Disconnected Operation in the Coda File System

JAMES J. KISTLER and M. SATYANARAYANAN
Carnegie Mellon University

Disconnected operation is a mode of operation that enables a client to continue accessing critical data during temporary failures of a shared data repository. An important, though not exclusive, application of disconnected operation is in supporting portable computers. In this paper, we show that disconnected operation is feasible, efficient and usable by describing its design and implementation in the Coda File System. The central idea behind our work is that *caching of data*, now widely used for performance, can also be exploited to improve *availability*.

Categories and Subject Descriptors: D.4.3 [**Operating Systems**]: File Systems Management:—*distributed file systems*; D.4.5 [**Operating Systems**]: Reliability—*fault tolerance*; D.4.8 [**Operating Systems**]: Performance—*measurements*

General Terms: Design, Experimentation, Measurement, Performance, Reliability

Additional Key Words and Phrases: Disconnected operation, hoarding, optimistic replication, reintegration, second-class replication, server emulation

1. INTRODUCTION

Every serious user of a distributed system has faced situations where critical work has been impeded by a remote failure. His frustration is particularly acute when his workstation is powerful enough to be used standalone, but has been configured to be dependent on remote resources. An important instance of such dependence is the use of data from a distributed file system.

Placing data in a distributed file system simplifies collaboration between users, and allows them to delegate the administration of that data. The growing popularity of distributed file systems such as NFS [16] and AFS [19]

This work was supported by the Defense Advanced Research Projects Agency (Avionics Lab, Wright Research and Development Center, Aeronautical Systems Division (AFSC), U.S. Air Force, Wright-Patterson AFB, Ohio, 45433-6543 under Contract F33615-90-C-1465, ARPA Order 7597), National Science Foundation (PYI Award and Grant ECD 8907068), IBM Corporation (Faculty Development Award, Graduate Fellowship, and Research Initiation Grant), Digital Equipment Corporation (External Research Project Grant), and Bellcore (Information Networking Research Grant).

Authors' address: School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213.

attests to the compelling nature of these considerations. Unfortunately, the users of these systems have to accept the fact that a remote failure at a critical juncture may seriously inconvenience them.

How can we improve this state of affairs? Ideally, we would like to enjoy the benefits of a shared data repository, but be able to continue critical work when that repository is inaccessible. We call the latter mode of operation *disconnected operation*, because it represents a temporary deviation from normal operation as a client of a shared repository.

In this paper we show that disconnected operation in a file system is indeed feasible, efficient and usable. The central idea behind our work is that *caching of data*, now widely used to improve performance, can also be exploited to enhance *availability*. We have implemented disconnected operation in the Coda File System at Carnegie Mellon University.

Our initial experience with Coda confirms the viability of disconnected operation. We have successfully operated disconnected for periods lasting one to two days. For a disconnection of this duration, the process of reconnecting and propagating changes typically takes about a minute. A local disk of 100MB has been adequate for us during these periods of disconnection. Trace-driven simulations indicate that a disk of about half that size should be adequate for disconnections lasting a typical workday.

2. DESIGN OVERVIEW

Coda is designed for an environment consisting of a large collection of untrusted Unix[1] clients and a much smaller number of trusted Unix file servers. The design is optimized for the access and sharing patterns typical of academic and research environments. It is specifically not intended for applications that exhibit highly concurrent, fine granularity data access.

Each Coda client has a local disk and can communicate with the servers over a high bandwidth network. At certain times, a client may be temporarily unable to communicate with some or all of the servers. This may be due to a server or network failure, or due to the detachment of a *portable client* from the network.

Clients view Coda as a single, location-transparent shared Unix file system. The Coda namespace is mapped to individual file servers at the granularity of subtrees called *volumes*. At each client, a *cache manager* (*Venus*) dynamically obtains and caches volume mappings.

Coda uses two distinct, but complementary, mechanisms to achieve high availability. The first mechanism, *server replication*, allows volumes to have read-write replicas at more than one server. The set of replication sites for a volume is its *volume storage group* (*VSG*). The subset of a VSG that is currently accessible is a client's *accessible VSG* (*AVSG*). The performance cost of server replication is kept low by caching on disks at clients and through the use of parallel access protocols. Venus uses a cache coherence protocol based on *callbacks* [9] to guarantee that an open file yields its latest

[1] Unix is a trademark of AT&T Bell Telephone Labs

copy in the AVSG. This guarantee is provided by servers notifying clients when their cached copies are no longer valid, each notification being referred to as a 'callback break'. Modifications in Coda are propagated in parallel to all AVSG sites, and eventually to missing VSG sites.

Disconnected operation, the second high availability mechanism used by Coda, takes effect when the AVSG becomes empty. While disconnected, Venus services file system requests by relying solely on the contents of its cache. Since cache misses cannot be serviced or masked, they appear as failures to application programs and users. When disconnection ends, Venus propagates modifications and reverts to server replication. Figure 1 depicts a typical scenario involving transitions between server replication and disconnected operation.

Earlier Coda papers [18, 19] have described server replication in depth. In contrast, this paper restricts its attention to disconnected operation. We discuss server replication only in those areas where its presence has significantly influenced our design for disconnected operation.

3. DESIGN RATIONALE

At a high level, two factors influenced our strategy for high availability. First, we wanted to use conventional, off-the-shelf hardware throughout our system. Second, we wished to preserve *transparency* by seamlessly integrating the high availability mechanisms of Coda into a normal Unix environment.

At a more detailed level, other considerations influenced our design. These include the need to *scale* gracefully, the advent of *portable workstations*, the very different *resource*, *integrity*, and *security* assumptions made about clients and servers, and the need to strike a balance between *availability* and *consistency*. We examine each of these issues in the following sections.

3.1 Scalability

Successful distributed systems tend to grow in size. Our experience with Coda's ancestor, AFS, had impressed upon us the need to prepare for growth *a priori*, rather than treating it as an afterthought [17]. We brought this experience to bear upon Coda in two ways. First, we adopted certain mechanisms that enhance scalability. Second, we drew upon a set of general principles to guide our design choices.

An example of a mechanism we adopted for scalability is callback-based cache coherence. Another such mechanism *whole-file caching*, offers the added advantage of a much simpler failure model: a cache miss can only occur on an open, never on a read, write, seek, or close. This, in turn, substantially simplifies the implementation of disconnected operation. A partial-file caching scheme such as that of AFS-4 [22], Echo [8] or MFS [1] would have complicated our implementation and made disconnected operation less transparent.

A scalability principle that has had considerable influence on our design is the *placing of functionality on clients* rather than servers. Only if integrity or

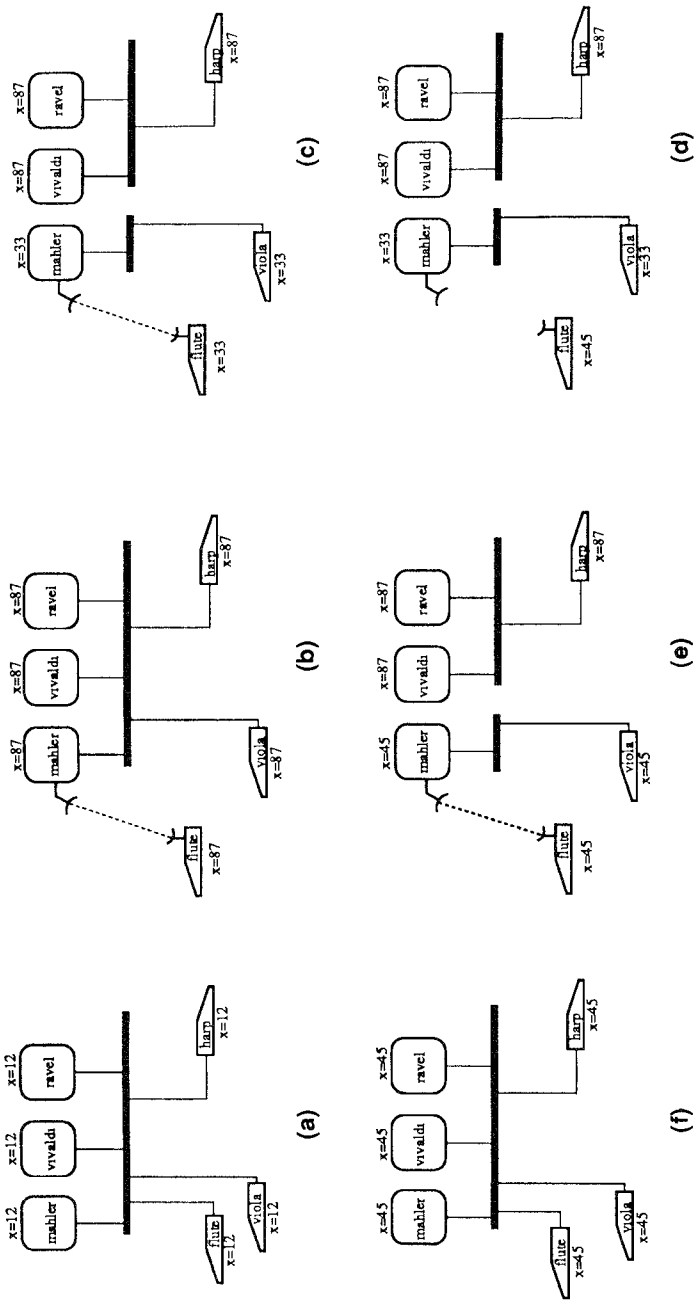

Fig. 1. How disconnected operation relates to server replication Three servers (*mahler*, *vwaldi*, and *ravel*) have replicas of the volume containing file *x*. This file is potentially of interest to users at three clients (*flute*, *viola*, and *harp*). *Flute* is capable of wireless communication (indicated by a dotted line) as well as regular network communication. Proceeding clockwise, the steps above show the value of *x* seen by each node as the connectivity of the system changes. Note that in step (d), *flute* is operating disconnected

security would have been compromised have we violated this principle. Another scalability principle we have adopted is the *avoidance of system-wide rapid change*. Consequently, we have rejected strategies that require election or agreement by large numbers of nodes. For example, we have avoided algorithms such as that used in Locus [23] that depend on nodes achieving consensus on the current partition state of the network.

3.2 Portable Workstations

Powerful, lightweight and compact laptop computers are commonplace today. It is instructive to observe how a person with data in a shared file system uses such a machine. Typically, he identifies files of interest and downloads them from the shared file system into the local name space for use while isolated. When he returns, he copies modified files back into the shared file system. Such a user is effectively performing manual caching, with write-back upon reconnection!

Early in the design of Coda we realized that disconnected operation could substantially simplify the use of portable clients. Users would not have to use a different name space while isolated, nor would they have to manually propagate changes upon reconnection. Thus portable machines are a champion application for disconnected operation.

The use of portable machines also gave us another insight. The fact that people are able to operate for extended periods in isolation indicates that they are quite good at predicting their future file access needs. This, in turn, suggests that it is reasonable to seek user assistance in augmenting the cache management policy for disconnected operation.

Functionally, *involuntary* disconnections caused by failures are no different from *voluntary* disconnections caused by unplugging portable computers. Hence Coda provides a single mechanism to cope with all disconnections. Of course, there may be qualitative differences: user expectations as well as the extent of user cooperation are likely to be different in the two cases.

3.3 First- vs. Second-Class Replication

If disconnected operation is feasible, why is server replication needed at all? The answer to this question depends critically on the very different assumptions made about clients and servers in Coda.

Clients are like appliances: they can be turned off at will and may be unattended for long periods of time. They have limited disk storage capacity, their software and hardware may be tampered with, and their owners may not be diligent about backing up the local disks. Servers are like public utilities: they have much greater disk capacity, they are physically secure, and they are carefully monitored and administered by professional staff.

It is therefore appropriate to distinguish between *first-class replicas* on servers, and *second-class replicas* (i.e., cache copies) on clients. First-class replicas are of higher quality: they are more persistent, widely known, secure, available, complete and accurate. Second-class replicas, in contrast, are inferior along all these dimensions. Only by periodic revalidation with respect to a first-class replica can a second-class replica be useful.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.