Snellius SLURM

Bryan Cardenas
Robert Jan Schilmbach
High Performance Machine Learning

SURF

Module environment

HPC systems are shared between many users. How to offer several versions of the same software?

- Module environment
- modules: `module av`
- We build a module environment every year. Load 2024: `module load 2024`
- `module av Python`
- `module show Python/3.11.3-GCCcore-12.3.0`
- Modules set environment variables like `PATH`, `LD_LIBRARY_PATH`, etc. These are used by the Linux OS to find binaries, libaries, etc

SURF

Module environment

Why using software from the module environment is a good idea

- Convenient!
- All dependencies are provided by our module environment
- Optimized for our specific hardware
  - Difference is most pronounced for CPU-based codes. CUDA binaries are typically (but not always!) optimized for a wide range of GPU models
- One software stack to debug in case of problems (easier to debug then when users each have their own software installations!)
- Many users use the same installation => bigger chance of spotting problems. Only have to solve them once, and they'll be solved for everyone.
- Less duplication (save disk space, reduce amount of files)

Combining virtual environment with modules

What do you do if you need additional Python packages?

- Load whatever you can/want to use from the module environment
- Create a virtual environment with `python -m venv <venv-name>`
- Activate the virtual environment, and pip-install your additional package

```
Module load 2023 Python/3.11.3-GCCcore-12.3.0

Python -m venv .llm
source .llm/bin/activate

pip install transformers
pip install bitsandbytes
```

More on the module environment

See https://servicedesk.surf.nl/wiki/display/WIKI/Environment+Modules

- module avail / module av: show what modules can be loaded
- module spider <modulename>: show what modules can be loaded after other modules (e.g. 2023) are loaded first
- module load <modulename>: load a module
- module display / module show <modulename>: prints the lua script that a ‘module load’ will execute
- module unload <modulename>: unload a module
- module list: show currently loaded modules
- module purge: unload all modules

Why is conda used so much?

Anaconda is used widely accross scientific domains. Why?

- Python virtual environments is limited to managing Python packages. Conda also manages non-Python installations (e.g. CUDA)
  - This leads to better reproducibility and less issues with missing libraries, particularly for novice users
- Many instructions for software installation reference conda
- Conda does proper dependency resolution (pip doesn't) to figure out if all packages in an environment are compatible.
  - If package A is used as dependency by packages B and C, and those package have version requirements A < 2.0 and A > 2.5, conda will simply error out and tell you it's impossible to satisfy both requirements. Pip will install A < 2.0 and then overwrite with A > 2.5.

SURF

Conda

So, why is conda bad on HPC systems?

- Conda uses generically optimized binaries => (potentially) bad for performance
- Conda makes incorrect assumption about the location of system libraries => Still issues at runtime
- Conda virtual environments produce enormous amounts of files, easily half your file quotum on an HPC system
- Conda modifies the `.bashrc` file, which can easily cause conflicts or unintended effects
- Conda environments are hard to support
  - Each conda environment is a full software stack. Each user might have multiple conda environments (i.e. multiple complete stacks). We simply cannot support all of those

See e.g. https://docs.alliancecan.ca/wiki/Anaconda/en

SURF

Containers…

https://docs.alliancecan.ca/wiki/Singularity

Containers are ok-ish on HPC systems. Some remarks

- Containers are nice for HPC file systems, since they save file I/O
- Containers generally have generically optimized binaries, so same performance issues apply as to conda / any binary install (Nvidia GPU performance ok, CPU may or may not be ok)
- Like conda, it is a software stack on its own. We can't help debug.
- Very *static* software environments, hard to add one package => Container rebuild needed
- Need to pull in full container to use one small script
- Multinode use with MPI applications can be difficult

SURF

Best practices: software on HPC (1/2)

- Use software from the module environment whenever you can
  - Build on top of the module environment when you have to
- Never use system installations of Python/GNU compilers/Perl/etc
- If you need additional Python packages: built a virtual environment on top of software from the module environment
  - Load the modules first, *then* create / load the virtual environment

SURF

Best practices: software on HPC (2/2)

- Avoid using conda
  - But if you do, install *everything* with conda. Don’t combine with modules.
  - Expect little support from us
- Use containers with care
  - Optimization usually ok for GPU, but may be poor for CPU
  - Expect little support from us
  - Multinode can be tricky (if MPI is used)

SURF

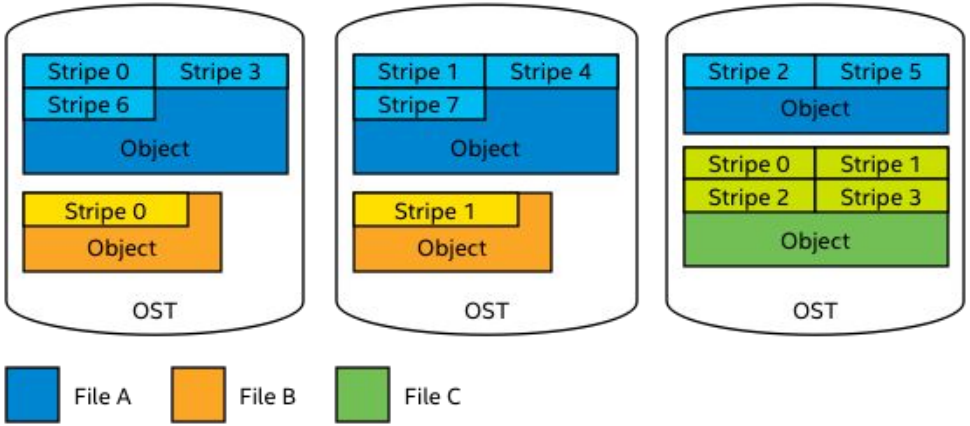
File System

HPC File System

HPC systems typically use parallel shared file systems.

- **Parallel file system**: one file can be distributed over many physical disks, to increase I/O bandwidth.

- **GPFS (Snellius)**
  - Metadata and actual file stored on the same server
  - Striping is managed automatically, by file system. User has no control.



HPC File System

HPC file systems are optimized for ***bandwidth*** not I/O operations per second (IOPS)

- E.g. Snellius GPFS: up to 10 GB/s write, 20 GB/s read
- Accessing small files means many metadata operations; network communication overhead
- Metadata operations are latency sensitive
- Important when e.g. a 100 node job needs to start from the same 20 GB input file…

Hence, HPC file systems are typically **not** good at AI

- AI workloads are generally: reading lots of small files / parts of a file

SURF

HPC File System

- An inode stores all information about a linux file *except* its name and data
  - File size, last change date, file permissions, etc
- On most HPC systems, you'll have an inode quota, in addition to a size quota
  - Like total size, each filesystem has a (fixed) maximum number of inodes
  - It is *also* a way to discourage doing to many IOPS

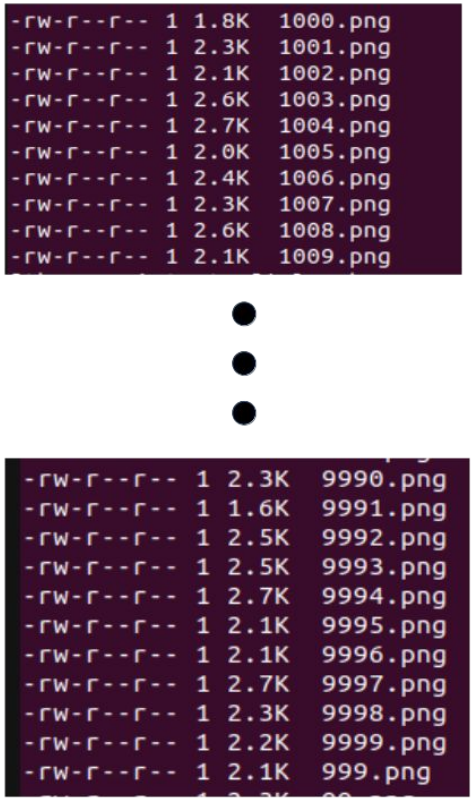
Seems we'll run out of inodes before we run out of space on e.g. our home1 filesystem:

```
$ df -ih /gpfs/home1/
Filesystem     Inodes IUsed IFree IUse% Mounted on
home1            287M  141M  146M   50% /gpfs/home1
$ df -h /gpfs/home1/
Filesystem      Size  Used Avail Use% Mounted on
home1           121T   56T   66T  46% /gpfs/home1
```

HPC File System

Suppose your dataset is 10k png files of 2k each…

- Home1: 121TB, 287M inodes, so 421k per inode
  - If everyone does this, the filesystem will be full of inodes, with 99.5% / 120 TB (!) storage still available…
- If you run 100 epochs, you'll do 1M I/O operations
  - Your sysadmin won't like you
  - Your fellow users won't like you: everyone suffers slow I/O because of your job!
  - You shouldn't like you: your I/O might be holding back your training performance…



Packed Data Format

What is a ‘packed data format’?

- Group multiple individual files together in a single ‘packed’/archive file
- Examples of packed data formats → ZIP, TAR, LMDB, HDF5, PARQUET, etc.
- Does not necessarily involve compression!
  - Compression is possible, but often runs on single process, so slow to decompress

Why packed data format

HPC systems use a **shared** filesystem. Packed data formats…

- Reduces inode consumption
- Reduces number of I/O operations/s
  - Reduces slowdown for yourself *and* other users

Best Practice for Data Formats in Deep Learning - SURF User Knowledge Base

SURF

Local Scratch Disks

Some HPC systems have **local disks** *in* the nodes

- On Snellius, these can be requested by adding `#SBATCH --constraint=scratch-node` to your batch script
  - Note: about half our GPU nodes have local scratch disk. You might be in the queue longer if you request `scratch-node`, so only do it if you think you benefit from it
- Local scratch disks can usually handle much more IOPs
- Local scratch disks are **shared** between all users in a node
  - Users on the same node *may* still experience some effects when you 'hammer' local scratch with I/O, *but it's way less bad than on a shared filesystem!*

SURF

Local Scratch Disks

Some HPC systems have local disks *in* the nodes

- If you really need to use *individual* files instead of a packed file format, use local scratch disks in the following way:

1. **Store** zip/tar on shared parallel Filesystem.
2. At start of job: **extract** to local scratch dir (for every node, if you use multiple)
3. **Do** any further I/O from that local scratch dir

Snellius filesystems - SURF User Knowledge Base

Example SLURM

SLURM Simple commands

Slurm is an open-source workload manager to schedule and manage jobs on HPC (High Performance Computing) clusters.

It allocates compute resources (CPUs, GPUs, memory, etc.)

Resource allocation: jobs run only when the requested resources are available.

Job queuing: slurm decides when/where to run them.

Monitoring and control: track job status, usage, and can enforce time/memory limits.

Creating and running jobs - SURF User Knowledge Base

SLURM Simple commands

Slurm is an open-source workload manager to schedule and manage jobs on HPC (High Performance Computing) clusters.

It allocates compute resources (CPUs, GPUs, memory, etc.)

Information

1. `sinfo`: View available resources/partitions.
2. `squeue`: View running/pending jobs.
3. `scancel`: Cancel jobs.
4. `scontrol`: Advanced control over jobs and nodes.

Creating and running jobs - SURF User Knowledge Base

SURF

SLURM Simple commands

Slurm is an open-source workload manager to schedule and manage jobs on HPC (High Performance Computing) clusters.

It allocates compute resources (CPUs, GPUs, memory, etc.)

Allocation

1. `sbatch`: Submit a job for batch processing.
2. `salloc`: Interactive session on a compute node
3. `srun`: Launch tasks.

Creating and running jobs - SURF User Knowledge Base

SURF

SLURM Simple commands

Slurm is an open-source workload manager to schedule and manage jobs on HPC (High Performance Computing) clusters.

Live Monitoring

1. `squeu -u $USER`
2. `sstat / scontrol show job`
3. `nvidia-smi` (after ssh'ing into a node)

Creating and running jobs - SURF User Knowledge Base

SURF

SLURM Simple commands

Slurm is an open-source workload manager to schedule and manage jobs on HPC (High Performance Computing) clusters.

Module

1. `module avail`
2. `Module spider <modulename>`
3. `Module show <modulename>`
4. `Module purge`
5. `Module list`
6. `Module unload <modulename>`
7. `Module load <modulename>`

1. show what modules can be loaded
2. show what modules can be loaded after other modules
3. load a module
4. prints the lua script that a ‘module load’ will execute
5. unload a module
6. show currently loaded modules
7. unload all modules

Creating and running jobs - SURF User Knowledge Base

Sbatch Example

```
#!/bin/bash
#SBATCH --job-name=my_job
#SBATCH --nodes=1
#SBATCH --time=00:10:00
#SBATCH --partition gpu_a100
#SBATCH --gpus 1
#SBATCH --ntasks 1
#SBATCH --cpus-per-task 8

# load environment
source ./venv/bin/activate
module load 2024
module load cuDNN/9.5.0.50-CUDA-12.6.0
cd ./ml/algorithms/esmfold/

cmd_args="--fastas_folder ./outputs/proteinmpnn/seqs/
--output_folder ./outputs/esmfold/"

# run algorithm
python esmfold.py ${cmd_args}
```

Interactive Allocation Equivalent:

```
salloc --time 00:10:00 --nodes 1
--partition gpu_a100 --gpus 1
--ntasks 1 --cpus-per-task 8
--job-name my_job
```

Creating and running jobs - SURF User Knowledge Base

Sbatch notebook Example

```
#!/bin/bash
#SBATCH -t 1:00:00
#SBATCH -N 1
#SBATCH -p thin
#SBATCH -o jupyter-notebook-job.out

# Make sure the jupyter command is available, either by loading the appropriate modules, sourcing your own virtual
environment, etc.
module load 2022
module load IPython/8.5.0-GCCcore-11.3.0
module load JupyterHub/3.0.0-GCCcore-11.3.0

# Choose random port and print instructions to connect
PORT=`shuf -i 5000-5999 -n 1`
LOGIN_HOST=${SLURM_SUBMIT_HOST}-pub.snellius.surf.nl
BATCH_HOST=$(hostname)

echo "To connect to the notebook type the following command into your local terminal:"
echo "ssh -N -J ${USER}@${LOGIN_HOST} ${USER}@${BATCH_HOST} -L ${PORT}:localhost:${PORT}"
echo
echo "After connection is established in your local browser go to the address:"
echo "http://localhost:${PORT}"

jupyter notebook --no-browser --port $PORT
```

Jupyter Notebooks on Snellius - SURF User Knowledge Base

Sbatch notebook Example

1. Modify the previous notebook contents depending on your setup. Run the notebook on a login node, not a GPU node. Use:

   sbatch notebook.job.

2. You will obtain an output, e.g. slurm-5065147. This file will tell you how to connect to your notebook

3. Connect to the notebook follow the command (from the output slurmfile) from a new *local* terminal. e.g.:

ssh -J bryanc@int6-pub.snellius.surf.nl
bryanc@tcn1113.local.snellius.surf.nl -L 5994:localhost:5994

4. Open a browser and copy paste what the terminal is telling you, e.g.:

http://localhost:5994/tree?token=9514f7a3f5d21e9cb959203e6e26d62896f3713
75711da84

Jupyter Notebooks on Snellius - SURF User Knowledge Base

1. Small or Large NWO Grant

https://www.surf.nl/en/research-it/apply-for-access-to-compute-services

https://servicedesk.surf.nl/wiki/pages/viewpage.action?pageId=30660193

https://servicedesk.surf.nl/wiki/display/WIKI/NWO+grants

2. Mail us!

bryan.cardenasguevara@surf.nl

damian.podareanu@surf.nl

matthieu.laneuville@surf.nl

hpml@surf.nl

https://www.surf.nl/en/consultancy-on-it-solutions-for-researchers

Apply for consultancy hours!