Self-Led Lab Exercises for Raspberry Pi Cluster

- Build a compute cluster using Raspberry Pis
- Set up an environment similar to a NASA supercomputer
- Run a hydrodynamics visual demo

Introduction

The goal of this document is to show you how to build a compute cluster using Raspberry Pis. In addition, we'll help you set up an environment that will be very similar to what is used on one of the fastest supercomputers in the world: The Pleiades supercomputer at NASA's Advanced Supercomputing (NAS) Division. When you're finished you'll have a cluster that runs the same scheduling software as Pleiades (Portable Batch Scheduler or PBS), and you'll be able to run a cool hydrodynamics visual demo that you can show off to your friends!

This guide does assume you have an understanding of networking, Linux environments, and technical troubleshooting. If you run into trouble, you'll need to use your favorite search engine to find answers.

Let's start with what you'll need:

- 8 Raspberry Pis (you can get by with less...but try for at least 4)
- 8 MicroSD cards (we went with 32GB cards)
- SD card reader - most modern laptops and desktop PCs have SD card slots, but you may need to purchase an SD card reader (they usually plug into a USB port) in order to copy the necessary software to the SD card
- USB Multiport Charging hub (we used the Anker Powerport-10)
- 8 1-ft USB to micro-USB charging cables to plug your Raspberry Pis into the USB charging hub
- Ethernet Switch – you will need enough ports for all of the Raspberry Pis and an additional port to connect to your internet switch
- 8 short (1-ft or 2-ft) ethernet cables
- 1 additional ethernet cable (however long you need it to be) to connect your Pi ethernet switch to your internet switch
- 1 HDMI display (if you want to show off the visual demos)
- 1 HDMI cable
- Case (optional) - we used an 8-layer GeauxRobot Dog Bone Stack clear case

Environment

Pleiades has thousands of machines (or nodes) whose sole purpose is to perform computations. A smaller number of machines are set aside for storage and scheduling. In our cluster, we'll use one Pi as the front end for the entire cluster. The front-end will run the scheduler that kicks off your programs. We'll also set aside another Pi just for storage. This is where your programs will be stored along with any data generated by your programs. That leaves six Pis dedicated for computation. So now that we have a general idea of what we want to build, let's build it!

Construction

If you've decided to purchase a case (or cases) for your Raspberry Pi nodes, assemble this now. Afterwards power up your Pi switch and connect each Pi to the switch. Then run another cable from your Pi switch over to your internet switch (it's very difficult to install software on your Pis without internet connectivity). We'll hold off on powering up the Pis until we've copied the necessary software to the MicroSD cards.

Flashing the MicroSD cards

First you'll need to download the operating system that will run on each of the nodes in your cluster. This can be downloaded from https://www.raspberrypi.org/downloads/raspbian. Download the ZIP file for Raspbian Stretch Lite to your local machine. This is a non-graphical version of the operating system. Installing a desktop windowing environment on each Pi is not necessary since you'll probably be accessing the cluster using a regular PC or Mac.

Next you'll need to download and install a program called Etcher to your local machine from https://www.balena.io/etcher/. This will allow you to copy the ZIP file you downloaded earlier to the SD cards. Once you have Etcher downloaded...

1. Attach one of the SD cards to your local machine.
2. Open Etcher and select the ZIP file downloaded earlier which contains the Raspbian operating system.
3. Select the SD card that you'd like to copy Raspbian to.
4. Review your selections and then click "Flash" to copy the operating system to the SD card.
5. Repeat the above instructions for all of SD cards.

Enabling SSH on the boot volume

When accessing any compute cluster, SSH is the method of choice. Fortunately, SSH comes preinstalled with the Raspbian operating system, but it needs to be enabled first. The easiest way to do this is to create a blank text file called "**ssh**" on the boot volume of each SD card:

1. You can access the boot volume by simply plugging the SD card into your local machine. It will show up on your computer as "**/boot**" if you're using a Mac or Linux machine. It will show up as a drive letter on a Windows machine.
2. Once the SD card is mounted, create a blank file called "**ssh**" in the root directory of the "**boot**" volume on the SD card.
3. Repeat this process for all of the SD cards.

Finding IP Addresses

You will need to get the IP addresses assigned to your Pis. Without knowing the IP addresses, it will be very difficult to configure. There are a couple of different ways to find the IPs. One way is to login to your internet router and view the list of IPs assigned by the DHCP service. This varies from router to router, but it should be fairly easy to find. Another method is to download and install a utility called **nmap**. **nmap** is a free utility for profiling networks. **nmap** can be used to scan your network and show you which IPs are in use. For example, assuming your network is **192.168.0.0**, you would use the following command to scan your network using **nmap**:

```
nmap 192.168.0.0/24
```

Whether you're using the DHCP service on your router, or nmap, you'll want to get a list of assigned IPs **before** turning on any of your Pis. After this, you can turn on each Pi – one at a time – and use **nmap** or DHCP to look for the new IP that shows up.

Once you've taken inventory of your existing IPs, do the following for each Pi:

1. Insert an SD card in the slot on the underside.
2. Connect a cable from the USB charging hub to the micro-USB port on the Pi. This will power on the Pi. Wait a minute or so for it to boot.
3. Once you've found the IP address (using DHCP or **nmap**), make note of the IP address and which physical Pi it corresponds to.

Hosts file

Now that we have IPs for all of the Pis, we can login to each using SSH (which we enabled earlier) and begin the configuration. You should now have a list of all of the IP addresses and which physical Pi they correspond to. Now is a good time to assign names to each Pi. We called our front-end Pi "**admin**", our storage node was called "**nfs**", and the compute nodes were named "**rpi1**", "**rpi2**", etc. It's your cluster, so you can give the Pis any names you want. Make a list of the names and the corresponding IPs, this will become the hosts file on each Pi (which will allow you to refer to each Pi by name rather than having to remember the IP address). Repeat the following on every Pi in the cluster:

1. From your local machine, connect to one of your Pis using the following command:

   ```
   ssh pi@x.x.x.x
   ```

   (where *x.x.x.x* is the IP number of your Pi)

   The login is **pi** and the default password is **raspberry**.

2. Open the hosts file in the nano text editor:

   ```
   sudo nano /etc/hosts
   ```

3. To the bottom of the file, append your list of IPs followed by the names. Each entry should look something like this:

   ```
   …
   192.168.1.200   front-end
   192.168.1.201   nfs
   192.168.1.202   node1
   ...
   ```

4. Save the file when you're finished editing. Just hit **Ctrl+X** and then **Y** to save.
5. Once the hosts file is saved and you're back at the prompt type '**hostname'** followed by the name of the Pi that you're currently logged into. For example, if you want the machine you're logged into to be called '**sleepy,'** type the following:

   ```
   hostname sleepy
   ```
6. Next it would be a good idea to change the default password. Just type '**passwd'** and follow the instructions.

Upgrade the operating system

All of your Pis should now be online with new fancy names and internet connectivity. Login to each Pi and run the following commands to upgrade the operating system on each Pi:

1. sudo apt update
2. sudo apt upgrade
3. sudo reboot

SSH configuration

SSH (or secure shell) is the how users connect to compute clusters to run their programs. It's also how the nodes communicate with each other. In order to allow this communication, we'll need to generate public and private keys on the front-end and distribute the public keys to all of the nodes. Start by logging in to your front-end node and do the following:

1. Generate a keypair:

   ```
   ssh-keygen -t rsa
   ```
2. You will be asked where to save the keys. Just hit **Enter** to accept the default.
3. You'll be asked to enter a passphrase. To keep things simple, just hit **Enter** to use no passphrase.
4. Use your list of hostnames and use the following command to copy the public key from the front-end to each of the other nodes:

   ssh-copy-id pi@*whateverthehostnameis*
5. After you've run the above command to copy the public key to the other nodes, reboot the front-end (**sudo reboot**) and make certain that you can **ssh** into the other systems without a password. Use the following command to test logging into each node from the front-end:

   ssh *whatevernameyouused*

NFS configuration

NFS (network file system) allows you to share files between your Pi nodes. Without it, you would need to copy your programs to each node before running them. With NFS, you can keep your programs at a single location and share them with all of the nodes. Start by logging into your storage node using **ssh**:

1. Install NFS server software:

   ```
   sudo apt install nfs-kernel-server
   ```

2. Create NFS server shared directory:

   ```
   sudo mkdir /mnt/nfsserver
   ```

3. Change the permissions of the new folder:

   ```
   sudo chmod -R 777 /mnt/nfsserver
   ```

4. Type '**sudo nano /etc/exports'** and add the following line to the end of the file:

   ```
   /mnt/nfsserver *(rw,sync)
   ```

   Close and save the file.

5. Tell NFS to read the exports file you created above:

   ```
   sudo exportfs
   ```

6. Reboot the storage node.

7. After reboot log back into the storage node and create an empty file in the shared directory:

   ```
   sudo touch /mnt/nfsserver/blankfile
   ```

   This file can be used to verify that files are being shared.

Next, you'll need to login to the front-end and each of the compute nodes to configure the NFS client software using the following steps:

1. Install NFS client:

   ```
   sudo apt install nfs-common
   ```

2. Create the mount point for the NFS share:

   ```
   sudo mkdir -p /mnt/nfs
   ```

3. Allow the pi user to access the NFS share:

   ```
   sudo chown -R pi:pi /mnt/nfs
   ```

4. Mount the NFS share. Replace ***x.x.x.x*** with the IP number of the NFS server:

   ```
   sudo mount x.x.x.x:/mnt/nfsserver /mnt/nfs
   ```

5. Type '**sudo nano /etc/fstab'** and add the following line to the end of the file (this will automount the nfs share whenever the machine boots):

   ```
   x.x.x.x:/mnt/nfsserver     /mnt/nfs      nfs     rw 0 0
   ```

6. Close the file and save.

7. Reboot. Log back into the node you just configured. Run the following command:

   ```
   ls /mnt/nfs/
   ```

Do you see the blank file in the listing? If so, NFS file sharing is working. Repeat the above steps on the next node…

Prep for scheduler and other parallel software

We're going to need to build the scheduler from source code, but before we do that we'll need to install multiple packages to make this work:

1. Log into the front-end and run the following install:

   ```
   sudo apt install gcc make libtool libhwloc-dev libx11-dev libxt-dev libedit-dev ncurses-dev perl postgresql-server-dev-all postgresql-contrib python-dev tcl-dev tk-dev swig libexpat-dev libssl-dev libxext-dev libxft-dev autoconf automake git
   ```

2. Install the following on the front-end. Also, install these on the compute nodes:

   ```
   sudo apt install expat libedit2 postgresql python postgresql-contrib sendmail-bin tcl tk libical2 libglew-dev mpich2
   ```

Install PBS

Now that all necessary software has been installed. We can download the source code for PBS and compile it. Start by logging into the front-end node:

1. Go the shared NFS directory:

   ```
   cd /mnt/nfs
   ```

2. Download the PBS source code:

   ```
   git clone https://github.com/PBSPro/pbspro.git
   ```

3. Change to m4 directory. We'll need to make some changes…

   ```
   cd pbspro/m4
   ```

4. Run the following command. This will allow you to compile PBS on the ARM processor (instead of x86):

   ```
   sed -i 's/x86_64-linux-gnu/arm-linux-gnueabihf/g' *.m4
   ```

5. Run the following commands to compile PBS (this could take a half hour or more):

   ```
   cd /mnt/nfs/pbspro

   ./autogen.sh

   ./configure –prefix=/opt/pbs

   make

   sudo make install
   ```

6. Run the following script to complete the install on the front-end node:

   ```
   sudo /opt/pbs/libexec/pbs_postinstall
   ```

7. Type '**sudo nano /etc/pbs.conf**'. Modify the file so it looks like this:

    ```
    PBS_SERVER=admin (change 'admin' to the name of your front-end node!)
    PBS_START_SERVER=1
    PBS_START_SCHED=1
    PBS_START_COMM=1
    PBS_START_MOM=0
    PBS_EXEC=/opt/pbs
    PBS_HOME=/var/spool/pbs
    PBS_CORE_LIMIT=unlimited
    PBS_SCP=/usr/bin/scp
    ```

8. Set file permissions and start PBS:

    ```
    sudo chmod 4755 /opt/pbs/sbin/pbs_iff /opt/pbs/sbin/pbs_rcp
    sudo /etc/init.d/pbs_start
    ```

Now login to each of the compute nodes and do the following:

1. cd /mnt/nfs/pbspro
2. sudo make install
3. Run the following script to complete the install on this compute node:

    ```
    sudo /opt/pbs/libexec/pbs_postinstall
    ```

4. Type '**sudo nano /etc/pbs.conf**'. Modify the file so it looks like this:

    ```
    PBS_SERVER=admin (change 'admin' to the name of your front-end node!)
    PBS_START_SERVER=0
    PBS_START_SCHED=0
    PBS_START_COMM=0
    PBS_START_MOM=1
    PBS_EXEC=/opt/pbs
    PBS_HOME=/var/spool/pbs
    PBS_CORE_LIMIT=unlimited
    PBS_SCP=/usr/bin/scp
    ```

5. Set file permissions and start PBS:

    ```
    sudo chmod 4755 /opt/pbs/sbin/pbs_iff /opt/pbs/sbin/pbs_rcp
    sudo /etc/init.d/pbs_start
    ```

Once PBS is running on the front-end and all compute nodes, log back into the front-end:

1. Create a default queue and set scheduler defaults:

```
sudo /opt/pbs/bin/qmgr -c “create queue dev queue_type=e,started=t,enabled=t”
sudo /opt/pbs/bin/qmgr -c “set server default_queue=dev”
sudo /opt/pbs/bin/qmgr -c "set server job_history_enable=true"
sudo /opt/pbs/bin/qmgr -c "set server flatuid=true"
```

2. Register each compute node on the front-end:

   `sudo /opt/pbs/bin/qmgr -c “create node X”` (replace “*X*” with the name of the compute node)

3. Check whether all of the nodes are listed:

```
pbsnodes -a
```

Running a test MPI program using PBS

We’ll run a **hello world** program on multiple processors to make certain that our PBS cluster is working properly. Log into the front-end node:

1. Change to shared NFS directory:

```
cd /mnt/nfs
```

2. Create a directory for your new program, and change to that directory:

```
mkdir helloworld
cd helloworld
```

3. Run ‘**nano helloworld.c**’ and enter the following code:

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char** argv) {
    // initialize MPI environment
    MPI_Init(NULL,NULL);

    //get # of processes
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    //get rank of the process
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    //get name of processor
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(processor_name, &name_len);
    //print a hello world message
    printf(“Hello world from processor %s, rank %d out of %d processors\n”,
        processor_name, world_rank, world_size);
    //Finalize MPI environment
    MPI_Finalize();
}
```

4. Exit the file and save.
5. Compile the code
   ```
   mpicc -o helloworld helloworld.c
   ```
6. Now that we have an executable that we can run on multiple nodes, let’s write a script that will distribute the program using PBS. Run ‘**sudo nano run_hello**’ and enter the following:
   ```
   #PBS -lselect=6:ncpus=4
   mpiexec /mnt/nfs/helloworld/helloworld
   ```
7. Exit the file and save.
8. Run the script using PBS:
   ```
   qsub run_hello
   ```
9. Take note of the job ID. A couple of files will be created in the same folder as the script called **run_hello.e[*job_id*]** and **run_hello.o[*job_id*]**. The file with the extension starting with “**e**” is an error file (hopefully, this is empty). The file with the extension starting with “**o**” is the output file. Viewing this file should give you the output of the above program.

TinySPH Visual Demo

For this last demonstration, we actually won’t use PBS, since this is an interactive process. As always, start by logging into the front-end node:

1. Change to the shared NFS directory
   ```
   cd /mnt/nfs
   ```
2. Pull the source code from github:
   ```
   git clone https://github.com/TinyTitan/SPH
   ```
3. Enter the SPH folder:
   ```
   cd SPH
   ```
4. Edit the makefile (‘**nano makefile**’). Change the 3rd line to this:
   ```
   LDFLAGS+=-L$(SDKSTAGE)/opt/vc/lib/ -lbrcmGLESv2 -lGLEW -lbrcmEGL -lopenmaxil -lbcm_host -
   lvcos -lvchiq_arm -lpthread -lrt -L../libs/ilclient -L../libs/vgfont -lfreetype
   ```
5. Exit the file and save.
6. Run the **make** script:
   ```
   make
   ```
7. If all goes well, copy **sph.out** to the **/mnt/nfs** directory:
   ```
   cp sph.out ..
   ```

8. Go back to the **/mnt/nfs** directory:

   ```
   cd ..
   ```

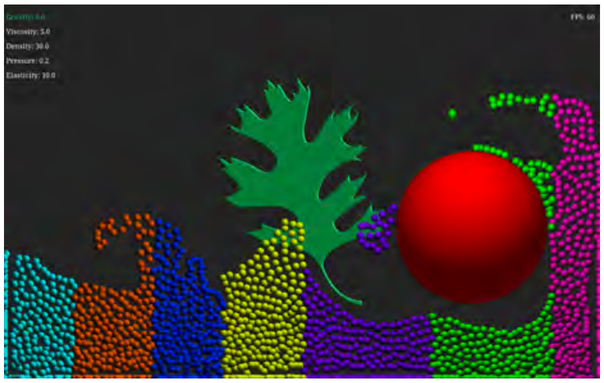
9. Create a machine file (‘**nano machinefile**’) add the IPs for the admin nodes and the compute nodes. There should be an IP address on each line.

10. Exit the file and save.

11. Run the program:

    ```
    mpiexec -f machinefile -n 7 /mnt/nfs/sph.out
    ```

If all went well, you should see something like this:



- **Ctrl+C** exits the program.
- The **L** key will switch between water and particle mode.
- Arrow keys can be used to modify the values in the upper-left-hand corner.