Arla—a free AFS client

Assar Westerlund
Swedish Institute of Computer Science
assar@sics.se
Johan Danielsson
Parallelldatorcentrum, KTH
joda@pdc.kth.se

Abstract

AFS is a world-wide distributed file system that supports several advantages over NFS, like better administration and availability. Arla is a free AFS-client, today available for a number of systems for which there are not any Transarc clients. Porting it to new systems is not hard. Arla also has some disconnected functionality and supports encryption of the data stream, features that are not present in the Transarc code. Our preliminary measurements of the unoptimized code shows that the performance of Arla is close to that of UFS.

1 Introduction

AFS is a world-wide distributed file system, originally from Carnegie Mellon University (CMU), but now commercialised by Transarc Corporation. It has several advantages over NFS and is quite similar to DCE DFS (actually DFS has been referred to as AFS version 4).

KTH has been using AFS for several years, and the authors, having grown used to it, would not like to abandon it without a decent replacement.

There are a few problems with AFS; it is not available for all platforms, it does not do everything we would like it to, and finally, we cannot give it away to anybody we want.

Therefore, we asked ourselves the question: how difficult would it be to implement an AFS-compatible client? This paper is our attempt at answering that question.

2 The Andrew File System

The development of the Andrew File System [1] started at Carnegie Mellon University (CMU). The main goals were scalability and security. Versions one (AFS1) and two (AFS2) of the Andrew File System were designed, implemented, and deployed at CMU.

During the development of AFS3, it was taken over by a spin-off company, Transarc. They later renamed it to the 'AFS File System'. When this paper only says AFS it refers to AFS3.

AFS is being used by organisations around the world, and there are currently approximately 150 public *cells* – a cell is a part of the global AFS name space managed by a particular organisation. All the users of AFS see the same name space and can transparently access files in other cells (subject to access rights).

2.1 Model

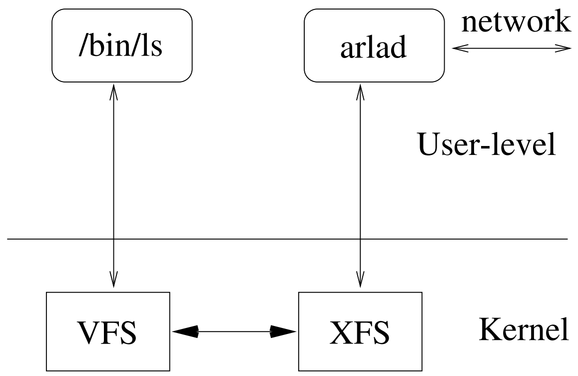
In the AFS model, files are stored on *file servers*. These servers are preferably dedicated to this task. The clients (or *cache managers*) do not need to be trusted at all and have to prove to the file servers that they have the right to perform the operation requested on behalf of a particular user.

The clients store copies of the files (typically on local disk) and the consistency of these are guaranteed by having the file server notify all hosts caching a file before allowing it to change. This notification is called a *callback* [2] (also called a *lease* [3]). It allows the clients to use a cached file without any network activity, as long as it has not changed.

3 Implementation of Arla

The main difference between Arla and the Transarc client is that Arla only puts a small portion of the code inside the kernel, and instead the protocol processing and cache handling are performed by a user-level daemon. This is very similar to, and inspired by, the Coda MiniCache [4].

This separation into two components has facilitated the implementation and the portability of the code. The potential problem with this organisation is that performance could be lacking. We will show that this is not the case. Another advantage is that it facilitates development, allowing people to use their normal development and debugging tools. Having all the protocol processing and policy code in user-space allows linking with normal libraries without having to port them to the kernel environment (which differs quite a lot between operating systems).



3.1 Kernel module (*xfs*)

The kernel module is fairly small, relatively file-system independent and consists only of the minimal functionality needed for obtaining reasonable performance. (The FreeBSD/i386 kernel module is around 5000 lines of code and compiles to 32 KB of object code.)

It implements a system call, a character pseudo-device driver, and a virtual file system. The file system hooks into the Virtual File System switch [5]. The character device is used for communicating with the daemon. This is performed by reading and writing messages on this device. The syscall is used by applications for manipulating the behaviour of arlad.

The VFS system varies a lot between operating systems. The kernel module still presents the same interface to the daemon, which does not need to take these differences into consideration.

The kernel module implements a cache of vnodes and a cache of name mappings. Misses to the caches are served by sending an *upcall* to the daemon. The daemon installs vnodes and name mappings by sending messages to the kernel module.

3.2 User-level daemon (*arlad*)

The daemon is responsible for communicating with the file servers (retrieving files on misses, sending updates, and servicing callbacks). It keeps the cache in normal UFS files and tells the kernel module where to find these.

3.3 Libraries and utilities

Some special libraries are used by arlad: *lwp* (user-level threads) and *rx* [6] (an RPC system). They were written as part of a research project jointly funded by DARPA and IBM. The DARPA funding requirements made them publicly available, which is why we can use them.

A more efficient stub generator for the rx stubs (*ydr*) was written instead of the rpcgen-based one used by the original rx code.

Included in the arla distributions are also versions of the client applications *fs, vos,* and *pts*, that are used for manipulating and examining AFS files, volumes, users, and groups.

3.4 Portability

Only approximately 10% of the code is operating system dependent (the kernel module and a small portion of arlad).

The daemon is portable to any Unix system that supports Berkeley sockets and for which there is an implementation of the context switch function of the *lwp* user-level threads. To witness to the fact that this is a rather narrow set of requirements, it currently runs under Windows NT with the cygwin32 [7] library.

There are currently kernel modules for: SunOS, Solaris, NetBSD, FreeBSD, OpenBSD, Linux, AIX, HP-UX, and Digital Unix.

4 Performance

The test is running the Andrew Benchmark [8] on an IBM ThinkPad 560 under FreeBSD 2.2. It consists of five phases that (I) create a directory structure, (II) copy files, (III) recursively stat files, (IV) recursively read files, and (V) compiles a program. This is not a head-to-head comparison; Arla is compared against UFS which is not a distributed file system. It is useful as a lower bound comparison; Arla stores the cached files in UFS files so it cannot be faster than UFS. All numbers are elapsed time in seconds.

	UFS cold	Arla cold	Arla warm
(I) mkdir	2	2	3
(II) cp	6	18	11
(III) r.stat	2	2	3
(IV) r.grep	5	5	5
(V) compile	34	36	34

Some interesting facts are shown in these results. Phases (III) and (IV) only refer to files created during phase (II) and should not cause any network traffic; this is compatible with the difference between UFS and Arla being small. The other three phases actually create files which means that Arla needs to send data over the network and is forced to perform more work than UFS.

When running the same benchmark with encryption enabled the results were very similar. This result also indicates that the overhead imposed by having part of the functionality running in user-space is not the limiting factor.

5 Security and encryption

Rx supports different types of authentication mechanisms. The one used in practice is *rxkad* which uses Kerberos 4. Due to the American export restrictions this module was not made available together with the publicly available rx implementation. Instead, we use Björn Grönvall's implementation of rxkad. It was written outside the US and is also an order of magnitude faster than the one distributed by Transarc.

The rxkad mechanism supports three different protection levels:

clear	checksum some selected header fields
auth	clear + protected packet length
crypt	auth + encrypt packet payload

Arla naturally allows the user to choose between these levels, defaulting to 'auth'. As we see it, there's no reason, except for potential server overload, not to use 'crypt'.

6 Disconnected operation

Many computers today are mobile or have intermittent network access for some other reason. This is one instance where it would be beneficial to trade consistency for availability. Always requiring some file server to be reachable, makes the system fragile. A simple way would be to allow access to cached data, even when the network is unavailable (and consistency therefore cannot be guaranteed). Additionally, it's useful to make sure that the cached data survives a reboot.

For handling mutating file operations it is necessary to perform them on the cached files and log the operations performed so that they can be merged later once the network is back up. For different usages and situations it has proved worthwhile to have an intermediate state between *connected* and *disconnected*, called *fetch-only*. The following table explains the relations between these states.

	fetch on miss	no fetch on miss
consistent	connected	N/A
not consistent	fetch-only	disconnected

Disconnected operation was an integral part of the Coda file system [9], a descendent of AFS2. It was later implemented in the Transarc code base [10]. Arla implements a subset of the functionality of these systems.

7 Future work

A file server and the database servers required for running an AFS cell are the obvious next steps once the client is stable enough. Development of them has already started.

Ports to new platforms and operating systems is something that will make Arla useful to more people. We are specially interested in a port to Windows 95/NT (but that seems to require a SDK only available in the US and Canada [11]).

Improving performance will make it more feasible to use Arla instead of a local file system.

Implementation and experimentation of more disconnected support is another item that is on our to-do list.

8 Acknowledgments

Björn Grönvall tricked the first author into starting writing Arla and is also responsible for large portions of code and encouragement. His rxkad implementation is also a necessary part of Arla.

Magnus Ahltorp
Robert Burgess
Artur Grabowski
Love Hörnquist-Åstrand

A large number of people have contributed bug-fixes, documentation, and encouragement. They are mentioned in the file `THANKS` in the distribution.

9 Availability

Arla is freely available under a BSD-style license. See `http://www.stacken.kth.se/projekt/arla`.

References

[1] J. H. Howard, *An Overview of the Andrew File System*, In Proceedings of the USENIX Winter Technical Conference, 1988.

[2] M. L. Kazar, *Synchronization and Caching Issues in the Andrew File System* In Proceedings of the USENIX Winter Technical Conference, 1988.

[3] C. G. Gray, D. R. Cheriton, *Leases: an efficient fault-tolerant mechanism for distributed file cache consis-*

tency, In Proceedings of the 12th ACM Symposium on Operating System Principles, 1989.

[4] D. C. Steere, J. J. Kistler, M. Satyanarayanan, *Efficient User-Level File Cache Management on the Sun Vnode Interface*, In Proceedings of the USENIX Summer Technical Conference, 1990.

[5] S. R. Kleiman, *Vnodes: An Architecture for Multiple File System Types in Sun UNIX*, In Proceedings of the USENIX Summer Technical Conference, 1986.

[6] B. Sidebotham, *Rx: A High Performance Remote Procedure Call Transport Protocol*, Information Technology Center, Carnegie Mellon University

[7] Cygnus, *The GNU-Win32 Project*, `http://www.cygnus.com/misc/gnu-win32/`

[8] J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham, M. J. West, *Scale and Performance in a Distributed File System*, ACM Transactions on Computer Systems, 6(1), February 1988.

[9] M. Satyanarayanan, *Coda: A Highly Available File System for a Distributed Workstation Environment*, In Proceedings of the Second IEEE Workshop on Workstation Operating Systems, Sep. 1989.

[10] L. B. Huston and P. Honeyman, *Disconnected Operation for AFS*, In Proceedings of the USENIX Mobile & Location-Independent Computing Symposium, August 1993

[11] Microsoft, *Windows NT IFS Kit—Ordering* `http://www.microsoft.com/hwdev/ntifskit/order.html`