Jørn Dietze (UiT/LUST)
21.9.2023

Introduction to Lustre and Best Practices

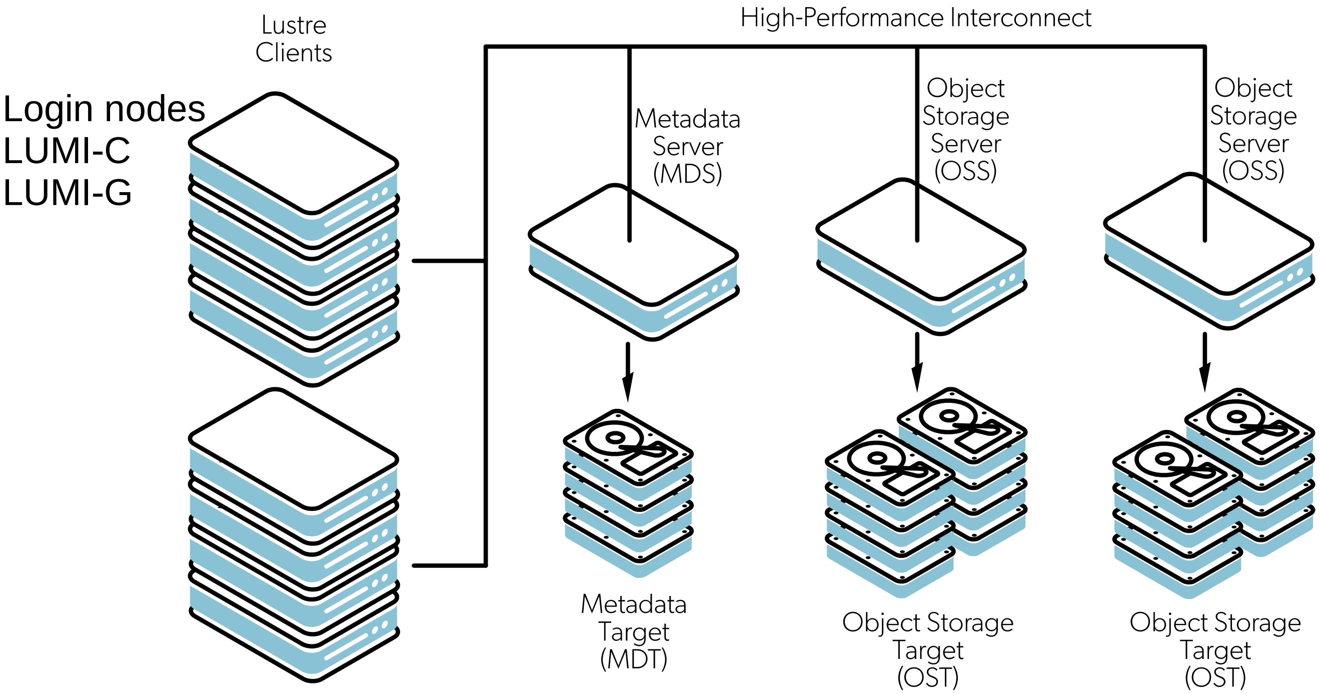
Lustre

LUMI has a highly parallel load

- Large amounts of data
- Compute and login nodes need access to storage
- Often multiple nodes require simultaneous read or write access to same data
- Danger of data corruption

Parallel file system to handle load

Lustre consists of 3 major functional units

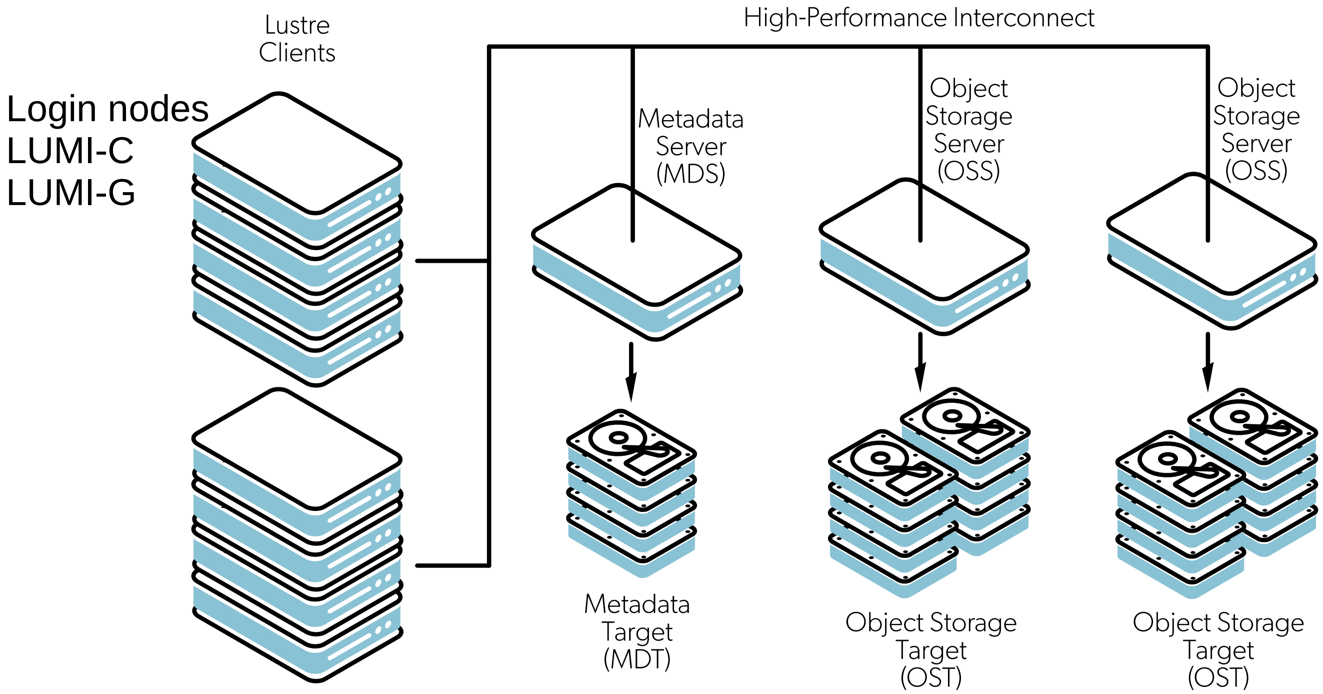


What steps happen when a file is accessed?

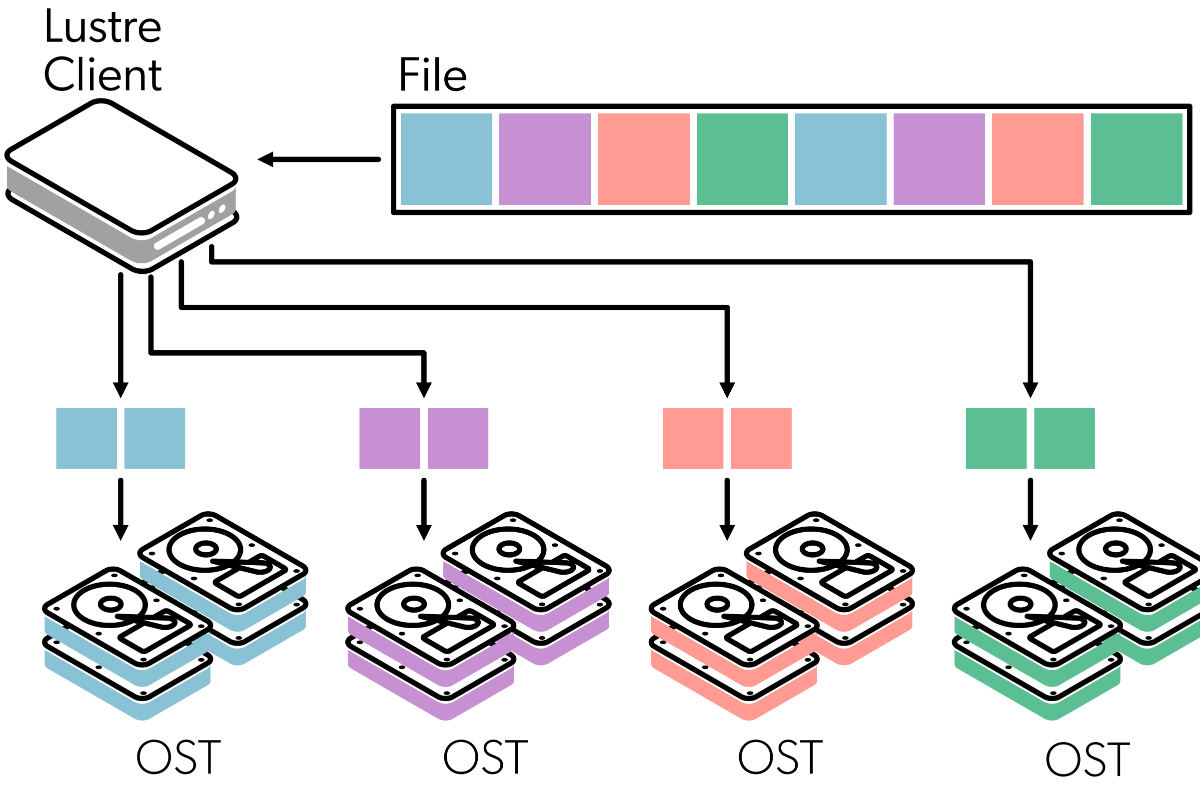
Client e.g. compute node wants read file

1. Client queries **Metadata Server** (MDS) for file
2. MDS returns location and layout
3. Client uses striping information to determine which **Object Storage Target** (OST) has which part of the file
4. Client requests file content from OSTs via **Object Storage Server** (OSS)
5. Data integrity it checked by client with checksums from OST

Lustre consists of 3 major functional units



Files are spread across multiple OSTs



Striping behavior can be adapted

Different tools to setting and displaying stripe properties

- `lfs setstripe` Set striping properties of a directory or new file
- `lfs getstripe` Return information on current striping settings
- `lfs df -h` Show disk usage of file system

Striping count and size are most important

Count: Number of OSTs to stripe over (0 default, -1 all)

- # files > # OSTs —> Set stripe_count = 1
  Reduce lustre contention and OST file locking and gain performance
- # files == 1 —> Set stripe_count = #OSTs or a number where your performance plateaus
  - Assuming you have more than 1 I/O client
- # files < # OSTs —> Select stripe_count so that you use all OSTs
  For example you write 8 files at the same time and have 32 OSTs, then select stripe_count=4

➡ Try to use all OSTs

Striping count and size are most important

Size: Bytes on each OST (0 filesystem default)

- No effect if stripe count is 1
- For large files
  - smallest recommended stripe size is 512 KB.
  - good stripe size is between 1 MB and 4 MB in most situations.
  - maximum stripe size is 4 GB but you should only use this value for very large files

Striping has to be set before file is created

```
jodietze@uan01:~> ls lfs.test
ls: cannot access 'lfs.test': No such file or
directory
jodietze@uan01:~> lfs setstripe -c 4 -S 2m lfs.test
jodietze@uan01:~> lfs getstripe lfs.test
lfs.test
lmm_stripe_count:  4
lmm_stripe_size:   2097152
lmm_pattern:       raid0
lmm_layout_gen:    0
lmm_stripe_offset: 10
obdidx    objid    objid    group
    10        110905348       0x69c4804             0
    12        110883990       0x69bf496             0
    14        110883882       0x69bf42a             0
    16        110888976       0x69c0810             0
```

Striping has to be set before file is created

```
jodietze@uan01:~> lfs setstripe -c 1 -S 1m lfs.test

lfs setstripe: setstripe error for 'lfs.test': stripe already set
```

Lustre is shared and finite

Metadata Storage Servers and Targets

- Are involved in many filesystem operations like creating, open, closing files
- Also queried everytime file attributes are looked up (e.g. with `stat` or `ls -l`)
- Are limited and can become bottleneck

For reading and writing OST are directly contacted

Some lustre **performance** tips

- Avoid stat() calls
- Open files read-only if that is the intention
- Read on rank-0 and broadcast instead of reading small files from every task
- Avoid very large directories
- Avoid appending to a file from many nodes (clients)

Many small files can be problem

- Slowdowns can occur when many (small) files are being opened
- Usually not restricted by bandwidth or actual file access latency
- But MDS is being flooded with request for files
- Especially installations and compilations can create hundreds of thousands of files
- Use archives or containers which are unpacked on compute node
- Special `lumi-container-wrapper` or `cotainr` for pip or conda environments

Storage on LUMI

LUMI has two storage systems

LUMI-P

- Disk based
- 4 independ Lustre file systems with each 20 PB
- Aggregated 240 Gb/s bandwidth

LUMI-F

- Solid-state (flash) based
- 8.5 PB
- 1740 GB/s bandwidth

LUMI has **four** storage areas

Area	Path	Quota	Files	Rention time
User home	`/users/<username>`	20 GB	100k	User lifetime
Project persistent	`/project/<project>`	50 GB	100k	Proj lifetime
Project scratch	`/scratch/<project>`	50 TB	2000k	90 days
Project flash	`/flash/<project>`	2 TB	100k	30 days

+ LUMI-O (object storage)

Weird errors → check your quota

Use `lumi-workspaces` to

- check for quota (file and size)
- see on which file system your home and project is located

Weird errors → check your quota

```
jodietze@uan02:~> lumi-workspaces
Quota for your projects:
Disk area                              Capacity(used/max)  Files(used/max)
----------------------------------------------------------------------
Personal home folder
Home folder is hosted on lustrep2
/users/jodietze                                  1,7G/22G         43K/100K
----------------------------------------------------------------------
Project: project_465000005
Project is hosted on lustrep2
/projappl/project_465000005                      4,1K/54G           1/100K
/scratch/project_465000005                       3,8G/55T          72/2,0M
/flash/project_465000005                         4,1K/2,2T          1/1,0M
----------------------------------------------------------------------
```

Temporary storage /tmp

- Compute nodes don’t have local disks/flash
- `/tmp` resides in memory
- Consumes space of your memory allocation
- Remember to allocate enough memory if you want to use `/tmp`

Conclusion

- Lustre achieves high performance through parallelism
  - Lots of bandwidth if used correctly
  - Metadata server can be a bottleneck
  - Striping options to optimize performance
  - Avoid large number of files
- LUMI has 4 storage areas with different quotas and lifetimes
- Check your quota with `lumi-workspaces`

LUMI

Jørn Dietze
LUMI User Support Team

jorn.dietze@uit.no

Follow us

Twitter: @LUMIhpc

LinkedIn: LUMI supercomputer

YouTube: LUMI supercomputer

www.lumi-supercomputer.eu

contact@lumi-supercomputer.eu

EuroHPC Joint Undertaking

The acquisition and operation of the EuroHPC supercomputer is funded jointly by the EuroHPC Joint Undertaking, through the European Union's Connecting Europe Facility and the Horizon 2020 research and innovation programme, as well as the of Participating States FI, BE, CH, CZ, DK, EE, IS, NO, PL, SE.

Leverage from the EU 2014–2020

European Union European Regional Development Fund

Kainuun liitto