CITI Technical Report 95–11

Communications and Consistency in Mobile File Systems

P. Honeyman
L.B. Huston

Center for Information Technology Integration
University of Michigan
Ann Arbor

ABSTRACT

To overcome availability, latency, bandwidth, and cost barriers of mobile networks, mobile clients of distributed file systems switch between connected and disconnected modes of operation. Lying between these are modes of operation that refine the consistency semantics of cached files, allowing a mobile client to select a mode appropriate for the the prevailing network conditions. Clients can take advantage of network opportunities unsuitable for connected operation, obtaining improved performance, more effective sharing, and more stringent consistency guarantees as a result.

This article will appear in *IEEE Personal Communications*, special issue on mobile computing (December 1995).

October 5, 1995

Center for Information Technology Integration
University of Michigan
519 West William Street
Ann Arbor, MI 48103-4943

Communications and Consistency in Mobile File Systems

P. Honeyman
L.B. Huston

Center for Information Technology Integration
University of Michigan
Ann Arbor

1. Introduction

Today's mobile computing technology lets nomadic users operate computers large and small from their offices and homes and places in between. However, mobile users are often frustrated by the quality of distributed services available to them. In this article, we describe ways to extend the advantages of distributed file systems to users of mobile computers.

File systems are fundamental *middleware*, gluing applications to the operating system. A file system that has the same appearance and behavior, whether at the office, home, or in transit, can support applications without requiring that they be modified to accommodate mobility. This important aspect of distributed systems — *location transparency* — is lost if files have different names, are found in different places on different computers, or have to be moved by hand before they can be used. Our goal is to extend the location transparency of distributed file systems to mobile computers.

Distributed systems are often structured with a few servers and many clients; this is also the most common architecture of distributed file systems. Such systems scale well if the workload can be shifted from servers to clients. Many distributed file systems advance this scaling principle by caching information in client memory and disks. Experience has shown that client caching is a powerful technique for distributed file systems, providing for file systems that perform and scale well, span multiple large enterprises, and offer global access through a shared name space [1].

Client caching is a form of replication; a file system that relies on caching must employ a mechanism to ensure that replication sites remain in agreement about the contents of replicated files. The cache consistency guarantees offered by a file system are a critical factor in determining the network requirements of the system. These guarantees determine the semantics of sharing and the types of guarantees that users and applications can expect from file systems. But maintaining consistency requires network communications of various degrees.

Users of mobile computers encounter enormous variation in network availability. At one extreme, mobile users can have access to high-speed, low latency networks, comparable to office networks, while at the other, they may have no network access at all. In between are networks of various sorts, some with high costs, others with low bandwidth, *etc.* Location transparency is sacrificed if adverse characteristics of the network — excessive latency, insufficient bandwidth, or network outages — delay or disrupt file operations.

Today, the *lingua franca* of mobile networking starts with the letter V, as in V.34, V.42*bis*, *etc.* File systems designed for megabits per second of bandwidth and milliseconds of latency misbehave when presented with tens of kilobits per second of bandwidth and dozens to hundreds of milliseconds of latency. As a result, individual file operations may be delayed or fail to complete because of congestion collapse in the transport layer.

It would be an understatement to say that these factors adversely influence usability and user acceptance of mobile file systems. So, the first step toward supporting mobile access to file systems is to provide a high performance transport layer, one that can accommodate the bandwidth and latency limitations common in mobile networks [2, 3]. Beyond this basic step, research in mobile file systems has focused on providing access to a distributed file system when networks are unavailable or so unreliable as to be unusable.

A most successful approach to network partition has been *disconnected operation* [4, 5, 6, 7], which allows the use of cached files and directories during periods of network unavailability. A disconnected client keeps a log of local modifications to the file system. When convenient, logged operations are replayed over the network.

Disconnected operation shows great promise in extending location transparency to the most hostile networking environments, where network connectivity is absent altogether. However, disconnected operation fails to take advantage of networking opportunities that arise, yet are inappropriate for fully connected operation because of cost or bandwidth considerations. By fine tuning the semantics of sharing associated with distributed file systems, we can provide continuous access to the common name space, with good performance and natural semantics. Our approach is to relax cache consistency guarantees to match network cost, bandwidth, and availability.

2. Sources of network traffic

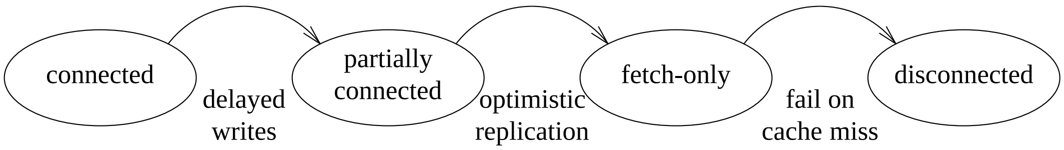
In this section, we examine the demands that a distributed file system places on network communications. Some communications are initiated by the client, the rest by servers. While both sorts can be provided on a network that is continuously available, an intermittent or costly network may interfere with server-initiated communications. For example, the server may be unable to locate a mobile client, or a mobile user may prohibit costly incoming calls.

Network communication is necessary to convey updates from the client to the server, to keep cached data consistent with other clients and servers, to fetch data missing from the client cache, and to service informational or maintenance requests. A mobile client that provides all of these operates in connected mode. Eliminating these services places a client in disconnected mode.

In the next few sections, we address the sources of network communications individually and describe techniques for eliminating them:

- Delayed writes reduce the network traffic caused by updates.
- Optimistic replication eliminates cache consistency traffic.
- Forcing references to uncached files to fail eliminates network fetches.

After examining ways to reduce or eliminate network traffic, we describe a sequence of modes of operation that apply these techniques progressively, as depicted below.



Mobile file system modes of operation. This figure shows the modes of operation as cache management operations are modified to avoid network communications. In connected mode, the file system offers the usual semantics, but places the greatest demands on the network. In disconnected mode, many of the semantic guarantees offered by the file system are eliminated, but so are all network requirements. In between are fine-grained modes of operation that trade communications for cache consistency.

2.1. Delayed writes reduce update traffic

In normal operation, applications use a file system to read and write files and directories. When a client writes to the file system, it is vital to propagate those changes to servers, as this promotes sharing and shifts responsibility for permanence of data from users to administrators. But low-speed networks can force applications to wait while synchronous updates are transmitted to servers. This kind of delay can frustrate interactive users.

Deferred writes make this latency transparent to users. Deferred operations must be saved on the mobile computer's disk to prevent loss of data if the system crashes (although vulnerability to catastrophic failure remains a potential). In our experience, laptop disks are as reliable as server disks; still, we try to propagate our updates within a day or so to coincide with server backup schedules.

Delaying writes offers two benefits. First, it unshackles file system performance from network performance so that operation latencies depend on the characteristics of local resources rather than on network characteristics. This improves file system performance when bandwidth is low or latency is high and removes a source of unpredictability from overall system performance. Second, asynchronous update defers the associated network traffic, so that work can continue when the network is unavailable.

Asynchronous update improves performance, but introduces the potential for concurrent updates by clients out of view of one another. Resolving these overlapping updates requires semantic knowledge specific to the applications in conflict [8, 9]. Delaying writes also prevents other clients from sharing new data right away.

One way to address the sharing problem is to inform the server of pending changes; when another client tries to access the new data, the server demands the deferred update. This scheme has shown some success in reducing operation latency and improving server performance in the Echo [10, 11, 12] and Sprite [13] file systems. In the mobile environment, though, the server's demand for updates might cause a well-connected client to inherit the penalties of slow or high latency networks as files are conveyed to the server from a poorly-connected client. Furthermore, in the face of potential network partitions the demand might be impossible for mobile clients to honor.

2.2. Optimistic replication eliminates cache consistency traffic

An object in the cache that is no longer the most current version is called *stale* (as opposed to *fresh*). To promote sharing, distributed file systems use network protocols to avoid the use of stale data. In a polling protocol, the client queries the server about the timeliness of cached data. NFS [14] uses a polling protocol. In a token protocol, the client calls the server to obtain tokens and the server calls the client to revoke tokens. In AFS [15], these tokens are called *callbacks*.

The difference between polling and token protocols is significant for a networked mobile client. Polling does not require continuous network availability, but generates considerable network traffic. Tokens use less network bandwidth, but require that the server be able to initiate communications. For mobile systems, this is often tantamount to a requirement of continuous network availability, as the server may have no way to locate a mobile client. In either type of protocol, network cost and availability may make it inconvenient or impossible to guarantee freshness of the client cache.

To provide for uninterrupted operation, some systems allow applications to use data that can not be guaranteed to be the most current at the time of use. These systems rely on *optimistic replication* [16] to justify the decision to use files under these circumstances. This optimism is based on the assumption that files that are modified are rarely shared; the incidence of write-sharing in UNIX and other operating systems that support distributed computing has long been observed to be low [17, 18], and in file systems that support mobile operations even lower [19].

Removing the network traffic associated with cache validation offers other benefits to clients of a distributed file system. First, it allows clients to access cached data when no networks are available. Second, clients do not incur network latencies associated with checking cache validity, which improves read performance. These benefits are offset by the potential for the use of stale information.

2.3. Aborting cache misses eliminates fetch requests

When an application attempts to use uncached data, the normal operation of the cache manager is to fetch a fresh copy from the server. The data might be file contents, access control lists, or name space information. Resolving cache misses allows the user to continue operating when necessary files are missing from the cache.

Occasional cache misses are a fact of life: except under extraordinary circumstances, references are eventually made to uncached data. Yet it is not always appropriate for the mobile client to fetch the missing data. For example, the volume of data being sought may be impractically large for some network conditions, or there may be no network available at the time of the request. Under these circumstances, an alternative to fetching the file is to abort the request and return a failure indication such as ''network is down'' to the application, which can then invoke its specific recovery procedures. Some applications are not prepared to recover from such errors, so it is worthwhile to assure that critical files are cached.

2.4. Miscellaneous operations

The remainder of the network communications used by file systems are in response to operations performed beyond the file system interface, such as adjusting access control lists or setting quota limits. These tend to be initiated by the client, although remote monitoring may originate externally. The specifics of these operations are file system dependent and can vary significantly among distributed file systems; it is difficult to generalize about their behavior.

3. Matching network demands with modes of operation

To adapt to the remote environment, the client must control network demands. The principal sources of network traffic exist to offer some guarantees about the validity of the cache. Synchronous writes guarantee that others see the same data as the writer. Cache consistency ensures that cached files are up-to-date. Fetching files missing from the cache ensures that files available to others are also available locally.

Hostile network conditions make it expensive or impossible to offer all of these guarantees. Our goal is to provide mechanisms that allow clients to adjust file system operation to match network availability. This allows the user to trade consistency guarantees for performance or cost improvements. Two of these modes — *connected* and *disconnected* — are well known in mobile file systems. *Partially connected* and *fetch-only* modes combine techniques for reducing the file system's dependency on network access, offering finer control over the file system's semantics.

The following table lists the modes of operation along with the methods they use to reduce or eliminate network traffic, their network requirements, and some network environments appropriate for their use. Each mode of operation balances network availability against the expense of maintaining consistency guarantees. These modes form a progression of decreasing consistency guarantees along with decreasing network requirements.

	MODES OF OPERATION			
	connected	partially connected	fetch-only	disconnected
method	normal operation	delayed writes	optimistic replication	abort on cache miss
network requirements	continuous, high bandwidth	continuous	on-demand	none
where	office	local dialup or PCS	ISDN, cellular, or long distance	anywhere

Mobile file system modes of operation and their roles. This table shows four modes of operation of a mobile file system and suggests the environment in which each is appropriate. Network economies are realized by weakening the sharing semantics using the methods shown.

3.1. Connected operation

Connected operation is the normal mode for file system clients that share a high-speed network with file servers. Changes to files and directories are propagated synchronously and stringent cache consistency is maintained.

Connected operation provides for consistent access to cached files but collapses in adverse networks. Synchronous file operations on a high latency or low bandwidth network translates into bothersome delays and user frustration when the application is interactive. Network partition renders cached files utterly unusable lest consistency requirements mandated by the semantics of connected operation be violated.

3.2. Partially connected operation

Partially connected mode augments connected operation by deferring writes. Although mutating operations are propagated asynchronously, native cache consistency protocols continue to be used when reading data. This provides confidence that any data read are the most recent seen by servers, with one exception: local modifications that have not yet been propagated are seen by the partially connected client.

Because data are not propagated as modifications occur, other users do not see updates immediately. If another user modifies a file that has pending writes, a conflict arises over which version of the file is current, requiring error recovery when the update is eventually propagated. The longer deferred updates sit in the cache, the more likely it is that this type of conflict will occur. On the other hand, by moving network dependencies out of the critical path, partially connected operation can significantly increase the speed at which mutating operations are performed [20].

Partially connected operation requires continuous network availability to support token-based cache consistency protocols. This may be too expensive for users who pay for network connect time, or impossible in places that don't offer network access. Partially connected mode is especially appropriate for low bandwidth or high latency networks, such as dialup access from home. PCS systems or low-speed wireless networks such as Metricom are a good match for partially connected mode.

3.3. Fetch-only operation

Fetch-only mode relies heavily on optimistic assumptions about cache consistency while also using delayed writes. Here, the cache manager assumes all cached data are valid, using the network solely to satisfy read misses.

Fetch-only mode provides the same performance benefits and potential for conflict as partially connected mode. Network demands are further reduced because no cache consistency protocol is used. The fetch-only client does not need continuous network availability, giving the mobile user the freedom to set up and tear down the network when convenient.

Fetch-only mode is attractive when the network has an associated charge for connect time, *e.g.*, over a cellular modem or ISDN. In this mode, the network is used as a last resort for the continued operation of applications.

3.4. Disconnected operation

Disconnected operation, which uses delayed writes, no cache consistency, and does not resolve cache misses, offers the weakest guarantees but requires no network access at all. If an application references uncached data, an error is returned. Disconnected operation is appropriate when no network is available or the network is too expensive to use.

Several systems that implement disconnected operation have been described in the research literature [4, 5, 6, 7, 19, 21]. Commercial products that keep mobile file systems synchronized with a workstation are also emerging, such as AirSoft's AirAccess, Relay Technology's AnyPlace, Traveling Software's LapLink, and IBM's Mobile FileSync.

4. Discussion

We have implemented the modes of operation described here in a modified AFS cache manager and use them daily. Portions of this article were written on office workstations connected to Ethernet, some on a home computer over ISDN, and some (most) on a subnotebook at a coffee shop on Main St. The convenience of having one and only one file system, whether at home, office, conference, or in transit, can not be overemphasized. There are no copies to compare, versions to merge, or removable media to tend. File system performance is predictable and consistent.

While most locations offer some network access, not all do, *e.g.*, the coffee shop mandates disconnected operation (at this time). Local cellular charges are low enough that it is feasible to employ fetch-only mode from network-limited locales, but the hardware associated with this sort of hookup is rather clumsy. Changes in radio spectrum allocation and packaging will soon combine to make access to wireless networks ubiquitous. In that environment, the ability to fine tune the file system to accommodate cost, speed, and latency constraints will be increasingly valuable.

The principal feature of the tuning process is to recognize the degree of network access required by the file system's consistency requirements, and to relax the requirements to match the circumstances that prevail. While most of the network traffic in an ordinary client arises from fetch and store operations issued by applications, this observation misses an important point: our goal is not to minimize network traffic, it is to adapt to adverse network conditions. Beyond the stark choice of a high-speed office network or no network at all are many opportunities for low speed or intermittent network communications. A fine-grained approach to managing file system semantics lets us take advantage of these intermediate-quality networks to extend the usability of our mobile computers and enhance personal productivity.

Acknowledgements

We thank Warren Widmayer and Rob Malan for their penetrating insights, and Andy Adamson, Dan Hyde, and Denise Miller for their careful reading.

References

1. 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* **6**(1), pp. 51–81 (February, 1988).
2. D. Bachmann, P. Honeyman, and L.B. Huston, ‘‘The Rx Hex,’’ pp. 66–74 in *Proceedings of the First IEEE International Workshop on Services in Distributed and Networked Environments*, Prague (June, 1994).
3. Lily B. Mummert, Maria R. Ebling, and M. Satyanarayanan, ‘‘Exploiting Weak Connectivity for Mobile File Access,’’ in *Proceedings of the 15th ACM Symposium on Operating System Principles*, Copper Mountain (December, 1995).
4. J.J. Kistler and M. Satyanarayanan, ‘‘Disconnected Operation in the Coda File System,’’ *ACM Transactions of Computer Systems* **10**(1), pp. 3–25 (February, 1992).
5. J.S. Heidemann, T.W. Page, R.G. Guy, and G.J. Popek, ‘‘Primarily Disconnected Operation: Experiences with Ficus,’’ pp. 2–5 in *Proceedings of the Second IEEE Workshop on the Management of Replicated Data*, Monterey (November, 1992).
6. L.B. Huston and P. Honeyman, ‘‘Disconnected Operation for AFS,’’ pp. 1–10 in *Proceedings of the USENIX Symposium on Mobile and Location-Independent Computing*, Cambridge (August, 1993).
7. Dorota M. Huizinga and Ken A. Heflinger, ‘‘Experience with Connected and Disconnected Operation of Portable Notebook Computers in Distributed Systems,’’ pp. 119–123 in *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz (December, 1994).
8. P. Kumar and M. Satyanarayanan, ‘‘Supporting Application-Specific Resolution in an Optimistically Replicated File System,’’ pp. 66–70 in *Proceedings of the Fourth Workshop on Workstation Operating Systems*, Napa (October 1993).
9. Peter L. Reiher, John S. Heidemann, David Ratner, Gregory Skinner, and Gerald J. Popek, ‘‘Resolving File Conflicts in the Ficus File System,’’ pp. 183–195 in *Proceedings of the Summer USENIX Conference*, Boston (June 1994).
10. T. Mann, A. Birrell, A. Hisgen, C. Jerian, and G. Swart, ‘‘A Coherent Distributed File Cache with Directory Write-behind,’’ SRC Research Report 103, Digital Equipment Corporation (June, 1993).
11. A. Birrell, A. Hisgen, C. Jerian, T. Mann, and G. Swart, ‘‘The Echo Distributed File System,’’ SRC Research Report 111, Digital Equipment Corporation (September, 1993).
12. G. Swart, A. Birrell, A. Hisgen, and T. Mann, ‘‘Availability in the Echo File System,’’ SRC Research Report 112, Digital Equipment Corporation (September, 1993).
13. Michael N. Nelson, Brent B. Welch, and John K. Ousterhout, ‘‘Caching in the Sprite Network File System,’’ *ACM Transactions on Computer Systems* **6**(1), pp. 134–154 (February, 1988).
14. B. Callaghan, B. Pawlowski, and P. Staubach, ‘‘NFS Version 3 Protocol Specification,’’ RFC 1813, Network Information Center, SRI International, Menlo Park, CA (June, 1995).
15. John H. Howard, ‘‘An Overview of the Andrew File System,’’ pp. 23–36 in *Proceedings of the Winter USENIX Conference*, Dallas (February, 1988).
16. S.B. Davidson, H. Garcia-Molina, and D. Skeen, ‘‘Consistency in partitioned networks,’’ *ACM Computing Surveys* **17**(3), pp. 341–370 (September, 1985).

17. 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,’’ pp. 15–24 in *Proceedings of the 10th ACM Symposium on Operating System Principles*, Orcas Island (December, 1985).

18. Mary G. Baker, John H. Hartman, Michael D. Kupfer, Ken W. Shirriff, and John K. Ousterhout, ‘‘Measurements of a Distributed File System,’’ pp. 198–212 in *Proceedings of the 13th ACM Symposium on Operating Systems Principles,* Asilomar (October, 1991).

19. M. Satyanarayanan, James J. Kistler, Lily B. Mummert, Maria R. Ebling, Pumeet Kumar, and Qi Lu, ‘‘Experience with Disconnected Operation in a Mobile Computing Environment,’’ pp. 11–28 in *Proceedings of the 1993 USENIX Symposium on Mobile and Location-Independent Computing*, Cambridge (August, 1993).

20. L.B. Huston and P. Honeyman, ‘‘Partially Connected Operation,’’ pp. 91–97 in *Proceedings of the Second USENIX Symposium on Mobile and Location-Independent Computing*, Ann Arbor (April, 1995).

21. H.-Y. Chang, Frank Novak, Carl Tait, and Peter Hortensius, ‘‘SFS: A Universal File System Cache for Disconnected FS Operations,’’ pp. 349–352 in *Proceedings of Joint Conference on Information Sciences*, Pinehurst, North Carolina (November, 1994).