Lustre file systems

A brief introduction

Patrick Keller

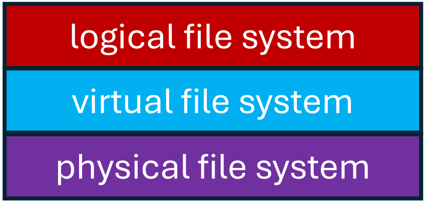
Peter-Bernd Otte

8.3.2024

Definition File System

- file system (FS): *data structure that controls how data is stored and retrieved*. (Wikipedia)
- without a file system, data placed:
  - would be one large body of data,
  - *no way to tell where one piece of data stopped and the next began,*
  - *or where – when retrieving - any piece of data was located*

- layered approach
  - logical file system
  - virtual file system (VFS)
  - physical file system



Logical file system (LFS)

logical file system

- LFS represents Physical FS
- logical file represents one or multiple physical files
- Logical files have no data. They have a description of the records found in one or multiple physical files.

- usually syscalls in Linux:
  `open, read, write, close, lseek, stat, …`
- same interface in userspace,
  different mechanisms in the lower layers

virtual file system

Virtual file system (VFS)

- **FUSE: F**ilesystem in **Use**rspace
    - Interface to VFS in Linux
- kernel module as an interface to userspace
- users can mount Filesystems directly
    - NTFS-3G: Windows file systems under Linux
    - SSHFS
    - CVM-FS

VFS concepts: inode

- „Everything after a backslash is an inode“
  `/directory/file`

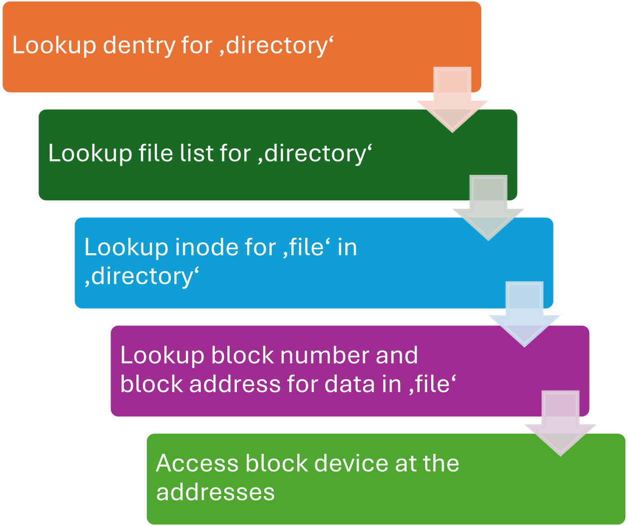
- inode contains metadata:
  - type
  - uid, gid
  - permissions
  - timestamps (atime, mtime, ctime)
  - file size
  - number of blocks and block pointers for actual data

VFS concepts: dentry/dcache

- directory entry (dentry)
  - Pathname to a file or a directory
  - `/some/directory/file.out`
- dentries usually have pointers to an inode object
  - addresses on block devices
  - memory for pseudo filesystems (e.g. /tmp)

- caching mechanism: “dcache”
  - in RAM
  - RAM limited → not all dentries included

Reading data from `/directory/file` within the VFS layer in a classical file system



Parallel File Systems

Parallel file system

- layer concept is not applicable to distributed file systems, because:

- parallel access (several clients)
- parallel storage (multiple servers)
- multiple servers with multiple disks store data

Parallel file system – Pros / Cons

+

- better performance (load distribution)
- scalability (in performance and volume)
- redundancy

-

- overhead
- locking
- complex
- (unintuitive)

Parallel file system - Examples

- Lustre (Opensource / DDN)
- GPFS (IBM)
- BeeGFS

Lustre Design: Object Storage

- data is:
  - stored within objects (not within files)
  - data is referenced by objectids and pointers

- striping:
  - files can be composed of multiple objects

Lustre design: Components

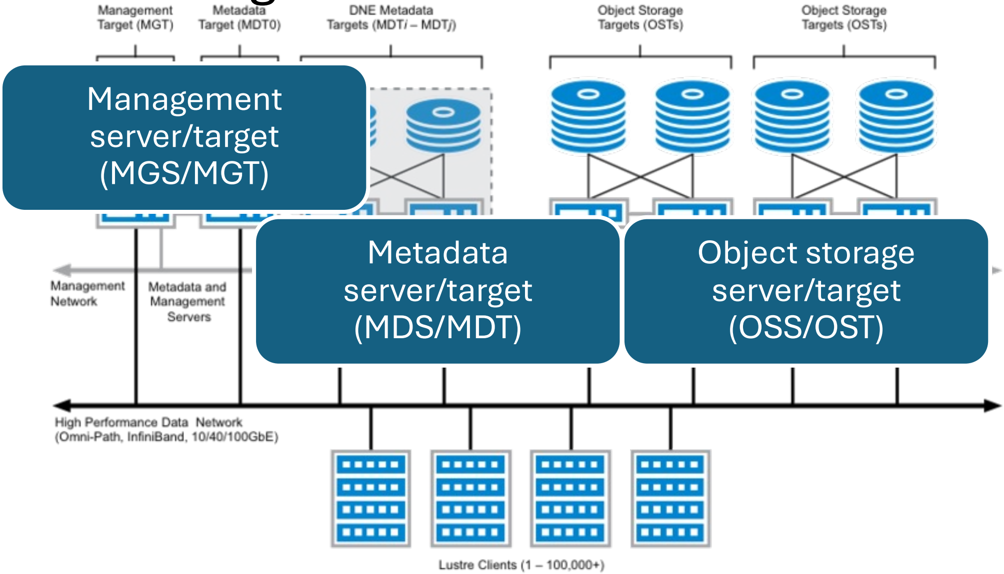
Management server/target (MGS/MGT)
- Central configuration, mounting on clients, locking

Metadata server/target (MDS/MDT)
- Translates files/directories to object ids
- Takes care of metadata information that is usually placed in an inode

Object storage server/target (OSS/OST)
- Stores object data on medium
- Data transfer to clients

add if required

Lustre design: Architecture



Physical storage in Lustre

- object based data is stored on „classical filesystems“:
  - ldiskfs (lustre disk filesystem)
  - ZFS (Zettabyte File System)

Example: Reading bulk data from Lustre

- User:
  `cat /my/file`

- Lustre client:
  - checks access rights for user at the MDS
  - gets objectids for file from the MDS
  - read objects from the respective OSTs

Example:

```
$ lfs getstripe testfile
testfile
lmm_stripe_count:  1
lmm_stripe_size:   1048576
lmm_pattern:       raid0
lmm_layout_gen:    0
lmm_stripe_offset: 40
      obdidx       objid       objid       group
          40     39776410      0x25ef09a                  0
```

Example:

```
$ lfs getstripe testfile
testfile
lmm_stripe_count:  4
lmm_stripe_size:   1048576
lmm_pattern:       raid0
lmm_layout_gen:    0
lmm_stripe_offset: 3
      obdidx       objid        objid       group
           3   110792105      0x69a8da9                 0
          20   107618569      0x66a2109                 0
          33    47100903      0x2ceb3e7                 0
          15   107404529      0x666dcf1                 0
```

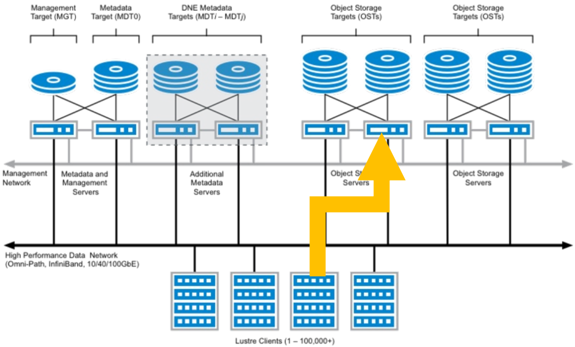
4GB testfile, default stripe layout

```
$ lfs getstripe testfile
testfile
  lcm_layout_gen:    3
  lcm_mirror_count:  1
  lcm_entry_count:   2
    lcme_id:             1
    lcme_mirror_id:      0
    lcme_flags:          init
    lcme_extent.e_start: 0
    lcme_extent.e_end:   1073741824
      lmm_stripe_count:  1
      lmm_stripe_size:   1048576
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 21
      lmm_objects:
      - 0: { l_ost_idx: 21, l_fid:
[0x100150000:0x5bc4615:0x0] }
```

```
    lcme_id:             2
    lcme_mirror_id:      0
    lcme_flags:          init
    lcme_extent.e_start: 1073741824
    lcme_extent.e_end:   EOF
      lmm_stripe_count:  4
      lmm_stripe_size:   1048576
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 9
      lmm_objects:
      - 0: { l_ost_idx: 9, l_fid:
[0x100090000:0x5e38725:0x0] }
      - 1: { l_ost_idx: 33, l_fid:
[0x100210000:0x1ca7418:0x0] }
      - 2: { l_ost_idx: 27, l_fid:
[0x1001b0000:0x6453a4e:0x0] }
      - 3: { l_ost_idx: 11, l_fid:
[0x1000b0000:0x5d1b9ea:0x0] }
```

Best practice

In Short: minimize meta data requests and read continuously.



Best practice 1/2

User behaviour resulting in slow Lustre:

- file size NOT on MDT
- getting the file size involves requests to all associated objects about their size
- listing a directory (eg `ls -als`) does this for every file
- one OST unavailable → process hangs

Best practice 2/2

- Solution:
  - Use of `ls` without parameters (check for aliases!)
  - `lazystatfs`: inaccurate file size saved on MDT

activated per default

- extended attributes
  - still saved on OSTs (also with "lazy file size" parameter)
  - are read from OSTs
  - avoid requests
- Data on metadata

Overview Mogon2 / Himster 2 File Servers

- /project 10 OSS, 44 OSTs, 5.4 PB
- /atlas 8 OSS, 24 OSTs, 2.2 PB
- /scratch 4 OSS, 12 OSTs, 1.1 PB
- /miifs04 4 OSS, 16 OSTs, 2.5 PB
- /miifs05 2 OSS, 8 OSTs, 0.75 PB
- Each OST consists of 14+1 disks in a ZFS raidz2

New:

- /"mogon3_lustre" 40 OST, 18.5 PB
- replacement for /miifs04 and /miifs05, 6-8TB useable
  - under construction

Hands on

- 4 examples with different IO patterns
- IO analysis with Darshan

- What to do:
  - login to MOGON 2 / Himster 2
  - go to `/lustre/project/m2_himkurs`
  - read `README`
  - Try to identify the problem behaviour in each example

- Don’t hesitate to ask for help

Solutions and discussion

Example A

- Straightforward blockwise IO
- 36 seconds vs 63 seconds (results vary! do statistics?)
- 1M vs 4k blocksize

Lesson learned

→ Small reads cause a lot of overhead

→ Try to increase the write size to ~1MB when possible

Example B

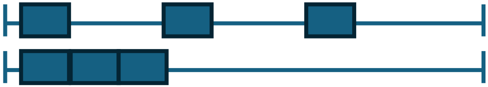
- 10000 writes to the same file
- B_0 opens the file, writes to it, closes it
- B_1 keeps the file open between writes
- ~40x faster
- Only one client involved, even worse if locking needs to be managed between clients

Lesson learned

→ Economically use open/close at all times

Example C

- 1M 4k reads
- Both read from start to end
- C_0 has gaps in between
- C_1 reads continuously



Lesson learned

→ No read ahead mechanism

→ Every lseek causes another IO operation in Lustre, no streaming IO

→ Try to read continuously when possible

Example D

- 50000 4k writes to the same file
- file is overwritten each time

- D_0 opens the file every time and truncates it (O_TRUNC)
- D_1 writes a buffer, resets the pointer to the start and writes the buffer again
- D_0 ~560x slower, due to open/close (compare with example B)
- O_TRUNC syncs files on Lustre (blocks until the changes are committed to disk)

Lessons learned

→ Avoid O_TRUNC whenever possible

→ use `/localscratch` if you need to constantly overwrite data