Increasing the Resilience of Atomic Commit, at No Additional Cost

Idit Keidar
Computer Science Institute
The Hebrew University of Jerusalem
idish@cs.huji.ac.il

Danny Dolev*
Computer Science Institute
The Hebrew University of Jerusalem
dolev@cs.huji.ac.il

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, *e.g.* because of a total crash), 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. 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.

1 Introduction

Reliability and availability of loosely coupled distributed database systems is becoming a requirement 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. In this paper we present a novel *atomic commitment protocol* (*ACP*) that *always* allows a majority (or quorum) to make progress.

In distributed databases when a transaction spans several sites the database servers at all sites have to reach a common decision whether the transaction should be committed or not. A mixed decision results in an inconsistent database, 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* (*ACP*), 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, that 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, *e.g.* because of a total crash), *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 [Kei94] we studied the problem of allowing an asynchronous system to make progress every time a quorum becomes connected, and we have developed a solution for it. In this paper we apply the principles of the solution suggested in [Kei94] to the quorum-based 3PC. The solution imposes a linear order on the quorums formed in the system, and sequentially numbers them using two counters maintained in two phases.

In this paper we present the *Enhanced Three Phase Commit* (*E3PC*) protocol, which is an enhancement of the quorum-based 3PC [Ske82]. E3PC *always* allows a quorum to make progress. It achieves higher availability than 3PC, simply by maintaining two additional

*This work supported by the United States - Israel Binational Science Foundation, Grant No. 92-00189

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.

A common way to increase the availability of data and services is *replication*. If data is replicated in several sites, it can still be available despite site and communication-link failures. Numerous replication schemes that are based on quorums were 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.

E3PC maintains consistency in the face of site failures and network partitions: sites may crash and recover, the network may partition into several *components*[1], and remerge. Whenever a failure is detected, a special recovery procedure is invoked. The protocol does not require that failures be correctly identified in order to work correctly. Undetected failures and false reports of failures may cause degradation in performance. In the absence of false failure reports, a *quorum* of connected sites may always reach a decision: 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 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].

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 (3PC) protocol [Ske82] is described in Section 4, and enhanced three phase commit (E3PC), in Section 5. Section 6 concludes the paper.

2 The Model

We assume that the set of sites running the protocol is fixed, and is known to all the sites. We assume that the sites are connected by an underlying communication network, that provides reliable FIFO 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 re-merge. Sites may crash and recover; recovered sites come up with their stable storage intact. We assume that failures are detected using a (possibly unreliable) fault detector, *e.g.* a timeout mechanism. We do not deal with message losses, we assume that the underlying communication layer discovers the loss and either recovers it, or reports of a failure.

2.1 Quorums

We use a *quorum system* to decide when a group of connected sites may resolve the transaction. To enable maximum flexibility we allow the quorum system to be elected in a variety of ways (*e.g.* weighted voting). For further flexibility, it is possible to set different quorums for commit and abort. In this case, a *Commit Quorum* of connected sites is required in order to commit a transaction, and an *Abort Quorum* – to abort.

We assume 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 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 sub-transactions, each running on a different site. The database manager at each site can unilaterally decide to ABORT the local sub-transaction, in which case the entire transaction must be aborted. If all the participating sites agree to COMMIT their sub-transaction (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 sub-transaction, once it has agreed to COMMIT it.

In order to ensure that all the sub-transactions are consistently committed or aborted, the sites run an *atomic commitment protocol* (*ACP*) such as *two phase commit* (*2PC*). The requirements of atomic commitment (as defined in Chapter 7 of [BHG87]) are:

- All the sites that reach a decision reach the same one.

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

- A site cannot reverse its decision after it has reached one.
- The COMMIT decision can only be reached if all sites voted **Yes**.
- If there are no failures and all sites voted **Yes**, then the decision will be to COMMIT.
- 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* (*2PC*) [Gra78]. Several variations of 2PC were suggested, 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, all the sites remain blocked indefinitely, unable to resolve the last transaction. The centralized version of 2PC is depicted in Figure 1.

Coordinator	Slave
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 2. 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 slave) 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.

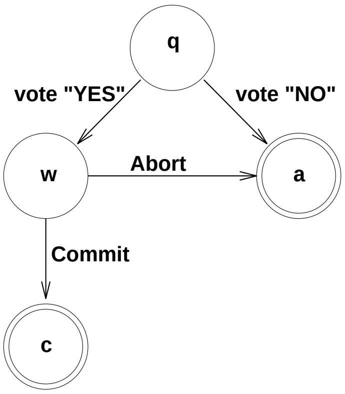

Figure 2: State Diagram for Two Phase Commit

w : WAIT state – In this state the coordinator waits for votes from all of the slaves, and each slave waits for the final word from the coordinator. This is the "uncertainty period" for each site, when it doesn't 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 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. proved [SS83] 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, that maintains consistency in spite of network partitions [Ske82]. This protocol is blocking; 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* (*E3PC*), that 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* (*3PC*) 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.

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:

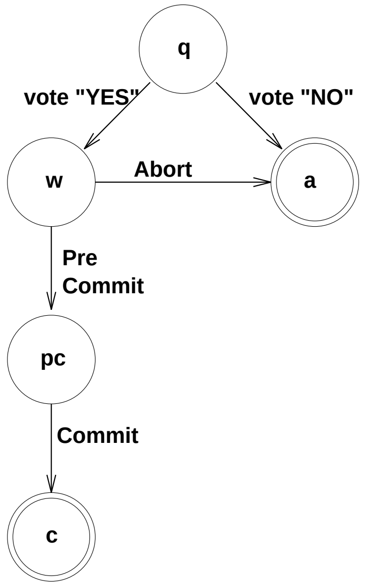

Figure 3: The Basic Three Phase Commit

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 4, and a corresponding state diagram is depicted in Figure 3. 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.

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].

[2]Election is a *weaker* problem than atomic commitment, only

Coordinator	**Slave**
Transaction is received: Send sub-transactions to slaves.	
	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 *Commit Quorum* of **ACKs**: Send COMMIT. Otherwise: Block (wait for more votes or until recovery)	
	COMMIT or ABORT is received: Process the transaction accordingly.

Figure 4: The Quorum-Based Three Phase Commit Protocol

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 7. 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 slave sends an ACK to r.
5. Upon receiving ACKs for PRE-COMMIT from a *Commit Quorum* or ACKs for PRE-ABORT from an *Abort Quorum*, r multicasts the corresponding decision: COMMIT or ABORT.
6. Upon receiving a COMMIT (ABORT) message: Process the transaction accordingly.

Figure 5: The Quorum-Based Recovery Procedure for Three Phase Commit

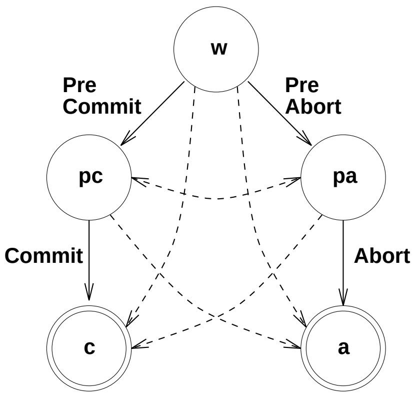

Figure 6: The Recovery Procedure for 3PC

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 *Commit Quorum* for COMMIT, or an *Abort Quorum* for ABORT. 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 5. The state diagram for the recovery procedure is shown in Figure 6. 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 doesn't include p_1) is formed, and another site, p_2 decides to COMMIT the transaction (this does not violate consistency, since 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.

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.

Therefore, p_1 makes a transition from PRE-ABORT to COMMIT.

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

Figure 7: The Decision Rule for The Quorum-Based Recovery Procedure

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 *Commit Quorum*. An ABORT is possible if the group of sites in the WAIT state together with the sites in the PRE-ABORT state form an *Abort Quorum*. The decision rule is summarized in Figure 7.

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, and the same quorums are designated for both commit and abort. Consider the following scenario:

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 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 if 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 that.

5 The E3PC Protocol

We suggest a three phase atomic commitment protocol, *Enhanced Three Phase Commit* (*E3PC*), 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 commit protocol [Ske82]. E3PC does not require more communication or time than 3PC; the improved resilience is achieved simply by maintaining two additional counters.

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.

5.1 E3PC: Enhancing Three Phase Commit

The basic E3PC is similar to the basic 3PC, the only difference is 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: there is no need to elect a new coordinator in each invocation of the basic 3PC, or E3PC, the re-election is only needed 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

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 value of *Last_Elected* and determines *Max_Elected*. 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 and *Last_Attempt* 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 9. 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 slave sets its *Last_Attempt* to *Last_Elected* and sends an ACK to r.
5. Upon receiving ACKs for PRE-COMMIT (PRE-ABORT) from a *Commit Quorum* (*Abort Quorum*), r multicasts the decision: COMMIT (ABORT).
6. Upon receiving a COMMIT (ABORT) message from r: process the transaction accordingly.

Figure 8: The Recovery Procedure for E3PC

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 that two contradicting attempts (*i.e.* PRE-COMMIT and PRE-ABORT), will not be made with the same value of *Last_Attempt*.

We use the following notation:

- $\mathcal{P}$ is the group of sites that are live and connected, and take part in the election of the new coordinator.
- *Max_Elected* is $\max_{p\in\mathcal{P}}(\textit{Last\_Elected}$ of $p)$.
- *Max_Attempt* is $\max_{p\in\mathcal{P}}(\textit{Last\_Attempt}$ of $p)$.
- *Is_Max_Attempt_Committable* is a predicate that is TRUE iff all the members that are in non-final states and their *Last_Attempt* is equal to *Max_Attempt* are in a committable state. Formally, *Is_Max_Attempt_Committable* is TRUE iff:

  $\forall_{p\in\mathcal{P}}(\textit{Last\_Attempt}$ of $p = \textit{Max\_Attempt}$ and p is in a non-final state $\rightarrow p$ is in a committable state)

5.2 Quorum Based Recovery Procedure

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 re-invoked to let them take part. 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. 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 8. The possible state transitions in E3PC and its recovery procedure are the same as those of 3PC, depicted in Figures 3 and 6; 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. It is possible to reach a decision before collecting the states from all the sites in $\mathcal{P}$. The sites are blocked until r receives enough states to allow a decision. 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 $\wedge Q_C(\mathcal{S})$	PRE-COMMIT
$\neg$*Is_Max_Attempt_Committable* $\wedge Q_A(\mathcal{S})$	PRE-ABORT
Otherwise	BLOCK

Figure 9: The Decision Rule for E3PC

The coordinator collects the states and the values of *Last_Attempt* 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 *Commit Quorum* – PRE-COMMIT.
- If *Is_Max_Attempt_Committable* is FALSE and $\mathcal{S}$ is an *Abort Quorum* – PRE-ABORT.
- Otherwise – BLOCK.

The decision rule is summarized in Figure 9. It is easy to see that with the new decision rule, if a group of sites is both a *Commit Quorum* and an *Abort Quorum*, it will never be blocked.

5.3 E3PC does not Block a Quorum

In E3PC, if a group of sites forms both a *Commit Quorum* and an *Abort 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, and the same quorums are designated for both commit and abort. We considered the following scenario:

- 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.
- 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 [KD94] we formally prove that E3PC fulfills the requirements of atomic commitment described in Section 3.1. In this section we describe the proof's outline.

We first prove that two contradicting attempts (*i.e.* PRE-COMMIT and PRE-ABORT), cannot be made with the same value of *Last_Attempt*. This is true due to the fact that every *Commit Quorum* intersects every *Abort Quorum*, and that a *Commit Quorum* of sites must increase *Last_Elected* before a PRE-COMMIT decision, and an *Abort Quorum*, before PRE-ABORT. 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.

Using the statements above we prove that if the coordinator reaches a COMMIT (ABORT) decision upon receiving a *Commit Quorum* (*Abort 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 this 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.

6 Conclusions

In this paper we demonstrated how the three phase commit (3PC) [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*[3] 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 posses 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 the counters is piggybacked on messages which are sent in 3PC as well as in E3PC: the value of *Last_Elected* is attached to messages used to elect a new coordinator, and *Last_Attempt* is sent to the new coordinator along with the site's state.

The importance of this paper is in demonstrating, using a very simple protocol, a general technique for making distributed systems more resilient, specifically, always allowing a connected quorum in the system to make progress. The technique uses two counters to impose a *linear order* on the quorums formed in the system, and guarantees that the decisions made by each quorum will be consistent with the decisions of the previous ones.

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. The same *Quorum System* should be used to determine when the data is accessible to a group of sites as for the atomic commitment protocol. Thus, in order to complete a transaction, 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 is *partially replicated*, then for each item accessed by this transaction, a quorum of the sites it resides on is required.

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 *Virtual Partitions* protocol [EASC85, EAT89], this is done every time the view changes. In this case, we suggest to use the view change as a "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.

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 replication [Kei94]. The same principle may be combined with a *dynamic voting* scheme for replication (cf. Section 7 in [Kei94]).

[3] If different quorums are used for commit and abort, then we say the group contains a *quorum* if it contains both a *Commit Quorum* and an *Abort Quorum*.

References

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

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

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

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

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

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

[GM82] H. Garcia-Molina. Elections in a Distributed Computing System. *IEEE Transactions on Computers*, 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.

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

[Her87] M. Herlihy. Concurrency versus Availability: Atomicity Mechanisms for Replicated Data.

ACM Trans. Comp. Syst., 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.

[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.

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

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