Casper and Hactar Beginner User Manual

Staff of HPC@Polito - November 18, 2016

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

Gentlemen Agreements

- 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 unusable *login-node* or other shared resources (by running such commands your account will be blocked). You MUST always pass through scheduler's computational queues (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.

Casper - Cluster Appliance for Scientific Parallel Execution and Rendering

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

Casper technical specificaion:

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	544
Number of Nodes	17
Total RAM Memory	2.2 TB DDR3 REGISTERED ECC
OS	ROCKS Clusters 6.1
Scheduler	GridEngine 2011.11p1

Hactar

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	9.7 TFLOPS
Computing Cores	360
Number of Nodes	15
Total RAM Memory	1.9 TB DDR4 REGISTERED ECC
OS	CentOS 6.6
Scheduler	GridEngine 2011.11

Storage

Storages technical specification:

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

First Access

Your account has to be considered active from the moment you received confimation email containing your access credentials.

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:

[Casper] *$ ssh username@casperlogin.polito.it*

[Hactar] *$ ssh username@hactarlogin.polito.it*

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:

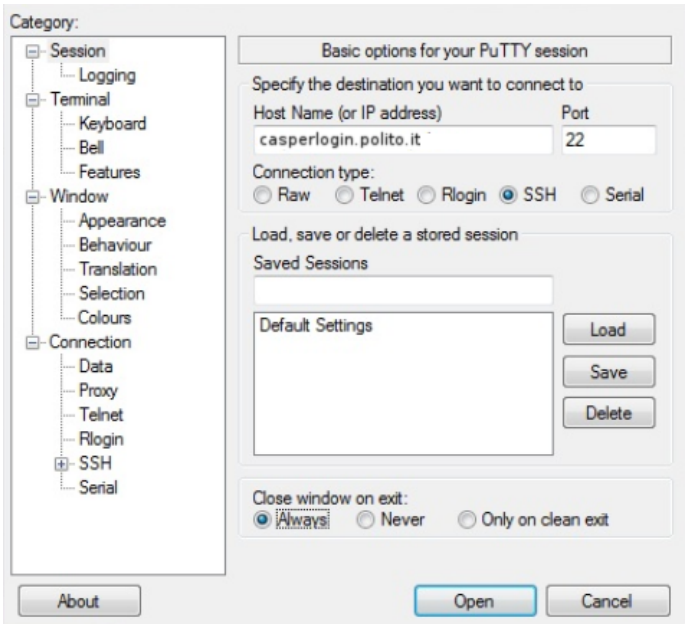

Figure 1: Putty - *PuTTY* Setting for CASPER access by Windows OS

We strongly suggest you to change your password after first log-in by usign command following:

$ passwd

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.

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 responsability 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 behaviours, please contact HPC's Staff immediately.

File Transfer

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

/home/username/

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 this storage each user can store at most 1 TB of data (eventually expandible).
To copy files/directories inside your main directory you can use *scp* command as following:

[Casper] *$ scp -r /path/to/local/dir/ username@casperlogin.polito.it:/home/username*

[Hactar] *$ scp -r /path/to/local/dir/ username@hactarlogin.polito.it:/home/username*

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

[Casper]*$ scp -r username@casperlogin.polito.it:/home/username /path/to/local/dir/*

[Hactar]*$ scp -r username@hactarlogin.polito.it:/home/username /path/to/local/dir/*

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

$ cp SOURCE DEST

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.

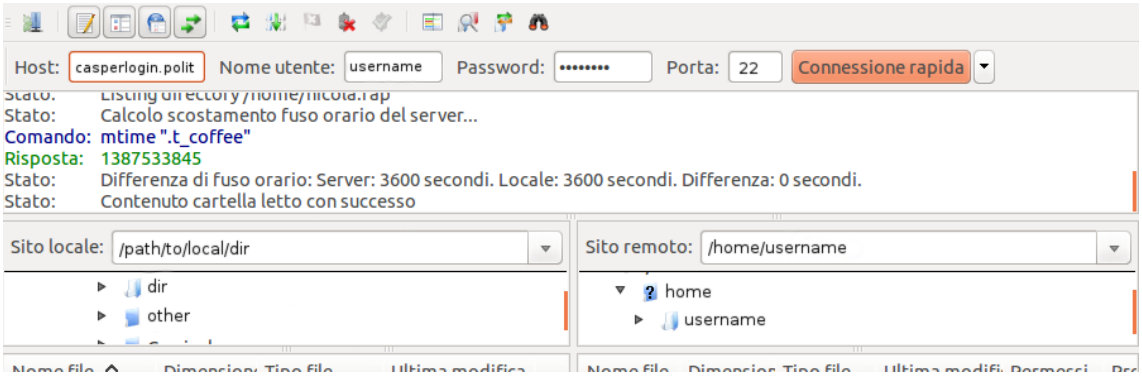

Figure 2: FileZilla configuration for file transfer on CASPER

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 mail 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 some jobs it is possible to temporary use the addiction space present on the local disk of the computational nodes.
This space can be accesses by following the path:

[Casper] */state/partition1* *210GB*

[Hactar] */scratch* *1TB*

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.
CASPER uses the Grid Engine scheduler to manage the waiting queues and availability of the computational shared resources.

There are two different queues for public access on CASPER:

- all.q — Default queue for public access, which includes all nodes (544 Cores on 17 nodes); the maximum duration of a job is 10 days.
- fast.q — High priority queue, 8 Cores on 2 nodes; the maximum duration of a job is 60 min.

The jobs that requires low computation and low resources should be inserted in the *fast.q* which deals with jobs launched for testing activity or with very low execution time.

There are two different queues for public access also on HACTAR:

- all.q — Default queue for public access, which includes all nodes (360 Cores on 15 nodes); the maximum duration of a job is 10 days.
- cuda.q — High priority queue for GPU task (compute-1-14), 2 Cores on 1 node + 2 GPU slot on a node; the maximum duration of a job is 10 days.

The *cuda.q* was planned for all the tasks that need GPU for their calculations (*rendering, CUDA, etc.*) Refer to appendix A to know how to submit job on this queue.

Controll and Submission of a Job
- qsub, qstat, qdel, qhost -

QSUB

All the jobs should be passed to the cluster scheduler through submission. It is possible to submit jobs using *qsub* command which recevices as argumnet a reference to a script file containing all the required information.
The script file, know as *qsub script*, is mainly composed by a set of directives and an execution command as should be appear on a command line. Some directives are mandatory(*), otherwise the job will stay in "*qw*" (waiting queue) indefinitely.
An example of this file is showed following:

[script.qsub]

```
#!/bin/bash
#$ -N Task_Name
#$ -M user_email@mail.com
#$ -m abes
#$ -cwd
#$ -j y
#$ -S /bin/bash
#$ -pe shared 32
#$ -l h_rt=HH:MM:SS
#$ -l virtual_free 16G
#$ -q all.q
example_task.bin
```

Directives:

-M	Email where the information on task will be sended.
-m	Events which cause an email notification (*es. abes* - abort, begin, end, suspend).
-cwd	If present cause the execution of task using the current dir as working dir.
-j y|n	Indicates if redirects or not stderr on output file.
-S	The preferred command interpreter.
-pe	(*) Specifies parallel environment in which the task will be executed. Any job requires a certain number of slots for execution, that can be "chosen" by the same node (max. 32) or more belonging to what is specified on this directive. The most commons are *shared*, *orte* and *mpirr*: **shared** - When it's required to execute many threads or processes on a single multi-core machine specifying the number of slot (*ex. -pe shared 16*). **orte** - When it's required to execute many parallel processes (usually MPI) on many nodes, but it tries to maximize the load on each one of them; Allocation Rule: "fill-up" (*ex. -pe orte 16*). **mpirr** - When it's required parallelize many processes using a certain number of slots, it tries to increase the distribution of each processes on many nodes; Allocation Rule: "round robin" (*es. -pe mpirr 16*). **mpisp** - When it's required to execute a certain number of MPI processes on each available node. Usefull for tasks that uses both MPI and OpenMP (example page 11).
-l h_rt	(*) Indicates the hard run time limit, which is about the time that processes needs to reach the end. This time is used by schedule to manage processes queue. This value must be less than 10 days (<240) for tasks on *all.q* queue.
-l virtual_free	Memory requirement for task execution.
-q	(*) Indicates the queue where each tasks has to be scheduled (for the most of the tasks use *all.q*).
-l gpu	Number of gpu asked to the gpu node (HACTAR only). Use this within cuda.q, *a*n exhaustive example on appendix A.

QSTAT

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

$ qstat

```
job-ID  prior   name   user    state submit/start at    queue                  slots
-----------------------------------------------------------------------------------
494180 0.51008 test   username  r  01/01/2015 00:00:00 all.q@compute-0-0.local  4
494182 0.51008 test   username  r  01/01/2015 00:00:00 all.q@compute-0-0.local  4
494183 0.51008 test   username  r  01/01/2015 00:00:00 all.q@compute-0-0.local  4
494184 0.51008 test   username  r  01/01/2015 00:00:00 all.q@compute-0-0.local  4
```

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

$ qstat -j {job-ID}

```
$ qstat -j 123456
==============================================================
job_number:                 123456
exec_file:                  job_scripts/123456
submission_time:            Thu Jan 01 00:00:00 2015
owner:                      username
uid:                        000
group:                      usergroup
gid:                        000
sge_o_home:                 /home/username
sge_o_log_name:             username
sge_o_path:                 ...
sge_o_shell:                /bin/bash
sge_o_workdir:              /home/username/...
sge_o_host:                 casper
account:                    sge
cwd:                        /home/username/...
merge:                      y
hard resource_list:         h_rt=720000
mail_list:                  username@casper.local
notify:                     FALSE
job_name:                   jobname
jobshare:                   0
hard_queue_list:            all.q
shell_list:                 NONE:/bin/bash
script_file:                run
parallel environment:       orte range: 32
usage    1:                 cpu=185:03:51:38, mem=2380221.15046 GBs, ...
scheduling info:            ...
```

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

*$ qstat -u **

Queues status can be checked by:

$ qstat -g c

```
CLUSTER QUEUE                   CQLOAD   USED    RES  AVAIL  TOTAL aoACDS  cdsuE
--------------------------------------------------------------------------------
all.q                             0.31    214      0    330    544      0      0
bioeda.q                          0.40     18      0     78     96      0      0
fast.q                            0.43      0      0     36     36      0      0
...
```

You can also check the queues status by node using:

$ qstat -f

```
queuename                       qtype resv/used/tot. load_avg arch          states
---------------------------------------------------------------------------------
all.q@compute-1-1               BIP   0/0/24         0.00     linux-x64
---------------------------------------------------------------------------------
all.q@compute-1-10              BIP   0/0/24         0.00     linux-x64
---------------------------------------------------------------------------------
all.q@compute-1-11              BIP   0/0/24         0.00     linux-x64
---------------------------------------------------------------------------------
all.q@compute-1-12              BIP   0/0/24         0.00     linux-x64
---------------------------------------------------------------------------------
all.q@compute-1-13              BIP   0/0/24         0.02     linux-x64
---------------------------------------------------------------------------------
...
```

QDEL

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

$ qdel {job-ID}

QHOST

The *qhost* command allows you to know state of nodes on the cluster at a given time:

$ qhost

```
HOSTNAME                ARCH         NCPU  LOAD  MEMTOT  MEMUSE  SWAPTO  SWAPUS
-------------------------------------------------------------------------------
global                  -               -     -       -       -       -       -
compute-0-0             linux-x64      32  2.00  126.2G    2.1G 1000.0M  394.7M
compute-0-1             linux-x64      32  2.41  126.2G   32.0G 1000.0M  601.8M
compute-0-10            linux-x64      32 20.02  126.2G    5.1G 1000.0M  166.9M
compute-0-11            linux-x64      32  1.03  126.2G    3.8G 1000.0M  210.4M
compute-0-12            linux-x64      32  2.02  126.2G    5.7G 1000.0M  336.6M
compute-0-13            linux-x64      32  2.04  126.2G    5.2G 1000.0M  174.6M
...
```

Moreover is possible to get information about distribution of a user's jobs on the cluster nodes by using:

$ qhost -u username

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 *qlogin* command.
This is the one and only allowed method to have an opened remote session on a compute node.

QLOGIN

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

$ qlogin -q {all.q} -pe shared {n.slot} -l h_rt {HH:MM:SS}

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

Usage of *qlogin* command is strictly monitored as defined by QLOGIN ABUSE SUPPRESSION policy.

An abuse is spotted when a *qlogin* 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" *qlogin* sessions is suppressed work in this way:
every day at 3:00 AM for any running *qlogin* 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 *qdel*.
The actual threshold is 30% (ratio = 0.3).

```
Example 1:
JOB-ID = 300112
STARTED AT = 1395741955 (UNIX time)
ELAPSED TIME = 106900 s
CPU SECONDS = 607772 s
LOAD\_AVG = 5.68
REQUESTED CPU = 32
RATIO = .17
---------------> KILLED

Example 2:
JOB-ID = 300113
STARTED AT = 1395741955 (UNIX time)
ELAPSED TIME = 106900 s
CPU SECONDS = 1215544 s
LOAD\_AVG = 11.36
REQUESTED CPU = 32
RATIO = .35
---------------> KEEP ALIVE
```

Graphical Sessions and X11 Forwarding

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

[Casper] *$ ssh -XC username@casperlogin.polito.it*

[Hactar] *$ ssh -XC username@hactarlogin.polito.it*

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

References

The complete reference manual is available here:

http://gridscheduler.sourceforge.net/htmlman/manuals.html

MODULES

The software *modules* (*Environment Modules Project*) enables dynamic modification of the environment variables during a user session.
The use of this software is strongly recommended in the *qsub script* because it provides an easy way to use different versions of the same application; it allows user to export environment variables.

You can get a list of all the available modules through the command following:

username@login-0-0$ module avail

The modules can be loaded (*load*) or unloaded (*unload*) from current user session. For example, to load the modules required to enable *openmpi over infiniband*, you can use:

[Casper] *username@login-0-0$ module load OpenMPI/1.6.2/gcc/module*

[Hactar] *username@login-0-0$ module load OpenMPI/1.8.4/gcc/module*

To remove all the current loaded modules from the session, you can use the command:

*username@login-0-0$ module unload **

To get the list of currently loaded modules you can use:

username@login-0-0$ module list

To be effective, in the most of the cases, the modules' commands should be directly added in the qsub script.

For example, to use *openmpi over infiniband* it's required to add on the qsub script following line:

[Casper] *module load OpenMPI/1.6.2/gcc/module*

[Hactar] *module load OpenMPI/1.8.4/gcc/module*

GANGLIA

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

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

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

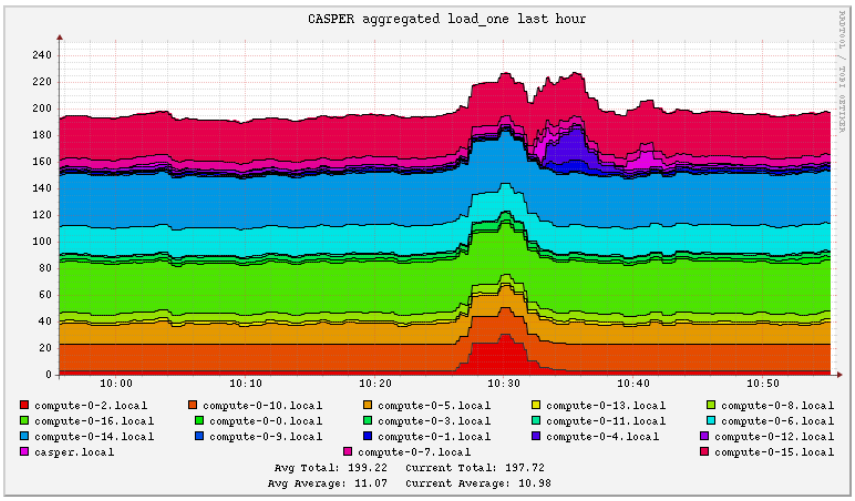

Figure 3: Ganglia - Ganglia example

APPENDIX A

It follows a list of script templates for some of the most used applications installed on CASPER.

Matlab [matlab.qsub]

```
#!/bin/bash
#$ -N test_matlab
#$ -M mail_utente@mail.com
#$ -m abes
#$ -cwd
#$ -j y
#$ -S /bin/bash
#$ -pe shared 16
#$ -q all.q
#$ -l h_rt=00:10:00

module load matlab/R2014b/module

matlab -r test_matlab
```

Blender [blender.qsub]

```
#!/bin/bash
#$ -N test_blender
#$ -M mail_utente@mail.com
#$ -m beas
#$ -cwd
#$ -j y
#$ -S /bin/bash
#$ -pe shared 24
#$ -q all.q
#$ -l h_rt=01:00:00

module load blender/2.76b/module

blender -noaudio -b BMW1M-MikePan.blend.1 -o Cycles_Benchmark -F PNG -f 1 -t 32
```

Comsol [comsol.qsub]

```
#!/bin/bash
#$ -N comsol_test
#$ -M mail_utente@mail.com
#$ -m abe
#$ -cwd
#$ -S /bin/bash
#$ -pe shared 16
#$ -q all.q
#$ -l virtual_free=64G
#$ -l h_rt=120:00:00

module load comsol/5.0/module

comsol -np $NSLOTS batch -inputfile input_file.mph -outputfile output_file.mph
```

Cplex [cplex.qsub]

```
#/bin/bash
#$ -N test_cplex
#$ -M mail_utente@mail.com
#$ -m abes
#$ -cwd
#$ -j y
#$ -S /bin/bash
#$ -pe shared 8
#$ -q all.q
#$ -l h_rt=01:00:00
#$ -l virtual_free=32G

module load cplex/12.6/module

cplex < problem.lp.cplex > problem.cplex.out
```

problem.lp.cplex

```
  read problem.lp
  set timelimit 41472
  set mip display 0
  set mip tol mipgap 0
  set parallel 1
  set threads 8
  set workmem 8192
  set mip limits treememory 32696
  optimize
  quit
```

MPI PROCESSES

MPI parallel processes can be distributed on Casper with two different allocation rules: *fill-up* or *round-robin*.

Fill-up tries to "fill" the node, exploiting the available (per node) resources and accelerates processes executions. It is enabled on *orte* Parallel Environment (PE).

Round-robin tries to spread processes over all available nodes, that is, each process potentially can reach more resources. It is enabled on *mpirr* Parallel Environment (PE).

MPI (FILL-UP) [mpi.qsub]

```
#$ -M mail_utente@mail.com
#$ -m abes
#$ -N mpi_quicksort
#$ -cwd
#$ -j y
#$ -S /bin/bash
#$ -pe orte 96
#$ -q all.q
#$ -l h_rt=05:00:00
#$ -l virtual_free=16G

module load OpenMPI/1.8.4/gcc/module

time /opt/openmpi/bin/mpirun -np 96 mpi_quicksort.x input_21GB.txt output.txt
```

MPI (ROUND-ROBIN) [mpirr.qsub]

```
#!/bin/bash
#$ -M mail_utente@mail.com
#$ -m abes
#$ -N mpi_quicksort
#$ -cwd
#$ -j y
#$ -S /bin/bash
#$ -pe mpirr 16
#$ -q all.q
#$ -l h_rt=05:00:00
#$ -l virtual_free=16G

module load OpenMPI/1.8.4/gcc/module

time /opt/openmpi/bin/mpirun -np 16 mpi_quicksort.x input_21GB.txt output.txt
```

In some cases it is required to exploit a certain number of MPI processes per node (e.g. one process on each node). It is possible to use the *mpisp* Parallel Environment (PE) for this scope.
(*Actually available only on CASPER*)

The mechanism is exploited by mean of a script.

The following data have to be passed at *mpirun* command:

- a list of available host (e.g. *-machinefile $JOB_ID.machines*)
  - *$JOB_ID.machines* is a file automatically generated by the Scheduler for this purpose;
- the number of processes involved (e.g. *-np $(($NHOSTS*$NPN))*
  - *$NHOST*, is the number of available host on a given moment;
  - *$NPN*, is the number of processes per node;
- number of processes per node (e.g. *-npernode $NPN*)

Finally the script shall appear as following:

..... MPI (NUMBER OF PROCESSES PER NODE) [mpisp.qsub] .....

```
#!/bin/bash
#$ -S /bin/bash
#$ -pe mpisp 16
#$ -cwd
#$ -V
#$ -N mpitest
#$ -q all.q
#$ -l h_rt=1:00:00

#############################################################
NPN=1 # set here the number of processes per node
#############################################################

module load OpenMPI/1.6.2/gcc/module

# $JOB_ID.machines, file containing the avaible host list, generated by sge
# $NHOSTS, number of avaible hosts passed by sge

mpirun -machinefile $JOB_ID.machines -np $(($NHOSTS*$NPN)) -npernode $NPN mergesort
```

.....

GPU [gpu.qsub]

```
#!/bin/bash
#$ -N cuda_task
#$ -M mail_utente@mail.com
#$ -m abes
#$ -cwd
#$ -S /bin/bash
#$ -pe shared 2
#$ -q cuda.q
#$ -l h_rt=05:00:00
#$ -l virtual_free=16G
#$ -l gpu=1

module load nvidia-cuda/7.5/module

./cuda-task-bin
```

The submission of job on this queue *cuda.q* is the correct way in which is possible to exploit GPU node.
-l gpu is a directive that specify the number of GPU that is intended to use for the next simulation (*actually it can only be 1 or 2*).

APPENDIX B

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 qsub 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-$parameter2.qsub
#/bin/bash
#$ -N task-$parameter1-$parameter2
#$ -M user_mail@mail.com
#$ -m abes
#$ -cwd
#$ -j y
#$ -S /bin/bash
#$ -pe shared 32
#$ -q all.q
#$ -l h_rt=01:00:00
#$ -l virtual_free=32G
example.bin  $parameter1 $parameter2
EOF
done
```

[parameters.csv]

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