Lower Bounds for a Primary–Backup Implementation of a Bofo Service

Navin Budhiraja Keith Marzullo Fred B. Schneider
Sam Toueg

Department of Computer Science, Cornell University
Ithaca NY 14853, USA

19 October 1991

1 Introduction

One way to implement a fault-tolerant service is the *primary-backup* or *primary-copy* approach [1]. With this approach, a service is implemented by a collection of servers. One server is designated as the *primary*; the others are called *backups*. Clients send requests to the primary and any responses to requests come from the primary. If the primary fails, then a *failover* occurs after which one of the backups assumes the role of the primary.

With the primary-backup approach, a request from a client to the service can be lost if sent to a faulty primary. However, periods during which requests can be lost are bounded by the length of time that elapses between the failure of the primary and the takeover by a backup. Such behavior is an instance of what we call a *bofo* service (*bounded outage finitely often*); an (i, Δ)–bofo service is one in which requests that are not processed fall into at most i intervals of time, each interval having a length of at most Δ. Thus, in an (i, Δ)–bofo service, even though some requests might be lost by the service, not too many will.

There exist lower bounds constraining i and Δ for implementing an (i, Δ)–bofo service using the primary-backup approach. These lower bounds are a function of message delivery delays and the number of failures that can be tolerated. The kinds of failures that that need to be tolerated constrain the degree of replication and the worst-case response time to client requests.

2 Characterization

To show the lower bounds described above, we must characterize the systems we are willing to consider. Unfortunately, there is no widely-accepted definition of the primary-backup approach. We believe that the following four attributes characterize a primary-backup system and note that several purported primary-backup protocols (*e.g.*, [1,2,5,8,3]) satisfy this characterization. There are, however, purported primary-backup protocols that do not satisfy one or more of these attributes (*e.g.*, [4]).

pb1: There exists a state predicate $PRMY$ on the state of any server such that $PRMY$ is true on the state of at most one server.

pb2: At each client c_i there exists a server identity $PRMY_i$. At any time, c_i may make a request by sending that request only to $PRMY_i$. Thus, a client interacts with a single server at any time. (This distinguishes primary-backup from active replication techniques such as [11].)

pb3: Suppose that requests are enqueued at the server and the only way a server can learn of a request is by executing a **receive**. A request r is received by a server s only when $PRMY$ is satisfied by that server's state.

pb4: The primary-backup system behaves like a single bofo server.

Note that in our specification we do not require that every request produce a response, that the identity of the primary remain static unless there is a failure, that the identity of the primary is known to all clients, or that the clients agree on the identity of the primary.

We consider a system consisting of n servers of which at most f_p can be faulty. Servers can be faulty by *crashing* in which case the server simply halts, by exhibiting *receive omission* failures in which case the server can crash or intermittently fail to receive a message, or by exhibiting *general omission* failures in which case the server can crash or fail in both sending and receiving messages. We do not assume that servers can fail in an arbitrary (*i.e.* Byzantine) manner [6] because arbitrary failures cannot be detected when there is a single source of responses [12].

We assume that server clocks are perfectly synchronized and that the processors communicate through a completely connected point-to-point network. We also assume that the links between processors are FIFO and that no more than f_l of the links can be faulty, where a faulty link may drop messages. We assume that if processors p_i and p_j are connected by a nonfaulty link, then a message sent from p_i to p_j at time t will be received by p_j no later than time $t + \delta$.

If no request requires a response, then it is trivial to provide a bofo service. We, therefore, assume at least one request type produces responses and that a client can initiate such a request at any time.

3 Lower Bounds

We now present lower bounds on any primary-backup system (*i.e.* any protocol satisfying **pb1** – **pb4**) for different failure models. We have proven these lower bounds for the stronger model in which only the links between servers can be faulty and in which omission failures can occur only for messages sent between servers and not between clients and servers. Although this stronger model is clearly unrealistic, it strengthens the lower bound results.

3.1 Bounds on Replication

Theorem 1 *Any primary-backup system that tolerates crash failures requires* $n \geq f_p + 1$.

Theorem 2 *Suppose there can be at most one link between any two processes and that the total number of process and link failures is bounded by* $f \leq min(f_p, f_l)$. *Then, any primary-backup system tolerating crash failures and link failures requires* $n \geq f + 2$.

Note, the scheme presented in [2] tolerates server crash failures and link failures. However, it uses only two processes to tolerate a single failure, which violates the lower bound of Theorem 2. This violation is possible because in [2] there are *two* links between the two processes. By making this assumption, the effects of a single link failure can be masked, and so only server crash failures need to be handled.

Theorem 3 *Any primary-backup system that tolerates receive omission failures requires* $n > \lfloor 3f_p/2 \rfloor$.

Theorem 4 *Any primary-backup system that tolerates general omission failures requires* $n > 2f_p$.

The protocol presented in [7] satisfies only a weaker version of **pb1**:

wpb1: There exists a state predicate $PRMY$ on the state of any server such that $PRMY$ is true on the state of at most one server except for bounded intervals during which it may be true on multiple servers.

The above four theorems hold even when **pb1** is replaced with **wpb1**. Thus, we have established that the protocol of [7] is optimal—it tolerates the general omission failure of one server by using three servers (a *primary*, a *backup* and a *witness*).

3.1.1 Bounds on Blocking

Informally, a *blocking protocol* for a primary-backup system s one in which the primary must, subsequent to receiving a request r, receive at least one other message from another server before it can respond to r. Provided that a primary cannot send a response to a request until it has received the request, we have:

Theorem 5 *All protocols for primary-backup for general-omission failures are blocking.*

This theorem also holds for the case where **pb1** is replaced by **wpb1**. The result shows that we cannot hope to get worst-case response times better than 4δ in systems that can exhibit general-omission failures where the server that receives the request is the server that sent the response. However, as shown below, the worst-case response time is lower for stronger failure models. In addition, we have developed non-blocking protocols for stronger failure models including receive-omission and link failures. We found it surprising that that non-blocking protocols in fact exist.

3.1.2 Bounds on Failover Times

In order to discuss lower bounds on failover times, we postulate a fifth characteristic of a primary-backup system. The protocols in [1,2,5,8,3] all satisfy this (seemingly reasonable) postulate.

pb5 A server that is the primary remains so until there is a failure.

Theorem 6 *Any primary-backup system tolerating at most f_p crash failures can exhibit an interval of time $f_p\delta$ during which there is no primary.*

Theorem 7 *Let f be $min(f_p, f_l)$. Any primary-backup system tolerating at most f failures, where failures can be crash failures and link failures, can exhibit an interval of time $2f\delta$ during which there is no primary.*

Theorem 8 *Any primary-backup system tolerating at most f_p general-omission failures can exhibit an interval of time $2f_p\delta$ during which there is no primary.*

Theorem 9 *Any primary-backup system tolerating at most f_p receive-omission failures can exhibit an interval of time $2f_p\delta$ during which there is no primary.*

Table 3.1.2 summarizes the bounds presented in Section 3.

Failure model	Replication	Blocking	Failover Time
Crash	$n > f_p$	no	$f_p\delta$
Crash + link[a]	$n > f + 1$	no	$2f\delta$
Receive Omission	$n > \lfloor \frac{3}{2} f_p \rfloor$[b]	no	$2f_p\delta$
General Omission	$n > 2f_p$	yes	$2f_p\delta$

[a] $f \leq min(f_p, f_l)$.
[b] This bound is not tight.

Table 1: Lower Bounds

4 Discussion

In addition to determining lower bounds, we have developed protocols that attain these bounds except for receive omission failures with $\lfloor 3f_p/2 \rfloor < n \leq 2f_p$. In this range, we have only been able to find solutions for $(n = 2, f_p = 1)$ and $(n = 4, f_p = 2)$. However, we can show that solutions in this range are impractical because they do not satisfy **pb5**.

References

[1] P.A. Alsberg and J.D. Day. A Principle for Resilient Sharing of Distributed Resources. In *Proceedings of the Second International Conference on Software Engineering*, pages 627–644, October 1976.

[2] J.F. Barlett. A NonStop Kernel. In *Proceedings of the Eighth ACM Symposium on Operating System Principles, SIGOPS Operating System Review*, volume 15, pages 22–29, December 1981.

[3] Anupam Bhide, E.N. Elnozahy, and Stephen P. Morgan. A Highly Available Network File Server. In *USENIX*, pages 199–205, 1991.

[4] Kenneth P. Birman and Thomas A. Joseph. Exploiting Virtual Synchrony in Distributed Systems. In *Eleventh ACM Symposium on Operating System Principles*, pages 123–138, November 1987.

[5] IBM International Technical Support Centers. IBM/VS Extended Recovery Facility (XRF) Technical Reference. Technical Report GG24-3153-0, IBM, 1987.

[6] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[7] Barbara Liskov, Robert Gruber, Paul Johnson, and Liuba Shrira. A Replicated Unix File System. Technical report, MIT Laboratory for Computer Science, 1990.

[8] Timothy Mann, Andy Hisgen, and Garret Swart. An Algorithm for Data Replication. Technical Report 46, Digital Systems Research Center, 1989.

[9] Keith Marzullo and Frank Schmuck. Supplying high availability with a Network File System. In *Proceedings of the 8th International Conference on Distributed Computing Systems*, pages 447–453, May 1988.

[10] Frank Pittelli and Hector Garcia-Molina. Database Processing with Triple Modular Redundancy. In *Fifth Symposium on Reliability in Distributed Software and Database Systems*, pages 95–103, jan 1986.

[11] Fred B. Schneider. The state machine approach: A tutorial. *Computing Surveys*, 22(3), September 1990.

[12] N. A. Speirs and P. A. Barrett. Using Passive Replicates in Delta-4 to Provide Dependable Distributed Computing. In *19 Int. Symp on Fault-Tolerant Computing Systems (FTCS 19)*, pages 184–190, June 1989.