Marchive
Extending MarFS to a Long Term Archive

Garrett Ransom
Los Alamos National Laboratory
LA-UR-20-26667
HPC Storage at LANL

Multiple clusters and a wide variety of storage requirements
HPC Storage at LANL

Used to orchestrate bulk data movement between storage systems
Local memory for each node of the compute cluster (DIMMs)
HPC Storage at LANL

Pure flash, high bandwidth storage intended to absorb ‘bursts’ of data
HPC Storage at LANL

General purpose workspace, mounted on every cluster node

Trinity
- CRAY XC30, 980K Cores, 2PB RAM

Fire/Ice: 1.7PF, 282 TB RAM
- CTS-1
  - Xeon E5-2695v4, 80K Cores

Luna - TLCC-2
- Viewmaster 2

Lustre

Campaign Storage
- 60 PB

VMS
- 576 TB

Shared Scratch
- 11 PB Lustre Filesystem

Net Scratch
- 300 TB

/home/projects
- 375 TB

Archive
- 70 PB stored, 50+ years of weapons data

IB Fabric

Ethernet Campus Network

Redcap

Discom 10 Gb/s Tri-Lab Network
Bulk disk storage intended for medium length residency (months/years)
HPC Storage at LANL

Tape archive intended for permanent data storage
Storage - Past and Future

Where We Were
- Memory
- Parallel FS
- Archive
  - Bandwidth = PBs/sec
  - Lifetime = Forever

Where We Are
- Memory
- Burst Buffer
- Lustre
- Campaign
- HPSS

Where We Want to Be
- Memory
- IOPs/BW Tier
- Capacity Tier
- Archive Tier

What was the problem?
Parallel FS doing too much:
- Low Latency
- High Bandwidth
- High Capacity
- Long Residency

Why aim for this?
Trying to avoid:
- Buying flash for capacity
- Buying tape for bandwidth
- Keeping bulk data forever
Campaign Storage Implementation

- Object storage has benefits
  - Easy scalability and resilience
- Object storage has limitations
  - Machines love object-IDs, people generally don’t
  - Applications expect POSIX file trees
- MarFS is LANL’s attempt to reconcile POSIX semantics with object storage
  - Focus on data protection and simplicity of design
What is MarFS?

- A near-POSIX interface layered over distinct metadata and data implementations
  - Scalability and resiliency of object storage for data
  - Actual POSIX metadata
- With tradeoffs, of course
  - No update in place
  - Restricted interactive use
What is MarFS?

Metadata mirrored within a parallel FS

Data stored as erasure coded pseudo-objects
What is MarFS?

- Object references stored as extended attributes of metadata files
- Multiple small files ‘packed’ into single objects
- Large files ‘chunked’ (broken up) over multiple objects
A MarFS-based Archive

- MarFS already offers:
  - Data validation via CRCs
  - Cross-server failure protection
  - Consistently sized objects via packing/chunking
  - Asynchronous garbage collection

- These sound like useful archive features
  - We can easily adapt the existing design to incorporate tape media
Marchive

Data Block 1 (1MB) | CRC (4B)
---+---
Data Block N (1MB) | CRC (4B)
---+---
Data Block (1MB) | CRC (4B)
---+---

Erasure Block 1 (1MB) | CRC (4B)
---+---
Erasure Block E (1MB) | CRC (4B)
---+---

Per-Block CRCs & Per-Stripe Erasure

Aggregates Checksum (8B)

Disk Arrays

Tape

Per-Chunk Checksums

Disk Pools (ZFS)

Tape Systems (TSM)
Marchive Interface

- Interactive access is not a good fit
  - Waiting for tape mounts is slow, regardless
  - Simultaneous user access compounds problems
  - Per-file I/O is likely grossly inefficient for tape
- A batch interface seems more promising
  - Far more efficient tape I/O
  - Far less abusive of tape media
  - Job priority can reduce wait time for essential tasks
Marchive Interface – Stage In

1. Submit stage in (write) job
2. Verify available space & trigger stage into MarFS
3. Disk Size Limit or Scheduled Staging Period Reached

Batch Interface

Job Scheduler

Standard MarFS Interface

Remote FS

ZFS Pool

Remote FS

Tape System

Tape System
Marchive Interface – Stage Out

(1) Submit stage out (read) job

(2) Backlog Size Limit or Scheduled Staging Period Reached

(3) Trigger stage out of MarFS

Batch Interface

Job Scheduler

Standard MarFS Interface

Remote FS

ZFS Pool

ZFS Pool

Tape System

Tape System
Marchive Components

New Components:
- Batch User Interface
- Obj Staging Utility
  - Front End
  - Back End

Existing Components:
- Job Scheduler
- Standard MarFS Interface
- Stage to/from remote FS
- Stage to/from tape system
  - Tape System

Submit Request
Submit path list
Submit Request
MarFS Improvements

- Extension to an archive system will require more functionality and stability from MarFS
  - Improved administration tools
  - Erasure code optimization
  - Improved config parser
  - Altered deletion / garbage collection process
MarFS Deletion Process

User Metadata Tree

Garbage Metadata Tree

File A

Object Xattr: <Object53>.<offset0>

Data Object #53
MarFS Deletion Process

User Metadata Tree

File A
Object Xattr: <Object53>.<offset0>

Duplicate Reference

Data Object #53

Garbage Metadata Tree

Garbage - File A
Object Xattr: <Object53>.<offset0>
MarFS Deletion Process

User Metadata Tree

File A

Object Xattr:
<Object53>.<offset0>

Garbage Metadata Tree

Garbage - File A

Object Xattr:
<Object53>.<offset0>

Data Object
#53
MarFS Deletion Problems

- Deletion is a non-atomic operation
  - Requires at least two syscalls (getxattr / unlink)
  - Interleaved operations could result in dropped object references
- For an archive, dropped references are a problem

Proc 1 -- Overwrite Target
Proc 2 -- Delete Target

Proc 1: Copy ObjRef
Proc 2: Copy ObjRef
Proc 1: Delete Target
Proc 1: Write new file
Proc 2: Delete Target
- LOST OBJECT REF -
MarFS Deletion Problems

- Can we scan objects/metadata to correct this?
  - Yes, but that is prohibitively costly
- Can we rename into the ‘trash’ tree instead?
  - Possibility of overwriting other trash files
  - Few filesystems implement the “renameat2” syscall
- What if we’re looking at this backwards? Maybe we can create the ‘trash’ reference at creation time?
Modified MarFS Creation Process

Reference Metadata Tree

User Metadata Tree

File A
Link Count: 1
Object Xattr: <Object53>.<offset0>

Data Object #53
Modified MarFS Creation Process

Reference Metadata Tree

Hardlink Into User Tree

User Metadata Tree

File A

Link Count: 2

Object Xattr:
<Object53>.<offset0>

Data Object #53

File A

Link Count: 2

Object Xattr:
<Object53>.<offset0>

Data Object #53
Modified Creation - Problems

- Won’t this double the size of our metadata?
  - Not quite, though it will double the number of directory entries (usually far smaller than inodes)
  - Even at scale, metadata simply isn’t that large
- What about collisions on creation?
  - The open syscall can avoid overwrites (O_EXCL)
  - Though unlikely, creation could fail with EBUSY
Modified Creation - Benefits

- Almost all metadata ops are trivial
  - Unlink / rename can freely target the user tree
- Garbage Collection and Quota calculation can be a single process
  - Scan the reference tree: inodes with a link-count of 1 are trash, otherwise they count against quota
  - Scans are highly parallelizable; likely no need for filesystem-specific scanning utilities
Modified Metadata Structure

Reference Metadata Tree
- File C (LC: 1)
  - File A (LC: 2)
  - File B (LC: 3)
User Metadata Tree
- GC Target
  - Add to Quota
  - Object C
  - Object B
  - Object A
Thank you for your attention!

Please take a moment to rate this session.

Your feedback matters to us.