System Installation Suite
Massive Installation for Linux

Sean Dague
japh@us.ibm.com

Abstract

The first hurdle that a user or administrator must overcome when migrating to Linux is the installation. In the not so distant past, this was a near Herculean task. Today, with a myriad of Linux distributions available, many focusing on the end user experience, installation of a single machine has become much easier. In some instances it is even Mom proof. This has given rise to a new issue, however, as these methods of installation tend to be distribution specific, and tend to have a single machine view of the world.

System Installation Suite attempts to solve the massive installation problem, i.e. how does an administrator handle installation and maintenance of hundreds or thousands of nodes at once, in Linux. The solution is agnostic of Linux distribution and architecture, and presents a uniform interface on every Linux platform. It does this through the creation of installation images, which are built on a centralized server somewhere. These images are then deployed over the network to client machines. The use of installation images, which are in fact fully instantiated Linux systems stored on an image server, gives rise to some interesting possibilities for system management and maintenance. The design process that went into System Installation Suite, and the possibilities that it provides for will be discussed further in this paper.

1 Background

System Installation Suite is a collaboration between two different open source massive installation tools, SystemImager and LUI (the Linux Utility for Cluster Installation). The design of System Installation Suite came largely from harvesting the strengths of both of these tools, while attempting to leave their short comings behind. It is appropriate that we explore some of the strengths and weaknesses of both LUI and SystemImager before delving into the design of System Installation Suite as a whole.

1.1 LUI - Linux Utility for Cluster Installation

The Linux Utility for Cluster Installation (LUI) was one of the first Open Source projects contributed by the IBM Linux Technology Center. The project was started by Rich Ferri to mature the state of Linux clustering. LUI version 1.0 was released to the world in April of 2000 under the GNU Public License.

LUI is a resource based cluster installation tool, conceptually based on NIM (Network Install Manager), the network installer for AIX. In LUI everything was driven by resources. LUI resources included: the list of packages (RPMs) that will be installed on a client, tarballs that would be expanded on the client, disk partition tables that would be used to setup the disks, custom kernels and ramdisks, post install scripts, or single files that would be propagated. A combination of these resources would fully describe the final makeup of a client.

Resources were first abstractly defined in the LUI database. Then clients were abstractly defined in the LUI database. Finally resources were assigned to clients. LUI also supported arbitrary grouping, so resource and client groups could be defined, and the allocation of resource groups to client groups could be utilized.

LUI installation required nodes with network interface cards that could network boot. For clients which did not have network bootable NICs, an floppy from the etherboot project could be made to simulate this process. The network booted kernel had a remote NFS root on the LUI server, and the installation logic was drive by a **clone** program contained within the NFS root.

LUI had many weak spots where things would often break down. The first issue was the reliance on PXE, TFTP, and NFS v2 which are not entirely reliable, secure, or scalable protocols. [1] When network booting worked properly, it was fantastic, when it failed, it was often extremely difficult to debug the failure. This was especially true due to the fact that there are various versions of the PXE standard which behaved slightly differently.

The second major issue was the timing of client instantiation. All the resources were instantiate into a working client machine on the client when running from the network booted kernel and NFS root. Although some sanity checks were run on the resources before they were allowed to be registered, many checks were either too expensive or too complex to be run. The most common failure was having an inconsistent list of RPMs, i.e. one which did not properly satisfy all package dependencies. By the time such an error was detected (during client installation), it was too late to recover gracefully. In a best case scenario, the machine had remote console access to debug the issue. In the more common case, the machine was hung in the middle of an installation, and a monitor and keyboard had to be wheeled over to the node to examine the failure.

The final issue with LUI was overly complicated with its resource model. Once a user understood all the possible resources, how they related, and which ones were really required to bring up a machine, it was great. However this learning curve was often rather steep.

Many of these issues were being looked at for a LUI 2.0 redesign during the spring of 2001. However the interaction with the SystemImager project made the redesign take an entirely different direction.

1.2 SystemImager

SystemImager is a project which was started by Brian Elliot Finley. Its first incarnation was under the name Pterodactyl, where it was a set off programs designed to replicate Solaris installations. It later became a far more robust product firmly rooted in Linux exclusively. SystemImager 1.0 was released in May of 2000 under the GNU Public License.

SystemImager, as the name implies, is an image based installation and maintenance tool. Unlike many other image based tools, SystemImager images are actually full live file systems which exist on the image server. These images are captured from a running machine that has been properly prepared, also known as a **golden client**. The image consists of the entire client file system, and is stored in a directly under the **/var/lib/systemimager/images** tree on the image server. These images can later

[1] Since the time of LUI's introduction, both NFS v3 and more robust implementations of tftp (such as atftpd) have become available on Linux

be deployed to other client machines using the SystemImager autoinstallation process.

The autoinstall process for SystemImager is different from that of LUI. Instead of network booting just a kernel, and using a remote NFS root to drive installation, SystemImager uses an embedded Linux, BOEL (Brian's Own Embedded Linux), to control the installation. The kernel and initial ramdisk for BOEL fit on a floppy, cd, or can be served from the network for machines that support network booting. Once BOEL has brought the client machine onto the network, it reconnects to the image server and fetches an autoinstall shell script. The autoinstall script drives the remainder of the installation. All file transfer between client and server is done using **rsync** [2], which provides a mechanism for remote file synchronization.

The use of live file systems on the server and **rsync** to transfer these file in SystemImager allows for a number of additional features. Because **rsync** can use a number of different underlying file transfer methods, the installation can happen over a secure ssh connection. This allows for unattended installation where client and server are separated by significant physical distance and the only route between them is a public, insecure network. Because images are replicated at a file level and not a package level, the process of file transfer is inherently distribution agnostic. Software on the node needs not have come from a distribution package (RPM, Deb, etc.), but may have been installed from source or binary tarball.

However, SystemImager is not an stand alone installation tool. It doesn't solve the "first node" issue. When using SystemImager for installation, one must first build the golden client using some native installation method. Only after the node is fully installed and configured manually can the image be captured. Before System Installation Suite, SystemImager also did only very minimal customizations to the images after installation. This meant that simple hardware differences like different models of network cards in client machines could require separate images on the server.

[2]http://rsync.samba.org

1.3 System Installation Suite

System Installation Suite takes the best parts of SystemImager and LUI and adds other features that neither of them previously had. The LUI project morphed into a project called SystemInstaller, which uses a much simpler model for instantiating a machine from a set of packages and disk partition files. In System Installation Suite, SystemImager stays largely unchanged from a user perspective, however some of the internals have changed to allow interaction with SystemInstaller, and all of the image customization code was removed from the autoinstallation process and replaced with calls to System Configurator. System Configurator is the third major component of System Installation Suite, a uniform configuration API for installation. It supports features such as network setup, boot loader setup, and ramdisk creation. System Configurator also makes images deployed under System Installation Suite more generic, and applicable to a wider range of hardware.

2 SystemInstaller

SystemInstaller is the image build tool for System Installation Suite. It is very similar to LUI in function, however instead of building clients directly, SystemInstaller builds images on the image server. The images are made to look just as if they were harvested from a live machine that was installed by a native installer.

The design of SystemInstaller came from many lessons learned during the LUI project. The SystemInstaller team attempted to keep all the strengths of LUI without retaining any of the weaknesses.

2.1 SystemInstaller Architecture

The biggest hurdle that any LUI user had to overcome was understanding all the possible resources that existed, and how they interacted with each other. In many instances resources were either co-dependent, redundant, or could have been auto detected and allocated during installation. An example of just such and instance was the ramdisk resource.

Most modern distribution kernels are built extremely modularly. This makes it very easy to use the same kernel on many different types of hardware by only changing the modules.conf file. If the root filesystem of the machine is on a device that needs one or more of these modules to access it, an initial ramdisk needs to be built to support this. LUI did not support the automatic create of an initial ramdisk, and required the user to create, and allocate one manually. If the creation of the initial ramdisk took place on a system running an smp kernel, but was attempted to be used on a up kernel system, the boot would fail. This was a common occurrence for beginner LUI users. Hence one of the early goals of SystemInstaller was to make the resource model much simpler, or even totally transparent to the user.

Even though LUI had weaknesses, it also had many strengths that other install tools did not have. Chief among these was the LUI database. All of the user facing LUI commands did nothing more then add the appropriate meta data into the LUI database. During installation, the **clone** script fetched data from the LUI database and used it to instantiate the resources on the client machines. After installation was done, LUI no longer needed this data at all. However, a decision was made that the data should be persistent, and an API created to access it, in the hopes that some other application might find it useful.

This is exactly what happened. When the OSCAR Clustering project was looking for an installation tool, they choose LUI because it had a cluster database already. OSCAR could just ride on top of LUI's database, and add extra information if required. As OSCAR was one of LUI's biggest users, when looking to replace LUI with System Installation Suite, the concept of a cluster database was a requirement that had to be kept.

With these two key points: simplify the resource model, and provide a cluster database, design on SystemInstaller began. SystemInstaller retained a flat file database, as LUI had, as it was still felt that requiring an SQL database was too significant an overhead for an installation tool. Although this did mean sacrificing some functionality, and the ability for remote access to the data store, it did mean the prerequisites were far lighter. The intent was always to move towards a model where various data stores (flat file, SQL, or even LDAP) could be accessed transparently. However, this goal was a bit beyond the scope of the initial release. In addition to the data store, many convenience functions were defined and exported so that products such as OSCAR would have a clean interface to interacting with System Installation Suite.

SystemInstaller has a far more simple command line interface than LUI had. Only the package list and disk partition table resources were retained. This was possible because the image could be tweaked after initial creation but before deployment manually. The addition of custom kernels, or hand compiled software could be handled at this stage, and did not need to be incorporated into the early SystemInstaller interface. This reduced complexity makes System Installation Suite much easier to use out of the box than LUI ever was.

2.2 SystemInstaller Interface

The SystemInstaller interface consists of a number of non-interactive command line utilities, an interactive **buildimage** program, and a graphical user interface, **tksis** which uses the Perl Tk bindings. Both buildimage and tksis are built on top of the command line interfaces of SystemInstaller and SystemImager. This separation between the functional interface and user interface was extremely important in making it possible for other user interfaces to be built on top of the command line utilities.

There are four extremely important commands in SystemInstaller that do all the real work for creating images and client definitions. These commands are as follows:

- mksiimage - Build an image from a list of packages
- mksidisk - Add disk partition information to an image
- mksimachine - Update, Delete, or Show a machine definition
- mksirange - Create a group of machine entries

The **mksiimage** , **mksidisk** , and **mksimachine** commands are entirely contained within the SystemInstaller package. The **mksirange** command is a wrapper on top of SystemImager's **addclients** command, which also inserts the client definitions into the System Installation Suite database.

Most SystemInstaller commands support the following actions: add (-A), delete (-D), and list (-L). The major exception is the **mksimachine** and **mksirange** combination of commands. This limitation is currently based on limitations of the **addclients** command, specifically its inability to add a single machine definition of arbitrary name.

All of the commands in SystemInstaller serve both manipulate the System Installation Suite database, and perform the actual instantiation of their commands. Because of this, failures in image and client instantiation are immediately returned to the user. The full advantages of this will be discussed later.

3 SystemImager Redux

In order to initially integrate SystemImager into System Installation Suite, only minor enhancements were needed. Many of these enhancements were generally applicable to SystemImager outside of the scope of System Installation Suite, and hence seen as merely additional function to existing SystemImager users.

SystemImager logically breaks up into three components: server, client, and autoinstallation. With the version 2.0 release of SystemImager, which integrated with the other components of System Installation Suite, only elements of the server component were changed. The following is a brief overview of merely the changes in those components. For further information on SystemImager design, please refer to the SystemImager manual.

3.1 SystemImager Server Changes

Before System Installation Suite, SystemImager was a complete product which consisted of a number of command line utilities. Principle among these were two commands that did most of the work on the server.

- getimage - captures images from a prepared client, and creates autoinstall script
- addclients - adds a range of client definitions associated with an image

In order to accommodate SystemInstaller and System Installation Suite the following changes were made:

The **getimage** command was broken into two separate commands. The new **getimage** was responsible only for harvesting the image from a client. This primarily involved rsyncing the entire contents of the golden client node into a directory within /var/lib/systemimager/images. An additional **mkautoinstallscript** command now became the interface for generating the autoinstallation script. This functional separation was needed by SystemInstaller, as its mksiimage and mksidisk commands performed the functional equivalent of getimage. This functional separation also enabled the ability to regenerate autoinstall scripts from within SystemImager. Prior to this change, users had to utilize a documented hack of pointing getimage against the localhost interface.

The second major change was the addition of a non interactive mode to **addclients**. Prior to SIS, addclients could only be run in an interactive mode. Although this provided a simple console user interface that was easy to understand, it hindered the ability for other interfaces to sit above the SystemImager commands. It would have been relatively easy to duplicate the function of addclients within SystemInstaller, however it was it was still felt that it would be extremely beneficial if the exact same math was used to calculate the list of clients irrespective of the interface the user was utilizing. As SystemImager's logic for this existed solely inside **addclients** it was simplest to wrap the **addclients** command from **mksirange**.

In addition to these command line interface changes, the beginnings of a SystemImager library was started. A number of functions were added to this library during the evolution of SystemImager 2.0 which allowed SystemInstaller to add entries to the SystemImager rsyncd.conf file reliably. In SystemImager, this process is handled by the getimage command. When building an image with SystemInstaller, getimage is never called, so this aspect of the SystemImager interface could not be used.

4 Image Deployment

Once an image is either built with SystemInstaller, or harvested with SystemImager, it resides on an image server, generally in /var/lib/systemimager/images. This image is then ready to be deployed to client machines. Although most of the logic for image deployment in System Installation Suite remains unchanged from SystemImager, it is significantly different from most package based install mechanisms that it is worthy of discussion. The differences in image based deployment over package based deployment lead to a number of interesting pros and cons of the two methodologies.

4.1 BOEL

The engine for the autoinstallation process is BOEL (Brian's Own Embedded Linux). BOEL consists of a monolithic Linux Kernel 2.2 and a ramdisk containing Glibc 2.1, BusyBox 0.60.0, rsync 2.4.6, sfdisk, and a number of other utilities. BOEL's entire job is to bring the client machine to a state where it can remotely access the image server, and then it hands off its job to the autoinstallation script. BOEL is based on Tom's Root Boot distribution. More details about BOEL can be found in a recent article in Embedded Linux Journal.

BOEL can be booted from a number of media, including floppy disk, cdrom, network boot, or even the local hard drive in the special case of an autoinstallation update of a client. When booted from a floppy disk or local hard drive, BOEL will attempt to access a local configuration file. This file can contain information about things such as the ip address of the client, the default gateway, and image server ip address. The local configuration file is very useful when doing installations on a network where the installation administrator does not have control over the site dhcp server. If the local configuration file is not available, a dhcp client is used to activate the network devices during boot.

Once the network device is configured, BOEL will attempt to determine which image server it should contact. This information may either be provided via local configuration file or dhcp options. After that, BOEL attempts to determine the host name of the machine it is running on. This is accomplished through either values in the local configuration file, reverse DNS lookup, or a special hosts file served from the image server.

With the network enabled, and the image server and local host name found, BOEL connects to the image server using rsync, and retrieves the autoinstall script for the host. This script is stored in the **scripts** rsync module as the file HOSTNAME.sh. At this point BOEL turns over control to the autoinstall script that it has downloaded.

4.2 The Auto Install Script

BOEL is an extremely constrained environment. It contains only a minimal glibc, a statically linked ash shell, and the BusyBox implementation of standard Linux commands. This means the autoinstall script must be POSIX shell. The tasks the autoinstall script must accomplish are as follows:

1. partition the disk drives with sfdisk
2. format all the partitions appropriately (ext2, ext3, and reiserfs are supported)
3. mount all the partitions to the proper mount points
4. rsync the appropriate rsync module from the server to the local disk
5. run systemconfigurator to setup network, modules.conf, and bootloader
6. execute a specified post install action (one of: beep, reboot, or shutdown)

If at any point the auto installation attempts an operation which fails, it will dump its console out to a shell and await human input. At this point no remote notification is provided in the event of failure.

Because the autoinstall script is merely a POSIX shell script, it can be easily modified to perform other actions beyond the straight scope of the autoinstallation process. The **mkautoinstallscript** command should be considered to generate a template autoinstall script. Although it will work fine for most scenarios without any modification, the possibility exists to easily modify it to add extra function.

4.3 Image Customization

Once the files from the image are transfered to the client machine, the job of installation is nearly complete. The only thing that remains is modifying the abstract image so that it has node specific information in it. For a machine to actually boot and connect to the local area network the network scripts must properly reflect the state of the node, and the bootloader must be installed. Setting up networking is something which tends to be very distribution specific. Making a machine bootable is very architecture specific. Instead of forcing that code into the autoinstall script, where one only has access to the POSIX shell environment, a different approach was taken.

System Installation Suite installs all the software in the image to the client machine. The client machine is a full instantiation of a runnable node. It has all the C, Perl, and Python libraries that a running machine would have. Why not exploit this fact by installing an additional program in the image or on the golden client which is transferred to the client during installation. This program would be called via the **chroot** function, and would present a unified API for configuration of networking, bootloader setup, and other required tasks to the autoinstall script, which would be exactly the same on any architecture or distribution. This program became known as the System Configurator project.

System Configurator implements a unified calling interface to setup both network scripts and boot loaders. In the process of setting up these features, it also can detect local hardware and modify the appropriate underlying files accordingly. This feature is even exploited from SystemImager outside the scope of System Installation Suite. This allows an image to be used on machines which have different network interface cards. Prior to SIS, this was not possible. System Configurator will also auto generate initial ramdisks upon request. This solves the long standing issue with LUI where one had to create an appropriate initial ramdisk if attempting to install with a modular kernel on SCSI hardware.

4.4 Foot Printing

There were many possible ways that System Configurator could have implemented its abstraction. One that was suggested, and firmly rejected, was classifying features in terms of distributions. The problem is to support a dozen or so distributions, over 3 or more releases, means 40+ code paths. Also, significant updates between stable releases of a distribution would be nearly impossible to track or support. The eventual design concept that won was "foot printing".

Let's say you know there are two different programs to create ramdisks, and each of them takes different options. You could either first try to find a comprehensive list of all Linux distributions that use one or the other, and then detect distribution, and use the right one (as stored in your matrix), or you could just say "If program a is there, run it like this, if program b, run it like that". This takes out a whole lot of indirection in detection, and provides for the possibility of supporting distributions that you didn't even know existed.

The idea of foot printing naturally leads to a modular architecture. Each module registers itself for a specific type of job (Network, Bootloader, Hardware, etc.), and when the phase for that job is executed, the modules are asked if their footprint is found. If so, their setup routine is executed. Depending on the type of job being accomplished, it may either be ok to execute all modules which footprint properly, or only the first one to do so. This decision is made per task module.

4.5 System Configurator

System Configurator can do many tasks, and does different tasks when used in a System Installation Suite context or a SystemImager only context. The basic setup directives include support for the following:

- pci hardware detection
- network setup
- initial ramdisk generation
- bootloader configuration file generation
- bootloader setup (running the proper bootloader)
- time zone setup
- network time sync

At every stage the main line code uses footprinting and plugin modules to accomplish tasks. Whenever possible System Configurator calls native setup tools for the distribution it is running on. The attempt is to make it very hard to determine that System Configurator created or modified files, as it did exactly what a native user or tool in the distribution environment would do.

A good example of this is the creation of modules.conf on the Debian distribution. In Debian there is a directory of files in /etc/modutils which are all merged into modules.conf using the **update-modules** command. System Configurator modifies the appropriate files and runs update-modules if the /etc/modutils directory is found.

In the current release of System Configurator 4 different types of networking are supported, which provides support for Red Hat, Mandrake, Conectiva, SuSE, TurboLinux, and Debian. An additional 2 types of networking have been identified that would add support for Caldera and Slackware, but haven't been implemented yet. System Configurator can support as many network adapters as your Linux system can handle, though SystemImager and System Installation Suite only support configuration of the primary network adapter at this time.

System Configurator also supports four different methods for bootloader setup, Lilo and Grub on i386, and two ways to setup Elilo on IA64. There is experimental support for PPC and PA-RISC setup at the moment, though the autoinstallation process does not yet support either of those platforms.

After System Configurator runs, the abstract image has become a real live node, tuned for the distribution and hardware that it is running on. This node will be able to reboot and become network accessible. Once the machine is network accessible, any additional custom setup could be performed by remote shell commands or programs such as cfengine.

5 Maintenance

Images are live file systems stored on an image server. Images get transfered across the network via rsync. Before rsync transfers files it first computes the difference between the source and destination for the files. One image can be applied to many machines.

All these facts taken together paint a picture of how SIS provides an extremely efficient update mechanism for client nodes. Suppose that some core library to your system has a security vulnerability, for instance zlib. Pushing this update to all your running machines is as simple as applying the update to the image, then rsyncing that image back out to the client nodes. In most cases updates of a running node can be performed without a reboot, the notable exceptions being an update to a kernel or commonly used shared library with a security vulnerability.

SystemImager provides an interface to performing this update process via the **updateclient** command. Updateclient is intelligent enough to exclude many directories that are used for variable or node specific information, such as /var/log, /var/run/, /var/spool, /tmp, /proc and ext3 journal files. This list of excludes is stored in a file on the client, so it can be tailored to meet site specific needs.

Because the rsync protocol computes the difference between the source and destination file systems, only those files that have changed get propagated. This reduces network traffic significantly. Rsync can even check for changed byte ranges within a large file, so that it can replicate on a sub file level.

6 Image vs. Client Instantiation

As has been shown in this paper, System Installation Suite takes a novel approach to installation. Most other tools used for unattended massive installation pull packages across the network, and instantiate them directly on the client during install. System Installation Suite does this instantiation on the server, then transfers the resultant image across the network. There are many advantages to this methodology. Maintenance mode, and support for non packaged software have previously been discussed. However there are many other advantages, some of which are still not fully exploited, to this approach.

All installation methods must have some code running on the client node to perform the installation. With SIS, all the code run on the client during install is SIS code. Packages are not allowed to run their own scripts during the portion of installation which occurs on the client. This means there are far less moving parts during the SIS install process, and hence less things that can go wrong with the install. In a package based installation, one bad package can prevent the install from working. With SIS, the one package can be manually force fit into the image where the user has far greater latitude (tools like alien might even be used to install non native packages). The SIS install process doesn't care where the content of the image came from. This reduced complexity during actual installation of the clients translates into a smaller number of problems that can occur during the autoinstall phase. This is true in theory, and in has been shown in practice as well.

One of the things that image installation does not do as well as package based install tools, is conserve disk space. In a package based installation environment the main server stores only packages. When the client installs, it will determine which combination of package it needs to complete installation, and fetch only those packages it needs to complete the process. The space required on the server is all the packages off the distribution CDs, plus any additional update packages. For most distribution releases this will amount to about 3 GiB of disk space per distro, per release, per architecture.

With image based installation the images are fully instantiated and stored on the server. An average image with a full load of software ranges from 1 to 2 GiB of data. There have been thoughts about providing an image normalization tool that would reduce the space required to store multiple images on the server, or to support multiple phase images, where many different images would be overlayed to create the final installation. Neither of these options are being seriously explored at this point because of one important fact: a 60 GiB EIDE drive costs less than $100. Disk space is cheap. Adding complexity to the project to save storage space requirements does not seem like a valuable use of developer resource.

The final advantage that System Installation

Suite's image implementation provides is the ability to live test an image on the server before deployment. The image is a live file system. Any user level program run chrooted from an image will run just as if it was a live machine. If you want to know whether an application will run properly in your image, you can chroot $IMAGEDIR $CMD.

This methodology was used when adding SuSE support to SystemInstaller. I harvested a SuSE 7.2 system onto my Mandrake 8.1 development machine. I then chrooted into the SuSE image and built additional SuSE images from inside it. This meant I could develop for multiple distributions on a single machine without having to reboot. This feature of SystemInstaller has begun to be exploited by a number of users that wish to create test and build environments for many distributions, but have a limited number of physical machines. The only limitation to this is testing software which needs access to physical hardware or kernel interfaces, as the host kernel will be used for that. There is a possible way around this limitation, discussed in the next section.

7 Future Work

SIS right now is at the very beginning of its life. There are a number of short comings that it has, and many directions it can go from here. What follows is a few of those thoughts, some of which are very pie in the sky, and some that will probably make it into the code stream by the end of this year.

7.1 Current Weaknesses

SIS has a number of weaknesses currently. The first major one is the fact that images currently contain more than just the software that is applied to the image. Because images also contain files like /etc/raidtab, and /etc/fstab, an image is bound to a partition model. This means that an image build for /dev/hda, cannot be applied to a machine with only SCSI devices. As most software doesn't care about the underlying disk devices, this should be able to be extracted from the main image and put into the autoinstall script.

The lack of multiple adapter support is also a big weakness. SIS currently only will set up the **eth0** interface during installation. Although there are hacks to change which interface is setup and to bring up additional adapters via dhcp, real multiple adapter support needs to be added to the autoinstall script to support this.

Remote logging existed in LUI, but there is no equivalent in SIS. This needs to be put in place before SIS can be considered enterprise ready, as lack of remote logging makes the discovery of failures far more difficult.

The autoinstallation kernel needs a Debian environment to build in. The main reason for this is that Debian provides libc_pic packages, which make it easy to create a smaller version of glibc to go on the autoinstall media. This is a serious limitation to having true source packages that can be rebuilt on any environment. There are a number of possible options here, the use of uClibc, dietlibc, or minilibc are top on the list.

Although building images directly on the server works for most packages, it doesn't for all of them. Occasionally a package will attempt to start a daemon which needs to communicate with other services or directly with hardware. Although it is questionable for a package to do this in a post install script without having a good way to shut it down, it does happen. There has been the possibility of doing some manner of freeze / thaw on post install scripts, so that certain package scripts would be stopped from running, then executed on first boot of the client. This is possible, but the full implications would need to be worked out.

7.2 New Directions

There are many new directions we would like SIS to take, however it is unlikely that more then one or two of these will get accomplished this year due to the size of the development team. So consider some of these pie in the sky ideas that hopefully some eager volunteers will help us do.

7.2.1 SIS to other Platforms

Currently in the pipe is work to bring SIS to PowerPC, HPARISC, and S/390 Linux. Some of these ports should see the light of day this summer. There has always been the thought that SIS could be applied to other operating systems, especially the *BSD family (FreeBSD, OpenBSD, and NetBSD) of operating systems. Any OS which Linux supports creation of, and read / write access to, their filesystem should be able to be supported by SIS in some manner.

7.2.2 Multicast SIS

Once upon a time a multicast library was written for SystemImager called multicaster. The library was never fully finished, but was posted to Source Forge anyway. The funny thing with open source projects, is they pop up in the oddest places. Sometime in October of last year, a new project was announced on Freshmeat called **mrsync** which was a derivative of multicaster with an rsync like command line interface. This is being used in production shops at the moment. The possibility exists of making SIS use mrsync instead of rsync during initial installation to allow installation to scale to hundreds or thousands of simultaneous nodes.

7.2.3 Diskless SIS

SIS creates fully chrootable images on a server which can then be deployed to clients. With very few modifications it should be able to build fully chrootable environments which could be used for diskless environments as well. There is significant interest from certain segments of the high performance computing community for this, so I believe this will happen in the near future.

7.2.4 UML Verification Suite

Because the images on the server are full installations of a running system, it seems possible that a more hearty verification on system integrity could be run on them. The natural choice for this would be User Mode Linux. After image instantiation, a custom verification program could be run in a UML instance which uses the image as its root filesystem. This would allow for a burn in test of the image before it was ever deployed, and could help track down possible conflicting libraries or software revisions. This type of burn it would be essential for large installations that want an assurance test on their images before deployment.

8 SIS in Action

One of the areas that Linux has penetrated extremely well, is the High Performance Computing arena, specifically High Performance Linux Clusters. This arena is very will suited to the strengths of SIS, as all the machines in a cluster tend to be nearly identical. Only a small number of images will be needed to deploy hundreds or thousands of machines. The installation method is distribution independent, so no matter what distribution the user chooses to deploy, the methodology is the same. SIS also has both a command line and graphic user interface, so it can be driven by another application very easily.

OSCAR (Open Source Cluster Application Resource) is a cluster building based on the best known practices in Linux clustering. As of OSCAR 1.2, System Installation Suite is the installer for all the client nodes in an OSCAR cluster. The OSCAR wizard is written in Perl Tk, and hence can use TkSIS panels directly. Every panel in TkSIS allows a callback to be registered. OSCAR uses this feature to do other cluster setup tasks at every stage. This integration was very easy to accomplished, and has given the OSCAR project a very robust distribution independent mechanism for installation. SIS was instrumental in allowing OSCAR support both Red Hat and Mandrake in OSCAR version 1.3.

Other clustering projects such as Clubmask, SCore, and SCE are looking at moving to SIS for their installation so they can support numerous underlying distributions. We expect many tools to leverage the image based framework for systems management that System Installation Suite has created in the future.

9 Conclusion

System Installation Suite is a novel approach to the massive installation problem in Linux which is both distribution and architecture agnostic. It provides an image based framework for extremely scalable installation and maintenance. It has always been the intent of the project to expose as many clean interfaces as possible to other applications, so System Installation Suite can be cleanly integrated into other projects or products requiring an install method that works on many distributions. The expectation is that other components could be easily added to System Installation Suite over time to exploit many of the capabilities of image based systems that have yet to be explored.

Our motto has always been: "Do it once, do it right, do it for every buddy". We want to support every distribution and every architecture that will run Linux equally well. By doing so, we raise the base line for Linux system's management, and make Linux easier to deploy for administrators everywhere.

For more information on System Installation Suite: how to use it, how to join the project, and what you can do to help, please visit our web site at **http://sisuite.org**. Links to all the component parts of System Installation Suite are provided there, as well as a network installer which will download and install the latest version of System Installation Suite.

10 Acknowledgements

Credit must always be given where credit is due. System Installation Suite has had many contributors, and hopefully many more (for a full list please see the credits files of all the various projects). Specifically mentioned should be the follow folks that have put long hours into making System Installation Suite happen over the past year, Michael Chase-Salerno, Brian Finley, and Dann Frazier. The following companies have donated a significant amount of time and effort to the project as well, and should be commended for that: Bald Guy Software, Hewlett Packard, and IBM.

Special thanks go out to my editing crew: Trey Belew, Michael Chase-Salerno, Joe Greenseid, and Barb Kane. Without their assistance, this paper would have been far less legible. I also want to thank my employer, IBM, for allowing me to spend the last year working full time on

the System Installation Suite project. I believe this project is a significant contribution to the Linux community, and am thrilled to have been a part of it.

11 References

LUI,
`http://oss.software.ibm.com/lui`

System Configurator,
`http://systemconfig.sf.net`

SystemImager,
`http://systemimager.org`

SystemInstaller,
`http://systeminstaller.sf.net`

System Installation Suite,
`http://sisuite.org`

OSCAR, `http://oscar.sf.net`

12 Trademarks

Proceedings of the
Ottawa Linux Symposium

June 26th–29th, 2002
Ottawa, Ontario
Canada