

LUMI



Files on LUMI: Using Lustre

Emanuele Vitali
LUMI User Support Team (LUST)
CSC

April 2026
Slides originally authored by Kurt Lust
(LUST, UAntwerp)

File systems on LUMI

- HPC since the second half of the 1980s has mostly been about trying to build a fast system from relatively cheap hardware and cleverly written software.
 - The Lustre parallel file system fits in that way of thinking:
 - Link several regular servers
 - with a good network to the compute resources
 - to build a single system with a lot of storage capacity and a lot of bandwidth
 - (though unfortunately not all IOPS - number of I/O operations - scale as nicely).
 - And it is the main file system on large HPE Cray systems.
- HPE Cray EX systems go one step further:
 - Lustre is the only network file system on the compute nodes.
 - as part of the measures taken to minimise OS jitter and reduce node memory use.

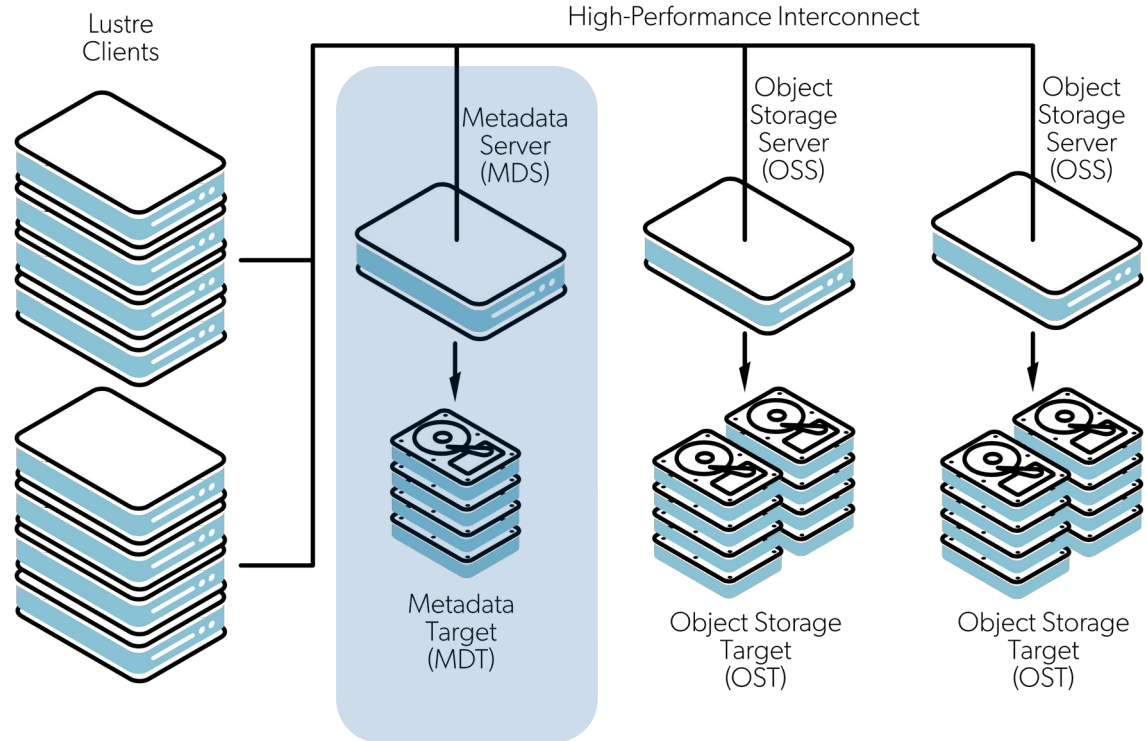
Lustre building blocks

Key element: Separation of data and metadata

L U M I

Lustre building blocks

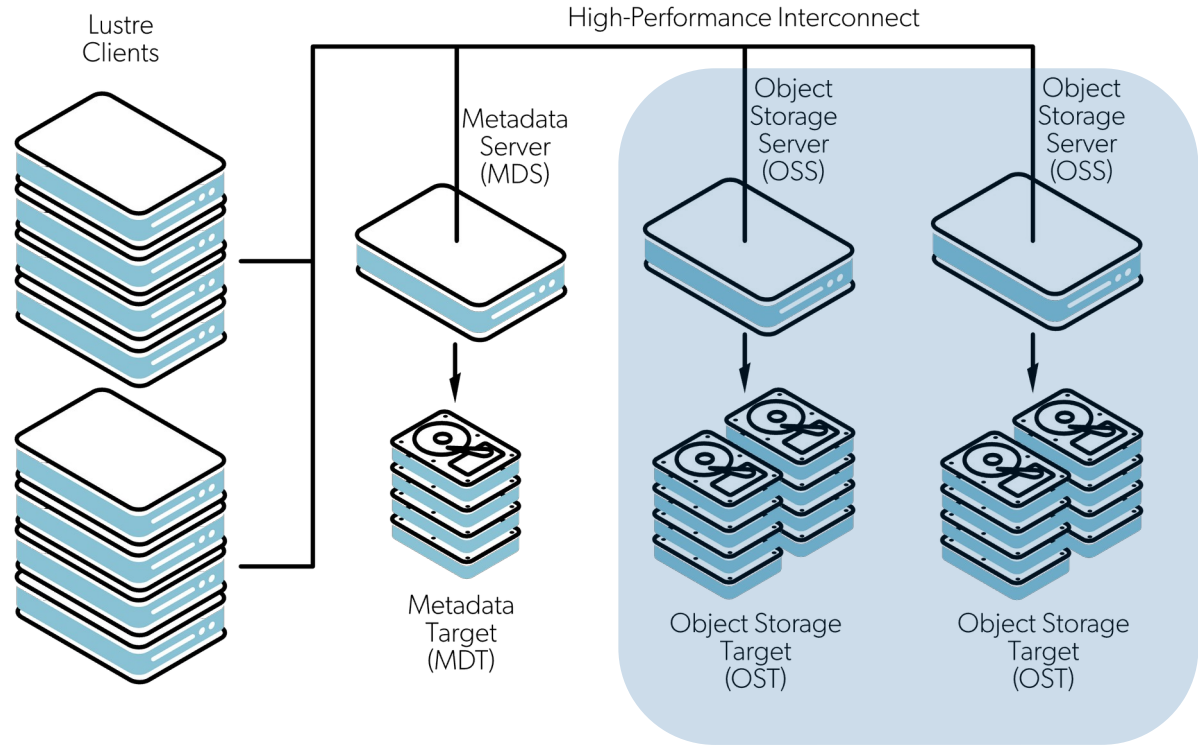
Metadata servers (MDSes) with one or more metadata targets (MDTs) each store namespace metadata (filename, access permissions, ...) and file layout.



Lustre building blocks

L U M I

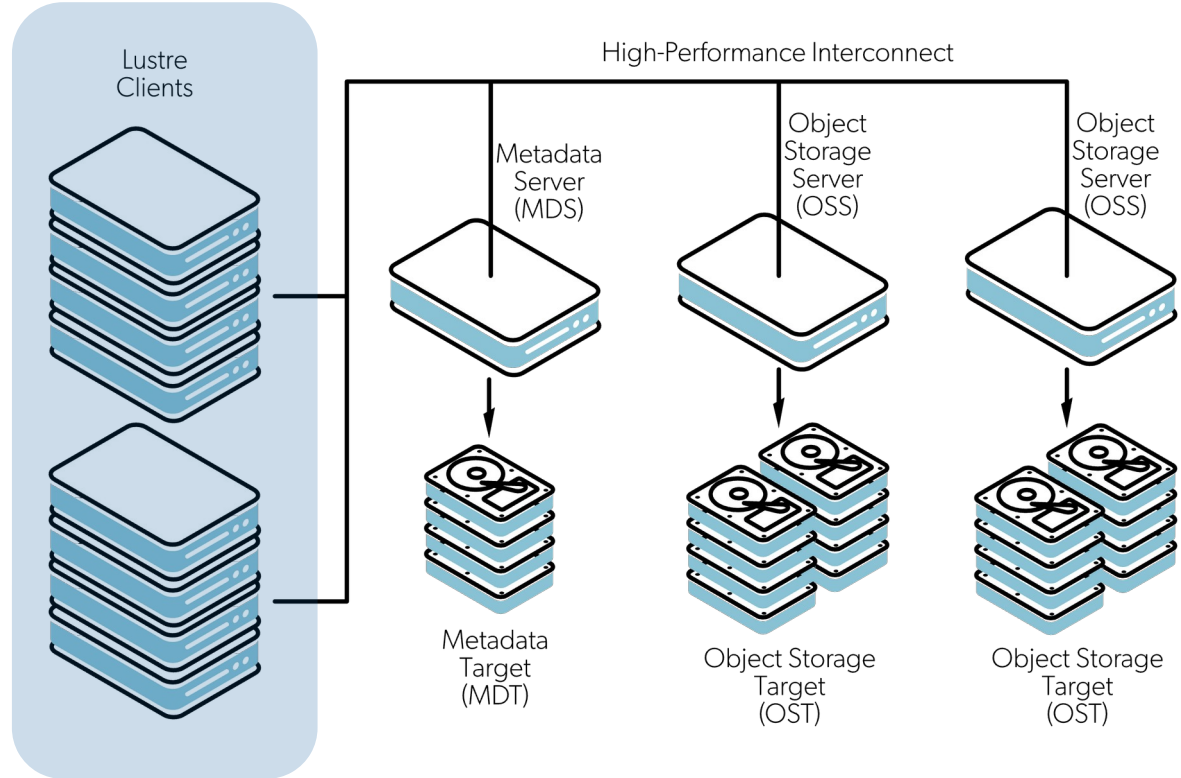
Object storage servers (OSSes) with one or more object storage targets (OSTs) each store the actual data.



Lustre building blocks

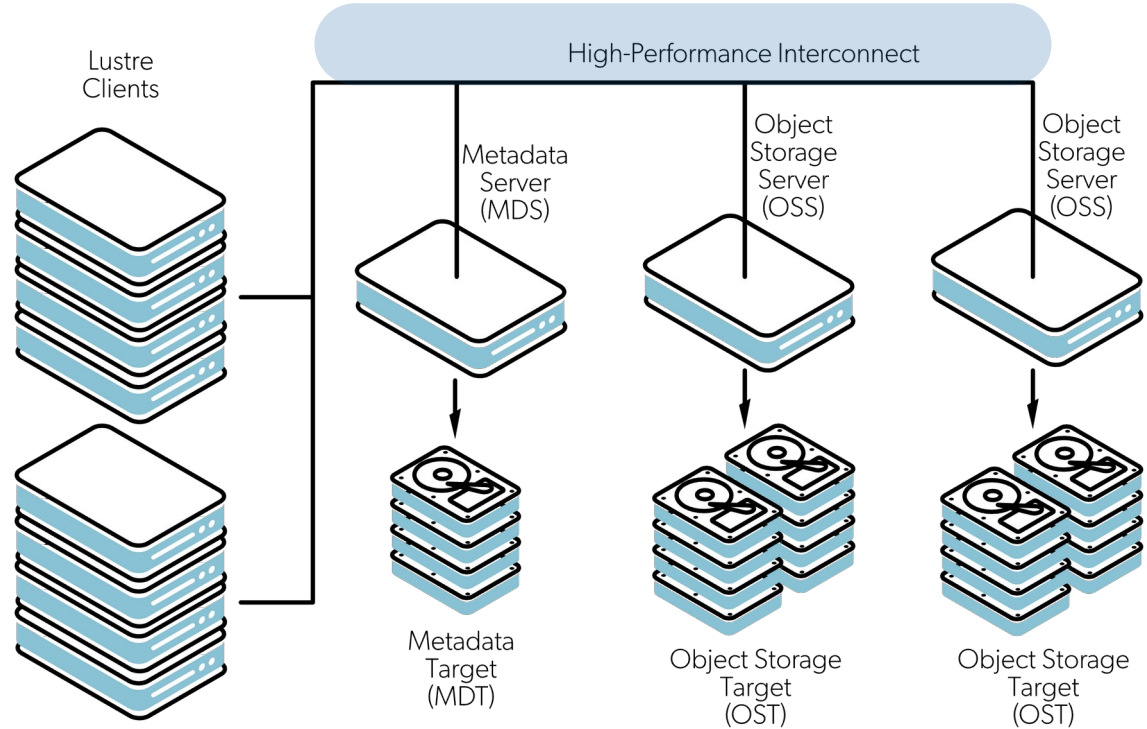
L U M I

Lustre clients that access and use the data and make the whole Lustre setup look like a single large file system



Lustre building blocks

L U M I



High-performance interconnect between all pieces of the storage system

Lustre building blocks (2)

- Lustre separates data and metadata as both are used differently
- Metadata servers (MDSes) with one or more metadata targets (MDTs) each store namespace metadata (filename, access permissions, ...) and file layout.
- Object storage servers (OSSes) with one or more object storage targets (OSTs) each store the actual data.
 - Capacity of Lustre is the sum of the capacity of the OSTs
- Lustre clients that access and use the data and makes the whole Lustre setup look like a single large file system
 - Transparent in functionality: You can use it as any regular Linux file system
 - But not transparent in performance: How you use Lustre can have a huge impact on performance
- All linked together through the high performance interconnect.

Lustre building blocks (2)

- Lustre separates data and metadata as both are used differently
- Metadata servers (MDSes) with one or more metadata targets (MDTs) each store namespace metadata (filename, access permissions, ...) and file layout.
- Object storage servers (OSSes) with one or more object storage targets (OSTs) each store the actual data.
 - Capacity of Lustre is the sum of the capacity of the OSTs
- Lustre clients that access and use the data and makes the whole Lustre setup look like a single large file system
 - Transparent in functionality: You can use it as any regular Linux file system
 - But not transparent in performance: How you use Lustre can have a huge impact on performance
- All linked together through the high performance interconnect.

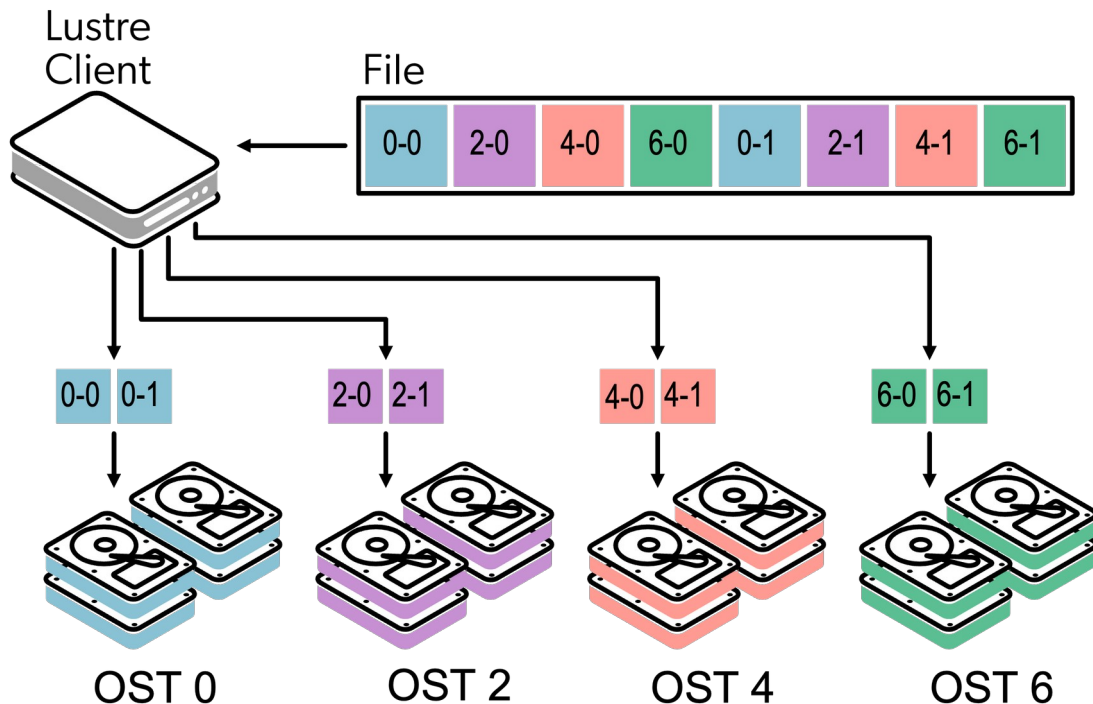
Lustre building blocks (2)

- Lustre separates data and metadata as both are used differently
- Metadata servers (MDSes) with one or more metadata targets (MDTs) each store namespace metadata (filename, access permissions, ...) and file layout.
- Object storage servers (OSSes) with one or more object storage targets (OSTs) each store the actual data.
 - Capacity of Lustre is the sum of the capacity of the OSTs
- Lustre clients that access and use the data and makes the whole Lustre setup look like a single large file system
 - Transparent in functionality: You can use it as any regular Linux file system
 - But not transparent in performance: How you use Lustre can have a huge impact on performance
- All linked together through the high performance interconnect.

Striping: Large files spread across OSTs

LUMI

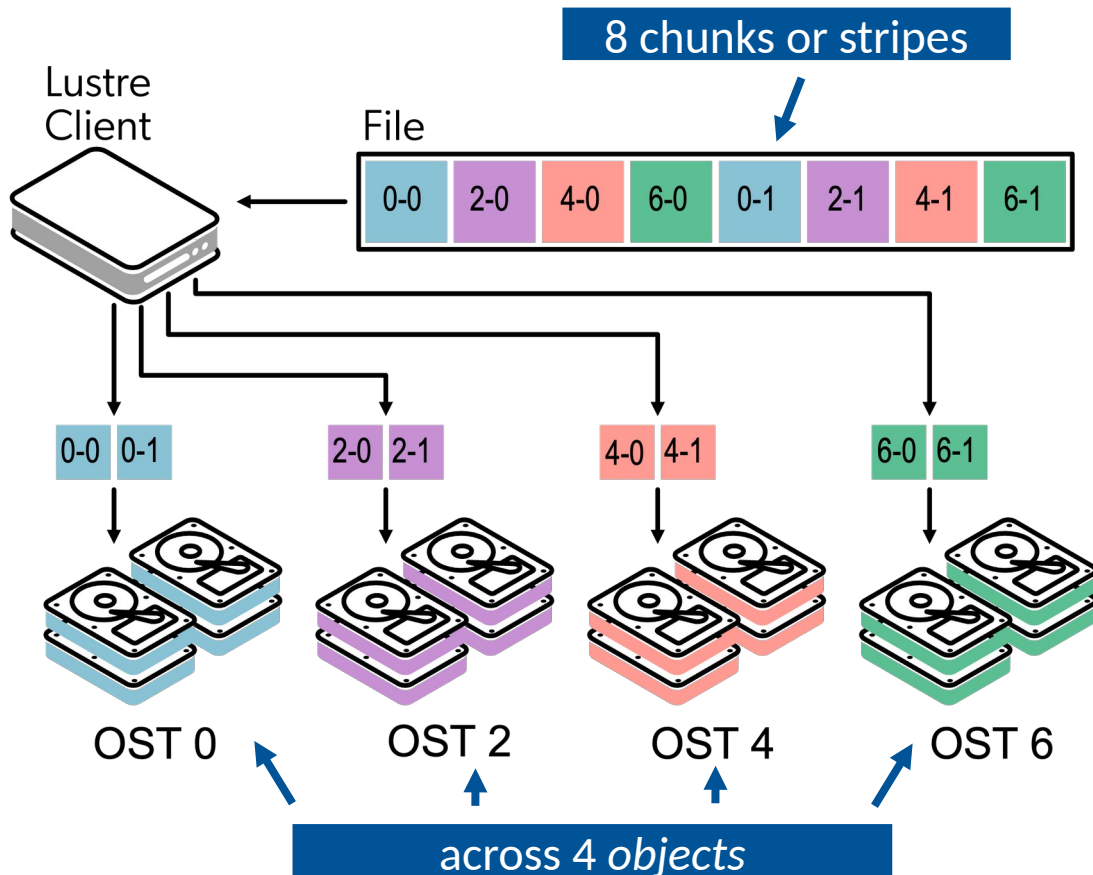
- Files broken in chunks/stripes, distributed cyclically across a number of chunk files/objects, each on a separate OST
- Transparent to the user with respect to correctness
- But large impact on performance
- 2 parameters:
 - Size of the stripes
 - Number of OSTs
- Default on LUMI is to use only one OST



Striping: Large files spread across OSTs

LUMI

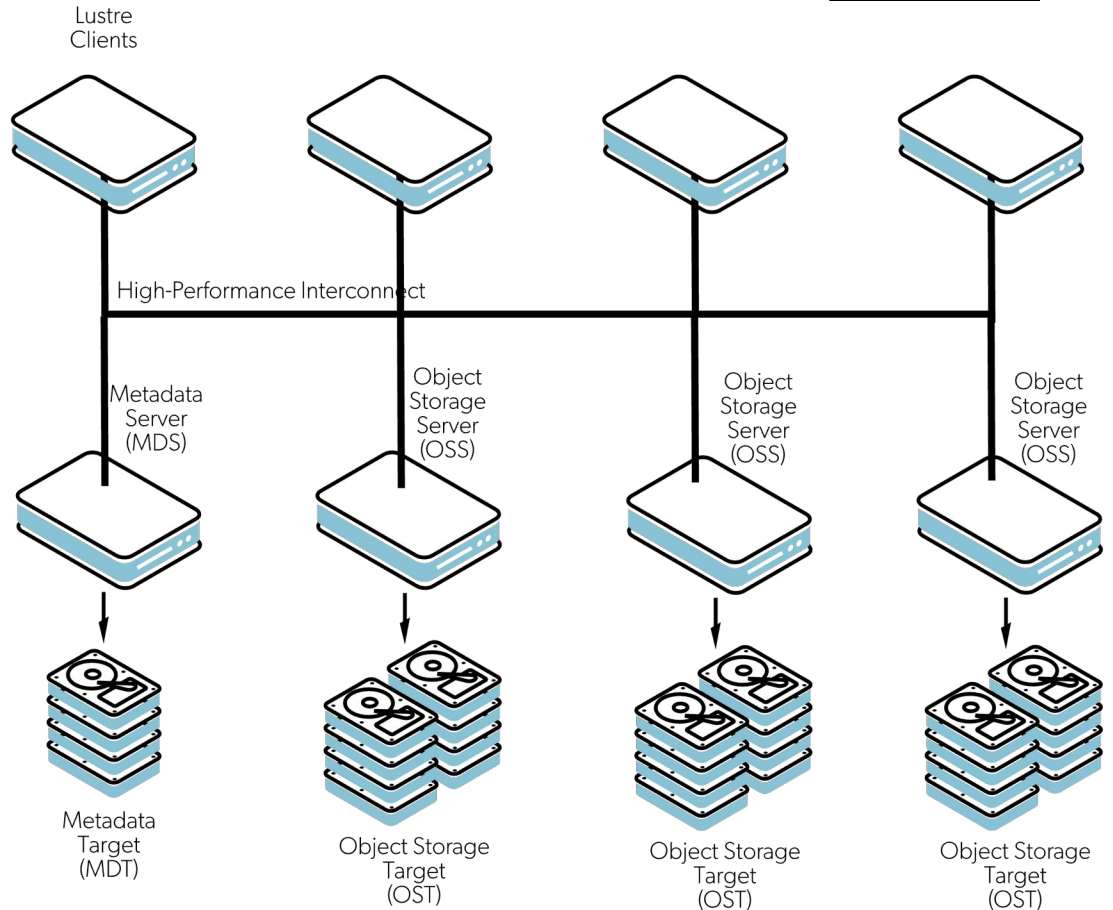
- Files broken in chunks/stripes, distributed cyclically across a number of chunk files/objects, each on a separate OST
- Transparent to the user with respect to correctness
- But large impact on performance
- 2 parameters:
 - Size of the stripes
 - Number of OSTs
- Default on LUMI is to use only one OST



Accessing a file

```
open(unit=12, file="out.dat")
```

L U M I

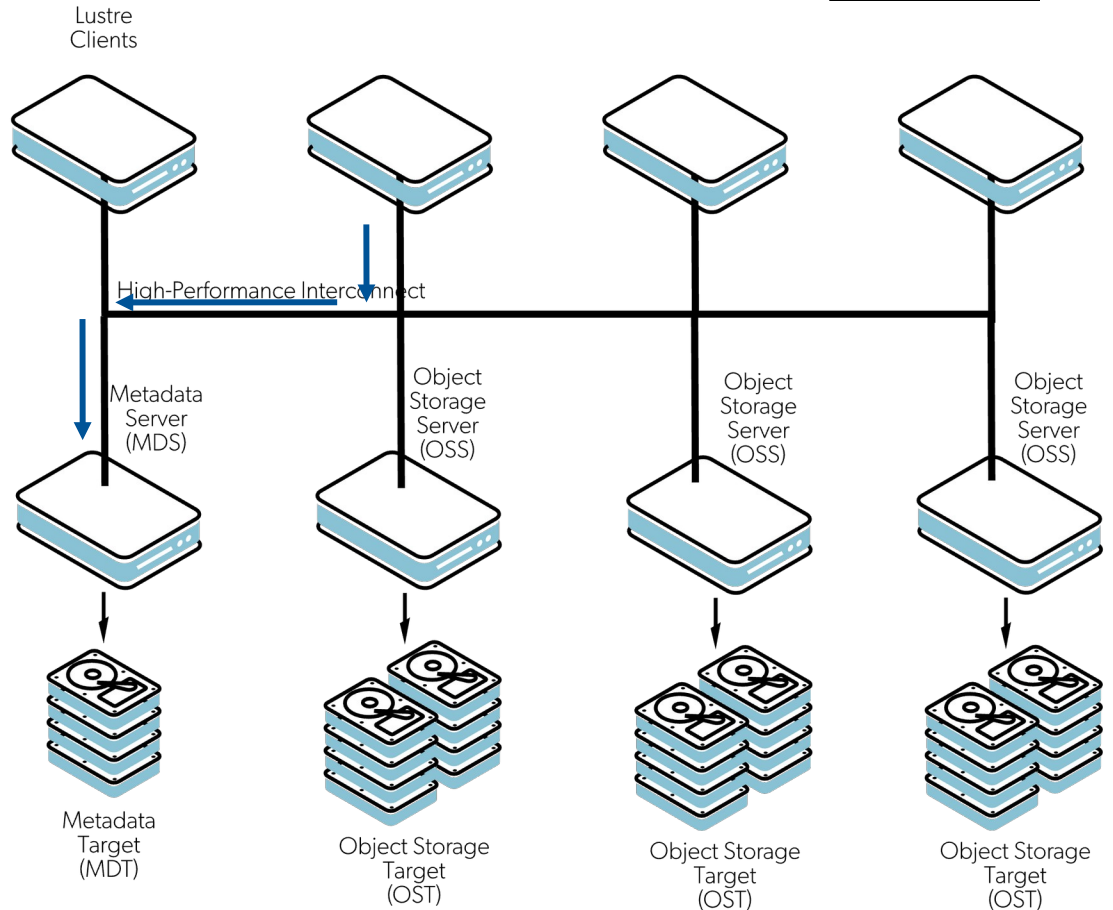


Accessing a file

Client queries MDS

```
open(unit=12, file="out.dat")
```

L U M I

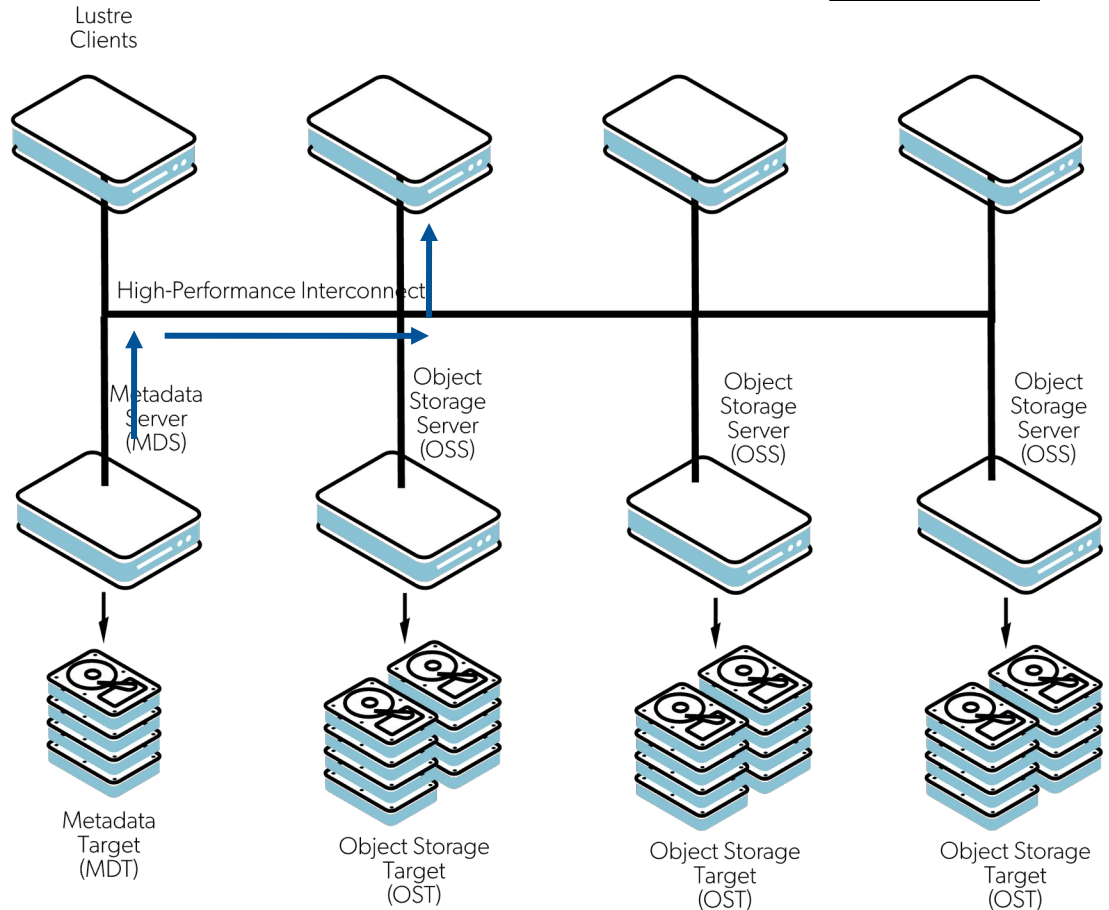


Accessing a file

```
open(unit=12, file="out.dat")
```

L U M I

MDS returns layout/location

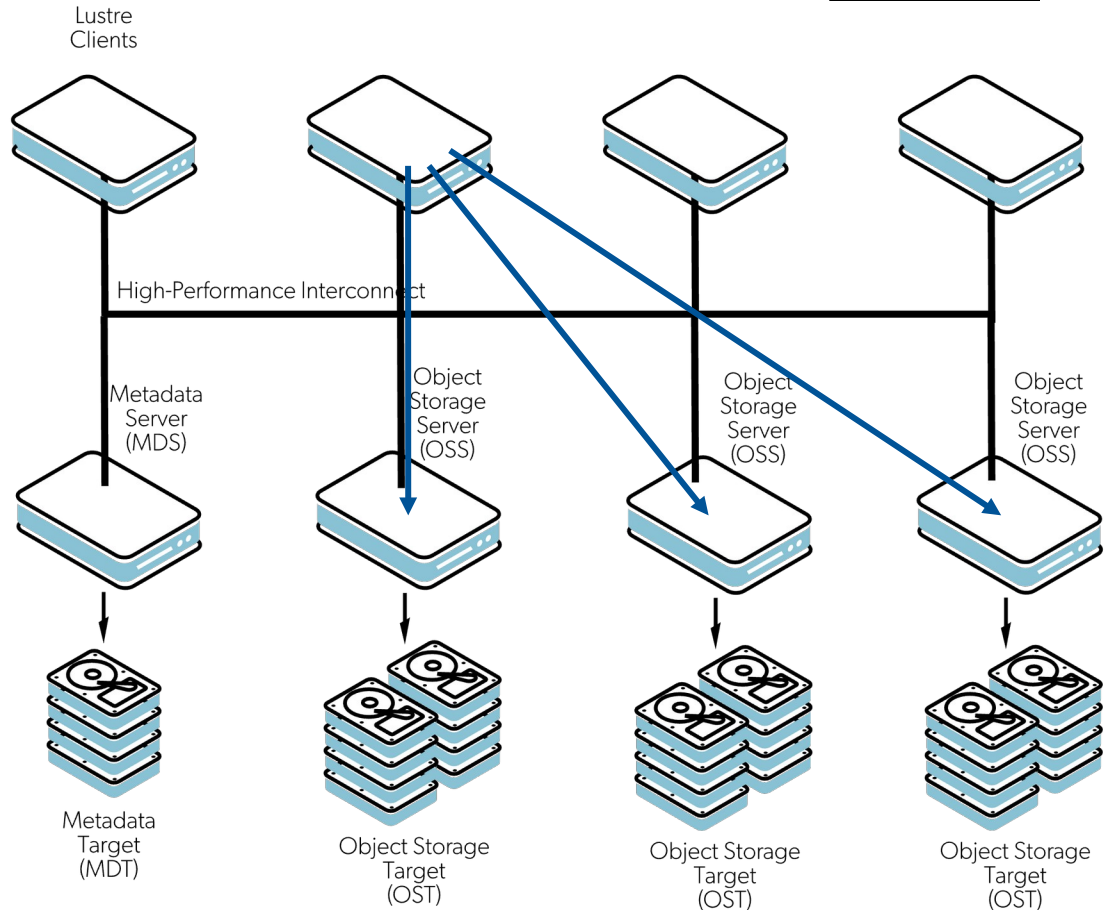


Accessing a file

`write(12,*) data`

L U M I

Subsequent read or write calls can talk directly to all OSSes involved



Parallelism is key!

- MDS access can be problematic
 - Difficult to spread across multiple MDSEs
 - Small accesses, so each MDS doesn't really exploit parallelism in RAID either
- But up to four levels of parallelism in reads and writes
 - Engage multiple OSSes
 - Which can each engage multiple OSTs
 - That typically engage multiple disks in a RAID setup for reliability
 - For an SSD file system: Modern SSDs are also highly parallel
- So large I/O operations needed
 - Very small I/O operations won't even benefit from RAID acceleration
 - Relatively large stripe size for more efficient I/O at the OST level (especially for hard drives)
 - And even larger I/O operations needed to engage enough OSTs (but that access can come from multiple nodes in the process)

Parallelism is key! (2)

- 😊 HPC file formats such as HDF5 and netCDF
 - When used properly, very good bandwidth possible
 - Old codes can be very good. But their authors have known floppy drives...

Parallelism is key! (2)

- 😊 HPC file formats such as HDF5 and netCDF
 - When used properly, very good bandwidth possible
 - Old codes can be very good. But their authors have known floppy drives...
- 😞 Codes that open one or more files per MPI rank
 - Won't scale to large numbers of ranks
 - Disaster for MDS as files will be opened more or less simultaneously
 - Potential disaster for ODS also as each ODS will serve many files with writes or reads coming in simultaneously
 - Also in old codes that were never meant to scale to 1000s or cores

Parallelism is key! (2)

- 😊 HPC file formats such as HDF5 and netCDF
 - When used properly, very good bandwidth possible
 - Old codes can be very good. But their authors have known floppy drives...
- 😞 Codes that open one or more files per MPI rank
 - Won't scale to large numbers of ranks
 - Disaster for MDS as files will be opened more or less simultaneously
 - Potential disaster for ODS also as each ODS will serve many files with writes or reads coming in simultaneously
 - Also in old codes that were never meant to scale to 1000s or cores
- 😞😞 Abuse the file system as an unstructured database by dumping data in thousands or millions of small files with each one data element
 - Local SSD not really a solution as you "own" a node only shortly
 - A Python or conda software installation by itself is already an example

How to determine the striping value?

- Small files accessed sequentially: 🤔 🤔 🤔
- Try to use all OSTs without overloading them.
 - #files \geq #OSTs: stripe count 1 is best
 - #files = 1: Set the stripe count to #OSTs, or a smaller number if the performance plateaus (benchmarking needed!). The latter will happen if not enough Lustre clients are used simultaneously to access the file.
 - #files < #OSTs: Chose such that stripe count * #files = #OSTs.
E.g.: 32 OSTs and 8 files: Use a stripe count of 4.
- Let the system choose the OSTs, don't try to impose them.
- An ideal stripe size will usually be 1 MB or more.
Maximum value is 4 GB but that is only useful for very large files.

Managing the striping parameters (1)

- The basic command line tool to manage striping in lustre is the `lfs` command.
- Use `lfs df -h` to get information about the file systems

UUID	bytes	Used	Available	Use%	Mounted on
lustref1-MDT0000_UUID	11.8T	16.8G	11.6T	1%	/pfs/lustref1[MDT:0]
lustref1-MDT0001_UUID	11.8T	4.1G	11.6T	1%	/pfs/lustref1[MDT:1]
lustref1-MDT0002_UUID	11.8T	2.8G	11.7T	1%	/pfs/lustref1[MDT:2]
lustref1-MDT0003_UUID	11.8T	2.7G	11.7T	1%	/pfs/lustref1[MDT:3]
lustref1-OST0000_UUID	121.3T	21.7T	98.3T	19%	/pfs/lustref1[OST:0]
lustref1-OST0001_UUID	121.3T	21.8T	98.2T	19%	/pfs/lustref1[OST:1]
lustref1-OST0002_UUID	121.3T	21.7T	98.4T	19%	/pfs/lustref1[OST:2]

- A way to find the number of OSTs

Managing the striping parameters (2)

- Use `lfs getstripe` to check striping information at the directory or file level

```
$ lfs getstripe -d /appl/lumi/SW
```



Only show directory itself

Managing the striping parameters (2)

- Use `lfs getstripe` to check striping information at the directory or file level

```
$ lfs getstripe -d /appl/lumi/SW  
stripe_count: 1 stripe_size: 1048576 pattern: 0 stripe_offset: -1
```

Managing the striping parameters (2)

- Use `lfs getstripe` to check striping information at the directory or file level

```
$ lfs getstripe -d /appl/lumi/SW  
stripe_count: 1 stripe_size: 1048576 pattern: 0 stripe_offset: -1
```

```
$ lfs getstripe -d --raw /appl/lumi/SW  
stripe_count: 0 stripe_size: 0 pattern: 0 stripe_offset: -1
```

Actually the defaults for the file system

Let the MDS chose

Managing the striping parameters (2)

- Use `lfs getstripe` to check striping information at the directory or file level

```
$ lfs getstripe -d /appl/lumi/SW
stripe_count: 1 stripe_size: 1048576 pattern: 0 stripe_offset: -1
```

```
$ lfs getstripe -d --raw /appl/lumi/SW
stripe_count: 0 stripe_size: 0 pattern: 0 stripe_offset: -1
```

```
$ lfs getstripe /appl/lumi/LUMI-SoftwareStack/etc/motd.txt
/appl/lumi/LUMI-SoftwareStack/etc/motd.txt
```

```
lmm_stripe_count: 1
lmm_stripe_size: 1048576
lmm_pattern: raid0
lmm_layout_gen: 0
lmm_stripe_offset: 2
objid 2          objid 292319061      objid 0x116c6f55      group 0
```

OSTs for the file

Managing the striping parameters (3)

- Use `lfs setstripe` to set the striping information

```
$ module use /appl/local/training/modules/2day-20240502
```

```
$ module load lumi-training-tools
```

```
$ mkdir testdir
```

```
$ lfs setstripe -S 2m -c 4 testdir
```

```
$ cd testdir
```

```
$ mkfile 2g testfile1
```

```
$ lfs getstripe testfile1
```

```
testfile1
```

```
lmm_stripe_count: 4
```

```
lmm_stripe_size: 2097152
```

```
lmm_pattern: raid0
```

```
lmm_layout_gen: 0
```

```
lmm_stripe_offset: 28
```

obdidx	objid	objid	group
28	66250987	0x3f2e8eb	0
30	66282908	0x3f3659c	0
1	71789920	0x4476d60	0
5	71781120	0x4474b00	0

Default striping for this directory

Managing the striping parameters (3)

- Use `lfs setstripe` to set the striping information

```
$ module use /appl/local/training/modules/2day-20240502
```

```
$ module load lumi-training-tools
```

```
$ mkdir testdir
```

```
$ lfs setstripe -S 2m -c 4 testdir
```

Default striping for this directory

```
$ cd testdir
```

```
$ mkfile 2g testfile1
```

Tool to create a new file of given size (2G here)

```
$ lfs getstripe testfile1
```

```
testfile1
```

```
lmm_stripe_count: 4
```

```
lmm_stripe_size: 2097152
```

```
lmm_pattern: raid0
```

```
lmm_layout_gen: 0
```

```
lmm_stripe_offset: 28
```

obdidx	objid	objid	group
28	66250987	0x3f2e8eb	0
30	66282908	0x3f3659c	0
1	71789920	0x4476d60	0
5	71781120	0x4474b00	0

Managing the striping parameters (3)

- Use `lfs setstripe` to set the striping information

```
$ module use /appl/local/training/modules/2day-20240502
```

```
$ module load lumi-training-tools
```

```
$ mkdir testdir
```

```
$ lfs setstripe -S 2m -c 4 testdir
```

Default striping for this directory

```
$ cd testdir
```

```
$ mkfile 2g testfile1
```

Tool to create a new file of given size (2G here)

```
$ lfs getstripe testfile1
```

```
testfile1
```

```
lmm_stripe_count: 4
```

```
lmm_stripe_size: 2097152
```

```
lmm_pattern: raid0
```

```
lmm_layout_gen: 0
```

```
lmm_stripe_offset: 28
```

And we get the values that we set for the directory

obdidx	objid	objid	group
28	66250987	0x3f2e8eb	0
30	66282908	0x3f3659c	0
1	71789920	0x4476d60	0
5	71781120	0x4474b00	0

The 4 OSTs

Managing the striping parameters (4)

- Use `lfs setstripe` to set the striping information

```
$ lfs setstripe -S 16m -c 2 testfile2
```

Create an empty file with given striping

```
$ ls -lh
```

```
total 0
```

```
-rw-rw---- 1 XXXXXXXX project_462000000 2.0G Jan 15 16:17 testfile1
```

```
-rw-rw---- 1 XXXXXXXX project_462000000 0 Jan 15 16:23 testfile2
```

```
$ lfs getstripe testfile2
```

```
testfile2
```

```
lmm_stripe_count: 2
```

```
lmm_stripe_size: 16777216
```

And we get the values that we set for the file

```
lmm_pattern: raid0
```

```
lmm_layout_gen: 0
```

```
lmm_stripe_offset: 10
```

obdidx	objid	objid	group
10	71752411	0x446dadb	0
14	71812909	0x447c72d	0

The 2 OSTs

The metadata servers (1)

- Finite and shared resource
- Involved in many file system operations:
 - Create/open/close
 - Get attributes
 - Managing file locking
- Slow or variable filesystem performance when overstressed
 - Less than 200k operations per second, depending on operation type also!

The metadata servers (2)

- Important to be careful with what you do
 - E.g., `ls -l` is rather costly on Lustre
 - Access to many small files from many processes is not a good idea (think Python): Run from a container or move to `/tmp` (which will eat from your RAM). Use file formats as HDF5, ADIOS, ...
 - The filesystem is not a communication device for shuffling data between nodes
 - Avoid very large directories
 - Use `lfs find` instead of `find`
 - And many more tips for programmers...

Lustre on LUMI

LUMI

- LUMI-P:
 - 4 disk based storage systems
 - 18 PB capacity each
 - 240 GB/s aggregated bandwidth each
 - 2 MDTs (1 per MDS), 32 OSTs (2 per OSS)
 - Serves /users, /project and /scratch
- LUMI-F
 - Solid State Drive based storage system
 - 8.5 PB capacity
 - >2 TB/s aggregated bandwidth
 - 4 MDTs (1 per MDS) and 72 OSTs (1 per OSS)
 - Serves /flash

Questions?

