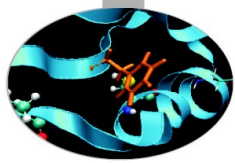
CINECA SCAI
SuperComputing Applications and Innovation



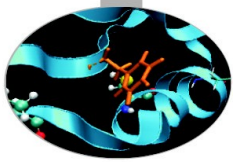
Production Environment

Introduction to Marconi HPC Cluster, for users and developers

HPC User Support @ CINECA

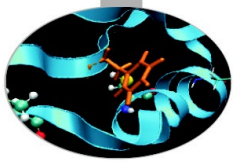
20/06/2016

CINECA



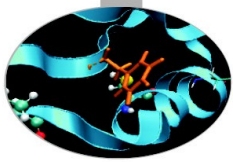
Overview

- Usage of a HPC system
  - Interactive mode
  - Batch mode
- Graphic sessions (using RCM)



Usage of a HPC system

- **Interactive mode**
  - general interactive operations
  - data movement
  - archiving
  - code development
  - compilations
  - basic debugger usage
  - …
- **Batch mode**
  - guarantees a fair access to resources
  - used for production runs



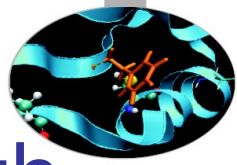
Interactive mode

- A **serial program** can be executed in the standard UNIX way:

  > ./program

This is allowed only for very short runs, since the **interactive environment has a 10 minutes time limit**: for longer runs please use the "batch" mode.

- A **parallel program** can be executed interactively only within an **"Interactive" PBS batch job**, using the "-I" (capital i) option:
  - the job is queued and scheduled as any PBS batch job
  - when executed, the standard input, output, and error streams are connected to the terminal session from which qsub was submitted.

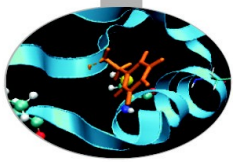


Example: an interactive session with MPI program “myprogram”

> qsub -A <account_no> -I -lselect=1:ncpus=2:mpiprocs=2 -- /bin/bash
qsub: waiting for job ... to start
qsub: job ... ready
> mpirun ./myprogram
> ^D (or “exit”)

If you want to **export variables** to the interactive session, use the **-v option**. For example, if "myprogram" is not compiled statically, you have to define and export the LD_LIBRARY_PATH variable:

> export LD_LIBRARY_PATH= ...
> qsub -I -v LD_LIBRARY_PATH ...

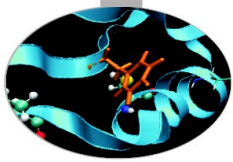


How to prepare a job for its execution on batch mode

On **MARCONI** we have the **PBS scheduler**

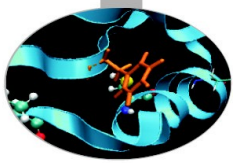
The job script scheme is:

```
#!/bin/bash
#PBS keywords
variables environment
execution line
```



PBS resources

- A job requests resources through the PBS keywords
- PBS matches the requested resources with available resources, according to rules defined by the administrator
- When resources are allocated to the job, the job can be executed



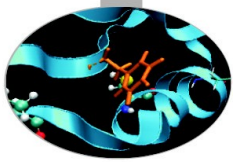
PBS resources

The syntax of the request depends on which type is concerned:
#PBS -l <resource>=<value> (server level resources, e.g. walltime)

#PBS -l select=N:chunk=…[+[N:]chunk=...] (chunk resources, e.g. ncpus, mpiprocs)

For example:
#PBS -l walltime=10:00
#PBS -l select=1:ncpus=1
Or
#PBS -l select=2:ncpus=8+4:ncpus=16

Moreover, resources can be required in one of two possible ways:
1) using PBS directives in the job script
2) using options of the qsub command



PBS directives: num. of processors

The number of cpus required for a serial or parallel MPI/OpenMP/mixed job must be required with the "select" directive:

#PBS -l select=NN:ncpus=CC:mpiprocs=TT

where:

NN: number of chunks (max depending on the queue)

ncpus=CC: number of physical cores per chunk

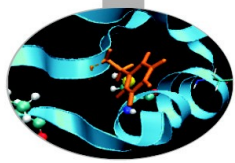
mpiprocs=TT: number of MPI tasks per chunk

for example:

#PBS -l select=1:ncpus=1 --> serial job

#PBS -l select=2:ncpus=8:mpiprocs=8 --> MPI job (2 chunks and 8 procs/tasks per chunk → 16 MPI tasks)

#PBS -l select=2:ncpus=8:mpiprocs=1 --> mixed job (2 MPI tasks and 8 threads/task)



PBS directives: processing time

Resources as the computing time, must be requested in this form:

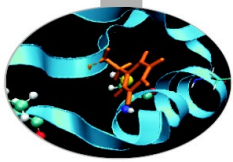
#PBS -l walltime=<value>

where

<value>: express the actual elapsed time (wall-clock) in the format hh:mm:ss

for example:

#PBS -l walltime=1:00:00 (one hour)



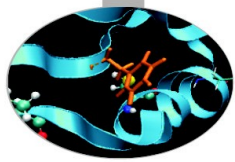
PBS directives: memory allocation

You can specify the requested memory up to to maximum memory available on the nodes using the "mem" directive:

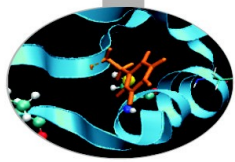
#PBS -l select=NN:ncpus=CC:mpiprocs=TT:mem=xxxGB

Please note: if you required more memory than the one “corresponding” to the number of assigned cpus (i.e., mem > ncpus* (total memory)/(number of cores in the node)), the number of "effective cores" and the cost of your job will increase

Other PBS directives



```
#PBS -l select=1:ncpus=18:mpiprocs=18:mem=120GB      # resources
#PBS -l walltime=1:00:00                             # hh:mm:ss

#PBS -A <my_account>                                 # name of the account

#PBS –N jobname                                      # name of the job

#PBS -o job.out                                      # standard (PBS) output file

#PBS -e job.err                                      # standard (PBS) error file

#PBS -j eo                                           # merge std-err and std-out

#PBS -m mail_events                                  # specify email notification
                    (a=aborted,b=begin,e=end,n=no_mail)

#PBS -M user_list                                    # set email destination
                          (email address)
```



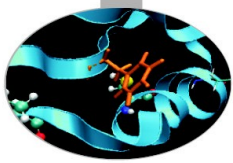
The User Environment

There are a number of environment variables provided to the PBS job. Some of them are taken from the user's environment and carried with the job. Others are created by PBS.

Short example lists some of the more useful variables:

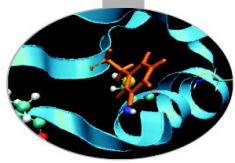
PBS_JOBNAME=jobb
PBS_ENVIRONMENT=PBS_BATCH
PBS_JOBID=453919.io01
PBS_QUEUE=shared

PBS_O_WORKDIR=/gpfs/scratch/usercin/aer0
PBS_O_HOME=/marconi/usercin/aer0
PBS_O_QUEUE=route
PBS_O_LOGNAME=aer0
PBS_O_SHELL=/bin/bash
PBS_O_HOST=nodexxx.marconi.cineca.it
PBS_O_MAIL=/var/spool/mail/aer0
PBS_O_PATH=/cineca/bin:/marconi/cineca/sysprod/pbs/default/bin: ...



PBS job script template

```
#!/bin/bash
#PBS -l walltime=2:00:00
#PBS -l select=1:ncpus=18:mpiprocs=18:mem=100GB
#PBS -o job.out
#PBS -e job.err
#PBS -A <account_no>
#PBS -m mail_events
#PBS -M user@email.com

cd $PBS_O_WORKDIR

module load autoload intelmpi/openmpi
module load somelibrary

mpirun -n 18 ./myprogram < myinput
```

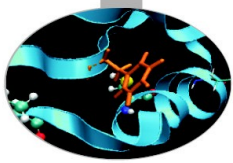


Job script examples

Serial job script:

```
#!/bin/bash
#PBS -o job.out
#PBS -j eo
#PBS -l walltime=0:10:00
#PBS -l select=1:ncpus=1
#PBS -A <my_account>
#
cd $CINECA_SCRATCH/test/
module load R
R < data > out.txt
```

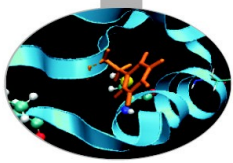
Job script examples



OpenMP job script

```
#!/bin/bash
#PBS -l walltime=1:00:00
#PBS -l select=1:ncpus=8
#PBS -o job.out
#PBS -e job.err
#PBS -A <my_account>

cd $PBS_O_WORKDIR       ! this is the dir where the job was submitted from

module load intel
./myprogram
```

N.B. Asking for ncpus=8 automatically sets OMP_NUM_THREADS=8.
Hence ./myprogram runs with 8 OMP threads

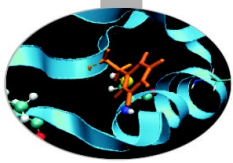


Job script examples

MPI Job Scripts

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

cd $PBS_O_WORKDIR        ! this is the dir where the job was submitted from

module load intel intelmpi
mpirun ./myprogram < myinput > myoutput
```

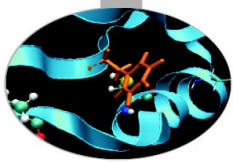
Job script examples



MPI+OpenMP job script

```
#!/bin/bash
#PBS -l walltime=1:00:00
#PBS -l select=2:ncpus=8:mpiprocs=2
#PBS -o job.out
#PBS -e job.err
#PBS -A <my_account>

cd $PBS_O_WORKDIR
export OMP_NUM_THREADS=4

module load intel intelmpi
mpirun ./myprogram
```

N.B. If you don't set the OMP_NUM_THREADS variable it will be equal to ncpus (8 in this example), hence you will have 2 MPI tasks and 8 threads/task → 16 threads on 8 cpus → core “overloading”

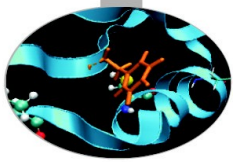


Array Jobs

- An efficient way to perform multiple similar runs, either serial or parallel, by submitting a unique job.

- The **maximum allowed** number of runs in an array job is set by the max_array_size parameter (typically a server attribute, the default is 10.000). If a limit is set on the maximum number of running jobs per user it will also apply to job arrays

```
#!/bin/bash
#PBS -N job_array
#PBS -l select=1:ncpus=1:mpiprocs=1
#PBS -l walltime=12:00:00
#PBS -A <account_name>
#PBS -J 1-20
#PBS -r y

cd $PBS_O_WORKDIR

./exe < input$PBS_ARRAY_INDEX.txt
```



PBS commands

qsub

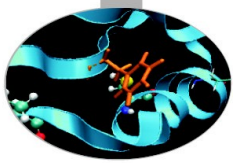
qsub <job_script>

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

qstat

qstat -u $USER

Shows the list of all your scheduled jobs, along with their status (idle, running, exiting, …) Also, shows the job id required for other qstat options



PBS commands

qstat

qstat -f <job_id>

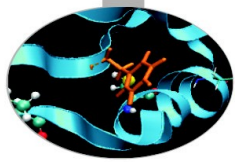
Provides a long list of informations for the <job_id> job.

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 be informed that the job will never start

qdel

qdel <job_id>

Removes the job from the scheduled jobs by killing it



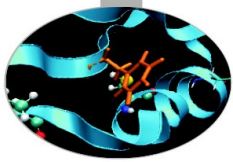
PBS commands

qalter

qalter -l <resources> <job_id>

Alter one or more attributes of one or more PBS queuing jobs

More information about these commands are available with the **man** command



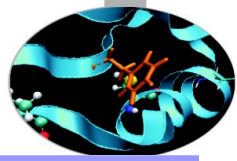
PSB Scheduler on MARCONI

- It is possible to submit jobs of different types, using only one "**routing**" **queue**: just declare how many resources (ncpus, walltime) you need and your job will be directed into the right queue with a right priority.

- The **maximum number of nodes** that you can request is around **167** (6000 cpus) with a maximum walltime of 24 hours

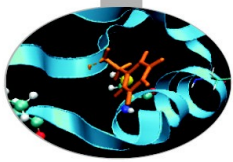
- If you do not specify the walltime, a default value of 30 minutes will be assumed; if you do not specify the number of cpus a default value of 1 will be taken

PSB Scheduler on MARCONI



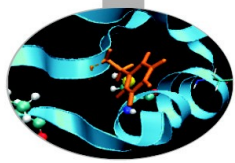
Queue Name		ncpus	Max Walltime	Max running jobs per user	Notes	Priority
route	**debug**	min = 1 max = 72	30 min	5	Managed by route	70
	prod	min = 1 max = 2304	24 h	20	Managed by route	50
	bigprod	min = 2305 max = 6000	24 h	[*]	Managed by route	60
special		min = 1 max = 36	180 h		Ask superc@cineca.it #PBS -q special	100
serial		1	4 h	[**]	on login nodes #PBS -q serial	30

[*] max_run_cpus = 6000 (per group) and 12.000 (total for the queue)
[**] max_run_ncpus = 12/16 (total for the queue)



Job's priority on MARCONI

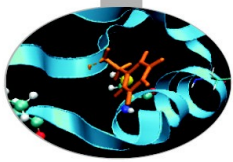
- The PBS scheduler estimates the job priority by taking into account the following parameters:
  - **prj_priority** = 0 [default_value]
  - **prj_quota_ratio** = left ratio of the monthly quota (the quota is calculated as total_budget / total_no_of_months), varies from 1 to 0
  - **resources** requested:
    - ncpus
    - walltime
  - **eligible time**



Usage of serial queue

Serial job script with specific queue request:

```
#!/bin/bash
#PBS -o job.out
#PBS -j eo
#PBS -l walltime=0:10:00
#PBS -l select=1:ncpus=1
#PBS -A <my_account>
#PBS -q serial
#
cd $CINECA_SCRATCH/test/
cp /gss/gss_work/DRES_my/* .
```



Graphic session

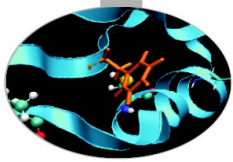
- It is recommended the usage of RCM, Remote Connection Manager, availave for:
  - –windows
  - –mac
  - –linux

- Client/server application
  every time you interact with the application, server side has to perform some operations that can take some time depending on bandwidth and latency of your Internet connection and workload of the clusters.

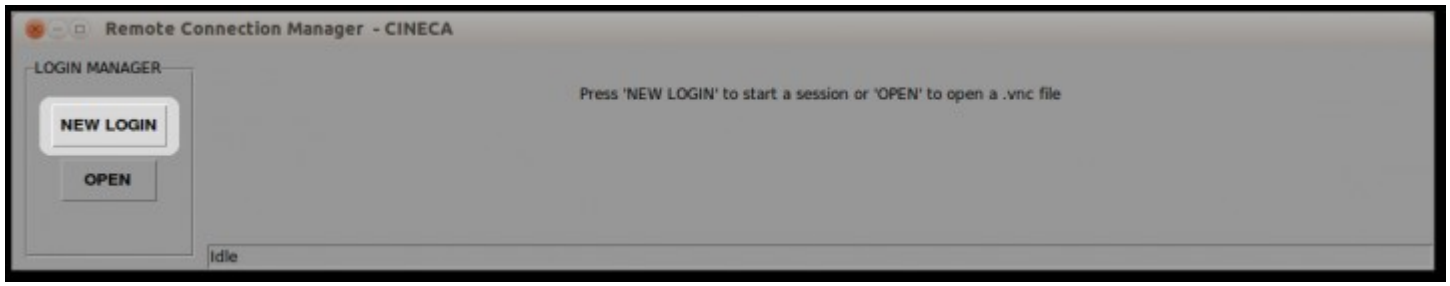
- Available at:

https://hpc-forge.cineca.it/svn/RemoteGraph/branch/multivnc/build/dist/Releases/?p=817

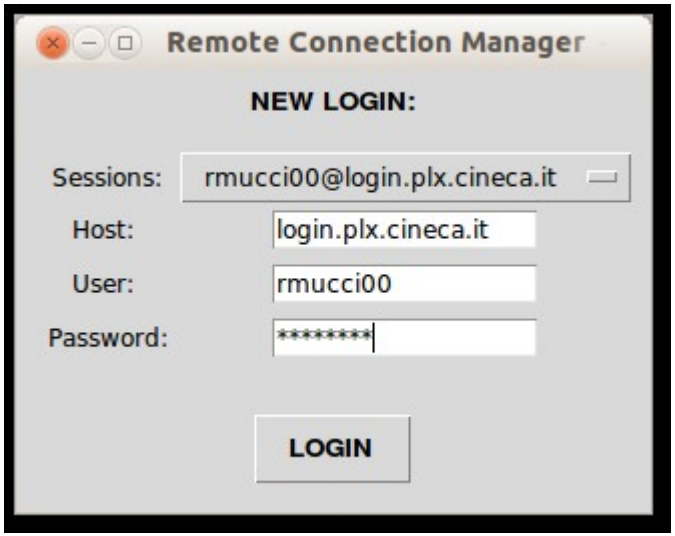
Working with displays



Start the application double clicking on the application icon and press the "NEW LOGIN" button:



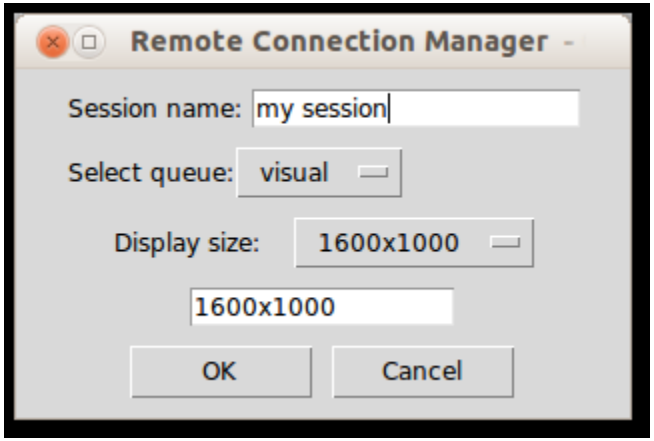
Insert the host name or select a previous session from the drop-down menu, your username and password, then press “Login”:



Working with displays

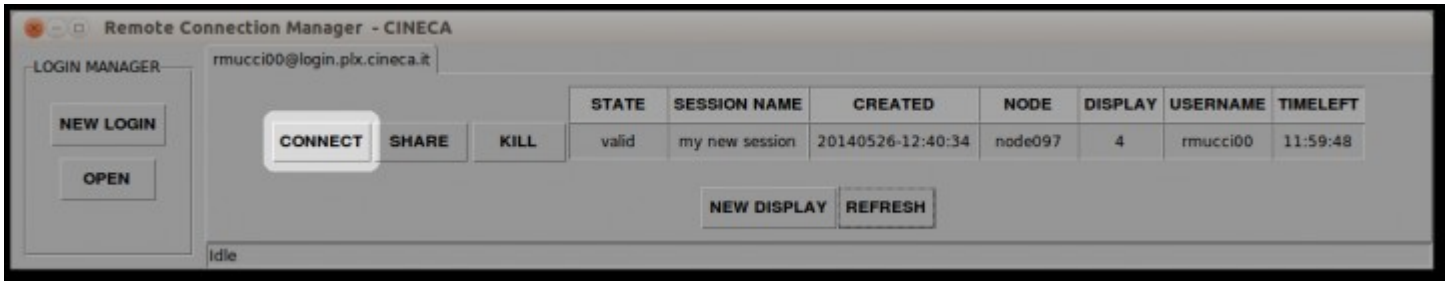
- Create a new display:

- A new window will appear to allow users to set:

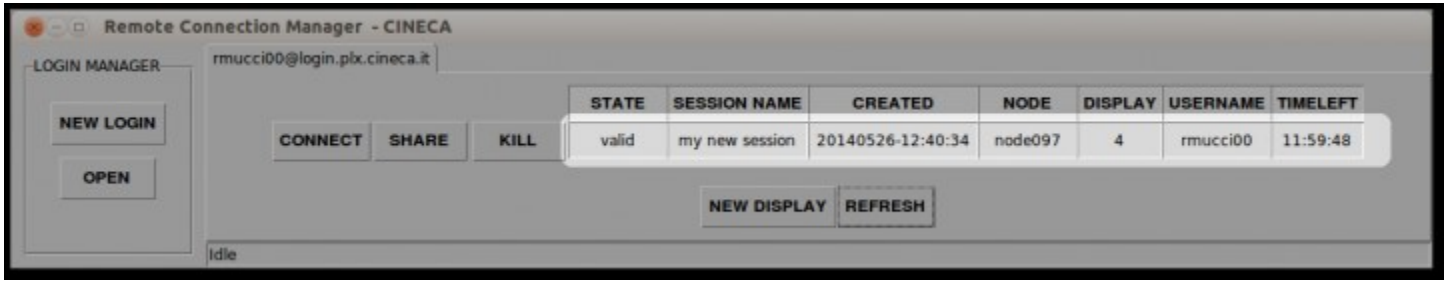


Working with displays

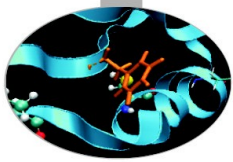
- Connect to a display:



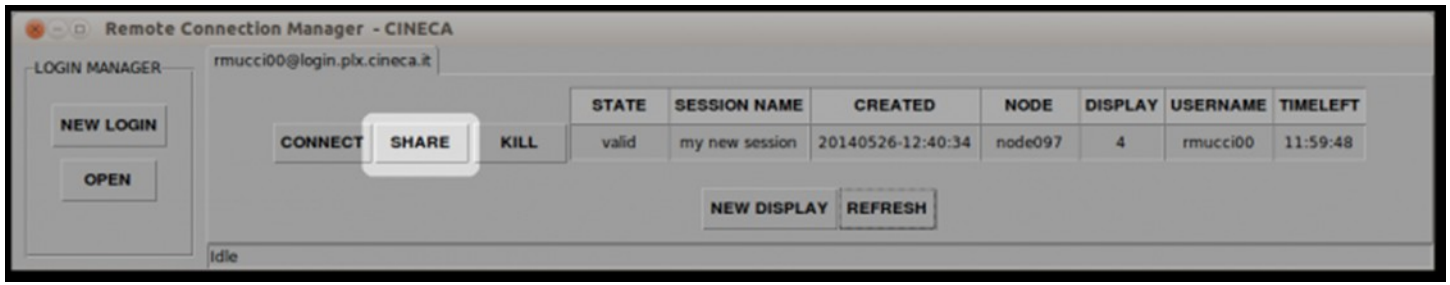
- Display information:



Working with displays

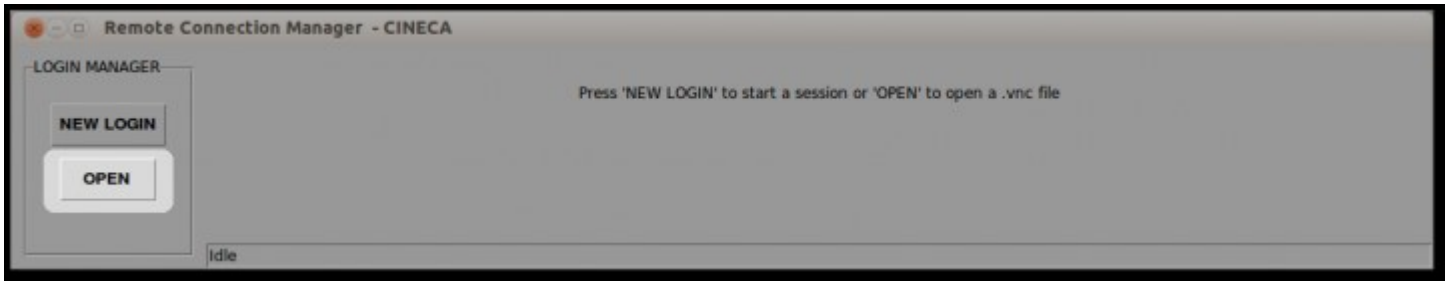


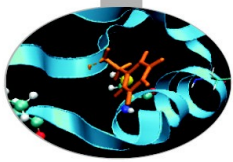
- Share a display:



- Send the saved file to the users who needs to access to the shared display.

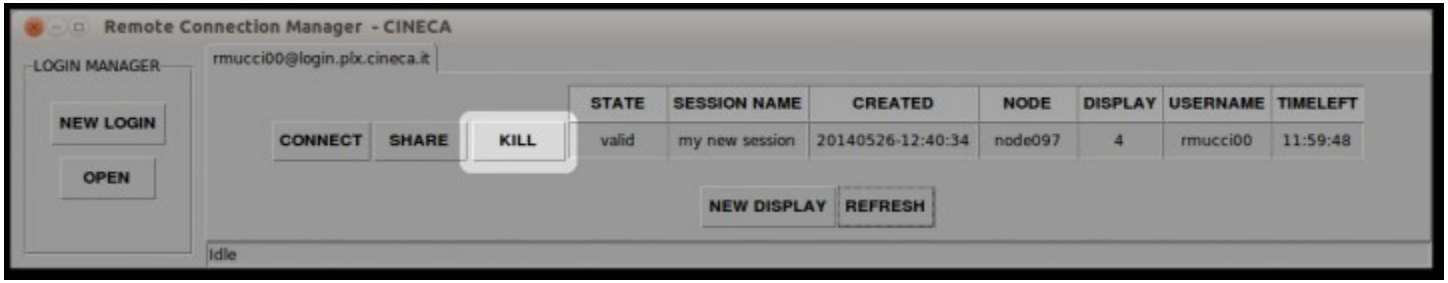
To connect to a shared display click on the "OPEN" button and select the received .vnc file:





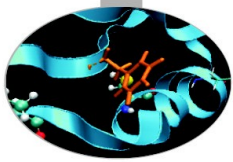
Working with displays

Kill a display:



Just press "**KILL**" beside the display you don't want to use anymore, and it will removed from the list of the available displays. This operation can take some time, depending on the workload of the clusters.

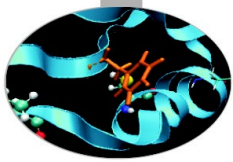
Note that by pressing "KILL", the related display will be not reachable anymore and you will lost not saved data.



Example of job execution using Totalview within RCM

- All the software that comes with a graphic user interface (GUI) can be used within a RCM session

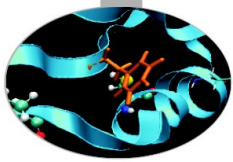
- With respect to other GUIs that can be run on RCM, Totalview is a little peculiar and must be run directly on the nodes that execute the parallel code



Example of job execution using Totalview within RCM

- establish connection through RCM with MARCONI
- open a terminal and prepare the job script

```
#!/bin/bash
#PBS -l walltime=00:30:00
#PBS -l select=1:ncpus=4:mpiprocs=4:mem=15gb
#PBS -N totalview
#PBS -o job.out
#PBS -e job.err
#PBS -A your_account_here        # account number (type saldo –b)

module load autoload <openmpi|intelmpi>      #select the compiler used to compile your program
module load totalview
cd $PBS_O_WORKDIR

totalview mpirun -a poisson.exe -n 4
```



Example of job execution using Totalview within RCM

Submit the job and pass the variable DISPLAY to the execution nodes.

qsub -v DISPLAY=`hostname`$DISPLAY job.sh

PS: In a terminal opened inside RCM, the shortcut to paste text copied elsewhere is "Ctrl+Shift+Insert"