Universidad Nacional Autonoma de Mexico
Facultad de Ingeniería

Semestre:
2025-2

Materia:
Sistemas Operativos

Grupo:
06

Profesor:
Gunnar Eyal Wolf Iszaevich

Integrantes:
Avila Reyes Iker
Gustavo Romero Christian

Tarea:
Documento de Presentación

Fecha de Entrega:
01/05/25

Documento de Presentación - BTRFS

¿Qué es?

BTRFS es un sistema de archivos en Linux diseñado para ser robusto, eficiente y escalable. Sus herramientas principales incluyen sistemas incorporados de compresión, control de versiones y detección de errores.[2][3][4] Está basado en la técnica Copy-On-Write, lo que le da un uso de memoria más eficiente. El sistema utiliza una estructura de datos de “árboles B”,[5] de ahí su nombre “B-Tree File System”, y está diseñado para poder manejar distintos usos y cargas de trabajo, por lo que se continúa actualizando constantemente para mantener un rendimiento decente a pesar de su antigüedad.[1]

Copy-On-Write.

Mencionamos que el sistema está basado en la técnica Copy-On-Write. ¿Pero qué significa esto?

Copy-On-Write, también conocido como “Shadowing”[6] o “Implicit Sharing”[7], es una técnica que busca maximizar el uso eficiente de recursos y minimizar el copiado innecesario de datos. Ésta dicta que, en casos de lectura, los datos se pueden compartir sin problema por varios procesos, y solo es necesario copiarlos cuando alguno intenta escribir sobre ellos. En éstos casos, la copia se guarda en una nueva dirección de memoria sin sobrescribir los datos originales. Como consecuencia de ésto, se mejora la integridad de los datos y se facilita la creación de snapshots.

Arboles B.

Otra cosa que se mencionó es que BTRFS utiliza una estructura de árboles B. Los árboles B, o B-trees, son una estructura de datos cuyo objetivo es administrar grandes cantidades de datos, manteniéndolos organizados y fácilmente accesibles para minimizar la cantidad de accesos al disco.[5][8]

¿Cómo es el proceso de escritura?

Digamos que queremos modificar un archivo llamado “archivo.txt”.

Lo primero que se realiza es la identificación de los bloques de datos. Siguiendo el sistema de árbol B, cada nodo contiene referencias a bloques de datos (por ejemplo,

bloques en el disco donde está guardado el archivo). Para encontrar archivo.txt, recorre desde el Árbol Raíz → Árbol de Metadatos → Entrada del archivo.

Una vez encontrado, como se quiere modificar el archivo, se realiza una copia. En lugar de sobreescribir el archivo sobre ese bloque, se asigna un nuevo bloque de memoria, y se escribe la nueva versión de los datos en ese bloque.

Finalmente, se actualizan los nodos del árbol B. Primero, se copia el nodo padre del bloque de datos antiguo a un nuevo nodo y éste se actualiza para que apunte al nuevo bloque. Este proceso se repite recursivamente hacia arriba hasta llegar a la raíz del árbol, la cual se copia y modifica para que apunte a la nueva versión del árbol.

Al mantener la versión anterior del archivo, se puede regresar a ella en caso de corrupción o algún otro fallo.

Snapshots.

También se mencionó que la técnica de Copy-On-Write facilita la creación de snapshots debido a que nunca se sobrescriben los datos originales. Un snapshot es básicamente un subvolumen de datos que mantiene referencias a versiones anteriores de su estructura. Como se vió en el ejemplo anterior, la versión original del archivo no se elimina, y se mantiene una referencia hacia ella. Es precisamente esta referencia a la versión original un snapshot. Una imagen del estado del sistema de archivos en cierto punto del tiempo.[2][9]

¿Cómo funcionan?

Utilicemos un nuevo ejemplo. Imaginemos que tenemos un archivo “holamundo.txt” que contiene el texto “Hola, mundo.”, y decidimos crear un snapshot en “snapshots/holamundo-v1.txt”. Entramos a esta carpeta, abrimos el archivo de texto, y vemos que contiene el mismo “Hola, mundo.”.

Lo interesante comienza cuando decidimos modificar nuestro archivo. Digamos que modificamos “holamundo.txt” para que diga: “Hola, mundo. Soy un archivo de texto.”. ¿Exactamente qué está haciendo? Básicamente, en vez de copiar completamente el texto de “Hola, mundo.”, el archivo está haciendo referencia a la cadena que ya existía en el snapshot que creamos, y simplemente hace las modificaciones necesarias en base a eso.

¿De qué nos sirve esto? Ahora tomemos como ejemplo un archivo de 5 GB, del cual queremos otra versión con tan solo unos pequeños detalles diferentes. En vez de copiar el archivo completamente y tener dos archivos que llegan a los 10 GB, simplemente se hace referencia al archivo original y de ahí se realizan las modificaciones necesarias.[9]

A esto nos referíamos cuando mencionamos que BTRFS hace un uso más eficiente de los recursos. Por supuesto, este mismo principio se aplicaría a nuestro ejemplo anterior de “archivo.txt” cuando fue modificado, y es gracias a la técnica de Copy-On-Write y el uso de árboles B.

Detección de Errores.

Aparte de la técnica de Copy-On-Write y el uso de Snapshots, otra manera en la que BTRFS protege la integridad de los datos es a través de las sumas de verificación, o “checksums”. Cada vez que se escribe un bloque en el disco, BTRFS realiza un checksum para ese bloque. Este checksum se guarda en un árbol de checksums separado. Así, cada vez que se lee un bloque, se vuelve a calcular su checksum y se compara con el almacenado para verificar que no haya corrupción.[2]

Volviendo al ejemplo inicial, tenemos nuestro “archivo.txt”, y lo modificamos. Digamos que el archivo modificado se va a guardar en el bloque de código “B1”. Antes de escribir nuestro archivo en el disco, se calcula su checksum con alguno de los varios algoritmos disponibles, y esto nos da como resultado “C1”. BTRFS entonces guarda “C1” en el árbol de checksums, y este guarda la dirección lógica del bloque (en este caso, “B1”).

Ahora, cuando intentamos leer “archivo.txt”, BTRFS lo localiza en “B1”, y realiza nuevamente el cálculo del checksum. Digamos que esto nos resulta en “C2”, y comienza a buscar en el árbol de checksums la entrada para “B1”. Como estaba en “C1”, y en el nuevo cálculo del checksum obtuvimos “C2”, esto significa que hubo algún tipo de corrupción. Si hubiera dado “C1” otra vez, esto hubiera significado que no se encontraron problemas.

Conclusión.

En conclusión, BTRFS es un sistema de archivos altamente eficiente gracias a su diseño y diversos algoritmos. La filosofía de Copy-On-Write junto con el uso de árboles B, evita casos de corrupción y nos ahorra valioso tiempo y memoria cuando se tienen varios procesos que solo leen datos sin realizar modificaciones. Además, facilita la

creación de snapshots, los cuales nos permiten alcanzar un mejor uso de la memoria del equipo, evitando copiar datos redundantes cuando simplemente se puede hacer referencia a los datos originales en archivos modificados. Otro uso de los snapshots es facilitar la gestión y control de versiones, que es bastante útil para la realización de backups y restauraciones de sistema. Finalmente, la detección de errores añade todavía otra capa de seguridad para mantener la integridad de los datos y detectar posibles problemas en el sistema. Estas son tan solo algunas de las herramientas con las que cuenta BTRFS, pero existen muchas otras que lo convierten en un sistema para manejar archivos sumamente robusto.

Bibliografía.

1. Rodeh, O., Bacik, J., & Mason, C. (2013). BTRFS: The Linux B-tree filesystem. ACM Transactions on Storage (TOS). https://doi.org/10.1145/2501620.2501623.

2. The Btrfs Wiki. (s.f.). Btrfs Wiki. https://btrfs.readthedocs.io.

3. The Linux Kernel Archives. (n.d.). Btrfs - Linux Kernel Documentation. https://www.kernel.org/doc/html/latest/filesystems/btrfs.html.

4. Red Hat. (s.f.). Chapter 4. Btrfs. https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/6/html/storage_administration_guide/ch-btrfs#ch-btrfs.

5. Bayer, Rudolf. (2008). B-tree and UB-tree. Scholarpedia. http://www.scholarpedia.org/article/B-tree_and_UB-tree.

6. Rodeh, Ohad. (2007). ACM Transactions on Computational Logic. IBM. https://web.archive.org/web/20170102212904/http://liw.fi/larch/ohad-btrees-shadowing-clones.pdf.

7. Qt group. (s.f.). Implicit Sharing. https://doc.qt.io/qt-5/implicit-sharing.html.

8. Spanning Tree. (29/04/2024). Understanding B-Trees: The Data Structure Behind Modern Databases. YouTube. https://www.youtube.com/watch?v=K1a2Bk8NrYQ.

9. Hartmann, Andreas. (30/12/2024). Working with Btrfs – Snapshots. Fedora Magazine. https://fedoramagazine.org/working-with-btrfs-snapshots/.