Euler Cluster Documentation

Release

Wisconsin Applied Computing Center

April 15, 2015

Contents

The Euler Cluster is operated by the Wisconsin Applied Computing Center at the University of Wisconsin - Madison.

CHAPTER 1

Contents

1.1 Accessing Euler

Euler is accessible via SSH at hostname `euler.wacc.wisc.edu`. Accounts for research projects may be requested by contacting Dan Negrut.

The following describes how to log in from various operating systems.

1.1.1 Linux / MacOSX / Unix

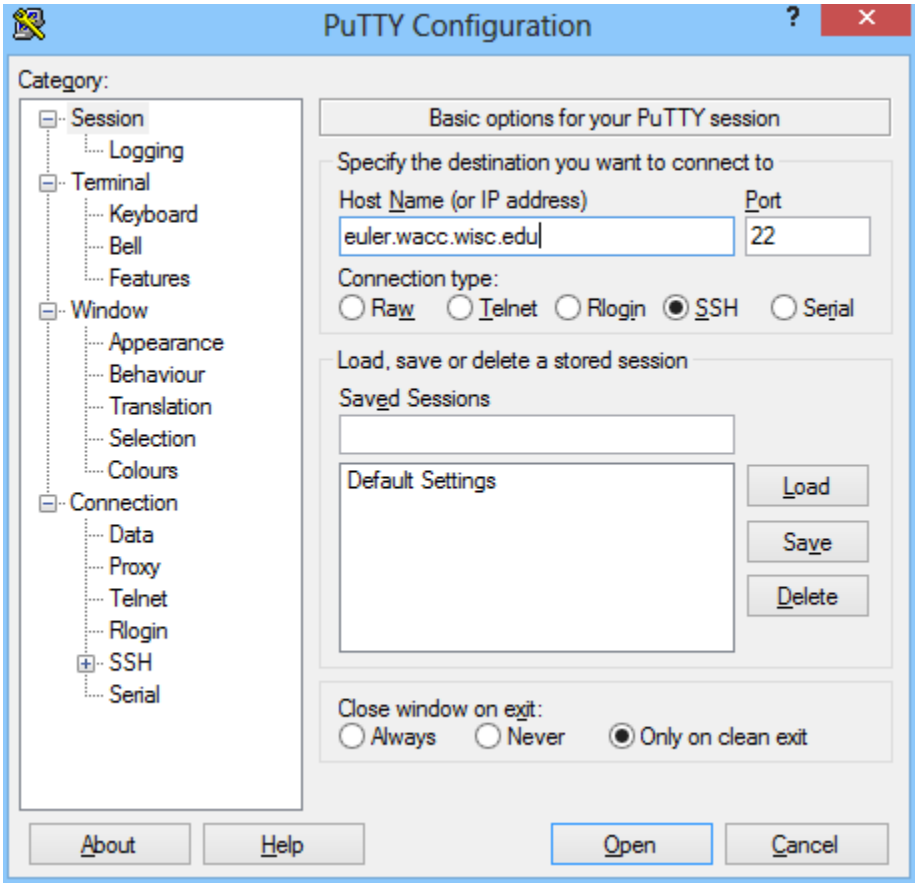
Most Unix-like systems include an SSH client. To access Euler, simply open the terminal and type `ssh username@euler.wacc.wisc.edu`, replacing `username` with the one provided to you.

```
macosx:~ demo$ ssh demo@euler.wacc.wisc.edu
demo@euler.wacc.wisc.edu's password:
Creating directory '/home/demo'.
[demo@euler ~]$
```

1.1.2 Windows

PuTTY is one of the most popular SSH clients for Windows. The installer version is preferred as it includes a few utilities that will be used later (PuTTYGen).

To use PuTTY, enter the server hostname (`euler.wacc.wisc.edu`) and click *Open*.




```
login: demo
demo@euler.wacc.wisc.edu's password:
[demo@euler ~]$
```

1.1.3 Key-Based Authentication (Optional)

Public key-based authentication allows you to access resources without explicitly entering a password every time. This section is optional and is not required in order to access Euler.

Linux / MacOSX / Unix

OpenSSH-based clients typically include a command named `ssh-keygen` which is used to generate the private and public keys. Once generated, the public key can be copied to the server using `ssh-copy-id`.

Note: MacOSX does not ship with `ssh-copy-id`; a copy is available here: ssh-copy-id.

On Linux:

```
[demo@linux ~]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/demo/.ssh/id_rsa):
Created directory '/home/demo/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/demo/.ssh/id_rsa.
```

```
Your public key has been saved in /home/demo/.ssh/id_rsa.pub.
[demo@linux ~]$ ssh-copy-id demo@euler.wacc.wisc.edu
demo@euler.wacc.wisc.edu's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'demo@euler.wacc.wisc.edu'"
and check to make sure that only the key(s) you wanted were added.
```

On MacOSX:

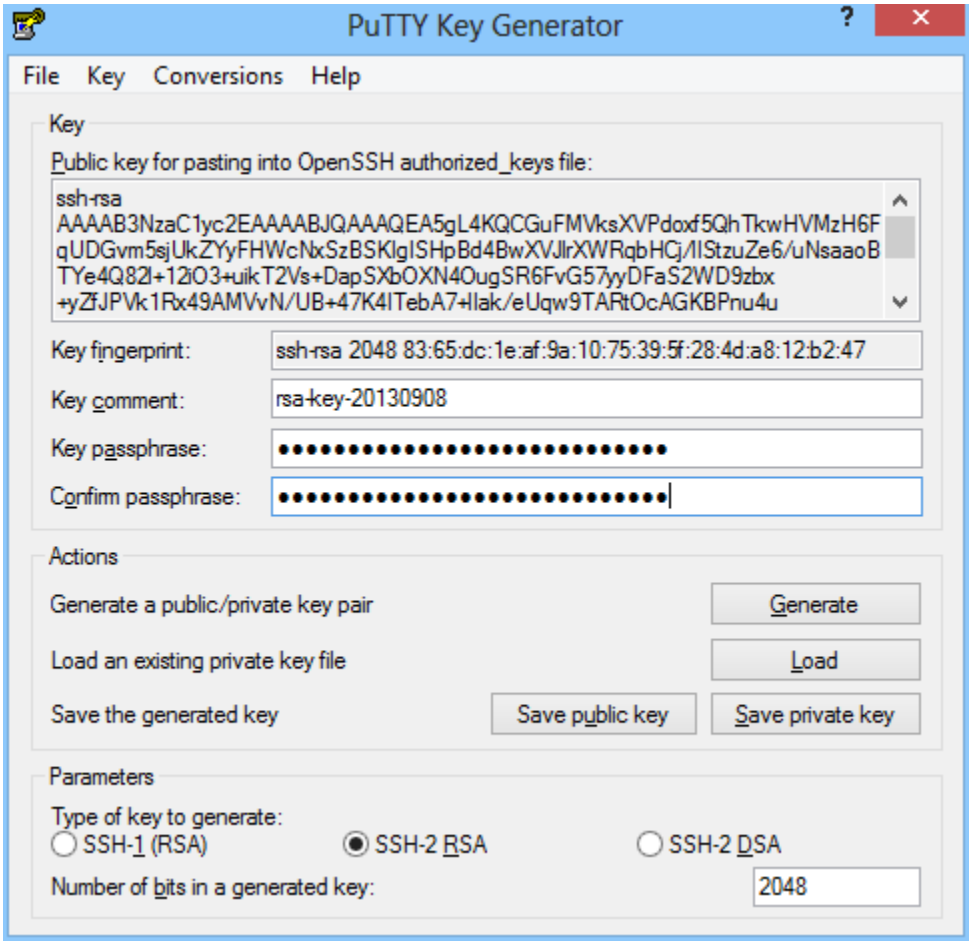
```
macosx:~ demo$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/demo/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /Users/demo/.ssh/id_rsa.
Your public key has been saved in /Users/demo/.ssh/id_rsa.pub.
macosx:~ demo$ curl -o ssh-copy-id https://gist.github.com/andrewseidl/6488345/raw/bffde4ea5cf23eebd8
macosx:~ demo$ less ssh-copy-id # inspect to make sure nothing changed
macosx:~ demo$ chmod +x ssh-copy-id
macosx:~ demo$ ./ssh-copy-id demo@euler.wacc.wisc.edu
demo@euler.wacc.wisc.edu's password:
Now try logging into the machine, with "ssh 'demo@euler.wacc.wisc.edu'", and check in:

  .ssh/authorized_keys

to make sure we haven't added extra keys that you weren't expecting.
```

Windows

In Windows, use the program PuTTYgen to generate the keys, saving the private key to a safe place. Copy the public key to the clipboard.



Next, log in to Euler as you normally would using PuTTY. Create a file named `~/.ssh/authorized_keys` which contains the public key you copied from PuTTYgen (right-click in PuTTY is mapped to paste).

```
[demo@euler ~]$ mkdir ~/.ssh && chmod 700 ~/.ssh
[demo@euler ~]$ cat >> ~/.ssh/authorized_keys << EOF
ssh-rsa AAAAB3NzaC1yc2...0seOoxPfSAwlQ== rsa-key-20130908
EOF
[demo@euler ~]$ chmod 600 ~/.ssh/authorized_keys
```

Finally, configure PuTTY to automatically send your username and the key. In a new session, set your username in *Connection* > *Data* > *Auto-login username* and select your private key in *Connection* > *SSH* > *Auth* > *Private key*. Return to *Session*, re-enter the hostname, enter a name under *Saved Sessions*, and hit *Save*. Double clicking on the session name should then automatically log you in to Euler.

1.1.4 Remote X Sessions via NX

It is sometimes desirable to use X11-/GUI-based programs from Euler. While Unix-like systems can forward this programs directly using `ssh -X`, Windows users typically need to install and configure an X server such as Xming. Alternatively, Euler allows users to create a remote X session using NoMachine NX.

> **Warning:** NX on Euler should be considered unsupported and legacy. It may not always work and may be replaced in the future.

To begin, download and install the NoMachine 4 Beta Client for your system. While that is installing, login to Euler

via SSH and run `cat /home/nx/client.id_dsa.key` to get a copy of NX's key. Save this to a file on your machine.

Next, open NoMachine and click the icon for *Add a computer*. Give the session a name ("Euler"), set the protocol to 'SSH', and enter the hostname (`euler.wacc.wisc.edu`). Click *Advanced* and select *Use the NoMachine login*. Click *Continue* and select the file you just created with NX's key. Click *Continue* to finish.

You should now be on a summary page with the new connection. Click *Connect* and enter your Euler login details. Once connected, click *Create a new virtual desktop* and select *Create a new GNOME virtual desktop*. After clicking *Continue* you should be presented with a desktop running on Euler where you can use GUI-based programs.

Note: Please remember to logout once you no longer require a desktop. NX is very resource-intensive compared to the terminal, which may cause issues if too many people are using it at once.

1.2 Using Euler

As of September 2013, Euler is running Scientific Linux 6.2.

There are numerous introductory tutorials to Linux available online, such as LinuxCommand.org.

1.2.1 Contents

Environment Modules

Euler uses Environment Modules to provide access to the various software packages available to users. These software packages include various versions of:

- Blender
- CUDA
- MVAPICH2
- OpenMPI
- Paraview
- Point Cloud Library

Running `module avail` will print a list of all modules available to you. See the Environment Modules documentation or `man module` for a list of all sub-commands. The main ones to pay attention to are:

`module avail` List modules available to you

`module load pkg/vers` Load the package given in *pkg/vers*

`module unload pkg/vers` Unload the package

`module initadd pkg/vers` Set *pkg/vers* to be loaded automatically at login

Submitting Jobs via TORQUE

The TORQUE Resource Manager is used for both resource management and job scheduling. Users wishing to run code on Euler must submit a job to TORQUE, which will then automatically send the job to a compute node which meets the job's requirements. The following provides only a cursory overview of how to submit and monitor jobs. Full documentation is available from Adaptive Computing.

Overview

Jobs submitted to TORQUE are configured by creating a shell script with special TORQUE/PBS-specific configuration comments. An example of these scripts is provided in your home directory, under the sym-linked directory *Example Jobs*. A few of these examples will be covered later on.

Each job is assigned on ID of the form `n.euler` where n is an incrementally-increasing integer. This ID is used to monitor job status, delete jobs, and to organize output and error logs.

`qsub`: Job Submission All configuration options may be set in either the job submission script or on the command line. The most basic job can be submitted by `qsub script.sh`, where `script.sh` is the name of your job submission script. One useful argument is: `-l nodes=n,ppn=x,gpus=y` to reserve *n* nodes and *y* GPUs per node, each with *x* processors (*ppn*: processors per node). Another useful argument is for tasks: `-t x-y,z` which will run the job *y-x+1* times, each with a different number from the set of the range *x-y*, and *z*. One place where this would be useful would be for selectively rendering frames of an animation.

`qstat`: Job Status Job status may be monitored via `qstat`. Without any arguments, this will list all recent jobs that have submitted. Next to each job ID will be a single character for the status. 'Q' for queued, 'R' for running, 'E' for error, and 'C' for completed. qstat will also show how long the job has run and how much time allotted remains. Depending on your submission settings, you will also receive an email saying whether your job completed or failed, and why. By default this is sent to your account on the cluster itself, which is accessible by the command mail.

`qdel`: Job Removal Jobs can be removed or cancelled via the command `qdel id` where *id* is the job's ID number.

`pbsnodes` and `pbstop`: Node Monitoring The command `pbsnodes -a` will give you details about all nodes in the cluster (available/used memory, number of jobs running on the node, various stats on the node's GPUs). The command `pbstop` will give you a nicer view of this data.

Job Submission Walkthrough

This section describes the steps required to log in to Euler and submit a basic GPU job.

A Basic Job As seen above, your home directory contains a symlink to the folder *Example Jobs*. This folder contains some example TORQUE job submission scripts. To run one of these, copy the `gpu-scan.sh` script to your home directory and submit it to the job manager using `qsub`.

```
#!/bin/sh
#PBS -N gpu-scan
#PBS -l nodes=1:gpus=1,walltime=00:01:00

$NVSDKCOMPUTEROOT/bin/linux/release/scan
```

The script above has two main sections: job settings and the script itself. Lines 3 and 4 set the job's name to *gpu-scan* (`-N gpu-scan`) and requests one node (`nodes=1`) with one GPU (`gpus=1`). Additionally, it tells the scheduler that the job will take at most 1 minute to execute (`walltime=00:01:00`). The last line executes the `scan` code example from the CUDA SDK.

There are three main things to note in this example. First, job settings are specified by `#PBS` followed by command line options for `qsub`. Since the settings are defined in comments, you can usually directly run this script from the command line without the TORQUE-specific options causing any issues. Next, all settings defined in the script can also be passed directly to `qsub` from the command line. See the `qsub` man page for more options (`man qsub`).

Lastly, note the use of the `$NVSDKCOMPUTEROOT` or `$CUDA_SDK` environment variables. This allows the same script to be used with differing versions of the CUDA SDKs, depending on which version you have selected via `module load cuda`.

```
[aseidl@euler ~]$ cp Example\ Jobs/gpu-scan.sh .
[aseidl@euler ~]$ qsub gpu-scan.sh
8117.euler
[aseidl@euler ~]$ qstat 8117
Job id        Name         User        Time Use S Queue
------------- ------------ ----------- -------- - -----
8117.euler    gpu-scan     aseidl      00:00:01 C batch
```

The example above shows how to copy the example job script to your home directory, submit it to the TORQUE job scheduler, and check on the status of the submitted job. When submitting the job, `qsub` will output the job ID assigned to it. This ID can then be use to check on the job's status with `qstat`. You can also enter `qstat` with no arguments to see the status of all recently submitted jobs.

The output of `qstat` shows the job ID assigned, the name given via the `-N` option, the user who submitted the job, walltime used, job status, and queue which the job was submitted to. Walltime is defined as the actual time that the job ran. Status is a single letter, the most common of which are: Q (queued), R (running), and C (completed). The queue allows you to submit your job to different sections of the cluster which may have different resources available or different restrictions on parameters such as walltime, number of GPUs requested, etc. Euler currently has a single queue, called 'batch'.

Interactive Jobs To run a job manually on a compute node, you can submit what is called an "interactive job" to TORQUE. This is done via the command `qsub -I`. Once the appropriate resources are available, you will be presented with a shell on one of the compute nodes, as shown below.

```
[aseidl@euler ~]$ qsub -I
qsub: waiting for job 8123.euler to start
qsub: job 8123.euler ready

[aseidl@euler01 ~]$
```

Note that unlike non-interactive jobs, interactive ones (and any processes your started during the interactive job) will terminate as soon as you exit the node or log out of Euler.

1.3 Developing on Euler

1.4 Accessing Newton

You can access Newton via SSH from Euler. If you do not know how to access Euler using SSH, you should read Accessing Euler first.

Before proceeding, you may not need to access newton directly! See Using Newton to be sure.

If you still feel that you need to access newton, follow the steps below.

1. Connect to Euler using SSH.
2. On Euler, run the command `ssh newton`.
3. Enter your password (optionally, run `ssh-copy-id euler` to do this automatically).
4. Use Newton as necessary, then exit by typing `exit` or by pressing `Control + D`.

1.4.1 Where is PBSWebmon?

This question has popped up enough that it seems to warrant a response. If you would like to have a graphical representation of the currently working jobs on the cluster, you can access the PBSWebmon service for newton here. Please note however, that due to some logistical concerns, this modified copy of the service does not provide many details about running jobs. If you would like to see lists of running jobs, use the `qstat [JobID]` command.

1.4.2 Requesting Software

If your project will not compile on Newton but normally would on Euler, there may be some packages not yet installed on the system. If you need something for this or some other reason, please contact the sysadmin to request that the package be installed.

1.4.3 User Etiquette

The usage of Newton and its sub-cluster is governed by the same rules of etiquette as the usage of Euler. Likewise, it is important that large compilations not be run on Newton, and instead are run on one of its worker nodes. These can be accessed in the same manner as Newton or any other node by running `ssh [nodename]` from Euler.

1.5 Using Newton

1.5.1 Submitting Jobs

Submitting a job to the Newton Sub-cluster is accomplished in a similar manner to the TORQUE jobs on euler. The easiest method is to add the directive `#PBS -q @newton` to your submission script. At this point, your job can be executed using `qsub [script]` from Euler or Newton.