Btrfs State Updates and Future

2 June 2017
Lu Fengqi lufq.fnst@cn.fujitsu.com
Qu Wenruo quwenruo@cn.fujitsu.com
Fujitsu Limited.
Index

- Btrfs Introduction
  - Btrfs existing features
  - Recent fixes and updates
  - Future of Btrfs

- PostgreSQL and Btrfs
  - Main dish
  - An interesting glance into the “uncommon” use case of btrfs and PostgreSQL
Btrfs existing features

- Btrfs is famous for its long feature list
  - Although other filesystems are catching up. E.g. XFS and OCFS2

- The following features will be introduced
  - Copy-on-write
  - Reflink
  - Checksum
  - Multi-device management

- That’s just the tip of the iceberg
  - For more, check btrfs wiki
  - [https://btrfs.wiki.kernel.org/index.php/Main_Page](https://btrfs.wiki.kernel.org/index.php/Main_Page)
Copy-on-write

- Copy-on-write is the **core** of btrfs
  - Mandatory metadata copy-on-write

1. Old Tree root
2. Old tree root
3. New tree root

- Data copy-on-write (Can be disabled)

1. File A
2. File A

- Better protection for power loss
- Provides basis for snapshot/relink
Reflink: A new method to “copy” data

- **Deep copy VS shallow copy**

- **Fast, space efficient**
  - XFS and OCFS2 join the reflink party
Checksum

- Btrfs supports checksum for both data and metadata
  - Mandatory metadata checksum
  - Default data checksum, can be turned off

**Pros**
- Detect corruption and auto-repair if backup is available
  - Important for **large** fs and **enterprise** usage
- Better than pure **block-level** backup
  - **RAID1** can’t really fix corruption without **checksum**

**Cons**
- Extra CPU usage
Multi device management and RAID

- Btrfs provides flex multi-device management
  - **Dynamic** allocation (like LVM thin-provision)
  - Metadata and data can have different RAID level
  - Co-operate with checksum, better recovery
  - **Stable** Single/DUP/RAID0/RAID1/RAID10 support
  - **Experimental** RAID5/6 support (v4.12 has btrfs RAID56 fixes)
Kernel Updates

- Btrfs is still **hot**
  - Kernel part
    - v4.8 – Oct 2016
    - v4.9 – Dec 2016
    - v4.10 – Feb 2017
    - v4.11 – May 2017

**Commits**

**Changed lines**

- Add
- Remove
- Net
Kernel Updates

► Btrfs kernel updates
  ▶ Most of them are bug fixes and cleanups
    • In v4.10, btrfs went through a large cleanup of internal APIs.
    • Bug fixes for variants parts
  ▶ ENOSPC rework
    • Much faster ENOSPC detection
    • Fast enough to expose existing but hidden bugs! (E.g. compression)
  ▶ Better space cache tree support

► Btrfs is getting more and more stable
User tool updates

- Btrfs-progs (user-tool) is **hot** too
  - v4.7 – July 2016
  - v4.8 – Oct 2016
  - v4.9 – Dec 2016
  - v4.10 – Mar 2017

Commit and Changed Lines graphs for versions v4.7 to v4.10.
User tool updates

- New feature along with bug fixes
  - Low memory usage mode btrfs check
    - Still experimental
    - No OOM checking large fs
    - Repair mode is pending
  - Convert/rollback rework
    - Support for new source fs is ready (Reiserfs?)
  - Better self tests
  - Tons of cleanup
    - Started in v4.9, continued in v4.10

- Btrfs-progs is getting larger and larger
  - No only a wrapper for btrfs ioctl
  - To modify unmounted btrfs just like kernel
Future of Btrfs

- Bug fixes will be the main part
  - RAID56
  - Quota
  - Compression ENOSPC
  - Better btrfs-corrupt replacement in btrfs-progs
  - Better RAID related test cases

- Hopefully some new feature
  - Sub-page size sector support
  - Inband de-duplication
Index

■ Btrfs Introduction
  ■ Btrfs existing features
  ■ Recent fixes and updates
  ■ Future of Btrfs

■ PostgreSQL and Btrfs
  ■ Main dish
  ■ An interesting glance into the “uncommon” use case of btrfs and PostgreSQL
Database and Btrfs

- **Common use cases of btrfs**
  - Backup storage
    - *Delta backup* makes it quite handy
  - Root fs
    - No need to worry about config error, thanks to *snapshots*

- **Database on Btrfs?**
  - No production usage of btrfs yet
  - Expected low performance, but is that true?
PostgreSQL on Btrfs

Advanced Database on Advanced Filesystem
- Not meant to enhance Btrfs nor PostgreSQL (yet)
- To find possible **bottleneck** and **workaround**
- Not a real-work benchmark
  - TPC-B focused
    - Measures throughput in terms of how many transactions per second a system can perform
    - **Spoiler**: Btrfs doesn’t perform poorly in SELECT only test, not interesting at all
  - Traditional HDD focused
  - Low load focused
    - **Spoiler**: Where btrfs falls far behind

Some early expectation
- Slow performance
  - Data checksum and cow will slow it
- Need heavy fine tuning to get normal performance
  - Xfs/Ext4 will dominate
TPC Performance

Different fs for /var/lib/postgres, pgbench TPC-B test

- **Low performance** for low concurrency use (only 50% of xfs/ext4)
- **Catch up** in high concurrency case
- Btrfs fine tuning seems **useless** in low concurrency case

But is that all?
PostgreSQL on Btrfs

- A different story: WAL on Ext4, DB files on btrfs
  - Ext4 to store Write-Ahead-Log (WAL) of PostgreSQL
    - WAL can be considered as an extra journal to ensure database integrity
  - Btrfs is no worse than ext4/xfs
    - Fine tuned btrfs even out performs ext4/xfs for certain cases
  - Nodatacow does not always help

![Graph showing performance comparison between filesystems](chart.png)
PostgreSQL on Btrfs

- Low performance: DB files on Ext4, WAL on Btrfs
  - Ext4 to store DB files
  - Btrfs **falls far behind** xfs/ext4 for low&mid concurrency cases
  - Default WAL is **fdatasync** focused
Fdatasync on Btrfs with HDD

How slow is btrfs fdatasync?

- **Single thread overwrite** 4K data and **fdatasync**

![Bar chart showing performance comparison between Btrfs, Btrfs with nodatacow, Btrfs with 4K single meta, Ext4, and Xfs.]

- **Btrfs (default)**
- **Btrfs (nodatacow)**
- **Btrfs (4K single meta)**
- **Ext4 (default)**
- **Xfs (default)**

Fine tuning doesn't help much:
- Amount of metadata affects, but **marginally**
- **Data CoW/Checksum** affects, but **marginally** too
Fdatasync on Btrfs with HDD

- **Time consumption of fdatasync**
  - Round 1 (1st 4K write + fdatasync)

![Graph showing time consumption of fdatasync for Btrfs, Ext4, and Xfs]

- **Write** is fast, thanks to write cache of HDD controller
- Ext4 doesn’t update its journal, while Xfs does (for first write)
- **Flush** and **Force Unit Access (FUA)** are slow
  - Libata disables FUA detection by default, FUA fallbacks to flush
- Btrfs and XFS are doing **2 slow operations**, other than **1** of Ext4
Fdatasync on Btrfs with HDD

- Time consumption of fdatasync

  - Round 2~5
    - Another 4K overwrite and fdatasync, just after Round 1

  ![Bar graph showing time consumption of fdatasync for Btrfs, Ext4, and Xfs](image)

- Performance get stable for **FUA** and **Flush**
- Btrfs’ extra **FUA** is **affecting** performance
  - Due to the Metadata CoW, btrfs must **update** its main **superblock**
- Xfs no longer updates its journal, **improving** the performance.
Fdatasync on Btrfs with HDD

- Ideas to enhance btrfs fdatasync/fsync
  - Reduce **metadata CoW** for fdatasync/fsync?
    - Only possible for **nodatacow** case
      - **CoW or append write** still needs to CoW log and log root trees
    - Avoid updating btrfs log tree, so only **data flush, no superblock update**
    - The most possible **software solution** yet for HDD

- **Native FUA** to reduce overhead?
  - Libata **disables FUA** detection by **default**
  - **Not all** HDD supports native FUA
  - Native FUA enhancement can be **marginal** for HDD
    - **Seek time** is still a big problem.
  - **SSD/NVMe** maybe the game changer?
    - Super fast FUA (same as cached write) detected for Intel 600P NVMe SSD
    - Did I say we’re focusing on HDD?
Fdatasync on Btrfs with NVMe

- Btrfs fdatasync on NVMe
  - With NVMe, the FUA for superblock update is **super fast**
    - FUA write is as fast as cached write for Intel 600P NVME
  - So btrfs fdatasync performance improves
    - Same level as Ext4 for finetuned btrfs, but still slower than XFS

![Graph showing performance comparison between different file systems](image-url)
 PostgreSQL performance also changes

- Low concurrency performance is **almost the same** as Xfs/Ext4
  - Thanks to super fast FUA
- But **new performance bottleneck** is exposed for mid&high concurrency case
  - High concurrency performance **falls far behind**

![Graph showing TPS vs number of clients for different file systems: Btrfs(default), Btrfs(nodatcow), Btrfs(compress=lzo), Ext4(default), Xfs(default).](image)
Fdatasync on Btrfs with NVMe

- **Btrfs with NVMe SSD**
  - **Improvement**
    - Fast FUA of NVMe **eliminate** the extra fdatasync **overhead** for btrfs
    - Extra superblock update is **no longer an obvious bottleneck**

- **New Bottleneck** for btrfs
  - Poor performance for **high concurrency load**

- **New work**
  - **Locate and fix** btrfs bottleneck on NVMe SSD
    - Tree lock ?
    - Tree layout ?
    - Extent allocator ?
    - ...

Copyright 2017 FUJITSU LIMITED
Practice Learnt

- **Data Cow/Checksum** not always the problem
  - Locating real bottleneck is more important

- **Metadata CoW** can cause problem
  - Extra superblock update can be **performance bottleneck**
  - Unavoidable, especially for fdatasync/fsync

- **NVMe SSD** is always a **game changer**
  - Super fast write/FUA/seek can expose hidden performance bottleneck
Q&A Time

- Q&A
shaping tomorrow with you