

Btrfs

Design, Implementation and the Current Status

Red Hat

Lukáš Czerner

February 18, 2012

Copyright © 2012 Lukáš Czerner, Red Hat.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the COPYING file.

Agenda

- 1 Btrfs and its place in the Linux file system world
- 2 Design and implementation of Btrfs
- 3 Btrfs Features
- 4 Current status
- 5 Who writes Btrfs



Part I

Btrfs and its place in the Linux file system world

Linux kernel file systems

- There are approximately 55 kernel file systems in Linux kernel tree
- A lot of them has limited or specific use
- ExtN file system family considered the *only* "general purpose file system" for a long time
- Most active local file systems = **xfs**, **btrfs**, **ext4**

Linux kernel file systems challenges

- **Scalability** - ability to grow and remain efficient
 - ext4 just breached 16TB limit
 - xfs scale up to 200TB and more
- **Reliability** - ability to recover from incidents
 - Silent data corruption
 - Metadata corruption tolerance
 - File system repair on huge file systems
- **Advanced Features** - more than just storing data
 - Snapshotting
 - Encryption
- **Ease of use** - reduce administration overhead
 - Growing storage stack - administration overhead

What Btrfs has to offer ?

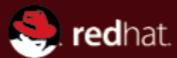
- **Scalability**
 - Does not have fixed positions for metadata (mostly)
 - 16 EiB file/file system size limit
- **Reliability**
 - Very fast file system creation
 - Possibly very fast file system check
 - Data + metadata checksumming
 - Incremental backup + snapshotting
 - Online scrub to find and fix problems
- **Advanced features**
 - Integrated volume management (RAID)
 - Integrated snapshotting support
 - Reblink
- **Ease of use**
 - Integrated easy-to-use volume management

The idea

- IBM researcher Ohad Rodeh at Linux Storage and File system workshop 2007
- COW friendly btree
 - Leaves can not be linked to their neighbors
 - Reference counting for easy tree cloning
 - Proactive merge/split

The file system

- Chris Mason liked the idea of COW friendly Btrees
- Lot of previous experience from Reiserfs
- Using COW btree for every object in the file system
 - COW advantages for **free**



Part II

Design and implementation of Btrfs

Generic btree implementation for all objects

- The tree does not care about object types
- Each tree block carries **header** and **key**
 - Header stores the location on disk + pointers to other blocks
 - The key defines order in the tree block layout
- Regardless on the operation btrfs uses the same code path
 - Reads
 - Writes
 - Data / Metadata allocation

Superblock

- Stored on all devices
- Mirror copies of superblock kept up-to-date (not in SSD case)
- Main trees positions
 - root tree
 - chunk tree
 - log tree

Everything is stored in Btrees

- Single btree manipulation code
- Few different types of trees
 - 1 root tree - roots of other trees
 - 2 chunk tree - logical to physical mapping
 - 3 device allocation tree - device parts into chunks
 - 4 extent allocation tree - file system wide
 - 5 fs tree - inodes, files, directories
 - 6 checksum tree - block checksums
 - 7 data relocation tree
 - 8 log root tree
- Each tree has its own specific ID
- Only leaves stores the actual data



Leaves and nodes

- Every tree block is either leaf or node
- Every leaf and node begins with the header
- **Nodes** [header, key_ptr0....key_ptrN]

```

struct btrfs_node {
    struct btrfs_header header;
    struct btrfs_key_ptr ptrs[];
}

struct btrfs_key_ptr {
    struct btrfs_disk_key;
    __le64 blockptr;
    __le64 generation;
}

```

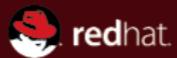
- **Leaves** [item0....itemN] [free space] [dataN...data0]

```

struct btrfs_leaf {
    struct btrfs_header header;
    struct btrfs_item items[];
}

struct btrfs_item {
    struct btrfs_disk_key key;
    __le32 offset;
    __le32 size;
}

```



Part III

Btrfs Features

Transactions in Btrfs

- There is no journal as extN, or xfs has
- COW is used to guarantee consistency
- On fs tree or extent tree modification
 - 1 Tree is cloned and the branch in question is copied and modified
 - 2 New roots are added into root tree (identified by transaction ID)
 - 3 New root of root tree added into superblock
 - 4 Wait on all respective data and metadata to hit the disk
 - 5 Commit the superblock to the disk
 - 6 Original trees can be freed (decrease refcount)
- In case of crash, the original root tree is used (the one in the most up-to-date superblock)

Snapshots in Btrfs

- Can be read only, or read write
 - If read only - block quota set to 1
- **Snapshots are subvolumes**
 - Share parts of the original root
- Created the same way as described in transactions
 - The original tree is not automatically freed
- Can be used in the same way as any subvolume
- Can be created instantly at any point in time



Checksums in Btrfs

- Both metadata **and** data are checksummed
- Checksummed blocks of different sizes (data extents)
- Calculated only before written out to the disk
- Data + Metadata can be verified after read from disk
- Improves reliability
- Online failover
- Online scrub

File system scrub

- Using checksums to validate all data + metadata
- Can fix errors if possible (mirror setup)
- Works online at a background
- start, cancel, resume, status
- `btrfs scrub start [-Bdqr] {<path>|<device>}`

Volume management

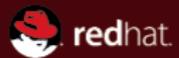
- Multi-disk support
 - Easy to add / remove drives - **pooling**
 - Multiple extent allocation trees
 - Raid0, Raid1, Raid10
- Subvolumes
 - Multiple fs roots allows creating multiple subvolumes
 - Each subvolume appears as directory in root volume
 - Create snapshot of a subvolume (snapshot of snapshot of ...)
 - You can easily mount a subvolume with `-o subvol=<name>` or `subvolid=<ID>`
 - Users manage subvolumes in their subvolume
 - Easy-to-use management

File system check

- It is ready now (sort of)
- Should be **really** ready by this summer (maybe)
- Has ambitions to be really fast
- Memory consumption should be quite low

And more

- Online defragmentation
- Online balancing
- Transparent compression
- Data deduplication
- Data encryption
- Volume balancing



Part IV

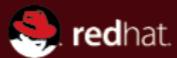
Current status

What is already done ?

- Online defragmentation
- Online balancing
- Compression
- Free space cache
- Reblink
- Recently modified files
- Offline file system check (sort of)

What is still missing ?

- **Offline file system check**
- Raid 5/6
- Deduplication
- Encryption
- Fragmentation problem ?
- More testing and stabilization



Part V

Who writes Btrfs?

Who writes Btrfs for the last year?

- 637 commits
- 38658 lines changes (26160 inserted, 12498 deleted)
- Contribution from 63 developers from the last year
 - Josef Bacik (Red Hat)
 - Li Zefan (Fujitsu)
 - Chris Mason (Oracle)
 - Al Viro (Red Hat)
 - and more...



The end.

Thanks for listening.