Casper and Hactar Beginner User Manual

- Staff of HPC@Polito -

This guide is written for all users of our clusters
to answer all frequently asked questions about the usage of HPC systems
If you need more information
don't hesitate to contact us at
hpc.dauin@polito.it

Contents

Intended Audience

This guide will show the usage of the HPC@POLITO systems that are running SLURM as a scheduler. This document can be used for both clusters, is enough substitute to {login-node} the right address of the respective login node of the cluster you want to use. Login nodes are:

[Casper] *casperlogin.polito.it*

[Hactar] *hactarlogin.polito.it*

If you are familiar with SGE scheduler here a list of thing to keep in mind when using SLURM:

1. SLURM does not have *parallel environments* (`orte`, `shared` or anything else), you will use the directives `--nodes` and `--tasks-per-node` to specify the distribution of the resources that you need.
2. The SGE queue are called *partition* in SLURM.
3. SGE let you chose hard limit and virtual free memory size, instead in SLURM you specify parameters for memory (`--mem` and `--mem-per-cpu`), which is the limit for your real memory usage and drives the decision where your job is started. If you don't know how many memory your program will use let the scheduler uses its default value.
4. Some environment variables may have changed (for example SGE `$JOB_ID`, in SLURM become `$SLURM_JOB_ID`, or SGE `$NSLOTS` in SLURM become `$SLURM_NTASKS`), for a complete reference you could read https://slurm.schedmd.com/sbatch.html.

For a more detailed comparative table, reference to https://slurm.schedmd.com/rosetta.pdf

Gentlemen's Agreement

- By using our systems for research purpose, you automatically authorize HPC@POLITO's staff to publish your personal data (name, surname, research group) and data associated with your research on our web site (hpc.polito.it) and in all the other papers published by HPC@POLITO (annual reports, presentations, etc.).
- By using our systems for research purpose, you agree to quote the HPC@POLITO's center in all your scientific articles on paper, conference or books. We kindly suggest this kind of acknowledgement:

  "Computational resources provided by HPC@POLITO, which is a project of Academic Computing within the Department of Control and Computer Engineering at the Politecnico di Torino (http://www.hpc.polito.it)".

  Or a shorter version as
  "Computational resources provided by HPC@POLITO (http://www.hpc.polito.it)".

Rules for Cluster Uses

- It is NOT permitted to run commands like simulations and computations directly from the command line, due to existing risk to make *login-node* or other shared resources unusable (by running such commands your account will be blocked). You MUST always pass through scheduler's computational partitions (also for compilation purpose).
- Any uses of shared resources except didactical or research purposes are NOT permitted.
- Every user is responsible for activity done by his account, it is not recommended to share your account credentials with others, it is just allowed by previously informing HPC Staff.

  Generally, it's NOT permitted to share sensible data, especially username and password, with others. The user agree to hold them safely.
- As a general rule, it is NOT permitted to do something which is not clearly permitted.

This document contains only technical and practical information and must be considered only as a supplement for the complete HPC@POLITO regulation.
Updated versions of regulation or this technical manual can be reached on the project's website: *http://hpc.polito.it*.
All the users must take vision of the regulation and comply with it.

HPC@POLITO Academic Computing Center

Politecnico di Torino HPC project is an Academic Computing center which provides computational resources and technical support for research activities in accademic and didattical purposes.
The HPC project is officially managed by LABINF (Laboratorio Didattico di Informatica Avanzata) under supervision of DAUIN (Department of Control and Computer Engineering) which granted by Board of Directors.

Casper technical specification

Architecture	Linux Infiniband-DDR MIMD Distributed Shared-Memory Cluster
Node Interconnect	Infiniband DDR 20 Gb/s
Service Network Gigabit	Ethernet 1 Gb/s
CPU Model	2x Opteron 6276/6376 (Bulldozer) 2.3 GHz (turbo 3.0 GHz) 16 cores
Performance	4.360 TFLOPS
Power Consumption	7 kW
Computing Cores	512
Number of Nodes	16
Total RAM Memory	2.0 TB DDR3 REGISTERED ECC
OS	Centos 7.4 - OpenHPC 1.3.4
Scheduler	SLURM 17.11

Hactar technical specification

Architecture	Linux Infiniband-QDR MIMD Distributed Shared-Memory Cluster
Node Interconnect	Infiniband QDR 40 Gb/s
Service Network Gigabit	Ethernet 1 Gb/s
CPU Model	2x Xeon E5-2680 v3 2.50 GHz (turbo 3.3 GHz) 12 cores
GPU Node	2x Tesla K40 - 12 GB - 2880 cuda cores
Performance	20.18 TFLOPS (June 2018)
Computing Cores	696
Number of Nodes	29
Total RAM Memory	3.7 TB DDR4 REGISTERED ECC
OS	CentOS 7.4 - OpenHPC 1.3.4
Scheduler	SLURM 17.11

Storages technical specification

Home Storage	140 TB on RAID 6, throughput near 200 MB/s
Lustre Storage	87 TB. throughput greater than 2.2 GB/s
Storage Interconnect	Ethernet 10 Gb/s

Account and First Access

Your account has to be considered active from the moment you received confimation email containing your access credentials.
We strongly suggest you to change your password after first log-in by usign command following:

$ passwd

In any case, the password expires annually, users can renew it by using the change command.
There is no limitation about length and composition of password, in any case is suggested to choose a password which is a combination of at least 8 characters, numbers, uppercase and lowercase letters.

The account expires as indicated in the application form. To renew the account it is necessary to send again the same form sent in the first request taking care to modify the fields with updated information.

How get access to clusters? It depends on which operating system is used by your computer.
Assume that you are using Linux, Unix or OSX system, you can use a simple ssh client from any terminal with following:

$ ssh username@{login-node}

to access on our systems please enter username and password associated to you by means of confirmation email.

If you are using a Windows system we suggest you to use PuTTY software (available at http://www.putty.org) by setting the configurations as following:

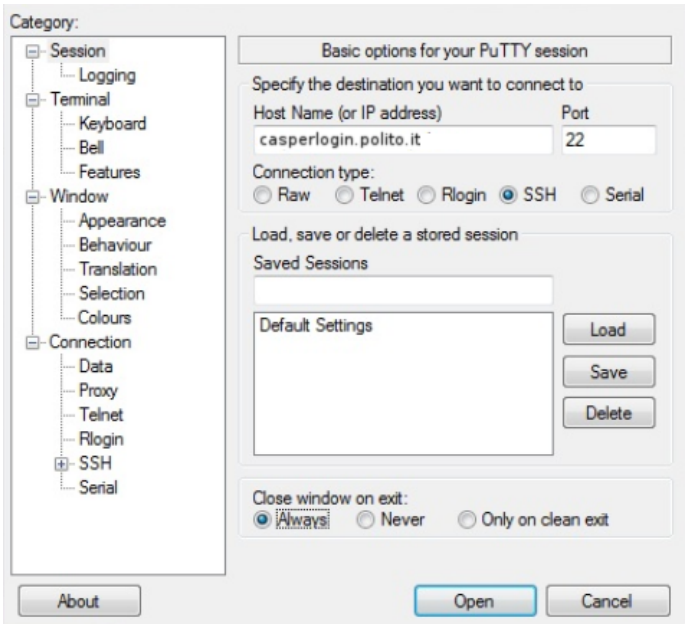

Figure 1: Putty - *PuTTY* Setting for access by Windows OS

Let us remind you that account sharing (the sharing of the same account among different users) is not permitted, moreover the person who made the request for the account has to take all the responsibility to keep the credentials safe: choose a strong password, don't share your sensible data with others. Your account should be secure, in any case if you observe unexpected account behaviors, please contact HPC's Staff immediately.

File Transfer & Storage

Home storage

After the access stage, the first showed directory is the user's *home*, located in the *Home Storage* and accessible by both clusters:

/home/user_name/

where the data have to be put in order to start a task and where the data will be written at the end of each task.
This directory can be accessed only by the owning user. Remember that in this storage each user can store at most 1 TB of data.
To copy files/directories inside your main directory you can use *scp* command as following:

$ scp -r /path/to/local/dir/ user_name@{login-node}:/home/user_name

Instead, to copy files from a CLUSTER on your local machine, use command following:

$ scp -r user_name@{login-node}:/home/user_name /path/to/local/dir/

It works similar as the *cp* command in Unix environment.

$ cp SOURCE DEST

NOTE: see the man page for *scp* command and its options

Third option is to use the SFTP server which is available within the cluster.
For example, you can use the FileZilla software configured as following.

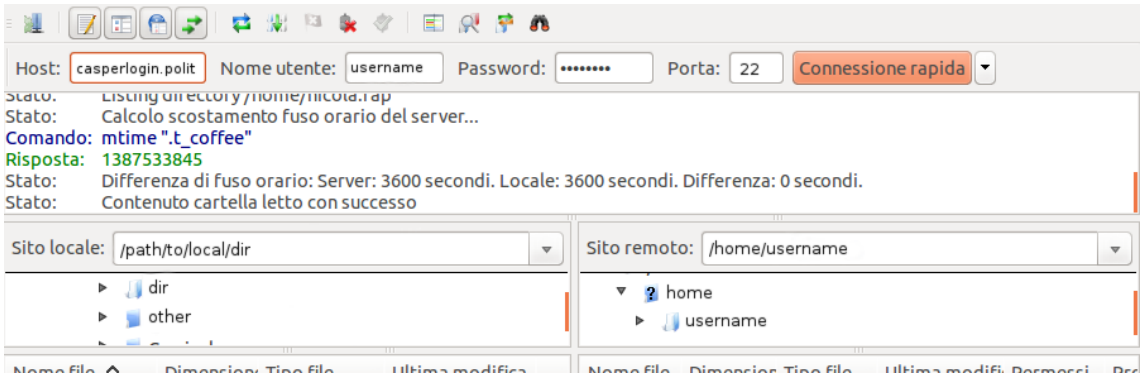

Figure 2: FileZilla configuration for file transfer on {login-node}

High performance storage

All users can access an additional high performance storage. It is provided with Lustre filesystem and it will improve all the tasks that make massive usage of I/O on files.

This storage is accessible by this path:

/work

This is as a secondary *home directory* containing for each user has a directory (named with his self username) and permissions to operate on it.

In order to move files from the working storage to the Lustre storage, you can use the *cp* command syntax as:

cp /home/user_name/file_name /work/user_name/file_name

cp -r /home/user_name/directory_name /work/user_name/directory_name

On this storage unit each user can wholly access 1TB of space (*soft quota*), but if strictly necessary it is possible to pass this threshold until 1.5TB (*hard quota*) for a short period (7 days).
To check your quota, use:

$ get-my-lustre-quota -h

The storage capacity is about 87TB and can reach a performance 1.1 GB/s used from CASPER and more than 2.2 GB/s from HACTAR.
Each user can send a request by email to obtain more space (extend their quota). The mail have to be sent to *hpc.dauin@polito.it* and contain a short explanation, the request will be evaluated by HPC@POLITO Staff.

Local Storage on Nodes

For each job it is possible to temporary use the additional space present on the local disk of the computational nodes.
This space can be accessed by the following path:

/scratch 1TB

inside the job is possible to refer to a specific temporary folder automatically created when the job is started on the node ad deleted at the end of the job. Environment variable $TMPDIR will contain the path to the temporary job folder.
PAY ATTENTION:

1. the space availability depends on others jobs' usage of the resource
2. is deprecated the usage of the path /scratch, use instead the automatic handled dir in the path `$TMPDIR`

Cluster Usage

The runnable processes on a cluster are "batch processes"; it means that are not interactive and execution of them can be postponed. Each task/job is composed by one or more processes that work together to achieve a certain result.
A job is executed after his schedulation; in order to schedule a job it must be inserted in a waiting list, managed by cluster, waiting to get the available resources held by oldest processes.
Our clusters uses the SLURM scheduler to manage the waiting queues and the availability of computational shared resources.

The public partitions for submitting the job are:

CASPER: **global** Default partition for public access, which includes all nodes; the maximum duration of a job is 10 days.

HACTAR: **global** Default partition for public access, which includes all nodes; the maximum duration of a job is 10 days.

cuda partition for GPU tasks (node *compute-1-14*), 2 GPU slots; the maximum duration of a job is 10 days. It is mandatory to use the option *--gres=gpu:{N_of_gpu}* (no curly braces)

Control and Submission of a Job

sbatch

All the jobs should be passed to the cluster scheduler through submission. It is possible to submit jobs using *sbatch* command which recevices as argument a reference to a script file containing all the required information.
The script file, know as *sbatch script*, is composed by a set of directives for the scheduler, followed by execution commands as should be appear on a command line.

An example of *sbatch* script is showed following, all lines start with `#SBATCH` are directives for the scheduler, and they are not interpreted from the Linux shell.

[script.sbatch]

```
#!/bin/bash
#SBATCH --job-name=job_name
#SBATCH --mail-type=ALL
#SBATCH --mail-user=name.surname@polito.it
#SBATCH --partition=global
#SBATCH --time=hh:mm:ss
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=32
#SBATCH --output=job_name_%j.log
#SBATCH --mem-per-cpu=1024M
example_task.bin
```

Now you can submit your job with the following command:

```
$ sbatch script.sbatch
```

In order to create the `script.sbatch` file you can boot create it on your local machine and copy it to the cluster, or edit the submition file directly on the cluster.
`NOTE:` Text files created on DOS/Windows machines have different line endings than files created on Unix/Linux. DOS uses carriage return and line feed as a line ending, while Unix uses just

line feed. You need to be careful about transferring files between Windows machines and Unix machines to make sure the line endings are translated properly.

Most Important Directives:

`--output`	the standard output is redirected to the file name specified, by default both standard output and standard error are directed to the same file.
`--error`	instruct Slurm to connect the batch script's standard error directly to the file name specified.
`--mail-user`	Email where the information on task will be sent.
`--mail-type`	Events which cause an email notification (*es. ALL*). Valid type values are NONE, BEGIN, END, FAIL, REQUEUE, ALL.
`--workdir={directory}`	If present cause the execution of task using the {directory} as working directory (the path of {directory} can be specified as full path or relative path).
`--ntasks-per-node`	number of tasks per node, if used with the `--ntasks` option, the `--ntasks` option will take precedence.
`--cpus-per-task`	number of CPU per task.
`--ntasks`	Total number of tasks for the job.
`--nodes`	Number of nodes that will be used.
`--time`	Indicates the hard run time limit, which is about the time that processes needs to reach the end. This value must be less than 10 days (<240 hours) for tasks on *global* partition.
`--mem-per-cpu`	Minimum memory required per allocated CPU, default units are megabytes (default=1000).
`--mem`	Specify the real memory required per node, default units are megabytes.
`--partition`	Indicates the partition where the job has to be scheduled (default=global).
`--exclude`	Explicitly exclude certain nodes from the resources granted to the job.
`--constraint`	Nodes have features assigned to them and users can specify which of these features are required by their job using this option, for example `--constraint="gpu"`. Available features can be shown in the output section *Scheduling pool data* of command `sjstat` (column *Other Traits*), or with `scontrol show node`.
`--gres=gpu:{N_of_gpu}`	Generic resource scheduling, used for specify the required number of GPU(s) per node .

For a complete reference you could read https://slurm.schedmd.com/sbatch.html.

squeue, sinfo, sjstat, scancel, sprio, sstat

In order to get information on the scheduled jobs with *sbatch*, as *job-ID*, *status*, *partition* and the used slots number, it's possible to use the followng command:

```
$ squeue -u username

      JOBID PARTITION     NAME     USER   ST       TIME     NODES  NODELIST(REASON)
        600    global     test   username  R   2-23:29:19      1   compute-0-1
        601    global     test   username  R      3:53:04      3   compute-0-[2-4]
        602    global     test   username  R         7:55      1   compute-0-8
```

To obtain more information about state of each specific job you can use:

```
$ scontrol show job 600
JobId=600 JobName=test
   UserId=test(500) GroupId=test(500) MCS_label=N/A
   Priority=30937 Nice=0 Account=test QOS=normal
   JobState=RUNNING Reason=None Dependency=(null)
   Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
   RunTime=04:07:05 TimeLimit=8-08:00:00 TimeMin=N/A
   SubmitTime=2018-02-13T10:25:11 EligibleTime=2018-02-13T10:25:11
   StartTime=2018-02-13T10:25:12 EndTime=2018-02-21T18:25:12 Deadline=N/A
   PreemptTime=None SuspendTime=None SecsPreSuspend=0
   Partition=global AllocNode:Sid=casperlogin:30786
   ReqNodeList=(null) ExcNodeList=compute-0-8
   NodeList=compute-0-[2-4]
   BatchHost=compute-0-2
   NumNodes=3 NumCPUs=96 NumTasks=96 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
   TRES=cpu=96,mem=240G,node=3
   Socks/Node=* NtasksPerN:B:S:C=32:0:*:* CoreSpec=*
   MinCPUsNode=32 MinMemoryNode=40G MinTmpDiskNode=0
   Features=(null) DelayBoot=00:00:00
   Gres=(null) Reservation=(null)
   OverSubscribe=OK Contiguous=0 Licenses=(null) Network=(null)
   Command=/home/test/test.bin
   WorkDir=/home/test/
   StdErr=/home/test/job_600.log
   StdIn=/dev/null
   StdOut=/home/test/job_600.log
   Power=
```

To obtain information about the entire cluster and the jobs executing on it, use:

```
$ squeue
```

Partition status can be checked by:

```
$ sinfo

PARTITION   AVAIL  TIMELIMIT  NODES  STATE NODELIST
global*        up 10-00:00:0      3    mix compute-1-[9,18,23]
global*        up 10-00:00:0     25  alloc compute-1-[1-8,10,12-17,19-22,24-29]
global*        up 10-00:00:0      1   idle compute-1-11
cuda           up 10-00:00:0      1  alloc compute-1-14
bioeda         up   infinite      1  alloc compute-1-15
desal          up   infinite      3  alloc compute-1-[16-17,26]
e3             up   infinite      1    mix compute-1-18
e3             up   infinite      4  alloc compute-1-[19-22]
srg            up   infinite      1    mix compute-1-23
srg            up   infinite      1  alloc compute-1-24
bluen          up   infinite      1  alloc compute-1-25
small          up   infinite      1  alloc compute-1-27
nqs            up   infinite      2  alloc compute-1-[28-29]
maintenance    up   infinite      3    mix compute-1-[9,18,23]
maintenance    up   infinite     25  alloc compute-1-[1-8,10,12-17,19-22,24-29]
maintenance    up   infinite      1   idle compute-1-11
```

or using:

```
$ sjstat

Scheduling pool data:
-------------------------------------------------------------
Pool        Memory  Cpus  Total Usable   Free  Other Traits
-------------------------------------------------------------
global*    128752Mb    32      4      4      0  local,public,amd6276
global*    128752Mb    32      3      3      0  local,private,amd6274
global*    128752Mb    32      1      1      1  local,private,amd6128
global*    128752Mb    32      3      3      3  local,private,amd6276
global*    128752Mb    32      5      5      5  local,private,amd6376
lisa       128752Mb    32      1      1      0  local,private,amd6274
bioeda     128752Mb    32      2      2      0  local,private,amd6274
bioeda     128752Mb    32      1      1      1  local,private,amd6128
nqs        128752Mb    32      3      3      3  local,private,amd6276
modena     128752Mb    32      2      2      2  local,private,amd6376
pt-erc     128752Mb    32      3      3      3  local,private,amd6376
maintenan  128752Mb    32      4      4      0  local,public,amd6276
maintenan  128752Mb    32      3      3      0  local,private,amd6274
maintenan  128752Mb    32      1      1      1  local,private,amd6128
maintenan  128752Mb    32      3      3      3  local,private,amd6276
maintenan  128752Mb    32      5      5      5  local,private,amd6376

Running job data:
---------------------------------------------------------------------------
JobID    User      Procs Pool      Status        Used  Master/Other
---------------------------------------------------------------------------
```

To view the components of all the jobs scheduling priority use:

```
$ sprio

          JOBID   PRIORITY        AGE  FAIRSHARE    JOBSIZE  PARTITION
```

useful Option:

`--jobs={jobID}`	Print the job priorities for specific job or jobs(comma separated list).
`--users={username}`	Print the job priorities for jobs of specific user or users(comma-separated list).

To stop and remove a job from a partition, use:

```
$ scancel jobID
```

To collect statistics about currently running jobs you could use:

```
$ sstat --format=JobID,MaxRSS,AveRSS,AveCPU,NTask -j $JOBID --allsteps
       JobID     MaxRSS     AveRSS     AveCPU   NTasks
------------ ---------- ---------- ---------- --------
JOBID.0           3248K      3248K  00:00.000        1
```

The above command works only for jobs executed through interactive method `srun`, for batch jobs append `.batch` to the `$JobID` like following:

```
$ sstat --format=JobID,MaxRSS,AveRSS,AveCPU,NTask -j ${JOBID}.batch --allsteps
       JobID     MaxRSS     AveRSS     AveCPU   NTasks
------------ ---------- ---------- ---------- --------
${JOBID}.batch       9488K      9488K  07:24.000        1
```

Interactive sessions on compute nodes

Any time when it is necessary to use an interactive session on a compute node could be helpful the usage of the *srun* command.
This is the one and only allowed method to have an opened remote session on a compute node.

Starting from the login node, use the command with the following syntax:

```
$ srun --nodes=1 --tasks-per-node=1 --pty /bin/bash
```

Command's options directly correspond with the directives of sbatch scripts (see *sbatch*).

Usage of *srun* command is strictly monitored as defined by SRUN ABUSE SUPPRESSION policy (see the generale rules).

An abuse is spotted when a *srun* session is signed running 'R' but actually does not execute any calculation. This could make the schedule equality policy uncertain.

The mechanism by which the "nasty" *srun* sessions is suppressed work in this way:
every day at 3:00 AM for any running *srun* it is calculated a certain value (ratio) as the average CPU usage divided by the number of requested CPU at submission time; if this value is below a certain threshold the job is terminated by mean of *scancel*.

Graphical Sessions and X11 Forwarding

By using *srun* command it is possible to create graphical interactive sessions on the computational nodes. To enable this feature it is required to initialize *X11 Forwarding* in the first connection to the login node using the command as following:

```
$ ssh -YC user_name@{login-node}
```

then use the command below to obtain a new session on a computational node and finally call the program intended to be used.

```
[Casper] $ srun --nodes=1 --tasks-per-node=1 --x11 --pty /bin/bash
```

The complete reference manual is available here: https://slurm.schedmd.com/

MPI job distribution

SLURM offers a large variety of option for configure how the resource should be reserved for the job. In the table below a summary of the main relevants one:

`--nodes`	Each server is referred as a node and is associated with the `--nodes` directive
`--ntasks`	Task corresponds with MPI ranks and is associated with the `--ntasks` directive
`--ntasks-per-node`	Give the possibility to control the number of task on one single node
`--cpus-per-task`	Set how many CPU per task will be reserved, when used with OpenMP represents the number of cores for single OpenMP process

MPI sbatch script example

MPI case 1 [mpi1.sbatch]

```
#!/bin/bash
#SBATCH --ntasks=16
#SBATCH --cpus-per-task=1
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=16
```

This example would request 16 tasks, corresponding to 16 MPI ranks. These will each have 1 core. All 16 tasks will be constrained to be on a single compute node.

MPI case 2 [mpi2.sbatch]

```
#!/bin/bash
#SBATCH --ntasks=32
#SBATCH --cpus-per-task=1
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=16
```

In this second example, 32 tasks are requested, and these are split between 2 nodes, 16 tasks on each.

MPI case 3 [mpi3.sbatch]

```
#!/bin/bash
#SBATCH --ntasks=32
```

In this third example, 32 tasks are requested and the scheduler decide how to distribute them.

MPI and OpenMP usecase [mpiOpenMP.sbatch]

```
#!/bin/bash
#SBATCH --ntasks=16
#SBATCH --cpus-per-task=4
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=4
```

In this case, we are running an application that uses multiple core for each MPI task or rank. We specify 16 tasks (`--ntasks=16`) and 4 cores per task (`--cpus-per-task=4`), for a total of 16*4=64 cores. The 16 tasks are split across 4 nodes (`--nodes=4`), with the `--ntasks-per-node=4` ensuring that they are distributed evenly across the nodes.

CUDA example

GPU CUDA example [cuda.sbatch]

```
#!/bin/bash
#SBATCH --time=24:00:00
#SBATCH --ntasks=1
#SBATCH --partition=cuda
#SBATCH --gres=gpu:1
#SBATCH --job-name=cudaExample
#SBATCH --mem=10GB
#SBATCH --mail-type=ALL
##
# load cuda module
module load nvidia/cudasdk/9.2
```

This example would request 1 GPU, in order to use the cuda functionality is necessary to load the CUDA module.

Scheduling and job priority

In all the clusters of the HPC@POLITO center, the resources are managed by the scheduler, which assigns the jobs to the available resources based on the current load and the job characteristics.
The algorithm used to manage queued jobs is multifactor with backfill. This mechanism goes into action in all those cases where the resources required for the start of the jobs are not immediately available and therefore the jobs remain in the queue, sorted in order of priority.
Jobs at the front of the queue will start as soon as the required resources are available, thanks to the backfill algorithm the scheduler starts minor priority jobs if this does not cause delays in starting the jobs with higher priority. For this algorithm to work as efficiently as possible, it is required that jobs have assigned a time limit that is as close as possible to the actual runtime time.

Priority computation

The factors that contribute to the calculation of the priority are the following:

- **Age Factor:** value directly proportional to the time spent in the queue.
- **Job Size Factor:** value directly proportional to the size (CPU, MEMORY, TIME) of the job, in this way it is guaranteed that jobs that require a lot of resources are able to start.
- **Partition Factor:** value used for the priority partitions together with the pre-emption of the resources.
- **Fair-share Factor:** calculated value with respect to the quantity of resources consumed in relation to the total resources available on a 30-day window.
- **TRES Factor:** assigned value if jobs require certain resources (GPU).
- **QOS Factor:** value calculated on the efficiency in the use of resources by the individual user on a 30-day mobile window.

QOS Factor

With the aim of encouraging the efficient and conscious use of available resources, the QOS Factor was introduced among the metrics. The algorithm used to determine the QOS Factor is the following: for each user the efficiency in the use of resources (CPU, memory, execution time) of each submitted job is evaluated on a 30-day mobile window by executing the division between

what actually consumed with the requested during submission. The calculated values are then normalized with respect to the maximum value obtained by users for the period in question.
The introduction of this metric aime at rewarding users who use resources more efficiently, and help reducing the problems listed below:

- **job with time limit not aligned with execution time:** inefficiency of the backfill algorithm, it is not possible to exploit portions of resources not used by the job with the highest priority waiting that all the resources necessary for its execution are freed with a consequent reduction in system throughput.
- **job with memory requests not aligned with what is actually consumed:** under-utilization of resources and longer waiting time in the queue, both as regards the job itself and regards the other jobs in the queue. Last but not least it heavily damages the pre-emption system on the priority queues because memory is a resource on which the suspension policy has no impact (the memory allocated by suspended processes continues to be allocated even in the event of suspension).
- **work with CPU requests not aligned with the expected consumption:** under-utilization of resources and longer waiting time in the queue.

To improve efficiency in the use of resources, you can use the **sacct** command os **seff** command to query the accounting system, or alternatively use the statistics in the notification emails at the end of the job.

Memory allocation

Slurm treats memory as a consumable resource, for this reason it is necessary to be precise in dimensioning the memory requests. Requests of memory not congruous to the effective use strongly penalize the overall throughput of the system, both in terms of resources usage and average time spent in queue. Given that those who do not have strong memory needs can safely use the default value set in the system (1GB per core), it could be done simply omitting the directives in which you specify the requests related to the memory(`--mem` or `--mem-per-cpu`), for all the others the following section will show how to size correctly the memory request.
If it is difficult or impossible for you to accurately estimate the amount of memory analytically, it is best to derive this information by deduction from previously completed similar jobs.

seff

To do this you can use the **seff** tool from the login node of both clusters:

```
[~]xxxxxxxx@hactarlogin$ seff $JOBID
Job ID: $JOBID
Cluster: hactar
User/Group: xxxxxxxx/xxxxxxxxx
State: COMPLETED (exit code 0)
Nodes: 1
Cores per node: 10
CPU Utilized: 10:17:10
CPU Efficiency: 10.00% of 4-06:53:00 core-walltime
Job Wall-clock time: 10:17:18
Memory Utilized: 8.43 GB
Memory Efficiency: 86.30% of 9.77 GB
```

The above mentioned tool is also used to generate at the end of job reports included in the e-mail notifications, if the user has requested them during submission. From the example above we notice that the Memory Efficiency is of 86% that is a reasonable value.

sacct

Another option is to query the accounting system via **sacct**:

```
[~]xxx@hactarlogin$ sacct --format \
JobID,JobName,Partition,AveRSS,MaxRSS,ReqMem,State,AllocCPUS -j $JOBID

       JobID    JobName  Partition     AveRSS     MaxRSS     ReqMem      State  AllocCPUS
------------ ---------- ---------- ---------- ---------- ---------- ---------- ----------
$JOBID            $JOBNAME  global                          1000Mc  COMPLETED         10
$JOBID.batch      batch              8596096K   8836936K     1000Mc  COMPLETED         10
```

where:

`ReqMem`	It is the memory required for SLURM. If the value has suffix *Mn* it indicates the memory per node in MB, if it has suffix *Mc* it indicates the memory per core in MB.
`MaxRSS`	It is the maximum memory used. The value is applied directly for jobs running on a single node, while for jobs running on multiple nodes it indicates the maximum memory used on each node on which the job was running, the value should be normalized for the number of cores used on that node.
`AveRSS`	Is the avarage used memory.
`Elapsed`	Is the execution time.

Note: Only statistics collected on completed jobs are to be considered reliable.

To obtain the amount of memory required per core, it is possible to divide the *MaxRSS* value by the number of cores requested per node or used on that node if the distribution of the processes is not uniform among the nodes, or directly use the value of *MaxRSS* if you use the –mem option. We recommend to multiply the value obtained with this method for 1.2 in order to tolerate variations between runs and do not incur unpleasant surprises.
For example, to make a memory request with a 20% more of margin than the used memory:

`--mem-per-cpu=` *(MaxRSS / cores-per-node) * 1.2*
`--mem=` *MaxRSS * 1.2*

We recommend that all users verify and validate their memory requests with this method in all cases where a memory request is specified greater than the default one (1GB per core). The administrator will carry out random checks on the active jobs, all the jobs in which the memory request will be more than twice the one actually used will be terminated without notice.

If your job is failing or is cancelled because of memory limit you could find some lines like the following at the end of the job's log file:

```
slurmstepd: error: Job $JOBID exceeded memory limit (2773388 > 1024000), being killed
slurmstepd: error: *** JOB $JOBID ON compute-x-x CANCELLED AT xxxx-xx-xxTxx:xx:xx ***
```

In this case you should elevate the memory limit from 1GB to 3GB for example (at 2773388KB minimum, but this is not recommended).

Environment Module System

The software *Modules* (*TACC Lmod*) enables dynamic modification of the environment variables during a user session.
The use of this software is strongly recommended in the *sbatch* script and *srun* session because it provides an easy way to use different versions of the same application; it allows user to export environment variables.
To load the needed modules add the right commands directly in the sbatch script.

Common *module* command options:

module list	List loaded modules
module avail	List available modules
module load {module-name}	Load the module
module unload {module-name}	Unload the module
module purge	Remove all loaded module

Example:

```
$ module avail

-------------------- /opt/ohpc/pub/moduledeps/gnu-mvapich2 ---------------------
   adios/1.11.0    mpiP/3.4.1              petsc/3.7.5        scorep/3.0
   boost/1.63.0    mumps/5.0.2             phdf5/1.8.17       sionlib/1.7.0
   fftw/3.3.4      netcdf-cxx/4.3.0        scalapack/2.0.2    superlu_dist/4.2
   hypre/2.11.1    netcdf-fortran/4.4.4    scalasca/2.3.1     tau/2.26.1
   imb/4.1         netcdf/4.4.1.1          scipy/0.19.0       trilinos/12.10.1

------------------------ /opt/ohpc/pub/moduledeps/gnu -------------------------
   gsl/2.2.1      mpich/3.2           ocr/1.0.1          pdtoolkit/3.23
   hdf5/1.8.17    mvapich2/2.2 (L)    openblas/0.2.19    superlu/5.2.1
   metis/5.1.0    numpy/1.11.1        openmpi/1.10.6

-------------------------- /opt/ohpc/pub/modulefiles --------------------------
   autotools        (L)    gnu/5.4.0  (L)    pmix/1.2.3
   clustershell/1.8        ohpc       (L)    prun/1.2        (L)
   cmake/3.9.2             papi/5.5.1        singularity/2.4

----------------------- /share/apps/casper-modulefiles ------------------------
   blender/2.79                       intel/python/2.7/2017.3.053
   converge/2.3.22                    intel/python/3.5/2017.3.052
   fire/2014.2                        lammps/16Feb16
   intel/libraries/daal/2017.4.239    matlab/2017b
   intel/libraries/ipp/2017.3.196     quantum-espresso/5.4.0
   intel/libraries/mkl/2017.4.239     quantum-espresso/6.2.1      (D)
   intel/libraries/mpi/2017.4.239     starccm+/12.06.011
   intel/libraries/tbb/2017.8.239

  Where:
   L:  Module is loaded
   D:  Default Module
```

Below there are some examples.
To load the modules required to execute *matlab*, use:

```
$ module load matlab/2017b
```

To automatically load needed modules based on the cluster where job is submitted.
Version of *starccm* is different on both clusters on CASPER the version "starccm+/12.02.010" is installed;
on HACTAR the version "starccm+/12.02.011" is installed.

Add in the submission script the following lines of code in order to load the right module for the software version:

```
if [ $SLURM_CLUSTER_NAME == "hactar" ]
then
    module load starccm+/12.02.011
else
    module load starccm+/12.02.010
fi
```

Ganglia Monitoring System

Ganglia is a scalable, distributed monitoring tool for high-performance computing systems, clusters and networks. The software is used to view either live or recorded statistics covering metrics such as CPU load averages or network utilization for many nodes.

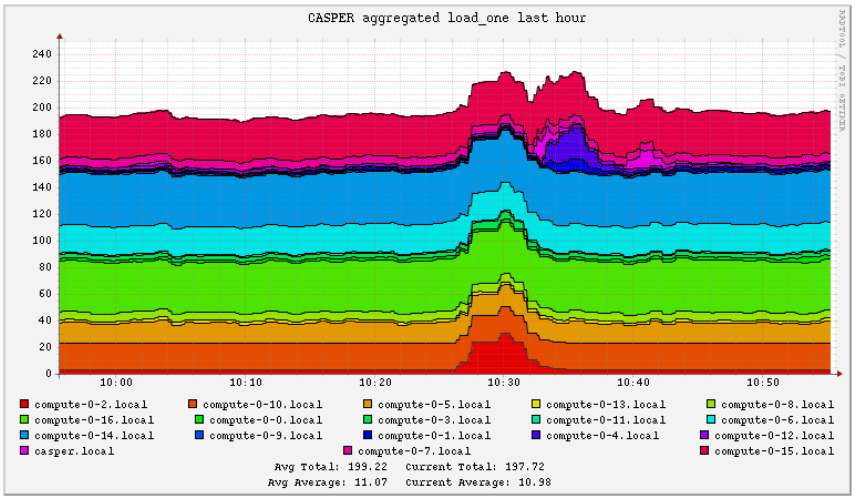

Figure 3: Ganglia - Ganglia example

Basically Ganglia is a web interface that provides information about the cluster's activities. It is available at following links:

[Casper] *http://casper.polito.it/ganglia/*

[Hactar] *http://hactar.polito.it/ganglia/*

APPENDIX A: sbatch script examples

It follows a list of script templates for some of the applications installed on clusters.

Matlab [matlab.sbatch]

```
#!/bin/bash
#SBATCH --job-name=job_name
#SBATCH --mail-type=ALL
#SBATCH --mail-user=nome.cognome@polito.it
#SBATCH --partition=global
#SBATCH --time=hh:mm:ss
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=32
#SBATCH --output=job_name_%j.log
#SBATCH --mem-per-cpu=1024M

module load matlab/2017b

matlab -r test_matlab
```

Blender [blender.sbatch]

```
#!/bin/bash
#SBATCH --job-name=job_name
#SBATCH --mail-type=ALL
#SBATCH --mail-user=nome.cognome@polito.it
#SBATCH --partition=global
#SBATCH --time=hh:mm:ss
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=32
#SBATCH --output=job_name_%j.log
#SBATCH --mem-per-cpu=1024M

module load blender/2.79

blender -noaudio -b BMW1M.blend -o BMW1M_out -F PNG -f 1 -t${SLURM_NTASKS}
```

Converge [converge.sbatch]

```
#!/bin/bash
#SBATCH --job-name=job_name
#SBATCH --mail-type=ALL
#SBATCH --mail-user=nome.cognome@polito.it
#SBATCH --partition=global
#SBATCH --time=24:00:00
#SBATCH --ntasks=96
#SBATCH --output=job_name_%j.log
#SBATCH --mem-per-cpu=2048M

module purge
module load converge/2.3.22

echo Starting on $SLURM_JOB_NODELIST with $SLURM_NTASKS processors

mpirun --bind-to core converge-2.3.22-openmpi-linux-64 $inFile > output
```

Quantum-espresso [qe.sbatch]

```
#!/bin/bash
#SBATCH --job-name=job_name
#SBATCH --mail-type=ALL
#SBATCH --mail-user=nome.cognome@polito.it
#SBATCH --partition=global
#SBATCH --time=24:00:00
#SBATCH --nodes=3
#SBATCH --ntasks-per-node=32
#SBATCH --exclude=compute-0-8
#SBATCH --output=job_name_%j.log
#SBATCH --mem-per-cpu=2048M

module purge

module load quantum-espresso/6.2.1

CASE_IN="ausurf.in"

echo Starting on $SLURM_JOB_NODELIST with $SLURM_NTASKS processors

prun pw.x -ntg 2 -ndiag 16 -input $CASE_IN
```

STAR-CCM+ [starccm.sbatch]

```
#!/bin/bash
#SBATCH --job-name=test
#SBATCH --mail-type=ALL
#SBATCH --mail-user=nome.cognome@polito.it
#SBATCH --partition=global
#SBATCH --time=24:00:00
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=32
#SBATCH --output=%x_%j.log
#SBATCH --error=%x_%j.err
#SBATCH --mem-per-cpu=1G

module purge
module load starccm+/12.06.011

#----------------------------
LICENSE_FILE='1999@flex.cd-adapco.com'
LIC_KEY='chiave_pod'
#----------------------------

echo '' > machinefile
for node in $(scontrol show hostnames $SLURM_JOB_NODELIST)
        do
        echo ${node}:${SLURM_NTASKS_PER_NODE} >> machinefile
        done

starccm+ -power -licpath $LICENSE_FILE -podkey $LIC_KEY -np $SLURM_NTASKS \
         -machinefile machinefile -mpi intel -mpiflags "-bootstrap slurm" \
         -fabric ibv -fabricverbose -batch-report -batch run $SLURM_JOB_NAME.sim
```

LAMMPS [lammps.sbatch]

```
#!/bin/bash
#SBATCH --job-name=nome_job
#SBATCH --mail-type=ALL
#SBATCH --mail-user=nome.cognome@polito.it
#SBATCH --partition=global
#SBATCH --time=20:00:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=32
#SBATCH --constraint=amd6274|amd6276|amd6376
#SBATCH --output=nome_job_%j.log
#SBATCH --mem-per-cpu=2048

module purge
module load prun/1.2
module load lammps/16Feb16

CASE_IN="in.chain"
echo Starting on $SLURM_JOB_NODELIST with $SLURM_NTASKS processors

prun lmp_mpi -echo screen -in $CASE_IN
```

APPENDIX B: generate submission scripts from a *csv* file

For advance tasks which requires multiple execution of the same program with different parameters, we suggest you to use following script which is able to generate multiple sbatch script file by starting from a text file containing all the parameters.

[generate.sh]

```
#!/bin/bash
#
list=$(cat ./parameters.csv)
#
for i in $list
    do
    parameter1=$(echo $i|cut -f 1 -d ',')
    parameter2=$(echo $i|cut -f 2 -d ',')
    cat << EOF > ./test-$parameter1-$parameter2.sbatch
    #!/bin/bash
    #SBATCH --job-name=job_name
    #SBATCH --mail-type=ALL
    #SBATCH --mail-user=nome.cognome@polito.it
    #SBATCH --partition=global
    #SBATCH --time=24:00:00
    #SBATCH --nodes=3
    #SBATCH --ntasks-per-node=32
    #SBATCH --output=job_name_%j.log
    #SBATCH --mem-per-cpu=2048M
    example.bin  $parameter1 $parameter2
    EOF
done
```

[parameters.csv]

```
A,1
A,2
B,1
B,2
```

APPENDIX C: Singularity container

In order to offer the maximun flexibility to the user, we introduced the possibility to run container inside the cluster.
The complete reference manual is available here: http://singularity.lbl.gov/user-guide
It follows an example to create and execute a container with TensorFlow.

Step1: create Singularity recipe file

There are several ways to create a container, one of tese is to start from a doker image. In this example we started from an official NVIDIA Docker image.
Create a file in your machine *tensorflowCentos.recipe* with the following content:

```
BootStrap: docker
From: nvidia/cuda:9.0-cudnn7-devel-centos7   # This is a comment

%runscript
   echo "Hello from tensorflow container"
   whoami

%post
   echo "Hello from inside the container"
   yum -y update && yum install -y epel-release
   yum install -y python-pip python-devel
   pip2 install matplotlib h5py pillow tensorflow-gpu keras scikit-learn
```

Step2: build Singularity container

There are multiple tipe of container, in this case we build a immutable img.

```
$ sudo singularity build  tensorflowCentos.img tensorflowCentos.recipe
```

Step3: copy the container to the cluster

```
$ scp tensorflowCentos.img {username}@{login-node}:/home/{login-node}/tensorflowCentos.img
```

Step4: create submission script

Enter the login node, and create the python file inside your home.
/home/{user_name}/helloTensorflow.py

```
import tensorflow as tf
hello = tf.constant("Hello, TensorFlow!")
sess = tf.Session()
print sess.run(hello)
exit()
```

Create a file *singularity.sbatch* that look like the following:

```
#!/bin/bash
#SBATCH --time=00:10:00
#SBATCH --ntasks=1
#SBATCH --partition=cuda
#SBATCH --gres=gpu:1
#SBATCH --job-name=singularityTest
#SBATCH --mail-type=ALL

module load singularity/2.4.5

singularity exec -s /bin/bash --nv tensorflowCentos.img bash -c 'python helloTensorflow.py'
```

Now submit the job with the sbatch command:

```
sbatch singularity.sbatch
```