SFS
Slow File System

Giuseppe A. Marino, Giancarlo Succi
DIST - Università di Genova

1. Introduction

During the last years there have been a growing interest in public archive systems; even a whole newsgroup of USENET is devoted to discussions and diffusion of information about public archives (comp.archives). Many sites of Internet offers anonymous FTP services; in the UUCP world there are some sites, mainly in the U.S., which allow for dial-in by any UUCP host, giving access to a large amount of "public domain" software; within EUnet, both the "superbackbone" mcsun.eu and many national backbones provide well managed archiving services. Why a public archive is a valuable service can be seen by looking at the following, frequently asked, questions:

(a) "Where can I find a copy of RFC1024?"
(b) "Where can I find the latest version of GNU goc?"
(c) "Are there any patches for X11R4?"
(d) "What public domain mail interfaces are available?"
(e) "What is InterViews?"
(f) "How can I know what new games are available for MacIntosh?"

Question (a) implies that data are important. Question (b) says that up-to-date files are more important. But (c) means that complete and documented files are even more important.

To answer (d), an archive must be organized in a congruent, well-structured hierarchy, while (e) is related to on-line documentation services. Finally, (f) calls for a service of automatic distribution of updates.

The first conclusion from this review is: there is a value in the structure itself of the archive not only in the archived files.

SFS is a first attempt in distributing well structured file systems among a community of geographically dispersed sites.

2. SFS from a user point of view

The SFS pseudo file system can be hosted into any ordinary UNIX file system. A user can browse through it using the standard UNIX commands (like ls or cd). Accessing individual files is a bit different: our target is to make it as transparent as possible, although the implementation we have completed so far requires the use of a specialized command (tick): when a user try to open a SFS file, the operation fails, returning an error: "Try again later, file is being transferred". In the meantime, a background process gets a copy of the real file from a remote archive host, substituting the placeholder file.
An example:

```
% ls -1 nsfnet-doc.tar.Z
-rwxrwxrwx 1 root 8 Dec 28 18:34 nsfnet-doc.tar.Z -wtmp
% ls -l nsfnet-doc.tar.Z
-rwxrwxrwx 1 root 8 Dec 28 18:34 nsfnet-doc.tar.Z
```

We plan to add an estimate of transfer time, characterizing more accurately the "later" above; up to now the user can infer how long it takes if he knows what kind of communication link is used for accessing the remote host.

SFS can use two different means for transferring files: FTP[1] and UUCP[2]. Both are "applications" (with reference to the OSI layers model), but as far as SFS is concerned, they can be seen as "data links" or channels. UUCP runs usually in time-activated spooling mode, i.e. with a time scale measured in hours, while FTP can complete transfers within minutes.

The user of a well-managed configuration of SFS can face a virtually unlimited archive, obtained by superimposing in the same directory hierarchy the archives of many remote hosts. The actual size is limited by two factors:

1. the disk space consumed by the hierarchy of directories and placeholder files (on a 4.2BSD filesystem approximately 1kbytes per directory and 10 bytes per file);
2. the disk space used by real files once transferred and installed by the SFS interface (this can be seen as a cache memory, with transit time measured in days or weeks).

From our experience, with about 100 Mbytes it is possible to represent archives with several Gigabytes of data.

3. Implementation of SFS

The prototypical version of SFS is running at the backbone site of EU nets in Italy (Carlia.dist.unipd.it which happens to be hosted at our department); its functional modules are implemented as a set of scripts written in a language suitable for "fast prototyping" (PERL). We have chosen this approach because we want to build up an experience base about all aspects of the proposed system before coding it in a more "serious" language, like C. Even in this present form, performance is quite good, and we have been able to change the specification and rewrite almost all scripts within weeks many times. Furthermore, the package has a high degree of portability among all existing flavours of UNIX, because PERL itself is quite easy to port.

3.1. PERL as a prototypical language

PERL[3] was designed by Larry Wall (<wall@jpl-devvax.jpl.nasa.gov>) as a substitute for both AWK and Shell scripts. It provides all the typical features of AWK (associative arrays, pattern matching) within a syntactic framework very similar to C (although some of its compound statements are quite peculiar and, for our taste, rather uncomfortable).

---

1 thanks to Pat O'oble copyeditors made but 6
to, for this idea.

2 with one, small, but mandatory, requirement: symbolic links.
updateSFS  installing for the first time a new remote archive, or updating an already mounted one is done by updateSFS. It takes a file which contains the list of all directories and files of the remote system and builds a local copy of the hierarchy, using symbolic links as placeholder files. Each link is a "fake" file, whose name is conventionally composed by concatenating the hostname (or hostid) of the remote system and the size of the real file. As an example:

| lwxrwxrwx 1 root 11 Dec 28 18:44 CLX:R4.1.text.Z -> mcsun:1088k |
| lwxrwxrwx 1 root 10 Dec 14 19:23 CLX:R4.3.text.Z -> exp1:101k |
| lwxrwxrwx 1 root 10 Dec 14 19:23 CLX:R4.4.text.Z -> exp2:103k |
| lwxrwxrwx 1 root 10 Dec 14 19:24 Pipes/ |
| lwxrwxrwx 1 root 9 Dec 14 19:23 SeeTeX-2.16.1.tar.Z -> expo:46.4k |

Usually a well-managed archive maintains on-line a file with the list of all available files; there is no agreement about the form this file should adopt: a few different forms are common, all based on different flavours of the Is-lR command. UpdateSFS uses a configuration file for associating the format and the name of the file to each remote host.

When it is possible to use FTP for retrieving files, updateSFS is able to retrieve the file from the remote host immediately, completing the whole operation in a single run. If the communication link is based on UUCP, the operation is split in two parts, using the same protocol described below for normal file transfer.

tick In the prototype version, there is a special command, visible to all users of the local SFS, which is used for signaling the attempt to access a placeholder file. It will eventually evolve into a library function, to be associated to the 'open' system call in a "Newcastle Connection[4]" style implementation, in order to make its use (almost) transparent.

Tick accepts a list of files as arguments and simply checks each file is a placeholder (otherwise nothing must be done), then queues a request for the real file, and marks the symbolic link as "in transit". As an example:

| lwxrwxrwx 1 root 9 Dec 14 19:23 SeeTeX-2.16.1.tar.Z -> expo:46.4k |

If the link type is FTP, a getSFS process is started in background, announcing the user that his logging for the file may last few minutes.

getSFS/putSFS The command responsible for most of the day-by-day work is DgetSFS. It is started either at given time marks during the day or by tick (a mechanism ensuring that at most one instance of getSFS per remote host is running is provided).

The script behaves differently for FTP- and UUCP-based communications. If the channel is FTP, it dequeues a request posted by tick, transfers the file, verifies it, and installs it into the right place (i.e. where the placeholder file was). The UUCP mode works in two phases: in the first one a copy request is issued to the local UUCP subsystem; when the transfer will completed, UUCP will start an auxiliary command, putSFS, which installs the file in the right place.

Both forms recover gracefully from errors, guaranteeing that files are eventually installed without human intervention.

3.3. Mounting remote file systems

The whole set of modules implements a "remote mount" abstraction, loosely resembling the "remote mount" of NFS. A configuration file, /etc/sfstab, lists all tuples (remote host, remote directory, local directory) which are defined for the local system.

As an example:

```
# Mount point table for SFS
# host/remote dir local dir
# funic:/pub/amiga /comp/amiga
# funic:/pub/doc /doc
# funic:/pub/TeX /text/TeX
# mcsun:/ripe /ripe
# mcsun:/documents /doc
# mcsun:/comp /comp
# mcsun:/gnu /gnu
# mcsun:/graphics /graphics
# mcsun:/mail /mail
# mcsun:/misc /misc
# mcsun:/network /network
# mcsun:/programming /programming
# mcsun:/security /security
# mcsun:/windows /windows
# exp:/contrib /windows/Xcontrib
# exp:/pub/h4 /windows/XR4
# exp:/pub/PEX /windows/XPEX
# exp:/pub/DOCS /windows/XDOCS
# exp:/pub/XTEST /windows/XXTEST
# unist:*tc /doc/utc
```

Note that all directories are "rooted" w.r.t. the home directory of SFS (usually "ftp" or "ftp/pub" or even "uucp"). Given a file /a/b/c/d/e on host remote, the algorithm that maps remote filenames to local filenames iteratively scans the table, looking for a key:

```
remote/a
remote/a/b
```

until either a tuple is found or the whole pathname is scanned. If a tuple, say:

```
remote/a/b/c/d
```

is found, the corresponding local pathname is /b/c/d, otherwise no mapping exists and the remote file must be discarded. In this way it is possible to select a subtree of a remote file system, and to rename subdirectories according to some local policy. A similar algorithm is used for reverse mapping from local name space to remote name space.

Note that it is allowed to have two or more remote hosts sharing the same local directory. In this case the two subtrees are completely merged; possible clashes of files with the same name are solved in updateSFS using a very simple priority scheme.
On a global basis, SFS may produce a significant increase in FTP and UUCP traffic. Indeed, this is one of the fundamental rules of technology: the easier the service, the higher the usage!

On the other side, using SFS we can avoid wasting communication bandwidth FTP-ing the same files many times, as we found when at least four users at our department retrieved the same version of GNU stuff from different anonymous FTP sites.

5. Conclusions

We have demonstrated the feasibility of the SFS approach for archive services. When wide-area networks based on limited bandwidth links, running heterogeneous protocols, probably involving different organizations, are taken into account, SFS could be the right solution. Although it was designed for solving the specific problem of archive sharing, other applications can be devised: there are many cases where replication of data, local availability, reliable transfer are rated more important than occasionally delayed execution.

We hope the availability of virtually unlimited archives will help widespread knowledge about existing public domain data and programs and will foster cooperation within computer networks.

SFS stands for "Slow File System" (stressing its slow response time), but malignant voices say it means "Stealing FTP Services".

References

- "PERL User Manual", available with the software distribution.
- "The Newcastle Connection or the World Unite": Software Practice and Experience, 1982.