Measuring the Quality of Service of Optimistic Replication

Geoffrey H. Kuenning Rajive Bagrodia Richard G. Guy Gerald J. Popek
Peter Reiher An-I Wang

June 29, 1998

Abstract

Optimistic replication has become an important tool in modern systems, allowing both read-only and read-write object accesses to continue even in the face of network outages and disconnected mobile computing. The quality of service delivered to a user by an optimistic system has traditionally been measured in terms of the rate of conflicting updates. We show that this measure does not accurately assess the user's requirements, and propose new criteria for evaluating optimistically replicated systems.

1 Introduction

In large-scale distributed systems, data replication is a useful technique for improving the accessibility of information and surviving the unreliability of communication networks. *Optimistic* replication [2] trades off consistency against availability, allowing updates to take place even in the presence of communication failures at a cost of sometimes violating single-copy serializability.

The *quality of service* (QoS) of optimistic replication systems in the face of updates is the degree to which the system presents an illusion of connectivity to a single up-to-date copy of all objects to all users. In real replicated systems, this illusion must necessarily be violated, and quantifying the user-visible effects of such violations is key to determining how well the system performs.

We will discuss and analyze several possible metrics that can be used to measure quality of service. We ignore important questions of system load and overheads in both time and space, since in general they are relatively easy to measure and characterize. We do not consider measures of availability, which are primarily of interest in pessimistic systems.

2 Measuring Quality of Service

The literature on replicated systems has made use of only a few measures of service quality. A well-designed replicated system achieves near-100% read availability. Therefore, the most interesting metrics are those that relate to updates. However, all measures of update quality, in particular those that have been used in past studies of replicated systems, have drawbacks that make them imperfect for characterizing and comparing replication methods.

2.1 Conflict Count

Conflict count and *conflict rate* [5, 6] are the most popular measures of the QoS of replicated systems. Optimistic replicated systems allow updates at any node, at any time. Thus, sometimes some replicas are aware of new data before others. A mechanism, commonly called *reconciliation* or *synchronization*, is required for informing all unaware replicas of updates. The lack of restrictions on update patterns in optimistic systems will sometimes lead to *conflicts*, in which two or more replicas have violated one-copy serializability by basing updates to an object on a common ancestral version (as opposed to having one update based on the results of the other). In most designs, conflicts are detected during the reconciliation process. Conflicts are undesirable because they generally require automated or manual mechanisms to *resolve* the difficulty [4, 7].

The conflict count quantifies the QoS of a replicated system by counting the number of conflicts observed during a measurement period. The conflict rate expresses the same information as a ratio of the conflict count to time or total operations. The advantages of these metrics lie in their simplicity: they are simple to express, to measure, and to understand. A system with few conflicts is desirable because it requires little human intervention and usually adheres to single-copy semantics. A system with a high conflict rate, by contrast, is troublesome to the user.

However, this metric has serious drawbacks [8]. Since conflicts are detected only during reconciliation, the frequency of reconciliation has a direct effect on the number of conflicts observed. In the limit, if reconciliation never occurs, no conflicts will be detected regardless of the degree of concurrent updates! (This effect is exacerbated by the the common optimization of compressing multiple changes to the same object into a single update to save time and space during reconciliation [3, 5, 6].)

A second drawback with conflict counts is the difficulty of measuring conflicts in systems with more than two replicas [1]. The measurement can depend on the pattern of both updates and reconciliations. Consider four replicas of a single object. Replicas 1 and 3 receive independent updates without any intervening communication, creating a conflict. If these two replicas now reconcile, this single conflict will be detected. However, suppose replica 1 first reconciles with replica 2, and 3 with 4. The conflicting updates now exist in two different places. If replicas 1 and 3 then reconcile, they will count the conflict, but when replicas 2 and 4 reconcile, they will erroneously count it a second time. Adding replicas and adjusting the reconciliation patterns allows almost arbitrary choice of total conflict count.

Another difficulty is how to count unresolved conflicts. If a conflict is created at a pair of replicas, but it is not resolved even after several reconciliations, it is difficult to design an algorithm that can correctly count it as only one conflict when faced with complex update propagation patterns.

Finally, because conflict counts only measure write/write effects, they ignore the important question of whether the user is accessing out-of-date information.

2.2 Stale-Access Metrics

Since conflicts are an artifact of the reconciliation process, it is better to count the underlying cause: simultaneous non-serializable updates. We define a *stale access* as any access to a data object that is globally out of date. An access to a particular replica is stale if (a) some other replica has been updated prior to that access, as measured by a global clock, and (b) that update has not yet propagated to the replica being accessed. We can further subdivide stale accesses into *stale reads* and *stale writes*. A sequence of stale writes will eventually cause one or more conflicts to be detected during reconciliation. A stale read will not cause conflicts, but is undesirable because it delivers outdated information.

Stale-access counts provide a very attractive measure of quality of service. They directly reflect what is important to the user: up-to-date data. Furthermore, since they are based on a global view of the world, they are closely connected to the single-copy serializability semantics that are the ideal every replication system strives to achieve. In addition, unlike conflict counts, stale-access counts do not tend towards zero as the reconciliation interval increases.

An alternative is to measure the *age* of a stale access, defined as the time elapsed between the latest update to an object seen by the accessing replica, and the time of the globally last update. For example, suppose replica 1 updates an object at time t_1, and this update is communicated to all other replicas through the normal reconciliation mechanism. Then suppose that replica 2 updates the object at time $t_2 > t_1$, and replica 3 accesses it at time $t_3 > t_2$, all without further reconciliation. Then replica 3's access is stale (because it has not seen the update from replica 2), and the age of the stale access is $t_2 - t_1$, the amount of time by which the information is out of date.

The age of stale accesses is closely related to the user's desire to see only the most up-to-date information, and gives us additional information about the departure from the ideal semantics in a replication system. In many systems, an access to hours-old data is far less acceptable than one to information that is only milliseconds out of date.

Unfortunately, the global nature of stale-access metrics causes them to be unusable in live systems, due to problems with obtaining global time. Although solutions to the global-clock problem exist (e.g., GPS receivers), they are of limited accuracy and generally require special hardware that is not available on most machines, limiting the utility of these measures to simulations and systems with special equipment. Nevertheless, we believe that stale-access metrics are extremely useful in comparing various designs for replication systems, since they do not exhibit the unusual sensitivities of conflict counts.

2.3 Propagation Time

An alternative measure of timeliness is *100% propagation time*, which is the time needed for an update to an object to become visible at all replicas. This value depends on many factors in the design of a replication system, including the choice of reconciliation method (e.g., broadcast vs. epidemic), the speed of the reconciliation process, and the topology across which updates are propagated.

Although the propagation time does not directly relate to the amount of stale data seen by the user, nor to the degree of staleness, it is reasonable to assume that a replication system with a low propagation time will provide the user with better service than one with a high time. Propagation time is somewhat easier to measure in a distributed system because there is no need for a global clock; instead each replica can record the cumulative time needed to propagate an update to the next replica.

The biggest disadvantage of propagation time is the cost of measurement: logging the time of each update, and tracking its progress, requires tremendous amounts of storage space. Also, propagation time assigns equal value to all updates and all replicas. It may not much matter whether a rarely-used object propagates quickly, or whether it takes a long time to reach an infrequently-connected replica. For this reason, it can be helpful to weight the propagation time by the object's usage, and to measure the time needed to reach some substantial fraction of sites rather than 100%.

3 Conclusions

Although optimistic replication systems have been in use for some time, we have shown that there is still a lack of adequate metrics for describing the quality of service delivered to the user by these systems. This deficiency is largely due to the difficulty of defining and calculating an appropriate metric; no single measure is best for all situations.

We have discussed three classes of possible metrics. The commonly used metric, conflict count, is subject to a number of anomalies that make it easy to misuse and inaccurate in the general case. Nevertheless, we have found that the conflict count is useful both because of its simplicity and because its weaknesses are easily minimized in most real-world situations.

Stale-access metrics seem to provide the most accurate picture of the service provided to the user. However, measuring them depends on a global system view, which is available only in simulations and other artificial environments.

The final metric, propagation time, is less directly related to the quality of service perceived by the user. Nevertheless, it has utility in describing certain system characteristics, making it a useful adjunct to the traditional conflict count.

Acknowledgments

The authors are affiliated with the UCLA Computer Science Department. Dr. Popek is also affiliated with Platinum *technology*. Authors' e-mail addresses: {geoff, rguy, reiher, awang}@cs.ucla.edu, popek@platinum.com. This work was partially supported by DARPA contract N00174-91-C-0107.

References

[1] J. Heidemann, A. Goel, and G. Popek. Defining and measuring conflicts in optimistic replication. Technical Report UCLA-CSD-950033, University of California, Los Angeles, Sept. 1995.

[2] M. Herlihy. Optimistic concurrency control for abstract data types. In *Proceedings of the Fifth Annual Annual ACM Symposium on Principles of Distributed Computing*, Aug. 1986.

[3] L. B. Huston and P. Honeyman. Peephole log optimization. In *Proceedings of the Workshop on Mobile Computing Systems and Applications 1994*, Santa Cruz, CA, Dec. 1994.

[4] P. Kumar and M. Satyanarayanan. Flexible and safe resolution of file conflicts. In *Proceedings of the USENIX Conference Proceedings*, New Orleans, LA, Jan. 1995. USENIX.

[5] B. D. Noble and M. Satyanarayanan. An empirical study of a highly available file system. In *Proceedings of the 1994 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, Nashville, TN, May 1994. Also available as technical report CMU-CS-94-120, School of Computer Science, Carnegie Mellon University.

[6] T. W. Page, R. G. Guy, J. S. Heidemann, D. Ratner, P. Reiher, A. Goel, G. H. Kuenning, and G. J. Popek. Perspectives on optimistically replicated peer-to-peer filing. *Software—Practice and Experience*, 28(2):155–180, Feb. 1998.

[7] 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. University of California, Los Angeles, USENIX, June 1994.

[8] A. A. Wang, P. L. Reiher, and R. Bagrodia. A simulation framework for evaluating replicated filing environments. Technical Report CSD-970018, University of California, Los Angeles, June 1997.