# A FORMAL MODEL OF CRASH RECOVERY

# IN A DISTRIBUTED SYSTEM

by
Dale Skeen
Michael Stonebraker

Memorandum No. UCB/ERL M80/48
December 5, 1980

Electronics Research Laboratory
College of Engineering
University of California, Berkeley
94720

# A FORMAL MODEL OF CRASH RECOVERY IN A DISTRIBUTED SYSTEM

*by*

*Dale Skeen and Michael Stonebraker*

Department of Electrical Engineering and Computer Science
University of California
Berkeley, California

## ABSTRACT

In this paper we introduce a formal model for transaction processing in a distributed data base system. We use this model to study both failures of single sites and communications failures. For site failures, we introduce a pessimistic crash recovery technique called *independent recovery*, and identify the class of failures for which a resilient protocol exists. For network partitions, we study the question of finding resilient protocols for the pessimistic case when messages are lost, and also for the optimistic case when no messages are lost.

## 1. INTRODUCTION

In this paper we present a formal model for transaction processing in a distributed data base and then extend it to model several classes of failures and crash recovery techniques. These models are used to study whether or not resilient protocols exist for various failure classes.

Crash recovery in distributed systems has been studied extensively in the literature [ALSB79, GRAY79, HAMM79, LAMP78, MENA79, ROTH77, STON79, SVOB79]. Many protocols have been designed which are resilient in some environments. All have an "ad-hoc" flavor to them in the sense that the class of failures they will survive is not clearly delineated.

The purpose of this paper is to formalize the crash recovery problem in a distributed data base environment and then give some preliminary results concerning the existence of resilient protocols in various well defined situations.

Consequently, in the next section we give a brief introduction to transactions in a distributed data base. Then, in section 3 we indicate the assumed network environment and our model for transaction processing. In section 4 we extend the model to include the possibility of site failure and give results concerning the existence of resilient protocols in this situation. Section 5 turns to the possibility of network failure and shows the class of failures for which a resilient protocol exists. The paper concludes with a summary and description of future work.

All results are presented without proof. The reader is referred to [SKEE81] for a more thorough treatment of the model and detailed proofs of all results.

---

This research was sponsored by the U.S. Air Force Office of Scientific Research Grant 78-3596, the U.S. Army Research Office Grant DAAG29-76-G-0245, and the Naval Electronics Systems Command Contract N00039-78-G-0013.

## 2. BACKGROUND

A distributed data base management system supports a data base distributed over multiple sites interconnected by a communications network. A *transaction* in a distributed database is an atomic operation, in the sense that it is indivisible: either it executes to completion or it appears not to have executed at all. The goal of distributed crash recovery is to provide transaction atomicity in the presence of failures for commands which may span several sites.

A transaction may not execute to completion because:

(1) one or more sites fails

(2) the network fails

(3) the transaction deadlocks with another transaction

(4) the user aborts the transaction.

During the processing of a transaction each participating site must be able to abort the transaction for any of the above reasons. When a transaction is aborted at a site, the state of the local data base is restored to its original state by local recovery procedures. The one site recovery problem is fairly well understood [GRAY79, LORI77].

At some point during transaction processing a site reaches a "commit point". Once a site has committed, it will complete the transaction even in the presence of a site failure.

For transaction atomicity to be preserved in a distributed environment, either all sites must abort or all must commit the transaction. A state where some sites have committed while others have aborted is an *inconsistent* state.

It is always an option for a distributed data base system to suspend operation whenever a failure occurs and only resume processing when the failure is repaired. Clearly, such a decision will render the distributed system exactly as resilient as the weakest link. In this paper we will be interested only in *nonblocking protocols* for which an operational site never suspends because of a failure.

Protocols designed to enforce atomicity are traditionally called *commit protocols*. A commit protocol is said to be *resilient* to a class of failures, if the protocol enforces transaction atomicity and is nonblocking for any failure within the appropriate class. The nonblocking constraint guarantees that a resilient protocol will always terminate. Similarly, a resilient protocol with an a priori upper bound on the number of messages always satisfies the nonblocking constraint. We will be interested exclusively in protocols with predefined upper bounds.

## 3. THE TRANSACTION MODEL

### 3.1. The Network Model

The network is assumed to provide point-to-point communication between any pair of sites. Moreover, it is assumed to have the following characteristics:

(1) it delivers a message within a preassigned time period, T, or

(2) it reports a "time out" to the sender.

When a time-out occurs, the sender can safely assume that the network or the recipient or both has failed. In the case of a network failure, it is not known whether the recipient received the message.

### 3.2. Transaction Processing

Transaction execution at a single site is modelled as a finite state automaton (FSA). During a transition a site can read one or more messages from the network, do

local processing and write one message to the network. A distributed transaction is then a collection of FSA's, one per participating site, and the network serves as a common input/output tape to all sites. Figure 1 presents a four site example.

There are several restrictions on this collection of FSA's:

(1) The FSA's are nondeterministic. The behavior of each FSA is not known apriori because of the possibility of deadlocks, failures, and user aborts. Moreover, when multiple messages are addressed to a site, the order of receiving the messages is arbitrary.

(2) The final states of the FSA's are partitioned into two sets: the "abort" states, A, and the "commit" states, C.

(3) There are no transitions from a state in A to a state not in A. Similarly, there are no transitions from a state in C to a state not in C. Therefore, once a site enters an "abort" state ("commit" state), the site remains in such a state. This corresponds to the requirement that abort and commit are irreversible operations.

(4) The state diagram describing a FSA is acyclic. This suffices to guarantee that a protocol is nonblocking.

FSA transitions are assumed to be instantaneous, and no two FSAs change state simultaneously. Therefore, the transitions made by a group of sites can always be linearly ordered.

### 3.3. An Annotated Example

We illustrate this FSA model by examining a two phase commit protocol (similar to [GRAY79, LAMP76]) for a two site transaction. This protocol is given in Figure 2. In the first phase each site receives the transaction, partially executes it, and indicates its readiness to commit ("ready"). The commit decision is made by the co-ordinator (site 1) which receives ready votes and sends a "commit" message only if all sites vote "ready". For a transaction to commit, three messages are exchanged: "start transaction" is sent to site 2; "ready" is sent to site 1; and "commit" is sent to site 2.

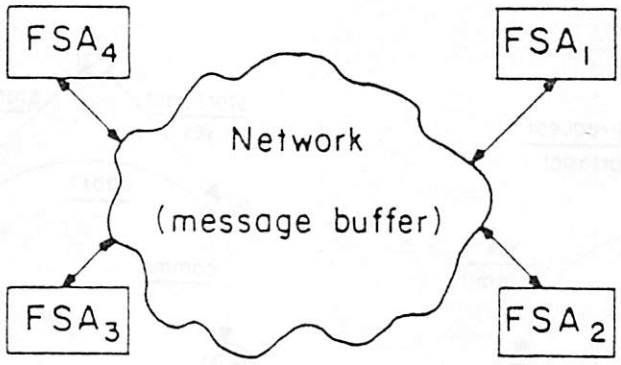


Figure 1. The model with 4 sites.

| SITE 1 | SITE 2 |
|---|---|
| (1) Transaction is received.<br>"Start Xact" is sent. | |
| | "Start Xact" is received.<br>Site 2 votes: "yes" to commit,<br>"no" to abort.<br>The vote is sent to site 1. |
| (2) The vote is received.<br>If vote="yes" and site 1 agrees,<br>then "commit" is sent;<br>else, "abort" is sent. | |
| | Either "commit" or "abort" is<br>received and processed. |

**Figure 2.** The two-phase commit protocol (2 sites).

The FSA state diagrams for this protocol are given in Figure 3. The initial states are $q_1$ and $q_2$. The execution of the protocol is initiated by the receipt of the special message, "Xact request," at site 1. Each FSA then proceeds to make transitions asynchronously. For each arc, the message received is indicated physically above the message sent. Final states are double circled and labelled Commit ($c_i$) or Abort ($a_i$). All

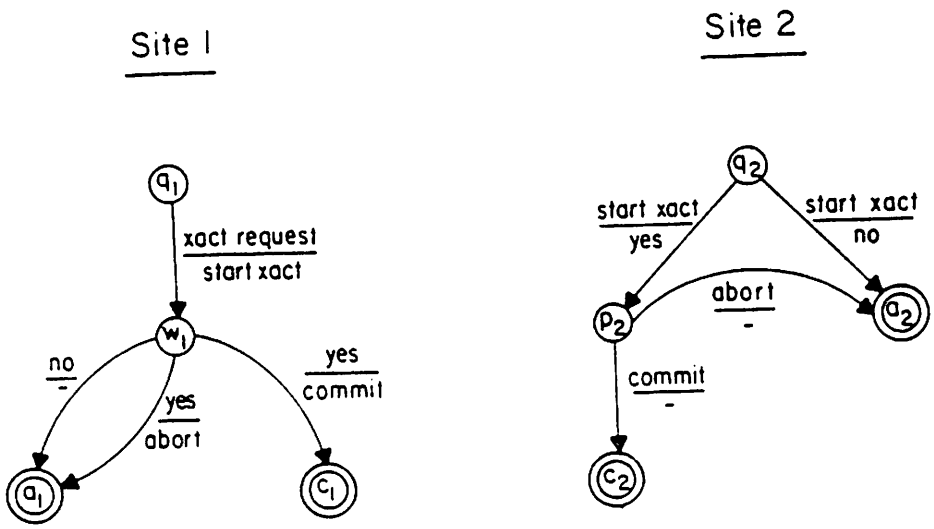


**Figure 3.** FSA's for the two-phase commit protocol (2 sites).

states are subscripted with their site number. This notation will be followed throughout the paper.

### 3.4. Global Transaction State

The *global state* of a distributed transaction is defined to consist of:

(1) a global state vector containing the local states of the participating FSA's and

(2) the outstanding messages in the network.

The global state defines the complete processing state of a transaction.

A *global state transition* occurs whenever a local state transition occurs at a participating site. Therefore, in a global state transition exactly one local state in the global state vector makes a transition while the others remain unchanged.[1]

If there exists a global state transition from global state **g** to global state **g'**, then **g'** is said to be *immediately reachable* from **g**. A global state, together with the definition of the protocol, contains the minimal information necessary to compute all of its immediately reachable states. The transitive closure of the immediately reachable relation yields all reachable states. Figure 4 contains the reachable state graph for the 2-phase protocol discussed earlier.

A *terminal* state is one with no reachable successors. Moreover, a path from the initial global state to a terminal global state in the reachable state graph corresponds to a possible execution sequence of the protocol.

A global state is said to be a *final state* if all local states contained in the state vector are final states. A global state is said to be *inconsistent* if its state vector contains both a commit state and an abort state. A protocol is *functionally correct* if and only if its reachable state graph contains no inconsistent states and all terminal states are final states. Figure 4 verifies that, in the absence of failures, the 2-phase protocol is correct.

Two local states are said to be *potentially concurrent* if there exists a reachable global state vector that contains both local states. We define the *concurrency set* of a local state $s_i$ to be all of the states of other FSA's that are potentially concurrent with it. We denote this set by $C(s_i)$. From this definition it should be clear that if state $s_i$ is a final state and the set $C(s_i)$ contains a final state of the opposite type, then there must exist an inconsistent (reachable) global state.

Consider a local state, $s_i$, and all incoming messages that can cause a transition. Define the *sender set* for $s_i$ to be the collection of all states, $t_j$, such that a transition from $t_j$ sends a message to $s_i$. We denote this set by $S(s_i)$.

Both the sender set and the concurrency set can be constructed from the reachable state graph. Moreover, the concurrency set for a state in a canonical protocol properly contains the sender set for that state.

## 4. SITE FAILURES

In this section we extend the model to include the failures of individual sites. The traditional method for detecting site failures, a *time-out*, is used. We model one recovery technique, and then show that only resiliency to single site failures is possible.

---

[1]This is true only in the absence of network partitions. When partitions are considered in a later section, we will introduce global transitions that change only the outstanding messages.

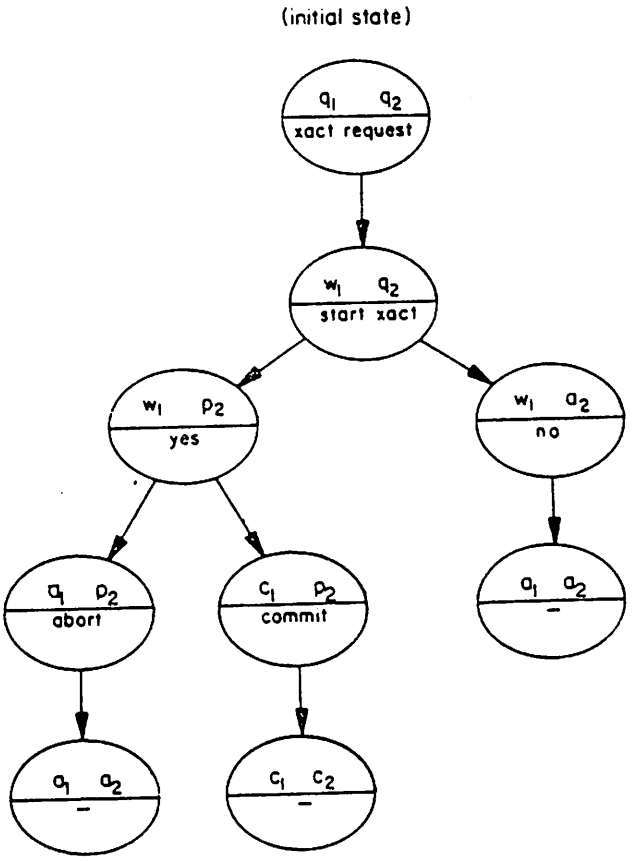


**Figure 4.** Reachable state graph for the two-phase commit protocol.

### 4.1. Failure Transitions

When a site fails a special type of transition, called a *failure transition*, is made. A failure transition reads all outstanding messages addressed to the site and writes a *time-out* message to each site. The failure transition originates in the state occupied at the time of failure and terminates in the state that the site will enter after it recovers. This recovery state could be one of the normally occupied states of the protocol or it could be part of a special recovery protocol. In a *resilient protocol* each local state must have a failure transition. Hence, the failure transition models the behavior of the site both at the time it fails and at the time it does local recovery.

The failure of a site is detected at an operational site by the receipt of a "time-out" message from the failed site. Such a "time-out" may (but not necessarily does) cause a transition to a special recovery protocol.

Like all other transitions, "failure" and "time-out" transitions must obey the rule of commit protocols: once a site has entered a commit (abort) state, all subsequent transitions must be to a commit (abort) state. The addition of "failure" and "time-out" transitions to a protocol greatly enlarges its reachable state space. Examination of the

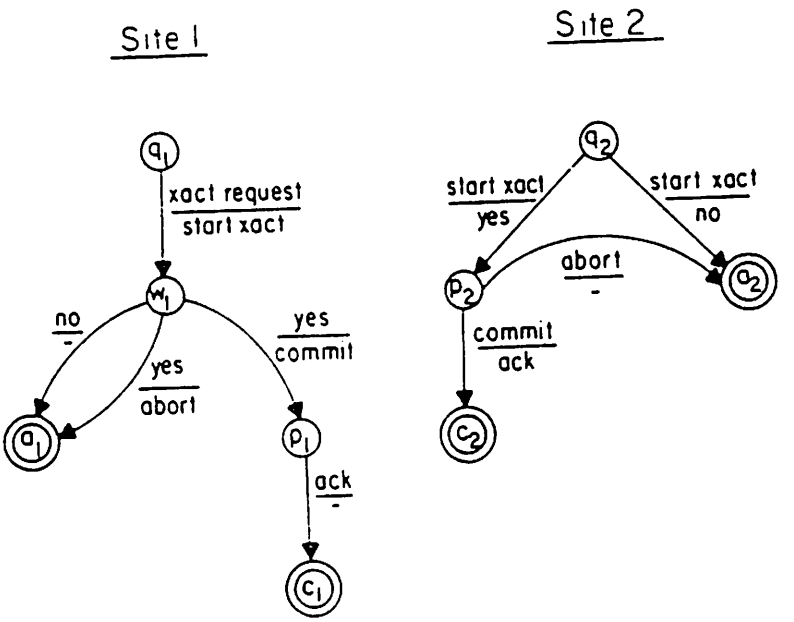


**Figure 5.** Two-phase commit protocol extended with an *ack* message.

extended reachable state space will reveal the mixtures of failures the protocol is resilient to.

### 4.2. Independent Recovery

In an independent recovery scheme, failed sites make a transition directly to a final state without communicating with other sites. Hence, no communication is attempted during the recovery process.

Independent recovery is interesting for several reasons. First, it is easy to implement and leads to simple protocols. One need not be concerned with messages to a down site being queued in the network or at another site which may be down when the failed site attempts to recover. Moreover, this model is of interest because it represents the most pessimistic recovery model. Proving the existence of a class of resilient protocols in this model implies its existence in all more sophisticated models of site failures.

The remainder of this section uses the independent recovery scheme.

### 4.3. Failure of a Single Site

Here we treat the restricted case that only one site can fail during the processing of a transaction. We first develop two rules for assigning "failure" and "time-out" transitions. The first rule deals with assigning "failure" transition. The failed site must make a transition consistent with the state of an operational site at the time of failure.

**Rule 1.** For a state $s_i$: if its concurrency set, $C(s_i)$, contains a commit (abort) state, then assign a "failure" transition from $s_i$ to a commit (abort) state.

The observant reader will note that the two-phase protocol of Figure 3 cannot satisfy this rule: the concurrency set of state $p_2$ contains both $c_1$ and $a_1$. Figure 5 gives a protocol similar to the two-phase one except for the addition of one state and an *acknowledgment* message to the commit message. Figure 6 gives the reachable global state graph for the protocol. Since no concurrency set contains both commit and abort final states, it is possible to assign "failure" transitions from all (nonfinal) local states according to rule 1.

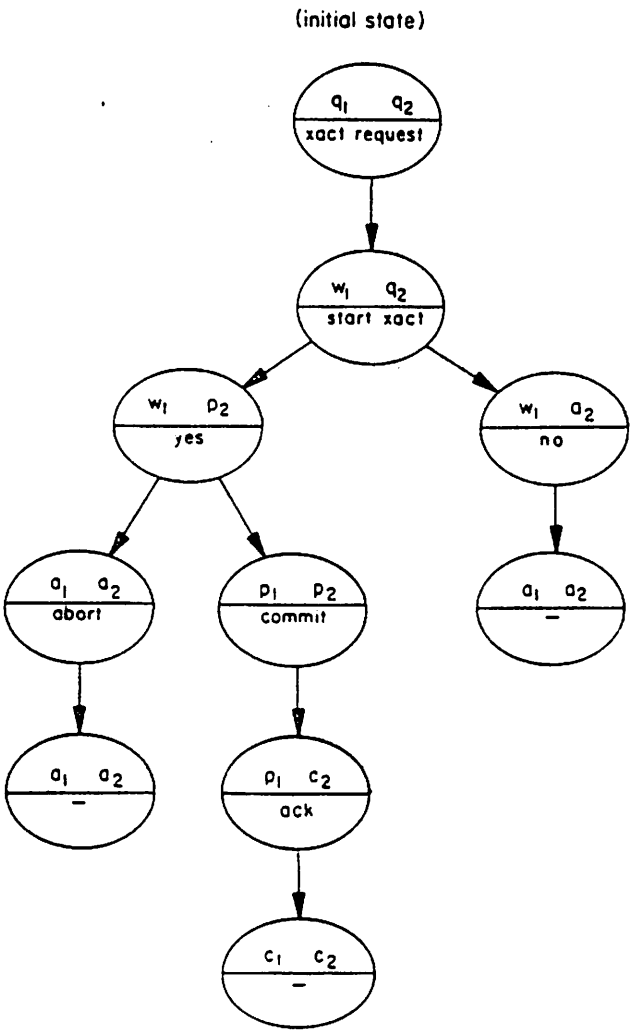


**Figure 6.** The reachable global state graph for the commit protocol in figure 5.

The second rule deals with "time-out" transitions.

**Rule 2.** For state $s_i$: if $t_j$ is in $S(s_i)$, the sender set for $s_i$, and $t_j$ has a failure transition to a commit (abort) state, then assign a "time-out" transition from $s_i$ to a commit (abort) state. If $S(s_i)$ is empty, then assign no "time-out" transition from $s_i$.

This rule is less obvious than the previous one. A "time-out" can be viewed as a special message sent by a failed site in state $t_j$ in lieu of a regular message. The "time-out" is received by the same state (in this case $s_i$) that normally receives the regular message. Moreover, the failed site, using independent recovery, makes a failure transition based solely on its local state. Hence, the site receiving the "time-out" must make a consistent decision.

Figure 7 illustrates the application of both rules. The protocol displayed is resilient to a single failure by either site. This can be verified by examining the reachable state graph for this protocol. In fact, the rules always yield a resilient protocol under independent recovery. Furthermore, since independent recovery is the most pessimistic (reasonable) model, *protocols obeying rules 1 and 2 are resilient to a single failure under any recovery model.*

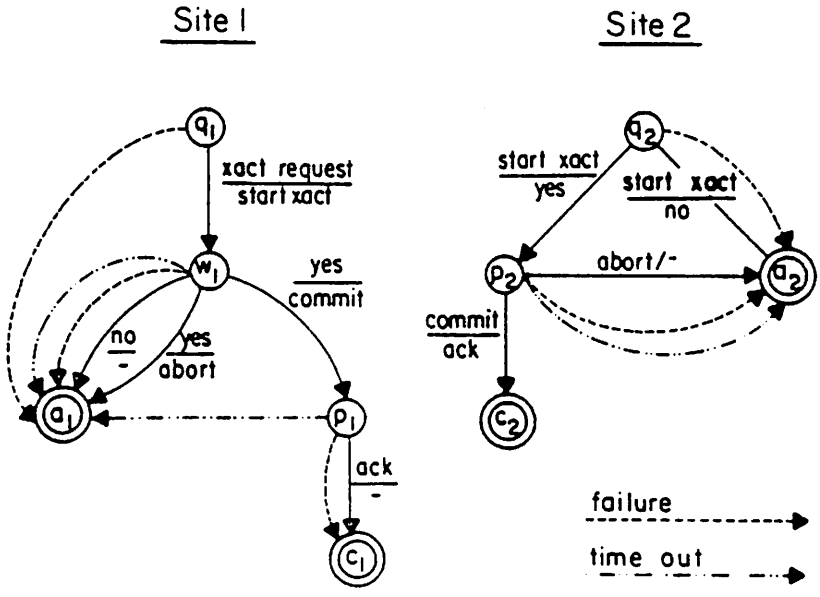


Figure 7. The extended two-phase protocol of figure 5 augmented with *failure* and *time-out* transitions according to rules 1 and 2.

**Theorem 1.** Rules 1 and 2 are necessary and sufficient for designing protocols resilient to a single site failure.

Although we have illustrated this result only for the two site case, it holds for multi-site protocols as well.

### 4.4. Two Site Failures

The rules given above are sufficient for protocols resilient to a single failure; however, such protocols are not resilient to the failure of two sites. This is demonstrated in the protocol of Figure 7. If double failures occur when site 1 is in state $p_1$ and site 2 is in state $p_2$, then an inconsistent final state results. In fact, no resilient protocol exists in this situation.

**Theorem 2.** There exists no protocol using *independent recovery* of failed sites that is resilient to two site failures.

Again, this result applies to the multi-site protocols as well as to the two site protocols.

## 5. NETWORK FAILURES

A network failure results in at least two sites which cannot communicate with each other. We model such a partition in two ways. In the first model, all messages are lost at the time partitioning occurs. In the second, no messages are lost at the time partitioning occurs; instead, undeliverable messages are returned to the sender.

We define a *simple partition* as one where all sites are partitioned into exactly two sets with no communication possible across the boundary. Since all partitions can be viewed as one or more occurrences of a simple partition, we specificly address two classes of failures: a single occurrence of a simple partition, and multiple occurrences of a simple partition (or multiple partition for brevity).

We consider a protocol to be *resilient* to a network partition only if it enforces the **nonblocking** constraint. That is, the protocol must insure that each isolated group of sites can reach a commit decision consistent with the remaining groups. Since the commit decision within a group is reached in the absence of communication to outside sites, this problem is very similar to the independent recovery paradigm presented in the previous section.

Throughout this section we will restrict our attention to network partitions exclusively and ignore the possibility of site failures.

### 5.1. Partitioning With Loss of Messages

As previously, a site detects the occurrence of a partition by a "time-out" and can make a transition on such a message. First, we treat the two site case.

A network partition is modeled as a special type of global state transition. Until now all global state transitions were triggered by one local state transition. However, a network partition is modelled as a global state transition that erases all outstanding messages and "time-outs" are sent to all sites.

After a partition has occurred, each site will make a "time-out" transition. In fact, we have a situation analogous to the double site failure in the independent recovery model of the previous section except that "time-out" rather then failure transitions are

made. It can be shown that a solution to the double failure problem implies a solution to this problem. An immediate consequence of this result is is the next theorem.

**Theorem 3.** There exists no two site protocol that is resilient to a network partition where messages are lost.

It is easy to generalize the model to partitions involving more than two sites and prove theorem 3 for the more general environment.

### 5.2. Partitioning with Return of Messages

In this situation we assume that the network can detect the presence of a partition and return undeliverable messages to their senders. This appears to represent the most optimistic model for partitions, while loss of messages is the most pessimistic one.

In this case a partition causes a global state transition that redirects all undeliverable messages back to their senders and writes "time-out" messages to the recipients of undeliverable messages. As before, a site makes a transition on a "time-out" message. Also, a site makes a transition when an undeliverable message is returned to it.

#### 5.2.1. Two Site Case

To study this optimistic situation, we now define two design rules that resilient protocols must satisfy.

**Rule 3.** For a state $s_i$: if its concurrency set, $C(s_i)$, contains a commit (abort) state, then assign a "time-out" transition from $s_i$ to a commit (abort) state.

Here site i in state $s_i$ was expecting a message when the partition occurred. Instead, it received a "time-out". This site will then make a decision to abort or commit the transaction consistent with the state of the site sending the undeliverable message.

The second rule deals with the site sending the undeliverable message. It must make a commit decision consistent with the decision of the intended receiver.

**Rule 4.** For state $s_j$: if $t_i$ is in $S(s_j)$, the sender set for $s_j$, and $t_i$ has a "time-out" transition to a commit (abort) state, then assign a "time-out" transition from $s_i$ to a commit (abort) state upon the receipt of an undeliverable message.

An observant reader will note that these rules are equivalent to the rules given for independent recovery of failed sites. In fact, the two models are isomorphic. To illustrate the equivalence, consider the information conveyed by a "time-out" message from a failed site. The following is true when the operational site, i, receives the "time-out" indicating a failure of the other site.

(1) the last message sent by site i was not received (the other site failed prior to its receipt),

(2) communication with the other site is impossible (it is down),

(3) the other site will decide to commit using independent recovery.

Exactly the same conditions hold when an undeliverable message is returned to site i.

Applying the above design rules to the protocol of Figure 5 yields the protocol illustrated in Figure 8. As expected, the protocol is identical, to the protocol of Figure 7.

In light of this isomorphism, theorem 4 is not surprising.

**Theorem 4.** Design rules 3 and 4 are necessary and sufficient for making protocols resilient to a partition in a two-site protocol.

### 5.2.2. Multisite Case

In the absence of site failures simple partitions in multisite protocols are not very different from partitions in a two-site protocol, since preserving consistency within a connected group of operational sites is straightforward. Thus, design rules 3 and 4 can be extended to multisite protocols in a straightforward way. This leads to the following result.

**Corollary 1.** There exist multisite protocols that are resilient to a **simple** partition when undeliverable messages are returned to the sender.

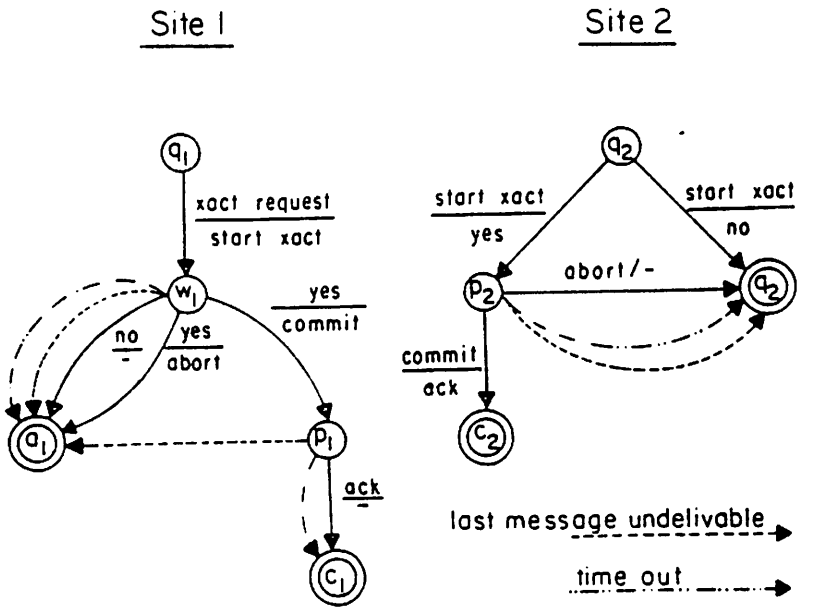


**Figure 8.** The extended two-phase commit protocol (of figure 5) augmented with *time-out* transitions and transitions on undeliverable messages according to rules 3 and 4.

This result is the complement of the results obtained from the pessimistic model discussed earlier. The models differ in their handling of outstanding messages when the network fails: in the pessimistic model, they are lost; whereas in the optimistic model, they are returned to their sender. Since this is the only difference between the two models, the next result is implied.

**Corollary 2.** Knowledge of which messages were undelivered at the time the network fails is necessary and sufficient for recovering from simple partitions.

We now turn to multiple partitions. Since we are dealing with an optimistic situation, we assume that "time-outs" and undeliverable messages are unaffected by additional partitions. This, in effect, is an assumption that the network is partitioned into all subsets simultaneously, and that the process does not happen sequentially.

Even in this (overly) optimistic model, our results are negative, which implies negative results for all realistic partitioning models.

**Theorem 5.** There exists no protocol resilient to a multiple partition.

Therefore, even complete information about message traffic during a partition, and in particular, information about which messages are undeliverable, is insufficient for recovering from multiple partitions.

## 6. CONCLUSIONS

We have presented a model of transaction processing in a distributed environment and used it to study both site failures and network partitions. Our results tend to be more illuminating than surprising. Using independent recovery, the class of recoverable site failures has been identified. Using an optimistic model for network partitions, we have shown that (nonblocking) recovery is possible only for simple partitions. In a more realistic model, recovery from a simple partition is not always possible.

We feel that the model is an appropriate vehicle for further study of resilient protocols. The topics that we are currently investigating include:

(1) Generalizations of independent recovery. We plan to include the possibility of queuing messages for down sites as in [HAMM79].

(2) Treatment of degrees of resiliency. In this paper protocols were either resilient or not. We plan to generalize this to a degree of resiliency between 0 and 1 and look for minimal state protocols with a given resiliency.

## Acknowledgments

The authors wish to thank Ken Birman, Ken Keller, and Larry Rowe for their useful comments and for valuable discussions.

[ALSB76] Alsberg, P. and Day, J., "A Principle for Resilient Sharing of Distributed Resources," *Proc. 2nd International Conference on Software Engineering*, San Francisco, Ca., October 1976.

[GRAY79] Gray, J. N., "Notes on Database Operating Systems," in *Operating Systems: An Advanced Course*, Springer-Verlag, 1979.

[HAMM79] Hammer, M. and Shipman, D., "Reliability Mechanisms for SDD-1: A System for Distributed Databases," Computer Corporation of America, Cambridge, Mass., July 1979.

[LAMP76] Lampson, B. and Sturgis, H., "Crash Recovery in a Distributed Storage System," Tech. Report, Computer Science Laboratory, Xerox Parc, Palo Alto, California, 1976.

[LORI77] Lorie, R., "Physical Integrity in a Large Segmented Data Base," *ACM Transactions on Data Base Systems*, Vol. 2, No. 1, March 1977.

[MENA79] Menasce, D. A. and Muntz, R. R., "Locking and Deadlock Detection in Distributed Databases," *IEEE Transactions on Software Engineering*, Vol. SE-5, No. 3, May 1979, pp. 195-202.

[ROTH77] Rothnie, J. B., Jr. and Goodman, N., "A Survey of Research and Development in Distributed Database Management," *Proc. Third Int. Conf. on Very Large Databases*, IEEE, 1977.

[SKEE81] Skeen, D., "Crash Recovery in a Distributed Database Management System," Ph.D. Thesis, EECS Dept., Univ. of Calif., Berkeley (in preparation).

[STON79] Stonebraker, M., "Concurrency Control and Consistency of Multiple Copies in Distributed INGRES," *IEEE Transactions on Software Engineering*, May 1979.

[SCHA78] Schapiro, R. and Millstein, R., "Failure Recovery in a Distributed Database System," *Proc. 1978 COMPCON Conference*, September 1978.

[SVOB79] Svobodova, L., "Reliability Issues in Distributed Information Processing Systems," *Proc. 9th IEEE Fault Tolerant Computing Conference*, Madison, Wisc., June 1979.