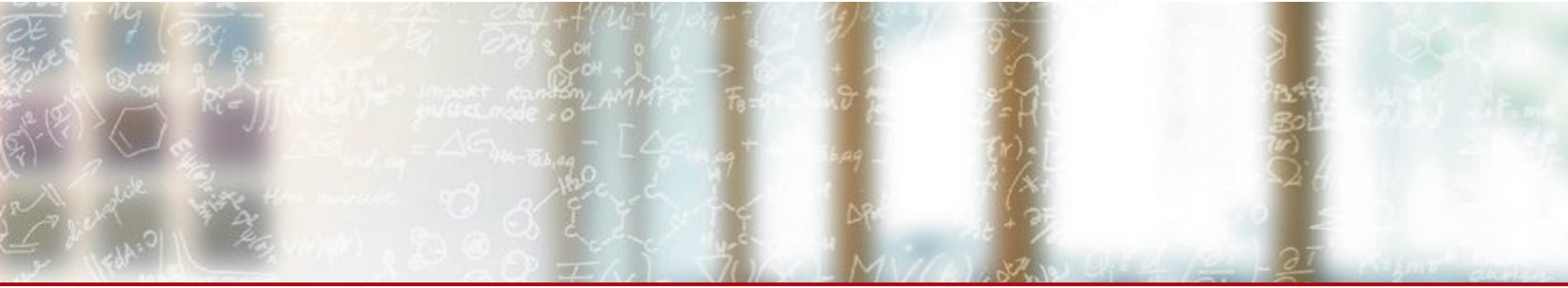
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



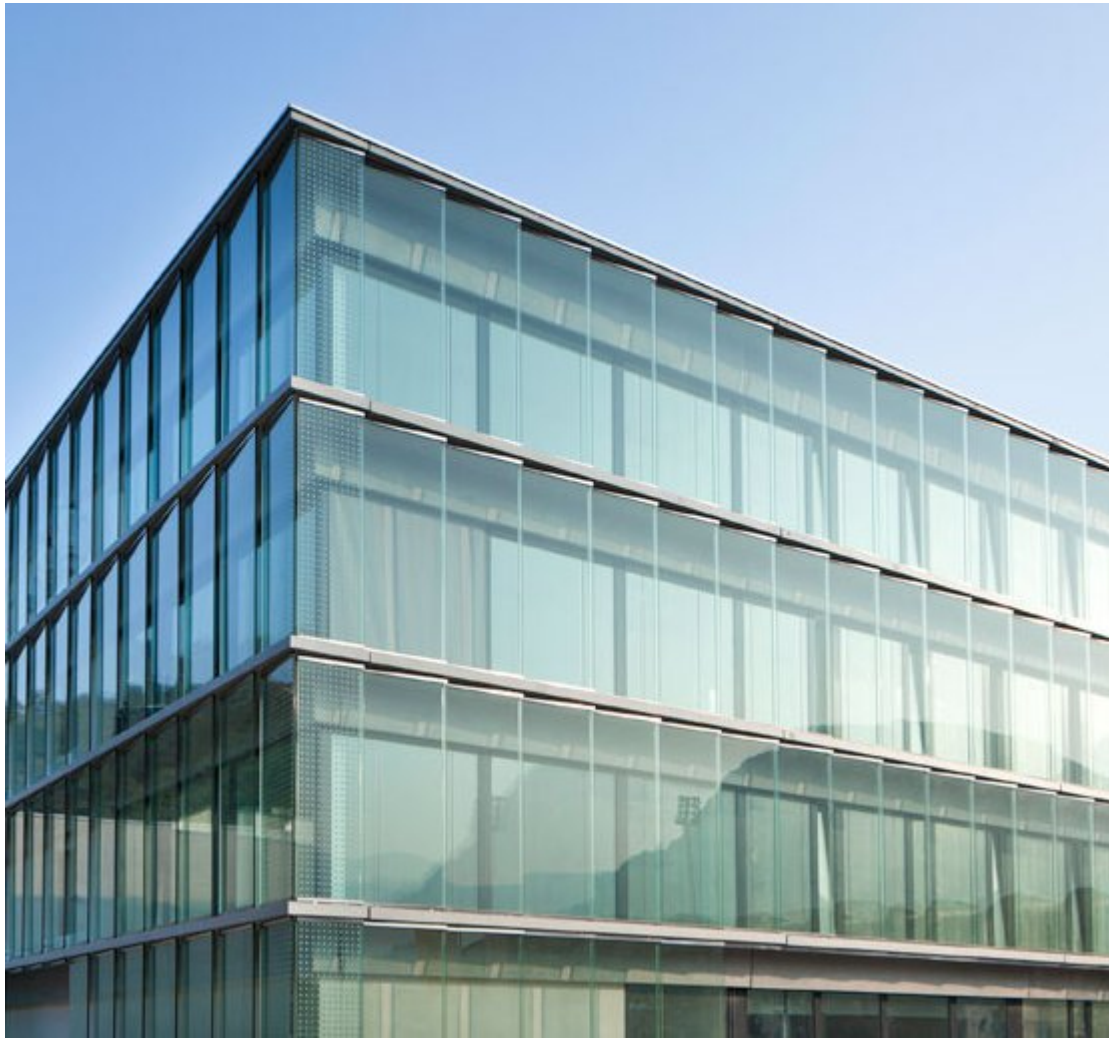
A Practical Introduction to CSCS HPC Infrastructure

CSCS User Lab Day, August 31st 2020

Luca Marsella, CSCS

Outline of the Presentation

- Policies and Resources
  - Login
  - Filesystems
- Development Environment
  - Features of the hybrid system
  - Programming Environment
- Job Submission and Monitoring
  - Slurm workload manager
  - Best practices
- Documentation and Troubleshooting
  - How to submit a support request



CSCS office building in Lugano

Policies and Resources

General Policies

The code of conduct outlines proper practices:

- **Access to Source Codes:** you agree to make codes available for support
- **Scientific Advisory Committee:** committee members must not be contacted
- **Acknowledgements:** you must acknowledge the use of CSCS resources in all publications related to your production with reference to the "*project ID ###*"

User Regulations define basic guidelines:

- **Accounts are personal** and sharing them is forbidden
- **Data ownership**: access to and use of data of other accounts without prior consent from the principal investigator is strictly prohibited
- **ETH Zurich Acceptable Use Policy for Telematics Resources ("BOT")**

Access to CSCS resources **may be revoked to users violating the policies**

Data Retention Policies

Data backup for **active projects**:

- Data in **users** and **project** folders is backed up (past 90 days)
- Data recovery is also possible with daily snapshots (past 7 days)
- Data in **project** folders removed **3 months** after the **end of the project**

As soon as a the project **expires**:

- Data backup is **disabled** immediately
- **No data recovery** after the final data removal

No backup for data in **scratch**:

- No recovery in case of **accidental data loss**
- No recovery of data deleted due to the cleaning policy

Fair Usage Policies

Slurm:

- The job scheduler is a shared resource among users submitting jobs
- **Do not submit large numbers of jobs** and commands at the same time
- We will be forced to kill jobs and limit new submissions

Login nodes:

- **Running applications on login nodes is not allowed**
- Submit your simulations with the Slurm scheduler on compute nodes
- **Heavy processes** running on login nodes will be **terminated**

Please check the summary of CSCS policies at
https://user.cscs.ch/access/accounting/#policies

How to access the systems

You should have already obtained an account at CSCS

The front end Ela is accessible via **ssh** as **ela.cscs.ch**:

```
$ ssh ela.cscs.ch
```

- It provides a minimal Linux environment
- You can **ssh** the computing systems from Ela

```
$ ssh daint.cscs.ch
```

- You can start an External Data Transfer with GridFTP from/to CSCS

Please note the following:

- **No programming environments** on the front end system
- User scratch space is **not directly accessible** from Ela

Filesystems

	/scratch (Piz Daint)	/scratch (Clusters)	/users	/project	/store
Type	Lustre	GPFS	GPFS	GPFS	GPFS
Quota	Soft quota 1 M files	None	50 GB/user 500K files	Maximum 50K files/TB	Maximum 50K files/TB
Expiration	30 days	30 days	Account closure	End of the project	End of the contract
Data Backup	None	None	90 days	90 days	90 days
Access Speed	Fast	Fast	Slow	Medium	Slow
Capacity	8.8 PB	1.9 PB	160 TB	6.3 PB	5.0 PB

Soft quota:

- Soft quota on **Piz Daint** to prevent excessive loads on the scratch filesystem
- Quota reached: **warning at submit time**, **no job submission** allowed

/scratch filesystem

Fast workspace for running jobs:

- Designed for **performance** rather than reliability
- **Cleaning policy**: files **older than 30 days deleted** daily
- **No backup:** transfer data after job completion

Performance of Piz Daint scratch (Lustre filesystem):

- **Soft quota on inodes** (files and folders) **to avoid** large numbers of small files
- Occupancy impacts performance:
  - **> 60%**: we will ask you to **remove unnecessary data immediately**
  - **> 80%**: we will free up disk space **manually removing data**

All CSCS systems provide a scratch personal folder:

- the variable **$SCRATCH on Piz Daint** points to **/scratch/snx3000/$USER**

/users and /project filesystems

Shared parallel filesystems based on the IBM GPFS software:

- Accessible from the login nodes using native GPFS client
- Storage space for datasets, shared code or configuration scripts
- Better performance with larger files (archive small files with **tar**)

Users are NOT supposed to run jobs here:

- The emphasis is on **reliability over performance**
- All directories are **backed up** with GPFS snapshots
- No cleaning policy until 3-months after the end of the project

Environment variables pointing to personal folders:

- **$HOME** points to **/users/$USER**
- **$PROJECT** points to **/project/<group_id>/$USER**

Computing Resources

Computing time on Cray systems is accounted in **node hours**:

- Resources are assigned over **three-months windows**
  - **Quotas reset** on April 1st, July 1st, October 1st and January 1st
- Use thoroughly the quarterly compute budget within the time frame
- Unused resources in the allocation periods **cannot be recovered**

Check your budget in the **current allocation window**:

- Group usage - **sbucheck**
  - reports group usage across the various systems
- Daily usage - **monthly_usage**
  - **monthly_usage --individual** usage per group member
- Overview of resources with the Account and Resources Tool
  - Check the details on the dedicated page of the User Portal

Development Environment

Piz Daint Specifications

Model	Cray XC50/XC40
XC50 Compute Nodes (Intel Haswell processor)	Intel® Xeon® E5-2690 v3 @ 2.60GHz (12 cores, 64GB RAM) and NVIDIA® Tesla® P100 16GB
XC40 Compute Nodes (Intel Broadwell processor)	Intel® Xeon® E5-2695 v4 @ 2.10GHz (2 x 18 cores, 64/128 GB RAM)
Login Nodes	Intel® Xeon® E5-2650 v3 @ 2.30GHz (10 cores, 256 GB RAM)
Interconnect Configuration	Aries routing and communications ASIC Dragonfly network topology
Scratch capacity	Piz Daint scratch filesystem: 8.8 PB

File Systems:

- **/project** is mounted with **read-only** access on compute nodes

NVIDIA CUDA Toolkit v10.1

- Comprehensive development environment to build **GPU-accelerated applications**
  - **compiler** for NVIDIA GPUs
  - optimized **math libraries**
  - **debugging** and **performance** tools

- Features **programming guides**, user manuals, API reference and **online documentation** to get started quickly

- NVIDIA developer portal:
  https://developer.nvidia.com/cuda-zone

NVIDIA Tesla P100 GPU Accelerator

Cray Linux Environment 7.0 UP01

- Cray Linux Environment (CLE) is the operating system on Cray systems
- CLE 7.0 UP01 is based on the SUSE Linux Enterprise Server version 15
- CLE 7.0 UP01 software release is available on Piz Daint since Nov 2019
- Read more on the HPE Cray Technical Documentation portal (pubs.cray.com)

Setting the Programming Environment

You should prepare the environment before running jobs:

- CSCS systems use the modules framework
  - The modules manage applications and libraries path
  - You can check currently loaded modules with **module list**

- Some **modules** are already loaded at login
  - The default environment on Piz Daint is **PrgEnv-cray**
  - The default architecture is XC50 (Intel Haswell): **craype-haswell**
  - You can browse the available modules with **module avail**

- You must **adjust your target architecture** (see **sbucheck**)
  - **daint-gpu** targets the XC50 (Intel Haswell and P100 Tesla GPUS)
  - **daint-mc** targets the XC40 (Intel Broadwell multicore)
  - These modules update the **MODULEPATH**

Setting the Programming Environment

```
$ module switch PrgEnv-cray/6.0.5 PrgEnv-gnu
$ module load daint-gpu
$ module list
Currently Loaded Modulefiles:
 1) modules/3.2.11.3                 9) cray-libsci/19.06.1                 17) dvs/2.12_2.2.151-7.0.1.1_5.6__g7eb5e703
 2) gcc/8.3.0                       10) udreg/2.3.2-7.0.1.1_3.9__g8175d3d.ari  18) alps/6.6.56-7.0.1.1_4.10__g2e60a7e4.ari
 3) craype-haswell                  11) ugni/6.0.14.0-7.0.1.1_7.10__ge78e5b0.ari  19) rca/2.2.20-7.0.1.1_4.9__g8e3fb5b.ari
 4) craype-network-aries            12) pmi/5.0.14                         20) atp/2.1.3
 5) craype/2.6.1                    13) dmapp/7.1.1-7.0.1.1_4.8__g38cf134.ari  21) perftools-base/7.1.1
 6) cray-mpich/7.7.10               14) gni-headers/5.0.12.0-7.0.1.1_6.7__g3b1768f.ari  22) PrgEnv-gnu/6.0.5
 7) slurm/19.05.3-2                 15) xpmem/2.2.19-7.0.1.1_3.7__gdcf436c.ari  23) daint-gpu
 8) xalt/2.7.24                     16) job/2.2.4-7.0.1.1_3.8__g36b56f4.ari

$ module avail ...
```

Cray XC Programming Environment

- Cray XC PE 19.10 includes the **Cray Developer Toolkit - CDT 19.10**
  - non-default Programming Environments can be accessed with **cdt** modules
- The following products have been updated within this release:
  - **Cray Compiling Environment - CCE**
    - cce 9.0.2, cray-mpich 7.7.10, cray-libsci 19.06.1
  - **Cray Performance Measurement & Analysis Tools - CPMAT**
    - Perftools 7.1.1
  - **Cray Environment Setup and Compiling support - CENV**
    - cray-modules 3.2.11.3 and craype 2.6.1
  - **Third party products**
    - GCC 7.3.0 and 8.3.0, cray-python 2.7.15.7 and 3.6.5.7, cray-R 3.4.2

Main default modules for supported applications on Piz Daint

Amber/18-14-14-CrayGNU-19.10-cuda-10.1	Amber/18-14-14-CrayGNU-19.10
Boost/1.70.0-CrayGNU-19.10-python3	Boost/1.70.0-CrayGNU-19.10-python3
CDO/1.9.5-CrayGNU-19.10	CDO/1.9.5-CrayGNU-19.10
CP2K/6.1-CrayGNU-19.10-cuda-10.1	CP2K/6.1-CrayGNU-19.10
CPMD/4.1-CrayIntel-19.10	CPMD/4.1-CrayIntel-19.10
GROMACS/2018.6-CrayGNU-19.10-cuda-10.1	GROMACS/2018.6-CrayGNU-19.10
LAMMPS/22Aug18-CrayGNU-19.10-cuda-10.1	LAMMPS/22Aug18-CrayGNU-19.10
NAMD/2.13-CrayIntel-19.10-cuda-10.1	NAMD/2.13-CrayIntel-19.10
NCL/6.4.0	NCL/6.4.0
NCO/4.8.1-CrayGNU-19.10	NCO/4.8.1-CrayGNU-19.10
ParaView/5.7.0-CrayGNU-19.10-EGL	ParaView/5.7.0-CrayGNU-19.10-OSMesa
QuantumESPRESSO/6.4.1-CrayIntel-19.10-cuda-10.1	QuantumESPRESSO/6.4.1-CrayIntel-19.10
VASP/5.4.4-CrayIntel-19.10-cuda-10.1	VASP/5.4.4-CrayIntel-19.10
	Visit/3.1.0-CrayGNU-19.10

For more details please check the User Portal at https://user.cscs.ch/computing/applications

Job Submission and Monitoring

The Slurm workload manager

- Slurm is the batch system/scheduler running on CSCS machines
- Permits users to run jobs with specific settings
- Job submission by calling sbatch with a job script

job.sh

```
#!/bin/bash -l
#SBATCH --nodes=10
#SBATCH --time=0:30:00
#SBATCH --partition=normal
#SBATCH --constraint=gpu
[...]

srun myprogram
```

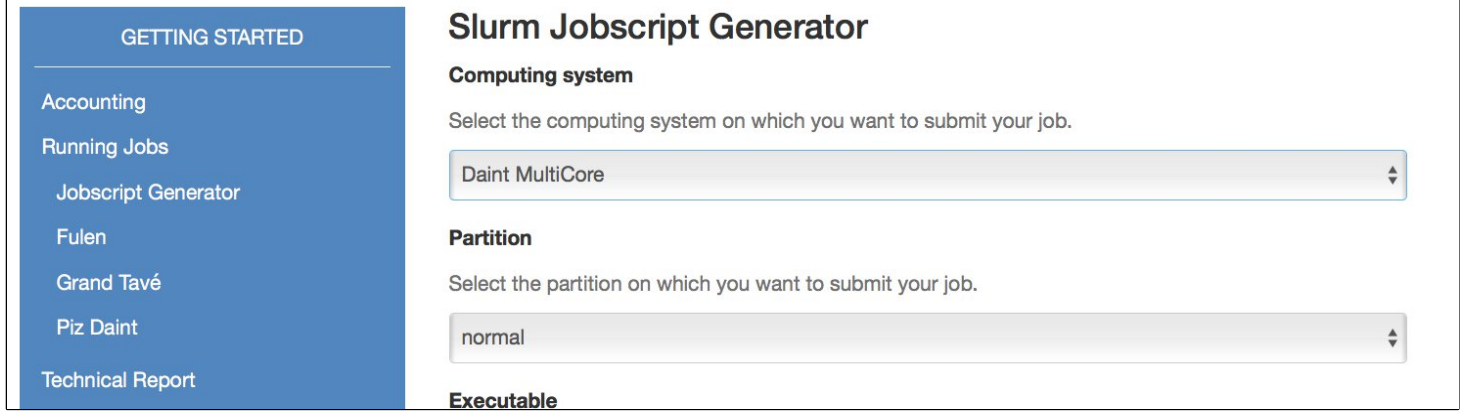
```
$ sbatch job.sh
```

Slurm jobscripts

- Our web interface for generating job scripts covers most cases

Slurm Jobscript Generator

https://user.cscs.ch/access/running/jobscript_generator/



- For a comprehensive list of all options please check `$ man sbatch`

Slurm queues on Piz Daint

- Corresponding Slurm option: **--partition**
- Allows users to run jobs on different queues

Name of the queue	Max time	Max nodes	Brief Description
debug	30 min	4	Quick turnaround for test jobs (one per user)
large	12 h	4400	Large scale work, by arrangement only
long	7 days	4	Maximum 5 long jobs in total (one per user)
low	6 h	2400(gpu)/512(mc)	Up to 130% of project's quarterly allocation
normal	24 h	2400(gpu)/512(mc)	Standard queue for production work
prepost	30 min	1	High priority pre/post processing
xfer	24h	1	Data transfer queue

More information at https://user.cscs.ch/access/running/piz_daint/#slurm-batch-queues

Slurm queues (2)

- Watch your jobs in queues with

```
$ squeue -u ${USER}
```

```
daint103:~$ squeue -u simbergm
  JOBID     USER ACCOUNT         NAME  ST REASON    START_TIME            TIME  TIME_LEFT NODES CPUS
11942503 simbergm csstaff hpx-3662-gcc-7   R None      16:36:57            30:26    5:29:34     1   24
11945966 simbergm csstaff hpx-3712-gcc-7   R None      16:44:24            22:59    5:37:01     1   72
11947200 simbergm csstaff hpx-3229-clang  PD BeginTime 17:34:15               0:00    6:00:00     1    1
11947180 simbergm csstaff hpx-3684-gcc-7  PD BeginTime Tomorr 00:19           0:00    6:00:00     1    1
```

- Observe state of queues with

```
$ sinfo -o"%P %.5a %.10l %.6D %.6t"
```

```
daint103:~$ sinfo
PARTITION AVAIL JOB_SIZE  TIMELIMIT   CPUS  S:C:T   NODES STATE      NODELIST
debug     up    1-4         30:00     72 2:18:2      2 allocated  nid00[448-449]
debug     up    1-4         30:00    24+ 1+:12+      14 idle      nid0[0008-0011,0450-0451,3508-3511,4276-4279]
xfer      up    1        1-00:00:00      9  9:1:1      2 idle      nordend0[3-4]
uftp      up    1        1-00:00:00      0  0:0:0      0 n/a
cscsci    up    1        1-00:00:00    24+ 1+:12+       7 down$     nid0[0125,0299,3541-3543,4579,5967]
cscsci    up    1        1-00:00:00    24+ 1+:1+:      28 maint     nid0[0124,0126,1144-1147,1804-1807,3492-3495,3576-
```

Job Priority

- Job priority is based on partition, fair share and waiting time: check it with `$ sprio -w`
- Check the reason why a job is pending with the command `$ squeue -u ${USER}`
- Check your budget with the command `$ sbucheck`
  Even if you still have lots of hours left, there may be other users/accounts with less usage and/or more hours allocated
- If the reason is “priority”, then you have to wait in the queue!
- Also, make sure there are no reservations in the system (maintenances, large runs,...):

```
$ scontrol show reservations
```

Good practices when submitting jobs

Specify accurate wall time

```
#!/bin/bash -l
#SBATCH --nodes=120
#SBATCH --time=0:30:00
#SBATCH --partition=normal
#SBATCH --constraint=gpu
[...]
```

Run jobs off ${SCRATCH}

```
#!/bin/bash -l
#SBATCH --nodes=120
#SBATCH --time=0:30:00
#SBATCH --partition=normal
#SBATCH --constraint=gpu
#SBATCH --mail-type=ALL
#SBATCH --mail-user=<your_email>

cd ${SCRATCH}
srun ${SCRATCH}/my_binary
```

For jobs with *many* steps, use greasy

```
#!/bin/bash -l
#SBATCH --nodes=120
#SBATCH --time=0:30:00
#SBATCH --partition=normal
#SBATCH --constraint=gpu

module load daint-mc
module load GREASY
greasy -f greasy_tasks.list
```

Make sure your srun commands work! (or sleep a little bit in between)

```
#!/bin/bash -l
#SBATCH --nodes=120
#SBATCH --time=0:30:00
#SBATCH --partition=normal
#SBATCH --constraint=gpu
function p() {
rt=$?;
if [[ ${rt} -ne 0 ]]; then
  sleep 2
fi
return ${rt}
}
srun mytask ; p
srun mytask2 ; p
```

What not to do when submitting jobs

Jobs that submit other jobs/steps in loops

```
#!/bin/bash
#SBATCH ...
while :
do
srun sbatch my_job.sbatch
sleep 1
done
```

Jobs with *thousands* of steps

```
# sacct -j 123456789 |wc -l
25337
```

Jobs that run off ${HOME}

```
#!/bin/bash -l
#SBATCH --nodes=120
#SBATCH --time=0:30:00
#SBATCH --partition=normal
#SBATCH --constraint=gpu

srun ~/my_binary ~/Large_input
```

Jobs with hundreds of steps in parallel

```
#!/bin/bash -l
#SBATCH --nodes=3
#SBATCH --time=0:30:00
#SBATCH --partition=normal
#SBATCH --constraint=gpu

export CRAY_CUDA_MPS=1
cd $SLURM_SUBMIT_DIR

date
srun --nodes=1 --bcast=/tmp/${USER} --ntasks=1 --ntasks-per-node=1 --cpus-per-task=12
tune_5x16x13_exe0 &
srun --nodes=1 --bcast=/tmp/${USER} --ntasks=1 --ntasks-per-node=1 --cpus-per-task=12
tune_5x16x13_exe1 &
srun --nodes=1 --bcast=/tmp/${USER} --ntasks=1 --ntasks-per-node=1 --cpus-per-task=12
tune_5x16x13_exe10 &
[...]
sleep 29m
```

What not to do on login nodes

squeue without filtering

```
$ squeue | grep ${USER}
$ squeue -u ${USER}
```

watch overloads the scheduler

```
$ watch squeue -u ${USER}
```

sacct + watch: even worse!

```
$ watch sacct
```

GNU make without number of tasks

```
$ make -j
$ make -j8
```

Avoid infinite loops

```
#!/bin/bash
while :
do
clear
squeue | grep JOBID
squeue | grep ${USER}
sleep 1
done
```

Loops that depends on variables are worse!

```
#!/bin/bash
for i in ${var}; do
sbatch my_job.sbatch
done
```

Use e-mail notification instead of loops with Slurm commands

```
#!/bin/bash -l
#SBATCH --nodes=2
#SBATCH --mail-type=ALL
#SBATCH --mail-user=<your_email>
```

Summary: How to submit jobs at CSCS

- Move input data to $SCRATCH

```
$ cd $SCRATCH
$ cp -r ~/input .
```

- Use the jobscript generator and accurately specify runtime

```
$ sbatch job.sh
```

- Monitor (manually) your jobs with

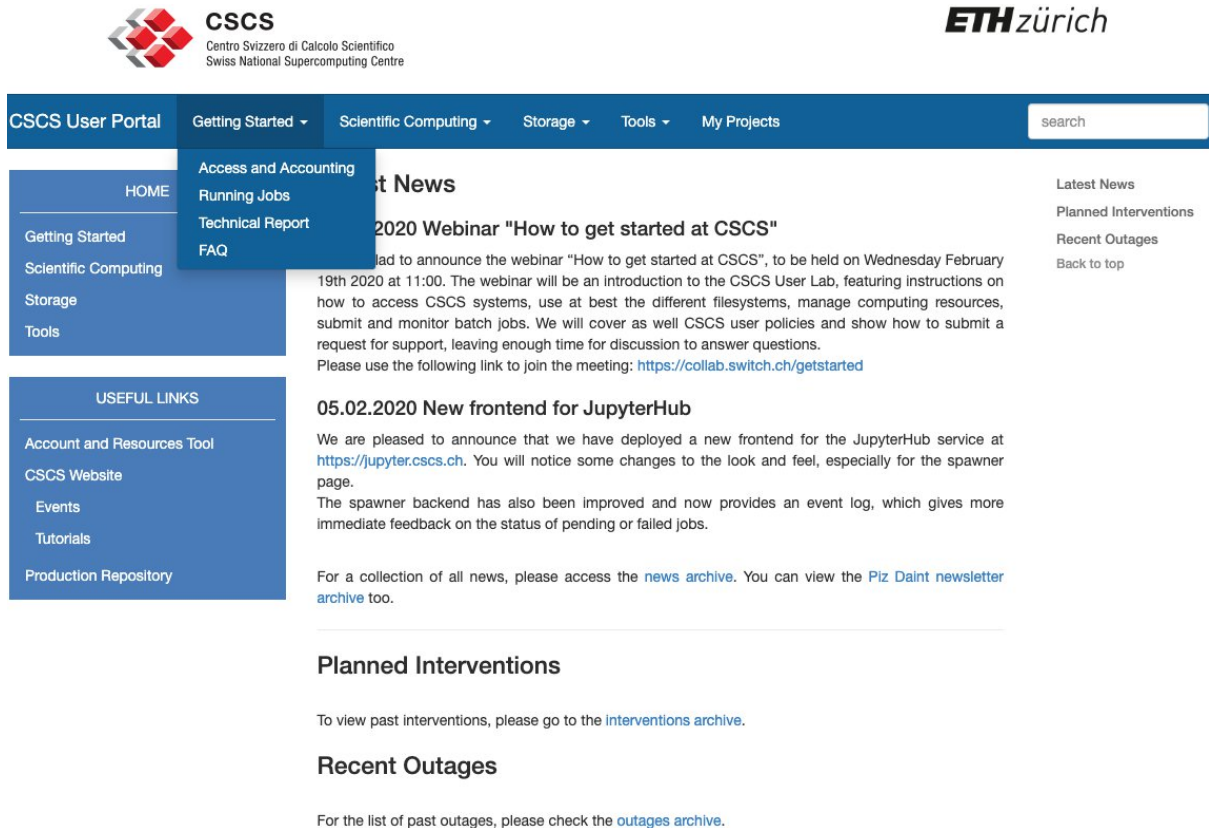
```
$ squeue -u ${USER}
```

- Use Slurm e-mail notification for live-updates on job status instead of repeated Slurm commands
- Copy important output data back to /project or /home

Documentation and Troubleshooting

What to do in case of trouble?

- Search the content of the User Portal
- Check the Frequently Asked Questions
- Additional user documentation:
  - **module help**
  - **man**
  - **HPE Cray Technical Documentation**



User Documentation

- General info with **module help**
  - **module help PrgEnv-cray**
- Product related details
  - **module help cce**
- Command specific guidelines
  - **man craycc**
- Advanced HPE Cray topics
  - CCE 9.1 Usage

```
The PrgEnv-cray modulefile loads the Cray Programming Environment, which
includes the Cray Compiling Environment (CCE). This modulefile defines the
system paths and environment variables needed tobuild an application using
CCE for supported Cray systems. For moreinformation on using targeting
modules see Cray Programming Environment User'sGuide, S-2529-114.

This module loads the following products:

      craype
      cce
      cray-libsci
      udreg
      ugni
      pmi
      dmapp
      gni-headers
      xpmem
      job
      dvs
      alps
      rca
      atp
      perftools-base
```

Cray Documentation

- **Cray man** pages:
  - Textual help files on the command line of Cray systems
  - **man** command followed by the name of the man page
  - Described on man(1) page accessible with **man man**

- **HPE Cray Technical Documentation** at http://pubs.cray.com
  - Quick access and search of Cray books
  - **man** pages and third-party documentation
  - Available in HTML and PDF formats

NVIDIA Documentation

- NVIDIA Documentation Portal
  - http://docs.nvidia.com
- Documentation on the system:
  - **module help cudatoolkit**
  - NVIDIA compiler
    - **nvcc --help**
  - CUDA profiler
    - **nvprof --help**

```
$ module help cudatoolkit

----------- Module Specific Help for 'cudatoolkit/10.1.105_3.27-7.0.1.1_4.1__ga311ce7' --

The modulefile defines the system paths and
variables for the Cuda Toolkit.

cray-cudatoolkit 10.1.105_3.27-7.0.1.1_4.1__ga311ce7
================

Release Date: December 6, 2012

Purpose:
--------
  cray-cudatoolkit 10.1.105_3.27-7.0.1.1_4.1__ga311ce7 provides a development environment
  for NVIDIA GPUs.  It provides a standalone compilation
  environment for CUDA for C/C++ (nvcc) and the infrastructure
  used by other compilation environments (CCE and PGI).

  Includes:
    NVCC compiler
    CUDA runtime support libraries
    Ptx assembler
    CUDA math libraries
    Profiler
    Cuda-gdb
    Cuda-memcheck
```

How to submit a support request

Contact us **if you can't find a solution**:

- Write an e-mail to **help@cscs.ch**
- Specify the **system** and your **project ID** in the subject
- Report the **Slurm job ID** and indicate the **Slurm job script**
- Copy scripts and source files to **$SCRATCH** and give us access

The **more detailed the request**, the **more effective the reply**!

Example of a request for support

Template message to be adapted for **help@cscs.ch**

Subject: Slurm job failed on Piz Daint (project <*project ID*>)

Content:

My username is <*user name*>, I submitted the job <*job ID*> on Piz Daint.

The job running <*code name*> exited with state **FAILED** but no error in output.

The job script (<*script name*>) and input files (<*file list*>) can be found here:

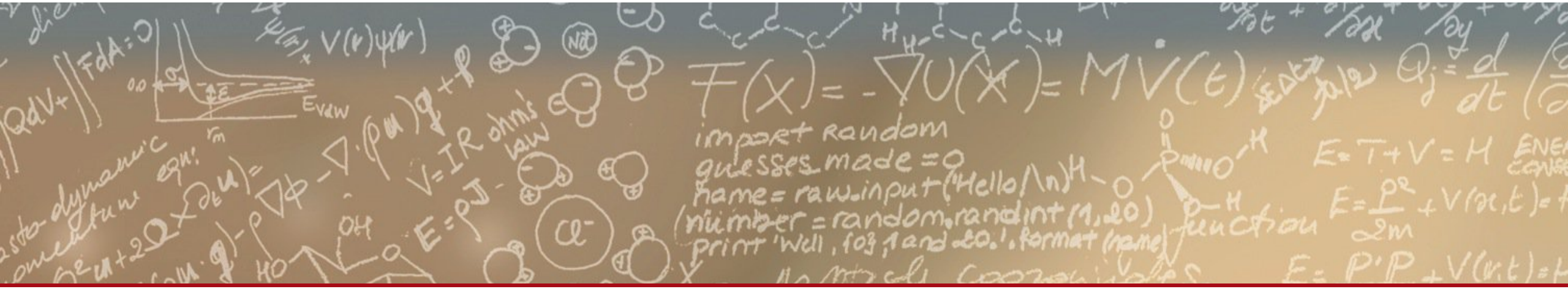
- **/scratch/snx3000/$USER/job**
- I have given read access with **chmod –R +r $SCRATCH/job**

Please let me know the reason of the failure.

Useful links

- CSCS User Portal
  - https://user.cscs.ch
- HPE Cray Documentation
  - https://pubs.cray.com
- NVIDIA Documentation
  - https://docs.nvidia.com
- Contact us
  - help@cscs.ch

Piz Daint in the machine room at CSCS



Thank you for your kind attention