Bisoños Usuarios de GNU/Linux de Mallorca y Alrededores | Bergantells Usuaris de GNU/Linux de Mallorca i Afegitons

Sistemas de Ficheros con *Journaling* en Linux (44169 lectures)

Per **Ricardo Galli Granada**, ***gallir*** (http://mnm.uib.es/gallir/)

Creado el 24/01/2002 23:10 modificado el 24/01/2002 23:10

Este es una versión en castellano levemente modificada de un par de artículos publicados en Novática[1] y Upgrade[2] (inglés, PDF[3]) donde explico la implementación de todos los sistemas de ficheros con journaling disponibles en Linux.

Sistemas de Ficheros con *Journaling* en Linux

Ricardo Galli
gallir@uib.es
Dept. de Matemàtiques i Informàtica
Universitat de les Illes Balears

English version[4]
Version en Français[5]

Antes de todo, no hay un ganador claro, XFS es mejor en algunos aspectos, ReiserFS en otros, y ambos son mejores que Ext2 en el sentido que son comparables en rendimiento pero ellos son sistemas de ficheros con journaling, y ya se sabe cuales son sus ventajas. Quizás la moraleja mas importante es que el sistema de buffer-cache del Linux sorprende por su rendimiento, afectó, positivamente, a todos los resultados de mis compilaciones, copias y lecturas/escritas aleatorias. Así que diría, compra memoria RAM e instala un sistema con journaling lo antes posible...

Palabras clave: Linux, sistemas operativos, page-cache, buffer-cache, journal file systems, Ext3, XFS, ReiserFS, JFS.

Introducción

El párrafo anterior es un extracto del que fue el segundo de una serie de artículos publicados en el web de Bulma[6], la Asociación de Usuarios de Linux en Baleares que describió los resultados de una segunda ronda de comparativas de rendimiento de sistemas de ficheros tradicionales con sistemas con *journaling*.

Aunque informales, ambas comparativas analizaron el Fat32, Ext2, ReiserFS, XFS y JFS para diferentes casos: las herramientas Mongo de Hans Reiser, copia de ficheros, compilación del kernel y un pequeño programa en C que simula patrones de acceso de sistemas de bases de datos de tamaño medio.

Ambos artículos estuvieron entre los primeros en hacer dichas comparativas de los sistemas de ficheros con *journaling* (registro o diario). Nuestro servidor web de Bulma se vio sobrecargado debido a su publicación en *slashdot.org*. Estas pruebas sirvieron principalmente para hacer desaparecer el mito que los sistemas de ficheros con *journaling* son significativamente más lentos que los sistemas tradicionales de Unix (UFS) y del Ext2, derivado del anterior y estándar de Linux durante varios años.

Desde esos días se han publicado más comparativas, pero la verdad es todavía la misma: no hay un claro ganador. Algunos sistemas se comportan mejor que otros en algunos casos, pero no en todos. Por ejemplo, ReiserFS es muy bueno leyendo ficheros pequeños a medianos, XFS tiene mejor rendimiento para ficheros grandes y se dice que JFS facilita mucho la migración de sistemas con OS/2 Warp y AIX a Linux.

En este artículo describimos todos los sistemas con *journaling* disponibles para Linux: Ext3, ReiserFS, XFS y JFS. También introducimos los conceptos básicos de *buffer-cache*, y *page-cache* del núcleo del Linux. El rendimiento de los diferentes sistemas de ficheros es afectado de forma importante por el uso de las dos técnicas anteriores. Y aún más importante, no sólo el rendimiento se ve afectado, sino también la re-programación de módulos importantes originalmente desarrollados para otros sistemas operativos. Por ejemplo, Silicon Graphics (SGI) tuvo que programar un nuevo módulo, *pagebuf*, que hace de interfaz entre su propia técnica de *buffering* en el XFS y los módulos del *page-cache* de Linux.

El sistema de ficheros virtual en Linux

Un *fichero* es una abstracción muy importante en programación. Los ficheros sirven para almacenar datos de forma permanente y ofrecen un pequeño conjunto de primitivas muy potentes (abrir, leer, avanzar puntero, cerrar, etc.). Los ficheros se organizan normalmente en estructuras de árbol, donde los nodos intermedios son directorios capaces de agrupar otro ficheros.

El sistema de ficheros es la forma en que el sistema operativo organiza, gestiona y mantiene la jerarquía de ficheros en los dispositivos de almacenamiento, normalmente discos duros. Cada sistema operativo soporta diferentes sistemas de ficheros. Para mantener la modularización del sistema operativo y proveer a las aplicaciones con una interfaz de programación (API) uniforme, los diferentes sistemas operativos implementan una capa superior de abstracción denominada *Sistema de Ficheros Virtual* (*VFS: Virtual File System*). Esta capa de software implementa las funcionalidades comunes de los diversos sistemas de ficheros implementados en la capa inferior.

Los sistemas de ficheros soportados por Linux se clasifican en tres categorías:

1. Basados en disco: discos duros, disquetes, CD-ROM. Estos sistemas son Ext2, ReiserFS, XFS, Ext3, UFS, ISO9660, etc.
2. Sistemas remotos (de red): NFS, Coda, y SMB.
3. Sistemas especiales: procfs, ramfs y devfs.

El modelo general de ficheros puede ser interpretado como orientado a objetos, donde los objetos son construcciones de software (estructura de datos y funciones y métodos asociados) de los siguientes tipos:

- **Super bloque**: mantiene información relacionada a los sistemas de ficheros montados. Está representado por un bloque de control de sistema almacenado en el disco (para sistemas basados en disco).
- **i-nodo**: mantiene información relacionada a un fichero individual. Cada i-nodo contiene la meta-información del fichero: propietario, grupo, fecha y hora de creación, modificación y último acceso, más un conjunto de punteros a los bloques del disco que almacenan los datos del fichero.
- **Fichero**: mantiene la información relacionada a la interacción de un fichero abierto y un proceso. Este objeto existe sólo cuando un proceso interactúa con el fichero.
- **Dentry**: enlaza una entrada de directorio (*pathname*) con su fichero correspondiente. Los objetos *dentry* recientemente usados son almacenados en una caché (*dentry cache*) para acelerar la translación desde un nombre de fichero al i-nodo correspondiente.

Todos los sistemas Unix modernos permiten que los datos del sistema de ficheros sean accedidos de dos formas distintas (Figura two).

1. Asociación (*mapping*) de memoria con **mmap**: La llamada de sistema mmap() permite a los procesos de usuario acceder de forma directa a los datos del *page-cache* mediante asociación de memoria. El objetivo de mmap es permitir el acceso a los datos mediante direcciones del sistema de memoria virtual, por lo que los datos de ficheros pueden ser tratados como estructuras de memoria estándares. Los datos del fichero son leídos y copiados a la *page-cache* bajo demanda (*lazily*) cuando se generan fallos de página por intentos de acceso a páginas no residentes en RAM.
2. Llamadas de sistema de acceso directo al sistema de E/S de bloques, tales como **read** y **write**: La llamada de sistema read() lee los datos de los dispositivos de bloques a la caché del núcleo (para CDs y DVDs se puede evitar esta copia mediando el parámetro O_DIRECT del *ioctl*), luego se copian al espacio de direcciones del proceso. La llamada de sistema write() copia los datos en la dirección opuesta, desde la memoria del proceso a la caché del núcleo y eventualmente, en un *futuro próximo*, a los bloques correspondientes del disco. Estas interfaces son implementadas usando el *buffer-cache* o el *page-cach*e para almacenar temporalmente en la

memoria del núcleo.

Page-cache y *buffer-cache* del Linux

En versiones anteriores del Linux (y en Unix en general), las operaciones de correspondencia de memoria (*mapping*) eran gestionadas por el sistema de memoria virtual (VM o MM), mientras que las llamadas de E/S por bloques fueron gestionadas independientemente por el subsistema de E/S. Por ejemplo, en Linux hasta la versión 2.2.x, el sistema de memoria virtual y el de E/S tenían cada uno sus propios mecanismos de caché para mejorar el rendimiento: *page-cache* y *buffer-cache* respectivamente.

Figure: Buffer-cache y page-cache

Buffer-cache

El *buffer-cache* mantiene copias de bloques de disco individuales. Las entradas del caché están identificadas por el dispositivo y número de bloque. Cada *buffer* se refiere a cualquier bloque en el disco y consiste de una cabecera y un área de memoria *igual* al tamaño del bloque del dispositivo. Para minimizar la sobrecarga, los *buffers* se mantienen en una de varias listas enlazadas: sin usar (*unused*), libres (*free*), no modificadas (*clean*), modificadas (*dirty*), bloqueadas (*locked*), etc.

Por cada operación de lectura, el subsistema de *buffer-cache* debe buscar si el bloque en cuestión ya está copiado en la memoria. Para hacerlo de forma eficiente se mantienen tablas de dispersión para todos los *buffers* presentes en la caché. El *buffer-cache* es también usado para mejorar las operaciones de escritura, en vez de copiar todos los bytes inmediatamente al disco, el núcleo los almacena temporalmente en el *buffer-cache* e intenta agrupar varias operaciones para escribirlas al disco simultáneamente. Un *buffer* que está esperando por ser copiado al disco se denomina *modificado* (*sucio* o *dirty*).

Page-cache

A diferencia del anterior, el *page-cache* mantiene páginas completas de la memoria virtual (4 KB en la plataforma x86). Las páginas pertenecen a ficheros en el sistema de ficheros, de hecho las entradas en el *page-cache* están parcialmente indexadas por el número de i-nodo y su desplazamiento en el fichero. Pero una página es casi siempre más grande que un bloque de disco, y los bloques que conforman dicha página pueden estar almacenados de forma discontinua en el disco.

El *page-cache* es principalmente usado para satisfacer los requerimientos de interfaz del subsistema de memoria virtual, que usa páginas de tamaño fijo de 4KB, con el VFS, que usa bloques de tamaño variable u otro tipos de técnicas tales como *extents* en XFS y JFS.

Unificación del *page-cache* y *buffer-cache*

Los dos mecanismos anteriores operaron de forma semi-independiente uno del otro. El sistema operativo tenía que tener un especial cuidado para mantener sincronizados los dos caches y así prevenir que las aplicaciones reciban datos inválidos. No sólo eso, si el sistema estaba escaso de memoria, el sistema operativo tenía que tomar decisiones muy difíciles para liberar memoria del *buffer-cache* o del *page-cache*.

El *page-cache* tiende a ser más sencillo de administrar debido a que representa más directamente los conceptos usados en los niveles superiores del código del núcleo. El *buffer-cache* además tiene las limitaciones que los datos siempre deben ser asociados al espacio de direcciones del núcleo, lo que agrega límites artificiales a la cantidad de datos que pueden mantenerse en caché, ya que el hardware moderno puede tener fácilmente más RAM que el espacio de memoria del núcleo.

Con el tiempo, partes importantes del núcleo han pasado de usar el *buffer-cache* al *page-cache* por razones de simplicidad de codificación. Pero los bloques individuales del *page-cache* estaban todavía gestionados por el *buffer-cache*, lo que creaba confusiones entre el uso y programación de ambos niveles.

Esta falta de integración llevó a un rendimiento ineficiente y falta de flexibilidad. Para lograr un buen rendimiento es importante que los sistemas de memoria virtual y E/S estén bien integrados. La forma elegida en Linux para reducir las ineficiencias de las dobles copias es almacenar los datos sólo una vez (en el *page-cache)* y mantener punteros de cada elemento en el *buffer-cache* (Figura shared).

Figure: Los datos son compartidos por el page-cache y buffer-cache

Las correspondencias temporales de memoria de las páginas en el *page-cache* para soportar read() y write() son raramente necesarias ya que Linux traduce permanentemente toda la memoria física al espacio de direcciones del núcleo. Linux además usa una técnica interesante, el número de bloque donde está una página en el disco está almacenada en forma de una lista en memoria con una estructura denominada *buffer_head*. Cuando una página modificada tiene que ser grabada en disco, las solicitudes de E/S pueden ser pasadas directamente al gestor del dispositivo, sin necesidad de hacer ningún tipo de operación adicional para determinar donde deben ser escritos los

datos.

La unificación del page-cache

Siguiendo los conceptos del ``Unified I/O and Memory Caching Subsystem for NetBSD'', Linus Torvalds quiso cambiar radicalmente el comportamiento del *page-cache* en el núcleo de Linux. El 4 de mayo de 2001, escribió el siguiente mensaje a la lista de desarrolladores (linux-kernel):

> *Quiero re-escribir block_read/write para que use el page cache, pero no porque impactará nada en esta discusión. Quiero hacerlo al principio del 2.5.x porque:*
>
> *- acelerará los accesos*
>
> *- reusará mejor el código existente y conceptualizará las cosas más claramente (i.e. convertiría un disco en sistema de ficheros _realmente_ simple con sólo un fichero enorme ;-)*
>
> *- hará que la gestión de memoria sea mucho mejor para cosas como el fsck - la presión de memoria está diseñada para trabajar en cosas como el page-cache.*
>
> *- será una cosa menos que use el buffer cache como una ``caché'' (quiero que la gente piense y use el buffer cache como una entidad de _E/S_, no como una caché).*
>
> *No cambiará el ``caché al arranque'' para nada (porque aún en el page cache, no hay nada en común entre la representación virtual de un _fichero_ (o metadata) y la representación virtual de un _disco_).*

Aunque estos cambios no estaban programados hasta el 2.5.x, Linus finalmente se decidió por integrar una cantidad considerable (más de 150) de parches de Andrea Arcangeli, más sus propios cambios, y liberó el 2.4.10, que finalmente unificaba el *page-cache* y *buffer-cache* (Figura unified). A raíz de estos cambios, se esperan importantes mejoras en el sistema de E/S, sobre todo cuando se ajusten mejor los parámetros del sistema de memoria virtual (al momento de escribir este artículo, la versión disponible del Linux es 2.4.13, que ya muestra importantes mejoras de rendimiento, sobre todo para sistemas con poca memoria, y un consumo bastante menor de RAM para el *page* y *buffer cache*).

Figure: Unificación a *page-cache*

Sistemas de ficheros con *Journaling*

Durante mucho tiempo, el sistema de ficheros estándar en Linux fue el Ext2. Éste fue diseñado por Wayne Davidson con la colaboración de Stephen Tweedie y Theodore Ts'o. Es una mejora al sistema anterior, Ext, diseñado por Rémy Card. El Ext2 está basado en i-nodos (asignación indexada). Cada i-nodo mantiene la meta-información del fichero y los punteros a los bloques con los datos ``reales".

Como hemos mencionado antes, para mejorar el rendimiento de las operaciones de E/S, los datos del disco son temporalmente almacenados en la memoria RAM a través del *page-cache* y *buffer-cache*. Los problemas surgen si hay un corte de suministro eléctrico antes que los datos modificados en la memoria (*dirty buffers*) sean grabados nuevamente al disco. Se generaría una inconsistencia en el estado global del sistema de ficheros. Por ejemplo, un nuevo fichero que todavía no fue ``creado" en el disco u otros que hayan sido borrados pero sus i-nodos y bloques de datos todavía permanecen como ``activos" en el disco.

El **fsck** (*file system check*) fue la herramienta que resolvía dichas inconsistencias, pero el fsck tiene que analizar la partición completa y verificar las interdependencias entre i-nodos, bloques de datos y contenidos de directorios. Con la ampliación de la capacidad de los discos, la recuperación de la consistencia del sistema de fichero se ha convertido en una tarea que requiere mucho tiempo, por lo que crea problemas serios de disponibilidad de los servidores afectados. Esta es la razón principal de que los sistemas de ficheros hayan importado de las bases de datos las técnicas de transacciones y recuperación, y así hayan aparecido los sistemas de ficheros con *journaling*.

Un sistema con *journaling* es un sistema de ficheros tolerante a fallos en el cual la integridad de los datos está asegurada porque las modificaciones de la meta-información de los ficheros son primero grabadas en un registro cronológico (*log* o *journal*) antes que los bloques originales sean modificados. En el caso de un fallo del sistema, un sistema con *journaling* ``integral" asegura que la consistencia del sistema de ficheros es recuperada. El método más común es el de grabar previamente cualquier modificación de la meta-información en un área especial del disco, el sistema realmente grabará los datos una vez que la actualización de los registros haya sido completada. A la hora de recuperar la consistencia luego de un fallo, el módulo de recuperación analizará el registro y sólo repetirá las operaciones incompletas en aquellos ficheros inconsistentes, es decir que la operación registrada no se haya llevado a cabo finalmente.

Los primeros sistemas de ficheros con *journaling* fueron creados a mediados de los ochenta e incluyen a Veritas (VxFS), Tolerant y JFS de IBM. La demanda de sistemas de ficheros que soporten terabytes de datos, miles de ficheros por directorios y compatibilidad con arquitecturas de 64 bits ha hecho que en los últimos años haya crecido el interés de la disponibilidad de sistemas con *journaling* en Linux.

Linux tiene ahora cuatro nuevos sistemas de ficheros: ReiserFS[7] de Namesys, XFS[8] de Silicon Graphics (SGI), JFS[9] de IBM y Ext3[10] desarrollado por Stephen Tweedie, co-desarrollador del Ext2

Mientras que ReiserFS es un sistema totalmente nuevo escrito desde cero, XFS, JFS y Ext3 son derivados de sistemas ya existentes. XFS está basado, y comparte parcialmente el mismo código, en el sistema desarrollado por Silicon Graphics para sus estaciones de trabajo y servidores. JFS fue desarrollado por IBM para su sistema OS/2 Warp, que a su vez es derivado del sistema JFS de AIX.

ReiserFS es el único incluido en el árbol de desarrollo estándar del núcleo, está programado que los otros sean incluidos en la versión 2.5, aunque ya están en fase totalmente estable y funcional.

Ext3 es una extensión a Ext2 y agrega dos módulos independientes: un módulo de transacciones y un módulo de registro. Ext3 está en un estado muy avanzado y su inclusión en el árbol estándar es prioritaria. La distribución RedHat 7.2 ya la incluye como opción y será el sistema de ficheros oficialmente soportado por dicha empresa.

B-Trees

La técnica básica para mejorar el rendimiento comparado a sistemas de ficheros tradicionales de Unix es evitar el uso de listas enlazadas o mapas de bits (para bloques libres, entradas de directorios y direccionamiento a bloques de datos) ya que sufren de problemas inherentes de escalabilidad. La complejidad típica para búsquedas es O(n), no muy apropiada para discos muy grandes. La mayoría de los nuevos sistemas usan *Árboles Balanceados* (*B-Tree*) o variaciones de ellos (*B+Tree*).

La estructura de los árboles balanceados están bien estudiadas, son más robustos en rendimiento pero al mismo tiempo los algoritmos de gestión y balanceo son más complejos. La estructura *B+Tree* ha sido usada para indexación de base de datos desde hace tiempo ya que les provee de formas rápidas escalables para acceder a sus registros.

El símbolo ``+'' en *B+Tree* significa, normalmente, que es una versión modificada del original que:

- Coloca todas las claves en las hojas.
- Las hojas pueden estar enlazadas entre ellas.
- Los nodos pueden tener tamaños diferentes.
- No se necesita cambiar un padre si una clave en un hijo ha sido borrada.
- Hace que las búsquedas secuenciales sean más simples y rápidas.

ReiserFS

ReiserFS está basado en *B+Trees* para organizar los objetos del sistema de ficheros. Dichos objetos son estructuras que mantienen información de ficheros: permisos, tiempos de creación, modificación y acceso, datos de ficheros, etc. En otras palabras, información tradicionalmente mantenida en los i-nodos, directorios y bloques de datos. ReiserFS llama a esos objetos:

1. *stat data items*,
2. *directory items* y
3. *direct/indirect items,*

respectivamente.

ReiserFS provee *journaling* de sólo la meta-información de los ficheros. En caso de un reinicio no planificado, los bloques de datos de ficheros que se estaban usando en ese momento podrían estar corruptos.

Los *nodos no formateados* son bloques lógicos sin un formato determinado y son usados para mantener los datos de ficheros, los *direct items* consiste de los propios datos de ficheros. Dichos items son de tamaño variable y almacenados en las hojas del árbol, algunas veces junto a otros items si hay espacio suficiente en el nodo. La meta-información de los ficheros es almacenada próxima a los propios datos, ya que el sistema intenta ubicar a los *stats items* y los *direct/indirect items* del mismo fichero en ubicaciones contiguas.

Contrariamente a los *direct items*, los datos de ficheros apuntados por *indirect items* no se almacenan dentro del árbol. Esta gestión especial de los *direct items* se debe al soporte para ficheros pequeños: *empaquetado de colas* (o *tail packing*).

El empaquetado de colas es una características especial del ReiserFS, las colas son ficheros más pequeños que un bloque lógico, o los últimos bytes de los ficheros que no rellenan un bloque completo. ReiserFS empaqueta dichos ficheros o paquetes en nodos hojas únicos para ahorrar espacio en disco. Generalmente esto permite un ahorro del 5% comparado con el Ext2. Además ReiserFS tiene un rendimiento excelente para ficheros pequeños ya que puede incorporar dichas colas en el *B-Tree*, así se encuentran muy próximos a su meta-información (*stat data*).

El problema de esta técnica es que puede producir fragmentación externa. Además, el empaquetado de colas consume más CPU y puede afectar negativamente al rendimiento, especialmente en procesadores más lentos. Esto se debe a la necesidad de desplazamiento de memoria cuando se agregan datos a un fichero. Namesys conoce este problema y permite que el empaquetado sea desactivado especificando *notail* como opción de montaje para el montaje (o inclusive si se hace un *remount*).

Para la asignación de espacio, ReiserFS usa bloques de tamaño fijo (4KB) que afecta negativamente al rendimiento en operaciones sobre ficheros grandes. El otro punto débil de ReiserFS es que el rendimiento sobre ficheros *esparcidos* (*sparse*, ficheros que no tienen todos los bloques de datos ocupados) es significativamente peor que Ext2, aunque Namesys está trabajando en este tema.

Figure: Asignación basada en bloques

XFS

El 1 de mayo de 2001, SGI puso a disposición la versión 1.0 de sus sistema de *journaling* XFS para Linux. XFS es reconocido por su buen soporte a ficheros grandes con velocidades de transferencias muy altas (verificado hasta 7GB/seg). XFS fue desarrollado para su variante de Unix, Irix, versión 5.3, cuya primera versión fue introducida al mercado en diciembre de 1994. El objetivo del sistema fue soportar ficheros muy grandes para tratamiento de vídeo en tiempo real y generación de imágenes sintéticas para la industria de Hollywood.

XFS usa *B+Trees* extensivamente, mantiene la información de *extents* libres, índices de directorios y la asignación dinámica de los i-nodos que se encuentran distribuidos por todo el sistema de ficheros. Además, XFS usa técnicas de registro asincrónicas para proteger la modificación de la meta-información de ficheros.

XFS usa una asignación de espacio basada en *extents* (ver figura), y tiene características como asignación retrasada (para acelerar las operaciones de creación y ampliación de ficheros), reagrupamiento de espacio luego del borrado de ficheros e intenta usar *extents* del máximo tamaño posible durante la asignación de espacio. Para hacer más eficiente la gestión de espacio contiguo, XFS usa grandes descriptores de *extents*. Cada uno de ellos puede describir hasta dos millones de bloques del sistema, lo que ahorra mucho tiempo de CPU al no tener que buscar por bloques contiguos, puede simplemente leer la longitud del *extent* en vez que verificar para cada entrada si es contigua con la anterior.

Figure: Asignación basada en *extents*

XFS permite bloques de tamaño variable por cada sistema, de 512 bytes a 64 kilobytes. El cambio de tamaños de bloques varía la fragmentación, sistemas con ficheros pequeños suelen usar bloques más pequeños para evitar el desaprovechamiento de espacio por fragmentación interna. Sistemas con ficheros grandes suelen elegir la opción inversa para reducir la fragmentación externa.

El código fuente e XFS es muy complejo y muy orientado a Irix, lo que obligó a re-escribir la interfaz para poder portarlo a Linux. El resultado es el módulo *pagebuf*, que provee la interfaz entre XFS y el sistema de memoria virtual como así también entre XFS y la capa de dispositivos de bloques del núcleo.

El XFS soporta listas de acceso (*ACL*), integradas con el servidor Samba, y sistemas de cuotas transaccionales. En Linus soporta cuotas por grupo, mientras que en Irix es por proyecto. Otras características avanzadas, especialmente para aplicaciones de vídeo bajo demanda en tiempo real, todavía no han sido portadas a Linux.

Como mencionamos anteriormente, el modo normal de trabajo es la grabación asincrónica del registro, que aún así asegura que los datos de ficheros no pueden ser grabados al disco antes que haya hecho lo mismo con los datos del registro. Con la grabación asincrónica del registro se gana en rendimiento por dos factores principales:

1. Múltiples modificaciones pueden ser agrupadas en un sólo operación de grabación del registro.
2. El rendimiento de las modificaciones de la meta-información (que deben ser registradas) se hace independiente de la velocidad del dispositivo. Esta independencia está limitada por la cantidad de memoria que se asigna para los *buffers* del registro.

JFS

IBM introdujo su sistema de ficheros JFS (*Journaled File System*) con la versión 3.1 del AIX. Dicho sistema luego fue modificado para su gama de sistemas corporativos OS/2 Warp Server, que es el que comparte código con la versión para Linux.

JFS está optimizado para altas velocidades de transferencias. Usa, como el XFS, asignación basada en *extents*, junto con políticas de asignación de bloques agrupados, para producir estructuras eficientes para hacer corresponder desplazamientos lógicos dentro de los ficheros a direcciones físicas del disco. Los extents en JFS son una secuencia de bloques contiguos asignados a un fichero como una unidad y descriptos por la tripla `<desplazamiento lógico, longitud, física>`. La estructura de direccionamiento es un árbol *B+Tree* de descriptores de *extents*, cuya raíz es el i-nodo y la clave es el desplazamiento dentro del fichero.

Los registros del JFS son mantenidos en cada sistema de ficheros y registran información de operaciones en la meta-información de ficheros. La semántica tradicional del registro es sincrónica, si una operación que involucra cambios en la mata-información retorna satisfactoriamente, los efectos de la operación ya han sido enviadas al sistema de ficheros y serán visibles aún si el sistema falla inmediatamente después de la operación.

En términos de rendimiento, la técnica sincrónica es una desventaja comparada con otros sistemas de ficheros que usan técnicas asincrónicas (tales como Veritas y XFS). JFS ha sido mejorado y ahora también provee operaciones de registro asincrónicas, el cual mejora sustancialmente el rendimiento.

JFS soporta tamaños de bloques de 512, 1024, 2048 y 4096 bytes, únicos por cada sistema de fichero. Los efectos de fragmentación son similares a los descriptos para XFS. El tamaño por defecto es 4096 bytes. Al igual que XFS, JFS asigna espacio para los i-nodos dinámicamente y los libera cuando ya no son necesarios.

Para los directorios se usan dos organizaciones:

1. Para pequeños directorios, el contenido se almacena dentro del mismo i-nodo, esto ahorra la necesidad de otra operación de E/S y uso de espacio adicional.
2. Para directorios más grandes, cada directorio es un árbol *B+Tree*, donde la clave es el nombre del fichero. Este sistema provee búsquedas, inserción y borrado más eficientes.

JFS soporta ficheros *esparcidos*, el tamaño de fichero reportado por el sistema es el byte más alto que ha sido escrito, sin embargo no se asignan bloques de discos hasta que haya operaciones de escritura sobre dichos bloques.

Ext3

Ext3 es compatible con Ext2, en realidad es Ext2 con un fichero adicional de registro. Ext3 es una capa adicional sobre Ext2 que mantiene un fichero de registro (por defecto en el directorio /jfs). Debido a que está integrado en el Ext2, sufre algunas de las limitaciones de dicho sistema, y no explota las posibilidades de los sistemas de *journaling* puros. Por ejemplo, todavía usa asignación basada en bloques y búsqueda secuencial de directorios, aunque se está trabajando es esta área para mejorarla.

Sus mayores ventajas son:

- Ext3 mantiene la consistencia tanto en la meta-información como en los datos de los ficheros. A diferencia de los demás sistemas de *journaling* mencionados, la consistencia de los datos también está asegurada.
- Las particiones Ext3 no tienen una estructura de ficheros diferentes a los de Ext2, por lo que no sólo se puede pasar de Ext2 a Ext3, sino que lo opuesto también funciona, útil sobre todo si en algún caso el registro se corrompe accidentalmente, por ejemplo debido a sectores malos del disco.

Ext3 reserva uno de los i-nodos especiales de Ext2 para el registro, pero los datos del mismo pueden estar en cualquier conjunto de bloques, y en cualquier sistema de ficheros. Inclusive se puede compartir el registro entre sistemas distintos.

Tres tipos de bloques de datos son grabados en el registro:

1. Meta-información: contiene el bloque de meta-información que está siendo actualizado por la transacción. Cada cambio en el sistema de ficheros, por pequeño que sea, es escrito en el registro. Sin embargo es relativamente barato ya que varias operaciones de E/S pueden ser agrupadas en conjuntos más grandes y pueden ser escritas directamente desde el sistema *page-cache* usando la estructura *buffer_head*.
2. Bloques descriptores: Estos bloques describen a otros bloques del registro para que luego puedan ser copiados al sistema principal. Los cambios en estos bloques son siempre escritos antes que los de meta-información.
3. Bloques cabeceras: Describen la cabecera y cola del registro más un número de secuencia para garantizar el orden de escritura durante la recuperación del sistema de ficheros.

Rendimiento y conclusiones

Las diferentes pruebas de rendimiento (ver Enlaces al final) demuestran que ReiserFS y XFS obtienen excelentes resultados comparados con el ya bien ajustado y optimizado Ext2. Ext3 es más lento pero aproximándose al rendimiento de Ext2. Suponemos que el rendimiento de Ext3 será mejorado considerablemente en los meses siguientes. Por otro lado, JFS obtiene los peores resultados globales, además de problemas de inestabilidad en algunas versiones anteriores de Linux.

XFS, ReiserFS y Ext3 han demostrado que son excelentes y muy confiables. Hay un área donde XFS es el claro vencedor, especialmente si se compara con ReiserFS: operaciones sobre ficheros grandes. Sin embargo esto es posible que cambie con el tiempo, ReiserFS usa las operaciones de lectura y escritura genéricas del núcleo, mientras que XFS ha heredado las operaciones sofisticadas de Irix, como el uso de *extents*, asignación retrasada y operaciones de E/S directas. Además, el código de ReiserFS usado para las pruebas hace un recorrido completo del árbol por cada 4KB que escribe, y luego inserta un puntero, lo que genera una sobrecarga importante del sistema al tener que balancear el árbol mientras copia datos.

ReiserFS ha mostrado sus mejores resultados para ficheros entre 100 y 10.000 bytes, especialmente si los ficheros aún no están en el *page-cache* (como ocurre durante el arranque del ordenador). En caso que se lean ficheros que ya están en la caché, la diferencias son prácticamente despreciables para todos los sistemas de ficheros mencionados.

Entre todos los sistemas de ficheros con *journaling*, ReiserFS es el único incluido en el núcleo estándar desde la versión 2.4.1 (y SuSE lo soporta desde hace unos dos años). Sin embargo, Ext3 será el estándar de una distribución tan conocida como Red Hat, y XFS está siendo usado en grandes servidores, especialmente en la industria del cine y los efectos especiales. Tanto Ext3 y XFS son prioritarios para su inclusión en el núcleo. La inclusión de XFS es un poco más complicada debido a los grandes cambios que introduce en el *page-cache*. Por otro lado, los programadores de Linux están siguiendo con mucho interés el esfuerzo que dedica IBM para solucionar los problemas de estabilidad del

JFS, aunque ya se dice que es una alternativa válida para migrar de OS/2 y AIX a Linux.

Enlaces

- Ext3 architecture[11]
- Introduction to Linux Journal File Systems[12]
- XFS Home Page[13]
- JFS Home Page[9]
- ReiserFS Home Page[14]
- Storage Foundry[15]
- OS News[16]

Pruebas de rendimiento

- Ext2, Ext3, ReiserFS, XFS and JFS benchmarks[17]
- Ext2, ReiserFS and XFS Benchmarks[6]
- Pruebas con XFS, ReiserFS, Ext2FS, y FAT32[18]
- Namesys benchmarks[19]
- Ext2, XFS, ReiserFS and JFS Mongo Benchmarks[20]

About this document ...

This document was generated using the **LaTeX**2HTML[21] translator Version 2K.1beta (1.48)

The translation was initiated by Ricardo Galli on 2001-11-04

Ricardo Galli 2001-11-04

Lista de enlaces de este artículo:

1. http://www.ati.es/novatica/
2. http://www.upgrade-cepis.org/issues/2001/6/upgrade-vII-6.html
3. http://www.upgrade-cepis.org/issues/2001/6/up2-6Galli.pdf
4. http://bulma.net/body.phtml?nIdNoticia=1154
5. http://bulma.net/body.phtml?nIdNoticia=1167
6. http://bulma.net/body.phtml?nIdNoticia=642
7. http://www.namesys.com
8. http://bulma.net/http//oss.sgi.com/projects/xfs/
9. http://oss.software.ibm.com/developerworks/opensource/jfs/
10. http://e2fsprogs.sourceforge.net/ext2.html
11. ftp://ftp.kernel.org/pub/linux/kernel/people/sct/ext3/
12. http://www.linuxgazette.com/issue55/florido.html
13. http://oss.sgi.com/projects/xfs/
14. http://www.namesys.com/
15. http://sourceforge.net/foundry/storage/
16. http://www.osnews.com/story.php?news_id=69
17. http://www.mandrakeforum.org/article.php?sid=1212
18. http://bulma.net/body.phtml?nIdNoticia=626
19. http://www.namesys.com/benchmarks/benchmark-results.html
20. http://bulma.net/body.phtml?nIdNoticia=648
21. http://www-texdev.mpce.mq.edu.au/l2h/docs/manual/

E-mail del autor: gallir _ARROBA_ uib.es
Podrás encontrar este artículo e información adicional en: http://bulma.net/body.phtml?nIdNoticia=1153