Dear Giancarlo,

AUUG 91 has received an overwhelming response to our call for papers, well over sixty papers were submitted, of which we could only accept some twenty-seven.

I am delighted to inform you that your paper was judged by the committee to be in that top twenty-seven, and we are looking forward to your joining us at the end of September at the conference.

None of the submitted papers indicated whether the submitter was a full-time student at a tertiary institution, and as such we seem to have no one eligible for our student prize. If you are such a student please inform me quickly.

This is the first AUUG conference in which we will be running parallel streams. After lunch each day, we will run both a commercial and a technical stream. In the near future we will be sending you a conference timetable to let you know when and where your paper is placed.

As noted in the call for papers, there will be no facility for projecting overhead transparencies at AUUG '91. For those speakers without prepared slides there is a facility to have slides made up at AUUG's expense. The guidelines for those slides are set out in the speaker's kit which you will have shortly.

As Programme Committee Chair I would like to thank you for your paper, and look forward to meeting you at AUUG '91.

Andrew Gollan

Programme Committee Chair. AUUG '91

A Virtual File System
for the Coordinated Management
of Geographically Dispersed Archives

Joy Marino and Giancarlo Succi

DIST - Università di Genova
via Opera Pia 11a
I-16145 Genova
ITALIA

E-mail: [joy.charmi]@dist.unige.it

Abstract

The access to remote archives through the Internet Network is one of the most widely used network services; nonetheless, there is a lack of a nice, simple and efficient tool for knowing "what is where" and for retrieving data.

Our aim is to develop a tool offering both a better functionality for accessing and retrieving informations from remote archives and for encouraging a coordinated management of a set of archives.

Our strategy consists in making visible a virtual version of a remote directory hierarchy to the local user of a system, giving the user the capability of retrieving a remote file by means of a demand-paged like mechanism. The system is called "Slow File System" (SFS) for emphasizing both the Unix-like approach and the fact that the speed of retrieving data can be measured in terms of hours since the communication channels used are inherently slow.

This approach can be easily extended to handle the more general issue of integrating remote machines into a single structure: we can augment a SFS-like kernel with some strategies for mounting partial "interesting" views of remote file systems.

1 Introduction

The access to on-line archives is one of the most requested services of all cooperative networks. Archiving sites are common in Internet, in EUnet and in many others networks. Unfortunately, the tools for retrieving data from these archives are usually very simple and primitive, giving much more attention to security issues rather than to functionality or ease to use. Among these there are Anonymous FTP (PR3), which is so widely used that it has...
become a common expression in the IP world, meaning software/document publicly accessible via Internet, the UUCP protocol[Now78] which is even more restrictive, allowing only site-to-site file copying, and some distribution mechanisms based on e-mail, like the infoserver system used at most of the backbones of EUnet. Since E-mail access is ubiquitous, it is apparently very easy to access the infoserver services; furthermore, there is usually some form of "friendly" user-interface (or help support, at least). Unfortunately, using E-mail for sending large packages (and especially those binary-coded) is rather inconvenient.

We would like to stress that a publicly accessible archive is not just a simple file repository: the structure itself of a well managed archive is important (and it takes human resource and time to maintain). Being informed of what is available in the archive, which versions of packages, which patches, what is new, etc. may be even more important than the retrieval service.

If we consider all the software, and documents, and electronic information build up by the world community during these years, we find all of these will fill a Gbys archive, and that the process of keeping these information up to date will be an overwhelming job for whatever organization. Our proposal is to set up the tools for:

(1) a better functionality for accessing and retrieving information from remote archives;
(2) coordinate management of a set of, possibly replicated, archives.

Furthermore, because of our UNIX background and of our organization being involved in EUnet, we are interested in multiprotocol implementations which suit well within the UNIX framework.

2 Our Approach

The starting point is to make visible a virtual version of a remote directories hierarchy[KHR85] to the local users of a system, giving them the capability of automatically retrieving remote files by means of a demand-paged like mechanism. The system is dubbed Slow File System (SFS) making clear that the speed of this process can be measured in hours (sometimes in dozens of hours), since the communication channels used are inherently slow (9.6k or 64k baud leased lines running IP, or even slower with dial-up IP), and sometimes they are even "batch-oriented" (like the UUCP protocol suite).

We have devised an evolutionary path for SFS, starting with a prototype version in PERL[Wal88] (SFS 0, now completed), a simple version for "leaf" sites (SFS 1) and a more complex version for the coordinate management of a group of geographically dispersed archives (SFS 2).

The design of SFS 1 is based on the following elements:

(1) a program for building a local image of a remote archive, and for updating it on a regular basis;
(2) a set of standard UNIX tools for "skimming" through the virtual archive hierarchy: some of them are standard Shell commands ( cd, ls, ...) and some are either special commands for "booking" missing files, or standard commands compiled with some form of Newcastle Connection-like[BMR82] system-call interface which traps the access to missing files, kindly requesting the user to try again later: file being retrieved;

(3) a set of demons for retrieving files from remote archives using either FTP or UUCP;
(4) scripts for keeping tidy the disk space based on some local poity (filesystem size, least recently-used files, file age, frequency of usage, etc.)

3 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 tries 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
lrwxrwxrwx 1 root 8 Dec 20 18:34 nsfnet-doc.tar.Z -> mcsun:8k
% tick nsfnet-doc.tar.Z
Try again later, file(s) are being Xfer'd
% ls -1 nsfnet-doc.tar.Z
-rw-r--r-- 1 root 8192 Jan 10 10:47 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[PR85] and UUCP[Now78]. Both are "applicative" (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 spool 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).
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 directo-
ries 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 conven-
tionally composed by concatenating the hostname (or hostid) of the remote system and
the size of the real file. As an example:

1rmrwxrwx 1 root 11 Dec 28 18:44 CLX.R4.1X.tar.Z -> acsun:1058k
1rmrwxrwx 1 root 10 Dec 14 19:23 CLX.R4.3.tar.Z -> exp:1103k
1rmrwxrwx 1 root 10 Dec 14 19:23 CLX.R4.4.tar.Z -> exp:1103k
drwx-r-x 15 root 512 Dec 14 19:24 fixes/

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, based on different flavours of the ls -1R 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 prototypical 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[BMR82]" 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 enqueues a request for the
real file, and marks the symbolic link as "in transit". As an example:


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

getSFS/putSFS The command responsible for most of the day-by-day work is getSFS. 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 in-
stalled without human intervention.

4 Implementation of Sk's

The prototypical version of SFS is running at the backbone site of EUnet in Italy-which
happens to be in our department- and 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.

4.1 Modules of SFS

Each function performed by the system is managed by one different program, all of them
sharing a common configuration file. See fig. 1 for an overview of the whole system and
interdependencies among modules and data structures. They can be summarized as follows.

From our experience, with about 100 Mbytes it is possible to represent archives with
several Gigabytes of data.
4.2 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/SFSdb, lists all tuples (remote host, remote directory, local directory) which are defined for the local system. As an example:

```bash
# Mount point table for SFS
# host:remotedir localdir
#
# funic:/pub/amiga /comp/amiga
# funic:/pub/doc /doc
# funic:/pub/TeX /text/TeX
# mcuser:/ripe /ripe
# mcuser:/documents /doc
# mcuser:/comp /comp
# mcuser:/gnu /gnu
# mcuser:/graphics /graphics
# mcuser:/mail /mail
# mcuser:/misc /misc
# mcuser:/network /network
# mcuser:/programming /programming
# mcuser:/security /security
# mcuser:/windows /windows
# expo:/contrib/windows/X/contrib
# orep:/pub/REX/windows/R4
# expo:/pub/PEX/windows/X/PEX
# expo:/pub/DOCS/windows/X/DOCS
# expo:/pub/XTTEST/windows/X/XTTEST
# uunet:/rfc /doc/rfc
```

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/f on host remote, the algorithm that maps remote filenames to local filenames iteratively scans the table, looking for a key:

```bash
remote:/a
remote:/a/b ...
```

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

```bash
remote:/a/b/c /x/y
```

is found, the corresponding local pathname is /x/y/d/f, 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.

5 A protocol for remote archive management

Based on the experience gained with the prototype version of SFS, we are now designing a more ambitious protocol, aimed to the management of a distributed archive over a set of remote sites. Its mechanisms should allow:

- to minimize the memory usage, the network traffic and the retrieving time;
- to maintain the consistency of the distributed archive, with multiple instances of the same file, local caching, automatic updates.

We use techniques borrowed from cache memory theory for addressing these points.

We have devised two schemes: a "backbone-leaf" scheme, and a "cooperating sites" scheme.

We are experimenting the first, while the second one will be approached in the near future. The scenario of a backbone site-leaf site configuration is as follows:

- A service-provider host, the backbone, imports some remote archives under its SFS system, and make the whole of it visible to the outside world (making its own "is:IR" file).

- A leaf node, directly connected to the backbone by one of the available channels, builds its SFS hierarchy, importing the whole or parts of the backbone archive. The typical scheme should be one where the backbone site uses FTP links, while the leaf node can afford only a "cheap" UUCP link. A file x is retrieved at the leaf site by the following procedure:

1. the placeholder file is tick-ed at leaf;
2. a copy request is issued via UUCP from leaf to backbone;
3. UUCP copy fails because the requested file is a placeholder even at backbone;
4. a copy request is issued via FTP from remote host to backbone;
5. the real file is installed in the right place at backbone and the UUCP command is restarted;
6. finally a copy of the file is received at leaf and installed into the right place.

All the steps above but (1) are executed automatically by SFS. It must be stressed that this scheme requires a minimum of cooperation: the backbone site must enable remote access to the SFSF hierarchy and the SFS trapping mechanism must be added to the UUCP subsystem.

The other scheme requires even more cooperation: two or more sites can agree to share a common hierarchy of directories, with each subtree under control of one host. Participating hosts will exchange update messages, invalidating remote copies of modified files, and even transferring control over subtrees.
6 Security and administration

No security restrictions are coded in SFS up to now. SFS does not pose treats to the security of the local system (although the whole SFS hierarchy is "world-readable" by definition, no system critical file can be attacked through it).

As far as remote security is involved, SFS inherits the security of the communication channels it uses. Both UUCP and FTP are reasonably secure, and however, many sites are willing to provide public accessible archives in open, cooperation-oriented, environments, like EUnet and Internet are.

Administrative control of resource usage may be a more important issue. On a local basis, an administrator asks that neither the SFS disk partition is filled up, nor that the communication channels are hogged by some nasty user. The running version of SFS have provision for fulfilling these requirements, but yet no resource usage rule is enforced.

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.

7 Concluding Remarks

solution of the specific problem of archiving, since an analogous philosophy can be applied to the general issue of integrating filesystems of remote, administratively unrelated, sites [Sat90]. In order to do this, SFS can be seen as a mean for exchanging data and keeping the global system consistent with respect to update/removal of files and within the constrains of the local disk space, which could be topped with a system for integrating many, possibly overlapped, filesystems into a single structure, allowing also to mount partial "views" [SK89].

Some open questions still remain, like those about integrity and resistance to failure and synchronization [Svo84] (up to now we simply trust the underlying communication channels: FTP and/or UUCP).

References


