Rumor: Mobile Data Access Through Optimistic Peer-to-Peer Replication

Richard Guy[1], Peter Reiher[1], David Ratner[2], Michial Gunter[3], Wilkie Ma[1], and Gerald Popek[4]

[1] University of California, Los Angeles, Los Angeles, CA 90095-1596, USA
`http://fmg-www.cs.ucla.edu/rumor98/replication.html`
`rumor-report@fmg.cs.ucla.edu`
[2] currently with Software.com
[3] currently with Silicon Graphics, Inc.
[4] also affiliated with PLATINUM *technology, inc.*)

Abstract. [1] Rumor is an optimistically replicated file system designed for use in mobile computers. Rumor uses a peer model that allows opportunistic update propagation among any sites replicating files. The paper outlines basic characteristics of replication systems for mobile computers, describes the design and implementation of the Rumor file system, and presents performance data for Rumor. The research described demonstrates the feasibility of using peer optimistic replication to support mobile computing.

1 Introduction

Mobile computers typically suffer from weaker connectivity than that enjoyed by wired machines. Latencies are significantly higher, bandwidth is limited, power conservation requirements discourage communication, some communications media cost money to use, and long-duration disconnections are the norm. In this context, data management techniques that dramatically reduce the need for continuous connectivity are highly desirable.

One such potential class of solutions is data replication, in which copies of data are placed at various hosts in the overall network, generally 'near' to users and often local to users. In the extreme, a data replica is stored on each mobile computer that desires to access (read or write) that data, so all user data access is local. Ideally, all replicas of a data item should have the same value at all times, and it is the responsibility of the replication system to maintain consistency in the face of updates. Specific goals for a replication system often include improved reliability, availability, data autonomy, host and network traffic load balancing, and data access performance.

[1] This work was supported by the United States Defense Advanced Research Projects Agency under contract number DABT63-94-C-0080.

This paper describes the Rumor replicated file system, which was designed for use in a mobile environment. The goal of mobility led to development decisions focusing on availability, autonomy, and network traffic reduction from the mobile machine's point of view. The paper discusses design alternatives for replicated file systems, the decisions made for the Rumor system, the architecture of that system, performance of the Rumor system, and lessons learned in developing Rumor.

2 Replication Design Alternatives

Replication systems can usefully be classified along several dimensions: conservative vs. optimistic update, client-server vs. peer-to-peer, and immediate propagation vs. periodic reconciliation.

2.1 Conservative vs. optimistic update

A fundamental question in replicated data systems is how to handle updates to multiple copies of the same data item. If the copies cannot communicate instantaneously, then concurrent updates to different replicas of the same data item are possible, violating the ideal semantics of emulating single copy data storage.

Conservative update replication systems prevent all concurrent updates, causing mobile users who store replicas of data items to have their updates rejected frequently, particularly if connectivity is poor or non- existent. Even when connected, mobile users will spend bandwidth to check consistency at every update. Conservative strategies are often appropriate in the wired world, but they work poorly in most mobile environments.

Optimistic replication allows any machine storing a replica to perform an update locally, rather than requiring the machine to acquire locks or votes from other replicas. Optimistic replication minimizes the bandwidth and connectivity requirements for performing updates. However, optimistic replication systems can allow conflicting updates to replicated data items. Both simulation results [17] and extensive actual experience [7], [14] have shown that conflicts are rare and usually easy to resolve.

2.2 Client-server vs. peer-to-peer

In client-server replication, all updates must be propagated first to a server machine that further propagates them to all clients. Peer-to-peer systems allow any replica to propagate updates to any other replica. Client-server systems simplify replication systems and limit costs (partially through imposing a bottleneck at the server), but slow propagation of updates among replicas and are subject to failure of the server. Peer systems can propagate updates faster by making use of any available connectivity, but are more complex both in implementation and in the states they can achieve. Hybrid systems use peer replication among servers, with all other machines serving as clients. These hybrid systems typically try to avoid the disadvantages of peer systems by requiring tight connectivity among all servers, implying that none of them can be disconnected mobile machines.

Client-server replication is a good choice for some mobile systems, such as mobile computers that disconnect from a central network and remain disconnected until they return. Workers who take their machines home at night, or go on trips with their portable computer and only connect to the home machine via modem while away, are examples of mobile computing suitable for client/server replication. Peer replication is a good choice when the connectivity patterns of the mobile computers are less predictable. Environments suited to peer replication include disaster relief teams that must carry their own infrastructure or a wireless office of cooperating workers.

2.3 Immediate propagation vs. periodic reconciliation

Updates to data replicas must be propagated to all other replicas. Update propagation can be attempted immediately when the update occurs, or at a later time. Immediate propagation notifies other replicas of the new state of the data as quickly as possible, when it works. However, it may use scarce, expensive resources to do so, perhaps when immediate propagation was not very important. Alternatively, updates can be propagated at a later, more convenient time, typically batched. This option of periodic reconciliation does not spread updates as quickly, but allows propagation to occur when it is cheap or convenient. In systems permitting disconnected operation, some form of periodic reconciliation must be supported, since immediate propagation will fail when machines are disconnected. Both options can be supported, at the cost of extra complexity and possibly higher use of scarce, expensive resources.

3 Rumor Design and Architecture

Rumor is an optimistic, peer-to-peer, reconciliation-based replicated file system. Rumor is built at the user level, which has advantages in portability and limiting replication costs. It has been operational for several years in our research laboratory and other sites worldwide, and runs on several Unix systems.

To achieve higher portability, Rumor is built strictly at the application level. Rumor has its intellectual roots in the Ficus replicated file system [2], which was an in-kernel, SunOS 4.0.3-based implementation of an optimistic, peer-to-peer replication system that used immediate propagation with reconciliation as a fall-back. Rumor borrows much of the internal consistency policies and algorithms from Ficus. Both systems allow updates whenever any replica is available. The reconciliation algorithms of both systems reliably detect concurrent file system updates and automatically handle concurrent updates to directory replicas. Both Rumor and Ficus permit users to write tools to automatically reconcile concurrent file updates for other kinds of files [13].

Rumor operates entirely at the application level. Rumor requires no kernel modifications or dynamically loadable libraries. Installation is accomplished entirely without super-user (root) privileges, allowing anyone to install or upgrade Rumor. The benefits of this application-level implementation include being easily portable across different systems and platforms, free distribution of Rumor source code with no license restrictions, and no Rumor performance overhead during the normal operation of the host machine. Nothing is installed in the critical path of the user's operations, so Rumor users pay no performance cost except when they choose to reconcile.

Rumor is purely optimistic, and uses peer replication. While Rumor's peer replication mechanically permits any replica to reconcile with any other replica, mechanisms exist to effectively constrain the patterns of reconciliation. Thus, Rumor can emulate a client-server system or any other constrained topology of update propagation suitable for particular situations. Rumor maintains consistency purely by a periodic process called *reconciliation*. Rumor does not attempt instant update propagation. Periodic reconciliation makes the problems of temporary network and machine failures easier to solve, as reconciliation guarantees to maintain consistency when communication can be restored. Additionally, multiple updates to a single file can often be batched and transmitted as one update, with the usual batching performance improvement. Furthermore, the costs of communicating with other replicas are amortized, since updates to multiple files are handled in a single reconciliation.

Rumor operates on file sets known as *volumes*. A volume is a continuous portion of the file system tree, larger than a directory but smaller than a file system. For example, a user's mail directory might constitute a volume. Volumes have been used in many systems because they offer several key benefits. They take advantage of locality for performance by grouping logically connected files into a single physical location. Performance-intensive tasks are initiated once for the volume entity, instead of once per volume member. Volumes provide natural firewalls that prevent the propagation of errors and help establish fundamental security barriers. Volumes also assist in naming, by allowing a collection of logically connected files to be identified and acted upon with a single name.

Reconciliation operates at the volume granularity. At reconciliation time, replicas synchronize a single volume. When machines store multiple volumes, reconciliation periodically executes separately on each volume. The reconciliation interval controls the balance point between consistency and system load. Users are free to customize or add intelligence such as load-threshold values to the reconciliation interval.

A disadvantage of pure volume replication is that entire volumes must be stored at each replica and reconciled as a whole. This is a significant disadvantage for a mobile computing system, where disk space available for storage may be limited, and small quantities of bandwidth suitable for propagating only very important updates might be available. Rumor overcomes this problem by *selective replication* [10] and a per-file reconciliation mechanism. Selective replication allows particular replicas to store only portions of a volume, while still guaranteeing correct operation. Per-file reconciliation permits individual files to be reconciled, rather than the entire volume, at much lower costs.

Reconciliation operates between a pair of communicating replicas in a one-way, pull-oriented fashion. A one-way mode is more general than a two-way model, and lends support for inherently uni-directional forms of communication, such as floppy-disk transfer. At reconciliation time, the target replica selects a source replica with which to reconcile; selection is based on a number of criteria, described below. Once a source replica has been identified and contacted, reconciliation ensures that the target learns all information known by the source. This information includes *gossip*: the source replica transfers not only its local updates, but also all updates it has learned of from previous reconciliations with other replicas. For instance, two machines that rarely or never directly communicate can still share data if a mutually-accessible third machine gossips on the others' behalf.

Reconciliation involves only pairs of replicas, rather than all replicas, because there is no guarantee that more than two will ever be simultaneously available. For example, mobile computers operating in a *portable workgroup* mode may only be connected to a single other computer. Additionally, operating in a point-to-point mode, with an underlying gossip-based transfer mechanism, allows a more flexible and dynamically changeable network configuration in terms of the machines' accessibility from each other. Broadcast or multicast reconciliation will often save significant amounts of bandwidth, but they are not yet implemented in Rumor.

Reconciliation is responsible for maintaining data consistency on all user-nameable files. Rumor uses Parker's *version vectors* [8] to detect updates and update/update *conflicts* (concurrent updates). A version vector is a dynamic vector of counters, with one counter per replica. Each counter i tracks the total number of known updates generated by replica i. Each replica independently maintains its own version vector for each replicated file; by comparing two version vectors, the update histories of the corresponding file replicas can be compared.

The particular choices of communication partners between sites forms the *reconciliation topology*. While the reconciliation algorithms are topology independent, the actual topology can affect both the number of messages exchanged between all replicas and the time required to reach consistency. Rumor utilizes an *adaptive ring* between all volume replicas, which reconfigures itself in response to current replica availability and provides that reconciliation occurs with the next accessible ring member. In the extreme, if only two replicas are in communication, the adaptive nature of the ring allows them to reconcile with each other. The adaptive ring requires only a linear complexity in the number of volume replicas to propagate information to everyone, and additionally is robust to network failures and reconfigurations. Due to its adaptive nature, the ring does not require point-to-point links interconnecting all members, and thus allows sharing to occur between rarely or never-communicating participants by relying on third-party replicas to gossip on their behalf. However, the ring does not scale well in number of replicas.

Rumor is designed to handle arbitrary numbers of replicas and correctly manage data consistency between all of them. Performance, rather than correctness, dictates replication factor scaling for Rumor. Rumor scales gracefully to approximately 20 replicas of any given volume. An extension to Rumor, called Roam, allows an order of magnitude better scaling [11].

Replicated systems face special problems when handling deallocation of system resources (such as disk space) held by unnamed file system objects. This deallocation process is called *garbage collection*. In a centralized system, garbage collection is relatively simple. Whenever the last name for a file is deleted, all of the file's resources can be reclaimed. Garbage collection is harder in a distributed, replicated environment due to dynamic naming. Dynamic naming allows users to generate new names for existing files; since a file should not be removed until all names have been removed, including new names at remote replicas, local removal of the last name may not indicate that the resources should be reclaimed. Rumor uses a fully distributed, two-phase, coordinator-free algorithm to ensure that all replicas learn of the garbage collection process and eventually complete it, even though any given set of participants may never be simultaneously present. Rumor enforces the Ficus *no lost updates* semantics [7], which guarantees that the most recent data version will be preserved so long as the file is globally accessible.

3.1 Rumor architecture

Rumor needs to manage attributes above and beyond the standard file system attributes, such as version vectors. All replication state is maintained by Rumor in a lookaside database hidden within the volume. Currently, Rumor uses a specially formatted file for this database, but it could use any general database facility that supports transactions, which are required to recover from failures and unexpected terminations during execution.

The attribute database is not updated at file modification time, since Rumor contains no kernel code and does not trap updates. Instead, updates are detected at reconciliation time. By comparing file modification times, and occasionally resorting to checksum comparisons, Rumor is guaranteed to detect all file updates. In the case of directories, a list of directory entries is utilized instead of a checksum; in general, checksum comparisons are rare and only required in special circumstances involving explicit manipulation of timestamps or updates that occur during reconciliations. Because the attribute database is only periodically updated, transient files created and removed between reconciliation executions are never even noticed by Rumor. Such temporary files are by definition unimportant, and periodic database updates saves the user the expense of replicating such files.

Reconciliation can be separated into three distinct phases: *scan*, *remote-contacting*, and *recon*. Figure 1 shows data and control flows during reconciliation. The scan phase is responsible for determining the set of files in the volume, including detecting new files, updating the attributes of existing files, and noticing the removal of previously managed files. The remote-contacting phase finds a remote replica, informs it of the reconciliation, and performs a scan

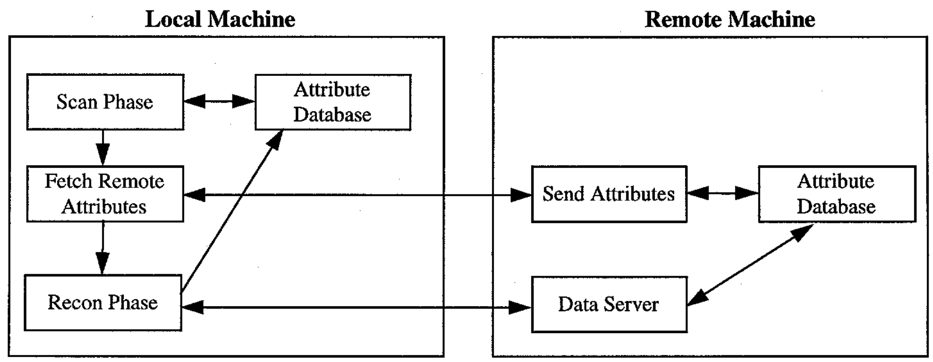

Fig. 1. Overall Rumor architecture, indicating control and data flow.

at that remote replica. The recon phase takes the list of local file attributes and the list of remote file attributes received from a remote replica during the remote-contacting phase and compares the two, taking the appropriate action (e.g., update the local replica) as determined by a comparison of the two sets of attributes.

The scan phase reads the previous set of file attributes from the database and recursively traverses the volume, detecting new files and modifying the attributes for previously known files if they have been updated. Updates are detected by examining modification timestamps and occasionally checksums, or a list of directory entries for directories. If an update has occurred, Rumor increments the version vector, obtains a new checksum, and updates the remaining attributes. The scan phase detects files that have been removed by the user by comparing the output traversal with the list of files from the previous scan, stored in the attribute database. When the scan phase completes, Rumor writes the new list of files and attributes into the lookaside database and provides them to the recon phase for processing.

The remote-contacting phase locates a remote replica of the volume according to the reconciliation topology and obtains a list of that replica's files and attributes. The remote site generates its list of files and attributes simply by initiating a scan phase on its volume. File data is not transferred during this phase, unless reconciliation is using a uni-directional transport mechanism, such as floppy disks. In this case, data must be sent by the remote site because there will be no further communications.

The recon phase performs the majority of the reconciliation work. Given the two lists of files and attributes, one local and one remote, file versions are compared and reconciliation actions are taken when appropriate. The comparison can yield four different results:

– The local version dominates (is more recent than) the remote version; no action need be taken, because the remote site will obtain the new version when it initiates a reconciliation.

– The remote version dominates; a request for file data is made of the remote site.
– The two versions are equivalent; no action need be taken.
– The two versions conflict (they each received concurrent updates); conflict-resolving and processing actions are taken. Often the conflict can be automatically resolved [13], but when it cannot, the user is notified of the conflict by email along with instructions on how to resolve it.

The recon phase also detects new files and deletions of old files. Data is requested for the new files, and they are created locally. File removals are processed by removing the appropriate local object and participating in the garbage collection algorithm.

Data requests are serviced by an asynchronous server at the remote site. Data requests are generated by the recon phase and are asynchronously processed by the remote server, which returns file data via the transport mechanism. Performing the data request and transfer asynchronously allows the recon phase to perform useful work during what would otherwise be wasted time waiting on network transmissions.

Rumor interacts with the specific data transport mechanism with very simple operations. Currently supported transport mechanisms include NFS, rshell, email, and floppy-disks.

The asynchronous data server receives data requests and processes them by performing a limited rescan of the file to ensure that the attributes to be sent with the file data are up to date. File updates could have been generated between the time that the list of files was sent and the data request was received, and Rumor does not trap such updates. Unless Rumor checks the attributes again before sending the data, updates might not be propagated correctly in some complex cases. Any new attributes are both written into the attribute database and shipped to the recon phase on the other machine. Similarly, before installing updates on the machine pulling the updates, Rumor must check the attributes of the file to ensure that simultaneous user activities have not updated them. Otherwise, Rumor would not always propagate data properly and might miss conflicting updates.

Rumor contains a selective replication facility that allows users to specify which files in a volume should be stored in a particular replica. Reconciliation does not transport updates to files that are not stored at the pulling site. However, such files are scanned locally during each reconciliation, allowing the local attribute database to be updated.

Reconciling an entire volume may be very costly when bandwidth is limited. Rumor permits users to specify individual files to be reconciled, giving the user more control over the costs paid to maintain file synchronization. This option limits the costs of reconciliation to the costs of shipping metadata related to the particular file and, if necessary, the file's contents. Basically, single-file reconciliation applies the Rumor reconciliation mechanism to one file instead of an entire volume.

Rumor itself provides no data transmission security, nor does it enforce policies on who may replicate particular files. The Truffles system [12] works with Rumor to provide those protections. Rumor provides the mechanism for portable computer users to store local replicas of files, but does not help them decide which files should be replicated. The Seer system [5] provides automatic file hoarding for this purpose, using Rumor as a replication mechanism to enforce Seer's decisions about which files to hoard. Better secondary storage space management is a growing requirement for portable computers.

4 Performance

Determining the overall performance of an optimistically replicated system is not easy [4]. In particular, determining the degree to which the system achieves the same semantics as a single-copy file system can be very difficult. The best available data of this kind comes from [13] and [17]. While not directly measuring Rumor, this data is suggestive of this aspect of Rumor's performance.

Space permits only limited discussion of Rumor performance, but we include data on two of the most important metrics: the run time to perform reconciliation on realistically large volumes and the disk storage overheads required to support Rumor. Rumor's design is such that both time and space overheads are more visible with larger numbers of smaller files, and so the data reported here focus on that portion of the exploration space.

We ran experiments on reconciling updates to a large volume replicated on two machines. The machines were Dell Latitude portable computers running 486 processors at 100 MHz, with 48 Mbytes of main memory. The communications media was a dedicated Ethernet running no other traffic. The test volume used 8 Mbytes of disk space to store 1779 files; the median file size was 3 Kbytes, and the maximum file size was 116 Kbytes.

In the experiments, various percentages of files in the volume were updated on one replica, and the other replica invoked reconciliation to pull the updated versions across. Figure 2 shows the resulting elapsed time to perform these reconciliations. These measurements represent five runs at each graphed point, with 95% confidence intervals displayed.

Reconciling a large volume with no updates thus took around 2 1/2 minutes. Reconciling the same volume with 10% of its files (by number, not by size of data) updated took a little less than 4 minutes. As Figure 2 shows, the increase in run time is close to linear. Running a minimal recursively executed command that only printed file names on the same volume took 6.3 seconds. The reconciliation times shown here are reasonable for periodic reconnections, such as via modem, but ideally they should be shorter.

The disk overhead to store Rumor attributes and other necessary information was 7.8% at each replica.

Figure 3 shows the number of bytes transported across the wire to perform the reconciliations. These figures include both the actual changed user data and Rumor's exchange of file lists, as well as any other communications overheads

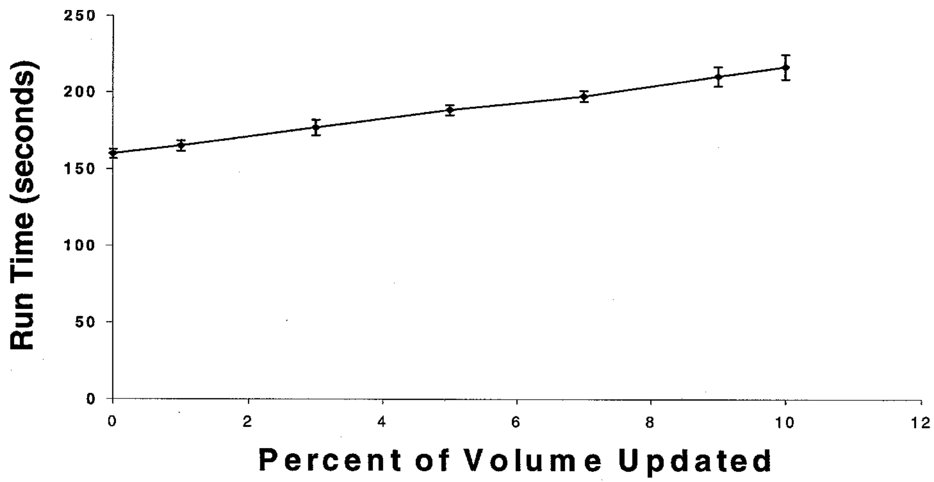

Fig. 2. Rumor Reconciliation Times

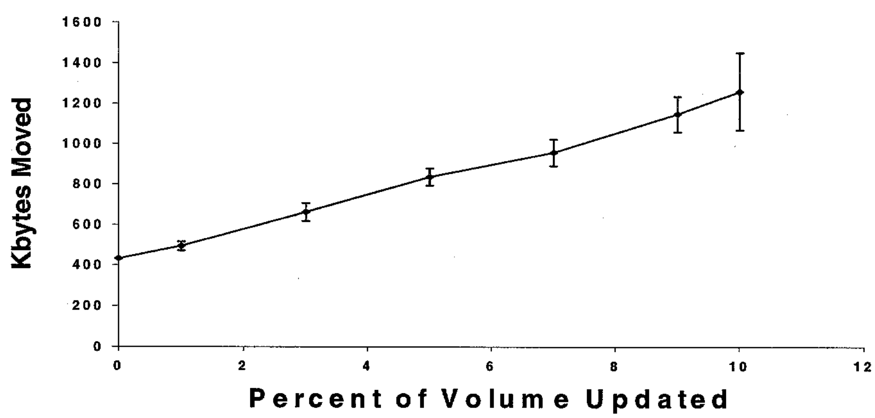

Fig. 3. Data Transferred During Reconciliation

added by Rumor. The maximum amount of data transferred was 1.2 Mbytes. Given the times required to perform reconciliation, even a modem could handle the data transfer rate required by Rumor without further slowing down the process.

Rumor has not been fully optimized for either space or time. The overheads reported here could be improved significantly with more optimization.

5 Related Work

There are many replicated file systems and database systems. Rumor's ancestor Ficus [2], which in turn was strongly influenced by Locus [9], shares many of the same goals as Rumor, but did not address mobility concerns. CMU's Coda [14] is an optimistic, client-server replication system targeted at mobility. The optimistic Bayou [16] system from Xerox PARC provides peer replication, but requires application-aware conflict detection and resolution, and does not yet provide selective replication. Mitsubishi's *reconcile* [3] facility was designed to support multiplatform (DOS and Unix) replicated directories, but is not currently available.

A number of commercial filesystem and database products support optimistic replication to varying degrees. Novell Replication Services [6] supports reconciliation-based, optimistic file replication with selective replication. Sybase's Replication Server [15] supports optimistic database replication, but the supporting documentation discourages such usage. Similarly, Oracle's Oracle7 relational database product allows optimistic updates, and provides semantic tools to help designers avoid conflicting updates wherever possible [1].

6 Summary

The combination of principles embedded in the design and implementation of Rumor is essential to effective mobile data access in the emerging computing and communications environment of the 21st century. Optimistic, peer-to-peer, reconciliation-based data replication techniques are well-suited to meet the challenges that lie ahead.

Rumor is a working system, implemented in an object-oriented style largely using C++ (with support code in Perl). A beta version of Rumor for Linux and FreeBSD is available at http://ficus-www.cs.ucla.edu/rumor.

References

1. Alan R. Downing. Conflict resolution in symmetric replication. In *Proceedings of the European Oracle User Group Conference*, 1995.
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, Anaheim, CA, June 1990. USENIX.

3. John H. Howard. Using reconciliation to share files between occasionally connected computers. In *Proceedings of the Fourth Workshop on Workstation Operating Systems*, pages 56–60, Napa, California, October 1993. IEEE.
4. Geoffrey H. Kuenning, Rajive Bagrodia, Richard G. Guy, Gerald J. Popek, Peter Reiher, and An-I Wang. Measuring the quality of service of optimistic replication. In *Proceedings of the ECOOP Workshop on Mobility and Replication*, Brussels, Belgium, July 1998.
5. Geoffrey H. Kuenning and Gerald J. Popek. Automated hoarding for mobile computers. In *Proceedings of the 16th Symposium on Operating Systems Principles*, pages 264–275, St. Malo, France, October 1997. ACM.
6. Novell, Inc. Novell Replication Services white paper. unpublished, http://www.novell.com/whitepapers/nrs, 1997.
7. Thomas W. Page, Jr., Richard G. Guy, John S. Heidemann, David H. Ratner, Peter L. Reiher, Ashvin Goel, Geoffrey H. Kuenning, and Gerald J. Popek. Perspectives on optimistically replicated, peer-to-peer filing. *Software—Practice and Experience*, 27(12), December 1997.
8. 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.
9. Gerald J. Popek and Bruce J. Walker. *The Locus Distributed System Architecture*. The MIT Press, 1985.
10. David H. Ratner. Selective replication: Fine-grain control of replicated files. Technical Report CSD-950007, University of California, Los Angeles, March 1995. Master's thesis.
11. David Howard Ratner. *Roam: A Scalable Replication System for Mobile and Distributed Computing*. PhD thesis, University of California, Los Angeles, Los Angeles, CA, January 1998. Also available as UCLA CSD Technical Report UCLA-CSD-970044.
12. P. Reiher, T. Page, S. Crocker, J. Cook, and G. Popek. Truffles—a secure service for widespread file sharing. In *Proceedings of the The Privacy and Security Research Group Workshop on Network and Distributed System Security*, February 1993.
13. Peter Reiher, John S. Heidemann, David Ratner, Gregory Skinner, and Gerald J. Popek. Resolving file conflicts in the Ficus file system. In *USENIX Conference Proceedings*, pages 183–195, Boston, MA, June 1994. USENIX.
14. Mahadev Satyanarayanan, James J. Kistler, Puneet Kumar, Maria E. Okasaki, Ellen H. Siegel, and David C. Steere. Coda: A highly available file system for a distributed workstation environment. *IEEE Transactions on Computers*, 39(4):447–459, April 1990.
15. Sybase, Inc. Sybase SQL Anywhere and Replication Server: The enterprise wide replication solution. White paper, http://www.sybase.com:80/products/system11/repserv.html, 1998.
16. Douglas B. Terry, Marvin M. Theimer, Karin Petersen, Alan J. Demers, Mike J. Spreitzer, and Carl H. Hauser. Managing update conflicts in Bayou, a weakly connected replicated storage system. In *Proceedings of the 15th Symposium on Operating Systems Principles*, pages 172–183, Copper Mountain Resort, Colorado, December 1995. ACM.
17. An-I A. Wang, Peter L. Reiher, and Rajive Bagrodia. A simulation framework for evaluating replicated filing environments. Technical Report CSD-970018, University of California, Los Angeles, June 1997.