

Breaking RSA
Using
Multiple Polynomial Quadratic Sieve (MPQS)
Using High Performance Clusters



Submitted by:

Muhammad Sami Khan

Supervisor:

Dr Athar Mahboob

Thesis

Submitted to

Department of Electronic and Power Engineering,
College of Marine Engineering (PNEC), Karachi
National University of Sciences and Technology, Islamabad

In partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING
With Specialization in Communications

March 2012

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

In the name of ALLAH,
the most Beneficent, the most Merciful

Abstract

High performance clusters are one of the emerging technologies and have been used in modern era to deal with challenging problems in the field of information security and breaking of cryptographic algorithms. Factorization of integers is a very difficult task which forms the foundation of security of many encryption algorithms, for example RSA algorithm. The largest integer factorized to date is of 232 decimal digits. We present results of our work on setting up single server image clusters using Kerrighed SSI based cluster system. We have attempted to use Multiple Polynomial Quadratic Sieve [MPQS] algorithm for integer factorization. This clustering recipe and the results presented shall be useful to others who are planning to setup Linux clusters (Kerrighed) for high-performance computing.

We have factored a 100 decimal digit integer from RSA Factoring Challenge by using MPQS algorithm implementation running on our 20 node Kerrighed SSI cluster within seven hours. The estimated price of our cluster is less than USD 5,000. We have by far improved upon the original RSA-100 cracking done in 1991 using MPQS on Massively Parallel [MasPar] computers. The estimated price of MasPar was close to a million USD for 16,384 processor elements and it took several days to factor RSA-100.

Acknowledgements

First and foremost I would like to offer my sincerest gratitude to Almighty ALLAH for his kindness and blessings that he has always bestowed upon me. I pray that, ALLAH shower his countless blessings and peace upon his most beloved and last Prophet Hazrat Muhammad Mustafa (S.A.W).

I would also like to express my special thank to my parents, brother and other beloved ones for supporting me throughout my study and showing great patience during this research work.

After that, I would like to express my sincere gratitude to my supervisor Dr. Athar Mahboob for his kind and continuous guidance with great patience and knowledge. I can not express in words the support, encouragement and motivation he has provided throughout the work and without him it could be very difficult to complete the work and achieve desired results.

At last but not least, I would like to thank all of Guidance Committee members.

- Dr. Pervaiz Akhter
- Dr. Vali Uddin
- Dr. Arshad Aziz

CONTENTS

1	INTRODUCTION	1
2	LINUX OPERATING SYSTEM & EVOLUTION.....	3
2.1	Types of Clusters	4
2.1.1	Load Balancing Clusters	4
2.1.2	Compute Clusters.....	5
2.1.3	Grid Clusters	5
2.1.4	Storage Clusters	5
2.1.5	Single Server Image Clusters.....	6
2.2	Limitations of Clusters.....	7
2.3	Types of Single Server Image Clusters.....	8
2.3.1	Mosix	8
2.3.2	OpenMosix.....	9
2.3.3	OpenSSI.....	9
2.3.4	Kerrighed	9
2.4	The Cluster Building Blocks.....	10
2.4.1	Cluster Nodes.....	11
2.4.2	Cluster Interconnect.....	12
2.4.3	The Cluster Boot Server.....	12
3	PUBLIC KEY CRYPTOGRAPHY & RSA.....	14
3.1	Advantages of Public-Key Cryptosystem.....	14
3.2	Disadvantages of Public-Key Cryptosystem.....	14
3.3	Public-Key Cryptosystems.....	14
3.4	RSA.....	14
3.4.1	Key Generation	15
3.4.2	Encryption.....	15
3.4.3	Decryption.....	15
3.5	RSA Numbers	16
4	MULTIPLE POLYNOMIAL QUADRATIC SIEVE (MPQS).....	17
4.1	Phases of Algorithm.....	17
4.2	The Algorithm.....	18
4.3	Partial relations and cycles.....	18
4.4	Producing a full relation.....	19
5	IMPLEMENTATION.....	20

5.1	General Description of Cluster Node Boot Procedure	20
5.2	Installation of Kerrighed Boot Server	22
5.2.1	DHCP Server	22
5.2.2	Setting up the TFTP Server and PXE bootloader	24
5.2.3	Installing Time Synchronization Server.....	26
5.2.4	Setting up the NFS Server and Creating the Root File System.....	26
5.2.5	Testing the diskless boot system.....	31
5.2.6	Building the Kerrighed Linux Kernel	32
5.2.7	Installing Kerrighed from the Source.....	33
5.2.9	Kerrighed Configuration	35
5.2.10	Starting Kerrighed.....	36
5.3	Testing Kerrighed	38
5.4	Application of Clusters to Solve the Integer Factorization Problem (IFP)	39
6	RESULTS.....	45
7	CONCLUSION	47
8	FUTURE WORK.....	48
9	REFERENCES	49
10	APPENDICES	50

LIST OF FIGURES

Figure 2.1 Trend of Operating System Share among Top 500 Super Computers	3
Figure 2.1.1 Load Balancing Clusters.....	5
Figure 2.1.4 Storage Clusters	6
Figure 2.1.5 Single Server Image Clusters	7
Figure 2.4 Generic View of Cluster Components.....	11
Figure 2.4.1 Components of a cluster Node.....	12
Figure 2.4.3 Cluster Boot Server Components	13
Figure 5.1 Interaction Between Cluster Nodes and Cluster Boot Server.....	20
Figure 5.4 CPU utilization of Kerrighed nodes at the time of Sieving	39
Figure 5.5 Overall Msieve process to factor 100 digit number.....	41
Figure 5.6 General Structure of Parallel Integer Factorization Program Using Clusters.....	42
Figure 5.7 Original problem is distributed among nineteen nodes to find 5000 relations	43
Figure 5.8 Multiple msieve processes performing sieving on the Kerrighed cluster.....	43
Figure 5.9 Official Kerrighed Registration	44
Figure 6.1: Bar Chart Performance of MPQS Algorithm on 19 Nodes Cluster.....	46

LIST OF TABLES

Table 2.3 Different types of SSI Clusters	8
Table 5.1 Time Complexity of Various Integer Factorization Algorithms	40
Table 6.1: Performance of MPQS Algorithm on 19 Nodes Cluster	45

1 INTRODUCTION

For more than 30 years Computer scalability and availability is a theory that has received much interest. The demand of high performance systems is increasing day by day and Clusters have supported this requirement. A Cluster reduces the cost of high performance computer hardware by combining existing low cost computers to form a single high performance computer that offers throughput matching that of expensive supercomputers. Linux operating System and high level languages has supported clusters by revolutionizing the software domain with open source kernel to support extensive variety of hardware platforms. Most of the clustering solutions use Linux as the operating system. With the help of portable code and well defined standards leads to increasingly efficient use of clusters.

The major domains where clusters are mostly used are as follows.

1) Business Domain

This domain requires load balancing and high availability clusters. Limited load is assigned to each cluster node in parallel to perform the execution of transaction.

2) Scientific Domain

This domain requires high performance clusters, which can handle large complex problems. The problem is then divided in smaller problems and distributed among a number of computational nodes for concurrent processing.

Clusters should be highly available and scalable to become accustomed to various sizes of problems. The availability of the system is one of the major requirements and cannot be ignored as it may result as a failure of the system. To overcome this problem auto failover techniques are used. "Hot plug" concept is introduced by Beowulf Cluster in which nodes can leave or join the cluster during the normal execution.

The hidden integration of resources and application and to present the target as single one big machine is achieved by using SSI (Single System Image). Kerrighed is one of the SSI Project which was launched in 2004. Kerrighed allows the program to be distributed among the nodes, shares all the physical memory into one combined shared memory which allows the processes to utilize more memory than available in a single computer. The migration of tasks and load balancing is done through Kerrighed scheduler.

This Thesis is organized as follows.

After a brief introduction we will review the Linux operating system in next chapter where the emergence of *Linux as an operating system* is defined, further we will move to the *Public key cryptographic & RSA algorithm* where its is discussed in detail. We have also included *Network Protocols* that are used in our thesis. The basic Idea of our thesis is to find prime factors of a given integer, for finding factors we have used Quadratic Sieve algorithm that is discussed in Chapter 4. In chapter 5 we have discussed the recipe of Kerrighed Cluster. In the last we have shown the results achieved using the Kerrighed cluster and MPQS algorithm.

2 LINUX OPERATING SYSTEM & EVOLUTION

Linux is a new implementation of the UNIX computer Operating System (or OS). Linux is an example of an OpenSource operating system. Linux is an example of an opensource operating system, which means that the source code is available free of cost.

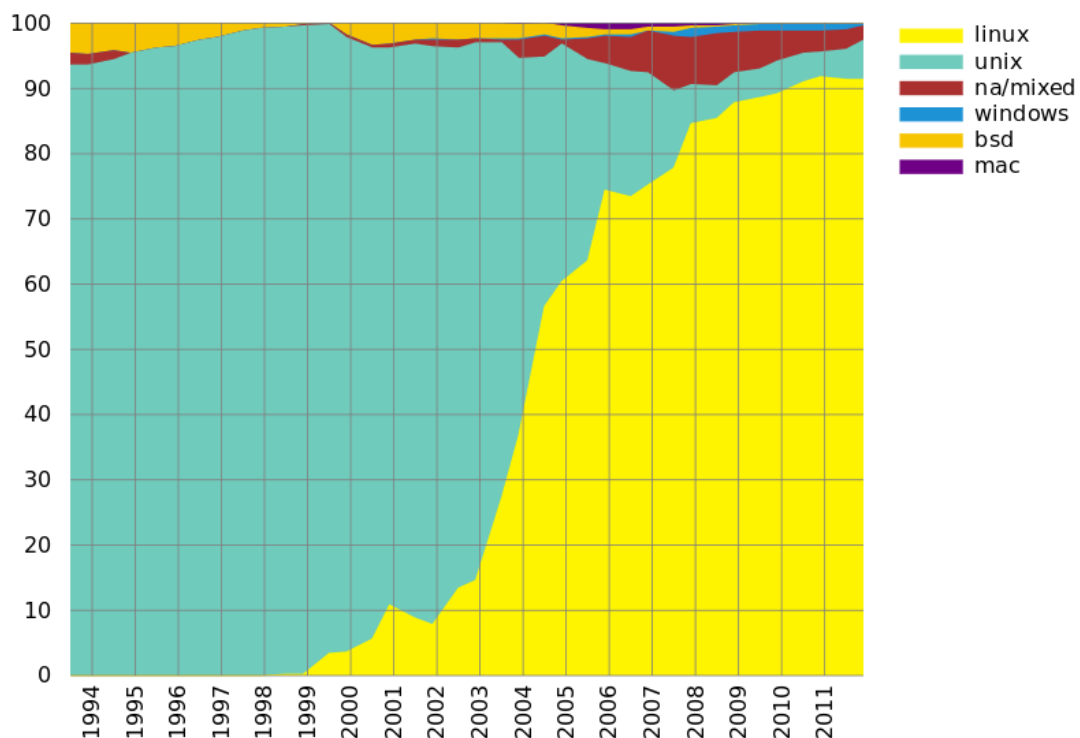


Figure 2.1 Trend of Operating System Share among Top 500 Super Computers

As the above figure shows the trend of Linux operating system used in top 500 super computers is exponentially increasing. The researchers and large organizations are now supporting and extensively using Linux because of reduced cost and opportunity of advancement. The operating system landscape has seen variety of changes during the last two decades. The rise of Linux as the pre-eminent and dominant Unix type of operating system is one such change. Some examples of advance Linux features are.

The amazing fact of Linux it is still actively used and the main reason behind this story is Linux kernel development community is strong and active. New features are being added into

the Linux Kernel at a very fast pace. All facets of the Linux operating system kernel have been developing steadily.

2.1 Types of Clusters

There are numerous types of clusters and each with its own properties. Several types of popular clusters and their uses are discussed below:

- 1) Load Balancing Clusters
- 2) Compute Clusters
- 3) Grid Clusters
- 4) Storage Clusters
- 5) Single Server Image Clusters

2.1.1 Load Balancing Clusters

Load balancing clusters work in such a way that front-end nodes accept the request from end user and the pass on the request to the back-end array server. The front end request that is received from the end user is distributed among multiple nodes based on a policy. The policy is maintained by the network administrator and is defined as per the requirement. For example a policy can be defined on each node to bear certain amount of load or if required it can dynamically adjust the load. An example of Linux Load balancing server is of LVS (Linux Virtual Server). The figure below shows how load balancing server works.

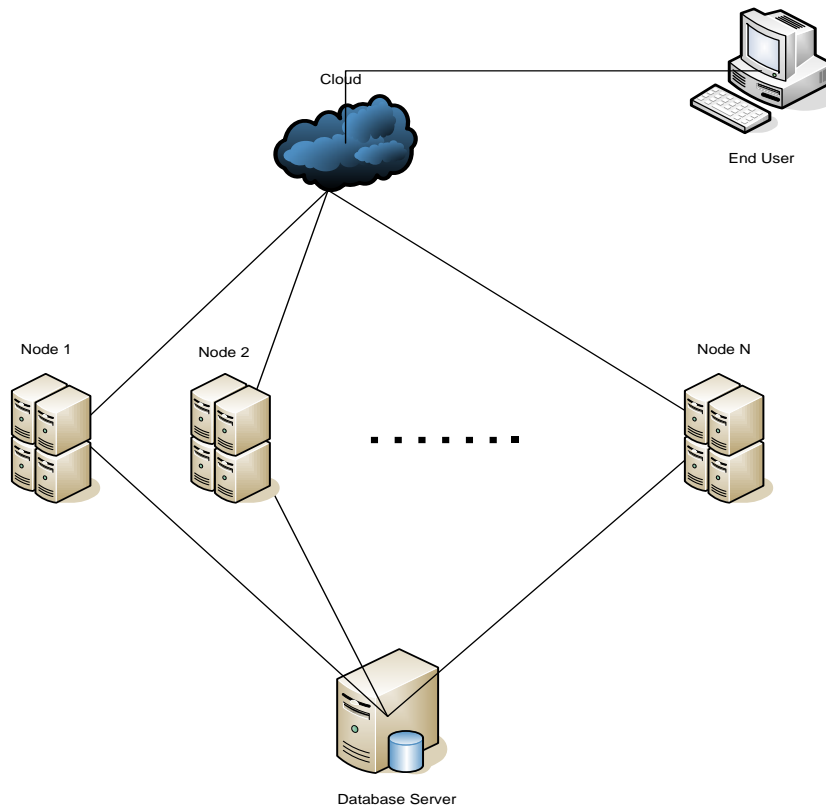


Figure 2.1.1 Load Balancing Clusters

2.1.2 Compute Clusters

The logic of compute clusters is based on MPI (Message passing interface), where each node runs its own operating system and its own physical memory and works by means of high level message passing. The message is passed among the nodes on a share conventional.

2.1.3 Grid Clusters

Grid Clusters are usually confused with Compute Clusters. Grid clusters allow sharing the computational power and data storage capacity over different networks or internet. This allows increasing the computing resources independent of where they are located. The characteristics include decentralization, distributed job management & Scheduling.

2.1.4 Storage Clusters

Redundant Array of Independent disks (RAID) technology is usually used for Storage Clusters. The objective of storage cluster is availability (by increasing redundancy), increase of disk space and speed and efficiency increase boundaries of storage area networks (SANs) to new levels. In Linux one form of storage cluster is Distributed Replicated Block device (DRDB). Storage clusters are built of storage server farms connected together that work on

similar tasks in a grid style. Storage clusters provides end-users with a virtual pool that can tolerate and recover from storage device failures.

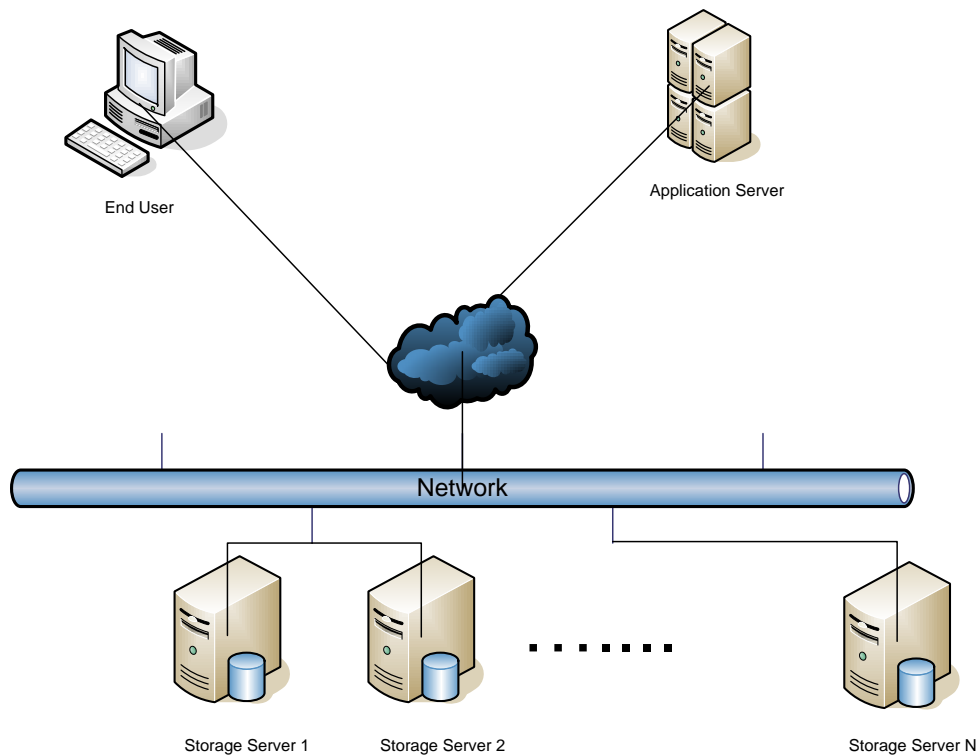


Figure 2.1.4 Storage Clusters

2.1.5 Single Server Image Clusters

In single Server Image (SSI) Clusters multiple nodes of computers combined to form a single large computer. This kind of cluster makes an application to view multiple computers as a single node. The program is divided into multiple processes and distributed among the nodes for resource balancing. The memory space and the process space of the entire cluster are merged. Almost all SSI Clusters provide process migration.

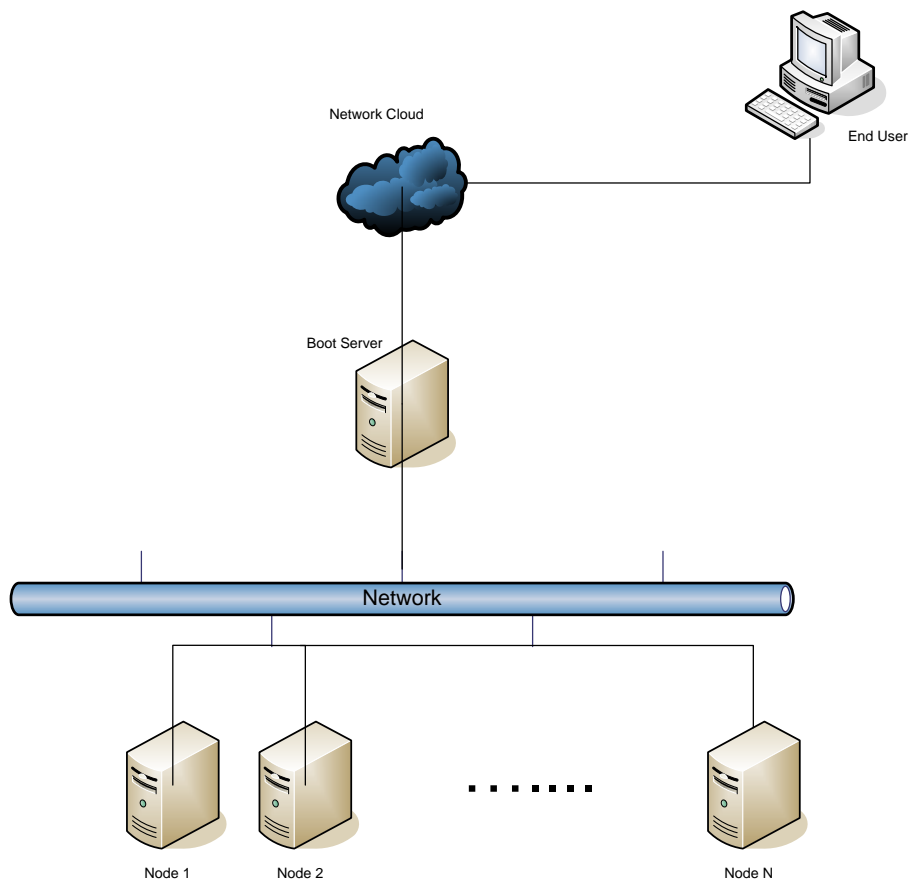


Figure 2.1.5 Single Server Image Clusters

2.2 Limitations of Clusters

As everything in this world has its own limitation, same applies to cluster. Some people thinks if a single computer takes 10 seconds to execute a problem then two computers would take 5 seconds to execute the same problem, but this is not the case in actual as adding computers in a clusters have overheads and even in some rare cases it decreases the actual performance of a computer. Following issues tells why we cannot achieve ideal speed using clusters.

- Parallelization in a program results in a considerable overhead. The overhead can be network as data is transferred among the nodes which takes sometime for message passing.
- Final combined result is dependent until all the nodes complete their assignment. If say for example there are 3 nodes in which one is comparatively less computationally powered then for final result cannot be compiled until the less computationally powered system completes its task.

- For simpler & small programs Clusters can take more time to execute as compare to single computer.
- Clusters are not made for every kind of problems. Limited areas of application can benefit from Clusters.
- SSI Clusters does not allow thread migration
- If a program cannot be break up process wise then Clusters will be of no use as the execution will be performed in Serial.

2.3 Types of Single Server Image Clusters

The major benefit of using Single Server Image (SSI) clusters is that they provide an well-designed view of the cluster as a single machine. In practical this is very hard to achieve given the variety of hardware and latencies of interconnected-nodes communication. Below figure shows the list of SSI clusters.

Table 2.3 Different types of SSI Clusters

Name	Process migration	Process checkpoint	Single process space	Single root
Amoeba	Yes	Yes	Yes	Yes
AIX TFC	Unknown	Unknown	Unknown	Yes
Kerrighed	Yes	Yes	Yes	Yes
LinuxPMI	Yes	Yes	No	Yes
LOCUS	Unknown	Unknown	Yes	Yes
MOSIX	Yes	Yes	No	Yes
openMosix	Yes	Yes	No	Yes
Open-Sharedroot	No	No	No	Yes
OpenSSI	Yes	No	Yes	Yes
VMScluster	No	No	Yes	Yes
Plan 9	No	No	No	Yes
Sprite	Yes	Unknown	No	Yes
TruCluster	No	Unknown	No	Yes

Below are some of the SSI clusters and features available as listed in the above figure.

2.3.1 Mosix

Mosix is a clustering project that pre-dates Linux and was developed for UNIX type of operating systems. MOSIX makes frequent releases and is keeping pace with mainstream

Linux kernel by releasing its versions based on current kernel releases. However, restrictive nature of its software licensing keeps its user base limited and keeps it out of open source category.

2.3.2 OpenMosix

openMosix was an attempt of the Linux MOSIX community to develop a community based version of MOSIX. However openMosix remained stuck with Linux Kernel version 2.4 and could not adapt to the version 2.6. Eventually the maintainer of openMosix announced plans to end the openMosix project effective March 1, 2008, claiming that "the increasing power and availability of low cost multi-core processors is rapidly making single-system image (SSI) clustering less of a factor in computing." Due to this reason openMosix is of historical interest only.

2.3.3 OpenSSI

OpenSSI has been an ambitious Linux clustering project [OpenSSI 2009]. The aim of the project was to avoid re-inventing the wheel and combine existing Linux clustering related projects in such a way so as to create the SSI cluster. Unfortunately the project has run out of steam. After a few years of active work only a few contributors remain committed to the project. Originally the project had a strong commitment and support from Hewlett-Packard. The last major release was made in 2006.

2.3.4 Kerrighed

The Kerrighed project was launched at the French National Research Laboratory (INRIA) in 1998. The project has thus been in existence for over a decade and may be regarded as mature. The project is still releasing newer version and adding features with every release. The current release is called Kerrighed 2 series. The Kerrighed 1 series had many more features but was not a stable platform. The Kerrighed 2 series removed many of the experimental features and is slowly adding back features as they are stabilized [Kerrighed 2009]. Admittedly the Kerrighed user community is also small.

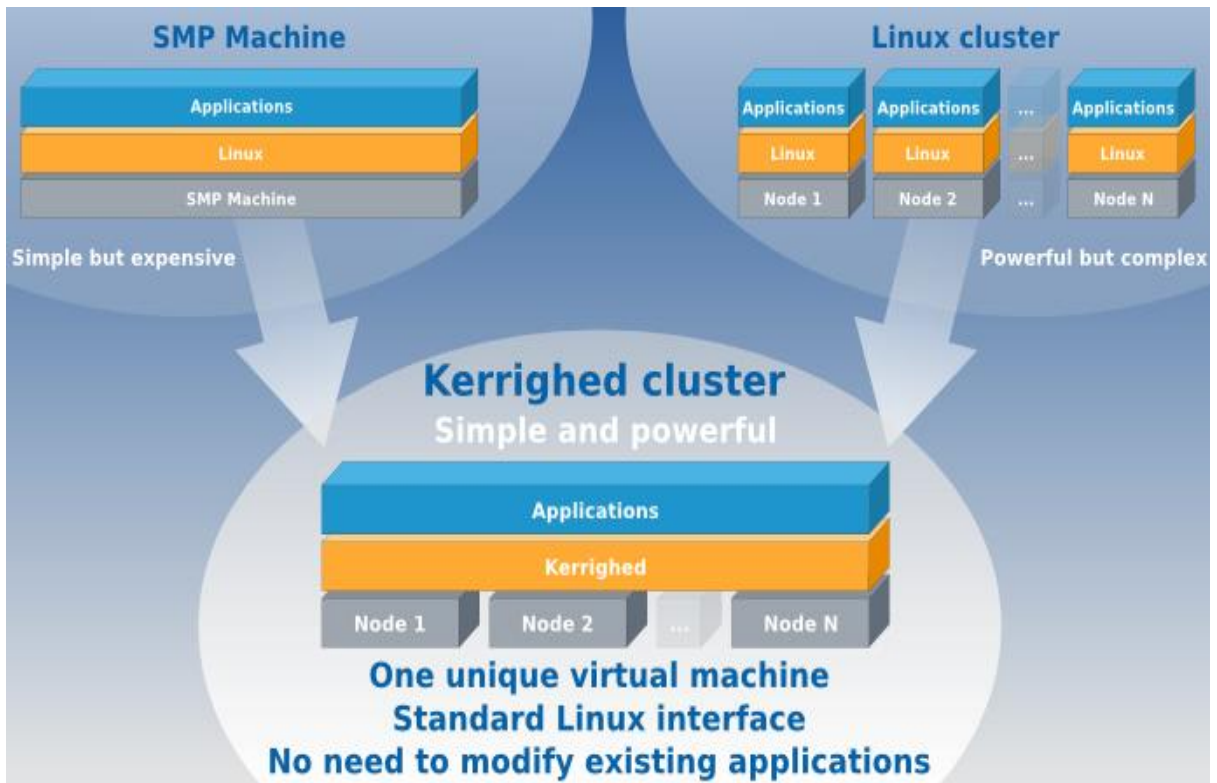


Figure 2.3 Kerrighed Cluster [Kerrighed 2012]

2.4 The Cluster Building Blocks

Now we review the hardware and software required for cluster in a generic fashion. We discuss the master or boot server node, the cluster interconnect, the cluster compute or member nodes, the cluster operating system software, the middleware libraries such as Kerrighed libraries, etc. In figure 5 we show the generic view of the cluster nodes, the interconnect and the cluster boot server.

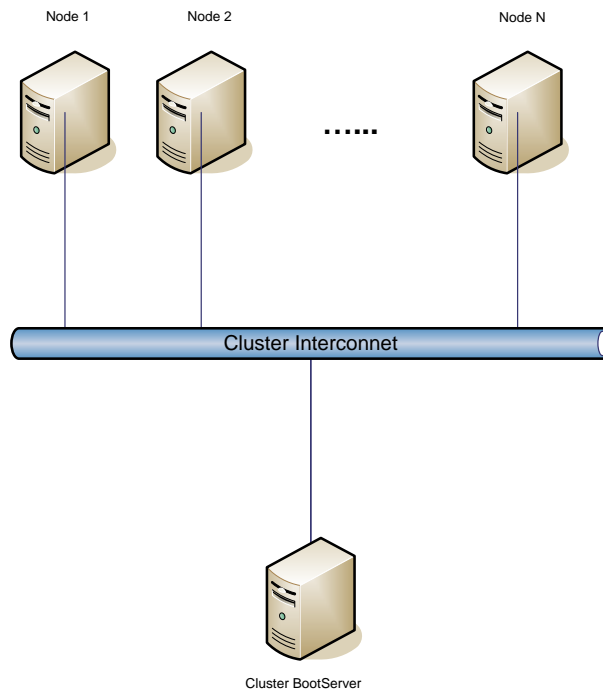


Figure 2.4 Generic View of Cluster Components

2.4.1 Cluster Nodes

The cluster node itself is made up of a number of components. These components are shown in the figure below. The hardware components consist of the network interface, the system bus within the node through which all of the data moves while traveling from its CPU and memory to the network and so on. The node CPU is in control of the hardware resources of the node and runs an instance of the operating system supporting clustering. In the case of SSI clusters each node runs the same operating system image in every respect.

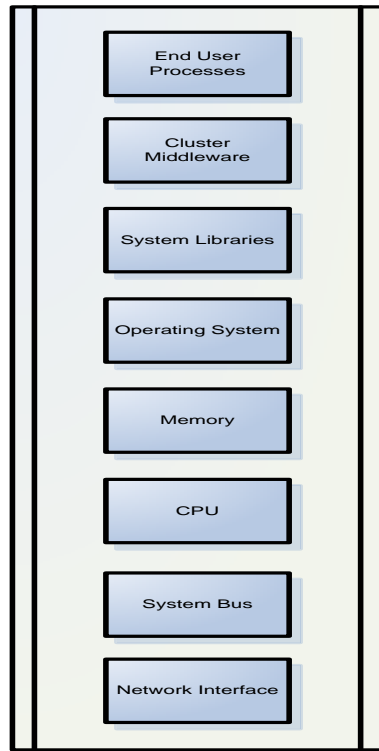


Figure 2.4.1 Components of a cluster Node

System libraries include the standard C and C++ libraries and any application libraries that may be needed. In SSI cluster the entire software stack is common to all nodes and is generally served as a network mounted root filesystem. In addition, libraries specific to cluster middle-ware may also be present.

2.4.2 Cluster Interconnect

The cluster nodes perform the inter-node communication using the interconnection network. One of the major limitations of the clustering technology is the speed and latency of the interconnection network. For those with limited budgets interconnection using Gigabit Ethernet is an appropriate choice. For the performance conscious clusters Infiniband is the technology of choice for cluster interconnect. Infiniband can give data throughputs reaching 100 Gbps with extremely low latencies.

2.4.3 The Cluster Boot Server

In order to allow the cluster nodes to easily load the required software we use the Cluster Boot Server as the solution. In this solution it is easy to add cluster nodes as no installation is required on the cluster nodes themselves. The cluster boot server itself is not part of the single server image cluster as its role is only to facilitate the cluster nodes in accessing and loading their software and configuration. For the sake of simplicity we can run the cluster boot

server by using one of the standard modern Linux distributions such as Ubuntu, Suse or Redhat Linux. The essential components of the cluster boot server which require proper configuration are shown in the figure 2.4.3.

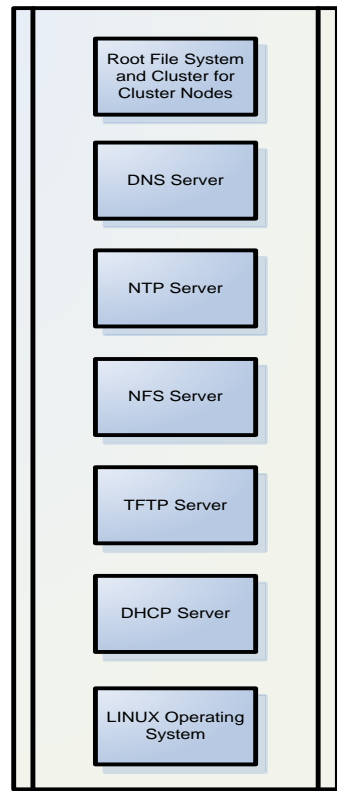


Figure 2.4.3 Cluster Boot Server Components

The cluster boot server performs an important role by providing the infrastructure software to all cluster nodes. It ensures that identical operating system software and libraries are made available to cluster nodes. It also functions as the cluster management and configuration management component of the cluster.

3 PUBLIC KEY CRYPTOGRAPHY & RSA

Public-Key Cryptography includes two types of keys, a private key and a public key, associated that needs to authenticate its identity by electronic means or to sign or encrypt data.

3.1 Advantages of Public-Key Cryptosystem

The public-key resolved the severe issues of secret key cryptography which involves:

- i) Key distribution Problem
- ii) Authenticity (Digital Signature)

3.2 Disadvantages of Public-Key Cryptosystem

The main disadvantage of using public-key cryptography for encryption is its speed. Due to complex computational requirements of very large numbers generally public-key systems are much slower than secret-key systems. These hybrid systems get the security advantages of public-key systems and the speed advantages of secret-key systems. The public-key system is utilized to generate and verify the digital signature and to establish a common secret key. The secret-key system is utilized for bulk encryption.

3.3 Public-Key Cryptosystems

Some popular public-key algorithms utilize long number modular exponentiation are:

- RSA
- Diffie-Hellman Key Exchange
- Digital Signature Algorithm, DSA

We will only be discussing RSA in this report. Other security algorithms are out of scope.

3.4 RSA

RSA is a block cipher scheme, It gets its name from names of authors Rivest-Shamir-Adleman (RSA). Since the publication, RSA scheme renowned as the most widely accepted and implemented general-purpose approach to public-key encryption. RSA security rely on a hard problem known as IFP (Integration Factorization Problem). It involves

computation complexity of factoring very large number. Following sections will explain the Key generation, Encryption and Decryption processes of RSA.

3.4.1 Key Generation

The keys for the RSA algorithm are generated the following way

1. Choose two large random prime numbers p and q
2. Calculate $n = p \times q$
3. Calculate $\varphi(n) = (p - 1) \times (q - 1)$
4. Choose d such that d is co-prime with $\varphi(n)$, i.e. $\gcd(d, \varphi(n)) = 1$
5. Calculate $e = d^{-1} \pmod{\varphi(n)}$
6. Publish $\{e, n\}$ as the public key and keep d secret as the private key.

3.4.2 Encryption

Alice transmits her public key (n, e) to Bob and keeps the private key secret [RSA 2009]. Bob then wishes to send message \mathbf{M} to Alice.

He first turns \mathbf{M} into an integer m , such that $0 < m < n$ by using an agreed-upon reversible protocol known as a padding scheme. He then computes the ciphertext c corresponding to

$$c = m^e \pmod{n}.$$

This can be done quickly using the method of exponentiation by squaring. Bob then transmits c to Alice.

Note that at least nine values of m will yield a ciphertext c equal to m , But this is very unlikely to occur in practice.

3.4.3 Decryption

Alice can recover m from c by using her private key exponent d via computing

$$m = c^d \pmod{n}.$$

Given m , she can recover the original message \mathbf{M} by reversing the padding scheme.

(In practice, there are more efficient methods of calculating c^d using the pre computed values below.)

3.5 RSA Numbers

Table 3.1 RSA Challenge numbers and date on which it was factored[RSANUM 2008]

Number	Digits	Factored (references)
RSA-100	100	Apr. 1991
RSA-110	110	Apr. 1992
RSA-120	120	Jun. 1993
RSA-129	129	Apr. 1994 (Leutwyler 1994, Cipra 1995)
RSA-130	130	Apr. 10, 1996
RSA-140	140	Feb. 2, 1999 (te Riele 1999a)
RSA-150	150	Apr. 6, 2004 (Aoki 2004)
RSA-155	155	Aug. 22, 1999 (te Riele 1999b, Peterson 1999)
RSA-160	160	Apr. 1, 2003 (Bahr et al. 2003)
RSA-200	200	May 9, 2005 (see Weisstein 2005a)
RSA-576	174	Dec. 3, 2003 (Franke 2003; see Weisstein 2003)
RSA-640	193	Nov. 4, 2005 (see Weisstein 2005b)
RSA-704	212	
RSA-768	232	Dec. 12, 2009 (Kleinjung 2010, Kleinjung et al. 2010)
RSA-896	270	
RSA-024	309	
RSA-536	463	
RSA-048	617	

4 MULTIPLE POLYNOMIAL QUADRATIC SIEVE (MPQS)

In 1981 Carl Pomerance invented a factorization algorithm for integers known as quadratic sieve which is actually an improvement to Schroepel's linear sieve algorithm [QS 2012]. It is considered to be the fastest algorithm for integer's factorization under 100 decimal digits and second fastest for above 100 or above digit number. It is a simple algorithm and its run time solely depends on the size of the integer that has to be factored.

4.1 Phases of Algorithm

MPQS algorithm is divided into two phases:

i) Data Collection

In this phase the algorithm calculates congruence of squares. This step can easily be parallelized and efficiently works on different nodes if multi-core processors or a cluster is available. After calculating these results are combined together.

ii) Data Processing

In this phase the algorithm calculates congruence of squares by pulling all the data collected in the data collection phase. This step cannot easily be parallelized and does not efficiently work on different nodes as it requires a huge amount of physical memory.

The QS algorithm sets up a congruence of squares modulo n (where n is the integer that has to be factored). The raw approach to finding a congruence of squares is to select a random number, square the number and assume that the smallest non-negative remainder modulo n is a perfect square in the integers. For example, $80^2 \bmod 5959$ is equal to 441, which is the square of 21. This congruence of squares can be accomplished by this algorithm only rarely for large n . This is roughly the basis of Fermat's factorization method.

The quadratic sieve is a modified form of Dixon's factorization method.

Normally the time required to factor n digit number using quadratic sieve equals

$$e^{(1+o(1))\sqrt{\log n \log \log n}} = L_n \left[\frac{1}{2}, 1 \right]$$

The e is the constant used as the base of the logarithm.

4.2 The Algorithm

The quadratic sieve algorithm is listed below.

1. Select a smoothness bound B [QSS 2009]. The number $\pi(B)$, denotes the number of prime numbers less than B , will controls the number of vectors needed and the length of the vectors.
2. Use sieving to find $\pi(B) + 1$ numbers a_i such that $b_i = (a_i^2 \bmod n)$ is B -smooth.
3. b_i is factored and generate exponent vectors mod 2 for each one.
4. Find a subset of these vectors which add to the zero vector. Multiply the corresponding a_i together naming the result mod n : a and the b_i together which yields a B -smooth square b^2 .
5. The only thing left is equality $a^2 = b^2 \bmod n$ from which can be achieved by two square roots of $(a^2 \bmod n)$, one by taking the square root in the integers of b^2 namely b , and the other the a computed in step 4.
6. The desired identity is achieved i.e. $(a + b)(a - b) = 0 \pmod{n}$. Calculate the Greatest Common Divisor of n with the difference (or sum) of a and b . This produces a factor, it might be a trivial factor (n or 1). If the factor is trivial, try again with different a , that was used before.

4.3 Partial relations and cycles

If some of the relations $y(x)$ are not smooth, it may be possible to combine two of these *partial relations* to form a full relation, if the two y 's are products of the same prime(s) outside the factor base. [Note that this is equivalent to extending the factor base.] For example, if the factor base is $\{2, 3, 5, 7\}$ and $n = 91$, there are partial relations:

$$21^2 \equiv 7^1 \cdot 11 \pmod{91}$$

$$29^2 \equiv 2^1 \cdot 11 \pmod{91}$$

Multiply these together:

$$(21 \cdot 29)^2 \equiv 2^1 \cdot 7^1 \cdot 11^2 \pmod{91}$$

and multiply both sides by $(11^{-1})^2$ modulo 91. 11^{-1} modulo 91 is 58, so:

$$(58.21.29)^2 \equiv 2^1 \cdot 7^1 \pmod{91}$$

$$14^2 \equiv 2^1 \cdot 7^1 \pmod{91}$$

4.4 Producing a full relation

A relation that is formed by combining two partial relations is called a *cycle*. Sometimes, forming a cycle from two partial relations leads directly to a congruence of squares, but it happens very rarely.

5 IMPLEMENTATION

In this chapter we explain our recipe of building a Kerrighed cluster. This recipe is further enhanced and free from errors that was written earlier [EUC 2011]. We first provide a general description of the cluster node boot procedure and then the specific commands to enable the installation and proper configuration of the required services.

5.1 General Description of Cluster Node Boot Procedure

To make the cluster node operations fault free and plug & play the cluster node must be booted over the network. Most modern PC motherboards come ready with the capability to boot their operating system over the network using the technology called PXE (Pre-boot eXecution Environment) within the BIOS firmware. The boot setting is done in BIOS and priority of network boot is increased. The following steps are taken place when a cluster node starts.

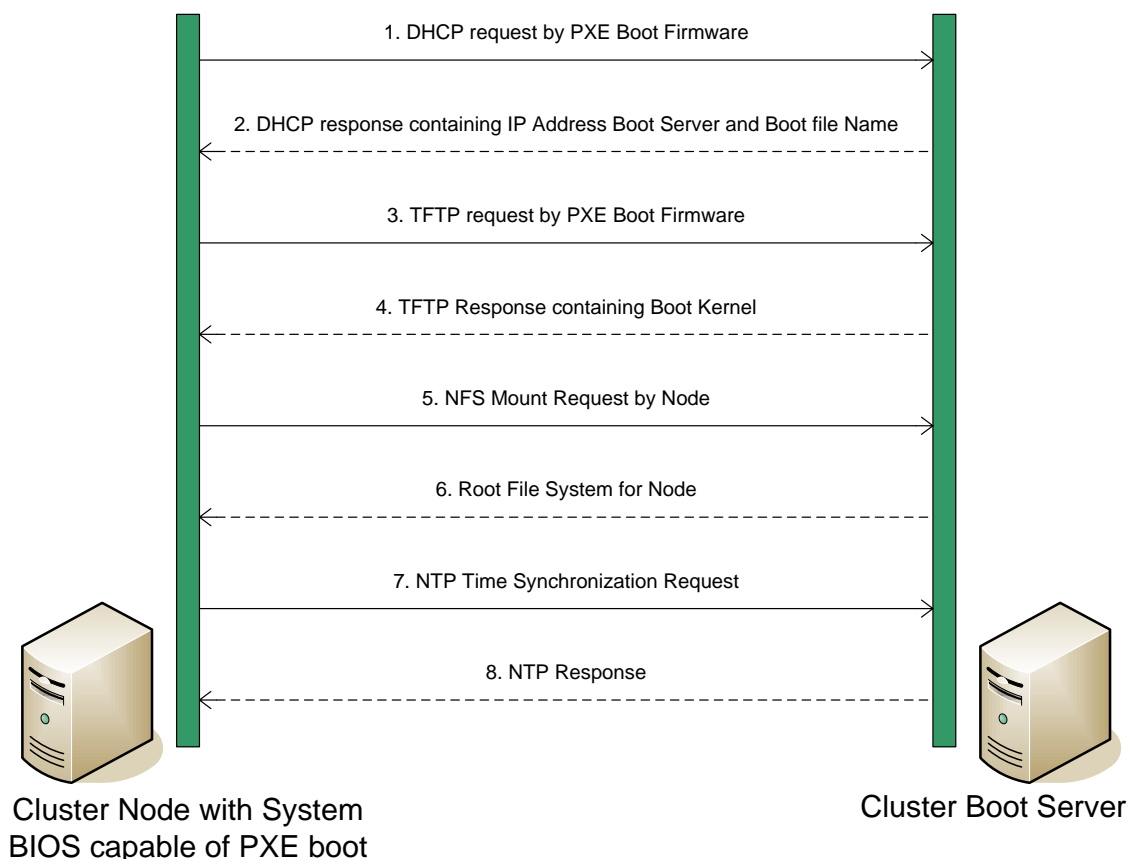


Figure 5.1 Interaction Between Cluster Nodes and Cluster Boot Server

Once any cluster node boots it sends a Dynamic Host Configuration Protocol (DHCP) broadcast requesting TCP/IP configuration. The cluster boot server is running a properly configured instance of DHCP server service and responds with the following parameters:

- Node IP Address
- Subnet Mask
- Default Gateway
- Domain Name
- DNS Server
- Boot Server
- Boot Image Name

On getting the DHCP response the PXE firmware configures its IP address and then sends a Trivial File Transfer Protocol (TFTP) request to the Boot Server identified in the DHCP response. The TFTP request is made for the file Boot Image Name provided to the node. In general, the boot image can be different for each node. However, in our case since it is a Single System Image cluster therefore each node is provided with the same boot file. The boot file is actually a properly configured and compiled Linux kernel instance supporting all the required hardware and system capabilities required by the cluster nodes. It should be noted that we need to record the MAC address of NIC of each cluster node before hand so that it can be assigned a deterministic TCP/IP configuration.

After the Linux kernel is booted the root filesystem is mounted over NFS. The Kernel boot parameters are configured as shown in the following example:

```
KERNEL vmlinuz-2.6.20
```

```
APPEND root=/dev/nfs nfsroot=10.1.1.111:nfsroot ip=dhcp rw
```

It should be noted that a second DHCP configuration request is made by the same node, this time it is made by the TCP/IP networking stack in the Linux kernel. Once the root filesystem is mounted over NFS standard system initialization is done using the init scripts and the remaining services are properly started. These would include the Secure Shell server allowing us to remotely log on to any one of the cluster nodes to initiate jobs for the cluster. Another

useful service would be the NTP client service. Each cluster node is able to synchronize its clock with that of the boot server using Network Time Protocol (NTP).

5.2 Installation of Kerrighed Boot Server

Following section will provide specific commands to enable the installation of Boot Server software components on Ubuntu Linux 10.04 distribution.

5.2.1 DHCP Server

The command to install DHCP server on Linux environment is

```
$ sudo aptitude install dhcp3-server
```

Sudo package is used for super user account also known as root. To execute a command as an administrator sudo is typed before the command.

To configure the DHCP server following steps has to be performed.

1. The file `/etc/default/dhcp3-server` contains the ethernet card to listen to DHCP requests. In our case it is `eth0` so the file should read:

```
# /etc/default/dhcp3-server #  
interfaces="eth0"
```

2. After setting up the Ethernet card edit the following file

```
$ nano /etc/dhcp3/dhcpd.conf
```

Nano is a text editor used to open a file. After opening it looks like the following:

```
# /etc/dhcp3/dhcpd.conf #  
# General options  
option dhcp-max-message-size 2048;  
use-host-decl-names on;  
deny unknown-clients;  
deny bootp;  
  
# DNS settings  
option domain-name "kerrighed";  
option domain-name-servers 10.1.1.254;
```

Information about the network setup

```
subnet 10.1.1.0 netmask 255.255.255.0 {  
    option routers 10.1.1.254;  
    option broadcast-address 10.1.1.255; # Broadcast address for your network.  
}
```

Declaring IP addresses for nodes and PXE info

```
group {  
    filename "pxelinux.0";          # location of PXE bootloader.  
    option root-path "10.1.1.111:/nfsroot/kerrighed"; # Location of the bootable filesystem on  
NFS server  
    host kerrighednode1 {  
        fixed-address 10.1.1.221;    # IP address for kerrighednode1.  
        hardware ethernet 01:2D:61:AB:17:86; # MAC address of the kerrighednode1's  
ethernet adapter  
    }
```

```
    host kerrighednode2 {  
        fixed-address 10.1.1.222;    # IP address for kerrighednode2.  
        hardware ethernet 01:2D:61:AC:17:87; # MAC address of the kerrighednode2's  
ethernet adapter  
    }
```

... additional nodes 17 nodes to be added....

```
    host kerrighednode19 {  
        fixed-address 10.1.1.239;    # IP address for kerrighednode2.  
        hardware ethernet 01:2D:61:AC:62:12; # MAC address of the kerrighednode2's  
ethernet adapter  
    }
```

```
server-name "bootserver"; # Name of the PXE server
```

```
next-server 10.1.1.111;    # The IP address of the dhcp/tftp/nfs server
}
```

3. In order for configuration changes to take effect the DHCP server can be restarted by issuing the following command.

```
$ sudo /etc/init.d/dhcp3-server restart
```

4. In addition we would like the DHCP server to automatically start when the system is booted up. Following command makes the appropriate symbolic links in the System V type init setup:

```
$ sudo update-rc.d dhcp3-server defaults
```

5.2.2 Setting up the TFTP Server and PXE bootloader

The operating system boot image is served by the TFTP server. The TFTP server package can be installed by issuing the command:

```
$ sudo aptitude install tftpd-hpa
```

The settings for TFTP server are done by editing the file

```
$ nano /etc/default/tftpd-hpa and making sure it looks like the following:
```

```
# /etc/default/tftp-hpa #
```

```
# Defaults for tftp-hpa
```

```
RUN_DAEMON="YES"
```

```
OPTIONS="-l -s /var/lib/tftpboot" To enable network booting of clients we install syslinux  
bootloader and copy the PXE bootloader code to the tftp server directory.
```

```
$ sudo aptitude install syslinux
```

```
$ sudo cp /usr/lib/syslinux/pxelinux.0 /var/lib/tftpboot/
```

We create a directory to store the default configuration for all the nodes

```
$ sudo mkdir /var/lib/tftpboot/pxelinux.cfg
```

Create the file `/var/lib/tftpboot/pxelinux.cfg/default` and add the following default configuration for the nodes to boot:

LABEL linux

KERNEL vmlinuz-<KERNEL_VERSION>

APPEND root=/dev/nfs initrd=initrd.img-<KERNEL_VERSION>

nfsroot=10.1.1.111:/nfsroot/kerrighed ip=dhcp rw

Copy a standard Linux kernel to /var/lib/tftpboot/ in order to test the diskless-boot system. Replace <KERNEL_VERSION> with whatever you are using. For example, you could use the same Kernel that comes with the standard Ubuntu 10.04 installation that is being used on the boot server. This would assume that the cluster nodes support the same hardware architecture that the boot server is using. The issue here is to ensure that if a 64-bit Linux kernel is being used on the boot server that cluster nodes should have a 64-bit processor to run the kernel. The case of 32 bit kernel does not cause any problems as 64 bit processors from Intel family can run both 32 bit and 64 bit versions of Linux.

\$ sudo cp /boot/vmlinuz-<KERNEL_VERSION> /var/lib/tftpboot/

Alternatively the kernel version can be found by using the command:

\$ uname -r

To enable the use of NFS based root file system it is necessary to change the initrd by performing the following steps:

5. Install the initramfs-tools by issuing the command:

\$ aptitude install initramfs-tools

6. Configure initramfs-tools so that the initial ramdisk (initrd) is correctly built to support NFS mounted root filesystem. By default the initrd supports the use of local devices (hard disks, USB, CD-ROM, etc.) for the root file system. Edit the file /etc/initramfs-tools/initramfs.conf:

#/etc/initramfs-tools/initramfs.conf#

NFS Section of the config

BOOT: [local | nfs]

nfs - Boot using an NFS drive as the root of the drive.

Change

BOOT=local

to

BOOT=nfs

7. Generate the new initrd by issuing the command

```
$ mkinitramfs -o /tmp/student/initrd.img -r `uname -r` -k
```

8. Finally, copy the initial ramdisk image to /boot

```
$ cp /tmp/student/initrd.img -r `uname -r` /var/lib/tftpboot/
```

Where student is the login username of Linux OS.

5.2.3 Installing Time Synchronization Server

The time of all the nodes should be same as that of boot server. To achieve this NTP has to be installed on bootserver, since cluster nodes will be sharing the root file system and other files therefore the time should be synchronized. Otherwise it might through some errors at the booting time of cluster nodes. The NTP package can be installed and configured on the boot server as follows:

1. *Install NTP software package:*

```
$ sudo aptitude install ntp
```

2. Adjust the configuration file /etc/ntp.conf as follows to allow cluster nodes to synchronize their time:

```
# /etc/ntp.conf
```

```
restrict 10.1.1.0 mask 255.255.255.0 nomodify notrap
```

5.2.4 Setting up the NFS Server and Creating the Root File System

The NFS server provides the root file system to all cluster nodes. The root file system contains standard libraries and programs which may be used in our cluster. The packages for the NFS server are installed by issuing the command.

```
# local - Boot off of local media (harddrive, USB stick).
```

```
$ sudo aptitude install nfs-kernel-server nfs-common
```

1. First we create a directory to store the bootable filesystem for cluster nodes

```
$ sudo mkdir -p /nfsroot/kerrighed
```

2. Next we edit /etc/exports by adding the following in order to export the the cluster nodes' root filesystem:

```
# /etc/exports
```

```
/nfsroot/kerrighed 10.1.1.0/255.255.255.0(rw,no_subtree_check,async,no_root_squash)
```

3. For the NFS changes to take effect we re-export the file systems

```
$ sudo exportfs -avr
```

4. We have shared the root filesystem using NFS but so far it is empty. Now we populate it with a usable file system. We install the packages needed and install the base system to the bootable filesystem folder. In this case it is a minimal install of Ubuntu Hardy (10.04).

```
$ sudo aptitude install debootstrap
```

```
debootstrap hardy /nfsroot/kerrighed http://archive.ubuntu.com/ubuntu/
```

5. Change the current root of the file system to the bootable filesystem directory.

```
$ sudo chroot /nfsroot/kerrighed
```

All of the following commands are executed within the chrooted file system.

6. Set the root password using the standard command

```
$ passwd
```

It may be noted that the use of sudo facility is not done within the chrooted filesystem. This is so because once in the chrooted file system the user identity is that of root user and the use of sudo is not required to execute privileged commands.

7. Mount the /proc directory of the current machine

```
$ mount -t proc none /proc
```

The /proc is a pseudo file system used by the Linux kernel to store its run-time data structures. Several of following commands would not work correctly if /proc file system was not available while in the chrooted environment.

8. The base root filesystem created in the steps above by debootstrap is missing certain language files. These will cause to give warning and error messages intermittently. The missing files can be installed by issuing the command:

```
$ aptitude install language-pack-en
```

9. Edit /etc/apt/sources.list in order to download the necessary packages. This file identifies the upstream repositories to be used for automatic software installation[[THREAD 2010](#)].

```
# /etc/apt/sources.list  
deb http://archive.canonical.com/ubuntu hardy partner  
deb http://archive.ubuntu.com/ubuntu/ hardy main universe restricted multiverse  
deb http://security.ubuntu.com/ubuntu/ hardy-security universe main multiverse restricted  
deb http://archive.ubuntu.com/ubuntu/ hardy-updates universe main multiverse restricted  
deb-src http://archive.ubuntu.com/ubuntu/ hardy main universe restricted multiverse  
deb-src http://security.ubuntu.com/ubuntu/ hardy-security universe main multiverse restricted  
deb-src http://archive.ubuntu.com/ubuntu/ hardy-updates universe main multiverse restricted
```

10. We need to update the current package listing for the cluster root filesystem thus:

```
$ aptitude update
```

11. Install the packages that our cluster nodes need for the DHCP, NFS and Secure Shell.

```
$ aptitude install dhcp3-common nfs-common nfsbooted openssh-server
```

12. Edit /etc/fstab of the bootable filesystem to have entries for proc and NFS mounted root filesystems.

```
# /etc/fstab #
```

```
#
```

```
# <file system> <mount point> <type> <options> <dump> <pass>
    proc/          proc      proc     defaults  0      0
    /dev/          nfs/      nfs      defaults  0      0
```

13. Edit /etc/hosts and add all cluster nodes and server to it. In our case it would look like the following:

```
# /etc/hosts #
127.0.0.1 localhost
10.1.1.111 bootserver
10.1.1.221 kerrighednode1
10.1.1.222 kerrighednode2
10.1.1.223 kerrighednode3
10.1.1.224 kerrighednode4
10.1.1.225 kerrighednode5
10.1.1.226 kerrighednode6
10.1.1.227 kerrighednode7
10.1.1.228 kerrighednode8
10.1.1.229 kerrighednode9
10.1.1.230 kerrighednode10
10.1.1.231 kerrighednode11
10.1.1.232 kerrighednode12
10.1.1.233 kerrighednode13
10.1.1.234 kerrighednode14
10.1.1.235 kerrighednode15
10.1.1.236 kerrighednode16
10.1.1.237 kerrighednode17
10.1.1.238 kerrighednode18
10.1.1.239 kerrighednode19
```

14. Do the following to create a symlink to automount the NFS shared filesystem at startup. This should not collide with other existing services e.g./etc/rcS.d/S35xxxxxxx

```
$ ln -sf /etc/network/if-up.d/mountnfs /etc/rcS.d/S34mountnfs
```

15. It is necessary to edit `/etc/network/interfaces` and disable the network manager from managing the nodes ethernet cards, as it can cause issues with NFS. Change the network configuration to be manual as shown below:

```
# /etc/network/interfaces #  
# Used by ifup(8) and ifdown(8). See the interfaces(5) manpage or  
# /usr/share/doc/ifupdown/examples for more information.  
  
# The loopback network interface  
auto lo  
iface lo inet loopback  
  
# The primary network interface, commented out for NFS root[LINUX 2009]  
iface eth0 inet manual
```

In our case the network interface is configured during the kernel booting by the DHCP client within the Linux Kernel TCP/IP stack.

16. Create an administrative user for the bootable system. Replace `<username>` with whatever you want.

```
$ adduser <username>
```

In our case user name is “Student”

17. Add an entry for the new user in the `/etc/sudoers` file. This will allow the user to perform administrative commands in the bootable file system.

```
# /etc/sudoers #  
#User privilege specification  
root ALL=(ALL) ALL  
Student ALL= (ALL) ALL
```

18. Install the NTP software for time synchronization:

```
$ sudo aptitude install ntpdate
```

19. Add the following to `/etc/default/ntpdate` for time to be synchronized with the Internet

Standard NTP servers:

```
# The settings in this file are used by the program ntpdate-debian, but not  
# by the upstream program ntpdate[LINUXAUTH 2009].
```

```
# Set to "yes" to take the server list from /etc/ntp.conf, from package ntp,  
# so you only have to keep it in one place.
```

```
NTPDATE_USE_NTP_CONF=no
```

```
# List of NTP servers to use (Separate multiple servers with spaces.)
```

```
# Not used if NTPDATE_USE_NTP_CONF is yes.
```

```
NTPSERVERS="ntp.pool.org"
```

```
# Additional options to pass to ntpdate
```

```
NTPOPTIONS=""
```

20. Add the following command to /etc/rc.local

```
# /etc/rc.local #
```

```
/usr/bin/ntpdate-debian
```

21. Exit from the chrooted bootable filesystem

```
$ exit
```

5.2.5 Testing the diskless boot system

In order to test the network based booting of cluster nodes we perform the following steps.

1. Restart the software providing boot services

```
$ sudo /etc/init.d/tftpd-hpa restart
```

```
$ sudo /etc/init.d/dhcp3-server restart
```

```
$ sudo /etc/init.d/nfs-kernel-server restart
```

2. Boot the cluster nodes after their MAC addresses have been entered into the DHCP configuration. We would observe the boot time messages on the cluster nodes indicating each of the steps:

- a. PXE Boot messages showing PXE, DHCP and TFTP activity
 - b. Kernel loading, hardware detection and initialization messages
 - c. NFS mounting of root file system
 - d. Starup of Linux services such as Secure Shell and others on the cluster node
3. The interaction between the boot server and the cluster nodes can also be observed on the boot server by watching the `/var/log/syslog` file on the boot server using the command:

```
$ tail -f /var/log/syslog
```

Now that we have got a diskless boot system setup, we need to build the Kerrighed kernel for the nodes to use and configure the Kerrighed settings in order to have a working SSI (Single System Image) cluster. Those steps are detailed in the next section.

5.2.6 Building the Kerrighed Linux Kernel

These instructions are mostly based on [Kerrighed 2009] and are duplicated here with necessary clarifications. Current Kerrighed release is based on Linux Kernel 2.6.20. A version based on Linux Kernel 2.6.30 is in the making and is available through the git repository. That should give a huge leap forward to the Kerrighed project in terms of general Linux kernel compatibility. We proceed to the instructions on downloading and building Kerrighed and the pre-requisite software.

1. On the boot server, once again chroot into the bootable file system.

```
$ sudo chroot /nfsroot/kerrighed
```

2. Install the necessary packages for compiling the Kerrighed Linux Kernel into the bootable filesystem.

```
$ aptitude install automake autoconf libtool pkg-config gawk rsync bzip2 gcc-3.3 libncurses5  
libncurses5-dev wget lsb-release xmlto patchutils xutils-dev build-essential
```

Gcc-3.3 is necessary else error will occur.

Above would install the GNU C/C++ compiler and other software building tools. Next we get the

latest kerrighed sources from INRIA's GForge and the vanilla 2.6 kernel.

5.2.7 Installing Kerrighed from the Source

Following steps are taken to build Kerrighed from sources.

1. To get the Kerrighed sources from INRIA's Gforge use wget as follows:

```
$ wget -O /usr/src/kerrighed-latest.tar.gz http://kerrighed.gforge.inria.fr/kerrighed-latest.tar.gz
```

Another method is to obtain the latest source from Kerrighed subversion repository as shown below:

```
$ svn checkout svn://scm.gforge.inria.fr/svn/kerrighed/branches/kerrighed-2.4
```

2. Optionally, download linux-2.6.20 sources tarball into '/usr/src'. You can once again use the wget utility to perform the download.

```
$ wget -O /usr/src/linux-2.6.20.tar.bz2 http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.20.tar.bz2
```

3. Decompress the downloaded Tarballs:

```
$ cd /usr/src
```

```
$ tar zxf kerrighed-2.4.0.tar.gz
```

```
$ tar jxf linux-2.6.20.tar.bz2
```

4. Configure the Sources:

```
$ cd kerrighed-2.4.0
```

```
$ ./configure
```

```
$ ./configure --with-kernel = /usr/src/linux-2.6.20 cc = gcc-3.3
```

```
$ cd kernel
```

```
$ make deconfig
```

```
$ make menuconfig
```

```
$ make kernel
```

```
$ make
```

\$ make kernel-install

\$ make install

\$ cd config

5.2.8 Configure the kernel

Running `./configure` sets up the kernel with a default configuration which may not suit your needs but gives you a running Kerrighed configuration[KRG 2010]. Use `./configure --help`` for possible options.

6. Build the sources:

\$ make

If you have more than one CPU core on the boot server you can use

\$ make -j 4

To increase the speed of compilation by initiating four instances of compiler processes. After the kernel is compiled you can go ahead and install all, as user **root**

\$ make install

To confirm if everything has been installed properly you can check the installation. You should now have the following dir/files installed as shown in Table 6.3.8.

Table 5.2 Kerrighed files installed

File	Purpose
<i>/boot/vmlinuz-2.6.20-krig</i>	Kerrighed kernel
<i>/boot/System.map</i>	Kerrighed kernel symbol table
<i>/lib/modules/2.6.20-krig</i>	Kerrighed modules
<i>/etc/init.d/kerrighed</i>	Kerrighed service script *
<i>/etc/default/kerrighed</i>	Service configuration

<i>/usr/local/share/man</i>	Manpages
<i>/usr/local/bin/krghadm</i>	Cluster administration tool
<i>/usr/local/bin/krghcapset</i>	Process capabilities tool
<i>/usr/local/bin/krghcr-run</i>	Process checkpoint/restart helper
<i>/usr/local/bin/migrate</i>	Process migration tool
<i>/usr/local/lib/libkerrighed-*</i>	Kerrighed library
<i>/usr/local/include/kerrighed</i>	Kerrighed library headers

5.2.9 Kerrighed Configuration

The Kerrighed kernel needs one parameter, the session id. This id is between 1 and 254 and can be set through either the Kernel boot command line or through a configuration file */etc/kerrighed_nodes*. Each instance of a Kerrighed cluster within the same network has a unique *session_id*. This allows Kerrighed cluster nodes to identify their siblings within the same cluster. If using the Kernel boot command line the boot parameter would be *session_id=XX* (XX is between 1 and 254 and same for all nodes within the same Kerrighed cluster). Alternatively, if using the second method edit the */etc/kerrighed_nodes* file to define the session ID for all nodes of the cluster, and the number of nodes that have to be available before the cluster autostarts. It should look like the following[HPC 2011]:

```
# /etc/kerrighed_nodes #
```

```
session=1 #Value can be 1 - 254
```

```
nbmin=19 #Number of nodes which load before kerrighed autostarts.
```

If *nbmin* is set to 0 the cluster would not start automatically and would have to be manually started.

Edit the file */etc/default/kerrighed* and ensure that it contains the following so the kerrighed service is loaded and started within the Kernel of a cluster node:

```
# /etc/default/kerrighed #
```

If true, enable Kerrighed module loading

ENABLE=true

Issue the following command so that dynamic libraries installed by Kerrighed are registered with the dynamic library loader within the cluster node root filesystem:

\$ ldconfig

Exit the chrooted bootable filesystem

\$ exit

Now we must reconfigure the TFTP boot configuration we set up earlier to use the Kerrighed kernel. First, copy the new Kerrighed kernel to the tftp directory.

\$ cp /nfsroot/kerrighed/boot/vmlinuz-2.6.20-krp /var/lib/tftpboot

Edit /var/lib/tftp/pxelinux.cfg/default so it boots the Kerrighed kernel. It should look like:

Cp debian /etc/init.d Kerrighed

Update -rc.d Kerrighed defaults

LABEL linux

KERNEL vmlinuz-2.6.20-krp

APPEND console=tty1 root=/dev/nfs nfsroot=10.1.1.111:/nfsroot/kerrighed ip=dhcp rw

Since all the configuring is done its time to restart the servers again.

\$ sudo /etc/init.d/tftpd-hpa restart

\$ sudo /etc/init.d/dhcp3-server restart

\$ sudo /etc/init.d/nfs-kernel-server restart

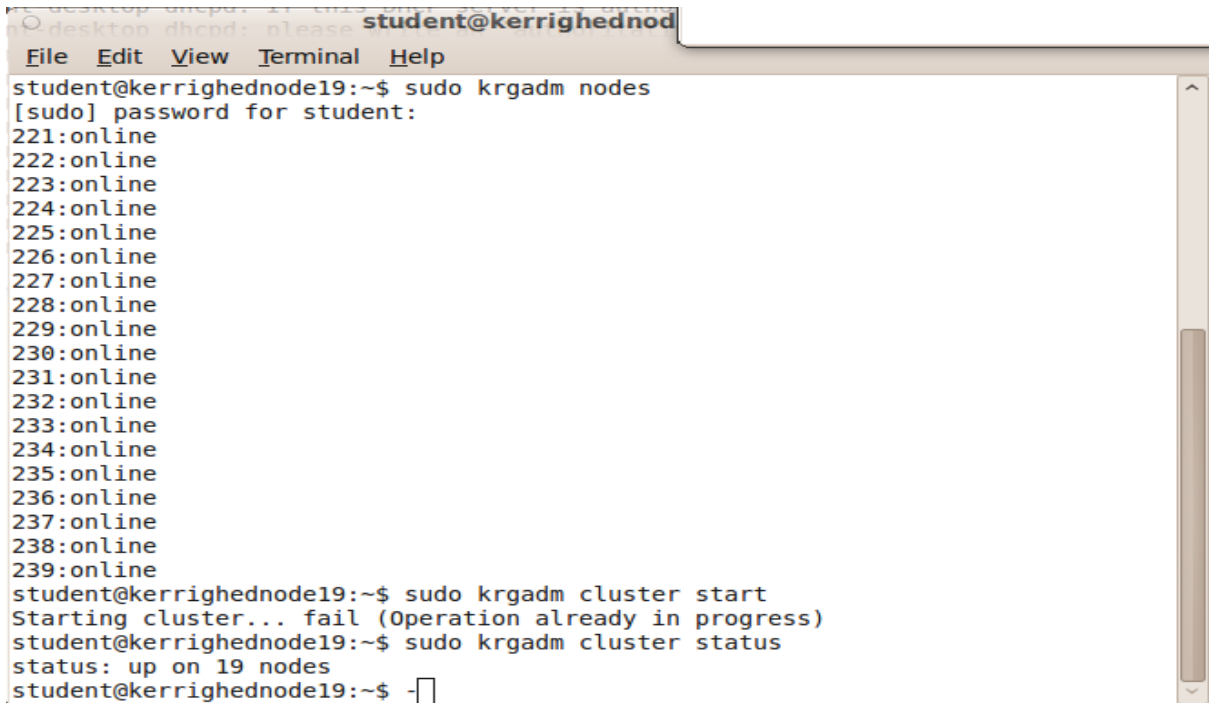
Once again boot up all of the nodes, and if the login prompt appears, the new kernel has booted fine. Login into one of the nodes (either by ssh or on the node itself).

5.2.10 Starting Kerrighed

We can check the list of connected nodes of the

```
$ sudo krgadm nodes
```

You should see a list of all nodes in the format node ID : Status Below figure shows the list of nodes



```
student@kerrighednode19:~$ sudo krgadm nodes
[sudo] password for student:
221:online
222:online
223:online
224:online
225:online
226:online
227:online
228:online
229:online
230:online
231:online
232:online
233:online
234:online
235:online
236:online
237:online
238:online
239:online
student@kerrighednode19:~$ sudo krgadm cluster start
Starting cluster... fail (Operation already in progress)
student@kerrighednode19:~$ sudo krgadm cluster status
status: up on 19 nodes
student@kerrighednode19:~$ -
```

Figure 5.3 Screenshot of Cluster Nodes

To start the kerrighed cluster type

```
$ sudo krgadm cluster start
```

To see if the cluster is running type the following:

```
$ sudo krgadm cluster status
```

To list the process capabilities for the kerrighed cluster type:

```
$ sudo krgcapset -s
```

To allow process migration to take place between nodes in the cluster type the following:

```
$ sudo krgcapset -d +CAN_MIGRATE
```

```
$ sudo krgcapset -e +CAN_MIGRATE
```

At this point of time the Kerrighed Cluster should be working fine. To check if it's working execute a program 20 times and then check if processor utilization is reaching to 100%.

5.3 Testing Kerrighed

In order to test Kerrighed we use the building of Linux Kernel as a processing intensive task. First we build the Linux Kernel using single process and then by using large number of concurrent processes. Table 3 below shows the results:

Results presented in Table 3 require some explanation. We ran trials with number of concurrent kernel compilation processes. For each number of concurrent processes except 1 and 2 concurrent processes we ran the kernel building make command with both the Kerrighed process migration disabled as well as enabled. It can be seen as the number of concurrent compilation processes increases, the instances where Kerrighed process migration is in effect require lesser time. Overall there is better performance with Kerrighed process migration but the overheads of inter-node communication make the instances where the 20 concurrent processes are used within the same cluster node to outperform those cases where process migration is in effect. This can be explained as follows. Kernel building process is sort of random and irregular, successive files do not require same amount of compilation times. It does not reflect the structure of many scientific computation scenarios where there may be a lot of regularity in the algorithm. Nonetheless this exercise does demonstrate the capability of Kerrighed to migrate processes automatically and perform load leveling. With large number of kernel compile processes we see that the Kerrighed software hangs. Several issues related to instability of Kerrighed have been reported in Kerrighed developer's forum. We were running the latest Kerrighed software on 64 bit platform. It appears that more work is required to get Kerrighed stable. Another explanation of kernel compile hanging the cluster could be race conditions arising due to Kerrighed scheduler. In the figure 10 we show the concurrent compilation processes and corresponding CPU utilizations within the cluster nodes.

```

student@kerrighednode19: ~
File Edit View Terminal Help
top - 10:19:53 up 1:58, 2 users, load average: 18.70, 8.16, 3.79
Tasks: 336 total, 21 running, 315 sleeping, 0 stopped, 0 zombie
Cpu7072: 92.3%us, 0.5%sy, 0.0%ni, 0.0%id, 0.0%wa, 1.2%hi, 6.0%si, 0.0%st
Cpu7073: 100.0%us, 0.0%sy, 0.0%ni, 0.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu7104: 0.0%us, 0.0%sy, 0.0%ni, 98.0%id, 0.0%wa, 0.0%hi, 2.0%si, 0.0%st
Cpu7105: 98.7%us, 0.0%sy, 0.0%ni, 1.3%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu7136: 0.0%us, 0.2%sy, 0.0%ni, 97.0%id, 0.0%wa, 1.0%hi, 1.7%si, 0.0%st
Cpu7137: 99.3%us, 0.0%sy, 0.0%ni, 0.7%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu7168: 0.0%us, 0.0%sy, 0.0%ni, 97.5%id, 0.0%wa, 0.7%hi, 1.7%si, 0.0%st
Cpu7169: 99.2%us, 0.0%sy, 0.0%ni, 0.8%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu7200: 0.0%us, 0.0%sy, 0.0%ni, 97.0%id, 0.0%wa, 0.5%hi, 2.5%si, 0.0%st
Cpu7201: 99.2%us, 0.0%sy, 0.0%ni, 0.8%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu7232: 0.0%us, 0.2%sy, 0.0%ni, 96.8%id, 0.0%wa, 0.5%hi, 2.5%si, 0.0%st
Cpu7233: 99.3%us, 0.0%sy, 0.0%ni, 0.7%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu7264: 0.0%us, 0.0%sy, 0.0%ni, 97.3%id, 0.0%wa, 0.7%hi, 2.0%si, 0.0%st
Cpu7265: 99.0%us, 0.0%sy, 0.0%ni, 1.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu7296: 0.0%us, 0.3%sy, 0.0%ni, 98.0%id, 0.0%wa, 0.0%hi, 1.7%si, 0.0%st
Cpu7297: 99.0%us, 0.0%sy, 0.0%ni, 0.7%id, 0.0%wa, 0.0%hi, 0.3%si, 0.0%st
Cpu7328: 0.0%us, 0.3%sy, 0.0%ni, 97.0%id, 0.0%wa, 0.3%hi, 2.3%si, 0.0%st
Cpu7329: 99.7%us, 0.0%sy, 0.0%ni, 0.3%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu7360: 0.0%us, 0.3%sy, 0.0%ni, 97.7%id, 0.0%wa, 0.7%hi, 1.3%si, 0.0%st
Cpu7361: 99.0%us, 0.0%sy, 0.0%ni, 1.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu7392: 0.0%us, 0.2%sy, 0.0%ni, 94.8%id, 0.0%wa, 1.0%hi, 4.0%si, 0.0%st
Cpu7393: 99.8%us, 0.0%sy, 0.0%ni, 0.2%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu7424: 0.0%us, 0.2%sy, 0.0%ni, 96.5%id, 0.0%wa, 0.8%hi, 2.5%si, 0.0%st
Cpu7425: 99.8%us, 0.0%sy, 0.0%ni, 0.2%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu7456: 0.0%us, 0.2%sy, 0.0%ni, 96.8%id, 0.0%wa, 0.2%hi, 2.8%si, 0.0%st
Cpu7457: 99.0%us, 0.0%sy, 0.0%ni, 1.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu7488: 0.0%us, 0.2%sy, 0.0%ni, 96.0%id, 0.0%wa, 0.2%hi, 3.5%si, 0.0%st
Cpu7489: 99.0%us, 0.0%sy, 0.0%ni, 1.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu7520: 0.0%us, 0.0%sy, 0.0%ni, 97.0%id, 0.0%wa, 0.7%hi, 2.3%si, 0.0%st
Cpu7521: 99.0%us, 0.0%sy, 0.0%ni, 1.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu7552: 0.0%us, 0.2%sy, 0.0%ni, 97.5%id, 0.0%wa, 0.2%hi, 2.0%si, 0.0%st
Cpu7553: 98.5%us, 0.0%sy, 0.0%ni, 1.5%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu7584: 0.0%us, 0.0%sy, 0.0%ni, 96.5%id, 0.0%wa, 0.5%hi, 3.0%si, 0.0%st
Cpu7585: 99.8%us, 0.0%sy, 0.0%ni, 0.2%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu7616: 0.0%us, 0.0%sy, 0.0%ni, 97.8%id, 0.0%wa, 0.5%hi, 1.8%si, 0.0%st
Cpu7617: 99.2%us, 0.0%sy, 0.0%ni, 0.8%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu7648: 0.2%us, 0.2%sy, 0.0%ni, 91.8%id, 0.0%wa, 1.0%hi, 6.7%si, 0.0%st
Cpu7649: 99.2%us, 0.0%sy, 0.0%ni, 0.8%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 9601496k total, 1330176k used, 8271320k free, 0k buffers

```

Figure 5.4 CPU utilization of Kerrighed nodes at the time of Sieving

5.4 Application of Clusters to Solve the Integer Factorization Problem (IFP)

The security of RSA algorithm lies in the adversary not being able to factor the publicly known modulus n . If the enemy were able to factor n , it could recover d from the publicly known e by using the equation in step 5 section 8.1. Hence RSA can be completely broken if there was a feasible way to factor an integer. For this reason RSA parameters p and q are chosen to be large integers, several hundred decimal digits long or equivalently several thousand binary digits long.

Over centuries attention has been paid to integer factorization (IF) problem. The best methods discovered so far require computationally infeasible amount of time for integers which are more than 200 decimal digits long. Table 6.5 is taken from [Aoki 2006] and shows the time complexity of the various integer factorization algorithms.

Table 5.1 Time Complexity of Various Integer Factorization Algorithms

method	complexity	effective range
TD	$L_p[1, 1]$	$p \leq 2^{28}$
ECM	$L_p[1/2, 1.414]$	$p \leq 2^{130}$
MPQS	$L_N[1/2, 1.020]$	$N \leq 2^{320}$
SNFS	$L_N[1/3, 1.526]$	$N > 2^{320}$
GNFS	$L_N[1/3, 1.923]$	$N > 2^{320}$
MPGNFS	$L_N[1/3, 1.902]$	$N > 2^{2000}(\?)$

$$L_x[s, c] = \exp((c + o(1))(\log x)^s(\log \log x)^{1-s})$$

One of the best known integer factorization algorithms is the Quadratic Sieve (QS) and its variants. The scheme for implementing integer factorization algorithm such as quadratic sieve using distributed processing of a cluster would use a structure.

Below figure shows how a 100 digit number is breakups to find prime numbers using high performance cluster using [MPQS] on 19 nodes.

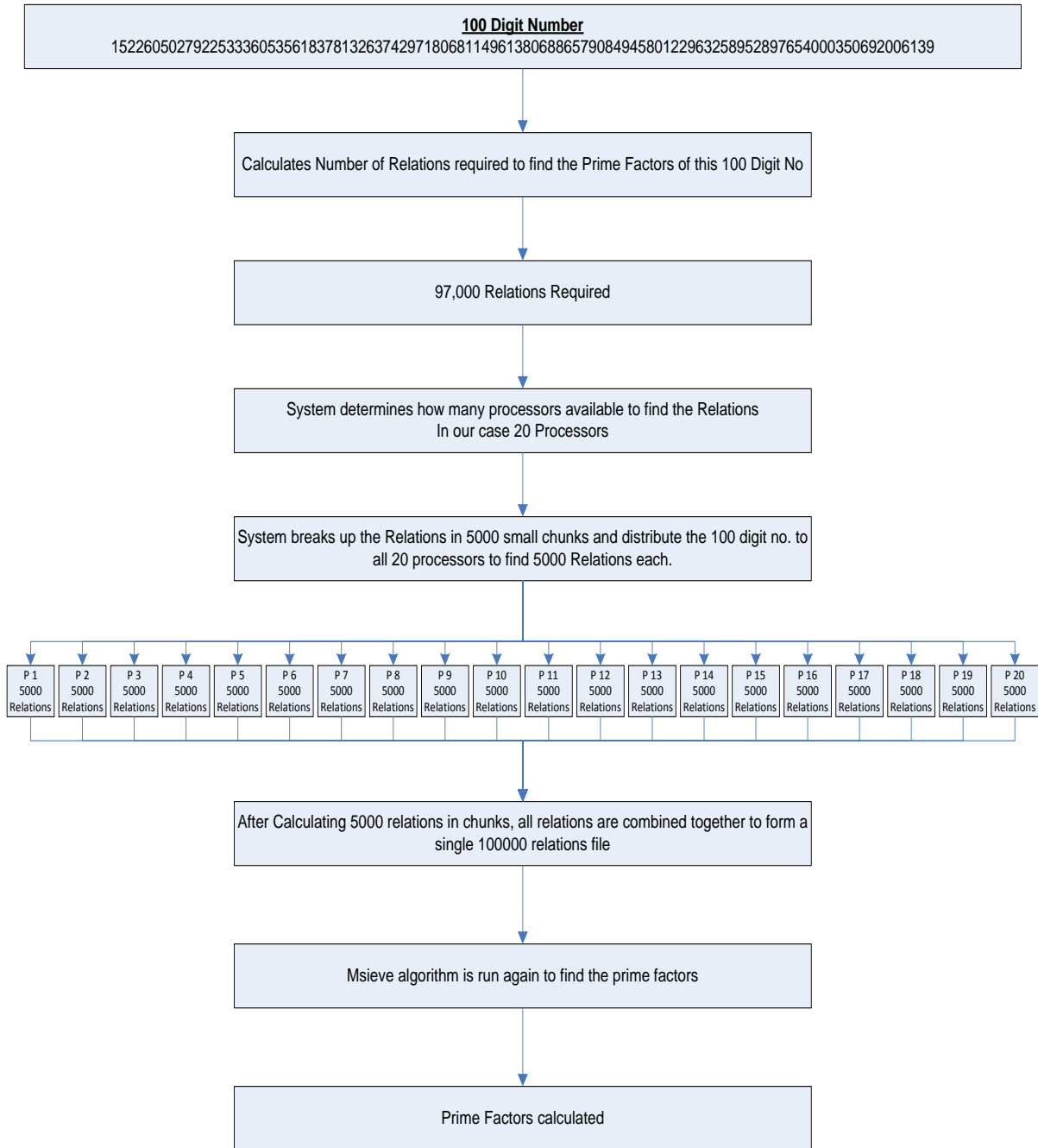


Figure 5.5 Overall Msieve process to factor 100 digit number

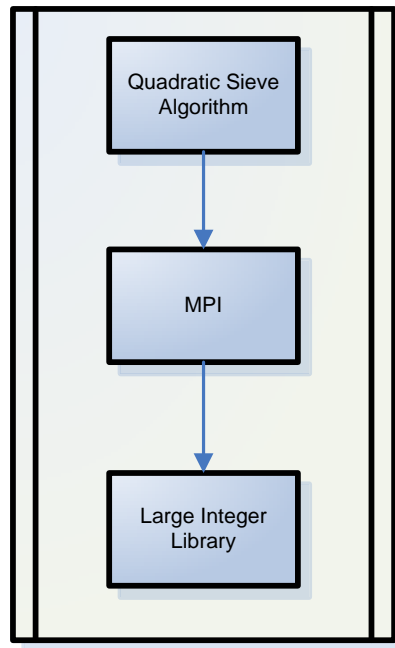


Figure 5.6 General Structure of Parallel Integer Factorization Program Using Clusters

The QS algorithm consists of a number of steps and some of the steps involved lend themselves to parallel implementation. These are the steps that can be sped up using a cluster. However, other steps are of a nature that will not benefit by the cluster. One of the well known and well maintained implementation of QS and few other IF algorithms is available in [Msieve 2009].

The below figures shows that original problem is distributed among nineteen nodes to find 5000 relations. Each node is responsible for finding its own relation. Figure 5.5d shows that the CPU utilization is almost 100% when the process is executed. Although there is some overhead such as I/O read-write and network but overall output speeds up the original problem to almost 9 times.

```

student@kerrighednode9: /usr/src/msieve-1.42
File Edit View Terminal Help
sieving in progress (press Ctrl-C to pause)
5055 relations (4236 full + 819 combined from 276708 partial), need 5000
sieving complete, commencing postprocessing
5074 relations (4275 full + 799 combined from 276822 partial), need 5000
sieving complete, commencing postprocessing
5105 relations (4347 full + 758 combined from 278628 partial), need 5000
sieving complete, commencing postprocessing
5198 relations (4382 full + 816 combined from 283392 partial), need 5000
sieving complete, commencing postprocessing
5231 relations (4418 full + 813 combined from 289515 partial), need 5000
sieving complete, commencing postprocessing
5235 relations (4381 full + 854 combined from 287304 partial), need 5000
sieving complete, commencing postprocessing
5121 relations (4295 full + 826 combined from 286863 partial), need 5000
sieving complete, commencing postprocessing
5285 relations (4409 full + 876 combined from 293930 partial), need 5000
sieving complete, commencing postprocessing
5121 relations (4307 full + 814 combined from 284073 partial), need 5000
sieving complete, commencing postprocessing
5132 relations (4345 full + 787 combined from 287200 partial), need 5000
sieving complete, commencing postprocessing
5272 relations (4379 full + 893 combined from 288771 partial), need 5000
sieving complete, commencing postprocessing
3341 relations (2986 full + 355 combined from 198442 partial), need 5000

```

Figure 5.7 Original problem is distributed among nineteen nodes to find 5000 relations

```

student@kerrighednode19: ~
File Edit View Terminal Help
top - 10:20:20 up 1:58, 2 users, load average: 19.87, 9.33, 4.28
Tasks: 336 total, 19 running, 317 sleeping, 0 stopped, 0 zombie
Cpu(s): 1.2%us, 0.1%sy, 0.0%ni, 97.8%id, 0.0%wa, 0.2%hi, 0.7%si, 0.0%st
Mem: 9601496k total, 1359436k used, 8242060k free, 0k buffers
Swap: 0k total, 0k used, 0k free, 279460k cached

  PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM    TIME+  COMMAND
 929041169 student  25   0 47956 25m 236 R  102  0.3   2:05.87 msieve
 929041176 student  25   0 47844 20m 208 R  102  0.2   1:52.59 msieve
 929041168 student  25   0 47940 30m 512 R  101  0.3   2:14.35 msieve
 929041170 student  25   0 47800 29m 520 R  101  0.3   2:11.08 msieve
 929041171 student  25   0 47616 29m 552 R  101  0.3   2:27.32 msieve
 929041172 student  25   0 48004 29m 512 R  101  0.3   2:10.89 msieve
 929041173 student  25   0 48036 28m 512 R  101  0.3   2:03.84 msieve
 929041177 student  25   0 47852 28m 492 R  101  0.3   2:03.36 msieve
 929041178 student  25   0 47788 28m 512 R  101  0.3   2:05.27 msieve
 929041186 student  25   0 47612 25m 340 R  101  0.3   1:51.04 msieve
 929041175 student  25   0 47536 29m 512 R  100  0.3   2:22.72 msieve
 929041179 student  25   0 47488 28m 512 R  100  0.3   2:15.71 msieve
 929041182 student  25   0 47988 28m 552 R  100  0.3   2:04.21 msieve
 929041183 student  25   0 47540 29m 512 R  100  0.3   2:25.56 msieve
 929041184 student  25   0 47784 25m 340 R  100  0.3   1:58.60 msieve
 929041180 student  25   0 47736 28m 552 R  99   0.3   1:59.87 msieve
 929041167 student  25   0 48048 19m 208 R  98   0.2   1:38.62 msieve
1004538456 root      10  -5    0    0    0  S    9  0.0   1:45.35 krgcom/0
1004538788 student  16   0 19108 1360 864 R  4  0.0   0:00.04 top
 929041185 student  18   0 47848 13m 208 R  2  0.1   1:15.94 msieve
   1 root      18   0 4012 340 64  S    0  0.0   0:00.57 init
   2 root      RT  0    0    0    0  S    0  0.0   0:00.00 migration/0
   3 root      34  19    0    0    0  S    0  0.0   0:00.00 ksoftirqd/0
   4 root      RT  0    0    0    0  S    0  0.0   0:00.00 watchdog/0
   5 root      RT  0    0    0    0  S    0  0.0   0:00.00 migration/1
   6 root      34  19    0    0    0  S    0  0.0   0:00.00 ksoftirqd/1
   7 root      RT  0    0    0    0  S    0  0.0   0:00.00 watchdog/1
   8 root      10  -5    0    0    0  S    0  0.0   0:00.01 events/0
   9 root      10  -5    0    0    0  S    0  0.0   0:00.00 events/1
  10 root      10  -5    0    0    0  S    0  0.0   0:00.00 khelper
  11 root      10  -5    0    0    0  S    0  0.0   0:00.00 kthread
 101 root      10  -5    0    0    0  S    0  0.0   0:00.00 kblockd/0
 102 root      15  -5    0    0    0  S    0  0.0   0:00.00 kblockd/1

```

Figure 5.8 Multiple msieve processes performing sieving on the Kerrighed cluster



Kerrighed Cluster Details

Administrative Informations

Cluster Name: RSA Buster
Organization/Department: Pakistan Naval Engineering College, NUST
Location: Karachi, Pakistan
Coordinates: Latitude: 0, longitude: 0. [Locate on a map](#)
Note: Team Members:- Dr Athar Mahboob, Sami Khan, Adil Khan

Hardware

CPU Type: Intel Pentium 4
Node Number: 19
CPU/Node: 1
Core/CPU: 1
Memory/Node: 1.00GB
Interconnect: Ethernet 100 Mbps

Software

Kerrighed Version: 2.4.4
Main applications: Factorizing using Quadratic Seiveing Alogorithm

Figure 5.9 Official Kerrighed Registration

6 RESULTS

We have factored the following 100 decimal digit integer from Factoring Challenge of RSA Security® by using msieve algorithm implementation. Although this challenge was already completed in 1991 and took several days for find the prime factors of this number.

RSA-100 = 15226050279225333605356183781326374297180681149613
 80688657908494580122963258952897654000350692006139

Prime factors generated with msieve using Quadratic Sieve algorithm are:

RSA-100 = 37975227936943673922808872755445627854565536638199
 × 40094690950920881030683735292761468389214899724061

The time required to factor with and without the cluster is given in Table 7.1 below.

Table 6.1: Performance of MPQS Algorithm on 19 Nodes Cluster

Number of Nodes	Number of Cores	Time (hours)	Speedup (Approx)
1	1	64	1X
4	4	22	3X
8	8	12	5X
12	12	10.5	6X
16	16	8.5	7.5X
19	20	7	9X

Using the processing power of different CPU (Dual core machine was used with a 2.0 GHz Pentium D processors and 512 megabytes of memory each) of a cluster helps to find the time consuming part of QS Algorithm that is *SIEVING*. The number is break up in as many processes as required and allowed to run on a cluster. Each process is responsible to generate the Relations as defined by the user, in our case 5000 relations were assigned to each process. Once all process completes generating the relations, all relations are combined to form a

single file. Next the process is run again, this time it will not calculate computationally intensive part to find the relations and only applies the post processing part to find the prime factor of that number.

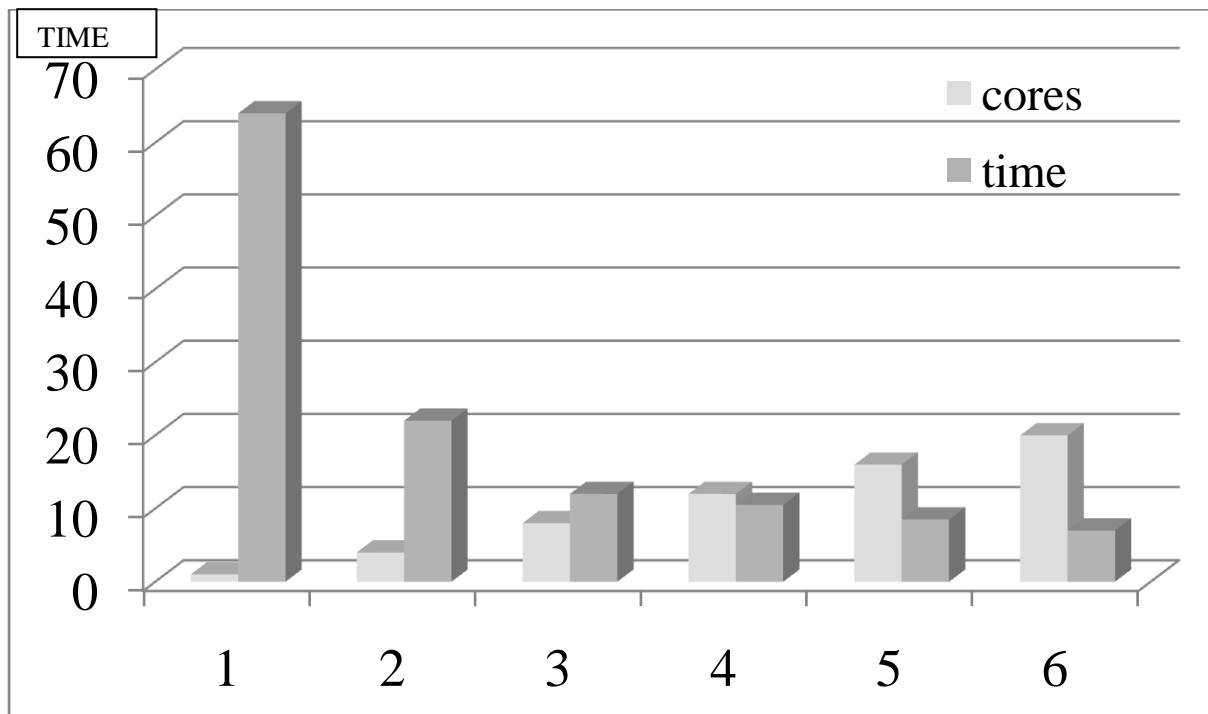


Figure 6.1: Bar Chart Performance of MPQS Algorithm on 19 Nodes Cluster

7 CONCLUSION

High performance clusters are one of the emerging technologies and have been used in modern era to deal with challenging problems in the field of information security and breaking of cryptographic algorithms. Factorization of integers is a very difficult task which forms the foundation of security of many encryption algorithms, for example RSA algorithm. The largest integer factorized to date is of 232 decimal digits. We present results of our work on setting up single server image clusters using Kerrighed SSI based cluster system. We have attempted to use Multiple Polynomial Quadratic Sieve [MPQS] algorithm for integer factorization. This clustering recipe and the results presented shall be useful to others who are planning to setup Linux clusters (Kerrighed) for high-performance computing.

We have factored a 100 decimal digit integer from RSA Factoring Challenge by using MPQS algorithm implementation running on our 20 node Kerrighed SSI cluster within seven hours. The estimated price of our cluster is less than USD 5,000. We have by far improved upon the original RSA-100 cracking done in 1991 using MPQS on Massively Parallel [MasPar] computers. The estimated price of MasPar was close to a million USD for 16,384 processor elements and it took several days to factor RSA-100.

8 FUTURE WORK

Based on our implementation and study we found that below areas are still open that can be considered as a future work.

- **Add a Big Integer class to QS**

In our Thesis we have factored a 100-digit number. We can use a bigger number in future to break the integer into primes.

- **Replace the existing Nodes with the latest technology hardware**

The existing Kerrighed nodes are Pentium D machines with 500MB of memory each. If the current nodes are replaced with the latest i7 processors with 8MB of memory, then we can solve a larger number Integer Factorization problem in less time.

- **Automation of process is required**

Quadratic Sieve algorithm is currently not 100% automated with Linux Kerrighed Cluster for finding relations and then finding prime numbers. Further modification in source code is required.

- **Newer version of Kerrighed 3.0**

Newer version of Kerrighed i.e. 3.0 is available that is compatible with Linux Kernel 2.6.30.

- **High-availability cluster option for Boot Server**

High Availability is another option apart from Performance that can be achieved through Kerrighed Clusters.

9 REFERENCES

[HPC 2011] High Availability Clusters <http://www.debianadmin.com/how-to-set-up-a-high-performance-cluster-hpc-using-debianlenny-and-kerrighed.html>.

[KRG 2010] Kerrighed Clusters
http://www.kerrighed.org/wiki/index.php/Installing_Kerrighed_2.4.0.

[RSA 2009] RSA Encryption Algorithm <http://www.secret-bases.co.uk/wiki/RSA>.

[LAM/MPI 2008] LAM/MPI Parallel Computing, <http://www.lam-mpi.org/> last accessed Aug 3, 2009.

[MPICH2 2009] MPI Implementation latest version released July 2009
<http://www.mcs.anl.gov/research/projects/mpich2/> accessed on Aug 3, 2009.

[OpenSSI 2006] OpenSSI version 1.9.1 for Fedora core 3, <http://openssi.org/cgi-bin/view?page=download.html> accessed on July 27th 2009.

[Kerrighed 2012] Kerrighed Cluster Technology <http://www.kerlabs.com/technology.en.html>.

[RSANUM 2008] RSA Numbers <http://mathworld.wolfram.com/RSA.html>.

[EBC 2009] Easy Ubuntu Clustering forum
<http://ubuntuforums.org/showthread.php?p=6495259> accessed on July 15, 2009.

[MOSIX 2009] MOSIX at Wikipedia, <http://en.wikipedia.org/wiki/MOSIX>

[THREAD 2010] <http://forums.phoenixlabs.org/printthread.php?s=&t=18230>

[LINUX 2009] <http://www.felipe-alfaro.org/blog/category/linux/>.

[Top500a] Top500 Supercomputing Sites, Operating Family Share Over Time, Available at <http://www.top500.org/overtime/list/33/osfam>

[LINUXAUTH 2009] <http://actuel.wikidot.com/projets:authenticationlinuxldapgroupe2>

[QSS 2009] http://www.danpritchard.com/wiki/Quadratic_sieve.

[Msieve 2012] Integer Factorization Source Code for Msieve at
<http://www.boo.net/~jasonp/qs.html>

[QS 2012] Quadratic Sieve introduction
http://en.wikipedia.org/wiki/Quadratic_sieve

[EUC 2011] Ubuntu Clustering Setup Guide
<https://wiki.ubuntu.com/EasyUbuntuClustering/UbuntuKerrighedClusterGuide>

10 APPENDICES

Below are the few relations that were calculated at the time of Quadratic Sieving.

2c8d59af47c81ab3725b472be417e3bf7ab85439af726ed3dfdf66489d155dc0b771c7a50ef7c5e58fb

A 77 87 88 8d 9e a0 a1 a2 a9 ae b8 ca 3e1
R -1c4e1 4 0 2 4 1a5 3f6 8cf d0d e44 lced 2a21 485c L 3f8dd7 d1f9663
R 59256 4 0 1 3 3 3 5 6 37 2bb 7b4 226e 6267 c370 17c1d L 30231d 7662fcd
R 3bae9 10 0 2 3 b 1c8 3f4 40d 19f5 12697 15ea6 169cb L 7debc7 152cf5c9
R 8244c 12 1 2 8 c 47 93 171 28c 99a 4f4f ce6d 147e8 L 3ca08b 1489dbf
R -5f0e7 15 0 2 2 b 2b fe 224 7a7 e24 2230 3340 7196 L 936f3d 10a2276f
R -7bb3c 1a 1 2 2 4 8 f 35 11d 25e 41f 6ea 804 d10 132fd L 571427 1511993
R 666a 1c 0 1 55 61 2c3 3bb 66f 1006 11ac 29f8 4e57 L 6330d9 e31c943
R 66749 1c 2 d07 116d 177f 43ea 55e6 57ca 74a6 L 2f5f6f 108049d3
R -32b90 20 0 1 3 23 2f 68 73 eb2 1d51 40fe 10885 1133a L 32562d 60e67d1
R 35f0b 20 0 2 2 2 2 3 6 33 45 85 95 1f2 3de cfd 2e7c 48be L 5b8863 c1fd67d
R b947 26 0 2 1b 1d 67 d7 168 1a3 9eb 1dc5 6c64 9c5f L 469d6f 1bc0f25
R -5c438 28 0 1 5 d e 14 32 19f b76 136b 2991 2db9 6060 11559 L 1 3c176cf
R 69e82 28 1 2 3 3 3 3 33 cc 186 2d1 9e8 14eb 3faa f221 L 7ec325 7b313f1
R -19713 2b 0 2 3 7 5c 161 2a0 3e8 de4 2428 11281 13175 L 829333 2d5552f
R 614fb 2e 0 2 10 19 24 37 336 339 360 102b eebf 12142 L ca1639 ec60a5
R -634f6 2f 0 1 2 3 3 3 f 14 4a 1f4 316 471 1718 aebf 14c5b L 30d0f7
315d65b
R 374c0 38 0 1 2 2 3 63 214 221 5cf 3641 6189 b034 e3de L 780995 13c3cc1
R 59d5f 3a 0 4 4 14 dc 123 127 1a0 830 879 2412 7654 L c90e15 8fa2185
R -2579a 44 0 1 6 a 16 2d 45 4d 110 814 6584 6a92 e2c3 L 5620bd 1271b18d
R -1f98d 44 0 2 3 4 a 31 10b 273 83b 1015 4565 4c9c 7e5c 139b0 L 1 3bc73a3
R 45511 45 0 2 3 6 8 1b 1d 70 509 8b1 8431 b86c 1777d L d08583 ccc8d43
R -48dff 4d 0 2 3 6 7 162 b17 246a 2a8b 3898 958e 116b6 L 6ff09d 7aa649
R 60df4 4f 0 1 3 3 5 8 e 17 2b 68 3d0 989 d9e 39f4 5ee8 L b6155f 2bc4b81
R 34f86 53 0 1 2 6 2e 4e5 506 f80 2953 2a0f 4ab4 d1fc L a6c445 b6d973
R 606a3 55 0 2 3 6 1f 5e 76 76 18c 4d8 1f14 7b36 ee3e 110b9 L 1 9509263
R -6455a 57 0 1 2 16 23 37 17e 85d 14e5 ldb1 2592 edal L dd8253 adbb6c5
R 19ca9 58 0 2 3 4 5 21 4e c6 83d 92d 2eb1 140ae 167d7 L 2af28c7 34f5d3d
R 8c5d1 59 2 2 5 8 46 4d 67 b4 6f8 b19 cf2 20e0 38c4 L 370bc5 2d663df
R 6a6d8 5a 1 2 3 3 3 8 8 a 28 c82 1c3b 78a9 17cda 17ceb L 67b203 7cb6d45
R -8350d 5b 2 3 4 4 b 1ec 2c6 aad eed 23c4 d53e da6d 11f55 L 1 cc1cb49
R 51396 5b 0 1 2 f 6e 75 e2 1fd 157c 3579 bc5d 14e83 L 20339c3 35b10e9
R 67a9a 5d 1 2 26 40 46 458 c7c fb0 2168 2c4a 4ffa 7933 L 1 153d09e7
R -21166 5f 0 1 2 2 2 4 6 c 368 405 653 8cb 21c2 262d 4e2e L 1712ab9
44cbf4b
R -8dc2b 61 2 3 4 5 5 5 5 6 d 21e 32e 2f9d 3c08 6f26 e430 179e4 L 1 8b87d5
R -173ea 62 0 1 2 2 3 3 3 5 4a 4b 21e 3f4 af7 41c4 4427 6a98 L 1052d9d
25eade9
R -11906 65 0 1 2 3 3 7 37 14b 171 1f1 5db 66b 70c 781 1d13 L 83bd93
10eb165
R -2b5f 65 0 3 3 6 5f 1414 31f3 44b4 ccc1 10988 120d7 L d6c025 1be22bb
R -3d509 6e 0 2 2 2 2 2 a5 1c5 4eb0 63f1 7cc2 dd67 16eae L 26014bd 47e14d3
R 5d519 79 0 21 2f 38 3cd 43f 4ba 4835 1142c 17136 L 834b47 1523d2e1
R 68a66 7a 1 2 3 3 3 3 d cf 10c 190 1b10 2b6b 9b3a 16b95 L 53c80d 4c7ab3b
R -2532a 7e 0 1 3 4 7 15 16 28 76 7f fb 3d5 69e ef4 d4e7 L 672bb7 716eb85
R -895ce 80 1 2 8 9 c 10 4f 6e 82 9b8 32e8 9555 11b87 L 1c327f7 526541b
R 69bdb 85 2 2 2 2 6 9 fd 120 677 1f98 3efc 919b 1678b L 3b40af 2ca6f4f
R 5941b 89 0 2 2 2 2 2 3 4 5 c 55 168 113c 1914 led5 8283 ef1c 119d1 L 1
563a43
R 4ece4 8b 0 1 2 2 3 7 8 17 63 16a 485 726 b34 215f 15769 L 2ffcb9d 31d8aad

R 70bc7 8b 2 4 7 67 da 135 354 d71 2995 3f10 8063 L 107009f 69ccec9
R -3168a 8e 0 1 2 3 5 2b 62 5b9 84e 4128 bf28 c1c9 ce7f L 3b351f dae75d
R -8c169 90 2 2 5 7 5e 174 ebb 6020 8e3a 11b57 12ed6 L ac2307 ad94f23
R 18c64 91 0 1 2 28 c3 d6 4d5 5c8 1bf7 2c31 3bb8 8027 ecd3 L 1 4f3dd8b
R 783ac 92 1 2 5 22 42 4b 55 562 5ef 5730 9726 fb9a L 126a703 2592701
R -15849 95 0 2 c 1c 29 5a 6b 15b 17f 10e9 1627 214c 287e 9271 L 1 28df8cf
R 6892b 97 1d 1e 23 24 42 17b 183 239 cdb 177f 1ebb 14b97 L 1 cfb788d
R -46cb5 9b 0 2 3 5 9 29 12d 33c 8bc c91 cd5 18ea a747 L cb21b1 414ee87
R -15372 9d 0 1 2 3 4 15 145 a50 12f1 2955 2e94 4a58 a8a2 L 296cdb 7df4c8f
R 237e5 a0 0 2 2 1a 9c 1dbe 24ed 4888 52ba 67f1 14f09 L 7aef0f 25b8929
R 794c4 a1 1 3 5 c 15 228 571 7c0 870 bb8 292c 42e6 L 1799bb5 42c5e19
R 50a74 a7 0 1 2 8 2a 56 5c 152 174 532 9fd c99b dfc5 L 5331eb a539269
R -891d7 a8 2 4 12 85 da 23d 28ae 28e0 5cf2 7b7c 80d1 865f L 1 3c6c3d
R e443 ad 0 2 4 5 18 135 659 907 4def 5958 62b4 c8ee L 2cae83 1ea4bc1
R -298b6 b1 0 1 2 110 166 1c2 8bf 11c4 81b1 cdfa 14085 L 751351 bd2e16b
R 64f30 b2 0 1 2 3 a 16 3d 15d 170 195 269 11df 1de7 f2f6 L 31507f ca5a6d
R -5c382 b4 0 1 2 10 1f 4d 8c0 1c32 24f7 2f1f 53dc 83bb c69a L 1 64477eb
R -872b4 b6 1 2 8 1e 96 128 bcf 1f73 544f ed5d 119f1 L 696bf5 b616a69
R 3b24 b7 0 1 2 2 11 13 56 96 1ed2 420b 637d bc26 be03 L 89286d 425afbb
R -42d63 b8 0 3 5 f5 85b 964 be52 f374 10e8e 15fd7 L c47687 76f48c1
R -33c12 b8 0 1 2 68 2d0 53f 598 881 bd6d d918 116e0 L 24ade87 2715689
R 674c0 ba 1 2 3 4 14 1f 2b 3e 5a ce cc8 1364 324c 9b1f 11048 L 1 de0e6b
R -6e5bc be 1 2 6 16 195 242 8e6 189b 3ee7 6696 11c9b L 9fe5c3 5139b2f
R 11e20 be 0 1 2 c 306 815 32f3 4c0a 4c18 722a ab5f L 181010f 2f4e9b7
R 6b71e bf 1 2 2 3 10 24 34 3b4 bb0 1127 1cc2 608b 74d5 8f9b L 1 e6be36d
R -82f86 c3 1 2 3 6 48 11c 5ab cd4 21fd 702e a915 b521 L 3efaaf 3eff1b
R -87901 c4 4 2f 108 a18 c7f 69dc cd2b dd67 ec82 L 67a9fb c08be1
R 3abc8 c9 0 1 b e 2c 62 f0 157a 3d05 4628 7a8c 1407c L 455f1b 48d67b
R 6e20b ca 3 7 25 4e 94 95 27c baf 2036 693e a3a2 L 3cff0d 2791f45
R 86297 ca 2 2 3 1a ab b2 f1 14c 25f 4323 7746 110a8 L 6bb70b 15cd5769
R 8d023 cb 2 2 3 4 41 12f 18c 29e 1000 9229 97b2 ff1e L 64ae4b 922d1c5
R 6e06a d3 1 2 4 5 11 15 4e 11e 1cb c87 1886 39c2 eca5 L 3945a1 11ed8301
R 5efd2 d5 0 1 2 3 7 7 b 1633 1ce3 4f6a 59e1 b866 d627 L 3304ab 1528d0f
R 5d10c d6 0 1 a 20 32 35 3fa 893 cel 10ce 464f 15dae L 31d81f 19386eb
R 29e66 df 0 1 2 2 3 4 7 22 437 4b1 629 1db9 2114 2693 570d L 40e631 55c273
R -6b466 e1 1 6 d 16 1d ac eb 226 6a6 710 344e 99bc L 568caf 6707e3f
R -5b5f7 e5 0 3 6 6 16 2a 1d4 c75 1add 510b 933a 16aee L 3ddb01 3f469f9
R 56c04 e8 0 1 2 3 3 4 12 14 28 63 67 97 3d3 1679 169a 4af1 L 9a495f
8c279ad
R -37b10 ea 0 1 2 3 8a 176 847 1d7e 3822 4d03 7bb0 f8da fd03 L 1 f6bddd
R 51ef2 ea 0 1 2 2 5 12 13 1a 220 416 b3c 2e84 8881 e1db L 768811 104d496f
R 74796 ee 1 2 1d8 296 2a3 7b6 150f 3472 4613 7a4b L 81f6c7 ecbebl d
R 6605d f2 3 5 6 d 10 10 4d 92 4a8b 732f 1221b 150df 16c5e L 1 3c4487
R -4d8ff f4 0 2 2 3 6 8 14 36 d9 1a43 2586 3146 7689 86f5 bea3 L 1 6ae34b5
R -17864 f4 0 1 9 a 10 17 b1 195 235 46a a0d8 10066 10f93 L 3a3caf 4610c9
R 89134 f7 1 2 2 3 8 3a 7c aa 10a5 130c 283f a705 16b99 L 65194d 2c01de7
R -65e9b f8 2 5 6 62 5c0 783 dbc 1c62 c1b0 10838 L 2ff2af 40db89b
R -782c2 fc 1 12 33 3b e2 851 215a ad36 afa6 e336 L 155e72d 63e09ff
R 5cb03 fc 0 2 3 f 6e c2 27c 3bd 70e 30a9 ae78 10e6d L 3675f7 2bed8f1
R 2e05 ff 0 2 2 3 9 d 13 4c 89 231 7fe ddb6 14d63 151d2 L 5a002b 1ef75a1
R -49f59 103 0 3 8 16 1c 28 4cel 59e6 14e95 16440 16ef7 L 6a403b 6b360ed
R -6fc60 108 1 2 4 6 e 1b b2 1739 4d6a b638 12404 169b5 L 5f51fb 1014259
R 6ed37 108 3 10 324 56e 13a7 3da3 9092 9618 ab36 1247c L 1 d181f47
R 29989 109 0 7 13 57 76 111 118 19ce 2398 cef1 12d50 L e0e879 6ae9aed
R -79a33 10a 2 2 2 4 a 12 32 62 9d 197 381 1090 2f27 52cc L 70ba0d 142e6e6b
R 7cca5 10a 2 2 2 3 3 5 6 b 103 155 162 5d0 2428 32db 10b9c L 120a513
4b5cd3d
R 7f45c 10a 1 2 4 5 22 26 27 41 1d1 17e6 3f3d 46d3 fc05 L 6fea79 8dca873
R -3d8cd 10b 0 4 11 22 1a7 395 b5a c86 labb a0e2 d06b L f4da79 139b63d
R -485f5 10c 0 2 2 2 2 2 4 14 1f 66 4356 520b 54e0 6be2 dbf5 L 2ea369
bcbe959