


TECHNICAL REPORT



**Department of Computer Science**
**Cornell University**
**Ithaca, New York**

# Primary-Backup Protocols: Lower Bounds and Optimal Implementations

IN-61-CR
72137
P/24

Navin Budhiraja*
Keith Marzullo*
Fred B. Schneider**
Sam Toueg***

TR 92-1265
January 1992

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*Supported by Defense Advanced Research Projects Agency (DoD) under NASA Ames grant number NAG 2-593 and by grants from IBM and Siemens. The views, opinions and findings contained in this report are those of the authors and should not be construed as an official Department of Defense position, policy or decision.
**Supported in part by the Office of Naval Research under contract N00014-91-J-1219, the National Science Foundation under Grant No. CCR-8701103, DARPA/NSF Grant No. CCR-9014363 and by a grant from IBM Endicott Programming Laboratory.
***Supported in part by NSF grants CCR-8901780 and CCR-9102231 and by a grant from IBM Endicott Programming Laboratory.

# Primary–Backup Protocols: Lower Bounds and Optimal Implementations

Navin Budhiraja* Keith Marzullo* Fred B. Schneider†
Sam Toueg‡

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

January 30, 1992

**Abstract**

We present a formal specification of primary–backup. We then prove lower bounds on the degree of replication, failover time, and worst-case response time to client requests assuming different failure models. Finally, we outline primary–backup protocols and indicate which of our lower bounds are tight.

**Keywords:** Fault-tolerance, reliability, availability, primary–backup, lower bounds, optimal protocols.

## 1 Introduction

One way to implement a fault-tolerant service is by using multiple servers that fail independently. The state of the service is replicated and distributed

*Supported by Defense Advanced Research Projects Agency (DoD) under NASA Ames grant number NAG 2–593 and by grants from IBM and Siemens. The views, opinions, and findings contained in this report are those of the authors and should not be construed as an official Department of Defense position, policy, or decision.

†Supported in part by the Office of Naval Research under contract N00014-91-J-1219, the National Science Foundation under Grant No. CCR-8701103, DARPA/NSF Grant No. CCR-9014363, and by a grant from IBM Endicott Programming Laboratory.

‡Supported in part by NSF grants CCR-8901780 and CCR-9102231 and by a grant from IBM Endicott Programming Laboratory.

among these servers, and updates are coordinated so that even when a subset of servers fail, the service will remain available.

Such fault-tolerant services have been structured in several ways. One approach is to replicate the service state across all servers and to present each clients request to all nonfaulty servers in the same order. This service architecture is commonly called *active replication* or *the state machine approach* [Sch90] and has been widely studied from both theoretical and practical viewpoints (*e.g.*, [PSL80, CASD85, JB89]).

Another approach to building replicated services is to designate one server as the *primary* and all the others as *backups*. Clients make requests by sending messages only to the primary. If the primary fails, then a *failover* occurs and one of the backups takes over. This service architecture is commonly called the *primary-backup* or the *primary-copy* approach [AD76] and has been widely used in commercial fault-tolerant systems. However, the approach has not been analyzed as extensively as the state machine approach, and little is known of the costs and tradeoffs, the degree of replication required, and the worst-case response time for various failure models. In this paper, we derive some of these tradeoffs. For example, some primary–backup protocols use more servers than the number of failures to be tolerated [LGG+91]. We are able to show that the number of servers needed depends on the failure model.

The key difference between the active replication and primary-backup approaches is how each masks failures. With active replication, server failures are completely masked by voting and the service implemented is that of a single non-faulty server. With the primary-backup approach, a request to the service can be lost if it is sent to a faulty primary.[1] Thus, clients can now observe the effects of server failures. Periods during which requests are lost, however, are bounded by the length of time that can elapse between failure of the primary and takeover by a backup. Such behavior is an instance of what we call a *bofo* service (*bounded outage finitely often*). Specifically, a *service outage* occurs at time $t$ if some client makes a request at that time but never receives a response to that request.[2] Furthermore, in a $(k, \Delta)$–bofo service, all service outages can be grouped into at most $k$ intervals of time, with each interval having a length of at most $\Delta$. Accordingly, even though some requests may not elicit a response from a bofo service, not too many will. Note that if clients are restricted to send requests only to

[1] The client can subsequently resend a copy of that request to the new primary.

[2] For simplicity, we assume in this paper that every request elicits a response.

a single server, then one cannot implement a service that is stronger than bofo. This is because if the client sends a request to a server and the server subsequently crashes, then the request can be lost and will not be processed.

In this paper, we give lower bounds for implementing a bofo service using the primary-backup approach. These lower bounds depend on the message delivery delay and the kinds of failures that can be tolerated. The lower bounds constrain the degree of replication, the time during which the service can be without a primary, and the worst-case response time of client requests. In some cases the results are surprising. For example, more than $f + 1$ servers are necessary to tolerate $f$ failures of certain types (crash and link failures, receive-omission failures, or general-omission failures). Also, if a majority of the servers can be faulty, then any primary–backup protocol for receive–omission failures will have a run in which the primary is non–faulty, but it is forced to become a backup, while a server that is faulty becomes the primary in its place.

Finally, in this paper we outline some primary–backup protocols. This allows us to determine which of our lower bounds are tight.

The paper is organized as follows. Section 2 gives a formal specification of a primary-backup protocol. Section 3 defines our system model. Section 4 discusses the lower bounds, and in Section 5 we outline our protocols and state which of the previously-shown bounds are tight. We conclude in Section 6.

## 2 Primary–Backup Protocols

To derive lower bounds, we have to give a precise definition of a primary–backup protocol. We believe that the following four properties characterize a primary-backup protocol and note that many primary-backup protocols (*e.g.* [AD76, Bar81, Cen87, BEM91]) satisfy this characterization.

Pb1: There exists predicate $Prmy_s$ on the state of each server $s$. At any time, there is at most one server $s$ whose state satisfies $Prmy_s$.[3]

For brevity, whenever we say that "$s$ is the primary (at time $t$)" we mean that the state of $s$ satisfies $Prmy_s$. Note that the *failover time* for a service is the longest period of time during which $Prmy_s$ is not true for any $s$.

[3]The protocol of [LGG+91] allows concurrent primaries, but only for bounded periods. If one replaces Pb1 by this property, then except for the bounds on failover times, the bounds shown in Section 4 continue to hold.

Pb2: Each client $i$ maintains a server identity $Dest_i$ such that to make a request, client $i$ sends a message only to $Dest_i$.

Property Pb2 distinguishes the primary-backup approach from active replication, where each client sends requests to every server in the service.

For the next property, we model a communications network by assuming that client requests are enqueued in a message queue of a server.

Pb3: If a request arrives at a server that is not the primary, then the request is not enqueued (and is therefore not processed).

Properties Pb1–Pb3 specify a protocol for interacting with a service, but not the semantics of the service. For example, the properties do not rule out a primary that ignores all requests. A fourth property eliminates such trivial implementations by stipulating that the server be bofo for some values of $k$ and $\Delta$:

Pb4: There exist fixed values $k$ and $\Delta$ such that the service behaves like a single $(k, \Delta)$–bofo server.

This property is not implementable if the number of failures is not *a priori* bounded. Assuming a bounded number of failures is just a modeling trick. When the number of failures is unbounded, bounding the rate of failures and including reintegration of recovered servers can provide service outages of bounded lengths. We do not address failure rates or reintegration in this paper.

### A Simple Primary–Backup Protocol

As an example of a service based on the primary–backup approach, consider the following protocol, which tolerates a single server crash. Assume that all communication is over point-to-point nonfaulty links and that each link has an upper bound $\delta$ on message delivery time[4]. Refer to Figure 1. There is a primary server $p_1$ and a backup server $p_2$ connected by a communications link. A client initially sends all requests to $p_1$ (indicated by the arrow labeled 1 in the figure). Whenever $p_1$ receives such a request, it

- processes the request and updates its state accordingly,

[4]To simplify exposition, we assume that the maximum message delay between the clients and the servers is the same as the delay between the servers. However, our results can be easily extended to the case when the delays are different.

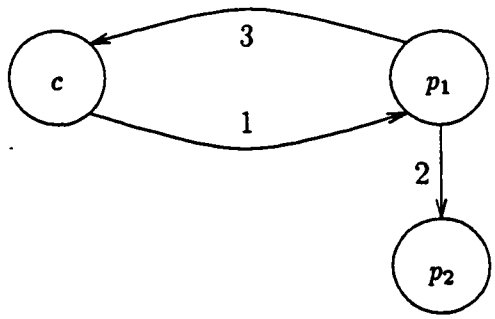


Figure 1: A Simple Primary–Backup Protocol.

- sends information about the state update to $p_2$ (message 2 in the figure),
- without waiting for an acknowledgement from $p_2$, sends a response to the client (message 3 in the figure).

The order in which these messages are sent is important because it guarantees that if the client receives a response, then either $p_2$ has received message 2 or $p_2$ has crashed.

Server $p_2$ updates its state upon receiving update messages from $p_1$. In addition, $p_1$ sends messages to $p_2$ every $\tau$ seconds. If $p_2$ does not receive such a message for $\tau + \delta$ seconds, then $p_2$ becomes the primary. Once $p_2$ has become the primary, it informs the clients (who update their copies of *Dest*) and begins processing any subsequent requests sent by them.

We now show that this protocol satisfies our characterization of a primary–backup protocol. Property Pb1 requires that there never be two primaries. This is satisfied by the following definitions of *Prmy*:

$$Prmy_{p_1} \stackrel{\text{def}}{=} (p_1 \text{ has not crashed})$$
$$Prmy_{p_2} \stackrel{\text{def}}{=} (p_2 \text{ has not received a message for } \tau + \delta)$$

The predicate $Prmy_{p_1} \wedge Prmy_{p_2}$ is always false in a system executing our protocol, and hence Pb1 is satisfied. The *failover time* for this protocol is the longest interval during which $\neg Prmy_{p_1} \wedge \neg Prmy_{p_2}$ can hold, and it is

$\tau + 2\delta$ seconds. Property Pb2 follows trivially from the description of the protocol. Property Pb3 is true because requests are not sent to $p_2$ until after $p_1$ has failed. Finally, Pb4 requires that the protocol implements a single bofo server for some values of $k$ and $\Delta$. Since $p_1$ sends message 2 before message 3, it will never be the case that $p_1$ sends a response to the client, and $p_2$ does not get information about that response from $p_1$. Using this fact, it can be shown that the service behaves like a single server. To compute $k$ and $\Delta$, we can let $k = 1$ and so it suffices to compute the longest interval during which a client request may not elicit a response. Assume that $p_1$ crashes at time $t_c$. Any request sent at $t_c - \delta$ or later may be lost since $p_1$ crashes at $t_c$. Furthermore, $p_2$ may not learn about $p_1$'s crash until $t_c + \tau + 2\delta$, and clients may not learn that $p_2$ is the primary for another $\delta$. So, the total period during which a request may not elicit a response is $t_c - \delta$ through $t_c + \tau + 3\delta$: the service is equivalent to a single $(1, \tau + 4\delta)$–bofo server.

# 3 The Model

We consider a system consisting of $n_s$ servers and $n_c$ clients. We assume that server clocks are perfectly synchronized with real time.[5] Clients and servers communicate by exchanging messages through a completely connected point-to-point network. Each message sent is enqueued in a queue maintained by the receiving process, and a process accesses its message queue by executing **receive**. We assume that links between processes are FIFO (*i.e.* if $p_i$ sends message $m$ followed by $m'$ to process $p_j$, then $p_j$ will never receive $m$ after $m'$) and if processes $p_i$ and $p_j$ are connected by a (non-faulty) link, then a message sent from $p_i$ to $p_j$ at time $t$ will be enqueued in $p_j$'s queue at of before $t + \delta$.

We are interested in identifying the costs inherent in primary–backup protocols, and so we assume that it takes no time for a server to compute a response. We also assume that a client can send a request at any time.

We model execution of a system by a *run*, which is a sequence of timestamped events involving clients, servers, and the message queues. These events include sending messages, enqueuing messages, receiving messages, and modeling computation at processes. Two runs $\sigma_1$ and $\sigma_2$ of the system are *indistinguishable* to a process $p$ if the same sequence of events (with the

[5] Extension to the case where clocks are only approximately synchronized [LMS85] is discussed in [Bud93].

same timestamps) occur at $p$ in both $\sigma_1$ and $\sigma_2$. We assume that if two runs $\sigma_1$ and $\sigma_2$ are indistinguishable to $p$, then at any time $t$, the state of $p$ at time $t$ in $\sigma_1$ is the same as the state of $p$ at time $t$ in $\sigma_2$. Again, it is not hard to extend our definition of indistinguishability to handle nondeterministic servers; the current definition does not.

We consider the following hierarchy of failure models:

*Crash failures:* A server may fail by halting prematurely. Until it halts, it behaves correctly. After it halts, a timeout can detect this fact.[6]

*Crash+Link failures:* A server may crash or a link may lose messages (but not delay, duplicate or corrupt messages).

*Receive-Omission failures:* A server may fail not only by crashing, but also by omitting to receive some of the messages sent to it over a nonfaulty link.

*Send-Omission failures:* A server may fail not only by crashing, but also by omitting to send some of the messages over a nonfaulty link.

*General-Omission failures:* A server may exhibit send-omission and receive-omission failures.

Figure 2 illustrates this failure hierarchy. Note that crash+link failures and the various types of omission failures are distinct. Although both represent loss of messages, each is dealt with by a different masking technique. In particular, crash+link failures can be masked by adding redundant communication paths, while omission failures can only be masked by adding sufficient redundant servers so that faulty processes can detect their own failure and halt. We discuss these masking techniques in Section 5.

Henceforth, we assume that no more than $f_s$ servers can be faulty, and for crash+link failures that no more than $f_l$ links can be faulty.

# 4 Lower Bounds

We now give lower bounds for implementing a single $(k, \Delta)$–bofo server using the primary–backup approach for each failure model.

[6]The lower bounds we derive for crash failures also hold for fail-stop failures [SS83] except for the bound on failover time. The lower bound on failover time depends on the maximum duration between when a server $p_i$ fails and when $failed_i$ becomes true.

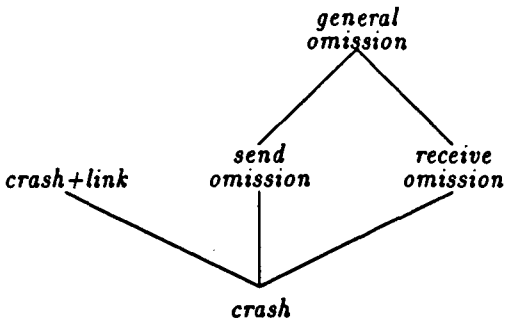


Figure 2: Failure Hierarchy

## 4.1 Bounds on Replication

The first theorem is obvious. However, to introduce our notation and the proof technique that will be used later in the section, we give a formal proof of the theorem.

**Theorem 1** *Any primary–backup protocol tolerating $f_s$ crash failures requires $n_s \geq f_s + 1$.*

**Proof:** We prove the result by contradiction. Suppose there is a protocol $P$ for $n_s < f_s + 1$. Thus, $P$ satisfies Pb4. Consider a run in which all $n_s$ servers are crashed initially and clients submit $R > k\lceil \Delta/d \rceil$ requests, where $d$ is the minimum time between the sending of any two requests ($d > 0$). By Pb4, at least one of these requests must elicit a response. This is because the number of requests that cannot have responses must fall into at most $k$ intervals of length at most $\Delta$, and each interval of $\Delta$ can contain at most $\lceil \Delta/d \rceil$ requests. However, such a response is impossible since, by assumption, all servers have crashed. □

The following lemma will be used for the rest of the theorems in this section.

**Lemma 4.1** *Consider any protocol that satisfies* Pb4. *Suppose two disjoint and nonempty sets of servers A and B can be found that meet the following three properties:*

1. *There exists a run $\sigma_a$ containing $R > 2k\lceil\Delta/d\rceil$ requests where $d$ is the minimum time between the sending of any two client requests ($d > 0$). Furthermore, in this run the servers in $A$ do not crash and all other servers crash at time 0.*

2. *There exists a run $\sigma_b$ containing $R$ requests. Furthermore, in this run the servers in $B$ do not crash and all other servers crash at time 0.*

3. *There exists a run $\sigma_{ab}$ containing $R$ requests. Furthermore, the servers in $A$ and $B$ do not crash, $\sigma_{ab}$ is indistinguishable from $\sigma_a$ to all servers in $A$, and $\sigma_{ab}$ is indistinguishable from $\sigma_b$ to all servers in $B$.*

*At least one of the above runs violates* Pb2.

**Proof:** Suppose for contradiction that the lemma is false and runs $\sigma_a$, $\sigma_b$ and $\sigma_{ab}$ all satisfy Pb2.

For $\sigma_a$, by Pb4 at least $R - k\lceil\Delta/d\rceil$ of the requests must have been received by servers in $A$. Similarly, for $\sigma_b$, at least $R - k\lceil\Delta/d\rceil$ of the requests must have been received by servers in $B$. Finally, since $\sigma_{ab}$ is indistinguishable from $\sigma_a$ to servers in $A$, they must execute the same number of receive events in both runs. The same holds for the servers in $B$. By Pb2, each request is sent to at most one server and so at least $2(R - k\lceil\Delta/d\rceil)$ requests must have been sent in $\sigma_{ab}$. Since only $R$ requests were sent, we must have $R \geq 2(R - k\lceil\Delta/d\rceil)$, or $R \leq 2k\lceil\Delta/d\rceil$, which contradicts the assumption that $R > 2k\lceil\Delta/d\rceil$.

□

Theorems 2 and 3 depend on two parameters of primary-backup protocols. Let $\Gamma$ be the maximum time between any two successive client requests (possibly from different clients) in any run of the system, and let $D$ be a duration such that if some server $s$ becomes the primary at time $t_0$ and remains the primary through time $t \geq t_0 + D$ when a client $c_i$ sends a request, then $Dest_i = s$ at time $t$. For simplicity of notation, we will write $D < \Gamma$ to mean that $D$ is bounded and $\Gamma$ is either unbounded or bounded and greater than $D$.

With both send-omission failures and crash+link failures, messages may fail to reach their intended destinations. The following theorem shows that crash+link failures are more expensive to tolerate as they require more replication.

**Theorem 2** *Suppose there is at most one link between any two servers and the total number of server and link failures that can occur is $f$, where $f \leq min(f_s, f_l)$. Then any primary-backup protocol tolerating crash+link failures and having $D < \Gamma$ requires $n_s \geq f + 2$.*

**Proof:** For contradiction, assume the existence of a protocol $P$ with $n_s < f + 2$. We will show that $P$ has three runs $\sigma_a$, $\sigma_b$ and $\sigma_{ab}$ that satisfy the conditions of Lemma 4.1. From the lemma, at least one of these runs violates Pb2, which implies that $P$ cannot be a primary–backup protocol.

Let $A$ be a set containing the one server $s_a$ and let $B$ be the set of remaining servers. Since $|A| = 1$ and $|B| = n_s - 1 \leq f$, $A$ and $B$ can be partitioned by link failures.

We first construct the run $\sigma_{ab}$ in which no server crashes, postulating that the links between the servers in $n_a$ and $n_b$ are faulty and do not deliver any messages. As required by Lemma 4.1, clients will send a total of $R > 2k\lceil\Delta/d\rceil$ requests. Let $0 < d \leq \Gamma - D$ be the minimum interval between any two such requests. We postulate that a request will be sent at time $t$ iff no request has been sent during the interval $[t - d..t)$ and one of the following rules hold.

1. A server $s$ is the primary during the interval $[t - D..t]$. This request arrives immediately and is enqueued (at $s$, by Pb3 and the definition of $D$).

2. There is no primary at time $t$. This request arrives immediately and by Pb3 will never be enqueued at any server.

3. A server $s$ is the primary at time $t$ but another server $s'$ is the primary immediately after time $t$. If this request is sent to $s$, then it arrives after $t$, and if it is sent to any other server, then it arrives immediately. In both cases, it arrives at a server that is not the primary, and so will not be enqueued (again by Pb3).

Note that, by construction, the maximum interval between any two client requests is $D + d$. This interval occurs when a server $s$ becomes the primary just before $d$ after a client message is sent, and $s$ remains the primary for at least $D$. Hence, the client will be able to send $R$ requests within time $R(D + d)$. This completes the construction of $\sigma_{ab}$.

We now construct $\sigma_a$ and $\sigma_b$, recalling that in $\sigma_a$ all of the servers except $s_a$ crash at time 0, and in $\sigma_b$ server $s_a$ crashes at time 0. The clients send the

same requests and at the same times in $\sigma_a$ and in $\sigma_b$ as in $\sigma_{ab}$. Furthermore, by construction these requests will arrive at the servers according to the same rules used in constructing $\sigma_{ab}$. Of course, a client request may not be delivered to the same servers in $\sigma_a$ or $\sigma_b$ as in $\sigma_{ab}$, since different servers are operational in these runs.

Since $s_a$ does not receive any messages from servers in $B$ in either $\sigma_{ab}$ or $\sigma_a$, these two runs are indistinguishable to $s_a$ as long as it receives the same client requests at the same times in both runs. We that this is the case by contradiction: let $t$ be the earliest time that $s_a$ can distinguish between these two runs.

Thus, at time $t$ either $s_a$ received a request $m$ in $\sigma_{ab}$ but not in $\sigma_a$ or it received a request $m$ in $\sigma_a$ but not in $\sigma_{ab}$. We will assume the former; the proof for the latter is similar. The request $m$ must have been enqueued at some time $t' \leq t$ at $s_a$ in $\sigma_{ab}$. Since $m$ was received by $s_a$, $m$ must have been sent by rule 1. By rule 1, $s_a$ must have been the primary through $[t' - D..t']$ in $\sigma_{ab}$ and therefore, by indistinguishability, in $\sigma_a$ as well. By the definition of $D$, $m$ would have been enqueued at $s_a$ at time $t'$ in $\sigma_a$ as well.

Since $s_a$ cannot distinguish between the runs before $t$, $s_a$ cannot receive $m$ before $t$ in $\sigma_a$, and $s_a$ must execute a receive in both $\sigma_a$ and $\sigma_{ab}$ at time $t$. So, it must be the case that $s_a$ receives another request $m' \neq m$ at time $t$ in $\sigma_a$. Assume that $m'$ was enqueued at time $t''$. By an indistinguishability argument similar to above, $m'$ must be enqueued at time $t''$ at $s_a$ in $\sigma_{ab}$ as well. Therefore, if $s$ received $m'$ in $\sigma_a$ at time $t$, it must receive $m'$ in $\sigma_{ab}$ as well, a contradiction.

A similar argument can be used to show that the servers in $n_b$ receive the same requests in $\sigma_b$ and $\sigma_a b$, and so these two runs are indistinguishable to the servers in $n_b$. Thus, by Lemma 4.1 $P$ cannot be a primary–backup protocol. □

The next theorem states that additional replication is required in order to tolerate receive-omission failures. The proof is similar to that of Theorem 2, and so it is omitted.

**Theorem 3** *Any primary–backup protocol tolerating receive-omission failures and having $D < \Gamma$ requires $n_s > \lfloor \frac{3f_s}{2} \rfloor$.*

The next lower bound holds independent of the relation between $D$ and $\Gamma$. However, before we prove the result, we need the following definitions.

Define $\prec$ to be the *potential causality* relation [Lam78] on server events $e_1$ and $e_2$ as follows: $e_1 \prec e_2$ iff

1. Both $e_1$ and $e_2$ occur at the same server $s$ and $e_1$ occurs before $e_2$ or
2. $e_1$ is a send event and $e_2$ is the corresponding receive event or
3. $(\exists e\colon e_1 \prec e \wedge e \prec e_2)$

We say that a request $m$ is an *update request* iff in any run $\sigma$ for which $m$ has a response $r$, any other response $r'$ sent after $r$ in real time causally follows $m$, *i.e.* if event $e(m)$ corresponds to the receipt of $m$ and event $e(r')$ corresponds to the sending of $r'$, then $e(m) \prec e(r')$. A primary-backup protocol is trivial to implement if there are no update requests, and so we assume that update requests exist and that clients can send them at any time.

**Theorem 4** *Any primary-backup protocol tolerating general-omission failures requires* $n_s > 2f_s$.

**Proof:** Assume for contradiction that there is a protocol for $n_s \leq 2f_s$. Partition the servers into two disjoint sets $A$ and $B$ of size at most $f_s$ each. We will construct two runs $\sigma_1$ and $\sigma_2$. In each run, one set of servers will be faulty and the other set will be nonfaulty.

$\sigma_1$: The servers is $A$ are faulty and fail to communicate with all servers in $B$, but behave correctly otherwise. Clients send update requests until the first response is sent (this must happen, by Pb4). Assume that the first response $r$ to a request is sent at time $t$. Say that this response is sent by server $s$.

$\sigma_2$: The same as $\sigma_1$ up to time $t$, but if $s$ is in $B$, then in $\sigma_2$ the servers in $B$ are faulty and fail to communicate with all servers in $A$. In either case, no server can distinguish $\sigma_1$ from $\sigma_2$ through time $t$ and therefore, the first response $r$ is sent at time $t$ in $\sigma_2$ as well.

By construction, $r$ is sent by a faulty server in $\sigma_2$. Let all of the faulty servers in $\sigma_2$ crash immediately after $r$ is sent and have clients continue to send requests until another response $r'$ is sent. This response must have been sent by a nonfaulty server which implies that $\neg(e(m) \prec e(r'))$. However this violates the fact that $m$ is an update request. □

## 4.2 Bounds on Blocking

Informally, a blocking primary-backup protocol is one in which the primary must, subsequent to receiving a request $m$, either receive a message from

another server or simply wait an interval before it can respond to $m$. We say that a primary-backup protocol is *$C$-blocking* if any request (received, say, at $t_m$) elicits a response in a failure-free run at time $t_r$, then $t_r - t_m \leq C$. For example, any primary-backup protocol in which the primary sends information about a request to the backups and waits for acknowledgement before sending the response to the client will be at least $2\delta$-blocking.

As shown in Section 5, 0-blocking primary-backup protocols are possible for crash and crash+link failure models. The simple protocol tolerating crash failures presented in Section 2 is 0-blocking. We call such protocols *nonblocking* because the primary can send a reply to the client as soon as the reply is computed. Nonblocking protocols tolerating receive-omission failures are also possible as long as $n_s > 2f_s$, but there is no nonblocking primary-backup protocol tolerating send-omission failures.

**Theorem 5** *Any primary-backup protocol tolerating receive-omission failures with $f_s > 1$, $n_s \leq 2f_s$ and $D < \Gamma$ is $C$-blocking for some $C \geq 2\delta$.*

**Proof:** For contradiction, suppose there is a primary-backup protocol for $n_s \leq 2f_s$ and $f_s > 1$ that is $C$-blocking where $C < 2\delta$. Partition the servers into two sets $A$ and $B$ where $|A| = f_s$ and $|B| = n_s - f_s \leq f_s$. We construct three runs. In all three runs, assume that all server messages take $\delta$ to arrive.

$\sigma_1$: There are no failures and all client requests take $\delta$ to arrive. Moreover, clients send update requests until some request $m$ evokes a response $r$. Let $m$ be received at time $t_m$ by server $p \in A$ and $r$ be sent at time $t_r$ by a different server $q \in A$. Notice that since the protocol is $C$-blocking where $C < 2\delta$, $t_r - t_m < 2\delta$. Also, since by construction all requests take $\delta$ to arrive, all client requests sent after time $t_m + \delta$ will be received after time $t_r$.

$\sigma_2$: Identical to $\sigma_1$ until $p$ receives $m$ at time $t_m$. At this point in $\sigma_2$, all servers in $A$ are assumed to crash and clients are assumed to send no request during the interval $[t_m + \delta..t_r]$. Finally, after time $t_r$ clients are assumed to repeatedly send requests at intervals of at least $d$ where $0 < d \leq \Gamma - D$ as follows. A request is sent at time $t$ if no request has been sent in $[t - d..t)$ and one of the following rules hold.

1. A server $s \in B$ is the primary during the interval $[t - D..t]$. This request arrives immediately and is enqueued (at $s$, by Pb3 and the definition of $D$).

2. There is no primary in $B$ at time $t$. This request arrives immediately by Pb3 will never be enqueued at any server.

3. A server $s \in B$ is the primary at time $t$ but another server $s' \in B$ is the primary immediately after time $t$. If the request is sent to $s$, then it arrives after $t$, and if it is sent to any other server it arrives immediately. In both cases, it arrives at a server that is no the primary, and so will not be enqueued (again, by Pb3).

Notice that eventually, there will be a response (say $r'$) in $\sigma_2$ because the protocol satisfies Pb4, and by construction it must be from a request sent by rule 1.

$\sigma_3$: The same as $\sigma_2$, except that the servers in $A$ *do not* crash at time $t_m$. Instead, the servers in $B$ commit receive failures on all messages sent after $t_m$ by servers in $A$. Clients send requests at the same times as in $\sigma_2$ which arrive using the same rules as $\sigma_2$.

Now, consider these three runs. By construction, the runs are identical up to time $t$. Since all server messages take $\delta$ to arrive, clients cannot distinguish $\sigma_1$ and $\sigma_3$ through $t_m + \delta$, and so clients send the same requests to the same servers in both $\sigma_1$ and $\sigma_3$. Similarly, since all server messages take $\delta$ to arrive, the servers in $B$ cannot distinguish between $\sigma_1$ and $\sigma_3$ through $t_m + \delta$. Therefore, since $t_r - t_m < 2\delta$, $p$ (the server that received request $m$ at time $t_m$ in $\sigma_1$) and $q$ (the server that sent response $r$ at time $t_r$ in $\sigma_1$) cannot distinguish between $\sigma_1$ and $\sigma_3$ through time $t_r$, and so $q$ sends response $r$ in $\sigma_3$ as well. Then, using an argument similar to the one in Theorem 2, servers in $B$ cannot distinguish $\sigma_2$ and $\sigma_3$, and so response $r'$ also occurs in $\sigma_3$. However, $\neg(e(m) \prec e(r'))$ which violates the assumption that $m$ is an update request. □

In run $\sigma_3$ of the above proof, a correct primary ($p$ in set $A$) becomes the backup, while a faulty server from set $B$ becomes the primary in $p$'s place. It is always possible to construct such a run. This is a disconcerting property: there does not exist a primary–backup protocol that tolerates receive-omission failures with $n_s \leq 2f_s$ in which a primary cedes only when it fails. Moreover, his lower bound is tight—we have constructed a receive-omission primary–backup protocol with $n_s = 2f_s + 1$ in which a primary cedes only when it fails.

The above lower bound holds only if $f_s > 1$. If $f_s = 1$, then the following theorem holds. Its proof is similar to the proof of Theorem 5, except that $p = q$.

**Theorem 6** *Any primary-backup protocol tolerating receive-omission failures with $f_s = 1$ and $n_s \leq 2f_s$ and having $D < \Gamma$ is $C$-blocking for some $C \geq \delta$.*

Primary–backup protocols tolerating send-omission failures exhibit the same blocking as those tolerating receive-omission failures:

**Theorem 7** *Any primary-backup protocol tolerating send-omission failures and $f_s > 1$ is $C$-blocking for some $C \geq 2\delta$.*

**Proof:** For contradiction, suppose there is a primary–backup protocol that is $C$–blocking where $C < 2\delta$. We consider the following two runs in which all server messages take $\delta$ to arrive.

$\sigma_1$: There are no failures and all client requests take $\delta$ to arrive. Moreover, clients send update requests until some request $m$ evokes a response $r$. Let $m$ be received at time $t_m$ by server $p$ and $r$ be sent at time $t_r$ by a different server $q$. Notice that since the protocol is $C$–blocking where $C < 2\delta$, $t_r - t_m < 2\delta$. Also, since by construction all requests take $\delta$ to arrive, all client requests sent after time $t_m + \delta$ will be received after time $t_r$.

$\sigma_2$: Identical to $\sigma_1$ through $t_m$. After $t_m$, $p$ and $q$ fail and omit to send all messages to all servers except each other. Since by construction all messages take $\delta$ to arrive, servers and clients cannot distinguish between $\sigma_1$ and $\sigma_2$ through $t_m + \delta$, and as a result $p$ and $q$ cannot distinguish the two runs through $t_m + 2\delta$. Therefore, since $t_r - t_m < 2\delta$, $q$ sends the response $r$ at time $t_r$ in $\sigma_2$ as well. Now let $p$ and $q$ crash at time $t_r$ and the clients send requests after time $t_r$. By Pb4, there eventually must be some request $m'$ that results in a response $r'$. However, $\neg(e(m) \prec e(r'))$, which violates the assumption that $m$ us an update request. □

**Theorem 8** *Any primary-backup protocol tolerating send-omission failures and $f_s = 1$ is $C$-blocking for some $C \geq \delta$.*

### 4.3 Bounds on Failover Times

The *failover time* is the longest interval during which $Prmy_s$ is not true for any server $s$. In this section, we present lower bounds on failover times. In order to discuss these bounds, we postulate a fifth property of primary–backup protocols:

Pb5: A server that is the primary remains so until there is a failure.

This is a reasonable expectation and it is valid for all protocols that we have found in the literature.

**Theorem 9** *Any primary-backup protocol tolerating $f_s$ crash failures must have a failover time of at least $f_s\delta$.*

**Proof:** Assume that the theorem is false. We derive a contradiction by induction on $f_s$.

**Base case** $f_s = 0$: trivially true since the failover time cannot be smaller than zero.

**Induction case** $f_s > 0$: suppose the theorem holds for at most $f_s - 1$ failures, but there is a protocol $P$ for which the theorem is false when there are $f_s$ failures. From the induction hypothesis, there is a run $\sigma$ with at most $f_s - 1$ failures and an interval $[t_0..t_1]$ at least $(f_s - 1)\delta$ during which there is no primary. Let $p_1$ be the server that becomes the primary at $t_1$. Consider the two runs $\sigma_1$ and $\sigma_2$ that extend $\sigma$ as follows:

$\sigma_1$: Assume $p_1$ crashes at time $t_1$. By assumption, there exists a new primary (say $p_2$) at time $t_2 < t_1 + \delta$. Since $p_1$ crashes at time $t_1$, $p_2$ does not receive any messages from $p_1$ that were sent sent after time $t_1$.

$\sigma_2$: Assume $p_1$ does not crash but all messages sent by $p_1$ after time $t_1$ take $\delta$ to arrive.

Since $p_2$ cannot distinguish $\sigma_1$ from $\sigma_2$ through time $t_2$, $p_2$ becomes the primary at time $t_2$ in $\sigma_2$. By Pb5, however, $p_1$ remains the primary at time $t_2$ in $\sigma_2$. This violates Pb1, and so $P$ is not a primary-backup protocol. □

The failover times for all other failure models have a larger lower bound.

**Theorem 10** *Any primary-backup protocol tolerating $f$ crash+link failures, where $f \leq min(f_s, f_l)$, has a failover time of at least $2f\delta$.*

**Proof:** We again assume that the theorem is false and derive a contradiction.

**Base case** $f = 0$: trivially true.

**Induction case** $f > 0$: suppose the theorem holds for at most $f - 1$ failures, but there is a protocol $P$ for which the theorem is false when there are $f$ failures.

From the induction hypothesis, there is a run $\sigma$ with at most $f - 1$ failures and an interval $[t_0..t_1]$ at least $(f - 1)\delta$ during which there is no primary. Let $p_1$ be the server that becomes the primary at $t_1$. Consider the three runs $\sigma_1$, $\sigma_2$ and, $\sigma_3$ that extend $\sigma$ as follows:

$\sigma_1$: Assume that $p_1$ crashes at time $t_1$ and that all messages sent after $t_1$ take $\delta$ to arrive. By assumption, there exists a new primary (say $p_2$) at time $t_2 < t_1 + 2\delta$. Since $p_1$ crashes at time $t_1$, $p_2$ does not receive any messages from $p_1$ that were sent after time $t_1$. Furthermore, since all messages take $\delta$ to arrive, any message that was sent after $t_1 + \delta$ can be received by $p_2$ only after time $t_2$.

$\sigma_2$: Assume that $p_1$ does not crash and that all messages sent after time $t_1$ take $\delta$ to arrive. Since there are no failures after time $t_1$, by Pb5 $p_1$ continues to be the primary through time $t_2$.

$\sigma_3$: The same as $\sigma_2$ except that the link between $p_1$ and $p_2$ is faulty and does not deliver any message sent by $p_1$ to $p_2$ after time $t_1$.

By construction, $p_2$ cannot distinguish $\sigma_1$ from $\sigma_3$ through time $t_2$, and so $p_2$ becomes the primary at time $t_2$ in $\sigma_3$. Similarly, $p_1$ cannot distinguish $\sigma_2$ from $\sigma_3$ through time $t_2$ and so $p_1$ remains the primary until time $t_2$ in $\sigma_3$. This violates Pb1, and so $P$ is not a primary–backup protocol. □

We omit the proofs of the following two theorems because they are similar to Theorem 9.

**Theorem 11** *Any primary–backup protocol tolerating $f_s$ receive-omission failures has a failover time of at least $2f_s\delta$.*

**Theorem 12** *Any primary–backup protocol tolerating $f_s$ send-omission failures has a failover time of at least $2f_s\delta$.*

# 5 Outline of the Protocols

In order to establish that the bounds given above are tight, we have developed a set of primary–backup protocols for the different failure models [BMST92]. In this section, we outline these protocols and use them to show which of the lower bounds in the previous sections are tight.

Our protocol for crash failures is similar to the protocol given in Section 2. Whenever the primary receives a request from the client, it processes that request and sends information about state updates to the backups before sending a response to the client. All servers periodically send messages to each other in order to identify server failures. This protocol uses $(f_s + 1)$ servers and is 0–blocking. Thus, Theorem 1 is tight and this protocol uses the optimal number of servers and incurs no additional delay. Furthermore, this protocol has the failover time $f_s\delta + \tau$ for arbitrarily small and positive $\tau$, and so Theorem 9 is tight.

In order for the protocol to tolerate crash+link failures, we add an additional server. By Theorem 2, this server is necessary. The additional server ensures that there is always at least one nonfaulty path between any two correct servers, where a path contains zero or more intermediate servers. The crash failure protocol outlined above is now modified so that a primary ensures any message sent to a backup is sent across at least one nonfaulty path. Note that this protocol uses $(f + 2)$ servers and is 0–blocking. Thus, Theorem 2 is tight and this protocol uses the optimal number of servers and incurs no additional delay. Furthermore, this protocol has the failover time $2f\delta + \tau$ for arbitrarily small and positive $\tau$, and so Theorem 10 is tight.

Most of our protocols for the different kinds of omission failures apply translation techniques [NT88] to the crash failure protocol. These techniques ensure that a faulty server detects its own failure and halts. The translations assume a round-based protocol. Since our crash failure protocol is not round-based, we must modify the translations so that a server can send and receive messages at any time rather than just at the beginning or the end of a round. This is not difficult to do. All of these resulting omission protocols have failover time $2f_s\delta + \tau$, and thus Theorems 11 and 12 are tight. The protocol for send-omission failures uses $f_s + 1$ servers and is $2\delta + \tau$–blocking. Furthermore, we also have a send-omission protocol for $f_s = 1$ that is $\delta$–blocking. Thus, Theorems 7, 8 and 12 are tight. The protocol for general-omission failures also uses $2f_s + 1$ servers and is $2\delta + \tau$–blocking, and so Theorem 4 is tight, and Theorems 7 and 12 are tight for general-omission failures as well.

We have not been able to determine whether Theorems 3 and 5 are tight. Our protocol tolerating receive-omission failures uses $2f_s + 1$ servers whereas the lower bound in Theorem 3 only requires $n_s > \lfloor \frac{3f_s}{2} \rfloor$. We have constructed receive-omission protocols for $n_s = 2$, $f_s = 1$ and $n_s = 4$, $f_s = 2$ but have not been able to generalize the protocols. The protocols in this region have the odd property that a nonfaulty primary can cede to a faulty

| failure model | degree of replication | amount of blocking | failover time |
|---|---|---|---|
| crash | $n_s > f_s$ | 0 | $f_s\delta$ |
| crash+link | $n_s > f + 1$ † | 0 | $2f\delta$ |
| receive omission | $n_s > \lfloor \frac{3f_s}{2} \rfloor$ * † | $\delta$ $f_s = 1$ †<br>$2\delta$ $f_s > 1$ * † | $2f_s\delta$ |
| send omission | $n_s > f_s$ | $\delta$ $f_s = 1$<br>$2\delta$ $f_s > 1$ | $2f_s\delta$ |
| general omission | $n_s > 2f_s$ | $\delta$ $f_s = 1$<br>$2\delta$ $f_s > 1$ | $2f_s\delta$ |

* Bound not known to be tight.
† $D < \Gamma$.

Table 1: Lower Bounds.

primary, and so we do not expect such protocols to have much practical importance. However, the protocol for $n_s = 2$, $f_s = 1$ is $\delta$–blocking and so Theorem 6 is tight.

Table 1 summarizes all of our results.

# 6 Discussion

This paper gives a formal characterization of primary–backup protocol for a synchronous system. It presents lower bounds on the degree of replication, the blocking time, and the failover time for a primary–backup protocol under various kinds of server and link failures. A set of primary–backup protocols is outlined and used to show which of our lower bounds are tight.

It is instructive to compare our results to existing primary–backup protocols. A two-server primary–backup protocol that tolerates crash+link failures is presented in [Bar81], which seemingly contradicts Theorem 2. However, this protocol assumes that there are two links between the two servers which effectively masks a single link failure. Hence, only crash failures need to be tolerated which can be accomplished using only two servers (Theorem 1).

A more ambitious primary–backup protocol is presented in [LGG+91]. This protocol tolerates the following failure model (quoted from [LGG+91]):

> The network may lose or duplicate messages, or deliver them late or out of order; in addition it may partition so that some nodes are temporarily unable to send messages to some other nodes. As is usual in distributed systems, we assume the nodes are fail-stop processors and the network delivers only uncorrupted messages.

This failure model is incomparable with the hierarchy we present. However, the protocol does tolerate general-omission failures and has optimal degree of replication as it uses $n_s = 2f_s + 1$ servers.

In Theorem 2, we assumed that $D < \Gamma$. This assumption is crucial: we have constructed a two-server primary–backup protocol tolerating one crash+link failure for which $D \geq \Gamma$. Recall that link failures are masked by adding redundant paths between the servers. Our two-server crash+link protocol essentially uses the path from the primary to the backup through the client as the redundant path. Thus, there appears to be a tradeoff between the degree of replication and the time it takes for a client to learn that there is a new primary.

The lower bounds on failover times given in Section 4.3 were derived assuming Pb5. We have constructed primary–backup protocols that have failover times smaller than the lower bounds given in Section 4.3, and as expected these protocols do not satisfy Pb5. This smaller failover time is achieved at a cost of an increased variance in service response time.

Finally, we have attempted to give a characterization of primary–backup that is broad enough to include most synchronous protocols that are considered to be instances of the approach. There are protocols, however, that are incomparable to the class of protocols we analyze [BJ87]. In addition, the protocols in [OL88, MHS89] are incomparable since they were developed for an asynchronous setting. Such protocols cannot be cast in terms of implementing a $(k, \Delta)$–bofo service for finite values of $k$ and $\Delta$. We are currently studying possible characterizations for a primary–backup protocol in an asynchronous system and expect to extend our results to this setting.

# References


[AD76] 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.

[Bar81] 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.

[BEM91] Anupam Bhide, E.N. Elnozahy, and Stephen P. Morgan. A Highly Available Network File Server. In *USENIX*, pages 199–205, 1991.

[BJ87] 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.

[BMST92] Navin Budhiraja, Keith Marzullo, Fred Schneider, and Sam Toueg. Optimal primary–backup protocols. Technical report, Cornell University, Ithaca, N.Y., 1992.

[Bud93] Navin Budhiraja. *Primary Backup in Synchronous and Asynchronous Systems.* PhD thesis, Cornell University, Department of Computer Science, 1993. In preparation.

[CASD85] Flaviu Cristian, Houtan Aghili, H. Ray Strong, and Danny Dolev. Atomic broadcast: From simple message diffusion to Byzantine agreement. In *Proceedings of the Fifteenth International Symposium on Fault-Tolerant Computing*, pages 200–206, Ann Arbor, Michigan, June 1985. A revised version appears as IBM Technical Report RJ5244.

[Cen87] IBM International Technical Support Centers. IBM/VS Extended Recovery Facility (XRF) Technical Reference. Technical Report GG24-3153-0, IBM, 1987.

[JB89] Thomas Joseph and Kenneth Birman. *Reliable Broadcast Protocols*, pages 294–318. ACM Press, New York, 1989.

[Lam78] Leslie Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, July 1978.

[LGG+91] Barbara Liskov, Sanjay Ghemawat, Robert Gruber, Paul Johnson, and Michael Williams. Replication in the Harp file system. In *Proceedings of the 13th Symposium on Operating System Principles*, pages 226–238, 1991.

[LMS85] Leslie Lamport and P. M. Melliar-Smith. Synchronizing clocks in the presence of faults. *Journal of the ACM*, 32(1):52–78, January 1985.

[MHS89] Timothy Mann, Andy Hisgen, and Garret Swart. An Algorithm for Data Replication. Technical Report 46, Digital Systems Research Center, 1989.

[NT88] Gil Neiger and Sam Toueg. Automatically increasing the fault-tolerance of distributed systems. In *Proceedings of the Seventh ACM Symposium on Principles of Distributed Computing*, pages 248–262, Toronto, Ontario, August 1988. ACM SIGOPS-SIGACT.

[OL88] B. Oki and Barbara Liskov. Viewstamped replication: A new primary copy method to support highly available distributed systems. In *Seventh ACM Symposium on Principles of Distributed Computing*, pages 8–17, august 1988.

[PSL80] M. Pease, R. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, April 1980.

[Sch90] Fred B. Schneider. The state machine approach: A tutorial. *Computing Surveys*, 22(4):299–319, December 1990.

[SS83] Richard D. Schlichting and Fred B. Schneider. Fail-stop processors: an approach to designing fault-tolerant computing systems. *ACM Transactions on Computer Systems*, 1(3):222–238, August 1983.