# Sistemas de archivo basados en bitácora

## ¿Qué es un sistema de archivo basado bitácora?

Mejor conocido como log-structured file system, es un sistema de archivos desarrollado por John Ousterhout y Mendel Rosemblum, en el cual los datos y metadatos son almacenados en un buffer de manera secuencial a manera de bitácora, para después ser escritos en el disco.

## Motivaciones

Conforme la memoria se va haciendo más grande con el paso del tiempo, más información se puede almacenar en la caché, la cual será utilizada principalmente para la lectura, lo cual provoca que el tráfico de datos en el disco sea mayormente de escritura, y por lo tanto, que su rendimiento sea dado por la eficiencia para esta operación.

A pesar de que los retardos causados por la búsqueda y la rotación de los dispositivos mecánicos ha disminuido, este cambio ha sido relativamente lento y reducirlos aún más resultaría muy complicado e incluso costoso, el rendimiento de operaciones de E/S en disco es menor cuando éste se hace de manera aleatoria que si se hiciera de manera secuencial, ya que se reduciría el número de rotaciones y búsqueda en disco.

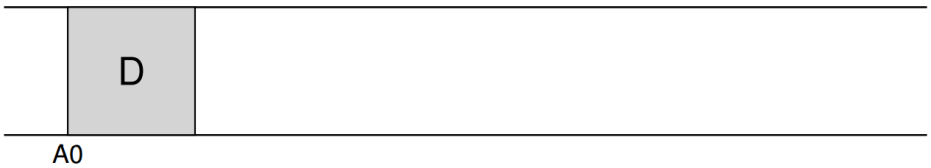
Los sistemas de archivos existentes tenían un rendimiento muy pobre, puesto que implicaban numerosas operaciones de escritura en disco al momento de crear o modificar un archivo. Además, los sistemas eran “inconscientes” si existía un RAID (Redundant Array of Independent Disks), por lo que dependiendo de su nivel, podían llegar a realizar hasta 4 tareas de escritura en disco al realilzarse alguna escritura lógica.

## Funcionamiento

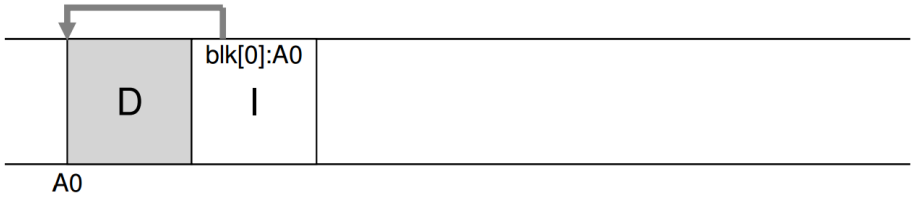
Antes de escribir en el disco, el sistema de archivos almacena y actualiza tanto los datos del archivo como sus metadatos en un segmento de la memoria principal, y cuando este segmento está lleno se escribe completamente en una parte vacía del disco, todo esto en una sola operación. Al ser los segmentos grandes, el disco se usa de manera eficiente.

Una de las interrogantes que surgen es: ¿cómo hacer que la escritura de las actualizaciones de un archivo sean secuenciales?

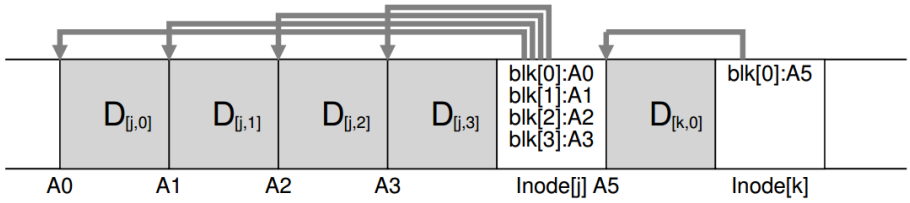
Primero se escribe nuestro bloque de datos en el disco, lo cual se puede visualizar de la siguiente manera:



Pero como se describió anteriormente, también se actualizan los metadatos de nuestro archivo, para lo cual se escribe también el i-nodo del archivo en el disco, y haciendo que éste apunte a nuestro archivo.




Es correcto pensar que escribir bloque por bloque no es muy eficiente, puesto que se deben de tener en cuenta la espera dada por la rotación del disco para colocarse de nuevo en un punto de tal manera que la escritura del siguiente bloque sea secuencial. Pero si el lector es atento, recordará que el sistema de archivos no guarda bloques de datos, sino segmentos de datos, dichos segmentos están compuestos de varios bloques de datos con sus respectivos i-nodos, y debe ser lo suficientemente grande para mantener una buena eficiencia, para poder recibir poca penalización por el tiempo de espera causado por la rotación.




El tamaño de los segmentos está dado por la siguiente expresión:

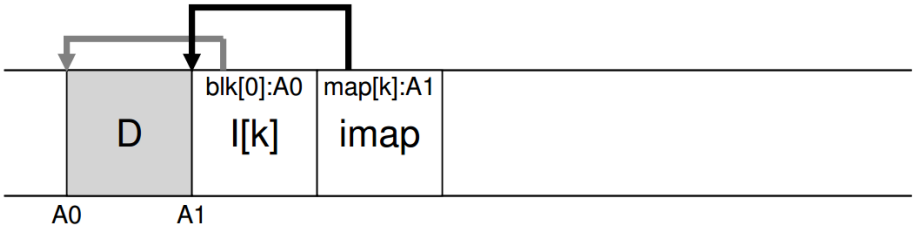
$$D = \frac{F}{1-F} \times R_{peak} \times T_{position}$$

Donde:

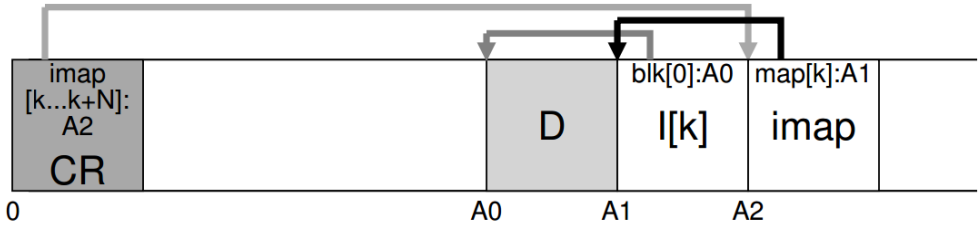
$D$: $Cantidad\ de\ datos\ a\ escribir\ (MB)$
$F$: $Tasa\ de\ efectividad\ (0 < F < 1)$
$R_{peak}$: $Tasa\ de\ transferencia\ a\ disco\ \left(\frac{MB}{s}\right)$
$T_{position}$: $Tiempo\ de\ posicionamiento\ (s)$

Uno de los inconvenientes a notar después de hacer varias escrituras en disco, es que tenemos i-nodos distribuidos por todas partes, contrario a lo que sucede en un sistema UNIX tradicional, donde todos los i-nodos están en un lugar fijo. Y peor aún, el i-nodo apuntando a la última versión del archivo se sigue moviendo.

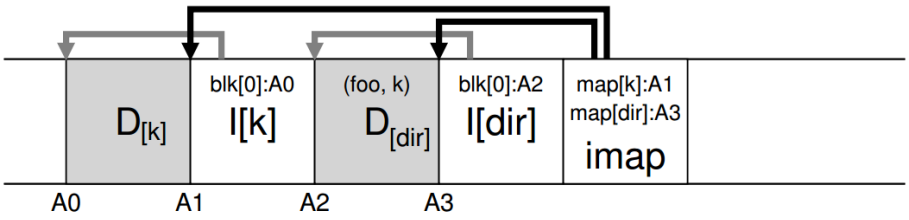
Para soluciona este detalle, se introduce lo que se llama i-mapa (mapa de i-nodos), donde se guardan los apuntadores a las versiones más recientes de los archivos. Curiosamente, este mapa de i-nodos no se guarda en una región fija del disco, como se podría pensar, en cambio coloca pedacitos de este mapa junto al bloque de datos que está por escribir




Y entonces ¿cómo localizamos este i-mapa si también está distribuido en todo el disco? Para ello se tiene una estructura llamada “Checkpoint Region” (CR) en una región fija del disco, el CR contiene apuntadores a las piezas más recientes del i-mapa y se actualiza periódicamente, por lo que no afecta de manera significativa al rendimiento.




En cuanto a los directorios, la información de éstos se genera al crear un nuevo archivo, añadiendo el bloque de datos correspondiente al directorio (contenido, i-nodos), así como el i-nodo correspondiente.




### Garbage Collector

Otro cuestionamiento que surge a partir de cómo se guardan en este sistema de archivos, es: ¿qué hace con las versiones viejas de los archivos? Bien podrían no borrarse, pero entonces el sistema de archivos pasaría a llamarse “sistema de archivos con versiones”, algo que no corresponde con el sistema basado en bitácora, que solo guarda la versión actual. Entonces el sistema de archivos debe hacerse cargo de limpiar estas versiones viejas (garbage) de datos de archivos, i-nodos, mapas y demás estructuras, y así poder tener espacio libre para las operaciones de escritura futuras.

La limpieza se realiza de la misma manera que la escritura, limpiando segmentos de datos, con lo cual se evita tener un gran número de espacios libres entre bloques que pueda perjudicar el rendimiento de las operaciones de escritura.

El método para esta limpieza es relativamente sencillo, se leen X segmentos, se determina qué bloques de datos pertenecen a versiones actuales, y compacta el contenido guardándola en Y segmentos, liberando así espacio que será usado para escrituras subsecuentes.

Tambien se debe tener políticas que determinen cuándo se debe limpiar y qué segmentos limpiar. Puede ser una limpieza periódica, cuando el sistema esté inactivo o se llegue a un umbral de capacidad del disco. Se puede hacer una separación entre bloques “calientes” (accesados con alta frecuencia) y “fríos” (baja frecuencia), tendiendo a limpiar los últimos.

## ¿En qué se diferencia con un sistema de archivos con bitácora?

El nombre de ambos sistemas de archivos es tan parecido, inclusive en inglés: log-structured file system corresponde a sistema de archivos basado en bitácora y logging (aunque recientemente es conocido como journaling) file system para sistema de archivos con bitácora, que pareciese que son lo mismo, pero no es así.

La principal diferencia radica en la estructura de la bitácora, en un sistema con bitácora, ésta se encuentra en una sección del disco distinta a la de los datos, llevando al menos dos operaciones de escritura en disco, la primera siendo la de los datos, y la segunda la de la actualización de la bitácora. En un sistema basado en bitácora, se lleva a cabo una sola escritura.

## Uso en sistemas embebidos

Hoy en día, tenemos sistemas que de una u otra manera se pueden denominar computadoras y que no solo se limita a la gran variedad computadoras personales, también tenemos los dispositivos móviles, y eso pequeños aparatos tales como routers, sistemas de GPS, relojes, y demás dispositivos que podemos denominar sistemas embebidos.

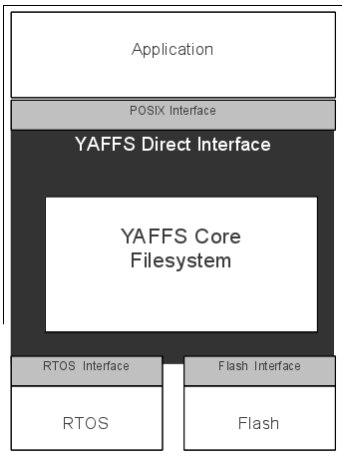
Las personas tratan muy diferente estos sistemas embebidos de lo que la mayoría denomina “computadoras”, dado que la gran mayoría no sabe que está tratando con una computadora muy pequeña, y la usan como cualquier aparatejo sin los cuidados que le darían a una “computadora”.

Para la gran mayoría de estos sistemas embebidos, los usuarios simplemente desconectan el aparato, pudiendo llegar a provocar alguna falla que bien puede ser pequeña, o puede dejar el aparato disfuncional. Por esta razón, se requiere diseñar los sistemas embebidos de manera muy diferente a la de una computadora usual.

Una parte de este diseño está enfocada al sistema de archivos del sistema embebido, ya que como se había mencionado, es común que los usuarios simplemente desconecten éste, pudiendo llegar a corromper los archivos si es que se estaban escribiendo en disco.

YAFFS (Yet Another File System) es un ejemplo claro de lo que como un sistema de archivos basado en bitácora se puede implementar en sistemas embebidos como medida de prevención ante la corrupción de datos. Al encender el sistema, para YAFFS siempre será un inicio normal dado que siempre construye el estado del sistema, sin necesidad de cargar algún código de recuperación, lo que quiere decir que no importa si el sistema se apagó de manera segura o abrupta, no habrá corrupción de datos.

## Integración en sistemas embebidos (YAFFS)




YAFFS Direct Interface: “Envuelve” a YAFFS de tal manera que sea fácil de integrar.

POSIX Interface: Usada por el código de la aplicación para acceder a YAFFS (open, close, read, write, etc.).

RTOS Interface: Funciones proporcionadas para que YAFFS pueda tener acceso a los recursos del sistema (initialise, lock, unlock, get time, set error, etc.).

Flash Interface: Funciones proporcionadas para que YAFFS pueda acceder al dispositivo flash (initialise, read chunk, write chunk, erase block, etc.).

## Referencias bibliográficas:

Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau. **Operating Systems: Three Easy Pieces.** Arpaci-Dusseau Books. March, 2015 (Version 0.90). http://pages.cs.wisc.edu/~remzi/OSTEP/file-lfs.pdf

Mendel Rosenblum and John K. Ousterhout. **The Design and Implementation of a Log-Structured File System**. July 24, 1991. https://people.eecs.berkeley.edu/~brewer/cs262/LFS.pdf

Documentos sobre YAFFS:

http://www.yaffs.net/documents/yaffs-direct-interface

http://www.yaffs.net/documents/considerations-choosing-flash-file-system

## Artículo de interés:

Valerie Aurora. **Log-structured file systems: There's one in every SSD**. Linux Weekly News. September 18, 2009. https://lwn.net/Articles/353411/