Increasing the Resilience of Distributed and Replicated Database Systems [*]

Idit Keidar[†] Danny Dolev[‡]

Institute of Computer Science,
The Hebrew University of Jerusalem,
Jerusalem, Israel, 91904
E-mail: {idish,dolev}@cs.huji.ac.il
Url: http://www.cs.huji.ac.il/{~idish,~dolev}

Abstract

This paper presents a new atomic commitment protocol, *enhanced three phase commit* (*E3PC*), that *always* allows a quorum in the system to make progress. Previously suggested quorum-based protocols (e.g., the quorum-based *three phase commit (3PC)* [Ske82]) allow a quorum to make progress in case of one failure. If failures cascade, however, and the quorum in the system is "lost" (i.e., at a given time no quorum component exists), a quorum can later become connected and still remain blocked. With our protocol, a connected quorum never blocks. E3PC is based on the quorum-based 3PC [Ske82], and it does not require more time or communication than 3PC. We describe how this protocol can be exploited in a replicated database setting, making the database *always* available to a majority of the sites.

1 Introduction

Reliability and availability of loosely coupled distributed database systems are becoming requirements for many installations, and fault tolerance is becoming an important aspect of distributed systems design. When sites crash, or when communication failures occur, it is desirable to allow as many sites as possible to make progress. A common way to increase the availability of data and services is *replication*. If data are replicated in several sites, they can still be available despite site and communication-link failures. Protocols for transaction management in distributed and replicated database systems need to be carefully designed in order to guarantee database consistency. In this paper we present a novel *atomic commitment protocol* (*ACP*) that *always* allows a majority (or quorum) to make progress. We describe how this protocol can be exploited in a replicated database setting, making the database *always* available to a majority of the sites.

In distributed and replicated database systems, when a transaction spans several sites, the database servers at all sites have to reach a common decision regarding whether the transaction

[*]Preprint of a paper to appear in J. Comut. Systems Science (JCSS) special issue with selected papers from PODS 1995, December 1998

[†]Idit Keidar's research is supported by the Israeli Ministry of Science.

[‡]This work supported by the United States - Israel Binational Science Foundation, Grant 92-00189.

should be committed or not. A mixed decision results in an inconsistent database, while a unanimous decision guarantees the *atomicity* of the transaction (provided that the local server at each site can guarantee local atomicity of transactions). To this end an *atomic commitment protocol*, such as *two phase commit* (*2PC*) [Gra78] is invoked. The atomic commit problem and the two phase commit protocol are described in Section 3. Two phase commit is a *blocking* protocol: if the coordinator fails, all the sites may remain blocked indefinitely, unable to resolve the transaction.

To reduce the extent of blocking, Skeen suggested the quorum-based three phase commit (3PC) protocol, which maintains consistency in spite of network partitions [Ske82]. In case of failures, the algorithm uses a *quorum* (or *majority*)-based recovery procedure that allows a quorum to resolve the transaction. If failures cascade, however, and the quorum in the system is "lost" (i.e., at a certain time no quorum component exists), *a quorum of sites can become connected and still remain blocked.* Other previously suggested quorum-based protocols (e.g., [CR83, CK85]) also allow a quorum to make progress in case of one failure, while if failures cascade, a quorum can later become connected and still remain blocked. To our knowledge, the only previously suggested ACP that *always* allows a quorum to make progress is the ACP that we construct in [Kei94]. The protocol in [Kei94] is not straightforward; it uses a replication service as a building block, while the protocol presented in this paper is easy to follow and self-contained.

In this paper we present the *enhanced three phase commit* (*E3PC*) protocol, which is an enhancement of the quorum-based 3PC [Ske82]. E3PC maintains consistency in the face of site failures and network partitions: sites may crash and recover, and the network may partition into several *components*[1] and remerge. E3PC *always* allows a quorum to make progress: At any point in the execution of the protocol, if a group G of sites becomes connected, and this group contains a *quorum* and no subsequent failures occur for sufficiently long, then all the members of G eventually reach a decision. Furthermore, every site that can communicate with a site that has already reached a decision will also, eventually, reach a decision. An operational site that is not a member of a connected quorum may be *blocked*, i.e., may have to wait until a failure is repaired in order to resolve the transaction. This is undesirable but cannot be avoided; Skeen proved that every protocol that tolerates network partitions is bound to be blocking in certain scenarios [SS83].

E3PC achieves higher availability than 3PC simply by carefully maintaining two additional counters and with no additional communication. The principles demonstrated in this paper can be used to increase the resilience of a variety of distributed services, e.g., replicated database systems, by ensuring that a quorum will always be able to make progress. Other protocols that use two counters in order to allow a majority to make progress are given in [MHS89, CT96, KD96, Lam89, DLS88].

Numerous database replication schemes that are based on quorums have been suggested [Gif79, Her86, Her87, EASC85, EAT89]. These algorithms use *quorum systems* to determine when data objects are accessible. In order to guarantee the atomicity of transactions, these algorithms use an ACP and therefore are bound to block when the ACP they use blocks. Thus, with previously suggested ACPs, these approaches do not *always* allow a connected majority to update the database. Using E3PC these protocols can be made more resilient. In Section 6 we describe in detail how E3PC may be incorporated into *accessible copies* protocols [EASC85, EAT89], in order to make the database always available to a quorum.

[1]A **component** is sometimes called a **partition**. In our terminology, a partition splits the network into several components.

E3PC uses a *perfect* fault detector: Every site has accurate information regarding which sites are connected to it. In Section 7 we discuss *unreliable* failure detectors [CT96, DFKM96] and the ability of our protocol to work with such failure detectors. In this case, the protocol solves the *weak atomic commit* problem [Gue95].

The rest of this paper is organized as follows: Section 2 presents the computation model. Section 3 provides general background on the atomic commitment problem. The quorum-based three phase commit protocol [Ske82] is described in Section 4, and enhanced three phase commit is described in Section 5. In Section 6 we describe how E3PC can be exploited in replicated database systems. In Section 7 we describe the protocol's behavior with an *unreliable* failure detector. Section 8 concludes the paper. In Appendix A we formally prove the correctness of E3PC.

2 The Model

Our protocol is applicable in an asynchronous message-passing environment. The set of sites running the protocol is fixed and is known to all the sites. The sites are connected by an underlying communication network that provides communication between any pair of connected sites. We consider the following types of failures: failures may partition the network, and previously disjoint network components may remerge; messages may be lost or delivered out of order. Sites may crash and recover; recovered sites come up with their stable storage intact. We assume that messages are neither corrupted nor spontaneously generated by the network.

Failures are detected using a fault detector: Every site has accurate information regarding which sites are connected to it. This assumption is weakened in Section 7.

3 Background – Distributed Transaction Management

This section provides general background on the atomic commit problem and protocols.

3.1 Problem Definition

A distributed transaction is composed of several subtransactions, each running on a different site. The database manager at each site can unilaterally decide to ABORT the local subtransaction, in which case the entire transaction must be aborted. If all the participating sites agree to COMMIT their subtransaction (vote **Yes** on the transaction) and no failures occur, the transaction should be committed. We assume that the local database server at each site can atomically execute the subtransaction once it has agreed to COMMIT it.

In order to ensure that all the subtransactions are consistently committed or aborted, the sites run an *atomic commitment protocol* such as *two phase commit*. The requirements of atomic commitment (as defined in Chapter 7 of [BHG87]) are as follows:

AC1: **Uniform Agreement:** All the sites that reach a decision reach the same one.

AC2: A site cannot reverse its decision after it has reached one.

AC3: **Validity:** The COMMIT decision can be reached only if all sites voted **Yes**.

AC4: **Non-triviality:** If there are no failures and all sites voted **Yes**, then the decision will be to COMMIT.

AC5: **Termination:** At any point in the execution of the protocol, if all existing failures are repaired and no new failures occur for sufficiently long, then all sites will eventually reach a decision.

3.2 Two Phase Commit

The simplest and most renowned ACP is *two phase commit* [Gra78]. Several variations of 2PC have been suggested (e.g., presume abort and presume commit [MLO86]), the simplest version is centralized – one of the sites is designated as the *coordinator*. The coordinator sends a transaction (or request to prepare to commit) to all the participants. Each site answers by a **Yes** ("ready to commit") or by a **No** ("abort") message. If any site votes No, all the sites abort. The coordinator collects all the responses and informs all the sites of the decision. In absence of failures, this protocol preserves atomicity. Between the two phases, each site *blocks*, i.e., keeps the local database locked, waiting for the final word from the coordinator. If a site fails before its vote reaches the coordinator, it is usually assumed that it had voted No. If the coordinator fails in the first phase, all the sites remain blocked indefinitely, unable to resolve the last transaction. The centralized version of 2PC is depicted in Figure 1.

Coordinator	**Participant**
Transaction is received: Send sub-transactions.	
	Sub-transaction is received: Send reply – **Yes** or **No**.
If all sites respond **Yes**: Send COMMIT. If some site voted **No**: Send ABORT.	
	COMMIT or ABORT is received: Process accordingly.

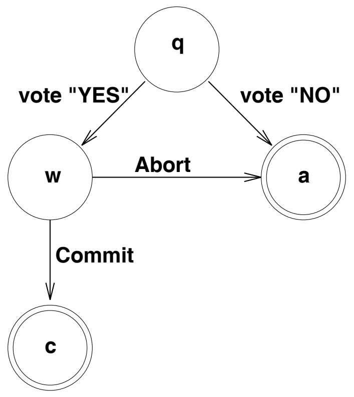


Figure 1: The Centralized Two Phase Commit Protocol

Commit protocols may also be described using state diagrams [SS83]. The state diagram for 2PC is shown in Figure 1. The circles denote states; final states are double-circled. The arcs represent state transitions, and the *action* taken (e.g., message sent) by the site is indicated next to each arc. In this protocol, each site (either coordinator or participant) can be in one of four possible states:

q : INITIAL state – A site is in the initial state until it decides whether to unilaterally abort or to agree to commit the transaction.

w : WAIT state – In this state the coordinator waits for votes from all of the participants, and each participant waits for the final word from the coordinator. This is the "uncertainty period" for each site, when it does not know whether the transaction will be committed or not.

c : COMMIT state – The site knows that a decision to commit was made.

a : ABORT state – The site knows that a decision to abort was made.

The states of a commit protocol may be classified along two orthogonal lines. In the first dimension, the states are divided into two disjoint subsets: The *committable* states and the *non-committable* states. A site is in a committable state only if it knows that all the sites have agreed to proceed with the transaction. The rest of the states are *non-committable.* The only committable state in 2PC is the COMMIT state. The second dimension distinguishes between *final* and *non-final* states. The *final* states are the ones in which a decision has been made and no more state transitions are possible. The final states in 2PC are COMMIT and ABORT.

3.3 Quorums

In order to reduce the extent of blocking in replication and atomic commit protocols, *majority* votes or *quorums* are often used. A *quorum system* is a generalization of the majority concept. E3PC, like Skeen's quorum-based three phase commit protocol [Ske82], uses a quorum system to decide when a group of connected sites may resolve the transaction. To enable maximum flexibility the quorum system may be elected in a variety of ways (e.g., weighted voting [Gif79]). The quorum system is *static*; it does not change in the course of the protocol.

The predicate $Q(S)$ is TRUE for a given subset S of the sites iff S is a quorum. The requirement from this predicate is that for any two sets of sites S and S' such that $S \cap S' = \emptyset$, at most one of $Q(S)$ and $Q(S')$ holds, i.e., every pair of quorums intersect. For example, in the simple majority quorum system $Q(S)$ is TRUE iff $|S| > n/2$, where n is the total number of sites running the protocol. Numerous quorum systems that fulfill these criteria were suggested. An analysis of the availability of different quorum systems may be found in [PW95].

For further flexibility, it is possible to set different quorums for commit and abort (this idea was presented in [Ske82]). In this case, a *commit quorum* of connected sites is required in order to commit a transaction, and an *abort quorum* is required to abort. For example, to increase the probability of commit in the system, one can assign smaller quorums for commit and larger ones for abort.

In this case, the quorum system consists of two predicates: $Q_C(G)$ is TRUE for a given group of sites G iff G is a commit quorum, and $Q_A(G)$ is TRUE iff G is an abort quorum. The requirement from these predicates is that for any two groups of sites G and G' such that $G \cap G' = \emptyset$, at most one of $Q_C(G)$ and $Q_A(G')$ holds, i.e., every commit quorum intersects every abort quorum.

3.4 The Extent of Blocking in Commit Protocols

The 2PC protocol is an example of a *blocking* protocol: operational sites sometimes wait on the recovery of failed sites. Locks must be held in the database while the transaction is blocked. Even though blocking preserves consistency, it is highly undesirable because the locks acquired by the blocked transaction cannot be relinquished, rendering the data inaccessible by other requests. Consequently, the availability of data stored in reliable sites can be limited by the availability of the weakest component in the distributed system.

Skeen *et al.* [SS83] proved that there exists no non-blocking protocol resilient to network partitioning. When a partition occurs, the best protocols allow no more than one group of sites to continue while the remaining groups block. Skeen suggested the quorum-based three phase commit protocol, which maintains consistency in spite of network partitions [Ske82]. This protocol is blocking in case of partitions; it is possible for an operational site to be blocked until a failure is mended. In case of failures, the algorithm uses a *quorum* (or *majority*)-based recovery procedure that allows a quorum to resolve the transaction. If failures cascade, however, *a quorum of sites can become connected and still remain blocked.* Skeen's quorum-based commit protocol is described in Section 4.

Since completely non-blocking recovery is impossible to achieve, further research in this area concentrated on minimizing the number of blocked sites when partitions occur. Chin *et al.* [CR83] define *optimal termination protocols (recovery procedures)* in terms of the average number of sites that are blocked when a partition occurs. The average is over all the possible partitions, and all the possible states in the protocol in which the partitions occurs. The analysis deals only with states in the basic commit protocol and ignores the possibility for cascading failures (failures that occur during the recovery procedure). It is proved that any ACP with optimal recovery procedures takes at least three phases and that the quorum-based recovery procedures are optimal.

In [Kei94] we construct an ACP that always allows a connected majority to proceed, regardless of past failures. To our knowledge, no other ACP with this feature was suggested. The ACP suggested in [Kei94] uses a reliable replication service as a building block and is mainly suitable for replicated database systems. In this paper, we present a novel commitment protocol, *enhanced three phase commit*, which always allows a connected majority to resolve the transaction (if it remains connected for sufficiently long). E3PC does not require complex building blocks, such as the one in [Kei94], and is more adequate for partially replicated or non-replicated distributed database systems; it is based on the quorum-based three phase commit [Ske82].

4 Quorum-Based Three Phase Commit

In this section we describe Skeen's quorum-based commit protocol [Ske82]. E3PC is a refinement of 3PC, and therefore we elaborate on 3PC before presenting E3PC. The basic *three phase commit* is described in Section 4.1, and the recovery procedure is described in Section 4.2. In Section 4.3 we show that with 3PC a connected majority of the sites can be blocked. We present a simplified version of 3PC that uses the same quorums for commit and abort.

4.1 Basic Three Phase Commit

The 3PC protocol is similar to two phase commit, but in order to achieve resilience, another non-final "buffer state" is added in 3PC, between the WAIT and the COMMIT states:

pc : PRE-COMMIT state – this is an intermediate state before the commit state and is needed to allow for recovery. In this state the site is still in its "uncertainty period."

The quorum-based 3PC is described in Figure 2, and a corresponding state diagram is depicted in Figure 3(a). The COMMIT and PRE-COMMIT states of 3PC are *committable* states; a site may be in one of these states only if it knows that all the sites have agreed to proceed with the transaction. The rest of the states are *non-committable*. In each step of the protocol, when the sites change their state, they must write the new state to *stable storage* before replying to the message that caused the state change.

Coordinator	**Participant**
Transaction is received: Send sub-transactions to participants.	
	Sub-transaction is received: Send reply – **Yes** or **No**.
If all sites respond **Yes**: Send PRE-COMMIT. If any site voted **No**: Send ABORT.	
	PRE-COMMIT received: Send ACK to coordinator.
Upon receiving a quorum of **ACKs**: Send COMMIT. Otherwise: Block (wait for more votes or until recovery)	
	COMMIT or ABORT is received: Process the transaction accordingly.

Figure 2: The Quorum-Based Three Phase Commit Protocol

4.2 Recovery Procedure for Three Phase Commit

When a group of sites detect a failure (a site crash or a network partition) or a failure repair (site recovery or merge of previously disconnected network components), they run the recovery procedure in order to try to resolve the transaction (i.e., commit or abort it). The recovery procedure consists of two phases: first elect a new coordinator, and next attempt to form a quorum that can resolve the transaction.

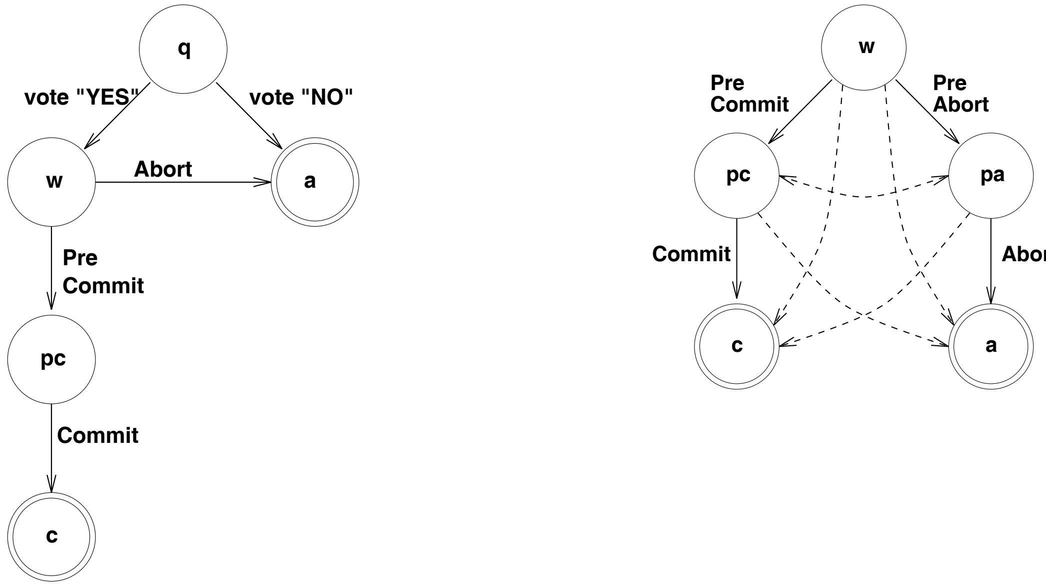
A new coordinator may be elected in different ways (e.g., [GM82]). In the course of the election, the coordinator hears from all the other participating sites. If there are failures (or recoveries) in the course of the election, the election can be restarted.[2]

The new coordinator tries to reach a decision whether the transaction should be committed or not and tries to form a quorum for its decision. The protocol must take the possibility of failures and failure repairs into account and, furthermore, must take into account the possibility of two (or more) different coordinators existing concurrently in disjoint network components. In order to ensure that the decision will be consistent, a coordinator must explicitly establish a quorum for a COMMIT or an ABORT decision. To this end, in the recovery procedure, another state is added:

pa : PRE-ABORT state. Dual state to PRE-COMMIT.

The recovery procedure is described in Figure 4. The state diagram for the recovery procedure is shown in Figure 3(b). The dashed lines represent transitions in which this site's state was not used in the decision made by the coordinator. Consider for example the following scenario: site p_1 reaches the PRE-ABORT state during an unsuccessful attempt to abort. The network then partitions, and p_1 remains blocked in the PRE-ABORT state. Later, a quorum (that does not include p_1) is formed, and another site, p_2, decides to COMMIT the transaction (this does not violate consistency, since

[2]Election is a *weaker* problem than atomic commitment, only the coordinator needs to know that it was elected, while the other sites may crash or detach without ever finding out which site was elected.



(a) The Basic Three Phase Commit (b) The Recovery Procedure

Figure 3: Three Phase Commit and the Recovery Procedure
q: INITIAL state; **w:** WAIT; **pc:** PRE-COMMIT; **c:** COMMIT; **pa:** PRE-ABORT; **a:** ABORT.

the attempt to abort has failed). If now p_1 and p_2 become connected, the coordinator must decide to COMMIT the transaction, because p_2 is COMMITTED already. Therefore, p_1 makes a transition from PRE-ABORT to COMMIT.

After collecting the states from all the sites, the coordinator tries to decide how to resolve the transaction. If any site has previously committed or aborted, then the transaction is immediately committed or aborted accordingly. Otherwise, the coordinator attempts to establish a quorum. A COMMIT is possible if at least one site is in the PRE-COMMIT state and the group of sites in the WAIT state together with the sites in the PRE-COMMIT state form a quorum. An ABORT is possible if the group of sites in the WAIT state together with the sites in the PRE-ABORT state form a quorum. The decision rule is summarized in Figure 5.

4.3 Three Phase Commit Blocks a Quorum

In this section we show that in the algorithm described above, it is possible for a quorum to become connected and still remain blocked. In our example, there are three sites executing the transaction: p_1, p_2, and p_3. The quorum system we use is a simple majority: every two sites form a quorum. Consider following the scenario depicted in Figure 6:

p_1 is the coordinator. All the sites vote **Yes** on the transaction. p_1 receives and processes the votes, but p_2 and p_3 detach from p_1 before receiving the PRE-COMMIT message sent by p_1.

p_2 is elected as the new coordinator. It sees that both p_2 and p_3 are in the WAIT state and therefore sends a PRE-ABORT message, according to the decision rule. p_3 receives the PRE-ABORT message, acknowledges it, and then detaches from p_2.

Now, p_3 is in the PRE-ABORT state, while p_1 is in the PRE-COMMIT state. If now p_1 and p_3 become connected, then according to the decision rule, they remain BLOCKED, even though they

1. Elect a new coordinator, r.
2. The coordinator, r, collects the states from all the connected sites.
3. The coordinator tries to reach a decision, as described in Figure 5. The decision is computed using the states collected so far. The coordinator multicasts a message reflecting the decision.
4. Upon receiving a PRE-COMMIT or PRE-ABORT each participant sends an ACK to r.
5. Upon receiving a quorum of ACKs for PRE-COMMIT or PRE-ABORT, r multicasts the corresponding decision: COMMIT or ABORT.
6. Upon receiving a COMMIT or an ABORT message: Process the transaction accordingly.

Figure 4: The Quorum-Based Recovery Procedure for Three Phase Commit

Collected States	Decision
$\exists$ ABORTED	ABORT
$\exists$ COMMITTED	COMMIT
$\exists$ PRE-COMMITTED $\wedge$ Q(sites in WAIT and PRE-COMMIT states)	PRE-COMMIT
Q(sites in WAIT and PRE-ABORT states)	PRE-ABORT
Otherwise	BLOCK

Figure 5: The Decision Rule for The Quorum-Based Recovery Procedure

form a quorum.

Analysis

In this example, it is actually safe for p_1 and p_3 to decide PRE-ABORT, because none of the sites could have committed, but it is *not* safe for them to decide PRE-COMMIT, because p_3 cannot know whether p_2 has aborted or not.

We observe that p_3 decided PRE-ABORT "after" p_1 decided PRE-COMMIT, and therefore we can conclude that the PRE-COMMIT decision made by p_1 is "stale", and no site has actually reached a COMMIT decision following it, because otherwise, it would have been impossible for p_2 to reach a PRE-ABORT decision.

The 3PC protocol does not allow a decision in this case, because the sites have no way of knowing which decision was made "later." Had the sites known that the a PRE-ABORT decision was made "later," they could have decided PRE-ABORT again and would have eventually ABORTED the transaction. In E3PC, we provide the mechanism for doing exactly this.

5 The E3PC Protocol

We suggest a three phase atomic commitment protocol, *enhanced three phase commit*, with a novel quorum-based recovery procedure that always allows a quorum of sites to resolve the transaction, even in the face of cascading failures. The protocol is based on the quorum-based three phase

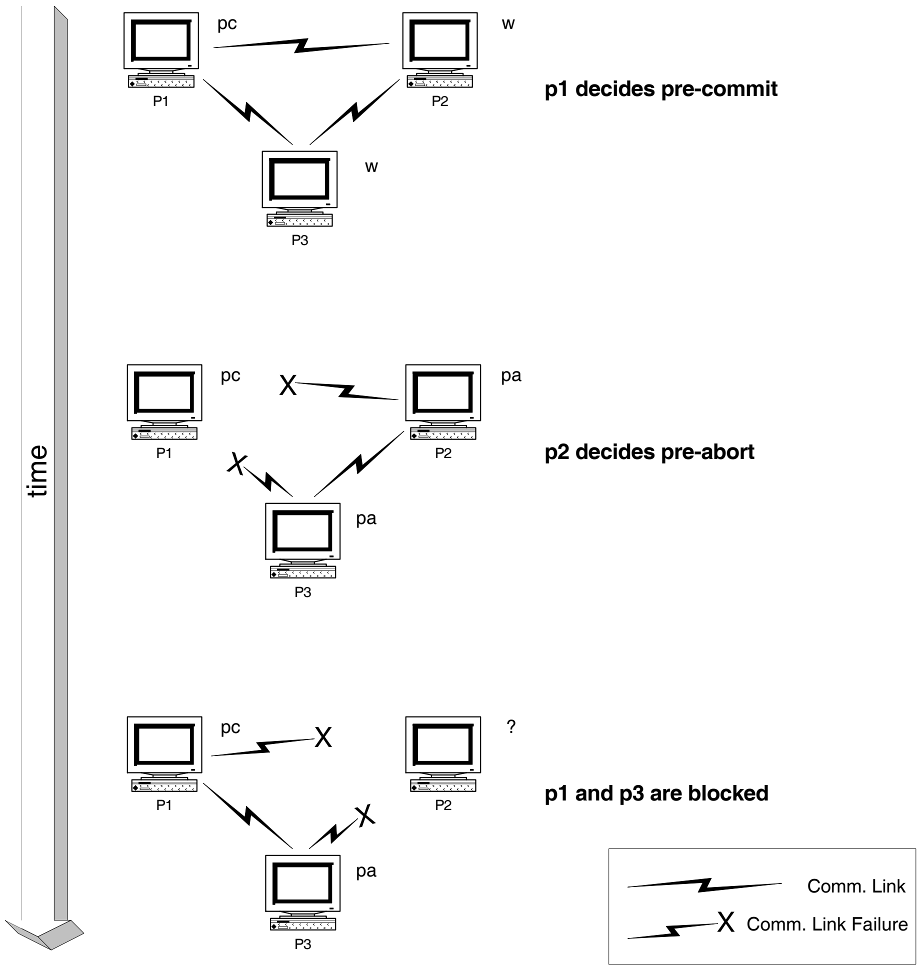


Figure 6: Three Phase Commit Blocks a Quorum

commit protocol [Ske82]. E3PC does not require more communication or time than 3PC; the improved resilience is achieved simply by maintaining two additional counters, which impose a *linear order* on quorums formed in the system.

Initially, the basic E3PC is invoked. If failures occur, the sites invoke the recovery procedure and elect a new coordinator. The new coordinator carries on the protocol to reach a decision. If failures cascade, the recovery procedure may be reinvoked an arbitrary number of times. Thus, one *execution* of the protocol (for one transaction) consists of one *invocation* of the *basic E3PC* and of zero or more *invocations* of the recovery procedure.

In Section 5.1 we describe how E3PC enhances 3PC. The recovery procedure for E3PC is described in Section 5.2. In Section 5.3 we show that E3PC does not block a quorum in the example of Section 4.3. In Section 5.4 we outline the correctness proof for E3PC. We first present a simplified version of E3PC that uses the same quorums for commit and abort. In Section 5.5 we describe a more general version of E3PC, which uses different quorums for commit and abort.

5.1 E3PC: Enhancing Three Phase Commit

The basic E3PC is similar to the basic 3PC, the only difference being that E3PC maintains two additional counters. We now describe these counters. In each invocation of the recovery procedure, the sites try to elect a new coordinator. The coordinators elected in the course of an execution of the protocol are sequentially numbered: A new "election number" is assigned in each invocation of the recovery procedure. Note that there is no need to elect a new coordinator in each invocation of the basic 3PC or E3PC; the re-election is needed only in case failures occur. The coordinator of the basic E3PC is assigned "election number" one, even though no elections actually take place. The following two counters are maintained by the basic E3PC and by the recovery procedure:

Last_Elected - The number of the last election that this site took part in. This variable is updated when a new coordinator is elected. This value is initialized to *one* when the basic E3PC is invoked.

Last_Attempt - The election number in the last attempt this site made to commit or abort. The coordinator changes this variable's value to the value of *Last_Elected* whenever it makes a decision. Every other participant sets its *Last_Attempt* to *Last_Elected* when it moves to the PRE-COMMIT or to the PRE-ABORT state, following a PRE-COMMIT or a PRE-ABORT message from the coordinator. This value is initialized to *zero* when the basic E3PC is invoked.

These variables are logged on stable storage. The second counter, *Last_Attempt*, provides a *linear order* on PRE-COMMIT and PRE-ABORT decisions; e.g., if some site is in the PRE-COMMIT state with its *Last_Attempt* $= 7$, and another site is in the PRE-ABORT state with its *Last_Attempt* $= 8$, then the PRE-COMMIT decision is "earlier" and therefore "stale," and the PRE-ABORT decision is safe. The first counter, *Last_Elected*, is needed to guarantee the uniqueness of the *Last_Attempt*, [3] i.e., that two different attempts will not be made with the same value of *Last_Attempt* (cf. Lemma 3 in Appendix A).

Notation

We use the following notation:

- $\mathcal{P}$ is the group of sites that are live and connected, and which take part in the election of the new coordinator.
- *Max_Elected* is $\max_{p\in\mathcal{P}}$(*Last_Elected* of p).
- *Max_Attempt* is $\max_{p\in\mathcal{P}}$(*Last_Attempt* of p).
- *Is_Max_Attempt_Committable* is a predicate that is TRUE iff all the members that are in non-final states and whose *Last_Attempt* is equal to *Max_Attempt* are in a committable state (i.e., in the PRE-COMMIT state). Formally, *Is_Max_Attempt_Committable* is TRUE iff $\forall_{p\in\mathcal{P}}$(*Last_Attempt* of p = *Max_Attempt* $\wedge p$ is in a non-final state $\rightarrow p$ is in a committable state)

[3] The value of *Last_Elected* is not guaranteed to be unique, two elections may be made with the same value of *Last_Elected*, in case the first election with this number did not terminate successfully at all the members. Also note that the same coordinator can not be chosen with the same election number twice.

5.2 Quorum-Based Recovery Procedure

1. Elect a new coordinator r. The election is non-blocking, it is restarted in case of failure. In the course of the election, r hears from all the other sites their values of *Last_Elected* and *Last_Attempt* and determines *Max_Elected* and *Max_Attempt*. r sets *Last_Elected* to *Max_Elected*+1 and notifies the sites in $\mathcal{P}$ of its election, and of the value of *Max_Elected*.
2. Upon hearing *Max_Elected* from r, set *Last_Elected* to *Max_Elected*+1 and send local state to the coordinator r.
3. The coordinator, r collects states from the other sites in $\mathcal{P}$, and tries to reach a decision as described in Figure 8. The decision is computed using the states collected so far, we denote by $\mathcal{S}$ the subset of sites from which r received the state so far. Upon reaching a decision other than BLOCK, r sets *Last_Attempt* to *Last_Elected*, and multicasts the decision to all the sites in $\mathcal{P}$.
4. Upon receiving a PRE-COMMIT or PRE-ABORT each participant sets its *Last_Attempt* to *Last_Elected* and sends an ACK to r.
5. Upon receiving a quorum of ACKs for PRE-COMMIT (PRE-ABORT), r multicasts the decision: COMMIT (ABORT).
6. Upon receiving a COMMIT (ABORT) message from r: process the transaction accordingly.

Figure 7: The Recovery Procedure for E3PC

As in 3PC, the recovery procedure is invoked when failures are detected and when failures are repaired. Sites cannot "join" the recovery procedure in the middle, instead, the recovery procedure must be reinvoked to let them take part.

All the messages sent by the protocol carry the election number (*Last_Elected*) and process id of the coordinator. Thus, it is possible to know in which invocation of the protocol each message was sent. A site that hears from a new coordinator ceases to take part in the previous invocation that it took part in and no longer responds to its previous coordinator. Messages from previous invocations are ignored. Thus, a site cannot concurrently take part in two invocations of the recovery procedure. Furthermore, if a site responds to messages from the coordinator in some invocation, it necessarily took part in the election of that coordinator.

The recovery procedure for E3PC is similar to the quorum-based recovery procedure described in Section 4.2. As in 3PC, in each step of the recovery procedure, when the sites change their state, they must write the new state to stable storage before replying to the message that caused the state change. The recovery procedure is described in Figure 7. The possible state transitions in E3PC and its recovery procedure are the same as those of 3PC, depicted in Figure 3; the improved performance in E3PC results from the decision rule, which allows state transitions in more cases.

In Step 3 of the recovery procedure, r collects the states from the other sites in $\mathcal{P}$ and tries to reach a decision. The sites are blocked until r receives enough states to allow a decision. It is possible to reach a decision before collecting the states from all the sites in $\mathcal{P}$; e.g., when a final state is received, a decision can be made. It is also possible to reach a decision once states are collected from a quorum, if one of the quorum members has *Last_Attempt=Max_Attempt*. We denote by $\mathcal{S}$ the subset of $\mathcal{P}$ from which r received the state so far; r constantly tries to compute the decision using the states in $\mathcal{S}$, whenever new states arrive and until a decision is reached. The decision rule is described below. If the decision is not BLOCK, r changes *Last_Attempt* to *Last_Elected*, and multicasts the decision to all the sites in $\mathcal{P}$.

Decision Rule

Collected States	Decision
$\exists$ ABORTED	ABORT
$\exists$ COMMITTED	COMMIT
Is_Max_Attempt_Committable $\wedge Q(\mathcal{S})$	PRE-COMMIT
$\neg$*Is_Max_Attempt_Committable* $\wedge Q(\mathcal{S})$	PRE-ABORT
Otherwise	BLOCK

Figure 8: The Decision Rule for E3PC

The coordinator collects the states from the live members of $\mathcal{P}$ and applies the following decision rule to the subset $\mathcal{S}$ of sites from which it received the state.

- If there exists a site (in $\mathcal{S}$) that is in the ABORTED state – ABORT.
- If there exists a site in the COMMITTED state – COMMIT.
- If *Is_Max_Attempt_Committable* is TRUE, and $\mathcal{S}$ is a quorum – PRE-COMMIT.
- If *Is_Max_Attempt_Committable* is FALSE and $\mathcal{S}$ is a quorum – PRE-ABORT.
- Otherwise – BLOCK.

The decision rule is summarized in Figure 8. It is easy to see that with the new decision rule, if a group of sites is a quorum, it will never be blocked.

5.3 E3PC does not Block a Quorum

In E3PC, if a group of sites forms a quorum, it will never be blocked. This is obvious from the decision rule: if some site has previously committed (aborted), then the decision is COMMIT (ABORT). Otherwise, a decision can always be made according to the value of *Is_Max_Attempt_Committable*.

We now demonstrate that E3PC does not block with the scenario of Section 4.3 (in which Skeen's quorum-based 3PC does block). In this example, there are three sites executing the transaction - p_1, p_2, and p_3 - and the quorum system is a simple majority: every two sites form a quorum. We considered the following scenario, depicted in Figure 9:

- Initially, p_1 is the coordinator. All the sites vote **Yes** on the transaction. p_1 receives and processes the votes, but p_2 and p_3 detach from p_1 before receiving the PRE-COMMIT message sent by p_1. Now *Last_Attempt*$_{p_1}$ is 1 while *Last_Attempt*$_{p_2}$ = *Last_Attempt*$_{p_3}$ = 0, and the value of *Last_Elected* is one for all the sites.

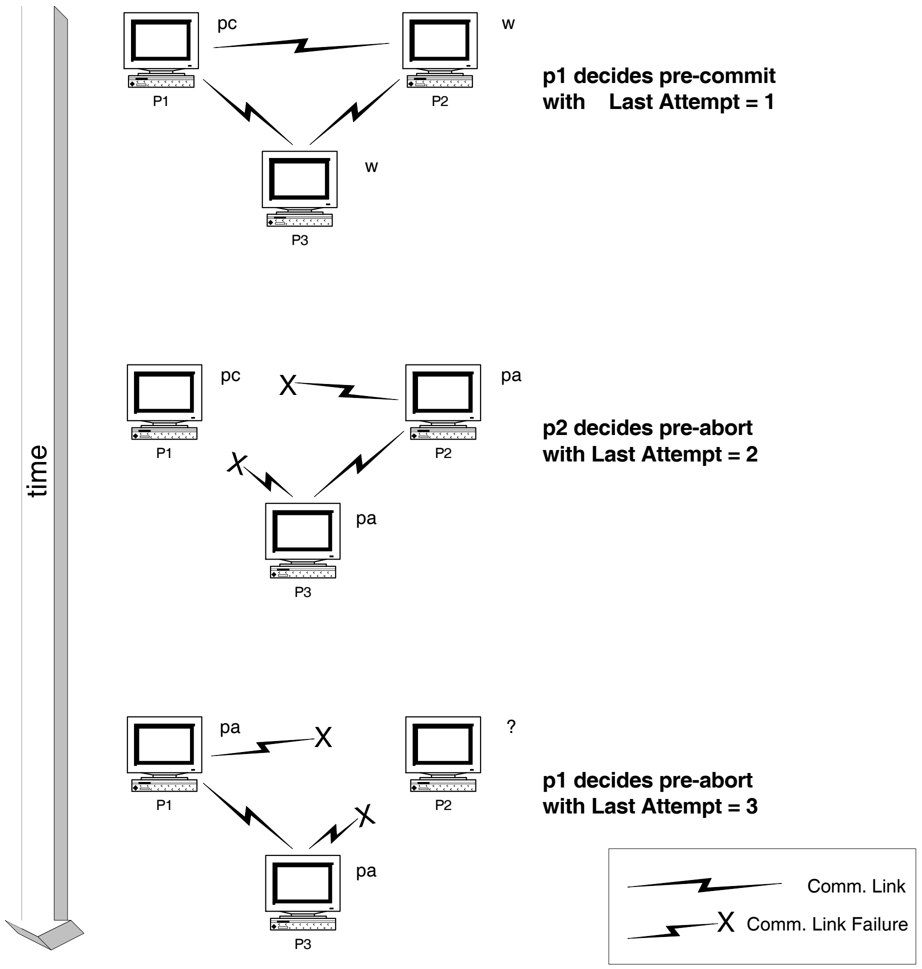


Figure 9: E3PC does not Block a Quorum

- p_2 is elected as the new coordinator, and the new *Last_Elected* is two. It sees that both p_2 and p_3 are in the WAIT state and therefore sends a PRE-ABORT message, according to the decision rule, and moves to the PRE-ABORT state while changing its *Last_Attempt* to two. p_3 receives the PRE-ABORT message, sets its *Last_Attempt* to two, sends an acknowledgment, and detaches from p_2.
- Now, p_3 is in the PRE-ABORT state with its value of *Last_Attempt* $= 2$, while p_1 is in the PRE-COMMIT state with its *Last_Attempt* $= 1$. If now p_1 and p_3 become connected, then, according to the decision rule, they decide to PRE-ABORT the transaction, and they do *not* remain blocked.

5.4 Correctness of E3PC

In Appendix A we formally prove that E3PC fulfills the requirements of atomic commitment described in Section 3.1. In this section we outline the proof.

First we prove that two contradicting attempts (i.e., PRE-COMMIT and PRE-ABORT) cannot be made with the same value of *Last_Attempt* (Lemma 3). This is true due to the fact that every two quorums intersect and that a quorum of sites must increase *Last_Elected* before a PRE-COMMIT or a PRE-ABORT decision. Moreover, *Last_Attempt* is set to the value of *Last_Elected*, which is higher than the previous value of *Last_Elected* of all the participants of the recovery procedure. Next, we prove that the value of *Last_Attempt* at each site increases every time the site changes state from a committable state to a non-final non-committable state, and vice versa (Lemma 5).

Using the two lemmas above we prove (Lemmas 6 and 8): If the coordinator reaches a COMMIT (ABORT) decision upon receiving a quorum of ACKs for PRE-COMMIT (PRE-ABORT) when setting its *Last_Attempt* to i, then for every $j \geq i$ no coordinator will decide PRE-ABORT (PRE-COMMIT) when setting its *Last_Attempt* to j. We prove these lemmas by induction on $j \geq i$; we show, by induction on j, that if some coordinator r sets its *Last_Attempt* to j in Step 3 of the recovery procedure, then *Is_Max_Attempt_Committable* is TRUE (FALSE) in this invocation of the recovery procedure, and therefore, the decision is PRE-COMMIT (PRE-ABORT).

We conclude that if some site running the protocol COMMITS the transaction, then no other site ABORTS the transaction.

5.5 Using Different Quorums for Commit and Abort

In this section we describe how to generalize E3PC to work with different quorums for commit and abort. Commit and abort quorums are described in Section 3.3. The following changes need to be made in the protocol:

Collected States	Decision
$\exists$ ABORTED	ABORT
$\exists$ COMMITTED	COMMIT
Is_Max_Attempt_Committable $\wedge Q_C(\mathcal{S})$	PRE-COMMIT
$\neg$*Is_Max_Attempt_Committable* $\wedge Q_A(\mathcal{S})$	PRE-ABORT
Otherwise	BLOCK

Figure 10: The Decision Rule for E3PC with Commit and Abort Quorums

1. In the second phase of the basic E3PC, the coordinator waits for a commit quorum of ACKs before sending PRE-COMMIT.

2. In Step 5 of the recovery procedure, the coordinator needs to wait for a commit quorum of ACKs in order to PRE-COMMIT, and for ACKs from an abort quorum in order to PRE-ABORT.

3. Likewise, the decision rule is slightly changed to require a commit quorum in order to PRE-COMMIT (in case *Is_Max_Attempt_Committable* is TRUE) and an abort quorum in order to PRE-ABORT (if *Is_Max_Attempt_Committable* is FALSE). The resulting decision rule is shown in Figure 10.

It is easy to see from the new decision rule that if a group of processes is both a commit quorum and an abort quorum, it does not remain blocked.

The correctness proof of the general version of E3PC is similar to the correctness proof of E3PC presented in this paper; we use the property that every commit quorum intersects every abort quorum in order to prove that two contradicting attempts (i.e., PRE-COMMIT and PRE-ABORT) cannot be made with the same value of *Last_Attempt*. The formal proof may be found in [KD94].

6 Replicated Database Systems

In replicated database systems, the sites continuously execute transactions. When the network partitions, it is often desirable to allow a quorum of the sites to access the database, but it is usually undesirable to allow sites in two disjoint network components to concurrently update the same data. Numerous replication schemes that are based on quorums have been suggested [Gif79, Her86, Her87, EASC85, EAT89]. In order to guarantee the atomicity of transactions, these algorithms use an ACP and therefore are bound to block when the ACP they use blocks. We propose to use E3PC in conjunction with these protocols in order to make the database *always* available to a quorum.

The same quorum system should be used to determine when the data are accessible to a group of sites as for the atomic commitment protocol. In a fully replicated database, a group of sites needs to be a quorum of the total number of sites in order to access the database. Hence, in order to resolve a transaction using the E3PC recovery procedure, a group of sites needs to be a quorum of the total number of sites and not just of the sites that invoked E3PC for the specific transaction.

If the data are *partially replicated*, then for each item accessed by this transaction, a quorum of the sites it resides on is required. In order to resolve a transaction using the E3PC recovery procedure, a group of sites needs to contain a quorum for each item accessed by this transaction.

There is a subtle point to consider with this solution: sites that did not take part in the basic E3PC for this transaction may take part in the recovery procedure. The local databases at such sites are not up-to-date, since they do not necessarily reflect the updates performed by the current transaction. Therefore, these sites need to recover the database state from other sites during the merge and before taking part in the recovery procedure. In the *accessible copies* protocols [EASC85, EAT89], this is done every time the view changes. In this case, we suggest using the view change as the "fault detector" for E3PC; thus, the recovery procedure is always invoked following a view change, after all the participating sites have reached an up-to-date state. Below, we describe in detail how E3PC may be incorporated into *accessible copies* protocols.

6.1 Using E3PC with Accessible Copies Protocols

Accessible copies protocols [EASC85, EAT89] maintain a *view* of the system to determine when data are accessible: A data item can be read/written within a view (component) only if a majority of its read/write votes are assigned to copies that reside on sites that are members of this view. This majority of votes is the "accessibility threshold for the item," not to be confused with read and write quorums used within the current view. In order to guarantee the *atomicity* of each transaction, these protocols use an ACP. We propose to use E3PC as this ACP using these accessibility thresholds as its quorum system. This way the sites that succeed in resolving the previous transaction are also allowed to access the database in new transactions.

A group of sites is considered a quorum (in E3PC) if and only if it contains a majority of the votes of each item accessed by this transaction. A connected quorum of the sites may invoke a transaction and access the data. When the sites running the transaction wish to commit it, they run E3PC for the transaction. The basic E3PC may be invoked by a *subset* of the sites, the members of the current view. The views maintained by the accessible copies protocol are used as *fault detectors* for E3PC; when the view changes, the recovery procedure is invoked.

In the course of the view change protocol, each site executes an *update_transaction* in order to recover the most up-to-date values of each data item. If the *update_transaction* is aborted, the view change is aborted; a successful view change implies that the "newly joined" sites have successfully performed the updates and thus have given up their right to unilaterally abort the transaction. When the recovery procedure is invoked with sites that did not take part in the basic E3PC for the current transaction, these sites are considered to be in the *wait* state with their *Last_Elected*= 1 and *Last_Attempt*= 0, as if they had voted **Yes** on the transaction, and detached.

- The basic E3PC may be invoked by a *subset* of the sites, the members of the current view.
- E3PC uses view changes as its fault detector, i.e., every time the view changes, the recovery procedure is invoked.
- When the recovery procedure is invoked with "newly joined" sites that did not take part in the basic E3PC, the "newly joined" sites are considered to be in the *wait* state with their *Last_Elected*= 1 and *Last_Attempt*= 0.

Figure 11: E3PC Adjusted to the Accessible Copies Protocol

Figure 11 summarizes the adjustments made in E3PC to make it suitable for the accessible copies protocol. With this protocol, the database is always available to a quorum of connected sites. We know of no previous database replica control protocol with this feature.

7 Failure Detectors and Weak Atomic Commit

The E3PC protocol presented in this paper uses a *perfect* failure detector: Every site has accurate information regarding which sites are connected to it. This assumption is not practical: in asynchronous systems, it is not always possible to tell failed sites from very slow ones. In practice, systems use unreliable mechanisms, e.g., timeout, in order to detect faults. Such mechanisms may make mistakes and *suspect* that a *correct (connected)* site is *faulty (disconnected)*.

Can we relax the *perfect* failure detection assumption? Guerraoui [Gue95] proves that the Atomic Commit Problem, as defined in Section 3.1, cannot be solved without a perfect failure detector; the non-triviality requirement (AC4) is too strong. He defines the *weak atomic commit problem* by changing the non-triviality requirement of atomic commit as follows:

Non-Triviality: If all sites voted **Yes**, and no site is ever *suspected*, then the decision will be to COMMIT.

The other requirements of atomic commit are unchanged. The weak atomic commit problem can be solved with non-perfect failure detectors.

Can the weak atomic commit problem be solved in a fully asynchronous environment that is not augmented with any failure detector? Unfortunately, the answer to this question is no. In a fully asynchronous environment, reaching consensus[4] is impossible [FLP85], in the sense that every protocol that reaches agreement is bound to have an infinite run. In particular, using any failure detector that can be implemented in such an environment, e.g., a time-out mechanism, E3PC does not fulfill the termination (AC5) requirement. However, when the protocol does terminate, the rest of the requirements of weak atomic commit are preserved.

7.1 Failure Detector Classes

We have seen that in order to solve weak atomic commit, the model must be augmented with some failure detector. Chandra and Toueg [CT96] classify failure detectors with different levels of *reliability*. These failure detector classes are defined in a crash-failure asynchronous environment. In [DFKM96] these definitions are extended to the model where network partitions may occur.

An *eventual perfect* failure detector (formally defined in [CT96] and [DFKM96]) may suspect correct sites, but there is a time after which correct sites are no longer suspected. Using such a failure detector, E3PC solves the weak atomic commit problem. E3PC terminates once a quorum of sites becomes connected and no failures or suspicions occur for sufficiently long. In a practical system, this assumption is likely to be fulfilled.

Similarly, [CT96] and [DFKM96] define weaker classes of failure detectors; Chandra *et al.* [CHT92] prove that the weakest possible failure detector to solve consensus is the *eventual weak* failure detector. Intuitively, an eventual weak failure detector may make mistakes and suspect correct sites, but there is a time after which there is some correct site that is not suspected by any other site that is connected to it. Guerraoui and Schiper [GS95] present a solution to the weak atomic commit problem in an environment without network partitions, using an *eventual weak* failure detector. Their protocol may be adapted to work in an environment with network partitions, using the technique presented in [DFKM96]. This technique yields a protocol that is less efficient (requiring more communication) than E3PC.

8 Conclusions

In this paper we demonstrated how the three phase commit [Ske82] protocol can be made more resilient simply by maintaining two additional counters and by changing the decision rule. The new protocol, E3PC, *always* allows a quorum of connected sites to resolve a transaction: At any point in the execution of the protocol, if a group G of sites becomes connected and this group contains a quorum of the sites, and no subsequent failures occur for sufficiently long, then all the members of G eventually reach a decision. Furthermore, every site that can communicate with a site that already reached a decision will also, eventually, reach a decision. We have shown that 3PC does not possess this feature: if the quorum in the system is "lost" (i.e., at a certain time no quorum component exists), a quorum can later become connected and still remain blocked.

E3PC does not require more communication or time than 3PC; the improved resilience is achieved simply by maintaining two additional counters. The information needed to maintain

[4]Guerraoui [Gue95] proves that the Weak Atomic Commit problem is reducible to consensus.

the counters is piggybacked on messages that are sent in 3PC as well as in E3PC: the values of *Last_Elected* and *Last_Attempt* are attached to messages used to elect a new coordinator.

We discussed how E3PC can be extended to work in an environment with unreliable failure detectors. In this case, the protocol solves the *weak* atomic commitment problem.

E3PC may be used in conjunction with quorum-based replication protocols, such as [Gif79, Her86, Her87, EASC85, EAT89], in order to make the database always available to a quorum. We demonstrated how E3PC may be incorporated in *accessible copies* protocols [EASC85, EAT89]; with the new protocol, the database is always available to a quorum of connected sites. The technique demonstrated here may be used to make other algorithms more resilient, e.g., an algorithm for maintaining a *primary component* in the network, to support processing of sequences of distributed transactions, as well as for ordering of messages [KD96] and replication [Kei94]. In [YLKD97, DKYL96] we exploit this technique in a *dynamic voting* scheme for maintaining the primary component in the network.

Acknowledgment

We thank the referees that reviewed this paper for their helpful comments.

A Correctness Proof of E3PC

In this section we prove the correctness of E3PC; we show that E3PC and its recovery procedure fulfill the requirements of atomic commitment (as defined in Chapter 7 of [BHG87]) described in Section 3.1. The proof follows:

AC1: **Uniform Agreement:** In Theorem 1 below we will prove that all the sites that reach a decision reach the same one.

AC2: In our protocol, a site cannot reverse its decision after it has reached one. When a site in a final state (COMMIT or ABORT) participates in some invocation of the recovery procedure, the decision in this invocation of the recovery procedure will correspond with its state.

AC3: **Validity:** The COMMIT decision can be reached only if all sites voted **Yes**: In the basic E3PC, a committable decision can be made only if all the sites vote **Yes**. If the recovery procedure is invoked with no site in a committable state, then according to the decision rule, a committable decision cannot be reached.

AC4: **Non-triviality:** If there are no suspicions during the execution of basic E3PC, then the basic E3PC succeeds in reaching a decision. If all sites voted **Yes**, then the decision is COMMIT. Since we assume a *perfect* failure detector, if there are no failures, there are no suspicions.

Without a *perfect* failure detector, the weak non-triviality requirement (defined in [Gue95] and Section 7) is fulfilled.

AC5: **Termination:** At any point in the execution of the protocol, if all existing failures are repaired and no new failures occur for sufficiently long, then all sites will eventually reach a decision. Our protocol guarantees a much stronger property:

At any point in the execution of the protocol, if a group G of sites becomes connected and this group contains a quorum of the sites, and no subsequent failures occur for sufficiently long, then all the members of G eventually reach a decision. Furthermore, every site that can communicate with a site that already reached a decision will also, eventually, reach a decision.

This property is immediate from the decision rule and from our assumption that the failure detector is *perfect*. This property is also fulfilled with an *eventual perfect* failure detector, since with such a failure detector, there is a time after which correct sites are no longer suspected.

We now prove that the decision made is *unanimous*, i.e., that if one site decides to COMMIT, then no site can decide to ABORT and vice versa.

Lemma 1 *If a coordinator r sets its local value of* Last_Attempt *to i and sends a* PRE-COMMIT (PRE-ABORT) *message to the participants in Step 3 of the recovery procedure, then a quorum of sites have set their value of* Last_Elected *to i during the same invocation of the recovery procedure.*

Proof. It is immediate from the protocol and from the fact that sites cannot "join" the recovery procedure in the middle, but rather the protocol must be reinvoked to let them take part. □

Lemma 2 *At each site, the value of* Last_Elected *never decreases.*

Proof. The value of *Last_Elected* is modified only in Step 2 of the recovery procedure, when it is changed to *Max_Elected*+1. A site may execute Step 2 only if it took part in the election of the coordinator in that invocation of the recovery procedure and its value of *Last_Elected* was used to compute *Max_Elected*, and therefore *Max_Elected*≥*Last_Elected*, and *Last_Elected* increases. □

Lemma 3 *If two sites, p and q, both set their* Last_Attempt *to the number i without changing to a final state, then either both of them set their* Last_Attempt *to i as a response to a* PRE-COMMIT *decision or both of them set their* Last_Attempt *to i as a response to a* PRE-ABORT *decision.*

Proof. A coordinator changes the value of *Last_Attempt* when it reaches a decision (in Step 3 of the recovery procedure or in the basic E3PC), and it remains in a non-final state if the decision is PRE-COMMIT or PRE-ABORT. Other sites change the value of *Last_Attempt* only in response to a PRE-COMMIT or a PRE-ABORT decision, in Step 4 of the recovery procedure, or in response to PRE-COMMIT in the basic E3PC.

Assume the contrary; then w.l.o.g., p set its *Last_Attempt* to i in response to a PRE-COMMIT decision in the course of some invocation, I_0, of the recovery procedure (or of the basic E3PC), and q, in response to a PRE-ABORT decision, in an invocation I_1. From Lemma 1, a quorum of sites set their *Last_Elected* to i in invocation I_0 and another quorum of sites set their *Last_Elected* to i in invocation I_1. Since the coordinator in invocation I_0 decided to PRE-COMMIT and the coordinator in I_1 decided to PRE-ABORT, I_0 and I_1 were *different* invocations of recovery procedure or of the basic E3PC.

Since every two quorums intersect, there exists a site, s, that set its *Last_Elected* to i in both invocations. W.l.o.g., s set its *Last_Elected* to i in I_0 before setting it to i in I_1. From the protocol, a site cannot concurrently take part in two invocations of the recovery procedure, furthermore, if a site responds to messages from the coordinator in some invocation, it necessarily took part in the election of that coordinator. Therefore, s took part in the election of the coordinator in I_1, *after* it set its *Last_Elected* to i, and from Lemma 2, in the course of the election, the coordinator heard from s that its value of *Last_Elected*$\geq i$ and determined that *Max_Elected*$\geq i$. The new value of *Last_Elected* for this invocation was *Max_Elected*+1, which is greater than i, which contradicts our assumption. □

Lemma 4 *At each site, at any given time,* Last_Elected≥Last_Attempt.

Proof. From Lemma 2, the value of *Last_Elected* never decreases, so it is sufficient to show that *Last_Attempt* is never increased to exceed it. We prove this by induction on the steps of the protocol in which *Last_Attempt* changes. *Base:* When E3PC is initiated, *Last_Elected* is set to one, and *Last_Attempt*, to zero. *Step:* Whenever *Last_Attempt* is changed in the course of the protocol, it takes the value of *Last_Elected.* □

Lemma 5 *The value of* Last_Attempt *at each site increases every time the site changes state from a committable state to a non-final, non-committable state and vice versa. The value of* Last_Attempt *never decreases.*

Proof. The only non-final committable state is PRE-COMMIT, and the only way to switch to a PRE-COMMIT state is in response to a PRE-COMMIT decision, when setting *Last_Attempt* to *Last_Elected.* Likewise, the only way to switch from a committable state to a non-final non-committable state is in Step 3 or in Step 4 of the recovery procedure, in response to a PRE-ABORT decision, when setting *Last_Attempt* to *Last_Elected.*

It is sufficient to prove that *Last_Attempt* increases when it is set to *Last_Elected* in Step 3 or 4 of the recovery procedure, i.e., that *Last_Attempt*<*Last_Elected* before Step 3. And indeed, in Step 2, *Last_Elected* is set to *Max_Elected*+1, which is greater than the value of *Last_Elected* was when the recovery procedure was initialized. From Lemma 4, *Last_Elected*≥*Last_Attempt* at all times, therefore, before Step 3, *Last_Elected* is greater than *Last_Attempt.* □

Lemma 6 *If the coordinator reaches a* COMMIT *decision upon receiving a quorum of ACKs for* PRE-COMMIT *when setting its* Last_Attempt *to i, then for every $j \geq i$ no coordinator will decide* PRE-ABORT *when setting its* Last_Attempt *to j.*

Proof. The proof is by induction on j. *Base ($j = i$):* This is immediate from Lemma 3. *Step:* We now assume that no coordinator decides PRE-ABORT with *Last_Attempt*$= k$ for every $j > k \geq i$, and prove for j. From the assumption, no site can be in a non-final non-committable state with its $j >$*Last_Attempt*$\geq i$. Now, assume some coordinator r sets its *Last_Attempt* to j in Step 3 of the recovery procedure, we have to show that r did not decide PRE-ABORT during this invocation of the recovery procedure. Assume the contrary, then r collected states, from a quorum of sites with *Last_Attempt*$< j$, and therefore, in this invocation *Max_Attempt*$< j$. Since every two quorums intersect, at least one member of G, p took part in this invocation of the recovery procedure and sent its state to r. Since $j > i$, from Lemma 5, p set its *Last_Attempt* to i (and switched to a committable state) *before* this invocation. But, no site can be in a non-final non-committable state with its $j >$*Last_Attempt*$\geq i$, and therefore *Is_Max_Attempt_Committable* is TRUE in this invocation, which contradicts the assumption that r decides PRE-ABORT. □

Lemma 7 *If the coordinator reaches a* COMMIT *decision when setting its* Last_Attempt *to i, then for every $j \geq i$ no coordinator will decide* PRE-ABORT *when setting its* Last_Attempt *to j.*

Proof. There are two cases to consider:

- If the coordinator reaches a COMMIT decision upon receiving a quorum of ACKs for PRE-COMMIT when setting its *Last_Attempt* to i, then from Lemma 6 for every $j \geq i$ no coordinator will decide PRE-ABORT when setting its *Last_Attempt* to j.

- If the coordinator reaches a COMMIT decision during the recovery procedure upon receiving a COMMIT state, then some coordinator has reached a COMMIT decision before, when its *Last_Attempt* was $< i$. We go back, by induction, to the first coordinator that reached a COMMIT decision. This coordinator must have reached a commit decision according to the previous case. Thus, we can conclude that for every $j \geq i$ no coordinator will decide PRE-ABORT when setting its *Last_Attempt* to j. □

Lemma 8 *If the coordinator reaches an* ABORT *decision upon receiving a quorum of ACKs for* PRE-ABORT *when setting its* Last_Attempt *to i, then for every $j \geq i$ no coordinator will decide* PRE-COMMIT *when setting its* Last_Attempt *to j.*

Proof. This lemma is dual to Lemma 6 and can be proven the same way. □

Lemma 9 *If the coordinator reaches an* ABORT *decision when setting its* Last_Attempt *to i, then for every $j \geq i$ no coordinator will decide* PRE-COMMIT *when setting its* Last_Attempt *to j.*

Proof. There are three cases to consider:

- If the coordinator reaches an ABORT decision during the basic E3PC, this decision is reached because some site voted **No** on the transaction. In this case, the coordinator does not PRE-COMMIT, and no site reaches a committable state in the course of the protocol. Note: If the recovery procedure is invoked with no site in a committable state, then according to the decision rule, a committable decision cannot be reached.

- If the coordinator reaches an ABORT decision during the recovery procedure upon receiving a quorum of ACKs for PRE-ABORT when setting its *Last_Attempt* to i, then from Lemma 8 for every $j \geq i$ no coordinator will decide PRE-COMMIT when setting its *Last_Attempt* to j.

- If the coordinator reaches an ABORT decision during the recovery procedure upon receiving an ABORT state, then some coordinator has reached an ABORT decision before, when its *Last_Attempt* was $< i$. We go back, by induction, to the first coordinator that reached an ABORT decision, according to one of the previous two cases, and conclude that for every $j \geq i$ no coordinator will decide PRE-COMMIT when setting its *Last_Attempt* to j. □

Theorem 1 *If some site running the protocol* COMMITS *the transaction, then no other site* ABORTS *the transaction and vice versa.*

Proof. A site may COMMIT (ABORT) only upon hearing a COMMIT (ABORT) decision from its coordinator. Assume that a COMMIT or ABORT decision was reached for some transaction T. Note: It is possible for more than one coordinator to reach a decision for the same transaction. Let i be the lowest value of *Last_Attempt* that a coordinator had when reaching a COMMIT or ABORT decision. There are two cases to consider:

1. Some coordinator reached an ABORT decision when setting its *Last_Attempt* to i:

   Assume for the sake of contradiction that some coordinator also reached a COMMIT decision, and let j be the lowest value of *Last_Attempt* of a coordinator reaching a COMMIT decision. From the assumption, $j \geq i$. Furthermore, since j is the lowest value of *Last_Attempt* of a coordinator reaching a COMMIT decision, no site could have started this invocation of the recovery procedure in the COMMITTED state, and the COMMIT decision must have been preceded by a PRE-COMMIT. But from Lemma 9 no coordinator can decide PRE-COMMIT when setting its *Last_Attempt* to j, and we reach a contradiction.

2. Some coordinator reached a COMMIT decision when setting its *Last_Attempt* to i:

   The proof is similar to the proof of Case 1 above, but there is one more case to consider: An ABORT decision reached in the course of the basic E3PC (not in the recovery procedure) is *not* preceded by a PRE-ABORT decision. In this case, *Last_Attempt* is set to 1, and the COMMIT decision could not have been reached with a lower value of *Last_Attempt*; therefore $i = 1$. This case reduces to Case 1 proved above. □

References

[BHG87] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, Reading, MA, 1987.

[CHT92] T. D. Chandra, V. Hadzilacos, and S. Toueg. The Weakest Failure Detector for Solving Consensus. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 147–158, 1992.

[CK85] D. Cheung and T. Kameda. Site Optimal Termination Protocols for a Distributed Database under Network Partitioning. In *4th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 111–121, August 1985.

[CR83] F. Chin and K. V. S. Ramarao. Optimal Termination Protocols for Network Partitioning. In *ACM SIGACT-SIGMOD Symposium on Principles of Database Systems (PODS)*, pages 25–35, March 1983.

[CT96] T. D. Chandra and S. Toueg. Unreliable Failure Detectors for Reliable Distributed Systems. *J. Assoc. Comput. Mach. (JACM)*, 43(2):225–267, March 1996.

[DFKM96] D. Dolev, R. Friedman, I. Keidar, and D. Malki. Failure Detectors in Omission Failure Environments. TR 96-13, Institute of Computer Science, The Hebrew University of Jerusalem, Jerusalem, Israel, September 1996. Also Technical Report 96-1608, Department of Computer Science, Cornell University.

[DKYL96] D. Dolev, I. Keidar, and E. Yeger Lotem. Dynamic Voting for Consistent Primary Components. TR 96-7, Institute of Computer Science, The Hebrew University of Jerusalem, Jerusalem, Israel, June 1996.

[DLS88] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the Presence of Partial Synchrony. *J. Assoc. Comput. Mach. (JACM)*, 35(2):288–323, April 1988.

[EASC85] A. El Abbadi, D. Skeen, and F. Christian. An Efficient Fault-Tolerant Algorithm for Replicated Data Management. In *ACM SIGACT-SIGMOD Symposium on Principles of Database Systems (PODS)*, pages 215–229, March 1985.

[EAT89] A. El Abbadi and S. Toueg. Maintaining Availability in Partitioned Replicated Databases. *ACM Trans. Database Systems*, 14(2):264–290, June 1989.

[FLP85] M. Fischer, N. Lynch, and M. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *J. Assoc. Comput. Mach. (JACM)*, 32:374–382, April 1985.

[Gif79] D.K Gifford. Weighted Voting for Replicated Data. In *ACM SIGOPS Symposium on Operating Systems Principles*, December 1979.

[GM82] H. Garcia-Molina. Elections in a Distributed Computing System. *IEEE Trans. Comput.*, C-31, NO.1:48–59, Jan. 1982.

[Gra78] J.N. Gray. Notes on Database Operating Systems. In *Operating Systems: An Advanced Course, Lecture Notes in Computer Science*, volume 60, pages 393–481. Springer-Verlag, Berlin, 1978.

[GS95] R. Guerraoui and A. Schiper. The Decentralized Non-Blocking Atomic Commitment Protocol. In *IEEE International Symposium on Parallel and Distributed Processing (SPDP)*, October 1995.

[Gue95] R. Guerraoui. Revisiting the Relationship between non-blocking Atomic Commitment and Consensus. In *International Workshop on Distributed Algorithms (WDAG)*, pages 87–100, September 1995.

[Her86] M. Herlihy. A Quorum-Consensus Replication Method for Abstract Data Types. *ACM Trans. Comput. Systems*, 4(1):32–53, February 1986.

[Her87] M. Herlihy. Concurrency versus Availability: Atomicity Mechanisms for Replicated Data. *ACM Trans. Comput. Systems*, 5(3):249–274, August 1987.

[KD94] I. Keidar and D. Dolev. Increasing the Resilience of Atomic Commit, at No Additional Cost. Technical Report CS94-18, Institute of Computer Science, The Hebrew University of Jerusalem, Jerusalem, Israel, 1994.

[KD96] I. Keidar and D. Dolev. Efficient Message Ordering in Dynamic Networks. In *15th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 68–76, May 1996.

[Kei94] I. Keidar. A Highly Available Paradigm for Consistent Object Replication. Master's thesis, Institute of Computer Science, The Hebrew University of Jerusalem, Jerusalem, Israel, 1994. Also available as Technical Report CS95-5, and via anonymous ftp at cs.huji.ac.il (132.65.16.10) in users/transis/thesis/keidar-msc.ps.gz.

[Lam89] L. Lamport. The part-time parliament. TR 49, Systems Research Center, DEC, Palo Alto, September 1989.

[MHS89] Tim Mann, Andy Hisgen, and Garret Swart. An Algorithm for Data Replication. Technical Report 46, DEC Systems Research Center, June 1989.

[MLO86] C. Mohan, B. Lindsay, and R. Obermark. Transaction Management in the R* Distributed Database Management System. *ACM Trans. Database Systems*, 11(4), February 1986.

[PW95] D. Peleg and A. Wool. Availability of Quorum Systems. *Inform. Comput.*, 123(2):210–223, 1995.

[Ske82] D. Skeen. A Quorum-Based Commit Protocol. In *6th Berkeley Workshop on Distributed Data Management and Computer Networks*, pages 69–80, Feb. 1982.

[SS83] D. Skeen and M. Stonebraker. A Formal Model of Crash Recovery in a Distributed System. *IEEE Trans. Software Eng.*, SE-9 NO.3, May 1983.

[YLKD97] E. Yeger Lotem, I. Keidar, and D. Dolev. Dynamic Voting for Consistent Primary Components. In *16th ACM Symposium on Principles of Distributed Computing (PODC)*, August 1997.