Appeared in the Proceedings of the
Second Workshop on Management of Replicated Data,
November 1992, pages 2-5

Primarily Disconnected Operation: Experiences with Ficus*

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

Department of Computer Science
University of California, Los Angeles

Abstract

Ficus is a flexible replication facility with optimistic concurrency control designed to span a wide range of scales and network environments. Support for partitioned operation is fundamental to the Ficus design but was not widely exercised in early Ficus use. This position paper reports recent experiences using Ficus in settings where some replicas are only occasionally connected to a network, and hence partitioned operation is the rule rather than the exception. We conclude that with some tuning, Ficus adapted quite well to primarily disconnected operation.

1 Introduction

In most work on replicated file systems, disconnected operation is assumed to be a temporary situation. Critical data is redundantly stored in order to improve the probability that at least one copy of the data remains accessible despite machine or communications network failures. Multiple copies of shared data may also be maintained on different sites in order to improve latency by increasing the probability that there is a copy of the data "near" where it will be accessed. However, the replicas are typically assumed to be *primarily connected*. Most of the time a replica can communicate with its siblings. In this environment, disconnected operation may be treated as a second order phenomenon, and the system optimized for normal operation in primarily connected mode.

There are many interesting uses of replicated data in a *primarily disconnected* fashion.

Home Use: Imagine having a machine at home which replicates most of the environment you use in the office. While you commute, your machine at home dials into the office, pulling down any changes made that day. By the time you get home, the machine there is up to date and you can work in the evening. When you are done for the night your machine dials in to the office and reconciles changes again, uploading all of your changes. While primarily disconnected, the machine at home is functioning almost as if it were constantly connected.

Portable Notebook Computers: You are about to take a trip, so you create replicas on your laptop of all files you are likely to need. While traveling on an airplane or sitting in your hotel room, you revise your next journal paper. If you get a chance during your trip, you connect your laptop to the Internet, either by modem or a direct network connection. Once connected, your changes are automatically reflected in the office replicas while the laptop also receives changes that occurred while you were gone.

Long Distance Networking: As is the case on the Ficus project, one of the members resides on Guam where the only network connection is by long-distance telephone. However, a machine there stores replicas of key file systems. Once a week, when phone tariffs are minimum, that machine connects to UCLA to exchange modifications.

The Ficus replicated file system was originally designed to span a wide range of scales and network environments. Not surprisingly, however, most of the experience with Ficus in its first two years of use has been in a local area network and over the Internet. In such an environment, while single-site partitions result from machine failures, network bandwidth and connectivity

*This work was sponsored by the Defense Advanced Research Projects Agency under contract N00174-91-C-0107. John Heidemann was also sponsored by a USENIX scholarship for the 1990-91 academic year, and Gerald Popek is affiliated with Locus Computing Corporation.

The authors can be reached at 3860 Boelter Hall, UCLA, Los Angeles, CA, 90024, or by electronic mail to ficus@cs.ucla.edu.

Appeared in the Proceedings of the
Second Workshop on Management of Replicated Data,
November 1992, pages 2-5

are quite different from the scenarios imagined above. Hence there may have been a tendency to optimize the implementation for the well-connected case.

We have recently begun employing Ficus in cases where disconnected operation predominates. This paper reports our experience in using Ficus in these cases.

2 Ficus Overview

Ficus is a distributed file system featuring optimistic replication. The default synchronization policy provides *single copy availability*; so long as any copy of a data item is accessible, it may be updated. Once a single replica has been updated, the system makes a best effort to notify all accessible replicas that a new version of the object exists. Those replicas then attempt to pull over the new version. Ficus guarantees *no lost update* semantics despite this optimistic concurrency control. Conflicting updates are guaranteed to be detected, allowing recovery after the fact.

Update *propagation* is the best-effort attempt to inform other replicas immediately of the presence of changes. In addition, a background process known as *reconciliation* runs on behalf of each replica after each reboot and periodically during normal operation. It compares all files and directories of a local volume replica with a corresponding remote replica, pulling over any missed updates and detecting any concurrent update conflicts. In the case of directories, most conflicts are repaired automatically by reconciliation, while for files, conflicting versions are marked as such and their owner notified. More information concerning the Ficus architecture and its reconciliation algorithms may be found in [2, 1].

3 Reconciliation Optimizations

Since each invocation of reconciliation is a unidirectional pull of information by one replica from another, some thought must be given to the reconciliation topology. Early Ficus designs called for all-pairs reconciliation, but the cost of $O(n^2)$ message exchanges proved too expensive. The alternative currently in use is to reconcile in a ring, each site pulling from the previous. To make this reconciliation topology resilient to failure, the ring skips sites which are inaccessible. In this model, primarily disconnected sites appear as spokes on the ring (Figure 1).

For sites which are primarily connected, the update propagation mechanism keeps most replicas consistent, with reconciliation serving to pick up the rare lost notifications. Primarily disconnected sites, however, rarely receive update notifications and rely almost entirely on reconciliation for consistency. They must reconcile with a better connected replica to pull down changes, and also must prompt a well-connected replica to reconcile with it.

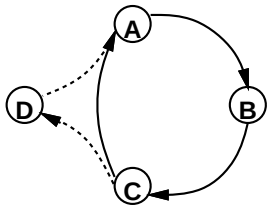

Figure 1: Sites A, B, and C are well connected. When site D becomes connected, it initiates reconciliation in the larger ring.

Our research group is actively using Ficus in the "home use" scenario. A user's environment is completely replicated on two workstations, one in the office, the other at home. These workstations are connected by modems to provide a mostly disconnected, slow Internet link. While the user travels between home and office, a connection is established and the replicas are reconciled. This reconciliation provides the user with the appearance of an identical working environment both at home and at work.

The Ficus architecture works well for basic home use. Because files are always stored on the local disk, access is consistently fast. Optimistic replication allows work to go on regardless of where the user is, as long as at least one replica can be contacted. Because a network connection is required only for reconciliation, the user's phone line is free most of the day; the home workstation does not require a second telephone line or a constant (possibly expensive) leased network connection.

Although the basic Ficus architecture adapted well to home use, there were several shortcomings in the initial implementation when applied to this new environment. Three performance problems that appeared are discussed below.

Ficus' original approach to detecting changes between two volume replicas was a first problem. The initial implementation detected updates by comparing the local and remote version vector of each file in a volume. This comparison required exchanging attributes of each file, resulting in sizable network traffic over slow links when comparing large numbers of files.

Only a fraction of the number of files of any given volume typically change between daily reconciliations.

Appeared in the Proceedings of the
Second Workshop on Management of Replicated Data,
November 1992, pages 2-5

This observation suggests that a significant amount of time can be saved by avoiding repeatedly re-examining files for changes. This observation motivated *time-based reconciliation*. A "last-update time" is associated with each file replica. Any change to the file at a particular replica is guaranteed to update this time. In addition, each pair of volume replicas record the time that the most recent completely successful reconciliation was last started. Since all known updates are guaranteed to be communicated by a successful reconciliation, files not updated since the last connection may be ignored. Time-based reconciliation therefore eliminates version vector comparison for all but recently changed files.

The second performance problem results from the approach Ficus uses to detect when a file has been removed in one partition and updated in another. This remove/update conflict has the potential for losing the update. To detect this conflict, the Ficus garbage collection algorithm [1] requires that file data propagate to all sites before it is removed. While this conservative strategy makes sense for important data files, many programs create temporary files which are quickly removed. Propagating these transient files to mostly disconnected sites wastes valuable bandwidth for data which will never be needed. We are currently investigating ways our garbage collection algorithm can be adjusted to avoid propagation of transient data.

The final performance problem we encountered stems from high latency typically encountered in mostly disconnected links. Reconciliation requires numerous interactions with both the local and remote replica. High latency network connections penalize frequent interaction. We expect that a combination of batching and caching could be applied to reduce the impact of this problem.

4 Observations

Our experience with Ficus in a home use setting has led us to several conclusions. We have been pleased with the low number of write conflicts encountered in practice. We have also found reconciliation a practical means of keeping consistent replicas connected by slow links alone. Finally, although first class replication works very well when files are required for long term use, our current implementation is a bit heavy weight for files only occasionally needed on different sites.

We have found surprisingly few update/update conflicts in our application of Ficus to home use. In three months of use we have seen only a handful of accidental conflicts. (We are currently modifying our software to log conflicts as they occur.)

Frequently-used, shared files are an exception to this pattern of low conflicts. Independent sources of update introduce potential conflicts, and frequent updates make conflicts probable. The only example of a frequently used, shared file in our environment is the UNIX fortune database. The simple semantics of this database allow automatic conflict recovery; we are currently investigating immediate and more general solutions to this problem. Since the vast majority of UNIX files in our environment are not shared databases, this problem has not significantly affected daily use.

We attribute this particularly low number of conflicts for non-database files to several factors. Most important is automatic conflict resolution of directory updates. Ficus automatically handles concurrent directory updates.

Another factor contributing to the rarity of conflicts is the effect of a *human write token*. Currently we replicate primarily system utilities and a user's personal data. Because updates to personal data come primarily from a single user, that person serves effectively as a "write token" for those files. By arranging a pattern of reconciliation corresponding to the presence of the user, the most recent data is almost always present when updates are made.

A second observation is that reconciliation need not be an an overly expensive method of promoting optimistic replica consistency. Updates typically affect only a small percentage of files. Since reconciliation overhead is proportional to the number of changes made since last reconciliation, cost of reconciliation is comparable to the benefits gained. In practice, current modems (V32.bis) result in 45 minute typical reconciliation times for a 63Mb home directory and about 50Mb of mostly static system files.

A final observation concerns the long-term cost of replication. Each site storing a Ficus replica must periodically communicate with other sites to participate in garbage collection. If these costs cannot be minimized, the need for long-term caching may require some sort of "second-class" replication, complicating the design. This issue is further discussed in the next section.

5 Related Work

There is at least one other research project focusing on the topic of deliberate disconnected operation described in the literature: CMU's Coda system [3]. Ficus and Coda share many of the same goals, and even some basic techniques such as optimistic replication. The primary difference in approach lies in the (AFS-inherited) Coda model of client-server filing support, versus the peer-to-peer Ficus model.

Appeared in the Proceedings of the
Second Workshop on Management of Replicated Data,
November 1992, pages 2-5

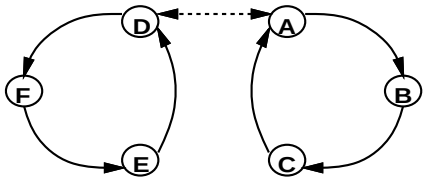

Figure 2: Two groups of sites are internally well connected, but they connect to each other only by a single, expensive link.

Coda combines client-side on-disk file caching with server file replication to support disconnected operation. When a client workstation is operating in "connected" mode, its cache manager maintains a set of recently accessed files on its local disk. The existence of a large cache allows the client to continue serving most file accesses out of the cache during disconnected operation. A third mode, "reintegration," is automatically entered when a client workstation is reconnected to a server. Reintegration ensures the mutual consistency of the client's cached files and the servers' copies. Files which have been concurrently updated are placed in a "co-volume" pending conflict resolution.

In Coda, client workstations are expressly "clients" and servers are expressly "servers". Client copies of files are, therefore, inherently second-class "replicas" in contrast to servers' first-class replicas. One subtle impact of this approach is that clients cannot directly share files: a server must be present to act as mediator. By contrast, Ficus' peer model makes no distinction between "classes" of replicas. Any two replica holders may always share data (and even propagate new versions) when they are physically able to communicate.

Consider, for example, expanding the one Ficus site on Guam to a cluster of sites (see Figure 2). Surely the cluster on Guam should be able to share data with each other even when not connected to the replicas on the mainland. Similarly, consider the case where several project members bring their laptops to a workshop. They should be able to inter-operate among themselves while disconnected from the replicas in the office. Both of these cases are feasible with the Ficus peer-to-peer model but not when clients possess only second-class replicas.

While Coda explicitly changes its state between connected, disconnected, and reintegrating, the Ficus model does not distinguish between connected and disconnected modes. Peers are dynamically connected to various degrees, determined in part by network bandwidth and latency. Thus, "connected" may be understood to imply high bandwidth and low latency, while "disconnected" implies epsilon bandwidth and practically infinite latency. A particular site might be strongly connected with respect to some volumes while disconnected from others. In practice, a number of interesting basic combinations of bandwidth and latency occur, which are further affected by sharing patterns.

Ficus' analogue of Coda's reintegration mode is reconciliation. However, in keeping with not distinguishing modes of operation, reconciliation is part of normal operation; normal user activity can proceed simultaneously with reconciliation.

6 Conclusions

Ficus was designed from the start to work in an environment in which partial operation (partial failure) was normal. This is the case in the Internet where some percentage of the sites are always inaccessible. However, initial experience with Ficus was primarily in the local network environment where all sites are accessible most of the time. This position paper examines the other extreme in which sites operate in primarily disconnected mode. In the future we plan to investigate the broad middle ground between the extremes.

Our conclusion is that the basic architecture of Ficus with optimistic replica management and reconciliation extends readily to the primarily disconnected environment. By modifying the ring-based reconciliation topology and using time to optimize reconciliations, it proved easy to adapt our implementation to an environment much different from where it was developed. Ficus is today in regular use in primarily disconnected mode.

References

[1] Richard G. Guy. *Ficus: A Very Large Scale Reliable Distributed File System.* Ph.D. dissertation, University of California, Los Angeles, June 1991. Also available as UCLA technical report CSD-910018.

[2] 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.

[3] James J. Kistler and Mahadev Satyanarayanan. Transparent disconnected operation for fault-tolerance. *ACM Operating Systems Review*, 25(1), January 1991.