Consistencia de Memoria en Sistemas DVSM

Rafael B. García Javier Echaiz* Jorge R. Ardenghi

Laboratorio de Investigación de Sistemas Distribuidos (LISiDi)
Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Bahía Blanca - Buenos Aires - Argentina
e-mail: {je,rbg,jra}@cs.uns.edu.ar

Palabras clave: Shared Memory, SM. Distributed Shared Memory, DSM. Distributed Virtual Shared Memory, DVSM. Uniform Memory Access, UMA. Non-Uniform Memory Access, NUMA. Massively Parallel Processing, MPP. Secuential Consistency, SC. Scope Consistency ScC.

Introducción a los DVSM

Las arquitecturas paralelo, obtenidas combinando varios procesadores, pueden suministrar potencia de cómputo órdenes de magnitud superior a las arquitecturas de procesador único.

Un aspecto clave que diferencia a estas arquitecturas es el mecanismo de soporte para la comunicación. En las arquitecturas de pasaje de mensaje cada procesador tendrá su propia memoria local solo accesible por el mismo, requiriendo que los procesos se comuniquen a través de mensajes explícitos. Contrariamente, multiprocesadores con un espacio único de memoria, sistemas SM, al permitir que toda la memoria pueda accederse desde los distintos procesadores, posibilita una comunicación con operaciones de lectura y escritura en memoria.

Los sistemas SM, referidos usualmente como rigidamente acoplados, proporcionan una memoria global la cual será accesible por igual por todos los procesadores. Esto permite compartir datos a través de un mecanismo uniforme de lectura/escritura sobre la memoria común, UMA, aunando facilidad de programación y portabilidad. Como contrapartida estos sistemas estarán expuestos a un incremento en la contención y latencia de los accesos a memoria, afectando perfomance y escalabilidad.

Si bien con sistemas eficientes de caché e interconexiones con un gran ancho de banda se atenúan estas limitaciones al punto de posibilitar arquitecturas con varias decenas de procesadores, ejemplo Starfire Ultra 10000 con 24 a 64 procesadores, este esquema resulta inviable para combinar un número muy superior de procesadores.

Se tiene así la necesidad de distribuir la memoria, reteniendo el concepto de un espacio único de direcciones, dando lugar a los sistemas DSM. Una porción de la memoria total compartida se asigna a cada procesador, dando lugar a dos tipos de acceso a memoria: local y remoto. Mejora la escalabilidad pero incorpora un mecanismo no uniforme para la memoria por la disparidad entre los tiempos de acceso locales y remotos, en lo que se conoce como sistemas NUMA. Su limitación es la asignación estática de los espacios de direccionado.

A su vez los sistemas distribuidos, llamados normalmente multicomputador, corresponden a nodos de procesamiento con su memoria independiente, vinculados por alguna red de interconexión. Estos sistemas han demostrado ser escalables y ciertamente posibilitan sistemas de enorme potencia de cómputo, ejemplo los sistemas MPP.

Tradicionalmente, la comunicación entre procesos en un sistema distribuido se basa en el modelo de pasaje de datos. Sistemas con pasaje de mensajes o sistemas que soportan llamadas a procedimientos remotos, RPC, adhieren a este modelo. En sí extiende el mecanismo de comunicación del sistema: abstracciones como pórticos o mailbox junto con primitivas como Send y Receive son usadas para la comunicación entre procesos, pudiendo ocultar esa funcionalidad a través del lenguaje como con el mecanismo RPC. En cualquier caso los procesos pasan información por valor, contrariamente a esto, sistemas con un espacio de dirección compartido podrán hacerlo por referencia.

Resulta de interés retener en estos sistemas debilmente acoplados el paradigma de memoria compartida, dando lugar a los sistemas con memoria virtual distribuida compartida, DVSM.

La principal ventaja de un DVSM sobre el convencional de pasaje de datos es la abstracción mas simple y entendible que posibilita al programador. Permite una transición mas natural de las aplicaciones secuenciales a las distribuidas. En principio, cómputos paralelo y distribuido escritos para SM podrán ser ejecutados en DVSM sin cambios. El sistema de memoria compartida oculta el mecanismo de comunicación remota de los procesos permitiendo que estructuras complejas puedan ser pasadas por referencia, simplificando substancialmente la programación y facilitando la migración de procesos. Más aún, el dato en un DVSM podrá persistir mas allá del tiempo de vida del proceso que accede la memoria compartida.

El modelo de pasaje de mensajes fuerza al programador a ser consciente todo el tiempo del movimiento de datos entre procesos, por el manejo explícito de las primitivas de comunicación y los pórticos o canales. Además dado que el pasaje de datos es entre múltiples espacios de dirección, resulta difícil pasar estructuras complejas de datos. Las mismas deberán ser marshaled y unmarshaled por la aplicación. (Marshaling se refiere a la linearización y empaquetado de una estructura de datos en un mensaje).

Los sistemas DVSM comparten metas con una variedad de sistemas como por ejemplo memoria caché de los procesadores de un sistema SM, memoria local en los sistemas NUMA, caché distribuida de los file systems y base de datos distribuidas entre otros. Particularmente, todos ellos intentan minimizar el tiempo de acceso a datos que son potencialmente compartidos y que serán mantenidos consistentes. Aunque similares a distancia, tendrán cuestiones de detalle e implementación que varían significativamente por la diferencia en el costo y uso de sus parámetros.

Clasificación de los sistemas DVSM

La gran atención que ha merecido éste tipo de arquitecturas y el importante número de trabajos de investigación y desarrollo relacionado con el tema lo podemos entender por dos cuestiones convergentes:

1. El gran desarrollo de los microprocesadores, tanto a nivel de su arquitectura como de la tecnología de circuitos integrados de gran escala VLSI

2. El desarrollo de las redes de interconexión del tipo local y de sistemas, que proporcionan significativos anchos de banda en la comunicación

Lo anterior hace potencialmente competitiva la combinación de cientos o miles de estos microprocesadores potentes y de reducido costo por su producción de gran escala.

Convengamos que un DVSM posibilita el paradigma de programación de memoria compartida en sistemas que poseen memoria físicamente distribuida, mas especificamente sistemas sin un espacio de direcciones único. Esta abstracción se logra con acciones específicas para acceder desde los diversos nodos a los datos del espacio virtual global compartido. Hay tres cuestiones fundamentales para la perfomance que se refieren a las acciones para traer el dato al sitio donde es requerido y a su vez mantenerlo consistente con respecto a los otros procesadores, las que dan a su vez espacio para clasificarlos según

- Cómo se realiza el acceso **Algoritmo**
- En dónde el acceso es implementado **Nivel de implementación**
- Cúal es la semántica de consistente **Modelos de consistencia**

Algoritmo de soporte de la memoria compartida

El primer tópico da lugar a diferentes algoritmos de memoria compartida según la existencia o no de múltiples copias de un dato y de sus permisos de acceso pudiendo aplicarse a la distribución de datos la estrategia de replicación o migración.

- SRSW single reader/single writer
  1. Sin migración
  2. Con migración
- MRSW multiple reader/single writer
- MRMW multiple reader/multiple writer

La replicación no está permitida en SRSW mientras que podrá darse en los algoritmos MRSW y MRMW. Resultan fundamentales en el desempeo del algoritmo las características particulares de la aplicación. Si bien los algoritmos predominantes son MRSW, los MRMW aunque más complejos de implementar, posibilitan un superior nivel de concurrencia que en ciertas implementaciones resultará decisivo.

El aspecto negativo de la perfomance en los algoritmos que implementan memoria distribuida es que son sensitivos al comportamiento de las aplicaciones. No se tendrá un único algoritmo que funcione aceptablemente en un amplio rango de aplicaciones. Esto sugiere un esfuerzo para adoptar el algoritmo apropiado dada una aplicación. Más aún, podría incrementarse la perfomance significativamente con sintonía fina sobre cómo la aplicación realiza el acceso a memoria o sintonía del propio algoritmo para el particular comportamiento de la aplicación, aunque eliminando la ventaja de transparencia en los accesos a memoria compartida.

Nivel de la Implementación

Como se realiza el acceso es una de las decisiones mas importantes, dada su incidencia tanto en la programación como en la perfomance y costo. Se tendrá:

- Software
  - Librerías de run time
  - Sistema Operativo

    2. Fuera del kernel
  - Compilador
- Hardware
- Combinación Hardware/Software

Las implementaciones por software son normalmente librerías de run time que serán linkeadas con la aplicación que usa dato compartido, ejemplo Trademark, o esta abstracción podrá realizarse a nivel de lenguaje de programación, dado que el compilador puede detectar accesos compartidos e insertar llamadas a rutinas de sincronización y coherencia. Otro tipo de solución se alcanza incorporando un mecanismo DSM en el sistema operativo distribuido, dentro o fuera del kernel. A menudo se combinan elementos de distintas alternativas, por ejemplo IVY que es una runtime library que incluye modificaciones al SO.

Las implementaciones de hardware buscan superar las limitaciones propias de las alternativas de software, como velocidad y granularidad de los accesos (tipicamente de una página) entre otras. Así, podemos mencionar MERLIN que a través de una memoria reflectiva mapeada dinamicamente permite al usuario compartir datos a relativa alta velocidad y con baja latencia, de manera automática y concurrente con el cómputo. Otro ejemplo de este nivel de implementación es el KSR1 que plantea, inspirado en lo que fue en su momento el advenimiento de la memoria virtual, algo análogo para las arquitecturas MPP, en el sentido que el usuario se desentienda del movimiento entre nodos tanto de los datos como del código, en lo que sería entre dos niveles para el ejemplo de memoria virtual.

Referido a la alternativa híbrida encontramos que soluciones de hardware, como COMA, se complementan con software como en Simple-COMA. En general se podrán hacer los casos infrecuentes por software para minimizar complejidad, o alternativamente a las soluciones de software incorporarles hardware para acelerar las operaciones frecuentes.

Modelos de consistencia

Dado que un sistema de memoria compartida permite que múltiples procesadores lean y escriban simultáneamente la misma locación de memoria, los programadores requieren un modelo conceptual para la semántica de la operación de memoria que le permita usarla correctamente. Esto se conoce como modelo de consistencia de memoria o modelo de memoria. Este modelo deberá ser intuitivo y fácil de usar. Normalmente optimizaciones a nivel de arquitectura y de compilador que posibilitan eficiencia en un espacio de direcciones único complicarán el comportamiento de memoria llevando a que diferentes procesadores tengan distinta visión de la memoria compartida. El problema en diseñar un sistema de memoria compartida se sintetiza en presentar al programador una visión del sistema de memoria que sea simple de usar y aun así posibilite las optimizaciones necesarias para un soporte eficiente del espacio de memoria compartido. Un tercer factor de importancia resulta ser la portabilidad.

Muchos de los modelos propuestos fallan en ofrecer una semántica razonable de programación, o bien sacrifican perfomance por la programabilidad.

En el estudio y evaluación de los diversos modelos tres cuestiones deberán ser tenidas en cuenta: primero cómo el modelo se presenta al programador y cómo este impacta tanto a la programación como a la portabilidad, luego las restricciones que el modelo impone en el sistema y las técnicas para una implementación eficiente, y por último la perfomance del modelo y complejidad requerida en la implementación para alcanzar dicha perfomance.

Los programadores prefieren razonar con un modelo simple e intuitivo como es el SC, luego el

información sobre las operaciones de memoria en una ejecución secuencialmente consistente. Esta le permite al sistema determinar optimizaciones en el ordenamiento que pueden ser explotadas sin violar la consistencia secuencial. Mas aún, la misma información podrá ser usada para portar el programa a distintas implementaciones, de manera automática y eficiente.

Lineamientos del Trabajo

Estamos trabajando en un sistema híbrido que, si bien es por software aprovechando los mecanismos de memoria virtual, introduce hardware para la actualización automática. Comparte con Brazos el trabajar con el modelo ScC pero se diferencian en que este último es por software unicamente y no trabaja con el concepto de home para las páginas. Las ventajas de la implementación en estudio son reducción significativa de la estructura de datos y de procesamiento, con perspectiva de mejora en perfomance y escalabilidad.

Referido a perfomance se reduce el trabajo a realizar dentro de las zonas críticas, muy conveniente por el efecto serializador en situaciones de gran contención, además de desaparecer efectos secundarios como cache polution. En cuanto a escalabilidad tiene la ventaja de que la estructura de datos escalaría con la complejidad del problema y no con el tamaño del cluster, como ocurre tipicamente en este tipo de sistemas. Además, el protocolo en cuestión menejaría satisfactoriamente aplicaciones con granularidad fina, por el prefetch implícito a través de la actualización automática.

Se está trabajando en su posible simulación y además en la posibilidad de manejar home dinamicamente para hacerlo más eficiente, sopesando la eventual sobrecarga inducida.

Referencias

[1] David Mosberger: Memory Consistency Models. TR 93/11 Department of Computer Science, The University of Arizona

[2] Pete Keleher, Alan L. Cox, and Willy Zwaenepoel. Lazy Release Consistency for Software Distributed Shared Memory, Departament of Computer Science, Rice Univesity

[3] Liviu Iftode, Cezary Dubnicki, Edward W. Felten and Kai Li. Improving Release Consistent Shared Virtual Memory using Automatic Update, 2nd IEEE Symposium on High-Perfomence Comuter Architecture, February 1996

[4] Liviu Iftode, Jswinder Pal Singh and Kai Li. Scope Consistency: a Bridge between Release Consistency and Entry Consistency, 8vo Annual ACM SPAA96

[5] Evan Speight and John K. Bennett. Brazos: A Third Generation DSM System, Proceedings of the First USENIX Windows NT Workshop, August, 1997

[6] Mathias Blumrich, Kai Li, Richard Alpert, Cezary Dubnicki, Edward Felten, and Jonathan Sandberg. Virtual Memory Mapped Network Interface for the SHRIMP Multicomputer, Proceedings of the 21st International Symposium on Computer Architecture, April 1994

[7] Creve Maples and Larry Wittie. MERLIN: A superglue for Multicomputer Systems, COMCON'90