
# Introduction to PICO

## Parallel & Production Enviroment

**Mirko Cestari** – m.cestari@cineca.it
**Alessandro Marani** – a.marani@cineca.it
**Domenico Guida** – d.guida@cineca.it
**Nicola Spallanzani** – n.spallanzani@cineca.it

SuperComputing Applications and Innovation Department

**BOLOGNA BUSINESS SCHOOL**
Alma Mater Studiorum Università di Bologna

# OUTLINE

- **A first step:**
  - **System overview**
  - **Login**
  - **Work environment**
- Production environment
  - Our first job!!
  - Creating a job script
  - Accounting and queue system
  - PBS commands
- Programming environment
  - Module system
  - Serial and parallel compilation
  - Interactive session
- For further info...
  - Useful links and documentation

# PICO CHARACTERISTICS

**Computing nodes**
**Processor**: Xeon E5 2670 v2 (2.5 Ghz)
**Number of processors (cores)**: 1320
**Number of nodes**: 66 (20 cores per node)
**RAM**: 128 GB/node

**Visualization nodes**
**Processor**: Xeon E5 2670 v2 (2.5 Ghz)
**Number of nodes**: 2 (20 cores per node)
**GPU:** Nvidia K40 (2 per node)
**RAM**: 128 GB/node

**“Big mem” nodes**
**Processor**: Xeon E5 2650 v2 (2.6 Ghz)
**Number of nodes**: 2 (16 cores per node)
**GPU:** Nvidia K20 (for 1 node)
**RAM**: 512 GB/node

**“Big insight” nodes**
**Processor**: Xeon E5 2650 v2 (2.6 Ghz)
**Numbero of nodes**: 4 (16 cores per node)
**Local disk space:** 32 TB
**RAM**: 64 GB/node

# HOW TO LOG IN

Establish a ssh connection

**ssh <username>@login.pico.cineca.it**

Remarks:

- **ssh** available on all linux distros
- **Putty** (free) or **Tectia** ssh on Windows
- *secure shell plugin* for Google Chrome!
- login nodes are swapped to keep the load balanced
- important messages can be found in the *message of the day*

Check the **user guide**!

https://wiki.u-gov.it/confluence/display/SCAIUS/UG3.3%3A+PICO+UserGuide

# WORK ENVIRONMENT

## $HOME:

Permanent, backed-up, and local to PICO.

50 Gb of quota. For source code or important input files.

## $CINECA_SCRATCH:

Large, parallel filesystem (GPFS).

No quota. Run your simulations and calculations here.

A cleaning procedure automatically deletes every file untouched since 50 days

## $WORK:

Similar to $CINECA_SCRATCH, but the content is shared among all the users of the same account.

1 Tb of quota. Stick to $CINECA_SCRATCH for the school exercises!

use the command cindata to get info on your disk occupation

https://wiki.u-gov.it/confluence/display/SCAIUS/UG2.4%3A+Data+storage+and+FileSystem

# OUTLINE

- A first step:
  - System overview
  - Login
  - Work environment
- **Production environment**
  - **Our first job!!**
  - **Creating a job script**
  - **Accounting and queue system**
  - **PBS commands**
- Programming environment
  - Module system
  - Serial and parallel compilation
  - Interactive session
- For further info...
  - Useful links and documentation

# LAUNCHING JOBS

As in every HPC cluster, PICO allows you to run your simulations by submitting **“jobs”** to the compute nodes

Your job is then taken in consideration by a **scheduler**, that adds it to a queuing line and allows its execution when the resources required are available

The operative scheduler in PICO is **PBS**

# PBS JOB SCRIPT SCHEME

The scheme of a PBS job script is as follows:

**#!/bin/bash**

**#PBS keywords**

**variables environment**

**execution line**

# PBS JOB SCRIPT EXAMPLE

```
#!/bin/bash
#PBS -N myname
#PBS -o job.out
#PBS -e job.err
#PBS -m abe
#PBS -M user@email.com
#PBS -l walltime=00:30:00
#PBS -l select=1:ncpus=20:mpiprocs=10:mem=10GB
#PBS -A <my_account>

echo “I’m working on PICO!”
```

# PBS KEYWORD ANALYSIS - 1

**#PBS -N myname**

Defines the name of your job

**#PBS -o job.out**

Specifies the file where the standard output is directed (default=jobname.o<jobID>)

**#PBS -e job.err**

Specifies the file where the standard error is directed (default=jobname.e<jobID>)

**#PBS -m abe (optional)**

Specifies e-mail notification. An e-mail will be sent to you when something happens to your job, according to the keywords you specified (a=aborted, b=begin, e=end, n=no email)

**#PBS -M user@email.com (optional)**

Specifies the e-mail address for the keyword above

# PBS KEYWORD ANALYSIS - 2

**#PBS -l walltime=00:30:00**

Specifies the maximum duration of the job. The maximum time allowed depends on the queue used (more about this later)

**#PBS -l select=1:ncpus=36:mpiprocs=18:mem=10GB**

Specifies the resources needed for the simulation.

**select** – number of compute nodes (“chunks”)

**ncpus** – number of cpus per node (max. 36)

**mpiprocs** – number of MPI tasks per node (max=ncpus)

**mem** – memory allocated for each node (default=3.5 GB)

# ACCOUNTING SYSTEM

**#PBS -A <my_account>**

Specifies the account that consumes the CPU hours allocated.

As an user, you have access to a limited number of CPU hours to spend. They are not assigned to users, but to **projects** and are shared between the users who are working on the same project (i.e. your research partners). Such projects are called **accounts** and are a different concept from your username.

You can check the status of your account with the command "*saldo -b*", which tells you how many CPU hours you have already consumed for each account you're assigned at (a more detailed report is provided by "*saldo -r*").

```
[amarani0@r000u07l02 ~]$ saldo -b

--------------------------------------------------------------------------------------------------------------------------
account            start        end        total       localCluster   totConsumed     totConsumed     monthTotal     monthConsumed
                                           (local h)   Consumed(local h)   (local h)            %     (local h)          (local h)
--------------------------------------------------------------------------------------------------------------------------
cin_staff          20110323    20200323   800000008          2017268     53875045             6.7     7299270             18527
smr_prod           20130308    20171215    12000000           353600      7536365            62.8      206540             55202
cin_priorit        20131115    20191231     4000000          1197301      3622341            90.6       53643             58183
cin_external       20150319    20201231       40000            14462        17972            44.9         567                 0
FUSIO_TEST         20161116    20200801          40               47           47           119.1           0                 0
train_scA2017      20170213    20170305        6000                0            0             0.0           0                 0
[amarani0@r000u07l02 ~]$
```

# ACCOUNT FOR THE COURCE

The account provided for this course is "**train_mdatsc17**"

(you have to specify it on your job scripts).

It will expire two months after the end of the school and is shared between all the students; there are plenty of hours for everybody, but don't waste them!

**#PBS -A train_mdatsc17**

# PBS COMMANDS

After the job script is ready, all there is left to do is to submit it:

**qsub**

qsub <job_script>

Your job will be submitted to the PBS scheduler and executed when there will be nodes available (according to your priority and the queue you requested)

**qstat -u**

qstat -u <username>

Shows the list of all your scheduled jobs, along with their status (idle, running, closing, …) Also, shows you the job id required for other PBS commands.
Hint: add the flag “-w” for an extended vision and the full name of your jobid

# PBS COMMANDS

**qstat -f**

qstat -f <job_id>

Provides a long list of informations for the job requested.
In particular, if your job isn’t running yet, you'll be notified about its estimated start time or, if you made an error on the job script, you will learn that the job won’t ever start

**qdel**

qdel <job_id>

Removes the job from the scheduled jobs by killing it

# OUTLINE

- A first step:
  - System overview
  - Login
  - Work environment
- Production environment
  - Our first job!!
  - Creating a job script
  - Accounting and queue system
  - PBS commands
- **Programming environment**
  - **Module system**
  - **Serial and parallel compilation**
  - **Interactive session**
- For further info...
  - Useful links and documentation

# AN EXAMPLE OF A PARALLEL JOB

```
#!/bin/bash
#PBS -l walltime=1:00:00
#PBS -l select=2:ncpus=20:mpiprocs=10
#PBS -o job.out
#PBS -e job.err
#PBS -A <my_account>

cd $PBS_O_WORKDIR     # points to the folder you are actually working into
module load autoload intelmpi
mpirun -np 20 ./myprogram
```

# MODULE SYSTEM

- All the optional software on the system is made available through the **"module" system**. It provides a way to rationalize software and its environment variables.
- Modules are divided in several *profiles*:

  •**profile/base** default - stable and tested compilers, libraries, tools

  •**profile/advanced** libraries and tools compiled with different setups that the default

  •**profile/chem** (phys, bioinf, astro,...) “domain” profiles with the application softwares specific for each field of research

  •**profile/archive** old or outdated versions of our module (we don't throw away anything!)
- Each profile is divided in 4 categories

**compilers** (GNU, intel, openmpi) **tools** (e.g. Scalasca, GNU make, VNC, ...)

**libraries** (e.g. LAPACK, BLAS, FFTW, …) **applications** (software for chemistry, physics, ... )

# MODULE SYSTEM

- CINECA's work environment is organized in modules, a set of installed libraries, tools and applications available for all users.
- "loading" a module means that a series of (useful) shell environment variables will be set
- E.g. after a module is loaded, an environment variable of the form "<MODULENAME>_HOME" is set

```
[amarani0@r000u06l01 ~]$ module load hdf5
[amarani0@r000u06l01 ~]$ ls $HDF5_LIB
libhdf5.a          libhdf5_fortran.so        libhdf5_hl.a          libhdf5hl_fortran.so        libhdf5_hl.la     libhdf5_hl.so.10.1.0  libhdf5.so
libhdf5_fortran.a  libhdf5_fortran.so.10     libhdf5hl_fortran.a   libhdf5hl_fortran.so.10     libhdf5_hl.so     libhdf5.la            libhdf5.so.10
libhdf5_fortran.la libhdf5_fortran.so.10.0.3 libhdf5hl_fortran.la libhdf5hl_fortran.so.10.0.3 libhdf5_hl.so.10 libhdf5.settings     libhdf5.so.10.2.0
[amarani0@r000u06l01 ~]$
```

- For certain modules, a specific profile must be loaded before ("module load profile/..."). Use the "modmap" command to understand which module is in which profile (try "modmap -h")

# MODULE COMMANDS

| COMMAND | DESCRIPTION |
|---|---|
| module av | list all the available modules |
| module load <module_name(s)> | load module <module_name> |
| module list | list currently loaded modules |
| module purge | unload all the loaded modules |
| module unload <module_name> | unload module <module_name> |
| module help <module_name> | print out the help (hints) |
| module show <module_name> | print the env. variables set when loading the module |

# MODULE PREREQS AND CONFLICTS

Some modules need to be loaded after other modules they depend from (e.g.: parallel compiler depends from basic compiler). You can load both compilers at the same time with “autoload”

```
[amarani0@r000u06l01 ~]$ module load hdf5
WARNING: hdf5/1.8.17--intelmpi--2017--binary cannot be loaded due to missing prereq.
HINT: the following modules must be loaded first: intelmpi/2017--binary
[amarani0@r000u06l01 ~]$ module load autoload hdf5
[amarani0@r000u06l01 ~]$
```

You may also get a “conflict error” if you load a module not suited for working together with other modules you already loaded (e.g. different compilers). Unload the previous module with “module unload”

# COMPILING ON PICO

- On MARCONI you can choose between three different compiler families: **gnu, intel** and **pgi**
- You can take a look at the versions available with "*module av" and then* load the module you want.

***module load intel*** *# loads default intel compilers suite*

***module load intel/pe-xe-2017--binary*** *# loads specific compilers suite*

| | GNU | INTEL | PGI |
|---|---|---|---|
| Fortran | gfortran | ifort | pgf77 |
| C | gcc | icc | pgcc |
| C++ | g++ | icpc | pgcc |

Get a list of the compilers flags with the command *man*

# PARALLEL COMPILING ON PICO

MPI libraries available: **OpenMPI/IntelMPI**

The library and special wrappers to compile and link the personal programs are contained in several modules, one for each supported suite of compilers

Load a version of **OpenMPI**:

```
module av openmpi
openmpi/1-10.3--gnu--6.1.0 (profile/base)
openmpi/1-10.3--intel--pe-xe-2017--binary
(profile/advanced)
module load autoload openmpi/1-10.3--gnu--6.1.0
```

Load a version of **IntelMPI**:

```
module av intelmpi
intelmpi/5.1--binary    (intel 2016)
intelmpi/2017--binary   (intel 2017)
module load autoload intelmpi/2017--binary
```

# PARALLEL COMPILING ON PICO

| | **OPENMPI/INTELMPI** |
|---|---|
| Fortran90 | mpif90/mpiifort |
| C | mpicc/mpiicc |
| C++ | mpiCC/mpiicpc |

Compiler flags are the same of the basic compiler (since they are basically MPI wrappers of those compilers)

OpenMP is provided with the following compiler flags:

gnu: -fopenmp

intel : -qopenmp

pgi: -mp

# JOB SCRIPT FOR PARALLEL EXECUTION

Let’s take a step back…

**#PBS -l select=2:ncpus=20:mpiprocs=5**

This example line means “allocate 2 nodes with 20 CPUs each, and 5 of them should be considered as MPI tasks”

So a total of 40 CPUs will be available. 10 of them will be MPI tasks, the others will be OpenMP threads (4 threads for each task).

In order to run a pure MPI job, ncpus must be equal to mpiprocs.

# EXECUTION LINE IN JOB SCRIPT

**mpirun -np 8 ./myprogram**

Your parallel executable is launched on the compute nodes via the command *“mpirun”.*
With the “-np” flag you can set the number of MPI tasks used for the execution. The default is the maximum number allowed by the resources requested.

**WARNING**:

In order to use mpirun, **openmpi-intelmpi has to be loaded** inside the job script:

**module load autoload intelmpi**

Be sure to load the same version of the compiler that you used to compile your code!!

# DEVELOPING IN COMPUTE NODES: INTERACTIVE SESSION

It may be easier to compile and develop directly in the compute nodes,

without recurring to a batch job.

For this purpose, you can launch an interactive job to enter inside a compute node by using PBS.

The node will be reserved to you as it was requested by a regular batch job

Basic interactive submission line:

```
qsub -I -l select=1 -A <account_name> -q <queue_name>
```

Other PBS keyword can be added to the line as well (walltime, resources,…)

Keep in mind that you are using computing nodes, and by consequence you are consuming

computing hours!

To exit from an interactive session, just type “exit”

# OUTLINE

- A first step:
  - System overview
  - Login
  - Work environment
- Production environment
  - Our first job!!
  - Creating a job script
  - Accounting and queue system
  - PBS commands
- Programming environment
  - Module system
  - Serial and parallel compilation
  - Interactive session
- For further info...
  - **Useful links and documentation**

# Useful links and documentation

- **Reference guide:**

  https://wiki.u-gov.it/confluence/display/SCAIUS/UG3.3%3A+PICO+UserGuide
  https://wiki.u-gov.it/confluence/display/SCAIUS/UG2.5.2%3A+Batch+Scheduler+PBS
  https://wiki.u-gov.it/confluence/display/SCAIUS/UG2.4%3A+Data+storage+and+FileSystem

- **Stay tuned with the HPC news:**
  **http://www.hpc.cineca.it/content/stay-tuned**

- **HPC CINECA User Support:** mail to **superc@cineca.it**

- **HPC Courses: corsi.hpc@cineca.it**