Crossroads
The ACM Student Magazine

Building a Nocturnal Cluster With Nonstandard Hardware: Linux on the iMac

by ***Eric J. Shamow***

Introduction

As parallel processing gains prominence in the computing world, it has expanded beyond the domain of well-funded research institutions and into the arena of university computer science departments and smaller organizations with intensive computing requirements. As this transition takes place, however, many smaller institutions have found themselves unable to afford the commercial clustering solutions offered by large companies. Often these universities and institutions can barely afford to maintain their modernized computing facilities, let alone purchase expensive dedicated hardware and software solely for the purpose of large-scale processing.

A middle ground involves the development of a part-time cluster. Many computing facilities are only used for a portion of the day; for the remainder, they sit idle, wasting valuable resources. This concept was the rationale for various shared-computing projects such as SETI@home and Distributed.Net, which utilize idle processor cycles to

perform intensive computation. At Queens College, we took a different approach: during the day, the computer should be devoted entirely to its primary function. In the evening, after the facilites are closed to the public, the systems switch into clustering mode, and devote their full resources to a shared-computing environment.

As with the design of any cluster, there are several decisions to be made regarding physical implementation that have an impact upon overall price and performance. Striking an appropriate balance between these two can be a challenging task, and there is much room for overcompensation in favor of one aspect at the expense of the other. This article will attempt to cover some of the implementation designs and dilemmas involved in building such a nocturnal cluster. At its centerpiece are two key notions: first, that the construction of the cluster must be low in cost, and second, that the cluster and all applications designed for it must be adaptable to larger cluster implementations should the cluster be expanded or replaced with more conventional clustering hardware.

In every such *ad hoc* implementation there are bound to be countless oddities and stumbling blocks to be overcome; this article will cover some of the unusual aspects involved in the ongoing design of the first Queens College cluster, how we adapted to them, and how we plan to further adapt the cluster over time.

Clustering the iMac

When designing a traditional cluster, one usually has the luxury of choosing a clustering platform to best suit the cluster's user base. Common clusters are already available from commercial vendors and involve a variety of different configurations, ranging from Cray multiprocessor machines, to IBM mainframes hosting a number of virtual Linux machines, to simple interconnected x86 PCs.

In the case of a nocturnal cluster, however, one is restricted by the configurations of hardware and software available. In the case of Queens College's first nocturnal cluster, the only available computer lab was a group of Apple iMac computers, each of which was equipped with a 500-MHz G3 processor and 256 MB of SDRAM. Each iMac was preloaded with Mac OS 9, and needed to continue to run that operating system during the day.

The decision was made early on that Mac OS 9 was an insufficient clustering environment for the nocturnal cluster. There were too few programming tools available, the operating system's performance was too inefficient, and the available

clustering software was expensive and not nearly as robust as that available for other operating systems. A logical next step was to consider an upgrade to Mac OS X. Apple's new operating system provided a complete UNIX-based operating environment, and was compatible with the standard GNU compiler utilities, making code portability easier to achieve. In addition, UCLA's Project AppleSeed very nearly provided plug-and-play clustering, which would have reduced setup to a minimal procedure. Most of the OS 9 applications currently running on these systems could easily be migrated to OS X, and OS X's multiuser features could be utilized to restrict access to the clustering software and data, allowing the lab's daytime users to be completely unaware of the systems' nocturnal activity, and preventing them from interfering with clustering operation.

OS X was abandoned as a clustering platform for several reasons. First, the cost of purchasing licenses for each node in the cluster was prohibitive, and that cost would continue to grow as the cluster expanded. In addition, in terms of code portability, OS X did not lend itself to cluster expansion; future generations of the cluster would be restricted to the PowerPC platform and to MacOS in particular. On both counts, OS X failed our above-stated requirements of cost-effectiveness and future expandability.

A Golden Client

There are two readily-available operating systems that meet our above requirements and will run on the PowerPC platform: Linux and the free variants of BSD. Both are robust and well-developed solutions, and there are multiple implementations of each. Due to the author's familiarity with Linux, that operating system was chosen in favor of BSD; in particular, the Debian distribution was chosen, as the x86 version of Debian is a particular favorite among programmers, and the transition was felt to be easier for them if they used a distribution with which they were familiar. BSD or another distribution (such as YellowDog Linux) would be equally valid choices for a future implementation of the iMac cluster.

Debian Linux met both of our requirements: it was free, and therefore cost was not a concern, and it was standardized and portable, which meant that expanding the cluster or moving it to another platform would not be a significant issue. Another major factor in favor of Debian Linux over another Linux distribution was its availability over a range of different platforms. Were the clustering platform to be changed to Alpha, ARM, x86, 680x0, or SPARC, we could continue to operate with the same distribution.

With the operating system decided upon, the next objective was to create a so-called golden client - a prototype client machine that we could easily duplicate for each desired node. The golden client was to contain separate installations of MacOS 9 and the latest edition of Debian Linux (Debian Woody), such that the Linux installation was to remain invisible to the OS 9 installation.

Since OS 9 requires a complicated series of partitions to store firmware and drivers, it was installed first. The iMac was booted from the OS 9 installer CD, and the installer's Drive Setup utility was used to partition the disk into one four-gigabyte HFS partition (for OS 9) and one empty partition (which would ultimately be partitioned for Linux). OS 9 was then installed into the HFS partition.

Partitioning

With OS 9 installed, the next step was to install Linux. Linux installation on an iMac can be divided into four stages: partitioning, initial boot, software installation, and configuration. While the OS 9 Drive Setup utility prepared the HFS partition for itself, it could not create the multiple partition types needed for a Linux installation. To create these partitions, we used the "pdisk" utility, available from **http://www.cfcl.com/eryk/linux/pdisk/**.

At the pdisk prompt "`Command (? for help):`" we entered "e" to enter disk editing mode, then specified "/dev/hde" to indicate the iMac's primary hard drive. pdisk's method of assigning disk names seems to differ slightly from Linux's, despite sharing a similar naming convention; in this case, the primary IDE drive, which would be known as /dev/hda in Linux, is /dev/hde in pdisk.

Once in editing mode, we entered "p" to list the complete partition table. This will produce a list similar to the following (adapted from Debian's **"Using pdisk"** guide):

```
Partition map (with 512 byte blocks) on '/dev/hde'
 #:                type name                 length   base     ( size )
 1: Apple_partition_map Apple                   63 @ 1
 2:      Apple_Driver43*Macintosh               54 @ 64
 3:      Apple_Driver43*Macintosh               74 @ 118
 4:    Apple_Driver_ATA*Macintosh               54 @ 192
 5:    Apple_Driver_ATA*Macintosh               74 @ 246
 6:  Apple_Driver_IOKit Macintosh              512 @ 320
```

```
 7:       Apple_Patches Patch Partition      512 @ 832
 8:           Apple_HFS MacOS           13304256 @ 1344     (  6.3G)
 9:          Apple_Free Empty           13302656 @ 1344     (  6.3G)
```

In this case, we see that the empty partition set aside for Linux is /dev/hde9. Once we determined the correct partition number, the task remained to repartition the empty space into something that Linux could use.

At a bare minimum, most Linux installations require two partitions: one of type Linux swap to be used as a swap partition (this partition is typically sized at twice the amount of physical RAM in the system) and a second, larger partition to hold the system software. In the usual installation this second partition is broken into a series of smaller partitions to separate the crucial system binaries from third-party applications and user data. In the case of a cluster node, however, which would not need to host users or third-party data, this additional partitioning is not necessary. In addition to the standard Linux partitions, we created a small partition to hold the yaboot boot loader. The decision was made, based on an analysis of the documentation and experience with Linux administration, that three partitions would be created: an 800K boot partition, a 256 MB swap partition, and a root partition encompassing the remaining free space.

In pdisk, we use the "C" command to create a partition and specify its type. To create the swap space inside the /dev/hde9 partition, we issued the following command:

```
Command (? for help): C 9p 800k boot Apple_Bootstrap
```

This command has the effect of creating an 800K partition of type "Apple_Bootstrap" and labeled "boot" in the place of the old /dev/hde9. A new /dev/hde10 partition will now appear as the "Empty" partition at the end of the partition table, so that a call to "p" will now yield something similar to the following two lines at the end of the display:

```
 9:     Apple_Bootstrap boot               800 @ 1344
10:          Apple_Free Empty           13302656 @ 1344     (  6.3G)
```

With the boot partition created, it was now necessary to create the swap partition. pdisk can sometimes determine partition type based on partition name. In the case of the swap partition, pdisk recognizes all partitions named "swap" as being Linux swap partitions. To allow pdisk to automatically determine the partition type we used the

lower-case "c" command:

```
Command (? for help): c 10p 256m swap
```

This command has the effect of creating a 256 MB partition of type "Apple_UNIX_SVR2" in place of partition /dev/hde10, and bumping the free space to /dev/hde11. In our idealized example, we will now have the following three lines at the end of the partition map:

```
 9:     Apple_Bootstrap boot                1600 @ 1344
10:     Apple_UNIX_SVR2 swap              262144 @ 1344
11:          Apple_Free Empty           13302656 @ 1344     (   6.2G)
```

Finally, we need to convert /dev/hde11 into a Linux root partition. The following command will accomplish the conversion:

```
Command (? for help): C 11p 11p /
```

After the final revision, our idealized partition map ends with the following three lines:

```
 9:     Apple_Bootstrap boot                1600 @ 1344
10:     Apple_UNIX_SVR2 swap              262144 @ 1344
11:     Apple_UNIX_SVR2 /               13302656 @ 1344    (   6.2G)
```

Once the table is set, it is written to disk using the "w" command. After exiting pdisk, the iMac must be rebooted to ensure that the partition table has been correctly written.

Initial Boot

Debian has a very robust internet-based installation system which allows an administrator to install Linux with little more than a few boot floppies, obviating the need for the installation CD. In the case of our iMacs, this posed another problem: the iMac comes equipped with a USB floppy drive, but the system cannot boot from it. However, as a member of a generation of Macintoshes known as "New World" Macs, the iMac possesses an "OpenFirmware" interface which allows a user to boot from a boot image resident on disk.

Debian provides a series of boot images designed for use with OpenFirmware, available at **http://http.us.debian.org/debian/dists/woody/main/disks-powerpc/**

current/new-powermac/. There are four files required: linux (the PowerPC Linux kernel image), yaboot (a dual-booting program for Linux), yaboot.conf (the configuration file for the yaboot booter), and root.bin (the installer program). These four files were downloaded and copied into the root directory of the OS 9 installation and the system was booted into the OpenFirmware interface. This is accomplished in a variety of ways depending on the system being used; on the New World iMac, OpenFirmware can be launched by holding down the Option, Command, o and f buttons while the system is booting.

To launch the Debian installer from the OpenFirmware interface, the only information necessary is the partition number of the OS 9 HFS partition, which can be obtained using the pdisk utility. In the case of our OS 9 installation, the HFS partition was partition nine of the disk. To boot the yaboot bootloader from partition nine with the OpenFirmware "0 >" prompt, the syntax is:

```
0> boot hd:9,yaboot
```

Once the yaboot bootloader loads, the "boot:" prompt appears. This bootloader gives the user the option to load one of several available kernel images. For the purposes of installation, we entered "install" here; after installation there is where we would be able to specify options to be passed to the Linux kernel at boot.

Software Installation and Configuration

The Debian installer program is known as "dbootstrap." Installing the Linux system software using dbootstrap was a fairly uncomplicated procedure; in the case of our golden client, as we were operating from boot images and not from an installation CD, we chose to use dbootstrap's option for installation over the internet. What this means is that, after the user completes the selection of software packages from dbootstrap's menu system, the software is downloaded directly from Debian or one of Debian's mirror servers and then installed automatically.

There are several important considerations involved in choosing software packages for cluster nodes. These considerations differ somewhat between the nodes themselves and the "master" node, on which compilation and job allocation will take place. All of the nodes need the software execution shared libraries necessary to execute not only their own system binaries, but also any potential distributed applications. They also

need a shell and a rudimentary set of system management tools. However, too many installations will create unnecessary bulk, make the process of imaging and updating the nodes slow and tedious, and may introduce unnecessary security holes. Therefore, software installation must be approached with care, and with an eye toward the ultimate goal of the cluster's end users. Otherwise, however, software installation does not differ from the procedure on an x86-based Linux installation.

Once installed, the selected software must be configured. In most cases this is a straightforward procedure and is documented clearly within the downloaded packages. Security is paramount here: every service which is not necessary for the operation of the cluster must be deactivated to protect system integrity. In terms of remote-access applications, typical cluster nodes need to be accessible via telnet and ftp (or ssh and sftp, if greater security is needed). Rsh is also typically required by clustering applications. Most other services can be deactivated.

The Future - Structuring the Cluster and Other Challenges

As of the date of this writing, the Queens College cluster project is poised to move on to replication of the golden client and implementation of the cluster. All that remains in completing the golden cluster design is to produce system scripts that will reboot the system from OS 9 to Linux and back again at the appropriate time of day. However, there are additional considerations involved in turning a single iMac running Debian Linux into a fully-functioning cluster.

Physically, the structure and connectivity of the cluster must be determined. There are multiple physical configurations of clusters which are idealized for different clustering applications, and different networking solutions designed to implement those configurations. Given the cost and manageability issues involved in implementing those configurations, it is likely that we will implement the most simple solution: 10/100 networking cards, which have the advantage of being integrated into the iMac, networked through a switch. This solution, while relatively slow, is one of the most adaptable to a wide variety of clustering applications and can be upgraded at a later date.

With the physical connectivity issues resolved, the golden client must be replicated to all of the client nodes. We are investigating various image-management solutions, and as of this writing have preliminarily settled upon SystemImager (available from **http://systemimager.org**) to manage client image update and distribution. Additionally,

with the client nodes in place, a master node must be designated. The master node tends to fall into one of two categories: the traffic cop (where the master node merely directs the distribution of tasks among the clients) and the master controller (where the master node serves as a file server for the clients, and also serves as a development platform for end users). Because of the scarcity of PowerPC Linux systems, we have settled upon the latter role for the master node. As a result, the master node must be constructed with considerably more care than the remainder of the cluster, as it will be open and available to the outside world and must handle multiple users. Ideally, this master node will boast more advanced hardware than the client nodes in the system.

Once the cluster is in place, it will remain to design software to take advantage of its structure. Various software applications take advantage of different parallel processing schedulers, so it is likely that the cluster will often be broken into several smaller mini-clusters. Once the supported schedulers are designated, a batch processor must also be implemented to schedule jobs according to system availability, and to repartition the cluster if necessary.

Conclusion

While the Queens College cluster project is still in its infancy, we have made considerable strides toward making the cluster a reality. In particular, given the obstacles of a nonstandard hardware platform and limited system time, we feel we have designed a system which is capable of providing some clustering functionality at no cost over that of the existing hardware and software. As an additional benefit, supporting Linux on the PowerPC platform allows us to expand the scope of supported operating systems on campus.

References

1

DiCarlo, A., Grobman, I., Perens, B., Rudolph, S., & Treacy, J. "Installing Debian GNU/Linux 2.2 For PowerPC," October 14, 2001, <**http://www.debian.org/releases/stable/powerpc/install.en.html**> (April 14, 2002).

2

using-pdisk.txt, <**http://prdownloads.sourceforge.net/debian-imac/debian-imac.sit**>, April 14, 2002.

Biography

Eric J. Shamow (**eric@qc.edu**)is a bachelor's student at Queens College of the City University of New York. Mr. Shamow is also Assistant System Manager for UNIX systems at Queens College's Office of Information Technology. He has done research with Dr. Christopher Vickery into persistant Java Virtual Machines, and together they designed the PJVM project, which remains under development. He is currently doing research in the areas of distributed and parallel computing and embedded processors.