**2014**

**Amandeep Singh**

**Computer Science Department**

Final Year

***BEOWULF CLUSTER : Enabling High Performance Computing***



Version 1.0

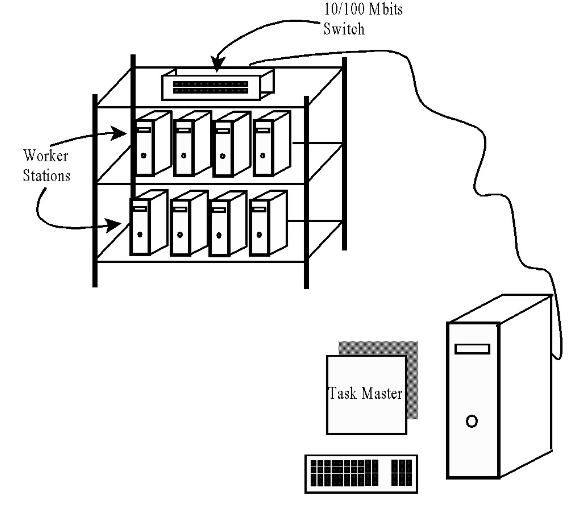
All about high performance Beowulf Clusters

This document describes about the Linux High Performance

Computing (HPC) clusters aka Beowulf clusters and their implementation.

# **BEOWULF CLUSTER : Enabling High**

# **Performance Computing**



*The article outlines the process required to install and configure Linux for use in a cluster environment. While no prior knowledge of either Linux or clusters is assumed, the reader should be forewarned that this is not a trivial task. Note that the document is written in reference to ubuntu linux operating system*

## **The MIT License (MIT)**

Copyright (c) [2013-2014] [Amandeep Singh]

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, dis-tribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AU-THORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

## **List of C­­­ontributors**

## Online Resources

GitHub : <https://github.com/aman-devy/beowulf>

Website : <http://kodevelop.com>/beowulf

## **Table of Contents**

Contents

[BEOWULF CLUSTER : Enabling High Performance Computing 1](#_Toc380498919)

[The MIT License (MIT) 2](#_Toc380498920)

[List of contributors in this document 3](#_Toc380498921)

[Online Resources 4](#_Toc380498922)

[Table of Contents 5](#_Toc380498923)

[Abstract 8](#_Toc380498924)

[High-Performance Computing 9](#_Toc380498925)

[Optimizing performance 10](#_Toc380498926)

[Introduction to cluster systems 11](#_Toc380498927)

[What is a Beowulf ? 12](#_Toc380498928)

[History of the Beowulf 14](#_Toc380498929)

[Uses of a Beowulf 17](#_Toc380498930)

[Cluster types and their applications 18](#_Toc380498931)

[High Availability (HA) clusters 18](#_Toc380498932)

[Web clusters and Web farms 19](#_Toc380498933)

[Load balancing cluster 20](#_Toc380498934)

[Web server cluster 20](#_Toc380498935)

[Application server cluster 20](#_Toc380498936)

[Database server cluster 20](#_Toc380498937)

[Server consolidation cluster 21](#_Toc380498938)

[Clusters in the real world 21](#_Toc380498939)

[Berkeley Open Infrastructure for Network Computing (BOINC) 21](#_Toc380498940)

[SETI@home 22](#_Toc380498941)

[Clusters in the ideal world 22](#_Toc380498942)

[Getting started with a Linux cluster 23](#_Toc380498943)

[The GNU/Linux story 23](#_Toc380498944)

[Why Linux ? 25](#_Toc380498945)

[Main hardware components of a cluster rig 25](#_Toc380498946)

[Head/Master Node 26](#_Toc380498947)

[Slave Computers 26](#_Toc380498948)

[Switch/Router 26](#_Toc380498949)

[Switch vs Hub 26](#_Toc380498950)

[Myrinet 27](#_Toc380498951)

[Interconnecting the Nodes 27](#_Toc380498952)

[Network Topology 28](#_Toc380498953)

[Some commonly used Network Topologies in a Beowulf Cluster 29](#_Toc380498954)

[IP addressing and TCP/IP Protocol 29](#_Toc380498955)

[Other Network setup options 30](#_Toc380498956)

[Software specifications 31](#_Toc380498957)

[Essential software packages (Detailed) 32](#_Toc380498958)

[Operating system 32](#_Toc380498959)

[Message passing interface 33](#_Toc380498960)

[Resource managers 33](#_Toc380498961)

[Necessary libraries 33](#_Toc380498962)

[Batch schedulers 33](#_Toc380498963)

[Compilers 33](#_Toc380498964)

[Monitoring 33](#_Toc380498965)

[File system 34](#_Toc380498966)

[Security 34](#_Toc380498967)

[Process Manager 35](#_Toc380498968)

[Network File System (NFS) 36](#_Toc380498969)

[Network Time Protocol (NTP) – Time Sync 36](#_Toc380498970)

[Integrated cluster scripts and packages 36](#_Toc380498971)

[PGI CDK®Cluster Development Kit® 37](#_Toc380498972)

[Rocks Cluster Distribution 38](#_Toc380498973)

[Oscar(Open Source Cluster Application Resources) 39](#_Toc380498974)

[YACI (Yet Another Cluster Installer) 39](#_Toc380498975)

[ScyldOS 39](#_Toc380498976)

[More about Message passing libraries 40](#_Toc380498977)

[Testing and verification 41](#_Toc380498978)

[The future: Using the cluster 42](#_Toc380498979)

[Final thoughts 43](#_Toc380498980)

[Step by step procedure for setting up a Beowulf cluster 45](#_Toc380498981)

[Detailed procedure to implement beowulf cluster : 45](#_Toc380498982)

[STEP 1: Set up Hardware 45](#_Toc380498983)

[STEP 2: Install linux 45](#_Toc380498984)

[STEP 3: Connect and Configure Network on all nodes 45](#_Toc380498985)

[STEP 4: Configuring the Hostname 46](#_Toc380498986)

[STEP 5: Create new users on each node 47](#_Toc380498987)

[STEP 6: Configure the slave nodes 48](#_Toc380498988)

[STEP 7: Configure the SSH client/server 48](#_Toc380498989)

[STEP 8: Setting up passwordless SSH for communication between nodes 48](#_Toc380498990)

[STEP 9: Configure the NFS client/server 49](#_Toc380498991)

[STEP 10: Configure the master node 50](#_Toc380498992)

[STEP 11: Sharing Master Folder 50](#_Toc380498993)

[STEP 12: Mounting /master in nodes 51](#_Toc380498994)

[STEP 13: Install compilers and other development tools (if not installed earlier) 51](#_Toc380498995)

[STEP 14: Install Message Passing Interface Library (Open MPI or MPICH) 51](#_Toc380498996)

[STEP 15: Setting up a machine file 52](#_Toc380498997)

[STEP 16: Testing 53](#_Toc380498998)

[STEP 17: Configuring NIS Network Information Services (NIS) 54](#_Toc380498999)

[List of commonly used Linux commands 55](#_Toc380499000)

[Tuning a Cluster 56](#_Toc380499001)

[Maintaining a Cluster 57](#_Toc380499002)

[MPI programming on clusters 58](#_Toc380499003)

[Works Cited 64](#_Toc380499004)

## **Abstract**

Real time data processing requires high computation capabilities. There is a trade-off between High Performance and the Cost of computation. In traditional computers we can achieve high performance using multiprocessors and more RAM etc. In cluster computing we can achieve the same performance in less cost using a cluster of general purpose computers. Basic idea in Cluster computing is that, we will have a cluster of general purpose computers which will be processing data or executing some functionality. In parallel processing, problem decomposition is achieved in two ways:

**Domain decomposition** (data are divided into pieces of approximately the same size and then mapped to different processors) or

**Functional decomposition** (We can also divide whole computation into smaller modules and assign each module to an individual processor of a group of processors in a cluster).

Thus we are achieving high performance in less cost.

As a programmer, one may find that he/she needs to solve even larger, more memory intensive problems, or simply solve problems with greater speed than is possible on a serial computer. We can turn to parallel programming and parallel computers to satisfy these needs. Using parallel programming methods on parallel computers gives us access to greater memory and Central Processing Unit (CPU) resources which are not available on serial computers. Hence, we are able to solve large problems that may not have been possible otherwise, as well as solve problems more quickly. One of the basic methods of programming for parallel computing is the use of message passing libraries. These libraries manage transfer of data between instances of a parallel program running (usually) on multiple processors in a parallel computing architecture.

## **High-Performance Computing**

High-Performance Computing (HPC) is a branch of computer science that focus on developing supercomputers, parallel processing algorithms, and related software. HPC is important because of its lower cost and because it is implemented in sectors where distributed parallel computing is needed to:

Solve large scientific problems

– Advanced product design

– Environmental studies (weather prediction and geological studies)

– Research

Store and process large amounts of data

– Data mining

– Genomics research

– Internet engine search

– Image processing

HPC architecture typically consists of massive number of computing nodes (typically 1000s) highly interconnected by specialized low latency network fabric which uses MPI for data exchange.

Nodes = cores + memory

Computation is divided into tasks, distributing these tasks across the nodes and they need to synchronize and exchange information several times a second.

Communication Overheads: Latency in startup time for each message transaction is 1 μs

Bandwidth: The rate at which the messages are transmitted across the nodes /processors is 10 Gbits/sec. You can’t go faster than these limits.

HPC through an example :

*To simulate a bio-molecule of 10000 atoms*

*Non-bond energy term ~ 10^8 operations*

*For 1 microsecond simulation ~ 10^9 steps ~ 10^17 operations*

*On a 1 GFLOPS machine (10^9 operations per second) it takes 10^8 secs (About 3 years 2 months)*

*Need to do large no of simulations for even larger molecules*

*PARAM Yuva – 5 x 10^14 – 3 min 20 sec*

*Titan – 5.7 seconds*

## Optimizing performance

The speed or performance of a computer system can be increased by increasing the bus width or the clock speed of the processor. However, there are limits to the performance benefits that can be achieved by simply increasing the clock speed or bus width. Another way to increase performance is to write optimized algorithms and techniques. But this also fails after a certain level .

So there's an alternative approach to increasing computing power. Instead of using one computer to solve a problem, why not use many computers, in concert, to solve the same problem? This is achieved through the HPC Linux clusters.

## **Introduction to cluster systems**

**What is a Cluster ?**

A cluster is a group of computers which work together toward a final goal. Some would argue that a cluster must at least consist of a message passing interface and a job scheduler. The message passing interface works to transmit data among the computers (commonly called nodes or hosts) in the cluster. The job scheduler is just what it sounds like, It simply takes job requests from user input or other means and schedules them to be run on the number of nodes required in the cluster. It is possible to have a cluster without either of these components, however. Consider a cluster built for a single purpose. There would be no need for a job scheduler and data could be shared among the hosts with simple methods like a CORBA interface.

By definition, however, a cluster must consist of at least two nodes, a master and a slave. The master node is the computer that users are most likely to interact with since it usually has the job scheduler running on it. The master can also participate in computation like the slave nodes do, but it is not required or even recommended in large clusters. The slave nodes are just that. They respond to the requests of the master node and, in general, do most of the computing.

## What is a Beowulf ?

*The [beowulf cluster](http://en.wikipedia.org/wiki/Beowulf_cluster) article on Wikipedia describes the Beowulf cluster as follows:*

*“A Beowulf cluster is a group of what are normally identical, commercially available computers, which are running a Free and Open Source Software (FOSS), Unix-like operating system, such as BSD, GNU/Linux, or Solaris. They are networked into a small TCP/IP LAN, and have libraries and programs installed which allow processing to be shared among them.” – Wikipedia,*[*Beowulf cluster*](http://en.wikipedia.org/wiki/Beowulf_cluster)*, 28 February 2011.*

Beowulf is mainly based on commodity hardware, software, and standards. It is one of the architectures used when intensive computing applications are essential for a successful result. It is a union of several components that, if tuned and selected appropriately, can speed up the execution of a well written application. So it is a multi-computer architecture which can be used for parallel computations. It is a system which usually consists of one server node, and one or more client nodes connected together via Ethernet or some other network. It is a system built using commodity hardware components, like any PC capable of running Linux, standard Ethernet adapters, and switches. It does not contain any custom hardware components and is trivially reproducible. Beowulf also uses commodity software like the Linux operating system, Parallel Virtual Machine (PVM) and Message Passing Interface (MPI). The server node controls the whole cluster and serves files to the client nodes. It is also the cluster's console and gateway to the outside world. Large Beowulf machines might have more than one server node, and possibly other nodes dedicated to particular tasks, for example consoles or monitoring stations. In most cases client nodes in a Beowulf system are dumb, the dumber the better. Client Nodes are configured and controlled by the server node, and do only what they are told to do.

In a disk-less client configuration, client nodes don't even know their IP address or name until the server tells them what it is. One of the main differences between Beowulf and a Cluster of Workstations (COW) is the fact that Beowulf behaves more like a single machine rather than many workstations. In most cases client nodes do not have keyboards or monitors, and are accessed only via remote login or possibly serial terminal. Beowulf nodes can be thought of as a CPU + memory package which can be plugged in to the cluster, just like a CPU or memory module can be plugged into a motherboard.

Beowulf is not a special software package, new network topology or the latest kernel hack. Beowulf is a technology of clustering Linux computers to form a parallel, virtual supercomputer. Although there are many software packages such as kernel modifications, PVM and MPI libraries, and configuration tools which make the Beowulf architecture faster, easier to configure, and much more usable, one can build a Beowulf class machine using standard Linux distribution without any additional software. If you have two networked Linux computers which share at least the /home file system via NFS, and trust each other to execute remote shells (rsh), then it could be argued that you have a simple, two node Beowulf machine. Beowulf programs are usually written using languages such as C and

FORTRAN, and use message passing to achieve parallel computation.

A Beowulf cluster can be as simple as two networked computers, each running Linux and sharing a file system via NFS and trusting each other to use the rsh command (remote shell). Or it can be as complicated as a 1024 nodes with a high-speed, low-latency network consisting of management and master nodes, and so on.

**2 classes of Beowulf Clusters:**

CLASS I: Built entirely using commodity hardware and software. The

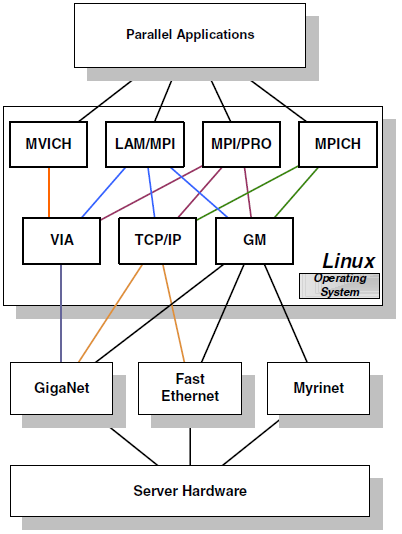
advantages are price, and the use of standard technology (SCSI, Ethernet,

IDE).

CLASS II: Not necessarily built using commodity hardware and software alone. The performance is better than CLASS I.

The true Beowulf is a cluster of computers interconnected with a network with the following characteristics:

1. The nodes are dedicated to the beowulf cluster.
2. The network on which the nodes reside are dedicated to the beowulf cluster.
3. The nodes are Mass Market Commercial-Off-The-Shelf (M2COTS) computers.
4. The network is also a COTS entity.
5. The nodes all run open source software.
6. The resulting cluster is used for High Performance Computing (HPC).



*Components of a beowulf cluster*

## History of the Beowulf

The first Beowulf was developed in 1994 at the Center of Excellence in Space Data and Information Sciences (CESDIS), a contractor to NASA at the Goddard Space Flight Center in Greenbelt, Maryland. It was originally designed by Don Becker and Thomas Sterling and consisted of 16 Intel DX4 processors connected by 10MBit/sec ethernet. Beowulf was built by and for researchers with parallel programming experience. Many of these researchers have spent years fighting with MPP vendors, and system administrators over detailed performance information and struggling with underdeveloped tools and new programming models. This lead to a "do-it-yourself" attitude. Another reality they faced was that access to a large machine often meant access to a tiny fraction of the resources of the machine shared among many users. For these users, building a cluster that they can completely control and fully utilize results in a more effective, higher performance, computing platform. The realization is that learning to build and run a Beowulf cluster is an investment; learning the peculiarities of a specific vendor only enslaves you to that vendor. These hard core parallel programmers are first and foremost interested in high performance computing applied to difficult problems. At Supercomputing '96 both NASA and DOE demonstrated clusters costing less than $50,000 that achieved greater than a gigaflop/s sustained performance. A year later, NASA researchers at Goddard Space Flight Center combined two clusters for a total of 199, P6 processors and ran a PVM version of a PPM (Piece-wise Parabolic Method) code at a sustain rate of 10.1 Gflop/s. In the same week (in fact, on the floor of Supercomputing '97) Caltech's 140 node cluster ran an N-body problem at a rate of 10.9 Gflop/s. This does not mean that Beowulf clusters are supercomputers, it just means one can build a Beowulf that is big enough to attract the interest of supercomputer users. Beyond the seasoned parallel programmer, Beowulf clusters have been built and used by programmer with little or no parallel programming experience. In fact, Beowulf clusters provide universities, often with limited resources, an excellent platform to teach parallel programming courses and provide cost effective computing to their computational scientists as well. The startup cost in a university situation is minimal for the usual reasons: most students interested in such a project are likely to be running Linux on their own computers, setting up a lab and learning of write parallel programs is part of the learn experience. In the taxomony of parallel computers, Beowulf clusters fall somewhere between MPP (Massively Parallel Processors, like the nCube, CM5, Convex SPP, Cray T3D, Cray T3E, etc.) and NOWs (Networks of Workstations). The Beowulf project benefits from developments in both these classes of architecture. MPPs are typically larger and have a lower latency interconnect network than an Beowulf cluster. Programmers are still required to worry about locality, load balancing, granularity, and communication overheads in order to obtain the best performance. Even on shared memory machines, many programmers develop their programs in a message passing style. Programs that do not require fine-grain computation and communication can usually be ported and run effectively on Beowulf clusters. Programming a NOW is usually an attempt to harvest unused cycles on an already installed base of workstations in a lab or on a campus. Programming in this environment requires algorithms that are extremely tolerant of load balancing problems and large communication latency. Any program that runs on a NOW will run at least as well on a cluster. A Beowulf class cluster computer is distinguished from a Network of Workstations by several subtle but significant characteristics. First, the nodes in the cluster are dedicated to the cluster. This helps ease load balancing problems, because the performance of individual nodes are not subject to external factors. Also, since the interconnection network is isolated from the external network, the network load is determined only by the application being run on the cluster. This eases the problems associated with unpredictable latency in NOWs. All the nodes in the cluster are within the administrative jurisdiction of the cluster. For examples, the interconnection network for the cluster is not visible from the outside world so the only authentication needed between processors is for system integrity. On a NOW, one must be concerned about network security. Another example is the Beowulf software that provides a global process ID. This enables a mechanism for a process on one node to send signals to a process on another node of the system, all within the user domain. This is not allowed on a NOW. Finally, operating system parameters can be tuned to improve performance. For example, a workstation should be tuned to provide the best interactive feel (instantaneous responses, short buffers, etc), but in cluster the nodes can be tuned to provide better throughput for coarser-grain jobs because they are not interacting directly with users. The Beowulf Project grew from the first Beowulf machine and likewise the Beowulf community has grown from the NASA project. Like the Linux community, the Beowulf community is a loosely organized confederation of researcher and developer. Each organization has its own agenda and its own set of reason for developing a particular component or aspect of the Beowulf system. As a result, Beowulf class cluster computers range from several node clusters to several hundred node clusters. Some systems have been built by computational scientists and are used in an operational setting, others have been built as test-beds for system research and others are serve as an inexpensive platform to learn about parallel programming. Most people in the Beowulf community are independent, do-it-yourself'ers. Since everyone is doing their own thing, the notion of having a central control within the Beowulf community just doesn't make sense. The community is held together by the willingness of its members to share ideas and discuss successes and failures in their development efforts. The mechanisms that facilitate this interaction are the Beowulf mailing lists, individual web pages and the occasional meeting or workshop. The future of the Beowulf project will be determined collectively by the individual organizations contributing to the Beowulf project and by the future of mass-market COTS. As microprocessor technology continues to evolve and higher speed networks become cost effective and as more application developers move to parallel platforms, the Beowulf project will evolve to fill its niche.

## Uses of a Beowulf

* Life Sciences and Scientific computing
* Making movies
* Commercial servers (web/database etc)
* Aerospace
* Military Applications
* Digital Biology
* CAD/CAM

A Beowulf can be used to perform complex and heavy calculations. We can use the cluster on cluster specific versions of [bioinformatics](http://en.wikipedia.org/wiki/Bioinformatics) tools that perform some sort of heavy calculations

Compute power from clusters and grid can be used for data mining for cyber attacks and fraud detection and control.

## Concurrency and Parallelism

Parallelism is the ability of many independent threads of control to make progress simultaneously toward the completion of a task.

Or in layman’s terms “The simultaneous use of more than one computer to solve a problem.”

Flynn’s taxonomy classifies parallel computing as:

Single Instruction/Multiple Data (SIMD): When the processor executes the same instruction on multiple data at the same time.

Multiple Instruction/Multiple Data (MIMD): When the processor executes different instructions on multiple data.

CPUs may either run independently or cooperate in solving a problem. In either case, they are usually under the control of another processor, which distributes the workload and collects results from them. This concept of parallelism is very important to keep in mind when designing and selecting components for a Beowulf cluster.

Concurrency is a property of the application and defines the parts of the program that can be computed independently. While parallelism is the ability to run concurrent parts of a program that are executed on separate processes at the same time. Parallelism is also a property of the hardware.

## Different forms of clusters

Parallel computing is performed on machines with either single or multiple CPUs and the hardware may be configured in several ways:

### Distributed clusters of machines

Each node is an autonomous computer. They perform parallel programming using message passing, sending data from a program running on one node to a program running on a different node by using Single Instruction Multiple Data (SIMD) or Multiple Instruction

### Multiple Data (MIMD)

Massively Parallel Processor machines (MPP): This has hundreds of

processors programmed on the SIMD model. It includes a large array of processors with one instruction unit that controls the entire array by fetching each instruction and commanding all the processors in the array to carry the instruction out in parallel, each with its own data.

### Shared Memory Processors (SMP)

These are computers with more than one CPU. You can run separate programs on each of the processors simultaneously or a single program in parallel across the processors.

### Distributed clusters of local memory machines

This is a combination of the single CPU and SMP machines via shared memory.

Beowulf may be considered a cluster of the SMP or local memory machines based on the MIMD model.

Since SMP machines communicate via shared memory, you may think that this solution is better because an overhead introduced by a network does not occur. That is almost true, but you also have limited scalability because of memory limitations; the machine cannot have infinite memory.

## **Cluster types and their applications**

All clusters basically fall into two broad categories: High Availability (HA) and High-Performance Computing (HPC). HA clusters strive to provide extremely reliable services. HPC is a cluster configuration designed to provide greater computational power than one computer alone could provide.

Some other cluster types:

1. High-Availability Clusters
2. Load-Balancing Clusters
3. High-Performance Clusters
4. Cloud Computing Clusters

Advantages of clustering:

• High performance  
 • Large capacity  
 • High availability  
 • Incremental growth

Applications of Clustering:

• Scientific computing  
 • Making movies and video processing  
 • Commercial servers (web/database/etc)

### **High Availability (HA) clusters**

HA clusters can be categorized based on function. For example, we would organize a database cluster or a server consolidation cluster under the heading of an HA cluster, since their paramount design consideration is usually high availability. Web clusters, while certainly an HA type of cluster, are often categorized by themselves.

In a typical HA cluster, there are two or more fairly robust machines which mirror each other’s functions. Two schemes are typically used to achieve this.

- In the first scheme, one machine is quietly watching the other machine and waiting to take over in case of a failure.

- The other scheme allows both machines to be active. In this environment, care should be taken to keep the load below 50 percent on each box or else there could be capacity issues if a node were to fail.

These two nodes typically have a shared disk drive array comprised of either a small computer system interface (SCSI) or a Fibre Channel; both nodes talk to the same disk array. Or, instead of having both nodes talking to the same array, you can have two separate arrays that constantly replicate each other to provide for fault tolerance.

Within this subsystem, it is necessary to guarantee data integrity with file and/or record locking. There must also be a management system in place allowing each system to monitor and control the other in order to detect an error. If there is a problem, one system must be able to incapacitate the other machine, thus preserving data integrity.

### **Web clusters and Web farms**

Web clusters generally bring elements from many other computing platforms or other clusters and are often a hybrid of various technologies. A typical Web cluster is more a collection of machines creating an infrastructure than an actual cluster. We will explore a typical Web cluster by starting at the top, where it interfaces with the Internet, and finish at the bottom, where the data content is kept.

### **Load balancing cluster**

In the first layer the hardware is connected through internet. Note that there are multiple redundant connections. These will tie in to a method of load balancing, either through dedicated hardware or through various software products, such as Linux Virtual Server (LVS).

### **Web server cluster**

The next layer is where the Web servers reside. This is simply a group of machines running a Web server application, such as Apache. These servers can present static pages (sometimes called brochure-ware) or they can have the core infrastructure of more complex pages with dynamic content. If a server fails, it will be noted by the load balancers, and future requests will be sent to other servers. If the load increases dramatically, additional servers can be easily added.

### **Application server cluster**

The application layer is where server side code is kept and run. Server side JavaTM is kept and run at this layer. This will generally be a type of high availability solution. At this level, it is still fairly easy to add additional machines to increase capacity.

### **Database server cluster**

Finally comes the database layer. This layer usually requires some form of HA solution. It is not uncommon, in a very large operation, for this to be a mainframe database. It is not difficult, using the procedures outlined above, to create a robust, high availability database for the backend using Linux and various choices for databases, such as IBM's DB2 or the open source MySQL. It is fairly difficult to increase capacity at this level without some thought and planning in the beginning. Also note that, in the Web server space, that various layers can be combined in one machine or one pair of machines. This is largely a consideration of the expected number of hits and how much hardware can be justified.

### **Server consolidation cluster**

It is not uncommon in today’s IT shops to have more and more file servers as departmental servers get added. At some point, there can be a pronounced cost and support advantage to consolidating these servers into a single large machine. If much of your data is stored here, it makes sense that some form of high availability solution be developed. There are also several methods of attaching network storage with multiple servers capable of sharing the data. Regardless of the architecture, the goal is to take many existing servers and combine them into a single solution that never fails.

## **Clusters in the real world**

There exist numerous examples of computer clusters in the practical world. Real World computer clusters generally use commodity hardware to power their nodes. Some of the real world clusters are discussed here:

### **Berkeley Open Infrastructure for Network Computing** (**BOINC**)

It is an open source middleware system for volunteer and grid computing. It was originally developed to support the SETI@home project before it became useful as a platform for other distributed applications in areas as diverse as mathematics, medicine, molecular biology, climatology, and astrophysics. The intent of BOINC is to make it possible for researchers to tap into the enormous processing power of personal computers around the world.

BOINC has been developed by a team based at the Space Sciences Laboratory (SSL) at the University of California, Berkeley led by David Anderson, who also leads SETI@home. As a high performance distributed computing platform, BOINC has about 540,130 active computers (hosts) worldwide processing on average 6.642 petaFLOPS. The framework is supported by various operating systems, including Microsoft Windows, Mac OS X and various Unix-like systems including GNU/Linux and FreeBSD. BOINC is free software which is released under the terms of the GNU Lesser General Public License (LGPL).

### **SETI@home**

("SETI at home") is an Internet-based public volunteer computing project employing the BOINC software platform, hosted by the Space Sciences Laboratory, at the University of California, Berkeley, in the United States. SETI is an acronym for the Search for Extra-Terrestrial Intelligence. Its purpose is to analyse radio signals, searching for signs of extra terrestrial intelligence, and is one of many activities undertaken as part of SETI.

SETI@home was released to the public on May 17, 1999, making it the second large-scale use of distributed computing over the Internet for research purposes, as Distributed.net was launched in 1997. Along with MilkyWay@home and Einstein@home, it is the third major computing project of this type that has the investigation of phenomena in interstellar space as its primary purpose.

## **Clusters in the ideal world**

To maximize the benefits of a cluster, the right hardware must be used. It is generally accepted that for optimal performance, all nodes except the master node must have identical hardware specifications. This is due to the fact that one node which takes longer to do its work can slow the entire cluster down as the rest of the nodes must stop what they are doing and wait for the slow node to catch up. This is not always the case, but it is a consideration that must be made. Having identical hardware specs also simplifies the setup process a great deal as it will allow each hard drive to be imaged from a master instead of configuring each node individually.

## **Getting started with a Linux cluster**

Although clustering can be performed on various operating systems like Windows, Macintosh, Solaris etc. , Linux has its own advantages which are as follows:-  
 • Linux runs on a wide range of hardware  
 • Linux is exceptionally stable  
 • Linux source code is freely distributed.  
 • Linux is relatively virus free.  
 • Having a wide variety of tools and applications for free.  
 • Good environment for developing cluster infrastructure.

In the following paragraphs, some more detailed reasons are provided for using Linux as the cluster operating system.

## The GNU/Linux story

In 1984 Richard Stallman, then working for MIT, became distressed with the way the software industry was evolving. Equipment and software were once shipped with its source code. Now, software is usually sent in binary and proprietary formats. Stallman felt that this closed source software defeated many of the mechanisms which he felt were important to software's continued growth. This concept was documented, much later, in The Cathedral and the Bazaar, by Eric Raymond.

Later that same year, Stallman quit MIT so he could pursue the development of software. He began developing new, free software. He called this the GNU project (pronounced “new”). GNU is a recursive acronym for “GNU’s Not UNIX.” To protect this free software, Copyleft and the GNU GPL (GNU General Public License) were written. In addition, the OSFTM or Free Software Foundation was created to promote this software and philosophy.

Stallman began by creating a compiler (called the GNU C compiler or gcc) and a text editor (GNU Emacs) as a basis for further development. This has evolved, over time, to be a very complete suite of applications and infrastructure code that supports today's computing environment.

Stallman's project was still moving along nearly a decade later, yet the

kernel–the core code running the computer–was not ready.

Then, in 1991, a Finnish student by the name of Linus Torvalds decided to write his own operating system. It was based on the concepts of UNIX®, but was entirely open source code. Torvalds wrote some of the core code. Then he did something quite original: he posted his code in a newsgroup on the growing Internet. His development efforts were complimented by others around the world and the Linux kernel was born.

There's a documentary hollywood movie also, describing the whole story about how the GNU/Linux evolved :

Today's Linux distributions combine the Linux kernel with the GNU software to create a complete and robust working environment. In addition, many companies, such as Red Hat®, SuSE® or TurboLinux®, add their own installation scripts and special features and sell this as a distribution.

## **Why Linux ?**

Linux provides the features typically found in standard UNIX such as multi-user access, pre-emptive multi-tasking, demand-paged virtual memory and SMP support. In addition to the Linux kernel, a large amount of application and system software and tools are also freely available. This makes Linux the preferred operating system for clusters.

The decisions in building a beowulf cluster will have to be made in roughly the four following categories:

1. Operating System

2. Network Topology

3. Communication

4. Software

## Main hardware components of a cluster rig

It is necessary to have at least two machines when building a cluster. It is not necessary that these machines have the same levels of performance. The only requirement is that they both share the same architecture. A switch is more desirable than a hub when designing clusters due the increased speed that they offer.

### **Head/Master Node**

There are four main considerations when building the master node. They are: processor speed, disk speed, network speed, and RAM.

The head node sends the computing tasks to the compute nodes, which in turn must send the result back, as well as sending messages to each other. The faster the better. The head node can also act as a NFS, PXE, DHCP, TFTP, and NTP server over the Ethernet network.

### **Slave Computers**

The slave nodes need to accomplish two tasks: perform the computations assigned to them and then send that data back out over the network. For this reason, their disk performance is not critical. In fact, it is common to have nodes without hard drives in a cluster. The three most important hardware considerations for slave nodes are processor speed, network speed and RAM.

### Switch/Router

A switch/hub or another connecting device is also required to connect the slave nodes with each other and with the master node. Ethernet cable is also required to provide the communication medium.

## **Switch vs Hub**

A switch is more desirable than a hub when designing clusters, due to the increased speed that they offer.

## Myrinet

Myrinet is used by the IBM Linux Cluster Solution. A high-speed switch fabric is used to provide low-latency, high-bandwidth, interprocess communications from node to node. Myrinet is configured to provide full bisectional bandwidth which gives full bandwidth capability from any node to any node at any time and can be scaled to thousands of nodes simply by adding additional switches.

## **Interconnecting the Nodes**

When considering how to connect a cluster’s nodes together, there are two decisions to be made:

a. What kind of network you will use; and

b. Your network’s topology.

The goal is to purchase the fastest network you can afford. Most Beowulf clusters use a switched ethernet fabric, though a few use ATM or Myrinet networks. Fast ethernet (100Base-T) is most common, as Gigabit ethernet (1000Base-T) is just becoming affordable. In many parallel computations, network latency is the primary bottleneck in the system, so a fast network is desirable.

## **Network Topology**

Choosing a topology depends on:

1. the kinds of computations your cluster will be running,
2. the kind of network in your cluster, and
3. how much money you have to spend.

A cluster’s bandwidth-needs depend on its computations. The vast majority of Beowulf clusters use a star topology. However, since network latency is often the bottleneck in a computation, some clusters augment this star with a ring, producing a star-ring hybrid. In theory, the additional bandwidth of these extra links allows each node to communicate with two others directly, reducing traffic through the switch, reducing latency. Some clusters add even more bandwidth by augmenting the ring with a hypercube. Other, more elaborate topologies also exist, especially for clusters of more than 48 nodes. The issue is complicated by manufacturers, who claim that their “smart switches” allow n/2 of a cluster’s n nodes to communicate with the other n/2 simultaneously. If this is true, it is unclear whether the extra connectivity of a ring or hypercube is worth the expense.

Some ranges of possibilities.

• Shared multi-drop passive cable, or

• Tree structure of hubs and switches, or

• Custom complicated switching technology, or

• One big switch

## Some commonly used Network Topologies in a Beowulf Cluster

• Direct wire. Two machines can be connected directly by a Ethernet cable (usually a Cat 5e cable) without needing a hub or a switch. With multiple NICs per machine, we can create networks but then we need to specify routing tables to allow packets to get through. The machines will end up doing double-duty as routers.

• Hubs and Repeaters. All nodes are visible from all nodes and the CSMA/CD protocol is still used. A hub/repeater receives signals, cleans and amplifies, redistributes to all nodes.

• Switches. Accepts packets, interprets destination address fields and send packets down only the segment that has the destination node. Allows half the machines to communicate directly with the other half (subject to bandwidth constraints of the switch hardware). Multiple switches can be connected in a tree or sometimes other schemes. The root switch can become a bottleneck. The root switch can be a higher bandwidth switch.

Switches can be managed or unmanaged. Managed switches are more expensive but they also allow many useful configurations.

## **IP addressing and TCP/IP Protocol**

The most prevalent protocol in networks is the Internet Protocol (IP). There are two higher level protocols that run on top of the IP protocol. These are TCP (Transmission Control Protocol) and UDP (User Datagram Protocol).

IPv4 protocol has 32-bit addresses while the IPv6 protocol has 128-bit addresses.

IP address range is divided into networks along an address bit boundary. The portion of the address that remains fixed within a network is called the network address and the remainder is the host address. Three IP

These addresses are permanently unassigned, not forwarded ranges are reserved for private networks.

• 10.0.0.0 – 10.255.255.255

• 172.16.0.0 – 172.31.255.255

• 192.168.0.0 – 192.168.255.255

by Internet backbone routers and thus do not conflict with publicly addressable IP addresses. These make a good choice for a Beowulf cluster. We will often use 192.168.0.0 – 192.168.0.255 in our examples.

The address with all 0’s in the host address, that is, 192.168.0.0, is the network address and cannot be assigned to any machine.

The address with all 1’s in the host address, that is, 192.168.0.255 is the network broadcast address.

Network address

Broadcast address

## **Other Network setup options**

There are three possibilities.

• Stand alone cluster

A cluster with all private IP addresses. Requires no Internet connection but has very limited access.

• Universally accessible cluster

All machines have public IP addresses and can thus be accessed from anywhere. This can be a high maintenance security nightmare! It can also be hard and expensive to get so many public IP addresses.

• Guarded cluster

There are two scenarios here.

* One machine has two NIC’s: one with a public IP address and the other with a private IP address. All other machines have only private IP addresses. Thus only one machine is vulnerable. IP Masquerading can be used to give the private nodes access to the Internet, if needed. This is the most common Beowulf configuration.
* The cluster machines still have public addresses but firewall software limits access to all but one machine from a limited set of outside machines. One machine is still publicly accessible. This is more flexible but requires a careful firewall setup. This would be the case when a system administrator wants to configure already existing machines into a cluster.

When assigning addresses and hostnames, it helps to keep them simple and uniform. For example: ws01, ws02, ws03, ..., ws08. Or ws001, ws002, ..., ws255. Labelling each machine with its host name, IP address and MAC address can be handy. It is also possible to use dynamically assigned addresses and names for each node, using one machine as the DHCP (Dynamic Host Configuration Protocol) server.

## **Software specifications**

For effective use of a Beowulf cluster as a shared, production tool, the following software list could be considered essential.

1. automated cluster installation

2. remote hardware management (remote power on/off, moni toring CPU temperature etc.)

3. cluster management (monitoring, system administration etc.)

4. job scheduling

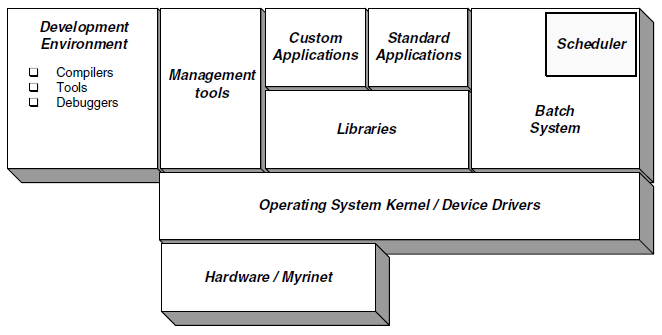
5. libraries/languages for parallel programming

6. tuning and analysis utilities

7. integrated distributions

Some more commonly used softwares and tools:

* MPI, for communication between processes
* NFS, to have a network disk visible and shared to all nodes
* NTP, to synchronize the time of the nodes so that you can compare log events and timestamps
* bootp to boot the nodes from a remote node, so that each node restart fresh with a guaranteed good and uniform setup.
* a set of cluster utilities to make your life easier, such as a distributed ssh to execute the same command on all nodes at the same time.
* a task scheduler, or queue manager, such as Condor, LFS or others, that allow you to prioritize job submissions and eventually measure them for limiting/pricing.
* a watchdog, so to reboot one node automatically if it gets stuck.
* software control for UPS (so to shut down automatically in case of prolonged loss of power)



Software components of a generic Beowulf cluster

## **Essential software packages (Detailed)**

### Operating system

Linux is the de facto OS for HPC clusters.

The latest version of the motherboard BIOS and firmware, which should be the same on all nodes.

### Message passing interface

message passing interface, necessary for the individual processes on the separate compute nodes to share the same data. OpenMPI is a no-brainer. Open MPI is one of the leading MPI-2 implementations. LAM/PVM is a popular choice. MPICH is a free version of MPI . LAM-MPI acts as a communication layer among the nodes.

### Resource managers

### Necessary libraries

MPI, LAM, PVM

### Batch schedulers

A Batch Scheduler allows you to schedule jobs, manage users, set priorities, memory access, node access, duration of job execution and much more. With your Batch Scheduler you can ensure that you are obtaining the most productivity out of your Cluster possible. Examples : torque and moab

### Compilers

Compilers transform source code from the source computer language to the target computer language. Compilers create executable which can be run on your HPC Cluster. Very often, performance varies depending on the compiler used to create the executable

### Monitoring

Each HPC Cluster needs an easy to use monitoring tool allowing you to view all important operating conditions through a web based Graphical User Interface (GUI). Through your web browser, you can remotely access details regarding cpu usage, cpu temperatures, chassis and cpu fan speeds, hard drive temperatures, memory utilization, hard drive swap space and many more metrics. You can even specify different time periods to gauge each metrics over those specified times. Computing overhead for cluster monitoring is extremely low due to the carefully engineered data structures.

One of the most important items to monitor is the computing environment. Maintaining a good computing environment is key to the success and longevity of your cluster

### File system

All computer systems have file systems and those file systems play an important role in data storage, access and reliability. Your HPC Cluster is no different. A clister includes a variety of file systems available depending upon your storage needs, data access and overall performance.

• Ext4 - The ext4 file system is a journaling file system that is 100% backwards compatible with all of the utilities created for creating, managing, and fine-tuning the ext2 & ext3 file system. The ext4 file system can support volumes with sizes up to one exibibyte.

• Network File System (NFS) - Distributed file system that allows NFS servers to give access to their local file system to NFS clients over a network using TCP/IP.

### Security

Hackers are everywhere! And they want nothing more than to hijack your computing resource. Port mapper and ipchains / iptables should be configured to help keep your HPC Cluster secure. In addition, access should be restricted only to verified users via a series of password protected files. Cron scripts must be included to keep user accounts and configuration files synchronized.

portable bash management system, such as the Torque Resource Manager, which allows you to break-up and distribute tasks to multiple machines. Pair Torque with the Maui Cluster Scheduler to complete the setup.

multi-threading math libraries and compilers to build your parallel computing programs

While there are a number of packages to choose from the Linux software, the only ones you are required to install to set up the basic cluster are NFS (network file system) and SSH (secure shell). Specifically, NFS makes it easier to share files between systems, affecting setup of the super computer. SSH is a secure, safe way to remotely connect from one computer to another, critical in the case of setting up a 10-computer super computer cluster.

### Process Manager

The process manager is needed to spawn and manage parallel jobs on the cluster.

The MPICH wiki explains this nicely: “Process managers are basically external (typically distributed) agents that spawn and manage parallel jobs. These process managers communicate with MPICH processes using a predefined interface called as PMI (process management interface). Since the interface is (informally) standardized within MPICH and its derivatives, you can use any process manager from MPICH or its derivatives with any MPI application built with MPICH or any of its derivatives, as long as they follow the same wire protocol.” – [Frequently Asked Questions - Mpich](http://wiki.mpich.org/mpich/index.php/Frequently_Asked_Questions).

The process manager is included with the MPICH package.

MPD has been the traditional default process manager for MPICH till the 1.2.x release series. Starting the 1.3.x series, [Hydra](http://wiki.mpich.org/mpich/index.php/Hydra) is the default process manager. So depending on the version of MPICH you are using, you should either use MPD or Hydra for process management. You can check the MPICH version by running mpich2version in the terminal.

### **Network File System (NFS)**

A distributed file system that enables users to access files and directories located on remote computers and treat those files and directories as if they were local. NFS is independent of machine types, operating systems, and network architectures through the use of remote procedure calls (RPC).

### Network Time Protocol (NTP) – Time Sync

The network time system is an accepted standard for synchronizing time on a system with a remote time server. On a cluster we would synchronize master with the time server.

## **Integrated cluster scripts and packages**

These automated scripts and pre-build packages make the cluster installation very easy on user's part. Many single click installation scripts are also available to put the user at ease.

### **PGI CDK®Cluster Development Kit®**

The PGI CDK® Cluster Development Kit® compilers and development tools enable the use of networked clusters of AMD or Intel x64 processor-based workstations and servers to tackle the largest scientific computing applications. For Linux, the PGI CDK includes pre-configured versions of MPI for Ethernet and Infini-Band. On Windows HPC Server 2008, the PGI CDK integrates with MSMPI and the job scheduler to enable development, debugging and tuning of high-performance MPI or hybrid MPI/OpenMP applications written in Fortran, C or C++.

PGI compilers offer world-class performance and features including auto-parallelization for multicore, OpenMP directive-based parallelization, and support for the PGI Unified Binary™ technology. The PGI Unified Binary streamlines cross-platform support by combining into a single executable file code optimized for multiple x64 processors. This assures your applications will run correctly and with optimal performance regardless of the type of x64 processor on which they are deployed.

PGI CDK Cluster Development Kit Key Features

* Floating multi-user seats for the PGI parallel PGFORTRAN™, PGCC® and PGC++® compilers. World-class single core and multicore processor performance
* Full native support for OpenMPI directive- and pragma-based SMP or multicore parallelization in PGFORTRAN, PGCC and PGC++
* Auto-parallelization for the latest AMD and Intel multicore processors
* Graphical parallel PGDBG debugger and PGPROF performance profiler for auto-parallel, thread-parallel, OpenMP and MPI programs
* Pre-configured MPI message-passing libraries and utilities for Linux
* Optimized BLAS and LAPACK math libraries for Linux
* Comprehensive support for all major Linux distributions and Microsoft Windows HPC Server 2008
* Installation utilities to simplify the setup and management of your Linux cluster
* PGI Roll option - The PGI Roll is maintained and distributed by Stanford University. The PGI Roll contains software only. A valid PGI license is required to use the software. A validPGI CDK license is required to enable remote MPI debugging and profiling.

### **Rocks Cluster Distribution**

The Rocks clustering package from the University of California at San Diego makes it easy to build and maintain a high-performance compute cluster with off-the-shelf hardware. Rocks is termed a cluster provisioning, management and maintenance package. It helps you set up the cluster in the first place (from bare metal); it provides the tools to run parallel programs, and it provides the tools to maintain and extend the cluster after it is created.

The package is delivered as a series of .iso images that you burn onto a series of CDs or DVDs. You then boot the machine that will become the head node from the appropriate DVD or CD, and the installation routine guides you from there. After asking a minimum number of questions in an interactive phase, the installation program builds the head node. Upon reboot, you invoke a single routine (insert-ethers) to add the rest of the machines as compute nodes. To add a compute node, you simply network boot it, and it will be added to the cluster, loaded and configured automatically. After the last node is complete, you have a functional cluster, ready to execute parallel applications.

### **Oscar(Open Source Cluster Application Resources)**

OSCAR (Open Source Cluster Application Resource) software package is a high performance cluster (HPC) used to simplify the complex tasks required to install a cluster. Its advantage is that several HPC-related packages like MPI implementations, LAM, PVM (Parallel Virtual Machine), PBS (Portable Batch Server) etc are installed by default and need not be installed separately.

### **YACI (Yet Another Cluster Installer)**

### **ScyldOS**

Some of NASA’s original Beowulf researchers have started selling ScyldOS , the Scyld Beowulf Cluster Operating System. ScyldOS is Redhat Linux preconfigured for a cluster, including a custom version of MPI. ScyldOS is commercial, open-source software (i.e., the source code is free, but you must pay for the documentation). The ScyldOS customizations presupposes that every cluster will have

1. a single master (i.e., no “two-user mode”), and
2. use a star topology. RHL does not have such presuppositions.

## **More about Message passing libraries**

Message passing libraries provide a high-level means of passing data between process executing on distributed memory systems. These libraries are currently at the heart of what makes it possible to achieve high performance out of collections of individual cluster nodes. Message passing libraries typically provide routines to initialise and configure the messaging environments as well as sending/receiving packets of data.

The two most popular high-level message-passing systems for scientific and engineering applications are MPI (Message Passing Interface) defined by the MPI Forum and the PVM (Parallel Virtual Machine) from Oak Ridge National Laboratory and the University of Tennessee at Knoxville, USA.

MPI I is the de facto standard for parallel programming, both on clusters, and on traditional parallel supercomputers, such as the Cray T3E and IBM SP2. MPI consists of a rich set of library functions to do both point-to-point and collective communication among parallel tasks. The first version of MPI did not specify the mean of spawning a MPI task on a run-time environment. Generally, however, conventions have been adopted by most MPI implementations. There are several implementations of MPI which are available from different sources. Most high-performance hardware vendors support MPI. This provides users with a portable programming model and means their programs can be executed on almost all of the existing platforms without the need to rewrite the program from scratch.

There are two popular and free implementations of MPI, MPICH and LAM. Each of these is a complete version of MPI I.

1. Argonne National Laboratory and Mississippi State University developed the MPI reference implementation, MPICH. It has been ported to most versions of UNIX.
2. LAM (Local Area Multicomputer) is an MPI programming environment and development system developed by Notre Dame University. LAM includes a visualization tool that allows a user to examine the state of the machine allocated to their job as well as provides means of studying message flows between nodes.

MPICH2 implementation relies solely on SSH and does not require a daemon running on each machine in the cluster, it is presumably easier to setup in an environment that involves a large number of nodes. But for my particular setup, either implementation should serve the purpose.

PVM is a software system that allows users to set up a controlling workstation that spawns child processes onto other machines. What makes PVM unique as a parallel programming environment is that it allows for the creation of an encapsulated virtual environment for running parallel programs. The virtual machine provided by PVM allows parallel programs can be run on heterogeneous collections of computers.

With PVM, each user may construct their parallel environment controlled from a single host on which child processes can be launched onto other machines. PVM is implemented via a daemon that is installed on each machine. Any user who has enough resources to compile and install the PVM package on a number of machines can therefore, run it. The PVM library has functions to support and aid the integration of parallel processing tools into the PVM environment. There are tools produced by researcher and vendors for PVM environments. The MPI infrastructure supports parallel tools and utilities by providing standard semantics for communications, contexts, and topologies.

## **Testing and verification**

The last thing you may want to do before releasing all this compute power to your users is test its performance. The HPL (High Performance Lynpack) benchmark is a popular choice for measuring the computational speed of the cluster. You will need to compile it from source with all possible optimizations your compiler offers for the architecture you chose. Compare your results on TOP500.org to compare your cluster to the fastest 500 supercomputers in the world !  
Reboot all the machines and see if each one comes on. See if you can connect to each node from the master. If the hardware appears to be running as designed, invoke your MPI application, whether LAM or another messaging passing interface software.

## **The future: Using the cluster**

The cluster can be used for research on object-oriented parallel languages, recursive matrix algorithms, network protocol optimization, and graphical rendering. Several non-CS faculty can also plan to use the cluster for computational modelling, including a physicist modelling electron behaviour in high-energy laser fields, a chemist modelling complex inorganic molecules, and an astronomer modelling the interaction of Saturn’s Rings and troposphere. Computer science students will receive extensive experience using the cluster in High Performance Computing project, seniors can use it for their senior projects.

Earlier there were single processor desktops and super computers (SMP’s). Now there are local clusters. Even Enterprise cluster /grid systems have also been set up, providing huge processing powers to the community. Global clusters/Grid systems and inter planet clusters are also not far from reality.

## **Final thoughts**

Building our cluster required a great deal of time. Roughly 10 different individuals — students, faculty, and administrators — have contributed to the construction of our cluster. The project of building a supercomputer has captured the imagination of our campus, and has provided an enjoyable focus this past year. As hardware continues to evolve, cluster technology is increasingly affordable. We hope that this report and the documentation available at kodevelop.com will encourage other institutions to build their own clusters.

# Step by step procedure for setting up a Beowulf cluster

 Brief steps to implement Beowulf :

* Collect and connect all the hardware components like master node, slave nodes, ethernet cable and switch
* Install Linux on all the pc's
* configure hostnames of all the nodes
* setup ssh server and client package on all pc's
* install MPI and other software packages using ssh
* install test package and run

## Detailed procedure to implement Beowulf Cluster

Here I’ll be providing steps and details about how a simple 8-node Beowulf cluster named ‘BEO’ was implemented in our college. These are meant for future reference only and follow the standard SRS documentation format.

## Hardware specification

List all hardware details

## Software specification

List all software used

## Networking Technique

Topology and wiring info

### **STEP 1: Set up Hardware**

To set up a cluster, you must assemble and connect the nodes to certain cluster hardware and configure the nodes into the cluster environment

### **STEP 2: Install linux**

Before installing Linux, it is a good idea to gather as much information as possible about the PC's on which Linux will be installed, like the monitor, video card, LAN card, RAM, processor speed, hard disk. Next install Linux. Linux will need to be installed on every node of the cluster.

### **STEP 3: Connect and Configure Network on all nodes**

If you have just two computers, then both machines must be able to reach each other over the network. We can connect them using an ethernet wire. The easiest is to put both machines in the same network with regard to hardware and software configuration, for example connect both machines via a single hub or switch and configure the network interfaces to use a common network such as 192.168.0.x/24. Make sure that IP addresses are assigned to them. If you don’t have a router to assign IP, you can statically assign them IP addresses.  Configuring network involves assigning IP Address, subnet mask and gateway for each node on cluster. It is compulsory to assign static IP address rather than DHCP for cluster private network. To make it simple, we will assign the IP address 192.168.0.1 to the master machine and192.168.0.2 to the slave machine.

Update /etc/hosts on both machines with the following lines: # /etc/hosts (for master AND slave) 192.168.0.1 master 192.168.0.2 slave

### **STEP 4: Configuring the Hostname**

To configure hostname (Machine Name) in Linux edit the hosts configuration file in each node. Open the /etc/hosts file of each machine and add the ip address and hostname of every machine. Also take out the line which associates 127.0.0.1 with the machines hostname. Leave the one which says 127.0.0.1 localhost. This ensures that the loopback address only refers to the local machine. Ex:127.0.0.1 localhost  
192.168.1.105 master\_node  
192.168.1.103 slave node

or node0 10.1.1.1  
node1 10.1.1.2

There may be other lines in this file, but these must be there for the cluster to function. Also modify the /etc/hosts.allow file and add the line ALL : ALL Suppose their are nodes running Ubuntu with these host names: ub0, ub1, ub2, ub3; Defining hostnames in etc/hosts/

Edit /etc/hosts like these:

127.0.0.1 localhost

192.168.133.100 ub0

192.168.133.101 ub1

192.168.133.102 ub2

192.168.133.103 ub3

Note that the file shouldn't be like this:

127.0.0.1 localhost

127.0.1.1 ub0

192.168.133.100 ub0

192.168.133.101 ub1

192.168.133.102 ub2

192.168.133.103 ub3

or like this:

127.0.0.1 localhost

127.0.1.1 ub0

192.168.133.101 ub1

192.168.133.102 ub2

192.168.133.103 ub3

otherwise other hosts will try to connect to localhost when they try to reach ub0.

### STEP 5: **Create new users on each node**

Create a new user in both the nodes. Let us call this new user as mpiuser. You can create a new user through GUI by going to *System->Administration->Users and Groups* and click "Add User". Create a new user called mpiuser and give it a password. Give administrative privileges to that user. Make sure that you create the same user on all nodes. Although same password on all the nodes is not necessary, it is recommended that you do so because it'll eliminate the need to remember passwords for every node. We define a user with same name and same userid in all nodes with a home directory in /mirror. Here we name it "mpiu"! Also we change the owner of /mirror to mpiu: omid@ub0:~$ sudo chown mpiu /mirror

### **STEP 6: Configure the slave nodes**

Now download and install ssh-server in every node. Then logout from your session and log in as mpiuser. To configure the machines that will act as slave nodes:  
  
Install package nfs-common  
Install package openssh-server  
  
If these were already installed, Ubuntu will simply tell you they are already the newest version. Running the install commands just to be sure doesn't hurt anything. Once nfs and the ssh server are installed, configuration of the slave nodes can proceed.

### STEP 7: **Configure the SSH client/server**

Now set up a public key for passwordless ssh logins. The reason for passwordless ssh is to be able to use all the nodes without having to login to each one every time you run MPICH2. The running of the cluster should be automatic across all nodes. Run this command in all nodes in order to install OpenSSH Server : sudo apt-get install openssh-server

### STEP 8: **Setting up passwordless SSH for communication between nodes**

First we login with our new user to the master node:

omid@ub0:~$ su - mpiu

Then we generate an RSA key pair for mpiu:

mpiu@ub0:~$ ssh­-keygen ­-t rsa

You can keep the default ~/.ssh/id\_rsa location. It is suggested to enter a strong passphrase for security reasons.

Next, we add this key to authorized keys:

mpiu@ub0:~$ cd .ssh

mpiu@ub0:~/.ssh$ cat id\_pub.dsa >> authorized\_keys

As the home directory of mpiu in all nodes is the same (/mirror/mpiu) , there is no need to run these commands on all nodes. If you didn't mirror the home directory, though, you can use ssh-copy-id <hostname> to copy a public key to another machine's authorized\_keys file safely.

To test SSH run:

mpiu@ub0:~$ ssh ub1 hostname

If you are asked to enter a passphrase every time, you need to set up a keychain. This is done easily by installing... Keychain.

mpiu@ub0:~$ sudo apt-get install keychain

And to tell it where your keys are and to start an ssh-agent automatically edit your ~/.bashrc file to contain the following lines (where id\_rsa is the name of your private key file):

if type keychain >/dev/null 2>/dev/null; then

keychain --nogui -q id\_rsa

[ -f ~/.keychain/${HOSTNAME}-sh ] && . ~/.keychain/${HOSTNAME}-sh

[ -f ~/.keychain/${HOSTNAME}-sh-gpg ] && . ~/.keychain/${HOSTNAME}-sh-gpg

Exit and login once again or do a source ~/.bashrc for the changes to take effect.

Now your hostname via ssh command should return the other node's hostname without asking for a password or a passphrase. Check that this works for all the slave nodes.

### **STEP 9: Configure the NFS client/server**

NFS allows us to create a folder on the master node and have it synced on all the other nodes. This folder can be used to store programs. To Install NFS just run this in the master node's terminal: sudo apt-get install nfs-server . To install the client program on other nodes run this command on each of them: sudo apt-get install nfs-client if you want to be more efficient in controlling several nodes using same commands, ClusterSSH is a nice tool. Set up an nfs mount in the /etc/fstab file. Use the home directory of the default user on the main node as an nfs share, which is one reason why user accounts need to be identical across all machines. This way, every machine has the exact same home directory.  
Change to the /etc directory and modify the fstab file. You want to append a line like this to the end of the file for the nfs directory to be mounted:  
master\_node:/home/mpi /home/mpi nfs user,exec  
  
That sets up the slave nodes to receive the directory exported by the master node. Now change back to the home directory.

### **STEP 10: Configure the master node**

Install the NFS server on the master node and configure the export:  
Install package nfs-kernel-server  
  
add this line to the /etc/exports file:  
/home/mpi \*(rw,insecure,sync)  
  
sudo exportfs -r to export the directory to all slave nodes

### STEP 11: **Sharing Master Folder**

Make a folder in all nodes with sudo mkdir /mirror, we'll store our data and programs in this folder. And then we share the contents of this folder located on the master node to all the other nodes. In order to do this we first edit the /etc/exports file on the master node to contain the additional line /mirror \*(rw,sync) This can be done using a text editor such as vim or by issuing this command: echo "/mirror \*(rw,sync)" | sudo tee -a /etc/exports Now restart the nfs service on the master node to parse this configuration once again: sudo service nfs-kernel-server restart Note that we store out data and programs only in master node and other nodes will access them with NFS.

### STEP 12: **Mounting /master in nodes**

Now all we need to do is to mount the folder on the other nodes. This can be done manually each time like this:

omid@ub1:~$ sudo mount ub0:/mirror /mirror

omid@ub2:~$ sudo mount ub0:/mirror /mirror

omid@ub3:~$ sudo mount ub0:/mirror /mirror

But it's better to change fstab in order to mount it on every boot. We do this by editing /etc/fstab and adding this line:

ub0:/mirror /mirror nfs

and remounting all partitions by issuing this on all the slave nodes:

omid@ub1:~$ sudo mount -a omid@ub2:~$ sudo mount -a omid@ub3:~$ sudo mount -a

### STEP 13: **Install compilers and other development tools (if not installed earlier)**

To be able to compile all the code on our master node (it's sufficient to do it only there if we do it inside the /mirror folder and all the libraries are in place on other machines) we need a compiler. Get gcc and other necessary stuff by installing the build-essential package:

mpiu@ub0:~$ sudo apt-get install build-essential

### **STEP 14: Install Message Passing Interface Library (Open MPI or MPICH)**

Make sure you install the package build-essential on the main node, otherwise you will have no build tools or compilers.  
Download MPICH2   
The last steps to setting everything up are to put the mpich2 folder on the path so that it can be found by the system. export PATH=/home/mpi/mpich2/bin:$PATH  
export PATH  
LD\_LIBRARY\_PATH="/home/mpi/mpich2/lib:$LD\_LIBRARY\_PATH"  
export LD\_LIBRARY\_PATH  
  
sudo echo /home/mpi/mpich2/bin >> /etc/environment  
  
Everything should now be installed and ready to go. To test this, use the following commands:  
which mpirun  
which mpiexec

Very last thing to do is set up a hosts file for the cluster in the user directory. The file should be named hosts and should be set up as follows - One line for each machine in the network listed by hostname  
Ex:  
master\_node  
slave\_node

Installing MPICH2

Now the last ingredient we need installed on all the machines is the MPI implementation. You can install MPICH2 using Synaptic by typing:

sudo apt-get install mpich2

To test that the program did indeed install successfully enter this on all the machines:

mpiu@ub0:~$ which mpiexec

mpiu@ub0:~$ which mpirun

### **STEP 15: Setting up a machine file**

Create a file called "machinefile" in mpiu's home directory with node names followed by a colon and a number of processes to spawn:

ub3:4 # this will spawn 4 processes on ub3

ub2:2 # this will spawn 2 processes on ub2

ub1 # this will spawn 1 process on ub1

ub0 # this will spawn 1 process on ub0

### **STEP 16: Testing**

Change directory to your mirror folder and write this MPI helloworld program in a file mpi\_hello.c (courtesy of [this blog](http://auriza.site40.net/notes/mpi/openmpi-on-ubuntu-904/)):

#include <stdio.h>

#include <mpi.h>

int main(int argc, char\*\* argv) {

int myrank, nprocs;

MPI\_Init(&argc, &argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &nprocs);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &myrank);

printf("Hello from processor %d of %d\n", myrank, nprocs);

MPI\_Finalize();

return 0;

}

Compile it:

mpiu@ub0:~$ mpicc mpi\_hello.c -o mpi\_hello

and run it (the parameter next to -n specifies the number of processes to spawn and distribute among nodes):

mpiu@ub0:~$ mpiexec -n 8 -f machinefile ./mpi\_hello

You should now see output similar to this:

Hello from processor 0 of 8

Hello from processor 1 of 8

Hello from processor 2 of 8

Hello from processor 3 of 8

Hello from processor 4 of 8

Hello from processor 5 of 8

Hello from processor 6 of 8

Hello from processor 7 of 8

### **STEP 17: Configuring NIS Network Information Services (NIS)**

NIS enables you to create user accounts that can be shared across all systems on your network. The user account is created only on the NIS server. NIS clients download the necessary username and password data from the NIS server to verify each user login. Configuring NIS is necessary in order to perform computation on cluster nodes by passing messages to each other, this requires a global shared space and password less logins. All this can be achieved by configuring NIS. Head node will be the NIS master and all other nodes (Compute nodes) will be NIS clients. An advantage of NIS is that users need to change their passwords on the NIS server only, instead of every system on the network. This makes NIS popular in computer training labs, distributed software development projects or any other situation where groups of people have to share many different computers. The disadvantages are that NIS doesn't encrypt the username and password information sent to the clients with each login and that all users have access to the encrypted passwords stored on the NIS server.

Congratulations! You have a working Beowulf cluster platform.

Enjoy supercomputing

# Commonly used Linux commands

Sudo ssh hostname@username

Sudo gedit /etc/hosts

Sudo gedit /etc/exports

# Tuning a Cluster

# Maintaining a Cluster

# MPI programming on clusters

## An introduction to MPI

Message Passing Interface is an industry-wide standard protocol for passing messages between parallel processors. MPI is a specification for the developers and users of message passing libraries. By itself, it is NOT a library - but rather the specification of what such a library should be. MPI is the most widely used message passing library in parallel processing and super computing tasks. MPI is mainly used for developing parallel processing algorithms and software-programs that can be divided into little pieces so that each piece can be executed simultaneously by separate processors.

MPI is both small and at the same time large enough to run hundreds of dumb nodes and execute multiprocessing tasks. MPI is small in the sense that it require only six library functions to write any parallel code and its vastness comes from the fact that there are more than 200 functions in MPI-2.

A nice online tutorial is also provided here at the following link: <http://mpitutorial.com/mpi-introduction/>

## MPI Programming Model

Originally MPI was designed for distributed memory architectures, which were becoming increasingly popular at that time. As architecture trends changed, shared memory SMPs were combined over networks creating hybrid distributed memory / shared memory systems. MPI developers adapted their libraries to handle both types of underlying memory architectures seamlessly. This means you can use MPI even on your laptops and old computer hardware.

The message-passing programming paradigm assumes that an application program is split into several tasks, each task having its own private data and addressing space. There is no shared global data between any tasks of the program. All synchronization and communication between the tasks is accomplished by explicit library calls between tasks, such as MPI\_SEND or MPI\_RECEIVE. Because the programmer has total control over task interaction, better scalability can be achieved by properly designing synchronization and communication between tasks, more so than with any other parallelization

technique today.

## Why MPI ?

Message passing has been widely adopted because of its close association with the physical attributes of SMP architecture. Message passing supports an interaction between sequential processes. One or more processes run on each processor and they interact via messages conveyed across the physical network.

Today, MPI runs on virtually any hardware platform:

• Distributed Memory

• Shared Memory

A hybrid MPI program is basically a C/C++, Fortran or Python program that uses the MPI library.

Amdahl’s Law Code = Serial Part + Part which can be parallelized. The potential program speedup is defined by the fraction of code that can be parallelized.

Can you get a speed up of 5 times using quad core processor ?

MPI Communicator

A set of processes that have a valid rank of source or destination fields. The predefined communicator is MPI\_COMM\_WORLD, and we will be using this communicator all the time. MPI\_COMM\_WORLD is a default communicator consisting of all processes. Furthermore, a programmer can also define a new communicator, which has a smaller number of processes than MPI\_COMM\_WORLD does.

MPI Processes

For this module, we just need to know that processes belong to the MPI\_COMM\_WORLD. If there are p processes, then each process is defined by its rank, which starts from 0 to p - 1. The master has the rank 0.

MPI Use SSH client (Putty) to login into any of these Multi - Processor Compute Servers with processors varying from 4 to 15

MPI Header Files for different languages

C: #include "mpi.h"

Fortran: include mpif.h

Python: mpi4py

The Smallest MPI library should provide these 6 functions:

1. MPI\_Init - Initialize the MPI execution environment
2. MPI\_Comm\_size - Determines the size of the group associated with a communicator
3. MPI\_Comm\_rank - Determines the rank of the calling process in the communicator
4. MPI\_Send - Performs a basic send
5. MPI\_Recv - Basic receive
6. MPI\_Finalize - Terminates MPI execution environment

Format of MPI Calls

C and Python names are case sensitive while Fortran names are not.

Example: CALL MPI\_XXXXX(parameter,..., ierr) is equivalent to call mpi\_xxxxx(parameter,..., ierr). Programs must not declare variables or functions with names beginning with the prefix MPI\_ or PMPI\_ for C & Fortran. For Python ‘from mpi4py import MPI’ already ensures that you don’t make the above mistake.

MPI\_Init

Initialize the MPI execution environment. This function must be called in every MPI program, and before any other MPI functions and must be called only once in an MPI program. For C programs, MPI\_Init may be used to pass the command line arguments to all processes, although this is not required by the standard and is implementation dependent. MPI\_Init (&argc,&argv)MPI\_INIT (ierr). For python no initialization is required.

MPI\_Comm\_size

Returns the total number of MPI processes in the specified communicator, such as MPI\_COMM\_WORLD. If the communicator is MPI\_COMM\_WORLD, then it represents the number of MPI tasks available to your application. MPI\_Comm\_size (comm,&size) MPI\_COMM\_SIZE (comm,size,ierr) where comm is MPI\_COMM\_WORLD. For python MPI.COMM\_WORLD.size is the total number of MPI processes. MPI.COMM\_WORLD.Get\_size() also returns the same.

MPI\_Comm\_rank

Returns the rank of the calling MPI process within the specified communicator. Initially, each process will be assigned a unique integer rank between 0 and number of tasks - 1 within the communicator MPI\_COMM\_WORLD. This rank is often referred to as a task ID. If a process becomes associated with other communicators, it will have a unique rank within each of these as well. MPI\_Comm\_rank (comm,&rank) MPI\_COMM\_RANK (comm,rank,ierr) where comm is MPI\_COMM\_WORLD. For python MPI.COMM\_WORLD.rank is the total number of MPI processes. MPI.COMM\_WORLD.Get\_rank() also returns the same.

MPI Hello World Program

https://gist.github.com/4459911

MPI blocking message passing routines are more commonly used. A blocking MPI call means that the program execution will be suspended until the message buffer is safe to use. The MPI standards specify that a blocking SEND or RECV does not return until the send buffer is safe to reuse (for MPI\_SEND), or the receive buffer is ready to use (for PI\_RECV).The statement after MPI\_SEND can safely modify the memory location of the array a because the return from MPI\_SEND indicates either a successful completion of the SEND process, or that the buffer containing a has been copied to a safe place. In either case, as buffer can be safely reused. Also, the return of MPI\_RECV indicates that the buffer containing the array b is full and is ready to use, so the code segment after MPI\_RECV can safely use b.

MPI\_SendBasic blocking send operation

This routine returns only after the application buffer in the sending task is free for reuse. Note that this routine may be implemented differently on different systems. The MPI standard permits the use of a system buffer but does not require it.

MPI\_Send (&buf,count,datatype,dest,tag,comm)

MPI\_SEND (buf,count,datatype,dest,tag,comm,ierr) comm.send(buf,dest,tag)

MPI\_Send

MPI\_Send(void\* message, int count, MPI\_Datatype datatype, int destination, int tag, MPI\_Comm comm)

message: initial address of the message- count: number of entries to send

datatype: type of each entry- destination: rank of the receiving process

tag: message tag is a way to identify the type of a message

comm: communicator (MPI\_COMM\_WORLD)

MPI\_Recv

Receive a message and block until the requested data is available in theapplication buffer in the receiving task. MPI\_Recv (&buf,count,datatype,source,tag,comm,&status)MPI\_RECV (buf,count,datatype,source,tag,comm,status,ierr)MPI\_Recv(void\* message, int count, MPI\_Datatype datatype, int source, int tag, MPI\_Comm comm, MPI\_Status \*status)-

source: rank of the sending process

status: return status

MPI\_Finalize

Terminates MPI execution environment

Note: All processes must call this routine before exit. The number of processes running after this routine is called is undefined; it is best not to perform anything more than a return after calling MPI\_Finalize.

MPI Hello World 2

https://gist.github.com/4459944

This MPI program illustrates the use of MPI\_Send and MPI\_Recvfunctions. Basically, the master sends a message, “Hello, world”, to the process whose rank is 1, and then after having received the message, the process prints the message along with its rank.

CommunicationCollective

It is a communication that must have all processes involved in the scope of a communicator. We will be using MPI\_COMM\_WORLD as our communicator; therefore, the collective communication will include all processes.

MPI\_Barrier(comm)

This function creates a barrier synchronization in a commmunicator(MPI\_COMM\_WORLD). Each task waits at MPI\_Barrier call until all other tasks in the communicator reach the same MPI\_Barrier call.

MPI\_Bcast(&message, int count, MPI\_Datatype datatype, int root, comm)

This function displays the message to all other processes in MPI\_COMM\_WORLD from the process whose rank is root.

MPI\_Reduce(&message,&receivemessage,int count,MPI\_Datatype datatype,MPI\_Op op, int root, comm)

This function applies a reduction operation on all tasks in MPI\_COMM\_WORLD and reduces results from each process into one value. MPI\_Op includes for example, MPI\_MAX, MPI\_MIN, MPI\_PROD, and MPI\_SUM, etc.

MPI\_Scatter(&message, int count, MPI\_Datatype,&receivemessage, int count, MPI\_Datatype, int root, comm)MPI\_Gather(&message, int count, MPI\_Datatype,&receivemessage, int count, MPI\_Datatype, int root, comm)

## Software application program interface (API) architecture

The API is the interface by which an application program accesses the operating system and other services. An API is defined at source code level and provides a level of abstraction between the application and the kernel to ensure the portability of the code.

An API can also provide an interface between a high level language and lower level utilities and services which were written without consideration for the calling conventions supported by compiled languages. In this case, the API’s main task may be the translation of parameter lists from one format to another and the interpretation of call-by-value and call-by-reference arguments in one or both

directions.

Parallel architecture may have:

Messages: May be implemented both for SMP and cluster machines; they require copying of data between nodes. Latency and the speed of this operation are the limiting factors. The advantage of writing code using messages on an SMP machine is that if you decide to move to a cluster, you can easily add a machine without have to recreate your application from scratch.

Threads: They are designed to work fast on SMP machines because of shared memory. They do not need copying as much as messages do. The disadvantage is that you cannot extend them beyond one SMP machine because data is shared between CPUs. NUMA may allow that, but we are not considering NUMA architecture at present due to its expense and because Linux is not yet mature on that platform.

## Batch and parallel jobs

You can have two kinds of jobs:

a batch job and

a parallel job.

A batch job is an application that runs independently on each node of the cluster without the need to interact deeply with the other nodes. This kind of application usually performs calculations on a set of data and at the end of the job passes the result to a control node responsible for collecting the results from all the nodes.

A parallel job is a job that needs to talk with the other compute nodes of the cluster. This operation needs message passing and network activity. The reason for using a cluster is to speed up the execution of a program, but sometimes SMP machines that are not in a cluster can be fast enough. With this in mind, we should consider using a cluster in the following circumstances:

* Problem size: If an application requires a lot of memory, a workstation may not be enough; a cluster might be a good solution.
* Real time answer: Problems that need a faster response. In this case, you should also consider the number of users who access the cluster at the same time. If too many users share the cluster resources at the same time, the cluster performance will slow down and maybe a workstation could be an adequate solution.
* Is the application suitable for a cluster or do I need to port it?
* How long does the application take to run?
* How large is the application? Can I fit it onto any machine?
* If at this point we have decided to use a cluster and to use parallel computing, so we need to create our application.

# Works Cited

A search on the WWW for "Beowulf clustering" or "parallel computing" will yield a wide variety of information.

<https://help.ubuntu.com/community/MpichCluster>

<http://byobu.info/article/Building_a_simple_Beowulf_cluster_with_Ubuntu/>

<https://www.linux.com/community/blogs/133-general-linux/9401>

<https://www.linux.com/community/blogs/133-general-linux/9401>

<http://yclept.ucdavis.edu/Beowulf/aboutbeowulf.html>

<http://cs.boisestate.edu/~amit/research/beowulf/beowulf-setup.pdf>

<http://www.slideshare.net/ankitmahato/hpc-workshop>

Linux HPC cluster installation, IBM Redbook

How to Build a Beowulf, by Thomas L. Sterling, et al

<http://mpitutorial.com/>

<https://computing.llnl.gov/tutorials/mpi/>

<https://computing.llnl.gov/tutorials/parallel_comp/>

<http://climate.ornl.gov/~forrest/osdj-2000-11/>

<https://computing.llnl.gov/tutorials/linux_clusters/>

<http://www.slideshare.net/mchaves/c1pcmpishort>

**Things to read**

xCAT

IBM director

Ganglia

MAUI scheduler

ALICE  
NFS vs AFS

Portland group compilers

CSM

Total view

Blast, starCD, IS dyna

Star cluster – MIT

PBS+LSF

PAPi

FPING

POV –RAY