1/1/2022

1DV512 Operating system

Individual

Assignment 3

Yetnayet Edeglign Belachew
LINNAEUS UNIVERSITY

1. Did some of the earlier file systems supported by Linux before BTRFS focus on support for larger volumes of stored data? If yes, which ones?

Yes, the Linux file system as the 4 extended file systems **EXT4** and **XFS** are the earlier file systems supported by Linux and also support larger volumes of stored data.

The extended file system (EXT4) Introduced in 2006 (BTRFS in 2009) with a hashed B-tree architecture. It is a family that now includes ext2(in 1993) ext3, and ext4. The ext4 file system adds some major features, including files up to 16 Bytes and file systems as large as 1 Ebyte. And also, it replaces the traditional block- mapping mechanism used with its earlier siblings.

The ext4 file system supports persistent pre-allocation. It is useful for applications like streaming media. File system checking is significantly faster, and it also supports delayed allocation.

XFS Introduced in 1994 with B+ tree architecture. It is initially developed by Silicon Graphics. It is developed or parallel I/O based on allocation groups and It is a 64-bit journaling file system. Designed for multiple storage devices and it includes its own volume manager. XFS file system is excellent for large files and claims to have great parallel I/O capability. One drawback of XFS is internal fragmentation - the block sizes are so large that small files and files that are not block aligned leave a large amount of space at the end of the blocks unuse

It uses B+ trees for the directories and files allocation. The file system is partitioned into allocation groups that have their own allocation and free space management. Files are allocated used extents that use contiguous blocks when possible. The number of extents usually grows as a file's size increases. XFS can handle variable block sizes, sparse files, and snapshots. XFS uses a logging system to track updates. This process can be synchronous or asynchronous based on the updates of the files [1].

2. Does BTRFS share any ideas and design goals with ZFS? If yes, which ones?

Yes, both file systems are designed with data integrity in mind, but they also bring many other useful features. For enterprise-grade file systems, which are file systems that are used for business- critical applications, scalability is an important aspect. Although some traditional Linux file systems, for example, XFS, have proven to scale well, **BTRFS** and **ZFS** are designed specifically with scalability in mind.

Hager (2009) have shown that BTRFS has the potential to be the de facto Linux file system, showing similar performance as ext4 and ZFS in experiments using a single disk.

ZFS is assumed to run in a big server with many disks, and it can be assumed that redundancy through RAID-Z is implemented. This means that ZFS can recover bad blocks through reconstruction; failing that, there is a backup. BTRFS cannot assume underlying RAID and multiple devices, hence it must be able to cope with bad blocks by other means.

BTRFS b-tree file system it is based on copy-on-write (COW) it was initially designed at Oracle Corporation for use in Linux

Btrfs is intended to address the lack of pooling, snapshots, checksums, and integral multi-device spanning in Linux file systems

its data resolution in nanosecond, data dedupulation are also their means data modification this are the features of Operating system

ZFS Originally developed by Sun Microsystems for Solaris (now owned by Oracle), ZFS has been ported to Linux. btrfs loose data but no file system corruption

ZFS is better file system then BRTFS because it simply offers the best level of data protection in a small office/home office environment.

ZFS achieves the kind of scalability every modern filesystem should have, with few limits in terms of data or metadata count and volume or file size

3. What is the "extent" concept used for storage by BTRFS? What are its pros and cons compared to the alternative approaches?

Extent: a contiguous on-disk area. It is page-aligned, and its length is a multiple of pages [1].

BTRFS provides **extent**-based file **storage** with a maximum file size of 50 TB and a maximum file system size of 50TB.

– All data and metadata are copy-on-write. This means that blocks of data are not changed on disk. **BTRFS** just copies the blocks and then writes out the copies to a different location [3].

PROS

The rational model allows for an objective approach that's based on scientifically obtained data to reach informed decisions. This reduces the chances of errors, distortions, and assumptions, as well as a manager's emotions, that might have resulted in poor judgments in the past.

This means that, due to the step-by-step methodology, decision-makers are equipped to deal with difficult problems incomplexenvironments.

Compared to the rational model, intuitive decision making allows for quick decisions to be reached, while a degree of gut feeling means managers can eliminate counter-intuitive ideas in reaching their decision. Since it considers the person's emotions, it ensures that positive feelings are used to their advantage, leveraging them as a way to motivate you through the process.

As opposed to the structure of the rational model, which progresses through steps, the intuitive model opts to see everything as a bigger picture. As a result, intuition can help managers to integrate pieces of isolated data, facts, and figures in order to form a cohesive vision of what needs to be done [5].

CONS

The process is sometimes constrained by insufficient information, which creates problems if a manager has to consider and then evaluate, any alternatives they need to reach a decision.

Time limitations can also be an issue. Since there's a lot of information needed, the necessary time for observation, collection and analysis are also essential. In a fast-paced business environment where time is crucial, the rational model is somewhat limited.

It's also an approach that tends to err on the side of caution. By limiting decision making based on what's only available, you may not be able to take the risks necessary for success [2] & [5].

- Btrfs provides extent-**based file storage with a maximum file size of 50 TB and a maximum file system size of 50 TB**.
- All data and metadata are copy-on-write. This means that blocks of data are not changed on disk Btrfs just copies the blocks and then writes out the copies to a different location.
- Not updating the original location eliminates the risk of a partial update or data corruption during a power failure.

- The copy-on-write nature of Btrfs also facilitates file system features such as replication, migration, backup, and restoration of data

4. Which structures does BTRFS use to support larger storage devices and map between physical and logical storage space?

Chunk tree: The chunk tree is used to perform the mapping from logical to physical addresses in BTRFS. All addresses used in BTRFS are logical addresses, which translate to one or more physical addresses depending on the pool configuration. Since also the chunk tree is referenced by its logical address, the superblock contains a part of the chunk tree, the system chunk items, for the initial mapping. This is required to build the chunk tree in the first place. A detailed description of the mapping performed by the chunk tree in BTRFS is given in Section 3.2. Besides, the chunk tree also contains information about the devices used in the pool [4].

A machine may be attached to multiple storage devices [1].

chunks. The rule of thumb is that a chunk should never be more than 10% of the device size. At the time of writing 1GB, chunks are used for data, and 256MB chunks are used for metadata.
A chunk tree maintains a mapping from **logical** chunks to **physical** chunks. A device tree maintains the reverse mapping. The rest of the filesystem sees **logical** chunks, and all extent references address **logical** chunks. This allows moving **physical** chunks under the covers without the need to back trace and fix references. The chunk/device trees are small and can typically be cached in memory. This reduces the performance cost of an added indirection layer. **Physical** chunks are divided into groups according to the required RAID level of the **logical** chunk. For mirroring, chunks are divided into pairs [1].

Larger files are stored in extents. These are contiguous on-disk areas that hold user data without additional headers or formatting. An extent-item records a generation number recording when the extent was created, a [disk block, length] pair to record the area on disk, and a [**logical** file offset, length] pair to record the file area. An extent is a mapping from a **logical** area in a file, to a **physical** area on the disk. If a file is stored in a small number of large extents, then a common operation such as a full file read will be efficiently mapped to a few disk operations. This explains the attractiveness of extents [6].

5. Does BTRFS support copy-on-write? If yes, is it possible to disable it? Could such an approach have impact on performance and fragmentation?

Yes, it supports copy-on-write, and **Yes**, it is possible to disable it.
Use the **nodatacow** mount option. This will only affect newly created files. Copy-on-write will still happen for existing files.
COW operations will also tend to fragment files, wrecking the nice, contiguous layout that the filesystem code puts so much effort into creating. Fragmentation hurts less with solid-state devices than on rotational storage, but, even in the former case, fragmented files will not be as quick to access [1].

6. Do changes to the file system immediately get written to the disk, when using BTRFS? How does this proceed?

No, **BTRFS** is intended to address the lack of pooling, snapshots, checksums, and integral multi- device spanning in Linux filesystems. **BTRFS** provides extent-based file storage with a maximum file size of 50 TB. All data and metadata are copy-on-writing. This means that blocks of data are **not** changed on disk. **BTRFS** just copies the blocks and then writes out the copies to a different location [3].

The copy-on-write nature of **BTRFS** also facilitates file system features such as replication, migration, backup, and restoration of data. Not updating the original location eliminates the risk of a partial update or data corruption during a power failure. It allows you to create both readable and writable snapshots. A snapshot is a copy of an entire **BTRFS** sub volume taken at a given point in time.

Writable snapshots allow you to roll back a file system to a previous state. You can take a snapshot, perform a system upgrade, and reboot into the snapshot if the upgrade causes problems. All snapshots are writable by default, but you also have the option to create read-only snapshots. The snapshots appear as normal directories and you can access the snapshot as you would any other directory.

In BTRFS extent-based storage is offered. The data along with metadata works on the basis of copy-on- write. In other words, it means that data blocks are not immediately changed on disk. They are only copied and written to a separate location.
Thus, the original location is not updated, and it also reduces the risk of data corruption and partial update. This is beneficial when there is a Power filurer.

7. Did Rodeh et al. discover large differences in performance between BTRFS and ZFS in their comparisons? How/why?

Rodeh et al. (2013) compare the performance of Btrfs against the most popular file systems on Linux at the time, XFS and ext4. The motivation for not including ZFS in this comparison, even though he recognized it as an important file system, was that despite the fact that ports existed to Linux, at the time it was not a native Linux file system.

Our contribution is a filesystem supporting important new features, such as snapshots and data checksums while providing reasonable performance under most workloads. An important filesystem that we do not **compare** against is **ZFS**. Section 2.1 covers a lot of its features, and also shows that it has significant differences compared to **BTRFS**. While **ZFS** has ports to Linux, it is not a native Linux filesystem, which prevents a real apple to apple comparison [1].

we chose several common benchmarks, to show that **BTRFS** performs comparably with its contemporaries. At the time of writing, the major Linux filesystems, aside from BTRFS, are XFS and Ext4. These are significantly more mature systems, so we do not expect to perform orders of magnitude better.

While ZFS and BTRFS have a lot in common, they also differ substantially.

For example

Checksums: both ZFS and BTRFS calculate checksums. However, ZFS keeps them in the block-pointers, whereas BTRFS keeps data checksums in a separate tree. This is necessary for BTRFS, since extents could be very large, and we would like to be able to validate each page separately.

Snapshots vs. clones: ZFS uses birth-times to build snapshots. BTRFS uses the reference-counting mechanism instead. The result is very similar, however, the BTRFS mechanism is more general, in that it supports clones as first class citizens.

RAID: ZFS uses RAID-Z and supports several RAID levels including single/dual parity (RAID {5,6}). BTRFS uses something closer to a standard RAID layout; support is currently available only for mirroring, striping, and mirroring striping (RAID levels {0,1,10}).

Backreferences: ZFS does not support back-references, while BTRFS does. ZFS is assumed to run in a big server with many disks, and it can be assumed that redundancy through RAID-Z is implemented. This means that ZFS can recover bad blocks through reconstruction, failing that, there is a backup. BTRFS cannot assume an underlying RAID and multiple devices, hence, it must be able to cope with bad blocks by other means.

Deduplication: ZFS supports deduplication, although this comes at a very significant memory cost. BTRFS does not support this feature at the moment. Due to the memory requirements, it might be a feature only fit for high-end servers.

- Btrfs is thought by many to be the next-generation file system for Linux but it is still under “heavy development” and new features are added continuously.
- ZFS has been around for a longer period of time than Btrfs and it was recently ported to Linux, bringing a stable and feature-rich file system to the platform.

8. Which workloads did Rodeh et al. use for their benchmark tests with FileBench? How did BTRFS compare to the more mature file systems in these tests?

The Web workload emulates a Web server that serves files to HTTP clients. It uses 100 threads that read entire files sequentially as if they were Web pages. The threads, in a 1:10 ratio, append to a common file, emulating a Web log.
The file workload mimics a server that hosts home directories of multiple users. A thread emulates a user, which is assumed to access only files and directories in his home directory. Each thread performs a sequence of creating, delete, append, read, write, and stat operations.

The mail workload mimics an electronic mail server. It creates a flat directory structure with many small files. It then creates 100 threads that emulate e-mail operations: reading mail, composing, and deleting. This translates into manysmallfileandmetadata. operations.
The OLTP workload emulates a database that performs *online transaction processing*. It is based on the IO model used by Oracle TM 9i. It creates 200 data-reader threads, 10 data-writer threads, and one log thread. The readers perform small random reads, the writers perform small random writes, and the log thread does 256KB log writes. The threads synchronize using a set of semaphores, so they all work in concert.

These are just four workloads which by no means exercises all the various ways one can use a filesystem. Hopefully, they are representative of the ways most file systems are used. In most of these cases, **BTRFS** was in the same ballpark as its **more mature** counterparts. In a few cases, low fsync performance was an issue, and this problem is being addressed at the time of writing. For heavy database workloads, the nodatacow mount option will be possible per file in kernel 3.7 [1].

References:

[1]

https://www.researchgate.net/publication/262177144_BTRFS_The_linux_B-tree_filesystem
BTRFS: The linux B-tree filesystem
Article in ACM Transactions on Storage · August 2013

[2]

https://www.jstor.org/stable/42911887?seq=1

JOURNAL ARTICLE

PROS AND CONS OF ALTERNATIVE APPROACHES TO THE TAXATION OF CONSUMPTION

George N. Carlson and Charles E. McLure, Jr.

[3]

https://www.thegeekdiary.com/features-of-the-btrfs-filesystem/

[4]

https://www.researchgate.net/publication/228255043_The_Pros_and_Cons_of_Using_Pros_and_Cons _for_Multi-Criteria_Evaluation_and_Decision_Making

The Pros and Cons of Using Pros and Cons for Multi-Criteria Evaluation and Decision Making

- October 2009
- SSRN Electronic Journal

[5]

https://www.researchgate.net/publication/262177144_BTRFS_The_linux_B-tree_filesystem

BTRFS: The linux B-tree filesystem

Article in ACM Transactions on Storage · August 201

http//www.thegeekdiary.com

[5] https://www.researchgate.net/publication/228255043_The_Pros_and_Cons_of_Using_Pros_and_Cons _for_Multi-Criteria_Evaluation_and_Decision_Making

The Pros and Cons of Using Pros and Cons for Multi-Criteria Evaluation and Decision Making

October 2009