It's Magic: SourceMage GNU/Linux as HPC Cluster OS

Jörg Cassens and Zoran Constantinescu
Norwegian University of Science and Technology (NTNU)
7491 Trondheim, Norway
{cassens|zoran}@idi.ntnu.no

May 26, 2003

Abstract

The goal of the presentation is to give an overview about how to build a commodity PC based GNU/Linux cluster for High Performance Computing (HPC) in a research environment. Due to the extreme flexibility of the GNU/Linux operating system and the large variety of hardware components, building a cluster for High Performance Computing (HPC) is still a challenge in many cases. At the Division of Intelligent Systems at the Norwegian University of Science and Technology (NTNU), we have build a 40 node HPC cluster for research purposes using the source-based GNU/Linux distribution Source Mage.

We describe a methodology for designing and installing a highly customized GNU/Linux cluster.

Different types of Linux distributions will be mentioned, binary-based and source-based, with their advantages and disadvantages.

The presentation will focus on using SourceMage for HPC, specifying the 'magical' ideas behind it: the ease of upgrading to the latest available version of the source code, a packaging system for keeping track of dependencies, optimized compiles for the hardware architecture used, easy integration of new packages, amongst others.

1 Introduction

High performance computing (HPC) has evolved into a small, stable, and high-priced market. When looking at HPC systems, we see two developments:

- A reduced number of vendors supporting the traditional type parallel computers and an
- increased performance and capacity of clusters based on off-the-shelf components in terms of CPU power, storage capacity, OS abilities, network, and availability of parallel software.

The Beowulf class of parallel computing machines started as a small research project at NASA in 1993, with the goal of building a one GFlops parallel workstation for less than $50.000. The idea was to use commodity off-the-shelf (COTS) hardware and software configured as a cluster of machines, and which exhibits excellent price-performance, provides a robust environment and has broad applicability.

In June 2002, there were about 80 Beowulfs in the top500 (the list of most powerfull computers, see see [8]), and the Beowulf population is estimated at several thousands.

Two are the main driving technologies for the success of Beowulf type clusters: cheap hardware and open source software. Most Beowulfs use mass market commodity off the shelf PC technology as hardware platform.

The main advantage is that by using no customized components it allows to buy the hardware from multiple vendors. This approach exploits components that are widely accepted by industry standards and benefit from prices resulting from heavy competition and mass production. Recent advances in performance of such components make them even more appealing for such clusters. One big advantage is their rapid response to technology trends. The latest advances in microprocessors, storage or other mass market technologies tend to find their way into PCs much faster than to other platforms. Technologies as old as only a few months can easily be integrated in Beowulf class systems.

Beowulf exploits readily available, usually free software systems. The success comes from the unification of public domain parallel tools and applications for the scientific software community. Such tools and applications are the result of decades of parallel processing research and on many attempts to use loosely coupled computers for a variety of applications. Some of these components include: parallel communication libraries, parallel file systems, tools for configuring, scheduling, managing and tuning parallel applications, higher level scientific libraries, etc. The quality of many such publicly available software is comparable with that offered by many commercial software vendors.

Most Beowulf class systems employ UNIX-like operating systems, with source code availability and low or no cost. The cost issue is very important because paying an OS license for each node of a large cluster (hundreds of nodes) could be prohibitively expensive. Source code availability is important because it allows custom modification to facilitate optimized performance. The most widely used operating systems in Beowulf clusters are Linux and BSD. There are also a few Beowulf clusters using Solaris or Windows as operating system.

One other important feature of Beowulf type clusters is that they enable do-it-yourself cluster computing, which fits perfectly for many academic and research environments.

The focus of most Beowulf class systems is science and engineering applications. These applications are mostly floating point intensive and require substantial amount of memory. There are many new applications in the area of data processing, especially in the new biocomputing field, where such systems are beginning to be used. Beowulfs are also finding significant use in computer science research, for the development of new tools and environments, such as metacomputing. One of fastest growing area of Beowulf use is in the computer science education. These systems offer flexibility and price advantages over other commercial offerings, which makes them ideal for

pedagogical purposes.

2 Cluster Architectures

Not every Cluster is built as a High Performance Computing (HPC) resource. The two most common cluster forms are:

- **Load Balancer:** Distribute uniform workload, e.g. web server farms.
- **Computational Cluster:** Conquer a computationally difficult problem by using several machines.

In this paper, we take a look at the second type: A computational cluster as an alternative to a conventional supercomputer.

The underlying architectural idea in computers like the common desktop PC is the use of one instruction on one piece of data at any given time. For HPC, different architectures were traditionally discussed and used.

Take for example one piece of data and apply different processing directives at any given time, and you get the multiflow machine.

The most common form of traditional supercomputers used the same instruction on different data at the same time. This can most easily be achieved by having several CPUs working together. This type can additionally be divided into systems were all CPUs share access to the same memory and systems where each systems sees a different memory. Applications can be of a type where memory access is shared or of a type where the teamworking is done by sending messages between them.

The theoretic step from a traditional supercomputer to cluster architectures seems now not to be very far fetched: what if the different memories and the different CPUs were not part of one single computer, but if one would take a collection of computers on a network that can function as a single computing resource with a distributed memory architecture.

2.1 ClustIS Architecture

For the original ClustIS project (see [2]), we have chosen simplified Desktop-PC's from one of our university's regular hardware suppliers. The computers have AMD Athlon CPU's, 100Mbit/Sec ethernet cards, Hard disks, a simple graphics card and up to 2GB of RAM. The cases are of standard Desktop type with a standard Power Supply and no CD-ROM or Soundcards. They are placed side-by-side on workhop shelfs. This reduced the costs considerably compared to rack servers.

One of the biggest advantages of specialized server hardware is the extendend mean time between failures: the power supply is of bigger dimension, casing and cabling are of better quality, and so one.

Downtime of computing nodes is no issue for COTS clusters: just replace it with a spare one or wait for the repair. This is different for the master node, which is in charge of tasks like job scheduling. Therefore, our master node is a server-type rack mounted computer. We have a second, smaller rack mount PC integrated into the cluster. Under

normal operating conditions, it works as a file server (NFS-mounts) for both the cluster and the Division of Intelligent Systems, offering an easy exchange of and access to data for the users of the cluster. In case of a failure of the master node, it can take over the jobs of the master node as well.

During the use of the cluster, a collaboration with another organizational unit of our Department was established. Therefore, some rack mounted PC's originally delivered for another cluster were integrated into ClustIS. Our concept proofed to be flexible enough to integrate these new systems despite of some hardware differences.

Per may 2003, ClustIS consists of the following systems:

- Master: AMD Dual Athlon MP 2100+ (1.66 GHz), 2GB RAM, 80+120GB IDE HD, 1*Gigabit Ethernet, 1*100MBit Ethernet
- Node type 1: (16 nodes: 1...16) AMD Athlon XP 1700+ (1.46 GHz), 2GB RAM, 1*40GB IDE HD, 1*100MBit Ethernet
- Node type 2: (12 nodes: 17...28) AMD Athlon XP 1700+ (1.46 GHz), 1GB RAM, 1*40GB IDE HD, 1*100MBit Ethernet
- Node type 3: (8 nodes: 29...36) AMD Athlon MP 1600+ (1.4 GHz), 1GB RAM, 1*18GB SCSI HD, 2*100MBit Ethernet
- Node type 4: (1 node: 37) AMD Dual Athlon MP 1600+ (1.4 GHz), 1GB RAM, 3*18GB SCSI HD, 2*100MBit Ethernet
- Storage: AMD Athlon XP 1700+ (1.46 GHz), 0.5GB RAM, 8+2*80GB IDE HD, 1*Gigabit Ethernet, 1*100MBit Ethernet

Important for clusters is the network interconnect used. For ClustIS, we went for a low cost alternative by using a private switched 100Mbit/sec Ethernet between the nodes and giving both the Master and the Storage Node Gigabit Ethernet access to the switch.

This network type seems suitable for the size of our cluster and the need of the applications we run. For larger clusters or applications which depend on a high bandwith or low latency interconnect, other network types might be more usefull. These solutions exist (see e.g. Myrinet), but are more expensive.

3 Linux

Most of Beowulf clusters are using GNU/Linux as an operating system because of its performance, availability of source code, better device support, and wider user acceptance. The extremely low cost is also very important. The key features for GNU/Linux in research are: sofistication, accessibility and low cost.

Currently there are many Linux distributions available. The choice of which distribution to use in a cluster environment is influenced by many factors. For example, which software packages come with a distribution and/or the familiarity with a certain distribution.

	Binary	Source
Bytes to download	less	more
Time to compile	short	long
Install time	short	long
Latest software versions	no	yes
Compilation logs	no	yes
Optimized binaries	maybe	yes
Architecture specific binaries	maybe	yes

Table 1: *Small comparison table: Binary and source based distributions*

3.1 Linux Distributions

There are basically two types of Linux distributions: binary based and source based. A binary distribution consists of a certain number of precompiled software packages, while a source based distribution is compiled at install time using the latest source code of the software.

Most binary distributions offer support for cluster use by including the necessary tools and libraries. System suppliers offers solutions based on existing commercial binary distributions like Red Hat or SuSE. Or Linux distributors offers complete systems with hard- and software, like in the case of the PPC-based Yellowdog distributions.

These system suppliers often focus on stability. The setup might therefore include older versions of software which is known to harmonize with each other and the hardware. This is a perfectly valid goal and often suits the needs in e.g. engineering domains or whenever the cluster is the tool, and not the resarch object.

Research systems, especially systems focusing on research in distributed or parallel systems itself, might be in need for more ”bleeding edge” versions. It is not always easy to combine a binary distribution in a stable way with newer software versions, given the complexity of dependencies.

A source based distribution consists of the source code of the packages which are compiled and configured at install time. The source codes for the packages are always downloaded directly from the software authors' homepages and mirrors. This means that it will always be getting the latest and greatest version of software packages, unlike other distributions that ship with outdated packages. Then, they are compiled with the architecture and optimizations that the system administrator specifies. Finally, they are installed, tracked, and archived for easy removal and upgrades. See table 1 for an overview about the differences between source based and binary distributions.

A source based distribution offers the system administrator more control over the GNU/Linux system. At the same time, if offers enhanced performance and customization. The cost for this is the time required to compile the software packages. However, time for compilation can be allocated from the computer when this is not used.

Having allways the latest version of the software, built from the current sources, makes the operating system free of known vulnerabilities and exploits.

There are a few source based GNU/Linux distributions: Source Mage, Gentoo, Lunar and Source Mage. They follow the same procedure of instalation, by downloading

the source code and compiling it. The major differences between these distributions are the way software packages are described (source code location, compile options and procedure, dependencies) and the number of available packages, i.e. which software packages can be installed using one distribution.

3.2 Source Mage

We decided to use the Source Mage GNU/Linux (SMGL) distribution, a source based distribution. It has many powerful features that set it apart from other distros.

Up-to-date packages can be auto-built using the optimization settings and build-time functionality the administrator wanted, rather than what other distro creator thought would be best for him. For example, if you don't want GNOME on the system, the applications won't have optional GNOME support enabled.

It has an advanced package management system, supporting a number of advanced features including dependencies, fine-grained package management, path sandboxing, safe unmerging, system profiles, virtual packages, config file management.

Source Mage is a simple yet powerful source based distribution for advanced systems administration. The package management system is called sorcery. Sorcery is comprised of modular, easily modified, command line and menu driven BASH scripts. A system administrator wielding sorcery can keep FHS 2.2 complaint Source Mage boxes current with the latest stable software releases.

Source Mage offers most of the features of modern GNU/Linux distributions. There is a menu driven installer on the Installation/Rescue CD image which simplifies the creation of a new box. After installation Source Mage has both command line and menu driven source management programs, making it easy to use both for beginners or more advanced administrators. It can also be used in automatic scripts.

Software installed by sorcery, with few exceptions, are the authors' latest stable release. Compiling software with the optimizations, options, and architecture that the sys admin specifies is how a Source Mage box achieves excellent run-time performance. Sorcery can discover and automatically fix broken library dependencies caused by an upgrade or removal of a library.

The Grimoire contains all of the "spells" that Source Mage uses. Each of these spells contains the instructions for downloading the source code, for compiling, and for installing a certain software package. By "cast"-ing the name of a program, the system will download, compile and install the program. For the first step, "wget" is used to download the source code from its web or ftp site. Next, following the instructions contained in the spell, the program is configured and compiled. Some of the spells, notably the kernel, require user input for further configuration options. Most spells are configures automagically, without needing human input beyond the initial command. The last step is the installation. The spell also contains instructions about where to put the compiled files. Logs documenting the filenames and locations of all these files are written to /var/log/sorcery/ so they can later be reversed, either manually or using the "dispel" command.

There are several cases in which casting of individual can become more complicated:

1. **Dependencies:** One spell may depend on another. Fortunately, cast takes care of dependencies automagically. You will be prompted to cast dependencies, which come in two flavors: required and optional. The dependencies you select will be cast before the main spell, so that everything will work right. You can say ”no” to a required dependancy, but it’s not recommended for the faint of heart. Say ”yes” to required dependencies unless you know what you’re doing!

2. **Post-Install setup:** Sometimes spells require post-install configuration and testing that can’t be taken care of by cast. An example is Free Type2 – after casting this spell, you will be reminded that you need to specify the font path of your X server manually, and run ttmkfdir in your True Type font directory. Because there is no agreed-upon standard location for True Type fonts in X-windows, cast cannot do these things for you. It doesn’t know where you’ve decided to install your fonts.

3. **Supported Spells:** Sometimes you will need to recompile a spell that is a dependancy for other programs. After the spell is recast, you may be prompted to recast the spells that depend on it in order to make certain they function with the new version. This is important for programs that are statically linked against common libraries, and particularly necessary when taking care of security vulnerabilities. For example, following the discovery of a severe vulnerability in the commonly used library zlib, a sorcery update recompiled zlib, and then prompted to recompile affected programs, thereby ensuring that the vulnerability was eliminated on that box.

4 Cluster Software

We use a job scheduling system in oder to share the resources fair between the users. The OpenPBS (see [5]) is such a general job scheduler: let’s say you need 5 nodes for 15 hours, you describe that in your job script, then submit it to the scheduler. When the requested number of nodes are available (i.e. not used by other jobs), your job (i.e. script) will be started. You will get also at that point a list of nodes ‘dedicated’ to your job (no other jobs will be scheduled on those nodes by PBS). If your job doesn’t finish after the ‘time’ you specified when submitting the job, it will be stopped by the PBS. The same applies to memory limits and other resources.

On the cluster now, the PBS is scheduling jobs on nodes 9 and up. The first 8 nodes can be used freely for testing purposes, running small programs, debugging etc., and are not touched by PBS. User are discouraged to run ‘long‘ jobs on those nodes (i.e. for more than a few hours).

Clusters are NUMA (Non-Uniform Memory Access) architectures. When we want to run some kind of parallelization, the first question that arises is: How does Information flow in Parallel Programs?

When fully implemented, ClustIS will offer three different answers to this problem:

1. Message Passing

2. Distributed Shared Memory

3. Distribution instead of Parallelization

4.0.1 Message Passing

The application consists of different process running on the different nodes. In order to exchange information with the other parts, they exchange messages. Two standards are important in this respect:

- **MPI (Message Passing Interface):** proposed as a standard for writing message passing programs.
- **PVM (Parallel Virtual Machine):** library enabling collection of heterogeneous computers to be used as one concurrent computational resource, see [6]

Today, the most common used form for parallel computing on ClustIS is the use of the MPICH implementation of MPI, see [3]. MPICH is heavily used in student assignments in courses on parallel computing.

4.0.2 Distributed Shared Memory

The underlying idea here is to implement a kernel extension distributing applications transparent to the user. This technique is not implemented on ClustIS yet, but we plan to use it least on a part of the cluster because of its ease of use for programmers not used to parallel programming.

We cite the general idea from the openMosix website (see [4]): 'openMosix is a Linux kernel extension. [...]

Once you have installed openMosix, the nodes in the cluster start talking to one another and the cluster adapts itself to the workload. Processes originating from any one node, if that node is too busy compared to others, can migrate to any other node. openMosix continuously attempts to optimize the resource allocation.

There is no need to program applications specifically for openMosix. Since all openMosix extensions are inside the kernel, every Linux application automatically and transparently benefits from the distributed computing concept of openMosix. The cluster behaves much as does a Symmetric Multi-Processor, but this solution scales to well over a thousand nodes which can themselves be SMPs.'

4.0.3 Distributed Computing

The third answer is not to parallelize a program at all, but to distribute it. Many applications have the need for high computing power, but it is sometimes sufficient to let the same algorithm run on different data sets instead of parallelizing an algorithm. We have an solution developed in-house for this kind of applications.

Q^2ADPZ `['kwod 'pi: 'si:]` is a modular C++ implementation of a free, open source, multi-user, multi-platform system for distributing computing requests in a TCP/IP network. The users of the system can submit, monitor, and control computing tasks (grouped into jobs) to be executed by computers participating in the Q^2ADPZ

system in form of dynamic shared libraries, executables, or interpreted programs (including Java applications).

Users can provide software, hardware, and platform requirements for each task and the proper computer is automatically selected. The system automatically delivers the input and output data files. Computers executing tasks detect users logging in, and the tasks are terminated or moved to other computers to minimize the disturbance of regular computer users. Q^2ADPZ can operate both in conditions of an open Internet environment or of a closed local TCP/IP network.

The internal communication protocol is based on optionally encrypted XML messages. The system provides basic statistics information on usage accounting. Several user modes are supported: from novice users submitting simple binary executable programs to advanced users who can alter the internal communication interfaces for their special needs. We are currently using the system for research tasks in the areas of large scale scientific visualization, evolutionary computation, and simulation of complex neural network models.

4.1 Application Areas

ClustIS is used both in education and research. The Division for Complex Datasystems (KDS) is using it in its undergraduate courses on parallel and distributed computing.

The Division of Intelligent Systems (DIS) developed the setup and is maintaining the cluster primarily for research purposes. In Intelligent Systems research, there are several areas in need for computational power:

- Data Mining
- Bioinformatics (Protein Folding)
- Large Scale Visualization
- Machine Learning (Cross Validation)
- Genetic Algorithms and Genetic Programming
- Artificial Life

An application domain on ClustIS with a huge need for computational power is data mining in real medical data. The task is to identify the role of different proteins in gene expression. A rough set approach to data mining is used with the in-house developed ROSETTA C++ library, a collection of C++ classes and routines that enable discernibility-based empirical modeling and data mining (see [7]).

Another domain is empirical work on machine learning algorithms. We use Weka, [9], a framework for Machine Learning written in Java Implements which implements all common ML algorithm. Its distributed version enables the cross validation step (multiple testing of the performance of algorithms on different learning and test data sets) to be distributed to the cluster nodes. We included our own framework CREEK (a knowledge intensive Case-Based Reasoner, [1]) into the range of algorithms to compare its performance with other approaches.

ClustIS is, amongst others, on a daily basis also used for computation intensive tasks in Genetic Algorithms, Visualization, and simulation of distributed systems.

References

[1] Agnar Aamodt. Knowledge Acquisition and Learning by Experience – The Role of Case-Specific Knowledge. In G. Tecuci and Y. Kodratoff, editors, *Machine Learning and Knowledge Acquisition – Integrated Approaches*, chapter 8, pages 197–245. Academic Press, 1995.

[2] http://clustis.idi.ntnu.no/. Website, May 2003.

[3] http://www-unix.mcs.anl.gov/mpi/mpich/. Website, May 2003.

[4] http://www.openmosix.org/. Website, May 2003.

[5] http://www.openpbs.org/. Website, May 2003.

[6] http://www.epm.ornl.gov/pvm/. Website, May 2003.

[7] http://rosetta.sourceforge.net/. Website, May 2003.

[8] http://www.top500.org/. Website, May 2003.

[9] http://www.cs.waikato.ac.nz/ml/weka/. Website, May 2003.