Coda: A Highly Available File System for a Distributed Workstation Environment

M. Satyanarayanan
School of Computer Science
Carnegie Mellon University

Abstract

Coda is a file system for a large-scale distributed computing environment composed of Unix workstations. It provides resiliency to server and network failures through the use of two distinct but complementary mechanisms. One mechanism, *server replication*, involves storing copies of a file at multiple servers. The other mechanism, *disconnected operation*, is a mode of execution in which a caching site temporarily assumes the role of a replication site. Disconnected operation is particularly useful for supporting portable workstations. The design of Coda optimizes for availability and performance, but provides the highest degree of consistency attainable in the light of those objectives. Measurements from a prototype show that the performance cost of providing high availability in Coda is reasonable.

1. Introduction

The use of a location-transparent distributed file system to share data among workstations is common practice today [5]. A consequence of growing dependence upon such file systems is concern about the availability of data stored in them. *Coda* is a distributed file system that is resilient to failures that typically occur in a workstation environment. It provides high availability through the use of two distinct but complementary mechanisms, *server replication* and *disconnected operation.* Our goal in building Coda was to offer the maximum availability at the best performance, and to offer the highest degree of consistency attainable within those constraints. This paper is a brief overview of the key architectural features of Coda.

2. Base Architecture

Many aspects of Coda are inherited from the *Andrew file system* (AFS) [4], a large-scale distributed file system that has been in use at Carnegie Mellon University since late 1984. Like AFS, Coda makes a distinction between *servers,* which are relatively few in number, and *clients,* which are far more numerous. Servers are physically secure, run trusted system software and are monitored by operational staff. Clients may be modified in arbitrary ways by users, are physically dispersed, and may be left unattended or turned off for long periods of time. Although clients and servers both run Unix[1], users can execute programs only on clients. Clients and servers communicate using a remote procedure call mechanism that supports encryption and performs mutual authentication

Each client runs a process called *Venus* that handles remote file system requests using the local disk as a file cache. Files and directories are cached in their entirety by clients. Once a file or directory is cached from a server, the client receives a *callback* on it. A callback is a promise by the server that it will notify the client before allowing any other client to modify the object. Callbacks allow clients to maintain cache consistency while minimizing client-server interactions.

The consistency model of Coda is an approximation to that of AFS, which in turn is an approximation to the model offered by Unix. Unix allows multiple processes to read and write a file simultaneously, with modifications by one process being immediately visible to all other processes. AFS offers a model of consistency that is close enough to

[1]Unix is a trademark of AT&T.

Unix to be compatible with most applications, yet is highly scalable. Processes sharing a file at a single site see exact Unix semantics. Processes at different sites see modifications at the granularity of file open and close operations.

In the absence of failures the Coda model of consistency is identical to that of AFS. In the presence of failures, Coda strives to maximize availability. It denies access to a copy of data only if it is certain that the copy is inconsistent, and maintains this policy even in the presence of network partitions. Although this strategy could lead to conflicting updates, the infrequency of write-sharing in Unix environments [2], makes such conflicts unlikely. We have adopted the view, first suggested by Locus [6], that conflicting updates are acceptable if they are rare, promptly detected, and often easily repaired. A *version vector* [3] mechanism is used by Coda to detect inconsistencies. The high-availability mechanisms of Coda can be supressed for files that must remain consistent at all times.

3. Server Replication

The unit of replication in Coda is a *volume.* A volume is a collection of files that are stored on one server and form a partial subtree of the shared file name space. The set of servers that contain replicas of a volume is its *volume storage group (VSG).* Data that does not have to be highly available may be stored in non-replicated volumes. Coda also supports read-only replication of volumes, a facility inherited from AFS.

For each volume from which it has cached data, Venus keeps track of the subset of the VSG that is currently accessible. This subset is referred to as the *accessible volume storage group (AVSG),* and is identical to the VSG in the absence of failures. A server is deleted from the AVSG when an operation on it times out. It is added back to the AVSG when Venus is able to reestablish communication with the server. Depending on the failure mode of the system and on the locations of clients, different clients could have different AVSGs for a volume.

The replication strategy we use is a variant of the *read-one, write-all* approach. When a file is closed after modification, it is transferred to all members of the AVSG. This approach is simple to implement and maximizes the probability that every replication site has current data at all times. Server CPU load is minimized because the burden of data propagation is on the client rather than the server. This in turn improves scalability, since the server CPU is the bottleneck in many distributed file systems. The key drawback, latency of synchronous propagation, is addressed in Coda by the use of a parallel remote procedure call mechanism.

When servicing a cache miss, a client obtains data from one member of its AVSG called the *preferred server.* The preferred server can be chosen at random or on the basis of performance criteria such as physical proximity, server load, or server CPU power. Although data is transferred only from one server, the other servers are contacted to verify that the preferred server does indeed have the latest copy of the data. If this is not the case, the member of the AVSG with the latest copy is made the preferred site and the AVSG is notified that some of its members have stale replicas. Once data is cached at a client, a callback is established with the preferred server.

4. Disconnected Operation

Disconnected operation makes possible a compromise between total dependence on servers and complete autonomy of clients, and begins when no member of a VSG is accessible. Unlike server replication, it provides resiliency without the storage overhead of multiple replicas or the performance penalty of replication protocols. But it only provides access to data that was cached at the client at the start of disconnected operation. When disconnected operation ends, modified files and directories from disconnected volumes are propagated to the AVSG.

Coda clients view disconnected operation as a temporary state and revert to normal operation at the earliest opportunity. Transitions between normal and disconnected operation are normally transparent to a user. In normal operation a cache miss is transparent to a user, and only imposes a performance penalty. But in disconnnected

operation a miss impedes computation until normal operation is resumed or until the user aborts the corresponding file system call. Consequently it is important to avoid cache misses during disconnected operation.

During brief failures, the normal LRU caching policy of Venus may be adequate to avoid cache misses to a disconnected volume. This is most likely to be true if a user is editing or programming and has been engaged in this activity long enough to fill his cache with relevant files. But it is unlikely that a client could operate disconnected for an extended period of time without generating references to files that are not in the cache.

Coda therefore allows a user to specify a prioritized list of files and directories that Venus should strive to retain in the cache. Objects of the highest priority level are *sticky* and must be retained at all times. As long as the local disk is large enough to accomodate all sticky files and directories, the user is assured that he can always access them. Since it is often difficult to know exactly what file references are generated by a certain set of high-level user actions, Coda provides the ability for a user to bracket a sequence of high-level actions and for Venus to note the file references generated during these actions.

Disconnected operation can also be entered *voluntarily*, when a client is deliberately disconnected from the network. This might happen, for instance, if the client is a portable personal computer and its user wishes to take it with him on his travels. With a large disk cache the user can operate isolated from Coda servers for an extended period of time. The file name space he sees is unchanged, but he has to be careful to restrict his references to cached files and directories. From time to time, he may reconnect his client to the network, thereby propagating his modifications to Coda servers.

By providing the ability to move seamlessly between zones of normal and disconnected operation, Coda may be able to simplify the use of cordless technologies in distributed file systems. Cordless media such as cellular telephone, packet radio, or infra-red communication. typically have limitations such as short range, inability to operate inside buildings with steel frames, or line-of-sight constraints.

5. Status

At the time of writing this paper, in early 1989, our prototype of Coda is functional in many respects. The essential elements of server replication and disconnected operation have been implemented. One can sit down at a Coda client and execute Unix applications without recompilation or relinking. Execution continues transparently when contact is lost with a server due to a crash or network failure. In the absence of failures, using a Coda client feels no different from using an AFS client.

Preliminary measurements with the Andrew benchmark [1] shows that the degradation due to replication is relatively small. With one replica, Coda performs about 5% worse than with a non-replicated volume. With two and three replicas, it performs 9% and 11% worse respectively. With three replicas, Coda performs 28% worse than a local Unix file system. It should be emphasized that these measurements are from an untuned prototype, and we expect significant improvement as we refine Coda.

6. Conclusion

The increasing dependence of users on distributed file systems makes the availability of these systems a matter of concern. The two mechanisms of Coda that address this issue, server replication and disconnected operation, provide resiliency in the face of many failures while minimally impacting performance. Although Coda is far from maturity, our initial experience with it reflects favorably on its design.

References

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

[2] Ousterhout, J., Da Costa, H., Harrison, D., Kunze, J., Kupfer, M., Thompson, J.
A Trace-Driven Analysis of the Unix 4.2 BSD File System.
In *Proceedings of the 10th ACM Symposium on Operating System Principles, Orcas Island*. December, 1985.

[3] Parker, D.S. Jr., Popek, G.J., Rudisin, G., Stoughton, A., Walker, B.J., Walton, E., Chow, J.M., Edwards, D., Kiser, S., Kline, C.
Detection of Mutual Inconsistency in Distributed Systems.
IEEE Transactions on Software Engineering SE-9(3), May, 1983.

[4] Satyanarayanan, M., Howard, J.H., Nichols, D.N., Sidebotham, R.N., Spector, A.Z., West, M.J.
The ITC Distributed File System: Principles and Design.
In *Proceedings of the 10th ACM Symposium on Operating System Principles, Orcas Island*. December, 1985.

[5] Satyanarayanan, M.
A Survey of Distributed File Systems.
In Traub, J.F., Grosz, B., Lampson, B., Nilsson, N.J. (editors), *Annual Review of Computer Science*. Annual Reviews, Inc, 1989.
Also available as Technical Report CMU-CS-89-116, Department of Computer Science, Carnegie Mellon University, February, 1989.

[6] Walker, B., Popek, G., English, R., Kline, C., Thiel, G.
The LOCUS Distributed Operating System.
In *Proceedings of the 9th ACM Symposium on Operating System Principles, Bretton Woods*. October, 1983.

Table of Contents