Tolerancia a Fallos es Sistemas de Memoria Compartida Distribuida

Mario Leandro Bertogna
Universidad Nacional del Comahue
mlbertog@uncoma.edu.ar

Introducción

A medida que es necesario mas poder de cálculo, aumenta la demanda de sistemas con múltiples procesadores, pero su desventaja es la programación que requiere gran esfuerzo y habilidad. Debido a esto, gran parte de su éxito depende del paradigma de programación que ofrezcan estos sistemas. Hay distintos métodos para mejorar la eficiencia y facilitar el uso de los sistemas paralelos, uno de estos es la *memoria compartida distribuida* (MCD).

Una memoria distribuida compartida es una abstracción que presenta un solo espacio de direcciones compartidos por un número de procesadores. Cualquier procesador puede acceder a los espacios de memoria directamente. Los administradores de mapeo de memoria implementan el mapeo entre las memorias locales y el espacio de direcciones compartido. Además de mapear, una de las principales responsabilidades de los administradores es mantener la coherencia todo el tiempo.

Un sistema de memoria compartida distribuida es atractivo desde el punto de vista del programador, debido a que simplifica varios aspectos en los desarrollos de aplicaciones. Sin embargo, a medida que el número de componentes crece, la probabilidad de fallas del sistema se incrementa en la misma medida.

Un sistema de MCD es robusto o tolerante a fallas si soporta la supervivencia a fallos de la aplicación de usuario. El mecanismo puede ser transparente a la aplicación o puede proveer un conjunto de primitivas para que la aplicación los utilice. Para implementar un sistema de MCD completamente confiable éste debe preservar una imagen consistente de la memoria compartida (los datos globales) y proveer mecanismos para que cada aplicación sea recuperada de forma consistente (estados de los procesos y datos privados). Distinguiremos entre: *recuperables;* que son los sistemas que aunque sobrevivan a fallos pueden tener inconsistencias; y *confiables;* que son los que garantizan consistencia e integridad.

Tolerancia a Fallos en MCD

A pesar de que un sistema completamente confiable provee al usuario de una preservación completa del estado de la aplicación (consistencia de memoria y procesos), un administrador de procesos tolerante a fallas realiza ésto de una forma completamente diferente. El principal objetivo del administrador de procesos es la supervivencia de los procesos, mientras que el principal objetivo de un sistema de MCD tolerante a fallas es la supervivencia de los datos compartidos. El mecanismo de recuperación del sistema de MCD puede ser implementado en el núcleo del sistema operativo o en el espacio de la aplicación. En la implementación sobre el núcleo, la preservación del sistema requiere mas atención, el impacto de la falla en éste siempre será de mayor importancia que en una aplicación del usuario.

Un punto importante en los sistemas de MCD basado en el núcleo es la correcta recuperación y la integridad de las estructuras del sistema operativo. Un segundo objetivo es la recuperación de las aplicaciones de MCD. Cuando un sitio con un sistema de MCD falla, pueden ocurrir los siguientes problemas:

Pérdida del estado interno o servicio de Directorio en los sistemas de MCD: En un sistema de MCD, la consistencia y la integridad de ciertas estructuras de datos internas nunca debe ser comprometida. Por ejemplo, en un sistema de MCD basado en páginas, un servicio de directorio provee la ubicación de estas. Si se considera la pérdida de la integridad del directorio del sistema, parte de los datos del usuario

almacenado estriarían inaccesibles, aunque los datos estuvieran disponibles en la red. Es esencial para construir un sistema confiable un servicio de directorio distribuido que provea acceso.

Pérdida de disponibilidad de datos en el sistema de MCD: Este problema puede ocurrir cuando un sitio con MCD pierde la disponibilidad de los datos. En algunos sistemas de MCD se mantienen copias replicadas de las páginas disponibles. En los sistemas que usan protocolos de actualización generalmente existen copias de las páginas compartidas que puede ser usadas para recuperación. En los sistemas que usan protocolos de invalidación, donde solo un sitio tiene la página actualizada en la cache, la probabilidad de falla afectando la disponibilidad de los datos es una función de la cantidad de lecturas y escrituras, mas específicamente si ocurren mas escrituras, la probabilidad de que los datos no estén disponibles aumenta.

Bloqueo infinito de procesos: Todos los sistemas de MCD dependen del subsistema de comunicación que envía los requerimientos de una manera confiable. Sin embargo, los sistemas de comunicación confiables pueden quedar bloqueados, si se pierden los datos compartidos, por ejemplo. Algunos sistemas tolerantes a fallas asumen que solo a los clientes se le permite enviar requerimientos y estos son enviados una sola vez, otros permiten que estos se reiteren. Estas dos opciones generan diferentes metodologías de diseño para sistemas de MCD. En la primer opción se usa un sistema llamado *non- timeout page fault*, un proceso de usuario que requiere una página se bloquea después de hacer el requerimiento. A pesar que se asume el requerimiento confiable será retransmitido por los niveles inferiores del sistema de comunicación, en este tipo de sistemas el cliente nunca reanuda su ejecución si el requerimiento falló. La ventaja de este método es que los administradores estándar no necesitan ser modificados debido a que el sistema continuará retransmitiendo el pedido, su desventaja es que otro sitio que reemplace al caído tendrá que responder para que el proceso pueda continuar su ejecución. De esta manera se pierde demasiado tiempo reconstruyendo un nuevo sitio con la información disponible en el sistema. Una posible solución es mantener copias consistentes, pero esto requiere tiempo de ejecución. En el segundo diseño, inicialmente el proceso esta bloqueado, sin embargo si la respuesta no es satisfecha dentro de un lapso, un mecanismo de timeout despertará al proceso bloqueado. El proceso bloqueado reintentará el pedido, posiblemente a otro sitio. Este tipo de sistemas es llamado *client-retry page fault system*. La ventaja de este método es la fácil reconstrucción de las estructuras de datos, como las colas de requerimientos. Si se considera una falla en un sitio que almacena la cola de requerimientos, el cliente solo reenvía el pedido al sitio recuperado o al reemplazo. Debido a que todos los clientes reenviaran sus requerimientos, las colas compartidas pueden ser reconstruidas factiblemente.

Inconsistencia de los datos luego de la recuperación: El cuarto problema es la validación de los datos después de la recuperación. No será necesario realizar siempre la implementación del chequeo de la consistencia en los datos recuperados, a pesar de que puede ser deseable. Algunas veces la integridad del núcleo de sistema operativo puede ser más importante que la consistencia de la aplicación. Solo se requiere consistencia en algunas partes del sistema distribuido que cooperan y mantienen los estados compartidos globales después de una falla del sistema. Es de gran importancia asegurar la consistencia del estado del sistema de MCD, no así la consistencia de los datos del usuario.

Como ejemplo de una aplicación donde habría beneficio de la recuperación, considerese una lista enlazada ordenada en memoria compartida distribuida. Varios procesos intentan encolar y desencolar sus ítems. El ordenamiento de la lista no necesariamente es determinístico debido a que los productores y consumidores actualmente usan la lista. Sutiles cambios en el algoritmo administración pueden influenciar radicalmente el orden de los ítems en la lista. Si el sitio falla, no significa que se necesita una recuperación consistente. Mientras que sería inaceptable que la lista permaneciera inaccesible, sería suficiente asegurar que todos los ítems en la cola pueden ser recuperados en algún orden, debido a que el ordenamiento de la lista fue creado por una serie de eventos no deterministicos.

Propuesta de Implementación

Como implementación se propone utilizar un sistema de MCD mediante software basado en objetos llamado Java Party[1], este sistema agrega objetos remotos a Java solo con la declaración sin las desventajas de la comunicación explicita de sockets y la sobrecarga programación de RMI, y la mayoría

de los problemas del método de pase de mensajes. Esta orientado al los clusters de estaciones de trabajo y por esto combina la programación Java con los conceptos de memoria compartida distribuida en redes heterogéneas. Originalmente este sistema no implementa replicación, un tema de vital importancia en los sistemas de MCD, habiendo una propuesta de implementación de esto en [2].

Uno de los trabajos mas conocidos sobre MCD confiable es el de Wu y Fuchs[3,4] que examinaron un esquema de puntos de chequeo transparente al usuario para implementar memoria virtual compartida distribuida recuperable. Su esquema permite al sistema comenzar el cálculo desde un punto de chequeo y no requiere que el sistema comience en forma global. Se usa la replicación de discos. Este sistema de almacenamiento se llama páginas-gemelas porque hay dos páginas en discos diferentes por cada página compartida en la memoria global. Su trabajo permite a los procesos transparentes de los puntos de chequeo ser integrados con el protocolo de coherencia de cache realizando un rápido roll-back. Esto contrasta con la técnica de deshacer todos los cambios desde el último punto de chequeo. Los autores solucionan la dificultad de realizar los puntos de chequeo y la recuperación, debido a que los procesadores solo mantienen una porción del espacio de direcciones. En particular, antes de que un procesador permita que sean vistas las actualizaciones locales por los demás procesadores, todas las páginas modificadas por el procesador son enviadas a almacenamiento estable y se marca un punto de chequeo.

Este sistema utiliza un algoritmo de administración similar a un monitor que mantiene al propietario y a las copias en una estructura de datos llamada *Info* en almacenamiento estable. Todos los sitios también mantienen sus tablas de páginas locales indicando los permisos de cada una de ellas en el espacio de direcciones global. También usa la técnica de escritura invalidación para la coherencia de memoria. Integrando el protocolo de coherencia y el esquema de puntos de chequeo, se produce la siguiente secuencia de eventos: mientras se administran las faltas de página, el propietario realiza un punto de chequeo antes de permitir que otro sitio tome la página. Esto se realiza si la página ha sido modificada desde el último punto de chequeo.

En el caso de la implementación las páginas que se mencionan serían reemplazadas por objetos y el monitor se menciona en la implementación del sistema de MCD como el Runtime Manager. Durante la liberación (release) de las variables según la implementación de JavaParty-R[2] se utilizaría el código para realizar el proceso de copia en el almacenamiento estable y marcar el punto de chequeo.

Conclusión

El objetivo de esta investigación es analizar los diferentes problemas que surgen luego de una falla en los sistemas de memoria compartida distribuida, plantear distintos métodos de tolerancia a fallos y como estos afectan los recursos y el rendimiento del sistema, tomando como experiencia el sistema Java Party-R.

Bibliografía

[1] M. Philippsen and M. Zenger. "JavaParty - transparent remote object in Java". In Concurrency: Practice & Experience, Volume 9, pp 1225-1242, 1997.

[2] M. Leandro Bertogna "Tesis de Magíster: Memoria Compartida Distribuida". Universidad Nacional del Sur, 2000.

[3] Kun-Lung Wu and W. Kent Fuchs. "Recoverable Distributed Shared Virtual Memory". IEEE Transactions on Computers, 39(4):pp.460-469,1990.

[4]B. Janssens and W. K. Fuchs. "Relaxing consistency in recoverable distributed shared memory". In Proceedings of the Twenty-Third Annual International Symposium on Fault-Tolerant Computing: Digest of Papers, pp 155-163,1994.