MBH'99: A Beowulf Cluster Capstone Project

Joel C. Adams W. David Laverell
Department of Computer Science
Calvin College
3201 Burton SE
Grand Rapids, MI 49546
{adams, lave}@calvin.edu

Mark A. Ryken
Ameritech IMS
Network Control Center
25 West Randolph Suite 28D
Chicago, IL 60606
mryken@ameritech.net

Abstract

The high costs of commercial parallel computing hardware and software place them beyond the reach of the Computer Science departments of many colleges and universities. By contrast, a *Beowulf cluster* is a dedicated system of commodity off-the-shelf PCs running free, open-source software and connected by inexpensive ethernet. A Beowulf cluster thus provides an inexpensive way to build a multiprocessor for parallel computing at a fraction of the cost of a commercial machine.

This paper is an overview of our experience building MBH'99, an eight-node hypercube-topology Beowulf cluster of cast-off 486 machines, running Linux and the Message Passing Interface (MPI), and connected with 100Mbps ethernet. Building such a cluster requires the integration and application of concepts from operating systems, networking, parallel computing, and computer architecture, making it an excellent subject for a capstone project. We discuss the decisions we made during the design and implementation of MBH'99, and how we will do things differently next time.

The total cost of MBH'99 was $6,100 in January 1999, all of which went for networking hardware. With recent price drops in 100 Mbps ethernet equipment, a similar system today can be built for roughly $3,000.

1. Introduction

The adage

"Many hands make light work"

captures the spirit behind parallel computing. For many computationally-intensive tasks, the task can be performed more quickly if the work is spread across multiple processors than if it must be done by a single processor. Because the work is done by multiple processors acting in parallel, such computation has come to be known as *parallel* or *high performance computing* (HPC). We will use these two phrases interchangeably.

To educate their students about parallel computing many Computer Science departments offer one or more courses on the topic. Such courses can be (and frequently are) taught in a "hands off" or theoretical fashion. However if students are to experience and appreciate the power of parallel execution, such courses should involve actual parallel hardware with which students can experiment and run their programs.

1.1 Problems of Commercial HPC Systems

Unfortunately, commercial HPC hardware and software are very expensive, costing from hundreds of thousands to millions of dollars. When combined with the fiscal budget-realities of many Computer Science departments, these costs have prevented many students from directly experiencing the benefits of parallel computation.

Support is also an issue with commercial HPC hardware and software. As a result of the high research and development costs and limited demand, the attrition rate among HPC companies is extremely high. As a result, a HPC hardware or software vendor may not exist a year after your purchase, leaving one stranded in terms of support.

1.2 A Solution: Beowulf Clusters

In 1994, Thomas Sterling and Don Becker of NASA's Goddard Space Flight Center found a solution to both of the previously-described problems with commercial HPC systems. They solved the "HPC systems are expensive" problem by connecting sixteen 486-DX4 PCs with Ethernet and installing free software (Linux, MPI, PVM) to make the PCs behave as a multiprocessor. The resulting multiprocessor provided significant speedup at a small fraction of the price of a supercomputer. They named their system **Beowulf** [1]. The terms **Beowulf cluster** and **Beowulf-class multiprocessor** have since been coined to classify the many similar systems that have been built since the original Beowulf [9].

Beowulf clusters also solve the "HPC system support problem." Hardware support becomes a non-issue because every piece of a Beowulf is a commodity off-the-shelf item that – upon failure – can be swapped with another equivalent item. Software support is also a non-issue because all of the Beowulf software (Linux, MPI, PVM, etc.) is *open-source* software. This allows the researchers to "roll their own" software as necessary. Thus, a Beowulf cluster empowers a researcher to provide their own hardware and software support, which some (many?) researchers seem to prefer to being dependent on a commercial HPC vendor [7].

Following our 1997 *Parallel Computing* course, our third author decided to build a Beowulf cluster as his senior project during the 1998-99 academic year. With mentoring and minor help from his professors (the first two authors), his cluster was a success. This paper describes our experiences.

2. DESIGN

Building a Beowulf cluster consists of these steps:

1. Acquiring the PCs that will make up the cluster and assembling them, if necessary;
2. Acquiring networking hardware (a hub or switch, cabling, etc.) and using it to connect the PCs;
3. Installing and configuring Linux on each PC;
4. Installing parallel programming software (e.g., MPI, PVM) on each PC; and
5. Using the resulting system to execute parallel programs.

As Beowulf novices, we decided to begin by studying existing clusters, to see how we should proceed in designing and building ours.

2.1 Existing Clusters

An on-line list of existing Beowulf clusters is given at [9]. Because the clusters listed there were built at different times, they differ in their ages, their CPUs, their memory capacities, and so on. There is much to be learned by studying these other systems, and so we spent a significant amount of time poring over their details.

We learned that there is near-universal agreement that the computational bottleneck of Beowulf clusters is the network by which its PCs communicate. 10Mbps ethernet is generally too slow for all but the most coarse-grained parallelism, but 1000Mbps ethernet and ATM are seen as too expensive, so fast (100Mbps) ethernet has been the fabric of choice (at least until the faster technologies drop in price).

For simplicity and low cost, the vast majority of Beowulf clusters use a *star* topology, with all communication routed through a central switch, as shown in Figure 1:

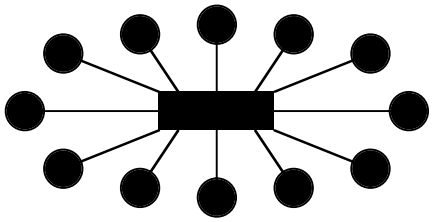

Figure 1
A Star-Topology Cluster

We also learned that if *all* communication is routed through a central hub or switch, then that switch can become the computational bottleneck. The problem is that as parallel communication increases, the switch can become *saturated*, causing delays in message transmissions. The result is an increase in the average **communication latency** (the time to transmit a bit from one machine to another) for the entire system.

Reducing communication latency has led to an interesting variety of network topologies, including rings, trees, and many hybrid structures. For example, the **LoBoS** cluster [3] at the National Institute of Health and **Wulfpack** cluster at Johns Hopkins Medical School [6] each use a ring-star hybrid topology similar to that shown in Figure 2:

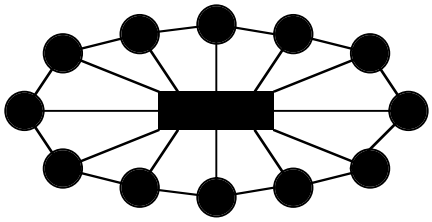

Figure 2
A Ring-Star Hybrid Cluster

Such a topology provides direct links between each node and its two nearest neighbors, and switched (1-hop) links between each node and its non-neighboring nodes. By allowing neighboring nodes to communicate directly, this topology reduces message traffic through the switch, and thus reduces the system's average communication latency.

2.2 Our Cluster

After much study, we decided to use a *hypercube-star hybrid* topology along the lines of **Loki** [10], a cluster built by Michael Warren of Los Alamos National Laboratories. In 1996, Loki was able to achieve 1.2 (measured) Gflops and cost $63,000 to build. Just a year later it could have been built for $28,000, and today its price would be much less.

In an N-dimensional hypercube, each of the 2^N machines has a direct link to each of its N nearest neighboring machines, and an indirect link to all other machines. Figure 3 illustrates this with a 3-dimensional hypercube:

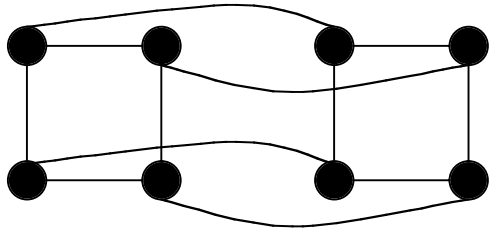

Figure 3
A 3-Dimensional Hypercube

Besides providing cost-effective, efficient communication, parallel algorithms based on other topologies can be mapped onto a hypercube. Figure 4 shows how a ring can be mapped onto the 3-D hypercube of Figure 3:

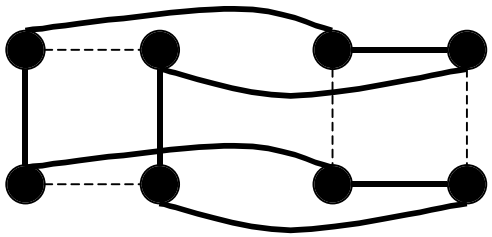

Figure 4
A Ring Mapped Onto a 3-Dimensional Hypercube

Loki uses a hypecube-star hybrid topology, meaning it uses a switch augmented with direct links between each node and its hypercube neighbors. Neighboring nodes thus have

direct links, while non-neighboring nodes have 1-hop links via the switch. This topology has lower communication latency than a ring-star hybrid, because each PC is directly connected to *N* other PCs, rather than just two, as shown in Figure 5:

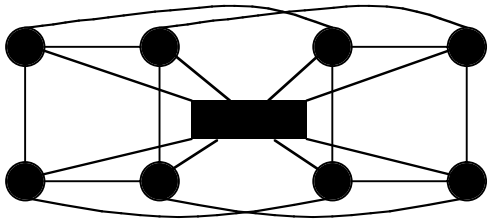

Figure 5
Our Hypercube-Star Hybrid Cluster

2.3 Software Decisions

Operating System. Virtually every Beowulf cluster uses *Linux* as its underlying operating system, for the following reasons:

1. It is *free*.
2. It is *stable*, allowing computations to run for weeks or months without rebooting.
3. It is *open-source* software, allowing its users to make changes as necessary.

Parallel Processing Support. For parallel programming, we chose the *Message Passing Interface* (MPI) [5] over the *Parallel Virtual Machine* (PVM) [4] for these reasons:

1. PVM's heterogeneous hardware interoperability adds unnecessary overhead in our homogeneous system.
2. PVM's language interoperability adds unnecessary overhead for our system.

The resulting system came to be known as **MBH'99**, which at various times of the year stood for *Miniscule Budget Hypercube '99*, *Mark's Beowulf Hypercube '99*, and *Mark's Big Headache '99* (Mark being our student author).

3. IMPLEMENTATION

3.1 Hardware Configuration

Thanks to an in-house grant, we had a project budget of just over $6,000. Unfortunately, a "Loki-like" cluster was not yet *that* inexpensive. After making inquiries, we learned that a dozen 486 machines in various states of repair were available for free as a result of a campus-wide upgrade to the Pentium platform. These machines had different ethernet cards, amounts of memory, disk space, and so on. By cannibalizing some of the twelve machines, we were able to construct eight machines that were more or less the same. The result was that each 486 had at least 16 Mb of RAM, and at least 400 Mb of disk space. Only one machine had a CD-ROM drive.

Thanks to these machines being free, we were able to devote our entire budget to our cluster's communication network: 32 3Com ISA 3C515 10/100 Mbps cards at $160 each, a Samsung SmartEther SS6208 10/100 Mbps switch for $800, and miscellaneous other hardware, for a total cost of $6100. These components have since dropped in price by roughly 50%.

3.2 OS Configuration

We chose Redhat Linux as the operating system for our cluster, primarily for its ease of installation. We installed the OS on the one machine that had a CD-ROM drive, and then used it to do a network-install on each of the other machines. Redhat 5.1 was the newest version at the time of the project. Subsequent releases of Redhat have supposedly fixed some of the problems we describe below.

Unfortunately, network installations through our "Linux compatible" 3Com 100Mbps cards succeeded on a machine or two, but failed on most. By replacing the

3Com card with a spare NE2000 compatible (10 Mbps) ethernet card, network installation would proceed smoothly. We ended up moving this same NE2000 card from machine to machine in order to get Linux installed onto each machine.

Once the operating system was installed on each machine, the four 3Com cards had to be installed. This was also easier to say than do – having multiple ethernet cards in the same machine caused a variety of problems.

The first problem was that our 3Com 100Mbps cards were MS-Windows 9x "plug-and-play" cards. Unfortunately, "plug-and-play" capabilities must be turned off for Linux, and these cards have no hardware or software switch for turning off those capabilities. As a workaround, we used a Linux *isapnp* utility in the system's initialization scripts to read each card's information from a configuration file whenever the system is booted. (To get the card's I/O port, IRQ, and DMA channel information for the configuration file, a second utility named *pnpdump* was used.) The correct Linux module was then loaded to control the card.

The next problem was that each time the system booted, the four cards might be set up in a different order, causing conflicts with their hardware settings. To fix this problem, we created a script that, once each card was set up, remapped their hardware addresses to their IP addresses.

This same script was also used to solve our next problem: setting up our routing tables to make neighboring nodes in the hypercube communicate directly. On each PC, this script sets up the tables so that (i) a message to a neighboring node is routed over the card linked to that neighbor and (ii) a message to a non-neighboring node is routed to the card linked to our switch. Writing this script was time-consuming, but not difficult.

3.3 MPI Configuration

Installation and configuration of MPI went smoothly. However, upon trying to run MPI, we hit our final problem. Since our cluster had just eight nodes with static IP addresses, we tried to map host names to IP addresses solely via the /etc/hosts file on each machine. However, this was insufficient; to let the PCs communicate we had to enable and run domain name service. That is, to actually map our PC's names to IP addresses, we had to install the *bind* utility, and then run *named*, the domain name service daemon. To avoid having name resolution consume network bandwidth, these were installed on each machine. With each machine acting as its own name-server, MPI then worked correctly.

4. OBSERVATIONS

4.1 Topology Costs

Each of the M machines in a hypercube-star hybrid has $log_2(M)$ direct links to its hypercube neighbors, plus one link to the switch. Each machine in a 3-D hypercube-star hybrid cluster has 4 links (3 direct, 1 switched); each machine in a 4-D cluster has 5 links (4 direct, 1 switched), and so on. M must be a power of two, and each machine in the cluster needs another link each time M doubles. The cluster's per-node bandwidth and cost thus grow *logarithmically* with M.

For comparison purposes, each of the M machines in a completely connected cluster has $M-1$ direct links to each of the other machines in the cluster. If M increases by one, each machine must gain another link. The cluster's per-node cost and bandwidth thus grow *linearly* with M.

By contrast, each of the M machines in a ring-star hybrid cluster has 2 direct links (to neighbors in the ring), plus 1 to the switch. A machine never needs more links,

regardless of how *M* changes. A cluster's per-node cost and bandwidth are thus *constant*, and independent of *M*.

The hypercube-star hybrid thus appears to offer a "middle-of-the-road" cost, in return for "middle-of-the-road" performance. Figure 6 shows the costs of each topology in terms of *network ports*:

Network Ports	*Completely Connected*	*Hypercube-Star Hybrid*	*Ring-Star Hybrid*	*Star*
F(8)	56	32	24	8
F(*M*)	$M \times (M-1)$	$M \times (\log_2 (M)+1))$	$M \times 3$	M
O(F(*M*))	$O(M^2)$	$O(M \log_2 (M))$	$O(M)$	$O(M)$

Figure 6
Topology Cost (in Network Ports)

While these costs for an 8-machine cluster are not that far apart, the differing costs quickly manifest themselves as *M* increases. Figure 7 shows the cost for each topology in terms of network ports for 16, 32, and 64-machine clusters:

Network Ports	*Completely Connected*	*Hypercube-Star Hybrid*	*Ring-Star Hybrid*	*Star*
***M*=16**	240	80	48	16
***M*=32**	992	192	96	32
***M*=64**	4032	448	192	64

Figure 7
Network Ports Required (as *M* Grows)

Even with dropping card prices, the cost of network ports can quickly strain a tight budget even if one is using the ring-star hybrid topology. This expense is one reason many Beowulf-builders choose the star or ring-star topologies.

4.2 Topology Scalability

A less obvious cost of choosing the hypercube topology for a Beowulf cluster is the relative difficulty of adding new nodes to the cluster compared to the star or ring-star hybrid topologies. We call this problem the *scalability problem*.

To surpass the performance of a desktop, a cluster of cast-off PCs must consist of many, many machines. Also, being able to add cast-off PCs to one's cluster is an attractive, inexpensive way to increase a cluster's processing power.

This approach is being used by the **Stone SouperComputer** [8], a Beowulf cluster at Oak Ridge National Laboratories. This cluster consists entirely of cast-off machines (currently 128 of them). The cluster grows as people donate their old machines, making scalability very important. To solve the scalability problem, the Stone SouperComputer uses the standard 10 Mbps ethernet bus topology shown in Figure 8:

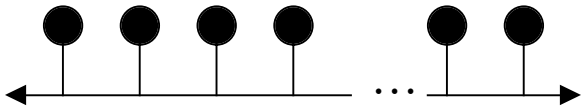

Figure 8
Bus Topology (ORNL's Stone SouperComputer)

With this topology, each machine in the cluster requires just one 10 Mbps port. If each cast-off already has such a card, the per-node hardware cost is $0. Of course, its bandwidth is quite limited, but such a cluster is fine for highly coarse-grained parallelism.

Compared to the Stone SouperComputer, our cluster is not easy to expand. To add machines to a hypercube, its dimension must increase from N to $N+1$ (e.g., from 8 machines to 16, 16 to 32, etc.), which adds one to the number of network ports needed by each machine. If 1-port network cards (the least expensive) are being used, then an additional card must be installed on each machine. Beyond the physical effort of opening

the cases to install the new card on *M* machines, each machine's OS configuration must also be modified, as described in 3.2.

The limited number of card slots on a PC motherboard (usually six or less) compounds the problem. Other subsystems (e.g., video) also use these slots, so that if a hypercube-star hybrid cluster has five-slot motherboards, a 3 is the highest dimension that hypercube can achieve. (The video card and switched-link network card each use one slot, leaving just three for the direct links.)

This problem can be addressed by using multi-port (e.g., 2-port or 4-port) network cards. However, such cards cost more per port than 1-port cards, so this has budget implications, and each machine's system must still be reconfigured to use the new port.

The ring-star hybrid topology represents a nice compromise between ORNL's easy-to-scale/low-bandwidth Stone Souper-Computer and a harder-to-scale/high-bandwidth hypercube-star hybrid. As seen in Figure 2, each machine in a ring-star hybrid requires three network ports: two for direct links and one for the switched link, regardless of how many machines are in the cluster. This becomes a real benefit when one wishes to add new machines to the cluster, since all that is required to integrate the new machine is:

1. Ensure that the new machine has three network cards;
2. Create an opening in the ring by disconnecting machine *M-1* from machine 0;
3. Connect machine 0 to the new machine, and the new machine to machine *M-1*;
4. Connect the new machine to the switch; and
5. Configure the routing tables on machine 0, machine M-1, and the new machine.

Unlike our hypercube-star hybrid, only two of the machines in a ring-star topology must be modified to add a new machine.

The only complication occurs when a new machine is to be added and all ports on the switch are filled. In this case, either the switch can be upgraded to one with more ports, or a new switch can be added and the switches joined via their uplink ports. The original machines can then either be left in place, or evenly distributed across the two switches. Figure 9 shows the cluster from Figure 2 with two additional nodes, and the machines evenly distributed across two 12-port switches:

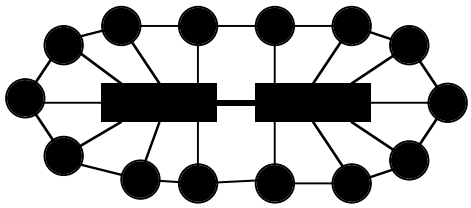

Figure 9
A Ring-Double Star (Multi-Switch) Hybrid

More nodes can then be added to each switch. When both switches are filled, the process can be repeated. When the cluster grows beyond three switches, the switches can be organized into a tree so as to minimize the number of hops between any two machines, as shown in Figure 10:

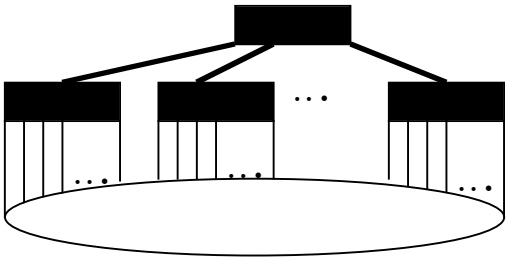

Figure 10
A Multi-Switch Ring-Tree Hybrid

Such reorganization can be done without modifying any of the cluster's machines. The ring-star hybrid is thus an extremely scaleable cluster topology.

5. CONCLUSIONS

Building a Beowulf cluster is a rewarding senior project in which one learns much more than the nuts and bolts of combining several computers. Such a project can be done on a very limited budget (or even no budget if one wishes to adopt the model of the Stone SouperComputer).

In selecting a cluster's topology, there are tradeoffs between cost, bandwidth, and scalability. A completely connected topology is only practical for a small cluster (e.g., 4 machines) that is unlikely to ever grow, as its superior bandwidth comes at a very high cost and poor scalability. A hypercube-star hybrid topology is practical for a small-to-medium sized cluster that is unlikely to grow, as it trades off high bandwidth against medium cost and low scalability. A ring-star hybrid topology is practical for a small, medium, or large cluster that is likely to grow, as it trades off good scalability and low cost against lower bandwidth. A star topology is good for a small, medium, or large cluster running coarse or medium-grained parallel computations and that is likely to grow, as it trades off ease of scalability and low cost against reduced bandwidth. A bus topology is good for a small, medium, or large clusters running very coarse-grained parallel computations and that is likely to grow, since it trades off extreme ease of scalability and very low cost against very low bandwidth.

We recommend the project of building a Beowulf cluster as a senior project for any student (or a team of students) interested in operating systems, networking, and parallel computing. The theoretical and technical challenges make it an absorbing and rewarding capstone project for an undergraduate student's education.

REFERENCES

[1] Becker, D.,Sterling, T., Savarese, D., Dorband, J., Ranawak, U., and Packer, C., BEOWULF: A PARALLEL WORKSTATION FOR SCIENTIFIC COMPUTATION, *Proceedings of the International Conference on Parallel Processing*, 1995.

[2] Becker, D., *Linux Network Drivers*, http://www.beowulf.org/linux/drivers/index.html.

[3] B. Brooks and E. Billings, *LoBoS and LoBoS2's Home Page*, August 1997, http://www.lobos.nih.gov/.

[4] Geist, Al, *PVM Parallel Virtual Machine*, August 1999, http://www.epm.ornl.gov/pvm/.

[5] Grop, B. and Lusk, R., *The Message Passing Interface (MPI) Standard*, February 1999, http://www-unix.mcs.anl.gov/mpi/.

[6] Grossfiel, A., *The Wulfpack Home Page*, http://wulfpack.med.jhmi.edu/.

[7] Hill, J., Warren, M., and Goda, M., "I'm not going to pay a lot for this supercomputer!", Linux Journal, 45, 1998.

[8] Hoffman, F., Hargrove, W., Schultz, A., *The Stone SouperComputer, ORNL's First Beowulf*, May 1999, http://www.esd.ornl.gov/facilities/beowulf/.

[9] Merkey, P., *The Beowulf Project at CESDIS*, http://www.beowulf.org/.

[10] Warren, M., *Loki - Commodity Parallel Processing*, http://loki-www.lanl.gov/.