Implementación de un entorno para estudios comparativos de simulación paralela y distribuida

Edgardo Szulsztein, Gabriel Wainer

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Pabellón I - Ciudad Universitaria
Buenos Aires (1428) – ARGENTINA
E-Mail: {edgardo, gabrielw}@dc.uba.ar

Abstract. En este trabajo se presenta el análisis del desarrollo de una herramienta que permite realizar estudios sobre distintos mecanismos y técnicas de simulación paralela. Se analizan los dos protocolos más difundidos de coordinación distribuida: los optimistas y los conservadores (o pesimistas). El objetivo fue incorporar técnicas conservadoras en un entorno optimista ya existente, para poder ejecutar simulaciones bajo cualquier modelo, y poder luego obtener conclusiones sobre cuestiones de performance de las mismas.

Palabras clave: simulación paralela, protocolo de simulación Time Warp, protocolo de simulación de Chandy-Misra.

1. Introducción

La simulación paralela de eventos discretos (PDES) se refiere a la ejecución de una simulación de eventos discretos en una computadora paralela o un entorno distribuido. El usar múltiples procesadores permite mejorar la velocidad de la simulación explotando el paralelismo inherente en el sistema, ya que la mayoría de los sistemas simulados consisten de muchos componentes operando en paralelo.

El proyecto Warped (Martin et al. 1997) permitió la definción de un kernel de simulación PDES basado en el protocolo Time Warp. El objetivo del presente trabajo fue extender los servicios ofrecidos por esta herramienta, de forma tal de poder realizar comparaciones de performance de diversas simulaciones bajo diversos protocolos. También se estudian diversas técnicas para lograr un mejor paralelismo en las simulaciones, estudiando las alternativas existentes para el manejo de rollbacks en Warped, y definiendo nuevos métodos e implementándolos en el kernel. El trabajo está organizado como sigue: la Sección 2 presenta el problema de sincronización, y se muestran las soluciones existentes. Luego se describe brevemente algunas características de un kernel de simulación Warped sobre el cual se basó el trabajo. La siguiente sección detalla las modificaciones realizadas para incorporar mecanismos pesimistas. Finalmente, se muestran algunos ejemplos con los que se han realizado simulaciones.

2. Sincronización en PDES

Los mecanismos para PDES organizan la simulación en un conjunto de *Procesos Lógicos* (PL) que interactúan entre sí intercambiando mensajes que representan eventos en el sistema. Además, es necesario utilizar algún mecanismo de sincronización para evitar errores de causalidad en el pasaje de mensajes (debido a la diferencia de velocidad entre los procesadores en ejecución).

Los mecanismos de sincronización evolucionaron en dos categorías: *conservadores* y *optimistas*. Las conservadoras *evitan* la posibilidad de errores de causalidad, basándose en alguna estrategia para determinar cuándo es seguro procesar un evento. Por otro lado, las aproximaciones optimistas usan una solución de *detección* **y** *recuperación*: los errores se detectan, y se invoca un mecanismo de rollback para recuperarse.

La idea de los mecanismos *conservadores* es determinar cuándo es seguro procesar un evento. Si un PL tiene un evento no procesado con un timestamp (y ningún otro con timestamp menor), y es imposible recibir un evento con timestamp inferior, entonces el evento se puede procesar de forma segura. En (Chandy y Misra 1979), se define que un PL con al menos un mensaje no procesado en **cada uno** de sus enlaces de entrada procesa todos los mensajes con ese timestamp. Todos los mensajes futuros tendrán timestamps posteriores al reloj local ya que llegan en orden cronológico en cada enlace. El procedimiento se repite mientras haya mensajes no procesados.

Una forma de resolver el problema de existencia de abrazo mortal trata de evitarlos especificando *estáticamente* enlaces que indican qué procesos pueden comunicarse con otros. En (Chandy y Misra 1979) se presenta un algoritmo basado en mensajes *nulos* (no corresponden a ninguna actividad en el sistema físico). Un mensaje nulo de PA a PB es una promesa de que no se enviará un mensaje con timestamp menor que el mensaje nulo desde el proceso A hacia el proceso B. Para que esta solución sea efectiva, es necesario tener información de *precálculo* (lookahead), es decir, habilidad para predecir qué pasará (o, más importante, qué NO ocurrirá) en el futuro simulado. Si un proceso en tiempo simulado T puede predecir todos los eventos que generará hasta T+L, se dice que el proceso tiene precálculo L. El precálculo mejora la habilidad de predecir eventos futuros, que pueden usarse para determinar qué otros eventos son seguros de procesar. Entonces, es útil utilizar el precálculo de un PL para determinar los timestamps asignados a mensajes nulos.

Los métodos *optimistas* no evitan errores de causalidad sino que los detectan y recuperan. En contraste con los mecanismos conservadores, no precisan determinar cuándo es seguro proceder; si no que detectan un error e invocan un procedimiento de recuperación, explotando paralelismo donde pueden ocurrir errores de causalidad pero de hecho no ocurren.

El mecanismo Time Warp es el protocolo optimista más conocido (Jefferson 1985). Cuando el mensaje de un evento recibido en un proceso lógico tiene timestamp menor al del reloj del proceso, hay un error de causalidad. Este evento provoca un rollback, y se llama *dispersor*. La recuperación se realiza deshaciendo los efectos de todos los eventos que han sido procesados prematuramente por los procesos que reciben el dispersor (es decir, los eventos procesados con mayor timestamp que el dispersor).

Los mensajes tienen dos timestamps: tiempo de emisión y de recepción. El reloj local (LVT - Local Virtual Time) de un proceso se pone en el mínimo tiempo de recepción de todos los mensajes no procesados. Los procesos pueden ejecutar eventos mientras tengan alguna entrada, por ende el LVT de un proceso puede adelantarse a los de sus predecesores, y puede luego recibir un mensaje de un evento del pasado. En este caso, el proceso hace rollback, y la simulación se rehace teniendo en cuenta el nuevo evento. Un rollback, además, puede enviar mensajes a otros procesos. Para lo primero se debe almacenar el estado del proceso (en el rollback se restaura un vector de estados pasados). Para anular un mensaje transmitido se envía un mensaje negativo o *antimensaje* que elimina al original cuando llega a su destino. Los mensajes de eventos se llaman mensajes *positivos*.

Si se recibe un antimensaje de un mensaje positivo procesado, debe deshacerse el proceso, y eliminar el mensaje positivo. Para cancelar los efectos de los mensajes erróneos deben almacenarse los estados de cada proceso desde el último tiempo "correcto" (llamado GVT - Global Virtual Time). Este es el mínimo tiempo entre los LVTs de todos los procesos y los tiempos de emisión de todos los mensajes enviados pero no procesados. Por otro lado, si un mensaje y su antimensaje están en la misma cola, se destruyen.

Al recibir un dispersor, el LP hace rollback y lo procesa. El proceso pone el estado actual en el último estado grabado antes del tiempo de recepción del dispersor, y pone su LVT en el tiempo de ese estado. Descarta todos los estados posteriores de su cola de estados, manda todos los antimensajes de su cola de salida con tiempo de emisión mayores que su LVT. Procesa el dispersor y procesa los mensajes en su cola de entrada con tiempos de recepción mayores o iguales al LVT, y avanza en el tiempo de simulación. Asumiendo memoria infinita, el algoritmo Time Warp no entrará en abrazo mortal, porque los procesos individuales no entran en abrazo mortal mientras tengan entradas, y el GVT siempre crece.

3. La Herramienta Warped

Warped es un kernel de simulación optimista y distribuida, basado en Time Warp. Fue diseñado para ejecutar en forma paralela y distribuida, ya sea en equipos multiprocesador o en redes heterogéneas de computadoras. Para tal fin, se utiliza el protocolo de envío de mensajes MPI (Message-Passing Interface). Los objetos de una simulación se modelan como entidades que reciben y envían eventos, y actúan en consecuencia a ellos mediante la actualización de su estado interno. El término evento se usa para referirse al elemento de comunicación entre los objetos de una simulación, y al término mensaje para la comunicación entre procesos del mismo kernel.

El kernel agrupa a los objetos de una simulación en procesos lógicos (de aquí en mas, un objeto en el contexto del kernel de Warped, se refiere a un proceso lógico tradicional). En cambio, se llama proceso lógico a la estructura que agrupa en un mismo procesador a todos los objetos que se simulan en él. Los objetos de un mismo proceso lógico se comunican sin la intervención del sistema de mensajes, lo cual permite una comunicación mucho más rápida. Por ende, se debe ubicar en el mismo proceso lógico a todos los objetos que se comunican entre sí con una alta frecuencia.

En una simulación, se alterna el control entre el kernel de Warped y el código a nivel de aplicación, por medio del uso cooperativo de llamadas a funciones. El kernel provee funciones para definir, entre otras cosas, cómo inicializar cada objeto de la simulación, y qué hace cada uno de ellos durante un ciclo de la misma. El kernel está definido como varios templates, por lo que el usuario puede definir parámetros del sistema sin reescribir código. El código de la aplicación se deriva del código del kernel, por lo que puede acceder a funciones del mismo en forma transparente.

4. Inclusión de Mecanismos Pesimistas

El primer objetivo planteado es el de adaptar el kernel de Warped de modo tal que el mismo se pueda comportar respetando el protocolo de Time Warp o respetando protocolos de sincronización pesimista. Warped utiliza una única cola de entrada de eventos para todos los objetos que pertenecen a un mismo proceso lógico (es decir, que simulan en un mismo equipo). Mediante la utilización de un planificador por menor tiempo de próximo evento, se planifica para ejecutar a continuación al objeto cuyo evento en la cola posee el menor tiempo de recepción. Sin embargo, esta estructura debe cambiar para implementar el protocolo pesimista de Misra.

Por ende, para cada objeto de la simulación se debe definir un arreglo para eventos de entrada, un arreglo para indicar los objetos que pueden recibir eventos de salida, y un entero, para almacenar el lookahead del objeto. Un esquema de esta nueva estructura puede verse en la Figura 1. El arreglo de eventos de entrada para un objeto i contiene en cada posición una estructura con el identificador j de un objeto de la simulación que puede enviarle eventos al objeto i, y una lista ordenada, con los eventos que el objeto j ya le ha enviado a i.

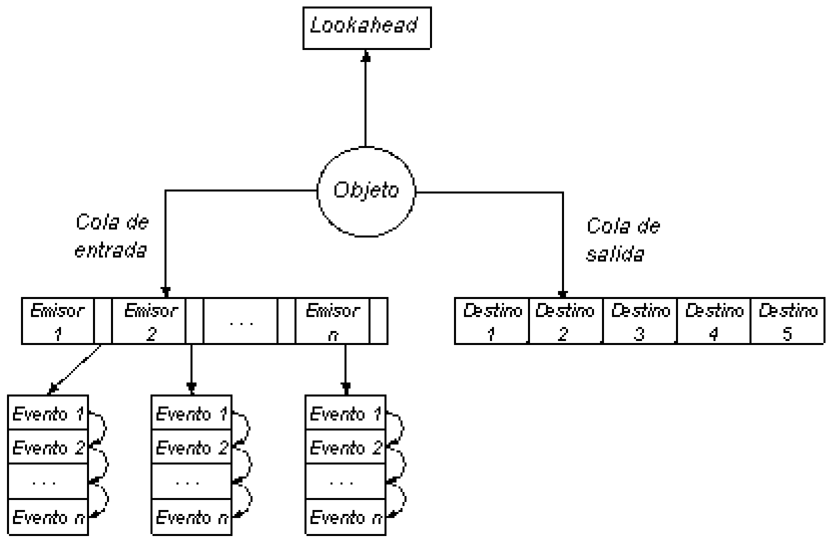

Fig. 1. Esquema de la nueva estructura de objeto en Warped

Originalmente, la aparición de un nuevo evento en la cola de entrada de un proceso lógico, provocaba que se entregue dicho evento al objeto receptor del mismo. En este caso, al recibir un evento, el proceso lógico deberá recorrer el arreglo de eventos de entrada del objeto destinatario del evento; cuando se haya encontrado la posición del arreglo cuyo elemento coincida con el identificador del emisor, se encola el evento. Luego se vuelve a recorrer el arreglo de eventos de entrada, chequeando eventos de entrada de todos los objetos que pueden enviar mensajes (esto se hace observando que cada lista de eventos no esté vacía), y se almacena el evento de entrada con menor timestamp.

El arreglo de eventos de salida contiene, en cada posición, un identificador de objeto destino, que puede recibir un evento enviado por este objeto en algún momento de la simulación. La finalidad de este arreglo es que el objeto sepa a quien debe enviar eventos nulos durante una simulación con protocolo pesimista. Por último, cada objeto almacena su lookahead, ya que se utiliza el mismo junto a la cola de eventos de salida para el envío de mensajes nulos a los objetos destino, para evitar la ocurrencia de deadlocks. Se generó un nuevo tipo de evento, correspondiente a los eventos nulos, mediante los cuales cada objeto informa de la llegada de un evento más su lookahead a sus posibles objetos destino, evitando deadlock.

El algoritmo original termina la simulación cuando circula por todos los objetos de la simulación (comenzando desde el objeto con id 0) un mensaje de control que no sufre modificación en su campo cancel, lo cual implica que todos los objetos están ociosos. Sin embargo, la inclusión de eventos nulos hace que la condición anterior pueda no ocurrir. Como cada objeto envía un mensaje nulo a todos sus posibles destinos cada vez que puede procesar un mensaje de entrada, y estos mensajes nulos generarán otros en cadena con los siguientes objetos sucesores, se cumple en las simulaciones que contienen objetos formando un ciclo que, a pesar de que no circulen más mensajes positivos en la simulación, y ésta debiera ya haber concluido, lo único que ocurrirá es la circulación indefinida de mensajes nulos. En este caso, cuando circula el mensaje de control por los objetos de la simulación, se dirá que un objeto está ocioso cuando: no está simulando evento alguno y no tiene eventos que no sean nulos sin procesar en su cola de entrada.

5. Ejemplo de Aplicación

Para mostrar el alcance de las modificaciones, mostramos el comportamiento de la herramienta al ejecutar un ejemplo. Se considera un esquema cliente/servidor, donde hay *N* clientes realizando solicitudes a un único servidor; los clientes quedan bloqueados luego de lanzar un requerimiento hasta que el servidor les devuelve una respuesta. Con esta simulación podemos analizar comportamientos de un típico esquema cliente-servidor, como por ejemplo los requerimientos HTTP de varios navegadores a un servidor de Web. Se destina el proceso lógico 0 para que albergue al objeto servidor, y a partir del proceso lógico 1 se pueden almacenar tantos clientes como se desee. También es configurable la cantidad de procesos lógicos con clientes que tendrá la simulación. En el siguiente esquema se muestra la estructura de una simulación con 2 clientes, alojados en un mismo proceso lógico.

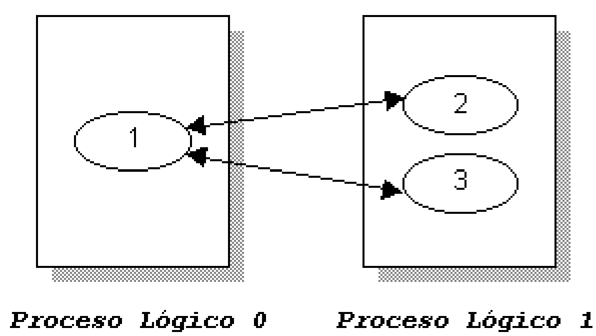

Fig. 2. Estructura de la simulación Cliente/Servidor.

Se configura a la simulación para que el objeto 2 realice dos pedidos al servidor, y el objeto 3 sólo uno. El servidor demora 3 unidades de tiempo en procesar cada solicitud y devolver una respuesta, y cada comunicación tiene un retardo de una unidad de tiempo. Se observa en la siguiente tabla la secuencia de eventos para una simulación pesimista (sin mostrar eventos nulos), asumiendo que el objeto 2 realiza un pedido al comienzo de la simulación, reenvía otro al recibir la respuesta, y el objeto 3 sólo realiza un requerimiento en tiempo 19 (Nota: (6,req21) simboliza timestamp 6 y requerimiento 2 del objeto 1):

Paso	Obj. 1 recibe	Obj. 1 envía	Obj. 2 recibe	Obj. 2 envía	Obj. 3 recibe	Obj. 3 envía
1				(0,req11)		
2	(1,req11)					
3		(4,req11)				
4			(5,req11)	(5, req21)		
5	(6,req21)					
6		(9,req21)				
7			(10,req21)			
8						(19,req12)
9						
10	(20,req12)					
11		(23,req12)				
12					(24,req12)	

Tabla 3. Secuencia para una simulación pesimista (sin eventos nulos).

La siguiente tabla muestra la secuencia de eventos que se suceden en la misma simulación con protocolo optimista; se agrega una columna mostrando el Local Virtual Time del objeto 1:

Paso	LVT obj. 1	Obj. 1 recibe	Obj. 1 envía	Obj. 2 recibe	Obj. 2 envía	Obj. 3 recibe	Obj. 3 envía
1	0				(0,req11)		(19,req12)
2		(1,req11) (20,req12)					
3	4		(4,req11)				
4	23		(23,req12)	(5,req11)	(5, req21)		
5		(6,req21)				(24,req12)	
6	**ROLL-BACK**						
7	6		(6,antim13)				
8			(9,req21)			(6,antim13) **ROLL-BACK**	
9				(10,req21)			(19,req12)
10		(20,req12)					
11			(23,req12)				
12						(24,req12)	
13							

Tabla 4. Secuencia de ejecución para una simulación optimista.

Por la naturaleza del protocolo, que intenta procesar un evento cada vez que éste está disponible, se procesa al primer evento del objeto 3 antes que el segundo evento del objeto 2, el cual tiene menor timestamp. Debido a ello ocurre un rollback en el objeto servidor en el tiempo 6, lo que produce en cascada otro rollback en el objeto 3 a continuación, obligando a este a reenviar su solicitud.

6. Incorporación de Nuevas Estrategias

El desempeño de los algoritmos de sincronización de PDES depende del grado de lookahead que pueda obtenerse (para las pesimistas), y de las diferencias de avance de la simulación en los distintos procesadores (que puede provocar alto grado de overhead debido a los rollbacks en el caso de las optimistas). En este caso estos parámetros sólo dependerán de la demora de cada proceso lógico. Una forma de mejorar el desempeño de estos factores se relaciona con la forma de procesar las listas de eventos. En general, las listas se procesan por medio de distintas tareas que ejecutan concurrentemente. El orden de selección de procesamiento de dichas listas puede determinar influencias importantes en los resultados. Pueden distinguirse varios casos:

El tiempo simulado avanza más lentamente en el procesador local que en los procesadores vecinos. Esto, en general, provocará que cuando llegue un mensaje, el proceso lógico tenga un tiempo virtual local inferior al de los timestamps de los mensajes recibidos desde los vecinos. Por ende, la probabilidad de errores de causalidad será baja, y será preferible dar mayor prioridad al procesamiento de la lista de eventos locales. De esta forma, la lista siempre dispondrá de mensajes con orden de procesamiento correcto y no habrá demoras en su tratamiento.
Si el procesador local procesa eventos a mayor velocidad que sus vecinos, los errores de causalidad ocurrirán frecuentemente. En este caso será conveniente dar mayor prioridad al proceso de entrada de nuevos mensajes.

Para detectar estas diferencias de velocidad se puede utilizar información implícita provista por los algoritmos de sincronización. En algoritmos optimistas, la existencia de un alto grado de rollback indica que el proceso local avanza a mayor velocidad que sus vecinos. En algoritmos pesimistas, se dispone de los peores timestamps en todos los enlaces de entrada, información que puede utilizarse para comparar con el reloj local y mejorar dinámicamente el desempeño.

Como fuera mencionado, una cuestión crítica enfrentada por las aproximaciones optimistas está relacionada con el overhead incurrido en el procesamiento de rollbacks. La ocurrencia de rollbacks provoca, por un lado, la explosión de antimensajes, que deben ser tratados. Por otro lado, un mal manejo del GVT trae problemas en cuanto al uso de memoria.

En su primera definición, el comportamiento de los algoritmos de sincronización optimista fue considerado similar al de los usados para administración de memoria virtual en sistemas operativos. La observación de un alto grado de overhead en el sistema es equivalente a un comportamiento de thrashing en sistemas con memoria virtual, pero aquí la mayoría del tiempo de procesamiento de la simulación se pierde en calcular resultados incorrectos, que provocan nuevos rollback.

Considerando estos hechos, se propone emplear una técnica que ha dado buenos resultados en cuanto a la relación overhead/uso de recursos para sistemas de memoria virtual. Esta técnica, llamada "Modelo de Frecuencias de Faltas de Página" (Wulf 1969), permite evitar el thrashing con bajo overhead. La técnica modificada será llamada Modelo de Frecuencia de Rollbacks (Wainer 1998), y usará la misma estrategia para estudiar el número de rollbacks para evitar su ocurrencia excesiva. La idea se basa en que cuando el número de Procesos Lógicos activos aumenta, el número de rollbacks también crecerá (exponencialmente), debido a que el número de antimensajes se multiplicará. En cambio, si el número de rollbacks es bajo, puede interpretarse como una señal de que el grado de paralelismo también será bajo, ya que la falta de rollbacks puede indicar un bajo grado de intercambio de mensajes.

Como en una simulación optimista o pesimista el número de procesos lógicos es fijo, la detección de una frecuencia de rollback alta para un proceso lógico es una indicación de que la simulación ejecutada por el procesador local debería demorarse. En este caso sólo se aceptarán nuevos mensajes de entrada que se agregarán en la lista de eventos locales, pero la lista de eventos locales no será procesada. De esta forma se reduce el número total de procesos lógicos activos y, por ende, en determinado momento el número de rollbacks se reducirá.

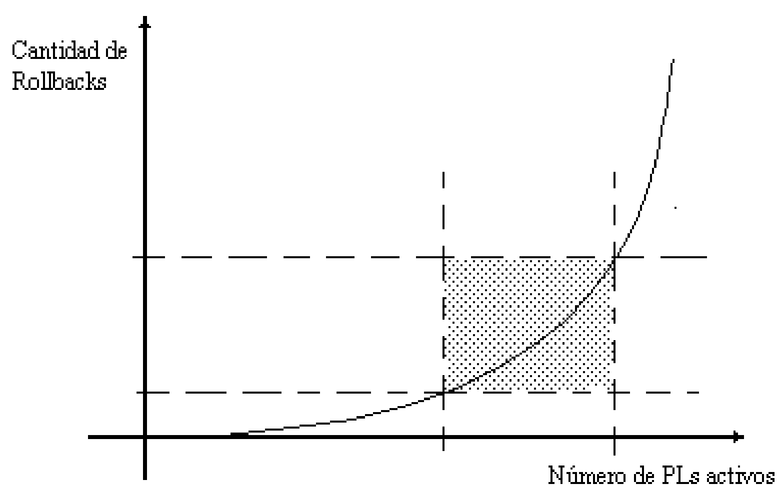

Fig. 3. Modelo de Frecuencia de Rollback (Wainer 1998).

Por otro lado, si el número de rollbacks es muy bajo, esto es una indicación de que el procesamiento de la lista de eventos local está avanzando lentamente. Por ende, si existe algún proceso lógico que haya sido demorado, deberá ser reactivado.

El fijar una frecuencia mínima y máxima de rollbacks (como se ve en la Figura 3), permite mantener el paralelismo en un nivel alto, reduciendo el overhead de la simulación. Si el número de rollbacks está por debajo de la cota inferior para cada proceso lógico y ninguno es demorado, el grado de paralelismo no puede aumentar. Por ende, la distribución de la carga debería modificarse. Para ello es necesario contar con algoritmos dinámicos para el balanceo de la carga.

6. Conclusión

En este trabajo se han analizado los resultados de implementación de los dos principales mecanismos de simulación distribuida: los optimistas y los conservadores. Cada uno de ellos resulta más eficiente de acuerdo al dominio de la simulación que se desee realizar. Por lo tanto, se modificó una herramienta de dominio público, que realiza simulaciones bajo el protocolo optimista, para permitir que ejecute el protocolo pesimista de Chandy-Misra. En este momento se están realizando comparaciones de eficiencia entre los protocolos, como así también entre diversas modificaciones de los mismos, permitiendo, de esta forma, contar con un estudio comparativo exhaustivo entre los diversos algoritmos existentes en la actualidad.

REFERENCIAS

Chandy, K; Misra, J.: Distributed simulation: a case study in design and verification of distributed programs. IEEE TOSE (1979).

Jefferson, D.: Virtual Time. ACM TOPLS, 7(3) (1995) 404-425.

Martin, D.; Mcbrayer, T.; Radhakrishnan, R.; Wilsey, P.: TimeWarp Parallel Discrete Event Simulator'. Technical Report. Computer Architecture Design Laboratory, University of Cincinnati (1997).

Misra, J.: Distributed discrete-event simulations. ACM Computing surveys. Vol. 18, No. 1 (1986) 39-65.

Wainer, G.: Introducción a la simulación de sistemas de eventos discretos. Informe técnico N° 96-005. Departamento de Computación. FCEN/UBA (1996).

Wainer, G.: Discrete event cellular models with explicit delays. Ph.D. Thesis. DIAM/IUSPIM. Université d'Aix-Marseille III (1998).

Wulf, W.: Performance monitors for multiprogramming systems. Proceedings of the 2nd. ACM symposium on Operating Systems Principles (1969) 175-181.