Crossroads
The ACM Student Magazine

Building an MPI Cluster

by ***Donald C. Bergen II*** and ***Boise P. Miller***

Introduction

Clustering provides for a cost-effective and easy to maintain method of supercomputing. Recently, more and more of the world's powerful supercomputers have been constructed from clusters (a group of computers working collectively on the same problem) of what most people would call ordinary machines. Surprisingly, with a little UNIX knowledge and some superfluous personal computers, almost anyone can build a cluster for parallel computing. As you will read, there can be some roadblocks, but these are easily overcome. The question then becomes, why would one want to and how does one go about building a cluster?

Motivation: A Largely Simple or a Simply Large Problem?

Recently we began an independent study in parallel programming. The study involved building a cluster and then writing several different programs to learn how to use the Message Passing Interface (MPI) to create parallel programs (programs that divide processing responsibilities amongst the machines in the cluster) for the cluster.

Clusters can be used to solve very large and very complex problems. They are used for everything from recordkeeping at large organizations to computational fluid dynamics, used in the design of modern aircraft, ships, and motor vehicles. At Lock Haven University, a cluster is being used to determine if a very large number, 1989!+1, is prime or composite.

Several years ago, a professor in the mathematics department at Lock Haven University gave an exam to his Number Theory class. The problem asked the student to find 1988 consecutive composite numbers. The problem was supposed to test the

students' knowledge of a theorem that states that there are N consecutive composite numbers from (N+1)!+2 through (N+1)!+N+1 . A student responded to the question with "the numbers from (1988+1)!+1 through (1988+1)!+1989." While this answer was not guaranteed by the theorem, the instructor had no way of proving that the answer was incorrect. To prove the answer incorrect, the professor would have to determine if 1989!+1 is prime or composite. The problem could have been solved easily on a powerful supercomputer; however with Lock Haven University being a small university, the instructor did not have one at his disposal. How could he determine if the answer was correct? Attempting to solve this is the major test case for our study. (Note that at the time of writing, the answer is still unknown.)

The Hardware

For our project we had eight identical Gateway PCs at our disposal. Each PC had:

- Intel Pentium III 550 MHz processor
- 256 MB of RAM
- 3Com 3C905C-TX 100Mbps Ethernet Card

This is the configuration we used; however, your configuration may vary. A cluster can be built using the same procedure we will outline below from nearly any modern computer capable of running the software specified below and supporting Transmission Control Protocol / Internet Protocol (TCP/IP) communications.

The Software

The operating system used for this project was Redhat Linux 7.2, however, almost any version of Linux and most commercial UNIX's will provide the same results. The middleware used to provide communications between processes was the Local Area Multi-computer (LAM) implementation of MPI (**http://www.lam-mpi.org**). LAM is included with the 7.2 distribution of Redhat Linux, however to gain full functionality you must download the latest version from the LAM website.

While there are several implementations of MPI available, we chose LAM because it is open source, has very good documentation, and supports most of the MPI-2 standard, which adds a great deal of functionality as well as provides bindings necessary for programming in C++. An alternative to LAM would be MPICH, another major free implementation of MPI

Making It All Work

Once the hardware was organized, we needed to connect it in a way that was easy to manage. This involved a wire-rack unit to hold the machines that were all connected to a Keyboard-Video-Mouse (KVM) switch. All eight machines were also connected to a 3Com SuperStack 3 100Mbps Ethernet switch. This however, is dependant on personal preference and availability of devices. We do, however, recommend that not less than a 100Mbps communication device be used due to the amount of inter-machine communications that takes place in a parallel computing environment.

In our first attempt to install Linux on the cluster machines, we installed only those packages we believed we would need to make MPI work. Our professor provided us with a copy of MPI/Pro for Linux, in the form of a single Redhat Package Manager (RPM) file, which we installed. We could find no documentation on how to configure and use MPI/Pro. While we were attempting to get MPI/Pro to work, we discovered that we needed remote shell (rsh) for MPI/Pro to work. We decided that it would be easier to reinstall Linux with all available packages than to try and figure out exactly what we needed.

After doing a complete install of Redhat Linux 7.2 on each of the machines, we again turned our attention to MPI/Pro. When we attempted to install the MPI/Pro RPM this time, we got a message about conflicting packages. After further research, we discovered that Redhat 7.2 ships with LAM / MPI. We decided to use LAM instead of MPI/Pro because LAM had much more extensive documentation as well as other reasons discussed in the Software Section. You may also find it easier to start with LAM unless you have purchased a full version of MPI/Pro and have documentation and support available to you.

Our next step was to configure rsh. Rsh is a service that allows you to execute commands on a remote machine without entering a user name and password. To do this, you must edit the file "/etc/hosts.equiv" to include the host name or IP address of each machine you wish to allow rsh commands from. Each name/address is listed on a separate line. To use rsh with LAM, each machine, except the root node, needs to have the name/address of the root node in its "hosts.equiv" file. In our case, security was not a huge concern because our cluster was built on a closed network using the private Class A IP network (10.0.0.0). If you are building your cluster on a publicly accessible network, you will likely want to specify usernames of allowed users in your "hosts. equiv" file to prevent unauthorized access to the cluster. For more on how the "hosts.

equiv" file works, run the command `info /etc/hosts.equiv`.

Once rsh was verified to be working, the next step was to configure LAM to use all eight machines. To do so, a file called "/etc/lam/lam-bhost.def" must be created containing the IP addresses of all nodes in the cluster. This file must appear on all nodes in the cluster and contain the same addresses. Additionally, an environmental variable needed to be added to the shell profile of each user using LAM on the root node. Adding the following line to the shell profile facilitated this: (check the documentation for the shell you are using for the location of your profile).

```
export LAMBHOST=/etc/lam/lam-bhost.def
```

It may be useful to note that because of the nature of parallel computing, to maximize efficiency, only one user should be running a job at a time. This is because running multiple jobs at the same time requires dividing the CPU time between the jobs, and the benefits of using a cluster will not be realized. For this reason, we have only created one user (on all nodes, having the same username) to run MPI jobs. We highly recommend this to anyone attempting to create an MPI cluster.

At this point configuration is complete and we are ready to "boot" the cluster. You can verify that the cluster is bootable using the `recon -vd` command. We were lucky enough to have `recon` succeed on the first try. Your results may vary. After verifying that the cluster is bootable, you can boot the cluster using the `lamboot` command. At this point we encountered problems, as will anyone using Redhat 7.2's default install of LAM. This is because Redhat installs the libraries for LAM in the wrong location. The only solution we found was to reinstall LAM. If LAM is pre-installed on your system, we strongly recommend uninstalling, going to the LAM website, downloading the latest version, and installing from source. We found building from source to be very easy, though somewhat time consuming, especially since it must be done on each machine. This process can be expedited by running `make` on one machine, then copying the directory to each machine and running `make install` on each machine. This way you only have to compile the source (the most time consuming part of the process) once. If you had configured a "lam-bhost.def" file before installing from source, you should verify that the file is intact, and correct.

At this point the cluster should be ready to use. You should again verify that the cluster

is bootable using the `recon -vd` command, and then boot the cluster using the `lamboot` command. You can verify that the cluster is running on all nodes with the `lamnodes` command. Once the cluster has been verified as operational, some minor testing should be done. There are several test programs available to facilitate this. We chose to use the canonical ring program (**http://www.lam-mpi.org/tutorials/lam/MPI_C_SAMPLE.c**) which is communication-intensive to test our cluster. To compile this code, it was necessary to use `mpicc` (a compiler wrapper) as opposed to the standard `gcc` compiler. Had the sample been written in C++, we would have used `mpiCC`, or for Fortran, `mpif77`. The test program you choose is entirely up to you, however, the aforementioned program was written with the LAM implementation of MPI in mind.

Once the cluster is running and some code is compiled (either a test program or code to do an actual parallel computing task), it must be sent to the LAM daemon (lamd) to run. To do this, the `mpirun` command is used. In our case, we used the following: `mpirun N -s n0 a.out`, where `mpirun` is the MPI executable, `N` signifies that all nodes should be used, `-s n0` designates that our executable should be copied from node 0 (our current root node), and `a.out` is our compiled program. Your environment may vary in that you do not wish to use all nodes, or your compiled program may not be named `a.out`. Additionally, you may not need to specify a source if you are using a shared file system such as Network File System (NFS) in your cluster (as discussed later).

In the event that things go badly with your MPI program, you can use the command `lamclean` to halt all MPI processes and messages. This restores the cluster to a clean state, and leaves it ready to start a new program. When you are done using the cluster, you use the `lamhalt` command to stop the cluster. It is important to do this before shutting down any of the machines in the cluster.

While we did not use Network Information Service (NIS) and/or NFS for our cluster, it may be desirable to do so. If NIS is used, this eliminates the need to replicate the users across each of the nodes. While this is not terribly important for our small eight node cluster, it would be especially convenient for larger clusters. Using NFS is also convenient, because you can place your compiled program on the NFS share, thereby eliminating the need to specify a source node. All nodes will have access to the

program through the NFS share.

Conclusion

We found that it is possible to build a cluster using several standard PCs, a 100Mbps Ethernet hub, or switch, Linux, and LAM / MPI, with a basic understanding of what a cluster is, how MPI works, and basic UNIX skills. Building a cluster of this type is relatively inexpensive, and offers significantly improved performance for programs written to solve problems that can be divided to run in parallel. The degree of improvement depends heavily on the ability of the problem to be solved in parallel.

Now that you have a functioning cluster, you will no doubt want to start writing programs for it. There are several good books and other references listed in the References section at the end of this article, we highly recommend them. Perhaps the most useful reference is the MPI standard itself that can be freely downloaded and is relatively easy reading. It describes in detail each of the functions associated with MPI, and their usage. You should also consult the documentation associated with the implementation of MPI you are using. This documentation should discuss how to compile and run your parallel programs. It has been our experience that LAM has exceptional documentation in this area.

Anyone wishing to build a cluster using Linux and LAM / MPI would be well advised to develop a basic understanding of Linux, remote shell, and LAM itself. Additionally they should consider the design of the network used, as it relates to the size of the cluster. Significant decreases in performance may be noticed if more than 10 machines are used on a hub. In cases where clusters of more than 10 machines will be developed, a switch should be used. On significantly larger clusters, clusters where extremely high performance is required, or clusters that will be solving very communications intensive problems, faster networks (e.g., Gigabit Ethernet, Myrinet) should be used.

References

1. Foster, I. *Designing and building parallel programs*. Addison-Wesley Publishing Company. 1995.
2. Groupp, W., Ewing L., and Skjellum, A. *Using MPI (Portable parallel programming with the Message-Passing Interface.* The MIT Press. 1996.
3.

LAM/MPI **URL: www.lam-mpi.org**

4

LAM/MPI Mailing List Archives: **URL: www.lam-mpi.org/MailArchives**

5

MPI Software Technology, Inc. MPI/Pro for Linux. **URL: www.mpi-softtech.com**

6

MPI Standard 1.1. **URL: www-unix.mcs.anl.gov/mpi**

7

MPI Standard 2.0. **URL: www-unix.mcs.anl.gov/mpi**

8

Pachero, P.S. *Parallel programming with MPI.* Morgan Kaufmann Publishers, Inc. 1997

9

RedHat Linux **URL: http://www.redhat.com**

10

Snir, M., Otto, S., Huss-Lederman, S., Walker, D. and Dongarra, J. *MPI . The complete reference.* The MIT Press. 1998.

11

Xavier, C., and Iyengar, S.S. *Introduction to parallel algorithms.* John Wiley & Sons, Inc. 1998.

Biographies

Donald C. Bergen II (**dbergen@wyoarea.k12.pa.us.**) graduated from Lock Haven University of Pennsylvania in May of 2002 with a Bachelors Degree in Computer Science. He is currently employed as a Computer Technician with the Wyomissing Area School District in Wyomissing, PA.

Boise P. Miller (**bmiller1@lhup.edu**) is a December 2002 graduate of Lock Haven University of Pennsylvania with a B.S. in Computer Science. He is currently employed as an Administrative Computing Application Developer with Lock Haven University of Pennsylvania.