# A Fault-Tolerant Commit Protocol for Replicated Databases *

Michael Rabinovich and Edward D. Lazowska

Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195

## Abstract

When failures occur during the execution of distributed commit protocols, the protocols may block in some partitions to avoid inconsistent termination of the transaction, thus making data items in these partitions unavailable for accesses. We present a protocol that incorporates two new ideas with the goal of improving data availability. First, a new two-level voting scheme is proposed for deciding in which partitions to terminate the transaction. In this scheme, a choice is made based on the number of data items available in the partition rather than on the number of individual nodes. Indeed, in replicated systems, a criterion based on the number of nodes may be misleading. Second, we propose a way to reduce blocking caused by accumulating network fragmentation. The idea employs the *views* mechanism previously used in replica management.

## 1 Introduction

The notion of a transaction [1] has proven to be a convenient general abstraction for data manipulation in the presence of failures, especially in distributed databases. In particular, being a unit of work that is guaranteed to be either completely done or not done at all, transactions make crash recovery feasible. In fact, the well-defined behavior of transactions in the presence of failures caused their proliferation to areas of distributed computing far beyond database systems [9, 15]. Consequently, the importance of implementation techniques for transaction management is growing.

Integral to transaction management are commit protocols, which ensure that a transaction is consistently done (committed) or not done (aborted) across all sites and data items involved. When failures occur during the execution of distributed commit protocols, the protocols may block in some partitions to avoid inconsistent termination of the transaction, thus making data items in these partitions unavailable for accesses. It has been shown that a commit protocol that never blocks under communication failures does not exist [13]. Given this fact, the goal is to reduce the negative effect on data availability caused by failures. The problem is further complicated if data replication is employed because, in this case, both the commit and the replica control protocols affect the availability of the data.

In this paper, we propose a protocol that incorporates two independent ideas to improve data availability under failures.

First, our protocol is based on a two-level weighted voting scheme which is used to resolve conflicts in the presence of failures. When failures occur during the execution of the commit protocol, two-level elections are held to choose partitions where the transaction in progress can be terminated. Nodes having a replica of a given data item form an electoral district. Each district holds local elections according to the replica control protocol used for maintaining consistency of the corresponding data item. Specifically, a partition wins in the district-level elections if it contains enough replicas of the corresponding data item to keep it available under the replica control protocol used. During the upper-level elections, the voting entities are districts. Each district has a number of votes equal to some weight pre-assigned to the corresponding data item. A partition receives *all* votes of the districts in which it won local elections, while all the "for" votes in the districts where it lost are wasted. The transaction can be committed (aborted) in a partition that obtained a minimum number of the upper-level votes, called a *commit quorum* $V_c$ *(abort quorum* $V_a$*)*, The sum of the commit and abort quorums must exceed the total number of votes, which ensures that the transaction will not terminate inconsistently in different partitions.

*This research was supported in part by the National Science Foundation (Grants No. CCR-8907666 and CCR-8619663), Digital Equipment Corporation, Apple Computer, and the Washington Technology Center.

The main advantage of two-level voting is that a partition containing enough nodes to keep most data items available *under the replica control mechanism used in the system* is chosen for terminating the transaction. In particular, if all data items have one vote and a majority of the total number of votes must be collected in the upper-level elections for both commitment and abortion to be possible, then a partition that wins the elections is the one that keeps a majority of data items available. In a non-replicated system, the districts consist of just one node and the local elections become trivial. Thus, in both replicated and non-replicated cases, a partition in which a majority of data items are available is chosen for terminating the transaction. Besides being a sensible strategy from the data availability standpoint, the above property specifies a well-defined and easily understood behavior of the system under failures. Another advantage is that if certain data items are known to be heavily used or otherwise especially important, our protocol can give preference to maintaining these items available by assigning them multiple upper-level votes (heavier weight). Finally, our scheme can work with a wide variety of replica control protocols such as voting [5], various structured coterie protocols [2, 8], dynamic voting [7], dynamic structured coterie protocols [10]. In fact, it can work with different replica control protocols used for different data items. Continuing our analogy, each district can independently establish its electoral law. The only important thing is that the party winning in a majority of districts (by the local laws in each case) wins the general elections. Thus, a system designer has a flexibility to choose a replica control protocol most suitable for each data item individually.

With our second idea, we propose a way to reduce the harmful effect of accumulating media failures on data availability. In distributed commit protocols, a transaction can be committed only if the coordinator of the protocol (usually the node that initiated the transaction) receives permission for commitment from all participating sites. Upon issuing permission for commitment, a participant enters a chain of vulnerable states in which it relies on instructions from the coordinator for making further progress. Should it become partitioned from the coordinator, blocking may occur. In the existing protocols, the likelihood of blocking increases if the system has already been fragmented due to prior failures, because in this case partitions will tend to contain fewer nodes and be less likely to win the elections necessary for transaction termination. To address this problem, we observe that if a set of nodes involved in the transaction is disjoint at the time the commit protocol starts, then, with high probability, the coordinator will not receive permissions for commitment from all sites, and hence the transaction will not commit. Therefore, if the individual sites could tell that, with high probability, the set of participants is disjoint, there would be no reason for them to issue permission for commitment. Instead, they could abort immediately and send a demand for aborting to the coordinator. This way, they avoid being in a state where blocking is possible. In our scheme, every node in the system maintains a periodically refreshed hint about what partition this node belongs to, called a *view*[1]. The commit coordinator piggybacks the list of participants with the request for commitment. If a participant discovers that some members of this list are outside its partition, it aborts the transaction locally and sends the demand for aborting to the coordinator. This causes all other participants who are in the same partition or can communicate with the coordinator to abort as well, freeing their data items for future accesses. Although this idea has been developed in conjunction with the protocol proposed here, it is independent from any specifics and can be used with other commit protocols as well.

## 2 Related Work

In [12], Skeen proposed a quorum-based fault-tolerant commit protocol. Skeen's protocol uses one-level voting in which partitions are chosen based on the number of nodes they contain. In the presence of replication, Skeen's protocol may choose a non-optimal partition for terminating transactions. As an example, consider a transaction that updates data items $x, y, z$ replicated on sites $x_1, x_2, x_3$; $y_1, y_2, y_3$; and $z_1, z_2, z_3$ respectively. Let commit and abort quorums both be equal to a majority of nodes. Assume a replica control protocol is based on simple voting which requires that a majority of replicas of a data item be operational for the data item to be available. Assume that the system partitions into $A = \{x_1, x_2, x_3, y_1, z_1\}$ and $B = \{y_2, y_3, z_2, z_3\}$. Skeen's protocol will terminate the transaction in partition $A$ and block the transaction in the partition $B$. However, the replica control protocol makes only one data item $x$ available in partition $A$ as opposed to two available data items in partition $B$. Hence, data availability for further accesses would be improved if the transaction would terminate in partition $B$. In contrast, our protocol counts the number of data items available in a partition rather than the number of nodes. In our example, partition $B$ wins in two local elections (districts $y$ and $z$) and hence this is where the transaction terminates, even though the total number of votes that $A$ has is greater.

In [6], two dual protocols were proposed, with the goal of establishing some cooperation between the commit and replica control protocols. These protocols are extreme particular cases of our two-level voting scheme. Our scheme converges to these protocols if one requires that a partition win in all districts to commit and in at least one district to abort

[1] To our knowledge, views were originally introduced in [3] in the context of replica control.

(or vice versa). However, these protocols lack the flexibility provided by our approach. In particular, if certain data items are known to be especially important, our protocol can be tuned so that it will give preference to keeping these items available. Also, by varying the relative sizes of commit and abort quorums, we can tune the protocol to reflect the specific performance and failure characteristics of the system. In fact, as discussed in section 7, in the modern systems, the most likely choice is to make abort and commit quorums equal, which is the farthest deviation from both dual protocols from [6].

## 3 Background: Replica Control Protocols

We first give some background information on replica control protocols, used to ensure consistency of replicated data in the presence of node and/or communication failures. Numerous replica control protocols have been proposed ([2, 3, 7] are a few examples). Different protocols are best suited for different usage patterns, failure models, etc. For the purpose of this paper, we consider the voting protocol ([5]), as it is most commonly used. In this protocol, every replica of each data item is assigned some number of votes. For each data item $x$, two numbers are chosen, $r(x)$ and $w(x)$ (called read and write quorums), such that $r(x) + w(x) > v(x)$ and $2w(x) > v(x)$, where $v(x)$ is the total number of votes assigned to replicas of $x$. A transaction must obtain at least $r(x)$ votes of the sites having a replica of $x$ before it can read $x$, and at least $w(x)$ votes before it can update $x$. Each site locks its replica prior to casting its votes. Because any read and write quorum of replicas of $x$ have at least one node in common, any successful read operation will find the most recent version of the data. Because any two write quorums of replicas of $x$ have at least one node in common, no two updates can perform in parallel, thus excluding the possibility of write conflicts.

## 4 The Protocol

In the protocol described here, we assume that the network is transitive (if $A$ can talk to $B$ and $B$ can talk to $C$, then $A$ can talk to $C$) and symmetric (if $A$ can talk to $B$ then $B$ can talk to $A$). Consequently, network partitionings are clean (each node within a partition can talk to all other nodes in the same partition, and no two nodes from different partitions can talk to each other). Although not needed for the correctness of the protocol, this assumption motivated some of the decisions in protocol design. We also assume that nodes and communication links are *fail-stop* [14], that is, they fail by crashing and do not behave maliciously. In this paper, we reserve the word "participant" for the nodes participating in the transaction. For instance, we never call members of partitions participants. Following [11], we will describe the commit protocols using the finite state machine (FSM) diagrams, one for each participating site. The structure of the state diagrams of our protocol is similar to that of the existing protocols [12, 6]; the novelty is in transition functions. Below, a set of nodes having a replica of a data item $x$ is called a district $x$. A subset of a district $x$ is *commit-sufficient* if it includes a write quorum of replicas of $x$, and is *abort-sufficient* if it includes a write *or* read quorum of such replicas[2]. Each district (data item) is assigned some (positive) weight. Two numbers, commit quorum $V_c$ and abort quorum $V_a$ are associated with every transaction, such that $V_c + V_a > V$, where $V$ is total weight of all data items participating in this transaction.

### 4.1 Normal Operation

In this section, we describe the behavior of the protocol when no additional failures occur during protocol execution. As in [6], we assume that, in the presence of replication, a replica control protocol is responsible for choosing the replicas on which to perform operations participating in the transaction. Hence, in the commit protocol, if any participating site demands to abort a transaction, the whole transaction should be aborted. The alternative is to make the commit coordinator itself decide whether each individual operation can perform based on responses from the replicas of the corresponding data item. The implications of the second approach are discussed at the end of this subsection.

The protocol is based on the three phase commit protocol [11] and is shown in Figure 1, with the transition function given in Table 1. Each node maintains a periodically refreshed list of all nodes in the system with which it can communicate. Following [3], we call this list a node's *view*. During the first phase, the commit coordinator broadcasts the request for commitment to all nodes participating in the transaction. With this request, the coordinator piggybacks a *participation set* which is the list of all participating sites. Upon receiving the request for commitment, a participant first checks if any members of the participation set received are outside its view. If this is the case, the participant immediately aborts the transaction (even if there are no local reasons for that) and sends the demand for abortion back to the coordinator. Otherwise, the participant either sends its permission for committing to the coordinator and enters state `W` where it waits to be informed by the coordinator on the global decision as to how to terminate the transaction, or it responds with the demand for aborting

[2] The dual and equally possible convention is to consider a subset commit-sufficient if it includes a write or read quorum of replicas and abort-sufficient if it includes a write quorum of replicas.

| Current State | Messages Received | Action |
|---|---|---|
| | **COORDINATOR** | |
| Q | Start protocol | Enter state W; broadcast request for commitment to all participants along with participation set PS. |
| W | `permission-to-commit` received from all participants. | Enter state PC; broadcast `prepare-to-commit` message to all participants. |
| | `demand-to-abort` received from some participants (inaccessible sites considered demanding to abort). | Enter state A; broadcast `abort` message to all participants; stop. |
| PC | `PC-ack`'s received from all nodes from commit-sufficient subsets of districts with total weight at least $V_c$. | Enter state C; broadcast `commit` message to all participants; stop. |
| | **PARTICIPANTS** | |
| Q | Request for commitment together with participation set PS received. | If PS $\not\subseteq$ VIEW, where VIEW is the node's view, then enter state A; send `demand-to-abort` to coordinator; stop.<br>else<br>if transaction can be committed locally, enter state W; send `permission-to-commit` to coordinator.<br>else enter state A; send `demand-to-abort` to coordinator; stop.<br>endif<br>endif |
| W | `abort` message received. | Enter state A; abort the transaction; stop. |
| | `commit` message received. | Enter state C; commit the transaction; stop. |
| | `prepare-to-commit` message received. | Enter PC state; send `PC-ack` to coordinator. |
| PC | `commit` message received. | Enter state C; commit the transaction; stop. |

Table 1: Transition function for the commit protocol (normal operation).

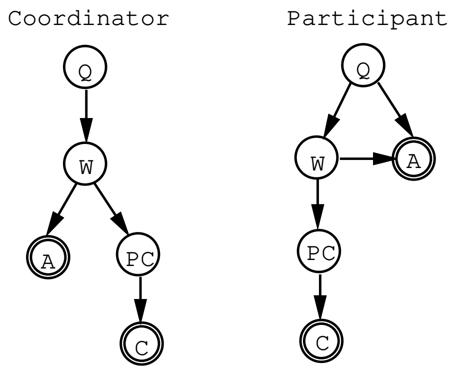


Figure 1: The state diagram for the commit protocol (normal operation).

and aborts the transaction without waiting. The rest of the protocol is similar to the three phase commit protocol. After receiving all responses (timeouts being considered demands for aborting), the coordinator initiates the second phase by broadcasting `prepare-to-commit` message if all participants permitted the commitment, or `abort` message otherwise. On receiving `abort` message, a participant aborts the transaction completing the protocol. On receiving a `prepare-to-commit` message, a participant enters `prepare-to-commit` state (`PC`) and responds with acknowledgement. The third phase starts when the coordinator receives the acknowledgements from all participants from commit-sufficient subsets of a commit quorum of districts. In the third phase, the coordinator broadcasts `commit` message to all participants causing them to commit the transaction.

Consider a rationale behind the described modification of the three-phase commit protocol. For a participant, its view is a hint on what partition the participant belongs to. So, if any members of the participation set do not belong to the node's view, the node infers that, with high probability, the set of sites involved in the transaction is disjoint. Then, by the assumption of cleanness of partitions, the coordinator will not receive the permission for commitment from all participants and will abort the transaction. Hence, there is no harm if the participant aborts the transaction immediately. In our protocol, the participant does just that thus freeing its data item for future accesses. In the existing protocols, in the same situation, the participant may well decide to issue permission for commitment and enter the waiting state. If additional failures occur during the execution of the protocol, voting will be initiated to find partitions where the transaction can be safely terminated. But because the network has already been fragmented, the chances that any partition will have enough operational nodes to win the elections will be slim.

Let $\Delta t$ be the maximum time needed for all nodes in the system to notice a change in the network topology and update their views to reflect the change [3]. Then, if a set of participants in the transaction was disjoint at $\Delta t$ before the start of the commit protocol and stays disjoint through the first phase of the protocol, then all participants will correctly identify the participation set as disjoint and abort the transaction. Hence, no blocking will occur at any site, even in the presence of additional node and communication failures during the execution of the protocol.

If a hint on network topology is optimistically incorrect at some sites (i.e., it identifies the participation set as connected while in fact it is disjoint; this may happen if the network partitions within $\Delta t$ before the protocol starts), then our protocol may show at worst the same blocking behavior as a similar protocol not using views. Indeed, a protocol without views can be considered as a protocol with views containing all nodes in the system (i.e., views are always optimistic).

Finally, if the communication failure that caused fragmentation has been repaired just before the protocol starts (within $\Delta t$), the views of some participants may be pessimistically incorrect, that is, they may identify the participation set as disjoint while in fact it is connected. In this case, the transaction will be unnecessarily aborted. Because changes in the network topology are infrequent compared to the transaction rate, this is unlikely to happen. But even if it does happen, it is a small price to pay for avoiding the harmful effect of network fragmentation on blocking. Indeed, the consequence of aborting a transaction is that it will have to be repeated, while the consequence of blocking is that data become unavailable.

Now consider briefly the case where the coordinator of the commit protocol concludes that a read (write) operation involved in the transaction can perform when the read (write) quorum of replicas of the corresponding data item responded with `permission-to-commit`, even if some replicas sent `demand-to-abort`. Then, the commit coordinator can commit the transaction when all participating operations can perform. In this case, it would be wasteful for participating sites to cause aborting simply because some of the participating sites are outside their partitions. Instead, the protocol should be modified as follows. As before, the nodes would maintain periodically refreshed views. With the request for committing, the commit coordinator would send to the participants the read and write sets of the transaction in progress [3]. Every participating site would use its view to guess if some but not all data items involved in the transaction are available in the site's partition, which means that the set of available data items is disjoint. A data item from the read (write) set is considered available in the site's partition

[3] The read (write) set of a transaction is a set of data items the transaction reads from (writes to).

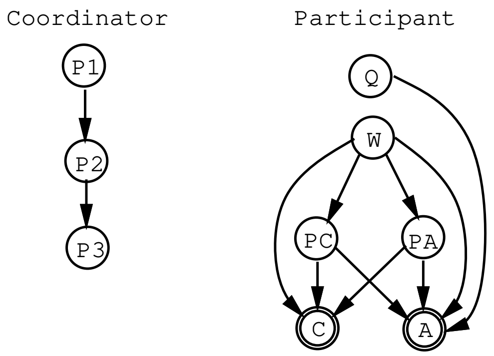


Figure 2: The state diagram for the termination protocol.

if the site's view contains a read (write) quorum of nodes having a replica of this data item. If a site discovers that the set of available data items is disjoint, it would abort the transaction and send a "strong" demand for aborting to the commit coordinator. Upon receiving a "strong" demand for aborting, the coordinator immediately aborts the transaction by sending an `abort` message to all participants. This is in contrast to the "normal" demand for aborting which, as just explained, does not necessarily cause an abort in this case.

### 4.2 Operation Under Failures (Termination Protocol)

A node concludes that a failure occurred if an expected message from another node does not arrive for sufficiently long time and a subsequent ping determines that this other node is not accessible. The only exception is the coordinator in state `W` which interprets the missing messages as the demands for abortion. If a node detects a failure during the commit protocol from the previous subsection, it initiates a *termination protocol* to either commit or abort the transaction consistently and to unblock the participating data items for future accesses. Note that each partition runs a separate instance of the termination protocol.

In the initial stage of the termination protocol, a coordinator of the termination is elected. Any existing election algorithm is suitable (see, for example, [4]). Once the termination coordinator is chosen, it runs the protocol shown in Figure 2 and Table 2. First, it polls the participants of the transaction about their states. If any node is in commit state (`C`), the transaction is immediately committed, no matter how many nodes are in the partition. (In this case, the transaction can only be either blocked or committed in all partitions, so consistency is not violated.) Similarly, the transaction is immediately aborted if a node in the initial (`Q`) or abort (`A`) state is found. Otherwise (i.e., if the partition does not contain sites in states `A, C, or Q`), the termination coordinator will count votes to decide how to terminate a transaction. It is possible to commit if (1) a partition contains at least one node in `PC` state, and (2) there are districts with total weight at least $V_c$ such that each is represented in the partition by a commit-sufficient subset of nodes in states `PC` or `W`. It is possible to abort if there are districts with total weight at least $V_a$ such that each is represented by an abort-sufficient subset of nodes in states `PA` or `W`. (Note that while it is quite possible that some partition satisfies the conditions for both committing and aborting, a situation where the condition for committing is satisfied in one partition and the condition for aborting is satisfied in another cannot occur.) If committing is possible, the termination coordinator moves the participants in state `W` into state `PC` and then commits the transaction. Moving the sites into `PC` state prior to committing is necessary to make sure that, should additional failures cause the termination process to be re-started from scratch, no partition will ever satisfy the condition for aborting. Similarly, if aborting is possible, the termination coordinator moves the participants in state `W` into state `PA` and then aborts the transaction. If both committing and aborting are possible, the coordinator chooses to commit. Finally, if neither committing nor aborting is possible, the transaction has to be blocked.

If additional failures are detected during the execution of the termination protocol, then a new termination coordinator is elected and the same protocol is restarted again.

## 5 Proof of Correctness

We now prove that the proposed protocol is correct. From now on, we will refer to the protocol in Table 1 as the normal commit protocol, and to the protocol in Table 2 as the termination protocol. We will say that a commit quorum is established in the system if all nodes from commit-sufficient subsets of the districts with total weight at least $V_c$ have visited state `PC` during the execution of the normal commit and the termination protocols. Similarly, we will say that an abort quorum is established if all nodes from abort-sufficient subsets of districts with total weight at least $V_a$ have visited state `PA`.

**Lemma 1.** *Once a commit quorum is established in the system, an abort quorum can never be established, and vice versa.*

**Proof.** Assume that a commit quorum is established and let $S_c$ be the set of nodes in the system that have visited state `PC`. By definition, $S_c$ includes all nodes from commit-sufficient subsets of districts with total weight $V_c$. Let $S_a$ be any set of nodes that must visit state `PA` for the abort quorum to be established. By definition, any such set must include

| Current State | Messages Received | Action |
|---|---|---|
| **TERMINATION COORDINATOR** | | |
| P1 | Start protocol | Enter state `P2`; request local states from all participants. |
| P2 | At least one participant reported state `C`. | Broadcast `commit` message to all participants; stop. |
| | At least one participant reported state `A` or `Q`. | Broadcast `abort` message to all participants; stop. |
| | None of the above combinations of messages received; and at least one participant reported state `PC`; and all nodes from commit-sufficient subsets of districts with total weight at least $V_c$ reported states `PC` or `W`. | Enter state `P3`; broadcast `prepare-to-commit` message to all participants. |
| | None of the above combinations of messages received; and all nodes from abort-sufficient subsets of districts with total weight at least $V_a$ reported states `PA` or `W`. | Enter state `P3`; broadcast `prepare-to-abort` message to all participants. |
| P3 | `PC-ack`'s received from all nodes from commit-sufficient subsets of districts with total weight at least $V_c$. | Broadcast `commit` message to all participants; stop. |
| | `PA-ack`'s received from all nodes from abort-sufficient subsets of districts with total weight at least $V_a$. | Broadcast `abort` message to all participants; stop. |
| **PARTICIPANTS** | | |
| Q, W, PC, PA, C, A | Request for a local state. | Stop executing the normal commit protocol; report the local state to the coordinator. |
| W, PC, PA | `commit` | Enter `C` state; commit transaction; stop. |
| Q, W, PC, PA | `abort` | Enter `A` state; abort transaction; stop. |
| W | `prepare-to-commit` | Enter `PC` state; send `PC-ack` to the coordinator. |
| | `prepare-to-abort` | Enter `PA` state; send `PA-ack` to the coordinator. |
| PC | `prepare-to-commit` | Send `PC-ack` to the coordinator. |
| PA | `prepare-to-abort` | Send `PA-ack` to the coordinator. |

Table 2: Transition function for the termination protocol.

all nodes from abort-sufficient subsets of districts with total weight $V_a$. Since $V_c + V_a > V$, where $V$ is the total weight of all districts, there exists a district $x$ such that $S_c$ includes a commit-sufficient subset of $x$ and $S_a$ includes an abort-sufficient subset of $x$. Due to the intersection property of read and write quorums (see section 3), any commit-sufficient subset and abort-sufficient subset of the same district have at least one node in common. This common node belongs to both $S_c$ and $S_a$. Since this node belongs to $S_c$, it has visited state PC. Hence, because state PA is not reachable from state PC, this node can never visit state PA. Therefore, not all nodes from $S_a$ can ever visit state PA. Because $S_a$ was assumed to be any set of nodes that must visit state PA for an abort quorum to be established, we conclude that establishing an abort quorum is impossible. A symmetric argument can be used to show that the dual claim of the lemma is also correct. □

**Lemma 2.** *If any participant has visited state* PC*, then there is a participant that was moved to state* PC *during the execution of the normal commit protocol.*

**Proof.** The proof by contradiction is trivial given the fact that in order to move any node to state PC, the termination protocol must find at least one node that has already been moved to this state. □

**Theorem.** *The proposed commit and termination protocols will terminate a transaction consistently in the presence of any number of fail-stop node and communication failures.*

**Proof.** First assume that some participant terminates the transaction by committing. We will show that no participant can ever abort. Since no site can commit before a commit quorum is established, a commit quorum has been established in the system. In particular, there are nodes that have visited state PC and hence, by Lemma 2, there is a node that was moved to state PC by the normal commit protocol. Before the normal commit coordinator moves any node into state PC, it makes sure that all participants have left the initial state Q and no participant in the system can ever abort during the execution of the normal commit protocol. Hence, all we need to show to prove that consistency is not violated in this case is that no participant can abort during the execution of the termination protocol. Indeed, before the first site can abort via the termination protocol, one of the following conditions must be satisfied. (a) There is a partition containing participants in states Q or A. Since we are considering the prerequisites necessary for the *first* participant to abort, this condition translates into the requirement that there be a partition containing a participant in state Q. But this is impossible because all participants have left state Q and this state is not reachable from any other state. (b) An abort quorum is established. This is also impossible due to Lemma 1.

Now assume that there is a participant that terminates by aborting during the execution of the normal commit protocol. Since the normal commit coordinator issues prepare-to-commit message only after making sure that no participant ever aborts during the execution of the normal commit protocol, the above assumption means that no participant will ever be moved into state PC by the normal commit protocol. Hence, by Lemma 2, no participant will ever move to state PC and a commit quorum will never be established. Since establishing a commit quorum is necessary before any node can commit, no participant will ever commit.

Finally, consider the case where no participant aborts during the execution of the normal commit protocol and there is a participant that is aborted by the termination protocol. Let $x$ be the first such participant [4]. We must show that no participant can ever commit. Indeed, $x$ could be aborted by the termination protocol only if there is a participant in state Q or A, or if an abort quorum is established. In the former case, because we are considering the *first* participant to abort, the condition reduces to the requirement that there is a participant in state Q. Then, since no node is moved to state PC by the normal commit protocol unless all participants leave state Q, no node moves to state PC during the execution of the normal commit protocol. As we showed in the previous paragraph, this means that no participant can ever commit. In the latter case, due to Lemma 1, establishing a commit quorum is impossible and no participant will ever commit.

We conclude that in all cases the proposed protocol terminates transactions consistently. □

## 6 Further Refinement of the Protocol

As a refinement, the coordinator of the normal commit protocol from Table 1 can broadcast the `commit` message as soon as there are districts involved in the transaction such that their total weight is at least $V_c$ and all nodes from commit-sufficient subsets of these districts either sent `PC-ack` or did not participate in the transaction at all. If we define a commit quorum to be established if all nodes from commit-sufficient subsets of districts with total weight at least $V_c$ either have visited state PC or did not participate in the transaction, then the proof of correctness from the previous section will still hold. By the same token, the termination coordinator can broadcast `prepare-to-commit` message as soon as there are districts involved in the transaction such that their total weight is at least $V_c$ and all nodes from commit-sufficient subsets of these districts either reported state PC

[4] More precisely, $x$ is such node that no other node aborts before $x$ does.

or did not participate in the transaction. Hence, with the request for committing, the normal commit coordinator can send the total number of participating replicas for each data item to all participants. Then, the termination coordinator would know how many replicas of each data item do not participate and could take them into account when deciding whether `prepare-to-commit` message can be broadcast. Similarly, the termination coordinator does not have to wait for `PC-ack`'s from all nodes from commit-sufficient subsets of districts with total weight $V_c$ before broadcasting the `commit` message. All it must ensure is that the nodes that sent `PC-ack` together with the replicas that do not participate in the transaction include commit-sufficient subsets of above districts.

The benefit of this refinement is twofold. First, the condition for terminating transactions is weakened. Second, the coordinators of the normal commit and termination protocols must wait for fewer responses before broadcasting messages. This shortens the execution time of the protocols which makes them less susceptible to additional failures.

## 7 Setting Parameters of the Protocol

In this section, we discuss how to choose parameters of the protocol. The weights for data items should be assigned depending on the importance of the individual data item's availability. The more important it is to have a data item available, the more weight it is assigned. The commit and abort quorums are chosen depending on how much time the nodes spend in states `W` and `PC` during the execution of the normal commit protocol [12]. Should a failure occur, the more time the nodes spend in state `W` (as compared to state `PC`), the more likely the termination coordinator will find all participants in the partition to be in state `W`, in which case the transaction can only be either aborted or blocked. Hence, the easier it should be to abort, which means a smaller abort quorum (and larger commit quorum) must be chosen. In modern systems, the work to be done by a node in state `W` includes logging the update and the state information (a site must keep its state in stable storage for recovery). This can usually be done in one disk operation. A node in a state `PC` must only log its state. This also requires one disk operation. In most cases, the time to execute both operations will be roughly the same. Then, assuming that communication delays are negligible compared to disk operations, if failures occur, the termination coordinator will equally likely find all the participants in state `W` or in state `PC`. Hence, the most likely choice is to make commit and abort quorums equal. However, these parameters can be fine-tuned to match the specific performance characteristics of the system.

## 8 Summary

In this paper, we propose a fault-tolerant distributed commit protocol which incorporates two new ideas. First, when choosing a partition where to terminate a transaction, we propose a way to count the number of data items available in the partition under the replica control protocol used in the system, as opposed to counting individual nodes. While for non-replicated systems our scheme converges to the existing one, in the presence of replication a partition that contains most nodes is not necessarily the one that contains most available data items, whereas the goal of a protocol must be to avoid blocking of as many available data items as possible. Also, by associating weights with data items rather than with individual nodes, our scheme allows for a very straightforward interpretation of weights: they are the measure of comparative importance of maintaining various data items unblocked (i.e., available for accesses).

Second, we propose a way to significantly reduce the harmful effect of network fragmentation on blocking. In essence, we eliminate the effect of fragmentation accumulated before the beginning of the protocol. The price for achieving it is that if the fragmentation has been repaired just before the commit protocol starts, the transaction may be unnecessarily aborted. Because the rate of failures and repairs is much lower that the transaction rate, this is unlikely to happen. But even if it does happen, it is a small price to pay for avoiding blocking. Indeed, the consequence of aborting a transaction is that it will have to be repeated, while the consequence of blocking is that data become unavailable.

## References


[1] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency control and recovery in database systems.* Addison-Wesley, Reading, Mass., 1987.

[2] S. Y. Cheung, M. H. Ammar, and M. Ahamad. The Grid protocol: a high performance scheme for maintaining replicated data. In *Proc. of the IEEE 6th Int. Conf. on Data Engineering*, pp. 438-445, 1990.

[3] A. El Abbadi, D. Skeen, and F. Cristian. An efficient, fault-tolerant protocol for replicated data management. In *Proc. of the 4th ACM SIGACT/SIGMOD Symp. on Principles of Database Systems*, pp. 215-229, 1985.

[4] H. Garcia-Molina. Elections in a distributed computing system. *IEEE Trans. Computers,* C-31(1), pp. 48-59, January 1982.

[5] D.K. Gifford. Weighted voting for replicated data. In *Proc. of the 7th ACM Symposium on Operating Systems Principles*, pp. 150-159, December 1979.

[6] C.-L. Huang and V. Li. A quorum-based commit and termination protocol for distributed database systems. In *Proc. of the 4th IEEE Int. Conf. on Data Eng.*, pp. 136-143, 1988.

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

[8] A. Kumar. Performance analysis of a hierarchical quorum consensus algorithm for replicated objects. In *Proc. of 10th Symp. on Distributed Computing Sys.,* pp. 378-385, IEEE, 1990.

[9] B. Liskov. Distributed programming in Argus. *CACM* 31(3), pp.300-312, March 1988.

[10] M. Rabinovich and E. D. Lazowska. Improving fault-tolerance and supporting partial writes in structured coterie protocols for replicated objects. To appear in *Proc. of the ACM SIGMOD Conf. on Management of Data*, June 1992.

[11] D. Skeen. Nonblocking commit protocols. In *Proc. of SIGMOD Int. Conf. on Management of Data*, pp. 133-142, 1981.

[12] D. Skeen. A quorum-based commit protocol. In *Proc. of the 6th Berkeley Workshop on Distr. Data Management and Comp. Networks*, pp. 69-80, 1982.

[13] D. Skeen and M.Stonebraker. A formal model of crash recovery in a distributed system. *IEEE Trans. on Software Eng.*, 9(3), pp. 219-228, May 1983.

[14] R. Schlicting and F. B. Schneider. Fail-stop processors: An approach to designing fault-tolerant computing systems. *ACM Trans. on Comp. Sys.*, 1(3), pp. 222-238, August 1982.

[15] F. Schmuck and J. Wyllie. Experience with transactions in QuickSilver. In Proc. of the 13th ACM Symp. on Operating Sys. Principles, pp. 239-253, October 1991.