Fault Tolerant Commit Protocols

Shyan-Ming Yuan and Pankaj Jalote

Department of Computer Science
University of Maryland College park
College Park, Maryland 20742

Abstract
To maintain consistency on a distributed database system, a commit protocol is often employed to ensure that either all the sub-transactions of a transaction commit or all abort. Special care needs to be taken to ensure proper functioning of a commit protocol in the face of node failures during the execution of the protocol. A protocol is said to be fault-tolerant if it properly commits or aborts, even if nodes fail.

In this paper, we present two different schemes to make commit protocols fault tolerant. The first scheme ensures that a failed site can directly recover to a state, which is consistent with all other sites, only using its local information. This scheme can only make commit protocols resilient to a single site failure. The second scheme ensures that failed sites can recover to a state consistent with all others with as few information exchanges as possible. This scheme can make commit protocols resilient to any type of site failure.

1 Introduction
In a distributed database system, maintenance of database consistency usually requires a transaction to be atomic. This requires that all the actions of a transaction successfully complete or none of them do. Commit protocols are needed to maintain the logical atomicity of the transactions.

Several commit protocols have been proposed [1,2,3,4,5]. The simplest one is called two-phase commit protocol. In two-phase commit protocol, there is one site called coordinator which starts the protocol and others called slaves. In the first phase, the coordinator sends a start message to all slaves upon receiving transaction request from users and then waiting for responses from slaves. If all the slaves and the coordinator agree to commit the transaction then the coordinator sends a commit message to all the slaves, otherwise it sends an abort message to all the slaves. This protocol guarantees transaction atomicity in the absence of failures. However, if a site fails, during the execution of the protocol, the operational sites may have to block the execution of the transaction until the failed site recovers.

A protocol is said to be nonblocking if the operational sites do not block when site failures occur during the execution of the protocol [6]. Skeen [6] has proposed a method to make the two phase commit protocol become nonblocking by adding another phase, called prepare phase. The resulting protocol is called three phase commit protocol.

Although nonblocking protocols do not block operational sites, a failed site may require information on recovery from failures to commit or abort. A protocol is independently recoverable if, on recovery, it is possible for a failed site to decide to commit or abort based on its local information only.

Skeen [7] has proposed an independent recovery scheme which can deal with any single site failure when there are only two sites in the system. However, as the number of sites in a system increase, the problem becomes complicated. Because a site can fail in the middle of a state transition, it may fail after sending some messages but before sending all messages. Therefore, some of the sites expecting the message from the failed site will receive it, while others will timeout. Sites receiving messages will have different view about the failed site from the sites having a timeout. In this paper, we propose an independent recovery scheme which can deal with any single site failure for systems with more than two sites.

It has been shown [7] that there exists no independent recovery scheme for multiple site failures. However, recovery schemes that are not independent can be designed to handle the case of multiple site failures. Several authors [2,8,9,10,11] have studied this problem. The usual way is to employ a termination protocol when failures are detected. A
termination protocol is a voting scheme which elects a new coordinator in central site protocols or takes a consensus in decentralized protocols. If new failures are detected in the current voting procedure, another voting procedure has to be involved. Such schemes can require many message interchanges if site failures continue to occur.

In this paper, a general recovery scheme is presented that is capable of handling multiple failures. Since independent recovery is not always possible, the scheme involves communication, if necessary, after the failure site recovers. However, communication is not required in all cases. In the cases where independent recovery can be performed by a site, the proposed scheme performs independent recovery.

The remainder of this paper is organized in four sections. Section 2 describes our system model. Section 3 presents an independent recovery scheme for single site failure case. Section 4 presents two recovery schemes for multiple site failure case. Section 5 gives a brief conclusion.

2 System Model

We assume that the network is fully connected and delivers every message within a preassigned time period. We also assume that nodes fail by stopping[12] and if a sender fails, we assume that the receivers will detect it by using a time-out mechanism.

The model used to formalize commit protocols in this paper is based on the model presented by Skeen in [7]. In this model, the execution of a commit protocol at each site is modeled as a finite state automaton (FSA). Each local state transition of FSA’s consists of receiving and/or sending of zero or more messages and is assumed to be atomic. Each FSA has two final states: a commit state and an abort state. If the commit protocol is correct, all the FSA’s should either reach their commit states or all should reach their abort states. As an example, the FSA’s of the two-phase commit protocol in a two site system is shown in figure 1.

On the FSA model of a commit protocol, the concurrency set of a local state $s_i$, represented as $C(s_i)$, is defined to be the set of local states which may be concurrently occupied by other sites when site $i$ is in state $s_i$. The sender set of a local state $s_i$, represented as $S(s_i)$, is defined to be the set of local states whose messages are received by $s_i$. For example, in the FSA for two-phase commit protocol in figure 1, $C(w_1)$ is $\{q_2, w_2, a_2\}$ and $S(w_1)$ is $\{q_1\}$.

It has been shown [7] that if a protocol contains a local state with both commit and abort states in its concurrency set, then it is impossible to make the protocol resilient to single site failure under independent recovery. If no local states have concurrency set containing both commit and abort states, then the following rules can make the protocol independently recoverable from any single site failures for a two site system.

1. For every intermediate state $s$ in the protocol if $C(s)$ contains a commit state, then assign a failure transition from $s$ to a commit state, otherwise assign a failure transition from $s$ to an abort state.

2. For every intermediate state $s$ if state $t$ is in $S(s)$ and $t$ has a failure transition to a commit (or abort) state, then assign a timeout transition from $s$ to a commit (or abort) state.

A failure transition from state $s_i$ to state $r_i$ means that if site $i$ fails in state $s_i$ then it will be in state $r_i$ upon recovering. A timeout transition from state $s_i$ to state $r_i$ means that if site $i$ get a timeout message due to some expected message missing at state $s_i$ then it will move to state $r_i$ instead of a normal transition.

The two-phase commit protocol in figure 1 is not independently recoverable because $C(w_2)$ contains both $w_2$ and $c_1$. It can be made independently recoverable by adding an additional "wait" state in the coordinator between state $w_1$ and $c_1$. If a correspondent state is added in the slave, then the resulting protocol, as shown in figure 2, is called a three-phase commit protocol.

In Skeen's model, local transitions were considered to be atomic [7]. This assumption holds for a two-site system where the coordinator or the slave has to send at most one message during each state transition. However, in a system
where there are multiple slaves, a local state transition of a commit protocol involves receiving several messages and sending several messages. In such a system a node may fail after sending or receiving some of the messages. This means that a site failure may occur in the middle of a local state transition and the transition is not atomic. For instance, the three-phase commit protocol involves receiving several messages and after sending or receiving some of the messages. This means that a site failure may occur in the middle of a local state transition and the transition is not atomic. For instance, the three-phase commit protocol shown in figure 2, contains no local states having both commit and abort states in the concurrency sets. If slave i fails in state wi, the coordinator will timeout in state pi; and move to state s1, but some slaves may timeout in their pi states and move to their commit states. Therefore, the final global state will contain both commit and abort states.

3 Single Site Failure in a Multi-Site System

In this section we present an extension to the Skeen's scheme to handle any single site failure in a multi-site system. A site may fail after sending or receiving some of the messages needed for a successful transition in the FSA. We assume that if a failure occurs before the transition from a state s to another state r, the FSA stays in state s. We say that the failed site fails at state s.

If a protocol has no local state with both commit and abort states in its concurrency set, like the three-phase commit protocol shown in figure 2, then it can independently recover from an arbitrary single site failure. For this, besides the failure transitions, we have to add some special transitions to the FSA's. A simple timeout transition will not be sufficient, because when a site s has a timeout while waiting for a message from site s', it has no way of knowing whether s' has failed, or that s' is unable to send the message since some other site s" has failed. To handle this situation, we introduce special timeout transitions which besides making a state transition, may also send messages.

A transition labeled by (timeout/toa) from state si to an abort state means that if site i detects timeout in state si then it sends toa to all other sites and moves to an abort state. A transition labeled by (timeout/toa) from state si to a commit state means that if site i detects timeout in state si then it sends toa to all other sites and moves to a commit state. A transition labeled by (toa/-) from state si to an abort state means that if site i receives toa in state si then it moves to an abort state. A transition labeled by (toc/-) from state si to a commit state means that if site i receives toc in state si then it moves to a commit state.

To make protocols independently recoverable from any single site failure, we present the following rules. We assume that no FSA has both commit and abort states in any concurrency set.

**RULE 3-1:** If a nonfinal local state s has a commit state in its concurrency set, then assign a failure transition from s to a commit state. Otherwise, assign a failure transition from s to an abort state (same as Skeen's rule 1).

**RULE 3-2:** For any intermediate local state si, if there exists a state s at other site in the sender set of si, such that s has a failure transition to a commit state then assign a transition labeled by (timeout/toa) from si to a commit state. If s has a failure transition to an abort state then assign a transition labeled by (timeout/toa) from si to an abort state.

Since the failure site may have sent out some outgoing messages before it fails, some sites may receive those messages and assume that the failure site made its normal transition. To ensure that every one has the same view of the failed site, those sites who detect a timeout should propagate to others such that every operational site will make same response for the failure.

**RULE 3-3:** For any intermediate local state si, assign a transition labeled by (toc/-) from si to a commit state and a transition labeled by (toc/-) from state si to an abort state.

Since sites detecting a timeout will propagate to others, all intermediate state should have transitions to handle the propagated timeout messages. We assume that timers are set such that no timeout occurs before a potential toa or toc messages arrives. And if a site receives toc (or toa) in
commit (or abort) states then it simply ignores them. The resulting protocol of applying above rules to three phase commit protocol is shown in figure 3.

**Theorem 3.1** If a commit protocol has no local state with both commit and abort states in its concurrency set, then rule 3-1, 3-2, and 3-3 are sufficient to make the protocol independently recover from any single site failure.

**Proof:** Assume that there are N sites, numbered as 1, 2, ... , N. Let site 1 be the failure site and s1, s2, ..., sN be the local states of site 1, site 2, ... , site N, respectively, when the failure occurs. There are two cases to be considered.

1. State s1 has a failure transition to a commit state. Since site 1 fails before it sends out all outgoing messages, there exists at least one site which will detect a timeout, send toc, and move to a commit state (rule 3-2). The other operational sites either are in the commit state already (definition of the concurrency set) or will receive toc and move to a commit state (rule 3-3). Therefore, all sites have a common final state commit.

2. State s1 has a failure transition to an abort state. The similar arguments can be used to show that all sites have a common final state abort.

If no failure occurs, the number of messages required for the protocol remains unchanged. In case of a failure, the total number of propagated timeout messages (toc or toe) is dependent on how many outgoing messages were sent before failure. If less sites detect a timeout then less propagated messages will be sent. The maximum is \((N - 2)^2\) and the minimum is \((N - 2)\). This overhead can be reduced by sending messages in some predefined order, or having one special site broadcast all the messages.

### 4 Multiple Site Failures

It is known that no independent recovery scheme can handle more than one site failure [7]. With multiple failures, different failed sites may move to different final states upon recovering. The conflict occurs only when the union of concurrency sets of failed sites contains both commit and abort states. In this section, we present a recovery scheme, which can handle multiple site failures including total failures, by utilizing extra recovery states. On recovery, a failed site moves to a recovery state, from where it communicates with others to determine whether to commit or abort the previous transaction. In some situations, where a site fails in some special state then it can independently recover to a final state. The scheme we present identifies such situations.

---

**Figure 3:** Three Phase Independent Recovery Protocol
and perform independent recovery in those cases. We first present the scheme without any attempt for independent recovery and then show how, in some cases, independent recovery can be performed.

4.1 A Simple Recovery Scheme

Suppose two nodes fail during the execution of a commit protocol with the FSA's in states $x_1$ and $y_1$, respectively. If $C(x_1)$ contains a commit state and $C(y_1)$ contains an abort state, then under independent recovery, the system may reach an inconsistent state after recovering. For the sake of resolving the conflict, we introduce two extra states: recovery state $T$ and recovery waiting state $TW$. Upon recovering, a failed site moves to the recovery state $T$, asks information from others, and waits for response in state $TW$. Upon receiving such requests, a site should respond commit if it has committed the transaction, abort if it has aborted it, or unknown if it has no knowledge about it. The following rules can be used to make a commit protocol, which does not violate the constraint of theorem 3-1, resilient to any number of site failures.

**Rule 4-1:** For all nonfinal states, assign a failure transition to the recovery state $T$.

**Rule 4-2:** For any intermediate local state $s$, if there exists a state $t$, at some other node, in the sender set of $s$ such that $C(t)$ contains a commit state, then assign a transition labeled by (timeout/toe) from $s$ to a commit state. Otherwise assign a transition labeled by (timeout/toa) from $s$ to an abort state.

**Rule 4-3:** For any intermediate local state $s$, assign a transition labeled by (toc/−) from $s$ to a commit state and a transition labeled by (toa/−) from $s$ to an abort state.

**Rule 4-4:** Assign special transitions to states $T$, $TW$, $e$, and $a$ as shown in figure 4.

**Theorem 4-1** If a commit protocol has no local state with both commit and abort state in its concurrency set, then rule 4-1, 4-2, 4-3 and 4-4 are sufficient to make the protocol resilient to any number of site failures.

**Proof:** Since no state has both commit and abort states in its concurrency set, all operational sites will reach the same final state (by theorem 3-1). A failed site on recovery starts from state $r$, then sends recovery message to all others and moves to state $rw$. In state $rw$, upon receiving a commit or an abort message, it moves to state $c$ or $a$, respectively. Because all operational sites reach the same final state, one commit or abort message is sufficient for failed sites to decide where to go. If a failed site receives the unknown messages from all others, it means that all sites failed during the transaction, and it can move to state $a$. Others will receive the abort message from this site and move to state $a$ eventually.

The total number of special messages required for operational sites and failed sites is at most $2kn + (n - k)n$ if the system has $n$ sites and $k$ sites fail.

4.2 An Efficient Recovery Scheme

If a site $i$ fails in state $s_i$, then all possible states which can be concurrently occupied by other sites must belong to the concurrency set of $s_i$. Hence, any other node that fails with node $i$ during the execution of a commit protocol must be in a state that belongs to $C(s_i)$. In other words, if nodes $i$ and $j$ fail concurrently in states $s_i$ and $t_j$, then $s_i \in C(t_j)$ and $t_j \in C(s_i)$. Using this property, we can determine whether independent recovery is possible or not.

The diagram below illustrates the special transitions for simple recovery scheme.
LEMMA 4-1 Suppose a site $i$ fails in state $s_i$. Site $i$ can perform independent recovery if and only if for each $t_j \in C(s_i)$, $C(t_j)$ contains the same final state as $C(s_i)$. In other words, if $C(s_i)$ contains a commit (abort) state then all $C(t_j)$ must contain a commit (abort) state also.

PROOF: Since all states which can fail with $s_i$ are in $C(s_i)$, if their concurrency sets have the same final state as the $C(s_i)$, then $s_i$ can perform independent recovery to the final state contained by $C(s_i)$. By the definition of concurrency set, some operational sites may already be in the final state, other sites should eventually move to that final state also.

If there exists a state $t_j \in C(s_i)$, such that $C(t_j)$ contains a final state different from the one in $C(s_i)$, then performing independent recovery at state $s_i$ may lead to inconsistency. $\square$

DEFINITION 4-1: A state $s$ is called a failure commit (abort) state if all states belonging to $C(s) \cup \{s\}$ have a commit (abort) state in their concurrency sets.

Since for any local state $s$, $C(s)$ can be statically determined, we can compute the concurrency sets for every state to identify the failure commit states and the failure abort states. For instance, in the three-phase commit protocol, the states $q_i, i = 1, 2, \ldots, n$ and the state $w_i$ are failure abort states. According to the previous lemma and definition, we can obtain an efficient recovery scheme by replacing the rule 4-1 by the following rule.

RULE 4-5: For all failure abort states, assign a failure transition to an abort state. For all failure commit states, assign a failure transition to a commit state. For all other nonfinal states, assign a failure transition to the recovery state $r$.

THEOREM 4-2 If a commit protocol has no local state with both commit and abort state in its concurrency set, then rule 4-5, 4-3, 4-3 and 4-4 are sufficient to make the protocol resilient to any number of site failures.

PROOF: Follows from theorem 4-1 and lemma 4-1. $\square$

This scheme can save $2n - 2$ messages for each failed site which fails in a failure commit or failure abort state. Figure 5 is the resulting protocol by applying the recovery scheme to the three-phase commit protocol.

![Figure 5: Three-Phase Efficient Recovery Protocol](image)
5 Concluding Remarks

Commit protocols are needed to ensure the atomicity of a distributed transaction. However, if sites fail during the execution of the protocol, some recovery schemes have to be applied to ensure that the atomicity of a transaction can be guaranteed and no operational sites block their executions to wait for the recovery of failed sites.

In this paper, we presented two kinds of recovery schemes to make a commit protocol resilient to site failures. The first scheme is an independent recovery scheme which is an extension of Skeen's method [7]. The advantages of our scheme are that it can make commit protocol resilient to any single site failure for systems with more than two sites, where local state transition are not atomic.

We also proposed two general recovery schemes which can make commit protocols resilient to any number of site failures. The simple recovery scheme can make a commit protocol resilient to an arbitrary number of site failures and total failures with at most \(N^2 + kN\) extra messages if there are \(N\) sites and \(k\) failed sites. The efficient recovery scheme can reduce the message overhead by identifying these situations where independent recovery is possible. The major advantage of our recovery schemes is that no backup coordinators have to be elected and no domino effect can occur when new failures occur in the execution of termination protocols. Currently we are working on ways to reduce the message overhead in our protocols.

References


