Secure Filesystems

Simon Wilkinson and Craig Strachan
School of Informatics
University of Edinburgh
simon@sxw.org.uk cms@inf.ed.ac.uk

February 26, 2007

1 Introduction

The School of Informatics at Edinburgh University is in the process of deploying a next generation, secure, networked filesystem for general computing use. This paper presents an overview of our experiences to date in this deployment. It discusses the local requirements for a networked file system, provides an evaluation of products in this space, and details our reasons for choosing AFS.

It addresses the issues involved in deploying OpenAFS over a medium scale site, discussing the AFS architecture, and considering disaster recovery and backup issue. Finally, it examines the lessons learned from our pilot program, and presents some solutions for the technical and social issues we encountered.

2 Background

The process of evaluating replacements for our existing networked file system began towards the end of 2004. At this time, the School of Informatics had approximately 1500 hosts in service located in four geographically separate sites and connected via the University of Edinburgh local area network. These hosts were mainly PCs running Red Hat Linux and maintained via a locally developed configuration system for large numbers of Linux hosts known as LCFG[1]. Six commodity file servers running Solaris 9 on Sun hardware and located at three of the four Informatics site exported the twenty terabytes of disk space available within Informatics using NFSv3. This file space was automounted onto local hosts using amd to create a site wide view of the available file space. The automounter maps defining this view were distributed and managed though LDAP. Authorisation and authentication services for the approximately 2000 active users, were provided by the local Kerberos V and LDAP infrastructures[2].

There had been a desire within Informatics for many years to look for an alternative to NFSv3. Its widely acknowledged security shortcomings, in particular the level of trust placed in client machines when controlling user level access, had led the School to impose severe restrictions on connection to its networked file space. In particular,

access was limited to trusted, centrally managed, hosts located on the school's intranet and users wishing to access their file space from the outside world were obliged to do so via Samba and our VPN.

The amd and LDAP combination presenting this filespace to the user was not as stable as we had hoped. In addition, the need to maintain and update the data sources and scripts which we used to generate the automounter maps imposed a significant workload on the School's computing staff. This local glue would also have presented a significant barrier to providing seamless access to machines outwith our control, even if our security concerns allowed this.

Management of the file system could also have been easier. Moving data between file servers or even between different partitions on the same server caused significant disruption to the users of that data, both because of the delay in the propagation of the new location of the data to all hosts, and the need to take the data offline whilst the move occurred. Any interruption to the NFS service could result in the need to reboot many clients due to stuck amd mount points. Should a server go out of service, making the affected data available via another server required significant resources and time. Given that it was clear that our amount of managed disk space was going to increase greatly (it has doubled in the last two years), it was apparent that our existing systems would prove inadequate for managing this volume of data.

The file system provided by NFSv3 was also increasingly failing to match the needs of the Informatics user base. The access model for NFSv3, particularly the limit of only being able to apply permissions to one user group was insufficiently flexible. Our self imposed access restrictions meant that collaborating with colleagues in other institutions was problematic. The increasing availability of disk space on clients, combined with the obvious performance gain of using local rather than networked file space led to a desire on the part of the user base to make more use of this local disk. Efficient management of this disk space under the existing system was impractical.

The final motivation for a new file system was the wish to reduce the level of administrative control we required over machines accessing it. As mentioned earlier, the weakness of the NFSv3 security model required us to maintain complete control over all clients, and only allow trusted computing staff super user access to those clients. We were already seeing a significant increase in Mac OS X usage, and a there was a clearly expressed desire to have more "self managed" Unix machines. On these machines configuration control rested with the user, rather than with the system administrators, making a security model that trusted the machine to say which user it was acting as no longer tenable. This desire, and the timescales attached to our delivery of it, were a significant motivator in the timing of our decisions.

3 Requirements

The following requirements were identified for the new file system:

- It should offer enhanced security both in terms of authentication to the file system and in the flexibility of the permissions which could be allocated to users. Integration with our existing security architecture would be highly desirable.

- It should offer enhanced performance either through the implementation of more efficient transfer protocols or through the use of local caching.
- It should be relatively easy to enable file system access through a firewall.
- It should be stable, have a large user base and be well and actively supported.
- It should run on Fedora Core on both the server and client end. Solaris server support would be an advantage.
- It should be scalable and capable of supporting well over 1000 clients.
- It should not have a significant cost, and have no per-client licensing requirements.

Desirable features were that there should be stable and well supported clients for Windows XP and Mac Os available. We also wanted adoption of the new file system to allow us to end our dependence on both Samba and amd with its associated support infrastructure.

4 The Selection Process

With these ideals in mind, an initial list of candidates was drawn up. The 6 products selected for initial consideration in spring of 2005 were OpenAFS, NFSv4, CIFS, DFS, Coda and Intermezzo, and the summary below reflects the state of the market at that point in time.

Coda and Intermezzo are both interesting research filesystems, offering the possibility of making considerable use of the client local disk, along with providing interesting next generation features such as disconnected operation. Both can trace their lineage back to AFS, with Coda having being developed as a spin off from AFS, and Intermezzo as a lighter weight spin off from Coda. Intermezzo is no longer actively developed, and Coda, whilst highly interesting, was considered to lack sufficient maturity to be deployed at the scale we were considering.

CIFS is the native filesystem protocol for Windows machines, with Samba providing the Unix implementation. At the time of evaluation, there was no stable Linux kernel module providing CIFS access, which ruled it out as a candidate for our Unix systems, as did the lack of Kerberos based authentication. We also had concerns regarding our ability to scale our Samba setup to deal with the significantly larger volume of data and clients we were considering deploying. DFS was also rejected due to the apparent lack of a sustainable user community, following IBMs decision to end of life the product, and its heavy dependence on DCE.

This left us with two serious candidates - NFSv4 and AFS, which are really the major players within this application area.

4.1 Background on NFSv4

NFS was developed by Sun Microsystems, making its first public appearance in 1984. It was originally designed as a simple, stateless protocol, operating over UDP. Successive revisions have added additional functionality, such as security and locking, TCP support, and most recently, state! Following the release of NFSv3 in 1995, Sun gifted change control over the NFS protocol to the IETF, and NFSv4 was developed under their auspices.

Whilst strong authentication had been available in some variants of NFSv3, it was with NFSv4 that it reached widespread availability. NFSv4 features a number of significant new features, including a focus on wide area operation, integrated lock management, file and directory ACLs and support for filesystem migration and replication between servers.

NFS is widely available in Unix influenced operating systems, although support for NFSv4 is still emerging. Clients are also available for Windows and Mac OS platforms, although the level of their v4 support is unclear.

4.2 Background on OpenAFS

AFS was originally developed at Carnegie Mellon University, as part of their Project Andrew campus computing initiative, hence its full name - the Andrew File System. Carnegie Mellon always reserved the rights to their implementation, and in 1989 Transarc Corporation was founded to commercialise AFS. Over this period AFS usage spread to many large academic and corporate installations, however widespread adoption was hampered by high client licensing fees. In 1998, IBM bought Transarc Corporation, and AFS became an IBM product, which it remained until IBM announced the end of life of the commerical product in 2002. Two years after buying Transarc, IBM released the AFS source code under an open source licence as OpenAFS. The OpenAFS project rapidly gathered steam, and when IBM made its end of life announcement, the OpenAFS product was already superior to IBMs in terms of both stability and features.

Meanwhile in the mid 1990s, with no sign of a freely available AFS offering, Stacken Computer Club at KTH in Sweden started developing their own independent implementation named Arla. Arla development continues to this day, in conjunction with the OpenAFS offering. Arla is still the preferred AFS client on a number of platforms.

AFS is centered around the idea of a global, distributed, filesystem. All of the filespace that AFS provides is joined into a single filesystem, usually rooted at /afs/ on the client machine. This amalgamation is performed transparently by the AFS cache manager, and allows all sites which have opted in to the global filesystem to access files at other sites using the same, globally unique, path. Files in AFS are arranged into logical groupings, termed 'volumes'. These volumes can be moved, and replicated, transparently to the user, allowing the migration of files between file servers without any user intervention.

Originally AFS used a native security protocol, heavily influenced by Kerberos v4, to provide authentication, integrity and privacy protection. As the weaknesses of the

Kerberos v4 protocol came to light, mechanisms were developed to move away from this native protocol towards external, Kerberos v5 based architectures. It's upon these mechanisms that any new deployment of OpenAFS should be based.

Current support for OpenAFS is provided via a vibrant free software community, with a number of companies also providing paid-for support, either on a per incident, or long term basis

4.3 Feature set comparison

On paper NFSv4 and AFS offer similar feature sets. This is not entirely surprising, as much of NFSv4's development was motivated by the features available in other filesystems such as CIFS and AFS. The major difference is in AFS's vision of a global filesystem - NFS still exports filesystems on a per machine basis, relying on local technology to glue those together, and allowing different clients to have different view of the filespace. With AFS there is a common namespace which can be shared not only by all clients within a single site but by all clients in all collaborating sites, too.

NFSv4's youth means that there is a significant discrepancy between the standardised feature set, and that implemented in the clients, as we discovered when we started to evaluate the available implementations

4.4 Initial evaluation

We conducted our evaluation on relatively low powered, desktop hardware, using the Fedora Core 3 Linux distribution. Whilst also being readily available, it was felt that using low powered hardware might be useful in identifying any load issues either candidate might have.

Support for NFSv4 was included in the default FC3 configuration. Installing NFSv4 on the test server and client simply required editing the appropriate system configuration files and creating Kerberos principals, neither of which posed any problem.

AFS was more problematic, as it does not ship as a standard component in the Fedora system. We also had a choice between using OpenAFS and Arla as our Linux implementation. We chose OpenAFS as it was significantly more mature and complete than Arla for all our target platforms. Whilst OpenAFS make RPMs available for some releases, no RPMs were available for FC3 at the time of our evaluation, so we were obliged to compile and install OpenAFS from source. Our initial OpenAFS setup was somewhat atypical. In a normal AFS installation, file servers and database servers would normally run on different hosts and there would be several of both types of server in a cell. In our initial test setup, all services were running on one host.

4.5 Initial Findings

When testing of the two candidate files systems began, two security issues immediately became apparent. Whilst the Kerberos based authentication built into OpenAFS worked seamlessly with the existing Informatics infrastructure, it soon became clear that while NFSv4 claimed to support user based authentication, the version we were

using only supported host based authentication. This meant that it offered little advantage over our existing NFSv3 setup, as it still required trusted hosts to identify their users.

To address the authentication issue, the latest released code for NFSv4 was downloaded, compiled and installed. Though this did go some way to addressing the problem, the new code only provided for checking user credentials at mount time, rather than file access time. This solution of providing access control was only tenable for systems being used by a single user, which is not applicable to our environment. For this reason, it was decided to conduct the rest of the evaluation using host based authentication with NFSv4, despite this not being an acceptable solution for final deployment.

One other problem emerged with the version of NFSv4. To evaluate the performance of the candidates, we were interested in using the Andrew[3] benchmark. As part of its measurements this builds and links a package contained in its test file tree. This benchmark consistently hung when run on a NFSv4 file system, further investigation showing that this occurred when the ar utility called chown. Whilst this transpired to be a known problem with the Linux NFSv4 server and client, it did not increase our confidence in the maturity of the system.

4.6 Performance

To measure the candidate filesystems' performance, two benchmarking programs were used, blogbench[4] and Iozone[5]. Whilst we had intended to also use the Andrew benchmark, we were unable to do so.

Blogbench attempts to simulate the load on a real world file system by simultaneously performing performing random writes, reads and re-reads in order to get some idea of the scalability and the concurrency a system can handle. Iozone makes a number of measurements of file system performance using varying file and record sizes but does not attempt to recreate an interactive environment. For our measurements, we measured how the candidate file systems performed reads, re-reads, writes and rewrites using file sizes from 64KB to 128MB and record sizes from 4KB to 16MB.

In the Blogbench tests, AFS came out slightly ahead of NFSv4 though the difference was not great. The initial results from iozone told a very different story with NFSv4 consistently and significantly outperforming AFS once the size of the file being used in the test had grown above the size of the local AFS disk cache. Often NFSv4 was more than 3 times as fast as AFS. Since these results were somewhat unexpected, the benchmarks were rerun using iozone's -c flag which includes the close() operation in the timing calculation. This time the results for writing and rewriting were much more evenly matched with NFSv4 showing a slight advantage with files sizes greater that 32MB. For reading, NFSv4 once again showed a big advantage, often performing reads twice as fast as AFS and sometimes approaching 5 times as fast. AFS had a slight advantage when performing rereads.

As well as these empirical tests, we attempted to get a feel for how fast the candidates were by building packages on the candidate filesystem from clients both within and outside the School's LAN. Though these results should be taken with an extremely large pinch of salt, the general consensus was that OpenAFS felt faster. Our iozone results highlighted AFS's poor performance when doing single operations without the

benefits of its local cache. Tests which attempted to simulate interactive operation, showed OpenAFS on a more equal footing.

4.7 Evaluation Conclusions

At the time of testing, only OpenAFS offered reliable user based authentication which would be suitable for hosts with multiple users logged in at the same time. The NFSv4 ACL language offered more flexibility than AFS's, not least because ACLs could be applied to both files and directories. Even with the Linux implementation's cut down ACL support NFSv4 supplied a more more verbose ACL language, although without the power of permitting user defined groups.

Neither candidate offered better performance in the general case than our existing NFSv3 setup. This is perhaps unsurprising, given the additional processor overhead of performing the encryption and authentication operations.

OpenAFS worked well through our firewall, and both OpenAFS and NFSv4 appeared to have a proactive support community. However, we had significant concerns about the maturity of the Linux NFSv4 solution.

Both candidates had both server and client support for Fedora Core 3. To use NFSv4 on our Solaris servers, we would have had to upgrade to Solaris 10. The OpenAFS server built and ran under Solaris 9.

When it came to our desirable features, OpenAFS provided both Windows and Mac OS clients, and could provide a solution free of amd and Samba.

It was clear from our evaluation that OpenAFS was a much closer match to our requirements. It appeared to be more stable than NFSv4, matched our available hardware better and was well supported. However, we were aware that adopting OpenAFS would require much more effort from the Schools computing staff and would require more adjustment on the part of the user base. For these reasons, we decided to proceed with a pilot OpenAFS service, with a view to gaining more experience with the system. If the pilot proceeded according to plan then it would be turned into a production system.

5 Introduction to OpenAFS

AFS has three units of data manipulation - the file, the volume, and the partition. The notion of files and partitions are identical to those used in local filesystems. The volume is a logical collection of files - for example, a user's homedirectory, or the set of files associated with a particular project. A volume is the basic unit of management in AFS, and are the key to some of its power. In a normal setup every volume will have a read-write version. For many volumes, this will be the only copy stored. For heavily used data, multiple read-only copies may be created, which are used by clients in preference to the read-write copy. This provides a mechanism both for load-sharing and providing redundancy. The process of copying from the read-write copy to the read-only replicas, known as releasing, is manually controlled - there is no automatic propagation from the read-write volume to the read-only one.

To aid the backup process special read only copies, known as backup volumes, can be created. These volumes provide a snapshot in time of the original data, and allow consistent backups to be taken without taking data offline. In addition, the backup volumes may be mounted and provide a means of providing direct user access to yesterday's data. Backup volumes must be stored on the same partition as their read-write parent volume, but only store the changes between themselves, and the read-write copy. This dramatically reduces disk usage, but makes them unsuitable as a means of doing disk-to-disk backup for disaster recovery.

AFS supports the migration of volumes between servers transparently to the end client. It is possible to move sets of data between file servers, whilst that data is being accessed by one (or more) clients, without the end user being aware that the move has occurred. This provides significant management benefits when dealing with partitions that are filing up, or even moving data off a file server that is about to be shut down.

AFS's client server model is comprised of three classes of host. There are AFS clients, fileservers, and database servers. The roles can be combined, so a single host may provide all three functions. Database servers are at the heart of AFSs filesystem and provide at least two functions. The volume location database allows clients to determine which file servers provide a given AFS volume. The protection database stores all of the user and group information necessary to evaluate ACLs. Whilst it is possible to run a site with only one database server, this is not recommended as it creates a single point of failure. Instead, AFS provides a powerful multi-master system named Ubik to allow the creation of redundant farms of database servers.

File servers perform as their name suggests - serving selections of files to AFS clients. Unlike most NFS servers, AFS does not store files transparently onto the native filesystem, so files are only accessible through an AFS client. This means that migrations can be performed without having to worry about local clients also accessing the data, but does mean that all accesses, including backups and restores, must be performed over AFS, rather than directly to the local disk. Some versions of OpenAFS's fileserver bypass the local filesystem completely, and store data directly into the disks inodes. These are faster than those which use the filesystem as a hash-based filestore, but are potentially dangerous if native operating system tools such as fsync are run on the partition in question.

AFS clients are comprised of a kernel module, and a user-space cache manager. The cache manager communicates with the volume location database in order to locate the fileserver for every volume the client accesses, fetches data from the fileservers, and administers the local disk cache. AFS's local disk cache provides it with a significant speedup when dealing with files (or directories) which are repeatedly accessed. AFS uses a callback based mechanism for ensuring the consistency of this cache. When a client caches a file (or portion thereof) it registers a callback with the fileserver holding that file. If that file is modified, then the fileserver breaks that callback with the client, causing it to mark its cached copy as invalid.

As mentioned earlier, modern OpenAFS deployments utilise Kerberos for user authentication - this turns authentication into a two fold process. Firstly, the user must obtain Kerberos credentials, then those credentials must be converted into an AFS authentication token. Tokens are the unit of AFS authentication - they are an in-kernel representation of a user's access rights and credentials. They can be associated either

with a user's Unix ID, or with a particular session (often referred to as a PAG). AFS tokens have a limited lifetime, in a similar way to the underlying Kerberos credential, meaning they must be renewed at regular intervals. The maximum duration between renewals can be set according to each organisation's security policy.

One drawback to current OpenAFS releases is that they still use security based around the DES encryption algorithm, and in some areas use a DES variant - fcrypt - with an even smaller key space. For sites with significant security concerns, or regulatory obligations, this can be a real problem. For this reason, the AFS community have been developing a replacement transport encryption mechanism, named rxgk. A derivative of rxgk, rxk5, is also in development and should be ready for deployment later this year.

Other forthcoming developments include support for disconnected operation. Whilst disconnected operation for AFS was developed at the University of Michigan in the early 90s, the closed nature of the AFS source at that time meant that it never made it into the main code base, and the code has suffered significant bitrot since that time. However, there is now significant interest in reviving the development, and introducing it into OpenAFS releases. AFS's cache consistency guarantees and file metadata make it possible to deploy disconnected operation without conflicts in a large number of usage scenarios.

Other work includes support for extending backup volumes beyond the current single copy, creating the potential for yesterday, last week, last month, last year sets of volume backups, and of creating the ability to use them for disk to disk disaster recovery backups.

While our primary interest is in Unix platforms, our goal of replacing Samba required robust Windows support from the selected filesystem. OpenAFS's Windows client has improved in leaps and bounds over the last few years to the point where its performance and reliability are now at least the equal of the Unix client. Support for both Windows and Mac OS X is freely available and easily installed.

6 Deployment

With the decision made, a project team of was set up to oversee the creation of the new file service. It was obvious that such a major change could not take place overnight and so a policy of gradually expanding the amount of AFS disk space in service and the user group using the service was implemented. Initially a small set of users were provided with AFS file space in addition to their normal NFS exported space. These users were carefully chosen both for their ability to push the uses to which the file system was being put and for their willingness to tolerate interruptions to the service. It was made clear to them that this was a pilot service, and was as much a learning experience for the administrators as it would be for them. Despite this, the offer of additional disk space is always an alluring one and there were a number of volunteers.

The next step was to obtain hardware to provide the file space they would be using, as it was clear that the evaluation hardware would be insufficient for the task. One of the first tasks of the project team was to build and install the AFS server on a Solaris system. This worked well and we decided to use two Solaris hosts as the initial servers.

One was a research machine on which 100GB of fibre channel RAID5 disk was allocated for AFS use, the other a recently retired server to which was attached a cluster of surplus SCSI disks adding up initially to around 150GB. Unfortunately, these surplus disks were less than reliable. and gave us early experience in restoring data hosted in AFS.

We moved user homedirectories gradually to the new system, starting with the project team, followed by the rest of the computing staff, and other interested early adopters. At the same time, project groups with large data storage requirements were asked to consider AFS for their storage needs. This process was accompanied by the acquisition of 3.5TB of fibre channel RAID5 disk dedicated to AFS, allowing us to pension off the surviving SCSI disks. Our existing NFSv3 servers now share the dual roles of NFS and AFS fileservers.

We are now at the stage where all new non-student users get an AFS home directory and the intention is that with the start of the new academic year in September 2007, all new users will get an AFS home directory by default. The pace of adoption was kept deliberately slow in order to avoid undue prejudice being generated against the system whilst it stablised. It was considered preferable to roll the system out to staff, before adding large numbers of student users.

This leaves one major adoption task to complete, moving all of our current users to AFS file space. It was decided at an early date that we would not move relatively short lived users, particularly current students, to AFS file space. Whilst this extends the life of our NFSv3 system for at least the next 3 years, it removes the need for an expensive transition. Significant support effort and computing staff time would have been required both to move all of the student homedirectories, and to support the students whilst they familiarised themselves with the new system.

Moving our remaining, non-student users, is not so much a technical problem as an exercise in user education. Members of the AFS project will be making presentations to small groups of users promoting the advantages of moving to AFS (not least of which is increased quota!) and giving some idea of the problems which may be encountered. It is also anticipated, at least in the early stages, that many users may need a higher level of support than normal to ease the transition to AFS. We are fortunate that several senior academic figures including the Head of School have already adopted AFS and are acting as evangelists.

6.1 Deployment Experience

One of the fundamental challenges of moving from an insecure filesystem such as NFSv3, to a secure one, is usability. Many features that users have taken for granted in less secure envirnoments are no longer available. A particular issue is that of credentials, both having to explictly gain tokens upon login, and the inconvenience of them expiring invariably in the middle of some long-running computation. Mitigating these two factors is a key element of ensuring the acceptability of any secure filesystem.

Locally, our infrastructure acquires Kerberos credentials upon user login to any managed machine. PAM based technology was already in place for our Unix machines to use these Kerberos credentials to gain X509 certificates on behalf of the user. We extended this technology with an additional PAM module to ensure that AFS tokens

were gained transparently at login time. Because our Unix home directories are hosted on AFS, it was vital that these credentials were gained as part of the login process, so that the shell startup scripts and the like still had access to the user's home directory.

This still left the issue of credential expiry. For a variety of reasons, we were keen to not extend our Kerberos credential lifetime indefinitely. This meant that we had to develop mechanisms to deal with credentials expiring under users processes. Initially, we extended our Kerberos lifetimes to 18 hours, which meant that there should be no problems with credentials expiring within a working day. We have modified our PAM libraries and screensaver configurations so that unlocking a screen has the effect of renewing both Kerberos tickets and all other associated credentials, including AFS tokens. These two actions mean that the majority of users now no longer see problems due to being unable to access their filespace.

However, further measures will be required to deal with users who have long running jobs that require filesystem access. We intend to setup up a system of allowing users to delegate certain parts of their access rights to Kerberos principals whose key material will be stored on the local disk. The use of utilities such as kstart will then allow processes to be run indefinitely with credentials and tokens created from that key material.

We still face problems with expiry whilst the machine is unattended. In general, this does not seem to be an insurmountable problem, but there are some applications, such as Thunderbird, which continue operating whilst a machine is locked and react badly to being unable to access files on the local disk. As yet, a solution to these is unclear, although its likely that it will be combination of a further increase in ticket lifetime, together with code patches for the affected applications wherever possible. An additional issue arises with screensavers themselves, which may require access to the .Xauthority file in the user's homedirectory. If credentials expire whilst the screen is locked, then it is no longer possible to create the window necessary to unlock the screen. The solution in this case is to configure the desktop manager to create the .Xauthority file on the local disk, rather than in the networked homedirectory. Whilst this is trivial to do, it does provide a good example of some of the problems that can be encountered when using a secure filesystem.

There are additional issues related to AFS's credential model. If you opt in to the global filesystem, by allowing AFS traffic beyond your site perimeter, users must realise that "anyuser" permissions mean any user, any where. The fact that "anyuser", AFS's version of anonymous access extends so widely can also cause problems for daemons that access the AFS file space. If you want web servers to have access to scripts in AFS that are not made available to the outside world, then these daemons must run with suitable credentials at all times. IP based ACLs do provide an alternative solution to this, but using them is return to trusting the network layer, which we were keen to avoid.

Another significant issue is the change in ACL model. Our users are familiar with the existing Unix model of user, group and other, and read, write and execute. AFS ACLs providing a significant learning challenge. Whilst in many ways they provide far more flexibility than convention Unix controls, the fact that they only operate on a per directory basis poses significant challenges. Whilst, in general, these challenges only affect our more advanced users, they do pose a significant barrier to adoption.

The lack of support for creating FIFOs and Unix sockets also pose a problem for advanced users, as well as requiring additional configuration of some window managers.

6.2 Backups

Our major outstanding technical issue is devising a suitable mechanism for performing disaster recovery backups of our data. OpenAFS's inbuilt backup mechanism is somewhat outdated, and would require significant scripting to bring it up to production quality.

Whilst many site-specific issues come into play here, and the details of selecting a backup policy could make a paper in their own right, it is worth examining one of the major considerations for AFS backup - what unit of backup you use. As explained earlier, it is natural to arrange backups by dumping each individual volume. This simplifies backup management, as well as ensuring that all of the AFS metadata is also captured to tape. However, it does complicate situations where only a single file need be restored. In these cases, the entire volume would have to be restored to a holding location, before a file can be copied out of it.

We are still in the process of evaluating the replacement for our backup technology - as an interim measure we use our current technology to walk the backup volumes in our AFS filesystem, and save these to tape on a file by file basis. This does result in the loss of any mountpoint, and ACL, information. We are urgently investigating replacements for this stop gap solution.

6.3 Performance

Performance of AFS in applications which require a high read or write throughput, but are unable to utlilise the local disk cache, is still concerning. We are undertaking some work to profile these applications, and to better optimize our client and file server configuration. Planned improvements to the OpenAFS source code should also reduce these problems further in future releases.

7 Conclusions

Our choice of filesystem has been justified by our experiences to date. In service AFS has proved itself to be more robust, more reliable and easier to manage than our existing NFSv3 setup. Recovery from service outrages is quicker and easier and we have yet to come across any unsolvable problems. Since much of our NFS problems stem from amd and its associated infrastructure, it is unlikely that we would have seen much benefit from moving to NFSv4 and leaving the remained of the infrastructure unchanged, even if the security and usability issues of NFSv4 could have been resolved within our timeframe.

Progress in adopting the new file system has been slower than we hoped (we are now almost exactly a year behind our original optimistic timetable) but this delay has

mostly been due to making the service as reliable as possible as early as possible and we believe this delay to have been unavoidable.

Many of the lessons learned and the problems encountered during this process are those applicable to any major change to the computing infrastructure. Users will always be resistant to change unless the change addresses issues which are causing them problems or it provides an immediate and obvious advantage over the old system. In our own case, it is fair to say that some of our users find that their use of the file system has become more complicated, firstly by the need to become and remain authenticated and secondly by the change in the way file permissions work. At the same time they may have no need or appreciation of the major benefits these changes bring, namely the improved security and access. This of course is only a problem with users used to the old filesystem which is why in many ways new users are much easier to deal with than old ones.

In addition, the take up of self managed machines which was originally one of the main drivers for deploying a secure filesystem for home directories has been considerably less than anticipated, which removes one of the advantages upon which the new filesystem could have been sold.

In our experience, the critical factor when making such a significant infrastructure change is to make the utmost effort to ensure that the new system is as reliable as possible before normal users make use of it and that the user base has been fully informed about why the change is necessary. Reliability is important since any breaks in service will be blamed on the new service whether directly related to problems with that service or not. If the new service is not to be labeled as broken before its full benefits can be realised, it is important to make as clear as possible where the true responsibility for service interruptions lies.

References

[1] Paul Anderson and Alastair Scobie *LCFG - The Next Generation* UKUUG Winter Conference 2002 http://

[2] Simon Wilkinson *Kerberizing our Network* UKUUG Winter Conference 2006 http://

[3] J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham, and M. J. West, *Scale and performance in a distributed file system* Transactions on Computer Systems, vol. 6, pp. 51-81, February 1988.

[4] *Blogbench - A filesystem benchmark for Unix* http://www.pureftpd.org/project/blogbench

[5] W. Norcott *The IOzone Filesystem Benchmark* http://www.iozone.org