The University of California at Santa Barbara
Technical Report TRCS99-11

The Performance of Database Replication using Atomic Broadcast Group Communication

JoAnne Holliday Divyakant Agrawal Amr El Abbadi

Abstract

Database replication with update-anywhere capability while maintaining global synchronization and isolation has long been thought impractical. Protocols have been proposed for distributed replicated databases that take advantage of atomic broadcast systems to simplify message passing and conflict resolution in hopes of making replication efficient. This paper presents performance measurements on a simulation of a distributed replicated database using those protocols. The results show that with the proper group broadcast mechanism, replication with update-anywhere capability is indeed practical.

1 Introduction

As Jim Gray et al., point out in "The Dangers of Replication and a Solution", [12] database replication with write-anywhere capability, full serializability and isolation is considered impractical. The analysis in that paper follows the traditional argument that deadlocks will rise as the third power of the number of nodes in the network. Also, since writing to all replicas is part of the transaction, the number of database operations (the transaction size) grows with the degree of replication and the probability of aborts and deadlocks grows as the fifth power of the transaction size [12]. This has lead to a great deal of research into primary-copy schemes that restrict where updates can occur and schemes which relax consistency, atomicity and isolation requirements. In order to reduce the cost of writes, the database community has moved from looking at the traditional read-one, write-all update protocols using Two Phase Locking (2PL) or Global Time-Stamp, to variations which use quorum-based protocols [12], lazy update [8] and epidemic [4] algorithms. Quorum protocols have the advantage of reducing the cost of writes, however, they add to the cost of reads. A study by Amir and Wool [7] shows that under modified assumptions appropriate to an Internet environment, dynamic quorum systems outperform static systems. Lazy update/epidemic algorithms reduce write cost by doing writes asynchronously, however, they are optimistic, so there is the danger that conflict resolution will require the abort of a previously committed transaction. The Bayou Project of Xerox PARC [18] has extensively explored a type of lazy update called *anti-entropy*.

Another part of the distributed systems community, being interested in fault-tolerance in distributed systems, has developed systems which provide group communication primitives to support fault-tolerant applications. Systems such as ISIS [10, 3], Amoeba [2], and Totem [6, 1], provide reliable message broadcast with delivery guarantees, atomicity and ordering properties. The expectation is that these systems will solve the replication problem. An application would issue a single command to the group communication system to send messages to all participating sites, thus updating all replicas at once. This would indeed reduce update costs, however, the sample applications presented consider transactions consisting of a single object update. Real transactions require reliability, atomicity, and ordering on a group of operations, not just a single operation. This mismatch is starting to receive some attention [20], and efforts for reconciliation have just begun [16, 5].

Two main benefits are claimed for group broadcast as compared to point to point communication. First, that the application design will be simpler. Second, the performance of the replicated system will be improved. Update protocols have been proposed for distributed replicated databases that take advantage of group broadcast systems [5]. We found the design of the database and update protocols was not noticeably simpler. In fact, the replicated system must still have a concurrency control protocol, and dealing with deadlock is not straightforward. The design process also showed us that the broadcast systems had many features we did not need which we thought would hurt performance. We were therefore skeptical about the performance improvements.

In this paper we explore the benefits of atomic broadcasts for the management of transaction based replicated databases. The update protocols use the atomic broadcast systems to simplify message passing and conflict resolution in hopes of making replication efficient. For concreteness we investigate a particularly important application of replication, namely a highly available server in an Internet environment. A heavily loaded database server, with requests continuously received from the Internet, may be implemented as a replicated database on a small number of servers (up to nine) on a local area network. In order to determine whether broadcast based protocols can improve performance for replicated databases, we built a detailed simulation of a distributed replicated database. We then conducted a series of experiments to determine the tradeoffs among the different protocols and to explore advantages of replication in a broadcast based system. The results show that, with the proper group broadcast mechanism, replication with update-anywhere capability is indeed viable.

Section 2 defines our system model and Section 3 discusses the specific replicated data update protocols used. Section 4 describes our simulation program. In Section 5 we present the results of our experiments. The paper concludes with a discussion in Section 6.

2 System and Communication Model

In this paper we investigate the design of a highly available database server. Such a server is particularly relevant in the context of the Internet, where users may submit various requests (queries or updates) to a server. If the server is implemented as a single database, it may easily become congested and be a bottleneck. We therefore consider replicating several copies of the database (or perhaps only the hot-spot pages) on multiple sites connected by a local area network. We refer to them as *server copies.*

Each server copy could be connected directly to the Internet or there could be a controller which accepts the Internet requests and forwards them to one of the server copies (see Figure 1). The former approach would avoid the potential bottleneck of the controller. The latter approach could ensure a balanced load among the replicas. In this paper, we allow both system configurations.

A replicated database is a collection of data items that are replicated; that is, they are stored redundantly at multiple sites. We assume that the database is fully replicated, i.e., every server copy stores a copy of all objects in the databases. Users interact with the database by invoking *transactions* at any one of the server copies. A transaction is a sequence of read and write operations

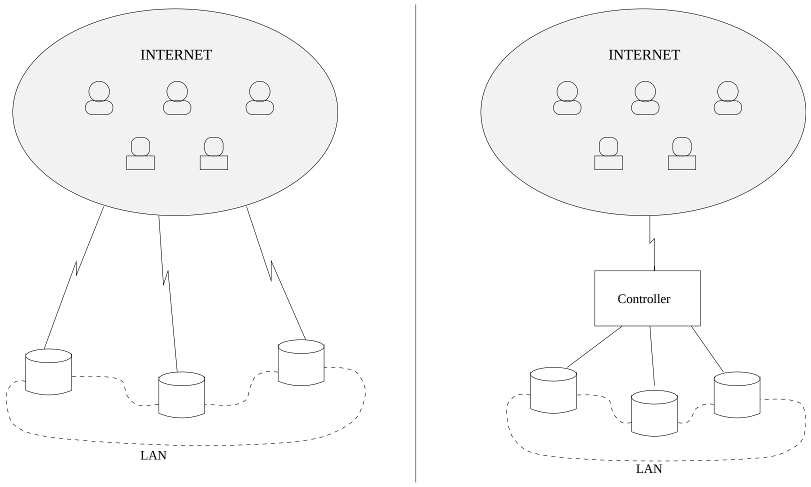


Figure 1: Two Possible Configurations for a Three Site Database System

on the data items that is executed atomically, i.e., a transaction either *commits* or *aborts* the results of all its operations [13]. For simplicity, we assume that all read operations are executed before all write operations. The criterion for correctness in databases is the *serializable* execution of transactions. Serializable executions are guaranteed by employing a *concurrency control* mechanism such as locking, timestamp ordering or optimistic concurrency control. Since strict two-phase locking[1] is widely used, we assume that concurrency control is locally enforced by strict two phase locking at all server copy sites.

In a replicated database, a transaction can be initiated at any one of the database copies and the multiple physical copies of the data items must appear to the transaction as a single logical copy. The results of the transaction operations must appear to the user as if there is only one copy of the database. That is, we want *one-copy serializability* [9]. Thus, the goal of replica control protocols is to find update protocols to keep the copies consistent and up-to-date while maintaining good performance.

We assume that the server copies are fail-stop and the communication subsystem among them provides primitives to broadcast messages to a group of sites. These group multicast services ensure reliable message broadcasting with total ordering properties. In order to achieve the ordered delivery of messages, messages that are received at a site are delayed and not delivered to the application until they can be delivered in the proper order. In addition to ordering and reliability, group multicast systems keep track of group membership and report site failures to ensure virtual synchrony.

In this paper we consider atomic broadcast with the following properties [14]:

1. If a correct (non-failed) site broadcasts a message m, the primitive ensures that the message will be delivered to all operational sites. Furthermore, if a site delivers a message m, then all operational sites deliver m. This is the "all-or-nothing" property.

2. A message is delivered at most once, and only if it was actually broadcast.

[1]In strict two-phase locking, transactions hold all locks until termination.

3. If sites p and q deliver broadcast messages m and m', then m and m' are delivered in the same order at all sites.

Several atomic broadcast systems have been proposed. We now discuss two such systems: Totem and Amoeba. Totem [1] was developed at UCSB and provides totally ordered multicasts to process groups with low, predictable latency. It uses a token circulating on a logical ring. A site can broadcast messages only when it has the token, giving the messages a total order. Performance remains good at heavy loads because the need to wait for the token is offset by the lack of contention on the Ethernet. Totem offers *agreed* delivery and *safe* delivery. Both options are totally ordered; however, safe delivery waits until all sites have received the message before any site can deliver the message to its application. Our protocols require only agreed delivery, however, the larger delay of safe delivery was used in our model to give realistic results under less-than-ideal conditions. Performance measurements were by R. Budhia [11] using Sun SparcStations on a 100 Mbit Ethernet with no special hardware.

Amoeba [2] is a group communication service developed at Vrije Universiteit. It has been in use in academia, industry, and government for over five years. It runs on the SPARC (Sun4c and Sun4m), the 386/486, 68030, and Sun 3/50 and Sun 3/60. Amoeba offers a single level of multicast which is totally ordered. To achieve this, one site is chosen as the sequencer and the sequencer determines message order. There are two variations on the Amoeba protocol, PB and BB. In PB, the sender sends its message to the sequencer, which then broadcasts it to all sites with a delivery order number. The BB variation avoids having the possibly long message on the network twice by having the sender broadcast it to all sites and then the sequencer broadcasts a short "accept" message with the ordering information so that the sites can deliver the message. We have modeled the BB protocol in our simulation. The measurements used in our model come from a study by Kaashoek and Tanenbaum [15] using 20 MHz MC68030 processors on a 10 Mbit Ethernet. Message latency is determined primarily by message size and only slightly influenced by the number of sites in the system.

3 Atomic Broadcast Based Update Protocols

We now describe a set of replicated data update protocols which use atomic broadcast primitives as their means for communication [5] with the other replicas. The idea underlying each of these protocols is that since the communication subsystem supports totally ordered atomic broadcasts, operations can be processed one at a time at every site in the same order. These protocols tolerate failures because of the masking of such failures by the broadcast mechanism. The details of the protocols and a proof that they enforce *one-copy serializability* appears in [5].

3.1 Protocol A1: Broadcast All

This is the simplest protocol for maintaining a replicated database and is presented primarily as a benchmark to compare with the other protocols. This protocol requires that a transaction initiated at a server copy site broadcasts all its operations (read or write) to all other sites using the atomic broadcast.

The initiating site waits until the operation has been delivered and executed locally before beginning the next operation. After executing all of the operations for a transaction, the site issues a commit or an abort operation which is broadcast to all sites. As a result the transaction is committed or aborted at every site when the operation is delivered and processed at each site.

Since lock requests can be delayed, transactions may deadlock. Since every site performs the same operations in the same order, deadlock resolution can be localized to each site. However, it is important for the correctness of this protocol (and the next one) that every site detects and resolves the deadlock the same way. In particular, if deterministic deadlock detection techniques are used at every site to resolve the deadlock, we are guaranteed that the same transactions will be selected for deadlock resolution.

3.2 Protocol A2: Broadcast Writes

An obvious drawback with the previous protocol is that read operations are performed globally (i.e., broadcasting read lock requests to all database sites), which is an overkill. In the database context, it is generally expected that read operations are significantly more frequent than write operations. Hence, localizing the execution of read operations is desirable and should result in significant performance improvements.

In this protocol, we execute read operations locally and broadcast only write operations. A read operation is executed locally after obtaining a read lock, and a write operation is broadcast to all copies of the database. When the lock manager at site S receives a write request from T_i, it checks if the lock can be granted. If granting is successful, the operation is executed. If the lock cannot be granted, it is because the requested data page, x, is either locked by readers or a writer. If x is locked by a writer, the write operation is blocked at S and any potential deadlock is resolved using the appropriate deadlock resolution mechanism. Since write operations are performed globally, a cycle involving write operations will be detected at every site in the system and resolved consistently.

If on the other hand, the lock is held by some readers, for every transaction T_j that holds a read lock on x, the lock manager checks whether T_j has already broadcast a commit. (Note that T_j was initiated at S.) If so, the lock manager blocks the write request until it receives T_j's commit. If S has not sent T_j's commit, T_j is aborted and T_i is granted the write lock. If T_j is not read-only, T_j's abort is broadcasted to release all locks held by T_j at other sites. There can never be a deadlock cycle involving read operations.

Finally, T_i terminates by broadcasting a commit operation. Since the atomic broadcast has the "all-or-nothing" property, there is no need to employ an atomic commitment protocol.

3.3 Protocol A3: Delayed Broadcast Writes

This protocol attempts to completely localize transaction execution. This is achieved by deferring update operations until commit time, when a single message with all updates is sent to all other sites. A transaction, T_i, executes a read operation locally while a write operation is deferred until T_i is ready to commit. To terminate, T_i broadcasts its deferred writes to all sites. On receiving the writes, the lock manager on site S grants all write locks to T_i atomically, and then the writes are executed at S. After all the writes of T_i are executed locally, T_i broadcasts its commit operation to all sites. T_i terminates after the delivery and execution of its commit locally.

This protocol is also based on reading one copy and writing all copies of replicated objects. It has the added benefit that deadlocks are not possible. This is because all the write locks for a transaction are obtained in a single atomic step at the local lock manager. Conflicts with read locks are dealt with by aborting read operations and the transactions they belong to. There is thus no need to check for deadlocks.

3.4 Protocol A4: Single Broadcast Transactions

Since all write operations are known to all sites and they will be eventually executed, the question arises if the extra broadcast of the commit operation is superfluous. To use a single broadcast, this protocol maintains a version number with each page in the database. A transaction T_i executes a read operation locally and a write operation is deferred until T_i is ready to commit. When the site that initiated T_i is ready to commit the transaction, if T_i is a read-only transaction, the decision to commit is done locally and no message is broadcast. Otherwise the site broadcasts the set of reads with their version numbers and the set of writes. On receiving the set of reads and writes of T_i, the lock manager at a site S first checks if the version of the items read by T_i are obsolete (that is if any of the versions on S are greater than the versions read by T_i). If so, T_i is aborted. Otherwise, S proceeds with the attempt to atomically grant all the write locks. If a write lock cannot be granted due to a conflict with a read operation, the reading transaction is aborted and T_i receives the lock. Once all write locks are obtained, S executes the write operations and increments the version numbers of each data-item. T_i terminates at S as soon as it atomically obtains the write locks and successfully executes its write operations.

3.5 To Broadcast or Not To Broadcast?

The above protocols use atomic broadcasting to manage replicated databases. Atomic broadcast tools can be quite expensive in terms of message passing overhead as compared to point-to-point communication. Due to the order guarantees, multiple rounds of message passing are needed as in Totem and Amoeba. However, the replicated data protocols A2, A3, and A4 exploit these total order guarantees to attain several advantages.

If point-to-point communication primitives are used, operations at remote sites must be acknowledged before transaction execution can proceed. When atomic broadcasts with delivery guarantees are used, acknowledgments for operation requests are redundant. Since the state in which an operation is executed is identical at all sites, the response to all operations is the same. Elimination of explicit acknowledgments at the application level reduces the communication cost of executing transactions. One of the main problems encountered when two-phase locking is used in a distributed environment is the need for distributed deadlock detection. In these protocols, global deadlock is impossible. Note also that atomic broadcast eliminates the possibility of deadlock on a single object, which is one of the main concerns when replication is implemented in a point-to-point communication system. That is, if two transactions attempt to lock multiple copies of the same page at different sites then depending on the order in which their requests arrive at different sites, two (or more) transactions may be involved in a deadlock. When atomic broadcasts are used, concurrent and competing requests for the same page will be delivered to and processed by every database site in the same order; thus, transactions could not be involved in single object deadlocks. Even local multiple-object deadlocks cannot occur in the protocols A3 and A4. Finally, a two-phase commit protocol to ensure atomic commitment is not needed since all sites will uniformly make the same decisions.

The main question this paper addresses is whether these advantages outweigh the overhead involved in implementing atomic broadcast. We address this question by designing a detailed

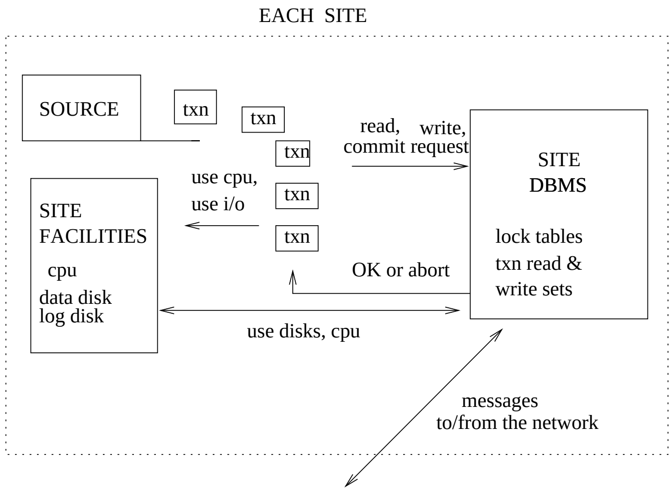


Figure 2: Model Structure of a Single Site

simulation to study the performance of replication using broadcasts.

4 The Simulation Model

We used a detailed model of a distributed replicated database for our experimental measurements. Previous studies [17] have shown that neglecting to model resources can greatly affect the conclusions. Therefore, in our model the transactions and site databases must contend for the CPU, data disk, log disk, and network services. The simulation studies in [16, 17] formed the basis for the design of our simulation. The structure of the model for each site is shown in Figure 2.

Each site in the database model has a *Source* which generates transactions. This corresponds to the transactional requests arriving from the Internet. The transaction interarrival time determines the load on the system. As the interarrival time is made shorter, the load increases (there is no multiprogramming level to limit the number of active transactions vying for data pages and resources). This is the "open" queuing model, and is appropriate for a system accepting requests from users on the Internet. This model would also be appropriate for a database application that used replication for fault-tolerance and increased availability.

Transactions are modeled as sequences of read and write operations, where all the reads are done before the writes. The write set is a subset of the read set, so there are no "blind writes". A transaction makes a read or write request to the site DBMS. After the operation is done, the transaction "thinks" for a while before presenting the next operation. This think time could represent the time needed to transmit the results to the user, analyze the results and send the next operation.

The *Site DBMS* has a collection of 1000 data pages which are replicated at each site. These 1000 data pages could be a subset of a larger database that is replicated for high throughput. By making the number of pages small, we will be able to study data contention issues more easily. A page is assumed to be 2K and is the locking granularity as well as the data disk granularity. The DBMS uses Strict Two-Phase Locking to ensure local serializability. The DBMS maintains the lock

Parameter Name	*Meaning*	Value
InterTime	Transaction interarrival time	varies
NumSites	Number of sites	varies
NumDisks	Number of data disks per site	1
MinDiskTime	Smallest data disk access time	4 ms
MaxDiskTime	Maximum data disk access time	14 ms
CPUInitDisk	CPU time needed to access disk	0.4 ms
LogDiskTime	Time needed for forced write of log	10 ms
LogPageSize	Number of log records per page	100
HitRate	Probability of cache hit	0.8
LockTime	CPU time needed to handle locks for read or write request	0.006 ms

Table 1: System Parameters

tables and a table of active transactions with the locks they currently hold. The site DBMS uses the network when the update protocol calls for the broadcast or receipt of a message. Each site DBMS has a deadlock detection routine for use with protocols that can suffer from local deadlocks, i.e., A1 and A2. A Wait-For-Graph is maintained and is checked for cycles each time an edge is added. The deadlocked transaction with the highest transaction ID is chosen as the victim. Victims are aborted and restarted after a delay.

The Site DBMS lock manager was carefully designed so that we could guarantee that each site grants locks to operations in the order in which they are broadcast to it. Thus, the operations will appear to be performed in the same order at all sites. Traditionally, there are no constraints on the lock manager as to the order in which it services operations from its input queue. When two-phase locking is used, operations arrive at the lock manager for consideration. If the lock is granted, the lock manager forwards the associated I/O (read/write) request to the data-manager. As soon as the operation is forwarded, the lock manager considers the next operation in its input queue. If the lock manager is multi-threaded, the order in which locks are granted may not be the same as the order in which requests are delivered by the broadcast service. Since the proposed protocols rely on the broadcast delivery order of operation, our lock manager has to be single-threaded. Also, when several lock requests are pending for a page, the choice of which request is considered next after the release of a lock on that page must be the same at all lock managers. This guarantees that all lock managers grant locks in the same order at every site and hence ensure the same synchronization decision at every site.

The system parameters of the model are given in Table 1 along with their values. The parameters governing the generation and behavior of transactions are given in Table 2. Transaction *response time* measured in milliseconds is the most important measurement. If an individual transaction had to be aborted, the restart delay, as well as the first partial and second total execution time (assuming only one restart was necessary), is included in the response time. Thus, the true cost of aborts is reported in the response time measurements. System *throughput*, measured in transactions committed per second, is also important in comparing results with different numbers of sites in the distributed system.

The multicast systems Totem and Amoeba are modeled with four parameters as shown in Table 3. Each broadcast and receive operation uses a certain amount of the CPU facility, *CPUMsgSend*

Parameter Name	*Meaning*	Value
ReadSetMin	Smallest transaction read set size	6
ReadSetMax	Maximum read set size	10
ROPercent	Percentage of read only transactions	75%
WriteSetMin	Smallest write set size for update transactions	1
WriteSetMax	Maximum write set size	4
CPUPageTime	CPU time spent on a data page	1.0 ms
IntThinkTime	Time between sucessive operation requests	10 ms

Table 2: Transaction Parameters

Parameter Name	*Meaning*	Totem Value	Amoeba Value
CPUMsgSend	CPU time needed to send a message	0.7ms	1.0+0.3*MsgSize
CPUMsgRcv	CPU time needed to receive a message	0.5ms	0.8+0.5*MsgSize
SendDelay	Message latency not caused by CPU usage	0.48*NumSites	1.1+1.6*MsgSize+0.004*NumSites
NetFacUse	Amount of LAN used	0.2*MsgSize	0.9

Table 3: Broadcast Parameters

and *CPUMsgRcv*. Before a message is distributed to the other sites by the network, it is delayed by *SendDelay*, which models waiting for the token in Totem and time required by the sequencer in Amoeba. The message also uses the network facility with *NetFacUse*. The simulation model achieves total order delivery by having a *network* module with a single input queue for messages to be broadcast. The order in which messages are put into the queue determines their eventual delivery order. The *network* module removes a message from its input queue and distributes it to all site input queues before considering the next message.

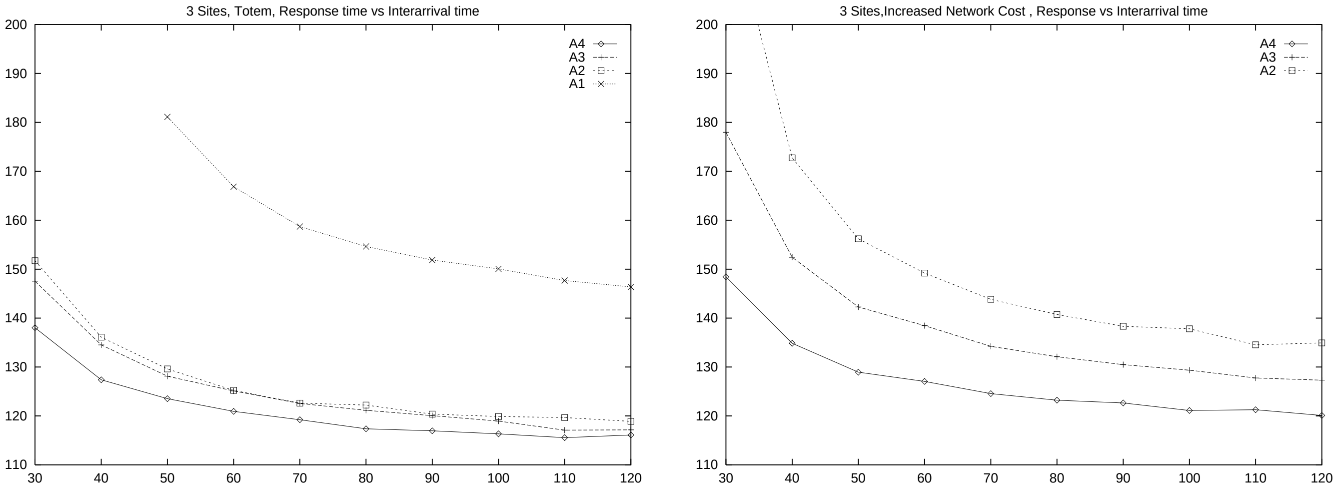
5 Experimental Results

In this section, we present the results of a variety of experiments that were conducted. The overall goals of the experiments were:

1. To evaluate the relative performance of the various replication protocols that use atomic broadcast as a means of communication.

2. To evaluate the relative advantages of different broadcast systems. In this paper, we use two such systems: Totem and Amoeba.

3. To ascertain the advantages of replication as a means for improving overall system performance.



(a) Response time for 3 sites

(b) Response time for 3 sites, Increased Network Cost

Figure 3: Response time vs Interarrival time in milliseconds

All measurements in these experiments were made by running the simulation until a 95% confidence interval was achieved for each data point. This was typically 15,000 to 30,000 transactions. The confidence intervals were typically one to two milliseconds wide.

5.1 Response Time Analysis

The first experiment measured response time in milliseconds as a function of transaction interarrival time, in milliseconds for all four protocols. An interarrival time of 50 means that every 50 milliseconds a new transaction starts at each site. One purpose of the experiment was to verify our intuitive assessment of how the protocols would perform. We modeled systems of three server copy sites on a LAN using Totem and the results are shown in Figure 3(a).

As expected, A1, which broadcasts all operations, had a much greater response time than the other protocols under all loading conditions. Protocol A1 is not included in subsequent experiments. Protocol A4, which requires a single broadcast per update transaction, performed best. We were, however, surprised that there was not a greater separation between A2, which requires a broadcast per write and A3 which only requires two broadcasts per update transaction. The main reason seems to be due to the increased resource contention. In fact, both A3 and A4 pay a penalty in terms of wait time for the disk and CPU that A2 does not have. For example, we considered the mean queue length for the data disk for A2, A3, and A4 at an interarrival time of 70 milliseconds on a three site system. For A2 the mean disk queue length was 0.292. For A3 and A4, it was 0.310 and 0.308 respectively. Recall that protocols A3 and A4 do all write operations at once, rather than one at a time with a *think time* in between as is done in A2. Thus, although all three protocols are making statistically the same requests, the distribution of the timing of those requests makes a difference. Apparently, the natural interleaving of resource use with other processing that A2 does (and that occurs in any multi-programming environment) is an advantage in spite of having to use the network to broadcast every write request. In our simulation, the data manager handles

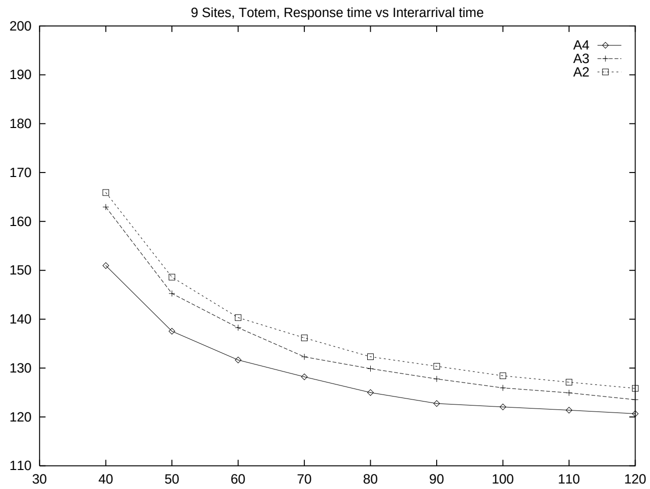


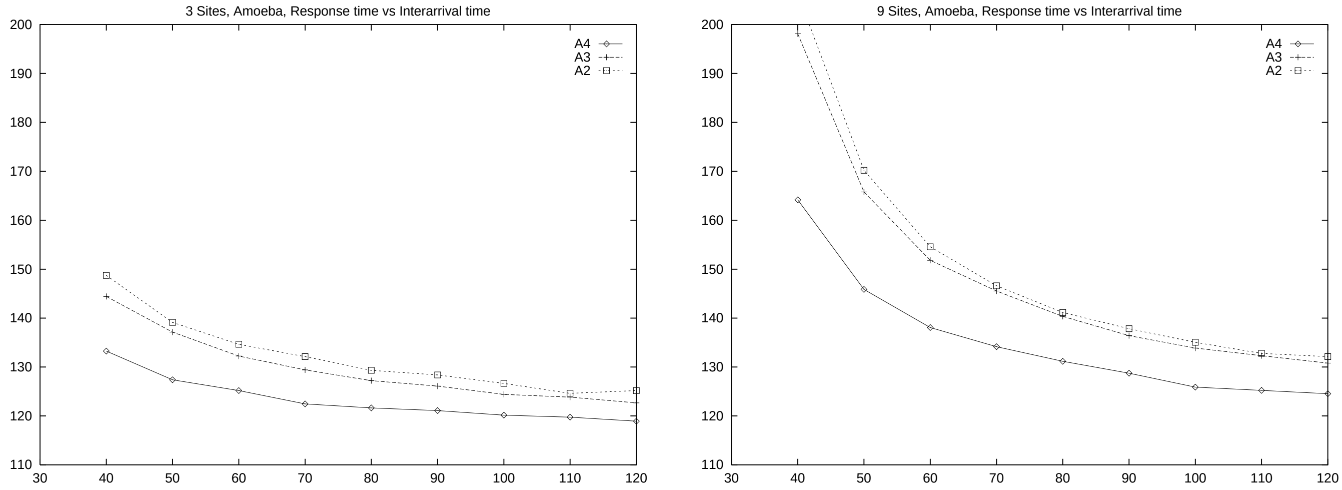
Figure 4: Response time vs Interarrival time for 9 site System

requests in the order they arrive, and does not reorder them for convenience or optimization. There is therefore a tradeoff between the benefit achieved by the data manager being able to rearrange operations and the benefit achieved by imposing a system-wide total order on the operations.

Another important observation is the substantially better response time of A4 when compared to both A3 and A2. This is primarily due to the fact that transactions using protocol A4 never have to wait for a page lock. Blocking readers are aborted and writers must have completed their writes and committed before the current transaction is considered. For A2 and A3 at an interarrival rate of 70 ms, the abort ratios (the ratio of transaction aborts to transaction commits) are similar and the blocking ratios (the ratio of the number of times a transaction was forced to wait for a lock to the number of transaction commits) were 0.008 and 0.004, respectively. For A4 under the same conditions, the abort ratio was slightly higher, but the blocking ratio was zero.

To further demonstrate the communication overhead of the protocols, we increased the message cost of broadcast. In Figure 3(b) we increase the message passing cost by five times. As anticipated, the increased network costs had very little effect on the response times in the A4 protocol, since A4 uses the network only once. The response time curve for A2 showed the most increase, as it uses the network for each write operation. Furthermore, the communication overhead starts to become more important than the disk access times.

Finally, we increased the number of server copy sites from three to nine and studied the relative performance of A2, A3, and A4. The response times (Figure 4) increased for all three protocols in similar proportions. The response times increased only about 10 to 25 milliseconds for all three protocols. The smallness of the increase was unexpected given that the system has tripled in size. We therefore studied the system with increased network delays. For example, the response time for A3 at an interarrival time of 110ms on a three site system is 117.08 milliseconds and on a nine site system it is 124.92ms. We tripled the network costs for nine site system, and got a response time of 134.79ms. We therefore conclude from this set of experiments that, given the relatively low network communication costs, the replicated data management protocols perform quite well.



(a) Three Site System

(b) Nine Site System

Figure 5: Response time vs Interarrival time for Systems Using Amoeba

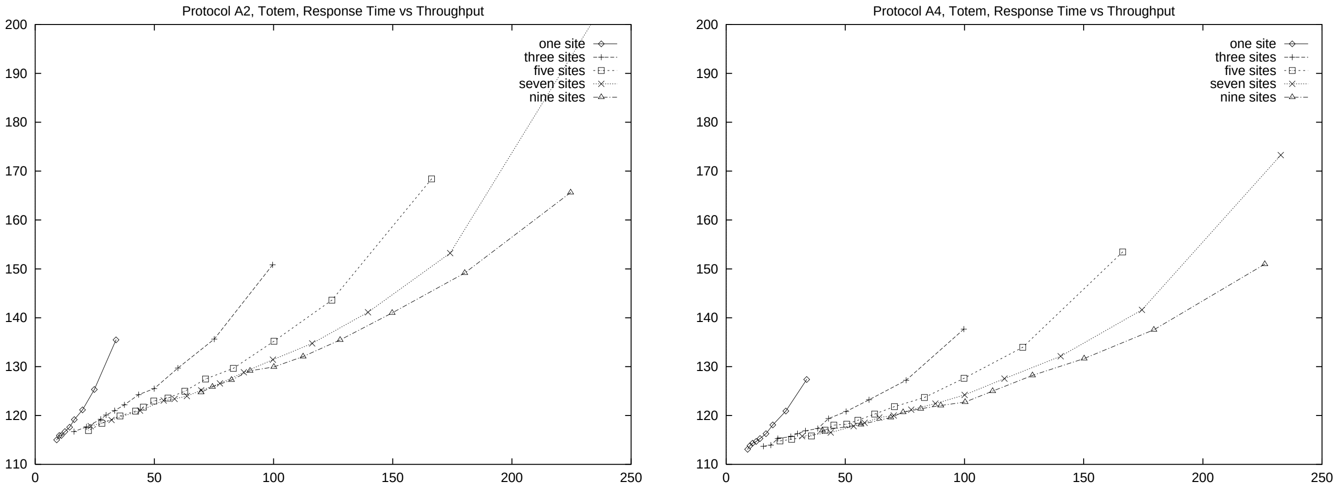
5.2 Experiments with Amoeba

The shape of the response time curve for a three site system using Amoeba (Figure 5(a)) is similar to that for Totem. The actual response times are about 5ms longer. Recall, our experiments for Amoeba are based on a 10Mbit Ethernet. In the nine site system (Figure 5(b)), the performance of protocol A2 and A3 are very similar. This can be explained by noting that the message cost depends heavily on the message size, and as A2 and A3 must broadcast the same amount of information their total message cost is nearly the same.

Since the message cost for Amoeba is largely determined by message length and only to a lesser degree by the number of sites, sending eight 1K messages has about the same overhead as sending a single 8K message. This is in contrast to Totem in which the message cost is most dependent on the number of sites and independent of the message length unless the load is so great that the Ethernet saturates. The Amoeba system copies the message several times and this cost dominates the overall message cost. The dominant factor in Totem message latency is the time spent waiting for the token to arrive. The time for the token to arrive is largely determined by the number of sites in the system. With this in mind, Amoeba would probably perform best in a system where the expected writeset was small and the size of the data that must be broadcast per write is small. When long messages are expected, Totem will probably perform better.

5.3 Varying Degree of Replication for a Given Protocol

In this experiment we investigate the benefits of replication on throughput. To compare the performance of a single protocol on systems with a varying number of sites, response time is shown as a function of throughput in Figure 6. This is because the transaction load on a nine-site system is three times as much as that on a three-site system for the same transaction interarrival time. Throughput is the number of transaction committed by the system per second. The measurements



(a) Protocol A2

(b) Protocol A4

Figure 6: Response time vs Throughput for Protocols A2 and A4

for a one-site system were made by setting the number of sites to one and the message broadcast cost to zero. In this way, the measurements represent the use of a centralized database using ordinary Two-Phase-Locking with no replication.

The experiment results in Figure 6 (a) show the advantages of replication for just the A2 protocol using Totem. A2 is a simple extension to protocols already in use, so these results are especially gratifying. Our first observation is that in a non-replicated system, the maximum throughput that could be reached before the effect of queuing for resources and the multiplicative effect of aborts overloaded the system was 34 transactions per second. If higher throughputs are desired, it seems that replication must be used. As the number of server copies increases, the system can handle larger throughput. For example, neither one- nor three-copy systems could handle a throughput of 150 transactions per second. However, a five-site system handled it with an average response time of 162msec, a seven-site system had response time 145msec, and a nine-site system, 140msec. Finally we note that a nine-site system can handle a throughput of 225 transactions per second with a fairly reasonable response time of 165 msec.

It is interesting to observe the system at low throughput. Figure 7 is a blow up of the response time versus throughput for protocol A2 and shows that even at low throughput, the multi-site systems continue to have better response times. In fact, at the maximum throughput of a single site system of 34 transactions per second with a response time of 135, a three site system has a response time of only 120 and a five site system has a response time of 119. Only at very small throughput (less than eight transactions per second) does the single site system outperform the three- and five-site systems. At this throughput, the messaging costs of the three- and five-site systems outweigh the advantage of additional resources.

The results for A3 were not much different from A2, and are not shown. The results for A4 were similar; however, the response times were consistently better. Response time, as shown in Figure 6 (b), increases with increased system throughput, as before. A nine-site system can handle

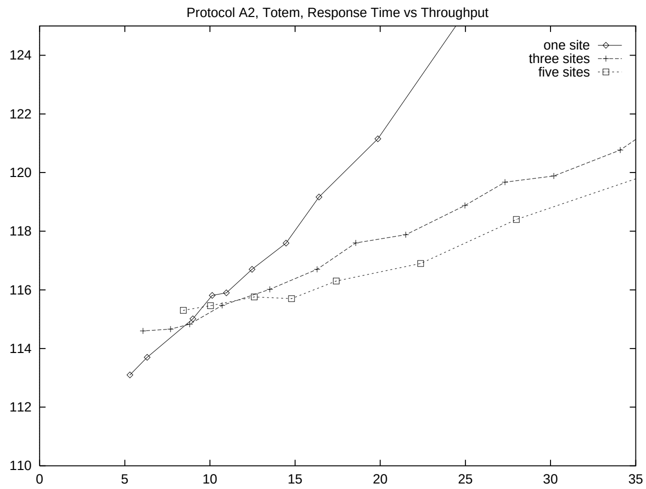


Figure 7: Detail of Response time vs Throughput

150 transactions per second with a response time of 130 ms, whereas with protocol A2 a nine-site system handling 150 transactions per second had a response time of 141 ms.

5.4 Reduced Resource Contention

One of the main challenges for replication is its ability to scale up well as system performance improves. One may question, with reduced resource contention (faster network, faster disk), would the protocols be able to handle the increased data contention resulting from increased load. We must demonstrate that the data contention caused by replication and our protocols will scale up reasonably with increased load. In the previous experiments, resource contention, particularly contention for the data disk, seemed to be the most important factor limiting throughput. How would our protocols perform with the best resources available today? Totem was evaluated on a 100 Mbit Ethernet [11]. One gigabit Ethernets are available today, so that will be reflected in this experiment by decreasing the network usage parameter. Also, the Totem *Agreed Delivery* will be modeled instead of the more expensive *Safe Delivery*. In their study of disk modeling [19], Ruemmler and Wilkes point out that disk drives such as the HP97560 with data caching can complete about 50 percent of requests in 3 milliseconds or less. In a replicated database, it may not be necessary to keep a copy on disk at all sites. Therefore the hit rate was increased and the disk access time shortened from 4 – 14 ms to 2 – 4 ms in this reduced resource contention model.

Table 4 summarizes the parameters changes. The results are shown in Figure 8. The scale on the Y-axis has been changed since the response times are much shorter. It is interesting to note that the difference in performance between the different protocols becomes negligible and all of the protocols perform better. For example with Totem in our previous example of a three site system, protocol A4 had a response time of 117 ms at an interarrival time of 80ms. Now, with reduced resource contention, protocol A4 has a response time of 97 ms at the same transaction interarrival time. Since network costs are negligible and the disk is faster so the queuing effects are less important, the overhead for all protocols becomes comparable.

Parameter Name	*Meaning*	Value
MinDiskTime	Smallest data disk access time	2 ms
MaxDiskTime	Maximum data disk access time	4 ms
CPUInitDisk	CPU time needed to access disk	0.1 ms
LogDiskTime	Time needed for forced write of log	3 ms
HitRate	Probability of cache hit	0.95
NetFacUse	Time spent in the network facility	0.02*MsgSize
SendDelay	Delay before a site can transmit	0.2*NumSites

Table 4: Parameter and Network Changes

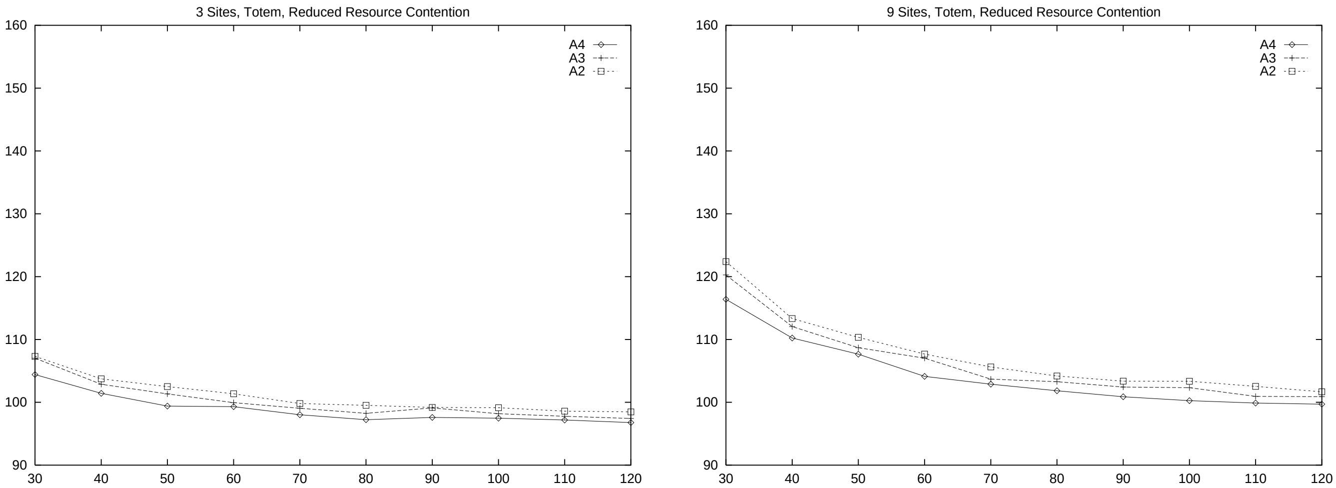


Figure 8: Response time vs Interarrival time for 3 and 9 sites, Reduced Resource Model

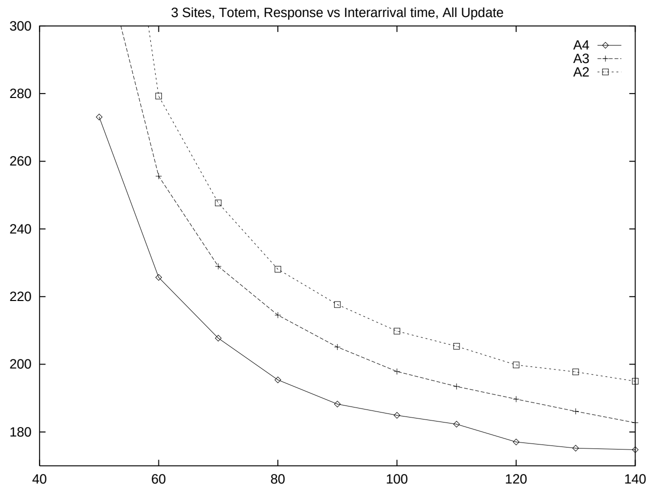


Figure 9: No RO, Response vs Interarrival time, 3 sites

5.5 No Read-Only Transactions

Our final experiment explores the case where the system is under heavy update load. Note that in all previous experiments, most of the transactions (75%) were read-only and only 26% of the update transaction's operations were write operations. This is a reasonable setting for most commercial databases. However, we would like to study the performance of the replicated protocols under stress. We therefore eliminated all read-only transactions and require all transactions to have on average 5 to 6 write operations out of a total of 10 to 12 operations. In Figure 9 we plot the resulting response time for a three site system using Totem. Obviously the average response time per transaction is significantly more than in the read-only dominated system. For example, in a three site system with Totem, protocol A4 had a response time of 117 ms at an interarrival time of 80ms. Now, with all transactions updating, protocol A4 has a response time of 195 ms at the same transaction interarrival time.

Although the relative performance of the protocols is still as expected, we notice a more significant difference in the absolute performance of the three protocols, due to the increased write rate. For example, at the interarrival time of 80 ms, A2 had a response time of 228 ms, A3's response time was 214 ms and A4 was 195 ms. In particular, note that the distinction between A2 and A3 is now substantial since more communication overhead is incurred by A2. A4 performs quite well considering the load on the system. At an interarrival time of 50 ms, A4 gives a response time of 270msec, which rapidly decreases as the interarrival rate increases to 184 msec at an interarrival rate of 100 and 175msec at an interarrival rate of 130.

6 Discussion

The simulation results for broadcast-based replication are quite positive, contradicting our initial reservations regarding the use of broadcast systems to support replicated data. In fact, some of the results seemed so positive that we need to address the question, "Why were the response times so good?" Familiar models of replication predict that data contention and message cost will multiply, causing response times to soar out of control as the number of sites and the load on the system increases.

With a group multicast system to handle messaging, sending a message to all other sites is a single, relatively fast operation. The time required does depend on the number of sites, N; however, the factor is small (240 microseconds) and the dependency is N rather than N^2. Our protocols may not perform well with 100 or 1,000 sites (the multicast systems have much different characteristics at those sizes), but replications up to nine sites appear to give a definite throughput advantage. In our protocols, there can be no global deadlocks. Local deadlocks are possible in A1 and A2, but they are extremely rare since a transaction that wants to write to a data page that is read-locked by another simply aborts the reader rather than waiting.

Broadcast-based replication does not add to the probability of aborts as it does in the point-to-point replication model. In the traditional model, a transaction must avoid abortion at each site in order to survive and commit and each site carries a small but non-zero probability of abortion. In our model, if a transaction survives at one site and is able to commit at that site, it must survive and be able to commit at every other site. Also, a transaction running at a site does not have to wait until its commit request is acknowledged and approved by all other sites. It is enough for the transaction to know that the commit message is assured a place in the total order of message delivery, and the transaction can commit locally. This has a positive effect on the response time.

Another way of looking at our model is by comparing it to the primary or master copy model. In the primary copy model, each data page or data object has a *primary* site and all other copies of the data are secondary. Read operations can be done on any copy, but a write can be performed only by obtaining a lock on the primary copy (the update is then propagated to the secondary copies). The restriction to the primary copy serializes write requests at a single site, thus overcoming the confusion of replication. The primary-copy model has been shown in numerous studies to have acceptable performance. The primary-copy system gives the improved performance because there is a single entity dictating the serialization order and the secondary copies are just copies. Our replicated system also has a single entity dictating the serialization order, except that this entity is not the primary copy but rather the group multicast service which imposes a total order on the operations. This is why we can expect to see performance measurements similar to the primary-copy model.

In conclusion, in spite of common attacks on the practical benefits of replication, we believe that broadcast support holds the promise of actually delivering these benefits. Recently, other researchers have started examining novel requirements for replication. For example, Wool [21] argues that quorum-based replication deserves a new look as new technological advances and different application requirements are brought to light. Similarly, we argue that broadcast-based replication should be revisited, now that the performance of broadcast systems has improved. The protocols we presented show that replication may be the sole avenue for achieving high throughput. The protocols we analyzed present a spectrum of options. Two of these protocols are especially worthy of note. The broadcast writes protocol (A2) provides good overall response time, while requiring only minor modifications to standard transaction execution models. The single broadcast protocol (A4), on the other hand, eliminates operation blocking and thus provides excellent response time performance. Finally, we observe that all the experiments considered less than 10 sites. This is due to restrictions on the broadcast systems, Totem and Amoeba. We are currently exploring wide area network versions of these protocols that would allow increased scalability both in terms of number of sites and the extent of the network.

References

[1] The Totem System [web page] http://alpha.ece.ucsb.edu/totem.html [accessed Nov, 1998].

[2] The Amoeba Distributed Operating System [web page] http://ftp.lip6.fr/pub2/distributed_systems/amoeba/amoeba.html [accessed Nov. 1998]

[3] The ISIS Project [web page] http://simon.cs.cornell.edu/Info/Projects/ISIS/ [accessed Nov. 1998]

[4] D. Agrawal, A. El Abbadi, and R. Steinke, "Epidemic algorithms in Replicated Databases," In *Proc. 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Tucson, Arizona, May, 1997.

[5] D. Agrawal, G. Alonso, A. El Abbadi, I. Stanoi, "Exploiting Atomic Broadcast in Replicated Databases," *EuroPar'97, Proc. 3rd International EuroPar Conf*, Passau, Germany, August 1997.

[6] Y. Amir, L.E. Moser, P.M. Melliar-Smith,D.A. Agarwal, P. Ciarfella, "The Totem Single-Ring Ordering and Membership Protocol," *ACM Transactions on Computer Systems* 13(4), Nov. 1995, p311-342.

[7] Y. Amir, A. Wool, "Evaluating Quorum Systems over the Internet," *IEEE Fault-Tolerant Computing Symposium*, Sendai, Japan, June 1996.

[8] T. Anderson, Y. Breitbart, H. F. Korth, and A. Wool. Replication, consistency, and practicality: Are these mutually exclusive? In *Proc. ACM SIGMOD Inter. Conf. Management of Data*, pages 484-495, Seattle, June 1998.

[9] P.A. Bernstein, V. Hadzilacos, N. Goodman, *Concurrency Control and Recovery in Database Systems.* Addison Wesley, Reading MA., 1987.

[10] K.P. Birman, T.A. Joseph, "Low Cost Management of Replicated Data in Fault-Tolerant Distributed Systems", *ACM Transactions on Computer Systems*,4(1):54-70, February 1986.

[11] R.K. Budhia. Performance Engineering of Group Communication Protocols. Ph.D. Dissertation. ECE department, University of California at Santa Barbara, August 1997

[12] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The Dangers of Replication and a Solution. In *Proc. ACM SIGMOD 1996 Int'l Conf. Management of Data*, Montreal, Quebec, pages 173-182, 1996.

[13] J.N. Gray. Notes on database systems. In R. Bayer, R.M.Graham, and G. Seegmuller, editors, *Operating Systems: An Advanced Course*, volume 60 of *Lecture Notes in Computer Science*, p. 393-481. Springer-Verlag, 1978.

[14] V. Hadzilacos and S. Toueg. "Fault-tolerant Broadcast and Related Problems". In S. Mullender, editor, *Distributed Systems*, chapter 5, p.97-147. Addison-Wesley, 1993.

[15] M.F. Kaashoek, and A.S. Tanenbaum, "Efficient Reliable Group Communication for Distributed Systems", 1994, Available via ftp: ftp.cse.ucsc.edu:/pub/amoeba/amoeba_papers

[16] B. Kemme and G. Alonso, "A Suite of Database Replication Protocols based on Group Communication," *18th International Conference on Distributed Computing Systems* (ICDCS 98), Amsterdam, The Netherlands, May 1998.

[17] R. Agrawal, M. Carey, M. Livny, "Concurrency Control Performance Modeling: Alternatives and Implications" and M. Carey, M. Livny, "Conflict Detection Tradeoffs for Replicated Data". In V. Kumar, editor, *Performance of Concurrency Control Mechanisms in Centralized Database Systems*, Prentice Hall, 1996.

[18] K. Petersen, M.J. Spreitzer, D.B. Terry, M.M. Theimer, and A. J. Demers, "Flexible Update Propagation for Weakly Consistent Replication," *Proceedings of the 16th ACM Symposium on Operating Systems Principles* (SOSP-16), Saint Malo, France, October 5-8, 1997, pages 288-301.

[19] C. Ruemmler and J. Wilkes, "An Introduction to Disk Drive Modeling," *IEEE Computer* 27(3) p17-29, March 1994.

[20] A. Schiper and M. Raynal. "From Group Communication to Transactions in Distributed Systems," *Communications of the ACM*, 39(4) p.84-87, April 1996.

[21] A. Wool, "Quorum Systems in Replicated Databases: Science or Fiction?" *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, Dec. 1998.