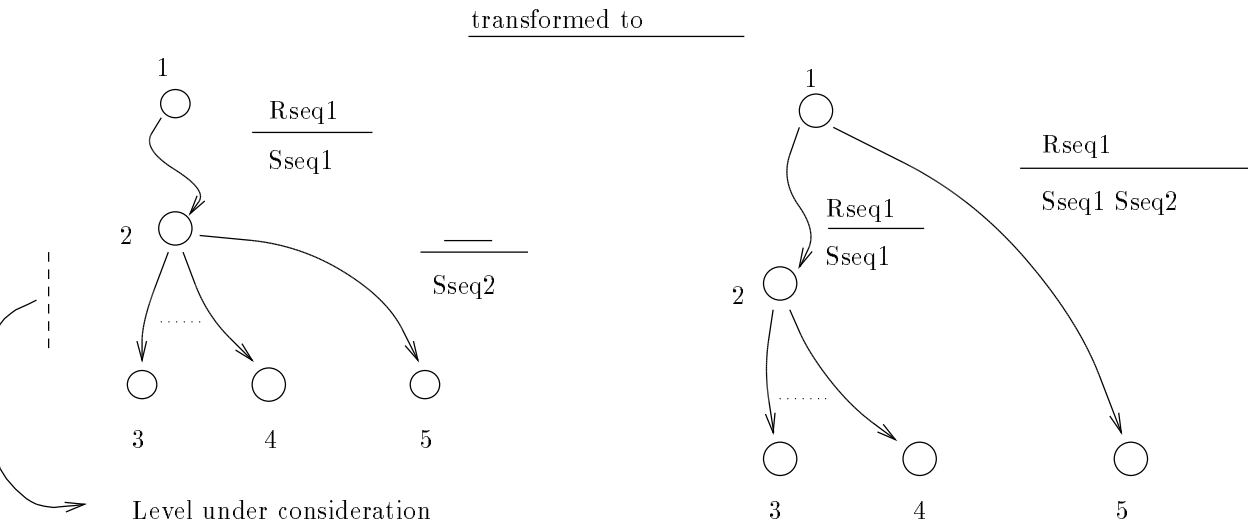


Figure 3: The Transformation for Lemma 1

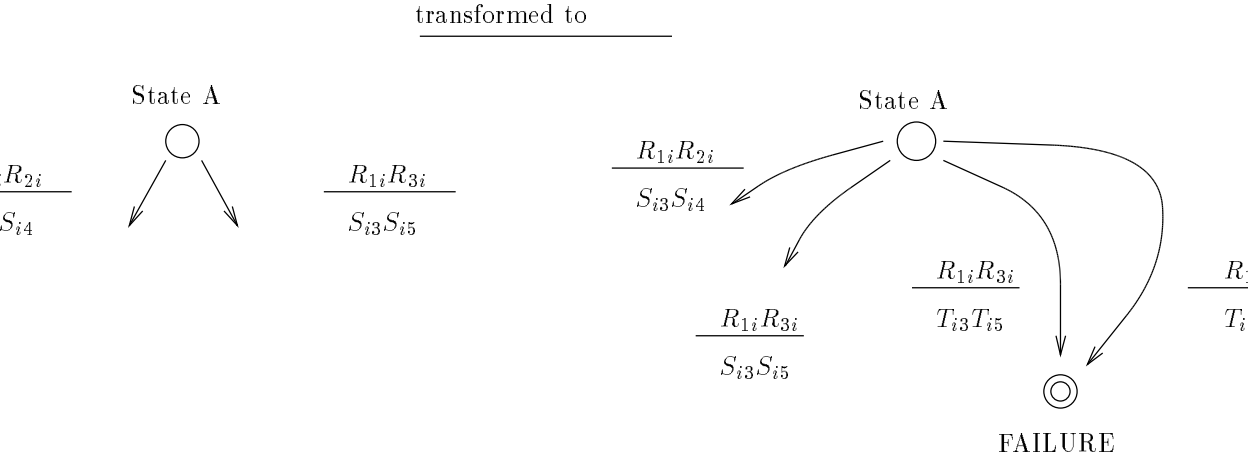


Figure 4: Modeling Failures

References

[1] Davidson, S., *et al.* 1985. Consistency in Partitioned Networks. *ACM Comput. Surv. Vol.17, No.3*, 341-370.

[2] Dolev, D., *et al.* 1987. On the Minimal Synchronism Needed for Distributed Consensus. *JACM. Vol.34, No.1* (Jan), 77-97.

[3] Dwork, C., *et al.* 1984. Consensus in the presence of Partial Synchrony. *3rd ACM Symp. on PODC.*, 103-118.

[4] El Abbadi, A., *et al.* 1985. An Efficient, Fault-Tolerant Protocol for Replicated Data Management. *4th ACM SIGACT-SIGMOD Symp. on PODS.*, 215-229.

[5] Fischer, M., *et al.* 1985. Impossibility of Distributed Consensus with One Faulty Process. *JACM. Vol.32, No.2* (April), 374-382.

[6] Skeen, D. 1982. Crash Recovery in a Distributed Database System. Doctoral Dissertation, Dept. of Elec. Engin. and Comp. Sci., Univ. of Calif., Berkeley (May).

[7] Soparkar, N.R., and Silberschatz, A. 1989. Data-value Partitioning and Virtual Messages. *Submitted for Publication.*

Proof : Consider two failure-prone processors linked by a failure-prone communication link. As discussed above, the theorem holds in this restricted case — and therefore, it holds for the general situation. □

It is clear that the detection and commitment problems are similar. The above negative result indicates a two-fold problem for network partitions. Commitment protocols based on partition detection cannot work in a general setting. Commitment protocols could be made to execute before, after, or concurrently with a detection protocol. If the commitment is done before the detection, then a positive detection of a partition failure implies that the termination decision may be incorrect. A decision of the absence of a partition failure reached before the commit protocol does not rule-out the possibility of subsequent failures. Finally, assume that the two protocols execute concurrently. Since the processors execute steps from the protocols atomically, it is easy to show that a protocol cannot exist. The proof lies in the observation that concurrent execution of two protocols is equivalent to a single protocol for commitment.

5.3 Delineating Failures Abstractly

Consider now the situation where network partitioning is not well-behaved. Also assume that the severed links get restored. In such situations, a natural question arises as to when one can assert that a partition has occurred. Dynamic partitions of this kind occur in realistic systems with links that may lose messages. In such environments, partitions may not have a precise definition, and consequently, their detection also lacks a precise description.

It may therefore be useful, to define failures more abstractly. For example, we may define a 'calamitous' failure as a system failure which precludes the existence of a bounded commit protocol. Similarly, other failures may also be defined abstractly. For example, we may consider failures where there does not exist a *finite* consensus protocol, or failures which only have *probabilistic* protocols applicable. In each, the worst situation would dictate what protocols are available. It may be noted that in [2], a taxonomy along these lines is available in a more restricted sense.

Let us examine how such a taxonomy would be useful for the design of practical systems. Given a system with certain characteristics, and a particular application that needs to be implemented, it becomes possible to decide *a priori* what protocols are necessary. Similarly, given that a particular application requires protocols with certain characteristics (e.g., a non-blocking characteristic), necessary system characteristics may be identified. Thus, information about the protocols dictated by the application can be matched with the failure classes.

6 Conclusions

The problems encountered in designing consensus protocols in failure-prone systems appear to be devoid of practical solutions. Some methods that have been previously suggested to avoid these problems are not generally applicable. Hence, applications designed using traditional methods that employed such protocols must either be redesigned, or they must be executed only in environments where partitions do not occur. If neither of the above alternatives is feasible, then the only way to solve the difficult problems caused by failures is to examine radically different techniques for the applications that presently make use of the protocols that we have examined (as we do in [7]). These techniques should account for the more practical characteristics exhibited by real systems.

Proof : For the case with three processors, the proof is as in [6]. In the case that there are more than two non-communicating groups, the result follows directly from the previous case, Lemmas 1 and 2, and discussions above. □

In fact, if we do not allow the unrealistic situation that undeliverable messages are returned to the original senders, and instead, impose the link failures with timeouts alone, then we have a negative result for $m \geq 2$, the proof of which is in [6] as well.

We thus conclude that, from a practical viewpoint, nothing can be done to obtain a non-blocking commit protocol for a distributed system.

5 Detection of Network Partitions

From the above discussions it is clear that if a network partition does occur, little can be done. Although some methods have been considered to bring about commitment inspite of network partitions, these are applicable to cases that are not entirely general (cf. [1, 4]), and are usually interesting only from a theoretical viewpoint. One of the problems that is not obvious is how a partition may be detected. This is an important aspect of failure resiliency since several protocols have been proposed that make use of the detection of network partitions. As we will see next, these have only limited applicability.

5.1 Detection and Communication Failures

Consider the case of two failure-prone processors connected by a communication link that fails by producing timeouts. In case a link fails, each processor is able to detect that it is isolated, but not whether a partition has occurred. Now consider the case with three processors that are interconnected by links that can return undeliverable messages. If all the links fail, the processors can determine that they are isolated from the others, but are unable to detect whether the remote processors are isolated from each other as well. This observation is important and we give an example to illustrate a situation where such information may be used. In a fully replicated distributed database, suppose that a protocol allows isolated groups to update data accessible to them if a majority of the processors are present in the group. If an isolated group ascertains that no other groups have a majority, then accessible data can be safely read.

In [6], the relation between independent recovery and network partitions in the context of commit protocols is discussed. We observe here that there is a close similarity in detecting whether an isolating network partition failure of the system has occurred, and the commitment problem. This is prompted by the fact that there are no commit protocols for either of the situations outlined above. In the case of two processors with the link having a property that undelivered messages are returned, there exist commit protocols, and it is easy to see that isolation of the other party can be detected.

Several methods have been suggested to allow processing to continue even after a network partition has been detected [1]. These methods assume that some method is used to continue processing within a group after the partition failure is detected.

5.2 Detection and Commitment

Let us consider the nature of the detection of a partition restricting attention to well-behaved partitions. Detecting a stable partition implies that every processor concludes that a partition has occurred within a bounded number of steps. Also, if one group of processors is able to detect a partition, so should the others. Let us make the detection less stringent, by requiring that the processors *agree* upon whether or not there has been a partition, and do so within a bounded number of steps — without necessarily there being a partition.

Theorem 2 *: It is not possible, in general, to detect network partitions.*

non-communicating groups of processors. We assume that a set of the links fail simultaneously so that their failure produces at least one more isolated group. Further, assume that all the link failures are essential in the sense that if any one of them did not fail, the number of non-communicating groups formed would be fewer. We call such a restricted partition failure to be *well-behaved*. Note that in a network partition failure, the non-communicating groups that form will have some fixed set of processors each. Well-behaved partition failures are restrictions of general partition failures.

Lemma 2 *: Consider a system where the only possible network partition failure is well behaved and forms the non-communicating groups g_1, g_2, ..., g_m of processors. If there is a commit protocol to handle such a situation, then there exists one to handle a system of m processors with the only possible network partition failures being well-behaved and resulting in the isolation of every processor from the others.*

Proof : The proof consists of a simulation of the actions of each group of processors in the system with the known protocol by one automaton for each processor in the other system. Consider the automata for the given protocol for the general case. Construct a global state graph G_i for each group g_i ignoring all automata not in the same group. Construct automata A_i from G_i for each processor in the other system as follows. The start state of G_i is made the start state of A_i; the commit (abort) state of G_i with no local failures is made the commit (abort) state for A_i; all other final states of G_i are coalesced to form the failure state for A_i. The other states in G_i are ordinary states of A_i. All transitions to the failure state in A_i are accompanied by timeout messages to all processors to which A_i is linked. Every message in the former system that is directed from a processor in a group g_j to one in group g_k is relabeled to indicate that it is directed from automaton A_j to A_k. It is appropriate to draw attention to the network topology implied in this lemma. In the system with m processors, processors P_i and P_j have communication links if processors in g_i and g_j exchange any messages. Also, in any system, timeout messages from different links are distinguishable but not those from the same link. Thus, timeouts generated due to link and processor failures are indistinguishable.

Every message sent within the group G_j gets subscripted by 'jj' after the above changes to the subscripts of the messages. Each such message is removed from the READ and WRITE sequences since these messages amounted to intra-processor communication. Any empty READ sequence may be removed by the construction of Lemma 1. Also note that the partition failure in the system that we have constructed occurs from one global state for all the A_i automata in the same manner as for the original system. At this point the description of the desired protocol is complete and the reader may verify it is correct by observing that the new system commits or aborts only when the original one does. □

Lemma 3 *: For a given system of n processors ($n > 1$), if there is a commit protocol to handle well-behaved network partitions that isolate the processors, then there is a protocol which can handle such partitions in a system with less than n processors.*

Proof : Consider a system of $(n-1)$ processors. From the protocol for n processors, construct an automaton B to simulate the global graph for the automata for two of the processors as in the proof of Lemma 2. □

Lemmas 2 and 3 reduce the task of obtaining the impossibility result concerning the non-existence of a commit protocol for a general system by allowing us to concentrate on a system with exactly n processors.

4.3 Non-Existence of Commit Protocols

Even restricting attention to subcases that are easier to handle, and favorable situations, we have the following results.

Theorem 1 *: There exists no commit protocol to handle the case where well-behaved partitions may occur in the network of a system with three interconnected processors. Furthermore, there exists no commit protocol to handle the case of well-behaved network partitions that form more than two non-communicating groups.*

4 Failures and Commit Protocols

In this section we first consider the modeling of failures in our formal system model. The infeasibility of commit protocols that are resilient to different types of failure is demonstrated by the use of the formal model.

4.1 Modeling Failures

Processor failures are modeled by failure transitions in their corresponding automata. A failure transition is added from every non-final state by an edge that has only 'timeout' messages (each denoted by T) for the SEND sequence. All these edges terminate at the final failure state of the automaton. To model the fact that a failure could occur irrespective of the set of messages that may be read at that state, for each non-failure edge, a failure edge is defined with the same READ sequence, directed to all automata that were to be sent messages.

We consider two kinds of failures in the communication links :

- Undeliverable messages are returned to the sending processor by the communication network. This models the ability of a processor to detect a link failure unequivocally. Thus, the buffer corresponding to the link in question changes the subscript of the message. We assume that the original sender is able to recognize that a returned message is one that it had sent earlier. A timeout message T_{ji} is introduced since the intended recipient of the original message should detect the failure.

- The processors cannot detect that the communication links have failed directly. Instead, a processor depends on timeouts alone to detect that some component has failed. To model this, the buffer replaces all messages in it by a pair of timeout messages directed to each automaton that it links.

In both these cases, the timeout messages are assumed, quite reasonably, to be indistinguishable with respect to their origin. Figure 4 provides an example of the manner in which timeout transitions are introduced.

Note that the situation of returned messages is more favorable as compared to the dual timeouts since all returned messages may be regarded as timeout messages. Similarly, assume infinite size buffers which is more favorable since any protocol for the finite case can be simulated by a protocol for the former. Further, assume that all communication is two-way. Consider two processors P_i and P_j which can communicate directly. Although the failure of communication from P_i to P_j may be independent of the communication from P_j to P_i, we assume that both fail simultaneously since, in general, both may simultaneously fail. Also, by restricting the links that may fail, we insist that a network partition occurs only due to the failure of some specified sets of links. Similarly, we insist that network partitions result from link failures alone — and due to processor failures. Furthermore, we make the realistic assumption that link failures take an arbitrary amount of time to be restored (lost messages are equivalent to *fail-stop* link failures, arbitrarily long delays also exhibit the characteristics of link failures). The fail-stop behavior of a link is a subcase of the situation that the link does get restored. Another restriction on a general situation occurs when all the links involved in a network partition fail while the system is in one global state. A general case may have links that fail one-by-one — thereby forming the non-communicating groups gradually.

In all the above cases, it is clear that an impossibility result for failure resilient protocols in the restricted cases implies the same for the general situation. However, it should be noted that the general cases are more relevant from a practical standpoint as they are more realistic.

4.2 Failures in a Large System

Let us assume that a network partition failures occur in the system in only one specific manner. We prove that it is not possible to have a commit protocol for even such partition failures, and therefore, a general case of partition failures could not have commit protocols. We require that only a fixed set of links fail to form m

amount of time to traverse the links. Message-order synchrony enforces the reception of messages to be in the same order in which they are sent. The mechanism for transmission differ between point-to-point and broadcast techniques. Finally, the atomicity of receives and sends refers to the atomicity of the transitions of the automata that model the processors.

Note that communication synchrony cannot be directly represented in the formal model; instead, we state it for a system. However, the presence of 'timeout' transitions [6] indirectly indicates that the communication may be synchronous. Also note that other types of failure (e.g., in the network, or malicious failures) could not possibly *improve* the resiliency of any protocol. This is because we could always restrict our attention only to those runs of the system where such failures are absent.

The different parameters govern the protocols that we can design for the system. Several of these are 'favorable' to the solution of the problem, while their counterparts are 'unfavorable' [2]. Essentially, the non-existence of a protocol in a favorable situation implies the same for the unfavorable case, and contrapositively, the existence of a protocol in an unfavorable situation implies the same for the favorable case. This may be established by simulating one system by the other. Since impossibility results are under consideration, we can restrict attention to the favorable parameters alone.

Consider a system with some set A of parameters. Suppose that by placing some restriction on the kinds of transitions that can occur in the system we are able to obtain a set of runs of the system which is the same as all the possible runs of the system with another set B of parameters. Then the resiliency exhibited by the system with parameter set A can be no better than that with parameter set B. Further, a set of parameters X is more favorable than a set Y if the actions of the system with Y can be simulated in a bounded number of steps by the system with X.

The transitions in the local automata are considered to occur as a result of the environment in which they function rather than some internal changes. Hence, we assume that every outgoing edge from a vertex of the automata defining the protocol has a non-empty READ sequence. It will become clear that this also allows us to create timeout transitions from every state to represent failures.

We demonstrate that every automaton in our model can be equivalently expressed in this form. Two automata A and B are *equivalent* if :

1. For every sequence of messages read by A, it is the case that every output sequence that can be produced by A, can also be produced by B, and *vice versa.*

2. For every message sequence read by A, and the set of possible message sequences produced by A that make it reach a final state, B also enters a similar type of final state with the same corresponding sequences, and *vice versa.*

Lemma 1 *: An automaton* A *can be equivalently represented by another automaton* B *which has a non-empty READ sequence for every.*

Proof: We construct an automaton from the description of A, and by a sequence of transformations, convert it into the automaton B. We use the transformation provided in Figure 3 repeatedly. The transformation applies to the level of A under consideration on all the transitions from a state in that level that have empty READ sequences.

Beginning with the start state, partition the edges of A into numbered levels according to their distance from the start state. For each level in turn, apply the transformation from Figure 3 where applicable to all edges in that level. There are only a bounded number of such levels that need to be considered. With each transformation, no paths with new labels are added or removed. Note that the number of vertices or edges does not increase. Clearly, the behaviors exhibited by A and B are the same. The fully transformed automaton is an example of B. □

All states, except for the distinguished start and final states, have both incoming and outgoing edges. The automata are restricted to have all paths of bounded length to model a non-blocking protocol. All outgoing edges from the start state are required to have a non-empty READ sequence to model the initiation of the protocol by an application program. The automaton depicted in Figure 1 provides an example of the above definitions.

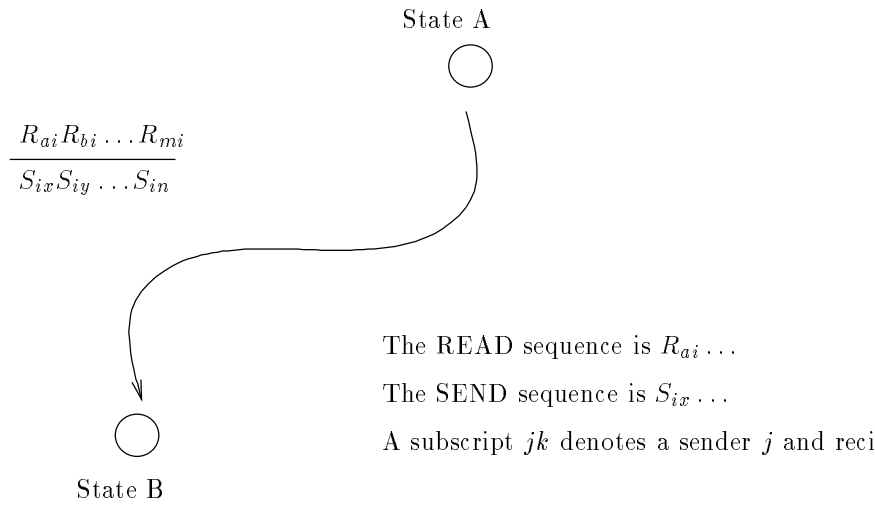


Figure 2: A Transition in Automaton i

An example of a transition from one state to another is provided in Figure 2. In the figure, assume that the transition, which is illustrated by the directed edge, occurs from state A to state B is part of an automaton with an identity i (the identity is used only to describe the model). The subscripts of the messages denote the identities of the sending and receiving automata.

Each link in the system is modeled as an infinite buffer. An automaton sends a message by 'placing' it in the buffer from which the receiving automaton 'retrieves' it. Note that there is a different buffer for each pair of processors that can send and receive — thereby allowing any one link to fail independently of the rest.

We define *global states, global transition graph, reachability, inconsistent final states* and *operational correctness* as in [6]. A *global commit* (*abort*) state is defined to be one where all local states are not failure states, and all non-failure local states are commit (abort) states. A global failure state is one where all local states are failure states.

The system of processors defined above and their interconnections is said to define a *commit protocol* if the global state graph is operationally correct, and there are paths from the global start state to every global commit state and to every global abort state. The concept of a 'unilateral abort' (i.e., the precipitation of a global abort by the decision of a single processor) is not an inherent part of the system. This is to account for protocols which may not require such unilaterally precipitated decisions.

3 System Parameters

To fully describe a system we must specify several parameters. Let us start with the following parameters that are formally defined in [2]. They are: *processor synchrony, communication synchrony, message-order synchrony, transmission mechanism* and *receive/send atomicity.* Processor synchrony implies that it is not possible for one processor to execute some unbounded number of steps without all the other processors executing at least one step each. Communication synchrony indicates that messages require some bounded

which a failed site is able to recover *without* any information obtained from the other sites. The impossibility results obtained in [6] are, unfortunately, quite general, and this is exhibited in our paper.

Detection of failures is important since it enables processors to take corrective actions. Most approaches that permit processing in the presence of failures make the assumption that the failures that they deal with may be detected. However, as we demonstrate, this assumption is very often not valid in most realistic situations.

The rest of this paper is organized as follows. The formal model for the system is defined in Section 2. In Section 3 the model is extended to include several different parameters of the system. In Section 4, the characteristics and modeling of the failures is described, and the effect that failures have on the development of commit protocols is examined. The difficulty of partition failure detection is demonstrated in Section 5 and, in view of these difficulties, a different approach to the taxonomy of failures is suggested. Finally, Section 6 constitutes the conclusions.

2 System Model

We abstract away from the details and present a model that solely deals with the problem of commitment. The description consists of the processor definitions, the communication link descriptions, and finally, the definition of a commitment protocol.

Let the system consist of n, $(n \geq 2)$, processors $P_1, P_2, \ldots, P_n$ that communicate with one another. Each processor follows a deterministic protocol involving the transmission and reception of messages. Without

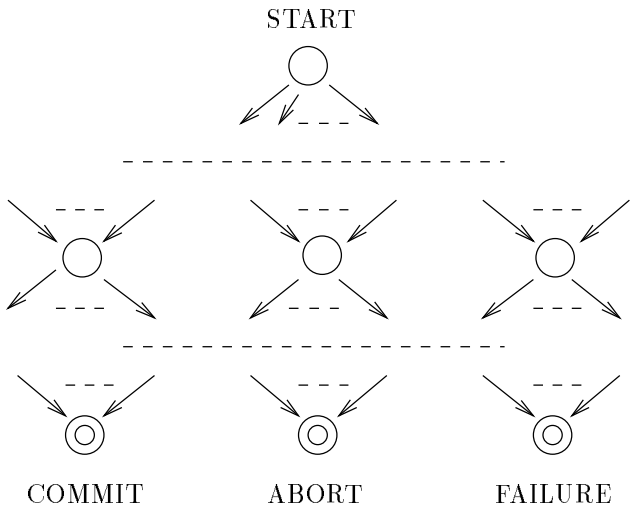


Figure 1: An Automaton

loss of generality, we consider each processor to be modeled by an automaton. Each automaton is defined by a directed acyclic graph with labeled edges denoting the sending and reception of messages, and vertices denoting the states of the automaton. The edges are labeled with READ and SEND sequences, and are regarded as atomic transitions made by the automaton. The READ sequence denotes the messages read by the automaton in the transition from the state at which the edge emanates, and the SEND sequence denotes the messages sent by the automaton in the transition. There is a distinguished *start* state with only outgoing edges, and two distinguished final states, *commit* and *abort*, with only incoming edges. We also have an optional *failure* state as another final state (in [6], such a state is not included since attention was restricted to independent recovery).

Distributed Systems Are Indecisive *

Nandit Soparkar
Abraham Silberschatz

Department of Computer Sciences,
University of Texas at Austin,
Austin, Texas 78712

Abstract

A major research topic in distributed systems is the issue of reaching a consensus on a value, say a single bit, in the presence of different types of failures. This is a widely investigated issue since several important distributed applications, when viewed abstractly, are essentially this problem. In this paper, we address the commitment problem — which is the consensus problem in the context of distributed databases. Specifically, the problem of reaching a consensus in the presence of network partition failures is considered. Several different characteristics that a system may have are examined and the problem is shown to be devoid of a solution for virtually any reasonable system. The related problem of detection of failures is seen to be difficult, often not possible. In view of these difficulties, suggestions for the study of failures from a different perspective are made.

1 Introduction

A distributed system consists of several computing units, the processors, that communicate by sending messages on communication links. For such a system to be of practical use, it must be capable of performing certain tasks inspite of failures of either the processors, or the communication links. Abstracting from several applications for such systems, the concept of a Distributed Consensus [2, 5] has emerged. The requirement is that all the processors should be able to agree on a value of a boolean variable. While this is not difficult to achieve for a system which is failure-free, it is quite difficult to realize when various components of the system are susceptible to failures.

The problem of designing protocols for achieving such a consensus has received wide attention [1, 2, 3, 4, 5, 6]. In this paper, we confine ourselves to *non-blocking* commit protocols, where the participants arrive at a consensus in a bounded number of locally measured steps. It is demonstrated that achieving non-blocking commitment for most practical situations is theoretically impossible. Our proofs are based on the solution of the problem when a *network partition* may occur. This failure is one where the failures divide the system into several groups of processors that are unable to communicate with one another. Related aspects, such as the detection of the failures, are also examined, and shown to be difficult. Finally, a different approach to the study of failures in distributed systems, which is more suited to the design of practical systems, is suggested.

The model that we use is similar to the one used in [6] extended in several ways. This is necessitated by the wider scope of our paper. In [6], the results are restricted mainly to *independent recovery* techniques in

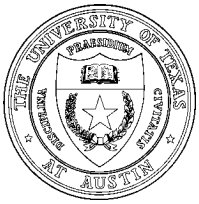
*This material is based in part upon work supported by the Texas Advanced Technology Program under Grant No. ATP-024, the National Science Foundation under Grant No. IRI-9106450, and grants from the IBM, NEC, and Hewlett-Packard corporations.

DISTRIBUTED SYSTEMS ARE INDECISIVE

Nandit Soparkar and Abraham Silberschatz

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712-1188

TR-92-17 April 1992



DEPARTMENT OF COMPUTER SCIENCES
THE UNIVERSITY OF TEXAS AT AUSTIN

AUSTIN, TEXAS 78712