Store — a System for Handling Third-Party Applications in a Heterogeneous Computer Environment

Anders Christensen* and Tor Egge**

University of Trondheim

Abstract. This paper presents the Store system administration tool suite, which helps administering third party applications in a heterogeneous Unix environment, giving better overview, more consistency, and less manual work. The problems targeted by Store are listed; the basic functionality of Store is described, and experiences from constructing and maintaining the system are presented.

1 The Problem

From experiences, we have found that a number of problems arise in the area of administering computer systems. These problems are often caused by the local, strategic organization of the computers, including heterogeneous hardware, multiple software versions, deviating user needs, multiple system administrators, geographical distribution, duplication of data, and organizational structure.

These give rise to a number of specific problems, where the more common ones are listed below:

Traceability. As the complexity of a computer system rises, the knowledge of "who-did-what" becomes more important, as does the "why-did-they-do-it-like-that" information. However, activities like logging, documenting, and registering are often ignored during particularly hectic periods in the daily system administering. A lot of self-discipline is required to maintain these activities in a highly result-oriented trade.

Affinity. In a traditional `/usr/local` file system, a large number of files are mixed fairly unorderly together. Files are somewhat grouped into applications, but it is often not obvious who maintains which applications. Typically, which files belong to which applications, and which applications consist of which files, are matters of uncertainty and vagueness.

Architecture dependency. Since different hardware architectures require different binary programs, system management tend to administer different hardware platforms separately. This adds unnecessarily to the costs; most applications are identical across platforms except from the differing binaries.

* SINTEF RUNIT (email: *anders.christensen@runit.sintef.no*)

** Division of Computer Science and Telematics (email: *tegge@idt.unit.no*)

Bit rot. There is no good explanation for it, but it is a well-known observation that unattended applications tend to stop working after some time, generally due to of changes in the surrounding environment. Therefore, users slowly move over to the "safe" computers; the computers that contain software being used. This speeds up the process of bit rot on the less used computers.

Source code. System administration does not involve program development, except for installation, bug-fixing, and local customization. The changes to program source code for these areas are often lost over time, as source code is often considered disk space overhead and deleted. This often results in minor differences between versions and installations on different architectures. For the users, these differences can be very annoying.

2 The Domain of Store

Store is a system administration utility that attempts to handle the problems listed in the previous section.

Store limits its scope to handling third party applications. That is, any group of programs and data files that form a logical unit and can be isolated from the operating system. Two main sources of third party software are downloadable freeware from international networks and packages from commercial vendors.

Furthermore, Store limits its operations on these third party applications to the installation, operation, maintenance, bug finding and -fixing parts of their life cycles.

That is, areas outside the domain of Store are: program development; user and network administration; OS maintenance, configuration, and surveillance.

Currently, Store is also limited to Unix systems. Initial testing is being performed for one-user, personal computers; specifically PCs running MS-DOS.

3 Requirements

The main requirements we have placed on Store are:

No OS modifications. No changes to the operating system should be necessary in order to run Store. Neither new device drivers, kernel modifications nor permanent daemons.

OS-supplied mapping. The operating system should provide the mechanism to map from the view (linktree) to the repository (store). A space-efficient such mapping for most Unix systems is soft-links.

Source code availability. The source code for the system should be available to the users.

Portability. The system should be able to run on any system that follows the POSIX.1 standard and has symbolic links.

Versioned binary files. The system should handle the existence of several versions of binary executables available in the repository, choosing the right one to become visible in the linktree.

4 Other Existing SCM Systems

Most existing SCM systems does not meet our specified requirements. To a large extent they target *programming* environments, instead of *installation* environments. Also, many of them can only manage textual files, are commercial products (available only for a subset of the platforms), or require changes in the operating system.

Two other existing SCM systems that meet our requirements are “Depot”[1] and “Lude”[3].

The problems described above have been targeted by several other systems. The initial implementation of Store (spring/summer 1991) was heavily influenced by “Depot” . However, no implementation of “Depot” was available and the local implementation turned out to be different from the original in many ways, so another name was given.

Other systems that target the problems in a similar ways are Ericsson [2]; and “Lude” .

In addition, Pleasant and Lear [4] describes “Track”, which is a basically a transport mechanism, and how it has been put into use in order to solve the same problems as Store.

Many university CM systems target configuration management of source code. However, for a system administrator, the one-time effort of compiling an application is often manageable and trivial compared to the continued effort that has to be put into continued, on-going administration of a complex system involving a large number of applications, organizations, architectures, and users.

Furthermore, most large Unix operating systems have a system for distributing software packages. But unfortunately, most of these commercial products are available only for a single OS and few offer more than insufficient, elementary support for the system administrators. Generally, they only offer support for loading software into the machine, a crude inventory list, and perhaps support for unloading.

5 The Concepts

The main concepts of Store are:

Separability. Everything related to a single application, including all versions and support for various hardware platforms, are isolated at one location in the computer file system. This “master location” of the application can be institution-wide; it can also span multiple cooperating institutions.

Consistency. An installation of an application should be consistent throughout the institution. If incorrect or buggy, the application should be consistently incorrect everywhere. Only then is there a fair chance of the problem being corrected; else users move over to computers that work, allowing the faults to continue and to accumulate.

Automation. Any kind of manually performed routine work will eventually be skipped or performed incorrectly. Thus, such work should be automated as much as possible.

Documentation. Locally written documentation should be as scarce as possible, and preferably be limited to: a) "gluing" together documentation accompanying the applications; b) documenting local conventions and deviations; and c) filling in fatal gaps in accompanying documentation. Locally written documentation tends to get outdated quickly, so the lesser there is to maintain, and the more automated is maintenance, the better.

Standardization. The more need a system administration tool has for specific and non-standard support, the more difficult and troublesome it is to port it to new architectures and organizations. Thus, when implementing solutions, standardization, portability, and robustness is more important than simplicity and elegance.

Strategy-less mechanisms. In constructing Store, the mechanism has been attempted isolated from the strategies that it implements. Thus, the Store transport mechanisms can be employed for various security schemes and to mimic various intra- and inter-organization relations.

6 Technical Description

Store separates applications from each other, in a two-level system: at the top there are one or more stores, or "store servers". At the lower level there are one or more applications. Multiple stores enable distributing the files in Store among several machines and organizations, e.g. duplication to counter slow or temporarily broken networks.

The directory tree shown in figure 1 indicates the directory structure for a Store system containing two servers and nine applications. The node named "/store/" is the top directory. Thus, path names for the locations of the applications are `/store/store/storlind/zip`, `/store/store/khym/tex`, etc.

The store names ("storlind" and "khym") can be identical to the names of the host on which they are located, but this is not a requirement. However, we have found this naming scheme to be fairly practical. One machine can host multiple stores.

Now we shift the focus to a single application. The structure of a small application is depicted in figure 2. For such a package there are one or more "ver-" directories, zero or more "src-" directories, and a file called "registration". The root of the tree shown in figure 2 is `/store/store/khym/shar`, if the application is "shar", and it is stored on the store "khym".

The "src-" directories contain the source code for the application. The original, official source code is stored in directories named "src-*version*" where *version* refers to the release number. In directories named "src-*version-architecture*", a link-tree (of symlinks) is used to contain modifications specific to that architecture; this linktree is said to be a "shadow" of the original linktree. The linktrees

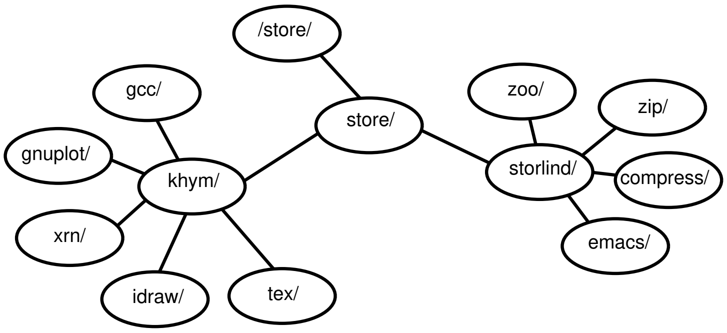

Fig. 1. Structure of Stores

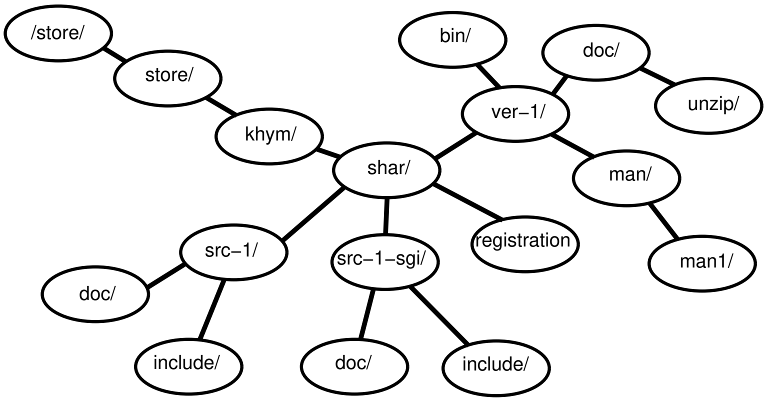

Fig. 2. Structure of Applications

can be configured in a way suitable for the application. For instance, a linktree can be a shadow of another linktree shadow.

In figure 3, a common situation is shown. One directory contains the official source; another contains local "hacks" and configurations, which shadows the official source; and several directories, each containing architecture-specific configuration changes, which shadows the local "shadow". The arrows in figure 3 refers to linktree-inheritance, i.e. shadowing.

Each "ver-" directory contains possibly multiple hardware-variants of one version of the application. Below the "ver-" directory, the structure of subdirectories is similar to the structures usually found in `/usr/local` and `/usr`. Figure 4 shows in detail the structure under a "ver-" tree.

In order to simultaneously handle support for multiple architectures, one need a method for distinguishing between hardware variants of a file in the file system name space. This is done by adding an identifier suffix to the filenames. For instance, the binary file `shar` becomes `shar@sun4os4` and `shar@hp700ux9` for SunOS 4 on Sparc and HP/UX 9.xx on HP-PA, respectively. In Store this is

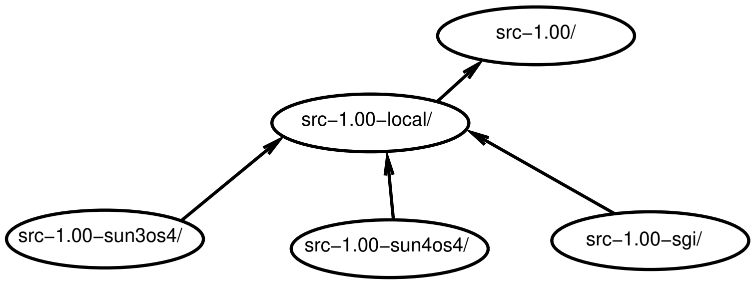

Fig. 3. Interrelationship of Source Directories

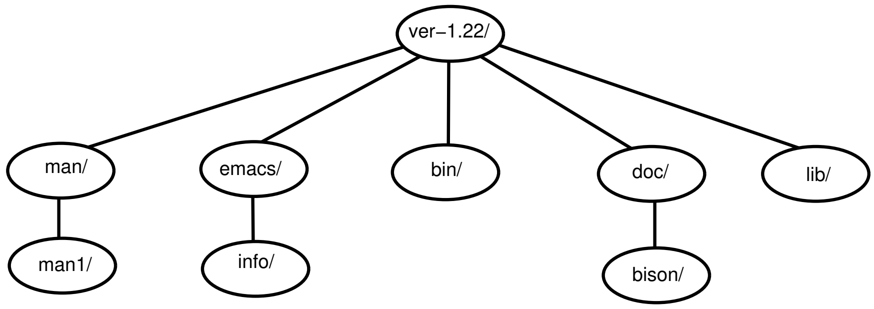

Fig. 4. Structure of a "ver-" Tree

perhaps a bit inelegant, but fairly simple and very portable and standard.

The names of the architectures are defined hierarchically, as shown in figure 5. Although most systems only need the name of the OS, some need to be related to the exact OS release level, or even to the specific configuration of a kernel. Normally, only files are given a file name suffix, but Store also supports suffixes for directories, in which case all files in that directory subtree implicitly inherit the directory.

The implementation of Store does not distinguish between OS-architectures and hardware architectures. Although these are two different dimensions (one OS can run on several hardware platforms, and each hardware platform can run many OSes), we have found it advantageous to collapse these two dimensions into one. The reason is that very few files for one OS are common for multiple hardware platforms; likewise for files specific for one hardware platform but common for multiple OSes. We might have implemented this differently if our target had been operating system software rather than third party applications.

We can now add filenames to the figure showing the subtree of a "ver-" directory (figure 4). Ovals denote directories, filenames are listed below the directory in which they are located. The new, expanded figure is shown in figure 6. Files containing the "@" character and a suffix, are architecture dependent files.

The last component in the application directory is the `registration` file. It

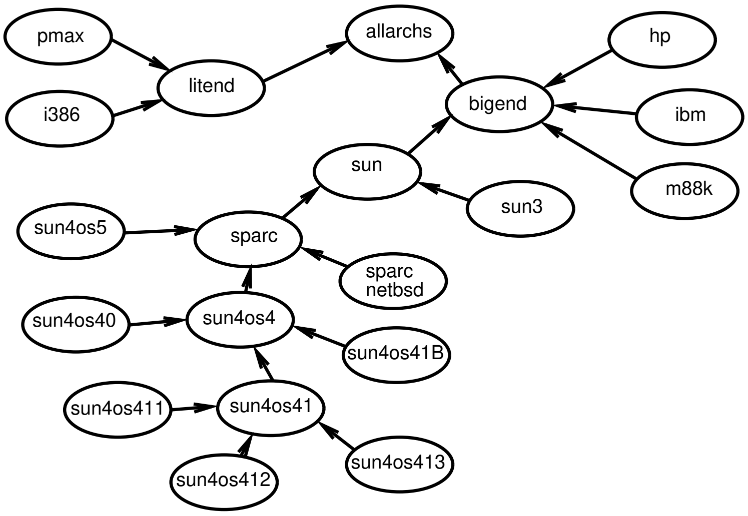

Fig. 5. Structure of Architectures

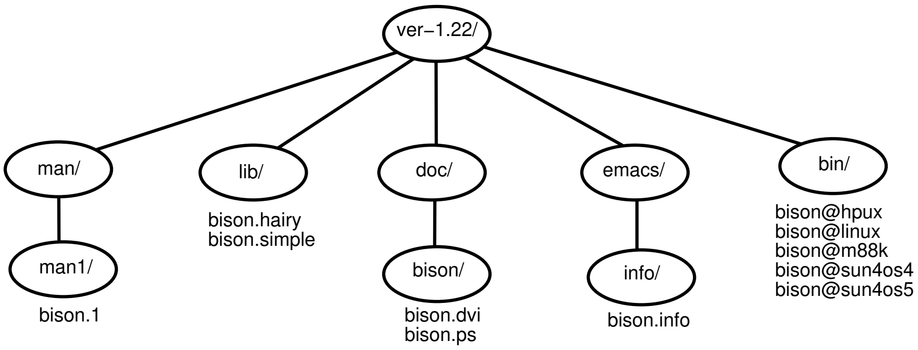

Fig. 6. Extended structure of a "ver-" Tree

contains information about the application, such as the name of the application, the source, the compilation history, who is responsible for local maintenance, which versions that are available and their stability, etc. It can be considered a configuration file for that application.

The system as it has been described so far is not very useful, since the files are distributed widely in a large number of directories. Therefore, the second part of Store automatically creates a linktree under `/store`, generated from all applications on all store servers. Effectively, it creates a directory hierarchy, which name space is the union of all name spaces in the "ver-" trees for all applications. However, there are some exceptions: if there are multiple versions of an application, only one version is used (e.g. the newest); if there are multiple variants of a file (for several architectures), only the file most relevant to the

```
/store/bin/bison              ->/store/store/khym/bison/ver-1.22/bin/bison@hpux
/store/lib/bison.hairy        ->/store/store/khym/bison/ver-1.22/lib/bison.hairy
/store/lib/bison.simple       ->/store/store/khym/bison/ver-1.22/lib/bison.simple
/store/man/man1/bison.1       ->/store/store/khym/bison/ver-1.22/man/man1/bison.1
/store/doc/bison/bison.dvi    ->/store/store/khym/bison/ver-1.22/bison/bison.dvi
/store/doc/bison/bison.ps     ->/store/store/khym/bison/ver-1.22/bison/bison.ps
/store/emacs/info/bison.info->/store/store/khym/bison/ver-1.22/emacs/info/bison.info
```

Fig. 7. Symlinks Generated for Application in Figure 6 (HP)

linktree is used, and the suffix part is removed from the links in the `/store` name space. The linktree is a view into the file repository under `/store/store`. For the filenames that are listed in figure 6, the corresponding symlinks generated are listed in figure 7.

Our next concerns are consistency and access time. Consistency requires that an application is only stored at one place in order to avoid two diverging copies. Access time requires that many copies are distributed among the computers in a network, in order to minimize access time and to ensure continuous access to the application even if parts of the network are inaccessible.

In order to satisfy these two requirements, Store implements "slave store servers". They are normal stores, except that applications stored on a slave store are automatically copied from the normal store (a.k.a. "master store"). The master store holds a repository of application code, the slave stores contains replicas of it, and the linktrees provides a view in to the repositories and replicas. Thus, consistency is ensured by uniqueness. Every piece of information should be stored at only one location. If further copies are required, they are generated from the master copy and repeatedly synchronized later.

All maintenance is performed at the master server. The main function of a slave store is mostly to cache applications from the master server in order to serve local linktrees. Calling it "master" and "slave" servers is a bit imprecise. Stores are either masters or slaves on a per-application basis: Given two servers and two applications, the first server can be a master store for the first application and a slave store for the second, and vice versa for the second store.

A simple directory naming convention notes the difference between slaves and masters. The directory `/store/store/khym/tgrind` is the master copy, while `/store/store/storlind/.tgrind` is a replica, due to the name starting with a dot. This convention also ensures that the replica directories stay out of sight in normal `ls`-listings.

Each linktree under `/store` will pick an application from a master or slave store for that application, and it points its symlinks towards it. Which server it chooses depends on which machine is closest measured in a distance metric, a metric that can be configured separately for each `/store` linktree.

Store slave servers only copy the files that they need, i.e. source code is not copied, are architecture-dependent files for hardware that none of the linktrees connecting to that server can ever use.

7 A Full-Scale Example

To flesh out the description given in the previous section, we here show a full-scale example, involving two stores, one application, four linktrees, and four architectures.

The application in question is "mpage", which consists of a binary file `mpage` (architecture dependent), and a manual page `mpage.1` (architecture independent). Figure 8 shows part of the file system on four machines: khym and storlind are servers, master and slave respectively; dekk and chur are linktree-only machines. In this figure, dotted, pointed lines indicate NFS-mounts; solid, pointed lines indicate symlinks; fat lines indicate directory structure; and thin lines group directories and files physically belonging on one machine.

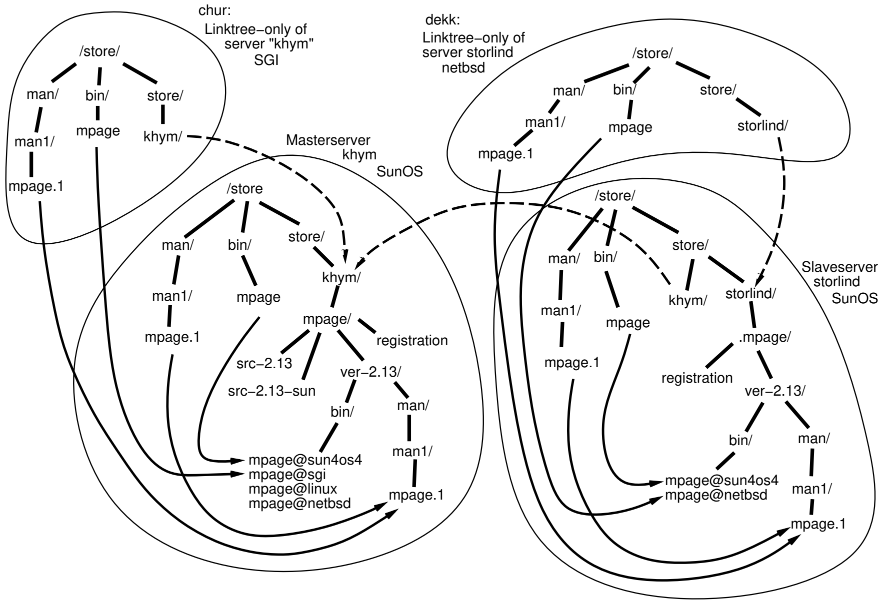

Fig. 8. Full-Scale Example of Store System

The figure shows that the slave server only copies the files for "sun4os4" and 386 "netbsd", the two architectures it serves. Names ending with a "/"-character refer to directories. The linktrees generated for the four machines are shown in figure 9.

Compared with a traditional `/usr/local` directory structure, there are several practical advantages of this scheme: Given a file in Store, one can determine which application and version it belongs to merely by looking at what the symbolic link expands to. By following the symbolic link, we can find any and all

For master store khym	
/store/bin/mpage	->/store/store/khym/mpage/ver-2.13/bin/mpage@sun4os4
/store/man/man1/mpage.1	->/store/store/khym/mpage/ver-2.13/man/man1/mpage.1
For linktree chur	
/store/bin/mpage	->/store/store/khym/mpage/ver-2.13/bin/mpage@sgi
/store/man/man1/mpage.1	->/store/store/khym/mpage/ver-2.13/man/man1/mpage.1
For slave store storlind	
/store/bin/mpage	->/store/store/storlind/.mpage/ver-2.13/bin/mpage@sun4os4
/store/man/man1/mpage.1	->/store/store/khym/mpage/ver-2.13/man/man1/mpage.1
For linktree dekk	
/store/bin/mpage	->/store/store/storlind/.mpage/ver-2.13/bin/mpage@netbsd
/store/man/man1/mpage.1	->/store/store/khym/mpage/ver-2.13/man/man1/mpage.1

Fig. 9. Symlinks Generated for Figure 8

other files related to that particular version of the application.

Users get their traditional `/usr/local` directory structure (although as symlinks), while system administrators get each application separated from the other applications. This means better overview and easier system maintenance. It also helps automatic generation of user documentation.

Each linktree may be configured to comply with various policies for updating to new versions. "Progressive" linktrees upgrade to the newest version of any application, even if it marked as alpha-test software. "Coward" linktrees only use versions marked as stable. Between these, several other policies exist. In addition policies may include the application's origin, i.e. which stores the linktree trusts.

Store is maintained by a number of Perl scripts. The scripts perform the majority of the tasks needed, except for most of the compilation process, which have proved difficult to automate on a general basis.

8 Where is it Being Used?

Currently, Store is being used at the University of Trondheim, where 10-20 of its departments and faculties cooperate in administering third-party applications through a common Store installation. A total of 300 Unix machines employ Store, that is approximately half the number of Unix machines at the University.

More than 400 applications are installed in Store. The degree to which hardware platforms are covered varies. However, most of the common Unix platforms (around 10) are reasonably covered. When checked, approximately one half to two thirds of the most frequently used user-commands at a typical workstation were loaded from Store. The remaining commands typically were shells (should always reside under `/bin`) and standard Unix commands.

Initial installations on institutions outside the University of Trondheim is being done, although none of these are yet of the same size as the installation at the University of Trondheim.

9 Experiences

We have learned many lessons while working with Store.

Higher competence. While Store decreased the amount of system work required, it increased the need for those fewer persons to have better skills regarding the applications they manage.

Prevalence. As soon as we reached a threshold (i.e. a large number of applications and a fair number of hardware platforms), new faculties and departments joining the Store-team got most of what they ever needed of freeware and third-party applications. Since their only cost was the price of extra disk space, this service tended to become very popular.

Vulnerability. After having used Store for a while, we have experienced higher vulnerability to incorrect installations. We are so to speak putting all the eggs into one basket; if someone introduces a fault into an application it is spread throughout the university within the next day. To counter this, much work has been put into authorization and configuration mechanisms. Also, Store makes it equally easy to correct faults throughout the university within a day.

A method for reducing this problem is to have an empiric form of quality assurance: No automatic updating of the slave stores, and only manually initiated updating of specific applications on the slave store servers.

Number of architectures. During the first period of Store, we experienced a strong trend towards standardization on a small number of OSes supported by Store. Lately, we have experienced a trend the other way; all it takes is one OS-bigot willing to recompile all Store-applications for "her" OS, and that OS is supported throughout the university on the same basis as major OSes.

Consistency. Using Store as a framework for installing applications ensures that one application is installed the same way, independent of hardware, ownership, and disk-sharing strategies. Only the first person installing a new application decides the strategy for where to put the files. Other persons, recompiling for new hardware architectures or upgrading versions, can simply follow the initial setup. This often implies less work, and the work remaining contains fewer decisions.

Symlinks. Store depends on symlinks, and that has posed some problems, although not as many as feared beforehand. For instance, the maximum number of links allowed by Unix (normally 8, although some OSes have more) has proven too small in some circumstances. When using the automounter, `/store` is itself a symlink, which increase the problem quite a bit. In addition, if the linktree is given a partition by itself, it might require an increased number of inodes, because of the large number of small files.

More disk usage. A side effect of using Store is an increased use of disk space. But this extra cost is small compared to the perceived reduction of system administration cost. To compute the overhead introduced by Store, assume a system consisting of 2 master stores serving 4 slave stores each, four architectures, 10 linktrees (one for each server), 200 applications. Per application, assume 3 architecture dependent files of an average size of 100 Kb and 12 architecture independent files of an average size of 20 Kb. The extra overhead in the linktrees is 10 servers $\times$ 14 symlinks $\times$ 200 applications, giving 24,000 symlinks. If each symlink occupies 512 bytes, this means a systemwide overhead of 12 Mb plus some extra, minimal space for directories. The files in a `/usr/local` tree would be 10 servers $\times$ 200 applications $\times$ [for each application 12 files of 20 Kb and 3 files of 100 Kb], resulting in a total of 1080 Mb. The extra files in Store which are not directly used are 2 servers $\times$ 200 applications $\times$ 3 variants $\times$ 3 files $\times$ 100 Kb, giving 360 Mb. This means a 33% disk-space overhead for this configuration.

Continued maintenance. Continued maintenance is important. We receive a daily mail, reporting the status of the Store installation. The output is adjustable. The reports may also be centralized if one support staff handles multiple Store installations.

Reduced Installation time. When a new computer lab with 10 machines was installed, it took only 20 minutes to link up all applications in Store that supported the given architecture. This is very fast, compared with the alternative situation: recompilation of all those applications.

The final decision is made locally. It is possible to *freeze* a replica, inhibiting further automatic updates for applications considered critically important. If a department wants to use a specific version of an application, they are free to do so. They can even set up their own master for that application. The `/usr/local/bin` directory can be placed in the PATH environment variable before `/store/bin`, in order to let locally installed applications take precedence over Store-applications.

Better interdepartmental cooperation. Departments using Store are implicitly encouraged to cooperate. This has resulted in positive side-effects at this university: a) university-wide documentation, b) sharing of human resources, and c) the quality per application is maximized.

Less duplication of work. Store has resulted in a reduced workload, mainly due to less duplication of work because of interdepartmental cooperation.

File name space mangling. The technique of mangling the attributes into the file system name space have proved sufficiently strong. In addition, it is simple, portable, and the limit of 255 characters in file names does not pose a problem. It is even precedented, as compilers use the "file type" to decide the type of contents of a file.

Name space dimensions. Store only uses two dimensions: architecture and domain. However, we envision at least five such dimensions: hardware, operating system, geographic location, organizational affiliation, and release type (alpha, beta, etc). Store uses one dimension (architecture) for the two first; and another (domain) the the two next. The last dimension is put into the configuration file for the application, as all files of a version in an application are considered to be of the same release type.

10 Further Developments

Possible future developments include:

Alternative transport mechanisms. Currently, NFS is the only supported transport mechanism. In the future, support for other mechanisms will be included: ftp, the alex file system, world wide web, etc. Also, support for transport to other systems than Store will be implemented.

Non-Unix platforms. Some initial experiments have been performed with PCs under DOS, using PC-NFS as the transport mechanism. PC-NFS handles symlinks, and technically it worked fairly well. However, some major conceptual problems turned up: most freeware applications are either Unix-only or DOS-only. Thus, the advantage of maintaining a multi-platform configuration of packages disappears.

Another problem is that site-specific-configuration and user-specific-configuration are often mixed together in the same configuration file under DOS; under Unix these are generally more separated.

Furthermore, under DOS and Windows, programs performing certain tasks during program installation that Unix typically performs during program startup, e.g. mail programs under DOS typically set the user name during installation, while under Unix the programs read the user name during startup. Thus, DOS-applications are often single-user applications that are difficult to incorporate into a system like Store.

Automatic documentation for the end-user. We have plans for a WWW gateway to semi-automatically generated hypertext documentation about the applications in Store, where the user can navigate between information about applications, the persons that installed those applications, the category of an application (e.g. image processing, games), the licensing conditions, the manual pages, info pages, examples, release notes, local diffs, and so on. Added to this, some search forms should also be present, so that the end user can easily find the program needed given some keywords about what the user wants to do. This would give one Web "home-page" per application, through which all information about the system is accessible.

Improved feedback. Feedback information should go directly by electronic mail to the person that installed an application, not just to a support team.

Availability monitor. NFS, although stateless, is very vulnerable to a situation where the server for a mounted file system becomes unavailable. If the server does not come up again, programs accessing those mount points will hang, causing the client machine to become rather unusable. Neither Sun's "automounter" nor "amd" provide the needed functionality to switch to a different server on-the-fly. By having identical stores with application replicas only, and an extra level of indirection, one can switch to a server that is available when the previous one becomes unavailable. Note that this is also possible with the traditional approach, just let `/usr/local` be a symlink to e.g. `/usr/locals/`*machine*.

Virtual stores. Currently, stores are located below `/store/store`. However, by adding an extra level of indirection, the level specifying the location of the store can be removed. If `/store/store/idt` is a virtual store, then it can dynamically be replaced by any of a set of identical stores. To become effective, this feature needs the availability monitor. However, for two stores to be identical, they cannot be master stores for any application.

Conflict handling. When conflicts between applications occur, they can be resolved, based upon metrics describing the importance of the application and the source of an application (i.e. which master store, who installed the application).

11 More Information

More information about the Store system can be found at the anonymous ftp site ftp.pvv.unit.no, in the directory "/pub/store", and through World Wide Web from http://www.pvv.unit.no/~arnej/store/storedoc.html.

12 Conclusion

We think that Store provides a suitable environment for installing third-party applications, and eases the the daily tasks of the computer systems administration staffs. The main effects are better overview; the automation of repetitive and trivial tasks; ensured consistency; the considering of many computers as one system.

We have got positive experiences from installing Store at various installations at the University of Trondheim, where currently, around half the Unix-computers use Store. The feedback from users have been generally pleasant, although most users are not aware of Store, even though most of their tools are installed in Store.

References

1. Stephen N. Clark Kenneth Manheimer, Barry A. Warsaw and Walter Rowe. The depot: A framework for sharing software installation across organizational and unix platform boundaries. Proceedings of the Fourth Large Installation Systems Administration IV. Colorado Springs, Colorado, October 1990.
2. Thomas A. Lundgren. A file server directory tree hierachy. Technical Report ETX/TX/DD-91:110, Ericsson Telecom, January 1992.
3. S. Boucher, M. Dagenais, R. Gérin-Lajoie, P. Laplante, and P. Mailhot. LUDE: A Distributed Software Library. University of Montreal, June 1993.
4. Melvin J. Pleasant Jr. and Eliot Lear. Transcending administrative domains by automating system management tasks in a large heterogeneous environment. USENIX Workshop Proceedings. New Orleans, Louisiana, April 1989.

This article was processed using the LaTeX macro package with LLNCS style