Escalabilidad en DSM: Modelos Relajados de Consistencia

Rafael B. García Javier Echaiz* Jorge R. Ardenghi

LISiDi – Laboratorio de Investigación en Sistemas Distribuidos
Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur, Av. Alem 1253, (8000) Bahía Blanca, Argentina
Tel ++54 291 4595135 - Fax: ++54 291 4595136
{rbg,je,jra}@cs.uns.edu.ar

Resumen

Los sistemas de Memoria Compartida Distribuida (DSM) posibilitan disponer de una memoria compartida en un contexto de escalabilidad superior a los de memoria compartida convencional. El modelo de consistencia mas aceptado es el secuencial, por ser el mas afín con la intuicion del programador. Otros modelos, a partir de imponer determinadas características al programa, aseguran una ejecución secuencial pero con el beneficio de una implementación superior. En nuestra opinión, un aspecto que incide fuertemente en la escalabilidad y/o complejidad de un sistema es el de coherencia de memoria. Los modelos de consistencia secuencial, y muchos de los derivados de este último, requieren mantener coherente la cache. Esto obliga a una serialización de los accesos a locaciones individuales, lo cual se aplica tanto en arquitecturas UMA (protocolos de *snooping*) como NUMA (generalmente esquemas de directorio). Dado que típicamente los programas son sincronizados, y en la hipótesis de que se debe ser mas eficiente en los casos mas frecuentes, analizaremos los resultados de diferir la coherencia, con el objetivo último de alcanzar acceso inmediato, *fast access*, y a su vez no obstaculizar la optimización a nivel de los compiladores.

Palabras Clave: DSM Memoria Compartida Distribuida, Coherencia de Cache

1. Introducción

Actualmente se puede observar una clara tendencia hacia los sistemas de memoria compartida, sustentado en la potencia de cómputo incremental y la facilidad de programación. Así podemos citar arquitecturas exitosas tanto UMA (*Uniform Memory Access*) como los servers SUN Enterprise, como NUMA (*Non Uniform Memory Access*) como la SGI Origin 2000/3000. Estos sistemas mantienen la coherencia de cache entre los procesadores individuales utilizando un protocolo de coherencia por hardware destinado a asegurar una consistencia secuencial. Esto constituye una limitación fundamental en cuanto a lograr un sistema eficiente y escalable. Las restricciones que impone van mas allá del orden parcial definido por las operaciones de sincronización en un programa paralelo, de lo que se deduce imponen un overhead innecesario.

Una propiedad interesante de los algoritmos que implementan un modelo de consistencia es cuanto tiempo toma una operación de memoria. Si dicha operación no requiere que se termine ninguna comunicación, y por ende puede ser completada solo basada en el estado local del procesador del acceso, se dice que la operación es *fast*. Attiya y Welch demostraron que

*Becario de la Comisión de Investigaciones Científicas de la Provincia de Buenos Aires, Argentina.

ningún algoritmo secuencial puede garantizar una ejecución *fast* para todas sus operaciones. Esto claramente restringe la eficiencia de cualquier implementación de un modelo secuencial.

Resulta conveniente suministrar un modelo de consistencia de memoria que pueda ser entendido en todos los niveles de software y de hardware, apartándose de las definiciones originadas en un nivel muy primitivo de ejecución y que además no incluya la coherencia de memoria.

La coherencia obstaculiza estos objetivos. Se puede puntualizar en tal sentido la distinción artificial que introduce en cuanto al tamaño del dato, en tanto entiende con el tamaño fijado para la línea. Por otro lado, el orden total de todas las operaciones de memoria no puede manejarse a nivel del programa fuente, mas aun habrá operaciones a memoria en el código compilado que no resultan visibles a nivel fuente. El que los subprogramas deban observar todas las escrituras a una locación en el mismo orden dificulta una visión modular a nivel del programa fuente. Por último, un programa multithreaded compilado por un compilador secuencial obligará a recurrir a prueba y error, como ser insertando declaraciones de "volatil" para determinadas variables o deshabilitando optimizaciones del compilador, [8].

Los objetivos de interés en el manejo de cache se resumen en:

- El hardware no necesite tener conocimiento que el dato compartido está en la cache de múltiples procesadores. Esto posibilita el uso de microprocesadores comerciales.
- La solución soporte de forma eficiente el *false sharing* de los bloques de datos.
- La solución aproveche las posibilidades de los modelos de consistencia *delayed*. A partir de que la mayoría de los programas paralelo identifican explícitamente las operaciones de sincronización, estos podrán ser ejecutados con modelos de consistencia mas débiles, y conservar la intuición de ejecución secuencial.
- Deslinde al programador y al compilador de la necesidad de considerar que el dato compartido está replicado en múltiples memorias locales.
- Se alcance una naturaleza local, *fast access*, para los accesos a memoria.

2. La Consistencia en los DSM

Los sistemas actuales de memoria compartida están generalmente implementados por hardware. Esto significa que la coherencia entre los caches individuales es mantenida en forma automática y transparente por el sistema. Dependiendo de la arquitectura base pueden distinguirse dos grupos: pasivos basados en protocolos de snooping, y activos basados en algún esquema de directorio con información acerca del contenido de los caches remotos.

El problema con los pasivos es que al depender de la propiedad del tipo broadcast de un bus, heredan las limitaciones de los sistemas basados en bus. Además requiere que todo el tráfico global sea propagado en un ciclo del bus limitando la frecuencia del reloj de éste.

Los protocolos de coherencia activa evitan la dependencia de la observación de los eventos de coherencia sobre un bus común y por lo tanto habilitan multiprocesadores con memoria compartida no basada en bus. Se han planteado distintos métodos para manejar esta información, incluyendo directorios (Stanford DASH o la SGI Origin 2000/3000, o listas enlazadas mantenidas en hardware (Scalable Coherent Interface SCI). En cualquier caso es necesario propagar todas las actualizaciones a los procesadores que tienen réplicas de los datos, provocando comunicación global. Además el mantenimiento de los directorios o listas enlazadas requiere hardware dedicado capaz de tener acceso directo al bus del procesador/memoria. Esto generalmente prohibe el uso de componentes comunes y de bajo costo.

En nuestro trabajo investigamos como evitar estos problemas mediante el uso de consistencia relajada y arquitecturas de memoria compartida sin soporte de hardware para la coherencia.

Las alternativas examinadas en este trabajo están orientadas a los mecanismos de coherencia de los sistemas de memoria compartida. La arquitectura base que provee la abstracción de una memoria global en hardware en un espacio de direcciones físico o virtual siempre se mantiene. Esto debe verse en contraste con las arquitecturas de los sistemas SW-DSM, las cuales crean la abstracción de una memoria global en software. Los SW-DSM pueden funcionar sobre cualquier substrato de comunicación pero tienen que enfrentar una mayor complejidad en el software. El soporte en hardware es simple de lograr y resulta de una complejidad mínima para un eficiente pasaje de mensaje en la memoria distribuida. Este tipo de soporte entendemos irá en aumento en los sistemas introducidos por los principales fabricantes, tanto del tipo UMA como NUMA.

2.1. Ventajas de la Arquitectura

La ventaja más significativa de las arquitecturas no-coherentes, comparadas a sus contrapartidas coherentes, es que pueden contar con módulos de procesador/cache completamente independientes. Se elimina la necesidad de cualquier componente o infraestructura global y de la de interconexión subyacente capaz de manejar la abstracción de una memoria global, la cual puede reducirse a la capacidad de lograr accesos remotos a memoria directamente desde el procesador local. Dado que esto no impone una complejidad significativa en el hardware se puede obtener un sistema con una buena escalabilidad, similar a la obtenida en sistemas distribuidos, pero con la ventaja de contar con memoria compartida a nivel sistema.

Los requerimientos de la estructura de interconexión se verán significativamente reducidos, pues las complejas estructuras que antes mantenían la coherencia ya no son necesarias. El sistema de interconexión ya no necesita estar integrado a un controlador de coherencia o conectado al bus del sistema. Más aún podría utilizarse una *System Area Network* (SAN) común, tecnología usualmente empleada en clusters, en lugar de emplear interconexiones de mayor complejidad.

En resumen, el uso de sistemas de memoria compartida no-coherentes redunda en sistemas más simples y menos especializados. Además abre la puerta al desarrollo de arquitecturas NUMA basadas en técnicas provenientes del clustering, logrando una notoria reducción en los costos de hardware y de desarrollo, a la vez que una mayor escalabilidad.

3. Modelo no coherente Dag Consistency

En el MIT Blumofe, Frigo, Joerg, et al. propusieron un modelo relajado para el sistema multithreaded de computación paralela Cilk, básicamente un lenguaje multithreaded basado en C y un *runtime system*.

Un programa multithreaded define un orden parcial de ejecución el cual se podrá ver como un grafo acíclico Dag, *Directed Acyclic Graph*. Informalmente en este modelo una instrucción de lectura puede "ver" una de escritura sólo si existe algún ordenamiento serial en la ejecución del dag en el cual la lectura ve dicha escritura.

La primer definición del modelo apareció en la tesis de Joerg y establecía que una memoria compartida M de una computación multithreaded G = (V,E) es Dag consistente si:

- Toda vez que cualquier nodo v $\in V$ lee alguna locación l $\in M$ recibe un valor x escrito por algún nodo u $\in V$ tal que $v \not\prec u$

- Para tres nodos cualquiera u, v, w $\in V$ que satisfacen $u \prec v \prec w$, si v escribe alguna locación l $\in M$ y w lee l, w no recibe el valor escrito por u.

Dados u, v $\in V$, si existe un camino de longitud no nula desde la instrucción u a la instrucción v en G se dice que u estrictamente precede a v y se nota $u \prec v$. Dos instrucciones serán incomparables si se tiene que $u \not\prec v$ y $v \not\prec u$.

Mas aun, el modelo permite que diferentes lecturas devuelvan valores que estan basados en diferentes ordenamientos seriales, toda vez que estos sean consistentes con las dependencias del dag. Esta característica dio lugar a una segunda definición del modelo, a fin de confinar el no-determinismo.

Establecía [7] que la memoria compartida M de una computación multithreaded G=(V,E) es Dag consistente si para cada locación l $\in M$ existe una función f_l:V$\rightarrow V$ tal que se satisfacen las siguientes condiciones:

- Para toda instrucción u $\in V$, la instrucción f_l(u) escribe en l
- Si una instrucción u escribe en l, luego f_l(u) = u
- Si una instrucción u lee l recibe el valor asociado a f_l(u)
- Para toda instrucción u $\in V$ se tiene u$\not\prec f_l(u)$
- Dadas las instrucciones u, v, w tal que $u \prec v \prec w$, si f$_l$(v) $\neq u$ se verifica que f$_l$(w) $\neq u$

Informalmente f$_l$(u) representa la observación en el nodo u del contenido de la locación l.

El último punto de la definición, más precisamente la contrapositiva de éste establece que si f$_l$(w) = u luego f$_l$(v) = u. Resulta de tal manera más fuerte que la anterior, la incluye en el sentido que el nodo intermedio, en nuestro planteo v, podrá ser indistintamente de lectura o escritura (para forzar la relación), y no excluyentemente de escritura como es el caso de la primer definición. De esta forma se logra confinar el no-determinismo que se había detectado en el modelo Dag.

4. Modelo no-coherente Location Consistency

Gao y Sarkar trabajando en un contexto diferente definieron un modelo equivalente al cual denominaron Location Consistency, LC. El modelo establecía que una operación de lectura puede recibir cualquier elemento de un conjunto de "escrituras más recientes". Asocia a cada locación de memoria un multiset parcialmente ordenado *pomset*, el cual se irá actualizando durante la ejecución. Cada elemento que se ingresa al pomset corresponde o bien a una operación de escritura o a una de sincronización.

Las operaciones básicas previstas en el modelo son lectura, escritura, *acquire* y *release*. Estas dos últimas son de sincronización del acceso a una o más locaciones.

El estado de una locación l se denota E$_l$ = (S,$\prec$), en donde S es el multiset y $\prec$ es el orden parcial en S. Este orden parcial está determinado completamente por la concurrencia y la sincronización del programa en ejecución. De esta forma este modelo provee una visión uniforme de la consistencia de memoria en todos los niveles, software y hardware. Sólo el orden parcial del programa necesita ser obedecido, no existe el requerimiento adicional de coherencia referido a que todas las escrituras a una misma locación sean observadas en el mismo orden en todas las operaciones de lectura.

Cuando se ejecuta una instrucción de lectura w siguiendo este modelo, el valor que devuelve provendrá de una instrucción de escritura u tal que se verifica:

- u$\prec w$ y u es la escritura precedente más reciente con respecto a esa lectura. Esto es u$\prec w$ y $\nexists$ una instrucción v de escritura a la misma locación l que verifique v$\prec w$ y u$\prec v$, o bien
- La escritura u no está ordenada respecto a la lectura según el *pomset*, esto es, se cumple u$\not\prec w$ y w$\not\prec u$.

5. Comparación y Conclusiones

Se puede ver que la definición de la LC resulta equivalente a la primer definición dada para la Dag. En nuestra opinión la afirmación de que no confina el determinismo esgrimida para la primer definición dada para la Dag no resulta extensible a la LC. La razón está dada que en su definición Gao y Sarkar establecieron para las operaciones de sincronización *acquire* y *release* la atribución de exclusión mutua para las locaciones involucradas. Es en este contexto que sin condición de carrera en el programa no existiría el no-determinismo, y para las aplicaciones que realizan accesos no sincronizados a locaciones compartidas, mediante el empleo de mecanismos de tipo barrera, se lograría el propósito de confinar el no-determinismo.

Para las barreras en principio podríamos simular un *acquire* seguido de *release* en todos los procesadores que deban sincronizar. Se está trabajando a nivel del protocolo de cache para lograr reducir el overhead de esta operación, manteniendo como objetivo del diseño el favorecer los casos más frecuentes.

Referencias

[1] K. Li, P. Hudak. Memory coherence in shared virtual memory systems. *ACM Transactions on Computer Systems (TOCS)*, 7(4):321–359, 1989.

[2] B. Tangney, A. Judge, P. Nixon, V. Cahill, S. Weber. Overview of distributed shared memory. Technical Report, 1988.

[3] M. Feeley, W. Morgan, F. Pighin, A. Karlin, H. Levy, C. Thekkath. Implementing global memory management in a workstation cluster. *SOSP*, 201–212, 1995.

[4] R. Lottiaux, C. Morin. Containers: A sound basis for a true single system image. *First IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2001.

[5] A. Barak, O. La'adan, A. Shiloh. Scalable Cluster Computing with MOSIX for Linux. *Linux Expo '99*, 1999.

[6] Y. Khalidi, J. Bernabeu, V. Matena, K. Shirriff, M. Thadani. Solaris MC: a multi computer OS. *USENIX Annual Technical Conference*, 191–204, 1996.

[7] R. Blumofe, M. Frigo, C. Joerg, C. Leiserson, K. Randall. An Analysis of Dag-Consistent Distributed Shared-Memory Algoritms. *Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 1996.

[8] W. Pugh. The Java memory model is fatally flawed. *Concurrency: Practice and Experience*, 2000.