Limitations on Database Availability when Networks Partition

Brian A. Coan, Brian M. Oki, and Elliot K. Kolodner

Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139

Abstract: In designing fault-tolerant distributed database systems, a frequent goal is making the system highly available despite component failure. We examine software approaches to achieving high availability in the presence of partitions. In particular, we consider various replicated-data management protocols that maintain database consistency and attempt to increase database availability when networks partition. We conclude that no protocol does better than a bound we have determined. Our conclusions hold under the assumption that the pattern of data accesses by transactions obeys a uniformity assumption. There may be some particular distribution for which specialized protocols can increase availability.

This research was supported in part by the Advanced Research Projects Agency of the Department of Defense, monitored by the Office of Naval Research under contracts N00014-83-K-0125 and N00014-85-K-0168, in part by the National Science Foundation under grants DCR-8503662 and DCR-83-02091, and in part by the Office of Army Research under contract DAAG29-84-K-0058.

© 1986 ACM 0-89791-198-9/86/0800-0187 75¢

1 Introduction

In designing fault-tolerant distributed database systems, a frequent objective is to make the system highly available in spite of component failures. We measure availability as the fraction of transactions presented to the system that complete. One technique to increase data availability is replicating the data at various sites in the network. In this paper, we examine several replicated-data management protocols that maintain database consistency and attempt to make replicated data highly available in the presence of network partitions. (Partitions are failures that divide a system into two or more components between which communication is impossible.)

The protocols we examine in this paper maintain one-copy serializability [1][8], and are of the *on-line* kind, that is, those that are required to make irrevocable commit/abort decisions at the time the transaction is processed. We do not consider other classes of protocols, such as *off-line* protocols, which are protocols that may defer commit/abort decisions until the partitions are rejoined; protocols that abandon one-copy serializability as the correctness criterion; and protocols that use type-specific information. Davidson's optimistic protocol [2] is an example of an off-line protocol. The partition-tolerant distributed databases project at the Computer Corporation of America [9] is an example of a system that abandons one-copy serializability to achieve higher availability. Herlihy [7] deals with replication methods for abstract data types.

The main objective of replicated-data management protocols is achieving availability while maintaining data consistency. No protocol whose correctness criterion is one-copy serializability can do better than a bound we have determined, under the assumption that the pattern of data accesses by transactions obeys a certain uniformity assumption that we explain. We believe this assumption is a reasonable one if we know nothing in particular about the transaction distributions; there might be some particular distributions for which specialized protocols achieve greater availability. Furthermore, this assumption permits us to do the analysis.

In the context of a simple model we have developed, we analyze the level of availability achieved by several replicated-data management protocols proposed in the literature. The protocols we look at use different rules to increase data availability during a partition. Given the authors' informal discussion of availability achievable by these protocols, it is difficult to determine how one protocol compares against the others. We provide a uniform basis for comparison. In addition, we show that several of the protocols achieve the upper bound for availability, so the bound is tight.

Our analysis shows that there is a severe limitation on the availability that can be achieved during a partition. Because of this limitation, networks should be designed to minimize the probability that partitions will occur.

This paper is organized as follows. Section 2 begins with some assumptions and definitions underlying our model, defines the notion of availability, and provides a bound on the level of availability achievable with replicated-data management protocols that maintain one-copy serializability. In section 3 we describe some of the known replicated-data management protocols and, for each protocol, give a quantitative measure of the availability achieved by the protocol. Finally, in section 4, we summarize our results.

2 Bounds on Availability

In this section, we define availability and then prove a bound on the availability achievable.

2.1 Assumptions and Definitions

The context of our work is a ***distributed database system*** in which the data is fully replicated. This system consists of a collection of n sites, numbered $1, \ldots, n$. We have chosen this special case of a distributed database system because it simplifies our analysis. *Transactions* can operate on data items by reading or updating. We assume no blind updates, where a blind update is one that updates a data item without first reading it. The set of data items updated by a transaction is called its *write set*.

A *partition* occurs in a system when two functioning sites are unable to communicate for a significant interval of time. A maximal set of sites that can communicate with one another is called a *partition group*, following Davidson [2]. We make the simplifying assumption that a partitioned network consists of only two partition groups, called a *majority partition* and a *minority partition*. This simplification does not change our conclusions because we believe that this kind of partition is rare and that more extensive partitioning is even rarer; we analyze the most common case. We define L_j as the *load* on the system for a site j based on the fraction of transactions that run there. That is,

$$L_j = \frac{\textit{number of transactions initiated at site } j}{\textit{number of transactions}}.$$

Then set S of sites is a *majority* if and only if $(\sum_{s \in S} L_s) > \frac{1}{2}$. We use this somewhat nonstandard definition of majority because it ensures the desirable property that during a partition more than one-half of the work submitted to the system is submitted to the majority partition.

In this paper, we are interested in availability. It is a measure of the amount of work that can be done by a system. We define availability as follows.

$$\textit{Availability} = \frac{\textit{number of transactions successfully completed}}{\textit{number of transactions presented to the system}}$$

We do not study other aspects of performance, such as the relative expense of read/write operations or the cost of rejoining partitions.

In order to quantify our observations concerning availability, we are interested in the following parameters.

t	=	total number of transactions presented during the partition
u_{maj}	=	fraction of t that are update transactions and are in the majority partition
u_{min}	=	fraction of t that are update transactions and are in the minority partition
r_{maj}	=	fraction of t that are read – only transactions and are in the majority partition
r_{min}	=	fraction of t that are read – only transactions and are in the minority partition

2.2 Analysis

We now show that no on-line replicated-data management protocol that maintains one-copy serializability can achieve a level of availability that is better than $u_{maj} + r_{maj} + r_{min}$.

Our proof depends on an assumption regarding system workload, which we call the *uniformity assumption*. One informal characterization, which is sufficient for the unifor-

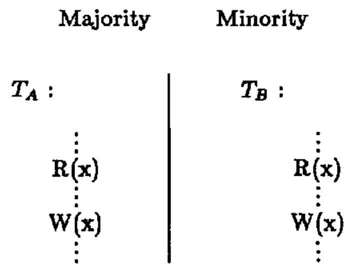

Figure 1: Execution $\mathcal{E}$

mity assumption to be satisfied, is that the transaction mix be the same at each site.

Uniformity assumption. For all D where D is a subset of the data items, and for all $j \in 1, \ldots, n$.

$$\frac{\textit{Number of } D-\textit{transactions initiated at site } j}{\textit{Total number of } D-\textit{transactions}} = L_j$$

where a D-transaction is a transaction with write set D. For example, suppose 10% of the update transactions run at site 1; then our assumption says that of those transactions that update a set of data items, 10% of them run at site 1.

We find it convenient to formulate the following correctness property, which in Theorem 1 we show is a necessary condition for maintaining one-copy serializability.

Correctness property. For all data items d, d is not updated on both the minority and majority sides of a partition.

Theorem 1. Any replicated-data management protocol that preserves one-copy serializability satisfies the correctness property.

Proof. Assume not. Then there exists a replicated-data management protocol that preserves one-copy serializability and that violates the correctness property. There must be some execution $\mathcal{E}$ of P in which the network partitions and in which some data item x is updated on both sides of the partition. Let A be the first transaction that updates x on the majority side and let B be the first transaction that updates x on the minority side. The execution $\mathcal{E}$ is illustrated in Figure 1. Because of the partition, transaction A cannot see the effects of the write to x by transaction B. Thus, transaction A must be serialized before transaction B. Analogously, transaction B must be serialized before transaction A. So, the execution $\mathcal{E}$ is not serializable. Contradiction. □

The following lemma is central to our proof of the upper bound on the availability achievable by a replicated-data management protocol that maintains one-copy serializability.

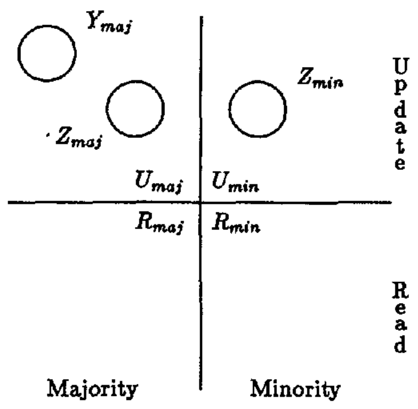

Figure 2: Sets of transactions used in proof of Lemma 2

Lemma 2. Let $\mathcal{A}$ be any replicated-data management protocol that satisfies the correctness property and that operates in a system in which the uniformity assumption holds. Any execution $\mathcal{E}$ of protocol $\mathcal{A}$ during a partition has availability at most $u_{maj} + r_{maj} + r_{min}$.

Proof. We use a counting argument to bound the availability from above. See Figure 2 for an illustration of the sets of transactions defined below.

We define the following quantities:

U_{maj} = set of update transactions presented on the majority side of the partition

U_{min} = set of update transactions presented on the minority side of the partition

R_{maj} = set of read-only transactions presented on the majority side of the partition

R_{min} = set of read-only transactions presented on the minority side of the partition

Note that
$$\begin{aligned} t &= |U_{maj}| + |U_{min}| + |R_{maj}| + |R_{min}| \\ u_{maj} &= |U_{maj}|/t \\ u_{min} &= |U_{min}|/t \\ r_{maj} &= |R_{maj}|/t \\ r_{min} &= |R_{min}|/t. \end{aligned}$$

Consider an arbitrary execution $\mathcal{E}$. Let $t_{\mathcal{E}}$ be the number of transactions that complete in execution $\mathcal{E}$. To bound $t_{\mathcal{E}}$, we make the worst-case assumption that all read-only transactions complete. To count the update transactions, let $Y_{maj} \subseteq U_{maj}$ be the set of update transactions that are initiated on the majority side and complete. Let $Z_{min} \subseteq U_{min}$ be the set of update transactions that are initiated on the minority side and complete. Then we bound $t_{\mathcal{E}}$ from above as follows:

$$t_{\mathcal{E}} \leq |R_{maj}| + |R_{min}| + |Y_{maj}| + |Z_{min}| \tag{1}$$

Let D be the set of all data items updated by transactions in Y_{maj}; similarly, let D' be the set of all data items updated by transactions in Z_{min}. By the correctness property, D and D' do not intersect. Let Z_{maj} be the set of transactions whose write sets are contained in D' and that are initiated on the majority side. By the uniformity assumption, for each possible write set in Z_{min}, there are more transactions with the same write sets in Z_{maj}; hence, $|Z_{maj}| > |Z_{min}|$. Substituting in equation (1) we get

$$t_{\mathcal{E}} \leq |R_{maj}| + |R_{min}| + |Y_{maj}| + |Z_{maj}|. \tag{2}$$

By the correctness property, no transactions in Z_{maj} complete during the partition, that is, $Z_{maj} \cap Y_{maj} = \phi$. Since $Y_{maj} \subseteq U_{maj}$ and $Z_{maj} \subseteq U_{maj}$, then $|Y_{maj}| + |Z_{maj}| \leq |U_{maj}|$. Substituting in equation (2), we have

$$t_{\mathcal{E}} \leq |R_{maj}| + |R_{min}| + |U_{maj}|. \tag{3}$$

To calculate availability from equation (3), we divide through by t, the total number of transactions presented during the partition. Then

$$Availability \leq r_{maj} + r_{min} + u_{maj}. \qquad \Box$$

The following theorem, the main result of this section, is an upper bound on availability.

Theorem 3. In a system where the uniformity assumption holds, no replicated-data management protocol that maintains one-copy serializability can achieve availability greater than $u_{maj} + r_{maj} + r_{min}$.

Proof. Assume that there exists an protocol $\mathcal{A}$ that has availability greater than $u_{maj} + r_{maj} + r_{min}$. By Lemma 2, protocol $\mathcal{A}$ violates the correctness property. By Theorem 1, protocol $\mathcal{A}$ does not preserve one-copy serializability. Contradiction. $\Box$

3 Analysis of Protocols

We are interested in database availability in the presence of long-lived partitions. The reasons why a transaction might fail to complete can be divided into two categories: transient and permanent. Transient problems, such as deadlock, will disappear if a transaction is retried sufficiently often. Permanent problems, such as the inaccessibility of data, will last as long as the partition persists. Only permanent problems are significant for analyzing the long-term behavior of a partitioned system. Therefore, we ignore the transient problems in the following analysis.

Using our new framework, we determine the level of availability achievable with each of four known replicated-data management protocols.

3.1 Gifford's Weighted Voting Protocol

Gifford [6] presents a simple and elegant protocol for maintaining the consistency of replicated data in a distributed computer system. The basic idea of the protocol rests on the notion of *quorum intersections.* Each copy of a replicated data item is assigned some number of votes. To read a data item, a transaction must collect a read quorum of votes; to write a data item, it must collect a write quorum of votes. To maintain the consistency of the replicated data, these read and write quorums must satisfy two constraints. First, read and write quorums must intersect, guaranteeing that any read quorum has a current copy of a data item. Second, write quorums must intersect, imposing an order on updates. Together, these two rules ensure one-copy serializability. The protocol has several additional benefits: it continues to operate correctly even if some copies are inaccessible, it is possible to change a data item's performance and reliability characteristics by altering quorum sizes, and it also copes with partitions without explicit detection. In this section, we consider the two ends of a range of possible quorum sizes.

Let S be some set of sites in the system. We define the vote of S in terms of the transaction load. That is,

$$Vote\ of\ S = \sum_{s \in S} L_s.$$

This apportionment of vote, which ensures that the majority partition has a higher vote than the minority partition, allows the maximum number of transactions to complete during the partition.

Scheme 1. Set S is a read quorum if the vote of $S > 0$; S is a write quorum if the vote of $S = 1$. The advantage of such a choice of quorum sizes is that if a transaction reads more data items than it updates, it is typically reading only a single copy of a data item (usually the local one); hence, it costs little more than the same transaction executing against a non-replicated database. The disadvantage is that if there is a greater proportion of updates than reads, then a transaction incurs the expense of updating all copies of a data item every time it executes a write. In the presence of a partition, read operations will succeed because only one copy of a data item need be read and that can occur on either side of a partition. Write operations, however, will never complete during a partition because all copies of a data item must be accessible. Therefore, read-only transactions are the only kinds of transactions that will succeed. In terms of our model, the level of availability achievable in this variation of Gifford's weighted voting protocol is

$$Availability = r_{maj} + r_{min}.$$

Scheme 2. This variation differs from the previous scheme in that it permits more writes because it does not require that all copies be available for writing. Set S is a write quorum in this scheme if it is a minimal set such that the vote of $S > \frac{1}{2}$; set S is a read quorum if it is a minimal set such that the vote of $S \geq \frac{1}{2}$. The penalty incurred for cheaper write operations is that a transaction must have at least half of the votes in order to read a data item. By definition the total vote of all the sites on the majority side of the partition is more than a half. Thus, read and write quorums can be constructed on the majority side. Analogously, the vote of all the sites on the minority side is less than a half; neither read nor write quorums can be constructed on the minority side. Both read and write operations will succeed on the majority side, but neither will succeed on the minority side. The availability is

$$Availability = u_{maj} + r_{maj}.$$

Analysis of Gifford's protocol. We observe that the above two variants of Gifford's protocol constitute the ends of a continuum of quorum choices afforded by his protocol. We shall show that the availability achievable by choosing intersecting quorums within the range of the two endpoints is either $u_{maj} + r_{maj}$ or $r_{maj} + r_{min}$.

Let w_{qrm} be the fraction of votes in a write quorum. $w_{qrm} > \frac{1}{2}$ because write quorums must intersect. When $w_{qrm} = 1$ the first scheme results. There exist values of w_{qrm} sufficiently close to $\frac{1}{2}$ such that the second scheme results.

The size of the read quorum must be greater than $1 - w_{qrm}$ in order that the read and write quorums intersect. For example, if $w_{qrm} = \frac{2}{3}$ then the read quorum size must be greater than $\frac{1}{3}$. The only possible values of w_{qrm} are greater than $\frac{1}{2}$ and less than or equal to 1. If p_{maj} is the fraction of votes in the majority partition and p_{min} is the fraction of votes in the minority partition, then $p_{min} = 1 - p_{maj}$. There are two cases to consider.

1. $p_{maj} < w_{qrm}$. This relationship implies that no transaction can update items because of an insufficient write quorum size. On both sides of the partition transactions can only read. The availability is $r_{maj} + r_{min}$.

2. $p_{maj} \geq w_{qrm}$. This relationship implies that there are enough votes on the majority side to satisfy the write quorum size. Thus, transactions can read or write on the majority side of the partition. The vote in the minority partition p_{min} is smaller than the required read quorum size, $1 - w_{qrm}$, so transactions cannot even read on the minority side. The availability is $u_{maj} + r_{maj}$.

Our result is that the availability of Gifford's protocol with parameter w_{qrm} is

$$Availability = \begin{cases} r_{maj} + r_{min} & \text{if } p_{maj} < w_{qrm}; \\ u_{maj} + r_{maj} & \text{otherwise.} \end{cases}$$

3.2 Missing Writes

Eager and Sevcik's protocol [3] is a more complicated replicated-data management protocol than Gifford's and attempts to achieve higher availability by switching between different quorum sizes; extra mechanism is required to accomplish this switch without loss of one-copy serializability. Transactions run in either of two modes: *normal mode* or *partitioned mode.* In normal mode, transactions follow scheme 1 of Gifford's protocol. When sites fail or when the network partitions, normal mode transactions make no progress if they must collect all possible votes to update a data item. To make progress, normal mode transactions abort. They restart in partitioned mode and follow scheme 2 of Gifford's protocol.

Since normal mode transactions can coexist with partitioned mode transactions, a normal mode transaction might be serialized before *and* after some partitioned mode transaction, leading to non-serializable behavior. To prevent this violation from occurring, the protocol guarantees that partitioned mode transactions are always serialized *after* normal mode transactions. The technique is to maintain data structures that keep track of the updates to data items that were not made at other sites storing copies of the data items because of failure—the so-called *missing write* information. Whenever a normal mode transaction has read a data item that has missing write information associated with it, it must abort. To make progress, it restarts in partitioned mode. Because we are interested in availability in the long term, the way in which this information is propagated and the speed with which it is propagated are irrelevant to our analysis.

All transactions executing on the majority side run to completion. The contribution to total system availability by these transactions is $u_{maj} + r_{maj}$. On the minority side of the partition, normal mode read-only transactions can run successfully (contribution to total availability is r_{min}). Writes are impossible because normal mode requires that all copies be available for writing. If update transactions restart in partitioned mode, they cannot make progress because a majority of votes for the read and write quorums cannot be attained. The total availability is

$$Availability = u_{maj} + r_{maj} + r_{min}.$$

3.3 Virtual Partitions

Another variation on Gifford's weighted voting protocol is the *virtual partition* scheme of El Abbadi, Skeen, and Cristian [4]. Virtual partitions attempt to track real changes in the network topology as closely as possible without being constrained by the need to cope with changes instantaneously. A virtual partition is a set of nodes that have agreed that they can communicate with each other and further agree that they will not communicate with any processors outside the partition. The ability to communicate within a partition may be subsequently lost due to failures. Although communication with processors outside a virtual partition may be physically possible, this communication may not be initiated until a special protocol is run to form a new virtual partition.

The virtual partition scheme permits cheap read operation as in Gifford's scheme 1; yet it allows write operations to be performed in the majority partition as in Gifford's scheme 2. The cheaper read operations come at the expense of a protocol that updates every item in the database when partitions are rejoined.

The authors enumerate rules and properties that they show are sufficient to ensure one-copy serializability. We list the ones that affect our calculations of availability. There are four rules that govern transaction execution. The *accessibility rule* is that a logical data object is accessible from a processor in a virtual partition if a simple majority of copies reside on processors in its virtual partition. The *read rule* is that a transaction may read from the nearest copy in its virtual partition, but the logical data object must first be accessible. The *write rule* is that a transaction must write to all copies in its virtual partition (which may be more than a simple majority), but the logical data object must first be accessible. The *one-partition rule* is that the execution of an individual transaction does not span virtual partitions.

Our analysis of availability depends heavily on the rule used to decide whether to form a new virtual partition. We call this rule the *tracking strategy*. Several tracking strategies are possible. The choice of strategy is not fundamental to the virtual partitions scheme. The particular strategy proposed by the authors of the scheme, which we call *aggressive tracking*, is to initiate the formation of a new virtual partition soon after a change in the physical network is discovered. Unfortunately, this strategy limits the level of availability achieved. We propose an alternative strategy, which we call *lazy tracking*, that achieves higher availability. This strategy is that a new virtual partition is formed only if read operations are possible in the virtual partition. In proposing this strategy, we depend on our assumption of full replication.

In terms of our model, we calculate the level of availability achievable in the virtual partition scheme for both tracking strategies mentioned above. As before, we suppose the physical network partitions into a majority side and a minority side.

Aggressive tracking. A virtual partition is formed for the majority side and another is formed for the minority side. Read operations are possible on the majority side due to the read rule. Update operations are also possible due to the write rule. Thus, both read and write transactions will complete if they run on the majority side (contribution to total availability is $r_{maj} + u_{maj}$). Consider transactions presented on the minority side of the partition. Because of the accessibility rule, no transaction can run at all; a majority of copies is not accessible. Thus, in the presence of a network partition, this scheme behaves in essentially the same way as Scheme 2 of Gifford's weighted voting protocol.

The availability is

$$Availability = u_{maj} + r_{maj}.$$

Lazy tracking. A virtual partition is formed for the majority side because read operations are possible. If a virtual partition were formed for the minority side, read operations would be impossible in that partition; therefore, the sites on the minority side retain their view that the network is unpartitioned. The same level of availability is achieved on the majority side as in the first case, $u_{maj} + r_{maj}$. The accessibility rule is satisfied on the minority side because the processors in the minority side still think they are in an unpartitioned network. Hence, read-only transactions are possible (contribution to total availability is r_{min}). The availability is

$$Availability = u_{maj} + r_{maj} + r_{min}.$$

A recent replicated-data management protocol similar to virtual partitions has been proposed by El Abbadi and Toueg [5]. Their new protocol provides greater flexibility in quorum choices and other improvements on the virtual partitions scheme. These differences, however, do not affect the analysis of the maximum availability possible. Their scheme is based on a concept similar to virtual partitions, which they call *views*. When the quorum sizes are set in their scheme in the way that provides the maximum availability, views behave the same way as virtual partitions. In that case, all the previous analysis, including tracking strategies, applies without change to the El Abbadi and Toueg scheme. Quorum sizes could also be chosen to provide the same availability as in Gifford's scheme 1, if availability of read-only transactions is deemed more important than that of update transactions.

3.4 Privileged Partitions

The notion of a *privileged partition* is suggested by Wright and Skeen [10] in their paper on class conflict graphs. In the *class conflict graph* scheme, the basic technique is to divide transactions into classes according to data access patterns, assign these classes to partitions, and construct a graph (called a class conflict graph) showing all possible interactions between the classes. Given this graph, one identifies those interactions between classes that could lead to non-serializable behavior and deletes those classes from partitions until the graph is acyclic. This pre-analysis is performed before any transactions are actually executed. When the system partitions, the transactions in the classes can be executed to completion without conflict.

This approach extends other replicated-data management protocols by relaxing some restrictions that those protocols impose on the reading and updating of a data item in different partitions. A simple application of their method that does not require pre-analysis is the notion of a "privileged" partition. We can adapt Gifford's protocol, where the read and write quorums require a majority of votes, to achieve greater availability while still guaranteeing one-copy serializability. We choose one partition to be the privileged partition, where an protocol can write whatever it did before, but in addition, it can read *any* item. The class conflict graph analysis guarantees the graph to be acyclic. The difference between this protocol and Gifford's is that partitions must be detected in some fashion and the data repaired after partitions are rejoined; in Gifford's protocol, partitions need not be explicitly detected, nor does the database require repair.

Suppose the privileged partition is the minority partition. Gifford's protocol is run on both sides of the partition. On the majority side, transactions can read and write data items; on the minority side, Gifford's protocol by itself allows no transactions. But read-only transactions can run on the minority side because of this special rule for privileged partitions. Assuming the minority partition is the privileged partition, the availability is

$$Availability = u_{maj} + r_{maj} + r_{min}.$$

Suppose the privileged partition is the majority partition. Both read-only and update transactions can complete on the majority side; neither can complete on the minority side. Assuming the majority partition is the privileged partition the availability is

$$Availability = u_{maj} + r_{maj}.$$

Given our assumption of full replication there is a method that ensures that the minority partition is the privileged partition. Unfortunately, this method does not generalize to systems without full replication.

4 Conclusions

A contribution of this paper is its uniform analysis of database availability that can serve as a basis for comparison of other replicated-data management protocols. We have shown, given our uniformity assumption, that no replicated-data management protocol that maintains one-copy serializability achieves availability greater than $u_{maj} + r_{maj} + r_{min}$. Additionally, we have shown that there are replicated-data management protocols that achieve the bound. Thus, the bound is tight.

Our analysis shows that there is a severe limitation on the availability that can be achieved during a partition. Because of this limitation, networks should be designed to minimize the probability that partitions will occur. We believe that it is technically feasible to make network partitions highly unlikely by building sufficiently reliable networks using a combination of the following techniques: high connectivity; software security, such as preventing malicious application programs from corrupting the operating system or gateway; and physical security, such as keeping people away from the machines.

But even though the likelihood of a network partition may be small given such a sufficiently reliable network, software should still preserve one-copy serializability in the presence of partitions. A system should not fail in a catastrophic way. When choosing a replicated-data management protocol from among several, the criteria one uses need not necessarily be availability alone; other factors, such as ease of implementation and cost of repairing a database when partitions rejoins, should be considered as well.

Acknowledgments

We thank Jim Gray of Tandem Computers for a helpful discussion on the merits of hardware versus software solutions to the problem of high data availability. We thank Gary Leavens who suggested we write this paper, and Sunil Sarin, who encouraged us. Finally, we thank Amr El Abbadi, Maurice Herlihy, Jennifer Lundelius, Sam Toueg, and Bill Weihl, whose comments proved helpful in preparing the final version of this paper.

References

[1] Philip A. Bernstein and Nathan Goodman. "Multiversion Concurrency Control–Theory and Algorithms". *ACM Transactions on Database Systems*, 8(4):465–483, December 1983.

[2] Susan B. Davidson. "Optimism and Consistency in Partitioned Distributed Database systems". *ACM Transaction: on Database Systems*, 9(3):456–481, September 1984.

[3] Derek L. Eager and Kenneth C. Sevcik. "Achieving Robustness in Distributed Database Systems". *ACM Transactions on Database Systems*, 8(3):354–381, September 1983.

[4] Amr El Abbadi, Dale Skeen, and Flaviu Cristian. "An Efficient, Fault-Tolerant Protocol for Replicated Data Management". In *Proceedings of the 4th ACM SIGACT/SIGMOD Conference on Principles of Data Base Systems*, 1985.

[5] Amr El Abbadi and Sam Toueg. "Maintaining Availability in Partitioned Replicated Databases". In *Proceedings of the 5th ACM SIGACT/SIGMOD Conference on Principles of Data Base Systems*, 1986. Held at the Hyatt Regency Hotel, Cambridge, Massachusetts, March 24-26, 1986.

[6] David K. Gifford. "Weighted Voting for Replicated Data". In *Proceedings of the 7th ACM Symposium on Operating Systems Principles in Operating Systems Review Vol. 13, No. 5 (December 1979)*, pages 150–162, 1979. Asilomar Conference Grounds, Pacific Grove, California, December 10-12, 1979.

[7] Maurice P. Herlihy. "A Quorum-Consensus Replication Method for Abstract Data Types". *ACM Transactions on Computer Systems*, 4(1):32–53, February 1986.

[8] Christos H. Papadimitriou. "Serializability of Concurrent Database Updates". *Journal of the ACM*, 24(4):631–653, October 1979.

[9] Sunil K. Sarin. "Robust Application Design in Highly Available Distributed Databases". In *Proceedings of the 5th IEEE Symposium on Reliability in Distributed Software and Database Systems*, pages 87–94, 1986. Held at the Marriott Hotel, Los Angeles, California, January 13-15, 1986.

[10] Dale Skeen and David D. Wright. "Increasing Availability in Partitioned Database Systems." Technical Report TR 83-581, Department of Computer Science, Cornell University, March 1984.