Synctree for Single Point Installation, Upgrades, and OS Patches

John Lockard, University of Michigan
Jason Larke, ANS Communications, Inc.

Synctree for Single Point Installation, Upgrades, and OS Patches

John Lockard – University of Michigan
Jason Larke – ANS Communications, Inc.

ABSTRACT

The combination of large networks, frequent operating system security patches, and software updates can create a daunting task for a systems administration team. This paper presents a system created to address these challenges with system security and “uptime” as the primary concerns. By using a file-form “database,” the Synctree system holds a full network’s configuration in an understandable, secure, location. This paper also compares this system with previously published works.

Introduction

Synctree is flexible enough to be useful for a small organization with only a few machines, to a large university with thousands. The idea behind Synctree is that once you’ve got the machine talking to the network, you can “sync” it to a Synctree template to bring that system to the same level as the rest of the machines on your network. You can usually rely on your vendor to help you get to the point where your system is up and running and communicating on your network. Sun supplies Jumpstart, HP ignite, etc. But as pointed out in [Anderson], these procedures will always be inadequate for one or more reasons.

You could “clone” a system, by making an image of a system that meets your guidelines, but that image would really only be good for the initial setup of that computer, and would need to be re-generated each time a new patch was installed. For some operating systems, this may be as often as every day. In many organizations, one department will need a slightly different configuration from the next. You would also need a master machine for every system architecture you are using.

You could use rdist to send out an update, but you would need to make sure that every machine is listening when you run the update. Also, this update may only be good for one shot. Many patches require portions of previous patches to be installed first. If one of your machines wasn’t “listening” when you updated with the previous patch, you can no longer count on your current patch to leave you with an operational system.

You installed a security patch on all of your systems last week, but you’ve just been “cracked” through the vulnerability that the patch was supposed to fix. How do you go about verifying that the patch on the system is valid? Or, are you sure that ‘ifconfig’ is really the ‘ifconfig’ that you put there? Has ‘ls’ been modified to pass the file size, date stamp and crc tests? The rdist datestamp/crc check is not infallible as datestamps can be forged, and a crc on a bogus file can adjusted by most script-kiddy “hackers” with the latest version of their favorite root-kit.

These are all challenges that we’ve faced in our duties as Systems’ Administrators. They don’t have to be as troublesome as they appear. With Synctree in place, I am able to install a patch in a central location, and be assured that the next morning all the target machines on my network will have that patch. I can also be sure that the files I expect are on my system.

The University of Michigan’s Computer Aided Engineering Network group through the direction of Paul Howell created a utility called Synctree that takes care of all these things.

Previous Work

gutinteg, machdb and packagelink [Fisk] is a very similar set of utilities viewed from a completely different angle. Unfortunately, its aim is the small to medium sized operation. The focus of this idea excels at installation of packages and basic machine configuration, but is not geared toward maintaining the integrity of the system. Since the system uses a “copy” method of install rather than a “compare-before-copy” method, this can get quite expensive as more machines are thrown onto the system. Because of this, a large installation would not be able to verify the integrity of its systems on a daily basis.

GeNUAdmin [Harlander] is a very intricate system that can be used to manage even the most minute of a system’s configuration. This minute focus makes this system quite daunting. The use of so many configuration “databases” leaves yet another learning curve to the use of a system that’s intent is to make system administration easier. GeNUAdmin does not aim at updating of any of the system’s files outside of general configuration.

Sasify [Shaddock] uses a method requiring a reboot to initiate the update to the system. For an operation that is 24/7/365, this is not a feasible method, as many of the updates that need to be done are as simple as changing one file, with no reboot required.

OMNICONF [Hideyo] allows for the storing and restoring of configuration and operating system files. OMNICONF admittedly is useful for Updates and configuration recovery. But, the storage of whole configurations for individual machines and the lack of any real "template" sharing limits the size of the operation this can support.

lcfg [Anderson] is a very robust system for updating subsystems on a class of machine. This system focuses more on the idea of rebuilding a system from scratch when the configuration has started to "rot." Such services as Jumpstart, Kickstart and Ignite are developed specifically for the purpose of getting a machine installed, and are very adaptable to the hardware configuration of the machine. As with many other update utilities, this is also a package that runs best as a boot-time update.

Many sites use *rdist*(1) and related scripts to automate file distribution. Synctree and rdist share many features. However, the rdist model of a single fileset, which is distributed to many machines from a single server, becomes problematic in large, complex environments where universal trust for a single server may be difficult to obtain.

Remy Evard [Evard] discusses several examples of configuration management in action. He points out several problems in ad-hoc systems that we feel Synctree addresses rather well. In particular, Synctree is suitable for managing servers as well as clients (all LS&A AFS, mail, and DNS servers are Synctree clients), and uses the "configure to" model for making changes to a workstation.

Many, many other systems for performing similar tasks exist. The need for this "wheel" is so obvious, and the degree of market penetration reached by any one solution so small, that it has been reinvented in any number of places. We've been using Synctree for more than five years, and so have never looked at many of these tools. We hope that Synctree might prove useful for some of you, and also that by presenting a somewhat different spin on a common problem, we can spur other re-inventors to produce even better work that might someday become an acknowledged standard.

What Is Synctree

Synctree is both a command and a software suite for large-scale systems administration.

As a command, it's an intelligent file copier.

As a suite, it's a whole set of programs for describing, generating, and managing system images.

Synctree, like many other packages (including *rdist*(1), uses a compare-before-copy strategy. If a file currently on the disk is just like the file in the image, there's no need to copy. Synctree's notion of "just like" is fairly rigorous, including date, size, and md5 digest. It also checks file ownership, group, and mode, and will correct them if they are wrong but the file's contents are correct.

Since the original UNIX cp command doesn't work the same on all OS's, and may not work in all situations, Synctree will actually print out a list of commands needed to synchronize a file with a system image. The commands are actually shell scripts that come with Synctree, and can easily be adapted to the need or peculiarities of any given OS. They're named after their shell equivalents, with a 'sync' prepended – i.e., Synctree copies with 'synccp'. The first argument to any command is the reason why the command is being run. So, to copy down a new version of sendmail, because of a changed modification time, Synctree might say something like Listing 1. synccp in turn is shown in Listing 2. Along with some other refinements, the Synctree engine and these scripts were the basis of the original package.

Eventually this system breaks down. Examining a file may be simpler than copying it, but several hundred clients all checking the modification time at once on several hundred files generates severe scaling problems.

SI Files

Synctree version 2 introduced the SI database, a way of storing all the relevant information about a package. Listing 3 shows an example stanza from an SI file.

```
synccp MTIMSYNC /srv/template-server/LSA/root/usr/lib/sendmail /usr/lib/sendmail
```

Listing 1: Copying a new version of sendmail.

```
#!/bin/sh
#synccp - bourne shell script to copy a file
# Synctree Vers. 2
#grue - May 1994: initial version
#dirt - Dec 1996: replaced with cp/mv combo
cp -p $2 $3.TOBEMOVED
mv $3.TOBEMOVED $3
```

Listing 2: Using synccp.

In order, this tells us:

- What file we're comparing (*is*).
- What file to compare it to (*like*).
- What the permissions (*st_mode*) of the file should be.
- Who the owner (*st_uid*) should be.
- What group (*st_gid*) the file should belong to.
- What the file's proper size (*st_size*) is.
- What the file's *md5* digest should be.
- When and how to correct any differences (*trigger*).

The trigger value controls the sync attributes used when comparing IS and LIKE, plus the trigger specifies how a filesystem object should be instantiated. The trigger consists of three hexadecimal digits. Reading from left-to-right, the trigger semantics are as follows:

First digit – file type

1 Place holder, do not sync or remove.
2 Instantiate IS as a symbolic link using LIKE as the link text.
4 Instantiate IS as a regular file.
8 Instantiate IS as a directory.
9 Instantiate IS as a directory. Avoid recursing into the directory when removing extra files (e.g., /afs) but maintain the mode of the directory.

Second digit – mode,uid,gid,size

0 Ignore these fields for syncing purposes.
1 Sync the mode based on st_mode.
2 Sync the owner based on the value specified by st_uid.
4 Sync the group based on the value specified by st_gid.
8 Sync the file if the size is different from st_size.

Third digit – mtime,md5,remove-only

0 Ignore these fields for syncing purposes.
1 Sync the file if the time last modified is different from st_mtime.
2 Sync the file if the MD5 one-way hash is different from md5.
4 Update the time last modified and access time but do not sync the file.
8 Remove IS only. No synchronization is performed. The file type specified in the first digit is used as a bit mask to specify the type of file that IS can be if it is to be removed. For example, a value of 'c08' would remove the IS if it were a regular file or a directory, but IS would not be removed if it were a symbolic link. A value of '208' would remove IS only if it were a symbolic link.

Trigger values are additive, allowing flexibility in how a file is instantiated and what attributes will trigger its replacement.

The program that creates these SI files is called statinfo. Statinfo takes a given file or directory to start with (-L, or 'like'), and a file or directory to sync (-I, or 'is'), and prints out SI files for making that comparison. For example, if I run a query like one shown in Listing 4, I'm asking "is /etc/hosts.equiv like /srv/template-server/PSC/root/etc/hosts.equiv," and I get the results shown in Listing 5.

The file specified by the **-I** switch doesn't have to actually exist, but the file specified by -L does, since it's the basis for the comparison.

Once the SI files are created, all the Synctree clients need to do is read them. It's much easier on the server to deliver five megs worth of SI files three hundred times than support comparison of five thousand files three hundred times.

To have an SI file, you have to have something to make it from. The standard organization method is to have a directory where all the templates live, and under each template you have a directory called "root." This directory is analogous to a machine's / directory. Any files you want to put in the template go under "root," just like they normally live under /.

Let's take a concrete example. The AFSMODS class includes four files: /usr/bin/login, /usr/X11R6/bin/xdm, /usr/sbin/in.ftpd, and /usr/sbin/in.rexecd. As a template, it looks like Listing 6.

```
% tail /srv/template-server/AFSMODS/SI/root.SI
{
is /usr/sbin/in.rexecd
like /srv/template-server/AFSMODS/root/usr/sbin/in.rexecd
st_mode 100555
st_uid 0
st_gid 0
st_size 366012
md5 b6a27adc0df35f401b54a6f9af3e342a
trigger 4f2
}
```

Listing 3: Example stanza from an SI file.

```
statinfo -I /etc/hosts.equiv -L /srv/template-server/PSC/root/etc/hosts.equiv
```

Listing 4: Similarity query.

```
/*
 * statinfo produced this output
 * executed by root on Tue Apr 14 14:18:10 1998
 * selected run-time options are:
 *      directory trigger = 870
 *      regular file trigger = 4f3
 *      symbolic link trigger = 200
 *      shadow tree trigger = 2f3
 */
{
 is /etc/hosts.equiv
 like /srv/template-server/PSC/root/etc/hosts.equiv
 st_mode 100664
 st_uid 0
 st_gid 0
 st_size 479
 st_mtime 875041276
 md5 8a0a0f460702260ba60b6098b5776887
 trigger 4f3
}
```

Listing 5: Result of similarity query.

```
% ls -laR root
root:
total 36
drwxrwxrwx   4 root     wheel      14336 Jun 24 06:25 ../
drwxr-xr-x   5 root     bin         2048 Sep  5  1996 usr/
root/usr:
total 20
drwxr-xr-x   5 root     bin         2048 Sep  5  1996 ./
drwxr-xr-x   3 jlarke   wheel       2048 Jul 12  1996 ../
drwxr-xr-x   3 root     wheel       2048 Jul 12  1996 X11R6/
drwxr-xr-x   2 root     bin         2048 Jul 15  1997 bin/
drwxr-xr-x   2 root     wheel       2048 Mar 18  1997 sbin/
root/usr/X11R6:
total 12
drwxr-xr-x   3 root     wheel       2048 Jul 12  1996 ./
drwxr-xr-x   5 root     bin         2048 Sep  5  1996 ../
drwxr-xr-x   2 root     wheel       2048 Jun  3  1997 bin/
root/usr/X11R6/bin:
total 716
drwxr-xr-x   2 root     wheel       2048 Jun  3  1997 ./
drwxr-xr-x   3 root     wheel       2048 Jul 12  1996 ../
-rwxr-xr-x   1 root     wheel     361556 Jul 25  1996 xdm*
root/usr/bin:
total 2228
drwxr-xr-x   2 root     bin         2048 Jul 15  1997 ./
drwxr-xr-x   5 root     bin         2048 Sep  5  1996 ../
-r-xr-xr-x   1 root     bin      1136292 Jun  7  1997 login*
root/usr/sbin:
total 1560
drwxr-xr-x   2 root     wheel       2048 Mar 18  1997 ./
drwxr-xr-x   5 root     bin         2048 Sep  5  1996 ../
-r-xr-xr-x   1 bin      bin       427648 Sep  5  1996 in.ftpd*
-r-xr-xr-x   1 root     wheel     366012 Mar 18  1997 in.rexecd*
```

Listing 6: Template of AFSMODS class.

Generally speaking (although this may vary in other implementations), directories on the local disk but not in a template are ignored by Synctree.

Presas and Posas

Sometimes copying a file isn't enough. For example, if we're distributing a new daemon, we want to stop an old version and start the new. Presas (pre-sync activities) and posas (post-sync activities) take care of this.

Presas and posas are keyed to a particular file and run before or after any changes to that file are made. At LS&A, we don't have much call for presas, because Synctree is able to update a running program without crashing it. We just use a posa to stop the old version and start the new after the update.

For example, Listing 7 shows an SI file entry for sendmail.

The posa line tells it to run that script after making changes to sendmail. When Synctree runs a posa or presa, it gives it two command-line arguments: the reason why the file had to be changed, and the name of the file. So the command line might look like Listing 8. meaning that sendmail was updated because its md5 hash changed. The script is very simple; see Listing 9.

CTRL Files

Sometimes Statinfo doesn't (or can't) get all the information about a program. For example, there's no way it could tell what posa was associated with a file just by examining the file. CTRL files allow administrators to manually override all or part of an SI file. They look just like SI files, but they're usually typed by hand, and they usually have parentheses instead of curly braces. When Synctree sees a curly brace, that stanza replaces any previous stanza for that IS file. Parentheses tell it to keep the old data, and only replace the specified fields.

In the previous section, we saw an SI file for sendmail that was generated by merging an SI file and

```
{
is /usr/lib/sendmail
like /srv/template-server/LSA/root/usr/lib/sendmail
st_mode 104751
st_uid 0
st_gid 3
st_size 401640
md5 473c1c1400281a032473c1f121d0049f
trigger 4f2
posa /usr/private/admin/synctree/posas/sendmail
}
```

Listing 7: An SI file entry for sendmail.

```
/usr/private/admin/synctree/posas/sendmail MD5SYNC /usr/lib/sendmail
```

Listing 8: posa command line.

```
#!/bin/sh
# Restart sendmail if the binary was synced.
. /usr/private/admin/synctree/posas/posa.env
echo "/etc/init.d/sendmail stop"
/etc/init.d/sendmail stop
echo "/etc/init.d/sendmail start"
/etc/init.d/sendmail start
```

Listing 9: posa script.

```
{
is /usr/lib/sendmail
like /srv/template-server/LSA/root/usr/lib/sendmail
st_mode 104751
st_uid 0
st_gid 3
st_size 401640
md5 473c1c1400281a032473c1f121d0049f
trigger 4f2
}
```

Listing 10: SI file.

a CTRL file. The SI file entry would look like Listing 10, and the CTRL file entry is shown in Listing 11.

Organization (Classes)

Once you have the ability to create and use a system image, you run into the fact that no single image works for everyone. Each department, or sometimes each machine, has unique needs. So the authors created the notion of classes.

A class is a named set of files. Each department in LS&A that uses Synctree has a class. Berkeley-style printing is installed with a class, as is wuftpd. We also use them to track changes to the system-one class, SOL251, is Sun's OS image with vendor patches and AFS installed. The LSA class represents all the changes we've made to suit the College's needs. If I notice that 'rdist' doesn't match the system I used to work on, I can easily find out that the LSA class replaces it with a different version.

Each class has its own SI file. Synctree reads the SI file and builds a database of files to compare and reconcile, keyed by the IS entry in the SI file.

Each machine belongs to a specific set of classes, determined by the CLASS= line in /etc/hostconfig. Every machine has its own class, which is appended to the list of classes specified in the machine's /etc/hostconfig file. The order of the classes determines which files take precedence over others. Classes later in the list win out over those listed earlier.

For example, let's say my /etc/hostconfig file includes Listing 12. Both the SOL251 class and the AFSMODS class include versions of the /bin/login program. Synctree first reads the SI file for the SOL251 class, which looks like Listing 13. and then, later on, reads the SI file for AFSMODS, which would include lines shown in Listing 14.

When Synctree sees the second entry for the same IS file, it automatically replaces the first one with the new version. When it has read all the SI and CTRL files for all the classes listed (plus the host's own class), it has a complete image of the system built up, and it's ready to start comparing and updating.

It may sound complicated, but the upshot of it is that this one program takes care of all the synchroniza-

```
(
is /usr/lib/sendmail
st_mode 104751
posa /usr/private/admin/synctree/posas/sendmail
)
```

Listing 11: CTRL file entry.

```
CLASS=SOL251:LSA:PRINT:NPI-CDDI:ODS4:AFSMODS:WUFTP242UM:PSC:PSCSOL
```

Listing 12: /etc/hostconfig line.

```
{
is /usr/bin/login
like /srv/template-server/.SOL251-sun4m_55/root/usr/bin/login
st_mode 104555
st_uid 0
st_gid 2
st_size 28800
md5 635130471cda6d64d5f2a6667dc8ecfd
trigger 4f2
}
```

Listing 13: SI file for the SOL251 class.

```
{
is /usr/bin/login
like /srv/template-server/AFSMODS/root/usr/bin/login
st_mode 104555
st_uid 0
st_gid 2
st_size 1136292
md5 3ac4528006a8ee5d0685e1e25d789673
trigger 4f2
}
```

Listing 14: SI file for AFSMODS.

tion chores – all the administrator of a machine has to do is run the command.

Installation Procedures

LS&A tries to use vendor installation procedures to install Synctree, and then let Synctree install the rest of our changes. We've only implemented this philosophy for Solaris thus far, but see no reason that it couldn't be done for other platforms.

Syncing a Machine

Just running the Synctree binary doesn't get us very far. It provides the comparison engine, but we use wrapper scripts to tell it which SI files to use, authenticate to Kerberos, and handle logging what it does.

Generally machines run Synctree from cron, starting around 3 am every morning. The crontab entry for this is shown in Listing 15. run_synctasks is a script which has a handy randomizer routine which will have systems sleep anywhere from 0 seconds to 6.5 minutes before starting their sync. This has been random enough for the college to avoid too large of a load all at once on the servers and the network.

If you want to run Synctree by hand, just run 'syncnode'. You can also use the '-norun' switch if you want to see what Synctree *would* do, without actually making any changes.

Making Changes to a Template

Before we change Synctree templates, we generally test the change on a particular machine first. This prevents the nightmare scenario where a change which breaks machines happens on hundreds of machines at 3 am.

Once we know that a modification works, we need to determine which class to put it in, and update that class.

Generally, we put changes in the most general template that makes sense. If we have a file that goes on every machine in a department, we don't put it in each machine's class; we put it in a class that every machine syncs to.

Updates of a class's version of a file are done with the 'cisync' command. The basic use is just

```
cisync <filename>
```

Cisync will ask which class to add the file to, or we can specify it on the command line with the '-c' flag.

```
cisync -c MATH <filename>
```

Once we've cisync'd all the files, we need to make sure the SI files get updated to match. At LS&A, the college runs a cron job that will take care of this at 2 am each night. If you want to sync machines before then, you need to run the mktemplate command.

```
mktemplateSI <CLASSNAME>
```

Creating a New Class

The command to create a new class is 'mkclass'. At LS&A, running AFS, you need to be in the AFS system:administrators PTS group for the lsa.umich.edu cell. Class names can be no longer than eight characters long and, by convention, are in all caps. The length limit is a product of AFS' limits on volume names, since each class gets its own volume.

```
mkclass CYRUS152
```

Commonly Used Classes

For Solaris, classes of general interest at the College of LS&A include:

- [TEST,TEST4c,TEST4m,TEST4u]: Test classes for new patches, updated software, etc. If it succeeds here, it gets rolled out to production.
- [SOL251]: Stock Solaris with AFS installed, no customizations. Some files deleted in order to link them into AFS.
- [LSA]: LS&A changes to the stock Solaris install.
- [AFSMODS]: AFS-aware login, xdm, xlock, etc.
- [AFSSERVER]: Useful AFS server mods, afs fstype, tcpwrappers, etc.
- [FTP242B13]: Wu-ftpd version 2.4.2b13 with AFS mods, and .principals support.
- [MAILSERV]: Programs needed to support SMTP, IMAP2bis, POP3, and KPOP services.
- [MAILSERV2]: Like MAILSERV, but keeps inboxes in hashed directories – i.e., /var/mail/r/o/root/INBOX.
- [PERL]: Install Perl 5.004 in /usr/private/.
- [PRINT]: BSD-style printing commands.
- [SAMBA]: Samba 1.9.17p2 with AFS and NT hashed password support.

Other Synctree Utilities

Precedence

Precedence helps system administrators determine what Synctree class(es) a given file comes from. The simplest form:

```
# precedence /etc/inet/inetd.conf
```

would display a list of each /etc/inet/inetd.conf file in Synctree for that machine, from most important to least important like that shown in Listing 16 that shows that inetd.conf exists in MATH, LS&A and

```
0 3 * * * /usr/private/admin/synctree/bin/run_synctasks 2>&1
```

Listing 15: crontab entry for Synctree.

SOL251 and that the MATH copy is the one that takes precedence from Synctree.

mksynclist

mksynclist list Synctree clients and/or classes. Its original purpose was to provide a list of all the possible targets for mktemplateSI, but it can also be used to list all the machines that sync to a certain class.

whosyncs

whosyncs displays a list of all machines and classes that contain 'filename'

cisync

cisync copies a file from the local machine to Synctree classes. Since it uses Synctree to make the copy, administrators can be sure that the mode, type, and ownership of the file will be the same in the template, as on the local disk.

cosync

cosync gets a particular file out of Synctree. Used when you only want one or two files, and a full Synctree run would be too expensive.

addclass

addclass installs a new Synctree class on a machine.

Normally, the way to do this would be to add the class to the /etc/hostconfig file and run a syncnode. However, if you only want to add the functionality provided by the new class, there's no need to check all the files on the entire machine against the template.

rmclass

rmclass removes Synctree classes when you no longer want them on your machine.

Taking a class out of a machine's hostconfig file makes sure that the class will not sync again, but rmclass goes the extra step of removing files installed by the class.

In the interests of being careful, the output of rmclass is a list of shell commands, which you can either pipe into sh for execution or save to a file for examination.

The Big Picture: A Usage Scenario

At the University of Michigan's College of Literature, Sciences, and the Arts, Synctree and the core classes are maintained by a small group of sysadmins employed by the College. Most machine installs are done by sysadmins employed by various departments within the College. With Synctree in place, the departmental sysadmin's activities tend to revolve around it.

For example, when a new desktop machine arrives from Sun, the sysadmin uses Jumpstart to load the OS and Synctree. When the machine reboots, it has guessed (from its fully qualified domain name) which classes to install. The sysadmin double-checks this list, inputs her AFS password and the machine syncs. When it reboots, it will be a fully configured and operational workstation, already tailored to her department's needs. This operation – which can be quite tedious without automation – has taken only five to fifteen minutes of her time.

If the machine was a server, she might have added classes to install imapd and qpopper. She might have to generate a sendmail.cf file, configure DNS, or perform several other tweaks. The advantage of Synctree is that she will *not* have to worry about getting out of date versions of her mail software, an insecure version of BIND, or any of the other common bugaboos for sysadmins. She will not have to build any software from source, or get permission from a higher authority. In fact, she could deploy an entire department's worth of machines, and the expert sysadmins working for the College would never have to worry.

Comparison to Other System Configuration Systems

A detailed comparison to all systems would be impossible. However, with so much prior art in the field, a little elaboration on Synctree's unique advantages seem appropriate.

Many of the systems in common use (notably *rdist*(1)) require a server to initiate a sync. In environments where not every sysadmin trusts every other, such systems require additional automation scripts to be written in order to manage privileges. Synctree uses file system permissions to control access to the templates, and only the root user on the client machine can order a sync. Only root on the client can control which classes the client syncs to. The only parties the client needs to trust are those with administrator access to the file system the templates live on – in AFS, those in the system:administrator group.

Systems that rely on a central server also face performance bottlenecks. Even high-end machines can only carry on an rdist to a certain number of machines. Using rdists' binary comparison mode for security drops this down even further.

The only central service required by Synctree is a file server. Synctree downloads roughly five megabytes of SI files from the server, and all further work (except for copying files from AFS) happens entirely on the client. This means that much smaller demands are

```
% precedence /etc/inet/inetd.conf
/srv/template-server/MATH/root/etc/inet/inetd.conf
/srv/template-server/LSA/root/etc/inet/inetd.conf
/srv/template-server/SOL251/root/etc/inet/inetd.conf
```

Listing 16: Listing of inetd.conf files.

placed on the file server than would be placed on the rdist server. Furthermore, the use of precomputed SI files for the template, instead of forcing the server to examine each file in the template as it checks them, provides an enormous savings.

Many systems depend on complete images being distributed. Synctree, like GNU cfgengine, allows images to be overlaid by one another, to any desired resolution. As a result, a client can build up a picture in memory of the finished system before any changes are made. Using overlays without the ability to resolve conflicts in memory, ''thrashes'' the machine and versions of the same file are replaced by one another. For example, we once used an AFSMODS Synctree class to install an AFS Kerberos-aware login program. The default Solaris login was superceded by this version while the SI files were being merged in memory. Rdist would require two different dists, with a window of time between them when the wrong version was installed on the client.

Synctree is modular and open. The only configuration file is /etc/hostconfig, where a machine's classes are listed. Otherwise, almost all changes can be made by simply copying files. Every step of Synctree's abstraction, from the initial ''I type syncnode and it works'' level down to watching file-by-file comparison, can be observed and understood by anyone who can read a man page and shell scripts. Other systems that hide more of their functionality, or depend on more complex grammar, can be more difficult to learn and troubleshoot.

Finally, we argue that basing the system on a file server has fundamental architectural superiorities. If the goal of a configuration system is to distribute the correct version of several hundred files to hundreds or thousands of clients, it makes sense to rely on software that has been designed for providing files on that scale. Other common concerns for configuration systems, such as security and scalability, have already been addressed and solved for file systems. Finally, many large sites already have file servers scaled to match their clients. Using this existing resource can save quite a bit of worry later on.

Reflection, Surprise, Terror, for the Future

We at the College of LS&A have been working on an update to this system that will allow Synctree to take software packages that are normally on the network, and install them on the system's local hard drive. The motivation for this, is that you usually get a new system today, where the hard drive will hold the OS around four times over. In many cases, you end up with at least a Gigabyte of space left over which may be wasted.

This idea will pull traffic off of the network and put it onto the hard drive of the local machine. By doing this, we will end up with a faster system all around. The machines with the large hard drives will load applications locally, not off the network, and the other machines, with smaller hard-drives, will no longer have to contend with the others over that network bandwidth to get at the application.

One obviously desirable feature for Synctree would be a way to sync partial files. For example, a class for installing imapd would work best if it could also add the appropriate lines to /etc/services and /etc/inetd.conf. We've discussed doing this with either M4 macros or an implementation of the GNU cfgengine, but have yet to do any actual planning or work toward this goal.

Existing Synctree systems assume that AFS is available for file distribution. While the authors rather like AFS, we understand that not everyone has it or wants it. Synctree could easily be adapted to synccp, krcp, or any other secure client-initiated copying system. The major bottleneck is that such a system requires acls, so that clients may copy down files that are only readable by root, but cannot make changes *as* root, or (probably) grab the server's /etc/shadow file. If anyone is interested in working on this, Jason Larke may be able to provide some support.

Acknowledgments

Thanks to Rik Farrow for his support, and for prodding John into thinking that this presentation would be of interest to anyone. Also, thanks to all the UofM Unix Admins that gave the encouragement to sit down and type up this paper.

Availability

Synctree is available for non-commercial use, and is the property of the Regents of the University of Michigan, Ann Arbor, MI. Synctree (v.2) can be sampled from: ftp.math.lsa.umich.edu/pub/Synctree/.

Author Information

John Lockard is the Systems Administrator for the Department of Mathematics in the College of LS&A at the University of Michigan. He is thinking of switching to the Emacs paintball team. After some thinking, is really glad that Jason talked him out of titling the paper MS-SMS (Multi System-Synctree Management System). John can be reached by mail at:

Department of Mathematics
2072 East Hall – B746
525 East University Avenue
Ann Arbor, Michigan 48109-1109

He can be reached by email at jlockard@umich.edu .

Jason Larke is a Systems Administrator at ANS Communications in Ann Arbor, Michigan. He was the lead Unix Administrator for the College of LS&A previously being a member of The University of Michigan's fabled F-Five. He will never forgive himself for missing the Emacs vs Vi paintball game. Jason can be reached by mail at:

ANS CO+RE
880 Technology Drive
Ann Arbor, Michigan 48108
He can be reached by email at jlarke@ans.net .

References

[Anderson] Anderson, Paul, ‘‘Toward a High-Level Machine Configuration System,’’ *LISA VIII Proceedings*, 1994.

[Evard] Evard, Remy, ‘‘An Analysis of UNIX Machine Configuration,’’ *LISA XI Proceedings*, 1997.

[Fisk] Fisk, Michael, ‘‘Automating the Administration of Heterogeneous LANs,’’ *Tenth USENIX System Administration Conference*, 1996.

[Harlander] Harlander, Dr. Magnus, ‘‘Central System Administration in a Heterogeneous Unix Environment: GeNUAdmin,’’ *LISA VIII Proceedings*, 1994.

[Hideyo] Hideyo, Imazu, ‘‘OMNICONF – Making OS Upgrades and Disk Crash Recovery Easier,’’ *LISA VIII Proceedings*, 1994.

[Shaddock] Shaddock, Michael E. and Mitchell, Michael C. and Harrison, Helen E., ‘‘How to Upgrade 1500 Workstations on Saturday, and Still Have Time to Mow the Yard on Sunday,’’ *LISA IX Proceedings*, 1995.