1989

Communication-based Recovery in Replicated Databases

Bharat Bhargava
Purdue University, bb@cs.purdue.edu

Shirley Browne

Report Number:

89-891

COMMUNICATION-BASED RECOVERY IN REPLICATED DATABASES

Bharat Bhargava
Shirley Browne

CSD-TR-891
July 1989

Communication-based Recovery in Replicated Databases *

Bharat Bhargava and Shirley Browne
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

Abstract. This research investigates the use of communication to improve the efficiency of recovery in a replicated database. Current technological trends indicate that network speeds are improving more rapidly than disk access times. The techniques described in this paper should help avoid the disk I/O bottleneck in replicated database systems. Communication-based recovery schemes are described in the contexts of quorum methods for replicated data and of synchronization of logical clocks. Previous algorithms for transaction processing are unchanged, with the exception that writing of recovery data to disk may be optimized at the cost of more expensive reads, which are expected to be rare. The recovery algorithms for a site attempt to recover data and quorum assignments by communicating with other sites and read from stable storage only as a last resort. Conditions under which the communication-based techniques can fail are given. The interaction of increased reliance on communication with another efficiency-improving technique, namely adaptability, is also discussed.

*This research is sponsored by NASA and AIRMICS under grant number NAG-1-676, by UNISYS Corporation, and by a David Ross fellowship.

1 Introduction

This research investigates the use of communication to improve the efficiency of recovery in a replicated database. Conventional recovery schemes rely on logging events to stable storage during transaction processing and recovering the state of the database from disk following a failure. An alternative strategy in the case of a replicated database is to copy the database state from neighboring sites. Since copying the entire state of a large database will not be practical, we describe how communication may be used to implement recovery by copying only when necessary. Our communication-based recovery schemes are described in the context of quorum-based methods for replicated data. Quorum-based methods maintain the consistency of replicated data in the presence of either site failures or network partitioning [12]. We are concerned not only with the recovery of the data objects themselves, but also with recovery of quorum assignment data. We point out cases for recovery of both types of data where the extent of failures may necessitate falling back on the use of stable storage as a last resort.

With disk-based recovery, a recovering site reads its local log to redo committed updates and to determine what data are blocked by dangling precommits. One problem with the disk-based recovery of data for a site is that the database state recovered from the local log will be out-of-date, especially if the site has been down for a long time. Thus, communication will be required anyway to access up-to-date copies of objects. Secondly, metadata that are being continuously updated in a distributed environment require communication to obtain the latest copy; storing such data on stable storage is of no use. Yet most proposed and/or implemented recovery schemes for distributed databases assume a priori that conventional disk-based recovery is carried out on a per-site basis. Quorum-based replication control schemes [12,1,13,15] make this assumption as well, since a copy at a recovered site is free to participate in a quorum once the local recovery is finished. In a static quorum scheme, the failure of a site does not exclude it from any quorums. With a dynamic quorum scheme, however, it is likely that a failed site has been excluded from the current view and is no longer included in quorum assignments for objects in that view.

Either a new view must be formed or the current view extended before the local copy can be included in quorum assignments. Both these actions require accessing a remote read quorum. Hence, any effort spent on local recovery of the object will be wasted. Log-based recovery may still be necessary as a last resort if enough copies have failed that a read quorum cannot be obtained from the non-failed sites.

Another aspect of this research is that it may be more efficient to copy from a neighboring site than to read from a local disk or from a file server. Current technological trends indicate that network speeds are improving more rapidly than disk access times. FDDI, a fiber-optic high-speed network currently under development, provides a data-transfer rate of 100 MB per second and can support up to 1000 connections at distances of up to 2 kilometers between nodes [9]. Systems as currently designed may be unable to take advantage of such a high-performance network. As reported in [9], a node that attempts to also do disk operations will be buried by the sustained throughput on the FDDI. Implementation of communication by means of distributed shared memory has been proposed for local-area networks [10] and more recently for wide-area network [11]. For homogeneous systems this approach has promise for matching the speeds of future high-speed networks. Distributed shared memory, where an attempt to access a portion of memory residing on another machine is treated analogously to a page fault on a virtual storage system, is likely to further increase the performance advantage of inter-node communication over local disk access. Another trend in networking is toward the use of dedicated remote file servers rather than local disk storage. If several diskless workstations have failed or if the file server is busy processing logging requests for committed transactions, the file server becomes a bottleneck for recovery. If memory-resident data at other workstations can be copied by the recovering workstations, such a bottleneck will not occur.

This paper is organized as follows: In section 2, we discuss issues concerning the use of disk storage. Communication-based recovery techniques for a replicated database that uses quorums are described in section 3. We also indicate the conditions under which the techniques can fail and explain how stable storage can be used as a backup. In section 4, the effect on logical clocks of increased reliance on communication is explored. Discussion of related work is in section 5, and conclusions are presented in section 6.

2 No Disk vs. Limited Use of Disk

The concepts and feasibility of communication-based recovery with complete elimination of disk-based recovery have been studied in [5]. If site failures can be assumed to be independent and of probability p, then writing the updates of a transaction to a threshold number T of sites commits the transaction with probability $1 - p^T$. The approach in [5] is for a fully replicated database but can be extended to partial replication by establishing the threshold separately for individual objects. Eliminating stable storage has the effect of speeding up transaction commitment, since log records need not be forced to disk.

Before discarding disk-based logging, however, we analyze the consequences of relying completely on communication for recovery. One consequence is the possible loss of updates in the event of total failure of all the copies of an object. Even if disk-resident copies exist, these copies may not be up-to-date at the time the failure occurs if disk writes are asynchronous to transaction processing. Another consequence is increased probability of blocking in a network that is subject to partitioning. For example, suppose that all the sites within a small partition have just recovered from a failure. If the number of failed sites is less than the threshold T, no updates will have been lost. Even if data are disk-resident at the recovered sites, however, these sites will have no way of determining what committed updates should be redone and what objects are blocked by dangling precommits. Although read quorums for some objects may be available in the small partition, access to these objects will be blocked until communication with a sufficient number of non-failed sites can be achieved.

Rather than eliminating disk-based recovery entirely, it could be used as a last resort. The idea is to streamline writing at the expense of increased cost for reading and processing log records. Several alternatives can be considered. The log could be written to a raw disk partition, rather than using the system call that provides I/O synchronization for block devices. A single specialized remote logging facility could be provided in each possible partition of the network, for example in each of two LAN segments joined by a bridge. This facility might make use of a small battery backed-up stable memory in which log records could be buffered, as in [19]. Another possibility for a memory-resident database would be to use a log-structured file system, as described in [20].

3 Communication-based Recovery for Quorum Methods

The use of quorums for maintaining consistency in replicated databases is a well-known technique, originating from [12]. A quorum assignment for a replicated object specifies how many or which copies must be accessed to carry out a read or write operation. A set of copies that suffices to carry out an operation is called a *quorum*. A dynamic quorum method that allows quorum assignments to be changed during transaction processing provides a generic framework that encompasses a wide variety of replication control methods, including token passing [23], available copies [2], and dynamic voting [17,21]. In sections 3.1 and 3.2, we explain how communication-based recovery may be integrated with static and dynamic quorum methods. In section 3.3, we investigate the effects of combining communication-based recovery with another efficiency-improving technique, namely adaptability.

3.1 Static Quorum Methods

With a static quorum method [12], quorum assignments are fixed at system startup time. A recovering site may obtain the quorum assignment for an object from any other site that has a copy of the object or from a name server site. When no such site is available, quorum assignments may be read from stable storage.

With disk-based recovery, a recovering site reads and processes log records from its stable storage in order to redo committed updates and to determine what objects are blocked by dangling precommits [3]. After the local recovery is finished and dangling precommits are resolved, copies at the recovered site may participate in quorums for both read and write operations. There is no need to bring local copies up-to-date with other copies, since the quorum intersection requirement will ensure that the most recent value is obtained from a read quorum.

With communication-based recovery, a copy at a recovering site does not participate in a read quorum until it has either participated in a write quorum or copied the value of the object from a read quorum. If the objects are large, an alternative is to read the version number from a read quorum to determine if the local copy needs to be updated. If not, the value could be loaded into memory from local stable storage. If network speed and bandwidth are high compared to that for disk, however, the value can be copied from another site instead.

When a site starts recovery, communication-based recovery blocks access to every object until the blockage is cleared by a write or a copy operation. With a memory-resident database scheme, it might be efficient to load the entire database into memory all at once but still be inefficient to read and process log records, especially if the checkpointing interval were large. In this case, in-memory logs at other sites could be scanned to determine what objects are blocked. The blocked objects are those for which committed updates are not known to have been recorded on disk or for which the outcome of a transaction that updated them cannot be determined. The in-memory log scheme described in [5] could be extended to partial replication and quorum methods by reading logs at a union of read quorum sites for all the objects residing at the recovering site. A blocked object would be "fail-locked" [6] until the fail-lock is cleared by a write or a copy. All other objects would be free to participate in read quorums. Clearing the blockages makes access to a read quorum potentially more efficient, since it reduces the probability that a quorum member will need to read the log. Experimental data in [6] provide evidence that a two-stage method of clearing blockages gives good results. During the first stage, fail-locks are cleared by transaction processing. During the second stage, the remaining fail-locks are cleared in batch mode.

3.2 Dynamic Quorum Methods

Background. With a dynamic quorum method [13,14], quorum assignments may be changed while transaction processing is taking place. This flexibility allows the tuning of the quorum assignments for performance reasons, as well as the addition or deletion of copies. Quorum assignments can be changed to adapt to failures without having to terminate all active transactions. This capability is important for handling a network partitioning failure, since all active transactions cannot necessarily be terminated, even if a non-blocking commit protocol is being used [22]. A dynamic quorum method can be used to enhance availability in the event of subsequent failures, for example by implementing dynamic voting, which has been hypothesized to provide optimal availability for replicated data [17].

Two types of quorum assignments may be maintained for an object:

1. The *active* quorum assignment is read to determine what copies of the object must be accessed to carry out an operation.

2. The *backup* quorum assignment is used following a site or partitioning failure that renders active quorums unavailable to determine what the new active quorum assignment should be.

Using both types of quorums yields better availability during failures without sacrificing performance in the absence of failures.

To ensure that the correctness criterion of one-copy serializability is met, changes to active quorum assignments are coordinated by using views [1,13,4]. A view is associated with a connected component of the network. The representation of a view typically includes a unique integer view id and a connection vector with a 1 entry for every site in the view's component and a 0 entry for each of the other sites. An object is accessible for a given operation in a view if it has a backup quorum for that operation in the view's component. With most view-based methods that tolerate network partitioning, explicit reconfiguration of all objects is carried out at view formation time by accessing backup quorums. A transaction is required to execute entirely within a single view. The quorum intersection requirements ensure that transactions executed within the same view are serializable. Transactions executed in different views are serializable in order of their view ids.

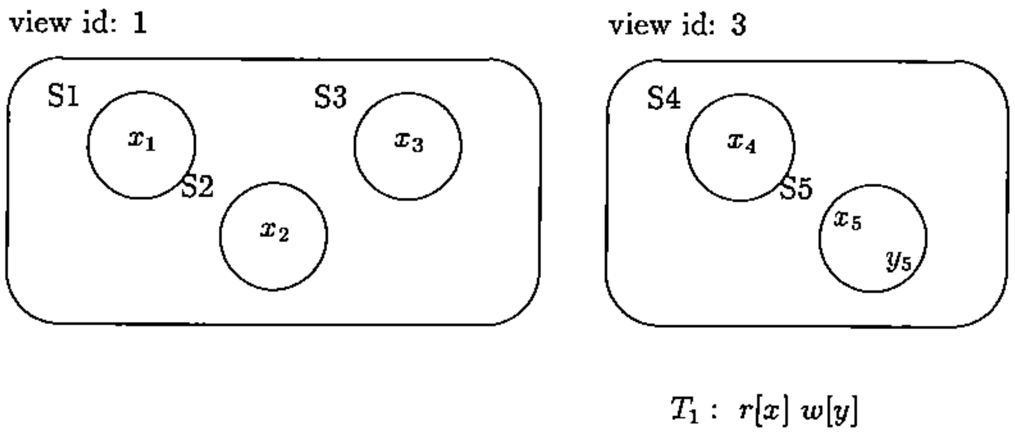
In most previous work, the assumption is made that changes to quorum assignments and view changes are written to stable storage and that this information is recovered from stable storage following a failure. Our research investigates ways in which either the reading or both the reading and the writing can be eliminated. With most previous view methods, a completely new view is formed whenever any type of failure or recovery occurs. Optimizations that allow existing views to be extended and old views to be re-used are investigated in [4]. Section 3.3 explores the effects of communication-based recovery on these optimization techniques. Below we discuss changes needed in dynamic quorum methods.

Quorum assignments. If a completely new view is formed when a site recovers, then there is no need to find out about the old active quorum assignments, since these will be reassigned anyway. Consequently, active quorum assignments may be maintained in volatile storage only. If total failure is a possibility, changes to backup quorum assignments must be written to stable storage As changes to backup quorum assignments require the participation of at least one backup read quorum and one backup write quorum, however, the backup quorum assignment for an object may be obtained without resorting to stable storage if either type of backup quorum is available among the non-failed sites. Any object for which a backup read quorum is available among the non-failed sites may also be reconfigured without accessing stable storage.

View ids. The question arises as to whether view ids must be written to stable storage at view formation time and read from stable storage when a site recovers. A strategy of writing a new view id before it is used at a site but of reading only as a last resort, as explained below, should give good results. Transactions executed in different views should be serializable in order of their view ids, since much of the theory involved with the use of views depends on this property [1,13,4]. The site establishing a new view normally collects the view ids from all sites in the new view and chooses a new view id greater than any of those received. If a failed site does not recover its view id from stable storage, however, a view id less than that for a view in which the failed site has participated may be chosen. This can result in transactions not being serializable in order of their view ids, as illustrated by Example 1 below.

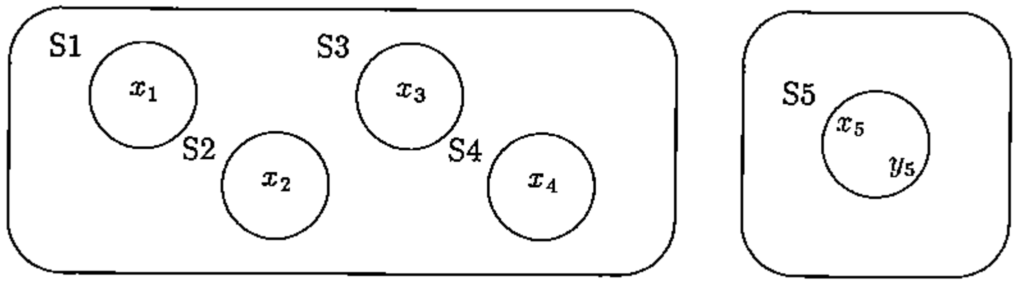
Example 1.

Suppose object X is replicated at sites S1, S2, S3, S4, and S5 and that object Y resides at site S5. Let the backup quorum assignment for X be read-two/write-four. Suppose sites 4 and 5 make up view 3 and that $T_1 : r[x]w[y]$ executes in view 3.



Then site 4 fails and comes back up in a partition with sites 1, 2, and 3, forming view 2. Transaction $T_2 : r[x]w[x]$ executes in view 2.

view id: 2



$T_2: \ r[x] \ w[x]$

Clearly T_2 should be serialized after T_1, since T_1 does not read the value written by T_2. Since the view formation for view 2 was able to access a backup read quorum for X from the non-failed sites, it did not have to resort to reading stable storage for this purpose. If we assume that new view ids are written to stable storage before they are used, then the following theorem shows that extending the requirements for access to backup quorums at view formation time will ensure serialization in order of view ids while still allowing access to stable storage to be avoided in many cases of site recovery.

Theorem 1. If the view formation protocol reads the view id from the sites in a backup write quorum for every object that is write-accessible in the new view, where the read is from stable storage at any site that has failed, then transactions from different views will be serializable in order of their view ids.

Proof. Suppose there are two transactions whose serialization order is different from their view ids. Let the transactions be T_1 and T_2 with view ids V_1 and V_2, respectively, where $V_1 < V_2$.

Case 1: T_1 reads some object X and T_2 writes the same object X, and T_1 reads the value written by T_2.

Since X will have been moved to V_2 at all sites in the write quorum used by T_2, the read quorum used by T_1 cannot intersect this write quorum if T_1 is to be able to execute in V_1, a contradiction.

Case 2: T_1 writes some object X and T_2 reads the same object X, but T_2 does not read the value written by T_1.

V_2 must have a backup read quorum for X, each site of which would have found out about V_2 at view formation time and written the view id for V_2 to stable storage. V_1 must either contain a backup write quorum for X among the non-failed sites or must read the view id from stable storage at enough failed sites to make up a backup write quorum. In either case, the backup write quorum for V_1 intersects the backup read quorum for V_2, and V_1 would have been made greater than V_2, a contradiction. $\bowtie$

In Example 1, view 2 would have had write accessibility for X but would not have had a write quorum among the non-failed sites. To satisfy the conditions of Theorem 1, the view formation protocol would have been forced to read the view id from stable storage at site 4 and would have found out about view 3 and hence would have chosen a view id greater than 3.

3.3 Adaptable Dynamic Quorum Methods

Recovery techniques for a dynamic quorum method that adapt the actions taken to the duration and extent of failures are described in [4]. These techniques include the following:

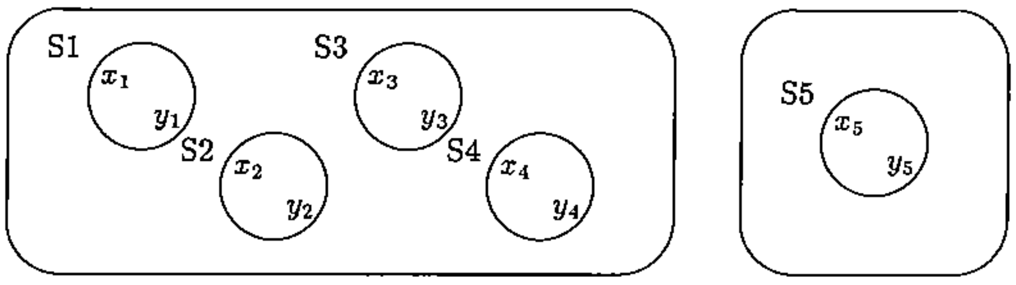
- a lightweight protocol for changing quorum assignments without forming a new view whenever possible,
- allowing objects to join a new view on demand,
- extension of an existing view to include a recovering site,
- inheritance of objects and their quorum assignments by a new view from an old view.

Extending a view to handle site recovery is cheaper than forming a new view since extension can be done using a one-phase protocol and only those objects with copies at the recovering site will need to reconfigure their quorum assignments. Inheritance allows an old view that is still largely intact to be re-used so that only those objects that have been deleted from the old view will need to be reconfigured.

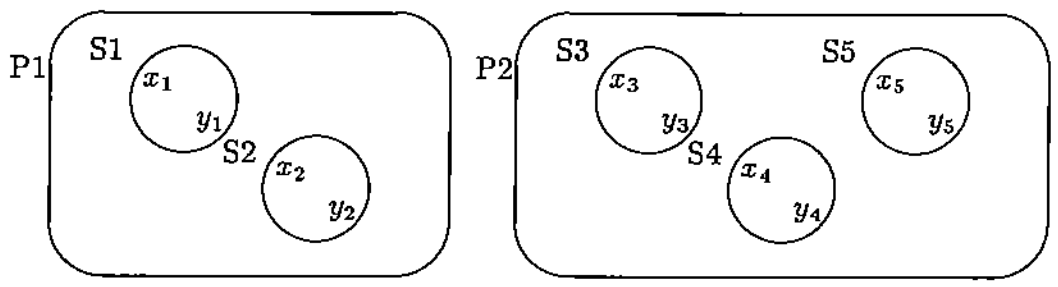
Quorum assignment changes. It is shown in [4] that a quorum assignment may be changed without forming a new view if both a read and a write quorum are available. This type of quorum assignment change is called *lightweight.* The assumption is made that changes to quorum assignments are written to stable storage. If site recovery does not force the formation of a new view and if active quorum assignments are not read from stable storage by a recovering site, a nonserializable execution can result, as shown by example 2 below.

Example 2.

Suppose sites S1, S2, S3, S4, and S5 all have copies of both X and Y. Let the backup quorum assignment for X be read-three/write-three, and the backup assignment for Y be read-four/write-two. Let S5 be partitioned from the other sites as shown below. A lightweight quorum change transaction coordinated at S1 changes the active assignment for X from read-three/write-three to read-two/write-four and the active assignment for Y from read-three/write-three to read-four/write-two.



Sites S3 and S4 fail following the quorum change and recover connected to S5 but partitioned from S1 and S2. Transaction T_1 coordinated at S1 attempts to run in P1, and T_2 coordinated at S5 attempts to run in P2.



$T_1: \ r[x] \ w[y]$ $T_2: \ r[y] \ w[x]$

If the new active assignments are used in P1, but the old assignments are used in P2, the incorrect execution will be allowed. If the recovery of S3 and S4 forces a new view formation, however, Y will not be read-accessible in P2 and T_2 will not be able to execute.

It is undesirable to force a new view formation after every site recovery. It results in greater communication overhead if a site failure is of short duration and a new view excluding the failed site has not been formed. More importantly, it rules out use of the view extension and inheritance techniques.

One solution for ensuring one-copy serializability with communication-based recovery is to keep active quorum assignments in volatile memory but to write lightweight quorum assignment changes to all old quorum members, instead of just requiring that they be written to a read and a write quorum. Another alternative is to write active quorum assignment changes to stable storage but read them on recovery only if access to stable storage is required anyway to access a read quorum for an object. In example 2, T_2 will need to read from stable storage to access a read quorum for Y and would find out about the quorum assignment change for Y to read-four/write-two. That such a strategy prevents non-serializable executions is proved by the following theorem:

Theorem 2. If lightweight quorum assignment changes are written to stable storage at at least an old read quorum and an old write quorum and are read from stable storage whenever the corresponding object is read from stable storage at a site, transactions executing within a single view will be one-copy serializable.

Proof. The proof of one-copy serializability within a single view from [4] still holds. ⋈

The above discussion illustrates how different techniques for making recovery more efficient can interfere with one another. If a new view is formed upon site recovery, then we can avoid writing active quorum assignments to stable storage, as explained in the previous subsection. When we try to make recovery more efficient by using view extension to handle site recovery, we find that we need to revert to writing quorum assignment changes to stable storage. Determining which technique is better would depend on the relative frequency of quorum assignment changes and site recovery and on the costs involved.

View extension. We now deal with the question of whether a view can be extended to include a recovering site without requiring the site to read its most recent view id from stable storage. When a site extends an existing view to include itself, it communicates with all sites in the view to ensure that none have joined a newer view. The recovering site also should not have joined a newer view. Example 1 from section 3.1 illustrates that if the view id s not recovered from stable storage, transactions may not be serializable in the order of their view ids. The following theorem shows how the view extension protocol from [4] can be modified for use with communication-based recovery.

Theorem 3. If the view extension protocol reads the view id from the sites in a backup write quorum for an object that is write-accessible in the extended view, where the read is from stable storage at any site that has failed, then the recovering site may participate in a write quorum for the object without violating one-copy serializability in order of view ids.

Proof. The proof in [4] for one-copy serializability within a view holds with minor modifications.

An argument similar to the proof of Theorem 1 shows that transactions executed in different views will be serializable in order of their view ids. $\bowtie$

The easiest way to enforce the condition of Theorem 3 would be to have a site indicate in its reply to the view extension request if it had also been included in the current view by an extension and had not read its most recent view id from stable storage. Such a site should not be counted as a member of a backup write quorum. A consequence of Theorem 3 is that a recovering site that holds the only copy of an object or, more generally, whose copy is needed to make up a backup write quorum, will not be able to participate in a write quorum for the object until it has read its most recent view id from stable storage. In Example 1, site 4 would not have been able include itself in a write quorum for X without forming a new view, since it must access stable storage to satisfy the condition of Theorem 3 and would have found out about view 3.

If the view extension protocol finds that the recovering site is still included in the current view, then no quorum assignments need to be changed, but the site must still verify that a backup write quorum is available among the non-failed sites before participating in a write quorum. If the site must be added to the view but an object is already accessible in the view, the active quorum assingment can be extended using the lightweight protocol. If the object is accessible only after the view is extended, the protocol described in [4] that allows objects to join a view on demand may be used.

Inheritance. Although inheritance may still be used with communication-based recovery, extra care is needed to ensure that essential information is not lost as a result of site failures. In this case, the information that might be lost concerns deletion of objects from the old view from which the inheritance is being done. Inheritance may be used if the sites in a new view are a superset of those in the old view. Instead of forming an empty new view, a view formation that uses inheritance allows all objects that have not been deleted from the old view to be included in the new view. No reconfiguration is needed for these objects, and their old quorum assignments may be used in the new view. It is essential for correctness, however, that all sites in the new view delete all objects that have been deleted at any site. Deletions are written to a backup read quorum and a backup write quorum. The coordinator of the inheritance must therefore ensure that deletions are read from backup quorums for all inherited objects. A site that has recovered from a failure can be counted in a backup quorum only if deletions are written to stable storage when objects change views and if the recovered site has read the record of any deletions from its stable storage. Any objects for which the backup quorum requirement cannot be met should be deleted from the new view in phase two of the view formation protocol.

4 Effects of Communication-based Recovery on Logical Clocks

Logical clocks have been proposed as a way of imposing a total order on the events taking place in a distributed system [18]. Logical time has been used in replicated database research to enforce serializability while increasing concurrency and availability. For example, in [15], event-based representation of objects allows write events to be serialized in order of their commit timestamps. For certain data types, such as a FIFO queue, the commit timestamp ordering permits greater concurrency and availability than the more conventional use of version numbers. In [13], event-based representation is used to permit blind writes, thus making an object available for writing even when a read quorum that would be required to determine the correct version number is unavailable. The event propagation protocol described in [13] is also based on logical time.

An implicit assumption in the use of logical clocks in a distributed system subject to failures is that the logical clock value at any site is monotonically increasing, even during failures. For this assumption to hold, the clock value would have to be forced to stable storage following every increment operation, an undesirable proposition for performance reasons. An alternative would be to record the logical time of any write event in a log and to recover the logical clock value from the log. For a database application that uses write-ahead logging, this alternative would not impose any additional I/O overhead. Even if the site's logical clock had been incremented past the maximum time found in the log, any events with greater times would have been lost anyway when the failure occurred. With communication-based recovery, setting the logical clock value to be greater than that of any site from which a recovery message is received would also seem to satisfy the assumption. Subtle problems with one-copy serializability based on commit-timestamp ordering may arise with both methods of recovery, however, as shown by Example 3 below.

Example 3.

> Suppose object X resides at site S1 and object Y at site S2. The logical clock values at S1 and S2 are 7 and 10, respectively, when $T_1 : r[x]w[y]$ executes at S2. Suppose site S1 fails following the commitment of T_1 and that write-ahead logging is being used. Then site S1 recovers the logical clock value 7 from its log and executes $T_2 : w[x]$ with commit timestamp 7.

Clearly T_2 must serialize after T_1, and the commit timestamp ordering of T_1 and T_2 is wrong. This example could be fixed by requiring read events to be logged to stable storage, but this requirement would oppose our goal of moving away from reliance on stable storage.

If we can assume i) that network partitioning does not occur, ii) that a site recovers its logical clock value by contacting all other operational sites, and iii) that not all the sites in a write quorum will fail, then the only transactions whose commit timestamps will be inconsistent with the serialization order will be read-only transactions, and for such a transaction an incorrect commit timestamp may not matter. If any part of the above assumption does not hold, however, the commit timestamps of two transactions that involve writes can be out of order. Example 3 illustrates how this can happen if part ii) does not hold. Adding a third site S3 from which site S1 recovers a clock value of less than 10 and having site S2 fail as well as site S1 results in the same incorrect ordering of commit timestamps when part iii) does not hold. Having site S2 be partitioned from sites

S1 and S3, rather than failing, illustrates how the misordering can result when i) does not hold.

This example proscribes the use of commit timestamp ordering for a database system that uses a static quorum method and relies on communication-based recovery. In such a system, it will be undesirable to log and recover all changes to the logical clock value by using stable storage. The inconsistency of commit timestamp ordering is serious, since the commit timestamps are relied on to enforce serializability. The problem can be solved, however, in a system that uses views to synchronize failures and recoveries of sites. If the view id is prefixed to the commit timestamp, then the conditions of Theorem 1 ensure that the commit timestamp ordering will be consistent with the serialization order.

5 Related Work

Ideas similar to ours have been used in ISIS [7]. The ISIS system provides resilient objects by replicating the state of an object in a number of components. A failed component can recover by copying the current state from an operational component. The ISIS communication primitives synchronize recovery with transaction processing. A read-any/write-all-available strategy is used for accessing replicated data. Site and process failures are tolerated, but the network is assumed not to be subject to partitioning. Recovery from total failure is implemented by restarting in the initial state and replaying the log from the beginning. No checkpointing or shadowing is done.

A regeneration approach for handling recovery in a replicated database that uses distributed shared memory is described in [16]. The recovery protocol relies on communication to regenerate "shares" of an object that are lost because of a site failure. The method does not handle network partitioning.

Memory-resident databases. Communication-based recovery is especially relevant to distributed memory-resident databases, since a remote site will not need to access its local disk to satisfy a request from another site. The recovery problem for a single-site memory-resident database system is discussed in [19]. A solution is given that relies on the use of a stable reliable buffer estimated to be on the order of tens of megabytes in size, much smaller than the main memory, and two to four times slower than main memory. The solution also makes use of two independent processors, a main CPU that performs transaction processing and a recovery CPU that handles writing to and reading from the log. The recovery CPU reorganizes log records in the stable memory before writing them to disk so that the memory copy of the database can be recovered on a demand basis for each partition one at a time. A partition is the basic unit of storage and of transfer to disk. The argument for partition-level recovery from the log, namely that it allows transactions to run as quickly as possible following a crash, applies equally well to communication-based recovery in a replicated main-memory database. With communication-based recovery, however, the delay of transaction processing could be even less, since instead of applying the log records to the checkpoint image of a partition, the system need only copy the partition from the most recent version as indicated by a read quorum. In addition, communication-based recovery permits recovery at the partition level of granularity without the requirement that log records be sorted and chained by partition numbers before being written to disk.

A design for the simple and efficient implementation of a small database that would be useful in a distributed system, for example for a name server, is described in [8]. Recovery from a crash consists of restoring the database from an old checkpoint and then replaying the log. The database is maintained as a strongly typed data structure in virtual memory and is replicated on multiple sites across the network. Recovery from a "hard" failure (when the local disk structures become unreadable) is carried out by restoring the data from another replica. Checkpointing the database to disk is expensive and takes about one minute, since it involves converting the strongly typed structure into a disk representation (55 sec) and a number of Unix "fsync" calls (5 sec total). The authors argue that if checkpoints are too rare, then the log file may consume excessive disk space and the restart time will be too long. If communication-based recovery were used to recover from crashes, however, the likelihood of reading the log file would be low. Less frequent checkpointing would not cause a long restart time unless total failure occurred. A long log file could be archived to reduce the amount of disk space used. The figures quoted for restart are 20 seconds to read the checkpoint plus 20 msec per log entry. Since the figure for a remote enquiry is 13 msecs (including the round-trip network communication costs), recovering the database from remote replicas incrementally either on demand or at low priority would seem to offer the potential for better recovery performance without sacrificing the simplicity of the design.

Optimizing disk writes. A log-structured file system, in which the file system's only representation on disk is in the form of an append-only log, is proposed in [20]. Such a file system is projected to achieve a 1000-fold improvement in I/O performance when combined with high-bandwidth disk arrays and large main- memory file caches. Although the log-structured file system is expected to outperform other file systems for writes, its performance for reading large files that are written piece-wise (for example, a database log to which log records are gradually appended) would be considerably worse, since many seeks will be required to read the file sequentially. A floating map structure scheme that allows random access is projected to provide read performance for small files or for large files written all at once that is at least as good as today's file systems. If communication-based recovery were used for a replicated database, the improvement in disk write performance could be taken advantage of while incurring the penalty of expensive reading of the log file only in the rare event of a total failure.

6 Conclusions

This research has explored the effects on quorum-based methods of moving away from reliance on stable storage toward reliance on communication for recovery from failures. Use of communication for recovery is a promising approach for the future, since network communication speeds and bandwidth are projected to show vast improvement in the near future. Other research [20] has pointed out the possibility that I/O will become the main bottleneck in future computer systems. Techniques such as ours will be useful for avoiding this bottleneck in database systems.

Looking carefully at the combination of communication-based recovery with replicated database algorithms shows that these algorithms typically make assumptions, sometimes implicitly, about the use of stable storage. These assumptions must be compensated for when communication is used instead. Our research has concentrated on quorum-based algorithms and shows the subtleties that can arise in attempting to adapt these algorithms to use communication-based recovery. We have shown that participation in quorums may need to be restricted following a failure. We have also shown that there may be a tradeoff between different techniques for improving the efficiency of recovery. In particular, our adaptability techniques [4] may require more information to be written to stable storage than nonadaptable dynamic quorum methods. For example, changes to active quorum assignments and deletions of object from views may need to be written to stable storage if adaptability is combined with communication-based recovery. As read access to stable storage will be used only as a last resort, however, writing to disk can be optimized at the cost of more expensive read access.

Although our research has concentrated on quorum-based algorithms, similar arguments are applicable to other replicated database algorithms. Experimental research is needed to determine if modifying these algorithms to rely first and foremost on communication for recovery will result in better overall performance and help adapt the algorithms to future technological advances. An adaptable database system should have the flexibility to use either centralized disk-based recovery or distributed communication-based recovery, depending on the relative speeds of each and the existing network conditions.

References

[1] A. E. Abbadi and S. Toueg. Availability in partitioned replicated databases. In *Proc. Fifth ACM SIGACT-SIGMOD Symp. on Principles of Database Systems*, pages 240–251, March 1986.

[2] P. Bernstein and N. Goodman. An algorithm for concurrency control and recovery in replicated distributed databases. *ACM Trans. Database Syst.*, 9(4):596–615, Dec. 1984.

[3] P. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[4] B. Bhargava and S. Browne. *Adaptability to Failures Using Dynamic Quorum Assignments*. Technical Report CSD-TR-886, Purdue University, June 1989. (Submitted to 6th IEEE International Conference on Data Engineering).

[5] B. Bhargava and S. Browne. Using replication to implement reliable storage. In *Proc. 26th Annual Allerton Conf. on Communication, Control, and Computing*, pages 368–377, Monticello, Illinois, Sep. 1988.

[6] B. Bhargava, P. Noll, and D. Sabo. An experimental analysis of replicated copy control during site failure and recovery. In *Proc. 4th IEEE International Conference on Data Engineering*, pages 82–91, Los Angeles, Feb. 1988.

[7] K. P. Birman. Replication and fault-tolerance in the ISIS system. In *Proc. 10th ACM SIGOPS Symp. on Operating Systems Principles*, pages 79–86, Dec. 1985.

[8] A. D. Birrell, M. B. Jones, and E. P. Wobber. *A simple and efficient implementation for small databases*. Technical Report 24, Digital Systems Research Center, Palo Alto, CA, Jan. 1988.

[9] S. Cooper. FDDI and the next generation of LANs. *UNIX Review*, 7(2):48–61, Feb. 1989.

[10] G. S. Delp and D. J. Farber. *Memnet: an experiment in high speed memory mapped local network interfaces*. Technical Report Udel-EE 85-11-1R, University of Delaware Department of Electrical Engineering, 1986.

[11] D. Farber and G. Delp. All systems in sync. *UNIX Review*, 7(2):72–77, Feb. 1989.

[12] D. K. Gifford. Weighted voting for replicated data. In *Proc. Seventh Symposium on Operating Systems Principles*, pages 150–162, ACM, Dec. 1979.

[13] A. A. Heddaya. *Managing Event-based Replication for Abstract Data Types in Distributed Systems*. PhD thesis, Harvard University, Oct. 1988. TR-20-88.

[14] M. Herlihy. Dynamic quorum adjustment for partitioned data. *ACM Trans. Database Syst.*, 12(2):170–194, June 1987.

[15] M. Herlihy. A quorum-consensus replication method for abstract data types. *ACM Trans. Comput. Syst.*, 4(1):32–53, Feb. 1986.

[16] M. Hsu and V. Tam. *Managing Databases in Distributed Virtual Memory.* Technical Report TR-07-88, Harvard University Center for Research in Computing Technology, March 1988.

[17] S. Jajodia and D. Mutchler. Integrating static and dynamic voting protocols to enhance file availability. In *Proc. 4th IEEE International Conference on Data Engineering*, pages 144–153, Feb. 1988.

[18] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, July 1978.

[19] T. J. Lehman and M. J. Carey. A recovery algorithm for a high-performance memory-resident database system. In *Proc. ACM SIGMOD '87 Annual Conference*, pages 104–117, San Francisco, May 1987.

[20] J. Ousterhout and F. Douglis. Beating the I/O bottleneck: a case for log-structured file systems. In *Proc. 12th ACM Symp. on Operating Systems Principles*, pages 11–28, Oct. 1988.

[21] J. Paris and D. Long. Efficient dynamic voting algorithms. In *Proc. 4th IEEE International Conference on Data Engineering*, pages 268–275, Feb. 1988.

[22] D. Skeen. Nonblocking commit protocols. In *Proceedings of the 1981 ACM-SIGMOD International Conference on Management of Data*, pages 133–142, Ann Arbor, Michigan, 1981.

[23] K. Vidyasankar and T. Minoura. An optimistic resiliency control scheme for distributed database systems. In *Proc. Second International Workshop on Distributed Algorithms*, University of Utrecht, The Netherlands, July 1987.