MarFS, Marchive, and GUFI – Long Term Storage Strategies at LANL

David Bonnie
Los Alamos National Laboratory
US Department of Energy
Simulation HPC site

- Large jobs (30%+ of system, up to ~80%)
- Run for 6-12 months for a computing campaign
- Defensive checkpointing in both N-1 and N-N forms – up to petabyte scale files
Storage @ LANL

Trinity: 41.5PF/s, 2PB RAM
Cray XC30, 980K Cores

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

Viewmaster 2

IB Fabric
Ethernet Campus Network
Redcap

Campaign Storage
VMS
Shared Scratch RL3/RL4
Net Scratch
/home/projects
Archive
Discom 10 Gb/s
Tri-Lab Network

60 PB
576 TB
11 PB Lustre Filesystem
300 TB
375 TB
70 PB stored, 50+ years of weapons data

2018 Storage Developer Conference. Los Alamos National Laboratory LA-UR-18-28675
Storage @ LANL

- Many layers of storage
  - Explicit tiering between layers by users
  - 2 new layers with Trinity – Burst Buffer and Campaign
  - Complicated the user’s job of shepherding data even further
Why complicate matters?

- Burst buffer for economic reasons ($ / GB/s)
- Campaign for...economic reasons ($ / GB/s and $ / GB)
  - Unintuitive at first, but much easier to scale disk than tape for bandwidth
- ...but existing POSIX solutions were expensive
Why build our own file system?

- Existing POSIX solutions either:
  - Expensive
  - Unsafe
  - Unsuitable to workload / users
  - Combination of the above 😊
Enter MarFS

- This is our current “Campaign” tier
- We compromise a key part of POSIX:
  - Update in place / seek on write
- Given that compromise, we gain a lot
  - IO shaping is possible (save IOPs on write)
  - IO protection is now easy™ (batch IO efficiency)
- Design goals: transparency, protection of data above all else, recoverability, and ease of administration
MarFS in a nutshell

- FUSE daemon – full POSIX metadata, full data read access
- Library – simple API for all access, abstracted calls for simplicity via DAL/MDAL
- pftool – optimized parallel data movement tool
- Admin utilities – quota generation, trash management, offline packing
Scalability at its core
Metadata scaling stunt

- We built a test harness with MPI to push the limits of the MarFS metadata design
- Initial MD scaling runs on Cielo (1.1 PF, ~140,000 cores, ~9000 nodes)
  - 968 billion files, one single directory
  - 835 million file creations *per second* to metadata repos on each node
  - No cheating! Every create traversed the network from client to server
    - QD=1 for each client, 8 clients per node, 8 servers per node
- Further testing on Mustang (230 TF, ~38,000 cores, ~1600 nodes)
  - Large directory readdir tested from 10-50 nodes (sequential and parallel)
  - Peak of 300 million files per second across 50 nodes
  - More than 400X speedup over a single client sequential readdir
Sample Multi File

MultiFile - Attrs: uid, gid, mode, size, dates, etc.
Xattrs - objid repo=1, id=Obj002., objoffs=0, chunksize=256M, ObjType=Multi, NumObj=2, etc.

/Object System 1
Obj002.1
Obj002.2

/Object System X

/MarFS top level namespace aggregation

/GPFS-MarFS-md1
Dir1.1
trashdir

/GPFS-MarFS-mdN

Dir2.1
Sample Packed File

/MarFS  top level namespace aggregation

/GPFS-MarFS-md1

Dir1.1

trashdir

/GPFS-MarFS-mdN

Metada

Data

UniFile - Attrs: uid, gid, mode, size, dates, etc.

Xattrs - objid repo=1, id=Obj003, objoffs=4096, chunksize=256M, Objtype=Packed, NumObj=1, Obj=4 of 5, etc.

Object System 1

Obj003

Object System X

2018 Storage Developer Conference. Los Alamos National Laboratory LA-UR-18-28675
Multi-Component Repositories

- Initially storage via commodity object storage
  - Very “black box”, vendor lock-in, design goals
- Developed our own “object” storage layer
  - Lean on local expertise in ZFS
  - Completely transparent layout
  - Erasure at two levels
Data Layout

- File Transfer Agent
- Meta-data servers
- Parity of 10+2

Each Zpool is a 17+3
Storage nodes in separate racks
Multiple JBODs per Storage Node
Data and Parity are round-robin to storage nodes
Storage Nodes NFS export to FTAs
File Format

Storage Path: `<prefix>/repo<N>/pod<P>/block<Q>/cap<X>/scatter<Y>/<obj_ID>.<part_num>`

- `<part_num> = 0`
  - Stripe 0
  - CRC
  - Stripe 0
  - CRC
  - Zero Fill
  - CRC

- `<part_num> = N - 1`
  - Stripe 0
  - CRC
  - Stripe 1
  - CRC
  - Zero Fill
  - CRC

- `<part_num> = N`
  - P - 0
  - CRC
  - P - 1
  - CRC
  - P - 2
  - CRC

- `<part_num> = N + E - 1`
  - Q - 0
  - CRC
  - Q - 1
  - CRC
  - Q - 2
  - CRC

Manifest Files:

( N, E, offset, block_size, part_size, comp_part_size, crc_sum, total_size )
Lessons learned (so far)

- Transparency at the lower levels of storage is absolutely key to problem analysis and repair.
- Reducing IOPs requirements at the bottom allows efficient use of un-agile disks (shingled HDDs).
- Simple design makes it easy to discover, analyze, and repair any problems as they come up.
Ongoing work

- RDMA native transport
- “Fuzzy” DAL
- Fine-grained IO timing
- Live capacity/storage migration
- Something even more “cold”…
So we have this MarFS thing…

- Scalable namespaces
- Quotas
- Easy to understand/administer
- Optimized write IO characteristics
- …can we make an archive out of this?
Enter Marchive

- MarFS + Archive == Marchive
- Very simple extension of the MarFS paradigm
  - Just replace the ZFS arrays with tape!
  - Lose agile read from FUSE
  - Batch process ingest/recall on tape
  - Mostly just automation and UX challenges
Many layers of storage

- By design – users will keep everything if allowed, and HSMs only contribute to that bloat
- Data management is entirely user-driven
  - Users go find unneeded data and delete, if prodded
  - Users have no easy way to find particular datasets unless they have a good hierarchy or they remember where they put it
  - Users have bad memories and bad hierarchies...(you can see where this leads)
  - ...lower (longer) tiers of storage systems accumulate cruft over time
Enter GUFI

- **Unified** index over home, project, scratch, campaign, and archive
- Metadata only with attribute support
- Shared index for **users** and admins
- Parallel search capabilities that are very fast (minutes for billions of files/dirs)
- Can appear as mounted file system where you get a virtual image of your file metadata based on query input
- Full/Incremental update from sources with reasonable update time/annoyance
- Leverage **existing tech** as much as possible both hdwr and software: flash, threads, clusters, sql as part of the interface, commercial db tech, commercial indexing systems, commercial file system tech, threading/parallel process/node run times, src file system full/incremental capture capabilities, posix tree attributes (permissions, hierarchy representation, etc.), open source/agnostic to leveraged parts where possible.
- **Simple** so that an admin can easily understand/enhance/troubleshoot
Initial thoughts

- Why not a flat namespace?
  - Performance is great, but…
  - Rename high in the tree is terribly costly
  - Security becomes a nightmare if users/admins can access the namespace
- Leverage things that already work well, reduce required records to scan:
  - POSIX permissions / tree walk (readdir+)
  - Breadth first search for parallelization
  - Our trees have inherent namespace divisions for parallelism
  - Embedded DBs are fast if not many joins and individual DB size < TB
  - Flash storage is cheap enough to hold everything with order ~10K IOPs each
  - Entries in file system reduce to essentially <dir count> * 3
  - Dense directories reduce footprint dramatically
  - SQL is easily utilized for general queries of attributes
Prototype

-Dir-Summary —
  DB with summary of this directory

-Tree-Summary —
  DB with summary of the tree below optional can be placed anywhere

-Entries —
  DB with name/stats/linkname/xattr info for each file or link

Process/Node Parallelism for different parts of the tree, within each system-namespace combination use thread based parallelism
Draft DB Schemas

- Parent-Inode mapping file  “directories-parent-inode directories Inode”
  - Parent inode is only kept for directories, not for files as that kills rename/move function performance
- "CREATE TABLE entries(
  - name TEXT PRIMARY KEY, name of file (Not path due to renames)
  - type TEXT, inode INT, f for file l for link inode
  - mode INT, posix mode bits
  - nlink INT, number of links
  - uid INT, gid INT, uid and gid
  - size INT, blksize INT, size and blocksize
  - blocks INT, blocks
  - atime INT, access time
  - mtime INT, file contents modification time
  - ctime INT, metadata change time
  - linkname TEXT, if link this is path to link
  - xattrs TEXT);"; single text string, key/value pairs w/ delimiters
"CREATE TABLE summary(
    name TEXT PRIMARY KEY,
    type TEXT, inode INT,
    mode INT,
    nlink INT,
    uid INT, gid INT,
    size INT, blksize INT, blocks INT,
    atime INT, mtime INT, ctime INT,
    linkname TEXT, xattrs TEXT,
    totfiles INT, totlinks INT,
    minuid INT, maxuid INT, mingid INT, maxgid INT,
    minsize INT, maxsize INT,
    totlk INT, totmtk INT, totltm INT,
    totmtm INT, totmtg INT, totmtt INT,
    totsize INT,
    minctime INT, maxctime INT,
    minmtime INT, maxmtime INT,
    minatime INT, maxatime INT,
    minblocks INT, maxblocks INT,
    totxattr INT,
    depth INT);";

summary info for this directory

name TEXT not path due to rename
d TEXT for directory inode
mode INT, posix mode bits
nlink INT, number of links
uid INT, gid INT
size INT, blocksize, blocks
atime INT, mtime INT, ctime INT, access time, dir contents mod time, md chg time
lfile INT, path to link, xattr key/value delimited string
tot files in dir, tot links in dir
minuid INT, maxuid INT, mingid INT, maxgid INT
min and max uid and gid
minsize INT, maxsize INT
minimum file size and max file size
totlk INT, totmtk INT, totltm INT, total number of files lt KB mt KB, lt MB,
totmtm INT, totmtg INT, totmtt INT, total number of files mt MB mt GB, mt TB
minctime INT, maxctime INT
min max ctime
minmtime INT, maxmtime INT
min max mtime
minatime INT, maxatime INT
min max mtime
minblocks INT, maxblocks INT
min max blocks
totxattr INT, number of files with xattrs
depth this directory is in the tree

SDDC18

2018 Storage Developer Conference. Los Alamos National Laboratory LA-UR-18-28675
"CREATE TABLE treesummary(
  totsubdirs INT,
  maxsubdirfiles INT, maxsubdirlinks INT,
  maxsubdirsize INT,
  totfiles INT, totlinks INT,
  minuid INT, maxuid INT, mingid INT, maxgid INT,
  minsize INT, maxsize INT,
  totltk INT, tomtk INT, totlmt INT,
  tomtmm INT, tomtmg INT, totmtt INT,
  totsize INT,
  minctime INT, maxctime INT,
  minmtime INT, maxmtime INT,
  minatime INT, maxatime INT,
  minblocks INT, maxblocks INT,
  totxattr INT,
  depth INT);"

summary info for this tree
tot subdirs in tree
maxfiles in a subdir max links in a subdir
most bytes in any subdir
tot files in tree, tot links in tree
min and max uid and gid
minimum file size and max file size
total number of files lt KB mt KB, lt MB,
total number of files mt MB mt GB, mt TB
total bytes in files in tree
min max ctime
min max mtime
min max mttime
min max blocks
number of files with xattrs
depth this tree summary is in the tree
Programs Included / In Progress

- DFW – depth first walker, prints pinode, inode, path, attrs, xattrs
- BFW – breadth first walker, prints pinode, inode, path, attrs, xattrs
- BFWI – breadth first walker to create GUFI index tree from source tree
- BFMI – walk Robinhood MySQL and list tree and/or create GUFI index tree
- BFTI – breadth first walker that summarizes a GUFI tree from a source path down, can create treesummary index of that info
- BFQ – breadth first walker query that queries GUFI index tree
  - Specify SQL for treesummary, directorysummary, and entries DBs
- BFFUSE – FUSE interface to run POSIX md tools on a GUFI search result
- Querydb – dumps treesummary, directorysummary, and optional entry databases given a directory in GUFI as input
- Programs to update, incremental update (in progress):
  - Lustre, GPFS, HPSS
Early Performance Indicators

- All tests performed on a mid 2014 Macbook (quad core + nvme SSD)
- No tree indexes used
- ~136k directories, mostly small directories, 10 1M entry dirs, 20 100K size dirs, and 10 20M size dirs
- ~250M files total represented
- Search of all files: 2m10s (~1.75M files/sec)
- Search of all files and dirs: 2m19s (~1.63 M entries/sec)
- Search of all files and dirs, but exclude some very large dirs: 1m18s
- Search of all files and dirs, but exclude all < 1000 file directories: 1m59s

...on a laptop!
Learn more!

- [https://github.com/mar-file-system/GUFI](https://github.com/mar-file-system/GUFI)
- [https://github.com/mar-file-system/marfs](https://github.com/mar-file-system/marfs)
- [https://github.com/pftool/pftool](https://github.com/pftool/pftool)

Open Source

BSD License

Partners Welcome