FS Design Around SMR

Seagate's Journey with EXT4

Adrian Palmer - Seagate

SNIA Legal Notice

- The material contained in this tutorial is copyrighted by the SNIA unless otherwise noted.
- Member companies and individual members may use this material in presentations and literature under the following conditions:
  - Any slide or slides used must be reproduced in their entirety without modification
  - The SNIA must be acknowledged as the source of any material used in the body of any document containing material from these presentations.
- This presentation is a project of the SNIA Education Committee.
- Neither the author nor the presenter is an attorney and nothing in this presentation is intended to be, or should be construed as legal advice or an opinion of counsel. If you need legal advice or a legal opinion please contact your attorney.
- The information presented herein represents the author's personal opinion and current understanding of the relevant issues involved. The author, the presenter, and the SNIA do not assume any responsibility or liability for damages arising out of any reliance on or use of this information.

  NO WARRANTIES, EXPRESS OR IMPLIED. USE AT YOUR OWN RISK.

Abstract

SMR Friendly File System

- SMR is a fundamental drive technology, embraced by drive vendors. SMR changes fundamental assumptions of file system management. This long-held notion of Random-Writes now resembles the write profile of sequential-access tape.
- Seagate is leading the way in providing a standards compliant IO stack for use with the new drives. Using the new ZAC/ZBC commands to make and maintain a file system is essential for performant operation. Seagate is sharing lessons learned from modifying EXT4 for use with SMR. This effort is called the SMR Friendly File System (SMRFFS).

Shingled Magnetic Recording

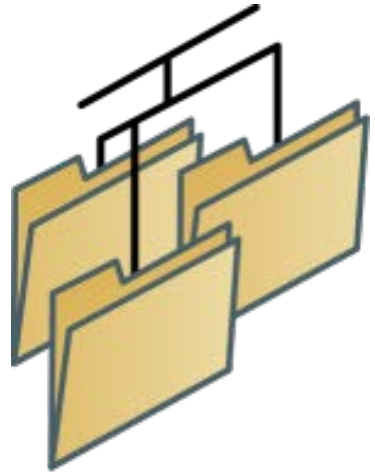
- Tutorial on SMR: SDC 2014
  - 1 new condition: Forward-Write preferred
  - http://www.snia.org/sites/default/files/Dunn-Feldman_SNIA_Tutorial_Shingled_Magnetic_Recording-r7_Final_0.pdf
- ZAC/ZBD spec: t10/13
  - Zones
  - http://www.t10.org/ - SCSI ZBC Standards
  - http://www.t13.org/ - ATA ZAC Standards
- Github reference
  - https://github.com/Seagate/SMR_FS-EXT4
- Assumptions
  - Familiarity with FS operation
  - Familiarity with SMR standards

Check out SNIA Tutorial:

Shingled Magnetic Recording Models, Standardization, and Applications

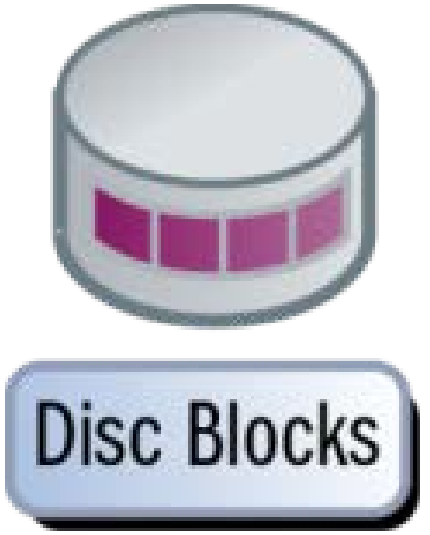
What is an FileSystem?

- An essential piece of software on a system
- Organizes and stores structured data on unstructured media
- Allocates and Manages Storage Space
- Stores metadata AND DATA (Differs from Database)
- Provides maintenance and usability functions



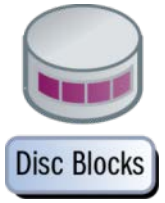
Basic FS requirements

- Write-in-place
  - Superblock – known location on disk
    - Mount operation cannot find dynamic location without a static location
- Sequential write
  - Journal – circular log structured buffer
- Unrestricted write type (Random or Sequential)
  - File records
  - Block maps
  - Indexes/Queries
  - Data



Drive parameters

Sector



- Atomic unit of read/write access
- Typically 512B size
- Independently Addressed
- Standard commands
  - Read
  - Write
- No states

Zone

- Atomic performant rewrite unit
- Typically 256 MiB size
- Indirectly addressed via sectors
- Modified with ZAC/ZBD cmds
  - ResetWritePointer
  - ReportZones
- Has state
  - WritePointer, Condition, Size, Type

Write Profiles

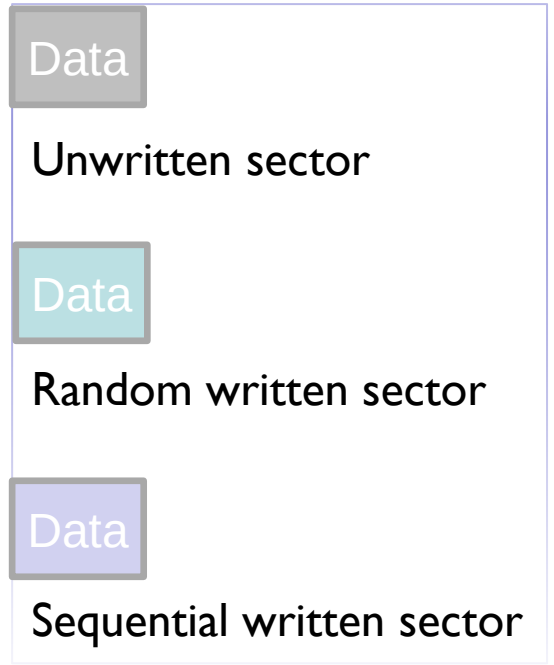
- Conventional – Random Access

- Tape – Sequential access

- Flash – Sequential access, erase blocks

- SMR HA/HM – Sequential access, zones

- SMR write profile is similar to Tape and Flash!



Allocation Containers

- Drive capacities are increasing – location mapping is EXPENSIVE
  - 1.56% loss (512B Blocks) or 0.2% loss (4k Blocks) *int64-->LBA*
- Remap the Block device as a … Block device
  - Partitions -> w*sector size
  - Block size -> x*sector size
  - Group size -> y*Block size
  - FS -> z*group size (expressed as blocks)
- Zones are a good fit to be matched with Groups!
  - Absorb, and mirror the zone metadata in the group!
  - *Do NOT continually query drive for metadata!

Solving the sequential-write problem
1) Separate the problem space (with zones)

- Dedicate zones to each problem subspace
  - User Data
  - File Records
  - Indexes
  - Superblock
  - Trees
  - Journal
  - Allocation Containers

Solving the sequential-write problem
1) Separate the problem space (with zones)

- GPT/Superblocks
  - Updated infrequently, and at dismount
  - First (and last) zones dedicated
  - Copy-on-Update with zone at WritePointer
  - Look at First blocks (known location) and Writepointer (updated information) on mount
  - Organized wipe and update algorithm

  - *Note: FS AND utilities have to be updated with HM
  - *Note: First and last zone of partition MAY be conventional, but not guaranteed.

Solving the sequential-write problem
1) Separate the problem space (with zones)

- Journal/soft updates
  - Updated very frequently
  - 2 (or more) zones, set up as a circular buffer
  - Checkpoint at each new zone
  - Wipe and Overwrite oldest zone
  - Can be used as non-volatile cache for metadata
  - Trade: Requires lots of storage space for efficient memory usage and non-volitility

Solving the sequential-write problem
1) Separate the problem space (with zones)

- Group Descriptors
  - Infrequently change
  - Changes on zone condition change
  - Changes on resize
  - Changes on free block counts
  - Write cached, but written at WritePointer

  - Because of updates, organize as B+Tree, and NOT as an indexed array.
  - The B+Tree will need to be stored on-disk

Solving the sequential-write problem
1) Separate the problem space (with zones)

- File Records
  - POSIX information – Updated very frequently
    - ctime, mtime, atime, msize (to name a few)
    - FS specific attributes
  - Can be on disk as a table, but organize with a B+Tree and add defragmentation
  - Allow records to be modified in memory, and written to journal cache
  - Gather records from journal, write to new blocks at WritePointer
    - Memory caching avoids walking tree problem

Solving the sequential-write problem
1) Separate the problem space (with zones)

- Mapping (File Records to Blocks)
  - A file is written (ideally) as a single chunk – single pointer needed
  - Through a file's life, it becomes fragmented – multiple pointers needed
  - Can outgrow file record space – will need its own B+tree
  - List can be in memory, and in the journal, but needs written out to disk at WritePointer

Solving the sequential-write problem
1) Separate the problem space (with zones)

- Data
  - Copy-on-write
  - Allocator chooses blocks at WritePointer
  - Writes are broken at zone boundary, creating new command and new mapping fragment

Solving the sequential-write problem
2) Cleanup

- We can no longer clean up as we go – it is now a separate step.
- Each zone will have holes created by normal FS operation.

- Garbage Collection
  - Journal GC
    - As each journal entry is consumed and written to disk, holes will form
    - Holes form non-sequentially, but the oldest “type” will be handled first
    - Pointers should be implemented in the journal
    - Trigger should be time based, and load based

Solving the sequential-write problem
2) Cleanup

- Garbage Collection
  - Zones GC
    - Zones less than half full can be combined, if they are the same “type” (except SB zones)
    - Trigger can be time based, capacity based, or event based
    - Requires compaction/defragmentation
    - Resets empty zones
  - Zone Compaction
    - Zones are re-written into a scratchpad zone of equal or greater size
    - File records are scanned – requires a reverse B+Tree for File Record -> Group
    - Metadata B+Trees are scanned – requires reverse B+Trees for metadata -> Group

Solving the sequential-write problem
2) Cleanup

- Garbage Collection
  - Defragmentation
    - Requires scanning of File Records to determine fragments and locations
    - Requires efficient compaction scans
    - Can be non-kernel code

Solving the sequential-write problem
3) Advanced Features

- Indexes
  - Implemented as needed, but either committed to disk as a whole, or broken with a tree.
  - Hash tables, and non-static linked lists are not compatible with SMR
  - Suggested POSIX path index or cache

- Queries
  - Indexes are good to avoid disk seeks
  - Drive needs cleaned to be efficient at reads

Solving the sequential-write problem
3) Advanced Features

- Extended Attributes
  - Can be in file record, but may be better in separate tree
  - Depending on size and organization, may share blocks
- Snapshots
  - Addition of trees and hard links can accommodate snapshots
  - New zones can be allocated for snapshot changes

Solving the sequential-write problem
3) Advanced Features

- Checksums/Parity
  - Must be calculated and written afterwards – no intermediate parity.
  - Because parity would be CoW also, intermediate writes are not protected
  - Intermediate writes can be achieved with use of additional disk space.
  - This approach creates either
    - Holes in the file
    - A separate file record or attribute requiring a tree
    - A opportunity for more aggressive compaction
- RAID/JBOD
  - Benefits from multi-size groups (for non-identical zones)
  - Parity needs a new standard

Questions

- Thanks for attending!
- Learning goals:
  - Forward-write only considerations for the block allocation schemes
  - Zones/BlockGroup/AllocationGroup alignment and use
  - Superblock, and other required write-in-place management schemes

Attribution & Feedback

The SNIA Education Committee thanks the following Individuals for their contributions to this Tutorial.

Authorship History

Name/Date of Original Author here:
Adrian Palmer – July 2015

Updates:

Additional Contributors

Please send any questions or comments regarding this SNIA Tutorial to ***tracktutorials@snia.org***

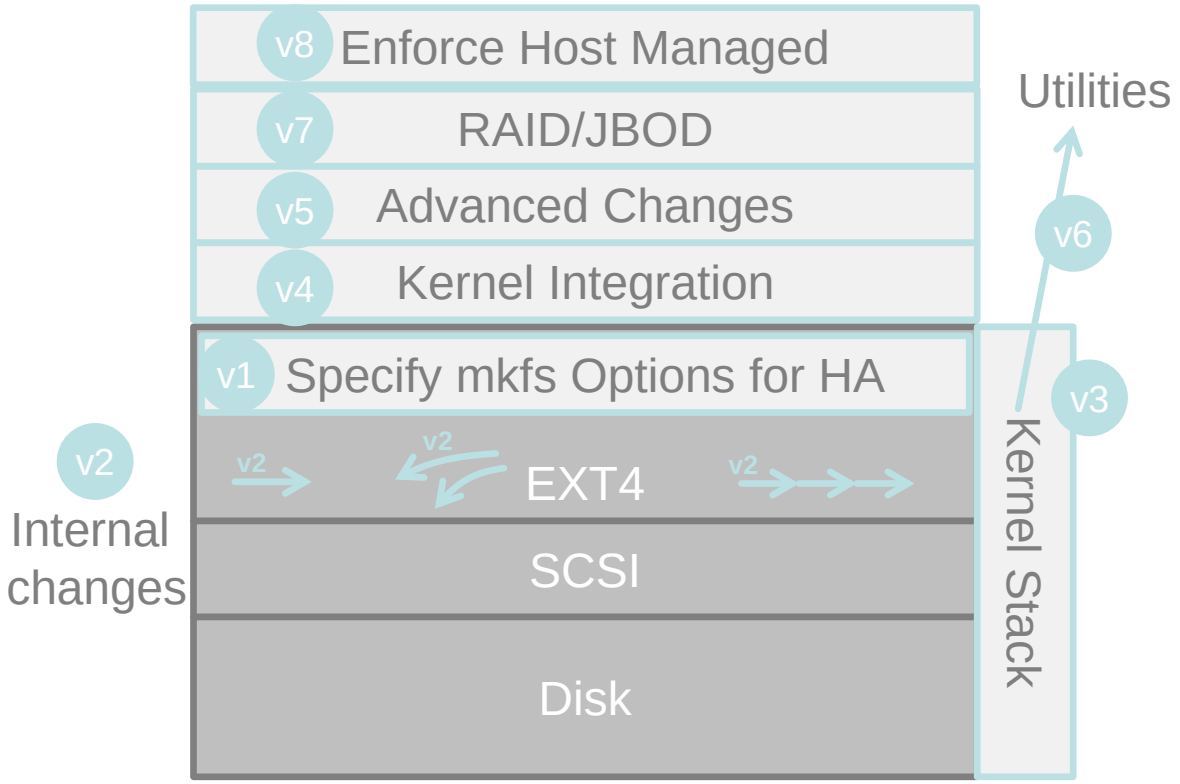
Appendix – EXT4 Group Purpose

- Metagroup: sets of 32 zones
  - Metagroup# = int(Zone# / 32)
  - Assignment of zone usage
    - 0: see table to right
    - 1-30: data
    - 31: inode

- First zone of Metagroup usage by Metagroup#
  - 0 – GPT/Superblock/Group Btree
  - 1 – Group Descriptors/Bitmaps/Inode Btree
  - 2 -18 – Journal
  - 19 – Extent lists/Extent Btrees
  - 20 – Swap/Scratchpad
  - 21 – Xattrs

  …repeat as needed

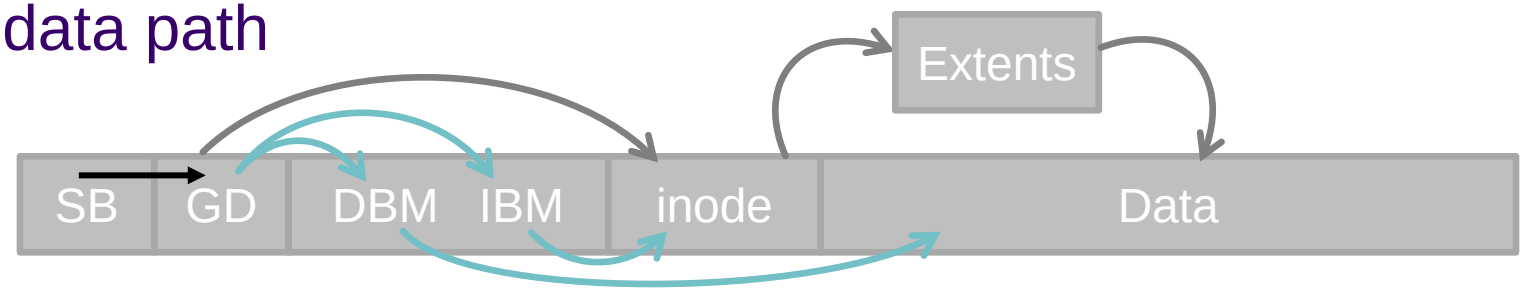
Appendix – EXT4 SMRFFS plan



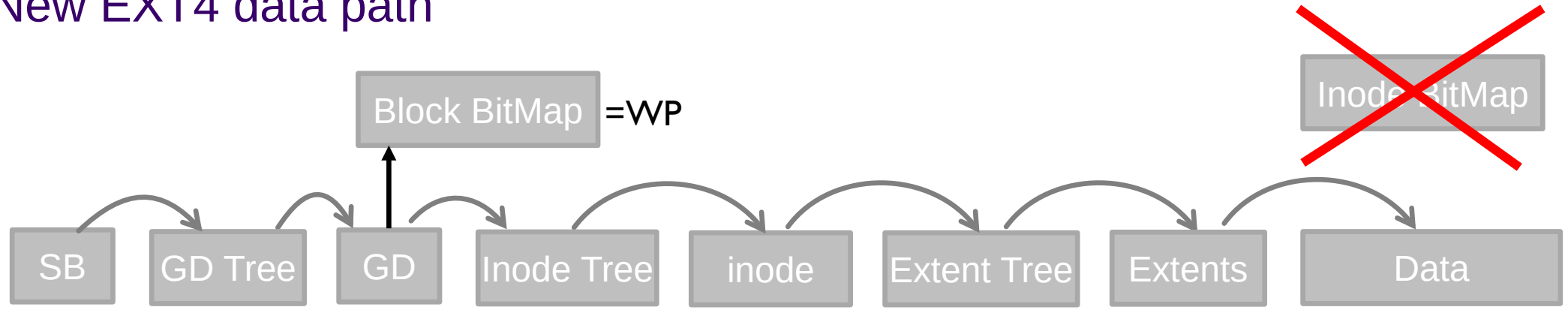
1. cmd line arguments
2. internal handling changes
3. kernel stack changes
4. IOCTL integrations
5. Algorithm enhancements
6. Utility updates
7. Multi-disk Enhancements
8. Host Managed Compliance

Appendix – EXT4 & SMRFFS Data paths

- Old EXT4 data path

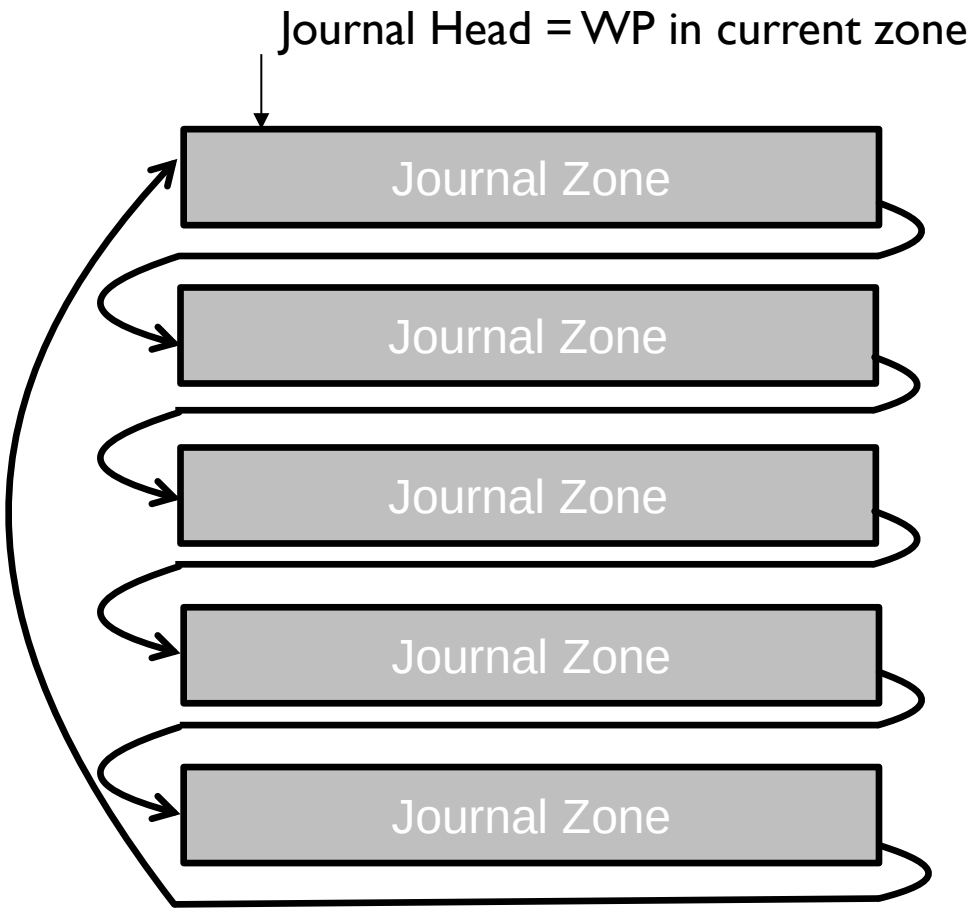


- New EXT4 data path



Appendix – SMRFFS Journal

- Journal is circular buffer
- Journal cannot be a regular file
- ResetWritePointer is issued at each zone change



Appendix – Zone combination

- JBOD
  - New zone = old zone
  - Block numbers/group numbers of all but first partition is shifted
  - Multi-sized groups

- Example
  - Disk 1 = 8 TiB (32768 zones @ 256-MiB) HDD
  - Disk 2 = 1 TiB (16777216 zones @ 32-KiB) Flash

  - LVM Volume = 9 TiB (16809984 zones, first 32768 @ 256 MiB, last 1677216 @ 32 KiB)

Appendix – Zone combination

- RAID 0 - Interleaving
  - New zone size = X x old zone
  - Write in stripes at WP in each zone
- Example
  - Disk 1 = 8 TiB (32768 zones @ 256 MiB)
  - Disk 2 = 8 TiB (32768 zones @ 256 MiB)
  - DM Volume = 16 TiB (32768 zones @ 512 MiB)

Appendix – Zone combination

- RAID 1, 10 - Overlay
  - New zone amount = 1/2 x old zone amount
  - Write in stripes at WP in each zone simultaneously
- Example
  - Disk 1 = 8 TiB (32768 zones @ 256 MiB)
  - Disk 2 = 8 TiB (32768 zones @ 256 MiB)
  - DM Volume = 8 TiB (32768 zones @ 256 MiB)

Appendix – Zone combination

- RAID 5, 50 - Parity
  - Uses interleaving and overlay of zones
  - Requires a minimum write size (X x stripe size)
  - Removes zones from the final report
- Example
  - Disk 1 = 8 TiB (32768 zones @ 256 MiB)
  - Disk 2 = 8 TiB (32768 zones @ 256 MiB)
  - Disk 3 = 8 TiB (32768 zones @ 256 MiB)
  - DM Volume = 16 TiB (32768 zones @ 512 MiB)