Nordic Journal of Computing 1(1994), 214–230.

A VIEW-BASED DYNAMIC REPLICATION CONTROL ALGORITHM

SUSHIL JAJODIA*
Department of Information and Software Systems Engineering
George Mason University
Fairfax, Virginia 22030
U.S.A.
jajodia@gmu.edu

RAVI MUKKAMALA
Department of Computer Science
Old Dominion University
Norfolk, Virginia 23529
U.S.A.
mukka@cs.odu.edu

K.V.S. RAMARAO
SBC Technology Resources, Inc.
550 Maryville Center Drive
St. Louis, Missouri 63141
U.S.A.
ram_ramarao@trigate.sbc.com

Abstract. Many algorithms exist in literature to manage replicated database objects. Some of these are *dynamic* and attempt to adapt to changing network configurations due to failures, particularly due to *network partitioning*. This paper presents a new dynamic algorithm for replication control with several desirable features: not only does it enhance the availability of read and write operations in failure conditions but also achieves this at a relatively low cost. This algorithm is based on the concept of *views* introduced originally by El Abbadi, Skeen and Cristian and improves on the dynamic voting ideas developed by many authors.

ACM CCS Categories and Subject Descriptors: D.4.3, H.2.4

1. Introduction

Replication is a well-known technique for improving the availability of objects in distributed operating systems and in distributed database systems: more than one copy of an object is stored in the system so that the object may be available in spite of failure of some sites. Of course, the physical availability of a copy does not necessarily guarantee that transactions can use it for reading/writing, owing to the consistency requirements. *Replica control* algorithms attempt to maintain mutual consistency among copies of an object. While it is relatively easy to design replica control algorithms that guarantee mutual consistency when no failures are allowed in the model,

*The work of S. Jajodia was supported in part by the Air Force Office of Scientific Research under grant # 90-0135.

Received November 1993. Accepted February 1994.

algorithms tolerant to failure are harder to design and analyze. Generally two types of failures are considered: site failures and network partitioning. Site failures are assumed to be benign: failed sites simply stop executing their protocol and do not exhibit any malicious behavior. Network partitioning occurs when the network gets fragmented into non-communicating groups of operational sites. Arbitrary, undetected network partitions are much harder to handle than site failures and thus algorithms handling both site failures and network partitioning are significantly more complex than failure-sensitive algorithms [8].

Two general approaches are taken in the literature to deal with network partitioning: the pessimistic and the optimistic approaches. The algorithms based on the pessimistic approach (e.g., [2, 3, 7, 12, 13, 18]) share the philosophy that mutual consistency is of greater importance than availability. Such a philosophy is appropriate in situations where temporary inconsistencies among replicated objects cannot be tolerated or where semantics of operations on objects are complex and inconsistencies in the copies of an object are difficult to detect and resolve. In contrast, the optimistic algorithms (e.g., [5, 8, 15, 16]) take the approach that the database must be available even when network partitions. Conflicting transactions are detected and rolled back when partitions are reunited. Since this is a very difficult task, these algorithms are useful in situations in which the number of database objects is large and the probability of conflicts is small (hence the label optimistic).

The work reported in this paper falls under the banner of the pessimistic approach. Thus, at most one group of a partitioned network will be allowed to continue with transaction processing under the replica control algorithms presented here. But what is new is that it improves the chances that one of the groups can in fact do some useful work by adapting closely to the topology of the network in transition towards a stable configuration. Simulatneously, it reduces the overhead due to its own execution by tightly controlling the cost of the adaptation.

The organization of the paper is as follows: next section motivates the derivation of the new algorithm by describing in detail two existing algorithms that form the basis for the new algorithm and pointing out their strengths and weaknesses. Section 3 presents and proves the correctness of the new algorithm. Finally, Section 4 concludes the paper.

2. Motivation for the New Algorithm

For the sake of simplicity and without loss of generality, we make the following assumptions:

- there is a single object x in the system,
- there are n sites in the system that have copies of x,
- arbitrary site failures and network partitioning are allowed, and
- the failures are only benign.

Following is the general structure of most pessimistic replica control algorithms. When failures are detected or suspected, some sites attempt to determine the composition of the groups that they belong to.[1] Then they try to ascertain whether or not their group is the *distinguished group* that can proceed with normal transaction processing. If the group is identified as the distinguished group, then reads and writes on the object can take place within that group. If on the other hand it is not a distinguished group, then the sites simply wait for a reconfiguration and the procedure is repeated.

Thus there are basically four issues that a replica control algorithm addresses and should provide answers for:

(a) how do sites determine the composition of their group,

(b) what makes a distinguished group,

(c) how do sites check if they form a distinguished group, and

(d) how are reads and writes performed in a distinguished group.

Of the four, (b) is the most critical issue and what most distinguishes different algorithms. Observe that the answer to this issue forms the crux of a replica control algorithm since it ensures that more than one distinguished group cannot coexist and thus enforces a serial order among the distinguished groups forming during the period of network transition which in turn enforces the mutual consistency when proper rules are devised. Issues (a) and (c) are relatively simple and in fact are answered identically by many algorithms. Issue (d) directly depends on the answer to issue (b), as explained above.

For example, consider the primary copy approach where one of the copies of the object is identified as the primary copy and all writes are first performed on it [4]. Under this replica control algorithm, any group that contains the site with the primary copy is distinguished. Clearly then there can be at most one such group in the system at any time. Operations in such a group are performed by writing to the primary copy first and then propagating the new value to the other copies and reading from any of the copies when stale values are acceptable and from the primary copy when the most recent value is needed.

A second example is the majority approach [9, 19]. Here, a group is distinguished if it has a majority of copies of the object. Under this approach, each copy of the object usually has an associated indicator representing how recent the value is. A fundamental property of the indicator is that its values are distinct and totally ordered. Examples of indicators are global time-stamps and version numbers. In a distinguished group, the most recent copy of the object is identified and read (this is possible due to the distinctness and total ordering of the indicator values) and a write is performed on all copies in the group, and the indicator is properly recorded atomically with the write.

[1] Since the failures are in general unpredictable and are also usually temporary, a site's belief on the composition of the group it is in does not have to reflect the reality. Nor do all sites in a group have to believe the same composition.

Examples presented above represent *static* algorithms that behave the same independent of the network configuration. There are several dynamic algorithms that have appeared recently in the literature. Of these, *dynamic voting with linearly ordered copies*, hereafter referred to as *dynamic-linear*, by Jajodia and Mutchler [12] and *accessibility thresholds protocol* by El Abbadi and Toueg [3] are particularly attractive. Each enjoys many virtues not found in other dynamic schemes; however, each suffers from some deficiencies as well. The present work introduces a new dynamic algorithm which attempts to retain the virtues of both algorithms, eliminating their disadvantages at the same time. More specifically,

- Like dynamic-linear, the write quorum for an object x in the new algorithm depends on the number of up-to-date copies in existence at the time of the update. On the other hand, the write quorum for x in accessibility thresholds protocol cannot be less than a value determined by the total number of copies of x (i.e., the write threshold of x).
- Like accessibility thresholds protocol, the new algorithm utilizes the view mechanism which permits a site to determine if it can read or write an object by consulting the view information available locally. Without such a mechanism, dynamic-linear requires much message passing to accomplish this.
- Like dynamic-linear, the new algorithm allows more flexibility in the selection of distinguished partitions than the accessibility thresholds protocol, resulting in greater read and write availability.

We now describe these two algorithms.

2.1 The Dynamic Linear Algorithm (DLA)

In this section, we provide an informal description of the dynamic-linear algorithm. Each copy under this algorithm has three variables (in addition to the value of the object) associated to it: a *version number (VN),* an *update sites cardinality (SC),* and a *distinguished site (p).* A group is distinguished if the number of sites in the group with the most recent version number VN is a majority of the corresponding SC if SC is odd; and is $SC/2$ if it includes the distinguished site and SC is even. The SC of the most recent version number in a distinguished group is also referred to as the group's update sites cardinality.

A site receiving a request (read or write), first determines if it is in a distinguished group. For this, it sends messages to all other sites, receives responses and applying the above criterion checks if they form a distinguished group. If they do then the request is serviced and all sites in the group are informed. What is more important and interesting is that a distinguished group can decide the criterion for the next distinguished group. For instance, it can install a new (higher) version number on a proper subset of the sites in the group and set SC and p accordingly. Or, when a write

is performed, a new version number and the corresponding SC and p may be installed. Since this is done atomically at all sites in the group, it is guaranteed that at most one distinguished group exists at any time.

Read and write quorums[2] can be incorporated easily within each distinguished group as follows. For a distinguished group P with SC as the update sites cardinality, the read quorum $q_r[P]$ and the write quorum $q_w[P]$ can be defined independently, and must satisfy the following equations:

$$q_r[P] + q_w[P] > SC \quad (1)$$

$$2q_w[P] > SC \quad (2)$$

Dynamic-linear has several very desirable properties:

- it has a simple statement that permits a clear correctness proof,
- it is easy to implement, and
- the availability afforded by the DLA is greater than the availability due to any static algorithm if the object is replicated at four or more sites in the network [12].

Furthermore, enough expertise can be imparted into an implementation of the algorithm to make its behavior truly reflective of the network configuration. Thus the algorithm can be effectively dynamic: directly dependent on the initial network topology as well as any factors that effect the availability of the system.

Conversely, there are several areas of DLA that warrant improvement too:

- Each request has a large message cost associated with it since the criterion for a distinguished group is asserted with each request. Thus, many messages may be wasteful during a transitional period.
- A two-phase protocol is employed in servicing requests: all sites receiving a message from a site attempting to service a request lock their copies of the object before responding and hold the locks until they receive another message from the site saying whether they are in a distinguished group (in which case the request is serviced and the effects are committed at all sites) or not (in which case the locks are released without any effect on the object). Clearly, in addition to the message overhead, this requirement potentially reduces the availability of the object.

We now describe the accessibility threshold algorithm which uses views that can be formed in a single phase.

2.2 The Accessibility Thresholds Algorithm (ATA)

In this section, we briefly describe the accessibility thresholds algorithm [3] which is a view-based algorithm. Under this algorithm, each site s maintains a list variable v_s known as the view of s. At any time, v_s lists the set of

[2] These specify the number of copies that must be read or written within a group.

sites that s believes to be in the same group as s at that time. Associated to an object are two integers: read accessibility threshold A_r and write accessibility threshold A_w. In addition each object in a view is associated with a read quorum $q_r[v_s]$ and a write quorum $q_w[v_s]$.[3] Let $n[v_s]$ be the number of copies of the object in the view v_s. The following relationships dictate bounds on the values of these variables. (In [1] and [2], the write accessibility threshold was bound by an additional constraint $2A_w > n$. This is removed in [3].)

$$A_r + A_w > n \tag{3}$$

$$q_r[v_s] + q_w[v_s] > n[v_s] \tag{4}$$

$$2q_w[v_s] > n[v_s] \tag{5}$$

$$1 \leq q_r[v_s] \leq n[v_s] \tag{6}$$

$$A_w \leq q_w[v_s] \leq n[v_s] \tag{7}$$

As in the case of the weighted voting approach [6, 9], there can be separate distinguished groups for reads and writes under ATA: a distinguished view for a read is one that has at least A_r copies in it and a distinguished view for a write is one with at least A_w copies in it. In a distinguished view v_s for a read, $q_r[v_s]$ copies are read and the most recent version among them is taken as the value. Similarly, $q_w[v_s]$ copies are atomically updated in a distinguished view for a write. Creation of a new view (corresponding to issue (a) listed in the motivation section) can be initiated by any site at any time, one possibility being when it detects that it cannot communicate with a site in its current view. View numbers (or identifiers) must be totally ordered globally and thus a site initiating a new view chooses a view number larger than any it has known previously and the set of sites it believes to constitute the new view. It then accesses A_r copies of the object (in general, as many copies as are required by the accessibility threshold for each object it has a copy of) from sites within the new view and updates its local copy to the most recent among the A_r read.[4] The messages sent to read the copies also double as invitations to other sites to join the new view. A site receiving such a message may choose to accept the invitation if the proposed view number is larger than any it has known (and/or several other possible factors, as discussed in [3]). In case it decides to accept the invitation, it updates the local copy of the object by accessing the appropriate number of copies within the new view and installs the new view.

There are at least three features of ATA that are highly impressive:

- The view formation can be achieved in a single phase: a site installing a new view (either by initiating the new view or after receiving an invitation from another site) does so at its own pace and convenience. No synchronization across sites is necessary.

[3] Values of these quorums depend upon the objects but we do not explicitly show that dependence here since we assume a single object.

[4] The installation of the view and the updates of all copies form a locally atomic operation.

- A site s where a transaction request to read/write arrives can, based on the locally available list v_s, makes a preliminary decision on whether it can service the request or not. In other words, determining whether or not the group a site (thinks) is in a distinguished group for a given operation can be done locally with no message passing. Message passing is required only if it believes that the request can be serviced. Thus, if the view of the site coincides or closely approximates the reality, then none of the messages used in an attempt to service a request are wasted.
- The quorums for read/write operations within a view can be dynamically determined by the members of the view. No global decision-making involving sites outside the view is needed.

Note that the dynamic aspect of ATA is in the ability to choose quorums dynamically within each view. The distinguished groups themselves are statically determined through the values of A_w and A_r. Clearly, this is an area for a possible improvement. There is another somewhat subtle but more serious issue about ATA that warrants improvement. Following proposition brings this issue out.

PROPOSITION 1. *Under the ATA, if an object has read and write access thresholds in a view v_s, then the write quorum in v_s must satisfy the following inequality:*

$$q_w[v_s] \geq \left\lceil \frac{n+3}{4} \right\rceil \tag{8}$$

PROOF. Since x is updatable in v_s,

$$n[v_s] \geq \max\{A_r, A_w\} \tag{9}$$

From Equations (3) and (9),

$$n[v_s] \geq \left\lceil \frac{n+1}{2} \right\rceil \tag{10}$$

From Equation (5),

$$q_w[v_s] \geq \left\lceil \frac{n[v_s]+1}{2} \right\rceil \tag{11}$$

The assertion now clearly follows from (10) and (11). □

Therefore, even if the write accessibility threshold A_w is quite small, the write quorum $q_w[v_s]$ cannot be less than a value determined by n, the total number of copies of the object. As an example, if the object is replicated at 100 ($=n$) sites, to write the object (in a view v), 26 or more (when $q_w[v_s] > 26$) physical copies will need to be accessed.

In the next section we develop a new algorithm that combines the virtues of both the algorithms described above while trying to avoid the pitfalls to the extent possible.

3. The New Algorithm

The replication control algorithm proposed in this section attempts to combine the advantages of both the dynamic and the view-based algorithms. More specifically, the objectives of the algorithm are as follows:

(1) It must not be necessary to exchange any messages in a relatively stable partition to determine the accessibility of an object.

(2) The number of copies needed in a partition to update an object *must not* depend on the total number of copies of the object (making the algorithm truly dynamic).

(3) Only single phase protocols must be run whenever possible, to guarantee the sustained availability of resources.

The first objective is realized by using the view mechanism in the groups while the second is realized by using ideas from dynamic-linear. Finally, effort is made to minimize the need for a two-phase protocol by eliminating the need for synchronization among sites whenever possible.

3.1 View Creation/Installation

The notion of a view used here is the same as that of [3]. Thus it is a logical partition (i.e., a set of sites) as viewed by some sites. A site s with a view v_s believes that it is in a group consisting of sites listed in v_s. This may or may not be the reality. But if a group is relatively stable and the number of read/write operations in the partition is sufficiently high, then the views of the sites in the partition can be expected to be identical to the physical partition. The process of view creation and installation in our algorithm is more flexible than that of [3], as we shall see below. In our algorithm also, any site can initiate the creation of a new view whenever it chooses to do so. A site s initiating a new view sends creation requests to some sites (determined in any way - all sites being an extreme case). A creation request consists of the following: a proposed view number that is larger than its current view number, and the proposed view (i.e., the list of sites). A site receiving a creation request either responds with a *reject* or does not respond at all if its own current view number is larger than the proposed view number. If on the other hand its current view number is smaller, then it may either choose to *reject* or to *accept* or not to respond.[5]

The initiating site may thus receive some *reject*s and some *accept*s in response to its creation requests. In fact many of the sites it has sent the requests to may not even respond (either because they chose not to or due to failures). Thus the initiating site uses a (local) time-out mechanism to decide when to make a decision on installing the view.[6] At that time, it

[5] When it chooses to accept, it will have to send the value of the object and its version number. We shall discuss this issue in detail in section 3.2.

[6] While a site can in principle create a new view consisting of any set of sites, it is not desirable that the view does not help to process any requests. Thus, as a practical matter,

records the new view as consisting of itself and the sites that have accepted its request. Then it sends that view to all the sites that have accepted it. Notice that theses two actions (local recording of a view and transmitting this information) do not have to be atomic (thus, there may be a view known only to the initiator). But since the intent of creating views is to make all sites in a stable partition to converge towards identical views, it is *desirable* that all sites accepting a view are notified as soon as possible on the composition of that view. A site receiving the new view installs it locally provided it has not already installed a view with a larger view number. In the latter case, it may choose to initiate a new view. But a site does not treat itself as being in a new view until it knows the view's composition.

3.2 Updates to Objects at View Creation

In addition to having the same view, the sites in a stable group should have the most up-to-date information on the values of the object in that view. To this end, we first introduce a concept needed for applying the ideas from the DLA: we associate with each view v_s a write access set $WA[v_s]$. The next section defines and shows how it evolves but for now it is sufficient to say that such a variable exists.

The site initiating a view requests the version numbers and the write access sets (WA) from each of the sites. Notice that the sites could send this information as part of the *accept* messages themselves. We have separated out these two aspects since they are logically distinct functions. The value with the maximum version number is taken and the local copy is updated. The corresponding WA is also recorded atomically with this update. Then these values are passed on to all the other sites in the view. Notice again that it is possible that even within a view the information associated with a view could be different at different sites. (Once again, no atomicity requirement is imposed across the sites in a view.) But if they can communicate physically, then eventually they will have identical values.

Finally, each view has to determine the read and write quorums for its object. This is the critical information to be used by a site to make a preliminary assessment of whether or not an object can be read/updated in the view. Thus consistency is clearly needed on this information. It is not relevant *how* the quorums are chosen. Any of a multitude of possible strategies can be used. But it is important that all sites that *believe* to be in a particular view have the same quorum information. Thus we have two options:

(i) atomically implement the two operations - installation of a view and recording the quorums for each object, and

(ii) allow the possibility of a site having many views, where a set of objects is associated with a view.

a site must refrain from creating views in which no requests can be serviced. Later we discuss when the requests can be serviced in a view. Those criteria can be applied at the view creation time itself to decide whether or not to create a new view.

In the second option, some objects may have newer views than others. Again, we choose the second option, following the objective (3) listed at the beginning of Section 3. It is noted though that choosing the first option may reduce the complexity of the replication control algorithm while the cost of a commit protocol to implement atomicity will be incurred.

The following points are consequences of the above decision:

- each object (copy) at each site has an associated view number and version number, and
- different copies of an object within the same view may have different view numbers at a given time.

This flexibility we have allowed during the view creation/installation time implies that we must be careful in installing reads/writes. These issues are treated next.

3.3 Update Algorithm

First we define the concept of write access sets introduced in the previous section. Since the concept is expected to facilitate the dynamic determination of distinguished views (or groups), it is natural that it directly relate to the update algorithm, as we shall see now. When a view v'_s is created from a view v_s (that is, the site s has a view v_s immediately before installing the view v'_s) then the initial value of $WA(v'_s)$ is the set of all sites in the view v_s. This value of $WA(v'_s)$ is retained until the successful execution of the first write operation in v'_s. At the end of the first successful write operation, $WA(v'_s)$ is changed to the set of sites in v'_s. Thus, our algorithm explicitly distinguishes between the first write and the later writes in a view.

Now consider the first write in a view v_s. At least a majority of sites in the write access set $WA(v_s)$ are accessed for this purpose. This may be implemented as follows: the locally available view information is first used as an approximation to predict the chances of success of the update in the view v_s (in other words, to determine whether or not the current group is a distinguished group); a choice of whether or not to proceed with contacting the other sites can be made based on the predicted chance of success. Recall that the most recent value of the object is determined at the view installation time and that each site installing the new view installs the most recent value as well. Thus the site receiving the write request invites other sites in the view to update the value of the object to the new value. The largest version number among them is incremented and used as the new version number. The write operation is deemed successful only if at least a majority of WA sites participate in the update (this is to be implemented atomically). In other words, the view is considered distinguished in this case. During the write, the WA set is modified to the set of copies in view v_s at each of the sites making the update (also atomically with the update). Observe that it does not matter which sites in view v_s has initiated the update.

The subsequent updates access any sites forming a write quorum in the view. The most recent copy is determined as before through the maximum

version number, that the version number is incremented, and the new value and the new version number are written at at least a write-quorum number of copies atomically. WA remains unaltered.

3.4 Read Algorithm

A transaction may request to read any object available in a view v_s. Since the reads usually far out-number updates in most practical environments, we intend to make reading as efficient as possible. In order to execute a read request in view v_s, at least a read quorum in v_s is to be accessed. (Unlike in the case of the first write, there is no need to use the view information here.) The copy with the largest version number among them is selected as the value of the object. Observe that the first read in a view needs no special treatment since all copies are updated to the most recent version as part of the view installation process itself.

3.5 View Creation revisited

During the discussion on view creation, we mentioned that a site should create a new view only if it is possible to perform some useful transaction processing in the new view. From the above description of update and read algorithms, it is clear that the write operation can be performed only if at least a majority of sites from the current distinguished view can be obtained from a new view. On the other hand, read-only transactions willing to use potentially stale data do not require such stringent criteria. It is sufficient in this case that a (read-only) transaction completely execute in the same view.

3.6 Correctness of the algorithm

Observe that the responsibility of a replication control algorithm is to order the accesses (reads and updates) to a single replicated object *properly* while the concurrency control mechanism is expected to order the transactions accessing several objects *properly*. Thus we shall restrict ourselves to proving that the replication control algorithm orders the accesses to any replicated data object correctly. Specifically, we show the following:

(a) any *update* to an object x is made to the (globally) *most recent* copy, and

(b) reads are properly ordered with respect to updates in that a read r from an update w precedes all updates *after* w (in the order given by (a)) and succeeds all updates preceding w (in the same order).

It is not difficult to show that any *standard* concurrency control mechanism on non-replicated databases (i.e., one that guarantees the acyclicity of the serializability graphs) together with our replication control algorithms produces only 1-copy serializable executions, once these two assertions are proven. In fact, this is the essence of the rather involved proof of [3]. Since

filling in those details would be almost exactly like in [3], we shall omit them and refer the interested reader to the original work.

PROPOSITION 2. *There is a total order among all updates made to x.*

PROOF. Observe that

(a) there is a total order among all updates (to x) within a view v, for any v, and

(b) there is a total order among all view numbers.

We show that these two together form the total order among all updates to x. Let v, v' be two views created from the same view u such that there is an update in u and let $v < v'$. Assume further that x is updated in both v and v'. Let w be the last update in v and w' the first update in v'. We show that w must precede w'.

Let w'' be the first update in v. By the algorithm, w'' must have updated x at a majority of sites in u. Thus v must have at least a majority of sites from u. Similarly, the write access set before w' is identical to u. Thus w' must have updated a majority of sites in u too. This could not have happened unless one of v, v' follows the other since the updates are implemented atomically. Furthermore, v could not have followed v' since the sites common to v and v' would not accept a view number less than their current view number. Hence v' must follow v. The same argument applies even if no updates were made in u except that the write access set then is the sites in a view that has most recently updated x before u. Similarly, if there is no update in one of v, v' then again the descendent of (say v) where an update is made takes the place of v in the above argument.

Thus v' must have been derived from v, *after* the first update in v. But then the write access set of v' before any update takes place in v' is v, not u (since the write access set is updated in v atomically with the update to x and its value after w'' is v). On the other hand, the write access set in v at the time of w (which is a non-first update) is also v (recall that the write access set is changed only once in an updating view, at the time of first update). Then again w, w' cannot take place concurrently and hence there must be an order among them. Again since $v < v'$, w must precede w'. □

PROPOSITION 3. *Let r be a read (of x) from the update w. Then there is an ordering in which r precedes all updates after w in the total order given in Proposition 2 and succeeds all updates before w in the same order.*

PROOF. Let w' be an update to x that *precedes* w. If r, w, w' are all in the same view, then it is obvious that w' must precede r (all reads are from the most recent update within a view that makes updates, by the algorithm's requirement that a read quorum be read and the value with the highest number be chosen). Assume that r, w are in the same view v but w' is in v' for $v' \neq v$. Then clearly view-number(v') < view-number(v) and r cannot

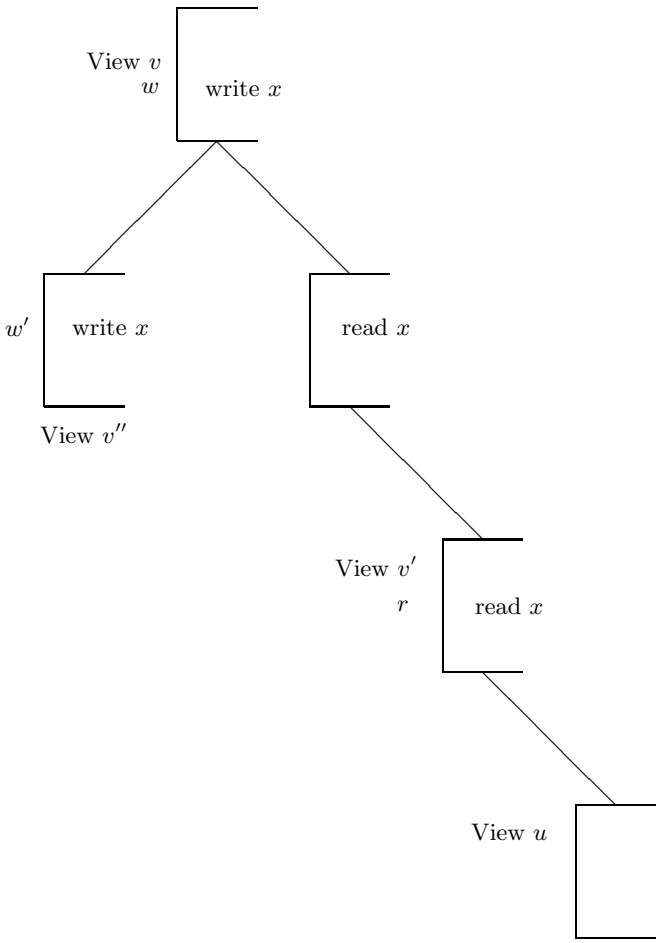

Fig. 1: Illustration for Proposition 3 when w' succeeds w

precede w'. same argument holds even if r, w are in different views. Now assume that w' is an update succeeding w. Then again the assertion follows immediately if w, w' are in the same view. Assume that r, w, w' are all in different views (it is easier if r, w are in the same view) and that w' is in the view v''. Assume for simplicity but without loss of generality that w' immediately succeeds w (that is, there is no update *between* w, w').

The assertion is immediate if no descendent view of v' ever updates x. Assume that some descendent view of v', say u (not necessarily distinct from v'), updates x. By Proposition 2, and the assumption that there is no update between w and w', u must be a descendent of v''. Again by Proposition 2, this means that w' precedes that update in u. But since r

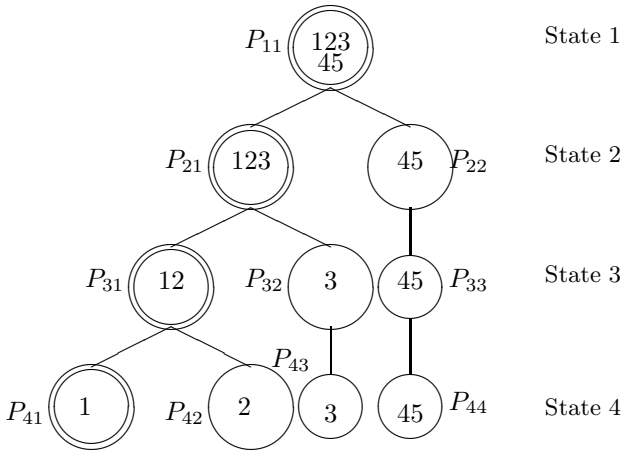

Fig. 2: Illustration of the Network Partitions

reads from w and not w', this implies that logically the view v' precedes the view v'' and hence r precedes w'. (It may be useful to imagine that v'' and v' have some common descendent u' where the first update after w' is made; then logically, all views in the path from v' to u', excluding u', precede v''). □

THEOREM 1. *Our replication control algorithm produces only one-copy serializable executions.*

PROOF. Follows from propositions 2 and 3. □

3.7 Illustration

We now illustrate through an example how the proposed algorithm achieves greater availability and efficiency than either of the two algorithms [3, 12].

Let us consider an object x with 5 copies in a database system. Let us refer to them as 1,2,3,4, and 5. Initially, all these copies (i.e., sites that contain them) can communicate with each other. This is referred to as group G_{11} in Figure 2. This state of the network is also referred to as State 1 in this figure. Due to a network partition the five copies are split into two non-communicating groups P_{21} and P_{22}. This is referred to as State 2. The other two states (States 3 and 4) can be similarly explained.

In this figure a circle refers to a physical partition. Similarly, a double circle refers to a *distinguished group*.

In the accessibility thresholds algorithm, the availability of x is determined by the access thresholds (A_r and A_w) and the read and write quorums

TABLE I: Availability table for the Access Thresholds Algorithm

Acc. Thresh./ Quorums	State 1		State 2		State 3		State 4	
	Read	Write	Read	Write	Read	Write	Read	Write
$A_r = 1, A_w = 5$								
$q_r = 1, q_w = 5$	All	All	All	None	All	None	All	None
$A_r = 2, A_w = 4$								
$q_r = 1, q_w = 5$	All	All	All	None	All	None	All	None
$q_r = 2, q_w = 4$	All	All	All	None	1,2,4,5	None	4,5	None
$q_r = 1, q_w = 4$	-	-	All	None	All	None	All	None
$A_r = 3, A_w = 3$								
$q_r = 1, q_w = 5$	All	All	All	None	All	None	All	None
$q_r = 2, q_w = 4$	All	All	All	None	1,2,4,5	None	4,5	None
$q_r = 1, q_w = 4$	-	-	All	None	All	None	All	None
$q_r = 3, q_w = 3$	All	All	1,2,3	1,2,3	None	None	None	None
$q_r = 2, q_w = 3$	-	-	1,2,3	1,2,3	1,2	None	None	None
$q_r = 1, q_w = 3$	-	-	1,2,3	1,2,3	1,2,3	None	1,2,3	None
$A_r = 4, A_w = 2$								
$q_r = 1, q_w = 5$	All	All	All	None	All	None	All	None
$q_r = 2, q_w = 4$	All	All	All	None	1,2,4,5	None	4,5	None
$q_r = 3, q_w = 3$	All	All	1,2,3	1,2,3	None	None	None	None
$A_r = 5, A_w = 1$								
$q_r = 1, q_w = 5$	All	All	All	None	All	None	All	None
$q_r = 2, q_w = 4$	All	All	All	None	1,2,4,5	None	4,5	None
$q_r = 3, q_w = 3$	All	All	1,2,3	1,2,3	None	None	None	None

($q_r[v]$ and $q_w[v]$). The relation between these parameters is described in Equations (3)-(7). The availability information for the four states in Figure 2 is summarized in Table I. As can be seen here, the maximum flexibility for choice of read and write quorums in a view is provided when $A_r = 3$ and $A_w = 3$. Let us look at this part of the table. There are six choices for the quorums. For the purpose of this example, we assume that all sites with copies of x within a physical partition attempt to form a single view. Thus, the last two choices are not relevant to State 1 (indicated by a "$-$" in Table I) since they violate Equation (4). If we consider $q_r = 3$ and $q_w = 3$ case, then *all* copies in State 1 can both be read and written. Since $A_r = 3$, it is possible to form a new view in State 2 consisting of copies in P_{21}. However, copies in P_{22} cannot form a new view. Thus, copies 1,2,3 can both be read as well as written in State 2. It is also possible to change the quorums in the new view (this is illustrated by the next two rows in the table). Using the same argument it may be shown that partitions in States 3 and 4 cannot form new views, and thus cannot change the quorums.

We will now refer to Table II that describes the behavior of the dynamic-linear algorithm with *majority quorums*. In State 1, the quorums are given as $q_r = q_w = 3$ (Equations (1)-(2)). This enables all copies in P_{11} to be read and written. In State 2, P_{21} is the distinguished group and this enables copies 1,2,3 to be read and written. Other states can be similarly explained.

Now let us consider the new protocol. Here, we are not bound by any accessibility thresholds to choose the quorums in views. Thus, the choice of

TABLE II: Availability table for the Dynamic-linear Algorithm

Dynamic	State 1		State 2		State 3		State 4	
Majority	Read	Write	Read	Write	Read	Write	Read	Write
$q_r = 3, q_w = 3$	All	All	-	-	-	-	-	-
$q_r = 2, q_w = 2$	-	-	1,2,3	1,2,3	-	-	-	-
$q_r = 1, q_w = 2$	-	-	-	-	1,2	1,2	-	-
$q_r = 1, q_w = 1$	-	-	-	-	-	-	1	1

TABLE III: Availability table for the New protocol

Quorums	State 1		State 2		State 3		State 4	
	Read	Write	Read	Write	Read	Write	Read	Write
$q_r = 1, q_w = 5$	All	All	All	None	All	None	All	None
$q_r = 2, q_w = 4$	All	All	All	None	1,2,4,5	None	4,5	None
$q_r = 1, q_w = 4$	-	-	All	None	All	None	All	None
$q_r = 3, q_w = 3$	All	All	1,2,3	1,2,3	None	None	None	None
$q_r = 2, q_w = 3$	-	-	1,2,3	1,2,3	1,2	None	None	None
$q_r = 1, q_w = 3$	-	-	1,2,3	1,2,3	1,2,3	None	1,2,3	None
$q_r = 2, q_w = 2$	-	-	1,2,3	1,2,3	1,2	1,2	None	None
$q_r = 1, q_w = 2$	-	-	-	-	1,2	1,2	None	None
$q_r = 1, q_w = 1$	-	-	-	-	-	-	1	1

quorums is much larger as shown in Table III. Thus it is even possible to read and write x when there is a single copy (as copy 1 in P_{41} of State 4) in a single partition. This was not possible in the accessibility algorithm. Similarly, the choices of the read and write quorums is much larger in the new protocol than the dynamic-linear algorithm. In addition to the increased availability, the proposed algorithm does not require two-phase commit to form new views (as in dynamic-linear).

4. Conclusions

This paper has presented a new dynamic replication control algorithm. The proposed algorithm is derived from two existing ones - the dynamic-linear and the accessibility thresholds algorithms. Three objectives have guided the design of the algorithm: 1) using local information to determine the chances of success of an operation, 2) making it possible to use very few copies (independent of the total number of copies) for reads/writes and thus increasing the availability, and 3) not requiring atomic implementations across a number of sites for implementing the algorithm itself.

The algorithm is tolerant to intermittent (non-malicious) failures of any number of sites and links. Database consistency is guaranteed at all times. The number of messages generated during the algorithm, when partitions are stable, is no more than in any of the two original algorithms. A simple constructive correctness proof is presented that gives clear insights into the workings of replication control algorithms.

References

[1] A. El Abbadi, D. Skeen, and F. Christian, "An Efficient fault-Tolerant Protocol for Replicated Data management," *Proceedings of the* 4^{th} *ACM Symposium on Principles of Database Systems,* pp. 215-229, March 1985.

[2] A. El Abbadi, and S. Toueg, "Availability in partitioned replicated databases," *Proceedings of the* 5^{th} *ACM Symposium on Principles of Database Systems,* pp. 240-251, March 1986.

[3] A. El Abbadi, and S. Toueg, "Maintaining availability in partitioned replicated databases," *ACM Trans. Database Systems*, 14 (2), June 1989, pp. 264-290.

[4] P. A. Alsberg, and J. D. Day, "A principle for resilient sharing of distributed resources," *Proceedings of the* 2^{nd} *International Conference on Software Engineering,* pp. 562-570, 1976.

[5] B. T. Blaustein, and C. W. Kaufman, "Updating replicated data during communication failures," *Proceedings of the* 11^{th} *International Conference on Very Large Data Bases,* pp. 49-58, 1985.

[6] J. J. Bloch, D. S. Daniels, and A. Z. Spector, "A weighted voting algorithm for replicated databases," *Journal of ACM,* Vol. 34, No. 4, pp. 859-909, 1987.

[7] D. Davcev, and W. A. Burkhard, "Consistency and recovery control for replicated files," *Proceedings of the* 10^{th} *ACM Symposium on Operating System Principles,* pp. 87-96, 1985.

[8] S. B. Davidson, H. Garcia-Molina, and D. Skeen, "Consistency in partitioned networks," *ACM Computing Surveys*, Vol. 17, No. 3, pp. 341-370, 1985.

[9] D. K. Gifford, "Weighted Voting for Replicated Data," *Proceedings of 7th ACM Symposium on Operating System Principles,* pp. 150-162, December 1979.

[10] S. Jajodia, "Managing replicated files in partitioned distributed database systems," *proceedings of IEEE* 3^{rd} *International Conference on Data Engineering ,* pp. 412-418, 1987.

[11] S. Jajodia and D. Mutchler, "A pessimistic consistency control algorithm for replicated files which achieves high availability," *IEEE Transactions on Software Engineering,* Vol. 15, No. 1, 1989.

[12] S. Jajodia and D. Mutchler, "Dynamic voting algorithms for maintaining the consistency of a replicated database," *ACM Trans. Database Systems*, 15(2), pp. 230-280, June 1990.

[13] J.-F. Paîis, "Voting with witnesses: a consistency scheme for replicated databases," *Proc. IEEE Int'l. conf. on Distributed Computing,* pp. 606-621, 1986.

[14] D. S. Parker, G. J. Popek, G. Rudisin, A. Stoughton, B. J. Walker, E. Walton, J.M. Chow, D. Edwards, S. Kiser, and C. Kline, "Detection of mutual inconsistency in databases," *IEEE Transactions on Software Engineering,* Vol. SE-9, No. 3, pp. 240-247, 1983.

[15] K. V. S. Ramarao, "Detection of mutual inconsistency in distributed databases," *Proceedings of the* 3^{rd} *IEEE International Conference on Data Engineering,* pp. 396-404, 1987.

[16] S. K. Sarin, B. T. Blaustein, and C. W. Kaufman, "System architectures for partition-tolerant distributed databases," *IEEE Transactions on Computers,* Vol. C-34, No. 12, pp. 1158-1163, 1985. 1983.

[17] J. Seguin, G. Sargeant, and P. Wines, "A Majority Consensus Algorithm for the Consistency of Duplicated and Distributed Information," *Proceedings of the* 1^{st} *International Conference on Distributed Computing Systems,* pp. 617-624, 1979.

[18] J. Tang and N. Natarajan, "A scheme for maintaining consistency and availability of replicated files in a partitioned distributed system," *Proceedings of the* 5^{th} *IEEE International Conference on Data Engineering,* pp. 530-537, 1989.

[19] R. B. Thomas, "A Majority Consensus Approach to Concurrency Control for Multiple Copy databases," *ACM Transactions on Database Systems,* Vol. 4, No. 2, pp. 180-209, June 1979.