Cluster Clonetroop: HowTo 2014

This section contains information about how to access, compile and execute jobs on Clonetroop, Laboratori de Càlcul Numeric's Cluster.

Description

Laboratori de Càlcul Numeric's cluster is a HPC Beowulf cluster that consists of 48 compute nodes (18 acquired in 2009 and 30 acquired in 2010) and one master node. All nodes are connected using an Infiniband network.

Compute nodes:

```
  1 x Dell Power Edge R820 4 x Octa-Core Xeon E5-4640 (2.4 GHz, 10MB L3
Cache, 1333MHz FSB) with 768 GB RAM - QDR Infiniband Network
  16 x Dell Power Edge M600 2 x Quad-Core Xeon E5440 (2.8 GHz/2x6MB, 1333Mhz
FSB) with 16 GB RAM - DDR Infiniband Network
  16 x Dell Power Edge C6100 2 x Hexa-Core Xeon L5640 (2.26 Ghz/12MB cache,
1333Mhz FSB) with 36 GB RAM - QDR Infiniband Network
  10 x Dell Power Edge C6100 2 x Quad-Core Xeon E5640 (2.66 Ghz/12MB cache,
1066Mhz FSB) with 36 GB RAM - QDR Infiniband Network
  4 x Dell Power Edge C6100 2 x Quad-Core Xeon E5640 (2.66 Ghz/12MB cache,
1066Mhz FSB ) with 72 GB RAM - QDR Infiniband Network
```

Access

The only method to connect to clonetroop is using SSH protocol (shell or SFTP). Username and password should be provided by LaCaN's systems admistrator.

This is the command to use. Flag '-X' is to tunnel X11 connections.

ssh -X clonetroop.upc.es

Connection is available from:

```
  All computers connected to LaCaN's network
  lordvader.upc.es (worlwide access)
```

We can download a SSH Client from http://www.putty.org/

Disk Quota

Disk use on clonetroop is controlled using a quota system. To check your quota status you can use the command “quota”

[username@clonetroop ~]$ quota Disk quotas for user username (uid 9999): Filesystem blocks quota limit grace files quota limit grace /dev/mapper/VolGroup00-LogVol03 6529396 10000000 10000000 70275 0 0 lordvader.upc.es:/lordvader 4720592 5000000 5100000 72663 0 0

User's Home directory is /clonetroop/username

For more disk space, we have a **scratch disk without quota limit**.

Available queues

This is the list of our available queues' properties.

serial4G

```
Max jobs: 32
Max memory/job: 4 GB
Max Cores/job: 1
Max execution time: none
CPU: Quad-Core Xeon E5440 (2.8 GHz/2x6MB, 1333Mhz FSB)
```

serial9G

```
Max jobs: 24
Max memory/job: 9 GB
Max Cores/job: 1
Max execution time: 8 days
CPU: Quad-Core Xeon E5640 (2.66 Ghz/12MB cache, 1066Mhz FSB)
```

serial16G

```
Max jobs: 16
Max memory/job: 16 GB
Max Cores/job: 1
Max execution time: 8 days
```

```
CPU: Quad-Core Xeon E5-4640 (2.4 GHz, 20MB L3 Cache, 1333Mhz FSB)
```

serial24G

```
Max jobs: 8
Max memory/job: 24 GB
Max Cores/job: 1
Max execution time: 8 days
CPU: Quad-Core Xeon E5-4640 (2.4 GHz, 20MB L3 Cache, 1333Mhz FSB)
```

parallel3G_A

```
Max slots:192
Max memory/job: none
Max nodes: 16
Max CPU/node: 12
Max CPU/job: 192
Max execution time: 4 days
CPU: Hexa-Core Xeon L5640 (2.26 Ghz/12MB cache, 1333Mhz FSB )
```

parallel3G_B

```
Max slots: 80
Max memory/job: none
Max nodes: 10
Max CPU/node: 8
Max CPU/job: 80
Max execution time: 4 days
CPU: Quad-Core Xeon E5640 (2.66 Ghz/12MB cache, 1066Mhz FSB)
```

parallel2G

```
Max slots: 64
Max memory/job: none
Max nodes: 8
Max CPU/node: 8
Max CPU/job: 64
Max execution time: none
CPU: Quad-Core Xeon E5440 (2.8 GHz/2x6MB, 1333Mhz FSB)
```

parallel_master

```
Max slots: 48
Max memory/job: none
Max nodes: 6
Max CPU/node: 8
Max CPU/job: 48
Max execution time: 4 days
CPU: Quad-Core Xeon E5440 (2.8 GHz/2x6MB, 1333Mhz FSB)
```

Submitting jobs

We use Sun Grid Engine (SGE) queue system to control and distribute jobs to the compute nodes.

Preparing your job

We have to follow these steps before sending a job to the queue.

1. Compile your program (if applicable)
2. Prepare our job script
3. Choose an appropiate execution queue

Useful SGE commands

Submit a job

```
qsub -q queue_name my_job_script.sh
```

View running jobs

```
qstatus -> show all running jobs (you can see your job id)
qstat -> show only your jobs
```

Delete a job

```
qdel jobid
```

Force deleting a job

```
qdel -f jobid
```

Jobs Scripts

Standard job

This section refers to serial executables compiled by yourself. Quick step by step guide

1. Compile your executable
2. Prepare your job script adapting the example below
3. Submit your job to a Serial execution queue

Standard job script example:

```
#!/bin/bash
#
#$ -cwd
#$ -j y
#$ -S /bin/bash
#$ -V
# -cwd means to execute the job for the current working directory.
# -j y means to merge the standard error stream into the standard output
stream instead of having two separate files
# -S /bin/bash specifies the interpreting shell for this job to be the Bash
shell.
# -V imports current environment variables (path, libraries, etc.) to the
new shell
echo "Current working directory is now: " `pwd`
echo "Starting job at `date`"
#Executable
./my_standard_job
echo "JOB run completed at `date`"
```

Note that lines starting with ”#$” are SGE commands and lines starting with ”# ” are comments.

Matlab Jobs

Matlab licences are limited. So it's possible that you experience problems submitting your job.

Please contact cluster administrator **if you plan to run more than 2 matlab jobs at the same time.** We will make a especial queue to save matlab licenses.

Please, use the command lmstat to check the number of licenses available on the server.

Quick step by step guide

1. Write your Matlab code
2. Prepare your job script adapting the example below
3. Submit your job to a Serial execution queue or to your special queue

Matlab installed version is R2013b (8.2.0.701) 64-bit (glnxa64)

Matlab job script

```
#!/bin/bash
#$ -cwd
#$ -j y
#$ -S /bin/bash
#$ -V
# SGE script for MATLAB batch job
# Check on some basics:
echo "Current working directory is now: " `pwd`
echo "Starting MATLAB at `date`"
matlab -nojvm -nodisplay -nosplash < input.m > output.out
echo "MATLAB run completed at `date`"
```

Castem Jobs

Quick step by step guide

1. Write your Castem code
2. Prepare your job script adapting the example below
3. Submit your job to a Serial execution queue (Be careful configuring the amount of memory that your program will need (L, XL, XXL, etc.) according with the choosen queue)

Castem installed version is Cast3M 2013

```
#!/bin/bash
#$ -cwd
#$ -j y
#$ -S /bin/bash
#$ -V
# SGE script for CASTEM batch job
export DISPLAY
echo "Current working directory is now: " `pwd`
echo "Starting CASTEM at `date`"
castem -m XL DPT2Dfina.dgibi
```

```
echo "CASTEM run completed at `date`"
rm fort.98
```

Compiling

GNU and Intel compilers are available in this cluster. Please use modules to manage your user environment. Follow this link for detailed information: Environment Modules info

GNU compilers version 4.3.4, 4.4.6 (default) and 4.8.2

```
  C: gcc
  c++: g++
  fortran: f95
```

More information about these compilers:

- http://gcc.gnu.org/onlinedocs/gcc-4.8.2/gcc/
- http://gcc.gnu.org/onlinedocs/gcc-4.8.2/gfortran/

Intel compilers version 13.0.1 and 14.0.0

```
  C and C++: icc
  fortran: ifort
```

More information about these compilers:

- http://clonetroop.upc.es/intel_compiler_c/index.htm
- http://clonetroop.upc.es/intel_compiler_f/index.htm
- http://clonetroop.upc.es/intel_mkl/mkl_userguide/index.htm

Scientific Libraries

MKL, and other scientific libraries are installed and ready to be used.

MPI Parallel Environment (distributed memory)

Quick step by step guide

If you are going to compile and execute MPI jobs, you should follow these steps (order is important):

1. Define your MPI environment
2. Compile your program
3. Write your job script (script to sending job to the queue)
4. Submit your job to a execution queue

NOTE: Is very mportant to compile and run programs using the same mpi implementation.

Set up your environment

We use modules to easily customize your shell environment (PATH, MANPATH, INCLUDE, LD_LIBRARY_PATH, etc). Modules allows you to cleanly set and unset your paths and environment variables.This is the list of MPI available configurations:

- rocks-openmpi (Rocks-compiled OpenMPI 1.6.2 with tcp,self transport)
- rocks_openmpi_ib (Rocks-compiled OpenMPI 1.6.2 with openib,dapl transports)
- intel-mpi/4.1.0 (Intel MPI 4.1.0 included in Cluster Studio XE 2013)
- intel-mpi/4.1.1 (Intel MPI 4.1.1 included in Cluster Studio XE 2013 SP1)

There are more implementations of MPI. Contact the administrator if you need a different one.

To view this list you can use the following command:

```
module available
```

Use the command module to change your environment. Follow this link for detailed information: Environment Modules info

Compiling

To compile MPI programs, we use the following wrappers. On each line, first command for openMPI, second command for intel:

- mpic++, mpiicpc
- mpicc, mpiicc
- mpiCC, mpiicc
- mpif77, mpiifort
- mpif90, mpiifort

Changing your MPI environment you can define which implementation of MPI is used to compile your programs (openmpi, intel, etc.)

Submit MPI jobs to the queue system SGE

This script is used to submit a MPI job with intel-mpi/4.1.1 configuration.

```
#!/bin/bash
#$ -cwd
#$ -j y
#$ -S /bin/bash
#$ -V
# Check on some basics:
echo "Running on host: " `hostname`echo "
Current working directory is now: " `pwd`echo "
Starting job at `date`"


#DEFINE YOUR PARALLEL ENVIRONMENT
# -pe <parallel-environment> min_procs-max_procs
# Use of <parallel_environment>:
# impi for Intel MPI
# orte for GNU OpenMPI
# min_procs means the minimum of processors that your program requires to
start
# max_procs means the maximum of processors that your program will use
# it is possible to put only one number, that means that this number is the
minimum and maximum
#$ -pe impi 24-48

#Some Parallel DEBUG information (removable)
echo "Current working directory is now: " `pwd`
echo "Starting job at " `date`
echo "Number of SLOTS: $NSLOTS"
echo "Running on host: " `hostname`
echo "PE_HOSTFILE: $PE_HOSTFILE"
echo "-------- HOST FILE -----"
cat $PE_HOSTFILE
echo "------------------------"

#Load MPI environment
module load intel-mpi/4.1.1

#EXECUTABLE
mpirun -np $NSLOTS -r ssh ./my_mpi_executable
echo "JOB run completed at `date`"
```

This script is used to submit a MPI job with rocks-openmpi configuration.

```
#!/bin/bash
#$ -cwd
#$ -j y
#$ -S /bin/bash
#$ -V
# Check on some basics:
echo "Running on host: " `hostname`echo "
```

```
Current working directory is now: " `pwd`echo "
Starting job at `date`"


#DEFINE YOUR PARALLEL ENVIRONMENT
# -pe <parallel-environment> min_procs-max_procs
# Use of <parallel_environment>:
# impi for Intel MPI
# orte for GNU OpenMPI
# min_procs means the minimum of processors that your program requires to
start
# max_procs means the maximum of processors that your program will use
# it is possible to put only one number, that means that this number is the
minimum and maximum
#$ -pe orte 24-48

#Some Parallel DEBUG information (removable)
echo "Current working directory is now: " `pwd`
echo "Starting job at " `date`
echo "Number of SLOTS: $NSLOTS"
echo "Running on host: " `hostname`
echo "PE_HOSTFILE: $PE_HOSTFILE"
echo "-------- HOST FILE -----"
cat $PE_HOSTFILE
echo "------------------------"

#Load MPI environment
module load rocks-openmpi

#EXECUTABLE
mpirun -v ./my_mpi_executable

echo "JOB run completed at `date`"
```

OpenMP Parallel Environment (shared memory)

Quick step by step guide

1. Compile your code
2. Prepare your job script adapting the example below
3. Submit your job to a Parallel execution queue

NOTE: It is important to decide how many threads is your job going to execute and setup the OMP_NUM_THREADS variable correctly. We recommend using as many threads as cores have the execution machine.

Compiling

This is a basic example witten in C to test compilation using OpenMP: “omp_hello.c”

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[])
{
int nthreads, tid;

/* Fork a team of threads giving them their own copies of variables */
#pragma omp parallel private(nthreads, tid)
  {

  /* Obtain thread number */
  tid = omp_get_thread_num();
  printf("Hello World from thread = %d\n", tid);

  /* Only master thread does this */
  if (tid == 0)
    {
    nthreads = omp_get_num_threads();
    printf("Number of threads = %d\n", nthreads);
    }

  }  /* All threads join master thread and disband */

}
```

And the fortran version: omp_hello.f

```
      PROGRAM HELLO

      INTEGER NTHREADS, TID, OMP_GET_NUM_THREADS,
     +        OMP_GET_THREAD_NUM

C     Fork a team of threads giving them their own copies of variables
!$OMP PARALLEL PRIVATE(NTHREADS, TID)


C     Obtain thread number
      TID = OMP_GET_THREAD_NUM()
      PRINT *, 'Hello World from thread = ', TID

C     Only master thread does this
      IF (TID .EQ. 0) THEN
```

```
        NTHREADS = OMP_GET_NUM_THREADS()
        PRINT *, 'Number of threads = ', NTHREADS
      END IF

C     All threads join master thread and disband
!$OMP END PARALLEL

      END
```

Intel compiler

C compiler

```
 icc -openmp omp_hello.c -o hello
```

Fortran compiler

```
ifort -openmp omp_hello.f -o hello
```

GNU compiler

C compiler

```
gcc -fopenmp omp_hello.c -o hello
```

Fortran compiler

```
gfortran -fopenmp omp_hello.f -o hello
```

Running your program

Before running your openMP executable you have to set the variable OMP_NUM_THREADS with the appropiate value. For example to execute 4 threads in our program, we can use the following command:

```
export OMP_NUM_THREADS=4
```

If we don't set this variable, by default it uses the maximum number of cores in the machine.

I recommend to use the maximum number of cores of a machine executing in queues, but not when we are testing in clonetroop.

Submit OpenMP jobs to the queue system SGE

This script is used to submit a OpenMP job. It takes the maximum number of cores available in the execution node.

```
#!/bin/bash
#
#$ -cwd
#$ -j y
#$ -S /bin/bash
#$ -V

# -cwd means to execute the job for the current working directory.
# -j y means to merge the standard error stream into the standard output
stream instead of having two separate files
# -S /bin/bash specifies the interpreting shell for this job to be the Bash
shell.
# -V imports current environment variables (path, libraries, etc.) to the
new shell

#$ -pe omp 8-12
export OMP_NUM_THREADS=$NSLOTS

echo "Current working directory is now: " `pwd`
echo "Starting job at `date`"
echo "Number of SLOTS: $NSLOTS"
echo "Number of THREADS: $OMP_NUM_THREADS"
echo "Running on host: " `hostname`

#Executable
./my_openmp_executable

echo "JOB run completed at `date`"
```