Sincronización de Relojes para Evaluación de Rendimiento: Experiencias en un Cluster Utilizado para Cómputo Numérico

Fernando L. Romero, Armando E. De Giusti, Fernando G. Tinetti*

III-LIDI, Facultad de Informática, UNLP
50 y 120, 1900, La Plata, Argentina
fromero@lidi.info.unlp.edu.ar, degiusti@lidi.info.unlp.edu.ar, fernando@info.unlp.edu.ar

Resumen

El presente trabajo es una evaluación del desempeño del conjunto de herramientas formado por la biblioteca para registros de tiempos en una computadora y una biblioteca para sincronización de relojes de computadoras de un cluster. Se trata de herramientas desarrolladas para medir tiempos tanto de ejecución como de comunicación en ambientes distribuidos, inicialmente de red local. La evaluacion se realiza a través de dos experimentos. El primer experimento es respecto a medida de tiempos de la propia herramienta para evaluar la escalabilidad para su uso en clusters de gran cantidad de computadoras. En el segundo se realiza la instrumentación de una multiplicación de matrices, en base a un algoritmo que emplea comunicaciones broadcast. Se reportan los datos de escalabilidad del primer experimento y, en el caso del segundo experimento, los tiempos en los que se producen las comunicaciones broadcast y las conclusiones de evaluar tiempos de multiplicación de matrices. Se presenta una reseña de los trabajos anteriores como introducción al tema.

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

Abstract

This paper presents an assessment of the performance of a toolset formed by the library to record times on a computer and a library for the synchronization of the clocks of the computers in a cluster. These are tools developed to measure both execution and communication times in distributed environments, initially within a local network. The assessment is carried out by means of two experiments. The first experiment is related to the measurement of time with the tool itself to assess its scalability in order to be used in clusters with a large number of computers. The second experiment involves the instrumentation of matrix multiplication based on an algorithm that uses broadcast communications. Scalability results are reported for the first experiment, whereas the times required for broadcast communications are reported for the second experiment, as well as the conclusions drawn after assessing matrix multiplication times. A summary of previous works is presented as an introduction.

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

* Investigador Asistente, Comisión de Investigaciones Científicas de la Provincia de Buenos Aires.

1 Introducción

Desde hace tiempo las computadoras poseen uno o varios dispositivos de hardware destinados a la medición del tiempo (timers), y relojes que proporcionan la hora, tales como el reloj de tiempo real [13] [21]. Las referencias de tiempo son esenciales a la hora de resolver problemas tales como el ordenamiento de eventos (ej: envío y recepción de correo electrónico, eventos dentro de transacciones, inicio de procesos en tiempo real, etc.) como para tareas propias del sistema, tales como planificación de procesos. Estos relojes permiten la medición de intervalos de tiempo. Dicha tarea cobra especial importancia a la hora de optimizar rendimiento tanto en sistemas monoprocesador como en sistemas paralelos y distribuidos [12] [4] [8]. En todos los casos, las aplicaciones con fuertes requerimientos de cómputo, procesamiento y comunicaciones son las que también requieren la optimización para el máximo aprovechamiento del hardware disponible. A partir de la monitorización de los tiempos de ejecución y comunicaciones se pueden analizar los problemas de rendimiento e intentar solucionarlos [4].
En el caso de sistemas distribuidos, esta monitorización requiere la existencia de una hora uniforme en todo el sistema. Esto se consigue a través de sincronizar los diferentes relojes que se utilizan en las mediciones [7]. La sincronización de relojes implica:

- Que los relojes tengan una referencia de hora común u hora inicial a partir de la cual actualizan su valor.
- Que la frecuencia de actualización sea congruente en todos los relojes. O sea que un tiempo después las horas actualizadas sean las mismas en todos los relojes, o que se mantenga la diferencia con que se ajustó la hora inicial lo mejor posible.

Dicha tarea, en un sistema distribuido no es trivial [19]. Se podría dividir dicho problema en tres etapas:

- Adquisición de las referencias de hora y frecuencia a partir de algún reloj maestro, en una máquina servidora de hora.
- Comunicación de la referencias a los computadoras donde residen los demás relojes.
- Actualización de los parámetros de hora y frecuencia de los relojes de los clientes.

Se debe aclarar que la actualización de parámetros no se realiza sobre el reloj de hardware propiamente dicho sino sobre variables en memoria, que guardan las diferencias tanto de frecuencia como de toma de valor inicial de los relojes de los clientes y del servidor, con lo que la actualización implica una adquisición de hora, comparación con la referencia y almacenamiento de los parámetros de sincronización en la memoria. Del análisis anterior, se deriva que los principales problemas a analizar son la adquisición de hora a partir de un reloj local, tanto en el servidor como en los clientes, y el problema de la comunicación de referencias de hora.
Referido a los relojes utilizados en los sistemas de cómputo a fin de realizar mediciones de tiempo de ejecución de procesos y de comunicaciones, es deseable que el registro de tiempos sea despreciable frente a los tiempos de ejecución de los mismos. Por otro lado, el requerimiento de resolución en las medidas de dichos tiempos ha crecido, debido a la evolución del hardware. Esta evolución ha llevado al incremento de las velocidades de los procesadores, la disminución de las latencias en las comunicaciones y el aumento del ancho de banda en las mismas. En la actualidad, para la medición de los tiempos implicados, se requieren resoluciones en el orden de los microsegundos. Los métodos provistos por el sistema operativo no son apropiados [14] debido no solo a problemas de resolución sino también por el nivel de interferencia (Ej: llamadas al Sistema Operativo). Los métodos y/o herramientas provistas por los lenguajes heredan este problema ya que dependen del sistema operativo. La solución es un mejor uso de las nuevas características del hardware, por ejemplo la utilización de la instrucción RDTSC (ReaD Time Stamp Counter) [13] [21].

Sería deseable que la 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. Es requerimiento de este trabajo que dicha sincronización se lleve a cabo sin la necesidad de incluir hardware adicional al del sistema, con lo que las comunicaciones deberán utilizar la red de interconexión existente entre computadoras. Asimismo, como sistema de medición, aprovechar el hardware de cada sistema de cómputo, sin agregados ni cambios. La herramienta de instrumentación que se utiliza en este trabajo cumple que [18]:

- Puede ser usada inicialmente en un cluster de PC's, con la posibilidad de ser extendido a clusters en general y luego a plataformas distribuidas aún más generales.
- Es de alta resolución, es decir que se puede utilizar para medir tiempos cortos, del orden de microsegundos.
- No altera el funcionamiento de la aplicación bajo prueba, o la alteración es mínima y conocida por la aplicación.
- Utiliza en forma predecible la red de interconexión. Más específicamente, se puede determinar, desde la aplicación, el uso que se le dará a 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.

A estas características se le agrega que los errores que se cometen tanto durante el proceso de sincronización como durante las mediciones están caracterizados y pueden ser conocidos por el usuario.

2 Trabajos Previos Relacionados

Se estudiaron tanto los algoritmos básicos como las implementaciones existentes [1]. Como requisito previo, se establece que cada computadora cuente con un reloj físico de frecuencia más o menos constante. En general, estos relojes utilizan como estabilizador del oscilador un cristal de cuarzo. Presentan valores de variabilidad de frecuencia que van desde 10^{-4} a 10^{-6} [15]. A partir de este reloj físico se derivan los relojes lógicos que son los que se sincronizan [3] [6] [10]. En todos los casos, lo que se tiende a resolver son las diferencias de [16] [17]:

1. Referencia fija en el pasado a partir de la cual se contabiliza el tiempo en cada computadora.
2. Frecuencia entre los relojes de las computadoras que se sincronizan.

Una vez realizada la sincronización mínima entre las computadoras se debe investigar el comportamiento del algoritmo utilizado en términos de escalabilidad. Usualmente la sincronización se da entre dos máquinas. La propuesta inicial de Cristian [2][3], de contar con un servidor que proporcionara una hora ajustada a un determinado estándar (sincronización externa) y clientes sincronizando con este servidor, sirvió de base para el desarrollo de protocolos más complejos en los que se consideraron otros aspectos del problema. Se puede citar a NTP (Network Time Protocol) [7] [9] y el algoritmo de Berkeley [5] como ejemplos, siendo este último un ejemplo de sincronización interna. En cuanto a ordenación de eventos sin un estándar en la hora, el algoritmo de Lamport es un ejemplo de otra orientación para solucionar el problema de sincronización. 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. Referido a análisis y experimentación de sistemas existentes, se han llevado a cabo pruebas con NTP y con el Algoritmo de Berkeley específicamente con su implementación en Linux, el timed (**time d**aemon). También se analizó el algoritmo de Lamport. Dichas pruebas están en [18]. Aquí se presenta un resumen de los aspectos más importantes de los diferentes algoritmos.

Uno de los primeros, conocido como algoritmo de Cristian [2], propone sincronizar relojes a partir de un servidor proveedor de la referencia de hora, a la cual llama hora verdadera. Se realiza previamente a la sincronización una estadística sobre los tiempos de ida y vuelta de mensajes de comunicación entre clientes y servidor. Los clientes solicitan la hora al servidor, el cual la devuelve. Simultáneamente el cliente mide el tiempo de ida y vuelta, descartando toda referencia que no se realice en tiempo de transmisión igual a la moda. Se dice que este método es estadístico ya que se ajustan los valores de error para garantizar que la referencia se conseguirá en tiempo de moda luego de una cantidad de retransmisiones menor a un cierto número, o sea que la probabilidad de que ello ocurra es cercana a uno. De este algoritmo se tomo la idea de un servidor y cliente, modificándose para que la información tanto de tiempos de ida y vuelta y de errores se centralice en el servidor.
Otro de los algoritmos, concretado en una implementación, NTP, corrige dos tipos de diferencias (*offsets*) para sincronizar dos o más computadoras:

- ***Offset* de hora:** Va realizando ajustes de acuerdo a que la diferencia respecto de la referencia sea 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 está 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.

Se rescata de NTP esta forma de corregir los relojes en hora y frecuencia. La herramienta de sincronización desarrollada en este trabajo utiliza estas dos formas de corrección. Un defecto de NTP es el poco control que se tiene sobre el error en la sincronización dado por la medida de distancia de sincronización.
A diferencia de NTP, el algoritmo de Berkeley tiene un proceso (o computadora) *Master* que se encarga de definir un tiempo sincronizado a partir del tiempo local de todos los demás procesos (o computadoras). De manera periódica, el Master realiza las siguientes tareas:

1. Requiere la hora local de cada uno de los demás procesos.
2. Calcula el promedio de los valores recibidos (descartando los que difieran demasiado).
3. Informa a los demás procesos el cambio que deben realizar para estar sincronizado.

En el cálculo de cada nuevo valor de tiempo a utilizar (con el promedio de los tiempos recibidos) no se tiene en cuenta el tiempo de comunicaciones necesario para las transferencias de datos hacia y desde el proceso *Master*.
Este algoritmo fue uno de los primeros incluidos en las distribuciones más conocidas de Linux, aunque algunas de ellas ya no lo incluyen más. Siempre es posible (y sencillo, de hecho) instalarlo en Linux, donde básicamente se pone en marcha el proceso timed a nivel de sistema operativo. En todos los casos la mejor resolución obtenida es de 1 milisegundo, a la que se llega bastante más rápidamente que con NTP. En principio, no cumple con uno de los objetivos originales de este trabajo que es llegar al orden de los microsegundos. Se rescata de este algoritmo que el Master controla en todo momento el proceso de sincronización y concentra la información, relativa a parámetros de sincronización y errores.

3 Evaluación de la Biblioteca de Sincronización

Los valores de hora en dos computadoras 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 [12] [11]. 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 [18] 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. Esta estimación, si bien puede no ser del todo cierta, el error cometido sería pequeño dado que la tarea llevada a cabo es simétrica, siendo las diferencias de ida y vuelta causadas por planificación de CPU, interrupciones de hardware y demás interferencias producidas por el propio sistema operativo. Mejoras al respecto se obtendrían utilizando un Sistema Operativo de Tiempo Real, lo cual sería poco práctico en un cluster dedicado a correr aplicaciones paralelas. No se descarta implementar solo el servidor de hora con un Sistema Operativo de Tiempo Real.
Teniendo estos tiempos caracterizados, se restan de los valores de ida y vuelta, divididos por dos, ya obtenidos en las estadísticas de moda realizadas al inicio de la aplicación. La biblioteca se implementó en un cluster de 16 máquinas con las características dadas en la Tabla 1.

Tabla 1: Detalles de las PC de la Tabla 4

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

La Tabla 2 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 en durante 20 mediciones en cada caso. Se reportan los valores máximo y mínimo en cada caso y expresados en microsegundos. Para el cálculo del error se considera que los mensajes con referencias solo 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. En el máximo, poco probable, y teniendo en cuenta que el tiempo local hasta los buffers es constante e igual a 22 en todos los experimentos, la posible asimetría se calcula a partir de restar los valores de moda menos 22. 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:

Tabla 2: Valores de tiempos de comunicación.

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

Por otro lado, en un caso más probable, la asimetría no debiera ser implicar 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 de hacer 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.
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, con lo que se sincronizó los relojes de 2 a 16 computadoras, obteniéndose los valores que se muestran en la Tabla 3, expresados en microsegundos:

Tabla 3: Tiempos de ejecución para 20 corridas de Sinchro en 2 a 16 máquinas.

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Mínimo	37972	38414	39902	41544	40819	39081	38596	41030	41910	40420	37522	39235	38525	36983	41774
Máximo	43362	44627	46933	51408	52908	50845	48233	49997	59397	52112	53436	51453	56029	51962	52791
Promedio	40475	41513	42444	44613	45314	44654	43585	44365	47353	44368	45433	45235	47645	47120	48312

En forma gráfica se puede apreciar la evolución de estos tiempos en la Fig. 1.

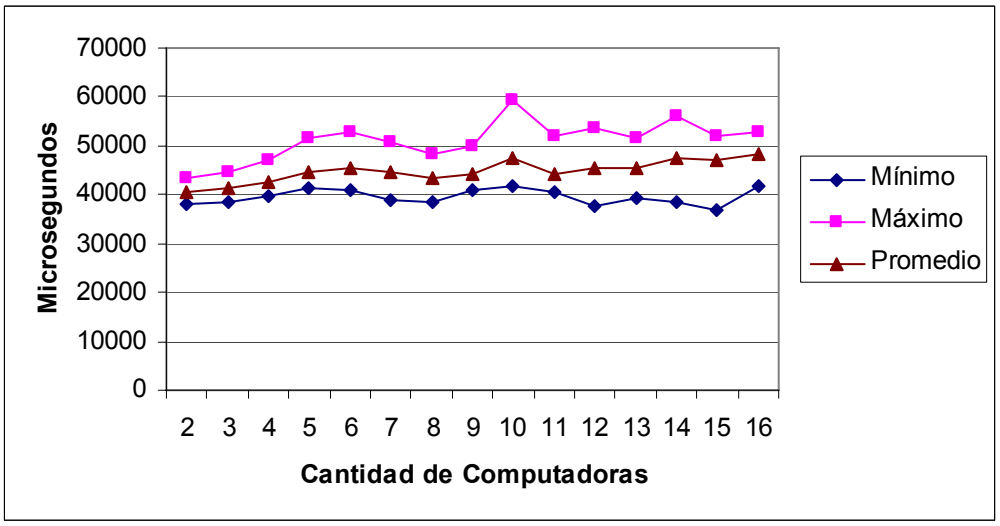


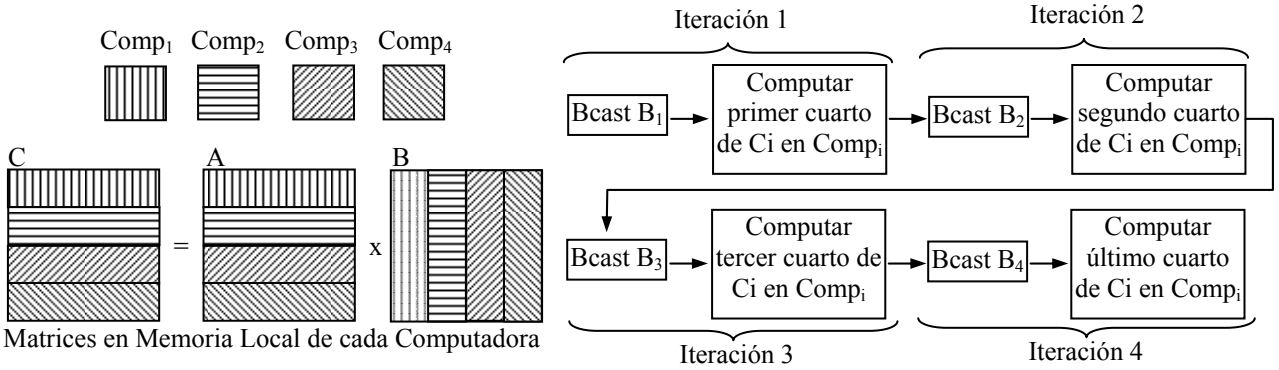
Figura 1: Tiempos de ejecución para 20 corridas de Sinchro en 2 a 16 máquinas.

Estos resultados permiten concluir que la escalabilidad de este algoritmo, a pesar de basarse en un modelo cliente-servidor, no representa mayores incovenientes en lo tamaños típicos de cluster. 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 preliminares al proceso de medición, y absolutamente aislados del mismo. Es de destacar 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.
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 [16], con lo que 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 [16] se reportan para gettimeofday(), una llamada al sistema operativo, opcional al

método utilizado de la lectura del Time Stamp Counter (con RDTSC), un tiempo de entre 402 y 615 ciclos.

3 Evaluación de un Programa Paralelo en un Cluster

Dentro de las pruebas realizadas a la herramienta de sincronización se encuentra la medición de un programa que implementa la multiplicación de matrices basado en mensajes broadcast, dado en [20]. La Fig. 2 muestra el esquema de dicha multiplicación de matrices para cuatro computadoras: la Fig. 2.a) muestra la distribución de las matrices y la Fig. 2.b) 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.



a) Distribución de Datos en cada $Comp_i$ b) Comunicaciones y Cómputo en cada $Comp_i$

Figura 2: Multiplicación de Matrices en Clusters.

El proceso completo de multiplicación es iterativo, tal como se muestra en la Fig. 2.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. 2 de la porción de Ci que debe calcular $Comp_i$. La implementación se llevó a cabo en el cluster que se describe anteriormente utilizando la biblioteca de pasaje de mensajes LAM/MPI. La Tabla 4 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 $Comp_0$ 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)**, comienza antes la ejecución del mensaje broadcast de la iteración siguiente (hace la llamada a la rutina broadcast de MPI). Dado que la computadora $Comp_2$ 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 $Comp_0$. 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.

Tabla 4: Datos de Instrumentación de la Multiplicación de Matrices con Tres Computadoras.

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

Es importante remarcar que la obtención de los valores que se muestran en la Tabla 4 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 de multiplicación de matrices *original*.

4 Conclusiones y Trabajos Futuros

Se realizaron las pruebas en las que se midió el escalado de tiempos de la sincronización de relojes con resultados favorables: en el peor de los casos 560 microsegundos por cada máquina que se agrega a la sincronización, con tiempos máximos de 60 milisegundos en 16 máquinas. Queda por analizar el tiempo inicial del que se parte con el fin de reducirlo, pero en todos los casos dicho valor es de 4 décimas de milisegundo, lo cual no representa inconveniente, ya que la sincronización de

relojes se realiza fuera de los tiempos de ejecución de la aplicación, con desacople total tanto en procesamiento como en comunicaciones. Se destaca lo trivial de introducir la instrumentación y la baja influencia en los tiempos de ejecución, del orden de una a dos decimas de microsegundo por tiempo adquirido [16]. La sincronización en los relojes hace que estos tiempos provean un orden global de los tiempos de los eventos producidos en el sistema distribuido. La utilización de la instrumentación no implica más de 20 líneas de código agregado al programa de multiplicación de matrices original. El orden global de los tiempos de los eventos permite identificar con claridad problemas de rendimiento de aplicaciones paralelas, tal como se mostró en la sección anterior.
Como trabajos futuros se espera experimentar en un cluster con mayor cantidad de computadoras, inclusive entre clusters conectados a través de Internet. En otro contexto, también sería interesante probar esta biblioteca con redes locales más rápidas (1 Gb/s Ethernet) y analizar la extensión de la herramienta a computadoras con múltiples procesadores o núcleos como también la utilización en aplicaciones con mayores requerimientos de precisión y cota de error de la biblioteca de sincronización como el propio broadcast.

REFERENCIAS

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

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

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

[4] D. A. Grove. "Performance Modelling of messagepassing parallel programs", May 2003.

[5] R. Gusella, S. Zatti "An Election Algorithm for a Distributed Clock Synchronization Program" EECS Department University of California, Berkeley Technical Report No. UCB/CSD-86-275 1986

[6] K. H. Kim, C. Im, P. Athreya, "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.

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

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

[9] D. L. Mills, “Network Time Protocol (Version 3) specification, implementation and analysis”. Network Working Group Report RFC-1305, University of Delaware, March 1992, 113 pp.

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

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

[12] D. L. Mills, “Precision synchronization of computer network clocks”, ACM Computer Communication Review 24, 2 (April 1994). 28-43.

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

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

[15] S. Mishra, C. Fetzer, F. Cristian, “The Timewheel Asynchronous Atomic Broadcast Protocol”, PDPTA 1997: 1239-1258

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

[17] F. L. Romero, F. G. Tinetti, "Sincronización de Relojes en Ambientes Distribuidos" IX Workshop de Investigadores en Ciencias de la Computación,Trelew, Mayo 2007.

[18] F. L. Romero, F.G. Tinetti, "Problemas de la Sincronización de Relojes en Clusters" XIII Congreso Argentino de Ciencias de la Computación (XIII CACIC) Octubre 2007

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

[20] 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

[21] P. Work, K. Nguyen, "Measure Code Sections Using The Enhanced Timer", http://www.intel.com/cd/ids/developer/asmo-na/eng/209859.htm, October 2005.