Creating SSI Clusters Using UML HOWTO

Brian J. Watson

Brian.J.Watson@hp.com

Revision History

Revision 1.04	2002−05−29	Revised by: bjw
LDP review		
Revision 1.03	2002−05−23	Revised by: bpm
LDP review		
Revision 1.02	2002−05−13	Revised by: bjw
Fixed minor typos and errors		
Revision 1.00	2002−05−09	Revised by: bjw
Initial release		

This is a description of how to create a Single System Image (SSI) cluster of virtual User−Mode Linux (UML) machines. After explaining how to use the pre−built SSI/UML binaries, this document demonstrates what an SSI cluster can do. Then it shows more advanced users how to build their own SSI/UML kernels, ramdisks and root images. Following that, it provides an overview of how to move to a hardware−based SSI cluster. It concludes with a set of links and an invitation to contribute to the SSI Clustering project.

Table of Contents

Table of Contents

1. Introduction

An SSI cluster is a collection of computers that work together as if they are a single highly−available supercomputer. There are at least three reasons to create an SSI cluster of virtual UML machines.

- Allow new users to easily experiment with SSI clustering, before investing time and hardware resources into creating a hardware−based cluster.
- Provide a friendly testing and debugging environment for SSI developers.
- Let developers test hundred−node clusters with only ten or so physical machines.

1.1. Overview of SSI Clustering

The *raison d'être* of the SSI Clustering project is to provide a full, highly available SSI environment for Linux. Goals for this project include availability, scalability and manageability, using standard servers. Technology pieces include: membership, single root and single init, single process space and process migration, load leveling, single IPC, device and networking space, and single management space.

The SSI project was seeded with HP's NonStop Clusters for UnixWare (NSC) technology. It also leverages other open source technologies, such as Cluster Infrastructure (CI), Global File System (GFS), keepalive/spawndaemon, Linux Virtual Server (LVS), and the Mosix load−leveler, to create the best general−purpose clustering environment on Linux.

1.1.1. Cluster Infrastructure (CI)

The CI project is developing a common infrastructure for Linux clustering by extending the Cluster Membership Subsystem (CLMS) and Internode Communication Subsystem (ICS) from HP's NonStop Clusters for Unixware (NSC) code base.

1.1.2. Global File System (GFS)

GFS is a parallel physical file system for Linux. It allows multiple computers to simultaneously share a single drive. The SSI Clustering project uses GFS for its single, shared root. GFS was originally developed and open−sourced by Sistina Software. Later they decided to close the GFS source, which prompted the creation of the OpenGFS project to maintain a version of GFS that is still under the GPL.

1.1.3. Keepalive/Spawndaemon

keepalive is a process monitoring and restart daemon that was ported from HP's Non−Stop Clusters for UnixWare (NSC). It offers significantly more flexibility than the `respawn` feature of **init**.

spawndaemon provides a command−line interface for **keepalive**. It's used to control which processes **keepalive** monitors, along with various other parameters related to monitoring and restart.

Keepalive/spawndaemon is currently incompatible with the GFS shared root. **keepalive** makes use of shared writable memory mapped files, which OpenGFS does not yet support. It's only mentioned for the sake of completeness.

1.1.4. Linux Virtual Server (LVS)

LVS allows you to build highly scalable and highly available network services over a set of cluster nodes. LVS offers various ways to load−balance connections (e.g., round−robin, least connection, etc.) across the cluster. The whole cluster is known to the outside world by a single IP address.

The SSI project will become more tightly integrated with LVS in the future. An advantage will be greatly reduced administrative overhead, because SSI kernels have the information necessary to automate most LVS configuration. Another advantage will be that the SSI environment allows much tighter coordination among server nodes.

LVS support is turned off in the current binary release of SSI/UML. To experiment with it you must build your own kernel as described in Section 4.

1.1.5. Mosix Load−Leveler

The Mosix load−leveler provides automatic load−balancing within a cluster. Using the Mosix algorithms, the load of each node is calculated and compared to the loads of the other nodes in the cluster. If it's determined that a node is overloaded, the load−leveler chooses a process to migrate to the best underloaded node.

Only the load−leveling algorithms have been taken from Mosix. The SSI Clustering project is using its own process migration model, membership mechanism and information sharing scheme.

The Mosix load−leveler is turned off in the current binary release of SSI/UML. To experiment with it you must build your own kernel as described in Section 4.

1.2. Overview of UML

User−Mode Linux (UML) allows you to run one or more virtual Linux machines on a host Linux system. It includes virtual block, network, and serial devices to provide an environment that is almost as full−featured as a hardware−based machine.

1.3. Intended Audience

The following are various cluster types found in use today. If you use or intend to use one of these cluster types, you may want to consider SSI clustering as an alternative or addition.

- High performance (HP) clusters, typified by Beowulf clusters, are constructed to run parallel programs (weather simulations, data mining, etc.).
- Load−leveling clusters, typified by Mosix, are constructed to allow a user on one node to spread his

workload transparently across all nodes in the cluster. This can be very useful for compute intensive, long running jobs that aren't massively parallel.
- Web−service clusters, typified by the Linux Virtual Server (LVS) project and Piranha, do a different kind of load leveling. Incoming web service requests are load−leveled by a front end system across a set of standard servers.
- Storage clusters, typified by Sistina's GFS and the OpenGFS project, consist of nodes which supply parallel, coherent, and highly available access to filesystem data.
- Database clusters, typified by Oracle 9I RAC (formerly Oracle Parallel Server), consist of nodes which supply parallel, coherent, and HA access to a database.
- High Availability clusters, typified by Lifekeeper, FailSafe and Heartbeat, are also often known as failover clusters. Resources, most importantly applications and nodes, are monitored. When a failure is detected, scripts are used to fail over IP addresses, disks, and filesystems, as well as restarting applications.

For more information about how SSI clustering compares to the cluster types above, read Bruce Walker's Introduction to Single System Image Clustering.

1.4. System Requirements

To create an SSI cluster of virtual UML machines, you need an Intel x86−based computer running any Linux distribution with a 2.2.15 or later kernel. About two gigabytes of available hard drive space are needed for each node's swap space, the original disk image, and its working copy.

A reasonably fast processor and sufficient memory are necessary to ensure good performance while running several virtual machines. The systems I've used so far have not performed well.

One was a 400 MHz PII with 192 MB of memory running Sawfish as its window manager. Bringing up a three node cluster was quite slow and sometimes failed, maybe due to problems with memory pressure in either UML or the UML port of SSI.

Another was a two−way 200 MHz Pentium Pro with 192 MB of memory that used a second machine as its X server. A three node cluster booted quicker and failed less often, but performance was still less than satisfactory.

More testing is needed to know what the appropriate system requirements are. User feedback would be most useful, and can be sent to <ssic−linux−devel@lists.sf.net>.

1.5. New Versions

The latest version of this HOWTO will always be made available on the SSI project website, in a variety of formats:

- HTML
- PDF
- SGML source

1.6. Feedback

Feedback is most certainly welcome for this document. Please send your additions, comments and criticisms to the following email address: <ssic-linux-devel@lists.sf.net>.

1.7. Copyright Information

1.8. Disclaimer

No liability for the contents of this documents can be accepted. Use the concepts, examples and other content at your own risk. As this is a new edition of this document, there may be errors and inaccuracies, that may of course be damaging to your system. Proceed with caution, and although this is highly unlikely, the author(s) do not take any responsibility for that.

All copyrights are held by their by their respective owners, unless specifically noted otherwise. Use of a term in this document should not be regarded as affecting the validity of any trademark or service mark.

Naming of particular products or brands should not be seen as endorsements.

You are strongly recommended to make a backup of your system before major installations, and back up at regular intervals.

2. Getting Started

This section is a quick start guide for installing and running an SSI cluster of virtual UML machines. The most time−consuming part of this procedure is downloading the root image.

2.1. Root Image

First you need to download a SSI−ready root image. The compressed image weighs in at over 150MB, which will take more than six hours to download over a 56K modem, or about 45 minutes over a 500K broadband connection.

The image is based on Red Hat 7.2. This means the virtual SSI cluster will be running Red Hat, but it does not matter which distribution you run on the host system. A more advanced user can make a new root image based on another distribution. This is described in Section 5.

After downloading the root image, extract and install it.

```
host$ tar jxvf ~/ssiuml−root−rh72−0.6.5−1.tar.bz2
host$ su
host# cd ssiuml−root−rh72
host# make install
host# Ctrl−D
```

2.2. UML Utilities

Download the UML utilities. Extract, build, and install them.

```
host$ tar jxvf ~/uml_utilities_20020428.tar.bz2
host$ su
host# cd tools
host# make install
host# Ctrl−D
```

2.3. SSI/UML Utilities

Download the SSI/UML utilities. Extract, build, and install them.

```
host$ tar jxvf ~/ssiuml−utils−0.6.5−1.tar.bz2
host$ su
host# cd ssiuml−utils
host# make install
host# Ctrl−D
```

2.4. Booting the Cluster

Assuming X Windows is running or the *DISPLAY* variable is set to an available X server, start a two node cluster with

```
host$ ssi-start 2
```

This command boots nodes 1 and 2. It displays each console in a new xterm. The nodes run through their early kernel initialization, then seek each other out and form an SSI cluster before booting the rest of the way. If you're anxious to see what an SSI cluster can do, skip ahead to Section 3.

You'll probably notice that two other consoles are started. One is the lock server node, which is an artefact of how the GFS shared root is implemented at this time. The console is not a node in the cluster, and it won't give you a login prompt. For more information about the lock server, see Section 7.3. The other console is for the UML virtual networking switch daemon. It won't give you a prompt, either.

Note that only one SSI/UML cluster can be running at a time, although it can be run as a non-root user.

The argument to **ssi-start** is the number of nodes that should be in the cluster. It must be a number between 1 and 15. If this argument is omitted, it defaults to 3. The fifteen node limit is arbitrary, and can be easily increased in future releases.

To substitute your own SSI/UML files for the ones in `/usr/local/lib` and `/usr/local/bin`, provide your pathnames in `~/.ssiuml/ssiuml.conf`. Values to override are *KERNEL*, *ROOT*, *CIDEV*, *INITRD*, and *INITRD_MEMEXP*. This feature is only needed by an advanced user.

2.5. Booting an Individual Node

Add nodes 3 and 5 to the cluster with

```
host$ ssi-add 3 5
```

The arguments taken by **ssi-add** are an arbitrary list of node numbers. The node numbers must be between 1 and 15. At least one node number must be provided. For any node that is already up, **ssi-add** ignores it and moves on to the next argument in the list.

2.6. Crashing an Individual Node

Simulate a crash of node 3 with

```
host$ ssi-rm 3
```

Note that this command does not inform the other nodes about the crash. They must discover it through the cluster's node monitoring mechanism.

The arguments taken by **ssi−rm** are an arbitrary list of node numbers. At least one node number must be provided.

2.7. Shutting Down the Cluster

You can take down the entire cluster at once with

```
host$ ssi−stop
```

If **ssi−stop** hangs, interrupt it and shoot all the **linux−ssi** processes before trying again.

```
host$ killall −9 linux−ssi
host$ ssi−stop
```

Eventually, it should be possible to take down the cluster by running **shutdown** as root on any one of its consoles. This does not work just yet.

3. Playing Around

Bring up a three node cluster with **ssi-start**. Log in to all three consoles as **root**. The initial password is **root**, but you'll be forced to change it the first time you log in.

The following demos should familiarize you with what an SSI cluster can do.

3.1. Process Movement, Inheriting Open Files and Devices

Start **dbdemo** on node 1.

```
node1# cd ~/dbdemo
node1# ./dbdemo alphabet
```

The **dbdemo** program "processes" records from the file given as an argument. In this case, it's `alphabet`, which contains the ICAO alphabet used by aviators. For each record, **dbdemo** writes the data to its terminal device and spins in a busy loop for a second to simulate an intensive calculation.

The **dbdemo** program is also listening on its terminal device for certain command keys.

Table 1. Command Keys for dbdemo

Key	Description
1–9	move to that node and continue with the next record
Enter	periodically moves to a random node until you press a key
q	quit

Move **dbdemo** to different nodes. Note that it continues to send output to the console where it was started, and that it continues to respond to keypresses from that console. This demonstrates that although the process is running on another node, it can remotely read and write the device it had open.

Also note that when a process moves, it preserves its file offsets. After moving, **dbdemo** continues processing records from `alphabet` as if nothing had happened.

To confirm that the process moved to a new node, get its PID and use **where_pid**. You can do this on any node.

```
node3# ps -ef | grep dbdemo
node3# where_pid <pid>
2
```

If you like, you can download the source for **dbdemo**. It's also available as a tarball in the `/root/dbdemo` directory.

3.2. Clusterwide PIDs, Distributed Process Relationships and Access, Clusterwide Job Control and Single Root

From node 1's console, start up **vi** on node 2. The **onnode** command uses the SSI kernel's `rexec` system call to remotely execute **vi**.

```
node1# onnode 2 vi /tmp/newfile
```

Confirm that it's on node 2 with **where_pid**. You need to get its PID first.

```
node3# ps –ef | grep vi
node3# where_pid <pid>
2
```

Type some text and save your work. On node 3, **cat** the file to see the contents. This demonstrates the single root file system.

```
node3# cat /tmp/newfile
some text
```

From node 3, kill the **vi** session running on node 2. You should see control of node 1's console given back to the shell.

```
node3# kill <pid>
```

3.3. Clusterwide FIFOs

Make a FIFO on the shared root.

```
node1# mkfifo /fifo
```

echo something into the FIFO on node 1.

```
node1# echo something >/fifo
```

cat the FIFO on node 2.

```
node2# cat /fifo
something
```

```
```

This demostrates that FIFOs are clusterwide and remotely accessible.

3.4. Clusterwide Device Naming and Access

On node 3, write "Hello World" to the console of node 1.

```
node3# echo "Hello World" >/devfs/node1/console
```

This shows that devices can be remotely accessed from anywhere in the cluster. Eventually, the node−specific subdirectories of `/devfs` will be merged together into a single device tree that can be mounted on `/dev` without confusing non−cluster aware applications.

4. Building a Kernel and Ramdisk

Building your own kernel and ramdisk is necessary if you want to

- customize the kernel configuration,
- keep up with the absolute latest SSI code available through CVS,
- or test your SSI bugfix or kernel enhancement with UML.

Otherwise, feel free to skip this section.

4.1. Getting SSI Source

SSI source code is available as official release tarballs and through CVS. The CVS repository contains the latest, bleeding−edge code. It can be less stable than the official release, but it has features and bugfixes that the release does not have.

4.1.1. Official Release

The latest SSI release can be found at the top of this release list. At the time of this writing, the latest release is 0.6.5.

Download the latest release. Extract it.

```
host$ tar jxvf ~/ssi−linux−2.4.16−v0.6.5.tar.bz2
```

Determine the corresponding kernel version number from the release name. It appears before the SSI version number. For the 0.6.5 release, the corresponding kernel version is 2.4.16.

4.1.2. CVS Checkout

Follow these instructions to do a CVS checkout of the latest SSI code. The modulename is *ssic−linux*.

You also need to check out the latest CI code. Follow these instructions to do that. The modulename is *ci−linux*.

To do a developer checkout, you must be a CI or SSI developer. If you are interested in becoming a developer, read Section 8.3 and Section 8.4.

Determine the corresponding kernel version with

```
host$ head −4 ssic−linux/ssi−kernel/Makefile
VERSION = 2
PATCHLEVEL = 4
SUBLEVEL = 16
EXTRAVERSION =
```

```
```

In this case, the corresponding kernel version is 2.4.16. If you're paranoid, you might want to make sure the corresponding kernel version for CI is the same.

```
host$ head -4 ci-linux/ci-kernel/Makefile
VERSION = 2
PATCHLEVEL = 4
SUBLEVEL = 16
EXTRAVERSION =
```

They will only differ when I'm merging them up to a new kernel version. There is a window between checking in the new CI code and the new SSI code. I'll do my best to minimize that window. If you happen to see it, wait a few hours, then update your sandboxes.

```
host$ cd ssic-linux
host$ cvs up -d
host$ cd ../ci-linux
host$ cvs up -d
host$ cd ..
```

4.2. Getting the Base Kernel

Download the appropriate kernel source. Get the version you determined in Section 4.1. Kernel source can be found on this U.S. server or any one of these mirrors around the world.

Extract the source. This will take a little time.

```
host$ tar jxvf ~/linux-2.4.16.tar.bz2
```

or

```
host$ tar zxvf ~/linux-2.4.16.tar.gz
```

4.3. Applying SSI Kernel Code

Follow the appropriate instructions, based on whether you downloaded an official SSI release or did a CVS checkout.

4.3.1. Official Release

Apply the patch in the SSI source tree.

```
host$ cd linux
host$ patch -p1 <../ssi-linux-2.4.16-v0.6.5/ssi-linux-2.4.16-v0.6.5.patch
```

4.3.2. CVS Checkout

Apply the UML patch from either the CI or SSI sandbox. It will fail on patching `Makefile`. Don't worry about this.

```
host$ cd linux
host$ patch -p1 <../ssic-linux/3rd-party/uml-patch-2.4.18-22
```

Copy CI and SSI code into place.

```
host$ cp -alf ../ssic-linux/ssi-kernel/. .
host$ cp -alf ../ci-linux/ci-kernel/. .
```

Apply the GFS patch from the SSI sandbox.

```
host$ patch -p1 <../ssic-linux/3rd-party/opengfs-ssi.patch
```

Apply any other patch from `ssic-linux/3rd-party` at your discretion. They haven't been tested much or at all in the UML environment. The KDB patch is rather useless in this environment.

4.4. Building the Kernel

Configure the kernel with the provided configuration file. The following commands assume you are still in the kernel source directory.

```
host$ cp config.uml .config
host$ make oldconfig ARCH=um
```

Build the kernel image and modules.

```
host$ make dep linux modules ARCH=um
```

4.5. Adding GFS Support to the Host

To install the kernel you must be able to loopback mount the GFS root image. You need to do a few things to the host system to make that possible.

Download any version of OpenGFS *after* 0.0.92, or check out the latest source from CVS.

Apply the appropriate kernel patches from the `kernel_patches` directory to your kernel source tree. Make sure you enable the /dev filesystem, but do *not* have it automatically mount at boot. (When you configure the kernel select 'File systems −> /dev filesystem support' and unselect 'File systems −> /dev filesystem support −> Automatically mount at boot'.) Build the kernel as usual, install it, rewrite your boot block and reboot.

Configure, build and install the GFS modules and utilities.

```
host$ cd opengfs
host$ ./autogen.sh --with-linux_srcdir=host_kernel_source_tree
host$ make
host$ su
host# make install
```

Configure two aliases for one of the host's network devices. The first alias should be 192.168.50.1, and the other should be 192.168.50.101. Both should have a netmask of 255.255.255.0.

```
host# ifconfig eth0:0 192.168.50.1 netmask 255.255.255.0
host# ifconfig eth0:1 192.168.50.101 netmask 255.255.255.0
```

cat the contents of `/proc/partitions`. Select two device names that you're not using for anything else, and make two loopback devices with their names. For example:

```
host# mknod /dev/ide/host0/bus0/target0/lun0/part1 b 7 1
host# mknod /dev/ide/host0/bus0/target0/lun0/part2 b 7 2
```

Finally, load the necessary GFS modules and start the lock server daemon.

```
host# modprobe gfs
host# modprobe memexp
host# memexpd
host# Ctrl-D
```

Your host system now has GFS support.

4.6. Installing the Kernel

Loopback mount the shared root.

```
host$ su
host# losetup /dev/loop1 root_cidev
host# losetup /dev/loop2 root_fs
host# passemble
host# mount -t gfs -o hostdata=192.168.50.1 /dev/pool/pool0 /mnt
```

Install the modules into the root image.

```
host# make modules_install ARCH=um INSTALL_MOD_PATH=/mnt
host# Ctrl-D
```

4.7. Building GFS for UML

You have to repeat some of the steps you did in Section 4.5. Extract another copy of the OpenGFS source. Call it `opengfs-uml`. Add the following line to `make/modules.mk.in`.

```
 KSRC           := /root/linux-ssi

 INCL_FLAGS     := -I. -I.. -I$(GFS_ROOT)/src/include -I$(KSRC)/include \
+                   -I$(KSRC)/arch/um/include \
                    $(EXTRA_INCL)
 DEF_FLAGS      := -D__KERNEL__ -DMODULE  $(EXTRA_FLAGS)
 OPT_FLAGS      := -O2 -fomit-frame-pointer
```

Configure, build and install the GFS modules and utilities for UML.

```
host$ cd opengfs-uml
host$ ./autogen.sh --with-linux_srcdir=UML_kernel_source_tree
host$ make
host$ su
host# make install DESTDIR=/mnt
```

4.8. Building the Ramdisk

Change root into the loopback mounted root image, and use the `--uml` argument to **cluster_mkinitrd** to build a ramdisk.

```
host# /usr/sbin/chroot /mnt
host# cluster_mkinitrd --uml initrd-ssi.img 2.4.16-21um
```

Move the new ramdisk out of the root image, and assign ownership to the appropriate user. Wrap things up.

```
host# mv /mnt/initrd-ssi.img ~username
host# chown username ~username/initrd-ssi.img
host# umount /mnt
host# passemble -r all
host# losetup -d /dev/loop1
host# losetup -d /dev/loop2
host# Ctrl-D
host$ cd ..
```

4.9. Booting the Cluster

Pass the new kernel and ramdisk images into **ssi−start** with the appropriate pathnames for *KERNEL* and *INITRD* in `~/.ssiuml/ssiuml.conf`. An example for *KERNEL* would be `~/linux/linux`. An example for *INITRD* would be `~/initrd−ssi.img`.

Stop the currently running cluster and start again.

```
host$ ssi−stop
host$ ssi−start
```

You should see a three−node cluster booting with your new kernel. Feel free to take it through the exercises in Section 3 to make sure it's working correctly.

---

5. Building a Root Image

Building your own root image is necessary if you want to use a distribution other than Red Hat 7.2. Otherwise, feel free to skip this section.

These instructions describe how to build a Red Hat 7.2 image. At the end of this section is a brief discussion of how other distributions might differ. Building a root image for another distribution is left as an exercise for the reader.

5.1. Base Root Image

Download the Red Hat 7.2 root image from the User−Mode Linux (UML) project. As with the root image you downloaded in Section 2.1, it is over 150MB.

Extract the image.

```
host$ bunzip2 −c root_fs.rh72.pristine.bz2 >root_fs.ext2
```

Loopback mount the image.

```
host$ su
host# mkdir /mnt.ext2
host# mount root_fs.ext2 /mnt.ext2 −o loop,ro
```

5.2. GFS Root Image

Make a blank GFS root image. You also need to create an accompanying lock table image. Be sure you've added support for GFS to your host system by following the instructions in Section 4.5.

```
host# dd of=root_cidev bs=1024 seek=4096 count=0
host# dd of=root_fs bs=1024 seek=2097152 count=0
host# chmod a+w root_cidev root_fs
host# losetup /dev/loop1 root_cidev
host# losetup /dev/loop2 root_fs
```

Enter the following pool information into a file named `pool0cidev.cf`.

```
poolname pool0cidev
subpools 1
subpool 0 0 1 gfs_data
pooldevice 0 0 /dev/loop1 0
```

Enter the following pool information into a file named `pool0.cf`.

```
poolname pool0
subpools 1
subpool 0 0 1 gfs_data
pooldevice 0 0 /dev/loop2 0
```

Write the pool information to the loopback devices.

```
host# ptool pool0cidev.cf
host# ptool pool0.cf
```

Create the pool devices.

```
host# passemble
```

Enter the following lock table into a file named `gfscf.cf`.

```
datadev:        /dev/pool/pool0
cidev:          /dev/pool/pool0cidev
lockdev:        192.168.50.101:15697
cbport:         3001
timeout:        30
STOMITH: NUN
name:none
node: 192.168.50.1      1       SM: none
node: 192.168.50.2      2       SM: none
node: 192.168.50.3      3       SM: none
node: 192.168.50.4      4       SM: none
node: 192.168.50.5      5       SM: none
node: 192.168.50.6      6       SM: none
node: 192.168.50.7      7       SM: none
node: 192.168.50.8      8       SM: none
node: 192.168.50.9      9       SM: none
node: 192.168.50.10     10      SM: none
node: 192.168.50.11     11      SM: none
node: 192.168.50.12     12      SM: none
node: 192.168.50.13     13      SM: none
node: 192.168.50.14     14      SM: none
node: 192.168.50.15     15      SM: none
```

Write the lock table to the cidev pool device.

```
host# gfsconf −c gfscf.cf
```

Format the root disk image.

```
host# mkfs_gfs −p memexp −t /dev/pool/pool0cidev −j 15 −J 32 −i /dev/pool/pool0
```

Mount the root image.

```
host# mount −t gfs −o hostdata=192.168.50.1 /dev/pool/pool0 /mnt
```

```
```

Copy the ext2 root to the GFS image.

```
host# cp -a /mnt.ext2/. /mnt
```

Clean up.

```
host# umount /mnt.ext2
host# rmdir /mnt.ext2
host# Ctrl-D
host$ rm root_fs.ext2
```

5.3. Getting Cluster Tools Source

Cluster Tools source code is available as official release tarballs and through CVS. The CVS repository contains the latest, bleeding-edge code. It can be less stable than the official release, but it has features and bugfixes that the release does not have.

5.3.1. Official Release

The latest release can be found at the top of the Cluster-Tools section of this release list. At the time of this writing, the latest release is 0.6.5.

Download the latest release. Extract it.

```
host$ tar jxvf ~/cluster-tools-0.6.5.tar.bz2
```

5.3.2. CVS Checkout

Follow these instructions to do a CVS checkout of the latest Cluster Tools code. The modulename is *cluster-tools*.

To do a developer checkout, you must be a CI developer. If you are interested in becoming a developer, read Section 8.3 and Section 8.4.

5.4. Building and Installing Cluster Tools

```
host$ su
host# cd cluster-tools
host# make install_ssi_redhat UML_ROOT=/mnt
```

5.5. Installing Kernel Modules

If you built a kernel, as described in Section 4, then follow the instructions in Section 4.4 and Section 4.7 to install kernel and GFS modules onto your new root.

Otherwise, mount the old root image and copy the modules directory from `/mnt/lib/modules`. Then remount the new root image and copy the modules into it.

5.6. Configuring the Root

Remake the ubd devices. At some point, the UML team switched the device numbering scheme from *98,1* for `dev/ubd/1`, *98,2* for `dev/ubd/2`, etc. Now they use *98,16* for `dev/ubd/1`, *98,32* for `dev/ubd/2`, etc.

Comment and uncomment the appropriate lines in `/mnt/etc/inittab.ssi`. Search for the phrase 'For UML' to see which lines to change. Basically, you should disable the DHCP daemon, and change the getty to use `tty0` rather than `tty1`.

You may want to strip down the operating system so that it boots quicker. For the prepackaged root image, I removed the following files.

```
/etc/rc3.d/S25netfs
/etc/rc3.d/S50snmpd
/etc/rc3.d/S55named
/etc/rc3.d/S55sshd
/etc/rc3.d/S56xinetd
/etc/rc3.d/S80sendmail
/etc/rc3.d/S85gpm
/etc/rc3.d/S85httpd
/etc/rc3.d/S90crond
/etc/rc3.d/S90squid
/etc/rc3.d/S90xfs
/etc/rc3.d/S91smb
/etc/rc3.d/S95innd
```

You might also want to copy **dbdemo** and its associated `alphabet` file into `/root/dbdemo`. This lets you run the demo described in Section 3.1.

5.7. Unmounting the Root Image

```
host# umount /mnt
host# passemble -r all
host# losetup -d /dev/loop1
host# losetup -d /dev/loop2
```

5.8. Distributions Other Than Red Hat

Cluster Tools has make rules for Caldera and Debian, in addition to Red Hat. Respectively, the rules are *install_ssi_caldera* and *install_ssi_debian*.

The main difference between the distributions is the `/etc/inittab.ssi` installed. It is the inittab used by the clusterized **init.ssi** program. It is based on the distribution's `/etc/inittab`, but has some cluster−specific enhancements that are recognized by **init.ssi**.

There is also some logic in the `/etc/rc.d/rc.nodeup` script to detect which distribution it's on. This script is run whenever a node joins the cluster, and it needs to do different things for different distributions.

Finally, there are some modifications to the networking scripts to prevent them from tromping on the cluster interconnect configuration. They're a short−term hack, and they've only been implemented for Red Hat so far. The modified files are `/etc/sysconfig/network−scripts/ifcfg−eth0` and `/etc/sysconfig/network−scripts/network−functions`.

6. Moving to a Hardware−Based Cluster

If you plan to use SSI clustering in a production system, you probably want to move to a hardware−based cluster. That way you can take advantage of the high−availability and scalability that a hardware−based SSI cluster can offer.

Hardware−based SSI clusters have significantly higher availability. If a UML host kernel panics, or the host machine has a hardware failure, its UML−based SSI cluster goes down. On the other hand, if one of the SSI kernels panic, or one of the hardware−based nodes has a failure, the cluster continues to run. Centralized kernel services can failover to a new node, and critical user−mode programs can be restarted by the application monitoring and restart daemon.

Hardware−based SSI clusters also have significantly higher scalability. Each node has one or more CPUs that truly work in parallel, whereas a UML−based cluster merely simulates having multiple nodes by time−sharing on the host machine's CPUs. Adding nodes to a hardware−based cluster increases the volume of work it can handle, but adding nodes to a UML−based cluster bogs it down with more processes to run on the same number of CPUs.

6.1. Requirements

You can build hardware−based SSI clusters with x86 or Alpha machines. More architectures, such as IA64, may be added in the future. Note that an SSI cluster must be homogeneous. You cannot mix architectures in the same cluster.

The cluster interconnect must support TCP/IP networking. 100 Mbps ethernet is acceptable. For security reasons, it should be a private network. Each node should have a second network interface for external traffic.

Right now, the most expensive requirement of an SSI cluster is the shared drive, required for the shared GFS root. This will no longer be a requirement when CFS, which is described below, is available. The typical configuration for the shared drive is a hardware RAID disk cabinet attached to all nodes with a Fibre Channel SAN. For a two−node cluster, it is also possible to use shared SCSI, but it is not directly supported by the current cluster management tools.

The GFS shared root also requires one Linux machine outside of the cluster to be the lock server. It need not be the same architecture as the nodes in the cluster. It just has to run **memexpd**, a user−mode daemon. Eventually, GFS will work with a Distributed Lock Manager (DLM). This would eliminate the need for the external lock server, which is a single point of failure. It could also free up the machine to be another node in your cluster.

In the near future, the Cluster File System (CFS) will be an option for the shared root. It is a stateful NFS that uses a token mechanism to provide tight coherency guarantees. With CFS, the shared root can be stored on the internal disk of one of the nodes. The on−disk format can be any journalling file system, such as ext3 or ReiserFS.

The initial version of CFS will not provide high availability. Future versions of CFS will allow the root to be mirrored across the internal disks of two nodes. A technology such as the Distributed Replicated Block Device (DRBD) would be used for this. This is a low−cost solution for the shared root, although it has a performance penalty.

Future versions will also allow the root to be stored on a disk shared by two or more nodes, but not necessarily shared by all nodes. If the CFS server node crashes, its responsibilities would failover to another node attached to the shared disk.

6.2. Resources

Start with the installation instructions for SSI.

If you'd like to install SSI from CVS code, follow these instructions to checkout modulename *ssic-linux*, and these instructions to checkout modulenames *ci-linux* and *cluster-tools*. Read the `INSTALL` and `INSTALL.cvs` files in both the `ci-linux` and `ssic-linux` sandboxes. Also look at the `README` file in the `cluster-tools` sandbox.

For more information, read Section 7.

7. Further Information

Here are some links to information on SSI clusters, CI clusters, GFS, UML, and other clustering projects.

7.1. SSI Clusters

Start with the SSI project homepage. In particular, the documentation may be of interest. The SourceForge project summary page also has some useful information.

If you have a question or concern, post it to the <ssic-linux-devel@lists.sf.net> mailing list. If you'd like to subscribe, you can do so through this web form.

If you are working from a CVS sandbox, you may also want to sign up for the ssic-linux-checkins mailing list to receive checkin notices. You can do that through this web form.

7.2. CI Clusters

Start with the CI project homepage. In particular, the documentation may be of interest. The SourceForge project summary page also has some useful information.

If you have a question or concern, post it to the <ci-linux-devel@lists.sf.net> mailing list. If you'd like to subscribe, you can do so through this web form.

If you are working from a CVS sandbox, you may also want to sign up for the ci-linux-checkins mailing list to receive checkin notices. You can do that through this web form.

7.3. GFS

SSI clustering currently depends on the Global File System (GFS) to provide a single root. The open-source version of GFS is maintained by the OpenGFS project. They also have a SourceForge project summary page.

Right now, GFS requires either a DMEP-equipped shared drive or a lock server outside the cluster. The lock server is the only software solution for coordinating disk access, and it is not truly HA. There are plans to make OpenGFS support IBM's Distributed Lock Manager (DLM), which would distribute the lock server's responsibilities across all the nodes in the cluster. If any node fails, the locks it managed would failover to other nodes. This would be a true HA software solution for coordinating disk access.

If you have a question or concern, post it to the <opengfs-users@lists.sf.net> mailing list. If you'd like to subscribe, you can do so through this web form.

7.4. UML

The User-Mode Linux (UML) project has a homepage and a SourceForge project summary page.

If you have a question or concern, post it to the <user-mode-linux-user@lists.sf.net> mailing list. If you'd like to subscribe, you can do so through this web form.

7.5. Other Clustering Projects

Other clustering projects include Mosix, Linux Virtual Server, Beowulf, HA Linux and FailSafe.

8. Contributing

If you'd like to contribute to the SSI project, you can do so by testing it, writing documentation, fixing bugs, or working on new features.

8.1. Testing

While using the SSI clustering software, you may run into bugs or features that don't work as well as they should. If so, browse the SSI and CI bug databases to see if someone has seen the same problem. If not, either post a bug yourself or post a message to <ssic-linux-devel@lists.sf.net> to discuss the issue further.

It is important to be as specific as you can in your bug report or posting. Simply saying that the SSI kernel doesn't boot or that it panics is not enough information to diagnose your problem.

8.2. Documentation

There is already some documentation for SSI and CI, but more would certainly be welcome. If you'd like to write instructions for users or internals documentation for developers, post a message to <ssic-linux-devel@lists.sf.net> to express your interest.

8.3. Debugging

Debugging is a great way to get your feet wet as a developer. Browse the SSI and CI bug databases to see what problems need to be fixed. If a bug looks interesting, but is assigned to a developer, contact them to see if they are actually working on it.

After fixing the problem, send your patch to <ssic-linux-devel@lists.sf.net> or <ci-linux-devel@lists.sf.net>. If it looks good, a developer will check it into the repository. After submitting a few patches, you'll probably be invited to become a developer yourself. Then you'll be able to checkin your own work.

8.4. Adding New Features

After fixing a bug or two, you may be inclined to work on enhancing or adding an SSI feature. You can look over the SSI and CI project lists for ideas, or you can suggest something of your own. Before you start working on a feature, discuss it first on <ssic-linux-devel@lists.sf.net> or <ci-linux-devel@lists.sf.net>.

9. Concluding Remarks

Hopefully, you find SSI clustering technology to be useful for your application that demands availability, scalability, and manageability at the same time.

If you have any questions or comments, don't hesitate to post them to <ssic-linux-devel@lists.sf.net>.