Carga Compartida en Sistemas Distribuidos Heterogéneos

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

Un sistema distribuido está compuesto de nodos, posiblemente heterogéneos, conectados mediante una red. Un sistema de esta clase puede utilizarse efectivamente sólo si el software es capaz de presentar al usuario el concepto de *single system image* (SSI) sobre el sistema físicamente distribuido. De esta forma todos los recursos de un nodo deberían poder accederse transparentemente desde cualquier otro nodo.
En esta línea de investigación se plantea la problemática y la implementación de un protocolo de migración de procesos como componente fundamental de un sistema distribuido con soporte de carga compartida.

Palabras Clave: carga compartida, balance de carga, migración de procesos, sistemas distribuidos.

1 Introducción

El desarrollo de microprocesadores de elevado poder de cómputo y bajo costo y las redes de alta velocidad propiciaron en las dos últimas décadas un cambio en el paradigma de la computación. Los grandes *mainframes* de ayer fueron reemplazados por clusters de pequeñas pero potentes computadoras conectadas mediante redes de alta velocidad. El usuario, en lugar de trabajar en una terminal boba conectada por un cable serie al mainframe, ahora tiene en su escritorio una poderosa computadora unida con pares mediante una red. Potencialmente todos los recursos que físicamente se encuentran en cualquier computadora están disponibles a cualquier usuario del sistema. Sin embargo este potencial no puede aprovecharse completamente a menos que los usuarios puedan acceder transparentemente a estos recursos. La transparencia, en este contexto, significa que los usuarios deben poder acceder a cualquier recurso sin preocuparse (y de hecho, sin siquiera saber) su ubicación física. Este es el objetivo de un sistema operativo distribuido, crear la ilusión en la mente de los usuarios de un sistema de tiempo compartido único en lugar de una colección de máquinas independientes pero conectadas.

Algunos de los sistemas más destacados son Amoeba [MvRT+90], Locus [PWC+86], Sprite [DO91], etc. La mayoría de estos sistemas trabajan corriendo una copia del mismo sistema operativo en cada computadora participante, y estas copias a su vez, cooperan para proveer SSI [EA03] a sus usuarios.

Sin embargo, si bien conceptualmente estos sistemas resultan atractivos, no presentaron un crecimiento real en su popularidad. Esto se debe principalmente al hecho de que ya existe

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

una gran cantidad de usuarios y software de base para UNIX; por lo tanto muchos sistemas distribuidos tratan de emular UNIX para que las aplicaciones existentes sigan funcionando con pocos o sin modificaciones, manteniendo un entorno de trabajo familiar a los usuarios.

Por otra parte el software de los sistemas distribuidos actuales debe resolver otro problema. El problema surge debido a que los sistemas distribuidos actuales están compuestos de una gran variedad de hardware (provisto por diferentes fabricantes) y de software de base (sistema operativo). Lograr SSI sobre esta heterogeneidad constituye un gran reto. Los sistemas como Amoeba, etc. no tratan de resolver el problema de la heterogeneidad de sistemas operativos, simplemente asumen que todas las máquinas participantes del cluster corren el mismo S.O. Hoy en día existen soluciones parciales a este problema, e.g. SunNFS [SGK+85] y AFS [MSC+86] a nivel del sistema de archivos distribuido con vista unificada. Otro recurso comúnmente compartido en los UNIX actuales son las impresoras.

Sin embargo el CPU *no* suele ser compartido transparentemente en los sistemas UNIX. Existen varios estudios que muestran que existe una gran disparidad en la carga de los nodos de un sistema distribuido en cualquier instante de tiempo. Mientras algunas máquinas están sobrecargadas otras se encuentran completamente ociosas. Es entonces deseable que estas máquinas puedan compartir la carga y así los usuarios podrían observar una mejora en la performance del sistema. Si bien existen comandos UNIX a nivel de usuario (e.g. `rsh`) que les permiten ejecutar sus procesos remotamente en una máquina a su elección, claramente estos mecanismos no son transparentes. En un ambiente ideal un usuario debería simplemente ejecutar una tarea y el sistema automáticamente seleccionar su mejor ubicación (i.e., la máquina menos cargada) para ejecutarla. Una forma más estricta de carga compartida es el llamado *balance de carga*, donde el sistema trata de equiparar la carga de todas las máquinas en todo momento. Mientras que con carga compartida el sistema simplemente debe seleccionar la mejor máquina para ejecutar una tarea recientemente disparada y transparentemente transferirla a esa máquina, con balance de carga típicamente se debería migrar una tarea a otra máquina, posiblemente *durante* su ejecución. Estas dos formas de migración de procesos se conocen como *migración no apropiativa* (o ejecución remota) y *apropiativa* respectivamente.

Obviamente la migración no apropiativa es más sencilla de implementar que la apropiativa. Esto se debe a que para implementar migración apropiativa, el sistema debe efectuar un *checkpoint* en el estado del proceso mientras se encuentra en ejecución y transferir dicho estado a la máquina destino. Claramente este problema no tiene una fácil resolución si las máquinas poseen arquitecturas y/o sistemas operativos diferentes, pues el checkpoint debería efectuarse a nivel del código fuente del programa y no a nivel del código ejecutable. Tui [SH96] es el ejemplo paradigmático de un sistema que permite que procesos en ejecución migren a sistemas con arquitecturas diferentes. Sin embargo, en este caso el proceso de migración no es transparente al programa de aplicación, i.e., la aplicación debe colaborar con el software de migración para que funcione correctamente. Por otra parte la tarea de migración es más costosa en términos de tiempo, pues la totalidad del estado (que puede ser de tamaño considerable) debe transferirse a la máquina destino. Hay estudios que muestran que este overhead adicional restringe severamente los beneficios en la performance que pueden obtenerse mediante la migración apropiativa frente a la no apropiativa [ELZ88]. La ejecución remota no incurre en este costo adicional debido a que únicamente las tareas recientemente disparadas son transferidas a otras máquinas, y por lo tanto no hay espacio de direcciones a transferir. Adicionalmente, en este caso, el problema de la heterogeneidad es más simple de resolver.

La carga compartida trae aparejados dos temas relacionados. El primero se relaciona con las políticas de migración. Por ejemplo, cuándo debe un nodo tratar de transferir un proceso a

otro nodo, qué proceso debe migrarse y a cuál máquina. El segundo tema está constituido por los mecanismos de transferencia de procesos, los cuales aseguran que la tarea migrada obtenga un entorno aproximadamente igual al de la máquina que lo originó.

2 Políticas de Migración para Carga Compartida

La planificación (*scheduling*) de tareas en un sistema distribuido de carga compartida involucra decidir no solamente cuándo ejecutar un proceso, sino también dónde ejecutarlo. Para ello la planificación se lleva a cabo mediante dos componentes: el *distribuidor* (políticas de distribución) y el *planificador* (políticas de planificación). El distribuidor decide dónde se ejecutará una tarea y el planificador dictamina cuándo una tarea recibe su porción de CPU. Típicamente cada nodo de un sistema distribuido tiene su propio planificador responsable de organizar los procesos en el(los) procesador(es) local(es), mientras que las decisiones de alto nivel de asignar un proceso a un nodo son tomadas por el distribuidor. Aunque existen pequeñas variaciones [Ous82], este esquema es el que surge naturalmente en un sistema distribuido. Esto se debe a dos motivos, el primero es que cada nodo tiene su sistema operativo propio encargado de planificar procesos, el segundo es la modularidad: los diseñadores pueden concentrarse mejor en los temas relativamente complejos de la distribución de la carga sin necesidad de involucrarse con los detalles de planificación. Las políticas de distribución tratan de repartir la carga total del sistema a sus nodos individuales transfiriendo procesos entre los nodos. Las políticas de planificación simplemente eligen un proceso del conjunto de procesos listos a ejecutarse en un nodo de forma tal de maximizar el *throughput*. La política de planificación de procesos de cualquier sistema operativo tradicional puede funcionar como una política de planificación del sistema distribuido. En realidad la planificación es efectuada por el S.O. por si mismo y por lo tanto debemos concentraremos principalmente en la política de distribución.

La política de distribución no realiza la migración de procesos, sino que colabora con el mecanismo de migración proveyendo información, e.g. cuándo un nodo debe tratar de migrar un proceso, qué proceso es factible para ser transferido, a cuál nodo debe enviarse el proceso seleccionado, etc. Una política de carga compartida puede dividirse en cuatro componentes basados en cuatro características funcionales distintas:

Política de transferencia. Decide cuándo un nodo es apto para una transferencia de procesos, tanto como emisor o como receptor.

Política de selección. Una vez que la política de transferencia decide que el nodo es emisor esta política elige el proceso local a migrar.

Política de ubicación. Encuentra un nodo par (*peer*) apropiado (emisor o receptor) para un nodo declarado emisor o receptor por la política de transferencia.

Política de información. Recolecta la información de estado del sistema necesaria para la toma de decisiones de distribución de la carga.

3 Mecanismos de la Migración de Procesos

La semántica de procesos UNIX no es simple de extender a un entorno distribuido debido a dos razones: el diseño contempla únicamente el caso de un nodo único e independiente, y más importante aún existe una fuerte relación entre el subsistema de procesos y otros subsistemas UNIX [Shi97]. A continuación mencionaremos brevemente algunos de los problemas que trae aparejada la extensión de un sistema UNIX convencional a un sistema distribuido:

Estado del proceso. Incluye el espacio de direcciones del usuario, información de control, credenciales, variables de entorno y contexto de hardware.

Migración apropiativa vs. no apropiativa. Debemos considerar ambas alternativas mientras trabajamos en el diseño pues esta decisión impactará directamente en la implementación y funcionalidad del sistema.

Implementación a nivel de usuario vs. kernel. Al igual que en el caso anterior, ésta constituye una decisión importante que influirá en la performance global del sistema y en la dificultad de la implementación.

Sistema de archivos. La mayoría de los sistemas asumen la presencia de un sistema de archivos distribuido que presenta una vista uniforme.

Descriptores de archivos. Un sistema UNIX presenta cuatro abstracciones para comunicar un proceso con el mundo externo: archivos, dispositivos, sockets, y pipes. La capacidad de nuestro sistema para acceder remotamente a estas abstracciones determinará qué procesos pueden ser tomados en cuenta como candidatos para la migración.

Credenciales de usuario. Cada usuario de un sistema UNIX tiene asignado un *user id* único y puede pertenecer a uno o más grupos, donde cada grupo tiene asignado a su vez un *group id* único. Evidentemente mantener la igualdad entre usuarios y grupos en cada nodo facilitará la tarea de migración.

Process id. Preservar la unicidad del *process id* (`pid`) en un sistema UNIX clásico es una tarea simple, sin embargo mantener esta propiedad cuando tenemos más de un nodo implica particionar de alguna forma el espacio de *ids*. Es necesario entonces que el process id sea único a nivel del cluster, que sea simple de generar y que sea fácil encontrar un proceso en el cluster a partir de su `pid`.

Relaciones entre procesos. Mantener la relación entre un proceso padre y su(s) hijo(s) constituye un desafío interesante en la extensión de un sistema convencional, pues el proceso padre y el hijo pueden estar ahora ejecutándose en nodos distintos. Debemos considerar también en este caso la preservación de la semántica de las señales (*signals*), mecanismo asincrónico clásico de los UNIX para notificar procesos.

Estandarización. Diversos parámetros deben preacordarse para posibilitar la heterogeneidad a nivel del sistema operativo. Algunos de ellos son: número de las señales, uso de recursos, prioridades, timers, límites, etc.

Tolerancia a fallas. Si bien la mayoría de las implementaciones actuales no direcciona (al menos no completamente) el problema de la tolerancia a fallas, es una característica fundamental en algunos sistemas; en ciertos casos llega incluso a tener mayor relevancia que la maximización de la performance global y por lo tanto debe considerarse.

4 Trabajos Futuros

Si bien existe una gran diversidad de opciones al considerar la implementación de un sistema operativo distribuido con capacidad de carga compartida, es nuestra intención implementar un sistema que cumpla con los siguientes objetivos: heterogeneidad a nivel del S.O., migración no apropiativa, cambios mínimos en la semántica UNIX, escalabilidad, eficiencia, y tolerancia a fallas (a nivel del nodo y de la red). Otro objetivo perseguido es mantener independientes los mecanismos de la migración de procesos y las políticas de carga compartida, i.e., dados los

mecanismos se debería poder emplear cualquier política de carga compartida. Esta posibilidad permitirá explorar las ventajas/desventajas de diferentes políticas según las necesidades particulares del sistema a considerar.

Para la implementación de este sistema se definirán ciertas convenciones que deben respetarse a nivel del cluster. Por ejemplo, la forma de asignar `pid`s, señales, etc. Además el sistema empleará UDP como protocolo de comunicación subyacente, y se asumirá la presencia de NFS (para el sistema de archivos distribuidos) y NIS (para mantener la unicidad de credenciales de usuario). Si bien el sistema todavía se encuentra en etapa de diseño, se espera tener una implementación Linux que siga estos lineamientos. Por otra parte dicha tarea servirá para extender y reevaluar el sistema a partir de la experiencia.

Bibliografía

[AEGC03] Jorge Ardenghi, Javier Echaiz, Rafael García y Karina Cenci. Migración de Procesos, Memoria Compartida Distribuida y Sistemas Multiagentes. *V Workshop de Investigadores en Ciencias de la Computación, WICC 2003*, páginas 32–35, Mayo 2003.

[DO91] F. Douglis y J. Ousterhout. Transparent Process Migration: Design Alternatives and the Sprite Implementation. *Software Practice & Experience*, 1991.

[EA03] Javier Echaiz y Jorge Ardenghi. Single System Image: Pilar de los Sistemas de Clustering. *V Workshop de Investigadores en Ciencias de la Computación, WICC 2003*, páginas 210–214, Mayo 2003.

[ELZ88] Derek L. Eager, Edward D. Lazowska y John Zahorjan. The Limited Performance Benefits of Migrating Active Processes for Load Sharing. *Conf. on Measurement & Modelling of Comp. Syst., ACM SIGMETRICS*, páginas 63–72, Mayo 1988.

[MSC+86] James H. Morris, Mahadev Satyanarayanan, Michael H. Conner, John H. Howard, David S. Rosenthal y F. Donelson Smith. Andrew: A Distributed Personal Computing Environment. *Communications of the ACM*, 29(3):184–201, Marzo 1986.

[MvRT+90] S. J. Mullender, C. van Rossum, A. S. Tanenbaum, R. van Renesse y H. van Stavern. *Amoeba: a distributed operating system for the 1990s.* IEEE Computer, Mayo 1990.

[Ous82] John K. Ousterhout. Scheduling Techniques for Concurrent Systems. En *Third International Conference on Distributed Computing Systems*, páginas 22–30, Mayo 1982.

[PWC+86] G. Popek, B. Walker, J. Chow, D. Edwards, C. Kline, G. Rudisin y G. Thiel. LOCUS: A Network Transparent, High Reliability Distributed System. En *ACM Joint Conf. on Comp. Perf. Mod., Meas. and Eval.*, 1986.

[SGK+85] Russel Sandberg, David Goldberg, Steve Kleiman, Dan Walsh y Bob Lyon. The Design and Implementation of the Sun Network File System. En *Proceedings of the USENIX Summer Conference*, páginas 119–130, Berkeley, CA, USA, Junio 1985.

[SH96] Peter Smith y Norman C. Hutchinson. Heterogeneous Process Migration: The Tui System. Reporte Técnico TR-96-04, Department of Computer Science, University of British Columbia, Febrero 1996.

[Shi97] Ken Shirriff. Building Distributed Process Management on an Object-Oriented Framework. En *Proceedings of the USENIX Annual Technical Conference (USENIX-97)*, páginas 119–132, Berkeley, Enero 6–10 1997. Usenix Association.