Using the USF Penguin Cluster

Department of Computer Science
University of San Francisco

Updated January 28, 2014

USF's penguin cluster has 24 compute nodes and an Infiniband interconnect. Each node has two dual-core AMD processors. So in some sense the cluster is a $24 \times 4 = 96$-processor system. The first time you use the cluster, things are a bit complicated. So here's a step-by-step outline of how to use it. After you've used it once, it should be pretty easy to use in the future.

1 SSH and Logging On

1. It's very important that you're able to log on to the nodes of the cluster without typing a password. So if you haven't created an ssh key before, log on to a CS department computer (e.g., `stargate.cs.usfca.edu`) and type

   ```
   $ ssh-keygen -t rsa
   ```

   (The dollar sign ($) is the shell-prompt: don't type it.) You should follow the prompts, accepting the defaults that the program gives. After you've generated your ssh-key, you need to copy the "public" key to a special file:

   ```
   $ cd ~/.ssh
   $ cp id_rsa.pub authorized_keys2
   ```

   While you're still in the `.ssh` directory, create a file called `config` and add the lines

```
Host *
    ForwardAgent yes
```

The permissions on this need to be 600. So leave the text editor and type

```
$ chmod 600 config
```

Be sure to remember the password you gave to `ssh-keygen`. The good news here is that you shouldn't need to do this step ever again.

2. Now every time you log on to a CS department computer from a machine outside of the CS department, type

```
$ ssh-add
```

This will prompt you for your *ssh* password. After doing this, you can log on to any other CS department computer without typing a password.

If typing `ssh-add` gives you the message

```
Could not open a connection to your authentication agent
```

try typing

```
$ exec ssh-agent bash
$ ssh-add
```

This should prompt you for the password.

Note: If you're using `tcsh` instead of bash, replace `bash` with `tcsh` in the `exec` command.

3. Log on to `steelhead`.

```
$ ssh steelhead
```

The first time you log on to a particular CS machine using ssh, ssh will ask you if you're sure: just say yes. Steelhead is the main host for accessing nodes of the cluster.

4. Log on to the cluster by typing

   ```
   $ ncs login -n <number of nodes>
   ```

   Unless you've spoken to me, `<number of nodes>` should be *small:* four at the most, but two is probably plenty for most program development.

   The `ncs` program "checks out" the number of nodes you request (if they're available). It also creates some special files and starts up some daemons (special processes) that the system uses to help with parallel program startup and communication in parallel programs. Finally it logs you on to one of the nodes.

2 Compiling and Running

1. This is the easy part.

2. To compile an MPI program, use `mpicc.` For example, to compile the `mpi_hello.c` program type

   ```
   $ mpicc -g -Wall -o mpi_hello mpi_hello.c
   ```

   The `mpicc` command is just a "wrapper" for `gcc`. It runs `gcc` and makes sure that `gcc` knows where the MPI header files are and how to link in the MPI libraries.

   Note: Your CS home directory is "mounted" on all of the nodes of the cluster. So you shouldn't need to copy your program onto the cluster, and the location of any file on the cluster should be the same as it is on any other CS department machine.

3. To run the program, type

   ```
   $ mpiexec -n <p> ./mpi_hello
   ```

   You should replace `<p>` with the number of MPI processes you want to use.

3 Finishing Up

1. When you're done using the cluster, you can just log off by typing

   ```
   $ exit
   ```

   This will log you off of the cluster and return your nodes to the pool of available nodes.

2. If you want to check to be sure that your nodes are available (a very good idea), type the following command on steelhead

   ```
   $ ncs status
   ```

   This will provide you with a list of the statuses of all of the cluster nodes.

3. If this shows you still have nodes checked out, note their numbers and use ncs to check them back in. For example, if nodes penguin01, penguin02 and penguin17 are checked out to you, type

   ```
   $ ncs checkin 1 2 17
   ```

4 Using Different MPI Implementations

There are three different implementations of MPI installed on the penguin cluster: MVAPICH, MPICH, and OpenMPI. The default implementation (the implementation that you get when you log on) is MPICH. In order to use the other implementations, you'll need to first execute a `module load` command from the shell:

```
$ module load mvapich2-x86_64 # Switch to MVAPICH
```

or

```
$ module load openmpi-x86_64 # Switch to OpenMPI
```

You can find out which (if either) of these is loaded by running

```
$ module list
```

Note that you should only load one of MVAPICH or OpenMPI at a time. For example, if you've been using MVAPICH and you want to switch to OpenMPI, you should type

```
$ module unload mvapich2-x86_64 # Go back to MPICH
$ module load openmpi-x86_64 # Switch to OpenMPI
```

To switch back to MPICH, you just need to unload the currently loaded module: you don't need to load anything.

Note that you need to use the correct combination of `mpicc` and `mpiexec`. For example, trying to run a program compiled with the MPICH compiler using the OpenMPI `mpiexec` is guaranteed to cause problems.

Unfortunately, the default behavior for all three implementations is to run your program on a single node of the cluster. To get the various implementations to run on different nodes, you need to point them to a "`machines`" file, which lists the nodes you have checked out. These machines files are created by the `ncs` command:

```
~/.machines.mvapich2
~/.machines.openmpi
```

However, the `mpiexec` commands don't know where to find them, and the details of using them depend on the implementation.

4.1 MPICH and MVAPICH

MVAPICH is a modification of the MPICH distribution that uses Infiniband. The MPICH distribution uses ethernet. Since MVAPICH is derived from MPICH, using it is pretty much the same as using MPICH: both implementations use the same format for the machines file, and both use the same command line format to access it. So you can use `machines.mvapich2` for both, and the `mpiexec` command should look something like

```
$ mpiexec -f ~/.machines.mvapich2 -n <p> \
      <mpich or mvapich executable>
```

So you might want to create an alias for the first part of this command and put it in your `.bashrc` file:

```
alias mpich='mpiexec -f ~/.machines.mvapich2'
```

With this change, you can run MPICH or MVAPICH programs with the command

```
$ mpich -n <p> <mpich or mvapich executable>
```

Note that you'll still need to use the module command to switch between MPICH and MVAPICH.

4.2 OpenMPI

OpenMPI also uses ethernet for network communication, and it is somewhat more difficult to use than MPICH/MVAPICH, but it often has better performance than MPICH. In order to start an OpenMPI program it needs the full path to the executable:

```
$ /usr/lib64/openmpi/bin/mpiexec -machinefile ~/.machines.openmpi \
      -n <p> <openmpi executable>
```

So it's a good idea to add the following alias to your `.bashrc` file:

```
alias openmpi='/usr/lib64/openmpi/bin/mpiexec -machinefile \
   ~/.machines.openmpi'
```