An Empirical Study of a Wide-Area Distributed File System

MIRJANA SPASOJEVIC
Transarc Corporation
and
M. SATYANARAYANAN
Carnegie Mellon University

The evolution of the Andrew File System (AFS) into a wide-area distributed file system has encouraged collaboration and information dissemination on a much broader scale than ever before. We examine AFS as a provider of wide-area file services to over 100 organizations around the world. We discuss usage characteristics of AFS derived from empirical measurements of the system. Our observations indicate that AFS provides robust and efficient data access in its current configuration, thus confirming its viability as a design point for wide-area distributed file systems.

Categories and Subject Descriptors: D.4.3 [**Operating Systems**]: File Systems Management—*distributed file systems*; D.4.8 [**Operating Systems**]: Performance—*measurements*

General Terms: Design, Experimentation, Measurement, Performance

Additional Key Words and Phrases: Andrew, Internet, scalability, usage, wide area, World Wide Web

1. INTRODUCTION

Over the last decade, distributed file systems such as AFS and NFS in the Unix world, and Netware and LanManager in the MS-DOS world, have risen to prominence. Today, virtually every organization with a large

This research was funded by the Advanced Research Projects Agency, under contract MDA972-90-C-0036, ARPA order number 7312. The views and conclusions expressed in this article are those of the authors and do not represent the official position of ARPA, Transarc Corporation, or Carnegie Mellon University. Mirjana Spasojevic is currently affiliated with Hewlett-Packard Labs, Palo Alto, Calif.
Authors' address: M. Spasojevic, Hewlett-Packard Labs, MS 1U13, 1501 Page Mill Road, Palo Alto, CA 94304; email: mirjana@hpl.hp.com; M. Satyanarayanan, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213; email: satya@cs.cmu.edu.

collection of personal machines uses such a system. The stunning success of the distributed-file-system paradigm is attributable to three factors.

First, a distributed file system simplifies the *separation of administrative concerns* from usage concerns. Users work on tasks directly relevant to them on their personal machines. Incidental but essential tasks such as backup, disaster recovery, and expansion of disk capacity are handled by a professional staff who focus primarily on the servers.

Second, the use of a distributed file system simplifies the *sharing of data* within a user community. Such sharing can arise in two forms: by a user accessing files from different machines and by one user accessing the files of another user. The ability to easily access one's files from any machine enhances a user's mobility within his or her organization. Although the accessing of someone else's files is not a frequent event (a fact confirmed by many previous studies [Baker et al. 1991; Ousterhout et al. 1985]), ease of access once the need arises is perceived as a major benefit by users. In other words, while sharing may be rare, the payoff of being able to share easily is very high. In this respect a distributed file system is like a telephone system: although a given individual only tends to call a tiny fraction of all telephone numbers, the latent ability to effortlessly reach any other telephone in the world is viewed as a major asset of the system.

Third, *transparency* is preserved from the users' and applications' points of view. Applications do not have to be modified to use a distributed file system. Because a distributed file system looks just like a local file system, a user does not have to learn a completely new set of commands or new methods of file usage.

The designs of modern distributed file systems reflect these observations. They use a client-server model, offer location transparency, rely on caching to exploit locality, provide fairly weak consistency semantics relative to databases, and support programming and user interfaces that are close to those of a local file system. The success and widespread usage of these systems confirm the appropriateness of these design choices.

But this success engenders a new question: "Is the distributed-file-system paradigm sustainable at very large scale?" In other words, how well can a very large distributed file system meet the goals of simplifying system administration, supporting effective sharing of data, and preserving transparency? Growth brings many problems with it [Satyanarayanan 1992]: the level of trust between users is lowered; failures tend to be more frequent; administrative coordination is more difficult; performance is degraded. Overall, mechanisms that work well at small scale tend to function less effectively as a system grows. Given these concerns, how large can a distributed file system get before it proves too unwieldy to be effective?

In this article, we provide one data point toward answering this question by reporting on the usage characteristics of AFS, the largest currently deployed instance of a distributed file system. Originally intended as a solution to the computing needs of the Carnegie Mellon University, AFS has expanded to unite about 1000 servers and 20,000 clients in 10 countries

into a single file name space. We estimate that more than 100,000 users use this system worldwide. In geographic span as well as in number of users and machines, AFS is the largest distributed file system that has ever been built and put to serious use.

Our study confirms that the distributed-file-system paradigm is indeed being effectively supported at the current scale of AFS. The measurements show that cache hit rates are above 96% and that servers are inaccessible only a few minutes per day. Even though most of data access (by volume) occurs within the boundaries of a single organization, the percentages of references to remote files is significant. Further, our data do not expose any obvious impediments to further growth of the system. While asymptotic limits to growth are inevitable, they do not appear to be just around the corner.

2. BACKGROUND

2.1 Design

The rationale, detailed design, and evolution of AFS have been well documented in previous papers [Howard et al. 1988; Morris et al. 1986; Satyanarayanan 1985; 1989; 1990; Spector and Kazar 1989]. In this section, we provide just enough detail of the current version of AFS (AFS-3) to make the rest of the article understandable.

Using a set of trusted servers, AFS presents a location-transparent Unix file name space to clients. Files and directories are cached on the local disks of clients using a consistency mechanism based on *callbacks* [Kazar 1988]. Directories are cached in their entirety, while files are cached in 64KB chunks. All updates to a file are propagated to its server upon close. Directory modifications are propagated immediately.

Backup, disk quota enforcement, and most other administrative operations in AFS operate on *volumes* [Sidebotham 1986]. A volume is a set of files and directories located on one server and forming a partial subtree of the shared name space. A typical installation has one volume per user, one or more volumes per project, and a number of volumes containing system software. The distribution of these volumes across servers is an administrative decision. Volumes that are frequently read but rarely modified (such as system binaries) may have read-only replicas at multiple servers to enhance availability and to evenly distribute server load.

AFS uses an *access list* mechanism for protection. The granularity of protection is an entire directory rather than individual files. Users may be members of *groups*, and access lists may specify rights for users and groups. Authentication relies on *Kerberos* [Steiner et al. 1988].

AFS supports multiple administrative *cells*, each with its own servers, clients, system administrators, and users. Each cell is a completely autonomous environment. But a federation of cells can cooperate in presenting users with a uniform, seamless file name space. The ability to decompose a

distributed system into cells simplifies delegation of administrative responsibility [Spector and Kazar 1989].

As originally designed, AFS was intended for a LAN. However, the RPC protocol currently used in AFS has been designed to perform well both on LANs as well as on wide-area networks. In conjunction with the cell mechanism, this has made possible shared access to a common, worldwide file system distributed over nodes in many countries.

2.2 Evolution

AFS was conceived in 1983 at Carnegie Mellon University with the goal of serving the campus community and spanning at least 5000 workstations. The design and implementation went through three major revisions: AFS-1 in 1984, AFS-2 in 1986, and AFS-3 in 1989. After 1989, responsibility for further development of AFS was transferred to Transarc Corporation. As of early 1989 there were four cells on the CMU campus with a total of 30 file servers and 1000 clients. In addition, remote cells were operating experimentally at MIT and the University of Michigan. This initial experience gave strong indications that AFS was appropriate for wide-area, shared file access.

In 1990 the Advanced Research Projects Agency (ARPA) awarded Transarc a contract to deploy and evaluate a file system to be shared by 40 to 50 Internet sites in the U.S. By mid-1991 there were 14 organizations included in the study. At the time of writing this article (November 1994) more than 130 organizations were part of this wide-area distributed file system.[1]

The wide-area nature of AFS is clearly visible from Figure 1, which shows the cells visible at the topmost level of AFS. All these directories, as well as the trees beneath them, are accessible via normal Unix file operations to any client anywhere in the system.

3. EVALUATION METHODOLOGY

Our goal in conducting this empirical study was to understand how effective various AFS mechanisms were in a large-scale, wide-area context. In this section we discuss our considerations in developing a measurement strategy, and then we briefly describe the data collection mechanism. We complete the section by characterizing the duration of our data collection and the size of the user community involved.

3.1 Considerations

One of the primary requirements in monitoring a system is minimal impact of instrumentation on the users and the system. The data collection should not significantly degrade the performance and availability of the system;

[1]There are more than 500 organizations worldwide that use AFS. However, many of these organizations do not have direct access to the Internet, and consequently their cells are not part of the wide-area file system which we discuss in this article.

cs.arizona.edu	graphics.cornell.edu	watch.mit.edu	spc.uchicago.edu
cs.brown.edu	theory.cornell.edu	ncat.edu	ucop.edu
bu.edu	kiewit.dartmouth.edu	eos.ncsu.edu	ni.umd.edu
gg.caltech.edu	northstar.dartmouth.edu	nd.edu	wam.umd.edu
cmu.edu	afsi.scri.fsu.edu	nsf-centers.edu	umich.edu
andrew.cmu.edu	iastate.edu	pitt.edu	citi.umich.edu
club.cc.cmu.edu	ucs.indiana.edu	psc.edu	math.lsa.umich.edu
ce.cmu.edu	isi.edu	huge.psc.edu	lsa.umich.edu
cheme.cmu.edu	alefnull.mit.edu	psu.edu	dmsv.med.umich.edu
cs.cmu.edu	athena.mit.edu	rose-hulman.edu	sph.umich.edu
ece.cmu.edu	rel-eng.athena.mit.edu	rpi.edu	cs.unc.edu
me.cmu.edu	media-lab.mit.edu	dsg.stanford.edu	utah.edu
sei.cmu.edu	net.mit.edu	ir.stanford.edu	cs.utah.edu
cs.cornell.edu	sipb.mit.edu	slac.stanford.edu	cs.washington.edu
msc.cornell.edu	soup.mit.edu	ece.ucdavis.edu	

(a) educational cells in the US

ads.com	ctp.se.ibm.com	transarc.com	dce.osf.org
bstars.com	locus.com	prc.unisys.com	gr.osf.org
cards.com	mtxinu.com	stars.reston.unisys.com	ri.osf.org
pub.nsa.hp.com	vfl.paramax.com	grand.central.org	syseng.osf.org
palo_alto.hpl.hp.com	stars.com	ciesin.org	
pittsburgh.ibm.com	telos.com	dementia.org	

(b) commercial cells in the US

anl.gov	nersc.gov	pppl.gov	nrlfs1.nrl.navy.mil
fnal.gov	alw.nih.gov	ssc.gov	es.net
inel.gov	ctd.ornl.gov	cmf.nrl.navy.mil	

(c) goverment cells in the US

jrc.flinders.oz.au	tu-chemnitz.de	rus.uni-stuttgart.de	rwcp.or.jp
glade.yorku.ca	uni-freiburg.de	rus-cip.uni-stuttgart.de	hepafs1.hep.net
writer.yorku.ca	urz.uni-heidelberg.de	zdvpool.uni-tuebingen.de	research.ec.org
afs.hursley.ibm.com	uni-hohenheim.de	in2p3.fr	others.chalmers.se
ethz.ch	rhrk.uni-kl.de	caspur.it	nada.kth.se
zurich.ibm.ch	geo.uni-koeln.de	pi.infn.it	bcc.ac.uk
lrz-muenchen.de	meteo.uni-koeln.de	cc.keio.ac.jp	pegasus.cranfield.ac.uk
ipp-garching.mpg.de	rrz.uni-koeln.de	sfc.keio.ac.jp	athena.ox.ac.uk
mpa-garching.mpg.de	ihf.uni-stuttgart.de	titech.ac.jp	ibm.uk
hrzone.th-darmstadt.de	mathematik.uni-stuttgart.de	etl.go.jp	

(d) cells outside the US

Fig. 1. A snapshot of cells visible from a typical AFS client. This figure shows the cells visible from a typical client in the system on November 1, 1994. The listing above was obtained by doing an "ls /afs" and then sorting the output according to the domain. As the figure shows, there were 59 educational cells, 22 commercial cells, 11 governmental cells, and 39 cells outside the United States at the time of the snapshot.

otherwise users will alter their behavior to circumvent these deficiencies. The scale of the system and our desire to study it over a long period of time complicate the logistics of data collection considerably. It is practically infeasible to require extensive cooperation of users or system administrators at many different cells to assist in the data collection. Hence we decided that our instrumentation should not require any regular administrative effort by the sites being monitored. But, the system administrator of

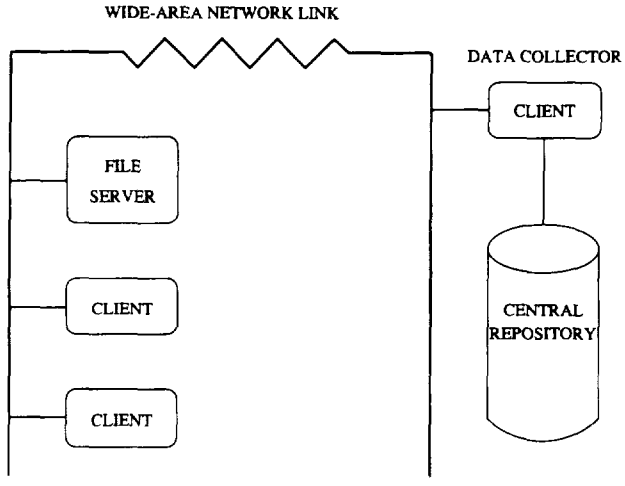


Fig. 2. Instrumentation for data collection.

a cell could turn off data gathering if that cell did not wish to participate in the study.

Instrumentation of a system like AFS requires advance planning. The instrumentation has to follow the standard software release cycle. For a commercial system like AFS, there is usually a period of at least six months between two successive versions. Consequently, there is a considerable time lag in seeing the effect of changes to the data collection tools.

A significant factor in our design was the need to balance the level of detail of event recording with providing adequate privacy and limiting data volume. Hence, many of our statistics report on aggregate behavior of the system rather than, for example, per-file operations.

3.2 Data Collection Mechanism

Our measurements were performed via the *xstat* data collection facility [Transarc 1991]. The AFS code was instrumented to allow collection of extended statistics concerning the operation of servers and clients. These statistics could be obtained remotely via an RPC call from any AFS client or server at any time.

A central data collection machine, located at Transarc, polled and obtained data from each participating machine once a day. The collected data was formatted and inserted into a relational database for postprocessing. The database provided a scalable tool for storing and manipulating large quantities of data. Figure 2 shows the structure of our data collection mechanism.

Not requiring the active cooperation of remote cells complicated the process of discovering which clients and servers should be contacted for data collection. Our solution to this problem was to run a discovery process once every few weeks. This process queried the Domain Name Service at

each cell to obtain a list of registered IP addresses. This list was then probed to discover new AFS clients and servers in that cell.

Our evaluation of AFS was based on data from clients and servers, collected over two 12-week periods. We supplemented it by circulating a questionnaire on various aspects of AFS to a sample of users and compared their responses to the measured data. This corroboration with anecdotal information served as a sanity check on our measurements.

One's confidence in the answers of an evaluation can be classified into four levels based on the origin of the information: *intrinsic* (direct examination of the system design), *empirical* (raw measurements), *evidentiary* (inferences based on raw data), and *anecdotal* (information requiring user judgment). In this taxonomy, our primary information is empirical and evidentiary while our secondary information is anecdotal.

3.3 Coverage

The first round of measurements was conducted during a 12-week data collection period from mid-May to mid-August, 1993. Our data spanned 50 file servers and 300 clients from 12 cells in 7 states. Preliminary results from this collection were presented in an earlier paper [Spasojevic and Satyanarayanan 1994].

The data collection code was refined and extended based on the insights gained in this preliminary study. The second round of measurements widened coverage to 70 file servers and over 900 clients from 14 cells. These measurements were conducted during another 12-week data collection period, from mid-October, 1993, to mid-January, 1994. In this article we present the results from the second set of measurements. We point out differences, when appropriate, between the preliminary and current observations.

4. OBSERVATIONS AND ANALYSIS

In this section we present and analyze data collected during the study. We begin by examining storage capacity and volume activity. We then discuss the nature of client-server interaction, including RPC traffic and bulk data transfers. Next, we explore cache performance and availability, two key parameters of any distributed system. Finally, we examine the extent to which AFS is used for collaboration and information dissemination.

4.1 AFS Data Profile

How much data does the wide-area file system contain? Figure 3 shows a snapshot of the data stored at 10 cells. These cells contained 70 file servers, housing about 32,000 volumes and constituting over 200GB of data. The data shows that although 52% of the volumes belong to individual users, they contain only 19% (41GB) of the data. The default quota on volumes in AFS is 5MB, which might explain why so many user volumes are rather small. Approximately 30% of the data (65GB) belongs to backup volumes. Only 2% of the volumes are read-only replicas, and they contain only 4% of the data. The remaining 25% of the volumes correspond to system binaries

Volume type	Total number	Total size (GB)	Average size (MB/vol)
User	16,622 (52%)	41 (19%)	2.5
Backup	6,654 (21%)	65 (30%)	9.8
Readonly	648 (2%)	9 (4%)	13.4
Other	8,171 (25%)	103 (47%)	12.6
ALL	32,095 (100%)	217 (100%)	6.8

Fig. 3. Storage capacities of 10 cells. This table is based on the volumes housed by 70 servers at 10 sites. The volume types were deduced from the volume names.

and data, bulletin boards, and other miscellaneous data. Together, these volumes contain almost half of the total data.

Extrapolating from this evidence, we estimate that the whole wide-area file system contains more than 400,000 volumes with 3–4TB of data. It is interesting to note that although the average volume size is only 6.8MB, the raw data indicate that some volumes contain more than 1.5GB of data. In other words, volumes span a wide range of sizes, but tend to be skewed toward the low end (Figure 4).

A related but distinct question pertains to how many of these volumes are in active use every day. To answer this question, we recorded the number of volumes that registered at least one read or write operation during the day. We also recorded sizes of these volumes. Our measurements indicate that every day approximately 40% of the volumes register at least one read or write operation. Those volumes comprise approximately 50% of the data (Figure 5).

We also tracked individual volume access patterns. Based on the observed patterns, we computed a *hazard rate* which represents the probability that a volume will be referenced on a particular day, assuming that the last reference was x days ago (Figure 6). Our data show a very high hazard rate of 80% on day 1. In other words, if a volume was accessed yesterday, there is 80% chance that it will be accessed again today. The hazard rate drops sharply to around 40% on day 2 and continues to slowly decrease thereafter. This indicates the existence of a set of "hot" volumes which are accessed daily.

Earlier studies of file lifetimes and access patterns by Smith [1981a; 1981b] have found that for a file migration algorithm it is useful to use the time since last reference, the file size, the type of file, and the file age. Direct comparison of the results of those studies with our observations of volume access patterns is not entirely appropriate, because of the substantial difference in data granularity—our measurements are per-volume whereas the earlier measurements are per-file. However, we tested the hypothesis that "time since last reference" is an important factor for a volume migration strategy. We built a simulation model in which an entire volume is migrated, if it has not been referenced for at least three days. Our simulation shows that more than 90% of requests are satisfied by the online volumes. This leads us to believe that Smith's work on file migration may in fact be applicable to volume migration also.

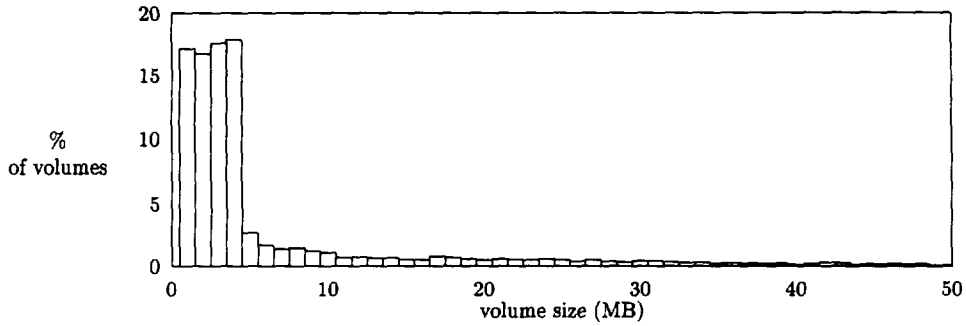
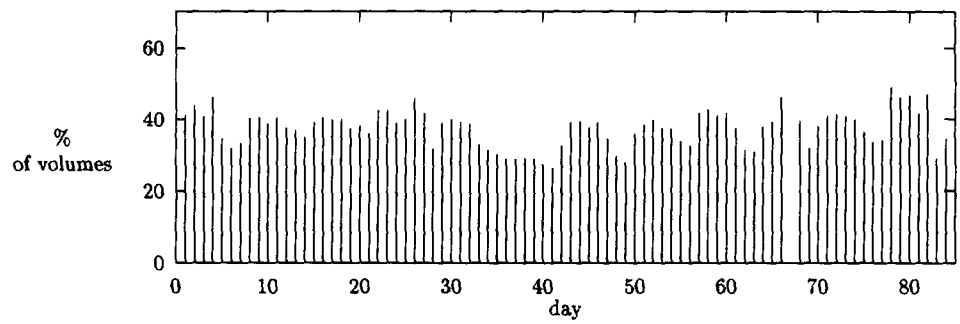
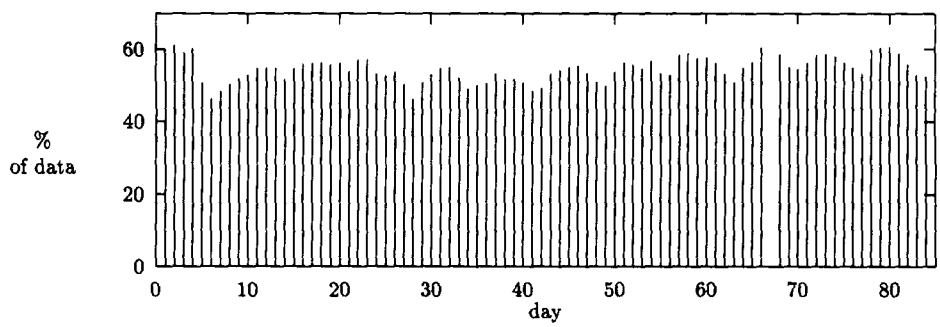


Fig. 4. Volume size distributions. This graph is based on the same data as in Figure 3. It shows the distribution of volume sizes for volumes up to 50MB.



(a) daily accessed volumes



(b) size of daily accessed volumes

Fig. 5. Volume activity on a daily basis. The graph in part (a) shows the percentage of volumes that were accessed on a particular day during the 12-week data collection period. The graph in part (b) shows the percentage of active-volume sizes with respect to the total data.

A file migration system based on AFS has been built and used at the Pittsburgh Supercomputing Center [Goldick et al. 1995]. Unfortunately, we

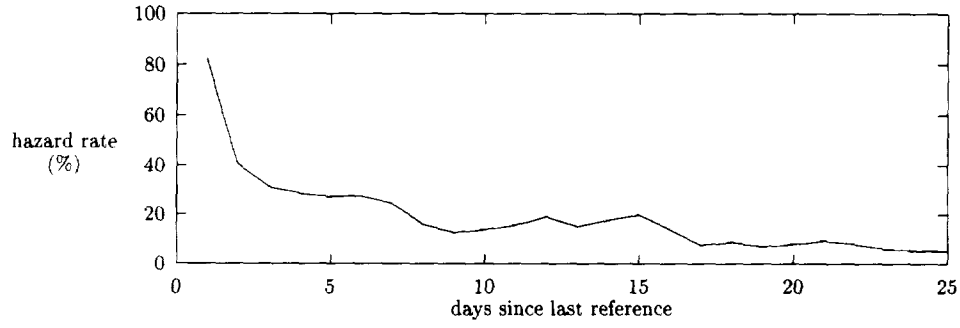


Fig. 6. Volume hazard rates. The inputs for this graph were the observed lengths of intervals between two consecutive access operations for active volumes. The length of intervals is expressed in whole days. The minimum interval is one day for a volume that has been accessed on two consecutive days. Based on the distribution of these intervals, the hazard rate represents the probability that a volume will be accessed today when the last access was x days ago.

could not validate our simulation using empirical data from this system because it migrates individual files rather than entire volumes.

4.2 Client-Server Interaction Profile

How do AFS clients and servers interact? The answer to this question is important because knowledge of the relative distribution of file system RPC calls helps characterize a normal system and identifies the commonest calls. This, in turn, allows performance tuning to be focused. Figure 7 lists the client-server RPC calls with short descriptions.

Both servers and clients have been instrumented to record the information regarding these calls. They keep statistics about the total number of calls, the number of successful calls, and the average time of execution of successful calls (with the standard deviation). During our study, statistics were collected from 50 file servers and 900 clients on a typical day.

4.2.1 *RPC Calls Observed by Servers.* Over 680 million calls were observed during the data collection period. About 93% of these were successful. Figure 8 summarizes the detailed statistics of calls accounting for at least 1% of the total calls.

The most frequent call is Fetch_Status. We conjecture that many of these calls are generated by users listing directories in parts of the file name space that they do not have cached. The significant number of unsuccessful calls (9.2%) suggests that these directories belong to protected areas of the file name space. It is interesting to note that despite caching, the number of Fetch_Data calls is considerably higher than the number of Store_Data calls.

The average times of execution of RPC calls account only for the time servers spend in servicing these calls and do not include network latency. Different software and hardware server configurations, as well as variations in server loads, result in significant variance of RPC execution times.

Fetch_Data	Returns data of the specified file or directory and places a callback on it.
Fetch_ACL	Returns the content of the specified file's or directory's access control list.
Fetch_Status	Returns the status of the specified file or directory and places a callback on it.
Store_Data	Stores data of the specified file or directory and updates the callback.
Store_ACL	Stores the content of the specified file's or directory's access control list.
Store_Status	Stores the status of the specified file or directory and updates the callback.
Remove_File	Deletes the specified file.
Create_File	Creates a new file and places a callback on it.
Rename	Changes the name of a file or directory.
Symlink	Creates a symbolic link to a file or directory.
Link	Creates a hard link to a file.
Make_Dir	Creates a new directory.
Remove_Dir	Deletes the specified directory which must be empty.
Set_Lock	Locks the specified file or directory.
Extend_Lock	Extends a lock on the specified file or directory.
Release_Lock	Unlocks the specified file or directory.
GiveUp_Call	Specifies a file that a cache manager has flushed from its cache.
Get_Vol_Info	Returns the name(s) of servers that store the specified volume.
Get_Vol_Status	Returns the status information about the specified volume.
Set_Vol_Status	Modifies status information on the specified volume.
Get_Time	Synchronizes the workstation clock and checks if servers are alive.
Bulk_Status	Same as Fetch_Status but for a list of files or directories.

Fig. 7. Client-server RPC calls.

RPC call	%	# of calls	(% err.)	Avg *ms*	(s.d.)
1. Fetch_Data	8.4	57,399,455	(0.4)	240	(878)
2. Fetch_Status	73.5	500,176,483	(9.2)	20	(426)
3. Store_Data	3.5	23,664,981	(0.9)	113	(398)
4. Store_Status	4.6	31,595,893	(0.5)	3	(110)
5. Remove_File	1.3	9,130,978	(0.9)	52	(218)
6. Create_File	1.5	10,274,032	(17.8)	32	(389)
7. GiveUp_Call	1.4	9,408,889	(0.0)	1	(40)
8. Get_Time	3.9	26,376,447	(0.0)	6	(198)
ALL	100.0	680,284,501	(7.14)	n/a	

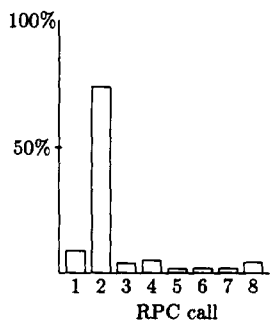


Fig. 8. Average distribution of RPC calls observed by servers. This figure presents statistics for RPC calls that account for at least 1% of the total calls. The numbers presented correspond to the percentage of a given call in the total number of RPC calls, the number of calls (with the percentage of unsuccessful calls), and the average time of execution of successful calls (with the standard deviation). The average times of execution account only for the time servers spend in servicing these calls. The results were averaged over all servers and all data collected during the 12-week data collection period. The graph on the right side is a graphical representation of the second column in the table.

Both Fetch_Data and Store_Data calls take considerably longer than other operations. This is to be expected, since they involve disk I/O. The GiveUP _Call is the call that takes the least amount of time on average. It involves only updating the server's internal callback structure.

Although Fetch_ACL call is not shown in Figure 8, our raw data showed that it takes considerably more time on average than Fetch_Status call

	Percentage of Calls							
week	Fetch_D	Fetch_S	Store_D	Store_S	Remove_F	Create_F	GiveUp_C	Get_T
1	9.0	77.3	3.2	1.5	0.9	1.3	1.3	4.0
2	7.6	79.7	2.9	1.5	1.1	1.2	1.5	2.8
3	8.3	78.7	2.8	1.6	1.0	1.2	1.4	3.4
4	8.7	76.6	3.3	1.8	1.1	1.4	1.3	4.0
5	8.9	76.5	3.4	1.9	1.0	1.4	1.2	4.1
6	9.8	72.9	3.9	2.1	1.6	1.8	1.5	4.5
7	7.3	76.2	3.6	1.9	1.2	1.5	1.4	5.3
8	9.4	67.7	4.6	6.9	1.7	2.0	1.1	4.6
9	10.0	74.8	3.3	1.9	1.0	1.3	1.1	4.9
10	7.0	75.2	3.0	5.0	1.3	1.3	2.0	3.4
11	6.0	62.7	3.5	17.2	2.1	1.7	1.5	3.0
12	8.5	60.7	4.4	14.7	2.4	2.1	1.3	3.4
all	8.4%	73.5%	3.5%	4.6%	1.3%	1.5%	1.4%	3.9%

Fig. 9. Weekly RPC call distributions observed by servers. This table is based on the same raw data as the table in Figure 8. It indicates weekly averages (in percentages), rather than averaging across all weeks.

(140ms vs. 20ms). This surprised us, since Fetch_Status returns access list information. This apparent anomaly was explained when inspection of the AFS code showed that the implementation of Fetch_ACL contains a call to a protection server, while the implementation of Fetch_Status does not. The same is true for the Store_ACL and Store_Status pair of calls.

Analysis of RPC calls on a weekly basis confirms that their distribution is stable over time. Figure 9 presents data that show only two significant deviations from the general profile shown in Figure 8. The anomalies are the very high number of Store_Status calls during weeks 11 and 12.

We discovered that more than 90% of Store_Status calls during these two weeks were concentrated on one file server at Transarc. The high number of Store_Status calls on this file server was also accompanied by a higher-than-average number of Store_Data calls. Further investigation revealed that this server contains many volumes occasionally updated with new software, thus explaining the unusual distribution of calls.

With this data, one can loosely characterize a normally running system as one with a very high number (above 60%) of Fetch_Status calls, and smaller, but still significant, number of Fetch_Data calls (about 8%). Other frequent calls in such a system include Store_Data, Store_Status, and Get_Time.

4.2.2 *RPC Calls Generated by Clients.* The set of machines from which we were collecting data did not represent a "closed system," i.e., there was no guarantee that participating servers and clients were contacting only each other. Thus, the number of calls observed by file servers does not match the number of calls generated by clients. Nevertheless, it is interesting to compare these two profiles. Figures 10 and 11 summarize the data collected from clients.

There were over 570 million calls, out of which 96.2% were successful. Again, Fetch_Status calls dominate. But the relative percentage of these

RPC call	%	# of calls	(% err.)	Avg *ms*	(s.d.)
1. Fetch_Data	14.9	85,680,312	(0.2)	371	(1128)
2. Fetch_Status	64.2	370,183,097	(5.0)	133	(1159)
3. Store_Data	4.6	26,396,225	(0.7)	251	(1134)
4. Store_Status	1.9	11,093,117	(0.3)	112	(1029)
5. Remove_File	1.3	7,613,467	(0.2)	149	(728)
6. Create_File	1.7	9,870,496	(16.3)	157	(837)
7. GiveUp_Call	1.8	10,320,046	(0.0)	160	(814)
8. Get_Vol_Status	2.1	12,307,880	(0.7)	197	(795)
9. Get_Time	5.1	29,189,958	(3.1)	97	(1194)
ALL	100.0	576,266,313	(3.8)	n/a	

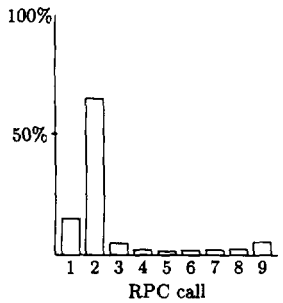


Fig. 10. Average distribution of RPC calls generated by clients. This figure presents statistics for RPC calls that account for at least 1% of the total calls. The numbers presented correspond to the percentage of a given call in the total number of RPC calls, the number of calls (with the percentage of unsuccessful calls), and the average time of execution of successful calls (with the standard deviation). The results were averaged over all servers and all data collected during the 12-week data collection period. The graph on the right side is a graphical representation of the second column in the table.

	Percentage of Calls								
week	Fetch_D	Fetch_S	Store_D	Store_S	Remove_F	Create_F	GiveUp_C	Get_V_S	Get_T
1	15.7	62.4	5.1	2.2	1.2	1.7	1.7	2.1	5.5
2	15.7	62.4	5.0	2.2	1.5	1.8	1.9	2.1	4.8
3	14.4	66.1	4.3	1.9	1.1	1.6	1.7	2.1	4.7
4	15.5	62.8	5.1	2.0	1.5	1.8	1.6	2.1	5.0
5	15.5	63.3	4.8	2.3	1.2	1.7	1.7	2.0	5.1
6	15.3	63.6	4.8	1.8	1.3	1.8	1.8	2.3	5.0
7	13.1	63.0	5.0	1.9	2.0	2.1	1.9	2.7	5.8
8	15.0	63.7	4.9	2.1	1.4	1.9	1.8	2.3	4.8
9	14.1	66.9	4.4	1.8	1.2	1.5	1.6	1.7	5.0
10	14.0	64.8	3.8	1.7	1.5	1.6	1.9	2.3	5.3
11	14.5	66.2	3.5	1.4	1.0	1.4	2.3	2.3	5.0
12	15.0	65.7	3.8	1.7	1.1	1.6	1.7	1.9	5.1
all	14.9%	64.2%	4.6%	1.9%	1.3%	1.7%	1.8%	2.1%	5.1%

Fig. 11. Weekly RPC call distributions observed by clients. This table is based on the same raw data as the table in Figure 10. It indicates weekly averages (in percentages), rather than averaging across all weeks.

calls was slightly lower than that reported in Figure 8 for servers. At the same time, the relative percentage of Fetch_Data calls was significantly higher.

The other significant difference between the two profiles is that in the client's case Get_Vol_Status call represents more than 2% of the total calls. We found that this anomaly is not typical and that it was confined to clients in only one cell.

The observed average times of execution of RPC calls account for both the time servers spend in servicing these calls and the network latency. Different software and hardware configurations of both servers and clients, as well as variations in loads and network congestion, result in significant variance of RPC execution times. However, on average, execution of an RPC call took about 100ms longer on a client than on a server.

Data chunk size	Servers		Clients	
	Fetched (%)	Stored (%)	Fetched (%)	Stored (%)
0 B - 128 B	28	35	20	16
128 B - 1 KB	5	11	6	15
1 KB - 8 KB	34	18	34	26
8 KB - 16 KB	5	6	6	8
16 KB - 32 KB	3	4	4	8
32 KB - 128 KB	24	24	29	27
over 128 KB	2	1	1	1
Daily per machine	390 MB	143 MB	33 MB	7 MB

Fig. 12. File transfer size distribution. This table represents distribution of transferred data chunk sizes as observed by servers and clients over the 12-week data collection period.

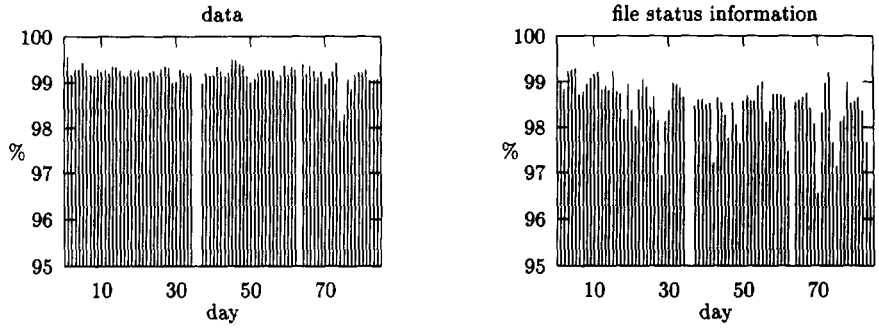
4.2.3 *Causes of RPC Failures.* As noted in the previous section, nearly 3.8% of the calls generated by clients failed. We were curious about the nature of these failures, since they may have been symptomatic of underlying performance or reliability problems. To study this, we instrumented AFS clients to keep track of failed RPC calls. Errors were divided into several categories: server problems, network problems, protection problems (insufficient authorization or expired authorization tickets), volume problems, occurrences of a busy volume (e.g., when a volume is moved to another server), and errors of unknown cause.

Our data showed that the majority of failed calls, 89%, were Fetch_Status calls. Most of them, 93.8%, failed because of protection errors. We hypothesize that there are periodic jobs on some machines that attempt to traverse the AFS tree and fail when they encounter a protected part of the tree. Another plausible explanation is continuous execution of some background daemons (e.g., xbiff) which always produce a failed call after the authorization ticket's expiration. A significant number of unsuccessful calls, 10%, failed for unknown reasons.

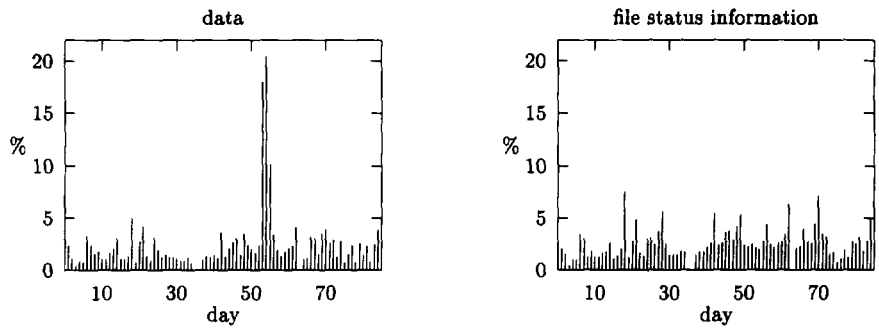
4.2.4 *Bulk-Transfer Profile.* Statistics concerning file transfers were recorded by both file servers and clients. AFS performs partial file caching, so the numbers reported here show transfers on a per-chunk basis, rather than on a per-file basis. The exceptions are directories which are cached in their entirety. Chunk size is 64KB by default, but may be changed on a per-client basis.

The collected statistics are summarized in Figure 12. Our data indicate that the most frequently fetched chunks are in the range 1–8KB. These correspond to entire files or directories. This result is consistent with many earlier studies of file size distributions which have reported small average file size [Ousterhout et al. 1985; Satyanarayanan 1981].

The distribution of data transfers on file servers and clients is roughly similar. The results from Section 4.1 indicate that the amount of data housed by active volumes is about 1.5GB per file server. Figure 12 shows that only about 25% of this data (390MB) is actually fetched by clients. Compared to the preliminary set of measurements, the results of the current round show much higher data traffic rates.



(a) combined cache hit ratio for native and foreign file references



(b) fraction of references to files in foreign cells

Fig. 13. Cache performance and reference mixes. This figure shows the observed cache hit ratios and relative proportion of native-cell and foreign-cell references over the data collection period. The gaps in histograms on several days correspond to missing data due to problems with the data collection machine.

4.3 Performance and Reliability

4.3.1 *Cache Performance.* Cache hit ratio is a critical factor in determining the overall performance of a system like AFS. Caching is especially valuable in masking the long latencies typical of wide-area networks. To study this aspect of AFS, we instrumented clients to keep statistics on cache hit rates and on the percentages of references made to native and foreign cells (Figure 13). Since the AFS file cache is split into a cache for data and a cache for status information, our statistics were kept separately for these two categories.

Our data indicate that the average cache hit ratio is over 98% for data and over 96% for status information. The percentage of references to files in foreign cells was around 5% for data and 8% for status information. The exceptions occurred on several days when the recorded percentages to foreign data were as high as 20%. This anomaly turned out to be a result of

Type of outage	Fraction of time	Time (min/day)
Servers in the same cell	0.07 - 0.82%	1.0 - 11.9
Servers in foreign cells	0.02 - 0.72%	0.3 - 10.4

Fig. 14. Average inconvenience time for clients. This table shows observed average inconvenience times for clients over the 12-week data collection period. The lower side of the range represents the case when for each client all daily failures occur simultaneously. The higher side of the range represents the case when daily failures do not overlap.

just one client accessing an unusually high number of remote files. When this client was excluded from the study, the percentage of references to foreign data dropped to less than 5% on these days. We statistically analyzed the possibility of foreign-cell references causing much lower cache hit ratios. Our analysis indicated that there was no such correlation.

4.3.2 *Frequency of File Server Failures.* Interruption of file service in a wide-area file system is a potential obstacle to providing transparency. One way of measuring file server downtimes is to have file servers record downtimes themselves and report them to the data collection agents. However, in our view, a much more important picture is the one that client machines have about the file servers' availability. Thus, we instrumented clients to record outages. A particular file server's downtime was observed only by the clients that could not access particular data from that file server (because of the server's failure and/or network problems). Such an approach weights failures by clients' interest in the files affected; in other words, the inaccessibility of a heavily used file contributes more to the metric than the inaccessibility of a lightly used file. Figure 14 reports average *inconvenience* time, which is the time during which a client cannot communicate with at least one file server that it needs to access.

Downtime incident statistics were collected from 800 clients on an average day. During the 12-week data collection period, the number of observed server downtime incidents was 20,763 for servers in the same cell and 2929 for servers in foreign cells. (Figure 15). It should be noted that a particular server's outage can be reported multiple times if observed by multiple clients. Also, on an average day only about 30% of the clients accessed data in foreign cells and thus were able to observe server downtimes in foreign cells. According to the numbers collected, on average, a client observes a server outage every three days for the local cell and every 12 days for the foreign cell, under the assumption that all clients are equally observant (active). The duration of more than half the outages is less than 10 minutes. Since this is shorter than the recovery time for a typical server, we conjecture that many of these short outages are really due to transient network failures.

4.4 Sharing in AFS

The existence of cross-cell file access in AFS is borne out by the data presented in Figure 13(b). That figure showed that the percentage of

Downtime durations	Servers in the same cell		Servers in foreign cells	
0 min - 10 min	12,021	(58.0%)	1,944	(66.4%)
10 min - 30 min	6,013	(29.0%)	503	(17.2%)
30 min - 1 hr	1,317	(6.4%)	114	(3.9%)
1 hr - 2 hr	563	(2.7%)	81	(2.8%)
2 hr - 4 hr	212	(1.0%)	118	(4.0%)
4 hr - 8 hr	288	(1.4%)	66	(2.2%)
> 8 hr	349	(1.7%)	103	(3.5%)
TOTAL	20,763	(100.0%)	2,929	(100.0%)

Fig. 15. Distribution of file server outage durations. This table shows durations of file server failures as observed by clients over the 12-week collection period.

references to the files in foreign cells was up to 5% for data and up to 8% for status information during the 12-week data collection period. Although 5% may not seem like much, it is significant because cells represent organizational boundaries, and most users tend to access data within their own organizations.

Figure 16 represents a histogram of the number of different cells contacted by each client during the 12-week period. The table shows that nearly 90% of the clients referenced data in at least one foreign cell while 6% of the clients referenced data in all available cells. Further, examination of the raw data shows that, on average, 30% of the clients referenced foreign data each day.

We also repeated the study originally reported by Kistler and Satyanarayanan [1992] on the extent of sequential write sharing on directories and files. Every time a user modified an AFS directory or file, the user's identity was compared to that of the user who made the previous modification. Our data, showing that 99.1% of all directory modifications were by the previous writer, are consistent with Kistler and Satyanarayanan's observations. Unfortunately, we are not able to report on write sharing on files due to a bug in the data collection tools.

5. CORROBORATION WITH ANECDOTAL EVIDENCE

To complement the quantitative data obtained by instrumentation, we constructed a questionnaire that touched upon a diverse set of issues. The purpose of the questionnaire was to elicit user perceptions as well as to obtain a profile of AFS usage. The topics of interest to us included characterization of the user community, extent of usage of native and foreign cells, and degree of collaboration within and across cells. We were also interested in obtaining user perceptions of performance and reliability of AFS for accessing native and foreign cells. Finally, we were interested in the value and adequacy of various AFS mechanisms such as access control lists, read-only replication, and data mobility.

The questionnaire was distributed in two ways: first, by posting on several Netnews bboards; second, by direct mailing to AFS contacts in different cells. We received about 100 responses from 50 cells. A detailed

Cells contacted	% of clients
≥ 1	100
≥ 2	89
≥ 3	78
≥ 4	64
≥ 5	29
≥ 10	22
≥ 20	13
≥ 50	7
≥ 70	6

Fig. 16. Client contacts with cells. This table shows the percentage of clients that contacted a given number of cells during the 12-week data collection period.

discussion on the results of this questionnaire is presented in our preliminary study [Spasojevic and Satyanarayanan 1994]. Here we summarize the most important findings.

The AFS user community consists of a number of academic, government, and commercial sites. Consequently, AFS users tend to have a very diverse background. However, responses to our questionnaire came from a technically sophisticated group of respondents. Most of them (81%) are serious programmers, and two-thirds of them rate their knowledge of AFS to be at an advanced or expert level. Close to 95% had experience with other distributed file systems, usually NFS. This renders their assessments of AFS quality more credible, but also leaves unanswered the question of how naive users view AFS.

The majority of users are satisfied with AFS performance when accessing both local and remote data. Compared to other distributed systems they have used, 73% of respondents feel that AFS provides comparable or better performance. However, nearly two-thirds of them also rate performance and reliability as aspects of AFS that have sometimes been unsatisfactory and that should be further improved.

Users tend to notice file server failures less frequently than what the empirical evidence indicates. Most of the respondents do not experience failures more than once a month. However, users perceive failures as lasting on average 30 minutes, which is much longer than an average server outage according to the empirical data in Section 4.3.2. This suggests that users notice only long-lasting failures which significantly affect their work.

Users confirm that the wide-area aspects of AFS are indeed valuable. In comparison to other collaboration tools (phone calls, surface mail, electronic mail, fax, bboards, FTP, other file systems) AFS was rated as the second-best tool, only after electronic mail. In their local cell, over 60% of the users tend to read or modify files that do not belong to them.

The extent of the cross-cell traffic (Section 4.3.1) is corroborated by reports of users accessing materials in other cells. About 80% of respondents possess accounts/authentication identities in foreign cells, and 38% of them participate in joint projects with people from different cells.

6. RELATIONSHIP BETWEEN AFS AND THE WEB

Over the last several years, the World Wide Web [Berners-Lee et al. 1994] has emerged as the dominant wide-area information service on the Internet. Although other examples of such services exist (such as Archie [Emtage and Deutsch 1992], Gopher [McCahill 1992], and, of course, AFS), none has demonstrated the popularity or growth rate of the Web. Since this growth shows no signs of leveling, it is fair to ask whether AFS will soon be rendered obsolete.

Our answer, as elaborated in the following sections, is that the Web and AFS have sufficiently different design characteristics that they are not really in competition. Rather, they represent complementary technologies that can be used in conjunction very effectively.

6.1 Comparison of Characteristics

6.1.1 *Interface Level.* AFS supports an application programming interface (API) and is primarily intended for use by programs. Consistent with the Unix heritage of AFS, the user interface is only of secondary importance. In contrast, the Web interface is primarily a graphical user interface (GUI) for humans. The ease of use of the Web by novice users has, in fact, been a major reason for its success. While it is possible to write programs that parse and interpret Web pages, this is not the intended use of the Web and is unlikely to be efficient.

6.1.2 *Targeted Access versus Browsing.* AFS assumes that the path names of objects to be accessed are known with accuracy. The only aids to search are the mnemonic significance of path names and the directory organization. Consistent with the Unix file system model, AFS directories offer no auxiliary annotation to help in browsing. While AFS' location transparency hides server names, its hierarchical name space inevitably reflects some aspects of administrative and organizational structure.

In contrast, a Web page is assumed to merely be the starting point of an exploration. The ability to mix annotational information with pointers to other Web pages greatly simplifies *browsing*. These pointers are often to pages in different administrative and organizational domains. Thus, a Web URL can be viewed as a *fuzzy pointer*, whose dereferencing can result in widely varying targets, depending on context.

6.1.3 *Workload Characteristics.* The different usage models of AFS and the Web result in substantial differences in client and server workload characteristics in the two systems. As shown earlier in this article, AFS references exhibit substantial temporal locality. This makes caching at clients useful. In contrast, Web references from a client tend to exhibit poor locality; this renders conventional client caching futile [Dharap and Bowman 1995].

On the other hand, there is substantial *collective* temporal locality in the Web references of an organization. In other words, Web documents that are accessed by one user in an organization are likely to also be accessed by

other users in that organization. Hence a shared cache at an intermediate node can be valuable. In contrast, intermediate caches have been shown to be of little value in AFS [Muntz and Honeyman 1992].

Finally, AFS allows remote objects to be updated by clients. Hence AFS server workloads include both read and write traffic. In contrast, Web pages are *read-only* objects. While the use of *forms* allows users to enter new data, the modicum of updatability this provides hardly compares with the full-fledged support for updates in AFS.

6.1.4 *Scalability.* Concern for scalability has pervaded the design of AFS from its very beginning [Satyanarayanan et al. 1985]. The evolution of AFS has consistently paid attention to the impact of large scale on performance, security, and system administration [Satyanarayanan 1992]. This concern is apparent in many aspects of the design of AFS:

—the use of callback-based caching [Howard et al. 1988] to minimize server and network load while offering a close approximation to the Unix consistency model,

—the use of *volumes* for ease of system administration [Sidebotham 1986],

—careful attention to security, including the use of access control lists, the ability to define groups of users, end-to-end authentication, and a model of limited trust [Satyanarayanan 1989], and

—the decomposition of the system into independent *cells* to support organizational autonomy.

In contrast, scalability has been an afterthought in the evolution of the Web. Only in the light of the Web's popularity has attention been focused on issues such as authentication, access control, and reduction of network and server load. Unfortunately, the need to be backward compatible with existing Web browsers and servers makes this a difficult task. It remains to be seen how successful attempts to retrofit scalability into the Web will be.

6.1.5 *Heterogeneity.* AFS is closely tied to the Unix file system model. Only late in its evolution were efforts made to provide cross-platform support. In contrast, the Web is platform neutral, and browsers became available for most platforms early in its evolution. The ability to access the Web from virtually any client played a significant role in its rise to prominence.

6.2 Combining AFS with the Web

The complementary design characteristics of the Web and AFS suggest the possibility of combining their strengths in a composite system. Indeed, a number of such systems are in use today.

For example, a recent paper [Katz et al. 1994] describes the use of AFS as the shared back end for a collection of Web servers at the National Center for Supercomputing Applications (NCSA). Combined with a customized DNS name resolver, the use of AFS enables the collection of Web servers to masquerade as a single, large, extensible, virtual Web server. The paper

reports that both performance and availability of the Web site have improved as a direct result of this approach.

Transarc and Carnegie Mellon University are two other examples of organizations that use AFS to store Web data. In both cases, a small number of Web servers provide access to Web data stored in AFS. Users create and update Web documents from any AFS client. Since these clients do not run Web servers, their identity is not part of the URLs. This preserves the ability to add or delete AFS clients and servers with minimal administrative overhead.

At Transarc, Web clients have been modified to determine if a URL refers to a document in AFS. If so, it is accessed directly through AFS. For example, the URL

http://www.transarc.com/afs/transarc.com/public/www/index.html

is effectively translated into the URL

file:/afs/transarc.com/public/www/index.html

by a Web client [Spasojevic et al. 1994].

The setup at Carnegie Mellon pays additional attention to issues of security. Web servers are authenticated to AFS as special users and can only access files in those directories with appropriate ACLs. This gives users fine-grain control over which AFS documents are visible via the Web. The default protection is such that Web access is prohibited.

7. CONCLUSION

Our goals in conducting this study were to observe wide-area AFS usage and to characterize its profile. We were also interested in determining how well AFS worked at the current scale of the system and to see if any imminent limits to its further growth were apparent.

The qualitative and quantitative data that we have presented confirm that AFS provides robust and efficient distributed file access in its present worldwide configuration. The caching mechanism is able to satisfy most of the file references from the clients' local caches. Even though file server and network outages can be disruptive for particular users, our observations show that prolonged server inaccessibility is rare. Our data show no obvious bottlenecks that might preclude further growth of the system.

AFS' divide-and-conquer technique of using semiautonomous cells for spanning widely disparate organizations has proven to be invaluable. By providing considerable flexibility in security and storage management policies, the cell mechanism reduces the psychological barrier to entry of new organizations. As a consequence, growth in AFS over time has not just been in the number of nodes in each cell, but also in the total number of cells.

In summary, this article provides conclusive evidence that AFS is a viable design point in the space of wide-area distributed file system designs. We are convinced that any alternative design must preserve the

aggressive caching policies, careful attention to security, and support for autonomous administration that are the hallmarks of AFS' approach. The absence of any of these features will be fatal in an attempt to build a file system that uses a wide-area network and spans many organizations.

ACKNOWLEDGMENTS

The *xstat* data collection facility was designed and implemented by Ed Zayas. Contributions to the evaluation methodology for wide-area distributed file systems were made by Bob Sidebotham, Alfred Spector, and Ed Zayas. Anne Jane Gray provided assistance in organizing this project. Comments by Maria Ebling, Mike Kazar, Jay Kistler, Qi Lu, Lily Mummert, and the anonymous referees were most helpful in improving the presentation.

REFERENCES

BAKER, M. G., HARTMAN, J. H., KUPFER, M. D., SHIRRIFF, K. W., AND OUSTERHOUT, J. K. 1991. Measurements of a distributed file system. In *Proceedings of the 13th ACM Symposium on Operating System Principles* (Pacific Grove, Calif., Oct.). ACM, New York.

BERNERS-LEE, T., CAILLIAU, R., LUOTONEN, A., FRYSTYK NIELSEN, H., AND SECRET, A. 1994. The World-Wide Web. *Commun. ACM 37*, 8 (Aug.).

DHARAP, C. AND BOWMAN, M. 1995. Preliminary analysis of wide-area access traces. Penn State Tech. Rep. CSE-95-030, Penn State Univ., University Park, Pa.

EMTAGE, A. AND DEUTSCH, P. 1992. Archie: An electronic directory service for the Internet. In *Usenix Conference Proceedings* (Winter). USENIX Assoc., Berkeley, Calif.

GOLDICK, J. S., BENNINGER, K., KIRBY, C., MAHER, C., AND ZUMACH, B. 1995. Multi-resident AFS: An adventure in mass storage. In *Usenix Conference Proceedings* (Winter). USENIX Assoc., Berkeley, Calif.

HOWARD, J. H., KAZAR, M. L., MENEES, S. G., NICHOLS, D. A., SATYANARAYANAN, M., SIDEBOTHAM, R. N., AND WEST, M. J. 1988. Scale and performance in a distributed file system. *ACM Trans. Comput. Syst. 6*, 1 (Feb.).

KATZ, E. D., BUTLER, M., AND MCGRATH, R. 1994. A scalable HTTP server: The NCSA prototype. *Comput. Netw. ISDN Syst. 27*, (Sept.).

KAZAR, M. L. 1988. Synchronization and caching issues in the Andrew File System. In *Usenix Conference Proceedings* (Winter). USENIX Assoc., Berkeley, Calif.

KISTLER, J. AND SATYANARAYANAN, M. 1992. Disconnected operation in the Coda File System. *ACM Trans. Comput. Syst. 10*, 1 (Feb.).

MCCAHILL, M. 1992. The Internet Gopher protocol: A distributed server information system. *ConneXions 6*, 7 (July).

MORRIS, J. H., SATYANARAYANAN, M., CONNER, M. H., HOWARD, J. H., ROSENTHAL, D. S., AND SMITH, F. D. 1986. Andrew: A distributed personal computing environment. *Commun. ACM 29*, 3 (Mar.).

MUNTZ, D. AND HONEYMAN, P. 1992. Multi-level caching in distributed file systems. In *Usenix Conference Proceedings* (Winter). USENIX Assoc., Berkeley, Calif.

OUSTERHOUT, J., DA COSTA, H., HARRISON, D., KUNZE, J., KUPFER, M., AND THOMPSON, J. 1985. A trace-driven analysis of the 4.2BSD file system. In *Proceedings of the 10th ACM Symposium on Operating System Principles* (Dec.). ACM, New York.

SATYANARAYANAN, M. 1981. A study of file sizes and functional lifetimes. In *Proceedings of the 8th ACM Symposium on Operating System Principles*. ACM, New York.

SATYANARAYANAN, M. 1989. Integrating security in a large distributed system. *ACM Trans. Comput. Syst. 7*, 3 (Aug.).

SATYANARAYANAN, M. 1990. Scalable, secure, and highly available distributed file access. *IEEE Comput. 23*, 5 (May).

SATYANARAYANAN, M. 1992. The influence of scale on distributed file system design. *IEEE Trans. Softw. Eng. 18*, 1 (Jan.).

SATYANARAYANAN, M., HOWARD, J. H., NICHOLS, D. N., SIDEBOTHAM, R. N., SPECTOR, A. Z., AND WEST, M. J. 1985. The ITC distributed file system: Principles and design. In *Proceedings of the 10th ACM Symposium on Operating System Principles* (Dec.). ACM, New York.

SIDEBOTHAM, R. N. 1986. Volumes: The Andrew File System data structuring primitive. In *European Unix User Group Conference Proceedings* (Aug.). EUUC Secretariat, Herts, U.K.

SMITH, A. J. 1981a. Analysis of long term file reference patterns for application to file migration algorithms. *IEEE Trans. Softw. Eng. SE-7*, 4 (July).

SMITH, A. J. 1981b. Long term file migration algorithms. *Commun. ACM 24*, 8 (Aug.).

SPASOJEVIC, M. AND SATYANARAYANAN, M. 1994. A usage profile and evaluation of a wide-area distributed file system. In *Usenix Conference Proceedings* (Winter). USENIX Assoc., Berkeley, Calif.

SPECTOR, A. Z. AND KAZAR, M. L. 1989. Wide area file service and the AFS experimental system. *Unix Rev. 7*, 3 (Mar.).

STEINER, J. G., NEUMAN, C., AND SCHILLER, J. I. 1988. Kerberos: An authentication service for open network systems. In *Usenix Conference Proceedings* (Winter). USENIX Assoc., Berkeley, Calif.

TRANSARC. 1991. AFS 3.1 programmer's reference manual. FS-00-D180, Transarc Corp., Pittsburgh, Pa. Oct.

Received December 1994; revised January 1996; accepted February 1996