Appeared in the Proceedings of the
Workshop on Management of Replicated Data,
November 1990, pages 20-25

Replication in Ficus Distributed File Systems*

Gerald J. Popek† Richard G. Guy Thomas W. Page, Jr.
John S. Heidemann

Department of Computer Science
University of California Los Angeles

Abstract

Ficus is a replicated general filing environment for UNIX *intended to scale to very large (nationwide) networks. The system employs an optimistic "one copy availability" model in which conflicting updates to the file system's directory information are automatically reconciled, while conflicting file updates are reliably detected and reported. The system architecture is based on a stackable layers methodology which permits a high degree of modularity and extensibility of file system services.*

This paper presents the motivations for replication and summarizes the case for optimistic concurrency control for large scale distributed file systems. It presents a brief description of the Ficus file system and concludes with a number of outstanding issues which must be addressed.

1 Overview

At UCLA, we have been working for some time on the design and development of an easily installed, replicated filing system that can be added to existing systems. Two fundamental characteristics of the work are its approach to modularity through a *stackable* file system architecture; and an *optimistic* view of update, in which any file or directory may be updated, so long as some copy is available. Conflicts are addressed when reconnection occurs. The system, which we call Ficus, is now operational. It has been constructed in a manner that can be added to an operating system that provides a faithful implementation of the vnode interface [10], even using conventional UNIX file systems such as UFS for file storage. Various optimizations are possible through extensions to the basic structure.

In the next section, we summarize the motivations for our replication work, providing a view of the goals of the research. The approach to modularity and optimistic operation are then described. Next, the actual Ficus project is summarized, and some observations about the utility of these approaches are offered. Finally, a number of other issues are raised.

2 Motivations for Replicated Files

In the general purpose and distributed filing environment, replication serves at least two important purposes: improving performance and increasing availability. Performance improvement occurs when access to data from a local storage medium is faster, cheaper, or less subject to congestion than from a remote store. In our view, a replication solution which does not take this consideration into

*This work was sponsored by DARPA under contract number F29601-87-C-0072.

†Author also associated with Locus Computing Corporation.

Appeared in the Proceedings of the
Workshop on Management of Replicated Data,
November 1990, pages 20-25

account will be limited in applicability. The availability argument, by contrast, is well known. It applies both in terms of a general network server, with replicated components such as mirrored disks, as well as at the workstation level, where each workstation may have copies of those critical items needed to permit isolated operation.

A suitable replication service must enhance availability in the face of communications outages, as well as system or storage failures. From our point of view, network partitioning is a fact of life. It occurs through a variety of means, even when there ostensibly are multiple communications paths. Examples abound: failure of the integrity of CSMA local networks due to a loose terminator, transmission jamming from errant transceivers, repeater or gateway failures, errors in routing protocols, or serious traffic congestion. Communications may be forcibly delayed until lower cost times of day, an effective failure. Organizational boundaries may limit communications, as in the airline industry between reservations systems. In addition, of course, one also has the expected communications outages. In all of these cases, local computing continues. The replication service should enhance this ability.

3 The Optimistic Model

One of the primary difficulties with replication services in a distributed environment concerns what to do about update. The issue is *mutual consistency*; keeping the multiple copies of an object consistent with one another. The usual view blocks copies from becoming inconsistent, by any of a variety of methods. Primary copy [1], majority consensus [16], weighted voting [4][1], and quorum consensus [9] are all variants of the same strategy; allow update to occur in at most one connected environment. Then propagate that update to other storage sites when communication is re-established.

Our view is different, motivated by two observations. First, the conventional strategies *decrease* availability for update. As the number of copies of an object is increased, the ability to update it tends to decrease. In a typical UNIX file system, for example, over 40% of the file opens are for update [3], and over 6% of directory access is for write [2], so this effect is significant. Second, while update traffic is significant, the frequency at which actual conflicting updates occur to data stored in a general purpose filing system (excluding directory information) is believed to be very low. The actual level of sharing is known to be low in general, and with the exception of databases, concurrent sharing is hardly measurable in many environments. It should be noted, however, that these "lack of ongoing sharing" statistics do not in general apply to directories [3, 2, 11, 12].

These facts suggest a much more optimistic approach to replication, if a feasible solution can be fashioned. Imagine an architecture in which update were allowed whenever a copy of the needed data were available. When multiple copies were reconnected, if any were out of date, the new version would be automatically propagated to make them current. In the rare event that conflicting updates had occurred to the data file in question, the problem would be detected, both versions saved, and the next higher level in the system (which might be the user) would be notified.

It is very important to consider carefully how directories are to be handled in this view, both because the directory system must maintain its integrity in the face of independent updates to itself so that the system can operate successfully; and because there is a higher likelihood that a given directory may be updated concurrently by different clients. Updates to directories (in fact, all supporting data structures as well) should be automatically reconciled by the system. In principle this step might appear easy. After all, one might argue that the only updates done to directories are record inserts and deletes. The proper reconciliation of two independently updated versions of a directory would be to take the union of all the entries, less those which had been deleted during partioned operation. In fact, however, the situation is rather more complex, due to a host of issues. These include the asynchronous nature of partitions and reconnections, the necessity to distinguish between deletion and creation of an entry, the necessity not

[1] The large number of variants to weighted voting published in the last decade are too numerous to list here.

Appeared in the Proceedings of the
Workshop on Management of Replicated Data,
November 1990, pages 20-25

to involve centralized algorithms, and to perform all necessary corrective actions during normal operation. The implementation of the suite of reconciliation algorithms in Ficus demonstrates that such a solution is indeed feasible, even in the face of these requirements.

4 An Approach to Modularity

It would be most attractive if a replication service such as we summarized above could be easily added to an existing filing system. In fact, this observation generalizes. A filing system that were structured in separable *layers*, analogous to protocol layers in UNIX System V Streams, would have many advantages. The concept is that of a stack of services, with the interface between all layers having an identical structure, so that one could assemble the services from an available set of building blocks. In Streams, one can even dynamically push a new protocol layer onto an executing stack of protocol functions.

One could then imagine an operating system's file system to be composed of a stack of services: extent management, data encryption, directory services, remote access facility, performance monitor, replication, etc. An important aspect of such an approach is the definition of the interface. It must be one that supports full function and allows high performance both when adjacent layers are in the same address space on a single machine, and when separated by network communication. We lay out a proposal for the stackable layers interface as part of Ficus.

5 The Ficus Project

Ficus [13] is a set of software packages that may be installed at the VFS layer of a file system, where Sun's Network File System [15] is typically connected. It is self contained, in that it makes no assumptions about the underlying filing storage system other than those which NFS makes. The system permits one to replicate files selectively within limits set by administrative control. That is, a collection of file volume replicas are set up at various storage sites for a given logical file subtree. A given file may be replicated at any subset of the sites hosting a volume replica.

Unlike NFS, Ficus guarantees name transparency among all of its volumes and replicas. This characteristic is achieved without a global replicated server such as found in AFS. The replicated information that is a necessity to achieve name transparency in a robust manner is merely represented in normal Ficus directories. In this manner, the system's standard directory reconciliation service is responsible for maintaining consistency among the necessary copies [7].

Two distributed algorithms are key to the Ficus architecture. The first faithfully detects update conflicts among data objects, and the second automatically reconciles directory conflicts to produce a correct integrated result. The conflict algorithm associates a version vector with each replica of each object and compares vectors to detect conflicts [14]. The reconciliation algorithm [8, 5], actually a family of related procedures, are two phase distributed algorithms (without coordinators) that guarantee all updates and deletions to a partially replicated directory structure (actually a general labeled graph structure) are seen by all copies, under minimum communications assumptions.

The Ficus implementation [6, 13] is divided into two stackable layers, "logical" and "physical". A given machine may or may not host either or both of these layers. The logical layer provides layers above with the abstraction of a single copy, highly available file; that is, the existence of multiple replicas is made transparent by the logical layer. It is responsible for replica selection, notifying physical replicas about the existence of updates, and managing reconciliation. The physical layer implements the abstraction of an individual replica of a replicated file. It uses whatever underlying storage service is available (such as a UNIX file system) to store persistent copies of files, pulls over file updates, and manages the extended set of attributes about each file, and each directory entry. When

Appeared in the Proceedings of the
Workshop on Management of Replicated Data,
November 1990, pages 20-25

the file is stored on the same site as the client, the logical and physical layers communicate via procedure calls (vnode operations) resulting in minimal overhead due to the layered architecture. When the physical storage site is remote, the logical and physical layers are separated by a communications channel. The initial Ficus implementation uses NFS to map vnode operations across address space boundaries. Figure 1 shows a file with two replicas, one of which is accessed via NFS.

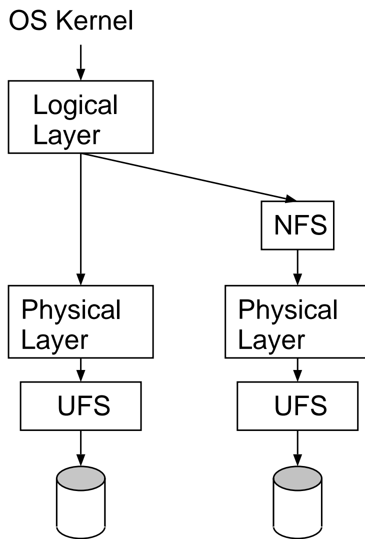

Figure 1: Ficus Stack of Replication Layers

The layered model provides a high degree of flexibility in the Ficus architecture. First, since the interface to each layer in the system is the standard vnode interface, NFS may be inserted between any pair of layers. In particular, using NFS above the logical layer makes a replicated file system available to any type of machine for which an NFS client implementation exists (e.g., a PC running MS-DOS), even though no implementation of Ficus exists for that system. Similarly, using NFS below the physical layer allows replicas to be stored on any type of machine that supports an NFS server interface (e.g., an IBM disk farm on a 3090 running MVS).

Second, the common layer interface means that additional file system functionality could be added to the stack transparently to all other modules. For example, a layer could be inserted anywhere in the stack which supports secure file storage, encrypting data as it is written and decrypting as it is read. We are in the process of implementing a measurement layer which records statistics about the traffic between any two modules in the stack. Thus stackable layers is also a methodology for extensible file systems.

6 Conclusions and Outstanding Issues

The modular replication work reported here seems like a promising component of distributed filing systems, especially those which are large or geographically dispersed. However, while pursuing this research, we encountered a number of other problems, and as a result have a somewhat different perspective in some cases. Here we raise a few: synchronization, consistency, and naming.

It is our position that the large scale distributed filing arena requires a different approach to concurrency control from that employed in distributed databases. In the database view, one guarantees that either all data objects are updated atomically as a unit or none are, in a manner that assures serializable schedules. By contrast, serializability and atomic transactions are not provided by today's local file systems and are not required, in general, in their large scale distributed descendants. We argue that one should not even apply transaction models to the collection of copies of an individual object, as that means update (and even, in some cases, reading) is impossible if any particular copy is missing. However, distributed file systems should provide the hooks whereby higher levels of consistency (up to and including serializability) can be provided to clients who require it.

Within some *region*, one may well wish some form of consistency. For example, on any given site, it may be important that the client experience monotonically increasing time, in the sense that the client never sees data older than data it has already seen, at least not without warning. One could of course insist on a "single system image" of replicated data in a very strong sense, as exhibited in IBM's Transparent Computing Facility. There, the

Appeared in the Proceedings of the
Workshop on Management of Replicated Data,
November 1990, pages 20-25

set of machines in the network cooperate to assure that all clients see synchronized data, including concurrent write sharing of file pages. However, the specific solutions employed there to achieve the single system image require global agreements and do not scale gracefully to very large networks. While such higher levels of consistency may not be feasible as a base level of service in a large distributed system, they may be quite feasible for a limited subset.

There are many other problems to be considered too. The directory reconciliation service that enables the optimistic strategy espoused earlier can be thought of as a general facility to manage loosely coordinated collections of records, where the semantics of the updates allowed to the records are restricted. Examples of such collections of records might include the UNIX password file or certain databases. How should such a service be made generally available?

Once a large filing system is installed, operating, and depended upon, it is increasingly unlikely that a "global reboot" to change protocols or make other globally visible alterations can be tolerated. This consideration puts a premium on decentralized architectures, self identifying protocols and versions, without global agreements.

A robust naming context system is also needed, which allows software and potentially replicated objects to be moved easily, without invalidating references (such as hypermedia links) in other objects. The name service must provide these transparency properties while at the same time being highly available, and hence replicated and mutually consistent. We are pursuing each of these outstanding issues in the context of Ficus.

Acknowledgements

Dieter Rothmeier and Wai Mak have played key roles in the implementation of the Ficus layers and reconciliation.

References

[1] P. A. Alsberg and J. D. Day. A principle for resilient sharing of distributed resources. In *Proceedings of the Second International Conference on Software Engineering*, pages 562–570, October 1976.

[2] Rick Floyd. Directory reference patterns in a UNIX environment. Technical Report TR-179, University of Rochester, August 1986.

[3] Rick Floyd. Short-term file reference patterns in a UNIX environment. Technical Report TR-177, University of Rochester, March 1986.

[4] D. K. Gifford. Weighted voting for replicated data. In *Proceedings of the Seventh Symposium on Operating Systems Principles*. ACM, December 1979.

[5] Richard G. Guy. *Ficus: A Very Large Scale Reliable Distributed File System*. Ph.D. dissertation, University of California, Los Angeles, 1990. In preparation.

[6] Richard G. Guy, John S. Heidemann, Wai Mak, Thomas W. Page, Jr., Gerald J. Popek, and Dieter Rothmeier. Implementation of the Ficus replicated file system. In *USENIX Conference Proceedings*, pages 63–71. USENIX, June 1990.

[7] Richard G. Guy, Thomas W. Page, Jr., John S. Heidemann, and Gerald J. Popek. Name transparency in very large scale distributed file systems. In *IEEE Workshop on Experimental Distributed Systems*, October 1990.

[8] Richard G. Guy and Gerald J. Popek. Reconciling partially replicated name spaces. Technical Report CSD-900010, University of California, Los Angeles, April 1990.

[9] Maurice Herlihy. A quorum-consensus replication method for abstract data types. *ACM Transactions on Computer Systems*, 4(1):32–53, February 1986.

[10] S. R. Kleiman. Vnodes: An architecture for multiple file system types in Sun UNIX. In *USENIX Conference Proceedings*, pages 238–247. USENIX, June 1986.

Appeared in the Proceedings of the
Workshop on Management of Replicated Data,
November 1990, pages 20-25

[11] Øivind Kure. Optimization of file migration in distributed systems. Technical Report UCB/CSD 88/413, Unviversity of California, Berkeley, April 1988.

[12] John K. Ousterhout, Hervé Da Costa, David Harrison, John A. Kunze, Mike Kupfer, and James G. Thompson. A trace-driven analysis of the UNIX 4.2 bsd file system. Technical Report UCB/CSD 85/230, UCB, 1985.

[13] Thomas W. Page, Jr., Gerald J. Popek, Richard G. Guy, and John S. Heidemann. The Ficus distributed file system: Replication via stackable layers. Technical Report CSD-900009, University of California, Los Angeles, April 1990.

[14] D. Stott Parker, Jr., Gerald Popek, Gerard Rudisin, Allen Stoughton, Bruce J. Walker, Evelyn Walton, Johanna M. Chow, David Edwards, Stephen Kiser, and Charles Kline. Detection of mutual inconsistency in distributed systems. *IEEE Transactions on Software Engineering*, 9(3):240–247, May 1983.

[15] Russel Sandberg, David Goldberg, Steve Kleiman, Dan Walsh, and Bob Lyon. Design and implementation of the Sun Network File System. In *USENIX Conference Proceedings*, pages 119–130. USENIX, June 1985.

[16] R. H. Thomas. A solution to the concurrency control problem for multiple copy databases. In *Proceedings of the 16th IEEE Computer Society International Conference*. IEEE, Spring 1978.