



Building a Virtual Cluster with Xen (Part Two)

Written by Angel de Vicente

Friday, 06 October 2006

Clustering software makes it all work

In the first part of this [article](#) I showed you how to get the basic settings for our virtual cluster: we started with a fresh install of Xen; then we created five virtual machines (one master and four slaves), and then we configured the network and NFS, so that the users could share their home directories across the cluster. This is the basis for this second part of the article, in which we see how to install a number of packages that will allow us to run parallel programs on it and manage the cluster more efficiently. Concretely you will learn how to install the C3 command suite, the Modules package for easily switching environments, a version of MPICH for running parallel programs, and the Torque/Maui combination for job queue management. These packages (specially Torque/Maui) can be configured extensively according to your needs. For this virtual cluster we will use a minimal configuration, but this should be enough to get you started. If you are interested in just trying out the virtual cluster but don't want to perform all the steps yourself, you can grab a ready-made cluster image from the [download:contrib:cluster](#) page at [Jailtime.org](#).

Creating a Snapshot of The Cluster So Far

In the first part of the article we did quite a lot of work, and getting this far again if something goes wrong later on would be quite tedious. So before we continue let's create a snapshot of the cluster so far. Doing it is really simple. We just make sure we stop every machine with the halt command and then create a new directory to keep all the files. For this snapshot, we will create a directory called `cray-network.configuration`:

```
angelv@yepes:~$ sudo mv /opt/xen/cray /opt/xen/cray-network.configuration
angelv@yepes:~$ sudo mkdir /opt/xen/cray
angelv@yepes:~$ sudo rsync -azv /opt/xen/cray-network.configuration/ /opt/xen/cray/
```

The configuration files are still pointing to `/opt/xen/cray` which could always be our working version of the cluster. Previous snapshots will be stored in different directories as shown above, and we could revert to a previous version anytime, by just copying the appropriate directory to `/opt/xen/cray`

We will also create a small script to help us start the cluster, which we will save as `/etc/xen/cray/start-cluster.sh` with the following contents:

```
#!/bin/sh
xm create /etc/xen/cray/master.cfg
echo "waiting for master to start ...."
sleep 10
xm create /etc/xen/cray/slave1.cfg
xm create /etc/xen/cray/slave2.cfg
xm create /etc/xen/cray/slave3.cfg
xm create /etc/xen/cray/slave4.cfg
```

We just have to make it executable and run the script to start the cluster. Note that the script uses the `xm create` command without the `-c` option, so that we will not be connected to the console. After the five machines have booted, we can connect to them by either the `xm console` command or by ssh-ing to the master node.

```
angelv@yepes:~$ sudo chmod 755 /etc/xen/cray/start-cluster.sh
angelv@yepes:~$ sudo /etc/xen/cray/start-cluster.sh
```

NOTE: sometimes the loops in the system remain busy, when they should not. If the system complains about a loop being busy or if we just want to test for this before starting the cluster and rectify it in case there are busy loops we can do it with:

```
angelv@yepes:~$ for file in /dev/loop[0-9]* ; do sudo losetup $file ; done
angelv@yepes:~$ for file in /dev/loop[0-9]* ; do sudo losetup -d $file ; done
```

Basic Cluster Configuration

A cluster should be something more than just a collection of nodes. The idea should be to make those nodes behave as close as possible as if it was just a single machine. To somewhat help us attain this we will install the C3 Cluster Command and Control tool suite and the Modules package, which will make the usage of different versions of software easier. We will also deal with something smaller but important, the configuration of the time zone.

C3 Installation

The **Cluster Command and Control** (C3) tool suite "implements a number of command line based tools that have been shown to increase system manager scalability by reducing time and effort to operate and manage the cluster". By reading the **installation instructions**, we see that we will need to configure *rsync*, *perl*, and *rsh* (in a production cluster you should probably consider using *ssh* instead, but for the moment we will configure it with *rsh*, even for the root account).

In the master node we install these packages and put the RPMs in the */cshare* directory for the slaves to access (remember that we do not have Internet access from the slaves):

```
-bash-3.00# yum install rsync rsh rsh-server xinetd
-bash-3.00# cp /var/cache/yum/base/packages/rsync-2.6.3-1.i386.rpm /cshare/
-bash-3.00# cp /var/cache/yum/base/packages/rsh-* /cshare/
-bash-3.00# cp /var/cache/yum/base/packages/xinetd-2.3.13-4.4E.1.i386.rpm /cshare/
```

Then, in the slaves we first install these RPMs (with the command *rpm -ivh /cshare/*rpm*) and then we modify the files */etc/xinetd.d/rsh*, */etc/hosts.equiv*, */etc/securetty*, and */etc/pam.d/rsh* so that they contain the following (since we don't have editors in the slaves, you can edit them in the master, put them in the */cshare* directory, and then copy them from the slaves to their destination):

- In the file */etc/xinetd.d/rsh* change the line *disable = yes* to *disable = no*
- In the file */etc/securetty* add a line containing *rsh*
- In the file */etc/pam.d/rsh* change the line *auth required pam_rhosts_auth.so* by adding at the end *hosts_equiv_rootok*

Also, create the file */etc/hosts.equiv*, containing:

```
boldo
slave1
slave2
slave3
slave4
```

Then, (remember, only in the slaves) we start *xinetd*:

```
-bash-3.00# service xinetd start
```

At last, we are now ready to install C3 in the master node:

```
-bash-3.00# wget -nd http://www.csm.ornl.gov/torc/C3/Software/4.0.1/c3-4.0.1.tar.gz
-bash-3.00# tar -zxf c3-4.0.1.tar.gz
-bash-3.00# cd c3-4.0.1
-bash-3.00# ./Install-c3
-bash-3.00# ln -s /opt/c3-4/c[^0-9]* /usr/local/bin/
```

We create the configuration file */etc/c3.conf* with the following contents:

```
cluster boldo {
    boldo:192.168.1.10
```

[MOSIX for HPC clusters](#)

Make the most of your cluster(s) [based on Linux 2.6 ; 32 & 64 bits]
www.mosix.com

[HPC middleware solutions](#)

Platform, the global leader in high performance computing solutions
www.Platform.com

```

    slave[1-4]
}

```

And following the **installation instructions** I add the following line to the `/etc/profile` file in the master node:

```
export C3_RSH=rsh
```

After starting a new session, we can verify that `cexec` works fine, for user `root` as well as for user `angelv`, for example by running the command `cexec uname -a`

```

[angelv@boldo ~]$ cexec uname -a
***** boldo *****
----- slave1-----
Linux slave1 2.6.12.6-xenU #2 SMP Thu Aug 17 10:30:05 WEST 2006 i686 i686 i386 GNU/Linux
----- slave2-----
Linux slave2 2.6.12.6-xenU #2 SMP Thu Aug 17 10:30:05 WEST 2006 i686 i686 i386 GNU/Linux
----- slave3-----
Linux slave3 2.6.12.6-xenU #2 SMP Thu Aug 17 10:30:05 WEST 2006 i686 i686 i386 GNU/Linux
----- slave4-----
Linux slave4 2.6.12.6-xenU #2 SMP Thu Aug 17 10:30:05 WEST 2006 i686 i686 i386 GNU/Linux
[angelv@boldo ~]$

```

In order to make use of the `ckill` command, we need to get Perl installed in the slaves so we do (in the master node):

```

-bash-3.00# cp /var/cache/yum/base/packages/perl* /cshare/
-bash-3.00# cexec mkdir /opt/c3-4
-bash-3.00# cpush /opt/c3-4/ckillnode
-bash-3.00# cexec rpm -ivh /cshare/perl-*

```

Now, from the master node, we can verify that `ckill` works without problems:

```

[angelv@boldo ~]$ cexec sleep 120 &
[angelv@boldo ~]$ cexec ps -u angelv
[angelv@boldo ~]$ ckill sleep
[angelv@boldo ~]$ cexec ps -u angelv

```

For the moment we are not interested in `cpushimage`, so that's all we have to do for C3. It is a small set of tools, but very useful for daily maintenance, specially in large clusters in which you can create subsets of nodes on which to execute commands (see the documentation regarding the syntax of the `c3.conf` file if you are interested in this).

Configuration Of The Time Zone

The images downloaded from Jailtime.org have the timezone set to EDT. Chances are that your timezone is not that, so that you would like to change it. In order to do it, we can follow the steps in this **Red Hat page**, by doing this in the master node: First, we create the file `/etc/sysconfig/clock` with the following (obviously, you should adapt it to suit your location):

```

ZONE="Atlantic/Canary"
UTC=false
ARC=false

```

Next, we do (by using the C3 commands just installed):

```

-bash-3.00# ln -sf /usr/share/zoneinfo/Atlantic/Canary /etc/localtime
-bash-3.00# cpush /etc/sysconfig/clock
-bash-3.00# cexec ln -sf /usr/share/zoneinfo/Atlantic/Canary /etc/localtime

```

If we want now to restart our cluster, we can also make use of the C3 commands. To orderly stop all the nodes in the cluster we just have to do the following in the master node (remember this recipe for the future):

```
-bash-3.00# cshutdown t 0 -h
-bash-3.00# halt
```

Note: In a real cluster you would like to install something like NTP in your nodes so that time is kept synchronized across the cluster, but with the Virtual Cluster with Xen this is not necessary, as all the virtual machines have the same time as the host machine.

Installation of Modules

The **Environment Modules** package "provides for the dynamic modification of a user's environment via modulefiles", which can prove very useful in a cluster, so that for example we can install different parallel programming libraries and we can change from using one to another with a simple command, without the need to manually modify environment variables. Installation is easy (we will need to install *tc/tk* as well, since the scripts use it):

```
-bash-3.00# wget -nd http://kent.dl.sourceforge.net/sourceforge/modules/modules-3.2.3.tar.gz
-bash-3.00# yum install tcl tcl-devel
-bash-3.00# tar -zxf modules-3.2.3.tar.gz
-bash-3.00# cd modules-3.2.3
-bash-3.00# ./configure
-bash-3.00# make
-bash-3.00# make install
-bash-3.00# cd /usr/local/Modules
-bash-3.00# ln -s 3.2.3 default
```

Now, for the initial configuration of the package we need to copy some *dot* files, and we recreate the *angelv* user account to reflect the changes to the */etc/skel* directory:

```
-bash-3.00# pwd
/root/modules-3.2.3
-bash-3.00# cp /etc/bashrc /etc/bashrc.OLD
-bash-3.00# cp /etc/profile /etc/profile.OLD
-bash-3.00# cp /etc/skel/.bash_profile /etc/skel/.bash_profile.OLD

-bash-3.00# cat etc/global/bashrc /etc/bashrc.OLD > /etc/bashrc
```

Apparently there is an error in the resulting */etc/bashrc*, and in line 13 we should change `\$MODULE_VERSION` for `/\$MODULE_VERSION` (the leading */* was missing). We continue:

```
-bash-3.00# cat etc/global/profile /etc/profile.OLD > /etc/profile
-bash-3.00# cp etc/global/profile.modules /etc/profile.modules
-bash-3.00# cat etc/skel/.profile /etc/skel/.bash_profile.OLD > /etc/skel/.bash_profile

-bash-3.00# cpush /etc/bashrc
-bash-3.00# cpush /etc/profile
-bash-3.00# cpush /etc/profile.modules

-bash-3.00# cp /var/cache/yum/base/packages/tc* /cshare/
-bash-3.00# cexec rpm -ivh /cshare/tc*rpm

-bash-3.00# cd /cshare/
-bash-3.00# tar -cPf modules-dist /usr/local/Modules
-bash-3.00# cexec tar -xPf /cshare/modules-dist
```

```
-bash-3.00# userdel -r angelv
-bash-3.00# useradd angelv
-bash-3.00# passwd angelv
```

OK, so that's all we need for the moment, but note that this is a very basic configuration, and later on we should look into the modified files (specially */etc/profile*) to tailor them to our needs, although for the moment it should be enough. Note as well that we do not need to distribute the file */etc/skel/.bash_profile* to other nodes as user accounts are always created in the master node. Now, we can verify that the modules package is working as intended (as user *angelv*) with the following commands:

```
[angelv@boldo ~]$ module load modules
[angelv@boldo ~]$ module avail
```

In the following section, when installing MPICH, we will see how to create a *modulefile* and how to instruct the users to change their environment by using *modules* instead of modifying the environment variables directly.

Installing MPI (Message Passing Interface)

The MPI library that we will install will be **MPICH**, version 1.2.7p1. The link below should point to the most recent version, which is 1.2.7p1 at the time of writing. The installation is very simple:

```
-bash-3.00# wget -nd http://www-unix.mcs.anl.gov/mpi/mpich1/downloads/mpich.tar.gz
-bash-3.00# tar -zxf mpich.tar.gz
-bash-3.00# cd mpich-1.2.7p1/
-bash-3.00# ./configure --prefix=/usr/local/mpich/mpich-1.2.7p1
-bash-3.00# make
```

We modify the file *util/machines/machines.LINUX* to contain the name of the machines where we will want the MPI jobs to run:

```
-bash-3.00# cat util/machines/machines.LINUX
# Change this file to contain the machines that you want to use
# to run MPI jobs on.  The format is one host name per line, with either
#   hostname
# or
#   hostname:n
# where n is the number of processors in an SMP.  The hostname should
# be the same as the result from the command "hostname"
slave1:4
slave2:4
slave3:4
slave4:4
```

Then we can proceed with the installation

```
-bash-3.00# make install
```

Configuration Of The MPICH Modulefile

As mentioned above, we will use the modules package to let users easily change from one version of the library to another one (useful if later on we decide to install, for example, the **MPICH2** version of the library). We follow the instructions in the INSTALL file of the modules package and create the directory `/usr/local/Modules/3.2.3/modulefiles/mpich/`. Inside it we create two files:

```
-bash-3.00# cat /usr/local/Modules/3.2.3/modulefiles/mpich/mpich127p1
##Module1.0#####
##
## mpich 1.2.7p1 modulefile
##
## modulefiles/mpich/mpich127p1
##
proc ModulesHelp { } {
    global version mpichroot

    puts stderr "\tmpich 1.2.7p1 - loads the MPICH version 1.2.7p1 library"
    puts stderr "\n\tThis adds $mpichroot/* to several of the"
    puts stderr "\tenvironment variables.\n"
}

module-whatism "loads the MPICH 1.2.7p1 library"

# for Tcl script use only
set    version      1.2.7p1
set    mpichroot    /usr/local/mpich/mpich-1.2.7p1

prepend-path    PATH          $mpichroot/bin
prepend-path    MANPATH       $mpichroot/man
prepend-path    LD_LIBRARY_PATH $mpichroot/lib
```

```
-bash-3.00# cat /usr/local/Modules/3.2.3/modulefiles/mpich/.version
##Module1.0#####
##
## version file for MPICH
##
set ModulesVersion    "127p1"
-bash-3.00#
```

And we replicate them to the slave nodes:

```
-bash-3.00# cexec mkdir /usr/local/Modules/default/modulefiles/mpich
-bash-3.00# cpush /usr/local/Modules/default/modulefiles/mpich/.version
-bash-3.00# cpush /usr/local/Modules/default/modulefiles/mpich/mpich127p1
```

Now we can verify that as user `angelv` we can use `Modules` to load/unload this version of MPICH

```
-bash-3.00# su - angelv
[angelv@boldo ~]$ echo $PATH
[angelv@boldo ~]$ module avail
[angelv@boldo ~]$ module load mpich/mpich127p1
[angelv@boldo ~]$ echo $PATH          (you should see now the path to mpich127p1 included)
[angelv@boldo ~]$ module unload mpich/mpich127p1
```

[Scali Manage](#)
Cluster
Management and
Monitoring of your
HPC environment
www.platform.com/Scali

```
[angelv@boldo ~]$ echo $PATH                (the path should be again as per the first call)
[angelv@boldo ~]$
```

And since for the moment we will only have this version of MPICH installed, we can load this module by default by modifying the file `/home/angelv/.bash_profile`. The relevant lines of this file would look like:

```
# put any module loads here
module add null
module load mpich/mpich127p1
```

Now we can do a quick test to verify that MPICH works as intended. As *angelv* we do:

```
[angelv@boldo ~]$ cp /usr/local/mpich/mpich-1.2.7p1/examples/cpi.c .
[angelv@boldo ~]$ mpicc -o cpi cpi.c
[angelv@boldo ~]$ mpirun -np 17 cpi
```

As there are no problems, we distribute it to the slave nodes, by doing this in the master node as *root*:

```
-bash-3.00# cd /cshare
-bash-3.00# tar -cPf mpich-dist /usr/local/mpich
-bash-3.00# cexec tar -xPf /cshare/mpich-dist
```

Installing the Resource Manager and Scheduler

If you just wanted a cluster for yourself, you could probably get away without a resource manager and scheduler, but if you intend to let a number of people make use of the cluster, a job queue management system will help you increase its utilization and at the same time share the resources amongst the users in a *fair* way. Our choice of software to do this will be **Torque** as the resource manager and **Maui** as the resource scheduler.

Torque Installation

Torque installation is really simple:

```
-bash-3.00# wget -nd http://www.clusterresources.com/downloads/torque/torque-2.1.2.tar.gz
-bash-3.00# tar -zxf torque-2.1.2.tar.gz
-bash-3.00# cd torque-2.1.2
-bash-3.00# ./configure --prefix=/usr/local/torque/torque-2.1.2
-bash-3.00# make
-bash-3.00# make install
```

In order to use the modules package with this version of Torque, we create the directory `/usr/local/Modules/3.2.3/modulefiles/torque/`. Inside it we create two files (*Note: we don't replicate this to the slave nodes, as this is mainly for users to have access to the man pages and the bin files, but normally users would not need this from the slaves*):

```
-bash-3.00# cat /usr/local/Modules/3.2.3/modulefiles/torque/.version
##Module1.0#####
##
## version file for Torque
##
set ModulesVersion      "212"

-bash-3.00# cat /usr/local/Modules/3.2.3/modulefiles/torque/torque212
##Module1.0#####
##
## Torque 2.1.2 modulefile
##
```

```
## modulefiles/torque/torque212
##
proc ModulesHelp { } {
    global version torqueroot

    puts stderr "\ttorque 2.1.2 - loads TORQUE version 2.1.2"
    puts stderr "\n\tThis adds $torqueroot/* to several of the"
    puts stderr "\tenvironment variables.\n"
}

module-whatism "loads TORQUE 2.1.2"

# for Tcl script use only
set    version      2.1.2
set    torqueroot   /usr/local/torque/torque-2.1.2

prepend-path    PATH          $torqueroot/bin
prepend-path    MANPATH       $torqueroot/man
prepend-path    LD_LIBRARY_PATH $torqueroot/lib
```

Since for the moment we will have only one version of Torque, and we want to provide it by default to all users, we add the following line to the end of the file `/etc/bashrc`:

```
module load torque/torque212
```

To have access to torque commands, libraries, etc. from the root account we also need to get dot files for the root account, so we create the following two files (*Note: Remember to log out and log in again so that these changes take effect*):

```
-bash-3.00# cat /root/.bashrc
module load torque/torque212

-bash-3.00# cat /root/.bash_profile
# start .profile
if [ -f /etc/profile.modules ]
then
    . /etc/profile.modules
# put any module loads here
fi

sh() { bash "$@"; }

# end .profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi
```

We configure Torque following using the **Torque Quick Start Guide**. Specifically, we will use the **manual configuration instructions**.

```
-bash-3.00# /usr/local/torque/torque-2.1.2/sbin/pbs_server -t create
-bash-3.00# qmgr -c "set server scheduling=true"
-bash-3.00# qmgr -c "create queue batch queue_type=execution"
-bash-3.00# qmgr -c "set queue batch started=true"
-bash-3.00# qmgr -c "set queue batch enabled=true"
```

```
-bash-3.00# qmgr -c "set queue batch resources_default.nodes=1"
-bash-3.00# qmgr -c "set queue batch resources_default.walltime=3600"
-bash-3.00# qmgr -c "set server default_queue=batch"
```

We create the following two files:

```
-bash-3.00# cat /var/spool/torque/server_priv/nodes
slave1 np=4
slave2 np=4
slave3 np=4
slave4 np=4
```

```
-bash-3.00# cat /var/spool/torque/mom_priv/config
$usecp */:/home /home
```

And we replicate the installation to the slaves:

```
-bash-3.00# tar -cPf /cshare/torque_slaves /usr/local/torque
-bash-3.00# tar -cPf /cshare/torque_spool /var/spool/torque
```

```
-bash-3.00# cexec tar -xPf /cshare/torque_slaves
-bash-3.00# cexec tar -xPf /cshare/torque_spool
```

We then can restart the server and start the MOMs in the slaves and verify that they report to the server:

```
-bash-3.00# /usr/local/torque/torque-2.1.2/bin/qterm -t quick
-bash-3.00# cexec /usr/local/torque/torque-2.1.2/sbin/pbs_mom
-bash-3.00# /usr/local/torque/torque-2.1.2/sbin/pbs_server
-bash-3.00# /usr/local/torque/torque-2.1.2/bin/pbsnodes -a
                (after a time, we have to wait for the nodes to report)
slave1
    state = free
    np = 4
    ntype = cluster
[...]
```

Automatic Start of Torque at Boot Time

We probably want to start Torque automatically at boot time, which can be easily accomplished by creating the init file `torque_server` in the master node as follows (similar scripts can be found [here](#) or in the `contrib/init.d` directory of the source code). Remember to change the permission of these to 755, with the commands `chmod 755 /etc/init.d/torque_server`.

```
-bash-3.00# cat /etc/init.d/torque_server
#!/bin/bash
#
# Red Hat Linux Torque Resource script
#
# chkconfig: 345 80 80
# description: TORQUE is a scalable resource manager which manages jobs in
# cluster environments.

# Source function library.
. /etc/init.d/functions

TORQUEBINARY="/usr/local/torque/torque-2.1.2/sbin/pbs_server"
```

```

start() {
    if [ -x $TORQUEBINARY ]; then
        daemon $TORQUEBINARY
        RETVAL=$?
        return $RETVAL
    else
        echo "$0 ERROR: Torque server program not found"
    fi
}

stop() {
    echo -n "Stopping $prog: "
    killproc $TORQUEBINARY
    RETVAL=$?
    echo
    return $RETVAL
}

restart() {
    stop
    start
}

reload() {
    restart
}

case "$1" in
start)
    start
    ;;
stop)
    stop
    ;;
reload|restart)
    restart
    ;;
status)
    status $TORQUEBINARY
    ;;
*)
    echo $"Usage: $0 {start|stop|restart|reload|status}"
    exit 1
esac

exit $?
exit $RETVAL

```

Similarly, we would create *torque_mom* init scripts in the slaves, stored as */etc/init.d/torque_mom*, and with permissions 755. These files are almost identical to the file */etc/init.d/torque_server*, except for the following two changes (output from the *diff* command):

```

-bash-3.00# diff /etc/init.d/torque_server torque_mom_boldo_slave1
12c12
< TORQUEBINARY="/usr/local/torque/torque-2.1.2/sbin/pbs_server"
---
> TORQUEBINARY="/usr/local/torque/torque-2.1.2/sbin/pbs_mom"
20c20
<         echo "$0 ERROR: Torque server program not found"

```

```
---
> echo "$0 ERROR: Torque mom program not found"
```

Then we create the necessary symbolic links in both the master and the slaves:

```
-bash-3.00# /sbin/chkconfig --add torque_server
-bash-3.00# cexec /sbin/chkconfig --add torque_mom
```

And lastly we link the libtorque.so.0 file to `/usr/lib/` because later on Maui will need it at start-up time, and it will not be able to find it in its current location:

```
-bash-3.00# ln -s /usr/local/torque/torque-2.1.2/lib/libtorque.so.0 /usr/lib/
```

Maui Installation

A basic installation of Maui is straightforward, but if you plan on using it make sure you read the [Online Administrator's Guide](#) to understand all that it has to offer:

```
-bash-3.00# wget -nd http://www.clusterresources.com/downloads/maui/maui-3.2.6p14.tar.gz
-bash-3.00# tar -zxf maui-3.2.6p14.tar.gz
-bash-3.00# cd maui-3.2.6p14
-bash-3.00# ./configure --with-pbs=/usr/local/torque/torque-2.1.2/
```

With this version there seems to be a problem with the Makefile, which is looking for the `libpbs` library (which was the name for the Torque library in previous versions, but now is called `libtorque`), so in line 26 of the Makefile we change `-lpbs` to `-ltorque` and then we continue:

```
-bash-3.00# yum install libnet
-bash-3.00# make
-bash-3.00# make install
```

As for Torque previously, in order to use the modules package with Maui, we create the directory `/usr/local/Modules/3.2.3/modulefiles/maui/` and inside it we create two files:

```
-bash-3.00# cat /usr/local/Modules/3.2.3/modulefiles/maui/.version
##Module1.0#####
##
## version file for Maui
##
set ModulesVersion      "326p14"

-bash-3.00# cat /usr/local/Modules/3.2.3/modulefiles/maui/maui326p14
##Module1.0#####
##
## Maui 3.2.6p14 modulefile
##
## modulefiles/maui/maui326p14
##
proc ModulesHelp { } {
    global version mauiroot

    puts stderr "\tmaui 3.2.6p14 - loads MAUI version 3.2.6p14"
    puts stderr "\n\tThis adds $mauiroot/* to several of the"
    puts stderr "\tenvironment variables.\n"
}
}
```

```

module-whatism "loads MAUI 3.2.6p14"

# for Tcl script use only
set    version      3.2.6p14
set    mauiroot     /usr/local/maui

prepend-path PATH          $mauiroot/bin
prepend-path MANPATH       $mauiroot/man
prepend-path LD_LIBRARY_PATH $mauiroot/lib

```

Also, as per Torque, we want users and root to have access to the bin files, the libraries, etc. so we add to both `/etc/bashrc` and `/root/.bashrc` the line:

```
module load maui/maui326p14
```

We restart the root session and we start maui manually.

```
-bash-3.00# /usr/local/maui/sbin/maui
```

If all goes well you can check the job queue with the command `showq` from either the `root` or the `angelv` user account. We will test this after the following section on how to make Maui start at boot time.

Automatic Start Of Maui At Boot Time

For the automatic start of Maui at boot time, we just need to create the init file `maui` (a similar script can be found in the `etc/maui.d` directory of the source code). Remember to change the permissions to 755, with the command `chmod 755 /etc/init.d/maui`, and then to create the necessary symbolic links with the command `chkconfig --add maui`:

```

-bash-3.00# cat /etc/init.d/maui
#!/bin/bash
#
# Red Hat Linux Maui Resource script
#
# chkconfig: 345 90 90
# description: Maui is a cluster scheduler which uses
# TORQUE to schedule jobs on that cluster.

# Source function library.
. /etc/init.d/functions

MAUIBINARY="/usr/local/maui/sbin/maui"

start() {
    if [ -x $MAUIBINARY ]; then
        daemon $MAUIBINARY
        RETVAL=$?
        return $RETVAL
    else
        echo "$0 ERROR: Maui program not found"
    fi
}

stop() {
    echo -n $"Stopping $prog: "
    killproc $MAUIBINARY
    RETVAL=$?
}

```

```

        echo
        return $RETVAL
    }

    restart() {
        stop
        start
    }

    reload() {
        restart
    }

    case "$1" in
    start)
        start
        ;;
    stop)
        stop
        ;;
    reload|restart)
        restart
        ;;
    status)
        status $MAUIBINARY
        ;;
    *)
        echo $"Usage: $0 {start|stop|restart|reload|status}"
        exit 1
    esac

    exit $?
    exit $RETVAL

```

Verification Of Parallel Programming Execution

After all this work, we are nearly finished with our first version of the virtual cluster. To test that everything is working correctly, we will execute a parallel program, submitted to the cluster through Maui. First of all, we should reboot the cluster (remember the recipe we saw above), to verify that all the services are started at boot time correctly. Then, as an example of how to submit jobs to Maui and in order to verify the execution of a parallel programs submitted to the queue, we create two files in the *angelv* user account (*cpu-eater.c* and *submit-eater*):

```

[angelv@boldo ~]$ cat cpu-eater.c
#include "mpi.h"
#include

int main(int argc, char *argv[])
{
    int rank, size;
    int t;
    long i,j = 0;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    printf("Hello World from process %d of %d\n", rank, size);

```

```

for (i=0;i<100000;i++)
  for(j=0;j<100000;j++)
    if (!(i % 10000) && (j == 0) && (rank == 0))
      printf(".");

MPI_Finalize();
return 0;
}

```

```

[angelv@boldo ~]$ cat submit-eater
#!/bin/sh

```

```

# This finds out the number of nodes we have
NP=$(wc -l $PBS_NODEFILE | awk '{print $1}')
cd $PBS_O_WORKDIR

# Make the MPI call
mpirun -np $NP -machinefile $PBS_NODEFILE ./cpu-eater

```

cpu-eater.c is just the parallel version of your typical "Hello World" program, with a wasteful loop, so that the job does not complete immediately. *submit-eater* is the file needed to submit this job to our queuing system. We compile it, launch it a number of times to the queue with Maui, and verify that everything is working as expected:

```

[angelv@boldo ~]$ mpicc -o cpu-eater cpu-eater.c
[angelv@boldo ~]$ qsub -l nodes=2:ppn=2 submit-eater
[angelv@boldo ~]$ qsub -l nodes=2:ppn=2 submit-eater
[angelv@boldo ~]$ qsub -l nodes=2:ppn=2 submit-eater
[angelv@boldo ~]$ qsub -l nodes=2:ppn=2 submit-eater
[angelv@boldo ~]$ qsub -l nodes=4:ppn=4 submit-eater

[angelv@boldo ~]$ showq
ACTIVE JOBS-----
JOBNAME          USERNAME      STATE  PROC   REMAINING      STARTTIME
9                angelv       Running  4     00:58:58  Fri Jun 16 01:58:35
10               angelv       Running  4     00:59:14  Fri Jun 16 01:58:51
11               angelv       Running  4     00:59:16  Fri Jun 16 01:58:53
12               angelv       Running  4     00:59:17  Fri Jun 16 01:58:54

    4 Active Jobs      16 of   16 Processors Active (100.00%)
                        4 of    4 Nodes Active      (100.00%)

IDLE JOBS-----
JOBNAME          USERNAME      STATE  PROC   WCLIMIT      QUEUE TIME
13               angelv       Idle    16     1:00:00  Fri Jun 16 01:59:05

1 Idle Job

BLOCKED JOBS-----
JOBNAME          USERNAME      STATE  PROC   WCLIMIT      QUEUE TIME

Total Jobs: 5   Active Jobs: 4   Idle Jobs: 1   Blocked Jobs: 0
[angelv@boldo ~]$

```

Conclusions

Phew!! We did quite a lot of work to get here, but now we have a more or less functional virtual cluster. Many improvements can be made, but by now you should have the basic understanding to configure your own cluster. I would suggest you to create a snapshot of the cluster (as we saw above) and continue experimenting with many of the other features that you would perhaps want in a real production cluster (DHCP, LDAP, SystemImager, Highly Available services, Parallel File Systems, cluster monitoring software, etc.). As mentioned in the introduction, if you just want to try out the virtual cluster obtained by following the steps in Part One and Two of this article, but without doing all the configuration steps yourself, you can obtain a ready-made cluster image from the [download:contrib:cluster](#) page at [Jailtime.org](#). Happy (virtual) clustering!

Angel de Vicente, Ph.D., has been working during the last three years at the **Instituto de Astrofisica de Canarias**, giving support to the astrophysicists about scientific software and being in charge of supercomputing at the institute. Being in the process of upgrading their Beowulf cluster, he lives of late in a world of virtual machines and networks, where he feels free to experiment.

[Software defined radio](#)

Solutions for
radio jtrs sca &
wireless
www.nrismtech.com

Comment on this article

You must [login](#) to leave comments...

Other Visitors Comments

There are no comments currently....

Last Updated (Monday, 29 January 2007)

Close Window