The Performance of Weak-consistency Replication Protocols

Richard A. Golding
Darrell D. E. Long

UCSC–CRL–92–30

July 6, 1992

Concurrent Systems Laboratory
Computer and Information Sciences
University of California, Santa Cruz
Santa Cruz, CA 95064

Weak-consistency replication protocols can be used to build wide-area services that are scalable, fault-tolerant, and useful for mobile computer systems. We have developed the *timestamped anti-entropy* protocol, which provides reliable eventual delivery with a variety of message orderings. Pairs of replicas periodically exchange update messages; in this way updates eventually propagate to all replicas. In this paper we present a detailed analysis of the fault tolerance and the consistency provided by this protocol. The protocol is extremely robust in the face of site and network failure, and it scales well to large numbers of replicas.

1 Introduction

We are investigating an architecture for building distributed services that emphasizes scalability and fault tolerance. This allows applications to respond gracefully to changes in demand and to site and network failure. It also provides a single mechanism to support wide-area services and mobile computing systems. It uses weak-consistency replication techniques to build a flexible distributed service.

We use data *replication* to meet availability demands and enable scalability. The replication is *dynamic* in that new servers can be added or removed to accommodate changes in demand. The system is *asynchronous*, and servers are as *independent* as possible; it never requires synchronous cooperation of large numbers of sites. This improves its ability to handle both communication and site failures.

Eventually or *weakly consistent* replication protocols do not perform synchronous updates. Instead, updates are first delivered to one site, then propagated asynchronously to others. The value a server returns to a client read request depends on whether that server has observed the update yet. Eventually, every server will observe the update. Several existing information systems, such as Usenet [1] and the Xerox Grapevine system [2], use similar techniques.

Delayed propagation means that clients do not wait for updates to reach distant sites, and the fault-tolerance of the replicated data cannot be compromised by clients that misbehave. It also allows updates to be sent using bulk transfer protocols, which provide the best efficiency on high-bandwidth high-latency networks. These transfers can occur at off-peak times. Replicas can be disconnected from the network for a period of time, and will be updated once they are reconnected. On the other hand, clients must be able to tolerate some inconsistency, and the application may need to provide a mechanism to reconcile conflicting updates.

Large numbers of replicas allow replicas to be placed near clients, and spread query load over more sites. This decreases both the communication latency for client requests and the amount of long-distance traffic that must be carried on backbone network links. Mobile computing systems can maintain a local replica, ensuring that users can use access information even when disconnected from the network.

These protocols can be compared with consistent replication protocols, such as voting protocols. Consistent protocols cannot practically handle hundreds or thousands of replicas, while weak-consistency protocols can. Consistent protocols require the synchronous participation of a large number of replicas, which can be impossible when a client resides on a portable system or when the network is partitioned. It is also difficult to share processing load across many replicas. The communication traffic and associated latency are often unacceptably large for a service with replicas scattered over several continents.

1.1 The Internet environment

The Internet has several behaviors that must be considered when designing a wide-area distributed service. These include the latency required to send messages, which can affect the response time of an application, and communication unreliability, which may require robust communication protocols. For example, two hosts on an Ethernet can exchange a pair of datagrams in a few milliseconds, while two hosts on the same continent may require 50–200 milliseconds. Hosts on different continents can require even longer. Packet loss rates of 40% are common, and can go much higher [3]. The Internet has many single points of failure, and at any given time it is usually partitioned into several non-communicating networks. This is a difficult environment for building distributed applications.

The application must also handle the vast number of users that can access a widely-available service. The Internet now includes more than 900 000 hosts [4]; the potential user base is in the millions, and these numbers are expected to increase rapidly. The **archie** anonymous FTP location service reported on the order of 10 000 queries per day (0.12 queries per second) using two servers in November 1991 [5]. Services used by a wider audience would observe load several orders of magnitude greater. We expect to find services in

the near future processing several hundred queries per second, a greater load than can be handled by single computer systems or individual network links.

Despite this environment, users expect a service to behave as if it were provided on a local system. The response time of a wide-area application should not be much longer than that of a local one. Further, users expect to use the service as long as their local systems are functioning. This is an especially difficult expectation to meet on portable systems, where the system may be disconnected from the network for a long time or may be "semi-connected" by an expensive low-bandwidth connection. Several researchers are investigating file systems that can tolerate disconnection [6, 7].

We assume that server processes have access to pseudo-stable storage such as magnetic disk that will not be affected by a system crash. Sites also have loosely synchronized clocks. Sites and processes fail by crashing; that is, when they fail they do not send invalid messages to other processes and they do not corrupt stable storage. Processes can temporarily fail and then recover. Sites have two failure modes: temporary recoverable failures, and permanent removal from service. The network is sufficiently reliable that any two processes can eventually exchange messages, but it need never be free of partitions. *Semi-partitions* are possible, where only a low-bandwidth connection is available.

2 Protocol description

Replicated data can be implemented as a group of replica processes that communicate through a *group communication protocol*. The group communication protocol generally provides a *multicast* service that sends a message from one process to all other processes in the group. The protocol also determines the *consistency* of each replica by controlling the order messages are sent among processes.

Weak consistency protocols guarantee that messages are delivered to all members but do not guarantee when. In this section we discuss how weak consistency compares to other kinds of consistency, and summarize one weak-consistency replication protocol.

The reliable delivery guarantee is not met in one important case: when a process permanently fails and loses data. No weak consistency communication scheme can be free from this, since a window of vulnerability exists while the data is being sent to other processes. In practice the duration is insignificant. We present an analysis of information loss in Section 3.

2.1 Kinds of consistency

In general, two processes are consistent at time t if they have received the same set of messages. Different degrees of consistency place different constraints on the orders in which the messages can be delivered. Unlike some other work on distributed consistency, we always measure consistency in *real* time, rather than using a virtual time measure.

The service provided by a process depends on the messages it has received. If a replica is to provide a certain level of consistency, it must use a communication protocol that guarantees a similar level of consistency. Communication protocols can provide guarantees on *message delivery, delivery ordering,* and *time of delivery.* In general, strong guarantees require multiphase synchronous protocols while weaker guarantees allow efficient asynchronous protocols.

Messages can either be delivered *reliably,* in which case arrival is guaranteed, or with *best effort,* meaning the system will make an attempt to deliver the message but delivery is not guaranteed.

Messages will be delivered to processes in some order, perhaps different from the order in which they are received. There are several possible orders, including total, causal, bound inconsistency, FIFO channel, and receipt orders. A *total* ordering means that all processes will see the same messages in the same order, though that order will not necessarily be the order messages were sent. *Causal* ordering implies that any messages with a potential causal relation will be delivered in the same order at all replicas [8, 9]. Messages with no causal relation can be delivered in different orders at different processes. A *bound inconsistency* ordering ensures that the database at one site never differs from the correct global value by more than a constant [10, 11]. Weaker orderings include a *per-process* or *FIFO channel* ordering, where the messages from any particular process are delivered in order, but the streams of messages from different processes may be interleaved arbitrarily. Finally, there is the possibility of simply delivering messages in the order they are received, without regard for order.

The communication protocol can deliver messages *synchronously,* within a *bounded* time, or *eventually* in a finite but unbounded time.

Strong consistency requirements are impossible to meet in the most general cases. For example, if there are no bounds on message delivery time it is not possible to guarantee consistency [12]. Further, if processes can fail in arbitrary ways, providing reliable delivery is equivalent to Byzantine Agreement. For most applications the Internet can be treated as an unreliable, bounded, broadcast (as opposed to strict point-to-point) network.

The weak consistency protocols we have developed provide reliable delivery, and can be modified to produce several delivery orderings, but only guarantee eventual message delivery. In particular, there is a non-zero probability that two processes have received all the same messages, and all processes are

guaranteed to agree in finite but unbounded time if no further messages are sent.

Grapevine [2] was one of the first wide-area systems to use weak consistency. In that system, replicated data was first updated at one site, then the results were propagated to other sites in the background. Updates were propagated three ways. A site might first use *direct mail,* an unreliable multicast, to get the update to as many sites as possible. Then it would use *rumor mongery* to propagate recent updates from one site to another. Finally, pairs of sites would periodically exchange all known updates in an *anti-entropy session* until they were mutually consistent. Of the three methods, only anti-entropy guaranteed delivery to all sites.

2.2 Timestamped anti-entropy

We have developed a new group communication protocol that provides reliable, eventual delivery, called *timestamped anti-entropy* [13]. Since the protocol is fault tolerant, messages will be delivered to every process in the group even if processes temporarily fail or are disconnected from the network. We have also developed a related group membership mechanism that handles adding and removing processes from the replica group [14]. The tradeoffs are that the protocol may have to delay message delivery (it is a blocking protocol), that replicas must maintain logs on disk that are not compromised by failure and recovery, and that timestamp information must be appended to every message.

Each replica maintains three data structures: a *message log* and two *timestamp vectors.* These should all be maintained on stable storage, so they are not corrupted when the site or process crashes. Stable storage is required to strictly meet reliability guarantees; however, we have found that careful buffering in volatile storage does not appreciably compromise this guarantee. Each site must also maintain a clock that is loosely synchronized with other sites.[1]

The message log contains messages that have been received by a process. The messages are stamped with the identity of the replica that initiated the message and a timestamp. Message are entered into the log on receipt, and removed when all other replicas have received them.

Replicas maintain a *summary timestamp vector* to record what updates they have received. The timestamp vector holds one timestamp for every replica in the group. Replica A records a timestamp t for replica B when replica A has received all update messages sent from B up to time t. The vector provides a fast mechanism for transmitting summary information about the state of a replica.

Each replica also maintains an *acknowledgment timestamp vector* to record what messages have been acknowledged by other replicas. A replica can determine that every other replica has observed a particular message by looking only at its local acknowledgment vector. If replica A holds a timestamp t for replica B, replica A knows that B has received every message from any sender with timestamp less than or equal to t. Process B periodically sets its entry in its acknowledgment vector to the minimum timestamp recorded in its summary vector. This mechanism makes progress as long as site clocks are loosely synchronized and the acknowledgment vector is updated regularly.

A replica can use the acknowledgment vector to detect when all other replicas have received an update. In particular, for an update message sent at (real) time t, there is a set M of replica processes at t. A replica process p can eventually determine that all replicas in M have either received the message or failed by comparing the message timestamp with the timestamps in its acknowledgment vector.

From time to time, a process A will select a partner process B and start an *anti-entropy session*. A session begins with the two processes allocating a session timestamp, then exchanging their summary and acknowledgment vectors. Each process determines if it has messages the other has perhaps not yet observed, by detecting that some of its summary timestamps are greater than the corresponding ones of its partner. These messages are retrieved from the log and sent to the other process using a reliable stream protocol.

[1] We have also developed a similar protocol that requires $O(n^2)$ state per process rather than $O(n)$, but allows unsynchronized clocks. This alternate protocol was discovered independently by Agrawal and Malpani [15].

The session ends with an exchange of acknowledgment messages. If any step of the exchange fails, either process can abort the session.

At the end of a successful session, both processes have received the same set of messages. Processes A and B set their summary and acknowledgment vectors to the elementwise maximum of their current vector and the one received from the other process.

After anti-entropy sessions have completed, update messages can be delivered from the log to the database, and unneeded log entries can be purged. A log entry can be purged when every other process has observed it, which is true when the minimum timestamp in the acknowledgment vector is greater than the timestamp on the log entry.

Here the protocol can produce different message delivery orderings. If the system guarantees a total or causal ordering, then only those messages with a timestamp less than the minimum timestamp in the summary vector can be delivered. Other messages will be delayed until messages arrive from other replicas. If per-process or weaker orderings are allowed, messages can be delivered immediately upon receipt.

2.3 Variations

There are several variations on the basic timestamped anti-entropy protocol. The timestamp vectors and message timestamps can be used to order messages before they are delivered from the log. Anti-entropy sessions can be augmented by a best-effort multicast to speed propagation, and replicas can use different policies to select partners.

As mentioned earlier, many different message delivery orders are possible. A total ordering is strongest, and ensures that all replicas will process all updates in the same order. This can be done by delivering messages in the order of their timestamps. When site clocks follow Lamport's happens-before condition [8], this order will respect causality as well. A message can be delivered at a replica when the summary vector has no timestamps less than the message timestamp. FIFO channel ordering results when messages from each replica are sorted by timestamp. Messages can then be delivered earlier than with total or causal orders. The delay required to correctly order messages depends on how fast messages propagate among replicas.

Best-effort multicast can be combined with anti-entropy to spread information rapidly. When a replica originates a message, it can multicast it to other replicas. Some of them will not receive the multicast, either because the network dropped the message or because the replica was temporarily unavailable. These sites will receive the message later when they conduct an anti-entropy session with another site that has received the message. This can speed dissemination when message delivery order is not important.

There are several different policies that replica processes can follow when selecting anti-entropy partners. *Random* selection is the simplest. However, other choices might improve the rate of update propagation or minimize long-distance communication. *Oldest-known* partner selection always selects the partner from which the replica has not received an update in the longest time, in the hope that this site will have the most updates that the replica has not yet observed. *Distance-biased* partner selection attempts to avoid long-distance communication as much as possible by weighting the chances of randomly selecting a partner based on its distance. Distance biasing was studied as an optimization for Grapevine [16]. The performance effects of different policy selections are reported in Section 4.

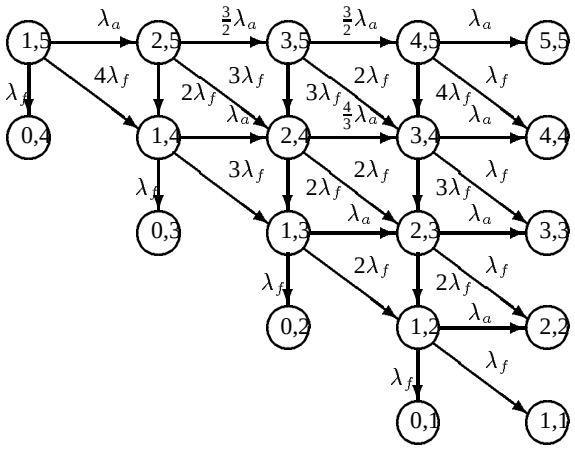

FIGURE 1: Model of failure for five processes. This model only includes permanent failure; temporary failure and recovery would add an additional dimension of states.

---

3 Fault tolerance

We first consider how often site failures would prevent a weak-consistency communication protocol from delivering a message to all replicas.

3.1 Analytical modeling

We modeled information loss using a state transition system like that shown in Figure 1. Each state is labeled with a pair $\langle m, f \rangle$, where m is the number of functioning replicas that have observed a message, and f is the total number of functioning replicas. The system starts with one replica having observed a message out of n possible (5 in the example). The system can then either propagate the information using anti-entropy, or a replica can be removed from service.

We treat anti-entropy as a Poisson process with rate λ_a. The rate of *useful* anti-entropy sessions, where a replica that has an update contacts one that does not, is a function of m and f and the partner selection policy. In particular, f replicas will be initiating anti-entropy sessions. If replicas choose their partners randomly, each replica that has observed the update has a chance $(f-m)/(f-1)$ of contacting a replica that has not yet observed the update. Since anti-entropy is a Poisson process, the rate of useful anti-entropy sessions is

$$f\frac{f-m}{f-1}\lambda_a.$$

For this evaluation we treated removal from service as a Poisson process with rate λ_f.

Since removal from service is a permanent event, the state transition graph is acyclic, with $O(n^2)$ states in the number of replicas. The probability p_i of reaching each state i can be computed using a sequential walk of the states. The probability density functions $p_i(t)$ of the time at which the system enters each state can be derived analytically or numerically. The analytic solution for $p_i(t)$ can be found by convolving the entry-time distribution $p_j(t)$ for each predecessor state j with the probability density of the time required for the transition from j to i. We obtained a numerical solution using a simple Monte Carlo evaluation.

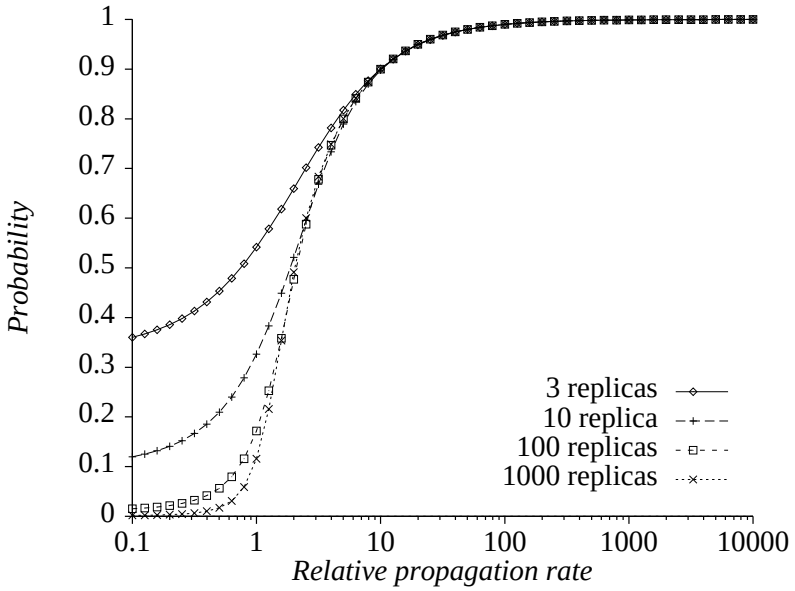

FIGURE 2: Probability of successfully delivering a message to all sites. The relative propagation rate ρ is the ratio of the anti-entropy rate to the permanent site failure rate.

3.2 Results

Figure 2 shows the probability of successful delivery for different numbers of replicas, accounting only for permanent failure. The probability is a function of $\rho = \lambda_a/\lambda_f$, the ratio of the anti-entropy rate to the permanent site failure rate. Sites generally are removed from service after several years of service, and unscheduled removals are rare. The anti-entropy rate is likely to be many thousands of times higher than the permanent failure rate. As a result, we expect that there will be no messages lost because of removal from service in a typical system.

Some implementations may buffer messages on volatile storage before copying them to stable storage. This is the default behavior of several operating systems, including Unix. These implementations will lose the information in volatile storage when a replica temporarily fails and recovers.

Volatile storage complicates the model. States must be labeled with four values: the number of functioning replicas that have not observed a message, the number that have written it to volatile store, the number that have written it to disk, and the number that have temporarily failed. The state transitions are complex and the solution is impractical for realistic numbers of replicas. We performed a Monte Carlo evaluation of the system to obtain estimates of fault tolerance.

The effect of volatile storage can be bounded by considering the probability that a failure will occur while there are unstable messages. Assume that temporary failure is a Poisson process with rate λ_t and that volatile data is flushed to stable storage every s time units. The probability that a failure occurs before writeback is

$$p = \frac{-2e^{-s\lambda_t} + s^2\lambda_t^2 - 2s\lambda_t + 2}{2s\lambda_t^2}.$$

For a typical value of $s = 30$ seconds and $1/\lambda_t = 15$ days, p is so close to zero as to be negligible.

4 Consistency and delay

Weak consistency protocols allow replicas to contain out-of-date information. There are two related measures of this effect—one concerning the propagation of an update, the other concerning the consistency of replicated values. The time required to propagate an update from one replica to others shows how quickly information will be made available to clients. The likelihood of encountering out-of-date information, and the age of this information, aggregates the effects of several updates.

4.1 Simulation modeling

We constructed two discrete event simulation models of the timestamped anti-entropy protocol: one to measure propagation delay and the other to measure information age.

The propagation delay simulator measured the time required for an update message, entered at time zero, and its acknowledgments to propagate to all available replicas. It used two events: anti-entropy propagation and permanent site failure. The simulator could be parameterized to use different partner selection policies. The simulator was run until either the 95% confidence intervals were less than 5%, or 10 000 updates had been processed. In practice 95% confidence intervals were generally between 1 and 2%.

The information age simulator was more complex. It used five events: one each to start and stop the simulation, one to send update messages, one to perform anti-entropy, and one to read data from a replica. The simulation was allowed to run for 1 000 time units so it would reach steady state. The simulation ended at 50 000 time units. In no case did the 95% confidence interval width exceed 4% of the mean. Read, write, and anti-entropy events were modeled as Poisson processes. The simulator included different partner selection protocols and an optional unreliable multicast on writes.

The simulator maintained two data structures for each replica: the anti-entropy summary vector and a data version number. It also maintained a global data version counter. Write events incremented the global version counter and copied that value to the version number for some replica. If an unreliable multicast was being used, the version number would be copied to other replicas if a simulated unreliable datagram was received. Anti-entropy events propagated version numbers between replicas, as well as updating the replicas' summary vectors.

Read events were used to collect measures of the expected age of data and the probability of finding old data. A replica was selected at random, and the version number for that replica was compared to the global version. The difference showed how many updates the replica had yet to receive.

4.2 Propagation delay

The time required to propagate a message from one replica to others was the first measure we investigated. If information is propagated quickly, clients using different replicas will not often observe different information, and loss of an update from site failure will be unlikely. The size of the message log is related to this measure, since messages are removed from the log when acknowledgments have been received from every replica.

Figure 3 shows the cumulative probability over time that an update has been received by all replicas. Time is measured as multiples of the mean interval at which replicas initiate anti-entropy events. Anti-entropy was assumed to occur 1 000 times as often as permanent site failure—as we have noted, a low estimate. The simulations in this graph use random partner selection with no initial multicast.

This figure also shows that our weak-consistency protocols scale well to large numbers of sites. The propagation delay increases roughly as the logarithm of the number of sites.

We also considered the effect of different partner selection policies on the speed of propagation. Figure 4 shows the distribution for different policies, using 500 sites and without initial multicast. Random and distance-biased selection are nearly nearly identical, while the oldest-first policy is generally somewhat

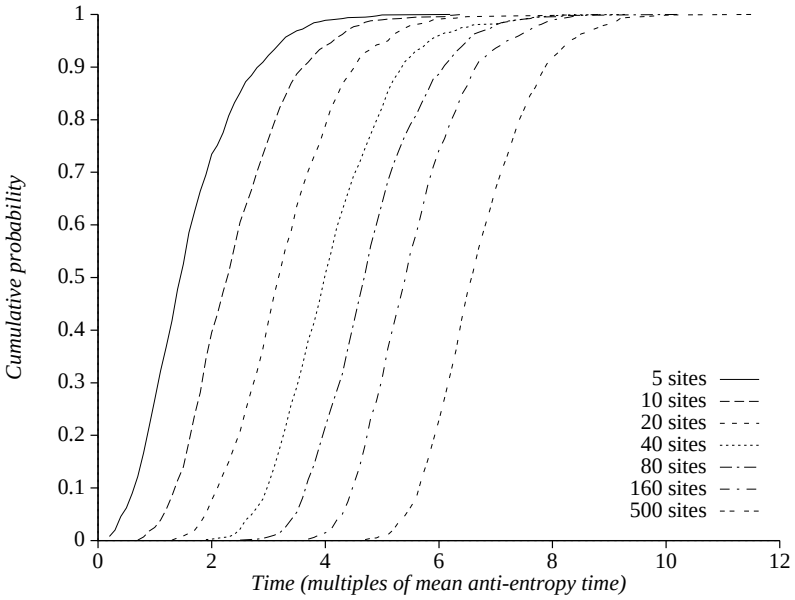

FIGURE 3: Cumulative probability distribution for propagating a message to all replicas. Measured for random partner selection with $\rho = 1000$.

slower than the others. The speed of acknowledgment propagation is also important. We have found that acknowledgment delay scales as well as propagation delay, and that the oldest-first policy is slower than the other two.

4.3 Age of information

While propagation delay measures the performance of one update, this measure shows the consistency of concurrent updates on a single data item. We took two measures: the probability that a replica would return old information, regardless of its age, and the expected age of information whether it was current or not.

The age of a database entry depends on the ratio of the anti-entropy rate to the update rate for that entry. Many wide-area services have extremely low update rates; some services write new entries and never change them. A low update rate means that anti-entropy has a better chance of propagating an update before another update enters the system. In the Domain Name Service [17], a host name or address rarely changes more than once every few months. In other databases new entries are added, corrected quickly, then remain stable. We expect the update rate for a single database entry to be about a thousand times lower than the anti-entropy rate. Most of the graphs in this section were generated using a mean time-to-update of 1 000 time units; the maximum anti-entropy rate investigated was only 200 times greater, giving a mean time-to-anti-entropy of five.

Figure 5 shows the likelihood of getting out-of-date information, while Figure 6 shows the expected age of that information. Clearly, adding an unreliable multicast on write significantly improves both measures. We have found that the message success probability is the most important influence on information age in large groups of replicas. For small numbers of sites, increasing the anti-entropy rate dramatically improves both the probability of getting up-to-date information and the expected age.

Figures 7 and 8 show how consistency depends on the number of sites. For these simulations the anti-entropy rate was fixed at 100 times that of writes. We expect this value might be typical for a database

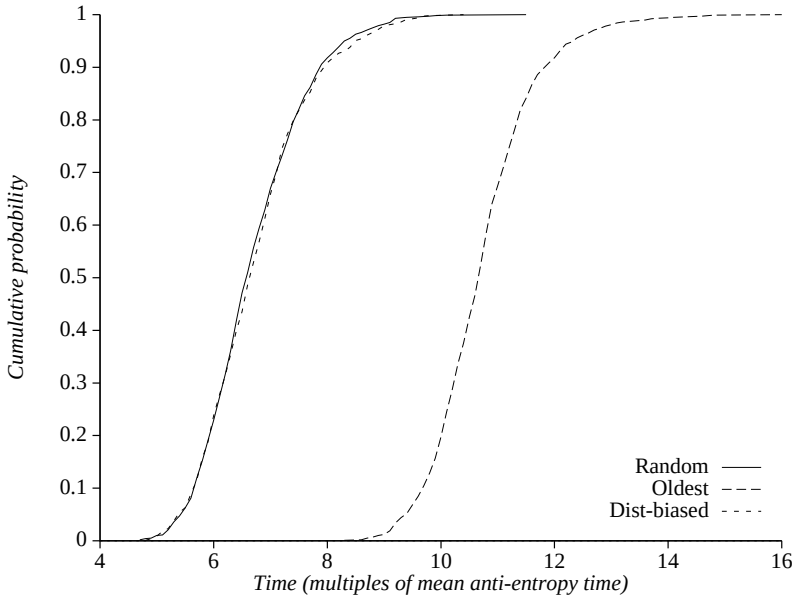

FIGURE 4: Effect of partner selection policy on propagation delay. Measured for 500 sites with $\rho = 1000$.

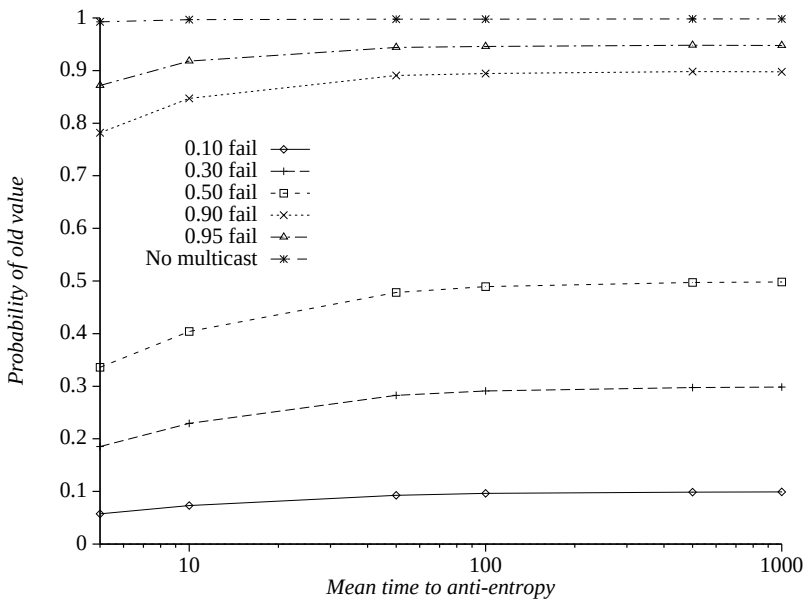

FIGURE 5: Probability of getting old value as anti-entropy rate varies, for 500 sites. Mean time-to-write 1 000; random partner selection.

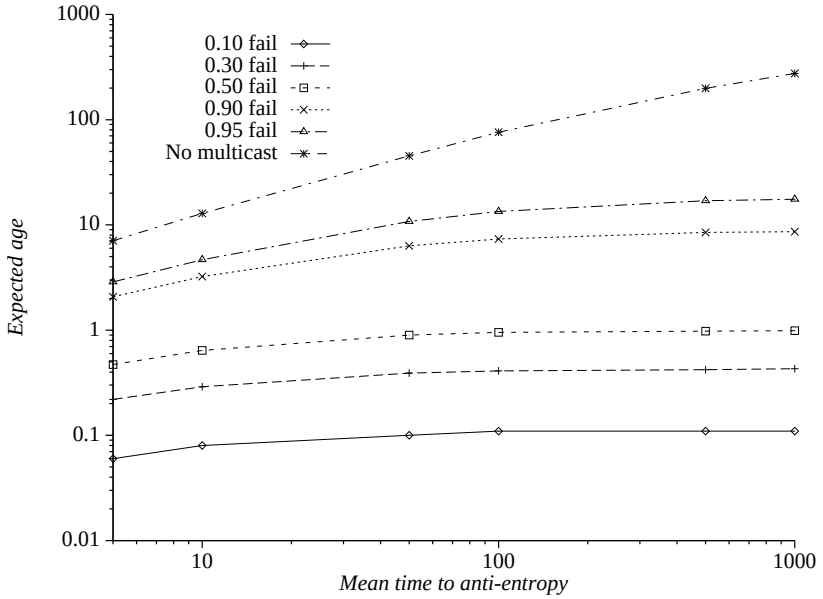

FIGURE 6: Expected data age as anti-entropy rate varies, for 500 sites. Mean time-to-write 1 000; random partner selection.

---

entry soon after it is entered, when updates are most likely. Later updates will be less frequent and the ratio will increase, improving the consistency. Once again an unreliable multicast provides considerable improvement.

We also investigated the effect of partner selection policy on information age, as shown in Figure 9. Since random and distance-biased selection showed identical propagation rates, it is no surprise that they have the same expected age. Oldest-first selection provides slightly older information, consistent with a slower propagation rate.

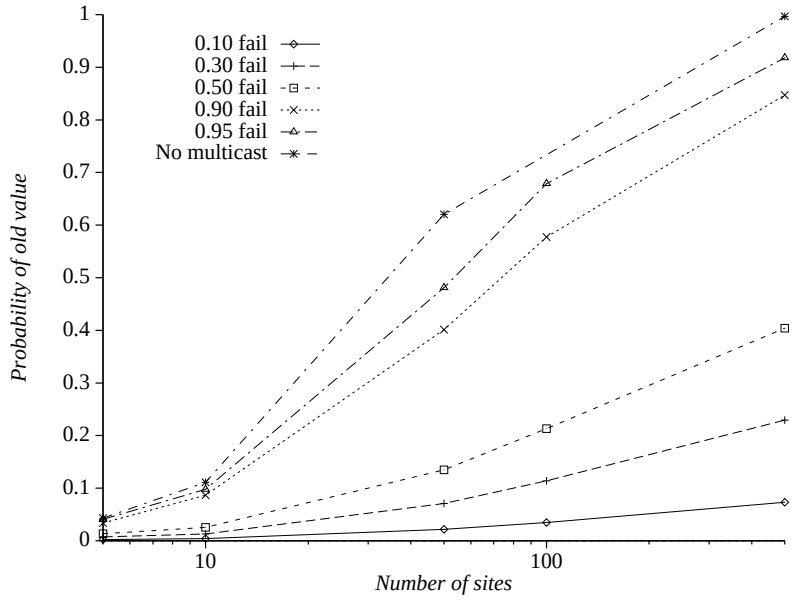

FIGURE 7: Probability of getting old value as the number of sites varies, with anti-entropy occurring 100 times as often as writes. Random partner selection.

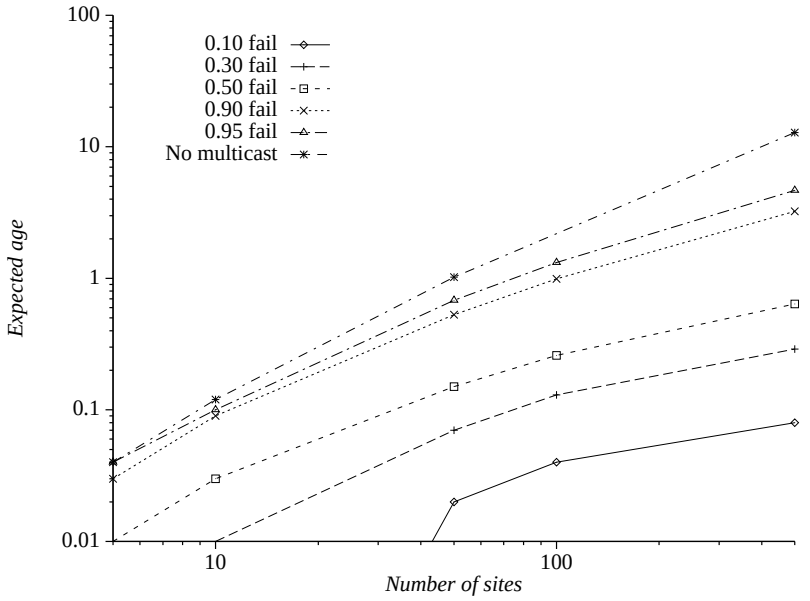

FIGURE 8: Expected data age as the number of sites varies, with anti-entropy occurring 100 times as often as writes.

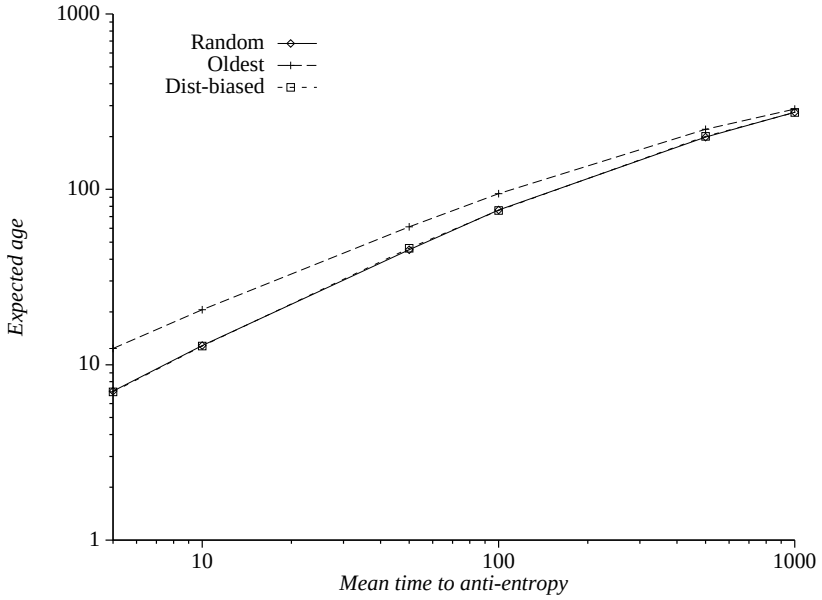

FIGURE 9: Effect of partner selection policy on expected data age. 500 sites; mean time-to-write 1 000.

5 Conclusions

Scalability, fault tolerance, and support for mobile computing are some of our goals for a wide-area distributed service architecture. Voting protocols cannot meet this goals in a wide-area network, so we have adopted weak-consistency replication protocols.

We have found that the timestamped anti-entropy protocol scales well to large numbers of replicas. The time required to propagate an update from one replica to all other replicas increases as the log of the size of the group. Each replica need only store $O(n)$ state in the group size.

Fault tolerance is the ability of a system to provide correct service even when some parts fail. The timestamped anti-entropy protocol avoids synchronous communication, instead communicating between pairs of replicas. When a site is partitioned from the rest of the network, it can continue to provide service and will receive updated information once it reconnects. Likewise, no special protocols are required for recovery from temporary failure.

These same features make timestamped anti-entropy ideal for mobile computing systems. Users can place replicas on their portable computers. The protocols can cope with the large number of replicas this produces, and will ensure they are updated when connected to the network.

This can be contrasted with consistent replication protocols, such as voting, that require synchronous communication with a majority of replicas. While consistent protocols can provide good availability and performance with small numbers of replicas, they are not practical for hundreds or thousands of replicas. Consistent replicas cannot continue to function when disconnected from other replicas, so they are not useful for mobile computing systems.

The negative aspect of weak consistency protocols is that replicas may return out-of-date information. Our investigation finds that an unreliable multicast can mitigate most of this problem, and that at reasonable propagation rates replicas are rarely more than a few updates behind. Many applications, including name services and bibliographic databases, are not concerned with consistency.

We are encouraged by the performance of the distance-biased partner selection policy. Similar policies can be used in the Internet to encourage traffic between nearby sites and to avoid saturating long-distance links. The random policy appears to be within a constant factor of optimal [18]. We are investigating its optimality, as well as several other selection policies.

Acknowledgments

John Wilkes, of the Concurrent Systems Project at Hewlett-Packard Laboratories, and Kim Taylor, of UC Santa Cruz, assisted the initial development of these protocols. George Neville-Neil, Bruce Montague, and Mary Long gave helpful comments on this paper.

The analytic solutions were found with the aid of *Maple*, a symbolic algebra program developed by the Symbolic Computation Group at the University of Waterloo. Many of the simulation results were obtained with the aid of SIMSCRIPT II.5, a simulation language developed and supported by CACI Products Company of La Jolla, California.

Richard Golding was supported in part by the Concurrent Systems Project at Hewlett-Packard Laboratories, and by a graduate fellowship from the Santa Cruz Operation. Darrell Long was supported in part by the National Science Foundation under Grant NSF CCR-9111220.

References

[1] J. S. Quarterman and J. C. Hoskins, "Notable computer networks," *Communications of the ACM*, vol. 29, pp. 932–71, October 1986.

[2] M. D. Schroeder, A. D. Birrell, and R. M. Needham, "Experience with Grapevine: the growth of a distributed system," *ACM Transactions on Computer Systems*, vol. 2, pp. 3–23, February 1984.

[3] R. A. Golding, "Accessing replicated data in a large-scale distributed system," Master's thesis, Computer and Information Sciences Board, University of California at Santa Cruz, June 1991. Published as Tech. Rep. UCSC–CRL–91–18.

[4] D. D. E. Long, J. L. Carroll, and C. J. Park, "A study of the reliability of Internet sites," in *Proceedings of 10th IEEE Symposium on Reliable Distributed Systems*, pp. 177–86, September 1991.

[5] A. Emtage and P. Deutsch, "archie – an electronic directory service for the Internet," in *Proceedings of Winter 1992 Usenix Conference*, pp. 93–110, January 1992.

[6] J. J. Kistler and M. Satyanarayanan, "Disconnected operation in the Coda file system," in *Proceedings of 13th ACM Symposium on Operating Systems Principles*, pp. 213–25, October 1991.

[7] R. Alonso, D. Barbará, and L. L. Cova, "Using stashing to increase node autonomy in distributed file systems," in *Proceedings of 9th IEEE Symposium on Reliable Distributed Systems*, October 1990.

[8] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, vol. 21, no. 7, pp. 558–65, 1978.

[9] R. Ladin, B. Liskov, and L. Shrira, "Lazy replication: exploiting the semantics of distributed services," *Operating Systems Review*, vol. 25, pp. 49–55, January 1991.

[10] C. Pu and A. Leff, "Replica control in distributed systems: an asynchronous approach," Tech. Rep. CUCS–053–090, Department of Computer Science, Columbia University, January 1991.

[11] D. Barbará and H. Garcia-Molina, "The case for controlled inconsistency in replicated data," in *Proceedings of the Workshop on the Management of Replicated Data*, pp. 35–8, November 1990.

[12] J. Turek and D. Shasha, "The many faces of consensus in distributed systems," *IEEE Computer*, vol. 25, pp. 8–17, June 1992.

[13] R. A. Golding, "The timestamped anti-entropy weak-consistency group communication protocol," Tech. Rep. UCSC–CRL–92–29, Computer and Information Sciences Board, University of California at Santa Cruz, July 1992.

[14] R. A. Golding and K. Taylor, "Group membership in the epidemic style," Tech. Rep. UCSC–CRL–92–13, Computer and Information Sciences Board, University of California at Santa Cruz, April 1992.

[15] D. Agrawal and A. Malpani, "Efficient dissemination of information in computer networks," *Computer Journal*, vol. 34, pp. 534–41, December 1991.

[16] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry, "Epidemic algorithms for replicated database maintenance," *Operating Systems Review*, vol. 22, pp. 8–32, January 1988.

[17] P. Mockapetris, "Domain names – concepts and facilities," Tech. Rep. RFC 1034, ARPA Network Working Group, November 1987.

[18] N. Alon, A. Barak, and U. Manber, "On disseminating information reliably without broadcasting," in *Proceedings of 7th International Conference on Distributed Computing Systems*, pp. 74–81, 1987.