Michael Nelson, Brent Welch, John Ousterhout. University of California

Caching in the Sprite Network File System

CARLOS MARTÍN FLORES GONZÁLEZ, Carné: 2015183528

Instituto Tecnológico de Costa Rica
Maestría en Computación
Sistemas Operativos Avanzados
Profesor: Francisco Torres Rojas, Ph.D

Overview of Sprite. Un sistema operativo implementado por la Universidad de California en Berkeley como parte del desarrollo de SPUR, un *workstation* multiprocesador de alto rendimiento. La interfase que Sprite provee a los procesos de usuario es muy parecido al que se proporciona en UNIX. Aunque aparenta ser similar a UNIX en función, el kernel ha sido reimplementado con el fin de proporcionar mejor integración con la red. La implementación de Sprite está basado en una facilidad de llamadas a procedimientos remotos kernel-a-kernel, lo que permite a kernels en diferentes *workstations* a solicitar servicios de unos y otros usando un protocolo similar al descrito en Birrell y Nelson. El sistema de archivos de Sprite user el mecanismo de RPC extensivamente para administración de caché.

Basic Cache Design. *Caches on Disk or in Main Memory:* en Sprite se pone en caché los datos de los archivos en la memoria principal. Razones: (1) Caché en memoria principal permite que los *workstations* no tengan discos (*diskless*) lo que los hace más baratos y silenciosos. (2) los datos pueden ser accedidos más rápidamente desde un caché en memoria principal que un caché en disco. (3) Memorias físicas en los *workstations* cliente son los suficientemente grandes para proveer altos tasas de acierto. (4) Los cachés del servidor estarán en memoria principal sin importar de dónde estén localizados los caché del cliente: al usar caché en memoria principal también para los clientes se está en la habilidad de construir un único mecanismo de *caching* para ser usado por ambos, servidores y clientes. *Cache Structure:* Los cachés de Sprite están organizados en base a bloques que usan un tamaño fijo de 4Kb. Los bloques de caché se tratan virtualmente, usando un identificador único proporcionado por el servidor y un número de bloque dentro del archivo. Se usan direcciones virtuales en lugar de direcciones físicas de disco para que los clientes puedan crear nuevos bloques en sus cachés sin contactar al servidor primero para encontrar sus direcciones físicas. Direcciones virtuales también permiten que los bloques en el caché sean localizados sin atravesar el archivo de mapa de disco. Para archivos accedidos de forma remota, los cachés de los clientes mantienen solo bloques de datos. Los servidores también pueden poner en caché mapas de archivos y otra información de administración de disco. Estos bloques se direccionan en el cache usando las direcciones de disco físicas de los bloques junto con un "identificador de archivo" especial correspondiente al dispositivo físico. **Writing Policy:** *Delay write-backs:* los bloques son inicialmente escrito solamente al caché y luego escrito a través del disco o el servidor un tiempo después. Para Sprite se escogió una política *delayed-policy* similar a la que se usa en UNIX: cada 30 segundos, tosos los bloques sucuios que no hayan sido modificados en los últimos 30 segundos son escritos de vuelta. Un bloque escrito en un cliente será escrito en el caché del servidor en 30-60 segundos, y será escrito en disco en 30-60 segundos más. Esta política evita retrasos cuando se escriben archivos y permite reducciones modestas en el tráfico disco/servidor, mientras que se limita el peligro que puede ocurrir en una caida.

Cache Consistency: Permitir a los clientes el poner archivos en caché presenta un probleam de consistencia: qué pasa si un cliente modifica un archivo que es también es puesto en caché por otros clientes? Pueden subsecuentes referencias de otros clientes al archivo retornar datos "obsoletos"? Sprite permite *write-sharing* secuencial y concurrente. Se garantiza que cuando un proceso lee datos de un archivo, se recibe los datos escritos más recientes, sin importar de cuándo o dónde los datos fueron escritos. Distinción entre consistencia y sincronización correcta: cada lectura retorna los datos más recientes. Sin embargo, el mecanismo de consistencia de caché no puede garantizar que las aplicaciones concurrentes realicen sus lecturas y escrituras en un orden sensible. Si el orden importa, las aplicaciones tienen que sincronizar sus acciones en el archivo usando *system calls* para bloque del archivo u otro mecanismo de comunicación disponible. La consistencia de caché simplemente elimina los problemas de la red y reduce el prolema a lo que estaba en sistemas de tiempo compartido. Sprite usa los servidores de archivos como un punto de control centralizado para consistencia de cache. Cada servidor garantiza consistencia de caché para todos los archivos en sus discos, y los clientes solamente tienen que lidiar con servidor para un archivo: no hay una interacciones cliente-client directas. *Concurrent Write-Sharing:* *Write-sharing* concurrente ocurre en un archivo cuando se abierto en múltiples clientes y al menos uno de ellos lo tiene abierto para escritura. Sprite lidia con esta situación desabilitando el caché del cliente para el archivo, de esta forma todos las lecturas y las escrituras del archivo van a ir a través del servidor. Cuando el servidor detecta que un *write-sharing* concurrent está por ocurrir en un archivo, toma dos acciones: (1) notifica al client que tiene le archivo abierto para escritura, diciéndole que no tiene que escribir todos los bloques sucios de vuelta en el servidor. (2) El servidor notifica a todos los cliente que tiene el archivo abierto, diciéndoles que el archivo ya no es candidato a ser puesto en caché. Una vez que estas dos acciones se llevan a cabo, los clientes enviarán todos los accesos para ese archivo al servidor. El kernel del servidor serializa los accesos a su caché, produciendo un resultado idéntico a correr todos los procesos cliente en un sola máquina de tiempo compartido. *Sequential Write-Sharing:* ocurre cuando un archivo es modificado por un cliente, cerrado y luego abierto por algún otro cliente. Problemas potenciales: (1) cuando un cliente abre un archivo podría tener bloques no actualizados en su caché. Para resolver el problema, los servidores mantienen un número de versión para cada archivo, el cual es incrementado cada vez que el archivo es abierto para escritura. Cada cliente mantiene la versión de todos los archivos en su caché. Cuando un archivo es abierto, el cliente compara la versión del servidor con la suya. Si difieren, el cliente limpia el archivo de su caché. (2) los datos actuales de un archivo podrían estar en el caché de otro cliente. Los servidores manejan esta situación manteniendo un rastro del último que escribió en dicho archivo; este cliente es el único que potencialmente puede tener bloques sucios en su caché. Cuando un cliente abre un archivo el servidor notifica el último escrito. Esto asegura que el cliente que lee recibirá la información más actualizada cuando solicite bloques del servidor. *Simulation Results:* En las simulaciones de tráfico que se mostró que el caché en el cliente reduce el tráfico del servidor en más del 70 % y en tasas de acerito de más del 80 %.

Virtual Memory and the File System. Sprite le permite que cada caché de archivo crecer y encogerse dinámicamente en respuesta a sus demandas de cambios en la memoria virtual de la máquina y en el sistema de archivos. El módulo del sistema de archivos y el módulo de memoria virtual administran un *pool* separado de páginas de memoria virtual. La memoria virtual mantiene sus páginas en orden LRU[1] aproximado a través de una version del algoritmo de reloj presentado por Michael Nelson.[2] El sistema de archivos mantiene sus bloques de caché en order LRU perfecto debido a que todos los accesos a bloques son a través de los *system calls* de "read" y "write". Cada sistema mantiene un *time-of-last-access* para cada página o bloque. Cuando un módulo necesita memoria adicional, compara la edad de sus página más antigua con la edad de la página más antigua de otro módulo. Si el otro módulo tiene la página más antigua, entonces se le fuerza a renunciar a esa página; de otra forma el módulo recibla su página más antigua. Este enfoque presenta dos problemas: (1) *Double-caching*,

[1]Last Recently Used
[2]En el artículo *Virtual Memory for the Sprite Operating System.*

puede ocurrir porque la memoria virtual es un usuario del sistema de archivos: almacenamiento de respaldo es implementado usando archivoso ordinarios y código de solo lectura el cargado bajo demanda directamente desde archivos ejecutables. Una implementación ingenua podría causar que las páginas que son leídas desde archivos de respaldo terminen en ambos, el caché del archivo y el *pool* de memoria virtual de página. Páginas que son eliminadas del *pool* de memoria virtual de página podrían simplemente ser movidas a los cachés de archivo, donde tendrían que envejecer por otros 30 segundos antes de ser enviados al servidor. Para evitar estas ineficiencias, el sistema de memoria virtual omite el caché local del archivo cuando lee y escribe en archivos de respaldo. (2) El segundo problema con la negociación entre la memoria virtual y el sistema de archivos ocurre cuando las páginas de memoria virtual son los suficientemente grandes como mantener varios bloques de archivos. Es el tiempo LRU de una página en el caché de archivo la edad del bloque más antiguo en la página, la edad del bloque más reciente en el página, o el promedio de edad de los bloques en la página? Cuando se ha determinado cuál página dar de vuelta a la memoria virtual, qué se debería de hacer con los otros bloques en la página si han sido accedidos recientemente? En la implementación de Sprite, que tiene páginas de 8Kb pero bloques de archivo de 4Kb, se usó una solución simple: la edad de la página es la edad del bloque más reciente en el página y cuando una página es abandonada todos los bloques son removidos.

Benchmarks. *Micro-benchmarks:* El primer conjunto de *benchmarks* midió el tiempo requerido para una invocación local y remore de tres operaciones comunes de búsqueda (*lookup*) en archivos. Las versiones remotas tomaron 3-4 veces más tiempo que las versiones remotas. Aproximadamente la mitad del tiempo transcurrido para las operaciones remotas fue gastado en ejecución de CPU en el servidor. El segundo conjunto de *benchmarks* midieron el rendimiento de la lectura y la escritura crudadel sistema de archivos de Sprite leyendo o escribiendo un archivo grande de forma secuencial. En los resultados se obtuvo que en el mejor caso, el caché del cliente permite a un programa de aplicación correr como mucho 6-8 veces más rápido que sin caché del cliente. El otro resultado importante es que un cliente puede leer y escribir al caché del servidor an aproximadamente la misma velocidad que un disco local. En la implementación de Sprite el caché del servidor es el doble de rápido que un disco local, pero esto es porque la política de configuración de disco de Sprite solamente permite que un bloque sea leído o escrito por revolución de disco. *Macro-benchmarks:* (1) *Application Speedups:* Sin caché de cliente, las máquinas sin disco fueron de 10-15 % más lentas que aquellas con discos. Con caché habilitado y un *warm-start*, las diferencias entre máquinas sin disco y aquellas con discos fue muy pequeña. En el peor caso del 12 %. (2) *Server Load:* uno de los mayores beneficios del caché del cliente es la reducción de la carga que se pone en los CPUs del server. En general, un cliente sin disco y sin caché utilizó cerca del 5-20 % del CPU del servidor. Con caché de cliente, la utilización del servidor cayó en un factor de dos o más a 2-9 %. (3) *Network Utilization:* conforme las velocidades de CPU aumentan el acho de banda de las redes se convierte en un problema. Sin caché de cliente los *benchmarks* promediarion 6.5 % de utilizacio4n de la red Ethernet de 10Mb. Cuando hay caché del cliente se reduce la utilizacio4n de la red cerca de un factor de cuatro, y en promedio cerca de un 1.5 % en los *benchmarks.* (4) *Contention:* Sin caché de cliente, se da una degradación significativa del rendimiento cuando más que unos clientes fueron activados a la vez. Con siete clientes y ningún caché, los clientes fueron ejecutados dos veces y media más lentos, el servidor fue utilizado casi el 80 % y la red fue utilizada más de un 30 %. Con caché de cliente y siente clientes activos, cada uno corrió a una velocidad dentro del 30 % de lo que podría haber logrado con un disco local. La utilización del servidor en esta situación estuvo cerca del 50 % y la utilización de la red fue solo de 5 %.

1. ¿CUÁL ES EL PROBLEMA QUE PLANTEA EL *PAPER*?

El artículo describe un mecanismo de distribución simple para *caching* de archivos en una red de *workstations* que fue implementado como parte del sistema operativo Sprite.

2. ¿POR QUÉ EL PROBLEMA ES INTERESANTE O IMPORTANTE?

Los cachés han sido utilizados en muchos sistemas operativos para mejoar el rendimiento del sistema de archivos. Típicamente, el *caching* es implementado al retener en la memoria principal algunos de los bloques de disco que hay sido accedidos recientemente. Accesos repetidos a un bloque en el caché puede manejado sin involucramiento del disco, lo que tiene dos ventajas: (1) reduce los retrasos: un bloque en el caché puede ser usualmente retornado a un proceso en espera de 5 a 10 veces más rápido que uno que tiene que ser recuperado del disco. (2) *Caching* reduce la contención por recursos del disco, lo cual podría ser ventajoso si varios procesos están intentando acceder a archivos en el mismo disco.

3. ¿QUÉ OTRAS SOLUCIONES SE HAN INTENTADO PARA RESOLVER ESTE PROBLEMA?

4. ¿CUÁL ES LA SOLUCIÓN PROPUESTA POR LOS AUTORES?

5. ¿QUÉ TAN EXITOSA ES ESTA SOLUCIÓN?