An SMR-aware Append-only File System

Chi-Young Ku
Stephen P. Morgan
Futurewei Technologies, Inc.
Huawei R&D USA
Future disk drives will be based on **shingled** magnetic recording.

- Conventional
  - track
  - gap
  - track
  - track
  - track

- Shingled
  - track
  - track
  - track
  - track
  - track

Higher recording density
But no random writes
SMR Technology (2)

- Drive divided into large “zones”.
  - Typically 256 Mbytes each.
- Per-zone **write pointer** for next write loc’n.
  - Write pointer advances as data is written.
  - Can **reset write pointer** on per-zone basis.
- Zone may be empty, full, or partially full.
  - Unwritten area filled with initialization pattern.
SMR Technology (3)

- Three kinds of SMR drives:
  - Drive managed
    - Has STL layer that accepts random I/Os.
    - Existing software runs correctly, poor performance.
  - Host managed
    - Writes must be performed at write pointer.
    - Requires new software to be written.
  - Host aware
    - Has STL layer that accepts random I/Os, but:
      - “Prefers” writes performed at write pointer.
      - Existing software may be tweaked to run better.
SMR Translation Layer (STL)

- Part of drive is reserved to buffer random I/Os.
- The data in this area is eventually moved to its home location after a read-modify-write cycle.
- Operation is performed in the background
  - When possible.
- Disk space could be replaced by flash memory
  - At a significant cost, but higher performance.
Common File Systems on SMR Drives

- Due to Dr. Hannes Reinecke (SUSE Labs)
  - btrfs “is nearly there”.
    - Writes sequentially due to its CoW nature.
    - Very few fixed data locations.
  - xfs “might be an option”
    - Roughly same zone usage as btrfs.
    - Hardly any sequential writes.
    - Report by Dave Chinner for adoption for SMR drives.
Changes to ext4 for SMR (SMRFFS)

- See [https://github.com/Seagate/SMR_FS-EXT4](https://github.com/Seagate/SMR_FS-EXT4)
- Optimizes sequential file layout
  - In-order writes and idle-time garbage collection
- Block groups laid out to match zone alignments
- Allocator changed to follow forward-write rqmts
- New extent layout
- Many more changes throughout stack
Append-only Applications

- Scientific sensor data.
- Financial time series data.
- Temporal business data.
- Surveillance data.
- Web logs.
- RocksDB / LevelDB (LSM-tree).
Circular Append-only Applications

- Probability of access to data in most append-only applications decreases with the age of data.
- Depending on the requirements, old data could be purged or migrated to cool storage.
- In both cases, it would be advantageous to design such applications to circularly append data.
Log-structured File Systems

- File system data and metadata are written to a large circular buffer called a log.
- Reads are satisfied from a large memory cache.
  - Unrealistic in practice.
- Disk seeks are minimized for writes, not reads.
- Garbage collection becomes frequent as file system fills up.
- Seemingly good match for SMR drives.
  - No update in place.
SMR-aware Append-only FS Overview

- Combination of a log-structured file system and a conventional file system.
  - Log is a (large) list of zones.
  - File comprises a zone or a list of zones.
    - Design also supports multiple files per zone.
  - Data initially written to log, then migrated to file.
    - Happens during log compaction, instead of LFS’s generational garbage collection.
SMR-aware Append-only FS Overview (2)

- Some FS data structures are rewritten in place
  - E.g., inodes, allocation maps
    - Host-aware drives support a small number of random I/O zones (e.g., 16)
- Log and files (frequently updated) written in order within zones, from start to finish.
- Log compaction “eats” a zone at a time
SAFS Layout

Inode m
Inode m+1
Inode m+2

Zone n
Zone o

Log head
Zone p
Zone q
Zone r

Log tail

Zone s
Segment Structure

SHB  Data  Log  Data  Log  Data  Log

Head  Segment  Segment  Segment  Segment  Segment  Segment

Zone

Tail  Zone

Zone
Compaction

- Inode Array
  - File 0
  - File 1
  - File 2

- Data
- Data
- Data

- SHB
- Data
- Log

- Log
- Data
- Log

- Data
- Log

- Copy

- Head
- Segment
- Zone

- Tail
- Zone

- Zone

- Zone

- Zone
SAFS Implementation

- Implemented with CSIM 20 simulator on Linux
- Coupled with Seagate 5TByte HA SMR drive
- Measured Performance of append-only applic’ns
- 256 zones in file system, 16 zones random RW,
  16 Gbytes of DRAM, x86-64 system
Disclaimer

- Not a production file system
- Purposes:
  - Explore potential of HA SMR drives
  - Explore combination of LFS and conventional file systems
  - Explore append-only file systems
CSIM 20 Discrete Event Simulator

- Use CSIM event to simulate semaphores
- Use CSIM ports to simulate IPC
- Use CSIM virtual time to account for SMR disk processing time
- Use CSIM processes to simulate POSIX threads
- SMR disk I/O performed via HA SMR drive
SAFS Simulator Components

- Workload simulation module
- File system commands simulation module
- Buffer cache simulation module
- Segment system simulation module
- Journaling system simulation module
- Lock manager simulation module
- SMR disk simulation module
Measured SAFS Applications (1)

- Creates four files
- Appends to all files (one block at a time to each file) until the system is ½ full
- Reads each file (one block at a time from each file) to the end
- Deletes all four files
Performance (1)

- File system size was 64 GBytes
- Total amount of data read/written was 64 GBytes
- Total time was 458 seconds
- Average processing rate was 143.1 MBytes/sec
Performance Comparison (1)

- Ran same steps on other file systems using a 4TByte conventional drive:

<table>
<thead>
<tr>
<th>File System</th>
<th>Time</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAFS*</td>
<td>458 sec</td>
<td>143.1 MB/sec</td>
</tr>
<tr>
<td>F2FS</td>
<td>504 sec</td>
<td>130.0 MB/sec</td>
</tr>
<tr>
<td>NILFS2</td>
<td>510 sec</td>
<td>128.5 MB/sec</td>
</tr>
<tr>
<td>EXT4</td>
<td>571 sec</td>
<td>114.8 MB/sec</td>
</tr>
</tbody>
</table>

- * Simulated, on a 5TByte, HA SMR drive.
Measured SAFS Applications (2)

- Creates four files
- Appends to all files (one block at a time to each file) until the system is ¾ full
- Deletes a file, re-creates and appends to it until the system is ¾ full again (for all four files)
- Deletes all four files
Performance (2)

- File system size was 64 GBytes
- Total amount of data written was 96 GBytes
- Total time was 698 seconds
- Average ingestion rate was 140.8 MBytes/sec
Ran same steps on other file systems using a 4TByte conventional drive:

<table>
<thead>
<tr>
<th>File System</th>
<th>Time</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAFS*</td>
<td>698 sec</td>
<td>140.8 MB/sec</td>
</tr>
<tr>
<td>NILFS2</td>
<td>742 sec</td>
<td>132.5 MB/sec</td>
</tr>
<tr>
<td>EXT4</td>
<td>988 sec</td>
<td>99.4 MB/sec</td>
</tr>
<tr>
<td>F2FS</td>
<td>DNF</td>
<td>N/A</td>
</tr>
</tbody>
</table>

* Simulated, on a 5TByte, HA SMR drive.
Conclusion

- Simulated SAFS on HA SMR drive performs better than modern production LFS and production conventional file system on conventional disk under append-only workload.
Questions?