Rev. Téc. Ing. Univ. Zulia. Vol. 30, Nº 2, 170 - 178, 2007

Cache memory coherence protocol for distributed systems

***José Lisandro Aguilar Castro*[1] *and Rodolfo Leonardo Sumoza Matos*[2]**

[1]*Escuela de Ingeniería de Sistemas, CEMISID, Universidad de Los Andes. Mérida, Venezuela, Código Postal 5101, Teléfono: 0274-2447111. aguilar@ula.ve*

[2]*Departamento de Computación, Universidad Simón Bolívar. Sartenejas-Baruta, Venezuela, Código Postal 1080, Teléfono: 0212-9063241. sumoza@ldc.usb.ve*

Abstract

This work proposes a protocol which manages the coherence in the cache memory in systems with distributed memory. Cache memory is distributed in different system's nodes, for this reason-stored information in them must to be maintained coherent. This protocol was proved using a methodology based on the formal description using a finites states machine and the Spin tool.

Key words: Cache memory, coherence protocols, distributed Systems.

Protocolo de coherencia de memoria cache para sistemas distribuidos

Resumen

Este trabajo propone un protocolo que gestiona la coherencia a nivel de la memoria cache en sistemas con memoria distribuida. La memoria cache se encuentra distribuida a través de los diferentes nodos del sistema, por lo que se debe mantener coherente la información almacenada en ellas. El protocolo se validó utilizando una metodología basada en la descripción formal, utilizando una máquina de estados finitos y la herramienta de software Spin.

Palabras clave: Memoria cache, protocolos de coherencia, sistemas distribuidos.

1. Introducción

Este trabajo propone un protocolo que gestiona la coherencia a nivel de la memoria cache en sistemas con memoria distribuida (no compartida), en el cual también la memoria cache está distribuida, es decir, la arquitectura subyacente es un sistema de Memoria Cache Distribuida, cuya red de interconexión puede ser de cualquier tipo. Principalmente se han realizado estudios en cuanto a la coherencia de la cache cuando se utiliza memoria compartida distribuida [1-8], como por ejemplo el caso de DASH cc-NUMA, que utiliza un mecanismo de coherencia para la memoria cache, pero bajo una arquitectura de memoria compartida distribuida. En cuanto a memorias distribuidas, hasta ahora la mayoría de los trabajos se han basado en plantear la distribución de la memoria principal, y su interacción con sus respectivas cache, y no abordan el tema de la cache distribuida específicamente [2, 9, 5]. Este trabajo se organiza como sigue: inicialmente presentamos el marco teórico usado, después presentamos el protocolo propuesto. Finalmente se presentan las verificaciones hechas y las conclusiones.

2. Marco Teórico

2.1. Memoria cache

Es una memoria temporal, generalmente de existencia oculta y automática para el usuario, que proporciona acceso rápido a los datos de

uso más frecuente. La forma de trabajar de la memoria cache es simple, los datos pedidos por el procesador son buscados primero en la memoria cache, y después, si no están disponibles en ella, se buscan en la memoria principal.

Existen una serie de mecanismos para la administración y gestión de la memoria cache que son utilizados por los distintos tipos de plataformas distribuidas (memoria compartida, memoria distribuida, etc.), los cuales influyen directamente en el rendimiento y el costo de este recurso. Entre los mecanismos está [2, 13, 4, 5] el denominado Write Through o Write Update en el cual la data es escrita en la memoria principal al mismo tiempo que es almacenada en la memoria cache.

La distribución de la memoria cache en sistemas con memoria principal distribuida, significa que cada procesador posee su propia cache y su propia memoria principal. Los mecanismos presentados previamente establecen una relación entre la memoria principal y su cache, sin considerar a los otros nodos del sistema. Es en estos casos donde el problema de coherencia deviene crucial. El uso de la Memoria Cache Distribuida es importante en ambientes donde hay muchos accesos a datos remotos por aplicaciones, en los cuales las actualizaciones de los mismos ocurren con muy poca frecuencia. Por ejemplo, en los Grid de Datos donde las consultas remotas son importantes.

2.2. Coherencia

La coherencia significa que cualquier lectura debe retornar el valor de la escritura más reciente. Mientras más estricta sea esta idea, más difícil de implementar será. Para explicarlo de otra forma, la coherencia se logra cuando cualquier escritura debe ser vista por una lectura, y todas las escrituras son vistas en el orden apropiado (serialización). Para el mantenimiento coherente de las cache se utilizan básicamente dos tipos de protocolos:

- **Protocolo entrometido, de sondeo o snoopy:** Su mecanismo básico es el de distribuir la información permanentemente usando las políticas de Invalidación en Escritura o Escribir-Invalidar y Actualización en Escritura, Escribir-Actualizar [2, 5].
- **Protocolo con directorio (mantiene la coherencia utilizando la memoria principal):** Implementa una estructura, llamada tabla o directorio, que registra cuál procesador o nodo ha gravado en su memoria cache cualquier bloque dado de la memoria principal [12].

3. Presentación de la Propuesta

3.1. Generalidades

Se propone el diseño de un protocolo para la gestión de la coherencia en sistemas distribuidos, con memoria principal y memoria cache distribuidas, cuyo espacio de direcciones (direccionamiento) se gestiona a nivel local. El entorno del que se está hablando es uno que utiliza la conexión en red de un conjunto de nodos, donde cada uno representa un computador con un CPU, una memoria cache y una memoria principal, conectado a través de una red de interconexión con el resto de los nodos.

Cada nodo, inicialmente tiene un conjunto de datos en su memoria principal, y para ese conjunto de datos, ese nodo representa su nodo hogar o Home. Además cuenta con un directorio local que registra ese conjunto de datos que pueden estar almacenados en memoria cache local o remota. El nodo hogar es el dueño de ese conjunto de datos, por ende, ningún otro nodo puede tener algún dato de ese conjunto en su memoria principal, pero si en su memoria cache.

El protocolo propuesto utiliza como base para la gestión de la consistencia el paradigma de Directorios Distribuidos, implementados localmente en cada nodo a través de Árboles Binarios Balanceados como estructura dinámica (Figura 1), como mecanismo de coherencia se utiliza la Actualización en Escritura (Write update), y la exclusión mutua distribuida la cual está implícita en el procedimiento. En la Actualización en Escritura implementada, cuando un procesador cambia un dato este mecanismo actualiza de forma inmediata la memoria principal y la memoria cache local donde pertenece el dato. En el caso de actualizaciones remotas, la cache remota queda actualizada junto con la cache y la memoria prin-

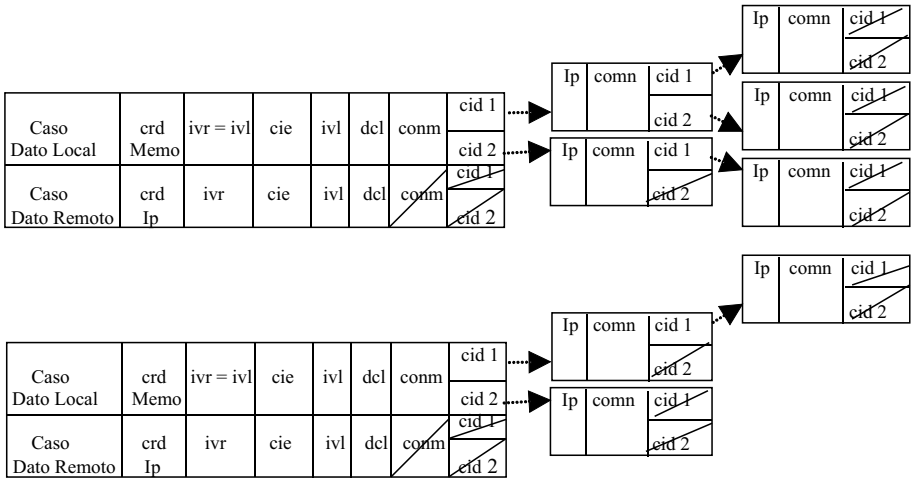

Figura 1. Estructura del directorio.

cipal del nodo al que pertenece el dato (nodo hogar). En todos los casos, el resto de las memorias cache de los otros nodos que comparten el dato y que están registrados en el directorio que se encuentra en el nodo hogar son invalidadas. Para identificar al nodo hogar (nodo propietario) se utiliza uno de los campos del directorio (Figura 1), el cual contiene una dirección de memoria principal (apuntador), en los casos donde el dato pertenezca al nodo local, o una dirección IP, para los casos donde el dato pertenezca a un nodo remoto.

Cada dato posee tres (3) estados posibles: Limpio, Inválido y Bloqueado. El primer caso representa un dato que no ha sido modificado por algún procesador desde su última actualización. El segundo caso ocurre cuando un dato ha sido modificado por algún otro nodo, tal que a su vez se actualiza la memoria en el nodo hogar y se invalida el resto de los nodos que comparten el dato, estas invalidaciones se hacen a través del envío de mensajes a los nodos registrados en el directorio (haciendo un broadcasting). El tercer caso ocurre cuando un nodo intenta modificar un dato, y esta acción no se ha completado en la memoria principal del nodo hogar; así, el estado en la memoria cache del nodo el cual realiza la actualización será el de bloqueado; de esta forma ningún otro proceso puede acceder a este dato hasta terminar la acción que se está realizando sobre él. La consistencia se garantiza por los mecanismos de exclusión mutua distribuida de acceso al dato objeto de estudio, y a través de la propagación de mensajes (broadcasting) a los sitios involucrados en las operaciones de manejo de coherencia. Cada directorio tiene los siguientes campos:

- Campo de Referencia del Dato o crd: Es un apuntador al dato que se almacena en la memoria, y es el índice de búsqueda del directorio. Para los casos en los cuales los datos no pertenecen al nodo, dicho apuntador será la dirección IP del nodo hogar del dato.
- Campo de Información del Estado o cie: Contiene la información del estado actual del dato (Limpio, Inválido o Bloqueado).
- Campo del identificador Remoto del Dato o ivr: Es el nombre que le asigna el nodo remoto al dato. Este se utiliza con la finalidad de evitar inconsistencias a nivel de nombres de datos locales y remotos. Para el caso cuando el dato pertenece a ese nodo el valor de este campo es el mismo que el campo ivl.
- Campo del Identificador Local del Dato o ivl: Es el nombre que le asigna el nodo local al dato. También se utiliza con la finalidad de evitar inconsistencias a nivel de nombres de datos remotos y locales.
- Campo de la dirección de memoria cache o dcl: Identifica el sitio en la memoria cache donde se encuentra el dato.
- Campos de información de la presencia del Dato o cid1 y cid2: Son dos campos apuntadores utilizados para construir el Árbol Binario que contiene la información (las direcciones IP) del resto de los nodos (compu-

tadoras) que en un momento dado comparten el dato (Figura 1). Cada uno de los "nodos" del Árbol Binario posee tres campos: Uno indica la dirección IP del nodo, el otro es un conmutador utilizado para el balanceo, y el tercero es un par de apuntadores a sus "nodos" hijos.

- Campo conmutador conm: Refleja la última asignación hecha en el árbol, es decir, si fue en su rama izquierda o derecha. Se utiliza para fines del balanceo de dicho árbol.
- Cuando el dato es remoto, es decir no pertenece al nodo, los campos conm, cid1 y cid2 son nulos.

El esquema de dicho protocolo se presenta como la interacción entre el procesador, la memoria principal, y la memoria cache de los nodos que conforman el sistema. La comunicación entre nodos se realiza a través de pases de mensajes administrados segun una política FIFO. En la Figura 2 se muestra la máquina de estados finitos que describe el comportamiento del protocolo a nivel de la memoria cache.

3.2. Reglas que rigen el comportamiento del protocolo en cada uno de sus componentes

3.2.1. Reglas para las solicitudes del procesador

La Tabla 1 muestra el conjunto de reglas asociadas al procesador.

Explicación y notación:

- ***Instrucción (a,v)***, representa una instrucción que puede ser: *Store(a,v) o solicitud de escritura de un dato y Load(a) o solicitud de lectura,* que utilizan la dirección de memoria *a* y el valor del dato *v* (puede ser opcional, según sea el caso).

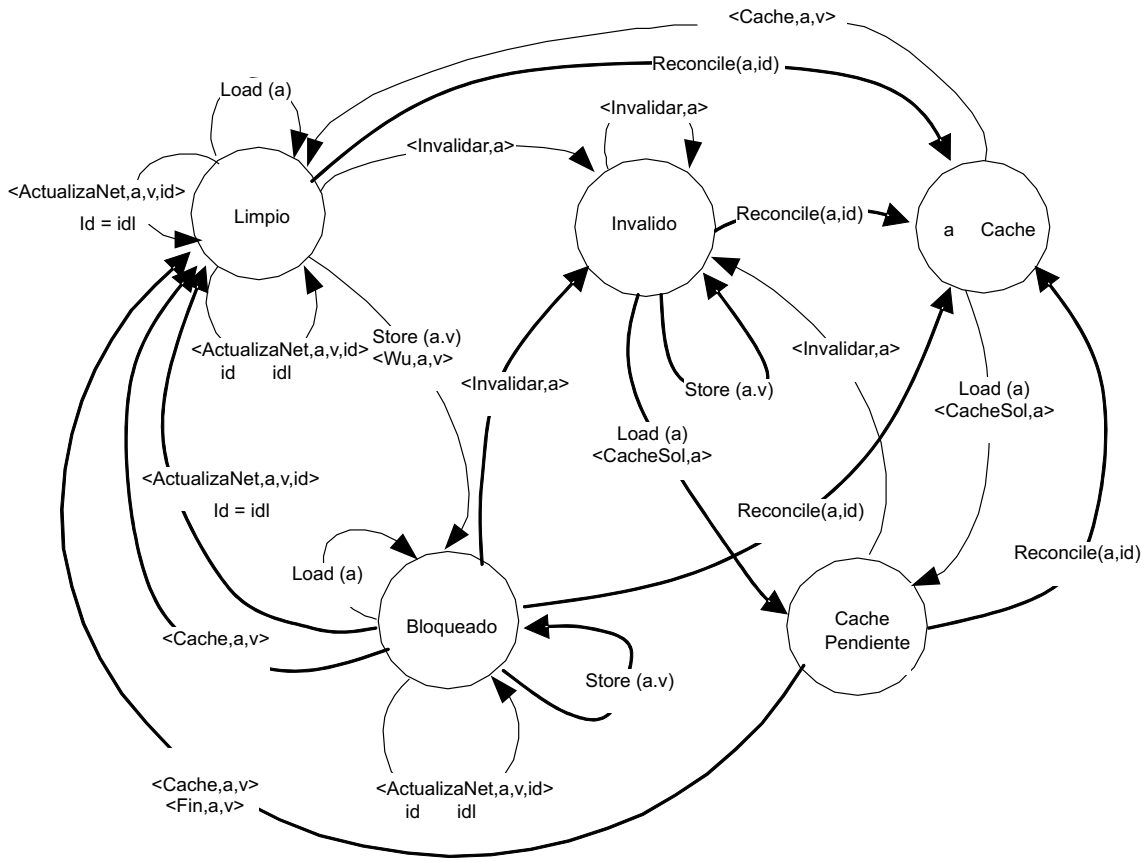

Figura 2. Máquina de estados finitos.

Tabla 1
Reglas del procesador

Instrucción	Edo. Inicial Cache	Acción	Edo. Final Cache
Load(a)	Celda(a,v,Limpio)	Finalizar Load	Celda (a,v,Limpio)
	Celda(a,v,Invalido)	<CacheSol,a> → Memo	Celda(a,-CachePendiente)
	Celda(a,v,Bloqueado)	Finalizar Load	Celda(a,v,Bloqueado)
	a ∉ cache	<CacheSol,a> → Memo	Celda(a,-,CachePendiente)
Store(a,v)	Celda(a,-,Limpio)	<Wu,a,v> → Memo	Celda(a,v, Bloqueado)
	Celda(a,-,Invalido)	Finalizar Store	Celda(a,v, Invalido)
	Celda(a,v1,Bloqueado)	Finalizar Store	Celda(a,v1,Bloqueado)

- En cuanto a los estados de la cache, ***Celda (a,v,Estado)*** representa una Celda o Bloque en la memoria cache con dirección *a*, valor del dato *v* (opcional) y su estado actual que puede ser Limpio, Inválido o Bloqueado.
- La acción que se representa con un patrón como el siguiente: ***<cmd,a,v>***, simboliza un mensaje cuyo comando asociado es *cmd*, con dirección en memoria principal *a* y valor del dato *v* (puede ser opcional, según el caso). Para el caso de las reglas del procesador, <cmd,a,v> → Memo, quiere decir que el mensaje se ha enviado a la memoria principal (Memo). El comando es la parte que indica qué es lo que el mensaje está requiriendo. Los mensajes pueden ser ***<Wu,a,v>*** y ***<CacheSol,a>.***
- ***<Wu,a,v>*** representa un mensaje de solicitud de escritura o actualización *Write Update*, a la dirección de memoria *a* con el valor del dato *v*.
- ***<CacheSol,a>*** representa un mensaje que solicita el valor o dato que posee la dirección de memoria *a*.
- Un estado ***CachePendiente*** es un estado momentáneo que representa la espera de la respuesta a la solicitud hecha a la memoria de un dato.
- a ∉ cache, quiere decir que la dirección de memoria *a* no se encuentra almacenado en la Cache.
- ***Reconcile(a,id)*** elimina físicamente el dato "a", con dirección "id", de la memoria cache local. Una de las posibles causas es cuando la memoria cache está llena y el mecanismo de reemplazo sustituye a "a". Ahora bien, cuando un nodo remoto, el cual no es el propietario del dato "a", decide sacar de su memoria cache al dato "a", éste le envía un mensaje al nodo hogar (id) para que actualice su directorio (árbol), desincorporando dicho nodo remoto del registro relacionado al dato "a".

Una instrucción *Load* finaliza después que el dato ha sido enviado al procesador, y una instrucción *Store* se finaliza después que la memoria principal y la cache han sido modificadas en el nodo hogar.

3.2.2. Reglas de la cache y de la memoria principal

Las reglas que establecen el funcionamiento lógico del protocolo en la memoria cache se muestra en la Tabla 2.

Las reglas de la memoria principal pueden verse en la Tabla 3.

Explicación y notación:

- El origen y el destino de un mensaje pueden ser un sitio en cache (*id y idl* representan la dirección IP, o IP local respectivamente), un conjunto de sitios cache o nodos (*dir*), la memoria principal local o remota (*Memo*), o el procesador.
- Un mensaje recibido en la memoria principal es siempre enviado del sitio *id*, del procesador, de una memoria remota o de la misma memoria local, mientras que un mensaje

Tabla 2
Reglas mandatorias del motor de cache

Mensaje (*)	Edo. inicial cache	Acción	Próximo edo. cache
<Cache,a,v>	a ∉ cache	<Fin,a,v>→ Memo	Celda(a,v,Limpio)
	Celda(a,-,CachePendiente)	<Fin,a,v>→ Memo	Celda(a,v,Limpio)
	Celda(a,v,Bloqueado)		Celda(a,v,Limpio)
<Invalidar,a>	Celda(a,-,Limpio)		Celda(a,-,Invalido)
	Celda(a,v,Inválido)		Celda(a,-, Invalido)
	Celda(a,-,Bloqueado)		Celda(a,-, Invalido)
	Celda(a,-,CachePendiente)		Celda(a,-, Inválido)
<ActualizaNet,a,v,id>	Celda(a,-,Limpio) id = idl	<Invalidar,a>→dir-idl	Celda(a,-,Limpio)
	Celda(a,v1,Bloqueado) id ≠ idl	<Invalidar,a>→id	Celda(a,v,Bloqueado)
	Celda(a,v1,Limpio) id ≠ idl	<Invalidar,a>→dir-id-idl <Cache,a,v>→ id	Celda(a,v,Limpio)
	Celda(a,-,Bloqueado) id = idl	<Invalidar,a>→dir	Celda(a,v, Limpio)

(*)Desde Memoria Principal del nodo local, o memoria cache o principal del nodo remoto hogar.

Tabla 3
Reglas mandatorias del motor de memoria

Mensaje	Edo. inicial de memoria	Acción	Próximo edo. memoria
<CacheSol,a>	Celda(-,-,D[dir]) a ∉ dir	<MemoSol,a>→Memo	Celda (-,-,D[dir])
	Celda(a,-,D[dir]) a ∈ dir	<MemoSol,a>→Memo	Celda (a,-,D[dir])
<Wu,a,v1>	Celda(a,v, D[idl | dir]) a ∈ Memo	<Invalidar,a>→dir-idl <Cache,a,v>→idl	Celda(a,v1, D[idl])
	Celda(a,v, D[id | dir]) a ∉ Memo	<ActualizaNet,a,v,idl>→id	Celda(a,v, D[id|dir])
<MemoSol,a,>	Celda(-,-,D[dir]) a ∈ Memo	<Cache,a,v>→id	Celda(a,v,D[id|dir])
	Celda(a,-,D[dir]) a ∉ Memo	<MemoNet,a,v>→Net	Celda(a,v,NetPendiente)
	Celda(a,-,D[dir]) a ∉ Memo a ∈ Dir	<MemoNet,a,v>→id	Celda(a,v,NetPendiente)
<MemoNet,a,v>	Celda(a,-,Memo) a ∈ Memo	<Cache,a,v>→id	Celda(a,v,D[id|dir])
	Celda(a,-,Memo) a ∉ Memo	Finaliza	
<Fin,a,v>	Celda(a,v,NetPendiente)	Finaliza	Celda(a,v,D[id|dir])

recibido en la cache es siempre enviado desde la memoria principal local o remota.

- **<*cmd,a,v*>→ *dir*** quiere decir el envío del mensaje con un comando cmd a un conjunto de sitios de cache (dir), con dirección de memoria a y valor del dato v (opcional).
- **<cmd,a,v>→Memo, a ∈ Memo, <Wu,a,v>, CacheSol, CachePendiente** tienen la misma significación que en las reglas del procesador.
- **<Cache,a,v>** representa un mensaje que le indica a la cache que almacene el valor o dato *v* con dirección de memoria *a*.
- **<Invalidar,a>** representa un mensaje que le solicita a la cache que invalide su contenido o dato, y elimina el apuntador en el directorio del nodo hogar a ese dato, de tal forma que el espacio del dato inválido pueda ser reutilizado.
- Para el caso de la memoria **Celda(a,v, D[dir])**, representa una celda en la memoria principal con dirección de memoria a, valor del dato v, y con un directorio el cual contiene almacenado el conjunto de referencias a nodos *dir*.
- **Celda(a,v,D[id|dir]),** también relacionado a la memoria, representa una celda en la memoria principal con dirección a, valor del dato v, y con un directorio el cual contiene el conjunto de referencias a nodos dir, en el que se incluye la referencia id, de forma específica.
- **Celda (a,v,MemoPendiente)** Representa una celda en la parte de la memoria principal que almacena el directorio, la cual posee un estado de espera por una respuesta que debe provenir de la otra parte de la memoria, pero fuera del directorio, es decir, en su contenido general.
- **Celda(a,v,NetPendiente)** Representa una celda en la parte de la memoria principal que almacena el directorio, la cual posee un estado de espera por una respuesta que debe provenir de la red (de otro nodo).
- **<MemoSol,a>** Representa un mensaje que envía la parte de la memoria que posee el directorio, a la parte de la memoria general de la memoria.
- **<MemoNet,a,v>** Representa un mensaje enviado por la memoria principal a la red, solicitando un dato con dirección *a* y valor *v*.
- **<ActualizaNet,a,v,id>** Es un mensaje enviado por la red desde *id*, el cual solicita la actualización del dato en los nodos. Para este caso en particular, si es el nodo hogar actualiza el dato, de lo contrario lo invalida.
- **<Fin,a,v,id>** Es un mensaje generado con la intención de cambiar el estado NetPendiente de la celda de la memoria.
- Cada regla de la memoria, cuando está asociada a un conjunto de referencias a nodos (con sus memorias cache), es decir, cuando envía un mensaje a un dir, la memoria lo distribuye a todos los nodos, y cada nodo lo gestiona de forma local.

4. Validación

La validación se realizó de dos formas, la primera, a la que se le dio el nombre de validación analítica, consiste en verificar analíticamente que en todos los procesos generados por el protocolo se obtenga siempre alguna respuesta (criterio de completitud). La segunda forma de validación consiste en evaluar algunos criterios de correctitud en forma práctica, es decir, criterios que permitan establecer si el protocolo realiza todos los procesos de forma correcta.

4.1. Validación analítica

En esta parte se evaluará el criterio de "completitud". La completitud consiste en una propiedad de los sistemas relacionada a la capacidad de finalizar de forma adecuada cada uno de los procesos que inicia, es decir, que cada proceso iniciado debe ser finalizado, y de una forma que se considere correcta. Para esto, en el primer tipo de evaluación para verificar la consistencia lógica se utilizó la máquina de estados finitos de la Figura 2. Este estudio se basa en establecer que cada uno de los estados que se representan pueden ser alcanzados de alguna manera por la dinámica del sistema. Así, la evaluación de completitud consiste en establecer de forma gráfica cada uno de los caminos posibles que pueden te-

ner los procesos que se pueden desarrollar utilizando el protocolo propuesto en este trabajo.

Este modelo representa la máquina de estados finitos de un solo nodo, y la dinámica se replica de igual manera para el resto de los nodos. El modelo del protocolo se sometió a un análisis exhaustivo de los estados que pueden o no ser alcanzados, a través de un algoritmo exhaustivo propuesto en [6]. Esta verificación es simple, aunque exhaustiva, ya que consiste en recorrer la máquina de estados finitos comenzando por un estado cualquiera, y se van chequeando el resto de los estados que pueden ser alcanzados. La segunda forma de evaluación de la completitud consiste en evaluar de forma gráfica que cada instrucción generada por un procesador (load, store) se ejecute en su totalidad. Esta evaluación se basa en el seguimiento de las reglas y dinámicas especificadas en las Tablas 1, 2 y 3, las cuales muestran las reglas del procesador, de la memoria principal y cache. Tal como se muestra en las Tablas 1, 2 y 3, al realizar un seguimiento de cada uno de los procesos ejecutados por un procesador, la culminación de cada ejecución o proceso debe ser llevado a cabo por una acción que indique la finalización de la instrucción de forma directa (por ejemplo, "*Finalizar Load*") o eliminando cualquier estado de transición (por ejemplo, CachePendiente), a través del desarrollo de cada uno de los pasos que se señalan en dichas tablas.

4.2. Validación experimental

En esta parte se evaluarán criterios de "correctitud". Consiste en encontrar de forma experimental un error en el diseño que impida la ejecución esperada y adecuada del protocolo. La herramienta utilizada se denomina Spin, la cual es un interpretador de un Meta Lenguaje denominado PROMELA (Process Meta Languaje) [14].

Para el modelo de esta investigación, sólo se consideraron los criterios de correctitud siguientes: Verificación de la no Existencia de abrazos mortales y/o lazos infinitos. Inicialmente se elaboró un modelo del protocolo propuesto utilizando PROMELA, en el cual se consideraron sólo dos nodos, cada uno con su procesador, su memoria cache y su memoria principal.

El análisis utilizando Spin mostró que se ejecutaron en la simulación 5 procesos. Pero lo más importante es que al final se observa un indicador que menciona que el modelo posee un estado válido de finalización (valid end state). Lo que quiere decir es que al ejecutar los procesos aplicando cada una de las reglas del protocolo no se generaron abrazos mortales, ni lazos infinitos, ya que de lo contrario la simulación no hubiese finalizado.

Por otro lado, se utilizó Spin como un verificador exhaustivo, capaz de verificar rigurosamente la validez de los requerimientos de correctitud especificados. De esta forma no se encontraron ningún tipo de conflictos, por lo que no se generaron estados finales inválidos. Los estados finales inválidos surgen cuando se generan lazos infinitos o abrazos mortales. De esta manera los criterios de correctitud fueron probados y no indicaron errores de diseño.

5. Conclusión

Para lograr el diseño del protocolo propuesto se realizó un análisis exhaustivo del problema de mantenimiento de la coherencia en las memorias cache distribuidas. El protocolo cumplió con las exigencias de correctitud y consistencia lógica requeridas en los protocolos en los sistemas distribuidos. Principalmente, el protocolo propuesto se caracteriza por utilizar un método que usa directorios distribuidos en cada nodo del sistema, los cuales almacenan la información sobre los datos guardados en las memorias cache, incluyendo sus estados actuales. Por medio de estos directorios distribuidos se logra manejar la coherencia de las memorias cache. Los directorios usan como estructura de datos dinámica un Árbol Binario Balanceado.

El protocolo se basa en un conjunto de reglas que establecen el manejo de la memoria principal y cache en cada nodo del sistema, así como los intercambios de información entre los nodos. Todo esto garantiza la coherencia de los datos almacenados en las diferentes memorias del sistema.

Se realizó una validación basada en dos criterios: el primero denominado *completitud*, para lo cual se utilizó por un lado un algoritmo exhaustivo, y por otro lado un análisis gráfico, dando como resultado un diseño que cumple con las exigencias de *completitud*. El segundo criterio se

llamo *correctitud*, que se evaluó a través de una simulación y utilizando un verificador exhaustivo (ambos usando *Spin*), las cuales demostraron que el protocolo no contiene errores de diseño. Falta aun evaluar el rendimiento del protocolo bajo distintos esquemas de funcionamiento sobre plataformas reales. Antes que nada, este protocolo no tiene requerimientos especiales a nivel de hardware o software, y en cuanto a su desempeño el costo mas importante es a nivel de la comunicación entre nodos. Si el protocolo se utiliza para soportar aplicaciones que involucren constantes actualizaciones de los datos, tendría tiempos de respuestas muy inferiores con respecto a su utilización en aplicaciones que se ejecuten bajo esquemas de consultas o con muy pocas actualizaciones.

Referencias Bibliográfícas

1. Aguilar J. y Leiss E.: "A Cache Memory System based on a Dynamic/Adaptive Replacement Approach". Revista Colombiana de Computación. Vol. 4, No. 1, (2003) 7-20.
2. Aguilar J. y Leiss E.: "Introducción a la Computación Paralela". Editorial Venezolana, Universidad de Los Andes, Mérida, 2004.
3. Aguilar J. y Leiss E.: "A Coherence-Replacement Protocol for Web Proxy Cache Systems". International Journal of Computers and Applications, Vol. 28. No. 1, (2006) 12-18.
4. Censier L. y Feautrier A.: "New Solution to Coherence Problems in Multicache Systems". IEEE, Trans. Comput, Vol. C(27), (1978) 1112-1118.
5. El-Ghazawi T.: "Introduction to Cache and Distributed Memory Concepts", Technical Report. The George Washington University. Estados Unidos, 2002.
6. Heinrich M.: "The Performance And Scalability of Distributed Shared Memory Cache Coherence Protocols". PhD Dissertation. Department of Electrical Engineering. Stanford University, 1998.
7. Hennessy J., Heinrich M. y Gupta A.: "Cache Coherent Distributed Shared Memory: Perspectives on Its Development and Future Challenges". Proceedings of the IEEE, Vol. 87, No. 3 (1999).
8. Vega M., Martín R., Zarallo F., Sánchez J. y Gómez J.: "SMPCache: Simulador de Sistemas de Memoria Caché en Multiprocesadores Simétricos". XI Jornadas de Paralelismo. Granada, 2000.
9. Chandra S., Richard B. y Larus J.: "Language support for writing memory coherence protocols". University of Wisconsin, Madison, WI, United States, 1996.
10. Shen X., Arvind y Rudolph L.: "CACHET: An Adaptive Cache Coherence Protocol for Distributed Shared-Memory". Technical Report. Massachusetts Institute of Technology. Proceedings of the 13th ACM-SIGARCH International Conference on Super Computing. Rhody, Greece, 1999.
11. Shen X., Arvind y Rudolph L.: "Commit-Reconcile & Fence (CRF: A New Memory Model for Architects and Compiler Writers)". Technical Report. Massachusetts Institute of Technology, 1999.
12. Archibald J. y Baer J.: "An Economical Solution to the Cache Coherence Problem, Technical Report". Department of Computer Science. University of Washington. Seattle. 1984.
13. Alexander T. y Kedem G.: "Distributed Prefetch-buffer/Cache Design for High Performance Memory Systems". Technical Report. Duke University, 1996.
14. Spin On line. Disponible en: http://spinroot.com/spin/whatispin.html.

Recibido el 05 de Mayo de 2006
En forma revisada el 14 de Mayo de 2007