Peer-to-Peer Reconciliation Based Replication for Mobile Computers

Peter Reiher, Jerry Popek, Michial Gunter, John Salomone, David Ratner
UCLA

1. Introduction

Data replication is particularly important for mobile computers, since disconnected or poorly connected portable computers must rely primarily on their own data resources. If those resources also need to be shared by other users, or require a more stable permanent location for backup and reliability, the best alternative is to replicate a copy of the data on the portable computer. Full replication is better than simple caching, as it better supports full functionality in the portable computer's data, and better generalizes to more than a single user.

Early forms of replication were used in environments different from mobile computing. In these environments, disconnection was uncommon, and replication had the primary purposes of providing fast local access and higher reliability in the face of failures. In mobile computing, disconnection (or, nearly equivalently, very poor connectivity) is a normal case. Replication is required here for availability, as well as for performance and reliability.

Because of its different requirements, mobile computing replication is more suitably handled by peer-to-peer models than by client/server models, and by reconciliation-based replication than by update-propagation based replication. This position paper will define these styles of replication, present arguments for why peer-to-peer and reconciliation-based methods are better for mobile computing, and describe a replicated file system for mobile computers that uses peer-to-peer, reconciliation-based methods.

2. Peer-to-Peer Replication

Peer-to-peer replication permits any replica of a data item to exchange update information with any other replica [1]. Client/server replication permits a data item replica to transmit its updates only to one or more specially designated server replicas [3]. The updates are transmitted from the servers to all other clients.

The client/server model of replication is can work very well in an office workstation setting, where connectivity is generally available and communications patterns are mostly fixed. In a more fluid setting, it has some disadvantages. Consider the case of two members of a research project who travel together to a conference, taking their portable computers with them. If those project members are performing cooperative work on the same data, very likely they will both have replicas of certain data items stored on their portable computers. If they make updates to some of those items, they would like to be able to trade their updates, effectively merging their work. In the client/server model, neither portable is likely to host a server copy of the data items, since the server replicas typically live on workstations or fixed server machines that will always be available. (After all, a disconnected server is not very useful.) Since the portable computers typically have only client replicas, they cannot directly trade their updates in the client/server model. Instead, each client must connect to a server machine, first to push their own updates to the server, then to pull the other client's updates from the server. If the clients are in Europe while the servers are on the West coast of the United States, getting the data from two portable computers that are within a meter of each other requires sending data practically around the world. If the only communications medium available connects only the two portable computers, then despite physical co-location and connectivity, they cannot trade updates.

In the peer-to-peer model, the two traveling replicas can immediately trade updates whenever they have connectivity, since any two replicas can exchange updates. In the scenario described above, the two traveling co-workers would simply attach their machines and invoke the action that causes updates to propagate.

The cost of using peer-to-peer replication, rather than client/server replication, is complexity of the algorithms used to control the replication. In client/server computing, the central location to which all updates must be posted substantially simplifies certain issues in replication, such as garbage collection. The full simplicity is only achieved when there is a single server replica, however. Single replica server systems have poor reliability, since the failure of the server makes it impossible for any other replicas to receive new updates or disseminate their own updates to others. Client/server systems that support multiple server replicas for higher reliability and performance must use peer-to-peer algorithms within the set of server replicas [3]. Assuming that all servers are highly available and always connected again simplifies matters, but if one must tackle the complexities of peer-to-peer replication at some level, anyway, less is gained from the simplifications of the client/server model.

Note that using peer-to-peer models for replication says nothing about the data access model used by the actual applications accessing the data. Client/server applications can just as easily use replicated data maintained by a peer-to-peer replication system as that maintained by a client/server replication system.

3. Reconciliation-Based Replication

Update propagation-based replication attempts to propagate updates made at one replica to the other replicas immediately, either directly or through some propagation graph spanning the overall set of replicas that minimizes communications costs. In a frequently disconnected environment, some or all of these update propagations are doomed to failure. Any effort spent trying to perform them is effort wasted. In a poorly connected environment, the situation might actually be worse. If the system attempts to propagate automatically all updates made to replicated data, the limited, expensive bandwidth available to a portable computer connected via a wireless network might be wasted on attempts to propagate relatively unimportant information. For example, the accidental creation of a core file in replicated space could result in megabytes of useless data being propagated over an expensive, slow network at tremendous cost and no benefit. While particular solutions may exist to deal with particular problems of this kind, the underlying characteristics of machines that are frequently poorly connected do not match well with relying on update propagation for dissemination of changes to replicated data.

Another alternative is reconciliation-based replication, in which no attempt is made to propagate updates automatically. Instead, periodically all changes made to the replicated data are batched together and sent to another site storing a replica. These batched changes can be sent during periods of high, cheap connectivity. With some effort, the system expends little or no effort on update dissemination at the time updates are actually made. No scarce bandwidth is consumed at those times, either. This alternative is particularly suitable for portable computers.

Even systems that rely on update propagation as their primary method of transmitting updates may find the use of reconciliation worthwhile. When such a system experiences a long period of disconnection (or, equivalently, one of its replication partners is disconnected for a long period), rather than maintain a queue of updates to be propagated when connectivity is re-established, the system can fall back on reconciliation. Since many updates are likely to be superseded, anyway, the reconciliation method can result in less load upon reconnection, and also removes the necessity of spending resources maintaining the update queue.

The cost of propagating updates at a later reconciliation time rather than immediately is that the updates are not disseminated to other replicas at the earliest possible moment. Other sites might use outdated versions of the data when they could be using the most recent version. In highly connected systems, this cost may be significant. In poorly connected systems, however, the attempt to propagate the update instantly would probably have failed, anyway, in which case there is no cost. If the attempt succeeded, it did so at the cost of using bandwidth perhaps better used for other purposes, so the benefit gained by the update propagation must be balanced against the cost of misuse of limited bandwidth. If users truly desire propagation of their replicated data updates while poorly connected, they can always request an immediate reconciliation. Providing reconciliation on a fine data granularity makes this alternative more feasible.

In systems that experience significant periods of high connectivity mixed with periods of poor connectivity, the system could try to detect the level of connectivity and attempt to propagate updates instantly when it was high. This solution is feasible to the extent that the connectivity is detectable automatically. It has costs in the complexity of the system.

4. The Rumor Replicated File System

Rumor is a replicated file system built for use in a mobile environment where poor connectivity is the rule. Rumor is a peer-to-peer reconciliation-based replication service. All sites store peer copies of the files they replicate, and updates are propagated solely through reconciliation. Rumor is a working system, and serves as a demonstration of the validity and suitability of peer-to-peer reconciliation-based data replication solutions for mobile computers. Rumor is an intellectual descendant of the Ficus file system [1].

Rumor has been built as an application-level service. It makes no use of any kernel facilities beyond those exported to normal applications. Rumor also does not use special libraries or privileged programs. Rumor interposes no code at all during file update or file access time. It is only active at reconciliation time, when Rumor is explicitly invoked by the user or a daemon process.

Rumor keeps records of the state of the replicated files it controls. These records are updated each time replication is run on the *volume* of replicated files. Rumor examines the information available about the current state of the files (such as modification time, modification time of meta-attributes, length, etc.) and compares it to stored information to deduce which files have experienced updates. Rumor then compares the state of the local replicated volume to that of a single remote replica of the volume and determines which files must have updates propagated.

Unlike some commercial products, such as Laplink and File Assistant, Rumor is a general replication service. Those products typically produce good, correct results for two replicas, but do not perform as well for more than two replicas. Rumor will handle arbitrary numbers of replicas. (Rumor has a practical limit of twenty replicas or so, due to overheads of storing meta-data and the speed at which updates will propagate through the system.) Rumor correctly detects and handles all cases involving various forms of conflicts, including update/update conflicts, update/delete conflicts, and name conflicts [2]. Rumor uses *version vectors* to guarantee that each update has a unique signature, thus ensuring that the same update need never be transmitted to the same replica more than once.

Since Rumor was designed to work at the user level, it is relatively portable. Complete portability is not possible, since Rumor must rely on the information about files made available by the underlying operating system. Since Unix systems and Windows 95 (for example) export different information about the files they store, and have different semantics for various file system behaviors, Rumor cannot behave exactly the same on top of both platforms. However, we have designed Rumor to be as portable as possible, by dividing the code into platform-independent and platform-dependent parts. Rumor currently runs on Linux and SunOS 4.1.1. Ports to other Unix-style systems are straightforward. A port to Windows 95 is under way.

Rumor is designed to replicate files, at the moment. However, little in the design is specifically tied to files, other than details of determining when updates have occurred and details of installing new updates. With some modifications, Rumor could replicate other data entities, such as objects or database relations. Certainly the basic methods used to replicate data in Rumor are not limited to file replication.

Rumor is a working system. It is implemented in an object-oriented style, largely using C++. An alpha version of Rumor is available on the World Wide Web at http:// ficus-www.cs.ucla.edu/rumor.

Research on replication in mobile computing continues at UCLA using Rumor as a base. This research includes systems for automatically caching necessary data on mobile computers prior to disconnection, providing consistency guarantees in environments that include both replication and remote data access, security concerns for data replication in a mobile environment, and replication at a much larger scale, up to hundreds of replicas.

5. Conclusions

Peer-to-peer replication is particularly well suited for maintaining replicated data in a mobile environment. When intermittently connected machines cannot be sure that the next replication partner they talk to will be a server, the ability to accept and propagate updates with any other partner is extremely valuable. For some very simple and common scenarios, the client/server model works poorly, while peer-to-peer replication works well.

Reconciliation-based replication is also particularly well suited for mobile environments. Using reconciliation to disseminate updates, rather than instant update propagation, makes better use and gives better control of expensive, limited bandwidth. In cases where machines are completely disconnected, attempting instant update propagation has no effect other than adding useless overhead to the system. Reconciliation adds costs only at the time it is invoked.

Rumor is a system that demonstrates these benefits. Rumor replicates files using a peer-to-peer, reconciliation-based strategy. Rumor is a working system that can be used to replicate real data. The basic methods used by Rumor could be used by other systems to replicate different types of data for the mobile environment. With some adaptation, Rumor itself would be able to handle different forms of data.

Bibliography

[1] Richard Guy, John Heidemann, Wai Mak, Thomas Page Jr., Gerald Popek, and Dieter Rothmeier, “Implementation of the Ficus Replicated File System,” Proceedings of the Summer Usenix Conference, 1990.

[2] Peter Reiher, John Heidemann, David Ratner, Gregory Skinner, and Gerald Popek, “Resolving File Conflicts in the Ficus File System,” Proceedings of the Summer Usenix Conference, 1994.

[3] M. Satyanarayanan, James Kistler, Puneet Kumar, Maria Okasaki, Ellen Siegel, and David Steere, “Coda: A Highly Available File System for a Distributed Workstation Environment,” IEEE Transactions on Computers, April 1990.