Next Generation File Systems and Data Integrity

Ric Wheeler
rwheeler@redhat.com
November 2008

Talk Overview

- Data Integrity Framework
- Ext4 and XFS Data Integrity Features
- BTRFS Data Integrity Features
- IO Stack Data Integrity Features
- Questions?

Data Integrity Framework

How Many Concurrent Failures Can Your Storage Device Survive?

- Protection against failure is expensive
  - Storage systems performance
  - Utilized capacity
  - Extra costs for hardware, power and cooling for less efficient protection schemes
- Single drive can survive soft failures
  - A single disk is 100% efficient in terms of raw capacity
- RAID5 can survive 1 hard failure & soft failures
  - RAID5 with 5 data disks and 1 parity disk is 83% efficient for capacity
- RAID6 can survive 2 hard failures & soft failures
  - RAID6 with 4 data disks and 2 parity disks is only 66% efficient with capacity!
- Fancy schemes (erasure encoding schemes) can survive many failures
  - Any “k” drives out of “n” are sufficient to recover data
  - Popular in cloud and object storage systems

What is the expected frequency of disk failure?

- How long does it take you to detect a failure?
- Hard failures
  - Total disk failure
  - Read or write failure
  - Usually detected instantaneously
  - Most common focus of existing file systems and storage software
- Soft failures
  - Can happen at any time
  - Usually detection requires scrubbing or scanning the storage
  - Unfortunately, can be discovered during RAID rebuild which implies at least partial data loss
  - Modern S-ATA drives are relatively reliable, but the huge size means that a second failure during rebuild is increasingly likely

How long does it take you to repair a failure?

- Repair the broken storage physically
  - Rewrite a few bad sectors (multiple seconds)?
  - Replace a broken drive and rebuild the RAID group (multiple hours)?
- Can we identify the damage done to the file system?
  - Are all of my files still on the system?
  - Are any files present but damaged?
  - Do I need to run fsck?
- Very useful to be able to map a specific IO error back to a meaningful object
  - A damaged user file needs restored from tape
  - Damaged metadata requires a file system check (fsck)
  - Damaged unallocated space is my favorite type of failure!
- Repairing the logical structure of the file system metadata
  - Fsck time can take hours or days
  - Restore any data lost from backup
  - Can take enormous amounts of DRAM to run for large file systems

Exposure to Permanent Data Loss

- Combination of the factors described:
  - Robustness of storage system
  - Rate of failure of components
  - Time to detect the failure
  - Time to repair the physical media
  - Time to repair the file system metadata (fsck)
  - Time to summarize for the user any permanent loss
- If the time required to detect and repair is larger than your failure rate, you will lose data!

Example: MD RAID5 & EXT3

- RAID5 gives us the ability to survive 1 hard failure
  - Any second soft failure during RAID rebuild can cause data loss since we need to read each sector of all other disks during rebuild
  - Rebuild can begin only when we have a new or spare drive to use for rebuild
- Concurrent hard drive failures in a RAID group are rare
  - ... but detecting latent (soft) errors during rebuild are increasingly common!
- MD has the ability to “check” RAID members on demand
  - Useful to be able to de-prioritize this background scan
  - Should run once every 2 to 4 weeks
- RAID rebuild times are linear with drive size
  - Can run up to 1 day for a healthy set of disk drives
- EXT3 fsck times can run a long time
  - 1TB FS fsck with 45 million files ran 1 hour (reports in the field of run time up to 1 week!)
  - Hard (not impossible) to map bad sectors back to user files using ncheck/icheck

Ext4 Data Integrity Features

EXT4 Integrity Features

- Checksums are computed for the journal transactions
  - Help detect corrupted transactions on replay of the journal after a crash
  - Does not cover corruption of user data blocks
- Extent based allocation structures do support quicker fsck and mkfs times
  - Uninitialized inodes can be skipped
  - Preallocated data blocks can be flagged us unwritten which will let us avoid writing blocks of zero data
  - These features – especially on larger file systems – need thorough testing!
- Better allocation policies lead to faster fsck times
  - Directory related blocks are stored on disk in contiguous allocations
  - High file count fsck times are 6-8 times faster
  - Improved allocation only helps native ext4 file systems, not those upgraded from ext3!

Updated Example: MD RAID5 & EXT4

- *RAID5 gives us the ability to survive 1 hard failure*
  - *Any second soft failure during RAID rebuild can cause data loss since we need to read each sector of all other disks during rebuild*
  - *Rebuild can begin only when we have a new or spare drive to use for rebuild*
- *Concurrent hard drive failures in a RAID group are rare*
  - *... but detecting latent (soft) errors during rebuild are increasingly common!*
- *MD has the ability to "check" RAID members on demand*
  - *Useful to be able to de-prioritize this background scan*
  - *Should run once every 2 to 4 weeks*
- *RAID rebuild times are linear with drive size*
  - *Can run up to 1 day for a healthy set of disk drives*
- EXT4 fsck times are much improved
  - 1TB FS fsck with 45 million files ran under 7 minutes – much faster than the ext3 1 hour time
  - Close to a 10x improvement for new file systems!
  - Credit goes to better contiguously allocated blocks for directories

BTRFS Data Integrity Features

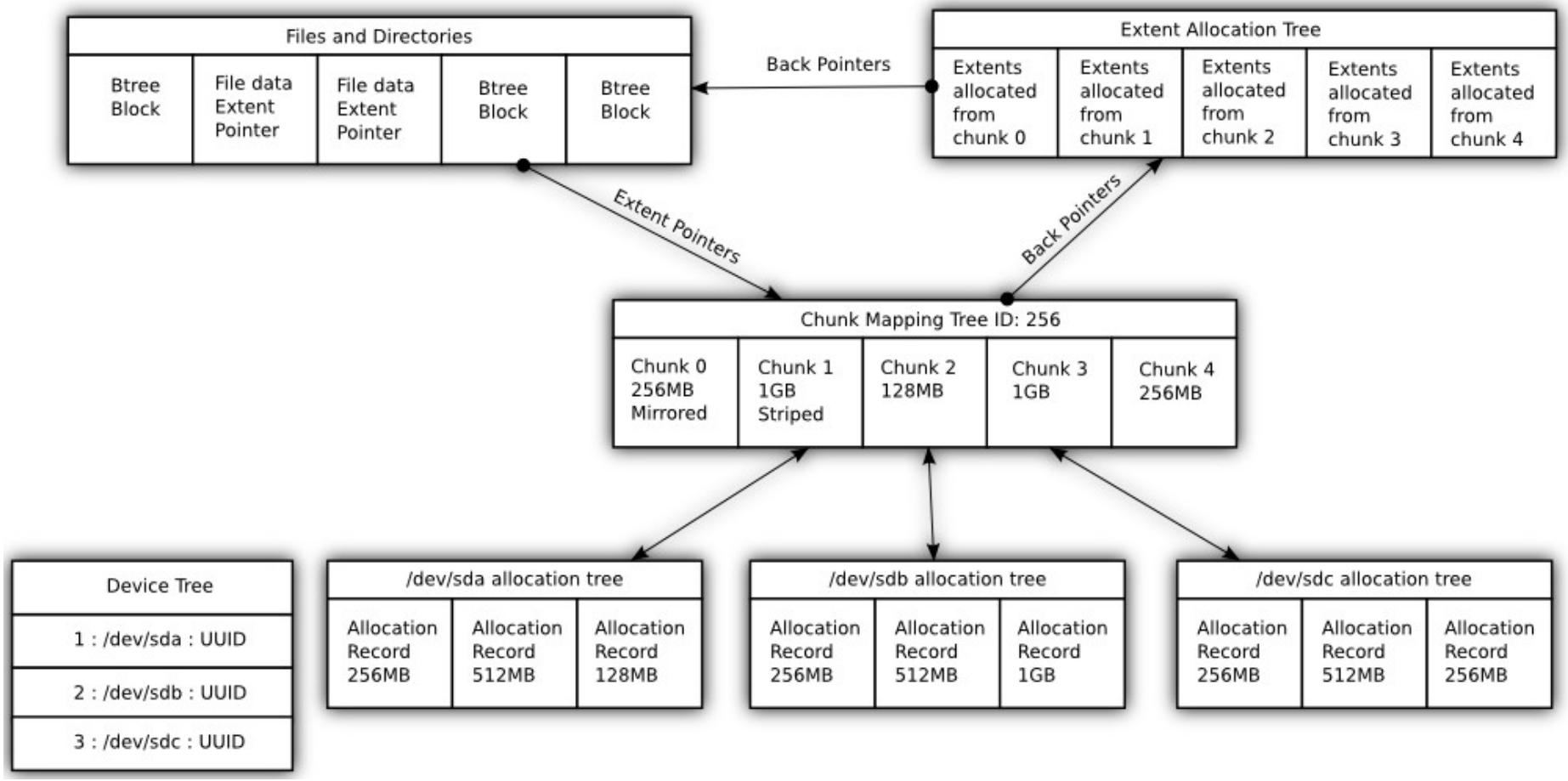
BTRFS Multi-device Support

- Has support for dynamically adding and removing devices from file system
  - Component devices can be simple disks, MD devices or high end arrays
  - Has a special allocation policy for SSD devices
- Existing File System to Block Layer API is limited
  - All IO errors look the same
  - No way to ask RAID1 device to read other mirror
- BTRFS Internally Implements RAID
  - RAID1 and RAID0 now, other RAID levels planned
  - Can use different policies for different types of data
  - Example: RAID1 for metadata, RAID5 for data blocks
- Upstream discussion on whether we can move all RAID code into a library
  - Take MD RAID code
  - Provide routines for device mapper, MD and new file systems to consume
  - Would prevent duplication of code and allow for more sophisticated API's

BTRFS Checksumming & Backpointers

- BTRFS supports checksumming of all data
  - Data blocks
  - Checksums are validated on read
  - With internal mirroring, a block with a bad checksum or IO error can be requested from the other mirror
- Metadata back references
  - Detect many types of corruptions
  - Help limit repair time and scope
  - Connect extents back to the files that reference them
    - Block 112233 is corrupted, which files do I need to fix?
    - Full read of storage can create a list of bad sectors which can be reversed mapped into files or metadata objects
- Disk scrubbing (planned feature)
  - A file read will validate data integrity for a specific file
  - Allows for a relatively easy (if slow!) full file system scrub routine

BTRFS Internal Structure



Copy on Write and Snapshots

- Btrfs is built on Copy on Write Primitives (COW)
  - Allows for innovative use of snapshots
- Snapshots
  - As efficient as directories on disk
  - Every transaction is a new unnamed snapshot
  - Every commit schedules the old snapshot for deletion
  - Snapshots use a reference counted COW algorithm to track blocks
  - Snapshots can be written and snapshotted again
- COW leaves old data in place and writes the new data to a new location
- These techniques provide built in support for snapshot based backup strategies

Data Ordering

- Ext3 data ordering writes all dirty data before each commit
  - Any fsync waits for all dirty data to reach disk
- Ext4 data ordering writes all dirty non-delalloc data before each commit
  - Much better than Ext3
- XFS updates inodes only after the data is written
- Btrfs updates inodes, checksums, and extent pointers after the data is written
  - Allows atomic writes
  - Makes transactions independent of data writeback

Synchronous Operation

- COW transaction subsystem is slow for frequent commits
  - Forces recow of many blocks
  - Forces significant amounts of IO writing out extent allocation metadata
- Simple write ahead log added for synchronous operations on files or directories
- File or directory Items are copied into a dedicated tree
  - One log tree per subvolume
  - File back refs allow us to log file names without the directory
- The log tree
  - Uses the same COW btree code as the rest of the FS
  - Uses the same writeback code as the rest of the FS and uses the metadata raid policy.
- Commits of the log tree are separate from commits in the main transaction code.
  - fsync(file) only writes metadata for that one file
  - fsync(file) does not trigger writeback of any other data blocks

Summary: Btrfs and Data Integrity

- Btrfs gives users easy RAID
  - Can survive and detect any single disk error
  - Can be used selectively to use the most expensive types of protection only for data or metadata
- Error detection window has been closed
  - Checksums are validated on all reads
  - Full disk scrubs can be done at the block or file level (by reading all files)
- Built in support for file system snapshots
  - Allows for sophisticated data reliability schemes
- File system check is really fast today
  - But it does not do as much work and checking as others!
- To do list
  - Implement full disk scrubbers
  - Continue to work on space allocation issues
  - FSCK improvements
  - http://btrfs.wiki.kernel.org/index.php/Project_ideas
  - Merge into mainline planned for 2.6.29!

IO Stack Data Integrity Features

SCSI DIF/DIX Support

- High end arrays and databases have long supported application to platter data integrity check
  - Applications like Oracle compute a CRC or checksum for each block sent down to the array
  - Checksum is validated by target storage device which must store the extra bytes in a larger disk sector, for example in a 520 byte sector
  - Historically, this has been a premium & costly feature
- SCSI DIF support in arrays brings this potentially into more types of arrays in a standard way
  - Adds an extra 8 bytes per 512 byte sector to store 3 tag fields
  - Tags can be set by array, HBA, SCSI, file system or application
  - T13 (ATA) committee also considering a similar proposal
  - HBA support for DIF can hide this new feature from the IO/FS stack
  - Sophisticated applications (like Oracle) can use it directly to get application to platter data intregrity
  - Optionally, file system could use one tag field
- Development lead by Martin Petersen at Oracle
  - http://oss.oracle.com/projects/data-integrity/

Questions?

- Project Wiki Pages:
  - http://btrfs.wiki.kernel.org
  - http://ext4.wiki.kernel.org
  - http://oss.oracle.com/projects/data-integrity
  - http://oss.sgi.com/projects/xfs
  -
- USENIX FAST Papers on Storage & File System Failures
  - http://www.usenix.org/event/fast07/tech/schroeder.html
  - http://www.usenix.org/event/fast07/tech/pinheiro.html
  - http://www.usenix.org/event/fast08/jiang.html
  - http:www.usenix.org/event/fast08/tech/gunawi.html
  -
- Contact Information
  - Ric Wheeler
  - rwheeler@redhat.com