# BtrFS: A fault tolerant File system

Kumar Amit Mehta

Date: 12/06/201
Coursework:
IAF0530

# Agenda

- Motivation
- Background
- BtrFS features
- Main principles and technical details
- Test results
  - Robustness
  - Performance
- Demo
  - Fault tolerance
- Conclusion
- Q&A

# Motivation

- Relevance


Source: dilbert.com

# Motivation contd...

- Main factors for Storage faults
  - Firmware/Software Bug
  - Hardware Failure
    - Aging
    - Faulty hardware
    - Annual disk replacement rates typically exceed 1%, with 2–4% common and up to 13% observed on some systems [7].
  - Complexity
    - SUN took 7 years to develop ZFS and another few years more until ZFS was deployed in mission critical systems.
    - “File System are hard” - Ted Ts'o (Maintainer of ext4)

# Motivation contd...


```
	/* Sun, you just can't beat me, you just can't.  Stop trying,
	 * give up.  I'm serious, I am going to kick the living shit
	 * out of you, game over, lights out.
	 */
	.align	8
	.globl	__csum_partial_copy_sparc_generic
```

<Snip from linux/arch/sparc/lib/checksum_32.S>

# Background

- Historical Perspective
  - “B-trees, Shadowing, and Clones Ohad Rodeh [1]” (USENIX, 2007)
  - Chris Mason (Combined ideas from ReiserFS and COW friendly B-trees as suggested by Rodeh )
  - Finally accepted in mainline Linux Kernel in 2009
  - Default root File system for SuSE, Oracle Linux
  - 2014, Facebook announced [2] to use BtrFS as Trial

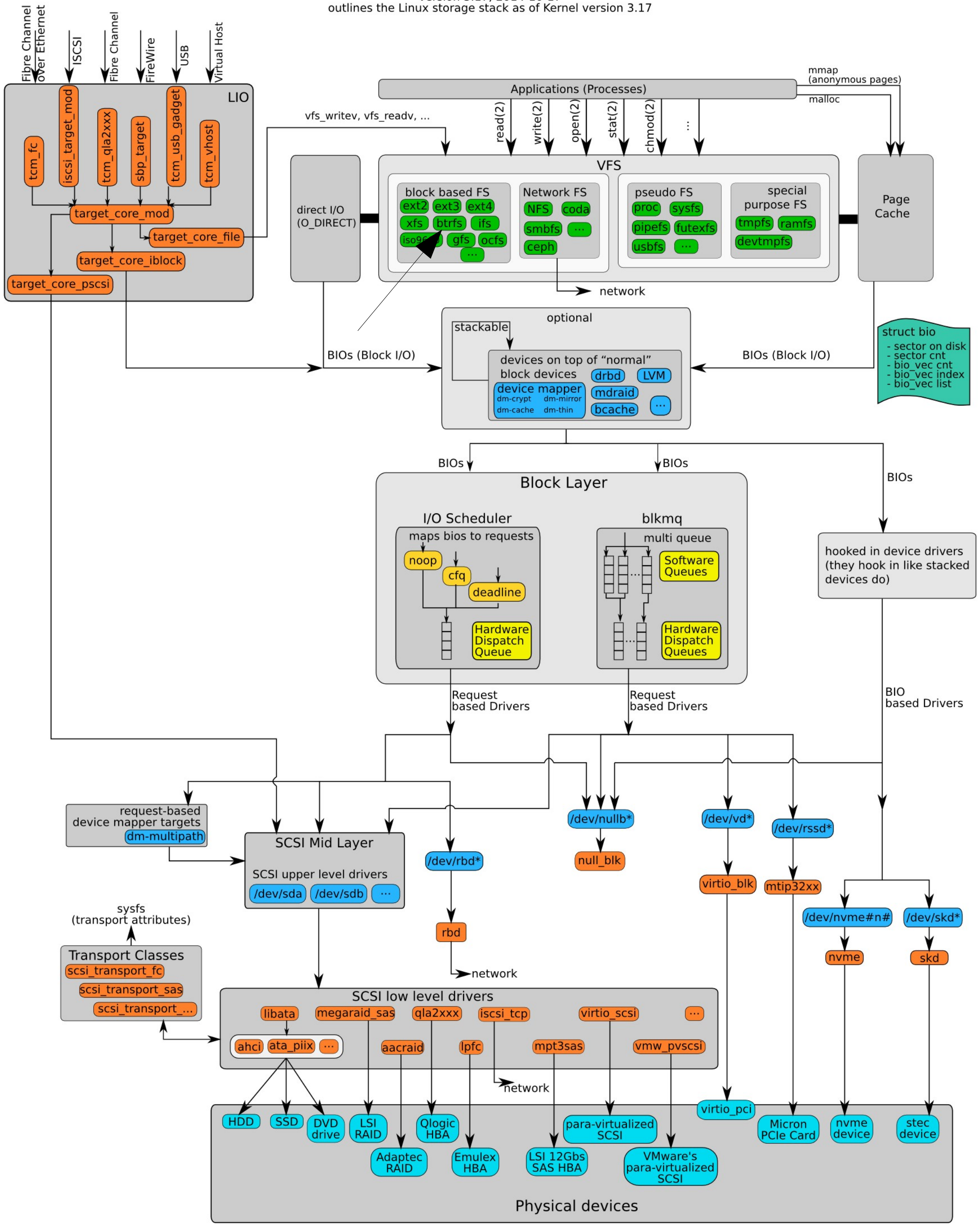
| | |
|---|---|
| **Date** | Tue, 12 Jun 2007 12:10:29 -0400 |
| **From** | Chris Mason <> |
| **Subject** | [ANNOUNCE] Btrfs: a copy on write, snapshotting FS |

```
Hello everyone,

After the last FS summit, I started working on a new filesystem that
maintains checksums of all file data and metadata.  Many thanks to Zach
Brown for his ideas, and to Dave Chinner for his help on
benchmarking analysis.
```

# The Linux Storage Stack Diagram

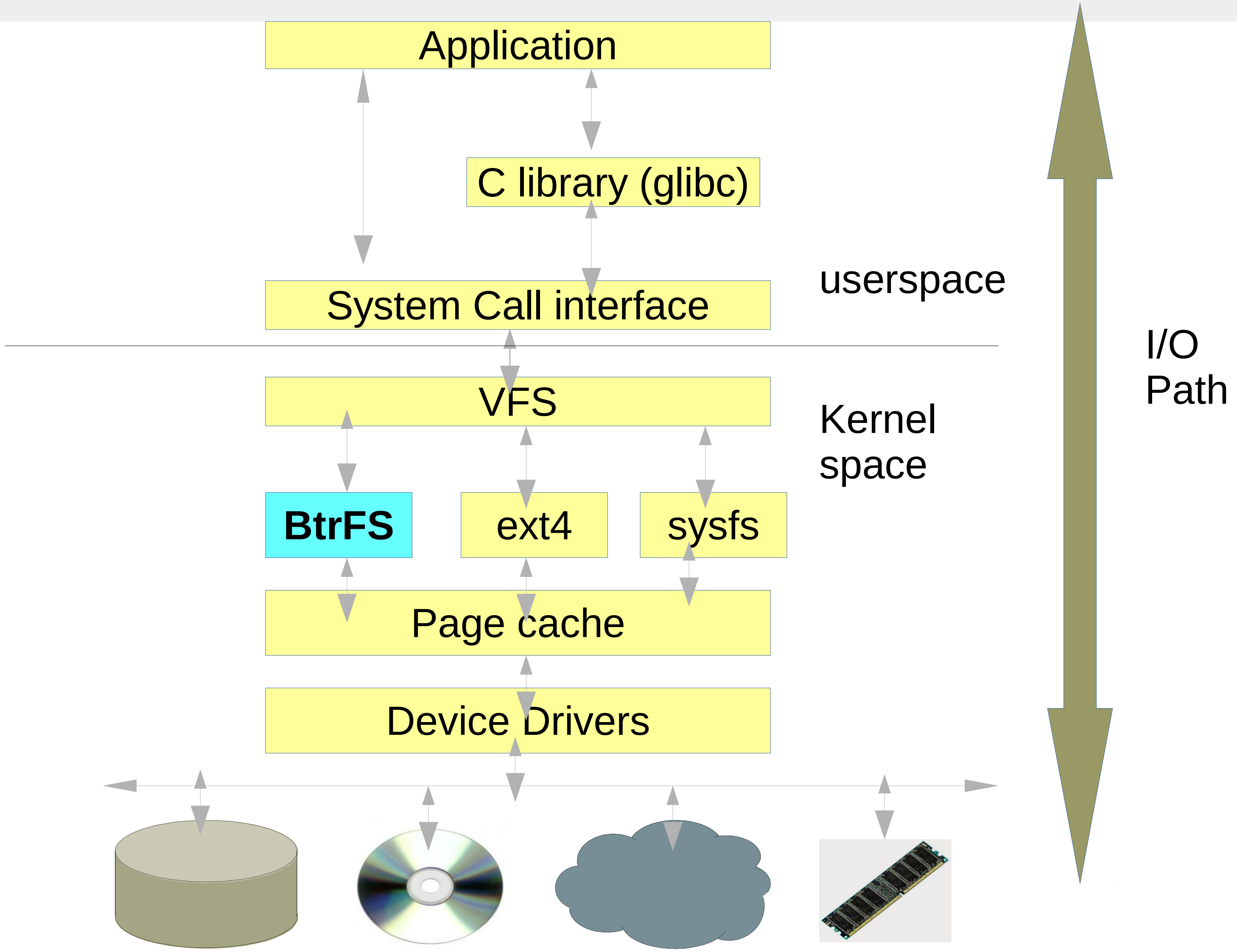
version 3.17, 2014-10-17
outlines the Linux storage stack as of Kernel version 3.17




THOMAS KRENN®
server.hosting.customized.

The Linux Storage Stack Diagram
http://www.thomas-krenn.com/en/wiki/Linux_Storage_Stack_Diagram
Created by Werner Fischer and Georg Schönberger

# Background – File System



# Background – File System

- Design Issue
  - Reliable Storage
    - Normal usage
    - Failure conditions
  - Fast Access
    - Different Scenarios
  - Efficient layout
    - Small files
    - Lots of files

- Operational Issues
  - Vulnerability windows
    - Log but only metadata
    - RAID write hole
  - Recovery
  - Defragmentation
  - Large directories
  - Resizing

Source: dclug.tux.org/200908/BTRFS-DCLUG.pdf

# Background – File System

- Design Issue
  - Reliable Storage
    - Normal usage
    - Failure conditions
  - Fast Access
    - Different Scenarios
  - Efficient layout
    - Small files
    - Lots of files

- Operational Issues
  - Vulnerability windows
    - Log but only metadata
    - RAID write hole
  - Recovery
  - Defragmentation
  - Large directories
  - Resizing

Source: dclug.tux.org/200908/BTRFS-DCLUG.pdf

# Background – File System

- Design Issue
  - Reliable Storage
    - Normal usage
    - Failure conditions
  - Fast Access
    - Different Scenarios
  - Efficient layout
    - Small files
    - Lots of files

- Operational Issues
  - Vulnerability windows
    - Log but only metadata
    - RAID write hole
  - Recovery
  - Defragmentation
  - Large directories
  - Resizing

Source: dclug.tux.org/200908/BTRFS-DCLUG.pdf

# BtrFS Features

- Key features
  - Extent based file system
  - Writable snapshots, read-only snapshots
  - Subvolumes (separate internal filesystem roots)
  - Checksums on data and metadata (crc32c)
  - Integrated multiple device support
    - File Striping, File Mirroring, File Striping+Mirroring, Striping with Single and Dual Parity implementations
  - Background scrub process
  - Offline filesystem check
  - Online filesystem defragmentation

# Main principles

- Inodes
- Extent based file system
- File system layout
- COW friendly B-trees
- Snapshot
- Software RAID
- Subvolume

# Inodes

- Everything in Linux is a file (Idea borrowed from Unix)
- Metadata (Data structure) associated with a file (Owner, permission, timestamp, actual file location(disk-blocks) and plethora of other information) VFS layer representation: struct inode
- Stored on a disk (Persistent storage)
- BtrFS representation: struct btrfs_inode_item

```
[amit@discworld papers]$ ls -li
total 7252
12853689 drwx------. 3 amit amit    4096 Oct 26  2014 FAST
12722069 -rw-------. 1 amit amit  130652 Oct 26  2014 flash-enterprise.pdf
12722070 -rw-------. 1 amit amit 3421258 Oct 26  2014 jcse_3-3_50.pdf
12722071 -rw-------. 1 amit amit  117576 Oct 26  2014 novos-hotos2011.pdf
12722072 -rw-------. 1 amit amit 1580486 Oct 26  2014 NVM13-Wheeler_Linux_and_NVM.pdf
13369519 drwxr-xr-x. 2 amit amit    4096 Nov  1  2014 OS-papers
12722073 -rw-------. 1 amit amit   95059 Oct 26  2014 storage-topology.pdf
12722074 -rw-------. 1 amit amit  902215 Oct 26  2014 vol4no9main_part22.pdf
12722075 -rw-------. 1 amit amit 1155377 Oct 26  2014 xie.pdf
```

# Extent

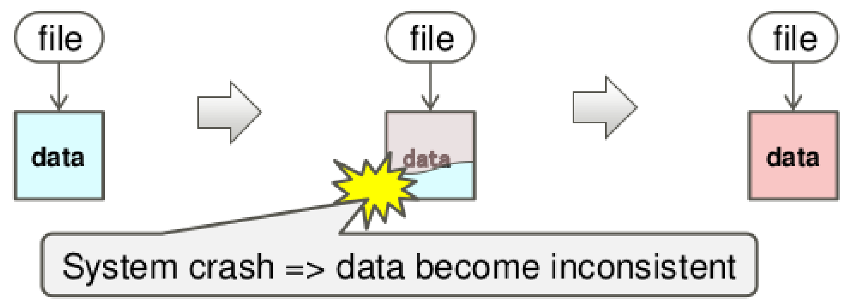
- Extents are physically contiguous area on storage (Say disks).
- File consists of zero or more extents.
- I/O takes place in units of multiple blocks if storage is allocated in consecutive blocks. For sequential I/O, multiple block operations are considerably faster than block-at-a-time operations.
- Easier book-keeping

# COW

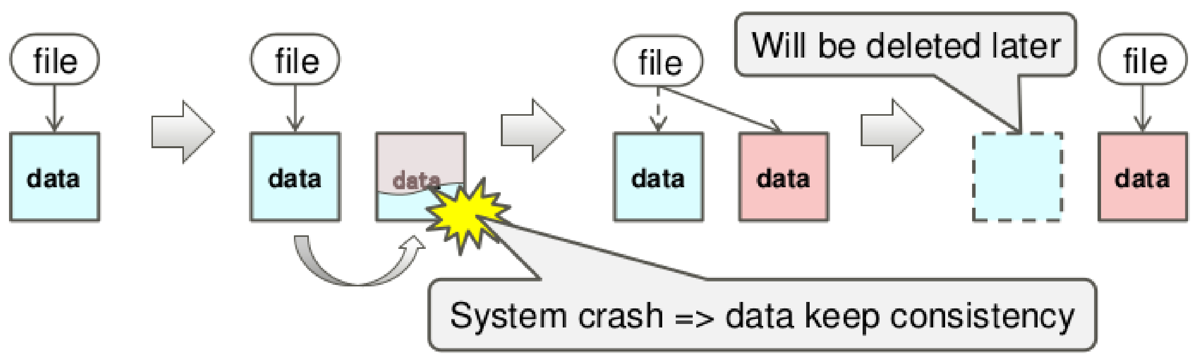
- Copy-On-Write
  - Technique
    - Every consumer is given pointer to the same resource.
    - Modification attempt leads to Trap
    - Create a local copy
    - Modify the local copy
    - Update the original resource
  - Benefit
    - Crash during the update procedure does not impact the original data.

# COW contd...

- Btrfs uses CoW style data/metadata update
  - Safer than overwrite style update by design
- Overwrite style: Update the data in place



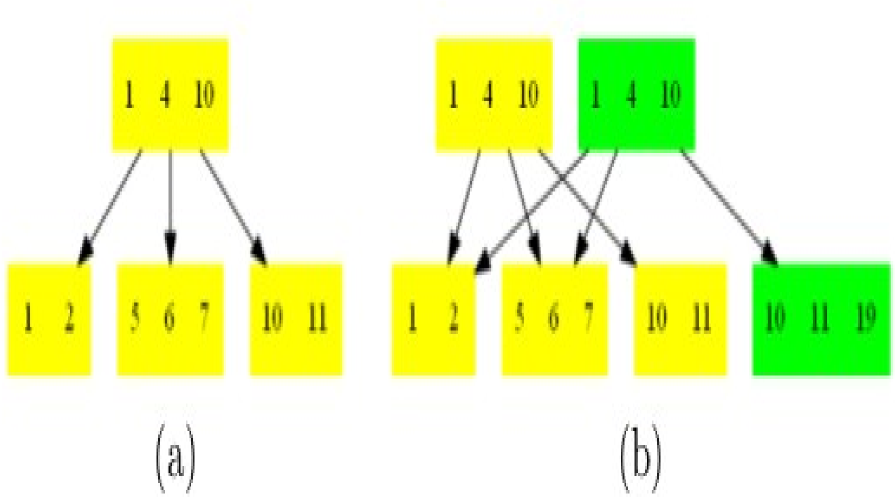

- CoW style: Copy, update, and replace pointer




Source: linux foundation[8]

# COW friendly (B){ayer|oeing|balanced|broad}-tree

- Main idea
  - B+ -tree
  - Top down update procedure
  - Remove Leaf chaining
  - Lazy reference counting (Cloning purposes, Utilizing DAG)




(a) A basic b-tree (b) Inserting key 19, and creating a path of modified pages. [3]

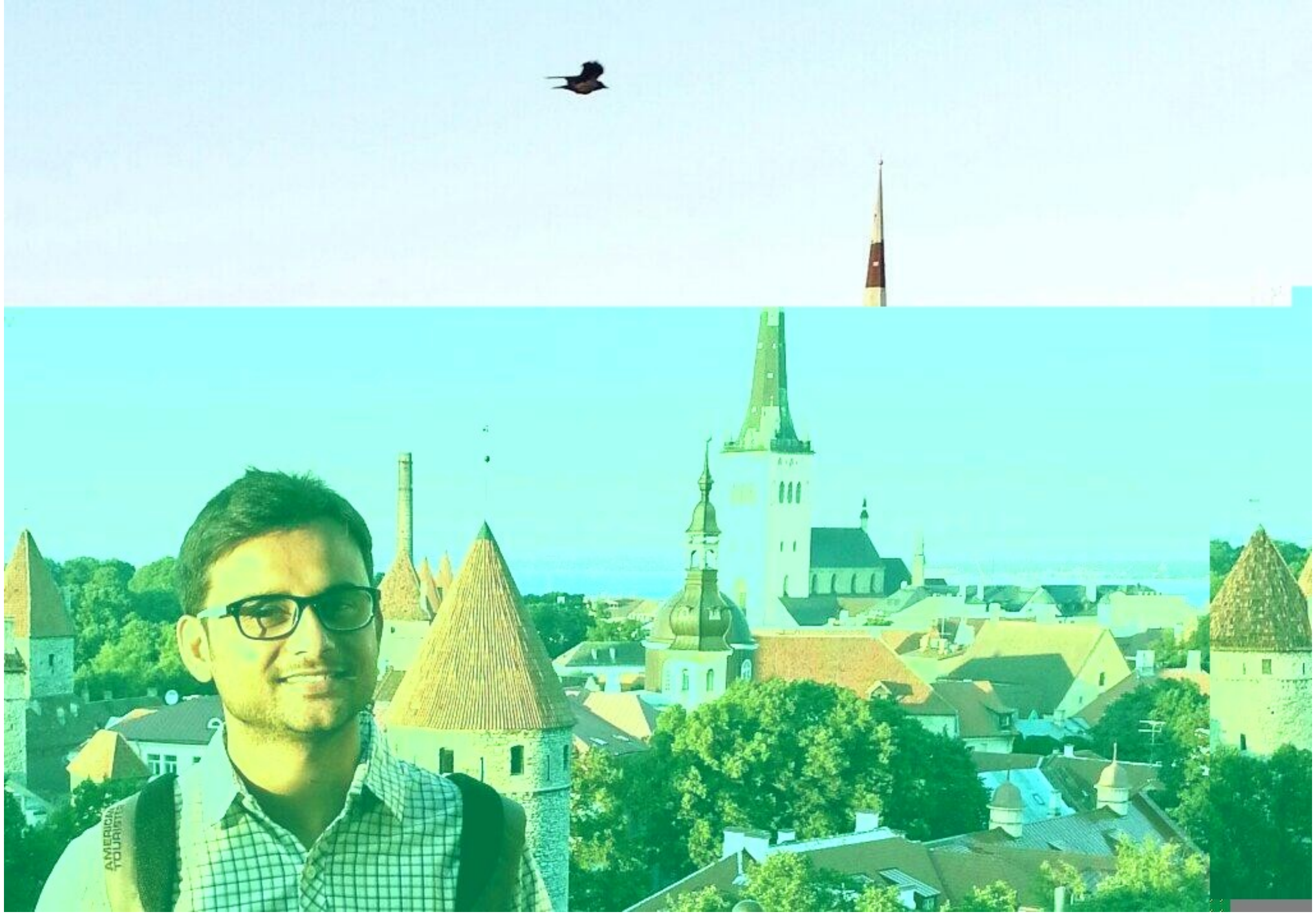
# Data/Metadata checksum

- Btrfs has checksum for each data/metadata extent to detect and repair the broken data.
- Metadata is redundant and checksummed and data is checksummed too.
- When Btrfs reads a broken extent, it detects checksum inconsistency.
  - With mirroring: RAID1/RAID10
    - Read a correct copy
    - Repair a broken extent with a correct copy
  - Without mirroring
    - Dispose a broken extent and return EIO
- With “*btrfs scrub*”, Btrfs traverses all extents and fix incorrect ones
  - Online background job
- Demo

# Data/Metadata checksum

Original

# Data/Metadata checksum




After flipping one bit

# Data/Metadata checksum

- Demo
  - Corrupt disk block by bypassing file system to simulate disk faults.
  - Result

    <snip from kernel logs>

    *BTRFS info (device sdb): csum failed ino 257 off 0 csum 2566472073 expected csum 3681334314*

    *BTRFS info (device sdb): csum failed ino 257 off 0 csum 2566472073 expected csum 3681334314*

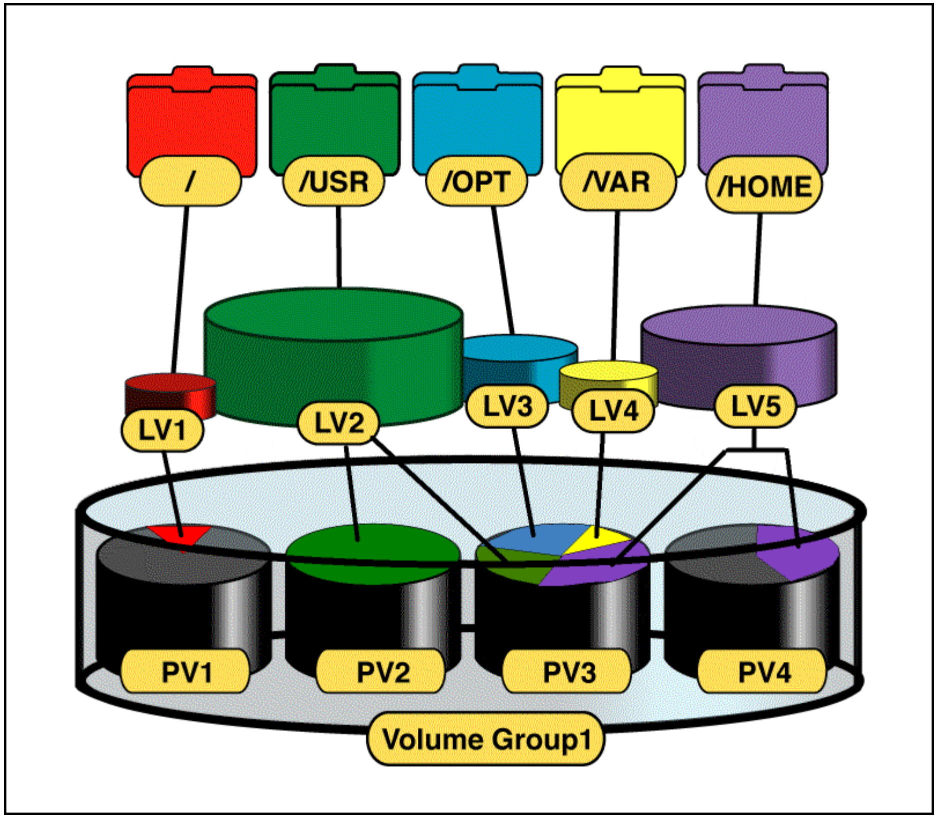
    *BTRFS: read error corrected: ino 257 off 0 (dev /dev/sdc sector 449512)*

    <snip from kernel logs>

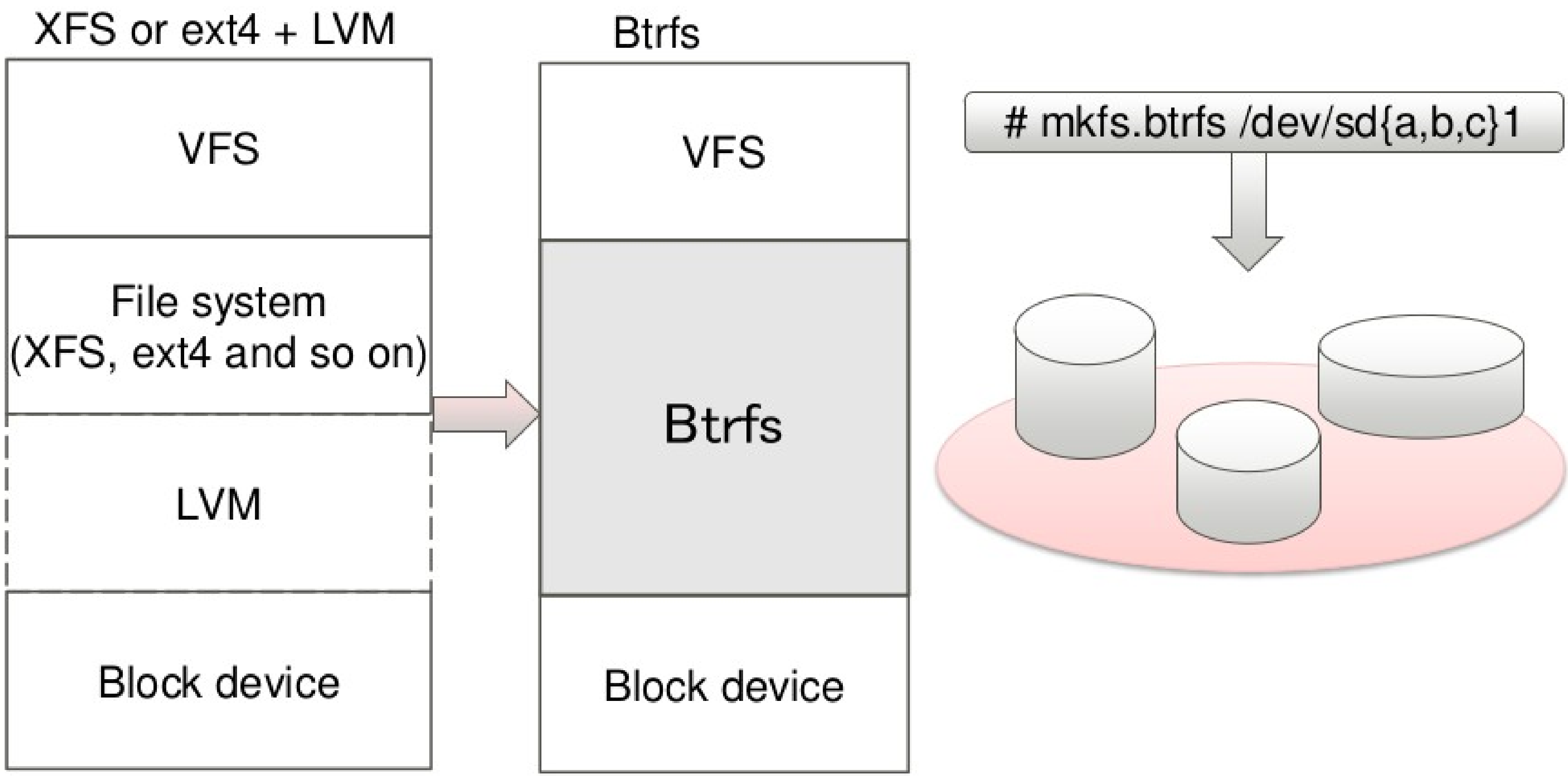
    - BtrFS is able to identify such errors and corrects them too :)

# Software RAID

- Motivation
  - Increased capacity
  - Greater reliability.
  - Software modules that take raw disks, merge them into a virtually contiguous block address space, and export that abstraction to higher level kernel layers.
  - Support mirroring, striping, and RAID5/6.



# Software RAID




Traditional approach

BtrFS approach

Source: Linux foundation [8]

# Software RAID contd...

- Raid 0, 1, 5 (single parity) and 6 (redundant parity) are built in BtrFS
- Metadata is usually stored in duplicate form in Btrfs filesystems, even when a single drive is in use. (Also allows administrator to explicitly specify the raid level for metadata and data separately).
- Dynamically increase the storage
- Example

  *# mkfs.btrfs -d <mode> -m <mode> <dev1> <dev2> …*

  *# mount /dev/<diskX> /<mount-point>*

  - Where diskX $\in$ {dev}

  *# btrfs device add /dev/<diskY> /<mount-point>*

# Scrub

- Btrfs CRCs allow to verify data stored on disk
- CRC errors can be corrected by reading a good copy of the block from another drive
- ONLINE background filesystem scrub (Not a full fsck)
- Scrubbing code scans the allocated data and metadata blocks
- Any CRC errors are fixed during the scan if a second copy exists

# Subvolumes

- Subvolumes allow the creation of multiple filesystems on a <u>single device</u> (or array of devices).
- Each subvolume can be treated as its own filesystem and mounted separately and exposed as needed (No need to mount the “root” device at all)
- Unlike subvolumes in LVM(an independent logical volume, composed of physical volumes), it has hierarchy and relations between subvolumes.
- Example

  *# btrfs subvolume create <user-defined-name>*

# Snapshot

- Works on subvolumes
- Snapshot is a cheap atomic copy of a subvolume, stored on the same file system as the original.
- Snapshots have useful backup function.
- Snapshot of snapshot of snapshot …
- Read-only or a Read-write snapshot

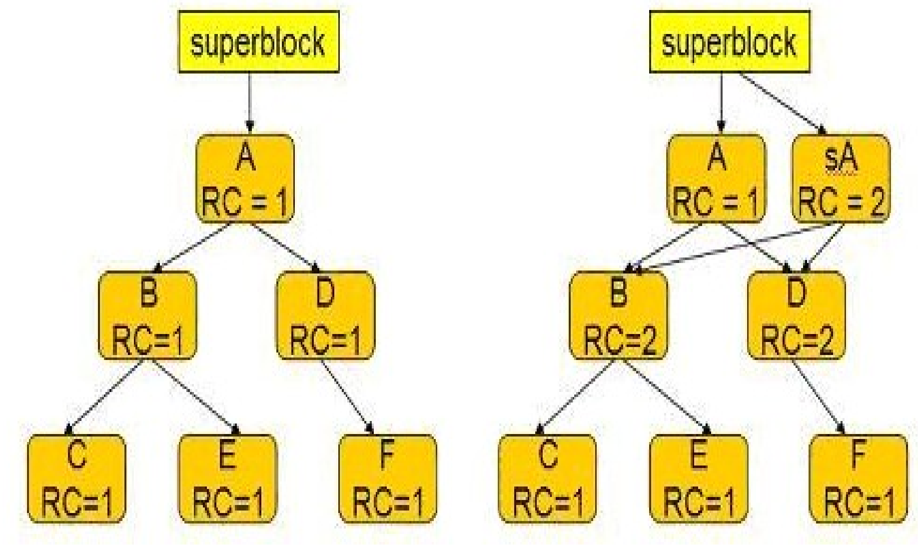



Fig: Snapshot of subvolume[4]

# Robustness

- **Test**
  - Tolerance to Unexpected Power Failure while Writing to Files
- **Environment**
  - Custom board (Freescale TWR-VF65GS10)
  - Memory : 1GB DDR3
  - Storage : 16GB Micro SD Card
  - Software: Linux Kenel 3.15-rc7
  - Application: Power Supply Control Unit Periodically Turns On and Off DC Power Supply every Minute, while a file writing application continuously Creates 4KB Files and Writes to it.
  - Test candidates: Ext4, BtrFS
- **Result**
  - Ext4 was corrupted and needed a file system check (fsck), while BtrFs didn't show any abnormalities.

Source: Fujitsu [5]

# Robustness

| | Number of Power Failure | Results |
|---|---|---|
| Btrfs | 1,000+ | No Abnormal Situation Occurred |
| Ext4 | 1,000+ | **Corrupted inode** had increased up to 32,000 and Finally Fell into Abnormal **Disk Full** State |

Table: BtrFS vs Ext4 power failure test result

Source: Fujitsu [5]

# Performance

- **Test**
  - Basic File I/O Throughput and Throughput under High Load
- **Environment**
  - Intel Desktop Board D510MO
  - Processor : 1.66 GHz Dual-Core Atom (4 Core with HT)
  - Memory : 1GB DDR2-667 PC2-5300
  - Storage : 32GB Intel X25-E e-SATA SSD
  - Software: Linux Kernel 3.15.1 (x86_64)
  - Application: FIO (Single (for Basic) and Multiple (for High Load) FIO Running)
  - Test candidates: Ext4, BtrFS
- **Result**
  - I/O throughput under single FIO
    - Read : in Seq, Btrfs was Slightly Faster than Ext4
    - Write : Ext4 was almost Twice Faster than Btrfs
  - I/O throughput under high load
    - Ext4: Every I/O Throughput Decreased Significantly, BtrFS: Decreased Less than Ext4

Source: Fujitsu [5]

# Performance

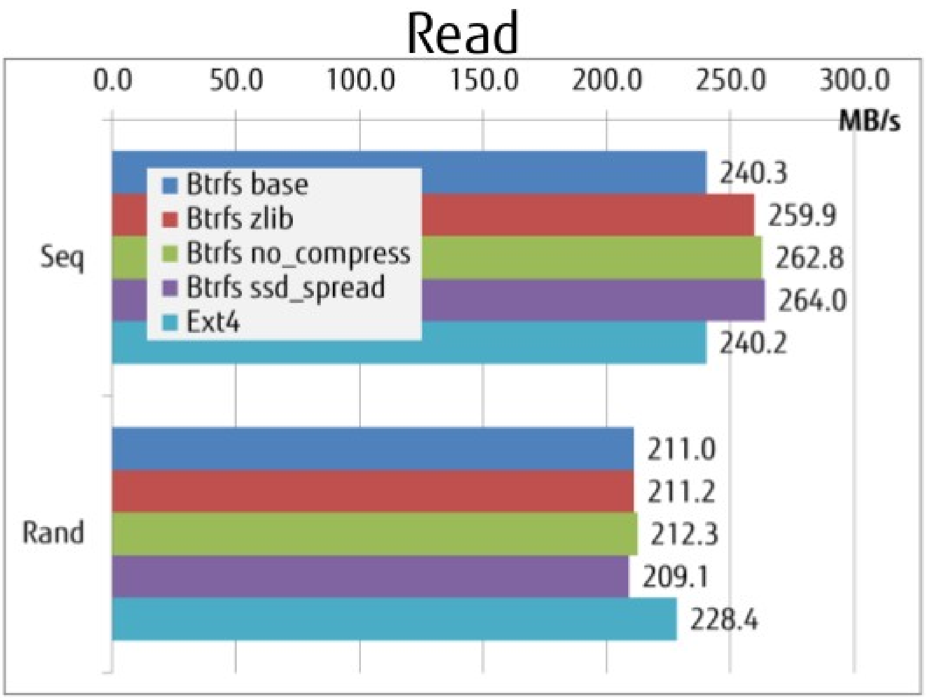



Fig: Read operation with single FIO

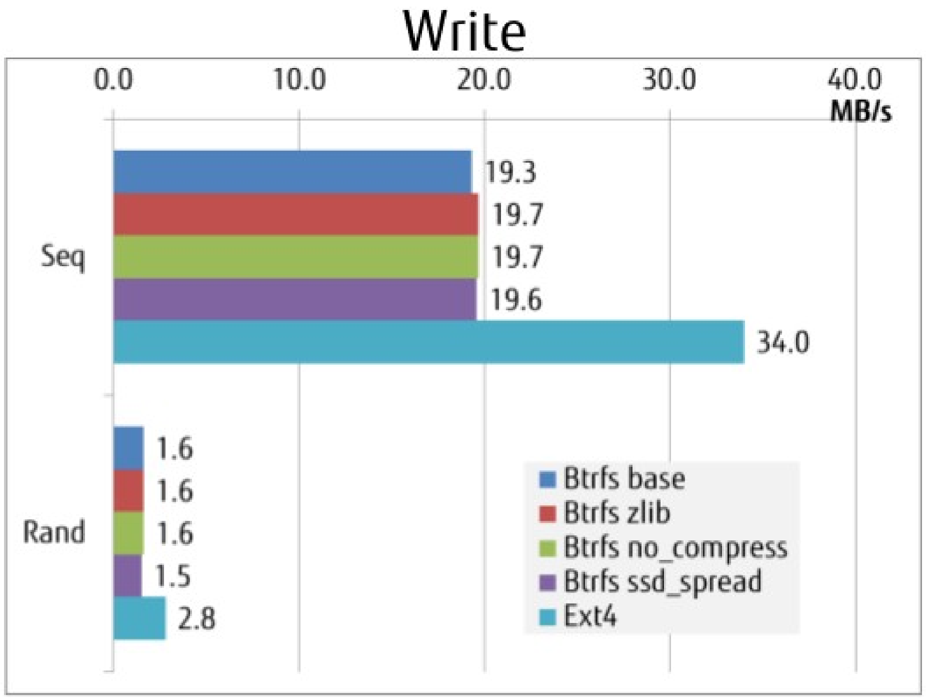



Fig: Write operation with single FIO

Source: Fujitsu [5]

# Who's using BtrFS

And many more...

# Gotchas

- Typically doesn't corrupt itself with latest kernel, but <u>sometimes</u> it does, So Always have backups.
- The Btrfs code base is under heavy development.
- Can get out of balance and require manual re-balancing.
- Auto de-fragmentation has problems with journal and virtual disk image files.
- Raid 5, 6 are still experimental.

# Conclusion

- Storage systems are not perfect, faults are inevitable.
- File systems play a crucial role in storage fault tolerance, data recovery, scalability and performance of the overall system.
- BtrFS[6] is a relatively new “copy-on-write” file system whose main design focus is fault tolerance.
- BtrFS adds an additional layer of storage fault tolerance capability to existing solutions and in some cases outperforms others or make them irrelevant.

# Reference

[1] Ohad Rodeh. 2008. B-trees, shadowing, and clones. Trans. Storage 3, 4, Article 2 (February 2008), 27 pages. DOI=10.1145/1326542.1326544 http://doi.acm.org/10.1145/1326542.1326544

[2] http://www.phoronix.com/scan.php?page=news_item&px=mty0ndk

[3] Ohad Rodeh, Josef Bacik, and Chris Mason. 2013. BTRFS: The Linux B-Tree Filesystem. Trans. Storage 9, 3, Article 9 (August 2013), 32 pages.DOI=10.1145/2501620.2501623 http://doi.acm.org/10.1145/2501620.2501623

[4] http://www.ibm.com/developerworks/cn/linux/l-cn-btrfs/

[5] events.linuxfoundation.jp/sites/events/files/slides/linux_file_system_analysis_for_IVI_systems.pdf

[6] https://btrfs.wiki.kernel.org/

[7] Bianca Schroeder and Garth A. Gibson. 2007. Disk failures in the real world: what does an MTTF of 1,000,000 hours mean to you?. In Proceedings of the 5th USENIX conference on File and Storage Technologies (FAST '07). USENIX Association, Berkeley, CA, USA, Article 1

[8] Satoru Takeuchi. Fujitsu. 2014. Btrfs Current Status and Future Prospects

# Be brave, try BtrFS TODAY