Adaptable Recovery Using Dynamic Quorum Assignments *

Bharat Bhargava and Shirley Browne

Department of Computer Sciences, Purdue University, West Lafayette, IN 47907

Abstract. This research investigates the problem of how to adapt the changing of quorum assignments for objects in a replicated database to the duration and extent of failures occurring in the underlying communication network. The concept of a view based on a connected component of the network is used to coordinate changes to quorum assignments of different objects. A dynamic view formation protocol is proposed that permits objects to join a new view on demand. A new technique called inheritance enables a new view to acquire quorum assignments from an old view, so that only those objects that were accessed during a failure need to change their quorum assignments back following repair of the failure. Extension of an existing view may be used to incorporate a recovering site into the network without forming a new view, thus localizing the effects of the failure. We have made analytical performance estimates for some sample network configurations and failure situations to show the improvements of our method over previously proposed methods. Following repair of a failure, our method can begin processing transactions almost immediately, but with less extra average overhead than for previous methods. We describe a prototype implementation of our method that will be used for future experimentation.

*This research is sponsored by National Science Foundation Grant IRI-8821398.

Proceedings of the 16th VLDB Conference
Brisbane, Australia 1990

1 Introduction

The need for adaptability and reconfigurability in a distributed system has been discussed in [2], [3], and [6]. In [6] a model for adaptability in a distributed database system is proposed and applied to distributed concurrency control and commit protocols, to network partitioning control, and to server relocation. Data replication increases the opportunities for adapting to failures and changing conditions, but adds the problem of maintaining mutual consistency of the replicated copies. Maintaining mutual consistency involves both concurrency control and replication control protocols. The purpose of these protocols is to achieve one-copy serializablity – that is, to ensure that the concurrent execution of transactions on the replicated database has the same effect and appearance as a serial execution on a one-copy database [4].

The use of quorums to deal with site failures and network partitioning was proposed in [9]. A quorum assignment for a replicated object specifies how many or which copies must be accessed to carry out an operation. A quorum method may be either static or dynamic. With a static method, quorum assignments are fixed. A dynamic method allows quorum assignments to be changed in order to increase availability. Methods for coordinating changes to quorum assignments have been proposed in [1], [10], and [11]. Changes to quorum assignments are typically coordinated by means of views. A view of the database is essentially a set of sites that can communicate with each other, together with the copies of objects residing at those sites. Dynamic quorum methods are expensive in processing and communication overhead. The cost is incurred either all at once when a new view of the database is formed, or incrementally as database objects are accessed in the new view. In [1] and [10], all objects in a connected component of the network change their quorum assignments at the same time by means of a single reconfiguration transaction. The same reconfiguration protocol is always

invoked, regardless of the extent of the failure. In [11], objects may change quorum assignments one at a time, but a cascade effect causes the quorum assignments for all objects to eventually be changed.

The problem addressed in our research is how to make dynamic quorum methods adaptable to the duration and extent of failures in order to reduce communication costs and overheads during recovery. The cost should be proportional to the severity of the failure, rather than to the size of the database.

Our approach to handling failures is to change the quorum assignment for an object only when that object is accessed by a transaction, as in [11]. By using a new technique called *inheritance*, however, quorum assignments that have not been changed during a network partitioning failure may remain in effect after the failure has been repaired. A new view is formed following repair, but this new view inherits quorum assignments from the old view that existed prior to the failure. Hence, the amount of work required following repair is proportional to the number of quorum assignment changes made during the failure, which depends on the length of the failure. Alternately, to handle repair of a lengthier failure, quorum assignments may be inherited by the new view from the current view for the largest partition of the network. In this way, the cost of repair is localized to objects that were accessed during the failure in smaller partitions. Objects that were accessed in the larger partition need only to extend their quorum assignments to include the copies being merged in.

To handle a recovery from a brief site failure, inheritance from the old view that existed prior to the site failure may be used. To recover from a lengthier site failure, we propose a new *view extension* technique followed by extension of quorum assignments to include copies at the recovered site.

Because our adaptable dynamic quorum method can change quorum assignments on demand and can re-use unchanged assignments following repair, it adapts to the access pattern of transactions that run during the failure. By localizing the effects of a site failure or of the separation of a small number of sites, our method also adapts to the extent of the failure.

In this paper, we describe our adaptable dynamic quorum method and briefly describe measurements from a prototype implementation based on a version of the distributed database system in [7]. These measurements are used to evaluate the estimated performance of our method analytically under various failure situations. A proof that our method ensures one-copy serializability of transaction processing is given in [8].

2 Quorum Model and Terminology

A set of copies that suffices to carry out an operation is called a *quorum*. With an appropriate quorum intersection requirement for conflicting operations, the use of quorums, together with distributed concurrency control and atomic commitment protocols, ensures the consistency of replicated data as seen by user transactions. The possible quorums that may be used are listed in a *quorum assignment*. We assume that an object's quorum assignments are stored with each copy of the object. 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.

A *coquorum* for a set S of quorums in a quorum assignment is a set of copies that intersects every quorum in S.

Quorum intersection rules that must be followed by quorum assignments for a dynamic quorum method have been given in [10, 11] for abstract data types. These rules are stated in the read-write model as follows:

1. An active write quorum must intersect all active read quorums (i.e., an active write quorum must be an active read coquorum).

2. A backup write quorum must intersect all backup read quorums.

3. An active write quorum must intersect all backup read quorums.

Using both types of quorums yields better availability during failures without sacrificing performance in the absence of failures. An object may change its active quorum assignment even when currently active quorums are unavailable, provided backup quorums are available.

To achieve one-copy serializability, objects coordinate changes to their active quorum assignments by means of views. In our model, a *view* is a set of copies of objects that agree to use a particular set of quorum assignments that satisfy the quorum assignment rules. Each view has a unique integer view id. The

view formation protocol given in section 3 ensures that the view ids for views in which a given copy of an object participates are increasing over time. A view is based on a connected component of the network, that is, a set of sites that can communicate with each other. A view is associated with a *connection vector*, indexed by the sites in the system, with a 1 entry for each site in the view's underlying component and a 0 entry for the other sites.

A transaction must execute entirely within a single view. The quorum intersection rules guarantee that transactions executed in the same view are serializable among themselves. Our proof of correctness, given in [8], shows that transactions executed in different views are serializable in order of their view identifiers.

3 Adaptability Techniques

Our dynamic quorum method has the following five components, with the last two specifically designed for adaptability:

- a view formation protocol,
- a protocol for changing quorum assignments that can be used when currently active quorums are unavailable but involves a move to a new view,
- a lightweight protocol for changing quorum assignments that requires a read and a write quorum to be available but does not require new view formation,
- an inheritance mechanism that permits a new view to acquire objects from an old view without changing their quorum assignments,
- a view extension protocol that changes the connection vector for a view without changing the view id.

Recovery actions that do not require formation of a new view, such as the lightweight and view extension protocols, are cheaper than those that form a new view with a new view id. There are two reasons for the difference in cost. The first is that formation of a new view requires participation of all sites in the new view, whereas a quorum assignment change that does not require a move to a new view need involve only those sites having copies of the object. The second reason is that changing the view id is likely to cause transactions that are in progress concurrently to abort and restart in the new view, because a transaction must execute entirely with a single view.

We assume that the distributed database system runs a correct distributed conflict-preserving concurrency control protocol, as well as conventional disk-based crash recovery algorithms [4]. We also assume the use of transaction commit and termination protocols, as described in [13].

3.1 New View Formation

New view formation may be invoked by a transaction manager when failures prevent the use of the quorum assignments associated with the current view, or when repair of a failure makes more copies available for inclusion in active quorum assignments. Following formation of a new view, which is initially empty of any objects, objects are moved to the new view on demand as they are accessed by transactions. Transactions will typically attempt to execute in the most recent view known to have been formed.

To request that a new view be formed, a transaction manager sends a view formation request to all sites in the new view's component. The request contains the connection vector for the new view. A site replies to a request by returning the highest view id it has seen so far. The view formation coordinator chooses a unique new view id greater than any of those received and sends it out in a commit message. Logging the new view id to stable storage is not required. If the view formation is interrupted by a failure, then any site may initiate still another new view formation, regardless of whether the site is able to determine the outcome of the previous attempt.

A separate view formation protocol is not actually required for correctness, as a new view could be formed incrementally in conjunction with moving objects to the new view. However, we conjecture that using a separate view formation protocol will help cut down on the number of transaction aborts that occur following repair of multiple failures.

3.2 Moving an Object to a New View

After a new view has been formed, it will not at first contain any objects. Hence, when a transaction attempts to execute in a newly formed view, it will find that the objects it wants to access are not present in the view. If an object that the transaction wants to access is accessible in the view's component (i.e., the object has a backup quorum in the component), the transaction can attempt to move the object into the new view with a new active quorum assignment. If

the object is both read- and write-accessible, then the new active quorum assignment can be any assignment that satisfies the quorum intersection rules given in section 2. If the object is only write-accessible, then the new active assignment is set equal to the backup assignment. If the object is only read-accessible, then the active assignment may instead be set equal to the most recent active assignment, because no other disjoint view can have write access. To move an object into a new view, a transaction manager carries out the following steps:

1. Send view change request messages with the new view id and new active quorum assignment to all copies of the object in the new view. If the object is read accessible in the new view, include a read request for the object to members of a backup read quorum. Each copy checks that its view id is less than the new view id. If so, the copy write locks its quorum assignment, logs the proposed change, and, if requested to do so as a member of a backup read quorum, returns information about the object. Otherwise, if its view id is greater or equal, the copy returns the newer view id and the corresponding active quorum assignment. If some copy returns a newer view id, the view change is aborted. Otherwise, proceed to step 2.

2. If the object is not read-accessible in the new view, go to step 3. Otherwise, send enough information about the object, obtained from a backup read quorum,to a new active write quorum so as to ensure that all members of the new write quorum are up-to-date After replies are received from members of the new write quorum indicating that they have logged any new events or values, proceed to step 3.

3. Send a commit message to all copies in the new view. Upon receiving a commit message, a copy marks itself deleted in the old view (it may retain read access there, however, as long as the old version is kept), adds itself to the new view with the new active quorum assignment, and releases the write lock on the quorum assignment.

3.3 Lightweight Quorum Assignment Change

If both a read quorum and a write quorum for an object are available, then its quorum assignment may be changed without forming a new view. Note that a read quorum is a write coquorum (i.e., a read quorum intersects every write quorum) and a write quorum is a read coquorum. Hence, notifying both a read quorum and a write quorum of the change ensures that at least one member of every old quorum has been notified. This rule applies equally well to both active and backup quorum assignments. Either type of quorum assignment may be changed without changing the other, provided the resulting active and backup quorum assignments satisfy the quorum intersection rules given in section 2.

To change a quorum assignment for an object, a transaction manager carries out the following steps:

1. Send a request for a quorum assignment change containing the old quorum assignment version number to all available old quorum members and to all new quorum members. If the change is for an active quorum assignment, include a read request for the object to members of an old read quorum.

2. Upon receiving a request for quorum assignment change, a quorum member write locks the quorum assignment and logs the new assignment to stable storage. It also checks if its quorum assignment version number agrees with the one in the request. If its own quorum assignment is more recent, it sends back its own quorum assignment. If requested to do so as a member of a read quorum, it sends back information about the object. After replies are received from all new quorum members and from at least one old read quorum and one old write quorum, proceed to step 3. (If a more recent quorum assignment was received from some site, restart the protocol with step 1).

3. If the change is only for a backup quorum assignment, proceed to step 4. Otherwise (the active quorum assignment is being changed), send enough information about the object (obtained from the old read quorum) to a new write quorum to ensure that all members of the new write quorum are up-to-date. After replies are received from members of the new write quorum indicating they have logged any new values or events, proceed to step 4.

4. Send a commit message to all new quorum members and to all available old quorum members. Upon receiving a commit message, a quorum member updates the quorum assignment and releases the write lock on the quorum assignment.

3.4 Inheritance of Quorum Assignments by Views

Inheritance essentially permits re-use of an old view that is still largely intact. When coordinating the formation of a new view, a site may check whether some old view has the same connection vector as the new one. If so, and if a large number of dynamic objects still reside in the old view, it may choose to have the new view inherit these objects from the old view. If it decides on inheritance, it sends the view id for the old view with the view formation request. Upon receiving the request, each site returns the names of any objects that it has deleted from the old view. The coordinator sends the names of all deleted objects with the commit message. Upon receiving the commit message, each site checks that all named objects have been deleted and switches the view id of the old view to that of the new one. If each object stores a pointer to the location of the view id, rather than the view id itself, then the view id for all the objects can be changed with a single write operation. Alternatively, the site can change the view ids of all the objects.

The idea behind our proof of correctness for inheritance is that at least one member of any old quorum must know about the deletion of an object. This knowledge prevents an old quorum assignment from being used for a deleted object. Propagating the names of deleted objects to all sites in the new view ensures that this condition holds for any object that was accessible in the old view.

It is not essential that the new connection vector be exactly the same as the new one. If the sites in the new view's component are a superset of those in the old one, then inheritance may still be used. Quorum assignments can then be extended to the new sites using the lightweight quorum change protocol. Thus, inheritance may be used to merge a small partition into the main network. If there are sites in the connection vector for the old view that are not in the new one, however, then using inheritance could result in transactions not being serializable in order of their view ids. Transactions should be serializable in the order of their view ids because our proof of correctness depends on this fact.

3.5 Extending a View

View extension can be used to avoid forming a new view when a site recovers from a failure. The connection vector for the current view is extended to include the recovered site. Then the lightweight protocol described in section 3.3 may be used to change the quorum assignments for those objects that have a copy at the recovered site.

To extend the current view, a recovering site carries out the following steps:

1. Copy the connection vector and view id for the current view from an operational site.
2. Send a join view request containing the view id to all sites in the view. If the current view id at a site receiving a request message agrees with the view id in the request, the site modifies its connection vector for the view and sends a positive acknowledgment. Otherwise, it returns the newer view id and corresponding connection vector.
3. If no site returns a newer view id, the view extension is completed. Otherwise, repeat step 2 using the newest view id received.

Because correctness of our method does not depend on the accuracy of a view's connection vector, the connection vector may be treated as a hint. Rather than using a separate protocol, we could even propagate changes in the connection vector along with lightweight quorum change requests. The one-phase protocol is not expensive, however, and should cut down on the disparity between the connection vectors at different sites. Because sites are only added and never deleted from the connection vector for a particular view, view extension may be used safely together with inheritance.

4 Prototype Implementation

As a first step in studying the behavior of our proposed algorithms, we carried out a prototype implementation of our quorum-based transaction processing and view formation protocols in an environment of a local-area network of Sun workstations. Our implementation was based on, but independent of, the distributed database system described in [7]. We adapted the existing software to incorporate support for partial replication and the use of views and quorums for replication control.

Transactions and view formations/extensions were run on five Sun 3/50's connected by a ten megabit/second Ethernet for configurations with three sites, five sites, and ten sites. The degree of data replication was set to two, three, and six, respectively (i.e., to approximately 0.6 times the number of sites). Active quorum assignments were set to

	T_{Norm}	T_{NV}	T_{LW}
3 sites	.48	.54	.64
5 sites	.51	.64	.57
10 sites	.69	.92	.74

Table 1: Execution time in seconds for read-write transactions

	Empty view formation	View extension
3 sites	.06	.06
5 sites	.10	.10
10 sites	.23	.20

Table 2: Execution time in seconds for view formation and extension

read-one write-all (in the current view) and backup quorum assignments were set to read-majority write-majority. Each user transaction accessed a single relation consisting of 100 tuples of length 50 bytes. A read operation read the entire relation, a write operation wrote 10 tuples. Each run of an experiment measured the elapsed time for execution of a single transaction, using the Sun clock chip with granularity of 20 msec. Data values were averaged over 20 runs to obtain 90 percent confidence intervals, calculated using the Student's distribution [12], of $\pm$ 0.05 sec for values under 0.40 sec and $\pm$ ten percent for values greater than or equal to 0.40 sec. In the tables of data, T_{Norm} denotes a normal transaction, T_{NV} a transaction that moves a relation to a new view, and T_{LW} a transaction that carries out lightweight quorum assignment change.

Normal transactions. Execution time for a transaction was measured at the coordinating site from the time the transaction was submitted for processing until the commit decision was reached. This time did not include the cost of interpreting the query nor of translating it to a transaction. Execution time for a normal read-write transaction in the absence of failures included a local read and phase one of the commit protocol. Phase one sends a round of approximately 1400-byte messages to the members of a write quorum, with each site of which does local logging. Times for normal transactions are shown in Table 1.

Empty view formation and moving a relation to a new view. Elapsed time for empty view formation included a round of approximately 60-byte messages to all sites, but no disk accesses. Times for empty view formation are shown in Table 2. Execution times for read-write transactions that move relations to a new view are shown in Table 1. These times are 20 to 40 percent greater than normal for the five- and ten-site cases, due to the need for two additional rounds of messages during the read phase to contact a backup read quorum and read an up-to-date copy of the relation. Additional rounds are not needed for the three-site case, however, since the degree of data replication for this case is two and backup read quorums are of size one.

View formation with inheritance. Execution times for view formation with inheritance are shown in Table 3. The increase with the number of deleted relations is due to the longer messages needed to contain the identifiers of deleted relations. Following view formation with inheritance, a transaction accessing a relation that had not been deleted from the old view took the same amount of time as in the absence of failures. A transaction accessing a relation that had been deleted took the same amount of time as if the transaction had followed empty view formation, because it had to move the relation to the new view.

View extension. Times for view extension are shown in Table 2. View extension took about the same amount of time as empty view formation. Fewer total messages are required for view extension, but the number and sizes of messages sent between when the timing starts and the view change is committed are the same in both cases.

Lightweight quorum assignment change. Execution times for transactions that carry out a lightweight quorum assignment changes following view extension are shown in Table 1. The increase over the normal execution time, about 10 to 20 percent for the five- and ten-site cases, is less than the additional time for a move to a new view. This is because only one additional round of messages is required in the read phase and only one site needs to be read from for lightweight change, compared to two additional rounds and reading from a backup read quorum for a move to a new view. The exception is the three-site case, where the degree of data replication is two. Our software was not smart enough to recognize in this case that, with a backup write quorum consisting of both sites having copies, the local copy of a relation was guaranteed to be up-to-date following view extension. Instead the transaction read at the remote site, making the lightweight change more expensive for the three-site case than a move to a new view.

	No deleted relations	5 deleted relations	10 deleted relations	50 deleted relations	100 deleted relations
3 sites	.08	.10	.10	.14	.16
5 sites	.10	.10	.12	.16	.18
10 sites	.22	.24	.25	.30	.34

Table 3: Execution time in seconds for view formation with inheritance

5 Analytical Performance Estimates

In this section, we evaluate different ways of handling new view formation and/or view extension following failures and subsequent repairs. We investigate how the relative performance of different techniques depends on the length of time between failure and repair. We have claimed that our techniques allow a replicated database system to adapt to the duration of a failure. In particular, we hypothesize that inheritance from the old view that existed prior to a failure will give good results if the failure is of short duration. For a longer failure, we expect that inheritance from the view for the larger component of the partitioned network (in the case of network partitioning) or extension of the current view (in the case of site recovery) will be better.

For our analytical model, we have made the following assumptions:

1. Copies of relations are distributed uniformly among the sites.
2. The access pattern of relations by transactions has a random uniform distribution.
3. Each transaction reads and writes a single relation out of a database of 100 relations.
4. Moves to a new view and lightweight quorum assignment changes are integrated with transaction processing.
5. Transaction throughput is limited by the rate at which servers can handle requests, rather than by network bandwidth. This assumption is based on the fact that high-bandwidth networks are becoming readily available.

The first two assumptions make the analytical model tractable. The third assumption allows us to give an explicit solution for an instance of the model. Below we discuss the implications when these three assumptions do not hold.

Our analytical model has the following parameters (Let T_{Norm} denote a normal transaction, T_{NV} a transaction that moves an object to a new view, and T_{LW} a transaction that carries out a lightweight quorum assignment change):

NVresp – the ratio of the response time for T_{NV} to the response time for T_{Norm}.

LWresp – the ratio of the response time for T_{LW} to the response time for T_{Norm}.

NVload – the ratio of the load on the Transaction Manager servers for T_{NV} to the load for T_{Norm}.

LWload – the ratio of the load on the Transaction Manager servers for T_{LW} to the load for T_{Norm}.

We estimated values for these parameters from measurements for the prototype implementation described in section 4. Using the response times measured for ten sites, rough estimates for *NVresp* and *LWresp* are 1.4 and 1.1, respectively. In our implementation, a normal transaction generates an average of 1.8 requests per Transaction Manager server, while a transaction that moves the relation to a new view generates an average of 2.4 requests per server. Hence, disregarding the lengths of the requests, a rough estimate for *NVload* is 1.3. The average number of requests per TM server for a transaction that does a lightweight quorum assignment change is 1.9 for a 10-site configuation, compared to 1.8 for a normal transaction. Thus, a rough estimate for *LWload* is 1.05.

Our calculations are based on the solution of a system of differential equations that describe how various quantities in the system change over time. These equations represent a continuous approximation to the discrete system. Experimental results in [5] give credence to a similar differential equation model for analysis of the fail-lock technique for doing site recovery.

The quantities solved for are the following:

move(t) – the number of relations remaining to be moved to a new view at time t following new view formation.

ltwt(t) – the number of relations still needing lightweight quorum assignment change at time t following new view formation or view extension.

Resp(t) – the ratio of average response time at time t following new view formation to normal average response time.

Th(t) – the ratio of throughput at time t following new view formation to normal throughput.

The quantities $Resp(t)$ and $Th(t)$ depend on one or both of $move(t)$ and $ltwt(t)$, depending on the type of view formation used. The equations and projected performance for different cases of failure and recovery are detailed in the following subsections.

5.1 Network partitioning failure and repair

Following a simple network partitioning, the network is divided into two components, and a new view has been formed for each component. If new view formation with reconfiguration has been used, all quorum assignments have been re-adjusted by the view formation transaction. If empty new view formation has been used, quorum assignments will be adjusted incrementally as relations are moved to the new view on demand by the transactions that access them.

Following empty view formation, the quantity $move_0(t)$ is the number of relations that remain to be moved to the new view at time t following the partitioning. Transactions will be committing at a rate $\lambda(t)$ which depends on the value of $move_0(t)$, because the additional overhead of moving relations to the new view will slow down the transaction processing rate. Given our assumption that relations are equally likely to be accessed, the transaction processing rate should be scaled by the fraction of relations that remain to be moved to obtain the rate of change of $move_0(t)$. Thus, a continuous approximation to the integer-valued quantity $move_0(t)$ is given by the solution to the following differential equation:

$$\frac{d\ move_0(t)}{dt} = \frac{-\lambda(t) move_0(t)}{dbsize}$$

where

$$\lambda(t) = Norm\lambda * Th(t)$$

is the transaction processing rate of the system at time t and

$$Th_0(t) = \frac{1}{(1 - \frac{move_0(t)}{dbsize}) + NVload * \frac{move_0(t)}{dbsize}}$$

is the ratio of throughput at time t to normal throughput. The initial condition for $move_0(t)$ is

$$move_0(0) = dbsize$$

The ratio of average response time at time t following new view formation to normal response time is given by the following equation:

$$Resp_0(t) = (1 - \frac{move_0(t)}{dbsize}) + NVresp * \frac{move_0(t)}{dbsize}$$

Although response time and throughput are normal following view formation with reconfiguration, successful transaction processing cannot resume until after the time required for reconfiguration. The time required for empty view formation is insignificant, and transaction processing can resume almost immediately following the failure. Then as relations are moved to the new view, response time and throughput gradually return to normal.

Repair of network partitioning. We compared the following ways of handling the repair of a network partitioning:

1. new view formation with reconfiguration,
2. empty new view formation,
3. view formation with inheritance from the old view that existed prior to the partitioning,
4. view formation with inheritance from the current view for the larger component.

For methods 3 and 4, we assumed that empty view formation was used following the network partitioning failure. Following 3, relations that were deleted from the old view while the network was partitioned need to be moved back. The number of such relations increases with the duration of the partitioning. Following 4, relations that have not yet been moved to the view for the larger component still need to be moved. Relations that have been moved to the view for that component but that have copies in the smaller component need to have their quorum assignments extended using lightweight quorum assignment change.

In the analysis that follows, we denote the average response time ratio for method i at time t by $Resp_i(t)$ and the throughput ratio for method i at time t by $Th_i(t)$. We let t_r denote the time of the repair. Expressions for $Resp_i(t)$ and $Th_i(t)$ are given in Tables 4 and 5, respectively.

i	$Resp_i(t)$
1	1
2	$(1 - \frac{move_2(t)}{dbsize}) + NVresp * \frac{move_2(t)}{dbsize}$
3	$(1 - \frac{move_3(t)}{dbsize}) + NVresp * \frac{move_3(t)}{dbsize}$
4	$(1 - \frac{move_4(t)+ltwt_4(t)}{dbsize}) + LWresp * \frac{ltwt_4(t)}{dbsize} + NVresp * \frac{move_4(t)}{dbsize}$

Table 4: Response time ratios at time t, for $t \geq t_r$

i	$Th_i(t)$
1	1
2	$1/((1 - \frac{move_2(t)}{dbsize}) + NVload * \frac{move_2(t)}{dbsize})$
3	$1/(\frac{1-move_3(t)}{dbsize} + NVload * \frac{move_3(t)}{dbsize})$
4	$1/((1 - \frac{move_4(t)+ltwt_4(t)}{dbsize}) + LWload * \frac{ltwt_4(t)}{dbsize} + NVload * \frac{move_4(t)}{dbsize})$

Table 5: Throughput ratios at time t, for $t \geq t_r$

For methods 2, 3, and 4, the differential equation for $move_i(t)$, for $t \geq t_r$, is

$$\frac{d\ move_i(t)}{dt} = \frac{-\lambda(t) move_i(t)}{dbsize}$$

The initial condition for method 2 is $move_2(t_r) = dbsize$. For method 3, the initial condition is $move_3(t_r) = dbsize - move_0(t_r)$. For method 4, the differential equation for $ltwt_r(t)$, the number of relations still needing lightweight quorum assignment change at time t, is

$$\frac{d\ ltwt_4(t)}{dt} = \frac{-\lambda(t)\ ltwt_4(t)}{dbsize}$$

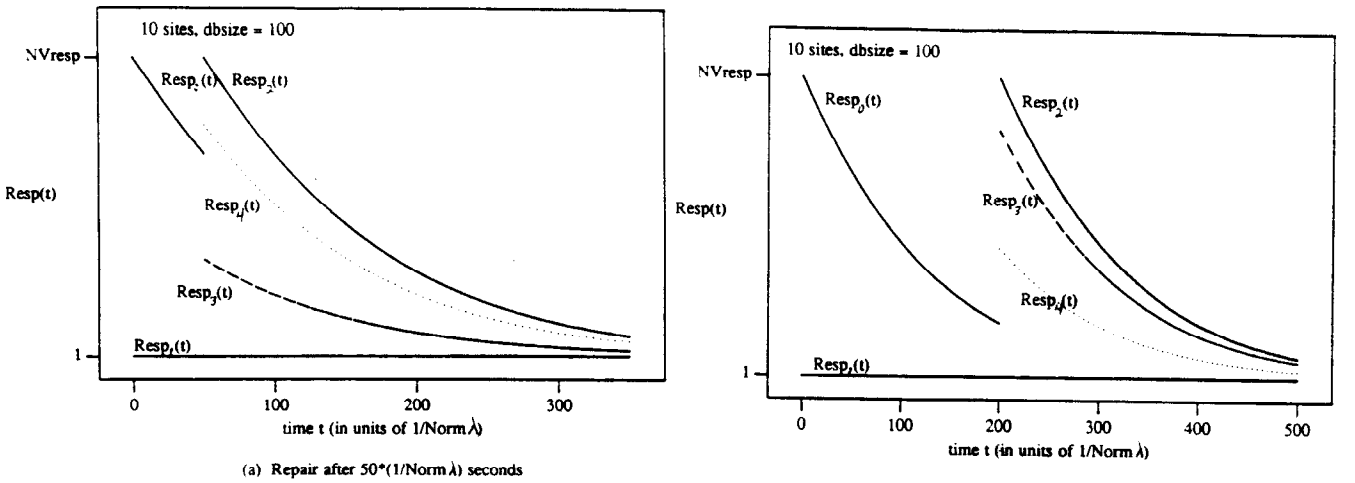
Based on the assumption of a uniform distribution of relations among the sites, we calculated values for the initial conditions $move_4(t_r)$ and $ltwt_4(t_r)$. Details of how $move_4(t_r)$ and $ltwt_4(t_r)$ were calculated may be found in [8].

Comparison of the different methods. We solved the above equations numerically with $dbsize = 100$ and $nsites = 10$ to obtain the curves in Figures 1 and 2 that show the calculated average response time and throughput ratios following the different methods of handling repair of the partitioning for different durations of the partitioning. For the 10-site case shown, the sizes of the two partitioned components were 4 and 6 sites. We used values for $NVresp$, $LWresp$, $NVload$, and $LWload$ as suggested by the preliminary experimental data reported in section 4.

The curves for view formation with reconfiguration (labeled $Resp_1(t)$ in Figure 1 and $Th_1(t)$ in Figure 2) illustrate the performance achieved by the methods in [1] and [10]. The curves for empty view formation (labeled $Resp_2(t)$ in Figure 1 and $Th_2(t)$ in Figure 2) illustrate the performance achieved by the method in [11].

The results of our preliminary experiments and analysis show that, under our assumptions of random access to objects and uniform distribution of objects among the sites, a significant improvement in transaction processing performance during the recovery period can be achieved by inheriting quorum assignments from a previous view. For a failure that is of short duration, Figures 1(a) and 2(a) illustrate saving of greater than 50 percent of the average overhead for moving objects to a new view, when inheritance from the pre-failure view is used. For a failure of longer duration, Figures 1(b) and 2(b) illustrate savings of close to 50 percent, when inheritance from the larger failure view is used.

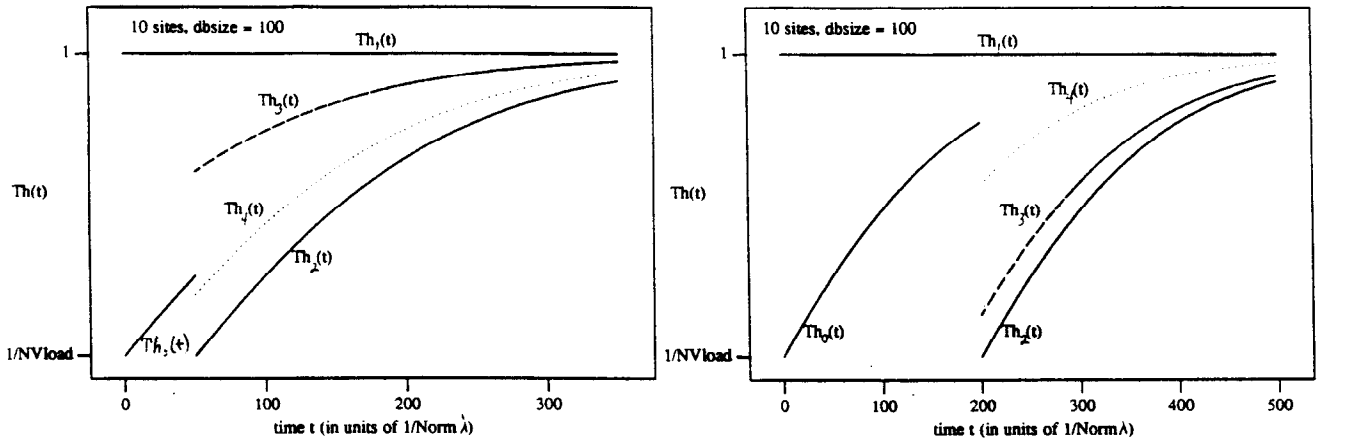
The analysis that produced Figures 1 and 2 is based on equal percentages of read and write operations. With a higher proportion of read operations, inheritance would achieve even greater savings by reducing the number of times read-only access must be converted to read-write access. The database size of 100 relations used in our analysis is fairly small. For a constant transaction processing rate, increasing the database size would increase the savings achieved by inheritance from the old pre-failure view, because the relative proportion of objects needing to be moved back to the old view would be smaller. We have assumed randon access to database objects. With an access pattern that exhibits high locality of reference, the savings achieved by inheritance from the



(a) Repair after 50*(1/Norm λ) seconds

(b) Repair after 200*(1/Norm λ) seconds

Figure 1: Comparison of response time for different methods of handling repair of network partitioning



(a) Repair after 50*(1/Norm λ) seconds

(b) Repair after 200*(1/Norm λ) seconds

Figure 2: Comparison of throughput for different methods of handling repair of network partitioning

pre-failure view would again be greater, for the same reason as for an increase in database size. The effect of a non-uniform distribution of relations among the sites is unknown and remains to be investigated.

5.2 Site recovery

We compared the following alternatives for handling the site recovery process:

1. new view formation with reconfiguration,
2. empty new view formation,
3. view formation with inheritance from the old view that existed prior to the site failure,
4. extension of the current view to include the recovering site.

As an alternative to method 4, inheritance from the current view would also be possible, but view extension is cheaper since fewer messages are sent. View extension also has the advantage that transactions in progress concurrently will not be aborted because of a change in view id. Hence, extending the current view seems clearly preferable to inheriting from it.

Using an analysis similar to that in section 5.1, we observed the same relationship between the methods for site recovery as for the repair of network partitioning. Details of this analysis may be found in [8].

6 Conclusions

We have investigated techniques for adapting the recovery actions of a dynamic quorum method to the length and extent of site failures and network partitioning. The major contributions of this research are the following:

- We have proposed a new recovery technique called inheritance that allows restoration of a previous configuration of quorum assignments with a minimal amount of work. Inheritance should be useful for recovery from a short-lived single site failure or simple partitioning, or from multiple site failures if all are repaired fairly quickly. For example, inheritance would be applicable to a redundant system with backup components that may be brought on-line quickly. For longer failures, inheritance allows the system to acquire quorum assignments from the largest current configuration, with the additional work required depending on the number of sites outside this configuration.
- For the special case of site recovery, we allow extension of the current view to include the recovered site. Such view extension is expected to provide efficient recovery from lengthy site failures.
- We have narrowed down the need for formation of a new view to the case where no active quorum is available for an operation. Quorum assignment changes invoked for other reasons, such as addition or deletion of copies or the restructuring of backups quorums, may be done without forming a new view.

We have implemented our dynamic quorum method in a distributed system collected preliminary experimental data. More work is needed to validate the experimental model.

Algorithms are needed that take as input all the information available about previous and current views and determine whether an old or current view should be used for inheritance or extension. Rather than making simplifying assumptions about the access patterrn and processing rate of transaction that run during the failures, as we did in our analyses, these algorithms should use actual transaction execution histories, or summaries thereof, to do a more accurate comparison of the different options for recovery.

The analyses for cases of multiple partitioning (when the network is partitioned into more than two components) and multiple site failures remain to be done. In these cases, repair may be partial, in that proper subsets of the network may recover, and perhaps be subject to further failures, before the entire network is reconnected. We expect that the analyses for these cases will be more complicated, but use will use similar techniques.

References

[1] A. E. Abbadi and S. Toueg. Maintaining availability in partitioned replicated databases. *ACM Trans. Database Syst.*, 14(2):264–290, June 1989.

[2] N. A. Alexandridis. Adaptable software and hardware: Problems and solutions. *IEEE Computer*, 19(2), Feb. 1986.

[3] A. Avizienis. Fault-tolerant systems. *IEEE Transactions on Computers*, C-25(12):1304–1312, Dec. 1976.

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

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

[6] B. Bhargava and J. Riedl. A model for adaptable systems for transaction processing. *IEEE Trans. on Knowledge and Data Engineering*, 1(4):433–449, Dec. 1989.

[7] B. Bhargava and J. Riedl. The RAID distributed database system. *IEEE Trans. on Software Engineering*, SE-15(6):726–736, June 1989.

[8] S. Browne. *Quorum-based Recovery in Replicated Database Systems.* PhD thesis, Purdue University, May 1990.

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

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

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

[12] A. M. Law and W. D. Kelton. *Simulation Modeling and Analysis*, chapter 4. McGraw-Hill Book Company, 1982.

[13] D. Skeen. A quorum-based commit protocol. In *Proc. 6th Berkeley Workshop*, pages 69–80, Feb. 1982.