
facebook

Scaling the Btrfs Free Space Cache

Omar Sandoval

Vault 2016

Outline

- Background
- Design Background
- Identifying the Problem
- Solution
  - The Free Space Tree
- Results
- Questions

Background

What is Btrfs?

- General-purpose filesystem
- Advanced features
  - First-class snapshots
  - Checksumming of all data and metadata
  - Transparent compression
  - Integrated multi-device support (RAID)
  - Incremental backups with send/receive
- Actively developed

Contributors

```
$ git log --since="January 1, 2015" --pretty=format:"%an" --no-merges -- fs/btrfs
```

Filipe Manana	Dan Carpenter	Borislav Petkov	Mike Snitzer
David Sterba	Forrest Liu	Andreas Gruenbacher	Mel Gorman
Zhao Lei	Yang Dongsheng	Alex Lyakas	Matthew Wilcox
Qu Wenruo	Tsutomu Itoh	Zach Brown	Konstantin Khebnikov
Anand Jain	Michal Hocko	Yauhen Kharuzhy	Jan Kara
Josef Bacik	Kirill A. Shutemov	Yaowei Bai	Guenter Roeck
Omar Sandoval	Jiri Kosina	Vladimir Davydov	Greg Kroah-Hartman
Chris Mason	Daniel Dressler	Tom Van Braeckel	Geert Uytterhoeven
Zhaolei	Wang Shilong	Sudip Mukherjee	Gabríel Arthúr Pétursson
Liu Bo	Satoru Takeuchi	Shilong Wang	Dmitry Monakhov
Chandan Rajendra	Naohiro Aota	Shaohua Li	Deepa Dinamani
Dongsheng Yang	Luis de Bethencourt	Shan Hai	Davide Italiano
Alexandru Moise	Kinglong Mee	Sebastian Andrzej Siewior	Dave Jones
Mark Fasheh	Kent Overstreet	Sasha Levin	Darrick J. Wong
Eric Sandeen	Justin Maggard	Sam Tygier	Colin Ian King
Jeff Mahoney	Jens Axboe	Robin Ruede	Chengyu Song
Byongho Lee	Holger Hoffstätte	Rasmus Villemoes	chandan r
Christoph Hellwig	Gui Hecheng	Quentin Casasnovas	Ashish Samant
Al Viro	Fabian Frederick	Pranith Kumar	Arnd Bergmann
chandan	David Howells	Oleg Nesterov	Adam Buchbinder
Geliang Tang	Christian Engelmayer	NeilBrown	

Btrfs at Facebook

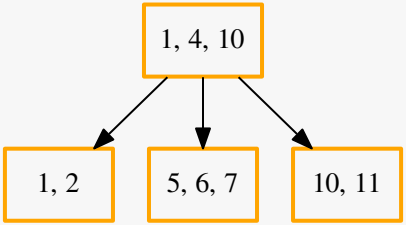
- GlusterFS
- Testing on other tiers
  - Web servers
  - Build machines
  - More...

Design Background

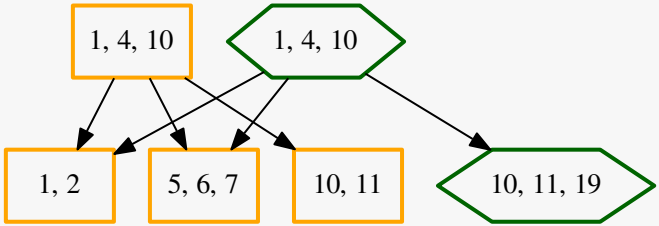
Copy-on-Write B-trees

- B+-trees with no leaf chaining
- Lock coupling
- Relaxed balancing contraints, proactive balancing
- Lazy reference-counting
- Update in memory, write out in periodic commits
- Due to O. Rodeh

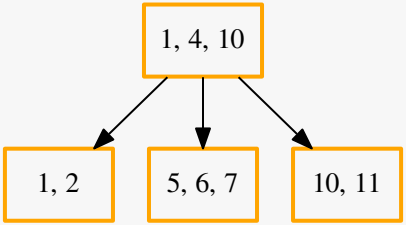
Copy-on-Write B-tree Insertion



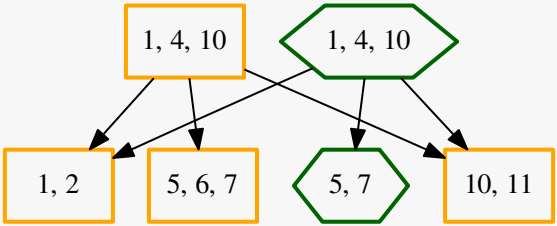
Copy-on-Write B-tree Insertion



Copy-on-Write B-tree Deletion



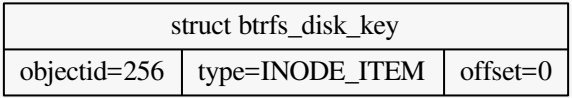
Copy-on-Write B-tree Deletion



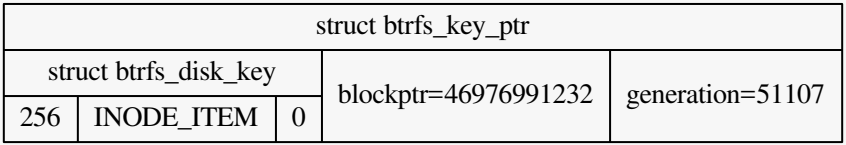
Btrfs B-trees

- Generic data structure
- Three main structures:
  - *Block header*: header of every block in the B-tree, contains checksum, flags, filesystem ID, generation, etc.
  - *Key*: comprises 64-bit object ID, 8-bit type, and 64-bit offset
  - *Item*: comprises key, 32-bit offset, and 32-bit size
- *Nodes* contain only keys and block pointers to children
- *Leaves* contain array of items and data

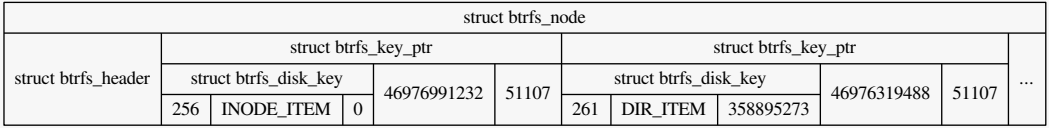
B-tree Nodes



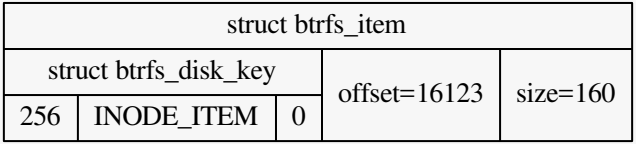
B-tree Nodes


struct btrfs_key_ptr
struct btrfs_disk_key
256
INODE_ITEM
0
blockptr=46976991232
generation=51107

B-tree Nodes


struct btrfs_node
struct btrfs_header
struct btrfs_key_ptr
struct btrfs_disk_key
256
INODE_ITEM
0
46976991232
51107
struct btrfs_key_ptr
struct btrfs_disk_key
261
DIR_ITEM
358895273
46976319488
51107
...

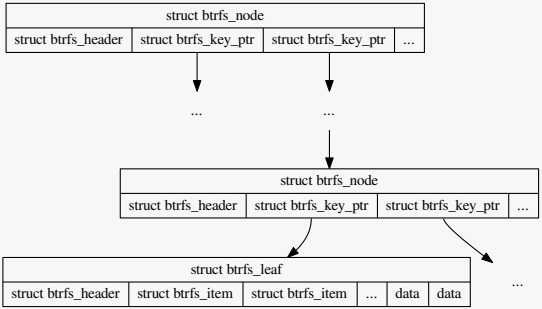
B-tree Leaves



B-tree Leaves

<table>
<tr><td colspan="12">struct btrfs_leaf</td></tr>
<tr><td rowspan="2">struct btrfs_header</td><td colspan="5">struct btrfs_item</td><td colspan="5">struct btrfs_item</td><td rowspan="2">...</td><td rowspan="2">struct btrfs_inode_ref</td><td rowspan="2">struct btrfs_inode_item</td></tr>
<tr><td colspan="3">struct btrfs_disk_key</td><td rowspan="2">16123</td><td rowspan="2">160</td><td colspan="3">struct btrfs_disk_key</td><td rowspan="2">16111</td><td rowspan="2">12</td></tr>
<tr><td>256</td><td>INODE_ITEM</td><td>0</td><td>256</td><td>INODE_REF</td><td>256</td></tr>
</table>

B-tree Structure



B-trees Used in Btrfs

- Root tree
- FS trees
- Extent tree
- Checksum tree
- Chunk tree, device tree
- Relocation tree, log tree, quota tree, UUID tree

B-trees Used in Btrfs

- Root tree
- FS trees
- Extent tree
- Checksum tree
- Chunk tree, device tree
- Relocation tree, log tree, quota tree, UUID tree
- Free space tree: subject of this talk!

Identifying the Problem

Symptoms

- Large (tens of terabytes), busy filesystems on GlusterFS
- Stalls during commits, sometimes seconds long
- Narrowed down the problem to write-out of the free space cache

Terminology

- *Chunk*: physical unit of allocation, 1 GB for data, 256 MB for metadata
- *Block group*: logical unit of allocation, comprises one or more chunks
- *Extent*: range of filesystem used for a particular purpose

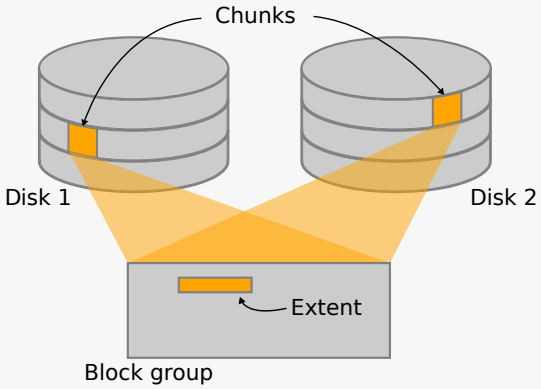


Figure: Basic terminology

Extent Tree

- Tracks allocated space with references
- Three types of items:
  - *Block group items*
  - *Extent items*
  - *Metadata items*

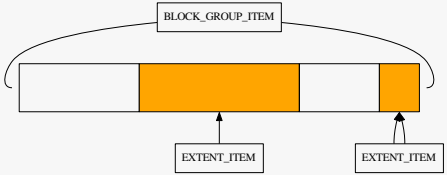


Figure: Data block group

Extent Tree

- Tracks allocated space with references
- Three types of items:
  - *Block group items*
  - *Extent items*
  - *Metadata items*

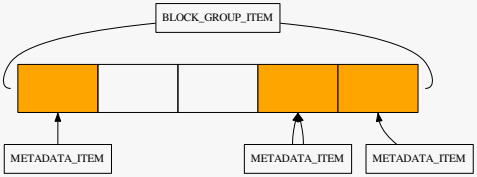


Figure: Metadata block group

Free Space Cache

- Track free space in memory after loading a block group
  - Red-black tree
  - Hybrid of extents + bitmaps
- Loading directly from the extent tree is slow
- Instead, dump the in-memory cache to disk
  - Special free space inodes, one per block group
  - Compact, very fast to load
  - Has to be to written out in entirety

Delayed Refs

- Don’t want to update extent tree on disk for every operation
- Instead, track updates in memory (red-black tree)
- Run at commit time as a batch

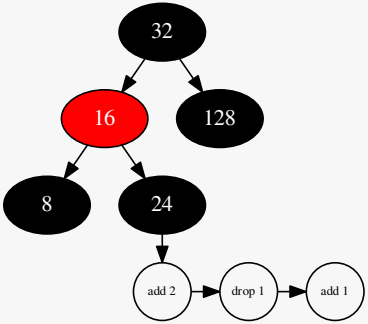


Figure: Delayed refs tree

Diagnosis

- Large, busy filesystem can dirty many block groups
- Have to write out the free space cache for each one
- Run during the critical section of the commit
  - Blocks other operations

Solution

False Start

- Start writing out free space cache outside of critical section
- Redo block groups that got dirtied again during that window
- Still possible for a lot to get dirtied again
    - Back to square one

The Free Space Tree

Key Ideas

- Use another B-tree instead
- Inverse of the extent tree

On-Disk Format

- Per-block group free space info
  - Extent count
  - Flags
- Free space extents
  - Start
  - Length
- Free space bitmaps
  - Start
  - Length
  - Bitmap

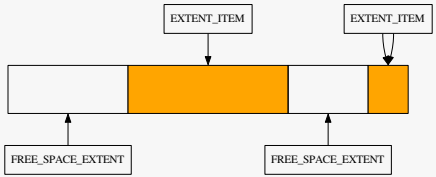


Figure: Free space tree extents

On-Disk Format

- Per-block group free space info
  - Extent count
  - Flags
- Free space extents
  - Start
  - Length
- Free space bitmaps
  - Start
  - Length
  - Bitmap

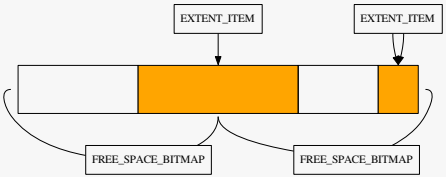


Figure: Free space tree bitmaps

Maintaining the Free Space Tree

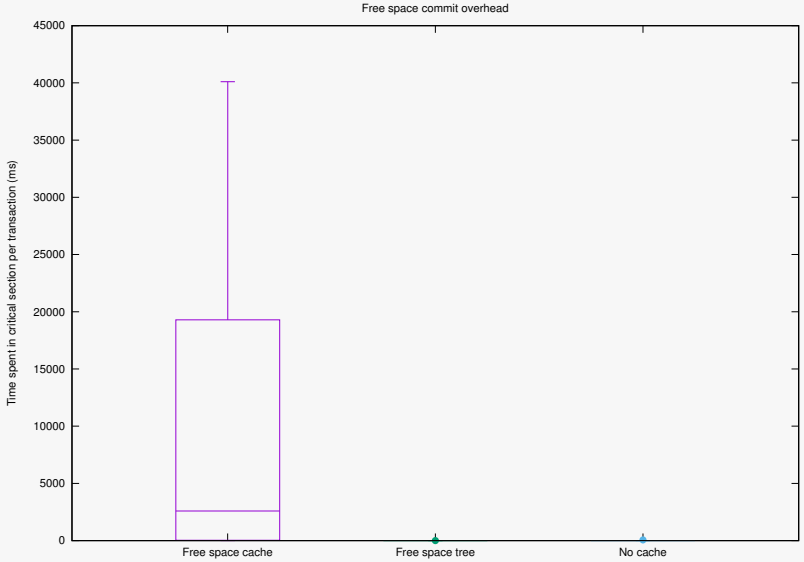
- Piggy-back updates on delayed-refs
  - Get batching for free
- Convert between extents or bitmaps
  - Thresholds for whichever is more space-efficient
  - Buffer to avoid thrashing between formats

Results

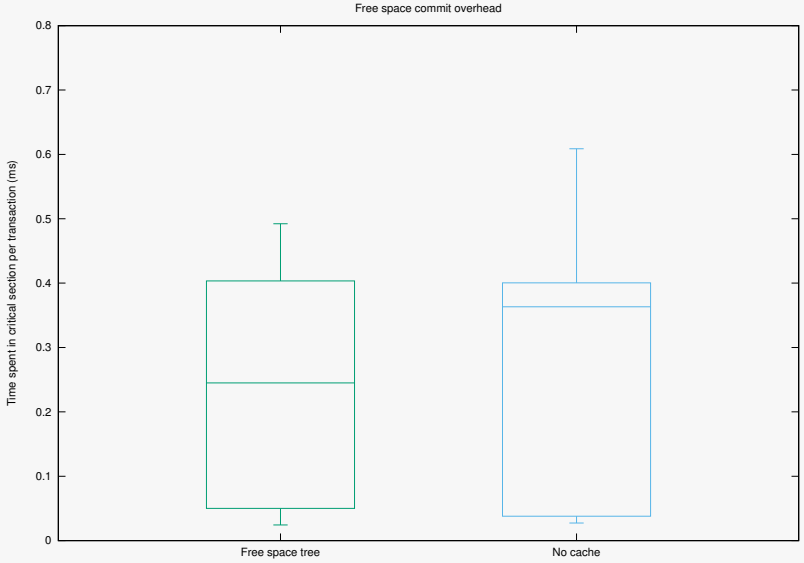
Measuring Commit Overhead

- Use `fallocate` and `unlink` to dirty a lot of block groups
- Sparse filesystem image on loop device to allow for exaggerated sizes (50TB of dirtied space)
- Measure time spent committing, esp. in critical section

Commit Overhead



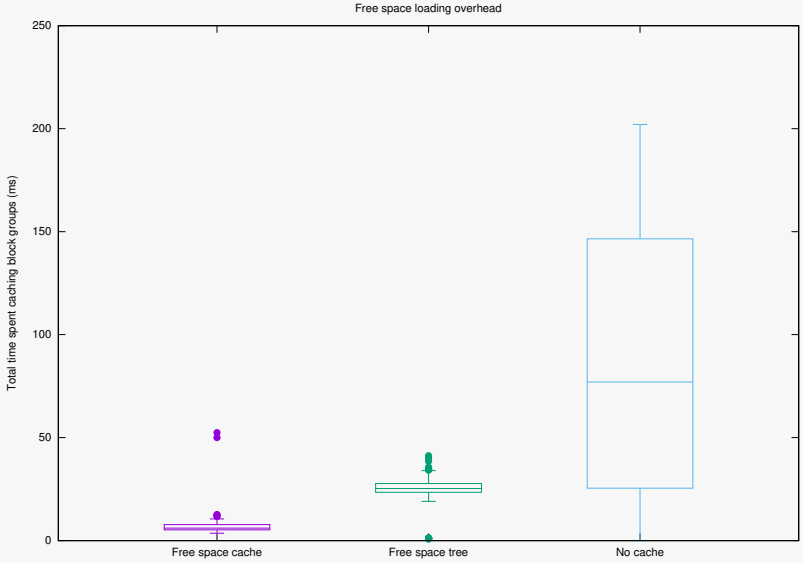
Commit Overhead



Measuring Load Overhead

- Create a fragmented filesystem with `fs_mark`
- Unmount, remount
- Run `fs_mark` again
- Measure time spent loading space cache

Load Overhead



Try It Out

```
mount -o space_cache=v2
```

(Since Linux v4.5)

Questions


facebook