An Experimental Analysis of Replicated Copy Control During Site Failure and Recovery *

Bharat Bhargava
Paul Noll
Donna Sabo

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

Abstract

The function of replicated copy control is to maintain the consistency of copies during periods of site failure and recovery. The objectives of our research are to examine the effect of failures on the consistency of replicated copies, to measure the rate at which inconsistency can be removed, and to measure the overheads associated with replicated copy control. To conduct our research we have extended the prototype system RAID to utilize the ideas of session numbers, nominal session vectors, fail-locks, and control transactions for replicated copy control. In this paper we discuss the results of three experiments. In experiment 1 we measured the overhead for fail-locks maintenance, the overhead for control transactions, and the overhead for copier transactions. In experiment 2 we observed the effects of site failure on data availability. In experiment 3 we examined the maintenance of consistency of replicated copies during site failure and recovery.

1 Introduction

A distributed database system consists of logical data items which are replicated at different sites to increase availability and fault tolerance. Maintaining the consistency of replicated copies during site failure and recovery is an ongoing area of research [Bern84] [ElAb85].

The ideas of *session numbers*, *nominal session vectors*, *fail-locks*, and *control transactions* have been proposed as a design of the read-one/write-all-available copies protocol as a solution to the replicated copy control problem [Bhar86a] [Bhar87]. To investigate these ideas, we implemented a stripped down version of the prototype system RAID [Bhar86b] and conducted experiments to obtain results on the impact of site failure on data availability and the consistency of replicated copies. We also measured the overhead of maintaining consistent copies through periods of site failure and recovery.

The next two subsections discuss a specific replicated copy control algorithm and how it was implemented. Three experiments are detailed in sections 2, 3, and 4 of this paper. Appendix A contains the transaction commit processing.

1.1 Background

To maintain consistency of copies during site failure, each transaction employs the read-one/write-all-available copies (ROWAA) strategy. A protocol using the ROWAA strategy allows transaction processing as long as a single copy is available. If a transaction on an operational site knows that a particular site k is down, the transaction does not attempt to read a copy from site k or to send an update to site k. This saves the time that would be wasted in waiting for responses from an unavailable site and also reduces the possibility of aborting or blocking transactions.

Our protocol design is based on the ideas of session numbers and nominal session vectors in conjunction with the ROWAA strategy. The proof of correctness of this protocol is given in [Bhar86a]. A session number identifies a time period in which a site is *up* (operational). A site which has failed such that it is no longer processing transactions is said to be *down*. A session number is also useful in determining if the status of a site has changed during the execution of a transaction. A nominal session vector for a site consists of the site's own session number and the perceived session numbers of the other sites in the system. A site uses its nominal session vector to determine which sites are operational (only operational sites can participate in a protocol based on the ROWAA strategy).

A **control transaction** is used to signal a change

*This work was partially supported by grants from UNISYS Corp., NASA, and AIRMICS.

in a nominal session vector. A control transaction of *type 1* is issued by a recovering site to announce that it is preparing to become operational. It causes the nominal session vectors of the operational sites to be updated with the recovering site's new session number and also obtains a copy of the session vector and fail-locks from an operational site for the recovering site. A control transaction of *type 2* is issued by a site to announce that it has determined that one or more sites which were operational have failed. It causes the nominal session vectors of the remaining operational sites to be updated to indicate the transition of the failed sites from an operational state to a nonoperational state.

To identify out-of-date data items on recovering sites, the concept of **fail-locks** is used [Bhar87]. This idea is adopted from the concept of a lock in concurrency control algorithms where a lock on a data item represents the fact to all other transactions that the locked item is being used by a transaction. A replicated copy control algorithm uses a fail-lock to represent the fact that a copy of a data item is being updated while some other copies are unavailable due to site failure or network partitioning.

In our replicated copy control algorithm, each copy for a data item has a fail-lock bit for each site. Fail-lock bits are set by an operational site on behalf of a failed site which has missed an update. A recovering site collects its fail-locks from the operational sites to determine which data items missed updates. A recovering site can distinguish out-of-date data items from up-to-date data items so the up-to-date data items are made available for transaction processing. A recovering site clears a fail-lock bit for a data item after it has become refreshed by a write on such an out-of-date data item. A recovering site can use a *copier transaction* to refresh an out-of-date copy. A copier transaction causes a read from a good data item on another operational site and a write to the data item on the recovering site. Fail-locks should be fully replicated to increase fault tolerance. The replication of fail-locks is discussed in [Bhar87].

1.2 Implementation Choices for Our Experiments

RAID is an experimental system being developed on VAXes and SUNs under the Unix operating system [Bhar86b]. Database sites in RAID communicate over an Ethernet network. We stripped down the processing of the RAID system in order to obtain results which were relevant to replicated copy control. Factored out were the effects of network communications, concurrency control, and data input/output. In the resulting system, which we call mini-RAID, database sites were implemented as Unix processes (on one processor with one process per site). Due to this, the influence of communication delays on the Ethernet is not considered. Each site kept a copy of the database, nominal session vector, and fail-locks and executed the same protocol to maintain the consistency of these objects.

Since our effort was focused on obtaining clear measurements of data availability and overhead costs the following assumptions were made:

1. We assumed that the distributed database system was equipped with a reliable message passing facility: no messages were lost; messages arrived and were processed in the order that they were sent; and no errors in transmission altered the messages. Our system did not attempt to check for communication errors or correct them.

2. Our system did not include concurrency control and transactions were processed serially. Due to this assumption, our measurements were independent of the interference and scheduling delays from concurrency control processing.

3. We assumed that each site performed data I/O as necessary. Our system kept data copies within the virtual memory of each process which represented a site. Due to this assumption, our measurements were independent of I/O overheads.

4. We assumed that the database was fully replicated. We also assumed, without loss of generality, that the data items used in our experiments represented only the portion of the database consisting of very frequently referenced data items.

We implemented a **nominal session vector** as an array of records, with each record representing a site. The information maintained for a site included its perceived session number and its state. The possible states of a site included the status information: site is up, site is down, site is waiting to recover, and site is terminating.

We implemented **fail-locks** with a bit map for each data item. The size of each bit map was less than or equal to the number of possible sites. Each bit represented a site with a value of 1 in the nth bit indicating that a fail-lock was set for the nth site for the data item. This implementation allowed the fail-lock operations to be performed very quickly.

We implemented a **managing site** to provide interactive control of system actions. It was used to cause sites to fail and recover and to initiate a

database transaction to a site. Site failure was simulated by sending a message to a site to indicate that the site should not participate in any further system actions. A failed site would remain inactive until recovery was initiated from the managing site. The following system parameters were defined through the managing site:

- the database size in terms of the number of data items
- the number of database sites for the transaction processing (not including the managing site)
- the maximum number of operations per transaction, where an operation was defined to be a read or write of a database data item.

A **database transaction** was generated by the managing site and consisted of a random number of operations (from 1 to the maximum specified for the system). There was an equal probability of an operation being a read or a write and each operation was for a randomly chosen data item from the database. This corresponds to our assumption of considering only the set of most frequently referenced data items in the database. All of these data items have approximately equal probabilities of being referenced. In the near future, we hope to repeat our experiments with the well-known benchmarks ET1 from Tandem Corporation [Anon85] and the Wisconsin benchmark from the University of Wisconsin [Bitt83].

Database transactions were processed by a two-phase **commit protocol**. The site which received a transaction from the managing site acted as the coordinator for the protocol. It will be referred to in this paper as the *coordinating site*. The remaining database sites which were operational at the time the transaction started will be referred to as *participating sites*. The precise actions of a coordinating site and a participating site are described more fully in Appendix A of this paper.

Fail-locks were maintained during the commitment of data copies. As a transaction committed a particular copy on a site, the nominal session vector was examined and the fail-lock bit was cleared for each operational site and the fail-lock bit was set for each failed site. Note that this resulted in some fail-lock bits being re-cleared for an operational site. However, for our system this implementation was more efficient than conditionally performing fail-lock maintenance based on a site's state.

A **copier transaction** was issued on demand by a coordinating site during its recovery period and was completed before phase one of the commit protocol began. If a coordinating site was unable to complete all necessary copier transactions the database transaction was aborted. A special type of transaction was implemented to allow a coordinating site to inform other sites of the fail-lock bits cleared by copier transactions. This special transaction was initiated by a coordinating site after all necessary copier transactions had been issued and the correct values for the data items in the copier transactions had been written at the coordinating site.

2 Experiment 1: Overhead Measurements

An experiment was conducted to measure the performance of the ROWAA protocol. The overhead associated with keeping consistent replicated copies was studied by three factors:

- the processing necessary for fail-lock maintenance
- the time required by control transactions
- the time required by copier transactions.

2.1 Design of Experiment

This experiment used the mini-RAID system described in the previous section of this paper. Sets of transactions were run through the system repeatedly over a two month period and the execution times of processing events were recorded after a stable state of transaction processing was achieved. The times presented here are the averages of the recorded times. Execution times were measured in the software by referencing the processor clock.

It should be stressed that the average times are not intended to represent the absolute performance of the system but rather the performance of the system for a particular configuration of system parameters. Thus the comparison of average times is of more interest than the numerical value of each average time.

Intersite communications were an important component of execution times. Since sites were abstracted as Unix processes on a single processor, these communication costs reduced to the cost of interprocess communications. The average time for a single communication from one site to another site was measured as nine milliseconds.

2.2 Measured Data

The following system parameters were used:

- Size of the frequently referenced portion of the database = 50 items
- Number of sites = 4
- Maximum transaction size = 10 items

2.2.1 Overhead for Fail-locks Maintenance

The overhead for fail-locks maintenance is the cost of clearing and setting fail-locks during the commitment of data copies. This commitment is part of the two phase commit protocol used for database transaction processing. Measurements were made by running a set of transactions with the fail-locks maintenance code removed from the software and then re-running the same set of transactions with the fail-locks maintenance code included in the software. The transaction time on a coordinating site was taken as the elapsed time between the initial reception of the database transaction and the completion of the two phase commit protocol. The transaction time on a participating site was taken as the elapsed time between the start of the site's participation in phase one of the protocol and the completion of the site's participation in phase two. The transactions did not generate any copier transactions.

	Transaction time without fail-locks code	Transaction time with fail-locks code
Coordinating site	176 ms	186 ms
Participating site	90 ms	97 ms

2.2.2 Overhead for Control Transactions

Times were measured for the processing of control transactions. The time for a type 1 control transaction to complete at a recovering site was 190 milliseconds. This includes the recovery announcement which goes from the recovering site to each operational site and the installation of a new session vector and fail-locks. The time for a type 1 control transaction is dependent on the number of sites in the system because an intersite communication is needed for each recovery announcement.

The time for a type 1 control transaction to complete at an operational site was 50 milliseconds. It consists of the formatting and sending of a message with session vector and fail-locks. Since only one intersite communication is required (between the operational site and the recovering site), this time is independent of the number of sites in the system. However, it is dependent on the size of the database because a large increase in the number of data items (and thus fail-locks) would require more storage for fail-locks and could increase the amount of time to transmit the message containing the fail-locks.

The time for a type 2 control transaction to complete was 68 milliseconds. This includes the sending of the failure announcement to a particular site and the updating of the session vector at that site. This time is independent of the number of sites in the system.

2.2.3 Overhead for Copier Transactions

Measurements were made for a database transaction which included the generation of one copier transaction. The scenario was that a coordinating site received a database transaction which included a read operation for a fail-locked copy. A copier transaction was then run to get an up-to-date copy. The transaction time was taken as the elapsed time between the initial reception of the database transaction, the completion of the copier transaction, and the completion of the two phase commit protocol. The transaction time was an average of 270 milliseconds.

The overhead for a site which received a copy request was 25 milliseconds. This includes the time to format a response with the specified copies and the sending of the response to the coordinating site.

The time for a transaction to clear fail-locks includes the sending of the message to a particular site and the clearing of the appropriate fail-locks on that site. This time was measured to be 20 milliseconds.

The transaction time of 270 milliseconds was an increase of 45% over the time for a database transaction which generated no copier transactions. However, approximately 30% of this cost was made up of the cost of transactions needed to clear fail-locks after a copier transaction. Therefore, an implementation which eliminated these transactions by embedding the necessary fail-locks information in the two phase commit protocol could significantly reduce this overhead.

2.3 Conclusions of Experiment 1

The overhead in fail-locks maintenance caused a slight increase in transaction processing times. The potential impact of this overhead was lessened by the incorporation of fail-locks processing into the commit protocol.

The overhead for a control transaction was comparable to the cost of a small database transaction. However, the relative infrequency of the control trans-

actions would greatly reduce the average cost of this overhead.

The overhead for copier transactions had an impact on a recovering site. There was a significant increase in time for a database transaction which generated a copier transaction. Therefore, this overhead may require the most consideration when making design choices. We discuss this point further in experiment 2. In any case, the increased overhead for a recovering site must be weighed against the increased data availability achieved for the site.

3 Experiment 2: Data Availability on a Recovering Site

The ROWAA protocol provides high data availability on a recovering site since a down site may become operational and quickly begin transaction processing using the up-to-date portion of its database. To further increase data availability and fault tolerance we must ensure that a site with an out-of-date copy refreshes the copy before all other sites with an up-to-date copy go down. In this experiment we studied the rate at which the fail-locks for out-of-date copies are cleared during the recovery period.

3.1 Design of Experiment

The mini-RAID system described earlier was used for this experiment. For a simple two site case, we failed one site and processed transactions until most of the copies on the down site were fail-locked by the operational site. We then brought the failed site up and processed transactions until the recovering site had completely recovered (i.e., all out-of-date copies were refreshed). Transactions were sequentially numbered from 1 for identification.

3.1.1 Measured Data

The parameters below were chosen to enable us to observe the recovery period for a simple system. If the ratio of the maximum transaction size to the number of data items being considered is large then most of the data items would be fail locked for the down site after processing only a few transactions.

- Size of the frequently referenced portion of the database = 50 items
- Number of sites = 2 (sites 0 and 1)
- Maximum transaction size = 5 items

To study a complete recovery period we used the following scenario for the experiment:

- Initially both sites were up with consistent and up-to-date copies.
- Before transaction 1, we caused site 0 to fail.
- For transactions 1-100 we kept site 0 down and processed transactions on site 1.
- Before transaction 101, site 0 was brought up.
- Transactions were processed on both sites until site 0 was completely recovered with all fail-locks cleared.

We observed that processing 100 transactions on site 1 while site 0 was down resulted in setting fail-locks for over 90% of the copies on site 0. After bringing site 0 up, 160 additional transactions were needed in order to completely recover site 0.

For each transaction we recorded the number of fail-locks set, the number of fail-locks cleared, and the number of copier transactions requested. The graph in Figure 1 illustrates the measured data in terms of the number of fail-locked copies after the processing of a certain number of transactions. The peak in the graph corresponds to the point at which site 0 was brought up and began its recovery. To the left of the peak site 0 was down and fail-locks were being set on site 1 for the copies that were being updated on site 1 but not on site 0. To the right of the peak fail-locks were cleared for the out-of-date copies being refreshed on site 0 by write operations in transactions and by copier transactions requested by the recovering site. Only two copier transactions were requested by site 0 during its recovery period.

3.1.2 Analysis of Data

The graph shows that while a high percentage of the database is fail-locked, fail-locks are cleared very rapidly. As the percentage of the data items fail-locked decreases, more transactions are required to clear fail-locks. The first 10 fail-locks were cleared in only 6 transactions and the last 10 fail-locks were cleared in 106 transactions!

3.2 Conclusions of Experiment 2

We conclude that the rate at which fail-locks are cleared is directly related to the percentage of data items fail-locked. Future implementations could take advantage of this fact to decrease the length of the recovery period.

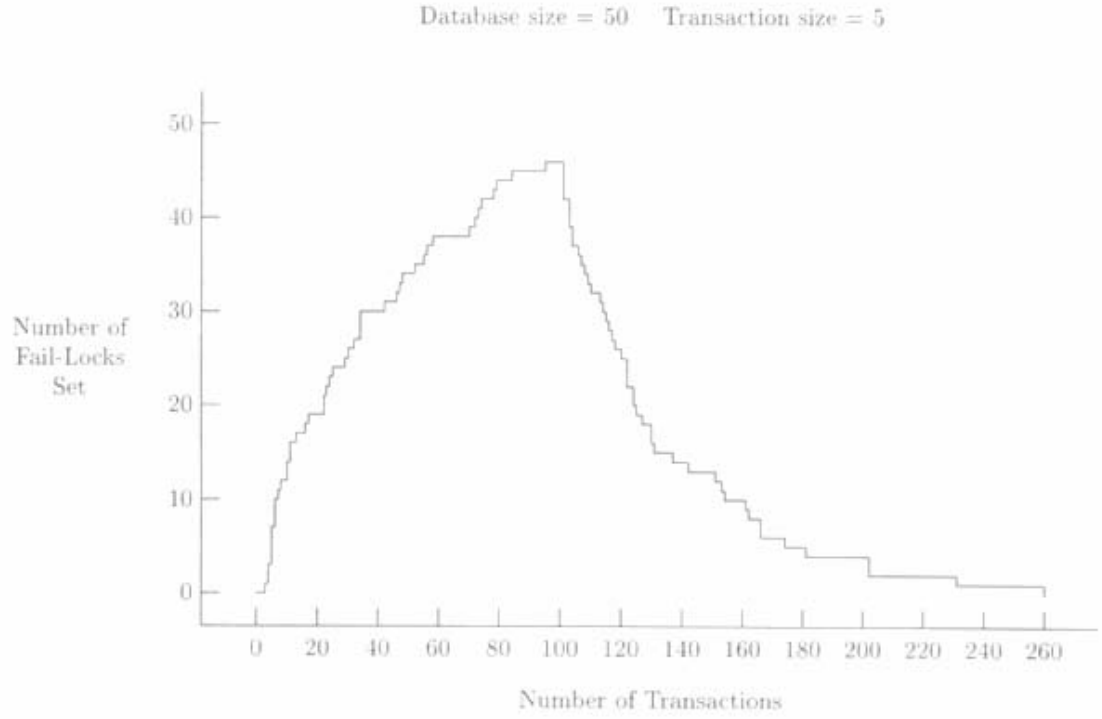

Figure 1: Data availability during failure and recovery

One idea is to divide the recovery period into two steps. Before beginning the first step, a recovering site computes the percentage of the frequently referenced copies that are fail-locked. If this percentage is greater than some threshold value, the recovering site enters step one. Otherwise the recovering site omits the first step and enters step two. In the first step copies are refreshed in the same manner as in our implementation, (i.e., by normal transaction processing and by the recovering site requesting copier transactions on demand). Once the percentage of copies fail-locked drops below the threshold the site enters step two of its recovery. In the second step the recovering site begins to issue copier transactions in a "batch" mode. Copier transactions are generated even though no transactions have arrived on the recovering site with a read request for any of the remaining out-of-date copies. This causes the out-of-date copies to be refreshed and hastens the completion of recovery. This two step method would increase fault tolerance by decreasing the possibility of a site's recovery being blocked by the failure of other sites.

In a partially replicated database system using the ROWAA protocol, data availability could be increased by creating a control transaction of *type* 3. Using this control transaction, a site having the last up-to-date copy of a data item would create a copy on a back-up site that has no copy of that data item (assume a back-up site exists or we have a partially replicated database). This increased data availability would have the cost of the type 3 control transaction plus the cost of removing copies of data items from sites once these additional copies were not needed any more.

4 Experiment 3: Consistency of Replicated Copies

In this experiment we observed how site failures affect the consistency of replicated copies. Note that since each set fail-lock represents an inconsistent copy, the number of fail-locks set is a measure of inconsistency.

4.1 Design of Experiment

The mini-RAID system that we implemented was used for this experiment. We observed two simple scenarios with multiple sites recovering concurrently.

4.2 Measured Data

4.2.1 Scenario 1

The following parameters were chosen for the first scenario.

- Size of the frequently referenced portion of the database = 50 items
- Number of sites = 2
- Maximum transaction size = 5 items

Transactions were processed and sites were failed in the following order.

- Initially both sites were up with consistent and up-to-date copies.
- Before transaction 1, site 0 failed.
- For transactions 1-25 we kept site 0 down and processed transactions on site 1.
- Before transaction 26, we brought site 0 up and failed site 1.
- For transactions 26-50 we kept site 1 down and processed transactions on site 0.
- Before transaction 51, we brought site 1 up.
- Transactions 51-120 were processed on both sites.

The graph in Figure 2 illustrates the measured data for both sites. The basic shape of the graph for each site is similar to the graph of a single site recovering shown in Figure 1. In this scenario the fact that site 1 went down for part of site 0's recovery period caused some of the data items fail locked on site 0 to be totally unavailable. During this time fail-locks were cleared on site 0 only by transaction writes because site 1 was not available to receive copier transactions. The inability to get up-to-date copies via copier transactions forced site 0 to abort 13 transactions.

4.2.2 Scenario 2

The parameters below were used for the second scenario.

- Size of the frequently referenced portion of the database = 50 items.
- Number of sites = 4
- Maximum transaction size = 5 items

Transactions were processed and sites were failed in the order listed below.

- Initially all sites were up with consistent and up-to-date copies.
- Before transaction 1, we failed site 0.
- For transactions 1-25 we kept site 0 down and processed transactions on the remaining sites.
- Before transaction 26, we brought site 0 up and failed site 1.
- For transactions 26-50 we kept site 1 down and processed transactions on the remaining sites.
- Before transaction 51, we brought site 1 up and failed site 2.
- For transactions 51-75 we kept site 2 down and processed transactions on the remaining sites.
- Before transaction 76, we brought site 2 up and failed site 3.
- For transactions 76-100 we kept site 3 down and processed transactions on the remaining sites.
- Before transaction 101, we brought site 3 up.
- Transactions 101-160 were processed on all sites.

The graph in Figure 3 illustrates the measured data for each site. Again, the basic shape of the graph for each site is similar to that for the single site recovery. Since the sites went down singly in succession and only once, an up-to-date copy of a data item was always available on some site. Thus the sites were able to recover without any aborted transactions due to data being unavailable.

4.3 Conclusion of Experiment 3

During site failure the copy of the database on the failed site quickly becomes inconsistent with the other copies of the database on operational sites. Write operations in database transactions and copier transactions requested by the recovering site are able to bring the database back to a consistent state relatively fast.

5 Conclusions

A distributed database system that employs the ROWAA protocol has a higher degree of data availability at the operational sites (since failed sites can be ignored) and at the recovering sites (due to fail-locks). In experiment 1 we determined the overheads present in our implementation of the protocol. The cost of the overhead for copier transactions may be

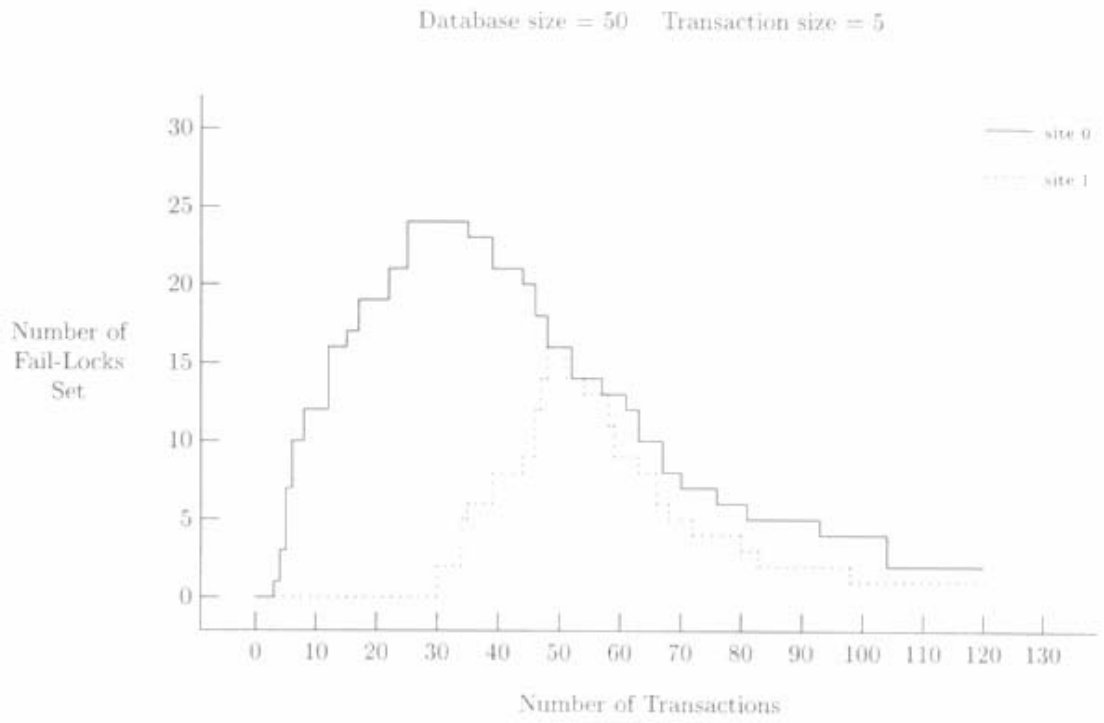

Figure 2: Database Inconsistency (scenario 1)

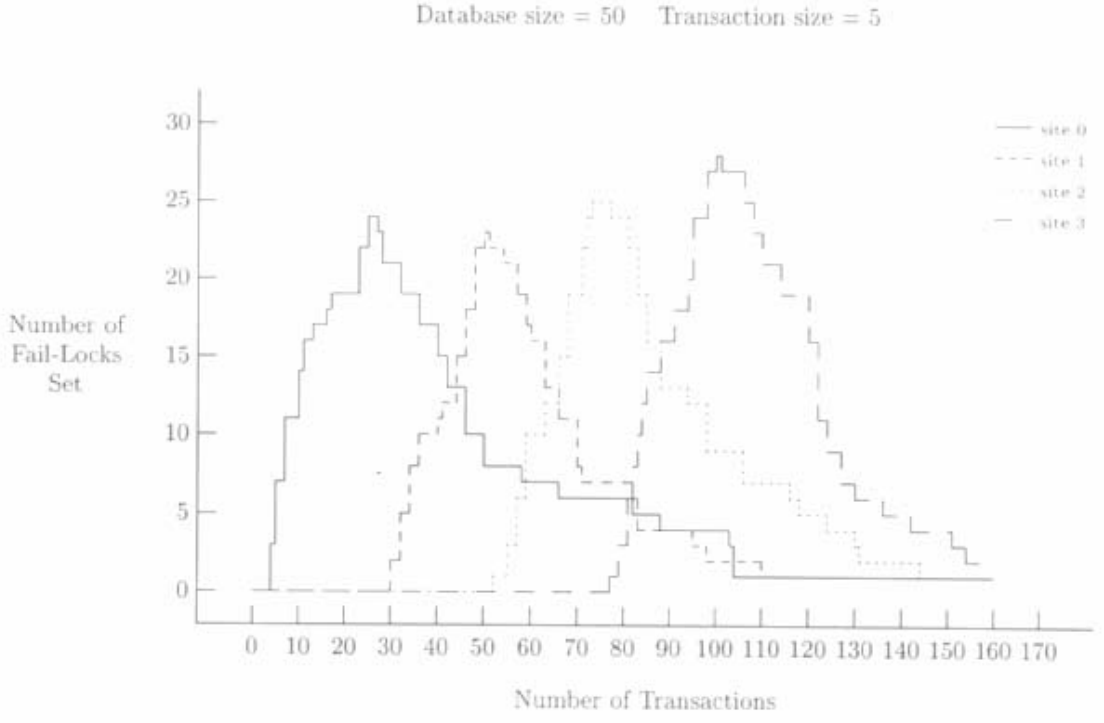

Figure 3: Database Inconsistency (scenario 2)

the most important consideration for implementors. We found that the response time for database transactions on a recovering site which required copier transactions was significantly higher than database transactions not requiring copier transactions.

In experiment 2 we observed that the rate at which fail-locks were cleared during site recovery was dependent on the percentage of copies fail-locked. Based on our analysis, we suggested a two step recovery period to increase the fault tolerance of a system. We also suggested the creation of control transaction 3 to increase data availability.

In experiment 3 we examined consistency control during site failure and recovery. We found that fail-locks can properly track the location of the correct values for data items even when these values are spread out over multiple sites. This allowed transaction processing to continue after the occurrence of successive site failures.

The fact that transactions were randomly generated in our implementation may be of concern to some who say that in reality transactions are not random and actually all data items are accessed with different probabilities. We made the assumption that in a database there is a subset of data items that is frequently referenced. We also assumed that the data items in this set have approximately equal probabilities of being accessed. The remaining part of the database is accessed less frequently. Including these rarely accessed data items in our experiments would not significantly alter our results.

Another issue that we must address is the fact that studies have shown that typically reads are far more common than writes. We have implemented the number of reads to be approximately equal to the number of writes. This assumption may be to our disadvantage during the time that fail-locks are being set. A fail lock is set for each down site every time a write operation is performed on a data item. This reduces our data availability more quickly than if we had assumed that writes occur less often than reads. However, this assumption also has the effect of increasing data availability more quickly during recovery with fewer copier transactions. In our implementation many of the fail-locks were cleared by writes instead of by copier transactions requested by a recovering site. If reads occur more commonly than writes then more copier transactions would probably be requested by a recovering site during recovery. Since the probabilities remained constant throughout the experiments, valid comparisons can be made between the scenarios.

We implemented our mini-RAID system in the C programming language under the Unix operating system. Interested researchers may contact us for a copy of the source code to repeat these experiments or to conduct their own.

In the near future, we hope to repeat our experiments using benchmark sets of transactions. We also plan to run this protocol in the complete RAID system and take into account other factors such as concurrency control and communication delays across machines.

A Transaction Commit Protocol

This appendix contains a pseudo-code description of the two phase commit protocol used by our system. Although it is not explicitly stated in this description, multiple copier transactions can be generated for a database transaction.

Actions at the coordinating site

```
receive database transaction from managing site;

if   transaction contains read operation for a
     fail-locked copy then
     /* run copier transaction */
     issue copy request to an operational site which
     has up-to-date copy;
     if   copy update returned by other site then
          update database and clear fail-lock(s);
          run special transaction to clear fail-locks
          at other sites;
     else /* other site is now down */
          abort database transaction;
          run control type 2 transaction to
          announce failure;
     end if
end if

if database transaction not aborted then
     /* begin phase one of protocol */
     issue copy update for written items to every
     operational site;
     if ack received from all participating sites then
          /* begin phase two of protocol */
          send commit indication to participating
          sites;
          if   commit ack not received from all
          participating sites then
               run control type 2 transaction to
               announce failure;
          end if
          commit database data items;
```

```
        update fail-locks for data items;
    else /* a participating site has failed */
        abort database transaction;
        run control type 2 transaction to announce
        failure;
    end if
end if
```

- **Actions at a participating site**

```
/* phase one of protocol */
receive copy update from coordinating site;
send ack to coordinating site;
if  commit indication received then
    commit database data items;
    update fail-locks for data items;
else if abort indication received then
    discard the copy updates;
else /* coordinating site has failed */
    run control type 2 transaction to announce
    failure;
end if
```

References

[Anon85] Anon, et. al., "A measure of transaction processing power", *Datamation*, vol. 31, no. 7, 112-118, April 1, 1985.

[Bern84] P. A. Bernstein, and N. Goodman, "An algorithm for concurrency control and recovery in replicated distributed databases", *ACM Trans. Database Syst.*, vol. 9, no. 4, 596-615, Dec. 1984.

[Bhar86a] B. Bhargava, and Z. Ruan, "Site recovery in replicated distributed database systems", *Sixth IEEE International Conference on Distributed Computing Systems*, 621-627, May 1986.

[Bhar86b] B. Bhargava, and J. Riedl, "The design of an adaptable distributed system" *IEEE COMPSAC Conference*, 114-122, Oct. 1986.

[Bhar87] B. Bhargava, "Transaction processing and consistency control of replicated copies during failures," *Journal of Management Information Systems*, Oct. 1987, vol. 4, no. 2.

[Bitt83] D. Bitton, D.J. DeWitt, and C. Turbyfil, "Benchmarking database systems: a systematic approach", *Proceedings VLDB Conference*, Oct. 1983.

[ElAb85] A. El Abbadi, D. Skeen, and F. Christian, "An efficient fault tolerant protocol for replicated data management", *Proc. 4th ACM Symp. on Princ. of Database Systems*, Portland, Oregon, 215-229, March, 1985.