HANDBOOK FOR THE USE OF THE EXCITON CLUSTER

© Aritz Leonardo & Carsten Ullrich.

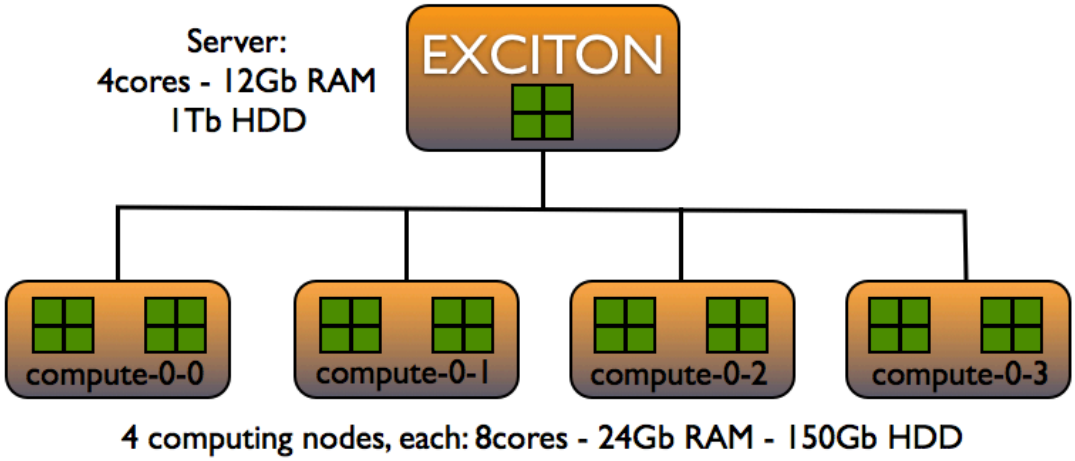
Last modification: October 2009

CHAPTER I:
CLUSTER DESCRIPTION

1.1 HARDWARE

The EXCITON cluster is composed of:

- ☑ 36 Cores Intel® Xeon® E5430, 2x6MB Cache, 2.66GHz, 1333MHz FSB.
- ☑ 108 Gb of RAM memory.
- ☑ Network Adapters: Gigabit Ethernet NIC (NetXtreme II 5708).
- ☑ 1Tb of storage mounted as the home directory and 4x150GB of local scratch.



EXCITON cluster has a maximum of 32 computing cores for running user's jobs. In addition, the cluster has 1.0 Tb of disk space mounted in the "/home/<user>" directory, that is common for all the cluster nodes (NFS). On the other hand, each node has a small hard drive of 150 Gb mounted in "/state/partition1", which is local, and thus only visible when you are inside the specific node. All these nodes are re-arranged into 2 queues:

- ☑ **p-small**: jobs with execution time below 24 hours
- ☑ **p-large:** jobs with execution time between 24 and 1500 hours

The execution priorities depend on the resources requested by the user's program, this is, the CPU time of the program and its RAM size. For example, a fast program that uses small amounts of RAM will be executed at the p-small queue with a higher priority.

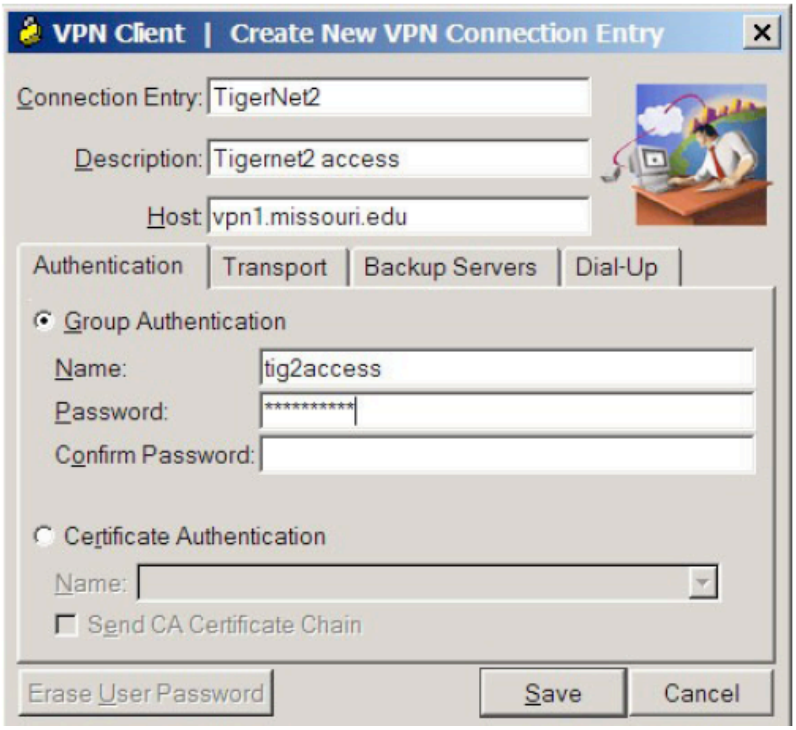
CHAPTER II:
SUBMITTING JOBS TO THE QUEUE
2.0 LOGIN TO EXCITON

☑ If you plan to login to EXCITON from any computer located inside MU's network (including laptops connected to the university's WIFI signal), simply type in your terminal:

ssh -Y username@exciton.physics.missouri.edu (-Y imports graphics)

☑ You may also connect to EXCITON from any computer outside MU provided you use a VPN client. Download CISCO's client and follow the next steps:

1. Create a **New Connection** from the Cisco VPN client.
2. At **Connection Entry**, enter a name for the profile.
3. At **Description**, type a brief description of the connection.
4. At **Host**, type in the host address of the VPN concentrator: vpn1.missouri.edu.
5. Select the **Group Authentication** radio button.
6. Enter the **Name** as "tig2access".
7. Enter the **Password** for tig2access: "sorrento5%".
8. **Confirm** the tig2access password.
9. **Save** the profile.



2.1 SHELL SCRIPT TO SUBMIT A SERIAL JOB TO THE QUEUE SYSTEM.

Regardless of the nature your job, wether is a parallel or a serial execution, the queue system of the cluster must be used always! NO user should execute directly in the server or in any of the computing nodes whatever program he is willing to run. In order to send a particular job to the queue system (TORQUE/openPBS) you must submit a little script like the one posted below with some specifications for the queue system to know.

```
vi job.pbs

#!/bin/tcsh -f
#PBS -q parallel                    (DO NOT change this line even for serial executions)*
#PBS -l mem=8gb                     (RAM memory required)*
#PBS -l nodes=1:ppn=1               (Number of NODES and PROCS per NODE required)*
#PBS -l walltime=25:00:00           (Estimated amount of time for the job execution, hh:mm:ss)*
#PBS -N Au_0001                     (Name of the job)
#PBS -m ea                          (email me when the job exists or aborts)
#PBS -M leonardoal@missouri.edu
#PBS -j oe                          (Merge the error and output files)
#PBS -r n                           (Do not rerun the job)

cd $PBS_O_WORKDIR       (MOVE TO the working directory)

echo ----------------------------------------------------
echo PBS: qsub is running on $PBS_O_HOST
echo PBS: originating queue is $PBS_O_QUEUE
echo PBS: executing queue is $PBS_QUEUE
echo PBS: working directory is $PBS_O_WORKDIR
echo PBS: execution mode is $PBS_ENVIRONMENT
echo PBS: job identifier is $PBS_JOBID
echo PBS: job name is $PBS_JOBNAME
echo PBS: node file is $PBS_NODEFILE
echo PBS: number of nodes is $NNODES
echo PBS: current home directory is $PBS_O_HOME
echo PBS: PATH = $PBS_O_PATH
echo ----------------------------------------------------

./program.x < input >& output
exit
```

```
-------------------------------------------
| All these echos print environmental
| variables of the queue system.
| It is only for the user's information
| and thus can be suppressed.
-------------------------------------------
```

*= NECESSARY LINES TO BE SPECIFIED

qsub job.pbs launches the script to the queue system.

As you can access your working directory at any time, you may have a look at the evolution of your job running at the queue system whenever you want. Note also, that if your program needs any other input file i.e. pseudopotentials, density matrixes etc. they should reside in the working directory or the path to them should be clear for the executable file. Eventually, the use of the home directories as working directories can become very slow if the amount of running jobs is high. Programs with a big amount of input/output data will make a big use of the cluster's net and hence of the mounted hardrive, becoming this way a possible bottleneck in speed performance. Read the FAQs to learn how to calculate using the local scratch directories.

2.2 SIMPLE SCRIPT FOR PARALLEL EXECUTION (mpich2)

Parallel scripts are pretty much identical to the serial one posted above. PBS lines do not change except for the fact of the number of nodes and procs. requested. The main difference resides then in the execution line. The parallel program one pretends to run should have been compiled previously using MPICH2 compilers (mpif90, mpicc, mpicxx...). MPICH2, unlike MPICH, uses an external process manager for scalable startup of large MPI jobs. The default process manager is called MPD, which is a ring of daemons on the machines where you will run your MPI programs and that has to be initiated at the time of execution.

```
#!/bin/tcsh -f
#PBS -q parallel
#PBS -l mem=8gb
#PBS -l nodes=3:ppn=2
#PBS -l walltime=100:00:00
#PBS -N mol3x3_A
#PBS -j oe
#PBS -r n
#PBS -m ea
#PBS -M leonardoal@missouri.edu

cd $PBS_O_WORKDIR

set NCPU `wc -l < $PBS_NODEFILE`
set NNODES `uniq $PBS_NODEFILE | wc -l`
```

```
echo ----------------------------------------------------
echo ' This job is allocated on '${NCPU}' cpu(s)'
echo 'Job is running on node(s): '
cat $PBS_NODEFILE
echo ----------------------------------------------------

echo PBS: qsub is running on $PBS_O_HOST
echo PBS: originating queue is $PBS_O_QUEUE
echo PBS: executing queue is $PBS_QUEUE
echo PBS: working directory is $PBS_O_WORKDIR
echo PBS: execution mode is $PBS_ENVIRONMENT
echo PBS: job identifier is $PBS_JOBID
echo PBS: job name is $PBS_JOBNAME
echo PBS: node file is $PBS_NODEFILE
echo PBS: number of nodes is $NNODES
echo PBS: current home directory is $PBS_O_HOME
echo PBS: PATH = $PBS_O_PATH
echo ----------------------------------------------------
echo ' '
echo ' starting up mpd daemons '

# Prevents later on from killing other jobs of the same user when mpdallexit is called
setenv MPD_CON_EXT $PBS_JOBID
# Starts the MPD daemon in all the nodes that will be used
mpdboot -n $NNODES -f $PBS_NODEFILE -v --remcons
# Testing communications between the selected nodes
sleep 10
mpdtrace -l
mpdringtest 100
# The parallel program is launched using mpiexec
mpiexec -n $NCPU ./program.x < input >& output
# kill the daemon in the nodes but preventing the killing of other daemon rings from other jobs
# of the same user thanks to MPD_CON_EXT
mpdallexit
exit
```

2.2 SIMPLE PARALLEL SCRIPT (OpenMPI)

Coming soon!

2.3 QSUB / QSTAT and MONITORING

Once the shell script that contains the PBS requirements together with execution orders and possible input file declarations is written, the way it is launched to the queue system, is simply typing:

```
qsub job.pbs
```

For a complete description of the submitting possibilities have a look at the online PBS documentation:

http://www.clusterresources.com/torquedocs/commands/qsub.shtml

The status of the jobs submitted by any user to the queue system can be checked using the command qstat with its numerous options(-a -f -n -u ....). For example:

```
qstat -a
```

prompts:

```
exciton.physics.missouri.edu:

                                                                  Req'd    Req'd     Elap
Job ID          Username Queue    Jobname         SessID  NDS   Memory   Time   S  Time
--------------- -------- -------- --------------- ------ ----- -- ------ ------- --- --------
85.exciton.physi leonardo p-small aupa_athletic6  19443     4  --    8gb   24:00  R  00:00
86.exciton.physi leonardo p-large  aupa_athletic6  19653     3  --    8gb    1500:  R  00:00
87.exciton.physi leonardo p-large  mol3x3_A        19856     3  --    8gb    1500:  R  00:00
88.exciton.physi leonardo p-large  mol3x3_B        20045     4  --    8gb    1500:  C  00:00
89.exciton.physi leonardo p-large  mol3x3_B         --       4  --    8gb    1500:  Q   --
```

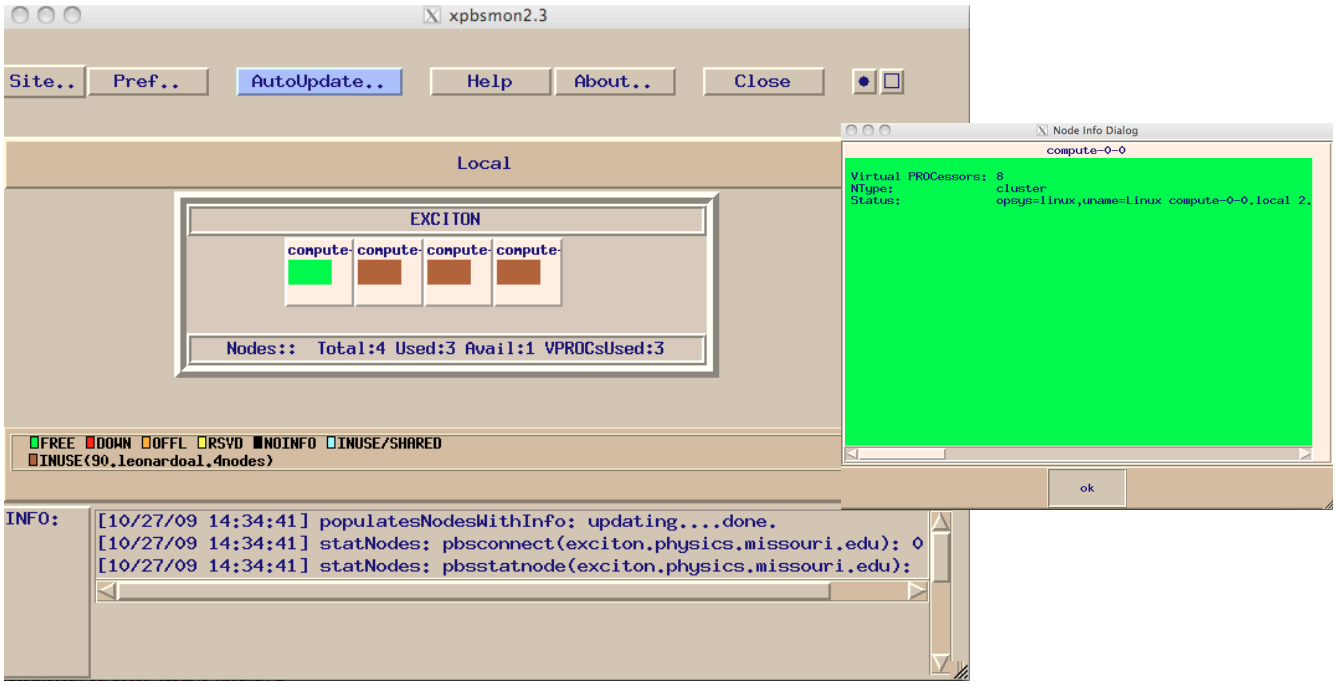
Any submitted job which is either RUNNING or QUEUING can be aborted using the command:

```
qdel Job_ID          ( “qdel 85 89” will kill the first and the last jobs from the list above )
```

There is a graphical tool that might result helpful when you want to check the workload of the cluster and particularly when one wants to visualize the status of the nodes. If you logged into EXCITON using “ssh -Y” you should be able to import graphics, so once you are in, type:

```
xpbsmon &
```

and it will show:



CHAPTER III:

SOFTWARE

All the software installed in EXCITON for public use is placed in the directory /opt. Among other programs one can find:

- ☑ Quantum-espresso
- ☑ Abinit
- ☑ Octopus
- ☑ Intel Compilers 11.0 + MKL libraries
- ☑ FFTs
- ☑ Mpich2
- ☑ OpenMPI
- ☑ GNU compilers: gfortran, gcc, g++
- ☑ GSL

Frequently Asked Questions

A) How can I speed up calculations?

a1. As a general rule and when possible, try not to use GNU compilers (gcc or gfortran). The INTEL compilers available in "/opt" and the MKL libraries (that contain Blas, Lapack, etc) are optimized for XEON processors. You will significantly improve the performance of your program. One other trick is to try optimization flags like "–O3" at the time of compilation. Be aware that sometimes your program may not support such an aggressive optimization aborting the compilation or giving weird errors at execution time. Try to lower down flags to "-O2", "-O1" or "–O0" (disable optimizations).

a2. If your program needs a large amount of temporary disk space, or produces data at a high rate accessing the disk continuously, the global "/home" directory could become a bottleneck decreasing dramatically the performance. One has to keep in mind that the home directory is a hard drive shared by all the users of the cluster and physically located in the entrance server. This implies that every time that a program running in a certain computing node wants to read or write something, it does it using the net! Moreover, if user "A" is writing, "B" has to wait for his turn, loosing precious computing time.

One way to overcome this problem is using the local disks that every node has instead of the home directory. The fee to pay is that one is forced to calculate using a SINGLE node and thus 8 processors at most. Always keep in mind that the local disks, as the name indicates, are local and thus not visible from one node to the other. The following script may result helpful:

```
#!/bin/tcsh -f
#PBS -q parallel
#PBS -l mem=8gb
#PBS -l nodes=1:ppn=8                    ( NOTE: nodes=1 !! we are running locally)
#PBS -l walltime=100:00:00
#PBS -N mol_3x4
#PBS -j oe
#PBS -r n
#PBS -m ea
#PBS -M user@missouri.edu
```

```
# Create a scratch directory which has the name of the JOBID. This number is different for every
# job launched so prevents from overwriting between 2 simultaneous running jobs
mkdir /state/partition1/<username>/PBS_$PBS_JOBID

# Copy all the files needed from the directory from which the script was launched to the
# previously created local scratch directory.
cp $PBS_O_WORKDIR/*.x          /state/partition1/<username>/PBS_$PBS_JOBID
cp $PBS_O_WORKDIR/input*       /state/partition1/<username>/PBS_$PBS_JOBID

set NCPU=`wc -l < $PBS_NODEFILE`
set NNODES=`uniq $PBS_NODEFILE | wc -l`

echo ' '
echo ' starting up mpd daemons '
setenv MPD_CON_EXT $PBS_JOBID
mpdboot -n $NNODES -f $PBS_NODEFILE -v --remcons
sleep 10

mpdtrace -l
mpdringtest 100

# Move to the local local scratch
cd /state/partition1/<username>/PBS_$PBS_JOBID

# start calculating locally
mpiexec -n $NCPU ./pw.x < si.inp2 >& si.out2
mpdallexit

# COPY back the results to the working directory inside home
cp -r /state/partition1/<username>/PBS_$PBS_JOBID  $PBS_O_WORKDIR
# DO NOT FORGET TO ERASE the local scratch!!!! It is only 150gb big so it is a MUST
rm -rf /state/partition1/<username>/PBS_$PBS_JOBID
exit
```

You will be able to check the evolution of your job by login into the specific node where your program is running (for example "ssh compute-0-0"). Once inside you should jump to the "/state/partition1/<username>/PBS_$PBS_JOBID" directory and check there. The way to know the specific node where your program is running is typing "qstat –n".

a3. Very often incorrectly finished parallel jobs can create processes at the EXCITON nodes that are not removed by the queue system and remain consuming resources. If a computing node has one of this "ghost" process running, it will be invisible for torque and hence the CPU time of the node will be shared with the real job that needs to be calculated. It is very recommendable to run a cleaning script once in a while to avoid this. Killing all these processes is simple. You have to wait until all your jobs at EXCITON are finished and then execute:

vi clean.sh

```
#!/bin/bash -f
cat > nodes.list << EOF
compute-0-0
compute-0-1
compute-0-2
compute-0-3
EOF

mpdcleanup --file=nodes.list --user=<username>
rm nodes.list

for i in `seq 0 3`;
   do
   ssh compute-0-$i pkill -u <username>
   echo NODE: compute-0-$i is clean!
   done
```

NOTE: This script will only work if you can access the nodes using **SSH without having to enter your password**. You have to generate a ssh-key once and you are set to go:

```
cd /home/<username>
rm -rf .ssh
ssh-keygen -t dsa
cd .ssh
cp id_dsa.pub authorized_keys2
```