Single System Image: Pilar de los Sistemas de Clustering

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 (8000), Argentina
T.E.: 0291–4595135 (Ext. 2616 y 2605)
{je,jra}@cs.uns.edu.ar

Resumen

El objetivo principal de este trabajo es comparar diferentes sistemas operativos distribuidos. En especial se busca comparar las distintas alternativas, ventajas y potenciales de SSI implementado en los distintos niveles de un sistema con soporte para clustering. Los resultados de este análisis brindarán el marco necesario para futuras investigaciones y desarrollos en este campo.

Palabras Clave: SSI, sistemas operativos distribuidos, clustering.

1. Introducción

Es evidente que una aplicación no accede de forma directa a los recursos físicos (procesadores, memoria, discos, placas de red), en vez de ello el sistema operativo brinda a la aplicación una vista lógica de dichos recursos (procesos y threads, espacios de direccionado, archivos, sockets).

Si cada nodo de un cluster utiliza su propio sistema operativo, los recursos serán locales a cada nodo, y no será simple lograr que dos procesos se comuniquen mediante un archivo compartido o un área de memoria compartida. Además, la migración de un proceso producirá una pérdida de su entorno, sin mencionar que los usuarios y programadores deberán encargarse de la distribución de los recursos en un ambiente de estas características.

Una forma elegante de resolver este problema es presentar una colección de recursos físicos como un recurso único y compartido. Ejemplo de ello es un sistema de archivos jerárquico el cual permita a cada nodo acceder a cada archivo. Una "ilusión" de este tipo es lo que usualmente se conoce como *Single System Image (SSI)* y posibilita que el usuario y las aplicaciones tengan una vista global de los recursos disponibles, sin importar a qué nodo específico del cluster estén físicamente asociados.

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

1.1. Servicios y Beneficios

Un cluster SSI es capaz de brindar los siguientes servicios:

- *Punto de entrada único:* un usuario puede conectarse al cluster como si fuese un host virtual (e.g. telnet beowulf.cs.uns.edu.ar) sin importar que el cluster tenga múltiples nodos con servicio de login. El sistema distribuye hacia los diferentes nodos las conexiones provenientes de los usuarios en forma transparente y con el objetivo de balancear la carga.

- *Interfase de usuario única:* el usuario debe poder utilizar el cluster empleando una única GUI. En lo posible debe ser una interfase conocida para el usuario, similar a la de CDE (Solaris, AIX, Tru64, etc.) o a la de la familia Microsoft Windows.

- *Espacio de procesos único:* todo proceso de usuario, sin importar en que nodo reside tiene un PID (*process ID*) único dentro del cluster. Un proceso puede crear nuevos procesos (`fork`) en el mismo nodo o en uno remoto. Además debe ser posible la comunicación entre cualquier par de procesos mediante señales y pipes. Obviamente para la existencia de este servicio debe implementarse cierto manejo global de recursos y control de procesos como si se tratase de procesos que corren localmente en un nodo.

- *Espacio de direcciones de memoria:* los usuarios "ven" la ilusión de una gran y única memoria principal, que por supuesto en realidad está compuesta por la suma de las memorias locales de los distintos nodos. El paradigma DSM (implementado en software) tiene éste como su objetivo principal. Alternativamente existen soluciones a nivel del compilador, las cuales apuntan a distribuir las estructuras de datos de una aplicación entre los distintos nodos. Este item sigue presentándose como un desafío interesante al área de computación distribuida, principalmente se trata de atacar el problema de la performance y de la independencia de las arquitecturas/S.Os. entre otras.

- *Espacio de I/O único:* este servicio permite que cualquier nodo efectúe operaciones de I/O en dispositivos (o discos) tanto locales como remotos. Mediante un diseño de este tipo se pueden lograr por ejemplo sistemas RAID (traducción de números de dispositivos/bloques mediante funciones matemáticas) en software.

- *File system único:* el usuario "ve" un gran file system (global) donde se integran al file system local los discos/disposivos remotos. Ejemplos de este servicio incluyen NFS, AFS, GFS, xFS, Solaris MC Proxy, MFS y oMFS.

- *Networking virtual global:* esto significa que desde cualquier nodo se puede acceder a cualquier conexión de red aún en el caso en el que la red no esté físicamente conectada a todos los nodos del cluster.

- *Sistema global de manejo de tareas:* un proceso de un usuario debe poder ejecutarse en cualquier nodo del cluster e incluso se debe poder "solicitar" la ejecución en varios nodos. Para lograr este objetivo se debe contar con un *scheduler* global capaz de planificar ejecuciones batch, interactivas y paralelas. Ejemplos de clusters que brindan este servicio incluyen GLUnix, LSF y CODINE.

- *Administración global:* el cluster (como un todo) y cada nodo puede ser configurado, monitoreado, testeado y controlado mediante un único conjunto de herramientas e interfases (GUI).

- *Checkpointing y migración de procesos:* el checkpointing es un mecanismo de software que permite almacenar en forma periódica el estado de un proceso (el cual por supuesto además del PCB incluye cómputos intermedios en memoria y disco). Esta técnica posibilita recuperación (*rollback*) frente a fallas. La migración de procesos complementa el checkpointing y es necesaria para un balance dinámico de la carga.

2. Implementación de SSI en los Diferentes Niveles

Las dos características más importantes de SSI son las siguientes:

1. Cada implementación de SSI tiene un límite, i.e., dentro de estos límites el sistema se comportará como una supercomputadora clásica, fuera de ellos será un conjunto de máquinas interconectadas.

2. Un sistema puede implementar SSI en distintos niveles, donde un nivel inferior puede simplificar la implementación de uno superior.

La implementación de SSI puede efectuarse en uno o más de los siguientes niveles:

- *Hardware*. Ejemplos de este nivel son el *Memory Channel* de Digital[1], técnicas SMP y los sistemas de *hardware distributed shared memory* (HW-DSM).

- *Sistema Operativo* (conocido también como *Underware*). Ejemplos de implementaciones a nivel del kernel incluyen al SCO UnixWare NonStop Cluster [WS99], Sun Solaris MC [KBM+95], GLUnix [GPR+98], Nomad [PB99], MOSIX [BL98] y OpenMosix [Bar01].

- *Middleware* (subsistemas en tiempo de ejecución). Ejemplos de implementaciones a este nivel incluyen entornos de programación tales como PVM [Sun90], *job management* y sistemas de scheduling tales como CODINE [Fer99] y Condor [LLM98], y sistemas basados en Java (JVM) como JESSICA [MWL00].

- *Aplicación*. A modo de ejemplo podemos incluir en esta clase a PARMON [Buy00], una herramienta (GUI) que ofrece una ventana única que representa todos los recursos o servicios disponibles. Otro ejemplo es el Linux Virtual Server (LVS) [Zha00].

En general toda implementación importante de SSI debe contar con la colaboración de todos los niveles para facilitar la implementación.

2.1. Ventajas y Desventajas de Cada Nivel

Cada nivel de implementación de SSI tiene sus propios pros y contras. El nivel de hardware ofrece el mayor nivel de transparencia, pero debido a la rigidez de su arquitectura no ofrece la flexibilidad requerida para extender y mejorar el sistema. Otro problema importante de este nivel es la dificultad económica de trabajar con arquitecturas reales pasada la etapa de simulación. El nivel de kernel ofrece SSI a todos los usuarios (desarrolladores de aplicaciones y usuarios finales). Sin embargo, la programación a nivel del kernel no brinda un ambiente "cálido" para desarrollar y mantener debido a que su mercado es y probablemente seguirá siendo limitado,

[1] Compaq en la actualidad.

además es difícil mantenerse al día con las emergentes innovaciones tecnológicas presentes en los sistemas operativos de mercado masivo.

El nivel de aplicación ayuda parcialmente en la implementación de SSI pero necesita que cada aplicación sea desarrollada con soporte SSI, presentando aquí una limitación muy importante. Una ventaja clave de este nivel comparada con el nivel de kernel es que la implementación puede realizarse en etapas y el usuario puede beneficiarse inmediatamente, a diferencia del nivel del kernel; pues a menos que todos los componentes sean desarrollados específicamente para soportar SSI, éste no puede ser utilizado en el mercado. El nivel medio (middleware) es una solución de compromiso entre los dos mecanismos anteriormente mencionados para proveer SSI. En algunos casos, como en el de PVM, cada aplicación debe implementarse empleando APIs especiales. Esto significa que hay un costo mayor de implementación y mantenimiento, de otra forma el usuario no puede obtener ningún beneficio a partir del cluster.

3. Conclusiones y Trabajos Futuros

SSI puede mejorar notablemente la aceptabilidad y utilidad de los clusters ocultando la existencia física de los múltiples nodos, presentándolos como un recurso único y unificado. SSI puede implementarse usando técnicas de software o de hardware, cada una con sus propias ventajas y desventajas. Si bien el nivel middleware parece ofrecer una mejor escalabilidad comparado con las otras estrategias, es evidente que no es capaz de brindar soporte completo de SSI como lo hace el nivel de sistema operativo. Es entonces el nivel del sistema operativo el que será escogido en futuras investigaciones y desarrollos[2]. No obstante es interesante mencionar nuevamente la importancia de un ambiente cooperativo entre los distintos niveles, lo que redundará probablemente en un sistema con capacidad SSI a nivel del S.O. pero con cierto grado de soporte de aplicaciones/librerías a nivel del usuario.

Cabe destacar como nota final que en cualquier caso, los diseñadores de software (sistema o aplicación) para clustering deben siempre considerar el soporte de *SSI* (*transparencia*) como uno de los objetivos de diseño más importantes además de los evidentes objetivos de una mayor performance y/o disponibilidad.

Referencias

[Bar01] Moshe Bar. OpenMosix Internals. *OSDN Open Source Development Network*, 2001.

[BF99] Mark Baker y Geoffrey Fox. Metacomputing: Harnessing informal supercomputers. En Rajkumar Buyya, editor, *High Performance Cluster Computing*, volume 1, Architectures and Systems, páginas 154–185. Prentice Hall PTR, Upper Saddle River, NJ, 1999. Chap. 7.

[BL98] Amnon Barak y Oren La'adan. The MOSIX multicomputer operating system for high performance cluster computing. *Future Generation Computer Systems*, 13(4–5):361–372, Marzo 1998.

[Buy00] Rajkumar Buyya. PARMON: a portable and scalable monitoring system for clusters. *Software, Practice and Experience*, 30(7):723–739, 2000.

[Ech02] Javier Echaiz. Survey: Single System Image. Reporte Técnico, Universidad Nacional del Sur, Dpto. Cs. e Ing. de la Computación, 2002.

[2]Un punto de partida sobre esta línea de investigación se desarrolla en [EGA02].

[EGA02] Javier Echaiz, Rafael García y Jorge Ardenghi. El paradigma DSM como base de un kernel distribuido. En *Proceedings of the 8vo Congreso Argentino de Ciencias de la Computación (CACIC '02)*, Octubre 2002.

[Fer99] Fritz Ferstl. Job and resource management systems. En Rajkumar Buyya, editor, *High Performance Cluster Computing*, volume 1, Architectures and Systems, páginas 499–518. Prentice Hall PTR, Upper Saddle River, NJ, 1999. Chap. 20.

[GLS94] W. Gropp, E. Lust y A. Skjellum. Using MPI: Portable Parallel Programming with the Message-Passing Interface. *MIT Press, Cambridge, MA.*, 1994.

[GPR+98] Douglas P. Ghormley, David Petrou, Steven H. Rodrigues, Amin M. Vahdat y Thomas E. Anderson. GLUnix: A Global Layer Unix for a network of workstations. *Software, Practice and Experience*, 28(9):929–961, 1998.

[HCW+99] Kai Hwang, Edward Chow, Cho-Li Wang, Hai Jin y Zhiwei Xu. Designing SSI Clusters with Hierarchical Checkpointing and Single I/O Space. *IEEE Concurrency*, 7(1):60–??, 1999.

[HX98] K. Hwang y Z. Xu. *Scalable Parallel Computing: Technology, Architecture, Programming.* WCB/McGraw-Hill, New York, 1998.

[KBM+95] Y. Khalidi, J. Bernabeu, V. Matena, K. Shrriff y M. Thadani. Solaris MC: A multi-computer OS. SMLI TR-95-48, Sun Microsystems Laboratories, 1995.

[LLM98] M. Litzkow, M. Livny y M. Mutka. Condor: A hunter of idle workstations. En *Proceedings of the 8th International Conference of Distributed Computing Systems*, 1998.

[MWL00] Matchy J. M. Ma, Cho-Li Wang y Francis C. M. Lau. JESSICA: Java-enabled single-system-image computing architecture. *Journal of Parallel and Distributed Computing*, 60(10):1194–1222, 2000.

[PB99] E. Pinheiro y R. Bianchini. Nomad: A scalable operating system for clusters of uni and multiprocessors, 1999.

[Pfi98] Gregory F. Pfister. *In search of clusters: The Ongoing Battle in Lowly Parallel Computing, 2nd ed.* Prentice Hall, Englewood Cliffs, NJ, 1995 edition, 1998. :1998 edition: ISBN 0-13-899709-8.

[Sun90] V. S. Sunderam. PVM: a framework for parallel distributed computing. *Concurrency, Practice and Experience*, 2(4):315–340, 1990.

[WPE+92] B. Walker, G. Popek, R. English, C. Kline y G. Thiel. The LOCUS distributed operating system. En Akkihebbal L. Ananda y Balasubramaniam Srinivasan, editores, *Distributed Computing Systems: Concepts and Structures*, páginas 145–164. IEEE Computer Society Press, Los Alamos, CA, 1992.

[WS99] B. Walker y D. Steel. Implementing a full single system image unixware cluster: Middleware vs. underware. En *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '99)*, 1999.

[Zha00] W. Zhang. Linux virtual server for scalable network services. En *Ottawa Linux Symposium 2000*, 2000.