A Quorum-based Commit and Termination Protocol for Distributed Database Systems*

Ching-Liang Huang and Victor O.K. Li

Department of Electrical Engineering
Univesity of Southern California
Los Angeles, CA 90089-0272

Abstract

Correctness and availability are two competing goals in the design of a fault-tolerant transaction processing strategy for distributed database systems. To achieve absolute correctness, availability of data may be reduced when failures occur. In this paper, a quorum-based commit and termination protocol is designed with the goal of maintaining high data availability in case of failures. The protocol proposed is resilient to arbitrary concurrent site failures, lost messages and network partitioning. The major difference between this protocol and existing ones is that the voting partition processing strategy is taken into consideration in the design. By doing so, our protocol is expected to maintain higher data availability.

1 INTRODUCTION

A distributed database system supports a database physically distributed over multiple sites interconnected by a computer network. Compared to centralized database systems, distributed database systems have at least the following distinctive advantages: (1) copies of data can be replicated in more than one site to improve performance and availability of the system. By storing copies of data in sites where they are frequently accessed, the need for expensive remote access can be reduced. By storing copies of critical data on processors with independent failure modes, the probability that at least one copy of the data will be accessible increases. (2) concurrent execution of multiple transactions at the same time can provide higher system throughput.

However, there are disadvantages also. Maintaining database correctness in a distributed environment is more difficult due to failures and synchronization problems. The generally accepted notion of correctness in a distributed database system is that it executes transactions so that they appear to users as indivisible, isolated actions on the database. This property, referred to as *atomic execution* [4], can be achieved by guaranteeing the following properties: *atomic commitment* for each transaction and *serializability* among all transactions in an environment where failures may occur. The failure scenarios considered are site failures, lost messages, and network partitioning (the network is partitioned into several disjoint components with no communication possible between them.) Atomic commitment means that either all of a transaction's updates are performed or none are performed. The execution of a set of transactions is serializable if it has the same effect as executing the same set of transactions one at a time in some order.

Atomic commitment is ensured by commit protocols and termination protocols. A termination protocol is invoked to consistently terminate transactions when failures occur and render the continued execution of a commit protocol impossible. Several commit protocols and termination protocols have been proposed. The two-phase commit protocol (2PC) [9,11] is the simplest one (Fig. 1). The protocol uses a designated site (usually the site where a transaction is issued) to coordinate the execution of the transaction at the other sites (participants). In the first phase of the protocol, the coordinator distributes the update values (in Vote-Req messages) to all sites which contain data items to be updated, and then each site individually votes on whether to commit ('yes') or abort ('no') the transaction. (An example in which a site may vote 'no' is when its I/O subsystem fails and it cannot implement the update.) In the second phase, the coordinator makes a decision on whether the transaction should be committed or aborted based on the responses it receives from the participants (the transaction can be committed iff every site votes 'yes'), and informs each site its decision. Each site will then commit or abort the transaction accordingly when the coordinator's decision is received. The termination (i.e. commit or abort) of a transaction at a site is an irrevocable operation. If a transaction is committed (aborted), it cannot be later aborted (committed).

In the absence of failures, two-phase commit works well. However, it is blocking under site failures or network partitioning. In the protocol, once a participant has voted 'yes', it cannot terminate the transaction until it has received the coordinator's decision. If the coordinator crashes and fails to send out its decision, or the network is partitioned and the decision cannot be delivered, the participants must block the transacton's execution and wait for the failures to recover. When a transaction is blocked, locks will be held on data items accessed by the transaction, rendering those data items inaccessible to the other transactions.

It has been proved that there does not exist a commit protocol nonblocking to concurrent site failures and network partitioning [14,17]. Since it is impossible to eliminate blocking, it is desirable to minimize the reduction in data availability when

*This work is supported in part by the Joint Services Electronics Program under Contract No. F49620-85-C-0071 and by the National Science Foundation under Grant No. DCI-8519101.

failures occur. There are two factors that effect the availability of data items. First, to ensure atomic commitment, data items locked by blocked transactions are not accessible to the other transactions. Secondly, to ensure serializability, conflicting operations issued by transactions executing in different partitions of the network should be controlled (by partition processing strategies), further reducing the availability of the data. All existing commit and termination protocols do not take the second factor into consideration in the design, therefore data availability is more likely to be reduced by both factors.

In this paper, we propose a quorum-based commit and termination protocol with the goal of maintaining high data availability in case of failures. The protocol proposed has the following salient features :

(1) It is resilient to arbitrary concurrent site failures, lost messages and network partitioning.

(2) It does not require the correct identification of the failure type and it does not require that the coordinator in each partition be unique.

(3) It can deal with additional failures that occur during the execution of the termination protocol, i.e., it is reenterable.

(4) It takes the voting partition processing strategy into consideration in the design. By doing so, we decrease the reduction in data availability due to failures.

This paper is organized as follows. In section 2, we give an overview of related work and then describe a problem with the existing protocols. In section 3, quorum-based commit and termination protocols are developed. Section 4 gives the proof of correctness of the protocols. We conclude in section 5 with some discussions on the protocols proposed.

2 RELATED WORK

The local transaction states of any commit protocol form two disjoint subsets: the committable states and the noncommittable states. A site will occupy a committable state only if all participating sites have voted 'yes' on the transaction. In [15], a three-phase commit protocol (Fig. 2) and a termination protocol were presented. By introducing a buffer state PC (Prepare-to-Commit) between the wait state (W) and the commit state (C), there exists no local state adjacent to both the abort state and the commit state and there exists no noncommittable state adjacent to the commit state, rendering the three-phase commit protocol nonblocking under site failures.

In [16], a quorum-based commit and termination protocol is proposed which reduces the probability that a large partition (one consisting of many participants) will be blocked in the event of a partitioning. The protocol uses a weighted voting scheme to resolve conflicts during failures. Each site is assigned some number of votes. When failures occur, a transaction is committed only if a minimum number of votes, called a commit quorum, are cast for committing. Similarily, a transaction will be aborted only if a minimum number of votes, called an abort quorum, are cast for aborting. The sum of the commit quorum and the abort quorum must exceed the total number of votes.

Serializability in nonpartitioned database systems can be achieved by using concurrency control algorithms [2,6,10,13]. When the database is partitioned, not only must serializability be ensured in each partition but also across partitions. Several partition-processing stategies for ensuring serializability in a partitioned environment have been proposed [1,3,4,5], [8,12,18,19] and most of them are based on the observation that a sufficient (but not necessary) condition for serializability is that no two partitions execute conflicting data operations. Among them, the voting scheme [8] is the most commonly used. In the scheme, every copy of each data item is assigned some number of votes. A transaction must collect $r(x)$ votes of a data item x before it can read the data item, and collect $w(x)$ votes of a data item x before it can write that data item. Two constraints must be satified : (1) $r(x) + w(x) > v(x)$, the total number of votes of data item x, and (2) $w(x) > v(x)/2$. The first constraint ensures that data read by any transaction will contain the most recent copy. (Version numbers are used to identify the most recent copy.) It also ensures that a data item cannot be read in one partition and written in another when the system is partitioned. The second constraint ensures that two writes on a data item cannot happen in parallel, or if the system is partitioned, that writes cannot occur in two different partitions.

The missing writes scheme [5] is an adaptive voting strategy that improves performance when there are no failures in the system.

If we use any existing commit and termination protocol to ensure atomic commitment and use the voting scheme to ensure serializability, then the correctness of the database can be maintained. However, we note that the availability of the data items is reduced twice, first by the commit protocol and the termination protocol and then by the partition-processing strategy. If a transaction is blocked in a partition by the termination protocol, then even though the partition may have enough votes for some data items in the writeset of that transaction, those data items are not accessible in the partition. On the other hand, even though a transaction is terminated in a partition by the termination protocol, if the partition does not have enough votes for some data items in the writeset of that transaction, then those data items are not accessible in the partition. The following examples illustrate this point.

EXAMPLE 1 A transaction TR issued at $site_1$ updates data items x and y. Data item x has copies x_1, x_2, x_3 and x_4 stored at $site_1$, $site_2$, $site_3$, and $site_4$, respectively. Data item y has copies y_5, y_6, y_7 and y_8 stored stored at $site_5$, $site_6$, $site_7$, and $site_8$, respectively. Suppose the quorum-based commit and termination protocol [16] is used to ensure atomic commitment and the voting scheme is used to ensure serializability. Assume the vote assigned to each site in the quorum-based protocol is 1 with commit quorum $V_c = 5$ and abort quorum $V_a = 4$ ($V_c + V_a > 8 = V_t$, the total number of votes), and the vote assigned to each copy of x and y in the voting scheme is also 1 with r(x) = r(y) = 2 and w(x) = w(y) = 3 (The two constraints are satisfied.). Suppose during the commitment procedure of TR, the coordinator ($site_1$) fails and the network is partitioned into three parts: $G_1 = \{site_1, site_2, site_3\}$, $G_2 = \{site_4, site_5\}$ and $G_3 = \{site_6, site_7, site_8\}$, leaving the local state of $site_5$ as PC and all the other active participants as W (Fig. 3). Since the votes contained in each partition is less than both the commit quorum and the abort quorum, transaction TR will be blocked in all the partitions, which causes data item x and y to be inaccessible in all the partitions even though partition G_1 has enough votes for reading data item x and partition G_3 has enough votes for updating data item y.

EXAMPLE 2 Suppose we have the same scenario as Example 1 except that the three-phase commit and termination protocol is used for ensuring atomic commit. The termination protocol of the three-phase commit protocol is designed for dealing with site failures only, and it dictates that if there exists a site in PC state or commit state, then the transaction should be committed; else the transaction should be aborted. Therefore, partition G_1 and G_3 will abort transaction TR while partition G_2 will commit the transaction, and transaction TR is terminated inconsistently.

3 THE QUORUM-BASED COMMIT AND TERMINATION PROTOCOLS

Atomic commitment is ensured by the quorum-based commit and termination protocols. The termination protocol is invoked to correctly terminate a transaction at all active participating sites (participants) when the normal commitment procedure is interrupted by failures. When it is invoked, a coordiantor will first be elected in each partition by an election protocol [7]. In the following, we will assume that an election protocol is availability. It should be noted that our protocols do not require the election of a unique coordinator in each partition.

3.1 THE QUORUM-BASED TERMINATION PROTOCOLS

In this subsection, we will assume that the commit protocol used is the three-phase commit protocol. A quorum-based commit protocol similar to the three-phase commit protocol will be designed in the next subsection. The quorum-based commit protocol will help speed up the commitment procedure.

Notation 1 : *W(TR) is the set of data items in the writeset of transaction TR.*

Definition 1 (partition state) *The partition state PS of a transaction TR in a partition is the set of local states of all active participants of TR in the partition.*

Definition 2 (concurrency set) *The concurrency set C(PS) of a partition state PS is the set of partition states which may be concurrent with PS.*

To terminate a transaction consistently in all the partitions, the following rules must be obeyed.

Rule 1 : *Given that the partition state of a transaction in a partition is PS. If C(PS) contains a partition state where at least one participant is in the commit state, then the partition should commit the transaction. On the other hand, if C(PS) contains a partition state where at least one participant is in the abort state, then the partition should abort the transaction.*

Rule 2 : *If a partition has partition state PS for a transaction TR, the partition should either block TR or terminate TR consistently with all the other partitions with state PS' for TR, where PS' is in C(PS).*

To minimize the reduction in data availability, we should design a termination protocol with the following property: if a partition has enough votes for a data item, that data item will not be blocked in the partition by the termination protocol; that is, if a partition has enough votes for a data item in the writeset of a transaction, the termination protocol should either commit or abort the transaction in the partition. Unfortunately, the following argument shows that such a termination protocol does not exist.

When the commitment procedure of a transaction is interrupted by failures, the mutually-exclusive, collectively-exhaustive partition states that a partition can be in and their corresponding concurrency sets are listed in Fig. 4. By rule 1, *PS3* should be aborted, *PS6* should be committed. By rule 2, both *PS1* and *PS2* should be blocked or aborted since *PS3* is in both *C(PS1)* and *C(PS2)*; *PS5* should be blocked or committed since *PS6* is in *C(PS5)*; *PS4* should be blocked or terminated consistently with *PS2* and *PS5*. Note that *PS2* is in *C(PS5)* and vice versa. When a partition G1 has enough votes for some but not all of the data items in W(TR) of a transaction TR and is in state *PS2* for the transaction, it is possible that some other partition G2 may have enough votes for some other data items in W(TR) and is in state *PS5* for TR. As partitions in state *PS2* can only be blocked or aborted and partitions in state *PS5* can only be blocked or committed, it is impossible to terminate TR in both G1 and G2 even though both of them have enough votes for some data item in W(TR). This argument can be generalized for any termination protocol working with any commit protocol.

In spite of the negative result shown above, we expect to be able to maintain higher data availability if the partition processing strategy is taken into consideration in the design of a termination protocol. The following two solutions follow such an approach.

3.1.1 TERMINATION PROTOCOL 1

This termination protocol consists of three phases. In the first phase the newly elected coordinator polls the participants of transaction TR in its partition about their local states, and their replies determine the action taken in the next two phases. If any participant has committed, then transaction TR is immediately committed at all participants in the partition. If any participant has aborted or is in the initial state, then transaction TR is immediately aborted at all participants in the partition. Otherwise, the coordinator will attempt to establish a quorum.

We introduce a new state, PA, and a new message, Prepare-To-Abort. A site will relinquish its right to participate in an abort quorum by moving to state PC when a commit quorum is formed and a Prepare-To-Commit message is received. A site will relinquish its right to participate in a commit quorum by moving to state PA when an abort quorum is formed and a Prepare-To-Abort message is received. A commit quorum is possible if at least one participant is in the committable state PC and the partition has *at least $w(x)$ votes* for *every* data item x in W(TR) from those participants which are not in state PA. If this is the case, the coordinator will attempt to move all participants in state W (wait) into PC by broadcasting Prepare-To-Commit messages. Barring additional failures, the coordinator will then commit the transaction in the partition.

An abort quorum is possible if not all participants in the partition are in state PC and the partiton has *at least $r(x)$ votes* for *some* data item x in W(TR) from those participants which are not in state PC. If this is the case, the coordinator will attempt to move all participants in state W (wait) into PA by broadcasting Prepare-To-Abort messages. Barring additional failures, the coordinator will then abort the transaction in the partition. Note that if a commit quorum is formed in one partition, then it is impossible for an abort quorum to be formed in another partition and vice versa. However, several abort quorums may be formed at the same time.

If there are additional failures and the coordinator does not receive enough acknowledgement messages, then the termination protocol will be repeated again. A prototype for termination protocol 1 is shown in Fig. 5. The longest end-to-end propagation delay of the network is assumed to be T.

The state transition diagram is given in Fig. 6. Note that there is no transition between PC and PA. A participant should ignore PREPARE-TO-COMMIT messages if it is in PA state and ignore PREPARE-TO-ABORT messages if it is in PC state. These are required to deal with additional failures and the possibility of more than one coordinators in a partition, each communicating with distinct intersecting subsets of the participants. It can be best explained by a counterexample.

EXAMPLE 3 A transaction TR issued at site 1 updates data items x and y which have copies x_2, x_3, x_4, x_5 and y_2, y_3, y_4, y_5 stored at sites 2, 3, 4, and 5, respectively. Assume the vote of each copy of both x and y is 1 and $w(x) = w(y) = 3$, $r(x) = r(y) = 2$. Suppose we adopt termination protocol 1. However, a participant will respond to a PREPARE-TO-ABORT message when in PC state and respond to a PREPARE-TO-COMMIT message when in PA state. Suppose during the commitment procedure of TR, the coordinator fails and the network is partitioned into two parts: $G_1 = \{site_1, site_2\}$ and $G_2 = \{site_3, site_4, site_5\}$, leaving the local state of $site_5$ as PC and all the other active participants as W (Fig. 7). After the election protocol, $site_2$ and $site_3$ will be elected as the coordinator of G_1 and G_2, respectively. However, just before $site_2$ starts collecting local state information, the network recovers, giving rise to two coordinators in the same partition. Assume that all the messages between $site_2$ and $site_3$ and from $site_2$ to $site_5$ are somehow lost in the network. So $site_2$ will only collect enough votes to abort TR and will send out PREPARE-TO-ABORT messages, while $site_3$ will collect enough votes to commit TR and will send out PREPARE-TO-COMMIT messages. If a site is allowed to respond to a PREPARE-TO-ABORT message in PC state and to respond to a PREPARE-TO-COMMIT message in PA state, then $site_4$ will respond to both the PREPARE-TO-ABORT message and the PREPARE-TO-COMMIT message. Therefore, $site_2$ will receive enough PA-ACK's to send ABORT commands, while $site_3$ will receive enough PC-ACK's to send COMMIT commands, and transaction TR is terminated inconsistently.

The following example illustrates that termination protocol 1 can maintain higher data availability than the protocol in [16].

EXAMPLE 4 Suppose we have the same scenario as example 1 except that the three-phase commit protocol and termination protocol 1 is used for ensuring atomic commitment. Since both partitions G_1 and G_3 satifiy the abort quorum in protocol 1, transaction TR can be aborted in G_1 and G_3. Now data item x in G_1 is not blocked any more and G_1 has enough votes for reading data x, so x can be read in G_1. Similarly, data item y can be updated in G_3.

3.1.2 TERMINATION PROTOCOL 2

Protocol 2 is the same as protocol 1 except that the criteria for forming a quorum is different. In this protocol, an abort quorum is possible if the partition has *at least $w(x)$ votes* for *every* data item x in W(TR) from participants which are not in PC state. A commit quorum is possible if at least one participant is in state PC and the partition has *at least $r(x)$ votes* for *some* data item x in W(TR) from participants which are not in PA state. A prototype for this protocol is shown in Fig. 8.

3.2 THE QUORUM-BASED COMMIT PROTOCOLS

Instead of using the three-phase commit protocol, we can design a quorum-based commit protocol for each termination protocol described above. These protocols are similar to the three-phase commit protocol except that the coordinator can send out commit commands before all the PC-ACKs' are received (Fig. 9), thus speeding up the commitment procedure. For commit protocol 1, the coordinator only has to wait for w(x) votes of PC-ACKs' for every data item x in the write set, because receiving these PC-ACKs' ensures that an abort quorum can never be formed for the transaction anymore. For commit protocol 2, the coordinator only has to wait for r(x) votes of PC-ACKs' for some data item x in the write set for similar reasons. So commit protocol 2 runs faster than commit protocol 1.

4 PROOF OF CORRECTNESS

In this section, we give the proof of correctness of commit protocol 1 and termination protocol 1. Similarly, protocol 2 can be proved.

Lemma 1 *If the first participant that terminates transaction TR commits the transaction, then all other participants will either commit or block transaction TR in the termination protocol when failures occur.*

Proof: Let the first participant that terminates transaction TR be $site_a$. Consider the following two cases:

Case(1): $site_a$ is committed by the quorum-based commit protocol.

In this case, participants which are in PC or commit state must constitute at least *w(x)* votes for every data item x in W(TR) before the termination protocol is executed. For any given partition, if the local states collected by its coordinator contains a commit, then the coordinator will send commit commands to all participants in the partition; else there must be less than *r(x)* votes for every data item x in W(TR) from participants which are not in PC state and the coordinator will not be able to move any participant to PA state. Therefore, no participant in the partition will abort transaction TR.

Case(2): $site_a$ is committed by the termination protocol.

In this case, it is impossible to have a partition that contains a participant in the initial state because there must exist a participant in PC state for $site_a$ to be committed. It is also impossible to have a partition where the coordinator can receive enough PA-ACK's from participants weighing a total of at least *r(x)* for some data item x in W(TR). This is because before $site_a$ is committed, participants weighing a total of at least *w(x)* for every data item x in W(TR) must have been moved to state

PC and none of them will respond to a PREPARE-TO-ABORT message. Therefore, no participant will abort transaction TR.

Q.E.D.

Lemma 2 *If the first participant that terminates transaction TR aborts the transaction, then all other participants will either abort or block transaction TR in the termination protocol when failures occur.*

Proof: Let the first participant that terminates transaction TR be $site_b$. Consider the following two cases :

Case(1) : $site_b$ is aborted by the quorum-based commit protocol.

In this case, all other participants of transaction TR must either be in the abort state, the wait state or the initial state before the termination protocol is executed. So it is impossible to have a partition that contains at least one participant in state PC, which in turn prevents any partition to have a commit quorum. Therefore, no participant will commit the transaction.

Case(2) : $site_b$ is aborted by the termination protocol.

In this case, it is impossible to have a participant that receives a delayed commit command because the participants in the same partition as $site_b$ cannot all be in state PC for $site_b$ to be aborted. It is also impossible to have a partition where the coordinator can receive enough PC-ACK's from participants weighing a total of at least $w(x)$ for every data item x in W(TR). This is because before $site_b$ is aborted, participants weighing a total of at least $r(x)$ for some data item x must have been moved to state PA and none of them will respond to a PREPARE-TO-COMMIT message. Therefore, no participant will commit the transaction.

Q.E.D.

Theorem 1 *The proposed termination protocol will terminate transactions consistently under concurrent site failures, lost messages and network partitioning.*

Proof: Follows as a direct consequence of Lemma 1 and Lemma 2.

Q.E.D.

5 CONCLUSIONS

We have presented two quorum-based commit and termination protocols which are resilient to arbitrary concurrent site failures, lost messages and network partitioning. By taking the voting partition processing strategy into consideration in the design, our protocols are expected to maintain higher data availability than existing ones. The idea can be generalized to work with other partition-processing strategies. Protocol 2 is expected to perform better than protocol 1 because its commit protocol runs faster, which not only shortens the commitment procedure of transactions when the system operates normally but also makes transactions less susceptible to failures.

References

[1] P. A. Alsberg and J. D. Day. A principle for resilient sharing of distributed resources. In *Proc. 2nd IEEE Int. Conf. on Software Eng.*, pages 627–644, 1976.

[2] P. A. Bernstein and N. Goodman. Concurrency control in distributed database systems. *ACM Computing Surveys*, 13(2):185–221, June 1981.

[3] S. B. Davidson. Optimism and consistency in partitioned distributed database systems. *ACM Trans. on Database Systems*, 9(3):456–481, September 1984.

[4] S. B. Davidson, H. Garcia, and D. Skeen. Consistency in partitioned networks. *ACM Computing Surveys*, 17(3):341–370, September 1985.

[5] D. L. Eager and K. C. Sevcik. Achieving robustness in distributed database systems. *ACM Trans. Database Systems*, 8(3):354–381, September 1983.

[6] K. P. Eswaran, J. N. Gray, R. A. Lorie, and I. L. Traiger. The notions of consistency and predicate locks in a database system. *ACM Communication*, 19(11):624–633, November 1976.

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

[8] D. K. Gifford. Weighted voting for replicated data. In *Proc. 7th ACM Symposium on Operating Systems Principles*, pages 150–162, 1979.

[9] J. Gray. Notes on database operating systems. In *Operating System: An Advanced Course*, New York: Springer-Verlag, 1979.

[10] H.T. Kung and J. T. Robinson. On optimistic methods for concurrency control. *ACM Trans. on Database Systems*, 6(2):213–226, June 1981.

[11] B. Lampson and H. Sturgis. *Crash Recovery in a Distributed Storage System.* Technical Report, Computer Sci. Lab., Xerox Parc, Palo Alto, CA, 1976.

[12] T. Minoura and G. Wiederhold. Resilient extended true-copy token scheme for a distributed database system. *IEEE Trans. on Software Eng.*, SE-8(3):173–189, May 1982.

[13] C. H. Papadimitriou. The serializability of concurrent database updates. *Journal of ACM*, 26(4):631–653, October 1979.

[14] D. Skeen. *Crash Recovery in Distributed Database System.* PhD thesis, Dept. of EECS, University of California, Berkeley, May 1982.

[15] D. Skeen. Nonblocking commit protocols. In *SIGMOD Int. Conf. on Management of Data*, pages 133–142, 1981.

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

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

[18] D. Skeen and D. Wright. Increasing availability in partitioned networks. In *Proc. 3rd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 290–299, 1984.

[19] M. Stonebraker. Concurrency control and consistency of multiple copies in distributed ingres. *IEEE Trans. on Software Eng.*, SE-5(3):188–194, May 1979.

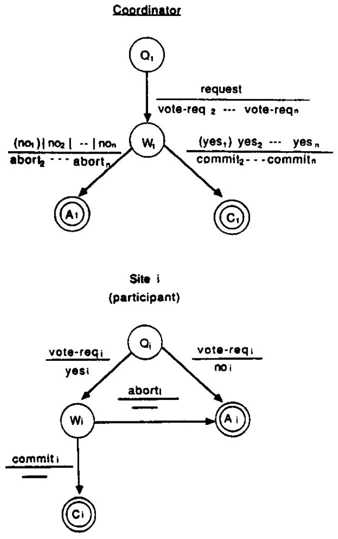


Fig. 1 The Two-Phase Commit Protocol.

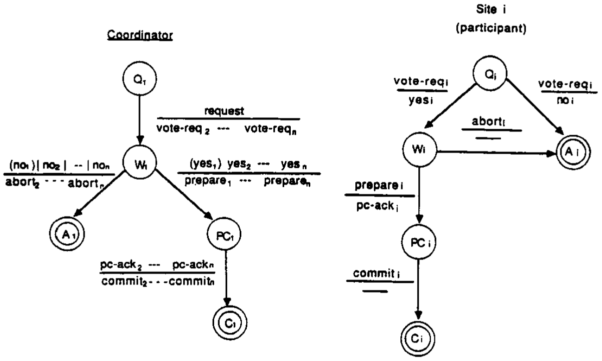


Fig. 2 The Three-Phase Commit Protocol.
(Adaptive from Dale Skeen [15].)

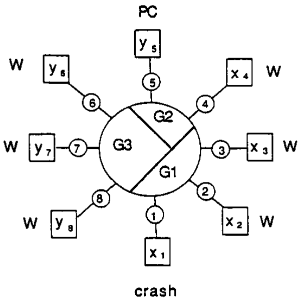
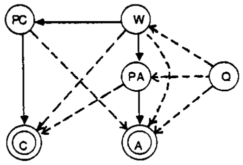


Fig. 3 Figure for Example 1.



---→ : indicates a path taken when a site does not participate in the formation of the quorum .

——→ : indicates a path taken when a site participates in the formation of the quorum .

Fig. 6 State Transition Diagram for the Termination Protocol.

Concurrent Partition States

Partition State

	PS1	PS2	PS3	PS4	PS5	PS6
PS1	○	○	○			
PS2	○	○	○	○	○	
PS3	○	○	○			
PS4		○		○	○	
PS5		○		○	○	○
PS6					○	○

PS1 : at least one participant in the partition is in the Q (initial) state and no participant in the partition is in the A (abort) state.

PS2 : all particiappnts in the partition are in the W (wait) state.

PS3 : at least one participant in the partition are in the A state.

PS4 : some participants in the partition are in the PC state, some participants are in the W state.

PS5 : all participants in the paartition are in the PC state.

PS6 : at least one participant in the partition is in the C (commit) state.

Fig. 4 The Concurreny Sets of Partitioin States.

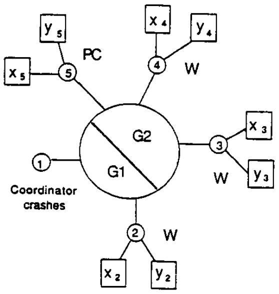
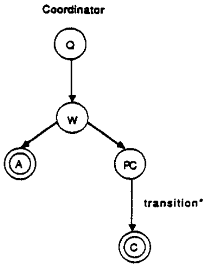


Fig. 7 Figure for Example 3.



For protocol 1, transition* requires w(x) votes of pc-acks' for every data item x in W (TR).

For protocol 2, transition* requires r(x) votes of pc-acks' for some data item x in W (TR).

Diagram for the participants is the same as 3PC.

Fig. 9 The Quorum-Based Commit Protocol.

PROTOCOL 1 PROTOTYPE

COORDINATOR

Phase 1

Request local states from all reachable participants

Phase 2

PARTICIPANTS' RESPONSES	COORDINATOR'S ACTIONS
if (≥ 1 commit state) OR (there are at least $w(x)$ votes for every data item x in W(TR) from participants in PC state)	send COMMIT commands to all reachable participants ; terminate ;
elseif (≥ 1 abort state or initial state) OR (there are at least $r(x)$ votes for some data item x in W(TR) from participants in PA state)	send ABORT commands to all reachable participants ; terminates ;
elseif (there exists a participant in PC state) AND (there are at least $w(x)$ votes for every data item x in W(TR) from participants not in PA state)	send PREPARE-TO-COMMIT to all participants in W state ; continue with (3a) ;
elseif (there are at least $r(x)$ votes for some data item x in W(TR) from participants not in PC state)	send PREPARE-TO-ABORT to all participants in W state ; continue with (3b) ;
else	block ;

Phase 3

```
(3a)
if     ( the participants which reply PC state in phase 1 and the participants which reply PC-ACK within
       the timeout period 2T in phase 2 constitute at least $w(x)$ votes for every data item x in W(TR) )
then   send COMMIT commands to all reachable participants ;
else   start the election protocol ;

(3b)
if     ( the participants which reply PA state in phase 1 and the participants which reply PA-ACK within
       the timeout period 2T in phase 2 constitute at least $r(x)$ votes for some data item x in W(TR) )
then   send ABORT commands to all reachable participants ;
else   start the election protocol ;
```

PARTICIPANTS

EVENTS	PARTICIPANTS' ACTIONS
(1) receive a request for local state of TR	send the local state of TR
(2) receive PREPARE-TO-COMMIT for transaction TR	*if* (TR is not in PA or Commit state) *then* enter PC state for transaction TR ; send PC-ACK back to the coordinator;
(3) receive PREPARE-TO-ABORT for transaction TR	*if* (TR is not in PC or abort state) *then* enter PA state for transaction TR ; send PA-ACK back to the coordinator ;
(4) receive COMMIT command for transaction TR	commit transaction TR and then terminate ;
(5) receive ABORT command for transaction TR	abort transaction TR and then terminate ;
(6) time out (* Occurs when the participant does not receive a response from the coordinator within 3T after sending a message to the coordinator *)	start the election protocol ;

Fig. 5 The Termination Protocol 1.

PROTOCOL 2 PROTOTYPE

COORDINATOR

Phase 1

Request local states from all reachable participants

Phase 2

PARTICIPANTS' RESPONSES	COORDINATOR'S ACTIONS
if (≥ 1 commit state) OR (there are at least r(x) votes for some data item x in W(TR) from participants in PC state)	send COMMIT commands to all reachable participants ; terminate ;
elseif (≥ 1 abort state or initial state) OR (there are at least w(x) votes for every data item x in W(TR) from participants in PA state)	send ABORT commands to all reachable participants ; terminates ;
elseif (there exists a participant in PC state) AND (there are at least r(x) votes for some data item x in W(TR) from participants not in PA state)	send PREPARE-TO-COMMIT to all participants in W state; continue with (3a) ;
elseif (there are at least w(x) votes for every data item x in W(TR) from participants not in PC state)	send PREPARE-TO-ABORT to all participants in W state ; continue with (3b) ;
else	block ;

Phase 3

```
(3a)
if    ( the participants which reply PC state in phase 1 and the participants which reply PC-ACK within
      the timeout period 2T in phase 2 constitute at least r(x) votes for some data item x in W(TR) )
then  send COMMIT commands to all reachable participants ;
else  start the election protocol ;

(3b)
if    ( the participants which reply PA state in Phase 1 and the participants which reply PA-ACK within
      the timeout period 2T in phase 2 constitute at least w(x) votes for every data item x in W(TR))
then  send ABORT commands to all reachable participants ;
else  start the election protocol ;
```

PARTICIPANTS

SAME AS TERMINATION PROTOCOL 1.

Fig. 8 The Termination Protocol 2.