DE LA RECHERCHE À L'INDUSTRIE

cea

www.cea.fr

Lecture on Parallel Filesystems

Lustre (5/5)

Jacques-Charles Lafoucriere

ENSIIE| 2018

Lustre HSM

Hierarchical Storage Manager

Definition

- Hierarchical storage management (HSM) is a data storage technique that automatically moves data between high-cost and low-cost storage media

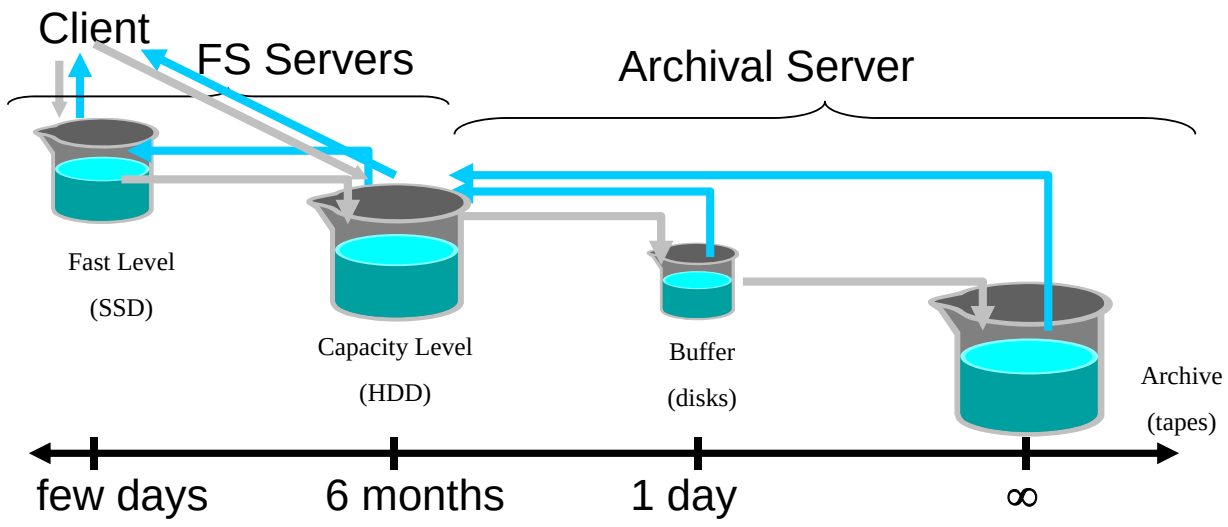
Objective

- All media does not have the same performances and same cost
- Having all data available on high-speed devices all the time is prohibitively expensive
- Managing data on multiple levels is difficult for users
- HSM do this automatically for the user/sys-admin

Different types

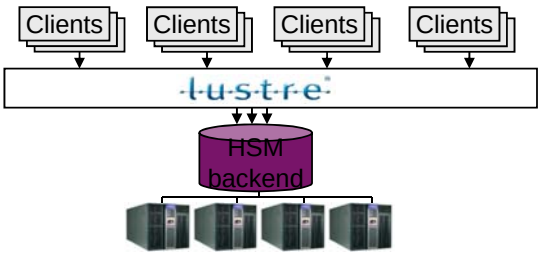
- FS
  - Disks to disks
- Archival Storage
  - Disks to tapes

Storage Tiers Levels



Lustre HSM Goals

HSM seamless integration



Takes the best of each world:

- Lustre: high-performance disk cache in front of the HSM
  - Parallel cluster filesystem
  - High I/O performance, POSIX access
- HSM: long-term data storage
  - Manage large number of disks and tapes, migration between them
  - Huge storage capacity

Lustre HSM Back-ends

HSM independent design

First supported storage back-ends

- HPSS
- POSIX
- Sam/QFS
- DMF
- Enstore
- TSM

Lustre HSM Design and Features

Features

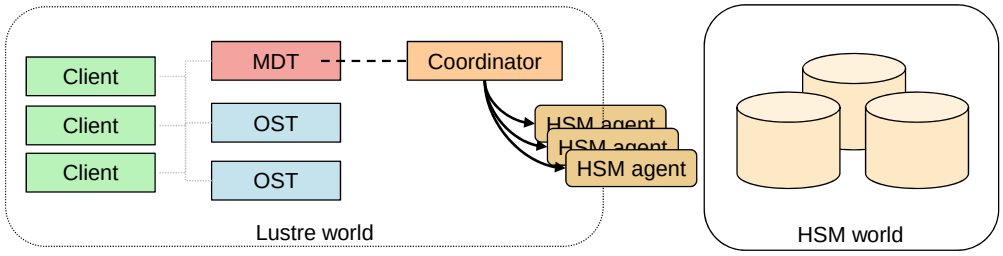
V1 Feature

- Migrate data to the HSM
- Free disk space when needed
- Bring back data on cache-miss
- Policy management (migration, purge, soft rm,…)
- Import from existing backend
- Disaster recovery (restore Lustre filesystem from backend)

New components

- Coordinator
- HSM agents (backend specific daemon)
- Policy Engine (user-space daemon)

Components (1/2)



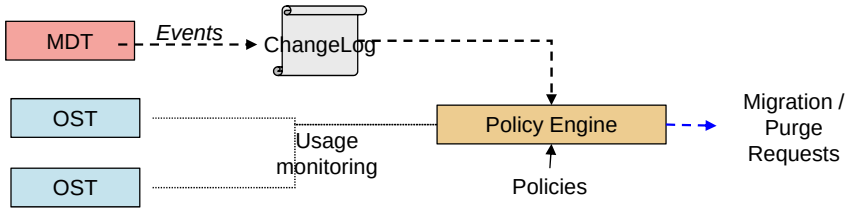
Coordinator

- Manages file locking on data restoration
- Centralizes copy requests
- Dispatches requests on agents

HSM agents (user space)

- Move data, cancel copy and remove external storage files
- Interface between Lustre and the HSM
- Know how to communicate with a specific HSM

Components (2/2)



Policy Engine (user-space)

- Monitors filesystem disk space usage
- Keeps track of files status and modification time
- Pre-migrates not recently modified data
- If free space is low, purges non recently accessed files (if already copied in the HSM)
- Deferred rm in HSM (soft rm)

Robinhood: PolicyEngine for Lustre-HSM

Robinhood is a Policy Engine implementation for Lustre-HSM

- CEA development
- OpenSource
- http://robinhood.sf.net

Policies

- File class definitions, associated to policies
- Based on files attributes (path, size, owner, age, xattrs…)
- Rules can be combined with boolean operators
- LRU-based migr./purge policies
- Entries can be white-listed

Robinhood: Example of Migration Policy

```
Fileset definition
Filesets {
          FileClass small_files_A {
              definition { tree == "/mnt/lustre/project_A" and size < 1MB }
              migration_hints = "cos=12" ;
…
          }
}
Migration policy
Migration_Policies {
          ignore { size == 0 or xattr.user.no_copy == 1 }
          ignore { tree == "/mnt/lustre/logs" and name=="*.log" }

          policy migr_A_small {
                    target_fileclass = small_files_A;
                    condition { last_mod > 6h or last_copyout > 1d }
          }
          …
          policy default {
                    condition { last_mod > 12h }
                    migration_hints = "cos=42";
          }
}
```

Robinhood: Example of Purge Policy

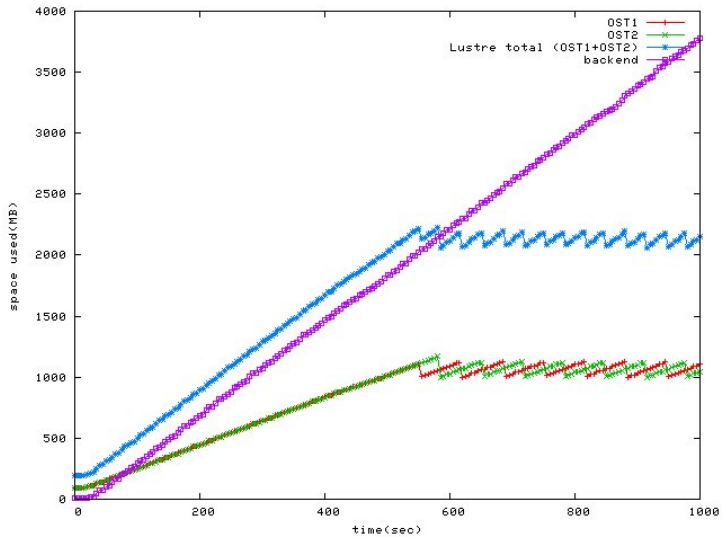
```
Purge trigger:
Purge_trigger {
          trigger_on = ost_usage;
          high_watermark_pct = 80%;
          low_watermark_pct = 70%;
}

Purge policy:
Purge_Policies {
          ignore { size < 1KB }
          ignore { xattr.user.no_release = 1 or owner == “root” }

          policy purge_quickly {
                    target_fileclass = classX;
                    condition { last_access > 1min }
          }
          …

          policy default {
                    condition { last_access > 1h }
          }
}
```


Policy Engine in action



Lustre HSM Commands

lfs hsm_state <*file(s)*>

- dirty/released/archived

lfs hsm_archive <*file(s)*> [--archive <*num*>]

- Copy file to the specified backend

lfs hsm_restore <*file(s)*>
open(<*file*>)

- Restore release file from backend

Lustre HSM Commands (cont.)

lfs hsm_release <file(s)>

- Release file data in Lustre

lfs hsm_remove <file(s)>

- Remove previous file copy in backend

lfs hsm_set --lost/dirty/no_release/no_archive… <file>

- Set status, set policy flags

Robinhood Admin Commands

rbh-hsm –sync

- Force archiving all modified files to the backend

rbh-hsm --archive-ost <ost_idx>

- Archive all modified files on the given OST

rbh-hsm –purge-fs <target_%>

- Release files, to the targeted usage level (LRU)

rbh-hsm –purge-ost <ost_idx>,<target_%>

- Release files per OST, to the targeted usage level (LRU)

rbh-hsm-report

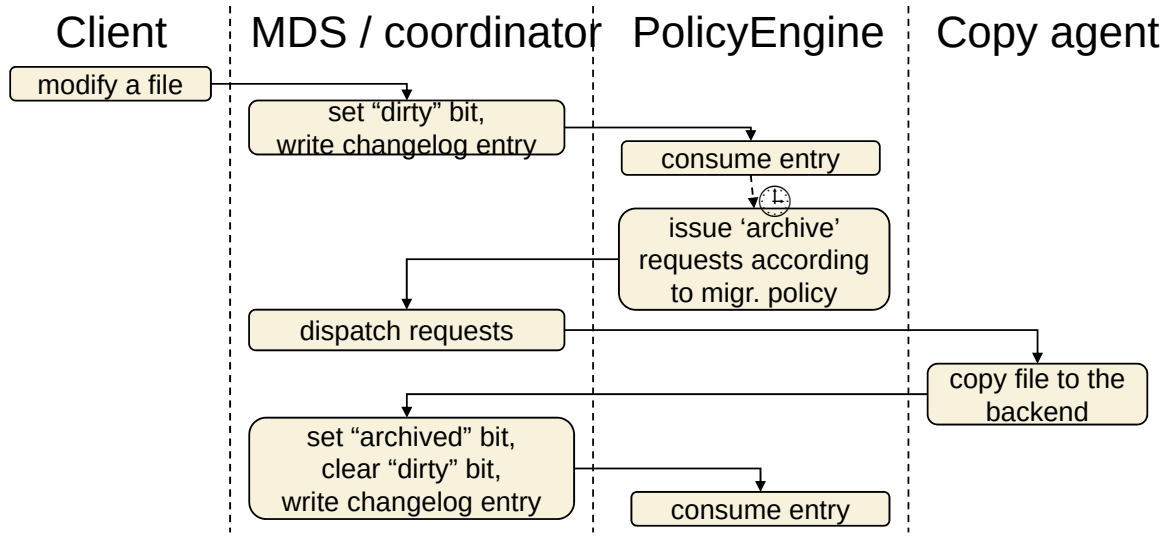
- fs content info (per status statistics)
- dump file list per OST, per status…

Lustre HSM V2: Incoming Features

Candidate features for Lustre-HSM v2

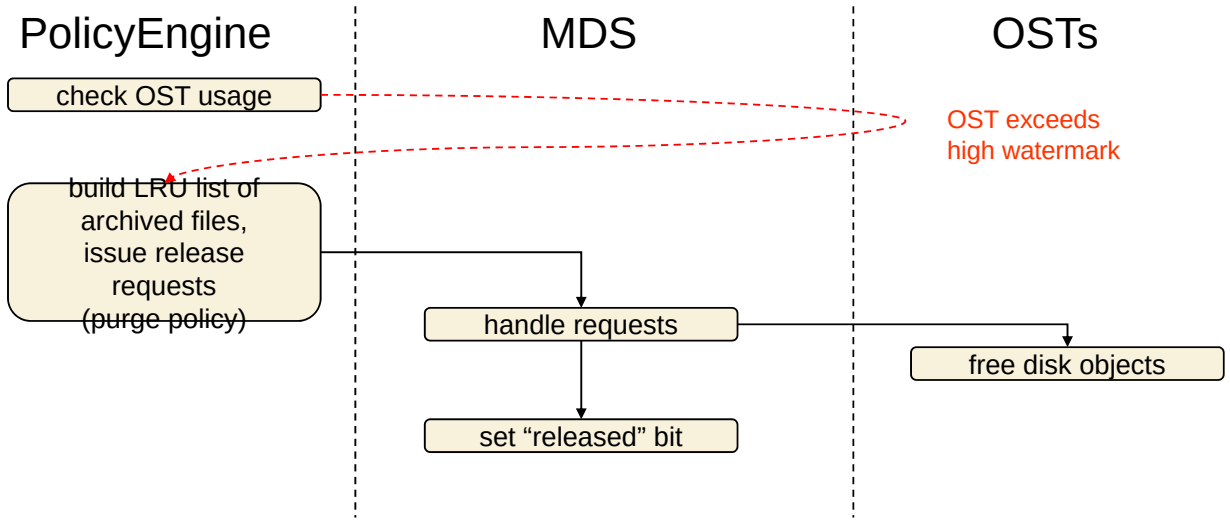
- Partial file release/restore (v1: whole file)
- Finest file locking on restore (v1: whole file)
- Online metadata snapshot
- Tape-friendly mass-restore operations
- Enhanced multiple archive support (mirroring…)
- File writes may cancel archive operation
- …

Use Case (1): Archive Modified Files



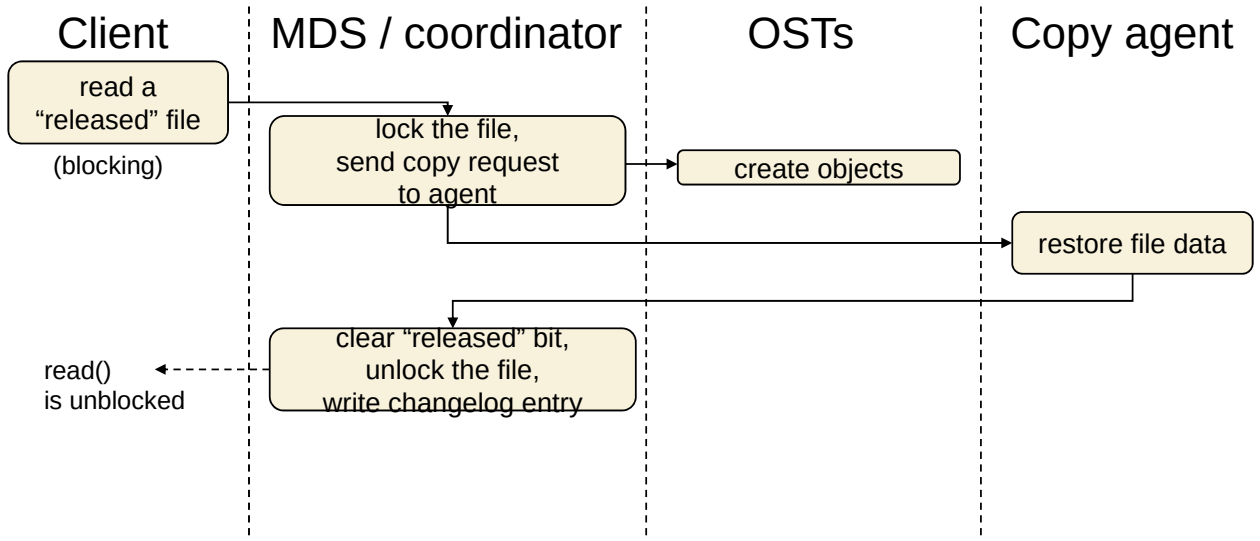
SFP 2018 Parallel Filesystems / Lustre 19

Use Case (2): Release Disk Space



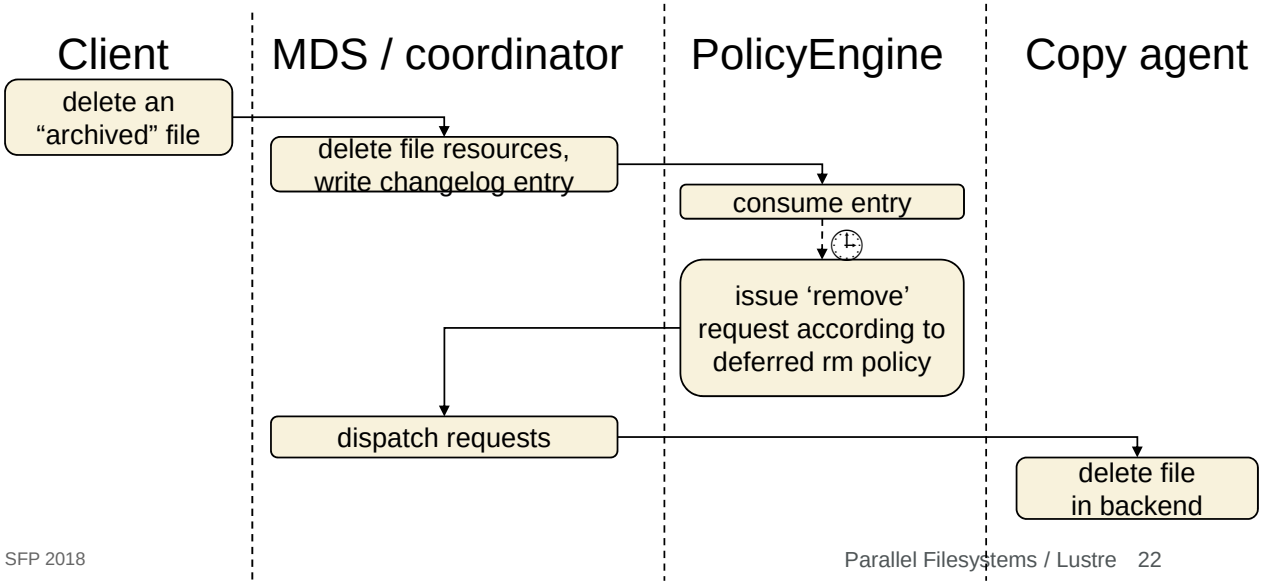
SFP 2018 Parallel Filesystems / Lustre 20

Use Case (3): Restore Released File



SFP 2018 Parallel Filesystems / Lustre 21

Use Case (4): Deferred Removal (soft rm)



SFP 2018 Parallel Filesystems / Lustre 22

Changelog

Changelog

Objective

- Provide a way to monitor object events

Implementation

- A journal of many Lustre events
  - Namespace changes
  - Object metadata changes
- Use permanent storage on MDT
- Purge after consumption

Changelog Events

```
enum changelog_rec_type {
        CL_MARK     = 0,
        CL_CREATE   = 1,  /* namespace */
        CL_MKDIR    = 2,  /* namespace */
        CL_HARDLINK = 3,  /* namespace */
        CL_SOFTLINK = 4,  /* namespace */
        CL_MKNOD    = 5,  /* namespace */
        CL_UNLINK   = 6,  /* namespace */
        CL_RMDIR    = 7,  /* namespace */
        CL_RENAME   = 8,  /* namespace */
        CL_EXT      = 9,  /* namespace extended record (2nd half of rename) */
        CL_OPEN     = 10, /* not currently used */
        CL_CLOSE    = 11, /* may be written to log only with mtime change */
        CL_LAYOUT   = 12, /* file layout/striping modified */
        CL_TRUNC    = 13,
…
```


Changelog Events (cont.)

```
enum changelog_rec_type {
…
        CL_SETATTR  = 14,
        CL_SETXATTR = 15,
        CL_XATTR    = CL_SETXATTR, /* Deprecated name */
        CL_HSM      = 16, /* HSM specific events, see flags */
        CL_MTIME    = 17, /* Precedence: setattr > mtime > ctime > atime */
        CL_CTIME    = 18,
        CL_ATIME    = 19,
        CL_MIGRATE  = 20,
        CL_FLRW     = 21, /* FLR: file was firstly written */
        CL_RESYNC   = 22, /* FLR: file was resync-ed */
        CL_GETXATTR = 23,
        CL_DN_OPEN  = 24, /* denied open */
        CL_LAST
};
```

Changelog Usage

Register a consumer

```
# lctl -device FSNAME-MDTXXXX changelog_register
FSNAME-MDTXXXX: Registered changelog userid 'cl1'
```

Get events

```
$ lfs changelog FSNAME-MDTXXXX [startrec [endrec]]
```

Ack events

```
$ lfs changelog_clear FSNAME-MDTXXXX userid endrec
```


Changelog Usage (Cont.)

Deregister

```
# lctl -device FSNAME-MDTXXXX changelog_deregister userid
```

A parameter can be set to mask some events

- mdd.FSNAME-MDT0000.changelog_mask

Changelog Events Example

```
# lfs changelog test-MDT0000 ; mkdir spooo ; touch spoo1 ; mkdir -p dir/dir2/dir3
# lfs setstripe -c3 spo23
# lfs changelog test-MDT0000
1 02MKDIR 14:27:57.634205651 2018.05.13 0x0 t=[0x200000401:0x1:0x0] ef=0xf u=0:0 nid=0@lo
p=[0x200000007:0x1:0x0] spooo
2 01CREAT 14:27:57.637429458 2018.05.13 0x0 t=[0x200000401:0x2:0x0] ef=0xf u=0:0 nid=0@lo
p=[0x200000007:0x1:0x0] spoo1
3 11CLOSE 14:27:57.639463292 2018.05.13 0x42 t=[0x200000401:0x2:0x0] ef=0xf u=0:0 nid=0@lo
4 02MKDIR 14:28:51.348938820 2018.05.13 0x0 t=[0x200000401:0x3:0x0] ef=0xf u=0:0 nid=0@lo
p=[0x200000007:0x1:0x0] dir
5 02MKDIR 14:28:51.350019437 2018.05.13 0x0 t=[0x200000401:0x4:0x0] ef=0xf u=0:0 nid=0@lo
p=[0x200000401:0x3:0x0] dir2
6 02MKDIR 14:28:51.350969196 2018.05.13 0x0 t=[0x200000401:0x5:0x0] ef=0xf u=0:0 nid=0@lo
p=[0x200000401:0x4:0x0] dir3
7 01CREAT 14:30:09.424942626 2018.05.13 0x0 t=[0x200000401:0x6:0x0] ef=0xf u=0:0 nid=0@lo
p=[0x200000007:0x1:0x0] spo23
8 12LYOUT 14:30:09.425691953 2018.05.13 0x0 t=[0x200000401:0x6:0x0] ef=0xf u=0:0 nid=0@lo
9 11CLOSE 14:30:09.425993215 2018.05.13 0x2 t=[0x200000401:0x6:0x0] ef=0xf u=0:0 nid=0@lo
```


FID Interface

How To Find FID/PATH couple?

Each Lustre Object is associated with

- A path name
- A FID

To get object FID

```
$ lfs path2fid object
```

To get PATH from FID

```
$ lfs fid2path FSNAME FID
```


Thank you for your attention

ENSIIE | 2018

Commissariat à l'énergie atomique et aux énergies alternatives
Centre DAM-Ile de France | 91297 Bruyères-le-Châtel Cedex
T. +33 (0)1 69 26 40 00 | F. +33 (0)1 69 26 70 86
Direction des applications militaires
Département sciences de la simulation et de l'information
Service informatique scientifique et réseaux

Lustre Glossary

DNE - Distributed Namespace Environment - feature to aggregate multiple MDTs (possibly on many MDS's) into a single filesystem namespace

IDIF - OST object ID In FID - specific FID range reserved for compatibility with pre-DNE OST objects

IGIF - Inode and Generation In FID - specific FID range reserved for compatibility from Lustre 1.x MDT inode objects

FID - File IDentifier - unique 128-bit identifier for every object within a single filesystem.

LMV - Logical Metadata Volume - client software layer that handles client (llite) access to multiple MDTs

LOD - Logical Object Device - MDS software layer that handles access to multiple MDTs and multiple OSTs

LOV - Logical Object Volume - client software layer that handles client (llite) access to multiple OSTs

MDC - MetaData Client - client software layer that interfaces to the MDS

MDD - Metadata Device Driver - MDS software layer that understands POSIX semantics for file access

MDS - MetaData Server - software service that manages access to filesystem namespace (inodes, paths, permission) requests from the client.

MDT - MetaData Target - storage device that holds the filesystem metadata (attributes, inodes, directories, xattrs, etc)

MGS - Management Server - service that helps clients and servers with configuration

MGT - Management Target - storage device that holds the configuration logs

OFD - Object Filter Device - OSS software layer that handles file IO

OSC - Object Storage Client - client software layer that interfaces to the OST

OSD - Object Storage Device - server software layer that abstracts MDD and OFD access to underlying disk filesystems like ldiskfs and ZFS

OSP - Object Storage Proxy - server software layer that interfaces from one MDS to the OSD on another MDS or another OSS

OSS - Object Storage Server - software service that manages access to filesystem data (read, write, truncate, etc)

OST - Object Storage Target - storage device that holds the filesystem data (regular data files, not directories, xattrs, or other metadata)