LUMI

Loading Training data on LUMI

Harvey Richardson, HPE
LUM AI Course, November 26-27 2024, Ostrava

Agenda

LUMI

- Storage on LUMI
- Lustre Filesystems
- LUMI-O
- Data access considerations

Storage on LUMI

- Parallel Lustre Filesystems (LUMI-P and LUMI-F)

	quota	Max-files	expandable	Backup	retention
User home	20GB	100k	No	No	User lifetime
Project persistent	50GB	100k	To 500GB	No	Project lifetime
Project scratch	50TB	2000k	To 500TB	No	Project lifetime
Project fast (flash)	2TB	1000k	To 100TB	No	Project lifetime

- Object Storage (LUMI-O)

	quota	Max buckets	Max objects-per-bucket	Backup	retention
Object Storage	150TB	1000	500000	No	Project lifetime

- /tmp (but need to have sufficient job memory request)

Storage on LUMI: filesystems

LUMI

LUMI-P/LUMI-F access

	Path	Intended use	Hardware Partition
User home	/users/<username>	User home directory for personal and configuration files	LUMI-P
Project persistent	/project/<project>	Project home directory for shared project files	LUMI-P
Project scratch	/scratch/<project>	Temporary storage for input, output or checkpoint data	LUMI-P
Project flash	/flash/<project>	High performance temporary storage for input and output data	LUMI-F

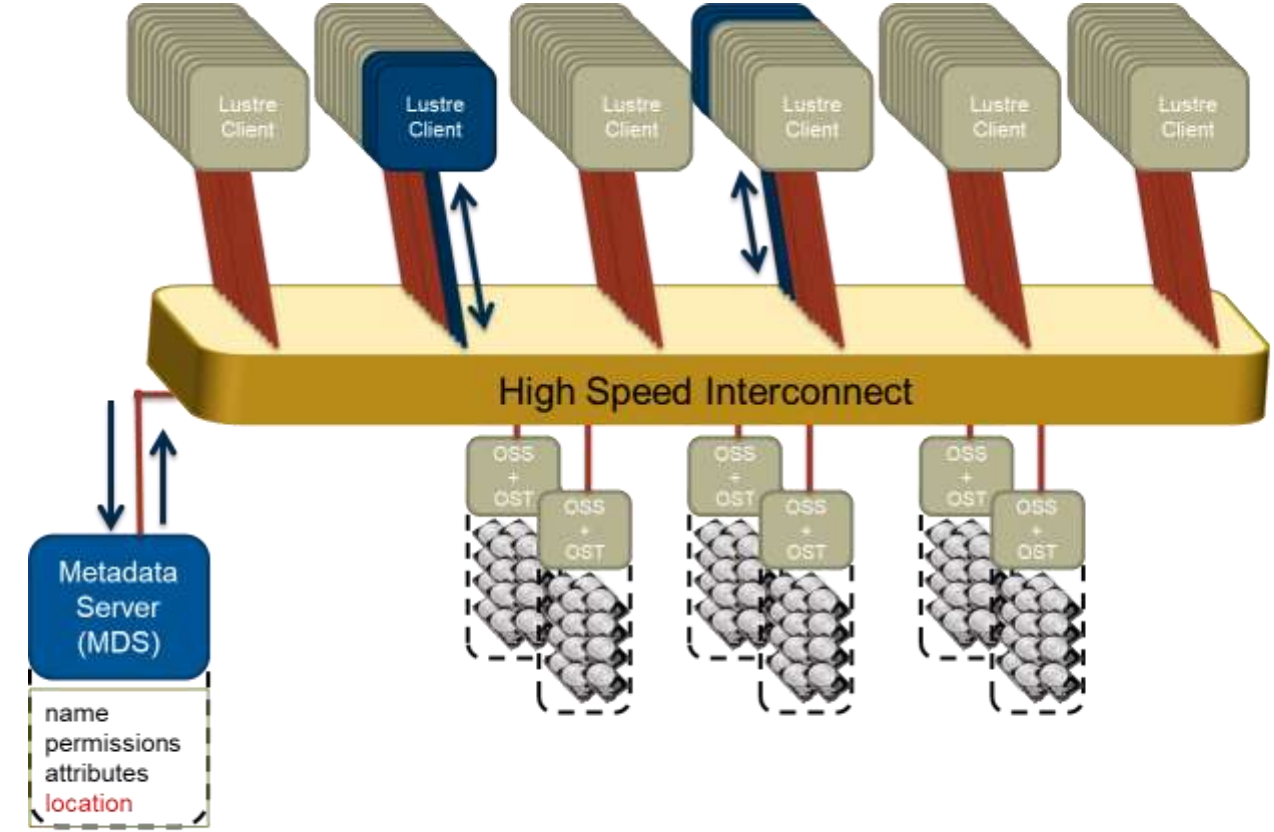
Run `lumi-workspaces` to see your specific locations

Lustre

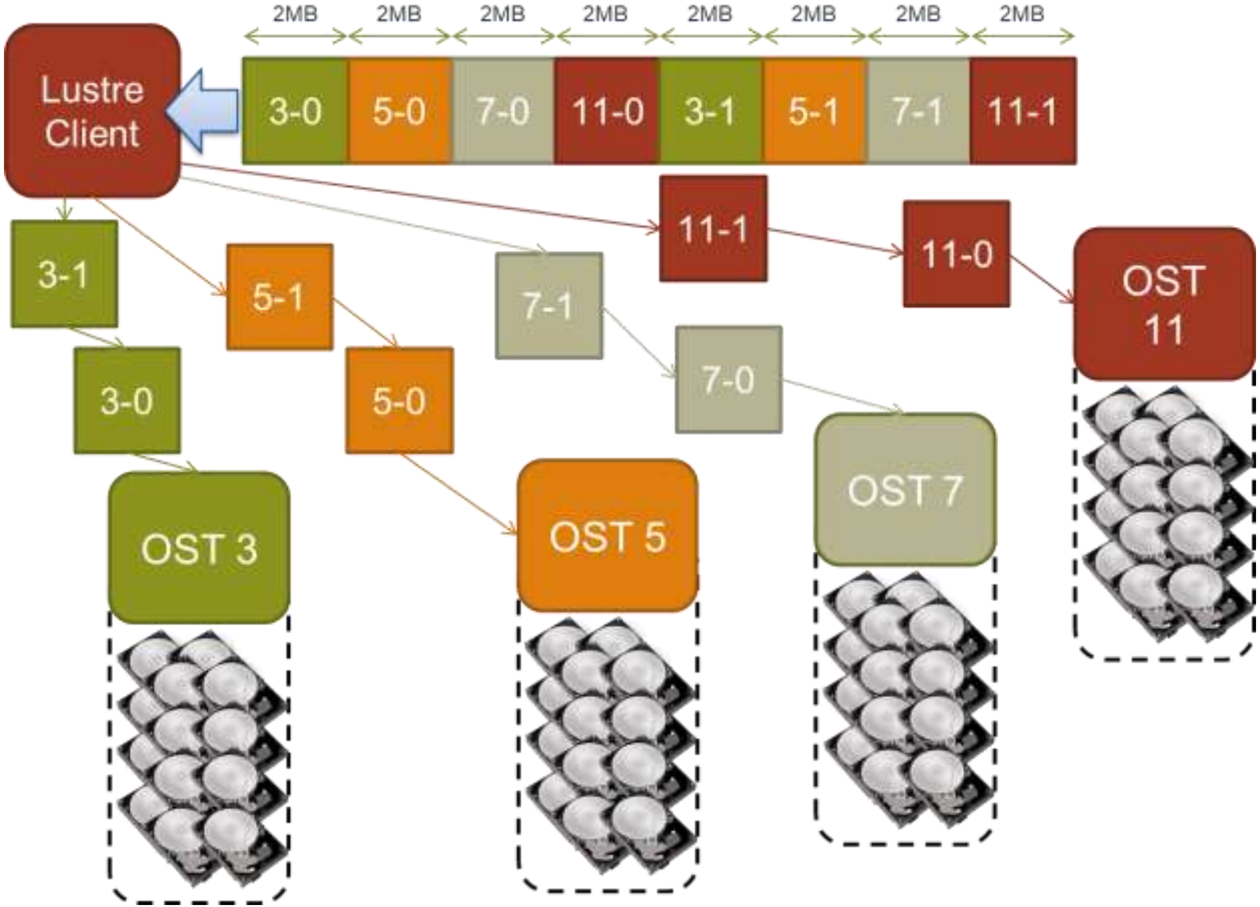
- Lustre is an open source parallel filesystem designed to support leadership class HPC systems
- Comprised of software subsystems, storage and associated network
  - Metadata servers (MDSs) providing metadata targets (MDTs) which store filesystem namespace information (directories, filenames, permissions etc.)
  - Object Storage Servers (OSSs) providing Object Storage Targets (OSTs) each hosting a local filesystem
  - Lustre clients (login nodes, compute nodes) access the global filesystem
- All clients see a unified namespace and the filesystem supports POSIX semantics providing concurrent coherent access to files.

Lustre components

One or more



File decomposition – 2MB stripes



Controlling striping with lfs setstripe

LUMI

- Sets the stripe for a file or a directory

```
lfs setstripe   <--stripe-size |-S size>
                      <--stripe-count|-c count> <file|dir>
```

  - size: Number of bytes on each OST (0 filesystem default)
  - count: Number of OSTs to stripe over (0 default, -1 all)

- Comments
  - Striping policy is set when the file is created. It is not possible to change it afterwards.
  - Can use lfs to create an empty file with the stripes you want (like the touch command)
  - Can apply striping settings to a directory, any children will inherit parent's stripe settings on creation.
  - There is an index option to choose the first OST, don't use this in normal circumstances.

Advice for striping settings

LUMI

- Selecting the striping values will have a large impact on the I/O performance of your application
- Rules of thumb: Try to use all OSTs
  - # files > # OSTs => Set stripe_count=1
    You will reduce the lustre contention and OST file locking this way 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
    Example : You have 8 OSTs and write 4 files at the same time, then select stripe_count=2

- Always allow the system to choose OSTs at random!

Lustre considerations

LUMI

- Lustre was designed for high performance streaming I/O for large amounts of data
- It will struggle with some usage patterns such as
  - Directories with huge number of files (reduce number, organize by client)
  - Small data transfers
- Python environments can be a challenge for Lustre, particularly if started in parallel on many nodes
  - Containerise them (LUMI tools can be used to help with this)
  - Possibly move into /tmp and run from there

LUMI-O Object Storage

LUMI

- Provides 30PB of storage for storing, sharing and staging data
- Supports private and public access
- Storage is object-based, you store objects in buckets you allocate…
  - **Buckets**: Containers used to store one or more objects. Object storage uses a flat structure with only one level which means that buckets cannot contain other buckets.
  - **Objects**: Any type of data. An object is stored in a bucket.
  - **Metadata**: Both buckets and objects have metadata specific to them. The metadata of a bucket specifies e.g., the access rights to the bucket. While traditional file systems have fixed metadata (filename, creation date, type, etc.), an object storage allows you to add custom metadata.

Accessing LUMI-O

LUMI

- Make commands available
  `module load lumio`
- To configure a connection to LUMI-O run
  `lumio-conf`
- Above command instructs you to go to https://auth.lumidata.eu/
- Follow instructions at: https://docs.lumi-supercomputer.eu/storage/lumio/auth-lumidata-eu/
  - Enter generated key into lumio-conf, it creates setup for rclone
  - Templates can be generated for shell, boto3, rclone, s3cmd, aws
- Keys have a lifetime so duration needs to outlast the workflow
  - For example move data from LUMI-O to scratch for job

Accessing LUMI-O

LUMI

- rclone and s3cmd can perform basic operations

Action	rclone comand	s3cmd command
List buckets	rclone lsd lumi-o:	s3cmd ls s3:
Create bucket *mybuck*	rclone mkdir lumi-o:mybuck	s3cmd mb s3://mybuck
List objects in bucket *mybuck*	rclone ls lumi-o:mybuck/	s3cmd ls --recursive s3://mybuck
Upload file *file1* to bucket *mybuck*	rclone copy file1 lumi-o:mybuck/	s3cmd put file1 s3://mybuck
Download file *file1* from bucket *mybuck*	rclone copy lumi-o:mybuck/file1 .	s3cmd get s3://mybuck/file1 .

- rclone and s3cmd can perform more complex operations (see manpages)

Endpoints for rclone and URL access

LUMI

- **lumi-o**: The private endpoint. The buckets and objects uploaded to this endpoint will not be publicly accessible.
- **lumi-pub**: The public endpoint.
  The buckets and objects uploaded to this endpoint will publicly accessible using the URL:

  https://<project-number>.lumidata.eu/<bucket_name>`

- Be careful to not upload data that cannot be public to lumi-pub

API access to LUMI-O

- LUMI-O can also be accessed via APIs such as boto3
  - For example to list buckets in project 465000001

```
import boto3


session =
   boto3.session.Session(profile_name='lumi-465000001')
s3_client = session.client('s3')
buckets=s3_client.list_buckets()
```

S3 client docs

Workflows

- As noted, LUMI has various filesystems and provides LUMI-O
- Most likely you will load data from the filesystems
- There are many APIs provided by languages, language modules and frameworks that you can use...

Considerations for data access

- ‘Containerise’ files in higher level formats (HDF5) — particularly for array-based data or images
- Use compressed file/image formats to save most on storage
- Use compact binary data formats
- User appropriate formats and loaders
  - csv, feather, parquet, jay, pickle; pandas, dask, datatables
  - Use multiple workers in data loaders (PyTorch Dataloader,… num_workers=…)
- Explore image loading libraries
  - (Python Imaging Library (PIL), pyspng, PyTurboJPEG
- Perhaps cache files in memory

A couple of perspectives on resolving bottlenecks

Solving Bottlenecks blog

LUMI

- Towards Data Science article by Chaim Rand. Solving Bottlenecks on the Data Input Pipeline with PyTorch Profiler and TensorBoard: PyTorch Model Performance Analysis and Optimization — Part 4
- https://towardsdatascience.com/solving-bottlenecks-on-the-data-input-pipeline-with-pytorch-profiler-and-tensorboard-5dced134dbe9
- Pipeline… CPU (load, collate, pre-process) then send to GPU
- Use Torch profiling to see if GPU is keeping busy, it might reveal other time-consuming parts
- Can set num_workers in the DataLoader perhaps up to number of available cpus
- But other parts might have threading capability (some image processing libraries have this capability)
- Look at applying transformations (say for images) on whole batches at once

Diagnosing and Debugging PyTorch Data Starvation

- Will Price blog: Diagnosing and Debugging PyTorch Data Starvation
- https://www.willprice.dev/2021/03/27/debugging-pytorch-performance-bottlenecks.html
- You identify GPU starvation: maybe cpu-intensive part of training loop or waiting on a batch of data
- Often you see stalls after multiple batch loads (this is due to workers)

Tuning Pytorch Data Loading

LUMI

- **num_workers**
  Increase this but don't overload cpus or memory
- **batch_size**
  The larger it is the more work is needed
- **shuffle**
  Good to randomize samples but not good for disk access, maybe shuffle within blocks of data
- **pin_memory**
  Makes cpu to GPU transfers more efficient
- **persistent_workers**
  Controls if workers are torn down and re-created per epoch

Other considerations

LUMI

- Where is the data?
  - Load it from faster storage – or even /tmp if you have the choice
  - Containerise it – even in a zip file can be useful
- May have to think about all aspects of performance
- Transforming data (image vide)
  - Likely to move to GPU

More Information…

LUMI

- LUMI-O https://docs.lumi-supercomputer.eu/storage/lumio/
- Generic Tutorial on reading large datasets: https://www.kaggle.com/code/rohanrao/tutorial-on-reading-large-datasets
- Best Practice for Data Formats in Deep Learning (SURF) https://servicedesk.surf.nl/wiki/display/WIKI/Best+Practice+for+Data+Formats+in+Deep+Learning
- Ray data loading: https://docs.ray.io/en/latest/train/user-guides/data-loading-preprocessing.html
- Pytorch Tutorial on pre-defined datasets/dataloaders: https://pytorch.org/tutorials/beginner/basics/data_tutorial.html
- Example of keeping training data in memory: "*Scaling Out Deep Learning Convergence Training on LUM*", Diana Moise & Samuel Antao, PDF