Weak consistency group communication for wide-area systems

Richard A. Golding
Computer and Information Sciences
University of California, Santa Cruz

Abstract
Replicated services can be implemented as process groups. Member processes use group communication protocols to communicate amongst themselves and group membership protocols to determine what processes are in the group. These protocols can provide various levels of consistency between members. I am investigating weak consistency protocols that guarantee that messages are delivered to all members, but do not guarantee when. I report on a new family of communication protocols, an associated group membership mechanism, and current progress in evaluating their efficiency and utility for real applications.

1 Introduction
A replicated service can be implemented as a set of processes forming a process group. The group members communicate by multicasting messages using a group communication protocol. Processes join and leave the group using a group membership protocol. The communication protocol guarantees that members will have a particular degree of consistency in their view of the messages that have been sent. Group membership and communication protocols are closely related: the membership protocol usually uses some form of the communication protocol to send membership information to processes, and the communication protocol uses membership information to identify what processes should receive messages.

Systems such as Isis have shown that group communication is a useful abstraction for building replicated services. A client of a replicated service can then become a group member, perhaps in a special role, or it can send its requests to a group member that will forward them to the group. The challenge is to find the most efficient ways to implement the group. Other researchers are considering strong consistency guarantees; I am exploring the weakest guarantees to determine what performance advantages they can provide, and what applications can use them [7].

Throughout this paper I will use the term "process" to refer to a process, running at some site, that provides replica service. I assume that processes have stable storage and a loosely-synchronized clock. They can either fail and recover, or are permanently removed from service. I assume the network is sufficiently reliable that any two processes can eventually exchange messages, but it need never be free of partitions. Many problems in group communication are consensus problems, so one must assume network and processors behavior that makes consensus possible [11]. The Internet generally approximates a system synchronous processors with bounded communication.

2 Weak-consistency protocols
The service provided by a process depends on the messages it has received, so I will only discuss communication consistency. Communication protocols can provide guarantees on:

1. Message delivery: reliable or best effort.
2. Delivery ordering: total, causal, per-process, or unordered.

In general, strong guarantees generally require expensive protocols while weaker guarantees allow more efficient protocols. Figure 1 lists a taxonomy of several existing group communication and replication protocols.

Weak consistency schemes can provide any delivery ordering, but only guarantee that messages are delivered eventually. In general, there is a non-zero probability that two processes have received all the same messages, and eventually all processes are guaranteed to agree if no further messages are sent.

Wide-area group communication systems can exploit eventual delivery in several ways. Since client operations can be performed first at one site, then propagated to other sites, clients need only communicate with a single nearby site. Delayed propagation means that clients do not wait for distant sites to be updated. It also allows messages to be transferred using bulk communication protocols, which are much more efficient on high-bandwidth high-latency networks. These transfers can occur at off-peak times. Finally, since the service remains available as long as a client can contact some replica, it can have extraordinarily high...
availability. Of course, the application must be able to tolerate inconsistency, and may need to provide a mechanism for reconciling conflicting update operations.

Grapevine [5] was one of the first widespread systems to use weak consistency. In that system, replicated data was updated first 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 when they were mutually consistent. Of the three methods, only anti-entropy guaranteed delivery to all sites.

3 Group communication

I have developed a new family of group communication protocols based on anti-entropy that provide reliable, eventual delivery: the timestamped anti-entropy protocols [7]. They can provide several different message delivery orderings, including total (but not causal), per-process, or no ordering. Causal orderings are possible if process clocks meet Lamport’s happens-before condition.

These protocols provide one new feature not provided by others: a process can detect when all other processes in the group have received a message. In particular, assume an update occurs at (real) time t, and let M be the set of processes that were members at time t. A group member process p can eventually determine that all processes in M have either observed the update or failed.

This feature allows processes to perform consistent operations on distributed data even though the communication protocols do not provide it. This is important, for example, when purging message logs. Existing weak consistency protocols rely on probabilistic methods to detect consistency, and hence must either accept erroneous results, or delay consistent operations for long periods. The group membership protocols I will present use this guarantee to detect when all group members have observed a membership change.

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 must exist while the data is being sent to other processes. In practice the duration can often be reduced by disseminating the new updates rapidly, but real networks do not allow complete certainty.

The timestamped anti-entropy protocols maintain three data structures: a message log and two timestamp vectors. They require that each process maintain 2n timestamps.¹

¹ 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 [1].
its summary timestamps with those of its partner. The messages are then exchanged using a reliable stream protocol. The session ends with an exchange of acknowledgment messages, and both processes update their timestamp vectors. If any step of the exchange fails, either process can abort the session.

This mechanism guarantees that every process will eventually observe all updates. Its performance can be improved by initially sending an unreliable multicast to other processes. Several different policies can be used to select partners, and these policies affect the time required to propagate messages.

4 Group membership

The group communication protocol must know what processes are in the group. The group membership protocol provides mechanisms for listing group membership, creating a new group, joining and leaving a group, and recovering from member failure [6].

Group membership protocols can use weak consistency for scalability and efficiency. Each process maintains an eventually-consistent view of the membership, indicating the status of each member. Views are updated during anti-entropy sessions, and eventually all processes can reach agreement if the membership stops changing.

To initialize a new group, a process $p$ creates a new view with only itself. To join a group, $p$ finds a sponsor process $s$ in the group. Process $s$ inserts $p$ into its view, and sends the view and group state to $p$. To leave a group, $p$ changes its status to leaving, then waits for every other process to observe the change. While waiting it performs anti-entropy sessions but does not send any further messages. To recover from failure, member processes must be informed by an outside mechanism that some member has failed. The process is marked as failed, and removed when every process has observed the change.

Inconsistent group views can make a system vulnerable to failure. Since a process can only contact processes in its view, the mechanism will fail if the only process to know about another fails. The "knows-about" graph is correct if the transitive closure of all views is equal to the group membership. To ensure the knows-about graph stays correct after up to $k$ failures, the minimum vertex-cut between any two processes must be $k + 1$ or greater. To ensure this, each process must obtain $k + 1$ sponsors when joining, and processes must wait to leave until no paths in the graph depend on them. When $p$ fails, other processes will perform anti-entropy to re-establish $k + 1$ paths.

In contrast to this system, previous group membership mechanisms ensure greater consistency of group views at the expense of latency and communication overhead. Both the ISIS system [3, 2] and a group membership mechanism by Cristian [4] are built on top of atomic broadcast protocols, and hence provide each process with the same sequence of group views. Ricciardi [10] is investigating an alternative group membership mechanism for Isis that does not use the underlying atomic broadcast. However, it still uses two- and three-phase commit protocols to maintain consistent group views. The Arjuna system [8] maintains a logically centralized group view via atomic transactions.

5 Evaluations

I am evaluating the performance and fault tolerance of these protocols, and they are being implemented as part of two different systems.

A system of $n$ replica processes can be modeled as a Markov system of $O(n^2)$ states, with each state labeled with the number of processes available and the number that have observed the update. Data loss due to permanent site failure appears to be negligible in systems like the Internet, where sites stay in service for several years. The usual approximations to stable storage, such as delayed writeback from volatile storage, also have negligible effect. I have found that the time required for a system to reach agreement increases approximately as the log of the number of processes, and that under reasonable update and read rates information is likely to propagate to most sites before it is needed. Figure 2 shows the distribution of time required to reach consistency for different numbers of processes.

While an update is propagating, some replicas will have out-of-date information and will send that information to clients, and the applications must be prepared for this. The expected data age of the system is the average number of update messages each replica has not yet received once the system is in steady state. Darrell Long and I have constructed simulations to measure the expected data age for various write and anti-entropy rates; some results are shown in Figure 3. These results show that the expected age increases roughly linearly in the number of replicas,
and that an initial unreliable multicast, even if it reaches only a few replicas, can substantially improve results.

I am currently extending this performance evaluation to include a more complex model of transient network and process failures, and to evaluate the message traffic induced by these protocols.

These protocols are being implemented in two systems, to determine how convenient the mechanisms are for constructing widely-used applications.

The refdbms system implements a distributed bibliographic database. It is based on a system in use for several years at Hewlett-Packard Laboratories [12] for sharing bibliographic information amongst a research group. Users can search databases by keywords, use references in TeX, locate copies of papers, and enter new or changed references. I am extending this system to distributed databases to widely dispersed sites. The system will provide both performance measurements of weak consistency protocols in a widely-used system, and experience in presenting weakly consistent information to users. Currently the system allows users to either limit their use to consistent information, or to use potentially inconsistent but more recent data.

The tattler system is a distributed availability monitor for the Internet [9]. It is being built by Darrell Long, at UC Santa Cruz, as part of his research into distributed system reliability. The system uses these group communication mechanisms to coordinate measurement of host availability, and to keep redundant logs of experimental results.

Acknowledgments

This work was supported in part by a fellowship from the Santa Cruz Operation, and by the Concurrent Systems Project at Hewlett-Packard Laboratories. Refdbms development has been assisted by the Mammoth Project at UC Berkeley. Darrell Long helped with the performance simu-

lators. Kim Taylor helped refine the proofs and co-authored the initial reports. George Neville-Neil provided support and suggestions.

References


