PhD Thesis Proposal

Russ Ross <rgr22@cl.cam.ac.uk>
University of Cambridge Computer Laboratory

31 January 2004

1 Introduction

Research into distributed filesystems has largely focused on two models: client/server systems where the data lives on a server and is accessed by a remote client (also called network filesystems), and peer-to-peer (P2P) systems where the data is distributed with little or no bias among participating members. In the former the overall topology is determined explicitly by administrators, and in the latter it may be deliberately randomized for resilience to partial failure.

I propose to design and implement a new distributed filesystem designed to reduce administrative overhead for a diverse range of users, while keeping data primarily on storage owned by the primary users and allowing different behaviour among different types of participating machines.

2 Goals and overview

2.1 Participants provide their own storage space

One attraction of P2P filesystems is the alluring promise of harnessing the unused storage capacity of thousands of other participants. With the decreasing cost of storage and the high cost to the system of compensating for unreliable participants[2], I choose instead to focus on the benefits of location transparency and simplified administration. I assume that each individual, organisation, or explicit group of participants is willing to provide for its own storage needs, possibly by arranging reciprocal agreements with other members or by hiring additional space from a provider.

Explicitly managed infrastructures have served well for many years. Instead of scattering content over a wide range of participants, I take the topology of existing storage models as a target worth emulating at least to some degree. Different participants may assume different roles that affect their behaviour in the system. A shared workstation in a public lab will keep local copies of popular application files but almost no personal data, while a server with high storage capacity and good connectivity may act in a role closely resembling that of an NFS[11] server to provide home directories to the users of that same lab. A laptop user may hoard[18] replicas of more content than a permanently networked workstation, and a web server will use different policies for caching replicas of the content it serves than a server running simulations will for input data it accesses.

Using more reliable storage provides a few practical benefits. The required degree of replication is related to the volatility of participants, so organisations that already have carefully managed storage capacity do not need as much replication as a system with random participants on the internet. In addition, many organisations will feel more secure when they can ensure that a local replica exists. Analysis of current P2P networks shows high volatility in many participants[17], and catastrophic events are a looming possibility for large scale P2P applications, e.g., a service provider turning on a filter that disables the protocol, a large university entering a vacation period and losing thousands of participants overnight, or a worm crippling large subnets of home users. Finally, local storage means that a group of participants can continue to function with most capabilities intact even when isolated from the rest of the network.

2.2 Replicas can be located by contents

Many different file storage and distribution problems can and should be addressed by a large-scale distributed filesystem. One general goal I have is to make a system with a global name space that is suitable as the primary means of both storage and distribution for many file users.

With a global name space, providing fast file transfers and reasonably sophisticated update propagation would obviate the need for separate, dedicated protocols for file transfers (FTP, Bittorrent[1], rsync[2], etc.). If the system is aware of content duplication (even under different names), transferring from a nearby replica or through a swarm approach using erasure codes or a simpler partitioning scheme like Bittorrent would be possible.

The system must be sensitive to the need for backups. For many users, replication (perhaps with the ability to place constraints on location to ensure sufficient distribution) can replace offline backups. For others, a server with the ability to copy appropriate subsets of the name space onto offline storage is desirable. Using extra storage capacity to track file updates and provide a version history is also desirable to minimise the need for offline backups and to provide a much more convenient alternative for retrieving an accidentally deleted or modified file[16].

To be acceptable as a primary filesystem, performance for the common case must approximate that of local filesystems. Intelligent and widespread replication of commonly used files (based on usage patterns and on explicit requests from more sophisticated users, or through pre-packaged profiles made available to users) and the ability to differentiate between the capabilities of different participant machines should make this possible in most cases[18].

Many of these goals will require refinement over time, and the behaviour of the system will vary quite a bit for different types of participants, as storage requirements and roles already vary among different users and machines today. My emphasis will be on defining primitive operations along with required and optional semantics for each participant to provide for flexible behaviour. Name space operations, data transfer primitives, security protocols, and metadata queries are examples of the components that will be common to all participants. Decisions about replica placement, migration, update propagation, and storage allocation (to different users) will vary with the participant profile and possibly even with different implementations.

Storage at the block level is not the emphasis of this research, and a prototype implementation will be layered above an existing local filesystem, ideally one with rich attribute support such as Reiser4[12]. To simplify implementation and allow more focus on the wire protocols and policy decisions, the prototype could be implemented as a loopback userspace NFS server. I have implemented a userspace NFS server before[13] and am confident that this will provide a sufficient environment at least for a prototype.

3 Related Work

3.1 Client/Server distributed filesystems

Also called network filesystems, client/server distributed filesystems are widely used. In network filesystems, files have a home on a particular server (or cluster of servers) and clients issue file requests to the server.

Various cache strategies improve performance to make this usable on a LAN or over the internet. NFS[11] uses a simple stateless server and relies on a more complex client implementation to manage caching. Requests between the client and server take place at the read/write request level. AFS[9] employs a persistent client-side cache that holds entire files and a token passing scheme to give write permission to one client at a time. Its successor CODA[3] takes this further to allow disconnected clients to continue functioning entirely from cache with changes propagated back to the server when connectivity is restored.

[1] `http://bitconjurer.org/BitTorrent/`
[2] `http://samba.anu.edu.au/rsync/`

Experience with CODA on mobile computers demonstrated[18] that optimistic write caching was an effective strategy as direct update conflicts were the exception rather than the rule.

Network filesystems by definition require a server to host all files and are administrated centrally. Some systems are designed around a cluster of servers while still exporting a single filesystem[7]. Others may create a single logical volume from a cluster of servers with a filesystem build above the logical volume[6]. In either approach the cluster could host a single participant in a larger distributed system.

3.2 Peer-to-peer filesystems

In contrast to client/server systems, peer-to-peer filesystems treat the entire set of participants as peers and distribute files across many different hosts. All participants are untrusted and unreliable, requiring a high level of replication to ensure that data is not lost. Replication like this works well in cluster applications[4, 6] that use commodity hardware with a relatively high rate of failure, but P2P systems must contend with a much greater failure rate and lower bandwidth as participants drop out without hardware failures[17].

P2P systems are attractive because they promise to harness the unused storage capacity of thousands of participants, offering cheap resources without central administration costs to those who require more storage, and possible remuneration to those with excess capacity. They also have the potential to be highly fault tolerant since data is widely distributed over a heterogeneous user base, though the communication cost to maintain this is quite high[2]. In addition, analysis of traffic over P2P file sharing systems suggests[8] that careful attention to locality can reduce bandwidth requirements considerably.

P2P filesystems such as Pasta[10] are typically layered over distributed hash tables (DHT) such as Pastry[14] or Chord[19] and the topology of the filesystem is largely determined by the DHT substrate. They achieve reliability through replication and widely distributed replica placement.

3.3 Serverless filesystems

The xFS[20] team defined a third category of distributed filesystems called serverless filesystems. xFS shares many goals with the current proposal, including a decentralized architecture with localized clusters of participants. Pangaea[15] bears some similarities as well, but it builds a network of trusted servers rather than a system of untrusted end-users.

4 Design overview

4.1 Global name space

Providing a global name space for all participants is essential to many of the goals of transparent distributed service. However, I propose a novel constraint to the way files are accessed. Normal file operations such as read and write are only available on files owned by the user or group that owns the file. Widespread read-only sharing of files is accomplished through a link operation that effectively copies the file for a user. The link operation may not actually result in a local replica being made, but it will provide a name with which the user can access the file normally. The system will index files by content (or a hash of the content) as well as the name, so it will maintain records of replicas with different names. Links may be made either to the content (with copy-on-write behaviour) or to the name (so changes to the original will be reflected in the link).

When a user attempts to access a file for which they own a name but no local copy, the system will retrieve a copy. The copy may come from any replica already existing on the system or through a swarm transfer (a la Bittorrent) from multiple hosts. The user's local copy will then be registered by content and available as a source for other users wishing to replicate it on their local hosts. With this scheme a popular file will become widely replicated and highly available while still being accessible through a

widely-known name (no explicit indirection to mirrors will be required) but not putting an excessive burden on the original owner. Also, everyone who accesses the file will become a source for further distribution.

This mechanism will give the system more information for making intelligent decisions about replication. Popular files that are not necessarily accessed often (such as many files in a typical Linux distribution that are always installed but rarely used) will be replicated enough to be widely available, but not as much as files that are actively read by many hosts. They will also be distinguishable to the system from files that are not widely used nor widely referenced, allowing the system to make more informed choices about replica placement.

4.2 Local storage

Another unique feature of my design is that participants provide their own storage space. This may be on a single machine, on many machines within an organization, or on hired space from a commercial storage pool. Unlike many P2P systems this system is not designed to make use of idle resources. It emphasizes reducing administration overhead, providing automatic replication, and allowing convenient, widespread distribution of files with a global namespace. It may also provide a useful model for the Xenoservers platform[5].

Virtually all content that a user/group creates will be hosted and replicated on storage owned by that user/group. Content that the user links to will also tend to be replicated locally (though if it is widely replicated and infrequently used it may not reside locally). A user's owned resources will be largely self-contained and a local machine or group of machines should function well when isolated from the rest of the network[18]. Replication of private content will happen on private storage, and will only escape into the wider system if it is published on a public part of the name space and linked to by another user. Even then, if the links do not persist it may not continue to live anywhere outside the user's private storage space.

Content that is linked to but not widely replicated will tend to be copied immediately, while widely replicated files may not by copied until they are actually used.

4.3 Participant roles

Different participants in the system may have different requirements. The system will be designed with a few primitive operations for transferring files between hosts, finding replicas of files by name or by content, finding available space usable by a given user, etc. Many of the decisions about when to transfer a file and when to access part of it from its current host, which hosts to transfer from, when to remove a replica, etc., will vary based on the characteristics of the particular machine and its relationship to the user. Many of these policy decisions will be best left to user customization, alternate implementations, or predefined profiles for different machine roles. A few examples include:

- Domain0 in a Xen[1] machine will want to host a replica of a Linux distribution and serve it (without additional replication) to other domains. Other domains will want to give high preference to replicas in the local domain0 and only replicate files from other machines.
- A laptop will want to replicate all content necessary to run the machine, and it will want to keep replicas of many current documents being actively modified on the user's desktop machine in anticipation of going offline.
- A user's home machine and office machine may want to mirror each other closely for some subtrees of the namespace and not for others. This remote replication may provide the primary backup for the user's email and work projects as well as making working at home convenient.

4.4 Replication

The decision to host a local replica of a file will depend on four factors: the existing degree of replication of that file in the system as a whole, the amount of demand for that file in the system as a whole, the demand for that file locally, and special considerations unique to the host's requirements as defined by its role. A file that is not used locally but is also not duplicated widely (meaning many duplicates or duplicates that are at distinct locations) may be replicated by a given participant where a file that is widely replicated already will not. Some host profiles may also replicate content more or less aggressively than others. Mobile computers will replicate more locally used content than a shared workstation with a permanent network connection, for example, while the latter may replicate more files that it doesn't actually use. The precise policies will likely be determined through experimentation, so the architecture will be designed to separate the policy decisions from the replication mechanism. Where policies may vary for different types of hosts, it is particularly important to design this with flexibility and the ability to change policies in an existing system.

Replication of read-only content enhances performance, particularly if participants are able to locate nearby (or local) instances of the files they seek, while synchronizing updates becomes more complicated with a high level of replication. Concurrent updates are relatively rare[18] and an optimistic strategy for resolving them will probably be suitable.

5 Timeline

I anticipate a few major stages in the progression of my PhD research:

1. Design and definition of requirements
   February 2004–June 2004
   During this time I will complete my userspace NFS implementation[13] and participate in the initial roll-out of the Xenoserver platform. This initial roll-out should be completed and some evaluation done by the May 2004 SOSP submission deadline so that a paper can be submitted describing the experience. As Xenoservers are a motivating application for my research, this experience will be a very useful source of requirements. I extend this period through June as I am planning to get married in June and do not anticipate starting a major new stage of my research until after that. In this time I will also do further research into related work.

2. Prototype implementation and refinement of design
   July 2004–December 2004
   A prototype implementation will likely be a loopback mounted userspace NFS server running over a local filesystem. This arrangement will allow a quick implementation, especially given my previous experience implementing a userspace NFS server.

3. Test deployment
   January 2005–March 2005
   The success of this project will be measured partly by the interaction of different types of participants, so a realistic deployment and testing will be a significant undertaking. Testing it in the context of Xenoservers will also be a goal of this stage.

4. Measurement and thesis planning
   April 2005–June 2005
   During this stage I plan to measure my system against other related work and identify the significant results, and from that I will outline the central arguments in my thesis.

5. Further measurement and write-up
   July 2005–March 2006
   On advice from others I allot a significant amount of time for writing up and performing additional tests.

6. Submit
   April 2006

References

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugabauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. *ACM SOSP*, October 2003.

[2] Charles Blake and Rodrigo Rodrigues. High Availability, Scalable Storage, Dynamic Peer Networks: Pick Two. *HOTOS IX*, May 2003.

[3] P. J. Braam. The Coda Distributed File System. *Linux Journal #50*, June 1998.

[4] G. Candea and A. Fox. Crash-only software. *HOTOS IX*, May 2003.

[5] K. Fraser, S. Hand, T. Harris, I. Leslie, and I. Pratt. The Xenoserver Computing Infrastructure. Tech Rep. UCAM-CL-TR-552, University of Cambridge, Computer Laboratory, January 2003.

[6] Svend Frlund, Arif Merchant, Yasushi Saito, Susan Spence and Alistair Veitch. FAB: enterprise systems on a shoestring. *HOTOS IX*, May 2003.

[7] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System. *ACM Symposium on Operating Systems Principles*, October 2003.

[8] K. Gummadi, R. Dunn, S. Saroiu, S. Gribble, H. Levy, and J. Zahorjan. Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload. *ACM Symposium on Operating Systems Principles*, October 2003.

[9] 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. *ACM Transactions on Computer Systems*. Feb 1988.

[10] Tim D. Moreton, Ian A. Pratt, Timothy L. Harris. Storage, Mutability and Naming in Pasta. *Proc. International Workshop on Peer-to-Peer Computing, Networking 2002*. May 2002.

[11] Brian Pawlowski, Chet Juszczak, Peter Staubach, Carl Smith, Diane Lebel, and David Hitz. NFS Version 3 Design and Implementation. *Proceedings of the Summer USENIX Conference*, pages 137–152, June 1994.

[12] Hans P. Reiser. Reiser4. http://www.namesys.com/v4/v4.html

[13] Russ Ross. First Year Report, January 2004.

[14] A. Rowstron, P. Druschel, Pastry: Scalable, Decentralized Object Location and Routing for Large Scale Peer-to-Peer Systems, *IFIP/ACM Conference on Distributed Systems Platforms*, Heidelberg (D), 2001.

[15] Yasushi Saito and Christos Karamanolis. Pangaea: a symbiotic wide-area file system. *ACM SIGOPS European Workshop*, September 2002.

[16] Douglas S. Santry, Michael J. Feeley, Norman C. Hutchinson, Alistair C. Veitch, Ross W. Carton and Jacob Or, Deciding When to Forget in the Elephant File System. *Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles*, December 12-15, 1999.

[17] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. *Proceedings of Multimedia Computing and Networking*, 2002.

[18] M. Satyanarayanan, J. J. Kistler, L. B. Mummert, M. R. Ebling, P. Kumar, and Q. Lu. Experience with Disconnected Operation in a Mobile Computing Environment, *USENIX Symp. on Mobile and Location Ind. Computing*, August 1993.

[19] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. *Technical Report TR-819, MIT*, March 2001.

[20] R. Wang and T. Anderson. xFS: A Wide Area Mass Storage File System. *Fourth Workshop on Workstation Operating Systems*, pages 71–78, October 1993.