MarFS: Near-POSIX Access to Objects

Jeff Inman
Los Alamos National Laboratory
Motivation

- See Gary Grider’s keynote presentation (Tuesday)
  - $B of POSIX
  - cost of replication (for those that care)
  - scalable BW
  - scalable namespace
  - cheap reliable capacity
- Trinity
  - 2 PB RAM
  - ~10 GB/s to archive
  - 5 tiers from RAM to tape
# Projected Trinity Reqs

<table>
<thead>
<tr>
<th>Tier</th>
<th>lifespan</th>
<th>bandwidth</th>
<th>change</th>
<th>capacity</th>
<th>change</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM</td>
<td>hours</td>
<td>9.1 PB/s</td>
<td></td>
<td>2 PB</td>
<td></td>
</tr>
<tr>
<td>BurstBuffer</td>
<td>hours</td>
<td>4 TB/s</td>
<td>( \times 0.0005 )</td>
<td>4 PB</td>
<td>( \times 2 )</td>
</tr>
<tr>
<td>Lustre</td>
<td>weeks</td>
<td>1.2 TB/s</td>
<td>( \times 0.3 )</td>
<td>100 PB</td>
<td>( \times 25 )</td>
</tr>
<tr>
<td>MarFS</td>
<td>year(s)</td>
<td>30 GB/s ++</td>
<td>( \times 0.025 )</td>
<td>30 PB ++</td>
<td>( \times 0.3 )</td>
</tr>
<tr>
<td>Tape</td>
<td>forever</td>
<td>10 GB/s</td>
<td>( \times 0.33 )</td>
<td>60 PB ++</td>
<td>( \times 2 )</td>
</tr>
</tbody>
</table>
Cost-Modeling (1/4)

- study examined cost of cheap EC disks for long-term archive
  - relevant, but not the same as LANL MarFS deployment
- Archive BW may be an HPC-specific concern
- “Cost of Tape versus Disk for Archival Storage” (CLOUD’14)
  - Linear Programming
  - spreadsheet -> CSV -> constraint eqs -> solver
  - real-world requirements for capacity and BW, per year
  - projected technological capabilities
  - choose disks or tape in any given year
  - buy tape-drives & tapes or JBODs, per-year
  - parameter studies
Cost Modeling (2/4)

Linear Programming reduced total projected costs of archives through 2025

- parameter-study varies tape-parms
- rigid tape-only solution: 8% cheaper
- disk-only solution: similar
- flexible per-year: ≤ 46% cheaper

Diagram showing the cost model with different solutions and their cost implications.
What the optimizer did in the best solution from previous slide

- solid-triangles: all-disk
- empty-triangles: all-tape
- red: per-year new archive storage
- green: min archive requirement
- blue: migration (from tape)
Cost-Modeling (4/4)

- **take-away**
  - clever all-tape deployment: reduced archive costs
  - clever all-disk deployment: similarly-reduced costs
  - cleverly-alternating storage: much cheaper

- **Study was “Disk versus Tape for Archival Storage”**
  - at LANL, MarFS occupies a BW tier above tape
  - 5-tier architecture should reduce archive demands
  - model assumes tape-drives can run 24/7/365
  - model doesn’t include labor costs

- **One could use MarFS for the all-disk technology in the models**
  - more-accessible metadata for cloud storage
Objects vs Files

- directories/files are human-friendly (familiar)
  - `/users/jti/projects/git/marfs/fuse/src/mdal.h`
  - `namespace` -- explicit structure & context
  - compartmented data, search, security
  - $Billions in apps that assume files

- objects are scalable and resilient
  - `http://host:port/bucket/09bac9f...`
  - `repository` -- cheap scalable capacity
  - awkward metadata access
MarFS Basic Structure

**Meta-Data (namespace)**

- `/`
- `dir/`
  - `file1`
  - `file2`
  - `file3`

**Data (repository)**

- `put`
- `get`
- `delete`
- `mknod`
- `setxattr`
- `mkdir`
- `opendir`

Operations:
- `open`
- `close`
- `write`
- `read`
- `...`

Data Location:

```
http://10.10.0.xx:81/bparc/proxy1/a2334ba0f3e...
```
MarFS File-Types

- DIRECT (legacy files)
- Uni (1 file -> 1 object)
- Multi (1 file -> N objects)
- Packed (M files -> 1 object)
  - offline packer
  - GC / repacker
  - pftool

- recovery-info at tail of file storage, after file-data
  - travels with packed files
MetaData File

- regular file
  - chown/chmod/rename/etc, as usual
  - truncated to size
  - xattrs (invisible to users)
    - objid: proxy/repo2/ver.001_004/ns.jti/F___/inode.0000027328/md_ctime.20160829_130133-0600_1/
      obj_ctime.20160829_130135-0600_1/unq.0/chnksz.40000000/chnkno.0
    - post: ver.001_004/U/off.0/objs.1/bytes.0/corr.0000000000000000/crypt.0000000000000000/flags.00/mdfs.
    - [restart]
“Near” POSIX?

- Writes must be sequential (per chunk)
  - relaxed for well-behaved parallel writers (pftool)
  - parallel access through pftool
- No update-in-place
  - overwrite entire file
- No sparse files
  - no seek() for writing
  - each chunk of a Multi is full-size (except maybe last)
  - easy computation of offset -> chunk
Libraries, Applications, etc.

- **FUSE**
  - `open`
  - `close`
  - `write`
  - `read`
  - `cp`
  - `ls`
  - `rm`
  - `rsync`

- **pftool**
  - `pfcp`
  - `pfcm`

- **Path**
  - `open`
  - `close`
  - `write`
  - `read`

- **POSIX**
- **MarFS**
  - `marfs_open`
  - `marfs_close`
  - `marfs_write`
  - `marfs_read`

- **PLFS**

**Libmarfs**
- **base**
  - `str_2_pre`
- **common**
  - `read_config`
  - `trash_unlink`
  - `expand_info`
  - `save_xattrs`
- **stream**
  - `str_open`
  - `str_put`
  - `str_get`
  - `str_sync`
- **ops**
  - `marfs_open`
  - `marfs_close`
  - `marfs_write`
  - `marfs_read`

**PA2X**

**aws4c**
- `iob_reset`
- `iob_append`
- `iob_extend`
- `s3_put`
- `s3_get`

**GC**
- **quotas**
- **packer**

**libcurl**
<config>
  
  <name>ODSU Testbed</name>
  <version>1.0</version>
  <mnt_top>/campaign</mnt_top>

  [repositories ...]

  [namespaces ...]

</config>
Configuration (repo)

```xml
<repo>
  <name>bparc</name>

  # 10.10.0.1 - 10.10.0.12
  <host>10.10.0.%d:81</host>
  <host_offset>1</host_offset>
  <host_count>12</host_count>

  <update_in_place>no</update_in_place>
  <access_method>SPROXYD</access_method>
  <chunk_size>1073741824</chunk_size>  # 1GB

  <security_method>HTTP_DIGEST</security_method>

  <enc_type>NONE</enc_type>
  <comp_type>NONE</comp_type>
  <correct_type>NONE</correct_type>
  <latency>10000</latency>
</repo>
```
<namespace>
  <name>admins</name>
  <alias>proxy1</alias>
  <mnt_path>/admins</mnt_path>

  <iperms>RM,WM, RD, WD, TD, UD</iperms>  # interactive (fuse)
  <iwrite_repo_name>bparc</iwrite_repo_name>  # matches some <repo> spec

  <bperms>RM, WM, RD, WD, TD, UD</bperms>  # batch (pftool)
  <range>  # batch writes for files in given size-range
    <min_size>0</min_size>
    <max_size>-1</max_size>
    <repo_name>bparc</repo_name>
  </range>

  <md_path>/gpfs/project/admins/mdfs</md_path>
  <trash_md_path>/gpfs/project/trash</trash_md_path>
  <fsinfo_path>/gpfs/foo/project/fsinfo</fsinfo_path>

  <quota_space>-1</quota_space>
  <quota_names>-1</quota_names>
</namespace>
open(RDONLY)

- can read at arbitrary offset
  - GET starts in read(), not open()
- concurrent reads OK
  - multiple file-handles coordinated at repo
  - single file-handle serialized (NFS)
- GET req has range = [open_offset, rest_of_chunk]
  - max GET req size can be configured
open(RDONLY)
read()
Read Notes

- cross chunk boundaries
- non-contiguous reads mean close/reopen
  - costly!
- NFS
  - multiple offsets / thread
  - multiple threads / file-handle
  - multiple file-handles
Stupid NFS Tricks

- defer close/reopen to wait for other readers
  - detect multiple-threaded reads
  - enqueue out-of-order requests
  - request-timeout -> close/reopen
open(WRONLY)

- no need for file-handle locks at open
  - require open-offset % chunk_size == 0
    - (accounting for recovery-info)
  - concurrent writes to different chunks via pftool
  - fuse writes overwrite
  - RESTART xattr locks file for read

- trash
open(WRONLY)
write()
Write Notes

- optimized for non-zero writes
  - starting PUT in open() is hard on MD benchmarks
    - cost of PUT zero-sized object (plus recovery-info)
  - work-around via DAL (explained later)

- NFS
  - arbitrary write-offsets illegal
  - work-around via DAL (different one)
MarFS Basic Structure (review)

Meta-Data (namespace)
- open
- close
- write
- read
- ...
- mknod
- setxattr
- mkdir
- opendir

Data (repository)
- put
- get
- delete

http://10.10.0.xx:81/bparc/proxy1/a2334ba0f3e...
MD Abstraction Layer (MDAL)

MDAL (per namespace)

POSIX

```
open
close
write
read
```

/mnt/project1

```
mknod
setxattr
mkdir
opendir
```

/mnt/project2

```
mknod
setxattr
mkdir
opendir
```

```
mknod
setxattr
mkdir
opendir
```

```
mknod
setxattr
mkdir
opendir
```

```
mknod
setxattr
mkdir
opendir
```

```
mknod
setxattr
mkdir
opendir
```

```
mknod
setxattr
mkdir
opendir
```
MDAL Implementations

- **init**
  - init custom-context, in MarFS file-handle
  - access to per-MDAL static state
- **context-based ops:** `open/close/read/write/etc`
  - use state in custom-context
- **context-free ops:** `mknod/unlink/setxattr/etc`
  - operate on paths
- **destroy**
  - clean-up custom-context, etc.
MarFS Basic Structure (review)

- Meta-Data (namespace)
  - /
  - dir/
    - file1
    - file2
    - file3

- Data (repository)
  - http://10.10.0.xx:81/bparc/proxy1/a2334ba0f3e...

- Operations
  - mknod
  - setxattr
  - mkdir
  - opendir
  - open
  - close
  - read
  - write
  - . . .
  - put
  - get
  - delete
  - mknod
  - setxattr
  - mkdir
  - opendir
Data Abstraction Layer (DAL)

DAL (per repository)

http://10.10.0.%d:81/bparc/proxy1/a2334ba0f3e...

/nfs/server_%d/repo/proxy2/hash1/hash2/a2334ba0f3e...

open
close
write
read

. . .
DAL Implementations

- init
  - init custom context, in MarFS file-handle
  - access to per-DAL static state
- context-based ops: open/close/get/put
  - use state in custom-context
- context-free ops: delete
  - operate on paths
- destroy
  - clean-up custom-context, etc.
Work In Progress ...

- scalable metadata
- scalable file-based storage
  - NFS
Scalable MetaData (1/3)

- Namespace (MDAL)
- remove practical limits on MD capacity
- scalable performance
- directory-MD – directory-tree MD only
  - /mnt/dmds/ns/path/to/parentdir/
  - access-perms cached
  - hash parentdir inode -> hash1
- file-MD – sharded directories
  - NFS exports from multiple servers
  - hash fname -> hash2
  - /mnt/fmds[S]/ns/scatter[M]/parentdir.inode/file
Scalable MD (2/3)

# Tentative extensions to namespace config to support scalable MD
# (assumes an array of NFS exports)
<namespace>
  <name> mds </name>
  <md_type> SHARDED </md_type>

  <d_md_path> /mnt/dmds/mds/ </d_md_path>
  <f_md_path> /mnt/fmds%d/mds/scatter%d/%s.%d </f_md_path>
  <shards> S </shards> # number of fmd servers
  <hash_width> M </hash_width> # width of “scatter-tree”

  <trash_md_path>…</trash_md_path>
  <fsinfo_path>…</fsinfo_path>
</namespace>
Scalable MD (3/3)

- benchmarking with distributed app
- “NO-OP” DAL
  - open for write still issues PUT
  - DAL-impl reports success
  - no consistency issues with zero-length files
- “MDScale” MDAL
  - Use context in MPI
  - no xattrs in /dev/shm
- object-IDs refer to non-existent objects
Multi-Component Data Store (1/3)

- Repository (DAL)
  - store data in files instead of objects
    - Cf. DIRECT storage
  - scalable storage
    - 48 JBOD DSUs, w/ ZFS
    - DAL implements EC + GET/PUT/DEL to NFS mounts
    - `/mnt/repo10+2/stripe[4]/blk[12]/cap[C]/parN/scatter[M]/objID`
    - hash fname % 4 -> “stripe” of 12 servers
    - hash fname % C -> capacity group
    - parN dir-names support load-balancing
    - hash fname % M -> scatter (for inode scaling)
    - object-IDs stringified into single filename (no ‘/’)

# Extensions to config for file-based repo using **x2 replication**
# (assumes an array of NFS exports)

```xml
<repo>
    <name>repo_copy2</name>
    <access_method> MC_COPY 2 </access_method>

    # /mnt/repo_copy2/srv[48]/cap[C]/parN/scatter[M]/objID
    <host> /mnt/copy3/srv%d/cap%d </host>
    <host_count> 48 </host_count>
    <host_cap_count> C </host_cap_count> # hash to cap

    <host_sub> /%s/scatter%d </host_sub>
    <hash_width> M </hash_width>
</repo>
```
# Extensions to config for a file-based repo using 10+2 EC
# (assumes an array of NFS exports)
<repo>
  <name> repo10+2 </name>
  <access_method> MC_ERASURE 10 2 </access_method>

  # /mnt/repo10+2/stripe[4]/blk[12]/cap[C]/parN/scatter[M]/objID
  <host> /mnt/repo10+2/stripe%d/blk%d/cap%d </host>
  <host_count> 4 </host_count>  # hash to “stripe”
  # block-count is 10+2
  <host_cap_count> C </host_cap_count>  # hash to cap

  <host_sub> /%s/scatter%d </host_sub>
  <hash_width> M </hash_width>
</repo>
Production Architecture (1/2)

- object-based storage
- "interactive" nodes
  - MarFS FUSE mount
  - RM,WM,RD,WD,TD,UD
  - marfs pipe tool (tar –czv files | marfs_pipe /marfs/…)
  - pfcp spawns to “batch” nodes
- “batch” nodes
  - no interactive logins
  - MDFS mount
  - run pftool MPI jobs
Production Architecture (2/2)

- Multi-Component storage
- “interactive” nodes
  - MarFS NFS mount
  - RM,WM,RD,WD,TD,UD
  - marfs pipe tool (tar –czv files | marfs_pipe /marfs/…)
  - pfcp spawns to “batch” nodes
- “batch” nodes
  - no interactive logins
  - MDFS mount
- NFS export MarFS FUSE (w/ DAL to MC storage)
  - run pftool MPI jobs
Team

- Dave Bonnie
- Hsing-Bung Chen
- Ron Croonenberg
- Chris DeJager
- Greg Geller
- Gary Grider
- Chris Hoffman
- Jeff Inman
- Brett Kettering
- Kyle Lamb
- Chris Mitchel
- Garrett Ransom
- George Sparrow
- Alfred Torrez
- Will Vining
Acknowledgements

- Gary Grider
- Vlad Korolev (vlad@v-lad.org)
  - aws4c library
  - git@github.com:vladistan/aws4c.git
  - (git@github.com:jti-lanl/aws4c.git)
Questions?

jti@lanl.gov

https://github.com/mar-file-system/marfs
https://github.com/pftool/pftool
Object IDs

http://xx.xx.xx.xx:port
/proxy/repo2/ver.001_004/ns.jti/F___/inode.0000027328
/md_ctime.20160829_130133-0600_1
/obj_ctime.20160829_130135-0600_1
/unq.0/chnksz.40000000/chnkno.0

- sproxyd:
  - “proxy” is NS alias, matching httpd FastCGI line
  - “repo2” is Repo, matching sproxyd driver-alias

- S3:
  - “proxy” is NS alias, matching created bucket

- Multi-Component:
  - entire “/proxy/repo2/…” has all ‘/’ translated to ‘#’, becomes a file-name