento y el desacople de la red de interconexión.

Además, se están llevando a cabo pruebas sobre los sistemas de relojes distribuidos empleados en este momento tales como NTP (Network Time Protocol) y DTS (Distributed Time Service), con el fin de analizar sus ventajas y desventajas. También un análisis de los sistemas que utilizan hardware especializado, como para tener un marco de referencia más amplio [6]. En todos estos casos, la idea final es contar con un conjunto de programas del estilo de los *benchmarks* para que provean automáticamente la caracterización del sistema de sincronización elegido/utilizado. Como mínimo, el objetivo es contar con una metodología de caracterización de las herramientas, bibliotecas, y/o protocolos de sincronización que incluso pueda ser aplicada a la nueva biblioteca que se desarrolle, mencionada antes.

Como paso posterior, se espera extender la biblioteca para su uso en Internet y múltiples clusters para cómputo paralelo [22]. Quizás en este punto se deban redefinir algunas características de la herramienta o biblioteca, tal como la capa de transporte de los mensajes con los cuales se implementa la sincronización.

## 4. FORMACION DE RECURSOS HUMANOS

En esta línea de I/D existe cooperación a nivel nacional e internacional. Inicialmente se tiene una posible tesis de maestría y está abierta la posibilidad para varias Tesinas de Grado de Licenciatura.

## 5. BIBLIOGRAFIA

[1] Coulouris G., Dollimore J., Kinberg T., "Sistemas Distribuidos. Conceptos y Diseño", 3ª edición. Pearson Educación, 2001. ISBN: 84-7829-049-4.

[2] Cristian F. "Synchronous and Asynchronous Group Communication", Comm. ACM, Vol.39, No. 4, April 1996, pp.88-97.

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

[4] Cristian F., C. Fetzer. "The Timed Asynchronous Distributed System Model" IEEE Transactions on Parallel and Distributed systems, June 1999, pp. 603-618.

[5] Cristian F., Fetzer C., "The Time Asyn-

chronous Distributed System Model", IEEE Transactions on Parallel Systems, June 1999, pp. 603-618.

[6] 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 (HotNets-1), Princeton, New Jersey, October 2002.

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

[8] Fetzer C., Christian F., "Integrating External and Internal Clock Synchronization", June 1996.

[9] Grove D. A.. "Performance Modelling of message-passing parallel programs", May 2003.

[10] 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 Objet-oriented Real-time distributed Computing, Washington, D.C., April 2002, pp. 263-270.

[11] Mills D. L. "Measured performance of the Network Time Protocol in the Internet System". ACM Computer Communication Review 20, Jan. 1990. pp. 65-75.

[12] Mills D. L., "Modelling and analysis of computer network clocks", Electrical Engineering Department Report 92-5-2, University of Delaware, May 1992.

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

[14] Mills D. L. "Measured Performance of the Network Time Protocol in the Internet System", ACM Computer Review 20, 1, January 1990, pp.65-75.

[15] Mills D.L., "Internet time Synchronization: the Network Time Protocol", IEEE trans. Communications COM39, October 1991, pp. 1482-1493.

[16] Mills D.L., "Network Time Protocol (Version 3) specification, implementation and analysis", DARPA Networking Group Report RFC-1305, University od Delaware, March 1992.

[17] Mills D. L., "A Brief History of NTP Time: Confessions of an Internet Timekeeper". ACM Computer Communications Review 33, 2 (April 2003), pp 9-22.

[18] Mills, D.L. "Unix kernel modifications for precision time synchronization". Electrical Engineering Department Report 94-10-1, University of Delaware, October 1994.

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

[20] Rubini A., Corbet J., "Linux Device Drivers 2nd Edition" ISBN 0-59600-008-1. June 2001.

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

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

# Herramientas para Instrumentación de Programas Paralelos en Ambientes Distribuidos

**Fernando G. Tinetti*, Fernando L. Romero**
III-LIDI, Facultad de Informática, UNLP
50 y 115, 1900, La Plata, Argentina
fernando@info.unlp.edu.ar, fromero@info.unlp.edu.ar

**Abstract**

This paper presents a methodology and tool for instrumenting parallel programs in distributed computing platforms with the highest possible resolution (microseconds, if possible) and the minimum interference/overhead in programs. In the context of time instrumentation in distributed environments, it is essential to synchronize the involved clocks. In order to design the basic synchronizing algorithm, the classical strategies used in distributed environments were followed and adapted to the environment of a cluster or, at least, of an interconnection network to which there is exclusive (or *controlled*) access for the all communications carried out among the synchronized computers. This environment is specifically represented by those of parallel computing in clusters.

**Keywords:** Parallel Performance and Instrumentation, Process Synchronization, Distributed Clocks, Parallel and Distributed Systems, Parallelism in Clusters and Interclusters, Internal and External Synchronization.

**Resumen**

En este artículo se presenta una metodología y una herramienta para la instrumentación de programas paralelos en plataformas de cómputo distribuidas con la mayor resolución posible (microsegundos, si fuera posible) y la mínima interferencia/sobrecarga en los programas. En el contexto de instrumentación de tiempo en ambientes distribuidos es fundamental la sincronización de los relojes que intervienen. Para el diseño del algoritmo básico de sincronización se siguieron las estrategias clásicas que se utilizan en ambientes distribuidos, adaptadas al entorno de un cluster o, al menos, de una red de interconexión sobre la que se tiene acceso exclusivo (o *controlado*) para todas las comunicaciones entre las computadoras que se sincronizan. Este ambiente es específicamente el de los entornos de cómputo paralelo en clusters.

**Palabras claves:** Rendimiento e Instrumentación Paralela, Sincronización de Procesos, Relojes Distribuidos, Sistemas Paralelos y Distribuidos, Paralelismo en Clusters e Intercluster, Sincronización Interna y Externa.

## 1 INTRODUCCION

Desde hace tiempo se dispone de un sistema capaz de proporcionar una referencia de tiempo en los sistemas de cómputo [20] [22]. Dicha referencia es esencial para resolver problemas tales como el ordenamiento de eventos (ej: envío y recepción de correo electrónico, eventos dentro de las transacciones, inicio de procesos en tiempo real, etc.).

* Investigador Asistente CICPBA

También cobra importancia la medición de tiempos en la optimización del rendimiento tanto en sistemas de cómputo monoprocesador como en sistemas paralelos y distribuidos [2] [4] [7] [13] [14]. En todos los casos, las aplicaciones con fuertes requerimientos de cómputo o procesamiento son las que también requieren la optimización para el máximo aprovechamiento del hardware disponible. La relación es bastante directa: a partir de la monitorización de los tiempos de ejecución se pueden analizar los problemas de rendimiento e intentar solucionar tales problemas [10].

Aún en el caso de las mediciones en una misma computadora (usualmente en el contexto de un sistema con un único procesador), es deseable que el registro de tiempos no influya en el tiempo de ejecución de la misma. En todos los casos, se necesitan resoluciones de reloj acordes a los tiempos que se deben medir en las aplicaciones. Usualmente, los métodos provistos por el sistema operativo no son apropiados [21]. Por otro lado, los métodos y/o herramientas provistas por los lenguajes dependen del sistema operativo y, por lo tanto, resultan inadecuados en muchos casos también.

En el caso de procesamiento en una arquitectura distribuida, 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 conlleva la necesidad de sincronizar los relojes de las diferentes computadoras que se utilizan [5] [17] [18]. Sería deseable que esta tarea de sincronización se lleve a cabo fuera del tiempo en que se ejecute el programa que se está monitorizando, y conociendo el tipo (o al menos magnitud) de error con que se sincroniza.

Por otro lado, es deseable que la sincronización se lleve a cabo sin la 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 y el sistema de medición existentes en cada sistema de cómputo.

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

- Pueda ser usada inicialmente en un cluster de PCs, 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.
- Que 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.

# 2 DESCRIPCIÓN DEL PROBLEMA Y EXPERIMENTOS ASOCIADOS

Se han estudiado tanto los algoritmos básicos como las implementaciones existentes. En este sentido, se cuenta con una amplia cantidad de información tanto de los algoritmos como de las implementaciones. Como requisito previo, normalmente se establece que cada computadora cuente

con un oscilador físico de frecuencia constante (al menos en términos ideales). A partir de este oscilador físico se derivan los relojes lógicos que son los que se sincronizan [8] [11] [16]. En todos los casos, lo que se tiende a resolver son las diferencias de [1] [3] [12]:

1. Referencia fija en el tiempo a partir de la cual se contabiliza el tiempo en cada computadora. Aunque normalmente es constante, es relativamente difícil establecer con precisión su valor.
2. Frecuencia entre los relojes de las computadoras que se sincronizan. En algunos casos, se suelen incluir en este punto las diferencias de las variaciones de los relojes, dado que los relojes no necesariamente tienen frecuencia constante a lo largo del tiempo.

La Fig. 1 muestra dos computadoras, cada una de ellas con su oscilador, que a su vez tiene sus propias características.

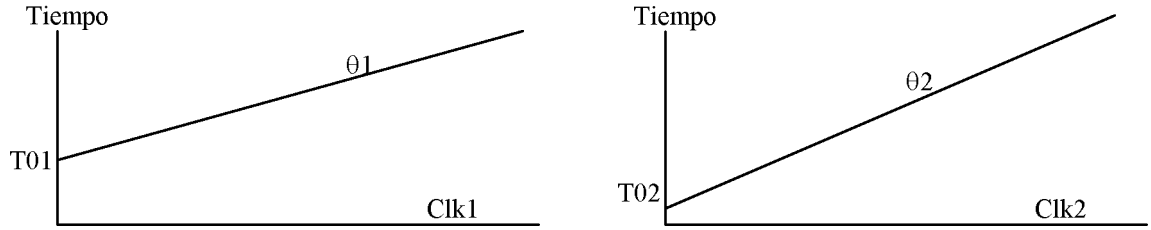


Figura 1: Relojes de Dos Computadoras.

En la Fig. 1, el instante de tiempo T01 es la referencia al tiempo real en una computadora y T02 es la referencia de tiempo equivalente en la otra computadora. La pendiente θ1 corresponde al oscilador de una computadora y θ2 es la correspondiente pendiente del reloj de la otra computadora.

Una de las formas más sencillas e intuitivas de sincronizar dos computadoras consiste en determinar el reloj de una de ellas en función del reloj de la otra teniendo en cuenta algún tipo de interconexión entre ellas. La Fig. 2 muestra esquemáticamente esta forma de sincronización, donde el reloj de una computadora se puede establecer a partir del reloj de la otra teniendo en cuenta el tiempo de las comunicaciones de sincronización entre las dos: t2-t1 (el *tiempo de ida y vuelta o roundtrip* de un mensaje).

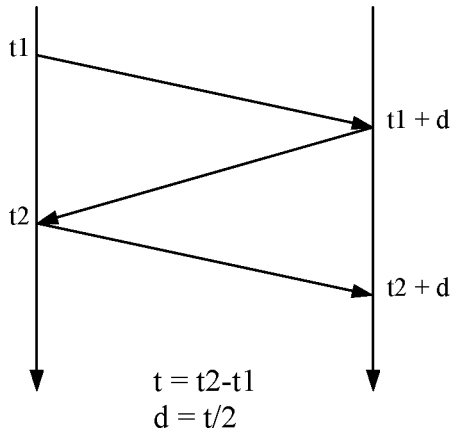


Figura 2: Sincronización Básica.

De acuerdo con la Fig. 2, el tiempo en la computadora que recibe el mensaje enviado en t1

solamente tiene que definir su propio reloj lógico como t1+d. De la misma manera, el tiempo lógico al recibir el mensaje enviado en el instante t2 debería ser t2+d. En todos los casos, d es la mitad del tiempo de *roundtrip* (o ida y vuelta) de un mensaje y hace referencia a un único reloj físico o lógico, evitando de esta manera los errores por las diferencias entre los relojes de las diferentes computadoras. Implícitamente, se está asumiendo que el tiempo de *ida* de un mensaje de una computadora a otra es igual al tiempo de *vuelta* del mismo mensaje entre el mismo par de máquinas. Los mensajes involucrados normalmente se denominan mensajes de sincronización y son de longitud mínima, dado que no es necesario transportar muchos datos de una computadora a otra para llevar a cabo la sincronización.

### 2.1 Acceso al Reloj Local de cada Computadora

La forma de hacer referencias al reloj físico y/o a los relojes lógicos con la mínima sobrecarga normalmente se basa en evitar llamadas al sistema operativo para, al menos, evitar el consiguiente cambio de contexto. Para ello se utilizaron instrucciones de ensamblador para consultar el registro correspondiente de la CPU utilizando la instrucción RDTSC (ReaD Time Stamp Counter). En este contexto, se han llevado a cabo mediciones de los ciclos de CPU invertidos en una llamada al sistema como gettimeofday y la utilizada por la alternativa presentada (con RDTSC) en diferentes computadoras. A modo de ejemplo, en la Tabla 1 se muestran los valores obtenidos en ciclos de CPU en dos computadoras, cuyas características se dan en la Tabla 2.

**Tabla 1: Ciclos de CPU de gettimeofday y RDTSC**

| PC | gettimeofday | RDTSC | RDTSC + 1 división PF |
|---|---|---|---|
| PII266 | 402 | 25 | 236 |
| XP1600 | 615 | 66 | 346 |

La primera columna de la Tabla 1 identifica la computadora (de la que se dan todos los detalles en la Tabla 2), la segunda columna muestra los ciclos de CPU necesarios para una llamada a gettimeofday, la tercera columna muestra los ciclos de CPU necesarios para una operación RDTSC, y la última columna muestra los ciclos de CPU necesarios para convertir los ciclos de CPU en microsegundos (usando la frecuencia del oscilador en la división, se podría disminuir haciéndolo en lenguaje de ensamblador). La primera columna de la Tabla 2 identifica la computadora, la segunda columna su CPU, la tercera columna la frecuencia del reloj del sistema y la última columna el sistema operativo utilizado en cada una de las computadoras.

**Tabla 2: Detalles de las PC de la Tabla 1**

| PC | CPU | Frecuencia (MHz) | Sistema Operativo |
|---|---|---|---|
| PII266 | Intel Pentium II | 266 | Linux 2.4.20-8 |
| XP1600 | AMD Athlon XP | 1600 | Linux 2.4.18-14 |

Nótese que, por un lado las cantidades de ciclos de CPU de la llamada al sistema gettimeofday y de RDTSC que se muestran en la Tabla 1 son significativamente diferentes (con su consiguiente sobrecarga *solamente* para medición/registro de tiempo).

### 2.2 Tiempos de Demora de las Comunicaciones de Sincronización

En cuanto a la sincronización, el tiempo d observado en la Fig. 2 es variable debido a diversos motivos, presentando una curva de distribución de frecuencias del tipo mostrada en la Fig. 3 [3].

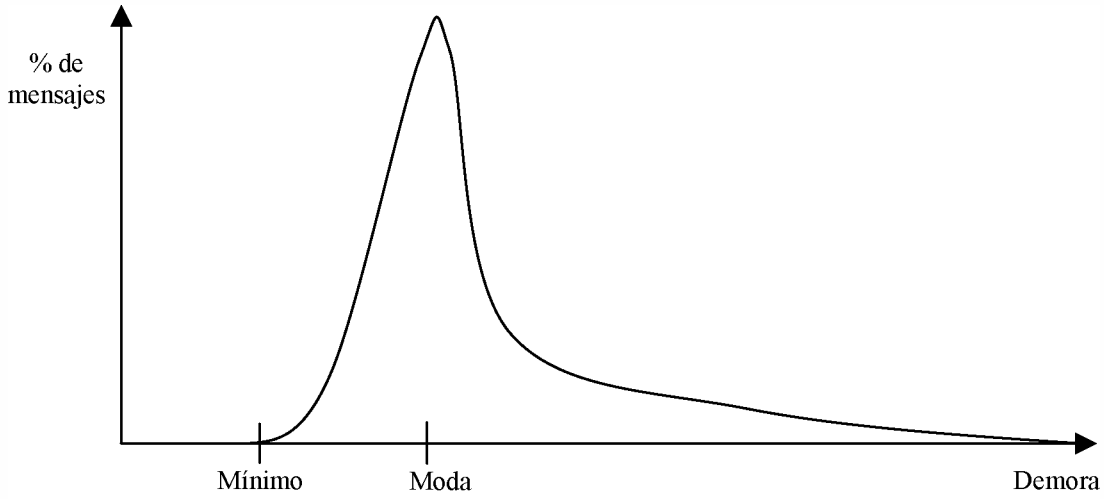


Figura 3: Distribución *Típica* de los Tiempos de Mensajes/Comunicaciones.

De acuerdo con la Fig. 3, la mayoría de los mensajes arribarán a destino con un tiempo *moda* de retraso. Se podrían considerar válidos los valores que están entre el tiempo mínimo y alrededor del tiempo *moda*. Uno de los primeros problemas detectados es que existe una asimetría entre el tiempo de ida y el de vuelta, con lo que la demora (d en la Fig. 2) será distinta del tiempo de *roundtrip*/2. Esto implica que para iguales tiempos de *roundtrip* pueden haber tiempos de ida diferentes.

Estas variaciones llevan a tener un margen de error en la sincronización. En la búsqueda de una cota, se analizó este tiempo descomponiéndolo en las etapas que se muestran en la Fig. 4, que se verifican tanto en la ida como en la vuelta de una comunicación [9]:

- Tiempo de envío: es el necesario para la construcción del paquete en las distintas capas hasta alcanzar la capa de acceso al medio. Es variable en función del estado del sistema operativo (context switch, scheduling, etc.). Podría reducirse en un sistema operativo de tiempo real.
- Tiempo de acceso al medio: depende del hardware de interconexión. En el caso de Ethernet, dependerá del tráfico en la red y en general es aleatorio. Si se usan switches para interconectar las máquinas se vuelve más determinístico y acotado.
- Tiempo de transmisión: depende de la velocidad de la placa de red y de la longitud del mensaje. En general es determinístico.
- Tiempo de propagación: en una red local será despreciable ya que es el tiempo necesario para que los datos recorran el cable entre las placas de red y el switch de interconexión (que en general existe, aunque no necesariamente siempre es parte del cableado). En general es determinístico.
- Tiempo de recepción M. (placa de la Máquina destino): es el tiempo necesario para llegar a los buffers o la memoria de la placa de interconexión. En general es determinístico.
- Tiempo de recepción P. (Proceso destino): es el necesario para desarmar el paquete en las distintas capas hasta ser entregado a la aplicación. Depende del sistema operativo como el tiempo de envío.

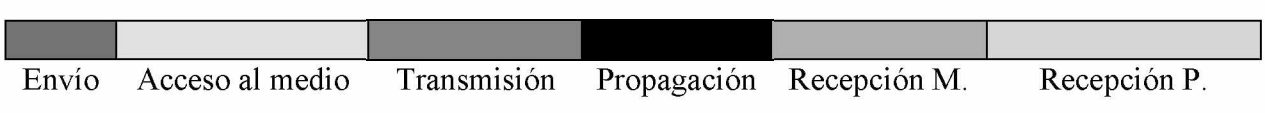


Figura 4: Tiempos Involucrados en el Envío y Recepción de un Mensaje.

Para reducir la varianza en estos tiempos, se ha hallado la conveniencia de:

- Realizar la sincronización fuera de todo proceso (es decir sin *competir* con otros procesos), para evitar variaciones en tiempos de envío.
- Tomar el tiempo con código sin llamadas al sistema operativo, evitando la sobrecarga del mismo (básicamente de los *context switches* involucrados).
- La utilización de redes con *switches* de interconexión de las máquinas, evitando las variaciones en los tiempos de acceso al medio.

Una vez tomados los recaudos necesarios para evitar la varianza, previamente a la sincronización se lleva a cabo una ráfaga de intercambio de mensajes, de los cuales se mide el promedio, el mínimo y el máximo, de tal manera de obtener una caracterización de los posibles errores cometidos al emplear la herramienta. Para el cálculo del promedio se eliminan los valores alejados en más de 2 veces del tiempo mínimo (que, además, son muy poco probables). Para las computadoras detalladas en la Tabla 2 interconectadas por una red local Ethernet de 100 Mb/s con switch de interconexión y sin uso exclusivo de la red, los valores totales (d, de la Fig. 2) obtenidos son los que se muestran en la Tabla 3.

**Tabla 3: Tiempo de Roundtrip/2 sin Uso Exclusivo de Red 100 Mb/s**

| Promedio | Mínimo | Máximo |
|---|---|---|
| 84µs | 76 µs | 91 µs |

Los valores menores al promedio de la Tabla 3 son, también en promedio, de 8µs menos y la diferencia promedio de los valores mayores es de 15µs. Con lo que las sincronizaciones tienen involucrados los errores relacionados con estas diferencias también. Por otro lado, con computadoras de mayor capacidad interconectadas por una red local Ethernet de 100 Mb/s con uso exclusivo y un switch de interconexión, los valores obtenidos son los que se muestran en la Tabla 4.

**Tabla 4: Tiempo de Roundtrip/2 con Uso Exclusivo de Red 100 Mb/s**

| Promedio | Mínimo | Máximo |
|---|---|---|
| 54µs | 53 µs | 56 µs |

En el caso de los valores de la Tabla 4, las diferencias promedio son de 1µs y 3µs mínimo y máximo respectivamente. La Tabla 5 muestra los detalles de las computadoras con las que se obtuvieron los valores de la Tabla 3.

**Tabla 5: Detalles de las PC de la Tabla 4**

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

### 2.3 Determinación de la Pendiente de los Osciladores

El valor de la frecuencia de reloj surge de obtener el valor del registro del oscilador de la CPU en dos instantes de tiempo t1 y t2, con lo que:

$$MHz = (tsc2 - tsc1) / (t2 - t1) \quad (1)$$

donde tsc1 y tsc2 son los valores del registro del oscilador de la CPU en los instantes de tiempo t1 y t2 respectivamente y, además, t1 y t2 son obtenidos a partir de una referencia de tiempo de una de las máquinas intervinientes en el proceso a partir de su reloj local con una llamada al sistema del tipo gettimeofday. Este método aunque es sencillo y claro no está libre de errores cuando se trata de medición de tiempo. Además de la sobrecarga de la llamada al sistema operativo que se menciona antes, los tiempos proporcionados por gettimeofday pueden tener un error y además habría un error en la variación de los tiempos de los mensajes involucrados para *transportar* las referencias de tiempo de una máquina a otra.

En todas las computadoras en las que se han hecho experimentos, que incluyen a las que se describen en la Tabla 2 y en la Tabla 5, todas las llamadas al sistema operativo (gettimeofday) tienen una resolución y sobrecarga (sumadas) del orden de 1μs en las computadoras con relojes del orden de GHz. Por lo tanto, el tiempo real tr1 involucrado, es decir teniendo en cuenta este error, tget < 2μs (porque son dos mediciones de tiempo, t1 y t2), sería

$$tr1 = t2 - t1 + tget \quad (2)$$

Por otro lado, habría que incluir el error determinado por la variación de tiempos de mensajes, td que sería la diferencia entre la moda y el mínimo. Con lo cual se tendría

$$tr2 = t2 - t1 + tget + td \quad (3)$$

Y, por lo tanto, para que el error total sea despreciable, t2 - t1 >> tget + td. Si, por ejemplo se tiene un td entre 0 y 8μs (valores similares a los derivados de la Tabla 3 y la Tabla 4) y se consideran datos útiles/aceptables con un error de $1/10^6$, se tendría que

$$t2 - t1 = 10^6 \times (2\ \mu s + 8\ \mu s) \quad (4)$$

Por lo tanto, dado que t2 - t1 = $10^6$ x (2 μs + 8 μs) = 10 segundos, se necesitaría un intervalo de tiempo de 10 segundos o más para que el error sea de $1/10^6$ (una millonésima) o menor. De esta manera, el error se puede hacer tan pequeño como sea necesario/útil haciendo mayor el intervalo de tiempo durante el cual se toman las referencias t1 y t2. Se debe notar que este método es usado para determinar los valores de MHz de cada CPU con error acotado/conocido.

# 3 RESULTADOS OBTENIDOS DE SINCRONIZACION

Se elaboró una biblioteca mínima en cuanto a la medición de tiempos de cómputo en un mismo sistema (usualmente en una máquina monoprocesador), para el sistema operativo Linux en PCs (procesadores Intel y compatibles) en las cuales se puede hacer referencia al contador de ciclos del oscilador. En lo referente a la sobrecarga, los datos preliminares muestran resultados altamente satisfactorios, dado que la sobrecarga no excede las decenas de ciclos de reloj. En este sentido, ya es posible la instrumentación de código y el análisis de las aplicaciones que se resuelven en un mismo sistema o máquina. Por otro lado, la precisión también es satisfactoria: del orden de los microsegundos. A modo de ejemplo, la Tabla 6 muestra la sobrecarga y la precisión de la rutina usada para registrar tiempos en una computadora de las mostradas en la Tabla 5, comparada con la llamada a gettimeofday. Se debe notar que la sobrecarga es igual a la precisión dado que no hay cambios de contexto al incluir la biblioteca en el binario de la aplicación. También a partir de la Tabla 6 se puede observar que la sobrecarga del registro de tiempos con la lectura de ciclos de CPU

es de poco más de 1/3 de la que se obtiene con la llamada al sistema operativo gettimeofday. Cabe aclarar que para la prueba , se provocó el desalojo de gettimeofday y rdtsc de memoria caché tanto de instrucciones como de datos, que es la condición real en la que serán usados en instrumentación.

**Tabla 6: Sobrecarga y Precisión de Tiempo Local**

| PC | gettimeofday (ciclos) | RDTSC (ciclos) | Precisión RDTSC (μs) |
|---|---|---|---|
| P42400 | 7876 | 2828 | ≅1.18 |

Se está desarrollando software (básicamente una biblioteca de una cantidad reducida de funciones) para instrumentación de programas paralelos que se ejecuten en clusters de PCs. Esta biblioteca tiene los lineamientos dados antes, con énfasis en la resolución del orden de los microsegundos, la mínima sobrecarga de procesamiento y el desacople de la red de interconexión. La Tabla 7 muestra los errores mínimo y máximo de sincronización con las computadoras detalladas en la Tabla 5 (P42400) interconectadas con una red Ethernet de 100 Mb/s de uso exclusivo.

**Tabla 7: Errores de Sincronización Medidos**

| PC | Red | Mínimo | Máximo |
|---|---|---|---|
| P42400 | 100 Mb/s, uso exclusivo | 0 μs | 5 μs |

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 la resolución dada en la Tabla 6: aproximadamente 1.18 μs.

# 4 CONCLUSIONES Y TRABAJOS FUTUROS

Como mínimo, se ha llegado a tener una biblioteca muy sencilla y de muy baja sobrecarga para la instrumentación de programas secuenciales. En el contexto de instrumentación de aplicaciones distribuidas, también se ha llegado a una biblioteca sencilla que no solamente permite registrar tiempos del orden de los microsegundos sino que, además, mantiene los relojes de las máquinas sincronizados de manera tal que se tenga una única referencia de tiempo.

Además, se están llevando a cabo pruebas sobre los sistemas de relojes distribuidos empleados en este momento tales como NTP (Network Time Protocol) y DTS (Distributed Time Service), con el fin de analizar sus ventajas y desventajas. También un análisis de los sistemas que utilizan hardware especializado, como para tener un marco de referencia más amplio [6]. En todos estos casos, la idea final es contar con un conjunto de programas del estilo de los *benchmarks* para que provean automáticamente la caracterización del sistema de sincronización elegido/utilizado. Como mínimo, el objetivo es contar con una metodología de caracterización de las herramientas, bibliotecas, y/o protocolos de sincronización que incluso pueda ser aplicada a la nueva biblioteca que se desarrolle, mencionada antes.

Por otro lado, también es importante analizar y, más específicamente, cuantificar las características de NTP (Network Time Protocol) para ser comparado con la biblioteca que se propone en este artículo. Esta cuantificación se refiere principalmente a:

- Resolución posible del reloj sincronizado. Aunque NTP es paramétrico esto no significa *a priori* que se puede obtener una resolución arbitraria cualquiera.

- Relación entre la resolución definida y la sobrecarga en la red de comunicaciones.
- Relación entre la resolución definida y la sobrecarga en la pila de protocolos TCP/ UDP/IP.
- Relación entre la resolución definida y la sobrecarga en el uso de CPU y la jerarquía de memoria.

Todo este análisis es básicamente experimental [15] [18] [19], 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.

Todavía está pendiente investigar el comportamiento en términos de escalabilidad de la sincronización implementada. Usualmente la sincronización se da entre dos máquinas y en el caso particular de NTP se lleva a cabo con el modelo cliente/servidor. Es claro que cualquier tipo de centralización (en el servidor, por ejemplo) tiene sus inconvenientes de escalabilidad y al menos debería ser posible su cuantificación. En este contexto específico es muy interesante la posibilidad de sincronización utilizando mensajes broadcasts con su consiguiente ahorro de comunicaciones punto a punto.

Como extensiones futuras, siempre es deseable la sincronización externa de los relojes [8]. Este paso está muy ligado también a la posibilidad de utilizar más de un cluster de computadoras para cómputo paralelo y en este contexto la sincronización de los relojes va más allá del análisis de rendimiento con el objetivo de optimizarlo.

También es necesario investigar en particular hasta qué punto las referencias a los relojes/osciladores físicos son portables o al menos tienen funciones u operaciones equivalentes en diferentes plataformas de hardware.

Como paso posterior, se espera extender la biblioteca para su uso en Internet y múltiples clusters para cómputo paralelo [23]. Quizás en este punto se deban redefinir algunas características de la herramienta o biblioteca, tal como la capa de transporte de los mensajes con los cuales se implementa la sincronización.

## REFERENCIAS


[1] Coulouris G., Dollimore J., Kinberg T., Sistemas Distribuidos. Conceptos y Diseño, 3ª edición. Pearson Educación, 2001. ISBN: 84-7829-049-4.

[2] Cristian F. "Synchronous and Asynchronous Group Communication", Comm. ACM, Vol.39, No. 4, April 1996, pp.88-97.

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

[4] Cristian F., Fetzer C. "The Timed Asynchronous Distributed System Model", IEEE Transactions on Parallel and Distributed systems, June 1999, pp. 603-618.

[5] Cristian F., Fetzer C., "The Time Asynchronous Distributed System Model", IEEE Transactions on Parallel Systems, June 1999, pp. 603-618.

[6] 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 (HotNets-1), Princeton, New Jersey, October 2002.

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

[8] Fetzer C., Christian F., "Integrating External and Internal Clock Synchronization", June 1996.

[ag2-9] Ganeriwal, S., Kumar, R., Srivastava, M.B., "Timing-sync protocol for sensor networks", Proc. of the 1st Int. Conf. On Embedded Networked Sensor Systems, Los Angeles, CA, USA (2003) p.138--149

[10] Grove D. A.. "Performance Modelling of message-passing parallel programs", May 2003.

[11] 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 Objet-oriented Real-time distributed Computing, Washington, D.C., April 2002, pp. 263-270.

[12] Mills D. L. "Measured performance of the Network Time Protocol in the Internet System", ACM Computer Communication Review 20, Jan. 1990. pp. 65-75.

[13] Mills D. L., "Modelling and analysis of computer network clocks", Electrical Engineering Department Report 92-5-2, University of Delaware, May 1992.

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

[15] Mills D. L. "Measured Performance of the Network Time Protocol in the Internet System", ACM Computer Review 20, 1, January 1990, pp.65-75.

[16] Mills D.L., "Internet time Synchronization: the Network Time Protocol", IEEE trans. Communications COM39, October 1991, pp. 1482-1493.

[17] Mills D.L., "Network Time Protocol (Version 3) specification, implementation and analysis", DARPA Networking Group Report RFC-1305, University od Delaware, March 1992.

[18] Mills D. L., "A Brief History of NTP Time: Confessions of an Internet Timekeeper", ACM Computer Communications Review 33, 2 (April 2003), pp 9-22.

[19] Mills, D.L., "Unix kernel modifications for precision time synchronization", Electrical Engineering Department Report 94-10-1, University of Delaware, October 1994.

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

[21] Rubini A., Corbet J., "Linux Device Drivers 2nd Edition", ISBN 0-59600-008-1. June 2001.

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

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