
 Eligir un Sistema de Ficheros

LA ELECCIÓN

La elección de un sistema de ficheros dependerá de sus necesidades. Vamos a profundizar en las diferencias que existen entre los diferentes sistemas de ficheros más populares de Linux de modo que no tenga que realizar una elección a ciegas.

POR BEN MARTIN

Es fantástico poder elegir, y un kernel moderno de Linux ofrece ciertamente una gran selección de sistemas de ficheros para albergar nuestros valiosos datos. En este artículo vamos a examinar algunos de los problemas del diseño de sistemas de ficheros y mostraremos cómo lo resuelven algunos de los sistemas de ficheros más populares.

Esta visión desde la perspectiva de un diseñador debería proporcionarle algunas de las claves a la hora de ver qué sistema de ficheros se ajusta mejor a su entorno.

Problemas, Problemas...

A primera vista, los sistemas de ficheros parecen bestias bastante mansas – todo lo que queremos es almacenar algunos datos en ficheros dentro de un disco y tenerlos disponibles de nuevo posteriormente, ¿correcto?

Según esta definición, lo único que necesitamos que haga el sistema de ficheros es seguirle la pista a los metadatos o

los nombres de los ficheros y directorios y las relaciones existentes entre unos y otros (el anidamiento dentro de los directorios), y probablemente almacenar un rango continuo de bytes para cada fichero como el “contenido” de los mismos. O, quizás el sistema de ficheros podría almacenar los metadatos en una tabla, listando los nombres de los ficheros, el tamaño, la fecha de modificación, etc, así como el contenido en bytes de los ficheros.

En esta etapa, el sistema de ficheros aún parece moderadamente simple. Hay que tener algunas primitivas de bloqueo para la tabla de metadatos para asegurarse de que los metadatos permanezcan consistentes, incluso con múltiples procesos ejecutándose al mismo tiempo, aunque la necesidad de alguna forma de bloqueo no es un problema. También podría ser interesante tener alguna clase de índice para la tabla de metadatos de modo que se pueda leer un directorio rápidamente.

Pero, espere... Si la luz se va y el PC se queda sin corriente eléctrica, el sistema de ficheros no debería sufrir daños y debería contener todos los datos cuando se vuelva a encender de nuevo la máquina. Y todo debería estar bien incluso si da la casualidad de que estaba escribiendo los datos en el disco cuando se fue la luz. Tal vez no consiga toda la información que se estaba escribiendo en el momento del corte eléctrico, pero el resto debería encontrarse perfectamente. Además, probablemente preferiremos no tener que esperar a que se ejecute una de esas operaciones *fsck* para comprobar el sistema de ficheros que tarda horas y horas en completarse en un resplandeciente disco de 2TB nuevo. Seguro que unos cuantos segundos son más que suficientes para que el pobre sistema de ficheros para determinar que estaba realizando 50 cosas a la vez cuando se fue la luz y recuperarse por sí mismo a algún estado seguro.

Ahora parece que las cosas se están empezando a complicar. Si el sistema de ficheros intenta actualizar los metadatos en el mismo lugar del disco, ¿cómo podrá determinar si un bloque de la tabla de metadatos es correcto o estaba a medio escribir cuando se fue la luz? ¿Qué hay de los índices de la tabla de metadatos? Es posible que parte de esos índices se escribieran al disco y los otros estuviesen esperando en la RAM a ser escritos en él cuando se produjo el fallo eléctrico. Usted de verdad, realmente no quiere perder ningún metadato, suceda lo que suceda. El índice debe ser válido también, ya que de otro modo, el sistema de ficheros no podría confiar en él y sería lo mismo que no tenerlo. Pero incluso si el sistema de ficheros puede detectar que no se apagó correctamente, la reconstrucción del índice podría tardar bastante tiempo, y esto es lo mismo que teníamos con el escenario del *fsck* que pretendíamos evitar.

Así que, quizás el sistema de ficheros podría disponer de un diario de los cambios realizados en los metadatos a lo largo del tiempo, y en algún “checkpoint”, construir una nueva tabla de metadatos e indexarla con las actualizaciones y cambiar de la vieja estructura de datos a la nueva. De esta forma, el sistema siempre tendrá al menos una tabla de metadatos y un índice válidos, y si la máquina fallara, podrían utilizarse la versión actual y el diario para avanzar hasta el punto más próximo en el momento de producirse el fallo. Aunque este plan podría ponerse en funcionamiento, la reconstrucción del índice cada vez que se deseen sincronizar los cambios del sistema de ficheros al disco lo haría muy lento.

Además de la consistencia de los datos, también se espera que un sistema de ficheros sea tan rápido como sea posible. Cuando se descomprime un archivo de 50MB o se ejecuta una compilación distribuida, cuanto más rápido se ejecuten las tareas, mejor. Afortunadamente, no tiene que intentar implementar su rápido y seguro sistema de ficheros propio; ya existen varios sistemas de ficheros para el kernel de Linux que resuelven de diversas formas estos problemas.

Almacenar los Metadatos

Hasta el momento hemos determinado que es buena idea almacenar los metada-

tos de los ficheros y los directorios en alguna clase de tabla abstracta y poseer un índice para la tabla con la idea de encontrar las entradas rápidamente. Cuando se considera cómo se puede gestionar un directorio, el índice cobra una importancia vital para el buen rendimiento del sistema de ficheros.

Normalmente, cada fichero recibe un número único: su inodo. Un directorio probablemente tendrá una lista de los inodos de todos los ficheros que contiene. De modo que para encontrar quién es el propietario y ver el grupo y los permisos de acceso para los ficheros del directorio, el sistema de ficheros utilizará una colección de búsquedas en la tabla de metadatos usando para ello los números de inodos.

La búsqueda de los metadatos de un fichero por medio de los números de inodos es una operación bastante frecuente, desde la comprobación de los permisos de acceso de los ficheros cuando se van a leer, hasta simplemente realizar un *ls -l* en un directorio.

Si el diseño del sistema de ficheros limita el número de inodos disponibles y se reserva una tabla contigua para todos los inodos en el momento de la creación del sistema de ficheros, el número de inodo puede utilizarse para encontrar la entrada directamente en la tabla de metadatos. Por ejemplo, si 128 inodos entran en cada página, entonces el inodo 600 se encontrará en la quinta página de la tabla de metadatos. El inconveniente de este esquema es que la tabla tiene que ser contigua y tiene que reservarse de antemano su espacio; de lo contrario, el uso de B-Trees para almacenar tanto la tabla de metadatos como sus índices se convertiría en una solución deseable.

Un B-Tree puede ser útil para implementar

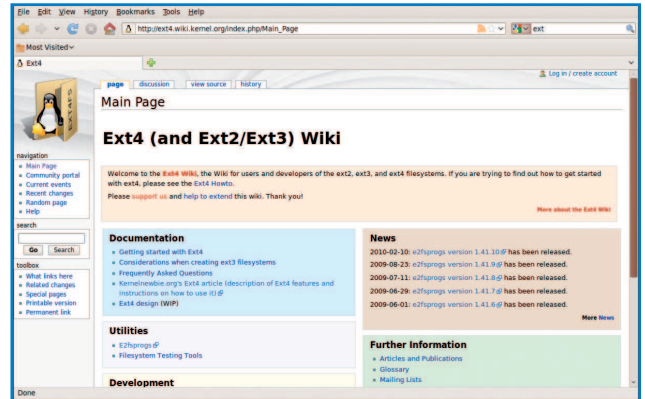


Figura 1: El sistema de ficheros extendido (ext) ya lleva varios años de desarrollo.

un índice que permita búsquedas de metadatos de ficheros por medio de inodos. Para aquellos que no estén familiarizados con los B-Trees, hay que decir que un B-Tree es un método de indexación basado en árboles (estructura jerárquica), que permite realizar búsquedas de una clave o un conjunto de claves de manera muy rápida. La estructura de datos B-Tree data de los años 70 y funciona bien en discos, ya que se puede buscar una clave con muy pocos accesos al disco, y mientras se añaden datos al B-Tree, éste permanece balanceado. (Para abreviar, no vamos a diferenciar entre B-Trees y B + Trees en este artículo).

Un B-Tree es una estructura de datos paginada, con lo que muchas búsquedas de claves se encuentran agrupadas en páginas individuales del disco (normalmente de 4KB), y una clave en una página referencia a otra página. Así que, si los números de página y las claves son de 8 bytes cada uno, la página raíz de 4KB del árbol podrá referenciar a otras 256 páginas, cada una de las cuales

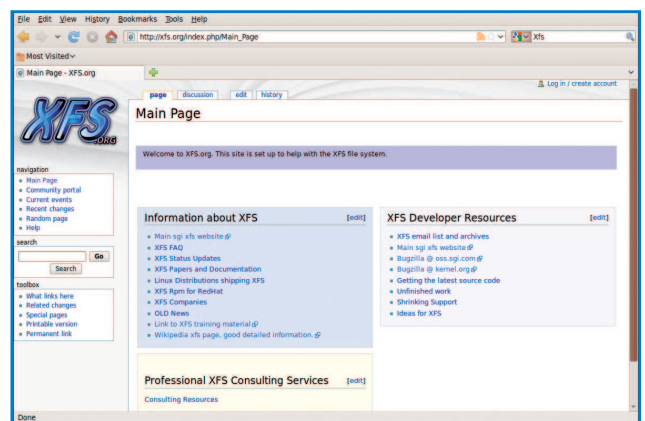


Figura 2: Originalmente desarrollado por Silicon Graphics, el sistema de ficheros XFS con diario es mantenido ahora como un proyecto independiente.

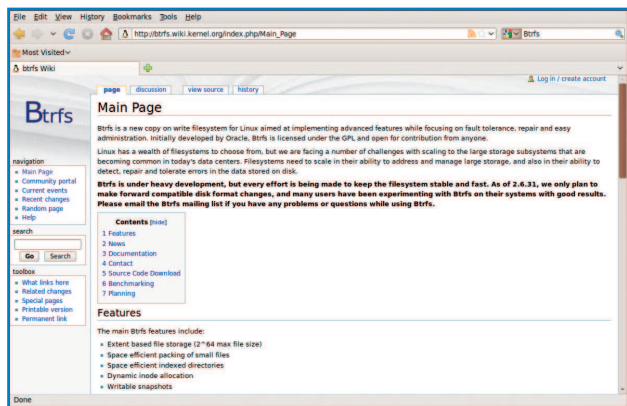


Figura 3: El sistema de ficheros B-Tree (Btrfs - pronunciado "Butter FS") ha creado expectación en la comunidad open source con funcionalidades avanzadas para la realización sencilla de imágenes y la expansión de múltiples dispositivos. Como indica la wiki, Btrfs es relativamente nuevo y aún se encuentra bajo un desarrollo intenso.

podrá a su vez referenciar a otras 256 páginas. Como se puede ver, no hacen falta muchas páginas para poder contener miles o millones de claves. Este gran "abanico" en cada página del árbol significa que se podría encontrar una única clave en un B-Tree de millones de claves con sólo mirar tres páginas.

Como cada página podría requerir una búsqueda de la cabeza del disco, lo cual es una operación extremadamente lenta, la clave del juego es la minimización del número de páginas que se necesitan para acceder. Por supuesto, estas reglas sobre las búsquedas están cambiando conforme entran en el mercado los discos de estado sólido, pero el coste por gigabyte de los medios tradicionales probablemente haga que la primera opción sea ésta durante algunos años más.

Las páginas correspondientes a las hojas del árbol B-Tree almacenan los "datos" de las claves, que pueden ser tanto un puntero a otra estructura de datos o los propios datos si no ocupan mucho espacio. Para el sistema de ficheros, los datos son la fecha de modificación, el propietario, los permisos, etc., los cuales ocupan poco espacio y pueden almacenarse directamente en el B-Tree. El sistema de ficheros XFS reserva y almacena los inodos en grupos de 64; el B-Tree de inodos referencia a cada uno de estos grupos de inodos.

Los B-Trees también son estructuras ordenadas, lo cual puede utilizarse en un sistema de ficheros para proporcionar un buen rendimiento. Por ejemplo, cuando se crean 50 ficheros en un directorio, si el

sistema de ficheros utiliza números secuenciales de inodos para esos 50 ficheros, se almacenarán en el B-Tree juntos y una búsqueda de los 50 ficheros podría hacer uso de la misma página. Como los números se agrupan juntos, las páginas que se requieren para seguir bajando por el árbol son las mismas, y podríamos ser capaces de localizar los 50 ficheros con tan sólo tres o cuatro accesos al disco.

Aunque todos estos índices son maravillosos, el problema de lo que le suceda al B-Tree cuando haya un corte eléctrico aún existe. Recuerde que no quiere tener que reconstruir el índice B-Tree y siempre quiere que sea completamente válido.

Diario y COW

Existen dos estrategias principales para mantener la consistencia de los metadatos del sistema de ficheros: el uso de un registro basado en un diario para mantener la lista de los cambios y el uso de una COW (Copy On Write) para el B-Tree. Cuando se usa un diario, hay que realizar periódicamente los cambios en las estructuras de datos principales y registrarlos en el diario. En el diseño de un COW B-Tree se utiliza un proceso similar, en el cual se realiza una migración haciendo uso de un nuevo nodo raíz.

La solución que utiliza el diario para mantener los metadatos la implementan XFS, ext3 y ext4 (Figuras 1 y 2). Los COW B-Trees son utilizados por Btrfs (Figura 3). Las series de sistemas de ficheros ext usan una tabla de inodos reservada de antemano. El sistema XFS utiliza B-Trees de inodos y Btrfs usa COW B-Trees de inodos.

COW B-Trees

Normalmente, cuando se actualiza un B-Tree, la página que contiene una clave se lee en la RAM, se actualiza allí y luego se escribe de nuevo en el disco. Si al añadir una clave la página crece lo suficiente como para no caber de nuevo donde estaba alojada, se divide en dos páginas y

se añade una clave nueva a la página padre antigua. En un COW B-Tree, cuando se inserta una clave, se deja al árbol antiguo completamente intacto y se crea una copia de la página que se va a actualizar. Para enlazar esta página con el árbol hay que realizar también una copia de todas las páginas padre. Recuerde que sólo debe haber entre tres y cinco páginas padre para cada página en el B-Tree, ya que la estructura se abre en abanico rápidamente, de modo que copiar todos estos padres no es una operación costosa. En este punto, se tiene un B-Tree que utiliza cuatro o cinco páginas nuevas y comparte las otras páginas con el B-Tree original. Como con la inserción se copian todas las páginas padre, se tendrá un nuevo nodo raíz cada vez, proporcionando de hecho una imagen del sistema de ficheros cada vez que se realice una modificación.

Por razones de rendimiento, un sistema de ficheros que utilice COW B-Trees intentará realizar en RAM la mayor parte de las operaciones sobre el B-Tree y sólo periódicamente se sincronizarán las páginas copiadas, creando de este modo un nuevo nodo raíz del árbol persistente. Si se mantienen los cambios, se pueden realizar muchas modificaciones sobre los nodos copiados en RAM sin tener que copiarlos de nuevo, reduciendo así los accesos al disco.

Estas imágenes permiten montar el sistema de ficheros en modo de sólo lectura tal como era en un momento determinado (por ejemplo, para ver nuestro directorio home tal y como era hace un mes). Una imagen que se pueda escribir se denomina clon y permite que se puedan realizar experimentos con diferentes cambios en el sistema antes de decidir qué clon o clones se desean mantener.

Aunque todas estas copias puedan parecer un desperdicio del espacio en disco, hay que tener en cuenta que se está hablando de duplicar páginas de 4KB, de modo que si incluso se duplican millones de páginas, aún se estará hablando de una modesta cantidad de espacio. Una vez que las páginas antiguas no se vayan a utilizar más, se pueden reutilizar para que el árbol no siga creciendo y devore el disco completo.

Btrfs utiliza árboles COW para todo, así que puede tomar rápidamente una instantánea del sistema de ficheros completo en cualquier momento.

Escribiendo en Disco

En algún momento, un sistema de ficheros va a necesitar conocer qué dato se encuentra escrito en el disco y por ello a salvo – por ejemplo, que el diario o cada página en un COW B-Tree nuevo se encuentra al completo en disco en vez de estar parcialmente en la pequeña memoria caché de 16MB que posee el disco.

Para asegurarse de que los datos se encuentran en el disco en vez de en la caché puede utilizar un comando SATA. Desafortunadamente, este comando SATA es del tipo “todo o nada”. No se puede decir “asegúrese de que los datos de los comandos SATA 12, 45 y 99 están en el disco”. Al menos no se puede decir de manera segura para todo tipo de hardware. Normalmente uno se ha de conformar con decir “escribe todo lo que tengas”. Dado que la caché del disco actualmente es de 16 ó 32MB, puede que no todo su contenido tenga que escribirse, y ya que puede escribir 100MB/s al disco, esto no debe suponer ningún problema. Pero cuando se considera que esa caché puede que no sea la única que contenga datos que hayan que escribirse – por ejemplo, la controladora de disco también podría tener su propia caché – y que los datos de la caché podrían tener localizaciones en diferentes puntos del disco, puede que resulte al final un retraso considerable cuando se ejecute la operación de escritura en disco del comando SATA.

Esta escritura SATA se conoce como “disk barriers” [1]. Los sistemas de ficheros ext4, Btrfs y XFS la usan por defecto. Aunque las operaciones pesadas con metadatos tales como la creación o el borrado de muchos ficheros se pueden ver ralentizadas por las “barriers”, el sistema de ficheros también será mucho más potente frente a problemas que surjan debido a los cortes eléctricos. Si se desea, se pueden desactivar las “barriers” para estos sistemas de ficheros, y hay que saber que no funcionarán con versiones del kernel anteriores a la 2.6.31, dependiendo de los parches que incorpore su distribución Linux. Afortunadamente, hay que tener en cuenta que si se está trabajando con una máquina que no posea el refuerzo de una batería y está preocupado por sus datos, entonces seguro que deseará utilizar las “barriers”. He descubierto que activándolas, las operaciones con metadatos y *fsync* se ven ralentizadas en un factor de 10. Estamos

claramente en una situación de compromiso entre la velocidad y la capacidad de asegurar los datos frente a un corte de luz.

Almacenar el Contenido

Hay muchas formas de almacenar los bytes de un fichero en disco, y cuando un sistema de ficheros almacena estos bytes, obviamente necesita conocer dónde se encuentran para poder recuperarlos posteriormente. Con el uso de bloques, el disco se divide en un número fijo de bloques, y cada fichero almacena una lista de números de bloques, lo que permite encontrar posteriormente todas las partes del fichero de nuevo.

Para ficheros pequeños, la lista de bloques se puede almacenar con la información de los inodos. Conforme el fichero crezca, la lista de números de bloques para encontrar todos los bytes del fichero también crece, de manera que el inodo almacena el número de bloque, que es un bloque que sólo contiene los números de bloque que permiten la localización del contenido del fichero. Conforme va creciendo el fichero, el bloque intermedio que contiene la lista de bloques del contenido del fichero se llenará, por lo que habrá bloques con referencias a bloques con referencias a datos y así sucesivamente.

Aparte de almacenar la lista de bloques, otro inconveniente de la utilización de bloques es que los ficheros grandes poseerán muchos bloques, y éstos podrían no estar físicamente cerca los unos de los otros en el disco. Por supuesto, el sistema de ficheros puede intentar obtener bloques próximos cuando se escribe un fichero.

Por otro lado, hay una solución basada en extensiones que se centra en rangos continuos de bytes [2]. En un mundo ideal, una película casera de 10GB se almacenaría en un rango continuo de bytes del disco, y esto sólo requeriría una pequeña cantidad de metadatos para saber dónde se encuentra almacenado el contenido; estos datos también podrían leerse del principio al final sin tener que realizar búsquedas en el disco.

Como se puede imaginar, si tiene una fragmentación mínima, podría borrar un fichero grande mucho más rápido que si se utilizan reservas de bloques. Un artículo de Christoph Hellwig [3] de SGI Open Source Project revela que el

tiempo requerido para borrar un fichero de 60GB tarda unos 50 segundos para ext3, cerca de 2 segundos para ext4 y 0.04 segundos para XFS (Figura 1). Como los dos últimos sistemas de ficheros emplean extensiones, puede obtener una ventaja considerable de los sistemas de ficheros basados en extensiones si a menudo crea y borra grandes ficheros (por ejemplo, si realiza trabajos de producción de vídeo).

El talón de Aquiles para la solución basada en extensiones aparece ante la presencia de ficheros fragmentados. Un caso patológico de esto consiste en descargar una distribución Linux por medio de una red P2P. Conforme se descarga el archivo, los distintos trozos llegan en momentos diferentes, siendo suministrados al sistema de ficheros. Así que se puede acabar con cientos de pequeñas extensiones que contengan estos pequeños fragmentos.

Para evitar la fragmentación de ficheros, el sistema de ficheros puede optar a evitar tener que escribir los datos al disco inmediatamente; esta estrategia se denomina reserva atrasada. Por ejemplo, un programa podría escribir un fichero de la siguiente manera: lo abre, luego entra en un bucle y escribe bloques de 64KB de datos muchas veces hasta terminar cerrando el fichero. Si el sistema de ficheros utiliza una caché para los datos escritos en memoria hasta que el programa cierre el fichero, puede saber de antemano cuál es el tamaño de los datos antes de escribir nada al disco. Para un sistema de ficheros basado en extensiones, puede encontrar una única extensión que esté próxima al tamaño deseado y escribir los datos ahí de una vez. No sólo se evita la fragmentación, sino que con una única búsqueda en disco se escriben todos los datos. Un esquema de reserva de bloques podría tratar de encontrar un grupo de bloques que se ubiquen en la misma zona del disco y escribir los datos en esos bloques de forma que se reduzca la sobrecarga de búsqueda en el disco cuando se produzca una lectura secuencial de nuevo de los datos.

Funciones tales como *fallocate(2)* permiten a un programa informar al sistema de ficheros del tamaño exacto que va a tener el fichero, aunque no todos los programas se aprovechan de estas funciones que le facilitan la labor al sistema de ficheros.

Incluso con toda la ayuda de la escritura retrasada y de *fallocate*, se acabará con el contenido de los ficheros fraccionados. Por ejemplo, si tiene un sistema que se encuentre lleno casi al 100% y hagan falta más de una extensión para un fichero grande, entonces se tiene que producir la fragmentación. Esta es una de las razones por las que muchos sistemas de ficheros poseen herramientas de desfragmentación.

Para XFS, se puede utilizar la herramienta de reorganización del sistema de ficheros *xfs_fsr* para optimizar los ficheros. Una de las ventajas de *xfs_fsr* es que se le puede indicar cuánto tiempo deseamos que se ejecute, y la próxima vez que lo haga, recordará dónde se quedó. De esta forma, se puede ejecutar durante unas horas en medio de la noche cada día, y cada vez continuará por donde se quedó la última vez.

El sistema de ficheros ext4 también posee su propia herramienta de desfragmentación. La utilidad *e4defrag* permite desfragmentar explícitamente un fichero, aunque a principios de 2010, este programa no se incluía en Fedora 11, 12 ni en Rawhide.

Elección de un Sistema de Ficheros

Con todos estos COWs, diarios y árboles, puede que se esté preguntando qué sistema de ficheros es el que mejor se ajusta a sus necesidades. La respuesta más simple es: depende. Y quizás la mejor solución consiste en utilizar más de uno en su sistema, aprovechándose de las ventajas que aportan cada uno de ellos para cada carga de trabajo en particular.

La familia ext almacena los inodos en una tabla, XFS utiliza B-Trees y Btrfs utiliza COW B-Trees para todo. Ext3, ext4 y XFS emplean registros basados en diario para mantener la integridad ante un posible corte del suministro eléctrico. A los sistemas de ficheros ext les gusta ejecutar una operación *fsck* de corta duración de vez en cuando durante la fase de puesta en marcha del sistema. Ext4, XFS y Btrfs activan por defecto las “barriers” de los discos, dejando a ext3 sólo en este aspecto.

Como guía general para el buen rendimiento, debe tenerse en cuenta que no debería llenar los discos demasiado. Una vez que el sistema de ficheros llega al 90-95 por ciento de su capacidad, tendrá que

comenzar a esparcir los ficheros, y posiblemente también los metadatos, por el disco.

Como nota, para crear menos bloques, muchos sistemas de ficheros utilizan un diseño del tipo “muchas subpartes”. Por ejemplo, el sistema de ficheros XFS está dividido en muchos “grupos de reserva”, lo que visto de otra manera, funciona como pequeños sistemas de ficheros autónomos. Esto permite al sistema de ficheros actualizar muchas estructuras de datos en paralelo, ya que las distintas partes no se afectan entre sí. Hay que tener en cuenta que la extensión de un fichero podría tener que dividirse y almacenarse en diversos grupos de reserva. De forma que si hay poco espacio en disco, sus efectos se ven amplificados por el uso de muchas partes pequeñas.

Muchos sistemas de ficheros permiten decidir qué tamaño han de tener los inodos cuando se crea el sistema de ficheros. Si está planeando utilizar atributos extendidos y desea un buen rendimiento a expensas de una pequeña pérdida de espacio en disco, probablemente desee crear los inodos algo más grandes de lo normal. El tamaño fijo de un inodo podría dividirse entre el almacenamiento de la lista de extensiones en el inodo, así como en el manejo de los atributos extendidos.

Irónicamente, cuando se utilizan los atributos extendidos extensamente, doblando explícitamente el tamaño de los inodos, incluso se podría hacer que el sistema de ficheros utilice menos espacio para sus metadatos y proporcionar un mejor rendimiento al mismo tiempo. Esta paradoja sucede porque la lista de extensiones y los atributos extendidos podrían caber dentro del inodo, evitando así que el sistema de ficheros tenga que mover los atributos extendidos a otro bloque de 4KB, ya que todo no cabría dentro del inodo.

Como caso concreto, para XFS, una lista de extensiones de 19 entradas podría caber en el tamaño por defecto de un inodo. Si se comienza utilizando uno o más atributos extendidos, tendrán que compartir el mismo espacio que las mencionadas 19 entradas, y cuando no quepan, los atributos extendidos se moverán a un B-Tree con una página propia.

Como Btrfs utiliza COW B-Tree, es fácil entender la excitación en torno a este sis-

tema de ficheros – un sistema de ficheros que utiliza reserva de extensiones y permite la realización de imágenes y clones de sistemas de ficheros rápidamente [4]. El mayor inconveniente de Btrfs es que es relativamente nuevo. Por otro lado, la mayor ventaja de los otros sistemas de ficheros es que llevan funcionando bastante tiempo. Cuando se habla del directorio home, valorará el hecho de que mucha gente lleva utilizando ext3 desde hace bastante tiempo, y esto nos da un grado de seguridad de que los datos continuarán estando disponibles.

Por otro lado, XFS lleva en el kernel de Linux desde la versión 2.4, y con su sistema de reservas de ficheros basada en extensiones y la desfragmentación, puede manejar ficheros muy grandes muy bien. Un inconveniente bien conocido de XFS es que es muy lento en las operaciones de creación y borrado de muchos ficheros. De modo que si planea descomprimir archivos con muchos ficheros pequeños como los códigos fuente de programas, XFS no sería la opción más óptima.

En resumen, si va a manejar ficheros grandes que pueda desfragmentar, XFS es una buena elección. Si la reserva dinámica de inodos y las extensiones no le llaman la atención, entonces ext3 es una buena solución. Ext4 trae la reserva de extensiones al diseño del sistema de ficheros ext3, y ext4 debería brindarle la posibilidad de desfragmentar (*e4defrag*). Si desea imágenes rápidas y clones de momentos específicos en el tiempo, entonces Btrfs es lo que anda buscando. Para desfragmentar Btrfs use *btrfsctl -d*.

Si desea velocidad y fiabilidad, es una buena idea almacenar el registro de diario de su sistema de ficheros en un buen SLC SSD de calidad o invertir en una buena batería y desactivar las “barriers”. ■

RECURSOS

- [1] Benchmarking barriers: <http://ldn.linuxfoundation.org/article/filesystems-data-preservation-fsync-and-benchmarks-pt-3>
- [2] ext2: <http://e2fsprogs.sourceforge.net/ext2intro.html>
- [3] ÖXFS: The Big Storage File System for LinuxÓ por Christoph Hellwig, October 2009: <http://oss.sgi.com/projects/xfs/papers/hellwig.pdf>
- [4] Btrfs: http://btrfs.wiki.kernel.org/index.php/Btrfs_design