RZ 2748 (# 88852) 06/06/95
Computer Science/Mathematics 8 pages

Research Report

Using Symbolic Links and Name Resolution of Distributed File Systems to Improve Version Management and to Optimize Access

R.C. Hauser* and A. Tanner

IBM Research Division
Zurich Research Laboratory
8803 Rüschlikon
Switzerland

*Present address: McKinsey & Company, Unterwerkstr. 3, 8065 Zürich, Switzerland

Research Division
Almaden · T.J. Watson · Tokyo · Zurich

Using Symbolic Links and Name Resolution of Distributed File Systems to Improve Version Management and to Optimize Access

R.C. Hauser* and A. Tanner

IBM Research Division, Zurich Research Laboratory, 8803 Rüschlikon, Switzerland

Abstract: The purpose of this paper is to show that the sophisticated exploitation of features of modern file systems enables system administrators to implement advanced version management and efficient data access without complex software management tools. User autonomy can be kept at a maximum with the described approach while the central administration remains simple. This can even be improved, if the proposed file system extensions are implemented.

*Present address: McKinsey & Company, Unterwerkstr. 3, 8065 Zürich, Switzerland

1 Introduction

Modern distributed file systems provide additional features beyond managing the data on local storage devices: They provide a global directory tree with transparent wide area file access. In the following explanations, we use AFS [1] as an example for such a distributed file system. AFS also accommodates heterogeneity of machine code of essentially compatible systems with its `@sys` path component of filenames.[1] If "Makefile"s and source code employ an appropriate directory structure, it is possible, for example, to compile applications using the same source files and the same storage medium from workstations built by different manufacturers using different machine-level code.

This paper discusses in Chapter 2 how different versions of a software package can be managed by appropriate directory structures and symbolic links such that it is possible to switch back and forth without restrictions. Next, it is shown how multiple versions of a package can be run in parallel and how test users can be accommodated. Chapter 3 explains how filename usage can optimize access patterns to persistent data among different cells in a global directory tree - within this paper we will henceforth call a system administrator's area of responsibility with the AFS terminology a "cell." Sections 3.3 and 3.4 outline a proposal concerning how the file name evaluation of the `namei` function could optimize the access pattern in a two-layer storage hierarchy inside a cell having central file servers and workstation disks connected by a local area network.

2 Beware of "install"

Network Resource Tools such as "Archie" [2] have made it easy to find useful code to install in an administrator's cell. The current problem is therefore no longer locating code to be installed, but avoiding cluttering the storage with obsolete versions.

It was already recognized that, for manageability reasons not all source code can be deposited in one big source directory, but that subdirectories should be built in a `/local/src`[2] directory for each package, for example Ghostscript in `src/GS`. This leads to the question where to place the binaries of such packages in order not to have too many directories in the search path. The common `install` commands move or copy binaries to such specified or standard places as `/usr/bin`, `/etc`, and `/usr/lib`. `Makefiles` copy or move files into the common directory tree for such purposes as help files, dynamic linking, library sharing and configuration.

This leads to the following issues:

- It is often unclear what files belong to which package or which version of the package and whether they are still in use.
- Many packages need extra environment variables in order to be full operational. Where should this be done?
- How can various users use different versions of the same package simultaneously?

[1] The "`namei`" function provides this feature, and is configurable by the command "`fs sysname`" of p. 59 of Ref. 1. (In some kernels, `namei` is the name of the function that maps the path names to the internal unique data identifiers, e.g. the "inodes.")

[2] We chose the name `local` as the root of the directory tree for the software we installed beyond what was provided by our operating system. Any other path name could stand as an arbitrary prefix.

2.1 The "curr" Approach

The goal of this approach is to keep several versions of the package fully operational and to be able to remove a version entirely with one command. This can be accomplished by having all versions of a package as mentioned in a subdirectory of `src` denominated with the version number, for example: `src/GS/2.4`.
In the `src/GS`, a symbolic link is made:

```
ln -s 2.4 curr
```

and in the `/local/bin` directory that is in the search path, another symbolic link is made such as:

```
ln -s /local/src/GS/curr/*path_inside_package_tree*/gs gs.
```

Similarly, links are made in `man/man?` to the "curr" location of the manual page, and potentially other files in `lib`, `etc`, etc. as well.
This allows all files belonging to a package to reside together in one defined place and the entire user community can be switched to a higher version of a software package simply by reassigning the "curr" link. If this new version turns out to have problems, the user community can be switched back with similar ease (see Fig. 1).

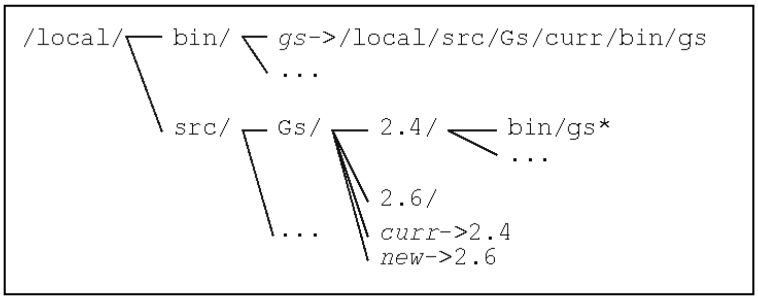

Figure 1: Usage of the "curr" Links.

`/local/` local root directory
`/local/bin/` directory in path contain the symbolic links
`/local/src/` directory with the complete packages.

2.2 Environment Variable Management and Test Users

Many packages require various further **environment** configurations on top of having the executable somewhere in the searched `PATH`. The authors of "Soft" [3] have developed a sophisticated tool to address this.
A simple alternative solution is to have in the `/local/bin` directory a *start-up shell-script with the package name* (`gs.sh`) instead of the generic link to `/local/src/GS/curr/*binary_inside_package_tree*`. The "Soft" tool provides shell independence, but this is not a problem with our proposal either: The common shell languages ({b}sh, ksh, csh) all entail test whether a variable is already set. If not, it can be set in this script with a minimum scope to keep the general environment small, namely only for the pertinent application. If the configuration

consists of ordering the components of a variable value such as the order of the components in the `PATH` variable, this environment variable preprocessing can still be done before launching the application binary. If the binary is launched by this script with the complete command-line argument array provided by the user, this script is transparent to the user because shell scripts inherit environment variables independent of which shell set them. This transparency of the start-up shell script is only missing when the user intends to interfere directly with the environment variables which are "preprocessed" by the start-up shell script as opposed to being set. The solution for this rare case is that the user copies the cell-wide start-up shell script into his personal workspace, configures the code himself, and ensures that it receives precedence before the administrator-provided start-up shell script.
This shell-script approach can also be used to specify **`test users`**: The result of the `whoami` is compared with a list of test users, and for these users the configuration is set to the version to be tested. For users not assigned to the test user group by the administrator, the new version of the application can be employed by calling the program with the extension "`.new`" (e.g. `gs.new`) (confer Fig. 2).

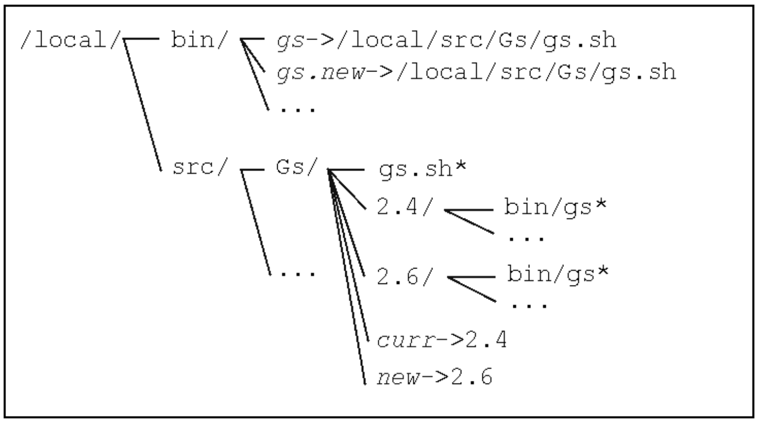

Figure 2: Usage of the start-up shell script.

If several versions of a package are to be used *truly* in parallel under the same name, all configurations should be performed by means of environment variable management in the mentioned start-up shell scripts because in this case, the `curr` symbolic links cannot be used, for example, for manual pages.
The command gs calls the start-up script gs.sh. Inside this script the `$rootDir` variable for example, is set to `/local/src/Gs/curr` or `/local/src/Gs/new` depending on the invocation name (`ARG[0]`) and/or the calling user (for "test users"). At the end, the script "exec"s to `$rootDir/bin/gs`.

2.3 Evaluation

- Outdated versions of a package can now be removed entirely with a simple `rm -r` in the pertinent directory. If the structure and naming of the contributory files changes, the worst case is that dangling `curr`-links might still exist, however, they do not occupy much space and are easily found.

- Having the system's `PATH` variable pointing to cell-wide binary directories that are not on the local disk is not significantly detrimental to access speed, because these directories hold only small amounts of information, namely the `curr`-symbolic links or the start-up shell scripts, and therefore will shortly be entirely cached.

- There are further mechanisms in Unix that may cause disks eventually to be cluttered with unused information such as the `man/cat?` versus `man/man?` mechanism. These problems can most likely all be solved on a case-by-case base: e.g. by writing a garbage collection routine that removes all `man/cat?/?*.?` entries if the pertinent `man/man?/?*.?` ceases to exist.

3 Data Access Efficiency

Owing to the decreasing cost of disk space, the workstations in distributed computing environments are increasingly being equipped with local storage. Therefore, there tend to be three classes of access speeds in such global file systems:

- access to the local disk through the bus,

- access to the cell's server by means of a fast LAN,

- and access to arbitrary data in the "wide area."

It appears that the sharing mode of file accesses in large distributed systems depends largely on the location of the accessor. Whereas users within the cell of the physical location of some data may happen to alter the data, remote users very rarely have other intentions than to sharably read it. In this scenario, caching approaches like the ones of AFS turn out to be inefficient because "warm reads" of users of the same workstation profit from the caching of remote data, but not other users in the same cell. Thus, even though the communication within the cell is much cheaper than over long distances, data is likely to be identically transmitted over the expensive long distance connection, but no effort is made to optimize transmission over the less-expensive LAN.
In the world of WWW, this has been recognized, and proxy servers that cache information on the cell level have been proposed [4]. Research has been done to integrate such two-layer hierarchical caching into filesystems to improve scalability (e.g. Frolic [5]), but they are not yet commercially available.

3.1 "ThisCell"

System administration appears to be a discipline in which progress propagates quickly. Therefore, much of the basic software infrastructure beyond the manufacturer's product code distribution is likely to be installed in an almost identical way at most sites. It is therefore advisable not to use the full cell-name when writing, for example, a `perl` [6] shell script to configure and manage an application X. If somebody from a remote cell wants to try out this new application X, the code of X obviously has to be transmitted long-distance. However, it is unnecessary to transfer the entire perl code as well.
Therefore, a symbolic link

```
cd /afs
ln -s <cellname> ThisCell
```

is made, and each occurrence of `/afs/<cellname>` in a script is replaced by `/afs/ThisCell`. On the <Info-AFS@transarc.com> mailing list, a "@cell" mechanism similar to the "@sys" has been proposed independently.[3] A kernel implementation of such an "@cell" mechanism has the advantage that the lengthy process necessary for the community of autonomous system administrators to agree on such a convention and implement it can be avoided. The above explanations, however, show that kernel implementation is not mandatory and can be substituted by the proposed symbolic links.

3.2 "ThisMachine"

The same idea also applies to disk-equipped workstations. The `ThisMachine` symbolic link points out of the `/afs` directory to the root of the workstation's physical disk. This permits the workstation owner to retain the privilege of configuring his workspace directly. For each `/afs/ThisMachine/local/bin` directory, it must be individually set whether the `curr` symbolic link points to the current version of the package installed on the local disk or to `/afs/ThisCell/local/src/Gs/curr/`.

3.3 "namei" Extension Proposal 1: "ThisMachine - ThisCell" Fall-back

The cell administrators should still, for example, be able to install patches or new versions of packages across their cells, i.e. also on workstation's local disks of the "autonomous" users.
In order to avoid having to find out whether the symbolic link `curr` in the workstation's package root path directory, e.g. `/local/src/Gs/`, was set by the user, we propose the following extension and policy for its use:

- `/afs/ThisCell/local/src/Gs/curr/` always points to `/afs/ThisMachine/local/src /Gs/*current_package_version_number*/`. If that directory is present on the workstation's disk, everything is fine; if that directory is not present, the `namei` will make another attempt with `/afs/ThisCell/local/src/Gs/*current_package_ version_number*/`. This allows one to decide whether a package is on the local disk or only available cell-wide.
- Furthermore, `/afs/ThisCell/local/bin/gs` points to `/afs/ThisMachine/local/src /Gs/curr`. By setting this link, the user can autonomously select the version he desires. If he omits setting any such link, the above setting prepared by the cell administrators will be assumed.

3.4 "namei" Extension Proposal 2: "ThisCell - Calling Cell" Fall-back

There remains the problem of how to convince the cell administrators in all cooperating cells to follow the "standards" of locations.
One approach is to introduce another level of fall-back: If some package is not found in the specified "`ThisCell`" location, the cell name in the path of the script that calls this package

[3]In the same discussion, further suggestions were made, such as having an @cluster, @dept, @group, or arbitrary variables instead. It appears that such hierarchies can be modeled without system support for example by keeping the appropriate order of directories in the `PATH` environment variable. An exception may be the "cluster" if there is truly a significant physical access difference between the cell, the machine, and the cluster.

is taken. It often happens that scripts invoke one another over many levels. This mechanism therefore must not only fall back to the parents but try all ancestors before it claims not to find a given file (for a pictorial representation see Fig. 3).

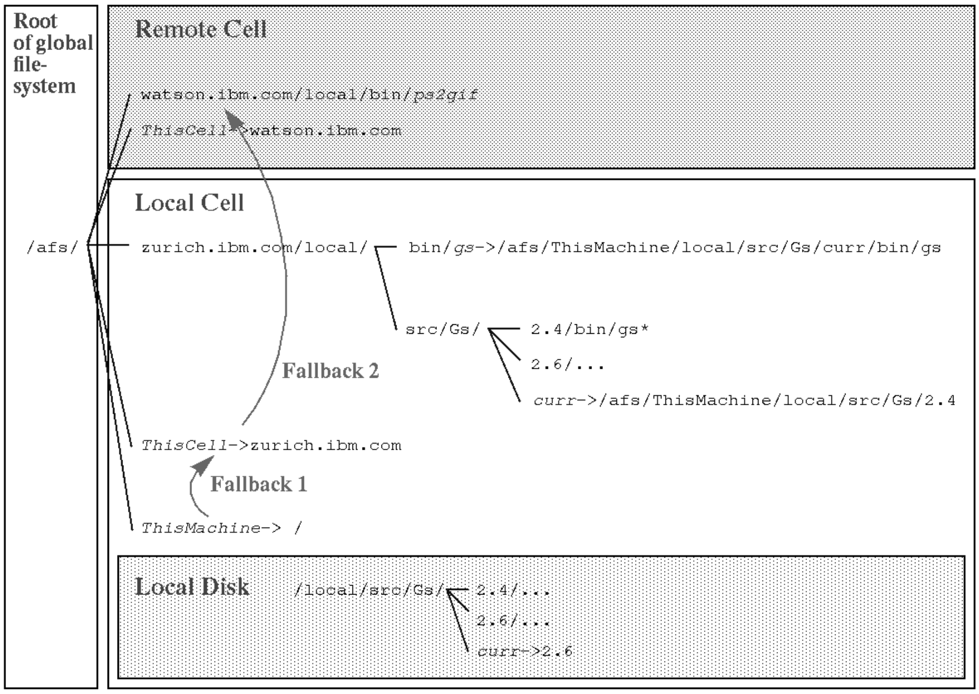

Figure 3: The "namei" Fall-back.

3.5 Evaluation

- The main difference between file-system-integrated caching and WWW proxies or the "ThisCell" mechanism is that system-integrated caching entails call-back promises from the server to the client, whereas proxies and ThisCell do not, i.e. they rely on expiration heuristics or polling of the server to determine whether a given cached file is still current.

- A problem appears if newer versions entail significantly different semantics, such as the step from perl version 4 to version 5: If the cell from which a script is coming has already upgraded to version 5 but the executing cell has not, the "ThisCell" mechanism will fail.
  Upward compatibility:
  The script must then call `/afs/ThisCell/local/src/perl/5/curr/` instead of `/afs/ThisCell/local/src/perl/curr/`. The `curr` will then only select among the semantically identical minor versions of semantically different major versions. Once a cell has migrated to, say, version 7 and wants to get rid of version 5, the directory `/afs/ThisCell/local/src/perl/5` will be removed and replaced by a link pointing to `6`. This mechanism uses only very little storage and therefore such a symbolic "upward compatibility"

link should only be removed long after the pertinent version is removed and no relevant cooperating cell is likely to have old scripts with a reference to this obsolete version number.

- These fall-back mechanisms in the file name resolution function can make the debugging of programs tricky because it is difficult to know from which location a package was finally taken. It is therefore recommended that a tracing flag be added such that the `namei` function always tells which path the information was taken from to e.g. `stderr`. Another flag should be added to turn the fall-back mechanisms entirely off.

The "curr" links and the start-up shell scripts have been used successfully in our cell for several years. The "ThisCell" has also been employed since several years, but getting other cells to cooperate has not been easy.

4 Future Work

An important step to having a wide majority of cell system administrators follow the outlined procedures is to rewrite the standard `Makefiles` so that they

- use an `@sys` component to model machine dependencies. If the file system of a machine does not know such a `namei` feature, this can easily be simulated manually by making a symbolic link with the name "`@sys`" to the respective directory,
- implement the `install` target such that all binaries and other data remain under the package root,
- add a `commit` target such that either the `curr` links are built or the start-up shell scripts are adapted and placed in the correct location.

The next step to facilitate the management of autonomous cells in a conglomerate of cooperating cells that trust each other at least partially might be the following:
Instead of fully installing a newer version in one's cell, the start-up shell script could merely force the test users to take the newer version from a cooperating cell that has already installed it. The test users will initially suffer some performance reduction until the new version is in the cache. Once they are familiar and satisfied with the new version, it should be possible to extract it from the cache and bring it into the explicitly administrated file system of the cell without having to obtain the sources nor compiling it. In order to find all relevant files in the cache, the above-mentioned tracing mechanism could be extended to monitor all file accesses out of a package. For this purpose and to provide robustness towards partial network failures, technologies such as "hoarding" that were developed for disconnected operation [7] will become a valuable building block.

5 Conclusion

With the efficient structuring of directories, symbolic links, and shell scripts tuned to this approach, good results concerning the version management of software packages can be achieved without having the overhead of complicated tools.
If the file-name-resolving function has some additional features such as the `@sys` mechanism in AFS or the extensions proposed here, transparency with respect to machine code level incompatibilities as well as efficient storage and access hierarchies for bulk data can be achieved.

The latter can be greatly facilitated if a consensus regarding appropriate `Makefiles` software package structures can be found among cell administrators, i.e. such "conventions" could be recommended as standards of good practice by their professional organizations.

Site Information

Our Distributed Computing Systems group manages the Unix and Intel based part of the computing environment of the IBM Research Laboratory in Rueschlikon, Switzerland. This currently consists of approximately 300 PCs, 100 XStations, 160 IBM RS/6000s and a 16-node IBM SP2 with some 300 active users spanning the entire spectrum from "naive" users to "hacking" computer scientists. The workstations are distributed between our two locations, which are about 5 km apart and interconnected by an 8-Mbit/s link. All user data and most of the applications are stored centrally in an AFS file system with five fileservers. Our cell is also connected to the global IBM-internal AFS tree.

Keywords:

Caching, symbolic links, version management, configuration management.

Acknowledgments

We thank Roger Larsson, who initiated many of these approaches, and Michael Niksch for helpful contributions.

References

[1] John H. Howard, Michael L. Kazar, Sherri G. Menees, David A. Nichols, M. Satyanarayanan, Robert N. Sidebotham, and Michael J. West. Scale and performance in a distributed file system. ACM Transactions on Computer Systems, 1988.

[2] EARN Staff. Guide to network resource tools. Internet Request For Comments: RFC 1580, March 1994.

[3] Rémy Evard and Robert Leslie. Soft: A Software Environment Abstraction Mechanism. In Proceedings of the Eight Systems Administration Conference (LISA VIII, San Diego, CA), pages 65-72. Northeastern Univ., September 1994.

[4] Ari Luotonen and Kevin Altis. World-wide web proxies. In First International Conference on the World-Wide Web, Geneva, May 1994. Elsevier Science Publishers.

[5] Harjinder S. Sandhu and Songnian Zhou. Cluster-based file replication in large-scale distributed systems. Technical Report 255, Computer Systems Research Institute CSRI, University of Toronto, Toronto, Canada, M5S 1A4, January 1992.

[6] Larry Wall and Randal L. Schwartz. Programming perl. O'Reilly & Associates, Inc., 1992. Practical Extraction and Report Language.

[7] M. Satyanarayanan, James J. Kistler, Lily B. Mummert, Maria R. Ebling, Puneet Kumar, and Qi Lu. Experience with disconnected operation in a mobile computing environment. In Proceedings USENIX Symposium on Mobile & Location-Independent Computing, pages 11-28. USENIX, August 1993.