The Ward Model: A Scalable Replication Architecture for Mobility*

David Ratner Gerald J. Popek† Peter Reiher

Department of Computer Science

University of California, Los Angeles

1 Introduction

The ability to replicate data objects 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 or reside at multiple locations, replicating them at all the interested hosts is often the best solution.

Early replication solutions assumed stationary and relatively well-connected environments, and thus had relatively simple architectures. Even systems designed to support common network disconnections and treat them as normal occurrences [2, 1] were designed assuming a static infrastructure. Mobile users are by definition not static, but mobile users of these systems are nevertheless forced to conform to their static infrastructures by acting as if they were not mobile. Extraordinary actions are taken to conform to the underlying static environment, such as long distance telephone calls over low-bandwidth links, costing the users additional money while providing them with degraded service. The system's statically designed communication topologies stress under the strain of mobility, and become overly complicated and often severely inefficient.

Peer-to-peer solutions [1, 6] solved some of the problems, but with a static infrastructure they can still become widely inefficient in mobile environments. As such, replication in a mobile world requires more than simply a peer-to-peer solution. It requires a new underlying architecture that supports both scalability and mobility in a dynamic and efficient manner. This position paper describes both the rationale behind the new architecture and the basics of the architecture itself, which we call the *ward* model.

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
- Replication at the object level
- Inherent support for wide-scale replication
- Mobile-transparent and low-maintenance solutions

Each of these will now be discussed.

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. Availability is therefore greatly increased. Of course, optimistic solutions must reliably

*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*.

detect concurrent updates and resolve the resulting *conflicts*, but recent studies have demonstrated that, in many environments, conflicts are rare and resolution is straight-forward, if not automatic [3, 5].

Peer-based algorithms [6], 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 wishing to communicate must do so through a third party. Peer solutions allow any set of replicas to directly communicate with each other, facilitating *mobile work-groups* and other dynamic collections of communicating replicas. However, peer algorithms have typically not scaled as well as the client-server solutions. Thus, while the algorithms must require support for peer-to-peer interaction, they must also share the scaling properties of the client-server solutions.

2.2 Replication at the object level

Many replication systems have provided replication based on large collections of objects. While such strategies may have been acceptable for static environments, mobility requires a more dynamic and flexible solution. First, mobile computers typically have less capacity than their stationary counterparts, and certainly lack the typical resources (e.g. fast access to a large disk pool) commonly associated with residing on a LAN. A desktop can afford both to replicate more than it absolutely requires and to forgo replication entirely in favor of accessing fast and inexpensive nearby replicas. Mobile machines have neither luxury; replication onto the mobiles must therefore be more selective in terms of which objects get locally replicated and the size of the minimal replication set. This is often referred to as *selective replication* [4]. Selective replication allows individual objects to be replicated without replicating the entire container or set of related objects.

Second, mobile machines by their very nature have the ability to encounter more data objects than static machines. Being mobile allows the machines to interact in many different environments, thereby discovering new objects. If replication forced that the replication of a single object forced the replication of all other objects in its container-set, the model could not possible be viable in a mobile world. There is simply too much available data. This is similar to the problems faced by static replication models and the vast expansion of the Internet—this position paper provides solutions to aid replication problems in these environments as well.

2.3 Wide-scale replication

Many replication systems have only considered environments that allow a handful of replicas of any single object. Mobility and mobile environments seem to negate the assumption that a handful is enough. While this paper does not argue for large-scale replication in terms of thousands of read-write copies, it does suggest that the type of environments common today and envisioned for the near future require replication factors to scale better than current systems allow. Mobile users clearly need a local replica on their laptop; if every user adds a second replica from their office environment to their laptop, replication factors suddenly double. Additionally, replication factors can often be kept small in office environments due to LAN sharing and NFS; one machine can store a replica that every machine in the office can access quickly, cheaply and easily. This is not the case in mobile environments. Mobile users require local replicas of critical information, and in most cases desire local replicas of non-critical objects as well for monetary and latency reasons. Thus, it would appear that mobile environments require much higher replication factors than traditional static environments. The ward architecture, described below, is designed to scale from one to several hundred replicas of any single data object.

2.4 Mobile-transparent and low-maintenance solutions

The system must whenever possible behave as if the mobile machine were not mobile. Special actions taken to accommodate or enable mobility should be hidden from users to facilitate easy system operation. While the underlying architecture may be more complex than that required by a static replication system, it should not appear so to the average user.

Additionally, the system must necessarily be low-maintenance and require little system administrator interaction. Mobile users are often off-site and therefore geographically separated from the system administrators; therefore, solutions that rely on common administrator intervention are not viable in a mobile environment. Users need to be able to understand and operate with their own system when traveling; otherwise, the system hinders mobility, rather than facilitating it.

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 a useful model for mobility. The concepts and architecture are currently being implemented in C++ at UCLA.

In the model, replicas are organized into dynamically formed groups called *wards*, or wide-area replication domains. Each ward is a collection of "nearby" replicas, geographically co-located in some area and possibly only loosely connected (i.e. disconnection is an expected occurrence within a ward). 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 communicate with one another, instead of communicating through the server, supporting *intra-ward mobility.* Intra-ward mobility occurs when users are mobile within a specific geographic area, such as within a building, a college campus, or a small city. Mobility occurs just enough to make ward master access become expensive or difficult, but not enough to justify changing wards all together.
- Since all ward members are peers, any ward member can serve as the ward master. Automatic re-election and *ward-master reconfiguration* occurs should the ward master fail or become unavailable. While correctness is not affected by a ward master failure, operationally the users of the ward will see better functionality and improved consistency should the ward master be available most of the time.
- The ward master is not required to store a superset of all replicated objects found within the ward, though it is required to at least be able to identify the set of all intra-ward objects.

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 any one replica. Ward masters store some additional knowledge. In a two-level ward model (i.e. there are only wards and ward masters, but no "super-wards" or other collections of wards) the ward master stores the identities of other ward masters. In a generic multi-level ward model, the ward master stores the identities of the higher-level ward to which it belongs. The two-level scenario is simply a special case of the generic model. The model does not scale perfectly: since ward masters must be knowledgeable of all objects replicated within their ward, there clearly is a hierarchical level such that the ward master cannot store all the required data. The specific level depends on the types of replicated data and the particular applications. Nevertheless, the model appears to scale quite nicely in small numbers of ward levels, and easily provides replication factors in the hundreds, which appears more than adequate given that few environments if any call for thousands of fully functional, read-write replicas.

Wards are formed at replica creation time. A heuristic is applied to determine if the new replica should join an existing ward or form a new ward, based on geographic location, the available bandwidth to the other replicas, and latency. New ward formation occurs simply by creating a new ward master and identifying a new ward. Ward destruction automatically occurs when the last replica in a given ward is destroyed, thus nullifying the need for the particular ward.

Two operations are provided to support inter-ward mobility. The first is called *ward overlapping*, the second is *ward changing*, collectively referred to as *ward motion.* Ward motion enables peer-to-peer communication between any two replicas in the ward model, regardless of their current ward membership.

Ward overlapping provides the ability to belong to more than one ward simultaneously, thus being able to directly communicate with members of each. Intended as a very light-weight mechanism for inter-ward mobility, ward overlapping provides support for users who find themselves temporarily in a new geographic location, but with the intent of returning to their original ward. Each replicas stores the identities of all ward members to facilitate peer-to-peer interaction with them; ward overlapping simply adds the members of a new ward to the replica's knowledge base. When the machine returns to the original ward, it "un-overlaps" by forgetting the identities of the overlapped-ward's members. Neither ward master takes any special action to support ward overlapping; in fact, the original ward never even learns that ward overlapping occurred, making it a light-weight operation both to perform and to undo.

In contrast, ward changing involves semi-permanently changing one's ward membership from one ward to another. While a more heavy-weight process, ward changing provides better user-end performance when the geographic change is semi-permanent, or at least a non-trivial amount of time. The moving replica actually changes its notion of the current ward members, forgetting all information from the previous ward. The new ward master learns about any new objects that may exist within the ward, carried in by the new replica; the old ward master learns that there is one less replica within the ward. Ward changing better utilizes the disk space, since the replica in question doesn't need to maintain information regarding two wards, but is more heavy-weight and more costly to reverse.

In both cases, ward motion occurs automatically. Ward overlapping is the assumed default, as it is a light-weight operation. However, if the system notices that the mobility seems more permanent, it automatically upgrades the ward motion type to an actual ward change. Users are therefore transparently provided with exactly the level of required service: light-weight for short mobile outings, and better disk utilization for longer trips. Additionally, since ward formation is itself dynamic, users can easily form *mobile-work-groups* by identifying a set of mobile replicas as a new (possibly temporary) ward. Ward motion and dynamic ward formation and destruction provide easy and straightforward communication between any set of replicas in the entire system.

4 Conclusions

Mobile environments are significantly different from static ones. These differences lead to different requirements for the replication services—requirements that the current replication systems cannot meet. Characteristics such as object-level replication and large replication factors must be present to facilitate proper replication in a mobile world. The algorithms must be optimistic and support a peer-to-peer framework for the systems to be easily usable. The ward model has these qualities, and as such is a suitable architecture for replicating arbitrary types of data in a mobile environment.

Additionally, the ward architecture is well-suited to non-mobile environments with the above characteristics. With the popularity of the World Wide Web and the rapid growth of the Internet, users are increasingly finding interesting and important data which happens to be distributed around the globe. Replication seems the obvious method for maintaining and using such data, and the ward architecture is well-suited for this problem as well. As with other areas of computer science, a general solution that works well in both environments (mobile and non-mobile) seems better and easier than two different and possibly not inter-operable solutions.

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

[3] 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.

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

[5] 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.

[6] 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.