Using the clusters

Before reading this document, you must have logged at least once to a cluster (see here for the details) and possibly tried out some examples.

This is a comprehensive general guide on how to use the clusters:

In the following examples, ***<username>*** means your Gaspar username.

Batch systems

The key to using the clusters is to keep in mind that all tasks (or jobs) need to be given to a batch system called SLURM. With this scheduler, your jobs will be launch according to different factors such as priority, availability of the nodes, etc.

Except for rare cases the idea is not to have real-time interaction and, even in such cases, the jobs are still managed by the batch system.

All the clusters are using SLURM which is widely used and open source **http://slurm.schedmd.com** .

Running jobs with SLURM

Currently, only Fidis has a serial partition with a "pay-as-you-use" policy. On the other clusters, you *will be charged for the whole nodes even if you use only a fraction of them*.

In the case you only need a few cores, please make sure to use Fidis' serial partition. Failing to do so will increase the cost of your simulation!

More details can be found on the 2020 Annual Maintenance page.

The normal way of working is to create a short script that describes what you need to do and submit it to the batch system using the ***sbatch*** command.

Here is an example of a script running a code called *moovit* :

```
#!/bin/bash
#SBATCH --chdir /scratch/<username>/moovit-results
#SBATCH --nodes 1
#SBATCH --ntasks 1
#SBATCH --cpus-per-task 1
#SBATCH --mem 4096
#SBATCH --time 12:30:00
echo STARTING AT `date`
/home/<username>/code/moovit < /home/<username>/params/moo1
echo FINISHED at `date`
```

Any line beginning with ***#SBATCH*** is called a directive to the batch system. Type the command *man sbatch* for the whole explanation and the full list of options.

The six options in the directives are more or less mandatory and do the following:

```
--chdir /path/to/working/directory
```

This is the directory in which the job will be run and the standard output files written. This should ideally point to your scratch space.

```
--ntasks 1
```

The ntasks is the number of tasks (in an MPI sense) to run per job

```
--cpus-per-task 1
```

This is the number of cores per aforementioned task

```
--nodes 1
```

This is the number of nodes to use - on Castor this is limited to 1 but it's good practice to request it anyway!

```
--mem 4096
```

The memory required in MB per node

```
--time 12:00:00 # 12 hours
--time 2-6 # two days and six hours
```

The time required. Note that there are many different formats to specify the time. See the manual of sbatch (by typing ***man sbatch***) and look for the details on this option.

If the time and memory are not specified then default values will be imposed - these may well be lower than required!

This script should be saved to a file, for example `moojob1.run` and in order to submit it we run the following command from one of the login nodes:

```
sbatch moojob1.run
```

The output will look something like

```
[username@frontend]$ sbatch moojob1.run
Submitted batch job 123456
```

The number returned is the Job ID and is the key to finding out further information or modifying the task.

Cancelling Jobs

To cancel a specific job:

```
scancel JOBID
```

To cancel all your jobs (use with care!):

```
scancel -u username
```

To cancel all your jobs that are not yet running:

```
scancel -u username -t PENDING
```

Getting Job Information

There are a number of different tools that can be used to query jobs depending on exactly what information is needed.

If the name of a tool begins with a capital **S** then it is a SCITAS specific tool. Any tool whose name starts with a small **s** is part of the base SLURM distribution.

Squeue

`Squeue` shows information about all your jobs be they running or pending.

```
[username@machine]$ Squeue
     JOBID         NAME  ACCOUNT       USER NODE  CPUS  MIN_MEMORY     ST       REASON
START_TIME             NODELIST
    123456         run1   scitas        bob    6    96       32000      R         None  2015-10-30T04:18:
37      r04-node[32-37]
    123457         run2   scitas        bob    6    16       32000     PD   Dependency                 N
/A
```

squeue

By default `squeue` will show you all the jobs from all users. This information can be modified by passing options to squeue.

To see all the running jobs from the scitas group we run:

```
[username@machine ~]# squeue -t R -A scitas
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
            123456  parallel  gromacs      bob  R      48:43      6 r04-node[32-37]
            123457  parallel     pw.x      sue  R   18:06:44      8 r01-node[03,11,21],r04-node[50,61-64]
```

See `man squeue` for all the options.

For example, the `Squeue` command described above is actually a script that calls:

```
squeue -u $USER -o "%.10A %.12j %.8a %.10u %.4D %.5C %.11m %.6t %.12r %.20S %.20N" -S S
```

scontrol

`scontrol` will show you everything that the system knows about a running or pending job.

```
scontrol -d show job <job id>


[user@castor jobs]$ scontrol -d show job 400
JobId=400 Name=s1.job
UserId=user(100000) GroupId=scitas(11902)
Priority=111 Account=scitas QOS=normal
JobState=RUNNING Reason=None Dependency=(null)
Requeue=1 Restarts=0 BatchFlag=1 ExitCode=0:0
DerivedExitCode=0:0
RunTime=00:03:39 TimeLimit=00:15:00 TimeMin=N/A
SubmitTime=2014-03-06T09:45:27 EligibleTime=2014-03-06T09:45:27
StartTime=2014-03-06T09:45:27 EndTime=2014-03-06T10:00:27
PreemptTime=None SuspendTime=None SecsPreSuspend=0
Partition=serial AllocNode:Sid=castor:106310
ReqNodeList=(null) ExcNodeList=(null)
NodeList=c03
BatchHost=c03
NumNodes=1 NumCPUs=1 CPUs/Task=1 ReqS:C:T=*:*:*
Nodes=c03 CPU_IDs=0 Mem=1024
MinCPUsNode=1 MinMemoryCPU=1024M MinTmpDiskNode=0
Features=(null) Gres=(null) Reservation=(null)
Shared=OK Contiguous=0 Licenses=(null) Network=(null)
Command=/home/user/jobs/s1.job
WorkDir=/scratch/scitas/user
```

Sjob

`Sjob` is particularly useful to find out information about jobs that have recently finished.

```
[user@deneb2 jobs]$ Sjob  296176
       JobID    JobName    Cluster    Account  Partition  Timelimit      User     Group
------------ ---------- ---------- ---------- ---------- ---------- --------- ---------
296176           s5.job      deneb     scitas      debug   00:10:00      user    scitas
296176.batch      batch      deneb     scitas
296176.0          sleep      deneb     scitas
296176.1          sleep      deneb     scitas
             Submit            Eligible               Start                 End
------------------- ------------------- ------------------- -------------------
2015-10-26T15:10:37 2015-10-26T15:10:37 2015-10-26T15:10:38 2015-10-26T15:10:59
2015-10-26T15:10:38 2015-10-26T15:10:38 2015-10-26T15:10:38 2015-10-26T15:10:59
2015-10-26T15:10:38 2015-10-26T15:10:38 2015-10-26T15:10:38 2015-10-26T15:10:49
2015-10-26T15:10:49 2015-10-26T15:10:49 2015-10-26T15:10:49 2015-10-26T15:10:59
   Elapsed ExitCode      State
---------- -------- ----------
  00:00:21      0:0  COMPLETED
  00:00:21      0:0  COMPLETED
  00:00:11      0:0  COMPLETED
  00:00:10      0:0  COMPLETED
    NCPUS   NTasks        NodeList    UserCPU  SystemCPU     AveCPU  MaxVMSize
---------- -------- --------------- ---------- ---------- ---------- ----------
       16               r02-node01  00:00.053  00:00.066
       16        1      r02-node01  00:00.038  00:00.034   00:00:00    395556K
        1        1      r02-node01  00:00.008  00:00.014   00:00:00    210292K
        1        1      r02-node01  00:00.005  00:00.016   00:00:00    210292K
```

Modules and provided software

Modules (LMod) is utility that allows multiple, often incompatible, tools and libraries to exist on a cluster.

Scientific tools and libraries are provided as modules and you can see what is available by running the command ***module avail*** :

```
$ module avail
-------------- /path/to/base/modules ---------------
cmake  gcc  intel  matlab
```

Initially you will only see the base modules - these are either compilers or stand alone packages such as MATLAB.

In order to see more modules including libraries and MPI distributions you need to load a compiler, typically `gcc` or `intel`:

```
$ module load gcc

$ module avail
--------------- /path/to/gcc/modules ----------------
gdb  fftw  hdf5  mvapich2  openmpi  python R

-------------- /path/to/base/modules ---------------
cmake  gcc  intel  matlab
```

The full guide to how to use modules can be found here.

In your submission script we strongly recommend that you begin with the command **`module purge`** and then load the module you need to as to ensure that you always have the correct environment.

Examples of submission scripts

There are a number of examples available on a git repository. To download these run the following command from one of the clusters:

```
git clone https://c4science.ch/source/scitas-examples.git
```

Enter the directory scitas-examples and choose the example to run by navigating the folders.

We have three categories of examples:

1. Basic (examples to get you started)
2. Advanced (including hybrid jobs and job arrays)
3. Modules (specific examples of installed software).

To run an example (here: hybrid HPL), do:

```
sbatch --partition=debug hpl-hybrid.run
```

Or, if you do not wish to run on the debug partition:

```
sbatch hpl-hybrid.run
```

Running MPI jobs

MPI is the acronym for ***Message Passing Interface*** and is now the *de facto* standard for distributed memory parallelisation.

It's an open standard with multiple implementations and we are now at version 3.

There are multiple MPI flavours that comply with the specification and each claims to have some advantage over the other.

Some are vendor specific and others are open source.

On the SCITAS clusters we fully support the following compiler/MPI combinations (July 2018 until July 2019):

Intel 2018 with Intel MPI 2018

GCC 6.4 with MVAPICH2 version 2.3rc2

This is a SCITAS restriction to prevent chaos - nothing technically stops one from mixing! They all work well and have good performance.

If we have a MPI code we need some way of correctly launching it across multiple nodes. To do this we use srun which is a SLURM's built-in job launcher:

```
srun mycode.x
```

To specify how many ranks and the number of nodes, we add the relevant **#SBATCH** directives to the job script.

For example to launch our code on 4 nodes with 16 ranks per node we specify:

```
#!/bin/bash
#SBATCH --nodes 4
#SBATCH --ntasks-per-node 16
#SBATCH --cpus-per-task 1
#SBATCH --mem 32000
#SBATCH --time 1-0
module purge
module load mycompiler
module load mympi
srun /home/bob/code/mycode.x
```

There is no need to specify the number of ranks when you call srun as inherits the value from the allocation.

Running OpenMP jobs

When running an **OpenMP** or hybrid **OpenMP/MPI** job the important thing to set is the number of OpenMP threads per process via the variable ***OMP_NUM_THREADS***.
If this is not specified it often defaults to the number of processors in the system.

We can integrate this with SLURM as seen for the following hybrid (4 ranks, 4 threads per rank) task:

```
#!/bin/bash
#SBATCH --ntasks 4
#SBATCH --cpus-per-task 4
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
srun mycode.x
```

This takes the environmental variable set by SLURM and assigns the value to ***OMP_NUM_THREADS***.

If you run such hybrid jobs we advise you to read the page on **CPU affinity**.

The Debug Partition

All the clusters have a few nodes that only allow short jobs and are intended to give you quick access to allow you to debug jobs or quickly test input files.

To use these nodes you can either add the **`#SBATCH -p debug`** directive to your job script or specify it on the command line:

```
sbatch -p debug myjob.run
```

Please note that the debug nodes must not be used for production runs of short jobs.

Any such use will result in access to the clusters being revoked.

Interactive Jobs

There are two main methods of getting interactive (rather than batch) access to the machines. They have different use cases and advantages.

Sinteract

The ***Sinteract*** command allows you to log onto a compute node and run applications directly on it.

This can be especially useful for graphical applications such as *Matlab* and *Comsol*.

```
[username@frontend ]$ Sinteract
Cores: 1
Time: 00:30:00
Memory: 4G
Partition: serial
Jobname: interact
salloc: Granted job allocation 579438 salloc: Waiting for resource configuration salloc: Nodes z15 are ready
for job
[username@z15 ]$
```

On the Izar cluster, the -g option is necessary to request the desired number of GPUs. For example:

Sinteract -g gpu:1

Use of graphical applications (X11) on the clusters

To be able to use graphical applications, there are two requirements:

Connection to the cluster

You must connect from your machine to the login node with the **`-X`** option. (The use of **`-Y`** is unnecessary and highly discouraged as it is a security risk.)

```
ssh -X username@fidis.epfl.ch
```

Connection within the cluster (login node to compute nodes)

Additionally you **MUST** have password-less login configured from the login node to the node **WITH** the default **RSA** ssh key (`~/.ssh/id_rsa`).

To verify this requirement the following command **MUST** work **without any user intervention** (in this example, it is executed from the Fidis login node). In particular if you are prompted to enter your password after the `ssh` command, that is a failure.

```
[username@fidis.epfl.ch]$ salloc -p debug
[username@fidis.epfl.ch]$ ssh -i ~/.ssh/id_rsa $SLURM_NODELIST
```

In case the above does **NOT** work, please follow this procedure:

The following operation will destroy any existing keys, please contact us if you are unsure of what you are doing.

To set up password-less login within the cluster and **only** if you **do not yet have a default RSA ssh key (`~/.ssh/id_rsa`)**, please run the following two commands:

```
[username@fidis.epfl.ch]$ ssh-keygen -b 4096 -t rsa -N '' -C "${USER}@clusters" -f ${HOME}/.ssh/id_rsa

[username@fidis.epfl.ch]$ cat ${HOME}/.ssh/id_rsa.pub >> ${HOME}/.ssh/authorized_keys
```

In case you already have or need other keys installed (which have a password set as recommended), you should rename them and use a `.ssh/config` file to make sure they are used instead of the default key for any services that need them.

`salloc`

`salloc` creates a reservation on the system that you can then access via srun.

It allows you to run multi-node MPI jobs in an interactive manner and is very useful for debugging problems with such tasks:

```
[username@frontend ]$ salloc -N 2 -n 2 --mem 2048
salloc: Granted job allocation 579440 salloc: Waiting for resource configuration salloc: Nodes z[17,18] are
ready for job

[username@frontend ]$ hostname
frontend

[username@frontend ]$ srun hostname
z17
z18

[username@frontend ]$ exit
salloc: Relinquishing job allocation 579440
```

Interactive shell

To gain interactive access on the node, we suggest you use `Sinteract`.

If you wish to achieve a similar result with `salloc`, you can type, after having had access to your job allocation:

```
srun --pty bash
```

or, if you need a graphical display (see other preconditions above)

```
srun --x11 --pty bash
```

Related articles

- Using the clusters
- How to configure PuTTY SSH to transfer files to Cluster from Windows
- Connecting to the clusters
- FAQ
- Running Docker images using Shifter