The Ward Model: A Replication Architecture for Mobile Environments*

David Ratner Gerald J. Popek† Peter Reiher
Department of Computer Science
University of California, Los Angeles

Abstract

Replication in mobile environments requires fundamentally different solutions than those previously proposed for static systems, due to requirement differences and to changes in how users work and operate within the system. Here we present a model for replication in mobile environments that solves these problems by blending together traditional client-server and peer-to-peer approaches.

1 Introduction

Replication is especially important in mobile environments, since disconnected or poorly connected machines must rely primarily on local resources. When resources need to be shared between multiple users and reside at multiple locations, replicating them at most or all interested hosts is often the best approach.

Early replication solutions assumed stationary and relatively well-connected environments, and thus had relatively simple architectures. Even systems designed to support frequent network disconnections and treat them as normal occurrences [1, 4] were built assuming static infrastructures.

However, mobile users are by definition not static, and have difficulty operating within a static infrastructure. Extraordinary actions, such as long distance telephone calls over low-bandwidth links, must occur to conform to the underlying static model, costing users additional time and money while providing them a degraded service. Additionally, mobile users have difficulty inter-operating with other mobile users, because communication patterns and topologies are typically predefined according to the underlying infrastructure. Often, direct interaction and synchronization between mobile users is simply not permitted.

The main problem is two-fold. First, mobility introduces unpredictability in that desirable communication partners and the optimal synchronization topology cannot be predicted *a priori*. One user may travel from Los Angeles to New York while another Los Angeles user travels to Seattle. Two machines that were geographically co-located and able to synchronize inexpensively with each other are suddenly very far apart. Communication between them would be expensive, time-consuming, and perhaps even physically impossible, given the often-disconnected nature of mobile networking. Additionally, these two machines are now in close proximity to other machines that they previously never communicated with directly and perhaps never even knew existed. Nevertheless, while traveling, the users want the ability to synchronize cheaply and easily with the neighboring, local users. Researchers who previously addressed this part of the problem turned to *peer-to-peer* algorithms [1, 10, 14] for their flexible control structures.

Second, scaling in the number of replicas becomes very important. Mobile users clearly require local replicas on their mobile machines; yet replicas must also be stored in the office environment for reliability, intra-office use by non-mobile personnel and system-administration activities like back-ups. As the number of mobile users increases, so does the required scalability of the replication system. Additionally, replication factors can often be kept small in office environments through LAN sharing and NFS: one machine stores a replica that other nearby machines access quickly, cheaply and easily. This is not the case in mobile environments. Mobile users require local

*This work was sponsored by the Advanced Research Projects Agency under contract DABT63-94-C-0080. The authors can be reached at the Department of Computer Science, UCLA, Los Angeles, CA 90095, or by email to {ratner,popek,reiher}@cs.ucla.edu.
†Gerald Popek is also affiliated with Platinum *technology*.

replicas of critical information, and in most cases desire local access to non-critical objects as well for monetary and latency reasons. Thus, mobile environments may require much higher replication factors than traditional static environments. Many researchers have therefore chosen *client-server* style solutions [2, 12, 13] because of their inherent ability to scale well.[1]

Unfortunately, practice has shown that neither approach is a complete solution. Peer algorithms do not scale well; client-server approaches restrict the system's functionality and flexibility by not allowing clients to intercommunicate and directly synchronize with each other. A mobile environment requires both good scaling and appropriate flexibility and functionality to be truly viable. Here we present a new approach that blends together traditional client-server and peer approaches. Called the *ward* model, the architecture and associated algorithms provide a complete solution to the problem of data replication in a highly mobile world. Initially designed for replicated file systems, the replication architecture also appears applicable to other data types.

2 Requirements of a Mobile Solution

A replication architecture for mobility has several requirements above and beyond those typically required for non-mobile environments. These can be briefly summarized as:

- Optimistic algorithms with peer-to-peer support
- Inherent support for wide-scale replication
- Large degree of replication flexibility

2.1 Optimistic, peer-to-peer support

Optimistic concurrency control is required to support the common network partitions caused by mobility. In contrast to conservative solutions such as primary-site or majority-vote, optimistic solutions allow updates to any replica at any time, and reliably detect and resolve the resulting *conflicts* [6, 9]. Availability is therefore greatly increased. Many systems [1, 12] have previously provided optimistic replication support, but optimism becomes even more essential in highly mobile environments, due to the commonality of disconnection and the increase in disconnection latency.

Peer-based algorithms [10], which allow all replicas to directly exchange information, are required to support interaction between mobile machines. Client-server algorithms allow no direct interaction between clients; mobile clients can only communicate with one another through a third party. Peer solutions allow any set of replicas to directly communicate with each other, facilitating *mobile workgroups* and easing problems caused by the seemingly random and sporadic geographic shifts of mobile users. Of course, the peer algorithms must additionally scale well, which has been a major problem for previous peer-based solutions.

2.2 Wide-scale replication

Most replication systems only provide for a handful of replicas of any given object, and traditional peer algorithms have never scaled well. Mobile environments seem to negate the assumption that a handful is enough. While we do not claim a need for thousands of read-write copies, it does seem likely that the environments common today and envisioned for the near future will require larger replication factors than current systems allow, because each mobile user needs local data access. Read-only strategies are not viable solutions because they force users to pre-select the writable replicas beforehand, and limit the number of writable copies themselves.

2.3 Replication flexibility

Replication flexibility is extremely important to mobile users because of the limited availability of resources when mobile. Well-connected users have vast resources, including networked disks and high bandwidth connections to remote disk pools. If important data is not replicated locally, it can typically be accessed cheaply and quickly.

This is not the case for mobile users. The decisions regarding what to replicate are critical, as the expense in servicing a *replication miss* (data should have been replicated locally but wasn't) becomes vastly more expensive.

[1] There are other beneficial aspects to and reasons to use client-server approaches; only one is outlined for the purposes of this discussion.

As a result, the system must provide a high degree of replication flexibility, in terms of being able to specify precisely and dynamically the data objects that should be locally replicated. Previous systems have only provided replication control at the *volume* [11] granularity, which does not provide the type of fine-grain control needed by the mobile user.

3 The Ward Architecture

The *ward* architecture combines classical elements of both the traditional peer-to-peer and client-server models, yielding a solution that scales well and provides replication flexibility, allowing dynamic reconfiguration of the synchronization topology. The algorithms and architecture are currently being implemented as part of UCLA's Travler project.

3.1 Architecture basics

The model's main grouping mechanism is the *ward*, or Wide Area Replication Domain. A ward is a collection of "nearby" machines, geographically co-located in some area and possibly only loosely connected. While all members of the ward are equal peers, the ward has a designated *ward master*, similar to a server in a client-server model but with several important differences:

- Since all ward members are peers, members of the ward can directly synchronize with one another, instead of synchronizing through the server. Typical client-server solutions do not allow client-client communication or synchronization. Whether by design or by accident, mobile users will encounter other mobile users; in such cases, direct access to the other ward member may be easier, cheaper and more efficient than access to the ward master.

- Since all ward members are peers, any ward member can serve as the ward master, in contrast to schemes that differentiate between clients and servers. Automatic re-election and *ward-master reconfiguration* can occur should the ward master fail or become unavailable, and algorithms exist to resolve multiple master scenarios. Correctness is not affected by a transient ward master failure; nevertheless, the system maintains better consistency if the ward master is typically available and accessible.

- The ward master is not required to store actual data for all intra-ward objects, though it must be able to identify (i.e. name) the complete set. Most client-server strategies force the server to store a superset of each client's data.

The ward master is the ward's only link with other wards; that is, only the ward master is aware of other replicas outside the ward. This is one manner in which the ward model achieves good scaling—by limiting the amount of knowledge stored at individual replicas. Traditional peer models force every replica to learn about other replica's existence; in the ward model, replicas are only knowledgeable about the other replicas within their ward.

All ward masters belong to a higher-level ward, forming a two-level hierarchical model.[2] Ward masters act on their ward's behalf by bringing new updates into the ward, exporting others out of the ward and gossiping about all known updates. Consistency is maintained across all replicas by having ward masters communicate with each other and having the information propagate independently within each ward. Figure 1 illustrates the basic architecture and communication paths.

Wards are dynamically formed at replica-creation time, and are dynamically maintained as the number of suitable ward-member candidates changes. Ward destruction automatically occurs when the last replica in a given ward is destroyed, nullifying the need for the particular ward.

3.2 Flexibility in the model

Replication flexibility is heavily stressed as an important feature of the ward model. The set of data stored within each ward, called the *ward set*, is dynamically adjustable, as is the set of ward members themselves. As ward members change their data demands and alter what replicated data they store locally, the ward set changes.

[2] The rationale behind the two-level hierarchy and its impact on scaling is discussed in Section 3.4.

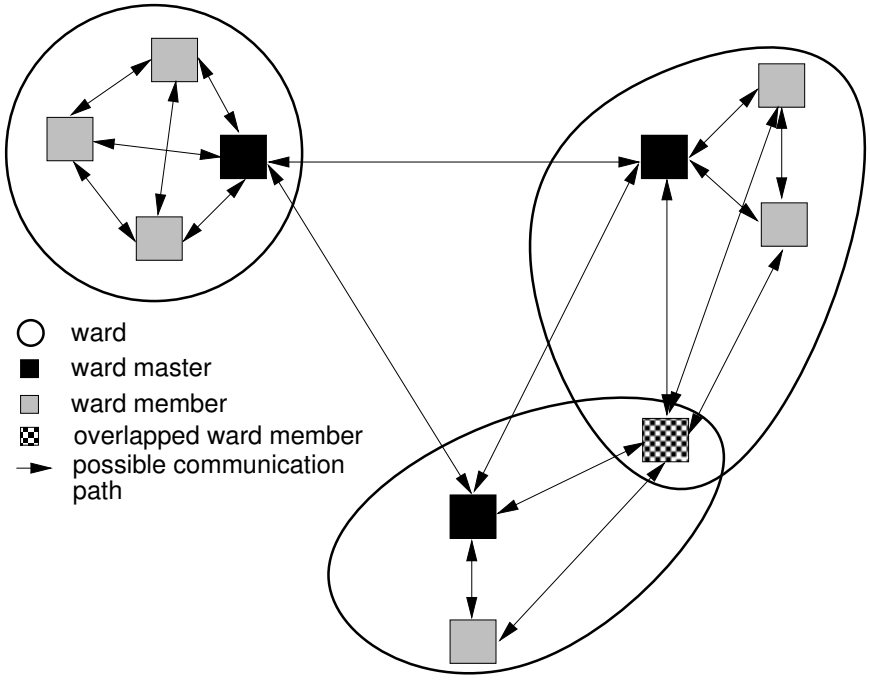

Figure 1: The basic ward architecture. Overlapped members are a mobility feature, and are discussed in Section 3.3.

Similarly, as mobile machines join or leave the ward, the set of ward participants changes. Both the ward set and ward membership are locally recorded and replicated in an optimistic fashion.

Additionally, each ward member, including the ward master, can locally store a different subset of the ward set. Such replication flexibility, called *selective replication* [7, 8] provides improved efficiency and resource utilization: ward members locally store only those objects in the ward set that they actively require. Replication decisions can be made manually or with automated tools [3, 5].

Since the ward set varies dynamically, it follows that different wards might store different sets of data as well; that is, not all ward sets will be equivalent. In other words, the model provides selective replication between wards themselves. The reconciliation topologies and algorithms [7] apply equally well within a single ward and between ward masters. Briefly, the algorithms provide that machines communicate with multiple other partners to ensure that each data object is synchronized directly with another replica. Additionally, the data synchronization algorithms support the *reconciliation* of non-local data via a third-party data-storage site. In this way the ward master can gossip, transmit and receive updates about data which it does not locally store but is stored somewhere within the ward.

3.3 Support for mobility

The model supports two types of mobility. *Intra-ward* mobility occurs when machines within the same ward become mobile within a limited geographic area; the machines encountered are all ward members. Since all ward members are peers, direct communication is possible with any encountered machine. Intra-ward mobility might occur within a building, when traveling to a co-worker's house, or at a local coffee shop.

Perhaps more interesting, *inter-ward* mobility occurs when users become mobile and travel (with their data) to another geographic region, encountering machines from another ward. Examples include businessmen traveling to remote offices and distant collaborators meeting at a common conference.

There are two chief issues to resolve. First, recall that due to the model's replication flexibility, two wards might not have identical ward sets. Thus, the mobile machine may store data objects not kept in the new ward and vice-versa. Second, consider the typical patterns of mobility. Often a users travels away from their "home location" only for a short time. The system would perform poorly if such transient mobile actions required global changes in data structures across multiple wards. On the other hand, mobile users occasionally spend long periods of time at other locations, either permanently or semi-permanently changing their definition of "home." In these scenarios, users should be provided with the same quality of service (in terms of local performance and time to synchronize data) as they experienced in their previous "home".

Our mobility solution resolves both issues by defining two styles of mobility—short-term (transient) and long-term (semi-permanent)—and having the ability to transparently and automatically upgrade from the former to the latter. The two operations are *ward overlapping* and *ward changing* respectively. Collectively, the two are called *ward motion* and enable peer-to-peer communication between any two replicas in the ward model, regardless of their current ward membership.

3.3.1 Ward changing

Ward changing involves a long-term, perhaps permanent, change in ward membership. The moving replica physically changes its notion of its "home" ward, forgetting all information from the previous ward; similarly, the various ward participants in the old and new wards alter their notion of current membership. Since ward membership is maintained optimistically, the problem of tracking membership in often-disconnected environments is straightforward.

The addition of a new ward member may change the ward set. Since the ward master is responsible for the inter-ward synchronization of all data in the ward set, the ward set must expand to properly encompass the replicated data stored at the moving replica. Similarly, the ward set at the old ward may shrink in size, as the ward set is dynamically and optimistically recalculated with the departure of the moving replica. The ward-set changes propagate to other ward masters in an optimistic, "need-to-know" fashion so that only the ward masters that care about the changes learn of them. Since both ward sets potentially change, and these changes are eventually propagated to other ward masters, ward changing is a heavyweight operation. However, users benefit in that afterwards, all local data can be synchronized completely within the local ward, giving users the same quality of service and reconciliation performance as in their previous ward.

3.3.2 Ward overlapping

In contrast, ward overlapping is intended as a very lightweight mechanism, and causes no global changes within the system. Only the new ward is effected by the operation. The localization of changes makes it a lightweight operation both to perform and to undo.

Ward overlapping allows simultaneous multi-ward membership, enabling direct communication with the members of each ward. To make the mechanism lightweight, we avoid changing the ward sets by making the new replica an "overlapped" member instead of a full-fledged participant. Ward members (except for the ward master) cannot distinguish between real and overlapped members; the only difference is in the management of the ward set. Instead of merging the existing ward set with the data stored on the mobile machine, the ward set is left unaltered. Data shared between the mobile machine and ward set can be reconciled locally with members of the new ward. However, data outside of the new ward set cannot be reconciled locally, and must either temporarily remain unsynchronized or else be reconciled with the original home ward.

3.3.3 Ward motion summary

When a replica enters another ward, there are only two possibilities: the ward set can change or remain the same. The former creates an improved-performance but heavyweight solution; the latter causes a moderate performance hit (when synchronizing data not stored in the new ward) but a very lightweight solution for transient mobile situations. Since both are operationally equivalent, the system can transparently upgrade the type of ward motion from overlapping to changing if it notices that the mobility seems more permanent than first expected.

Additionally, since ward formation is itself dynamic, users can easily form *mobile workgroups* by identifying a set of mobile replicas as a new (possibly temporary) ward. Mobile workgroups can be formed without leaving the old ward using ward overlapping. Ward motion and dynamic ward formation and destruction provide easy and straightforward communication between any set of replicas in the entire system.

3.4 Scalability

The scalability of the ward model is directly related to degree of replication flexibility. Ward sets can dynamically change in unpredictable ways; therefore the only method for a ward master to identify its ward set is to list each entry individually. The fully hierarchical generalization of the ward model to more than two levels faces scaling problems due to the physical problems of storing and indexing these lists of entries.

Nevertheless, the proposed model scales well within its intended environment, and allows several hundred read-write replicas of any given object, meeting the demands of everyone from a single software developer or a medium-sized committee to a large, international company. The model could be adapted to scale better by restricting the degree of replication freedom. For instance, if ward sets changed only in very regular fashions, they could be named as a unit instead of naming all members, dramatically improving the model's scalability. However, we believe that replication flexibility is an important design consideration in the targeted mobile environment, and one that users absolutely require.

4 Conclusions

Mobile environments are significantly different from static ones. These differences lead to different requirements for the replication services—requirements that current replication systems cannot provide. The algorithms must be optimistic and support a peer-to-peer framework while scaling well and supporting dynamic mobility within the system. The ward model has these qualities, and is therefore a suitable architecture for replicating many types of data in mobile environments. The world is becoming increasingly mobile. Users will require a replication service that matches their needs and requirements; they will not want to adapt their behavior to the replication service.

References

[1] R. G. Guy, J. S. Heidemann, W. Mak, T. W. Page, Jr., G. J. Popek, and D. Rothmeier. Implementation of the Ficus replicated file system. In *USENIX Conference Proceedings*, pages 63–71. USENIX, June 1990.

[2] P. Honeyman, L. Huston, J. Rees, and D. Bachmann. The Little Work project. In *Proceedings of the Third Workshop on Workstation Operating Systems*, pages 11–14. IEEE, Apr. 1992.

[3] J. J. Kistler. *Disconnected Operation in a Distributed File System.* Ph.D. dissertation, Carnegie-Mellon University, May 1993.

[4] J. J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda file system. Technical Report CMU-CS-91-166, Carnegie-Mellon University, July 1991.

[5] G. H. Kuenning. Design of the SEER predictive caching system. In *Proceedings of the Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, Dec. 1994.

[6] P. Kumar and M. Satyanarayanan. Supporting application-specific resolution in an optimistically replicated file system. In *Proceedings of the Fourth Workshop on Workstation Operating Systems*, pages 66–70, Napa, California, Oct. 1993. IEEE.

[7] D. Ratner, G. J. Popek, and P. Reiher. Peer replication with selective control. Technical Report CSD-960031, University of California, Los Angeles, July 1996.

[8] D. H. Ratner. Selective replication: Fine-grain control of replicated files. Master's thesis, University of California, Los Angeles, Mar. 1995. Available as UCLA technical report CSD-950007.

[9] P. Reiher, J. S. Heidemann, D. Ratner, G. Skinner, and G. J. Popek. Resolving file conflicts in the Ficus file system. In *USENIX Conference Proceedings*, pages 183–195. USENIX, June 1994.

[10] P. Reiher, J. Popek, M. Gunter, J. Salomone, and D. Ratner. Peer-to-peer reconciliation based replication for mobile computers. In *Proceedings of ECOOP'96 II Workshop on Mobility and Replication*, July 1996.

[11] M. Satyanarayanan, J. H. Howard, D. A. Nichols, R. N. Sidebotham, A. Z. Spector, and M. J. West. The ITC distributed file system: Principles and design. In *Proceedings of the Tenth Symposium on Operating Systems Principles*, pages 35–50. ACM, Dec. 1985.

[12] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere. Coda: A highly available file system for a distributed workstation environment. *IEEE Transactions on Computers*, 39(4):447–459, Apr. 1990.

[13] M. Satyanarayanan, J. J. Kistler, L. B. Mummert, M. R. Ebling, P. Kumar, and Q. Lu. Experience with disconnected operation in a mobile computing environment. In *Proceedings of the USENIX Symposium on Mobile and Location-Independent Computing*, pages 11–28, Cambridge, MA, Aug. 1993. USENIX.

[14] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. 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, Dec. 1995. ACM.