Evolving Ext4 for Shingled Disks

Abutalib Aghayev
Theodore Ts’o (Google), Garth Gibson, Peter Desnoyers (Northeastern University)

PARALLEL DATA LABORATORY
Carnegie Mellon University
Shingled Magnetic Recording (SMR) increases disk capacity with a small change to current technology

- Shingled Magnetic Recording (SMR): high capacity at low cost
- Complements advanced techniques, not going away any time soon
- Two kinds of SMR disks: drive-managed and host-managed
- Millions of drive-managed SMR disks have shipped
- For this talk: SMR disk ≡ drive-managed, CMR disk ≡ conventional
- SMR disks use Shingle Translation Layer (STL) like FTL in SSDs
- STL exposes a block interface on top of device constraints
Capacity increase comes with a cost: On SMR disks, continuous random writes lead to ultra-low throughput.
Our contribution: A small change to ext4 that results in a big impact on SMR and CMR disks

- Eliminate major source of random writes: metadata writeback
- Ext4 + journaling code ~ 50,000 LOC
- Small change: ~40 LOC modified, ~600 LOC added in new files
- On metadata-light workload (<1%): 1.7-5.4x improvement on SMR disks
- On metadata-heavy workload: 2-13x improvement on SMR and CMR disks
- Fix ext4 bottleneck resulting on 40x improvement on certain workloads
• Introduction
• Background
• Mechanism
• Evaluation on CMR
• Evaluation on SMR
• Conclusion
On SMR, naive handling of random writes results in expensive RMW operation for each write

- An SMR disk consists of sequence of bands
- A band must be written sequentially, like erase block in NAND flash
- Sectors are mapped to bands statically or dynamically
- Random write of a sector requires RMW of a band
- RMW becomes RWW with static mapping
- 4 KiB random write → 90 MiB w/ static mapping, 60 MiB w/ dynamic mapping
STLs in SMR disks use persistent cache to absorb random writes, deferring their cost until the cache fills

- SMR disks have a **large persistent cache invisible to host**
- STL can detect sequential writes, **bypass persistent cache**
- Random writes go to persistent cache, **dirtying bands**
- Dirty bands **cleaned** in idle times (or in the background)
With a full persistent cache, the range of random writes determines the disk throughput.

- STL needs 3 RMW operations to free space for the next 3 random writes.
- Assume RMW is 1 second → Throughput is 1 blocks/sec.

- Assume RMW is 1 second → Throughput is 3 blocks/sec.
- Range of random writes → Dirty bands → Cleaning load → Throughput.
Agenda

• Introduction
• Background
• Mechanism
• Evaluation on CMR
• Evaluation on SMR
• Conclusion
On ext4, metadata writes dirty bands in addition to those required by the user workload

- Partition made up of `flex_bgs`: 16MiB metadata, the rest is data
- A write to data region results in small writes to metadata region
- 4,000 `flex_bgs` on 8TB partition; metadata maps to 4,000 bands
- Touching all `flex_bgs` → 360 GiB of cleaning load for just for metadata
- Avoid metadata writes → Dirty bands ↓ → Cleaning load ↓ → Throughput ↑
On ext4, metadata is written twice: sequentially to the journal and randomly to file system

- Ext4 uses journaling for metadata consistency
- Journaling implemented in a generic kernel layer JBD2
- Small (128 MiB) journal, managed like a circular log
- 1. JBD2 writes metadata to location $J$ in the journal and marks it dirty
- 2. After dirty timer expires, writeback thread writes block to static location $S$
Ext4-lazy: Ext4 with Lazy Writeback Journaling

- ① JBD2 writes metadata to location \( J \) in the journal and marks it clean
- ② JBD2 inserts a tuple to an in-memory map that maps \( S \) to \( J \)
- Large (10 GiB) journal with LFS-style cleaning
- Metadata reads in ext4 go through the map
- On mount, reconstruct in-memory map from journal
Agenda

- Introduction
- Background
- Mechanism
- Evaluation on CMR
- Evaluation on SMR
- Conclusion
On metadata-write heavy workloads, ext4-lazy is 2x faster on the CMR disk

- Benchmark: Create directories
  - 800,000 directories
  - Tree depth 10
  - 8 threads
Creating directories: The disk perspective

![Graph of ext4 I/O offsets](image1)

- **Disk Offset (GiB)**
- **Time (s)**
- **Metadata Writes** (red)
- **Metadata Reads** (purple)
- **Journal Writes** (blue)

![Graph of ext4-lazy I/O offsets](image2)

- **Disk Offset (GiB)**
- **Time (s)**
- **Metadata Reads** (purple)
- **Journal Writes** (blue)
Notoriously slow operation: Massive directory traversal

Doing an `rm -rf` on a massive directory tree takes hours

Is there anyway to speed it up? Is there any command that does the same as `rm -rf` and doesn't take hours and hours?

7

```
fileystems  files  performance  rsnapshot
```

8 Answers

No.

"rm -rf" does a recursive depth-first traversal of your filesystem, calling `unlink()` on every file. The two operations that cause the process to go slowly are `opendir()`/`readdir()` and `unlink()`. `opendir()` and `readdir()` are dependent on the number of files in the directory. `unlink()` is dependent on the size of the file being deleted. The only way to make this go quicker is to either reduce the size and numbers of files (which I suspect is not likely) or change the filesystem to one with better characteristics for these operations. I believe that XFS is good for `unlink()` on large files, but not...

Carnegie Mellon
Parallel Data Laboratory

http://www.pdl.cmu.edu/
On massive directory traversal workloads, ext4-lazy is 4x faster on the CMR disk

- Benchmark: Remove directories
  - 800,000 directories
  - Tree depth 10

![Runtime](image)

![ext4 I/O distribution](image)

![ext4-lazy I/O distribution](image)
Removing directories: The disk perspective

ext4 I/O offsets

Metadata Writes
Metadata Reads
Journal Writes

ext4-lazy I/O offsets

Metadata Reads
Journal Writes

Carnegie Mellon
Parallel Data Laboratory

http://www.pdl.cmu.edu/
Agenda

• Introduction
• Background
• Mechanism
• Evaluation on CMR
• Evaluation on SMR
• Conclusion
On pure metadata workloads, ext4-lazy results in \(~\text{zero}\) cleaning load on the SMR disk

- Benchmark: Create directories
  - 800,000 directories
  - Tree depth 10
  - 8 threads
On metadata-light (<1% of writes) workload, ext4-lazy achieves a large (1.7-5.4x) speedup on SMR disks

- Benchmark: Emulate a file server
  - 10,000 files in 25,000 dirs
  - File size up to 1 MiB
  - 100,000 transactions
  - I/O size 1 MiB
  - 38 GiB writes, 31 GiB reads
Ext4-lazy reduces the number of dirtied bands achieving higher throughput with full persistent cache.

![Graph showing throughput and I/O offsets over time for Ext4 and Ext4-lazy](chart.png)
Conclusion

- Ext4-lazy separates metadata from data and manages it as a log
- Consolidating metadata is not new:
  - MFS[SOSP’91], DualFS[ICS’02], hFS[EuroSys‘07], TableFS[ATC‘13]
- Ext4-lazy is an evolution of a production file system
  - Evaluates metadata separation idea for SMR disks
- Reducing the number of dirtied bands is a good rule of thumb for SMR

- Spreading metadata was invented 30 years ago. Time to revisit?
  - Explicit assumption: Random Read cost is high
  - Implicit assumption: Random Read cost == Random Write cost
- Assumptions do not hold anymore: read/write costs are asymmetric