

The Creation of the Gentoo Clustering Live Medium

Eric Thibodeau

September 9, 2008

Contents

1	A Short Definition of a Beowulf Cluster	2
2	Single System Images, the node's environment	4
2.1	The Boot Image	4
2.2	Server Side Daemons	6
2.3	The Pre Boot Process	6
2.3.1	PXE booting	6
2.4	The Kernel Boot Process	9
2.4.1	The <code>stateless.sh</code> Init Script	9
2.5	Openrc, The Diskless <code>init</code> System Of Choice	14
3	The Master Server	15
3.1	The Daemons	15
3.2	Configuring The Server	16
3.2.1	The Configuration File Format	17
3.2.2	Setting up <code>net-nds/ldap-auth</code>	17
3.2.3	Configuring DHCP, TFTP and DNS	20
3.3	The <code>cluster-setup</code> Utility	20
4	Building your own Live Medium	21
4.1	Setting up the Build Environment	21
4.1.1	Required Packages	21
4.1.2	Setting up the Catalyst Environment	22
4.1.3	Fixing up a Few Quirks	23
4.2	Creating the Live Medium	24
4.3	Using the Live Medium	24

List of Tables

List of Figures

1.1	Typical Beowulf Cluster topology.	2
2.1	LiveCD folder organization	5
2.2	Filesystem layers on the <i>server</i> and the <i>nodes</i>	5
2.3	Booting PXE and TFTP	7
2.4	<code>pxelinux.0</code> searching for the configuration at start up.	8
2.5	Hand off to kernel booting, calling init script	10

Listings

2.1	Contents of /tftproot/pxelinux.cfg/default	8
2.2	Contents of /boot/stateless.sh	10
2.3	Contents of /root/mkDataDisk.sh	13
3.1	Contents of /etc/gentoo/ldap_auth.conf	18
4.1	Commands for Adding Catalyst Work Dirs	22
4.2	Contents of /etc/catalyst/catalystrc	22
4.3	Commands for Setting up the Release Engineering SVN Root	22
4.4	Commands for Getting a Recent Portage Snapshot	23
4.5	Commands for Getting a Recent Stage 3	23
4.6	Commands for Getting the Gentoo Clustering Live CD git . .	23
4.7	Commands for Correcting OpenLDAP dependancies	23

Acronyms

COTS Computers Off The Shelf

DHCP Dynamic Host Configuration Protocol

DNS Domain Name Service

GSSAPI Generic Security Services Application Program Interface

HPC High Performance Computing

HDD Hard Disk Drive

IP Internet Protocol

LDAP Lightweight Directory Access Protocol

MAC Media Access Control

NIC Network Interface Card

NFS Networked File System

PXE Pre eXecution Environment

RW Read/Write

SGE Sun Grid Engine

SSI Single System Image

TFTP Trivial File Transfer Protocol

Abstract

With the advent of multi-core processors and the recent stagnation in processor clock frequency scaling, parallel processing is becoming more and more important in the evolution of computation in any class. Single-threaded applications will no longer be able to rely on processor clock speed and Instruction Level Parallelism to gain speed, leaving only parallel processing as a solution to this physical and technological bottleneck. Unfortunately, parallel processing has typically been reserved to an elite class of programmers and developers due to the complex nature of parallel environments when it comes to configuration and use of tools for programming, debugging and performance assessment.

Building machines for High Performance Computing (HPC) and Scientific computing, especially in a parallel environment, is a non-trivial task. It requires intricate operating system knowledge and highly specialized tools, most of which tend to be permanently masked in the Gentoo tree mostly due to their complexity and cross-dependencies.

As part of the 2008 Google Summer of Code project for Gentoo, a Clustering Live CD was created provide an to provide both users and administrators the means to quickly build a Gentoo based Beowulf clustering solution without having to hand pick and configure all the low-level system packages. Many default configuration settings as well as automatic configuration scripts and wizards are included to guide the administrator and users into successfully using (and abusing) their HPC machines.

Introduction

The building and configuration of a Beowulf clustering environment involves the manipulation of multiple system packages as well as scientific libraries combined with the addition of user and administrator tools (scripts) to ease the usage of such beasts. The application of this environment to the restraints of Live Mediums such as a *LiveCD* or *LiveDVD* adds significant complexity to the entire process.

In this document, we detail the the creation, the booting and the use of the Gentoo Clustering Live Medium environment. Although there are specificities pertaining to the creation of the Live Medium, most of the material presented here is applicable to a permanent installation of a Beowulf cluster using Gentoo as the base OS. All packages created in the context of this project had this possibility in mind and should thus facilitate the setting up of the environment.

We will start by detailing the Clustering environment composed of the server and its Single System Image which is used for the node's environment. The actual recreation of this system on LiveCD will then be explained. Most of the work is performed using **Catalyst**¹, the same tool used by the Gentoo Release Engineering Team.

The elements in this document are presented in the order by which they need to be understood and built. It may seem paradoxical that one builds the node's environment prior to the server's but this is done purposefully to reflect the build process of the entire environment.

¹From the `dev-util/catalyst` package

Chapter 1

A Short Definition of a Beowulf Cluster

Our interpretation of a *Beowulf Cluster*¹ is of the *classic type*, meaning that one takes Computers Off The Shelf (COTS), connects them together using regular network fabrics such as Ethernet or Gigabit Ethernet and that all these computers reside on an isolated network with a single point of access, being the master node. Figure 1.1 is a simple representation of such a setup.

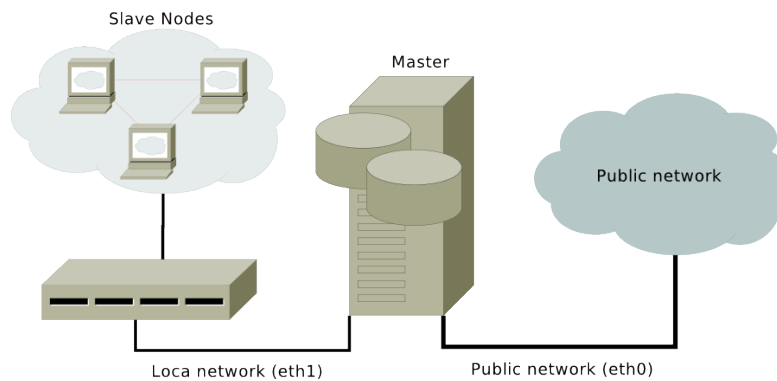


Figure 1.1: Typical Beowulf Cluster topology.

Most of the work we've performed assumes such this topology but can be easily extended to others where high performance fabrics such as *Infiniband* or *Mirynet* are used to connect the nodes. We also take for granted that the

¹<http://www.beowulf.org>

Master server and the nodes share the same hardware architecture, which in our current work is `x86_64` in it's most generic form. This implies that the Live Medium is built so that both AMD and Intel's 64bit processors will function. This said, the addition of node images of differing architecture is meant to be simple thanks to our approach to implementing the node's Single System Image (SSI) as described in Chapter 2 coupled with the use of Gentoo's `catalyst`.

Chapter 2

Single System Images, the node's environment

The idea of using a SSI is not new to the clustering environment. Many variants and interpretation of the approach exist¹ so we believe it's worthwhile to describe the details of our implementation, which provides the following advantages:

- Rapid deployment,
- Single point of management,
- Concise environment

In this section, we will mostly describe the quirks and requirements for an Networked File System (NFS) booted SSI to work. The actual building of one is detailed in Chapter XXX.

2.1 The Boot Image

The SSI we build consists of an entirely independent system image contained in a directory within the root of the server installation. As can be seen in Figure 2.1, this translates into having the CD file system containing an

¹Ours was inspired by prior work from Prof. Wilhelm Meier, available at <http://mozart.informatik.fh-kl.de/download/Software/GentooDiskless/gdxs.html>

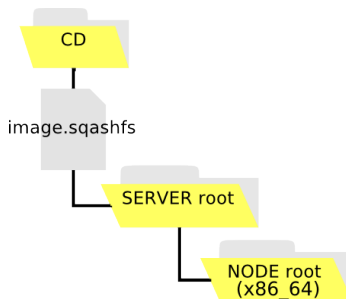


Figure 2.1: LiveCD folder organization

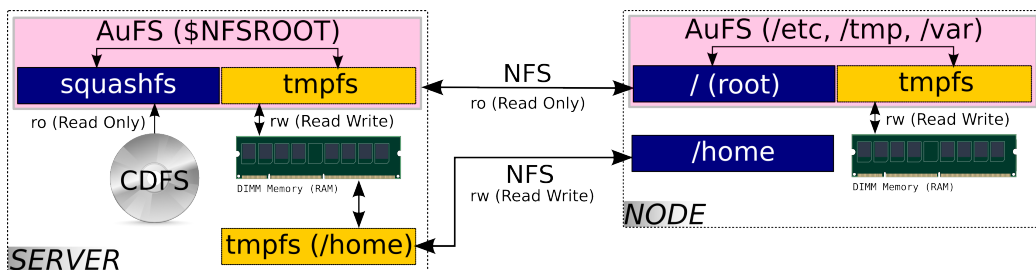


Figure 2.2: Filesystem layers on the *server* and the *nodes*.

`image.squashfs` file which contains the server system image to be bootstrapped...which contains the node's NFS root image. Figure 2.2 demonstrates a gross summarisation of the involved filesystem layering required on the *server* side and the *node* side. Even though the `$NFSROOT` is exported read-only, we have to loop mount it using *AuFS* since `squashfs` doesn't support being exported by NFS.

When the SSI is of identical architecture to the server, it may seem redundant to have a separate root for the nodes as one could use the server's root for the nodes and simply manage that root. The independent root approach was chosen so that it would be easy to add differing node architectures in the case of heterogeneous environments. Furthermore, it also provides an separate *playgrounds* in which one may test different compilation settings, kernels and library combinations without impacting the entire cluster. Switching between images becomes a simple case of changing a configuration file and rebooting the affected node, as described in Section 2.3.

2.2 Server Side Daemons

To be able to *serve* such images, it is required that the node's Network Interface Card (NIC) support Pre eXecution Environment (PXE) booting² and that the server be configured with the following daemons:

- Dynamic Host Configuration Protocol (DHCP),
- Trivial File Transfer Protocol (TFTP),
- Networked File System (NFS)

The detail of their configuration is presented in Chapter 3.

2.3 The Pre Boot Process

2.3.1 PXE booting

The boot process can be described as follows, observing Figure 2.3, PXE initiates the *bootp* protocol requesting an Internet Protocol (IP) address as well as extra information such as the *boot file* (option 67). The DHCP server responds with an IP address as well as the file to be used to initiate the boot process, in occurrence, `pxelinux.0`³. At this point, TFTP is used to download the `pxelinux.0` executable which then scans for the appropriate configuration file within `/tftproot/pxelinux.cfg/` using TFTP. The file name search order is as follows:

1. Media Access Control (MAC) address formatted as `00-0c-29-df-1e-5c`,
2. IP address in hexadecimal such as `C0A801D4` for `192.168.1.212`⁴. The search mechanism strips down the address until it matches a configuration file,
3. If no configuration file was matched, the `default` file is used as configuration file.

²Although not a strict requirement, since it's possible to boot using Etherboot from `http://etherboot.org`, we will concentrate on this simpler case.

³This file is provided by the `sys-boot/syslinux` package.

⁴This hexadecimal value can be obtained using `sipcalc --addr-ipv4 192.168.1.212` from the `net-misc/sipcalc` package.

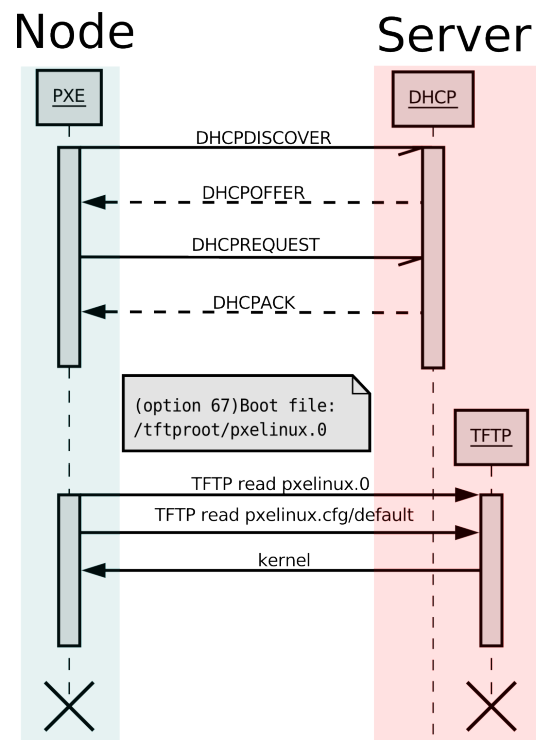


Figure 2.3: Booting PXE and TFTP

```

CLIENT MAC ADDR: 00 0C 29 2B E4 D7  GUID: 564D8FBE-EAAE-ADFA-D13D-65709B2BE4D7
CLIENT IP: 192.168.1.212  MASK: 255.255.255.0  DHCP IP: 192.168.1.2
GATEWAY IP: 192.168.1.1

PXELINUX 3.11 2005-09-02  Copyright (C) 1994-2005 H. Peter Anvin
UNDI data segment at:  0009BF0
UNDI data segment size: 4D60
UNDI code segment at:  0009E950
UNDI code segment size: 0BBC
PXE entry point found (we hope) at 9E95:0106
My IP address seems to be C0A801D4 192.168.1.212
ip=192.168.1.212:192.168.1.2:192.168.1.1:255.255.255.0
TFTP prefix: /tftproot/
Trying to load: pxelinux.cfg/01-00-0c-29-2b-e4-d7
Trying to load: pxelinux.cfg/C0A801D4
Trying to load: pxelinux.cfg/C0A801D
Trying to load: pxelinux.cfg/C0A801
Trying to load: pxelinux.cfg/C0A80
Trying to load: pxelinux.cfg/C0A8
Trying to load: pxelinux.cfg/C0A
Trying to load: pxelinux.cfg/C0
Trying to load: pxelinux.cfg/C
Trying to load: pxelinux.cfg/default
boot: _

```

Figure 2.4: pxelinux.0 searching for the configuration at start up.

Figure 2.4 illustrates this process when no specific configuration file is present. We can see that the `default` configuration file is finally used. This file resides on the server under `/tftproot/pxelinux.cfg/default` and this relates to the TFTP prefix and `pxelinux.cfg/default` displayed. Listing 2.1 details its contents. Note that that the file is actually automatically generated by the `sys-cluster/beowulf-head-0.1` package, which uses its configuration file to set paths to the configured architecture. The kernel to be downloaded via TFTP is specified by the line `kernel nfsroot/x86_64/boot/kernel`

NOTE:

All paths to files, during the PXE boot process are *relative* to `tftproot` folder. In other words, when the `default` config file defines `kernel nfsroot/x86_64/boot/kernel` the location of the file, on the server, is actually: `/tftproot/nfsroot/x86_64/boot/kernel`

Once the specified menu delay has expired, `pxelinux.0` hands off the execution to the kernel.

Listing 2.1: Contents of `/tftproot/pxelinux.cfg/default`

```

1 | prompt 1
2 | timeout 50
3 | say Press F1 for boot profiles , default is x86_64_node in 5 seconds...
4 | F1 BootProfiles
5 | default x86_64_node
6 |
7 | label x86_64_node
8 | kernel nfsroot/x86_64/boot/kernel

```

```

9 | append ip=dhcp nfsroot=192.168.1.2:/tftproot/nfsroot/x86_64/,hard,intr init=/boot/
10 | stateless.sh unionmod=aufs rwdev=tmpfs
11 | label local
12 | localboot 0

```

NOTE:

DHCP option 93 ‘Client System Architecture Type’⁵ is sent by PXE but we’ve noticed the 64Bit VMWare machine sends IA x86 as architecture and not x86_64. We can therefore not assume that this is a means of identifying the architecture reliably *a priori*. This might come in handy in the near future to enable the provisioning of multiple SSI by being able to select the proper NFS root image accordingly.

2.4 The Kernel Boot Process

The boot sequence is taken on by the kernel as depicted by Figure 2.5⁶. Another DHCP request is sent, by the kernel code this time, because we configured the kernel to do so by setting `CONFIG_IP_PNP=y` and `CONFIG_IP_PNP_DHCP=y` in the kernel’s configuration. Note that this is a gross summarization of the actual network exchanges, the reader is referred to RFC2131⁷ for a detailed description of the DHCP protocol.

2.4.1 The `stateless.sh` Init Script

The kernel hands off the execution to `/boot/stateless.sh`, the init script defined in the `/tftproot/pxelinux.cfg/default` configuration file⁸.

NOTE:

At this point, the path of the script is relative to the architecture’s NFS root, which translates to `/tftproot/nfsroot/x86_64/` and we refer to it as `$NFSROOT`, which in turn is configurable, more on this in Chapter 3.

As can be seen in Listing 2.2 this script performs the following actions using information from both the kernel’s last DHCP request and the kernel’s command line:

1. Mount a Read/Write (RW) device,

⁶Obviously, there is much more involved here but we’re limiting the description to the elements which we configure explicitly for this project.

⁷<http://tools.ietf.org/html/rfc2131>

⁸See line 9 of Listing 2.1.

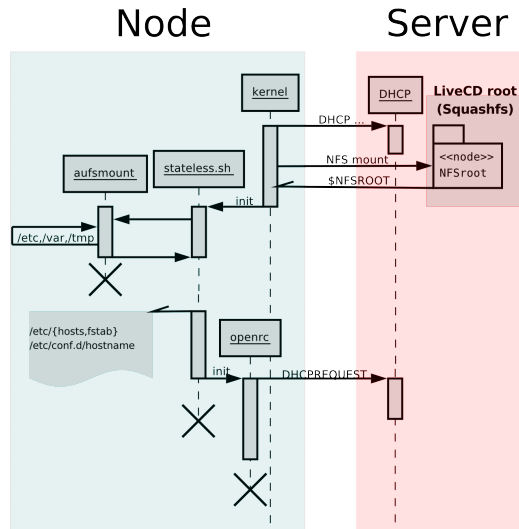


Figure 2.5: Hand off to kernel booting, calling init script

2. Load a unionfs-based kernel module⁹,
3. Mount `/etc,/var,/tmp` as RW using the unionfs mount option,
4. Set the hostname, if DHCP didn't provide one, we generate it using `node` and the last octet from the IP address.
5. Prepare the NFS mounts by generating `/etc/fstab`

Listing 2.2: Contents of `/boot/stateless.sh`

```

1  #!/bin/bash
2  # (C) Eric Thibodeau 2006-2008 GPL v2
3  # Replacement initrc/linuxrc script hack to get diskless NFS nodes booted
4  # This script is largely inspired by prior work from
5  # http://mozart.informatik.fh-kl.de/download/Software/GentooDiskless/
6  #
7  #
8  MODPROBE=/sbin/modprobe
9  IFCONFIG=/sbin/ifconfig
10 #MYHOST=$(/sbin/dhccpd -H; /bin/hostname) # By default, we'll use the DHCP assigned
    hostname
11
12 NODE_NAME="node"
13
14 ahostname(){
15     if [[ -z ${MYHOST} || ${MYHOST} == ${MYIP} ]]; then
16         echo "DHCP didn't tell me my name. Generating my own hostname..."

```

⁹We currently use the `sys-fs/aufs` package, which is part of the `sunrise` overlay.

```

17 |             MYHOST="${NODE_NAME}${MYIP###}"
18 |             echo "I proclaim that I am $MYHOST !!"
19 |         else
20 |             echo "DHCP told me my hostname is ${MYHOST}..."
21 |         fi
22 |
23 |         echo "$0: Setting Hostname to $MYHOST"
24 |         echo "HOSTNAME=\"\$MYHOST\" " > /etc/conf.d/hostname
25 |         /bin/hostname "$MYHOST"
26 |
27 |         echo "Setting domainname to DHCP's settings: $DOMAIN"
28 |         /bin/domainname $DOMAIN
29 |     }
30 |
31 |
32 | # Used to mount the RW part of the ROOT, it should be easy to add other
33 | # FS than tmpfs here.
34 | mount_rw_dev() {
35 |     echo "$0: Mounting $RWDEV as Read Write device ..."
36 |     if [ $RWDEV == "tmpfs" ]; then
37 |         mount -n -t tmpfs -o defaults tmpfs_rw /mnt/rw_mounts/
38 |     fi
39 | }
40 |
41 | mount_rw_dir() {
42 |     while [ "$1" != "" ]; do
43 |         echo "$0: Mounting $1 using $UNIONMOD ..."
44 |         mkdir -p /mnt/rw_mounts/$1
45 |         mount -i -n -t $UNIONMOD -o dirs=/mnt/rw_mounts/$1=rw:/$1=ro ${UNIONMOD}
46 |             -$1 /$1
47 |         shift
48 |     done
49 | }
50 | # ahosts was added so that we could dynamically change the NFS server address
51 | # using the rootserver= DHCP option as the NFS mount server. This is usefull
52 | # when you want to split the load onto different NIC interfaces within a same
53 | # logical network.
54 | ahosts() {
55 |     echo "Setting rootserver to $ROOTSERV in /etc/hosts file..."
56 |     echo "$ROOTSERV rootserver" >> /etc/hosts
57 |     echo "Setting up fstab"
58 |     echo "# Auto-generated at boot time by $0"
59 |     echo "$ROOTSERV:$ROOTPATH / nfs ro,defaults,hard,intr,actimeo=120,
60 |         timeo=14,tcp 0 1" > /etc/fstab
61 |     echo "$ROOTSERV:/home /home nfs rw,defaults,hard,intr,actimeo=120,
62 |         timeo=14,tcp 0 1" >> /etc/fstab
63 | }
64 | # get_param: parses parameters in PARSELINE which could be, for example, the kernel's
65 | # command line
66 | # (This implies PARSELINE=$(cat /proc/cmdline))
67 | # Example:
68 | # ip=dhcp nfsroot=192.168.1.2:/tftpboot/nfsroot/x86_64/,hard,intr init=/boot/
69 | # stateless.sh softlevel=unionfs
70 | # IN:
71 | # $1: parameter we want (ie: nfsroot)
72 | # $2: which token # we want, ie: the IP address of nfsroot it token 1
73 | get_param() {
74 |     PARAM=$2
75 |     for opt in $PARSELINE
76 |     do
77 |         case $opt in
78 |             ${1}*) set $(echo $opt | sed -e's/[=:,]/ /g' )

```

```

76 |         if [[ -z $PARAM ]]; then
77 |             shift
78 |             echo $*
79 |         else
80 |             shift $PARAM
81 |             echo $1
82 |         fi
83 |         return 0
84 |     ;;
85 |     esac
86 | done
87 | echo "Parameter '$1' not found" >&2
88 | return 1
89 | }
90 |
91 | # Used to extract useful information for the rest of the configuration
92 | # It's a horrible hack that parses dmesg but, unfortunately, /proc/net/pnp
93 | # is insufficient even with /proc/cmdline
94 | # Here is an example output of dmesg (partial)
95 | #IP-Config: Complete:
96 | #   device=eth0, addr=10.0.1.140, mask=255.255.255.0, gw=10.0.1.129,
97 | #   host=thinkbig24, domain=cluster.local, nis-domain=(none),
98 | #   bootserver=10.0.1.129, rootserver=10.0.1.129, rootpath=/tftpboot/AthlonXP
99 |
100 | import_dhcp_info() {
101 |     PARSELINE=$(dmesg | grep -A3 'IP-Config: Complete:' | sed -e 's/[[:,]]/ /g' | tr -
102 |         d '\n')
103 |     MYIP=$(get_param addr 1)
104 |     MYHOST=$(get_param host 1)
105 |     DOMAIN=$(get_param domain 1)
106 |     ROOTPATH=$(get_param rootpath 1)
107 |     ROOTSERV=$(get_param rootserver 1)
108 | }
109 | # if mounting /proc becomes a problem we have to use the following:
110 | # PARSELINE=$(dmesg | grep '^Kernel command line' | sed 's/^Kernel command line://g')
111 | import_cmdline() {
112 |     mount -n -t proc none /proc
113 |     PARSELINE=$(cat /proc/cmdline)
114 |     UNIONMOD=$(get_param unionmod 1)
115 |     RWDEV=$(get_param rwdev 1)
116 | }
117 |
118 |
119 | import_dhcp_info
120 | import_cmdline
121 | [ ! -z $RWDEV ] && mount_rw_dev
122 | if [ ! -z $UNIONMOD ]; then
123 |     echo "Loading $UNIONMOD"
124 |     # we should detect whether it's a mod or kernel builtin...
125 |     $MODPROBE $UNIONMOD
126 |     mount_rw_dir etc var tmp
127 | fi
128 | touch /etc || /bin/bash
129 | ahostname
130 | ahosts
131 | exec /sbin/init

```

Stateless.sh, presented in Listing 2.2, is the script to be modified if any node-specific tasks are to be performed at boot time and that couldn't be dynamically defined in configuration files within the node's NFS root image.

Any tasks that can be performed by the host after booting, like formatting a local disk, should be added as part of `/etc/conf.d/local.start` script. Listing 2.3 is an example of such an implementation where we would add `KillHDD` as a kernel command line keyword to indicated that the script is free to wipe the local Hard Disk Drive (HDD) and mount it to `/data/` as local scratch space. The script calls upon `/root/ClusterPartitions.sfdisk`, which is used as a template to partition the local HDD. Both `mkDataDisk.sh` and `ClusterPartitions.sfdisk` reside within `$NFSROOT/root`.

Listing 2.3: Contents of `/root/mkDataDisk.sh`

```

1  #!/bin/bash
2  if [ -b /dev/sda ];
3  then
4      HDD="/dev/sda"
5  else
6      HDD="/dev/hda"
7  fi
8  DATA="${HDD}2"
9  SWAP="${HDD}1"
10 DATAMOUNT="/data"
11 PART_TABLE="/root/ClusterPartitions.sfdisk"
12 HDD.WHIPE.KEYWORD="KillHDD"
13
14 # First, we check the kernel command line to make sure we want to mangle the HDDs:
15 dmesg | grep -q "`Kernel command line.*${HDD.WHIPE.KEYWORD}"
16 if [ $? != 0 ]
17 then
18     echo "Quitting HDD whipe procedure since ${HDD.WHIPE.KEYWORD} is not in kernel
19         command line"
20     exit 2
21 fi
22 # check to see if the disk exists
23 if [ -b $HDD ]
24 then
25     mount ${DATA} ${DATAMOUNT}
26     if [ ! -e /data/${echo $HOSTNAME} ]
27     then
28         # The HDD is not tagged therefore:
29         # 0- We unmount, just in case there was another FS
30         umount ${DATAMOUNT}
31         # 1- We partition it:
32         sfdisk --force /dev/sda < ${PART_TABLE}
33         # 2- We format the partition we want to use as data
34         mkreiserfs -f -f --label "DATA" -q ${DATA}
35         # 3- We also kept some spare space for eventual use of swap
36         mkswap ${SWAP}
37         # 4- Now we mount the disk
38         mount ${DATA} ${DATAMOUNT}
39         # 5- We tag the disk with the hostname
40         touch /data/${echo $HOSTNAME}
41     else
42         echo "Found the file tag /data/${echo $HOSTNAME}, not formatting the
43             drive!"
44     fi
45 else
46     echo "The data disk (${HDD}) is missing!"
47     #echo "Removing ${DATAMOUNT} so no one fills up the ram (unionfs here)"

```

```
47 |         #rmdir ${DATAMOUNT}
48 |         exit 1
49 |     fi
50 |
51 |     chmod 3777 ${DATAMOUNT}
52 |     swapon ${SWAP}
```

2.5 Openrc, The Diskless init System Of Choice

There are many common tasks performed by the *init system* at boot time such as HDD integrity verification, network setup and many maintenance tasks which *don't* apply to a diskless system and actually tend to prevent proper booting. The `sys-apps/openrc` package is a replacement to Gentoo's current *init system* management tools and scripts and is the obvious choice since it natively implements booting off read-only media, such as NFS roots. The use of `openrc` also implies we'll be using the `sys-apps/baselayout-2.*` package series. This fact is important to note since the syntax of the configuration files within the NFS root will differ from the one in the Live Medium since the latter still uses the current *init system*.

Chapter 3

The Master Server

Trivially named Master¹, this is the entry point machine as defined in a typical Beowulf cluster. The node's image(s), user accounts as well as resource management system (Torque), all run from this server. Users will typically compile and launch parallel jobs from this single point of access to the cluster.

3.1 The Daemons

In Section 2.2, we mention that three daemons are minimally required to boot NFS root based nodes. These are NFS, TFTP and DHCP. To these we must add Domain Name Service (DNS) and Lightweight Directory Access Protocol (LDAP) for the nodes to be of any use to us. Fortunately, `dnsmasq`² implements three of the required daemons, DNS, DHCP and TFTP. The local Domain Name Service plays a critical role in being able to identify the nodes correctly. Getting the names right, although a seemingly trivial task, is required for the well being of any cluster since many tools depend on consistent name resolution. But there is more to a cluster than just having nodes booting, the following is a reflection on the additional minimally required mechanisms.

¹OK, so you can change the name in `/etc/gentoo/cluster.conf`.

²From the `net-dns/dnsmasq` package.

Authentication and Authorization

For the nodes to be of any use, we need to provide a means by which users can identify themselves on all the nodes of the cluster. Strictly speaking, *authentication* is the mechanism by which one is *clearly* identified as being who he/she claims to be. Typically accomplished by the use of a user name and a password, such mechanisms are also implemented using biometrics as well as hardware and software keys. Kerberos is a robust (and complex) example of such a mechanism.

Authorization is the means by which one controls the access to resources that an *authenticated* user has access to. These resources can vary quite a bit, it's therefore logical that the mechanism be flexible and adaptive, such as a database. Since authorization is frequently required and read (ie: each file access requires authorization), the mechanism must be rapid and efficient.

Furthermore, since we're managing a network of machine, it is required that both of these be centrally managed and that all the systems remain in sync.

Choice of Mechanism

Prior research in using Kerberos for authentication had exposed complex issues that would arise with the integration of job control systems and thus would limit our Live Medium to using a Generic Security Services Application Program Interface (GSSAPI) aware queue manager³. Also, a user management interface that can manage both Kerberos and LDAP seamlessly is not readily available. These are some of the reasons why we've chosen to use LDAP⁴ for both roles as it provides all that is required by our cluster setup with reasonable security, especially since it will only be used for the cluster within a private network. User account management is accomplished using the `app-admin/diradm` package.

3.2 Configuring The Server

We've gone through the trouble of creating two packages for setting up the environment and aforementioned daemons on the server, the `net-nds/`

³A few discussions on the Beowulf mailing list revealed Sun Grid Engine (SGE) to be one of the few to implement the GSSAPI.

⁴We use the `net-nds/openldap` package version 2.4.10.

`ldap-auth` package and the `sys-cluster/beowulf-head` package. This way, anyone wanting to set up a cluster can do so rapidly and then learn the required tweaks as needed. Although explicitly created for the Live Medium, they were created so that they could be used directly on a *Stage 3* installation of Gentoo Linux.

3.2.1 The Configuration File Format

Each package has its configuration file which is sourced by the ebuild from within the `pkg_config()` routines. This implies the options in these files are in the form of *BASH* styled variables. Each configuration element is preceded by a description of its use, the scope of its impact and one or more example of its use.

3.2.2 Setting up net-nds/ldap-auth

Listing 3.1 presents an example of `/etc/gentoo/ldap_auth.conf` and editing it is almost all that is required. To one extreme, one could simply set the `$CONFIG_OK` variable to `'yes'`⁵ and launch `emerge --config =net-nds/ldap-auth-0.1`, which would set up all the required files on the server. There are also files that need to be configured accordingly on the node's image, this is accomplished by specifying the `$ROOT` variable beforehand such as `ROOT=$NFSROOT/${ARCH} emerge --config =net-nds/ldap-auth-0.1` where `$NFSROOT` is a configuration variable which is set within the `ldap_auth.conf` file and `$ARCH` is the architecture of the node. This approach is done for every architecture *roots* one has within `$NFSROOT`. These commands will take care of creating the following configuration files, the first three are unique to the server, the *italicized* entries in the list are the only ones that required modification on the nodes.⁶:

- `/etc/openldap/slapd.conf`
- `/etc/openldap/ldap.conf`
- `/etc/ldap.secret`

⁵But don't do that, unless you really want a *huge* security hole, the LDAP's administrator password being set to default by the `$LDAP_ADMIN_DN_PWD` in that file!

⁶The differentiation between the node's installation and the server's installation is controlled by the `authmaster USE` flag.

- */etc/ldap.conf*
- */etc/nsswitch.conf*
- */etc/pam.d/system-auth*

The initial LDAP database will also be created and has been configured to reside in \$DBDIR as defined in */etc/gentoo/ldap_auth.conf*, good to know for backups!

NOTE:

Extra attention has been put into the creation of these files so that it would be possible to modify them after the initial configuration is performed. It is therefore wise to still get acquainted with the LDAP quirks as performance tuning and optimization might be required in large installations.

Note that this is a "one shot deal" and that the automated configuration should not be called upon more than once per SSI. In the case of the Live Medium, both the Master server and the node's SSI get configured automatically by the */usr/sbin/cluster-setup* utility, which we describe in Section 3.3.

Listing 3.1: Contents of */etc/gentoo/ldap_auth.conf*

```

1 |#!/bin/bash
2 |# By Eric Thibodeau
3 |# 01 July 2008
4 |#
5 |# NOTES:
6 |#   * it is usually suggested to keep all LDAP definitions in
7 |#     _lower case_... but do as you wish!
8 |#   * all *_DESC variables are optionnal and are simply used
9 |#     in the description field of the LDAP db.
10 |
11 |# The following is to tell pkg-config that this files has been revised
12 |# Set it to "yes" once you're done
13 |CONFIG_OK="no"
14 |
15 |# Name of the LDAP server, if USE=authmaster, then this should usually
16 |# be set to the equivalent of LDAP_SERVER=$(hostname -f). Otherwise, point
17 |# this variable to your domain's LDAP server:
18 |LDAP_SERVER="master.gentoo.local"
19 |
20 |# What is the name you want to give the LDAP domain?
21 |# say we wanted the domain to be gentoo.local you would
22 |# put the following:
23 |#DOMAIN="gentoo.local"
24 |# or, in a more automated-generally-speaking sense:
25 |#DOMAIN=$(hostname -d)
26 |# Since this is the Clustering template, our Domain Context
27 |# is actually more like : "machine.domain.com". Assuming
28 |# the machine is set up right, we can pull that in autmatically
29 |# using the following:
30 |#DOMAIN=$(hostname -f)

```

```

31 | # but we hardcode it for the LiveCD, else you'd be getting my
32 | # machine's name :)
33 | DOMAIN="gentoo.local"
34 | DOMAIN_DESC="This the LDAP server residing on $DOMAIN"
35 |
36 | # Which part of your organisation is this machine filling in
37 | # for? In the present example, this is the cluster so we'll call
38 | # this Organizational Unit (ou) cluster:
39 | OU="cluster"
40 | OU_DESC="Clustering department branch. All units defined under this branch are for use
    |         by the cluster"
41 |
42 | # We will create some specific branches under that cluster,
43 | # logically, we'll have users and groups to manage these, so
44 | # we will minimally impose the definition of these two:
45 | USERS_OU="users"
46 | USERS_OU_DESC="Cluster specific Users"
47 | GROUPS_OU="groups"
48 | GROUPS_OU_DESC="Cluster specific Groups"
49 |
50 | # OUTHER_OU is parsed to automatically create other sub-OU under
51 | # the one defined above (as OU). This could be, for example:
52 | #OTHER_OU="aliases networks hosts"
53 |
54 | # if you want descriptions to be added to each of these groups
55 | # automatically in the LDAP database, define a separate _DESC
56 | # vairable for each. For example:
57 | #aliases_DESC="This is the container for user aliases"
58 | #hosts_DESC="This is the container for static host descriptions"
59 |
60 | # Although this is often a philosophical debate, we'll stick
61 | # with having an admin for the ldap user database and one
62 | # for managing it's contents. Here is the dirrerence in their
63 | # role:
64 | #
65 | # The ADMIN_DN will be the user used to create the ldap db
66 | # and have total control over it. This user is typically useful
67 | # only at creation and dumping/migration of the database. This
68 | # user _always_ has TOTAL access to the LDAP db where it's
69 | # defined. One typically _doesn't_ use this user to mange the
70 | # LDAP database, the user defined in ADMIN_DN is the one to use.
71 | #
72 | LDAP_ADMIN_DN="admin"
73 | # This is the paswword to use for LDAP management tasks and is the
74 | # one that is stored in /etc/openldap/slapd.conf (but we at least
75 | # hash it ;)
76 | LDAP_ADMIN_DN.PWD="default"
77 |
78 | # The following user will essentially be identical to root, you're better
79 | # off not renaming him.
80 | ADMIN_DN="root"
81 | ADMIN_DN_DESC="root account (under LDAP)"
82 |
83 | # The following is the name of the group used to identify people
84 | # with full access to the LDAP db. One advantage is that the members
85 | # of this group can be dynamically changed within the LDAP db. NOTE:
86 | # Since it's given the same gid as wheel, it's fonctionnaly equivalent!!!
87 | ADMIN_GROUP_DN="wheel"
88 | ADMIN_GROUP_DN_DESC="Users in this group can freely modify the LDAP directory at will"
89 |
90 | # The following is the place to put the resulting generated files
91 | # usually we'd want this to be ROOT="/" ...if you trust the script
92 | # entirely ;). We use the environment's $ROOT if one is available...

```

```
93 | [[ -z $ROOT ]] && ROOT="/"
94 |
95 | # The LDIF_OUT defines the name of the ldif file that will be automatically
96 | # created by the script. It's only really useful if you want to keep that file
97 | # afterwards for xyz reason (obviously, LDIF_OUT_KEEP has to be set to yes):
98 | LDIF_OUT="./create-db.ldif"
99 | LDIF_OUT_KEEP="yes"
```

3.2.3 Configuring DHCP, TFTP and DNS

These three daemons are serviced by the `net-dns/dnsmasq` package and automatically configured by the `sys-cluster/beowulf-head-0.1` package, created for this project. As with the LDAP authentication configuration process,

3.3 The cluster-setup Utility

The `cluster-setup` utility is to be called once all configuration files within `/etc/gentoo/` have been edited and the `$CONFIG_OK` variable of each files have been set to "YES".

Chapter 4

Building your own Live Medium

In this chapter, we present the required packages and steps to create your own Live Medium. It is preferable to create your own Live Medium as well as the node's SSI. A redistributable CD is prevented from including packages for many licensing reasons while a personal build aren't¹.

4.1 Setting up the Build Environment

The build host will need to be Gentoo based and of compatible architecture for the destination Live Medium as per Catalyst's requirements². The current Catalyst *spec files* are configured for building a generic `x86_64` bootable CD so it can be used on both AMD and Intel's 64bit architecture. Furthermore, the host build system must be running a kernel with `CONFIG_BLK_DEV_LOOP` built in or as a module. The latter is required for the last Catalyst build stage.

4.1.1 Required Packages

Apart from a working Gentoo system, the following packages will be required to prepare the build environment:

¹It's also the author's belief that any half-decent administrator would *want* to build their own ... "Look ma, a cluster!"

²http://www.gentoo.org/proj/en/releeng/catalyst/index.xml#doc_chap4

- the `dev-util/subversion` package
- the `dev-util/git` package
- the `dev-util/catalyst` package

4.1.2 Setting up the Catalyst Environment

Currently, Catalyst doesn't create the skeleton directories so the following sequence must be performed

Listing 4.1: Commands for Adding Catalyst Work Dirs

```
1 mkdir -p /var/tmp/catalyst/ /var/tmp/catalyst/snapshots /var/tmp/catalyst/tmp/default
```

Edit `/etc/catalyst/catalystrc` and add relevant environment variables such as:

Listing 4.2: Contents of `/etc/catalyst/catalystrc`

1	<code>export MAKEOPTS="-j6"</code>
2	<code>export FEATURES="parallel-fetch"</code>
3	<code>export PBS_SERVER_NAME="master.gentoo.local"</code>

The variables are the same ones as defined in `/etc/make.conf`. Note that `$PBS_SERVER_NAME` is only useful if the intention is to build a Live Medium with the `pbs USE` flag and it has to be set to *the same hostname* as the Master's intended name. This variable is normally deduced at install time and since the build system is most probably not named `master.gentoo.local`, we override it here.

The Release Engineering SVN Root is not strictly required as we only pull in the `README` files defined in their overlay³. Nonetheless, it is also a good source of examples on how all other architecture Live CDs and release medias are defined.

Listing 4.3: Commands for Setting up the Release Engineering SVN Root

```
1 mkdir -p /var/svnroot/
2 cd /var/svnroot/
3 svn co svn://anonsvn.gentoo.org/releng
```

The build environment uses its own *Portage Snapshot*. This is definitely derisive as a live snapshot would be a moving target and it would be close

³The catalyst definition of overlay, which is literally a file system tree overlaid to the one resulting from the catalyst process

to impossible to set up a stable environment, especially since we tend to use *bleeding edge* packages.

Listing 4.4: Commands for Getting a Recent Portage Snapshot

```
1 wget ftp://mirrors.tera-byte.com/pub/gentoo/snapshots/portage-20080806.tar.bz2 -O /var/
  tmp/catalyst/snapshots/portage-20080806.tar.bz2
```

Even though we need to create our own stage tarballs, we need a functional one to start off the process. Here we use the ones created by Daniel Robbins since they are released frequently and will have less chances of hitting circular dependencies.

Listing 4.5: Commands for Getting a Recent Stage 3

```
1 mkdir -p /var/tmp/catalyst/builds/drobbins/
2 cd /var/tmp/catalyst/builds/drobbins/
3 wget http://www.funtoo.org/linux/amd64/funtoo-amd64-2008.06.27/stage3-amd64-2008.06.27.
  tar.bz2
```

Finally, we are ready to get the Gentoo Clustering Live CD git from the Gentoo overlays web site:

Listing 4.6: Commands for Getting the Gentoo Clustering Live CD git

```
1 mkdir -p /var/git/
2 cd /var/git/
3 git clone git://git.overlays.gentoo.org/proj/clustering-livecd.git
```

4.1.3 Fixing up a Few Quirks

Well, this is the part where you put on the *Release Engineer's* haat and do a little fiddling with the ebuilds. On a regular system, this wouldn't be required but since we'll be creating our own stages, some weird dependency problems do show up.

net-nds/openldap, Bug 142637

As per Bug #142637⁴, you have to edit the openldap ebuild (currently the setup pulls in 2.4.10) and add `DEPEND="sys-apps/shadow"`:

Listing 4.7: Commands for Correcting OpenLDAP dependencies

```
1 vi /var/tmp/catalyst/snapshot_cache/20080629/portage/net-nds/openldap/openldap-2.4.10.
  ebuild
2 ebuild /var/tmp/catalyst/snapshot_cache/20080629/portage/net-nds/openldap/openldap
  -2.4.10.ebuild digest
```

⁴http://bugs.gentoo.org/show_bug.cgi?id=142637 "missing dependency in eutils.eclass"

4.2 Creating the Live Medium

The creation of the Live Medium requires that the SSI image has already been created since it will be integrated into the Medium's overlay.

4.3 Using the Live Medium