PelicanHPC: A Linux Cluster Distribution for MPI-based Parallel Computing

Michael Creel[1]

Department of Economics and Economic History, Edifici B, Universitat Autònoma de Barcelona, Cerdanyola del Vallès (Barcelona) 08193, Spain

Email address: michael.creel@uab.es (Michael Creel)

[1]Universitat Autònoma de Barcelona, Barcelona Graduate School of Economics and MOVE. This research was funded by projects MICINN-ECO2009-11857 and SGR2009-578.

Abstract

This paper describes the PelicanHPC distribution of GNU/Linux, which can be used for rapid and simple creation of a cluster for MPI-based parallel computing. Different usage models are discussed, including short term ad-hoc clustering versus a long term, remotely administered cluster, and the possibilities for mixing real and virtualized hosts. Means of customizing the distribution for specific tasks are presented, and some examples of derived distributions are noted. Documentation, performance, and security are discussed.

Keywords: linux cluster, MPI, live CD/USB

1. Introduction

This paper introduces and describes PelicanHPC[2] (Creel, 2008), a GNU/Linux distribution focused on MPI-based parallel computing on a cluster. PelicanHPC first appeared in January 2008, as the continuation and evolution of the ParallelKnoppix project (Creel, 2005), which had its first release in 2004. The main goal of this line of work has been and continues to be the provision of a flexible and versatile system for simple and rapid creation of a cluster for MPI-based computing. A basic PelicanHPC cluster can be created in less than five minutes, with no special technical knowledge. An ad hoc cluster of this sort can be useful for a number of purposes, including making use of idle resources for a pressing computational problem, demonstrating a computational solution during a presentation, or for teaching or learning about clustering and parallel computing. It is also possible to use PelicanHPC on a long term basis, with automatic setup, use of permanent storage, and with compute nodes can be booted automatically or remotely. Respecting the goal of simplicity, which keeps maintenance needs to a minimum, the project has evolved in the direction of improving the flexibility and customizability of the system, so that is can be useful to researchers in a variety of fields. ParallelKnoppix, PelicanHPC and derivatives have been used as research tools in a number of fields, including image processing (Dietlmeier, Begley and Whelan, 2008), solid state physics (San Sebastian et al. 2008), mathematical programming (Grytsenko and Peratta, 2007), econometrics (Creel and Kristensen, 2011), cheminformatics (Abreu et al., 2010) and bioinformatics (Chew et al., 2011). This paper describes the system used to create PelicanHPC, how PelicanHPC may be used, and how it may be adapted to meet specific needs.

2. Description

PelicanHPC is released as a binary-hybrid ISO image that is built from packages from Debian GNU/Linux[3], using the Debian Live system[4]. The image can be copied to a CD or USB storage device, which may then be used to boot a computer. Alternately, the ISO image may be used to boot a virtual machine, or may be booted directly using a bootloader capable of booting ISO images from disk (e.g., grub2). Once a computer, either real or virtual, has been booted from the image, it is referred to as the "frontend" node. Once logged into the account "user" on the frontend, one may run the `pelican_setup` script, which configures the frontend as a netboot server (providing services such as dhcp, tftp, nfs). After this is done, other computers on the network that are configured in BIOS to netboot will boot as compute nodes, receiving their filesystems from exactly the same image as was used to boot the frontend node. It is possible to run the frontend and/or the compute nodes as a virtual appliances on any of the popular operating systems, and real and virtualized systems may be mixed on a cluster. Because all computers in the cluster are booted from the same image, software versions are automatically synchronized across all nodes. The compute nodes obtain their filesystems by decompressing a single file that is exported by the frontend node. This scheme minimizes network traffic, at the expense of higher memory requirements for the compute nodes. Additionally, the `/home` directory of the frontend node is NFS-shared with all of the compute nodes. New software installed in `/home` on the frontend is automatically available on the whole cluster.

PelicanHPC has a single user account, with the username "user". No queuing or job scheduling software is included on the released images. PelicanHPC is designed to be used by one user at a time, who has full access to all of the resources of the cluster. A PelicanHPC cluster is often created when needed, and the machines that form the cluster return to their normal uses when the clustering work is finished. Such a scenario could take place in the computer room facilities of a university, for example, where the cluster could be used for research at night, while the computers are used during the day by students[5]. It is also possible to run a PelicanHPC cluster for a long period of time, with unsupervised booting of all nodes (supposing the compute nodes have been configured for wake-on-LAN) and recuperation of state across boots. If a long term installation is to be shared by researchers, one can add user accounts as needed. The entire `/home` directory is shared across nodes, so new user accounts will

[2] http://pareto.uab.es/mcreel/PelicanHPC/
[3] Debian: http://www.debian.org/
[4] Debian Live: http://live.debian.net/
[5] See the video http://www.youtube.com/watch?v=NBgAeKEt9uo for such an example.

also also able to use the cluster. If this is to be done, it may be desirable to add queuing software. It is expected that in most cases, PelicanHPC will be used by a single researcher, and this paper focuses on this form of usage.

As noted, PelicanHPC evolved from the ParallelKnoppix project. ParallelKnoppix images were created by remastering the Knoppix GNU/Linux distribution (refs). The remastering process could be partly scripted, but still involved a fair amount of interactive work, with the consequent possibilities for errors, and difficulties of exact replication of the process. The process also required a fair amount of knowledge on the part of the person performing the remastering. The fundamental difference between PelicanHPC and ParallelKnoppix is that the PelicanHPC image is created by running a single script, `make_pelican`. This can easily be done by any user with moderate knowledge of the Linux operating system. The script has only a few dependencies (debootstrap, rsync, wget, live-build) that can easily be installed on any version of Linux. For example, it is possible to run the script on a running PelicanHPC system, as all dependencies are already installed. To make a custom version with different packages, for example, one can edit the script to add the needed Debian package names, and then run it. This will result in a customized image file which can be used in the same way as the released PelicanHPC images. Thus, PelicanHPC is flexible and can easily be customized to met the needs of researchers in different fields. Two very good examples of specialized distributions that were built using modified versions of `make_pelican` are birgHPC[6] (Chew *et al.*, 2011) and MOLA[7] (Abreu *et al.*, 2010). The `make_pelican` script provides a fairly easy to use and robust means of generating a tailor-made clustering distribution with any specialized software that may be needed. Because of this scope for customizability, PelicanHPC itself will remain simple, providing clustering and a single implementation of MPI (Open MPI is the installed implementation), with few "bells and whistles". The simplicity of the basic setup is intended to make it easy to understand and modify, and easy to maintain.

3. Use

The frontend node first must be booted, using a CD or USB storage device, or directly from the ISO image if the frontend is a virtual machine. As it boots, the user is prompted to input some information, such as the desired password, or the network device to use for the cluster, if more than one is available. Upon completion of booting, the machine is at the Linux console login prompt. One logs in with the username "user" and whatever password was supplied. One may then enter the graphical desktop environment by typing "`startx`" or continue in the console. To set up the cluster, one enters "`pelican_setup`". This script configures services such as dhcp, tftp, and nfs. When this is done, a prompt to turn on the compute nodes appears. Then the frontend cycles, checking and reporting how many compute nodes have booted. When all are booted, the user indicates this is so, and a basic Open MPI bhosts file is written. This file can easily be modified to achieve better load balancing on a heterogeneous cluster. Next, a simple Fortran MPI program that computes the total number of flops that are available is compiled and run, assuming one slot per node. This achieves the simple goal of testing that the cluster is functional. After this, the console prompt appears and setup is done. From booting the frontend to completing the process takes about five minutes.

It is very helpful if the frontend node has two network devices, one for the cluster, and one for internet access. If the frontend is a virtual machine, one can create two virtual net devices, one that uses NAT, which allows ssh to and from the virtual frontend, and one which uses bridged networking, in order to connect to the cluster's network. It is very important that the network interface used for the cluster be isolated from any public network, or the dhcp server that runs on the PelicanHPC frontend node can interfere with the public network.

PelicanHPC inherits its features from the Debian Live system. This system includes the aufs kernel module, which is a layered file system that allows modifications to a file system that originates from a read-only medium such as a CD. Because of this, it is possible to install software in the normal way to the running PelicanHPC frontend node.

Example 1. To install the emacs editor on the frontend, execute

```
sudo apt-get install emacs
```

[6] birgHPC: http://birg1.fbb.utm.my/birghpc/
[7] MOLA: http://www.esa.ipb.pt/~ruiabreu/mola/

to install an editor that is not provided on the released images.

The same applies to the compute nodes, which share the frontend's internet connection using IP masquerading: they may be accessed from the frontend by ssh, and software may also be installed to each compute node. The package dsh is installed, which simplifies running a set of commands on each of a set of hosts.

Example 2. To install the GNU scientific library on all nodes, one can execute

```
dsh -f /home/user/tmp/bhosts sudo apt-get -y install libgsl0-dev
```

on the frontend node.

This provides a simple solution for adding a small number of packages on the fly. Packages added in this way will not be available after a reboot, however, because aufs overlays live in RAM. For newly installed packages to survive a reboot, there are two options. This first, which is recommended for users who intend to use PelicanHPC on a long term basis, is to prepare a customized image, as is described in the next section. The second, which may be more suitable for a project which requires adding a limited number of packages, is to use permanent storage for `/home`, and to install new software to a destination in `/home`. As `/home` is NFS shared to all nodes, a single installation there on the frontend provides the new software to the entire cluster. The configuration switch `--with-prefix=/home` will achieve this goal for much software compiled from source.

To use permanent storage, there are two methods. The first is to enter the device name (e.g., `/dev/sda4`) of an ext2, ext3 or ext4 formatted partition at boot time, instead of accepting the default of using tmpfs (in RAM) for `/home`. If this is done, then the specified partition is mounted at `/home`, and all changes under `/home` will survive a reboot. Software installed there will be available permanently. A more sophisticated option first appeared in PelicanHPC v2.2, thanks to Robert Petry. At this time, the `pelican_config` mechanism was added. Upon boot, PelicanHPC scans for a partition labelled `PELHOME`. If this volume name is found, the user will be prompted as to whether or not the partition should be mounted at `/home`. Next, as is usual at boot time, the user will be asked if the example software should be copied over to `/home`. If this is done, the `pelican_config` file, among others, will be copied to the permanent storage. The contents of `pelican_config` can be edited to modify the behavior of PelicanHPC at boot and setup time. It is possible to configure automount of the `PELHOME` partition, automatic setup, addition of a list of usernames, configuration of a static IP address, use of a network different than the default 10.11.12.x, use of a firewall, whether or not prompts should wait for user intervention, and a number of other features. The file is self-documented, and serious users should read it carefully, as it provides features which can make the cluster much more flexible and powerful when used on a non-transitory basis.

Example 3. The means of assigning a label depends on the type of filesystem of the partition. To assign the volume label `PELHOME` to the partition `/dev/sda6` which is formatted as ext2/ext3/ext4, one can execute

```
sudo /sbin/e2label /dev/sda6 "PELHOME"
```

while running PelicanHPC. At the next reboot, the partition will be found.

After booting and setting up the cluster using `pelican_setup`, the cluster is ready for use. The Open MPI implementation of MPI is installed on the released images, and the bhosts file created at setup time is located at `/home/user/tmp/bhosts`. This file simply lists the machines in the cluster by IP address, with no attempt at load balancing. Users should edit the file as appropriate for the hardware of the cluster. This is important to obtain good performance if the hosts are heterogeneous, for example, with different numbers of cores. If compute nodes are added or removed after initial setup, one can run `pelican_restarthpc` to re-configure the basic MPI environment, including re-generating the bhosts file.

Example 4. To obtain a list of the hosts in the cluster, one can run

```
mpirun --hostfile ~/tmp/bhosts hostname
```

4. Customization

PelicanHPC releases contain the Open MPI implementation of MPI, as well as a reasonably complete set of tools and libraries for compiling C, C++ and Fortan applications. GNU/Octave with MPI extensions and a fairly extensive set of examples related to the author's research and teaching are also provided, but these examples will be of interest to a limited set of users. Python with mpi4py is also provided, and examples are found in `/home/user/mpi4py-$VERSION/demo/`. The Octave and Python examples may be of some interest to some users, but most users with specific needs will require software that is not installed on the released images. As was noted above, PelicanHPC images are made by running a single script, `make_pelican`. This script has a small set of dependencies: wget, rsync, deboostrap, and live-build version 2.x (as of this writing - future versions of will use newer versions of live-build). All of these packages can easily be installed on any Linux system, and they are installed on PelicanHPC releases. To build an image, one just runs the script (as root, or using sudo). An internet connection is required, to download packages from Debian mirrors. Also, a fair amount of disk space (several GB is sufficient) is needed, to accomodate the downloaded packages and to build the image. To run the script on a running PelicanHPC system, one most likely needs to be using permanent storage, as discussed above, because most systems will not have enough RAM to build an image in memory.

Example 5. To run the current release at this writing, do

```
sudo sh ./make_pelican-v2.6
```

on a running PelicanHPC system

The `make_pelican` script is mostly self-documented, and it is also discussed in the PelicanHPC Tutorial (see the following section on Documentation). In the script, one can specify the architecture (amd64 or i386), the Debian release (stable is the default), Debian mirror to use, and the list of Debian packages to include on the image. All of these options are at the top of the script, and are easy to locate.

The script looks for a directory `./pelicanhome` in the build directory (the location of `make_pelican`). All of the contents of this directory are copied to the image, and will be in `/home/user` after the image that is built is booted. Furthermore, if this directory contains a script `make_pelicanhome.sh`, then this script will be run during the build process to perform any needed actions such as downloading source code, compilation of source code, etc. This provides a means of including software and data that is not part of Debian on a PelicanHPC image. The script is executed in the chroot environment that contains the filesystem that will appear on the image, at the last stage before the image is made. Thus, all libraries and packages that were included from Debian packages will be available for compilation of special software that is not in Debian. The software can also be installed to any location such as `/usr`, `/opt` or `/usr/local`, it does not have to be installed under `/home`. Octave, the Octave examples, and mpi4py are included on the released images using this mechanism, and the `pelicanhome.tar.bz2` archive that is available on the PelicanHPC web page provides an example of how the mechanism may be used.

Example 6. The following lines of code, contributed by Stanislav Godiška of Slovak Technical University of Technology, are in the `make_pelicanhome.sh` script that is used by default. If these lines are uncommented, then a NIC module that is not part of Debian will be available. A similar strategy could be used to add other kernel modules.

```
# Linux headers (for Atheros driver installation)
# uncomment below this line
# cd /etc/skel
# wget http://launchpadlibrarian.net/86474951/linux-headers-2.6.32-37_2.6.32-37.81_all.deb
# dpkg -i linux-headers-2.6.32-37_2.6.32-37.81_all.deb
# wget http://launchpadlibrarian.net/86453234/linux-headers-2.6.32-37-generic_2.6.32-37.81_an
# dpkg -i linux-headers-2.6.32-37-generic_2.6.32-37.81_amd64.deb
# # Atheros 81 Network family driver
# cd /etc/skel
# mkdir AR81Family
# wget http://k003.kiwi6.com/hotlink/5h69ss0y70/ar81family_linux_v1_0_1_14_tar.gz
```

```
# mv ar81family_linux_v1_0_1_14_tar.gz AR81Family
# cd AR81Family
# tar zxvf ar81family_linux_v1_0_1_14_tar.gz
# make install
# modprobe atl1e
```

5. Limitations and Cautions

Some caution is needed when setting up a PelicanHPC cluster, especially if one has little experience with networks. The PelicanHPC frontend node begins to act as a netboot server after `pelican_setup` has been run. This means that it will provide an IP address to any client that solicits one. If the network of the cluster is not isolated from larger networks, the PelicanHPC dhcp service may interfere with the dhcp service running on the larger network, causing connectivity problems for the users of the larger network.

A second caution is that attempts to use permanent storage should be made only after one understands what is the Linux device name given to hard disk or USB storage partitions. In its default configuration, PelicanHPC does not use permanent storage, and there is no risk of data loss. If one uses permanent storage, it becomes possible to erase or compromise data on the storage devices. Users with little experience with the Linux operating system should be especially careful. It is advisable to ensure that important data has been properly backed up.

A final caution concerns security and privacy. The user of a PelicanHPC cluster can easily access all data on all of the machines in the cluster. For example, an instructor of a class should be cautious about letting students use machines that contain sensitive data to make a cluster, as the students will be able to access the data while running PelicanHPC. With regard to external threats, there is little risk of unauthorized access from external networks as long as a strong password is used. PelicanHPC uses a firewall and software that blocks IPs from which too many failed logon attempts originate.

6. Documentation and Support

The main documentation for PelicanHPC is the PelicanHPC Tutorial[8] (Creel, 2008) which is included on released images and is also available as a web page. The Tutorial explains how to set up and use a cluster, as well as how to make customized images. Documentation is also provided by README files in appropriate places on the image, as well as by comments in scripts. For example, `make_pelican` and `pelican_config` are extensively commented. Support is available primarily through the user forum[9]. The ParallelKnoppix and PelicanHPC projects have been in continuous existence since 2004, and will continue to exist in a similar form for the foreseable future, so the level of support that currently exists will continue to be available.

7. Performance and Examples

A PelicanHPC cluster is a normal Beowulf-style cluster for MPI after all setup has been done. It is possible to use permanent storage, swap space, and scratch partitions. Performance of a PelicanHPC cluster depends upon the hardware that is used, and upon proper tuning of the MPI environment, especially if the machines that make up the cluster are heterogeneous. Tuning of the MPI environment, for example by editing the bhosts file used by Open MPI, is not an issue specific to PelicanHPC, and for this reason it is not discussed here. One can obtain nearly linear speedups for appropriate problems.

Example 7. PelicanHPC contains a number of example of GNU Octave scripts that perform various tasks in econometrics and statistics, including multivariate nonparametric kernel regression. To run the example code for kernel regression on a single core, one enters Octave, and executes

[8] PelicanHPC Tutorial: http://pareto.uab.es/mcreel/PelicanHPC/Tutorial/PelicanTutorial.html
[9] User forum: http://pelicanhpc.788819.n4.nabble.com/

```
kernel_example(5000, false, false)
```

To run on all 8 cores of a dual quad core machine, one executes (from the terminal, not from the Octave prompt)

```
mpirun -np 8 octave -q --eval "kernel_example(5000, true, false)"
```

To run on all the 24 cores of 3 dual quad core machines in a cluster, one executes

```
mpirun -np 24 -h ~/tmp/bhosts octave -q --eval "kernel_example(5000, true, false)"
```

Timings for these three commands are 10.41 seconds, 1.77 seconds, and 0.53 seconds, respectively, which shows good scaling performance for this problem. To learn what the example does and how kernel regression is implemented, one can enter Octave and then enter "help kernel_example" or "help kernel_regression".

A number of similar results for a variety of problems are reported by Creel (2005) using ParallelKnoppix, which operated fundamentally in the same way as does PelicanHPC. Abreu et al. (2010) provide performance measurements (see especially Figure 2) in the context of virtual screening of small molecules, using MOLA, a customized and enhanced version of PelicanHPC. Chew et al. (2011) find that the performance of a PelicanHPC cluster can be as good as that of a traditionally installed cluster that uses the same hardware (see their Table 1).

One interesting possibility that arises when building a Live CD/USB image is that of including complicated software environments that are difficult or time-consuming to reproduce, especially by non-specialists who may be users of different operating systems. A researcher can include all of the software needed to replicate results on a PelicanHPC image, and this can allow other researchers to replicate and extend results immediately, without the need to install and configure a long list of software dependencies. It may also be a very useful tool for demonstrations at seminars or conferences. To give an example of this possibility, and to give some more evidence of the performance of a PelicanHPC cluster, we can consider the file `runme.sh`[10] that is provided on PelicanHPC images. This file allows replication of the results found in Table 7 in Creel and Kristensen (2011). To obtain these results, one needs to have installed GNU Octave[11], Open MPI[12], the MPI extensions for Octave[13], the Dynare package for solving dynamic stochastic nonlinear models[14], and the ANN[15] library for finding nearest neighbors to points, as well as additional code provided by the author. Given this list of items, it would be difficult to replicate the results if everything were to be installed from scratch. However, given a recent copy of PelicanHPC, it is very easy to replicate the results, as one just executes the `runme.sh` script. This means of sharing code is free and open, but it is also practical and user friendly, and it can lead to more rapid and effective dissemination of scientific results. On a heterogeneous cluster made of 3 dual quad core machines (24 cores), the script runs in 48 seconds, while on a single core it takes 570 seconds. This application involves a fair amount of file I/O during the run, and a substantial portion that is not parallelized, thus the less than linear scaling is to be expected.

8. Conclusion

This paper has briefly introduced PelicanHPC. The most obvious feature of this clustering solution is the rapidity with which it can make available a cluster. Less obvious features include the customizability of the scheme, which allows it to be adapted to meet the needs of a large variety of users, and the possibilities for using PelicanHPC to create a cluster that may be remotely administered and used on a long-term basis. This paper has attempted to point out these less obvious features, to provide examples, and to give sources of additional information.

[10] The full path of the file on PelicanHPC is `/home/user/Econometrics/MyOctaveFiles/Econometrics/IL/DSGE_PO/runme.sh`
[11] GNU Octave: http://www.gnu.org/software/octave/
[12] Open MPI: http://www.open-mpi.org/
[13] Open MPI extensions for Octave: http://octave.sourceforge.net/openmpi_ext/index.html
[14] Dynare: http://www.dynare.org/
[15] ANN: http://www.cs.umd.edu/~mount/ANN/

[1] Abreu, R., Froufe, H., Queiroz, M., Ferreira, I. (2010) "MOLA: a bootable, self-configuring system for virtual screening using AutoDock4/Vina on computer clusters", *Journal of Cheminformatics,* 2010, 2:10.

[2] Teong Han Chew, Kwee Hong Joyce-Tan, Farizuwana Akma and Mohd Shahir Shamsir (2011), "birgHPC: creating instant computing clusters for bioinformatics and molecular dynamics", *Bioinformatics,* 27, 1320-1321.

[3] Creel, M. (2005) "User-friendly parallel computations with econometric examples", *Computational Economics*, 26, 107-128.

[4] Creel, M. (2008) "PelicanHPC Tutorial", UFAE and IAE Working Papers, 74908, http://econpapers.repec.org/RePEc:aub:autbar:749.08.

[5] Creel, M. and D. Kristensen (2011) "Indirect likelihood inference", Barcelona GSE Working Paper Series, 558, http://research.barcelonagse.eu/tmp/working_papers/558.pdf.

[6] Dietlmeier, J, S. Begley and P. Whelan (2008) "Cost-effective HPC clustering for computer vision applications", IMVIP '08 Proceedings of the 2008 International Machine Vision and Image Processing Conference, 97-102.

[7] Grytsenko, T. and Peratta (2007) "Implementation and performance assessment of a parallel solver for sparse linear systems of equations and rules for optimal solution" in *Data Mining VIII: Data, Text and Web Mining and their Business Applications,* A. Zanasi and C. Brebbia, eds., WIT Transactions on Information and Communication Technologies, WIT Press.

[8] San Sebastian, I, J. Aldazabal, C. Capdevila, C. Garcia-Mateo (2008) "Diffusion simulation of Cr–Fe bcc systems at atomic level using a random walk algorithm", *physica status solidi (a),* 205, 1337–1342.