Mendel Rosenblum y John Ousterhout. University of California at Berkeley

The Design and Implementation of Log-Structured File Systems

CARLOS MARTÍN FLORES GONZÁLEZ, Carné: 2015183528

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

Se presenta una nueva técnica para gestión de almacenamiento de disco llamada *log-structured file system*, en donde se escriben todas la modificaciones a disco secuencialmente en un estructura similar a una bitácora (*log-like*), de este modo se aceleran las escrituras en disco y la recuperación a fallos. La bitácora es la única estructura en disco, contiene información indexada de tal forma que los archivos pueden ser leídos de vuelta desde la bitácora de forma eficiente. Se ha construido un prototipo llamado Sprite LFS, el cual está ahora en producción como parte del sistema operativo Sprite.

Design for file systems on the 1990's. El diseño de sistemas de archivos está gobernado por dos fuerzas: tecnología, que proporciona un conjunto de bloques básicos, y la carga de trabajo (*workload*), que determina un conjunto de operaciones que tiene que ser llevadas cabo de forma eficiente.

Technology: componentes de tecnología significatios en el diseño de sistemas de archivos: procesadores, discos y memoria. Los procesadores son significativos debido a que su velocidad está incrementando en casi una tasa exponencial. La tecnología de los discos está mejorando rápidamente pero no en áreas de rendimiento. Hay dos componentes de rendimiento de disco: ancho de banda de la transferencia y tiempo de acceso. El ancho de banda de la transferencia puede ser mejorado substancialmente con el uso de arreglos de discos y cabezas paralelas de discos pero no hay mejoras significativas en el tiempo de acceso. El tercer componente es la memoria, que está incrementando a tasas exponenciales. Los sistemas modernos ponen en caché datos de archivos utilizados recientemente en la memoria principal y, memorias principales más grandes hacen posible poner en caché archivos más grandes.

Workloads: las cargas de trabajo de los sistemas de archivos son diferentes dependiendo de las aplicaciones. Las aplicaciones de oficina e ingeniería tienen a ser dominadas por accesos de archivos pequeños. Para cargas de trabajo dominados por accessos a archivos grandes existen una serie de técnicas para asegurar que tales archivos permanezcan en el disco de manera secuencial, de tal forma que el rendimiento de I/O tiende a verse limitado por ancho de banda del I/O y memoria en lugar de políticas de asignación de archivos.

Log-structured file systems. La idea fundamental de un sistema de archivos *log-structured* es mejorar el rendimiento de escritura aplicando *buffering* en una secuencia de cambios del sistema de archivos en el caché del archivo y luego escribiendo todos los cambios en disco secuencialmente en una sola operación de escritura. La información escrita en disco en la operación de escritura incluye bloques de datos del archivo, atributos, bloques de índices, directorios y casido toda la otra información usada para administrar el sistema de archivos. Para cargas de trabajo que contiene mucho archivos pequeños, un sistema de archivos *log-structured* convierte las escrituras aleatorias sincrónicas de sistemas de archivos tradicionales en una transferencia secuencial asincrónica grande que puede utilizar cerca del 100 % del ancho de banda del disco. Problemas clave a ser resueltos: cómo recuperar la información de la bitácora y cómo gestionar el espacio libre en el disco de tal forma que grandes extensiones de espacio libre estén siempre disponibles para escribir nuevos datos.

File location and reading: Sprite LFS escribe estructuras de índices en la bitácora para permitir recuperaciones de accesos aleatorios. Las estructuras básicas usadas por Sprite LFS son idénticas to aquellas usadas en el Unix FFS: para cada archivo existe una estructura de datos llamada *inode*, las cual contiene los atributos de los archivos, además de direcciones de disco del los primeros 10 bloques del archivos. Sprite LFS usa una estructura de datos llamada *inode map* para mantener la localización actual de cada inode. Dado el número de identificación de un archivo, el *inode map* debe ser indexado para determinar la

dirección del disco del inode. El *inode map* se divide en bloques que son escritos en la bitácora; una región de punto de verificación fija en cada disco identifica las localizaciones de todos los bloques *inode maps*. Afortunadamente, los *inode maps* son los suficientemente compactos para mantener las porciones activas en caché en la memoria principal: las búsquedas de *inode maps* raramente requieren accesos a disco.

Free space management: segments la parte de diseño más difícil para sistemas de archivos *log-structured* es la gestión de espacio libre. La meta es la de mantener grandes extensiones libres para escribir datos nuevos. Opciones del sistema de archivos: (1) Uso de *threading* → causa fragmentación y no hace posible que un sistema de archivos *log-structured* sea más rápido que uno tradicional. (2) *copying* → desventaja, su costo. Particularmente en archivos de largas duración. Sprite LFS usa una combinación de *threading* y *copying*. El disco se divide en extensiones grandes de tamaño fijo llamados segmentos (de tamaño 512Kb y 1Mb). Un segmento es siempre escrito secuencialmente de inicio a fin, y todos los datos vivos deben ser copiados de un segmento antes que el segmento sea re-escrito. El tamaño del segmento se selecciona lo suficientemente grande de tal forma que el tiempo de transferencia para leer o escribir un segmento completo sea mucho mayor que el costo de una búsqueda al inicio de un segmento. Esto le permite que las operaciones de segmento-completo corran a un ancho de banda cercano al total del disco, sin importar el orden en que los segmentos so accedidos.

Segment cleaning mechanism: El proceso de *copying* datos en vido a un segmento es llamado *segment cleaning*. En Sprite LFS es un proceso de tres pasos: leer un número de segmentos en memoria, identificar los datos en vivo y escribir los datos en vivo de vuelta a un número más pequeño de segmentos limpios. Luego que esta operación se completa, los segmentos que fueron leídos son marcados como limpios, y puede ser usado para nuevos datos o para limpieza adicional. *Segment summary block:* identifica cada pieza de información que se escribe en cada segmento; por ejemplo, para cada bloque de datos de archivo, el *summary block* contiene el número de archivo y el número de bloque de un bloque. Los segmentos pueden contener múltilples *segment summary blocks* cuando más de una escritura en la bitácora es necesaria para llenar el segmento. Sprint LFS también usa información de *segment summary* para distinguir entre bloques vivos de aquellos que han sido sobreescritos o borrados. Una vez que se sabe la identidad de un bloque, su *liveness* verificando el inode del archivo o bloque indirecto para ver si el puntero de bloque apropiado aún referencia a este bloque. Si lo hace, entonces el bloque está vivo, sino, se marca como muerto.

Segment cleaning policies: (1) Cuándo se tiene que ejecutar el *segment cleaner* → continuamente en *background* a baja prioridad, o cuando el disco está casi exhausto. (2) Cuántos segmentos se debería limpiar a la vez? → entre más segmentos se limpien a la vez, más oportunidades de reordenamiento. (3) Cuáles segmentos tienen que ser limpiados? → los más fragmentados, aunque podría no ser la mejor técnica. (4) Cómo deberían de agruparse los bloques vivos cuando son escritos? → agrupándolos en el mismo directorio u ordenarlos por el tiempo en que fueron modificados. ***Write cost:*** término que se utiliza para comparar políticas de limpieza. Para sistemas de archivos *log-structured* con segmentos grandes, búsquedas y latencia rotacional son despreciables tanto para la escritura como para la limpieza, por lo que el costo de la escritura es el número total de bytes movidos desde y hacia el disco divido por el número de esos bytes que representan nuevos datos. Este costa es determinado por la utilización in los segmentos que son limpiados. En el estado estacionario, el limpiador debe generar un segmento limpio para cada segmento de datos nuevos escritos. Para hacer eso, se leen N segmentos en su totalidad y se escriben $N * u$ segmentos de datos vivos[1]. Esto crea $N * (1 - u)$ segmentos de espacio libre continuo para datos nuevos.

$$\begin{aligned}\text{write cost} &= \frac{\text{total bytes read and written}}{\text{new data written}} \\ &= \frac{\text{read segs + write live + write new}}{\text{new data written}} \\ &= \frac{N + N * u + N * (1 - u)}{N * (1 - u)} = \frac{2}{1 - u}\end{aligned}$$

[1] donde u es la utilización de los segmentos y $0 \leq u < 1$

La clave para lograr alto rendimiento a un bajo costo en sistemas de archivos *log-structured* es forzar el disco a un distribución de segmentos bimodal en donde la mayoría de los segmentos están casi llenos, algunos están vacíos o casi vacíos , y el limpiador pude casi siempre trabajar con los segmentos vacíos. Esto permite una alta utilización de la capacidad del disco y proporciona un bajo costo de escritura. *Simulation results: Cost-benefit cleaning policy*

$$\frac{\text{benefit}}{\text{cost}} = \frac{\text{free space generated * age of data}}{\text{cost}} = \frac{(1-u) * age}{1+u}$$

. Permite que los segmentos fríos sean limpiados a un nivel de utilización mucho mayor que los segmentos calientes. Esta política reduce en un 50 % el costo de la escritura en comparación con la política voraz (de la simulación) y permite que un sistema de archivos *log-structured* supere al mejor Unix FFS incluso a tasas altas de utilización. *Segment usage table:* estructura utlizada en Sprite LFS para darle soporte a la política de limpieza costo-beneficio. Para cada segmento, la tabla graba el número de registros de bytes vivos en el segmento y el tiempo de modificación más reciente de cualquier bloque en el segmento. Estos dos valores son usando por el limpiador de segmentos cuando selecciona qué segmentos limpiar. Esos valores son inicialmente establecidos cuando el segmento se escribe y el contador de bytes vivos se decrementa cuando los archivos son borrados o los bloques son sobreescritos. Si el contador cae a cero entonces el semento puede ser reutilizado sin necesidad de limpieza. Los bloques de la *segment usage table* son escritos en la bitácora, y las direcciones de los bloques son almacenados en regiones de punto de verificación. Con el fin de ordenar bloques vivos por edad, la información de resumen de segmento graba la edad del bloque más joven escrito en el segmento.

Crash recovery. En un sistema de archivos *log-structured* las localizaciones de la última operación de disco es fácil de determinar: están al final de la bitácora. Asi debería de ser posible recuperarse muy rápido luego de un fallo. Sprite LFS usa dos enfoques para recuperación de fallos: (1) *checkpoints:* Un punto de verificación es una posición en la bitácora en la cual todas las estructuras del sistema de archivos son consistentes y completas. Sprite LFS usa un proceso de dos fases para crear un punto de verificación. Primero, escribe toda la información modificada de la bitácora, incluyendo bloques de datos de archivo, bloques indirectos, inodes y bloqeus del *inode map* y *segment usage table*. Segundo, escribe un *checkpoint region* a un posición fija especial en disco. El *checkpoint region* contiene las direccioens de todos los bloques en el *inode map* y *segment usage table*, más el tiempo actual y el puntero al último segmento escrito. (2) *roll-forward* Con el fin de recuperar toda la información posible, Sprite LFS escanea a través de los segmentos de bitácora que fueron escritos luego del último *checkpoint*. Durante el *roll-forward* Sprite LFS usa la información del resumen de bloque para recuperar datos que fueron escritos recientemente. Cuando un resumen de bloque indica la presencia de un nuevo inode, Sprite LFS actualiza el *inode map* que lee desde el *checkpoint*, de modo que el *inode map* referencia a la nueva copia del inode. Esto automáticamente incorpora los nuevos bloques de datos del archivo dentro del sistema de archivos recuperado. Si bloques de datos son descubiertos para un archivo sin una nueva copia de los inode del archivo, entonces el código *roll-forward* asume que la nueva version del archivo en disco está incompleto y se ignoran nuevo bloques de datos.

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

En la última década las velocidades de CPU se han incrementado dramáticamente mientras que los tiempos de acceso al disco solamente han mejorado lentamente. Esta tendencia es probable que continue en el futuro y puede causar que más y más aplicaciones se vean limitadas por el disco.

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

Problems with existing file systems: (1) esparcen información alrededor del disco en una forma que causa muchos accesos pequeños. Se necesita al menos 5 operaciones de I/O, cada una precedida por un seek para crear un nuevo archivo en el FFS de Unix. (2) Tienden a escribir de forma síncrona: la aplicación tiene que esperar que la escritura se complete, en lugar de continuar mientras la escritura se maneja en *background*.

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

La noción de *logging* no es nueva, y un número de sistemas de archivos tiene incorportado una bitácora como una estructura auxiliar para accelera las escrituras y la recuperación fallos. Sin embargo, estos otros sistemas usan la bitácora solo para almacenamiento temporal, el hogar permanente para la información es una estructura de acceso aleatorio tradicional en disco.

En contraste, un sistema de archivos *log-structured* almacena datos permanentemente en la bitácora: no hay ninguna otra estructura en el disco. Esta bitácora contiene información de indexación para que los archivos puedan leer de vuelta con una eficiencia comparable con los sistemas de archivos actuales.

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

Sistemas de archivos *log-structured* que se basan en la suposición que los archivos están en caché de memoria principal y que el creciente tamaño de la memoria harán los cachés más y más efectivos en satisfacer solicitudes de lectura. Como resultado, el tráfico de disco será dominado por escrituras. Un sistema de archivos *log-structured* escribe toda la información nueva al disco en una estructura secuancial llamada bitácora. Este enfoque aumenta el rendmiento de escritura dramáticamente al eliminar casi todas las búsquedas. La naturaleza secuencial de la bitácora también permite que la recuperación a fallos sea más rápida. Para que los sistemas de archivos *log-structured* operen eficientemente, deben asegurar que hay siempre grandes extensiones de espacio libre disponible para escribir nuevos datos. Este es el reto más difícil en el diseño de un sistema de archivos *log-structured*. En el artículo se presenta una solución basea en grandes extensiones llamada segmentos, donde un proceso limpiador de segmentos regenera continuamente segmentos vacíos comprimiendo los datos vivos desde segmentos fuertemente fragmentados.

5. ¿QUÉ TAN EXITOSA ES ESTA SOLUCIÓN?

Microbenchmarks: En un *benchmark* en donde se crearon, leyeron y borraron una gran cantidad de archivos pequeños, Sprite LFS fue casi 10 veces más rápido que SunOS en las fases de creación y borrado. Sprite LFS es también más rápido para leer los archivos de vuelta, esto es porque los archivos son leídos en el mismo orden creado y el sistema de archivos *log-structured* empaqueta los archivos densamente en la bitácora. Sprite LFS muestra también un rendimiento competitivo para archivos grandes. Tiene un ancho de banda de escritura mayor que SunOS en todos los casos de prueba. Es substancialmente más rápido para escrituras aleatorias porque las transforma en escrituras secuenciales en la bitácora. En sistemas de archivos *log-structured* logran localidad temporal: la información que es creada o modificada al mismo tiempo será agrupada cerca del disco.

Crash recovery: Varía con respecto al número y el tamaño de los archivos escritos entre el último punto de verificación y la caída/fallo. Los tiempos de recuperación se pueden acotar al limitar la cantidad de datos escritos entre los puntos de verificación. Cuando se utilizaron archivos de tamaños de un tamaño estándar (en sus pruebas) y tráficos de escritura diarios, un intervalo de punto de verificación tan largo como de una hora podría resultar en tiempos de recuperación de alrededor de un segundo. Usando la tasa máxima observada de escritura de 150Mb/hora, el tiempo máximo de recuperación podría crecer en un segundo por cada 70 segundos del largo del intervalo del punto de verificación.

Other overheads in Sprite LFS: Más del 99 % de los datos vivos en disco consiste de bloques de datos de archivo y bloques indirectos. Sin embargo, cerca de 13 % de la información escrita en la bitácora consiste de inodes, *inode map blocks* y bloques de *segment map*, todos los cuales tienden a ser sobreescritos rápidamente. El *inode map* por si solo abarca más del 7 % de todos los datos escritos en la bitácora.