Comparing Primary-Backup and State Machines for Crash Failures

Jeremy B. Sussman and Keith Marzullo
Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093–0114

Two of the more prevalent methods of constructing a highly available service are the *primary-backup* (e.g., [2]) and the *state machine* (e.g., [5]) approaches. Both methods are widely used, and a common wisdom has developed about their relative strengths and weaknesses. Our research attempts to make this comparison of relative strengths and weaknesses in a more formal manner under the crash failure model and in a synchronous system model.

The metrics upon which this comparison are based refer to the *service response time*, which is the time elapsed from when a client initially sends a request to the service until the client receives the response from the service.

Common wisdom says that in terms of best-case and expected service response times, primary-backup is better than state machines, but in terms of worst-case service response time, state machines are better than primary-backup. Hence, common wisdom argues that the state machine approach is a better choice for real-time systems in which schedulability is a concern, but primary-backup is a better choice for most other cases.

Our comparison necessitated fully specifying the two methods. The primary-backup approach has been specified [2] in terms of how a service must appear to a client, but not in terms of how a client interacts with the service except that the client only sends requests to the process it believes to be the primary. We have considered three different client-service protocols. It is easy to show that the primary-backup approach for any of these client-service protocols is optimal for the best-case service response time.

Specifying the state machine approach has proven to be more difficult. We have found it convenient to categorize the approach into three different cases, depending on how the ordering of the reliable broadcasts from clients to the service is done. We refer to these as the *sequencer* approach (for example, [4]), the *consensus* approach (for example, [6]), and the *a priori* approach (for example, [3]).

In the *a priori* approach, the order of delivery is determined by timestamping requests with a real clock value, and enqueuing them at the server until any earlier messages have been received. Thus, the *ordering latency*—that is, the time from when a client sends a request to the service until when the service delivers the request—is the same for all failure patterns, including failure free runs. This approach is provably optimal in the worst case, but a penalty is paid in that every request has the same ordering latency.

In the consensus approach, the order of the requests are determined jointly by all of the (non-faulty) servers in the system. In order to attain this consensus, a consensus protocol either must be run repeatedly [1] or must be initiated when a process of the service receives a client's request. Running consensus repeatedly can yield a small ordering latency, but is very expensive in terms of the number of messages used. Starting consensus from the receipt of a client's request, on the other hand, has a larger ordering latency but uses fewer messages. Neither is optimal in either the best-case service reponse time or the worst-case service response time.

In the sequencer-based approach, there is infinitely often a single server that unilaterally decides the order of a set of requests. In this sense, it is analogous to the primary-backup approach, and some of the lower bounds for primary-backup apply to sequencer-based state machines as well. Furthermore, there exist protocols of this approach that are optimal in the best-case service response time and are better than primary-backup in expected service response time.

Hence, contrary to common wisdom, we have found that in terms of best-case service response time, both primary-backup and state machines can be optimal. In terms of worst-case service response time, an optimal state machine service can be constructed, while an optimal primary-backup service cannot. However, the best-case service response time for this state machine service will be very poor. Finally, there are state machine protocols that have expected service response times better than any primary-backup protocol.

[1] P. Berman and A. A. Bharali. Quick atomic broadcast. In *Distributed Algorithms, 7th International Workshop, WDAG '93*, pages 189–203, Sept. 1993.

[2] N. Budhiraja, K. Marzullo, and F. B. Schneider. Primary-backup protocols: Lower bounds and optimal implementations. In *Dependable Computing for Critical Applications 3*, pages 321–343, Sept. 1992.

[3] F. Cristian, H. Aghili, H. R. Strong, and D. Dolev. Atomic broadcast: From simple message diffusion to byzantine agreement. In *Proc. of the 15th International Symposium on Fault-Tolerant Computers*, pages 200–206, 1985.

[4] M. F. Kaashoek, A. S. Tanenbaum, S. F. Hummel, and H. E. Bal. An efficient reliable protocol. *Operating System Review*, 23(4):5–19, Oct. 1989.

[5] F. B. Schneider. Implementing fault tolerant services using the state machine approach: A tutorial. *Computing Services*, 22(4):299–319, Dec. 1990.

[6] D. Skeen. *Crash Recovery in a Distributed Database System*. PhD thesis, University of California, Berkeley, 1982.

PODC'96, Philadelphia PA, USA
© 1996 ACM 0-89791-800-2/96/05..$3.50