Evaluación de la Sincronización Periódica de Relojes de Computadoras para Instrumentación

Fernando L. Romero, Armando E. De Giusti, Fernando G. Tinetti[1]
III-LIDI, Facultad de Informática, UNLP
50 y 115, 1900, La Plata, Argentina
{fromero, degiusti, fernando}@lidi.info.unlp.edu.ar

Resumen. Se presenta la evaluación de la sincronización de relojes en un cluster para contar con una referencia de tiempo común en un sistema paralelo. La finalidad es instrumentar experimentos de medición de rendimiento en dichos sistemas. Se analiza la posibilidad de resincronizar los relojes de computadoras durante el experimento de medición de rendimiento en una aplicación de procesamiento paralelo tal como una multiplicación de matrices. La evaluación se lleva a cabo con una fuerte competencia con la aplicación por la utilización de recursos. Dicha competencia por recursos necesariamente tendrá un impacto sobre los tiempos necesarios para sincronización.

Palabras clave: Sincronización de Procesos, Relojes Distribuidos, Evaluación de Sistemas Paralelos y Distribuidos, Rendimiento e Instrumentación.

1 Introducción

A partir de haber sido sincronizados, los relojes de las computadoras sufren corrimientos en sus valores de hora, los que traen aparejados errores en los tiempos relativos cuando se realizan las mediciones. En trabajos anteriores se analizó la tarea de sincronizarlos y realizar medidas de tiempo con un error acotado y conocido, con la menor intrusión posible. Como contexto de los experimentos, se desacoplaron los tiempos de ejecución de la aplicación bajo prueba y los tiempos dedicados a la sincronización. En general, se muestra por experimentación que error se incrementa desde la sincronización en adelante, lo cual permite corregir los tiempos post medición por interpolación. Ahora se analiza la posibilidad de resincronizar los relojes durante el experimento pero de una manera controlada, analizando el tiempo que agrega al experimento y en cuánto mejora el error.

Para la sincronización se utiliza el algoritmo de Cristian, por lo que se repasa su funcionamiento. Como el error aumenta a medida que pasa el tiempo a partir de la sincronización de relojes, se evalúa la resincronización después de determinado período. Entre los factores mejorables por software se ve la utilización de un Sistema Operativo de Tiempo Real, aunque sea para el caso de la máquina que actúa como servidor en la sincronización.

[1] Investigador Comisión de Investigaciones Científicos de la Provincia de Bs. As.

2 Sincronización con restricciones

La sincronización de relojes en computadoras [5] [7] [8] es compleja y presenta tópicos tales como resolución, precisión, exactitud, fiabilidad, tolerancia a fallas, interoperabilidad, sobrecarga del sistema y seguridad. Implica un compromiso entre estas variables. 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. Por ello, se plantea la *sincronización de relojes* pero *con restricciones* en los aspectos a resolver.

La sincronización se necesita como herramienta básica de instrumentación para programas paralelos, inicialmente para ser utilizada en un cluster de PC's. Debe ser de alta resolución, para medir tiempos del orden de microsegundos, sin alterar el funcionamiento de la aplicación bajo prueba, o con alteración mínima y conocida por la aplicación.

Se analizan la evolución del error de sincronización a partir de la misma, durante un tiempo de 2000 segundos. También analiza la deriva entre dos relojes en una computadora. En el caso de tener que realizar una sincronización durante un experimento, se analizan lo tiempos que lleva resincronizar sin desacoplar.

3 Análisis del error de sincronización

En el algoritmo de Cristian se propone sincronizar una red a partir de la hora en una computadora mediante el uso de mensajes para intercambiar referencias de hora entre las computadoras utilizando la red de interconexión entre las mismas. En este esquema se asume, por experimentación [1] [2] en forma estadística que:

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

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

- ρ tasa de deriva de la frecuencia del reloj local de *p*, en partes por millón.
- $H_p(t)$ valor leído en el contador del reloj de hardware de *p* en el tiempo real *t*
- *t* es definido como el más cercano a algún estándar como UTC.
- $H_p(s)$, el medido en el tiempo *s*.
- σ demora máxima en la cual un proceso es despertado por el sistema operativo para darle el uso de la CPU.

La hora será correcta respecto a *t*, si está dentro del intervalo:

$$(1-\rho)(t-s) \le H_p(t) - H_p(s) \le (1+\rho)(t-s) \qquad \textbf{(1)}$$

Esta fórmula marca los límites de deriva de un reloj de *W* unidades de tiempo, mientras σ pertenece al intervalo:

$$[t + (1-\rho)W, t + (1+\rho)W + \sigma] \quad (\mathbf{2})$$

Este método es estadístico, ya que estima que con probabilidad cercana a 1 que conseguirá un valor dentro de la banda de error *A*. El tiempo de ida y vuelta (límite superior de *A*) no puede ser mayor a

$$(t_2\text{-}t_0)(1+\rho) \quad (3)$$

Límite que se usa para determinar un tiempo máximo para la demora del mensaje en un sentido del mensaje m_2. El tiempo min surge de estadísticas sobre mensajes como la mitad del mínimo *rtt (round trip time)*.

$$\max(t(m_2)) \equiv (t_2 - t_0)(1+\rho) - \min \quad (4)$$

Y *A* estará limitado por lo valores de [min, max].

$$min(1\text{-}\rho) < A < max(m_2)(1\text{-}\rho) \quad (5)$$

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

$$[t_1 + \min(1-\rho), t_1 + \max(m_2)(1+\rho)] \quad (6)$$

Por lo tanto, el punto medio quedaría:

$$C_q(t_2, p) \equiv t_1 + \frac{\max(m_2)(1+\rho) + \min(1-\rho)}{2} \quad (7)$$

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:

$$\mathrm{E}_q(t_2, p) \equiv \frac{\max(m_2)(1+\rho) + \min(1-\rho)}{2} \quad (8)$$

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.

Además, el tiempo de envío de un paquete entre dos máquinas se puede descomponer como en la fig. 1, donde [3]:
Envío: construcción del paquete. Variable por estado del Sistema Operativo (context switch, scheduling, etc.).
Acceso al medio: En protocolo Ethernet dependerá del tráfico. Aleatorio.
Transmisión: Depende de velocidad de placa de red y largo mensaje. Determinístico.
Propagación: en red local despreciable.
Recepción: tiempo que tarda el mensaje en atravesar hasta la MAC layer. Es determinístico.
Recibo: Es el de desarmar el paquete. Depende del SO.

Fig. 1: Descomposición de tiempo de envío de un mensaje entre computadoras.

De lo visto surge que la sincronización depende de tres factores:
1) Variabilidad en los tiempos de comunicaciones de referencias
2) Variación medida en partes por millón (ppm) de la frecuencia de los relojes
3) Latencia en empezar a ejecutarse de los procesos involucrados.
Las primeras dos causas son dependientes fuertemente del hardware, y en el caso de las comunicaciones, del protocolo utilizado. Con respecto a la tercera, se puede disminuir con el uso de RTOS (Real Time Operating System). De lo anterior, se podría distinguir dos tipos de errores: uno a corto plazo, originadas a partir de 1) y 3), y otra que crece con el tiempo, 2), o a largo plazo.

4 Experimento sobre Deriva de Relojes en una Computadora

Se realizó un experimento, con un sistema operativo de tiempo real, RTAI [4] con el objetivo de analizar el comportamiento de dos relojes internos a una computadora. Se ejecuta una tarea como módulo del kernel, con frecuencia 4 segundos que lee el valor del contador interno TSC. Dado que la periodicidad, es fijada por el planificador de tareas usando el timer se dispone 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, las diferencia de TSC mediciones debieran ser constantes.

Mostrarían el error de medición, determinado por la deriva entre estos dos relojes. La figura 2 muestra los resultados del experimento y en la tabla 1 figuran los valores que se muestran en el gráfico. Como puntos aclaratorios y destacables del gráfico, se señala que:

- El eje horizontal muestra las partes por millón 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 ppm (ver detalle en la tabla 4.1), lo que concuerda con experimentos realizados con otro hardware por otros investigadores [3][1].
- 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.

En un oscilador de cuarzo como los usados se observan variaciones en la frecuencia del mismo de 1 ppm por ºC de cambio de la temperatura [3][1].

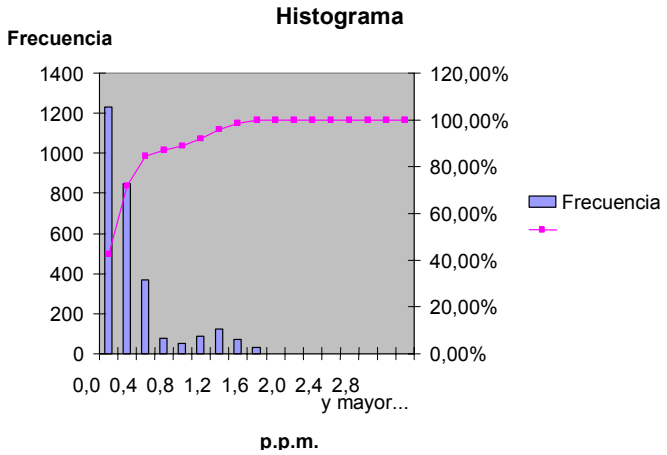

Fig. 2: Deriva de tiempo de TSC respecto del valor del timer del scheduler.

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.

Tabla 1: Frecuencias de los valores de deriva.

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%

5 Evolución del Error

Se llevó a cabo un experimento [9] sobre la evolución del error a partir de sincronizar 4 clientes con un servidor de hora durante 2000 segundos:

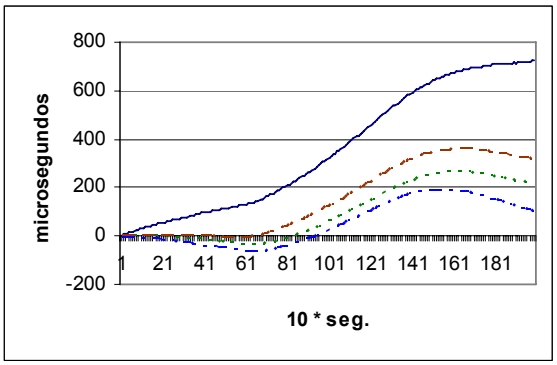

Fig. 3: Evolución del error de 4 máquinas sincronizadas con un servidor.

En la escala del gráfico prácticamente es despreciable el error a corto plazo, del orden del microsegundo. Se aprecia que el error se incrementa con el tiempo, producto de la deriva en partes por millón explicada antes. Con lo que transcurrido un cierto período pudiera requerirse una resincronización si se quiere mantener el error acotado.Para

ello caben dos posibilidades: 1) Interrumpir el programa instrumentado para realizar la resincronización de relojes, ó 2) Realizar la resincronización, compitiendo con el uso de recursos que haga la aplicación. Acá pudiera presentarse un problema con la intrusión, con lo que debe evaluarse los tiempos que llevaría una resincronización en estas condiciones.

6 Intrusión por (Re)Sincronización: Impacto en Escalabilidad

Inicialmente se analizan los tiempos de sincronización sin competencia de recursos (procesamiento y red) para distintas cantidades de computadoras con el fin de evaluar la escalabilidad de la implementación hecha para la sincronización de relojes de computadoras en un cluster. Para ello, se sincronizaron los relojes de 2 a 16 máquinas y se obtuvieron los valores que se muestran en la Tabla 2, dados 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.

Tabla 2: Tiempos de Ejecución para Sincronizar de 2 a 16 máquinas (μs).

Cant. Máq.	2	4	8	12	16
Mínimo	37972	39902	38596	37522	41774
Máximo	43362	46933	48233	53436	52791
Promedio	40475	42444	43585	45433	48312

En forma gráfica se puede apreciar la evolución de estos tiempos en la Fig. 4, donde además, se muestran los valores intermedios que se omitieron en la Tabla 2.

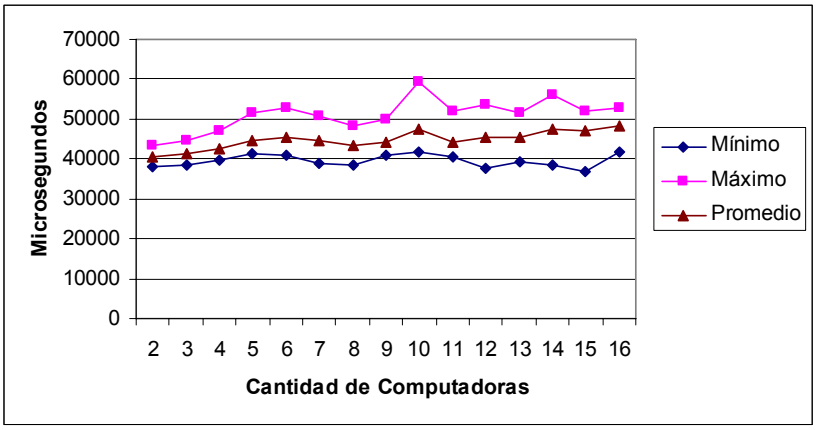

Fig. 4: Tiempos de ejecución para 20 corridas de Synchro en 2 a 16 máquinas.

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.

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 ejecución 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. Cuando en experimentos posteriores se sincronizaron los clientes con un *delay*, desaparece dicho tiempo inicial, ya que solicitan sincronizarse recién cuando el servidor ya terminó su inicialización.

A continuación se presentan los resultados con competencia de recursos corriendo simultáneamente en las mismas máquinas el experimento anterior con una aplicación paralela (multiplicación de matrices). La aplicación paralela tiene períodos bien definidos de cómputo y de comunicaciones: iterativamente se suceden intervalos de tiempo relativamente extensos con uso intensivo de CPU y luego se utiliza, también intensivamente la red de interconexión del cluster. La Tabla 3 muestra los resultados obtenidos y la Fig. 5 muestra los mismos resultados pero de manera gráfica.

Tabla 3: Tiempos de Ejecución para Sincronizar de 2 a 16 máquinas (μs).

Cant. Máq.	2	4	8	12	16
Max	66527510	202831064	539638	67348165	32424234
Min	601	1686	3771	17933	21903
Prom	7485613	11302582	79349	6251575	6338856

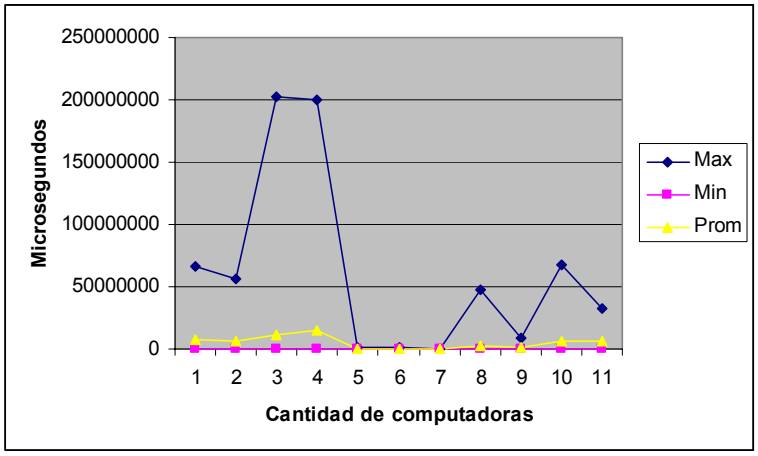

Fig. 5: Tiempos de Ejecución para Sincronizar de 2 a 16 máquinas (μs).

A partir de los datos de la Tabla 3 y de la Fig. 5 se pueden notar características importantes respecto del funcionamiento de la sincronización cuando la aplicación no se desacopla de la sincronización y directamente compite por los recursos disponibles

en el cluster (CPUs y comunicaciones). En principio, el tiempo máximo que puede necesitar una sincronización depende del nivel de competencia por los recursos que tengan las demás aplicaciones que se ejecutan en el cluster. Específicamente, analizando las estadísticas de tiempos de comunicaciones ha quedado claro que la competencia sobre la red de interconexión es la que definitivamente afecta el tiempo de sincronización de las computadoras. Esto, por otro lado era de esperar dada la dependencia tan fuerte de la sincronización siguiendo los lineamientos de Cristian con los tiempos de comunicaciones. Dada la similitud entre los tiempos mínimos y promedio de los datos de experimentación se puede concluir que, aunque la competencia puede causar efectos muy negativos sobre el tiempo máximo, la probabilidad de que esto suceda es muy baja: la mayoría de las veces que se sincroniza no hay problemas de exceso de tiempo por la competencia por recursos. En general, los tiempos no son afectados por la competencia por recursos y en la mayoría de los casos general los tiempos de sincronización son similares a los que se pueden dar sin competencia por los recursos. Es muy significativo que los mínimos con competencia (Tabla 3) pueden ser menores a los encontrados sin competencia (Tabla 2). En este caso, lo que sucede es que la aplicación en cierta forma "colabora" con una ordenación favorable de la ejecución del servidor y de los clientes del algoritmo implementado de sincronización. Más específicamente, es como se explica antes la relación del tiempo mínimo de 40 milisegundos iniciales que pueden hacer casi directamente reducir el tiempo de sincronización al tiempo de dos o tres paquetes TCP en la red de interconexión, dado que todos los procesos y la red están en las condiciones óptimas de ejecución.

7 Conclusiones y Trabajos Futuros

La sincronización de relojes de computadoras variará siempre en función del cumplimiento de requerimientos. A la hora de querer satisfacerlos a todos, no se llega a resultados satisfactorios. Dada las variaciones a largo plazo de los parámetros físicos de un reloj, se hace imposible lograr una sincronización duradera en el tiempo, por lo que la resincronización se hace necesaria. Ya que por el momento no parece probable que se cambie la tecnología (relojes estabilizados a cuarzo) de estos relojes, el problema seguirá existiendo en el futuro. La utilización de sistemas operativos con características de tiempo real tales como fijar prioridades y desalojo del uso de CPU puede mejorar la situación. Las últimas versiones de kernel Linux, 2.6, contemplan estas características, con lo que no sería tan complicado implementar clusters con sistemas de sincronización de este tipo. Sin embargo, todavía queda por analizar y cuantificar el impacto cuantitativo de esta posibilidad.

El sistema de sincronización implementado ha mostrado ser muy efectivo en el caso de no existir competencia por los recursos de un cluster. En este artículo se presentan resultados de experimentos llevados a cabo con competencia por recursos con una aplicación de cálculo numérico que se podría considerar típica en cluster usado para cómputo paralelo. En estos experimentos se puede notar que en la mayoría de los casos la competencia no presenta inconvenientes de tiempo para sincronizar o resincronizar un cluster, aunque hay excepciones. Cuando se dan las excepciones, es

decir los casos en los cuales se produce una competencia real por recursos agregan un tiempo de ejecución a la (re)sincronización no cuantificable a priori y dependiente, en realidad, del uso de recursos de las aplicaciones con las cuales la sincronización compite en el cluster. En ningún caso la sincronización se torna "imposible", solamente se necesitará un tiempo mayor al esperable. Una de las primeras líneas de extensiones a analizar e implementar consiste en tratar de identificar al menos de manera heurística la existencia de competencia para evitar esas situaciones y adelantar o postponer el período de (re)sincronización. Esto tiene que ser analizado desde la perspectiva de la cota de error que el esquema de Cristian tiene bien definida.

Referencias

1. S. Aggelos Bletsas, "Evaluation of Kalman Filtering for Network Time Keeping", IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control, Vol. 52, No. 9, Sep. 2005.
2. Flaviu Cristian, "Probabilistic clock synchronization". Distributed Computing, Springer verlag, 1989.
3. Jeremy Elson, Lewis Girod and Deborah Estrin, "Fine-Grained Network Time Synchronization using Reference Broadcasts". Department of Computer Science, University of California, Los Angeles, USA.
4. P. Cloutier, P. Mantegazza, S. Papacharalambous, I. Soanes, S. Hughes, and K. Yaghmour. Diapm-rtai. Real Time Operating Systems Workshop, November 2000.
5. Mills D.L., "Network Time Protocol (Version 3) specification, implementation and analysis", DARPA Networking Group Rep. RFC1305, University of Delaware, March 1992.
6. F. L. Romero, A. E. De Giusti, F. 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, Univ. Nac. de Chilecito, Chilecito, Argentina, Octubre 2008. ISBN 978-987-24611-0-2.
7. F. G. Tinetti, F. L. Romero, A. E. De Giusti "Clock Synchronization in Clusters for Performance Evaluation: Numeric/Scientific Computing". Proc. 2009 World Congress on Computer Science and Information Engineering, IEEE Computer Society, March 2009, Los Angeles, USA, ISBN 978-0-7695-3507-4/08.
8. F. Romero, "Sincronización de Relojes en Ambientes Distribuidos", Tesis de Maestría en Redes de Datos, Fac. de Informática, UNLP, Mayo de 2009.