COMMITMENT IN A PARTITIONED DISTRIBUTED DATABASE

K.V.S. Ramarao
Department of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260

Abstract

Network partition is among the hardest failure types in a distributed system even if all processors and links are of *fail-stop* type. We address the **transaction commitment** problem in a partitioned distributed database. It is assumed that partitions are detectable. The approach taken is **conservative** - that is, the same transaction cannot be committed by one site and aborted by another.

A new and very general formal model of protocols operating in a partitioned system is introduced and protocols more efficient than the existing ones are constructed.

1. Introduction.

Fault-tolerence is among the most important requirements of a distributed system. Various failure types have been studied in the literature and numerous protocols have been proposed to handle these failure types. Most prominent among these failure types are the processor (site) failures. Two classes among these have attracted much attention: *malicious failures* and *fail-stop failures*. A component with a fail-stop failure property either follows the protocol it is expected to follow, or simply stops at some arbitrary point during the protocol. A malicious component on the other hand can exhibit behaviour arbitrarily different from its expected behaviour and in fact can intentionally try to sabotage the operations of the system. A simultaneous failure of several sites and/or links can, depending on the topology of the network, lead to a **partition** of the system, even if all components are of fail-stop type. A partitioned network is divided into a number of subnetworks such that no subnetwork can communicate with another subnetwork. Focus of this paper is on distributed systems consisting of only fail-stop sites and links.

Simplest and most important among the fundamental problems in Distributed Computing is that of **reaching consensus.** The problem considered in this paper is a special case of this general problem and is known as the **transaction commitment** problem in the literature [23]. Protocols exist in the literature that solve the commitment problem when only sites fail [21]. Failure of only communication links not leading to a network partition can be handled by lower level protocols. Finally, it is not possible to solve the transaction commitment problem in a system where arbitrary partitions are possible [23], while there are certain special cases that allow solutions [23], [19].

The work presented in this paper focusses on the design of protocols that solve the transaction commitment problem in a large subsystem when arbitrary partitions occur while guaranteeing that the same transaction cannot be commited by a site and aborted by another.

The organization of the paper is as follows : Section 2 presents the problem specification and describes the formal model used. Section 3 studies the characterization of commit protocols by seperating their essence and the means. Section 4 explores the possible protocols in dealing with network partitions. Related work is discussed in section 5 and Section 6 concludes the paper.

2. Problem specification.

Let G = (V,E) be the network representing the distributed system. Each member of V is identified with a *site* and each member of E with a *link*. A subgraph T of G is a **partition** of G if G\T is disconnected. A connected component S of G\T is a *group* of T. We identify S with the sites in S and ignore the links in S when no confusion can arise. Following is an important assumption we make:

Network partitions are detectable.

We shall not consider here how a network partition can be detected. It is a hard problem in itself and the reader is referred to [15] for the description of a detection procedure and its integration with a concurrency control protocol. The technique is essentially time-out based and assumes the appropriate level of synchrony in the sytem (among the processors and communication and preservation of the order of messages on each link, etc). For our purposes here, we only need that the messages on each link are received in the order they are sent. No other synchrony is explicitly used.

The transaction commitment problem is defined as follows: *initially, each site i has a binary private value v_i (both 0 and 1 possible) and any protocol solving the problem guarantees that each site irreversibly decides on a binary value (dependent on all initial values) by the end the protocol such that a) all sites decide on the same value, b) a value of 1 is decided upon only if the initial values are all 1, and c) every operational site decides by a fixed bounded global time t(n) where n is the number of sites participating in the protocol (this global clock need not be accessible to the processors themselves).*

It is clear that the variant of the transaction commitment which does not require the operational sites to decide within a bounded time is always solvable. Protocols that solve this variant of the transaction commitment problem are known as **commit protocols** in the literature. A commit protocol is **nonblocking** to a failure type if it can guarantee the termination within a bounded time in spite of an arbitrary instance of that failure type [21].

Much of the terminology used in the following formal model is taken from [11], [8].

Commit protocols can be modelled by (possibly infinite-state) automata, one at each participating site. In the rest of the paper, we identify the automaton running at a site with that site itself when no confusion can arise. We need to clarify what is possible within an atomic step at a site: we allow local processing and a state change, the receipt of a set of messages, and the generation of the output messages (thus a site cannot fail after performing some but not all of these functions). We shall show that this rather strong assumption does not provide much assistance in dealing with the partitions. Also, we allow a site to output messages to any number of its neighbors in a single atomic step.

An *event* is the receipt of one or more messages by a site. For an event e = (p,m) where p is a site and m is a collection of messages, p is said to be the *agent* of e. For a commit protocol P, a **configuration** consists of the states of the sites and all undelivered messages in the system. An event e = (p,m) is *applicable* to a configuration C if p is allowed to take the next step in accordance with the degree of synchrony, if any, among the processors, and m is the collection of messages sent to p by its neighbors but are not delivered yet. (Thus, we allow a site to receive all in-transit messages at once.) A *run* of a commit protocol is a sequence of configurations $C_1, C_2, \cdots$, where C_1 is an initial configuration and the configuration C_{i+1} is obtained from C_i by a single event applicable to C_i for all $i \geqslant 1$. A configuration C is **reachable** if there is an initial configuration C' and a run r such that C is the last configuration and C' is the initial configuration of r. For a subset S of V, the **subconfiguration of C on S** consists of the states of the sites in S and all undelivered messages destined to the sites in S.

For each automaton, we denote by *Com* and *Ab* the final states with decision values 1 and 0 respectively. A configuration C has a decision value 1 (0) if one of the sites is in a state from Com (Ab) in C. Denote the initial states of each automaton by I, all (other) states that lead to both Com and Ab states by N, all states that lead only to Com states by CS, and all states that lead only to Ab states by CA. This does not mean that all automata have the same states; only that the states of any automaton can be classified alike.

The effect of a partition in the worst case, apart from interrupted communication among the groups, is that all in-transit messages in a group destined to sites in the other groups are lost. This is the **lost-message** scenario [23]. Since the partitions are assumed to be detectable, we model a partition as a special event delivering a special message to all the sites. Thus in the lost-message scenario, if a partition event occurs in a configuration C, each site q, including the agent of the non-failure event applicable to C, receives a special message indicating the failure, at the nearest C_i at or after C such that q is the agent of the non-failure event applicable to C_i. In [23], the authors consider another scenario: where all undelivered messages are returned to the senders (when the senders are operational). In general, it may be possible that some of the undelivered messages are returned to the senders while others are not. We call this the **returned-message** scenario. This can be modelled by a special message indicating the partition failure to each site and the delivery of returned messages to the corresponding senders with a notification to that effect. When we consider the subconfigurations on groups formed due to partitions, the messages in a subconfiguration vary depending on whether the scenario is of lost-message or returned-message type accordingly.

A run of a protocol is **partition-prone** if there is a partition event in it. A configuration is **decidable** if a site is in a final state in that configuration. A run is a **deciding run** if one of its configurations is decidable.

A run of a commit protocol is associated with the execution of each transaction to ensure the atomic implementation of the transaction. Thus when a failure occurs, each operational site participating in a transaction may be in one of several possible states of the automaton corresponding to the commit protocol being used (the possible states depend on the progress of the protocol till that instant). Naturally an attempt must be made to move as many sites as possible into their final states so that the resources currently being used by the transaction can be made available to the other waiting transactions. Three approaches are possible here - a) the commit protocol proceeds ignoring the failure information (while still guaranteeing the atomicity), voting-based protocols being examples of this, b) each site, after notified of the failure, attempts to complete the transaction unilaterally (**unilateral termination**), and c) a new protocol which uses the information available in a group as a whole is invoked after the partition occurs. Such new protocols are known as *cooperative termination protocols* or CTPs (these are called "termination protocols" in [23]). Similarly, the protocols used for unilateral termination will be called **unilateral termination protocols** or UTPs. Formally, a CTP can be defined as a *commit protocol* run by **each of the groups** formed due to the partition (that is, each group behaves like a single site) where the initial configuration for a group S is the subconfiguration on S of the configuration C to which the partition event is applied, such that all messages destined to sites outside S are erased in the lost-message scenario and certain undelivered messages are returned to the senders in the returned-message scenario.

Observe that any CTP degenerates into an UTP in the extreme case when each site forms a group by itself. In this sense, any UTP is a special case of a corresponding CTP (but several CTPs may degenerate to the same UTP).

The static approach of ignoring the failure is not very desirable when failures are detectable since it is overly pessimistic. The UTP-based approach is easy to implement since there is no need for inter-site communication after a failure is detected. The CTP-based approach on the other hand requires a degree of coordination among the sites in a group to project the impression of a single entity. In an actual implementation, this may mean that a unique site is *elected* in each group as the *leader* of that group and it runs the CTP while all other sites in that group follow the decision made by the leader (or one of several other possible implementations). Since there can be further partitions (a group may break into several smaller groups) or mergings (several groups merge together into a single group) during the execution of the CTP, the implementation of cooperative termination protocols is complex. The UTP-based approach is clearly insensitive to such network reconfigurations. On the other hand, it is likely that more sites can complete a transaction while using a CTP than otherwise (since a CTP uses more "information" than each individual site). We shall consider both of these approaches in this paper.

3. Properties of commit protocols.

While investigating the commit protocols, we would like to explicitly seperate the two aspects of a protocol - its essence (what it does) and its **means** (how it does that), and study them independently. This approach, as we shall see, not only leads to a better understanding of the problem itself, but also to better protocols than the previously known protocols. We first consider the *essence* part of the commit protocols.

Let A be a commit protocol and let C be a reachable configuration of A. The "history" of the run leading to C from an initial configuration C' can be modelled as an **information graph** as follows (this construction is similar to the "communication graph" of [9] but they consider a system with lock-step synchronization among the processors, tightly synchronous communication, and the topology of a complete graph) : nodes in the graph correspond to the ordered pairs <p,t> where p is a processor and t is the local time; a directed arc is placed from the node <p,t> to the node <q,t'> if a message sent by p at local time t is received by q at its local time t' and for all $t \geq 0$, an arc is placed from <p,t> to <p,t+1> for all p. It is now clear that given a run of A, a corresponding information graph can be easily constructed and this graph shows the flow of information among the sites in the system and the causal relationships among the messages passed. Let us assume for convenience that each site's local time is 0 when it is in the initial state.

Proposition 1 [9]. Let C be a reachable configuration of a commit protocol A such that p is in a Com state in C. Let G be the information graph corresponding to the run leading to C and let <p,t> be the node in G representing the status of p in C. Then, for each q, there is a path in G from <q,0> to <p,t> and all initial values are 1.
Proof follows directly from the definition of a commit protocol since a site cannot reach a Com state unless all initial values are 1 and the only way a site can know this fact is by either directly or indirectly receiving a message from each of the other sites. □

The following counterpart of Proposition 1 for Ab states is straight-forward:

Proposition 2. Let C and G be as above except that p is in an Ab state in C. Then there exists a q with initial value 0 such that there is a path in G from <q,0> to <p,t>. □

Let C be a reachable configuration of a commit protocol A and let G be the information graph of C (we omit to say that it is the graph of the run leading to C, when no confusion can arise). Let <p,t> be the node in G representing the status of p in C. We say **p has full-information in C** if for every q, there is a path in G from <q,0> to <p,t>. By Proposition 1, having full information is a necessary but not sufficient condition to be in a Com state. This condition is neither necessary nor sufficient to be in an Ab state, by Proposition 2. This asymmetry is due to the fact that the conditions for moving into Com states must be mutually exclusive from the conditions for moving into Ab states. For future reference, we use the following notation:

P_q ≡ **" q knows that all initial values are 1".**

Notice that if r,r' are two finite runs such that C,C' are their final configurations respectively and r is a prefix of r', then a processor p with full information in C has full information in C' also. The notion of having full information can be extended to sets of sites in an obvious way as follows: for a set of sites S, let $\{<p,t_p> \mid p \in S\}$ be the nodes in G representing the statuses of the sites of S in C; then S **has full information in** C if for each q, there is a $p \in S$ such that there is a path from <q,0> to $<p,t_p>$. We denote the predicate "*S has full information and all initial values are 1*" by P_S.

Notice that in the above discussion, we have only considered the paths in the information graph between nodes and not the individual message passings. This formalism abstracts the means of the protocol and specifies only the essence. Now we consider the means aspect of a protocol.

3.1 Virtual Communication Vehicles.

All commit protocols considered in the literature have highly structured information flow patterns. We find it easy to express these patterns via the notion of **virtual communication vehicles:** a virtual communication vehicle (VCV) refers to the interaction among the sites directed by the commit protocol and is a directed graph on n nodes where n is the number of sites participating in the commit protocol (some authors refer to these as "communication topologies" [2]). Messages can be passed on both directions of an edge but in only one direction at any time. For instance, the **centralized commit protocols** have the VCV of a star network, edges directed away from the center of the star. By this, we mean the following: there is a prefixed special site (the center of the star) which communicates with every other site while no two other sites communicate among themselves. A centralized commit protocol can be implemented on any connected network: the links in the VCV correspond to paths in the actual network. Similarly, the **decentralized commit protocols** have the complete graph as their VCV (an edge can be directed either way). Several other VCVs are considered in the literature - straight-line [13], ring [13], and tree [16] being the most used among these. But in theory, any graph on n nodes is a possible VCV.

A commit protocol may have several "phases" so that in a phase, one or more sites try to receive messages from all the other sites, and/or one or more sites try to propagate their messages to all the other sites. A seperate VCV may be associated with each site during each phase of such a protocol. Thus, for instance, in a commit protocol with two phases, the first phase (in which a site receives messages from all others) may use a star while the second phase (in which a site propagates a message to all of the others) may use a tree. Similarly, in a decentralized version of this protocol, each site could be using a seperate VCV to receive the messages from all other sites.

3.2 Classification of Commit protocols.

We say a commit protocol is of the **fast-commit type** if every site q moves into a Com state as soon as P_q is true (that is, satisfying P_q and moving into a Com state can be considered to be an *atomic event local to q* by all other sites). A commit protocol that is not of the fast-commit type is said to be of the **delayed-commit type.** Thus, a delayed-commit protocol A has a set S_A of sites associated to it (independent of the run) such that no site moves into a Com state while a site s from S_A does not satisfy P_s. We shall refer to the sites in S_A as **delay sites** of the protocol A since they delay the other sites from moving into Com states. The following examples exemplify this. Let us consider the **basic two-phase commit** protocol [13]. Under this protocol, a site starts in an initial state, sends out its private value while moving into an N state if it is 1 and an Ab state if 0, and if in an N state, moves into a Com or Ab state directly as soon as it gains full information or finds that a private value is 0, respectively. Thus, the basic two-phase commit is of the fast-commit type. Several variations of this protocol exist in the literature where a central site exists and a certain number of other sites act as back-ups to it. In such a protocol, the central site does not move into a Com state immediately after gaining the full information but first informs all of the back-ups and then moves into a Com state. We can easily see that it is a delayed-commit protocol and the back-ups are its delay sites. The **three-phase commit** [21] is another delayed-commit protocol: no site moves into a Com state unless all other sites have gained full information (that is, all sites are delay sites).

A commit protocol can now be described in two parts: a) the **conditions** under which each of the sites moves into different states, and b) the VCV used to achieve the conditions. Extending this insight to the termination protocols, we find it intuitively appealing to view a termination protocol also as a two-stage process: the specification of the predicates to be satisfied by a site/group, and the actual implementation. We shall further explore these ideas in the following sections.

4. Behaviour after the detection of a partition.

We first consider the UTP-based approach in which individual sites **not already in final states** try to move into final states consistently with others. We express the essence of the protocol used by the individual sites as an ordered pair of predicates we call **termination predicates:** *<commit predicate, abort predicate>*. Commit predicate specifies the predicate to be satisfied by a site to move into a Com state while the abort predicate is to be satisfied to move into an Ab state. Clearly commit and abort predicates must be mutually exclusive (since two sites may independantly move into Com and Ab states otherwise).

We do not specify the means of the termination protocol explicitly. Instead we let a site choose its own means; it is free to use whatever information it could gather in trying to assert the predicates. Thus unlike the existing solutions, our approach encourages a site to actively gather (before the failure) any information that it can use to assert a termination predicate.

By Proposition 1, a necessary condition for a site to move into a Com state is that it has full information and all initial values are 1 (that is, the site q satisfies P_q) and it can be easily checked that this is valid **even after a failure.** Thus P_q is the **weakest possible commit predicate.** We note that by contrast, the condition of Proposition 2 is **NOT** a necessary condition to move into an Ab state **after a failure.** The following is the **weakest possible abort predicate:** *no other site has already committed.* Clearly these two predicates are NOT mutually exclusive but are the two extremes. Given a commit (abort) predicate, its negation is the weakest possible abort (commit) predicate it can be paired with. We summarize this below:

Decision type	Strongest predicate	Weakest predicate
Commit	some site is in a Com state	P_q for some q
Abort	NOT P_q for any q	no site can be in a Com state

The following termination predicates form the two extremes, for a given commit protocol: <strongest commit predicate, weakest abort predicate> and <weakest commit predicate, strongest abort predicate>. Any valid pair of termination predicates must fall between these two. We call the former of these two **the conservative protocol** and the latter **the progressive protocol.** Notice that the progressive protocol tries to commit whenever possible while the conservative protocol tries to abort. We first show that the nature of a commit protocol may make the choice of the commit and abort predicates trivial.

Proposition 3. Let A be a fast-commit protocol. Then, irrespective of the VCV, there is exactly one pair of possible termination predicates.
Proof. By the hypothesis, if P_q is true then q can be considered to be in a Com state. But this implies that the strongest and weakest commit predicates coincide for A. Hence there is a unique possible commit predicate and a corresponding unique abort predicate.□

This result has an important implication: fast-commit protocols cannot handle network partitions. To show this, we observe that under the unique TP possible for such a protocol, a site not already in a final state can move into a final state after a partition only if *it is in an initial state.* But this is only a trivial termination since for all practical purposes, a site in an initial state can move into an Ab state unilaterally. Consider the basic two-phase commit for concreteness. Notice that it is likely that most sites are in an N state at a randomly chosen instant during the execution of this commit protocol, independent of the assumptions made on the behaviour of the system (since the sites have a private value of 1 for most of the transactions in realistic environements). This implies that the sites simply wait until a total recovery if the commit protocol used is the basic two-phase protocol. Consequently only the delayed-commit protocols can be of any practical assistance in handling partitions. Consequently we shall only deal with the delayed-commit protocols in the rest of the paper.

4.1 The Means of Termination protocols.

Delayed-commit protocols can have a wide range of possible termination predicates, two extremes being the progressive protocol and the conservative protocol. An actual implementation of the protocols based on termination predicates (that is, the *means of the protocol*) can vary depending on the commit protocol and its VCV. As an example, let us consider the centralized three-phase commit (with the star as its VCV during all phases) which works as follows: each site (including the central site) with an initial value of 1 moves into an N state and sends its initial value to the central site (a site unilaterally aborts (moves into an Ab state) before sending the message if its initial value is 0); the central site, after receiving all of the initial values (when they are all 1), or after receiving a 0, moves into a CS state or an Ab state, and sends a "prepare to commit" or "abort" message to all sites, respectively; a site receiving a "prepare to commit" message acknowledges it and moves into a CS state; a site receiving an "abort" message moves into an Ab state; the central site, after receiving all acks, sends a "commit" message to all sites and moves into a Com state; a site receiving a "commit" message moves into a Com state. Finally, assume that the progressive protocol is used after a partition.

Let us consider the lost-message scenario first. Notice that, under the three-phase commit protocol, a site satisfies P_q if and only if it is in a CS state. Hence, it is trivial for a site to assert the commit predicate. Observe that the central site is the first to move into a CS state. Thus, the central site satisfies the abort predicate if it is in an initial or N state (that is, it is able to exploit the VCV). Any other site can assert that it satisfies the abort predicate only if it is in an initial state. A site in an N state, even if the abort predicate is satisfied, cannot assert it unilaterally and hence must wait. (Thus, under the progressive protocol, all sites except the central site wait when they are in N states.)

In general, a non-delay site of a delayed-commit protocol (if exists) can be easily checked to have a unique possible UTP. It satisfies the commit predicate when in a

CS state and satisfies the abort predicate when in an I state. The following table summarizes the extremal conditions to be satisfied by a delay site other than the central site for a delayed-commit protocol:

Condition type	Local state of q
Strongest commit	from Com
Weakest commit	from CS
Strongest abort	from I
Weakest abort	from N

Consider the returned-message scenario now. A site receiving some returned messages can use the knowledge it gains through those messages in asserting the termination predicates. Thus, for instance, consider the centralized three-phase commit. If a site p in an N state is returned its message to the central site, then it can safely abort since the weakest abort predicate is now satisfied (no site can know that all initial values are 1). Thus, in a sense, a site receiving certain returned messages may be able to "roll back" its state to a previous state (such as p effectively considering itself to be in an I state in the example above). Thus the chances that the abort predicate is satisfied are improved in the returned-message scenario. The entries in the above table remain valid in this case also, with the interpretation that the "rolled back" state of the site is considered instead of the original state.

4.2 Cooperative Termination Protocols.

Informally, in comparison to the UTP-based approach, all sites in a group here pool together the information they have and then try to move into a final state. Thus, a group here plays the role a single site has played in the above approach. Consequently many of the conclusions drawn there hold for the CTP based approach, too. The counterpart of the Proposition 1 is given below. Proof is obvious and is omitted.

Proposition 4. Let A be a commit protocol, f a CTP of A and C a subconfiguration on S. Then f moves S into a Com state only if S has full information in C and all initial values are 1 (that is, P_S is true). □

Now it is easy to express CTPs also by termination predicates. More than in the case of UTPs, this approach here can be of considerable use because it leaves all the details of asserting a prescribed condition to the group. Thus under this approach, a group is free to use all the information it can accumulate, including the VCV of the commit protocol and any messages returned to the sites in the group. Also, it is free to use any VCV for the implementation of the CTP.

The weakest commit predicate for groups is given by Proposition 4. Similarly, the strongest commit predicate is, "some site is already in a Com state". The weakest abort predicate is the "group version" of the one given in the previous section: C on S satisfies the weakest abort predicate if the subconfiguration on V\S does not satisfy the strongest commit predicate. The strongest abort predicate similarly is the negation of the weakest commit predicate. Let us consider the lost-message scenario first. We extend the notion of states to groups as follows: a group S is in Com, Ab, CA, or I state if one of the sites in S is in a state from the respective class; S is in CS state if P_S is true; S is in AP ("abort possible") state if there is a delay site q in S such that q is in an N state; and finally S is in N state if it is in none of the above states. Now we can easily establish the correspondence between the extremal commit/abort predicates and the states of a group, given in the following table:

Condition type	Local state of the group
Strongest commit	Com
Weakest commit	CS
Strongest abort	I or CA or Ab
Weakest abort	AP

In [1], Barbara and Garcia describe CTPs by **quorum agreements:** a quorum agreement is an ordered pair <*commit quorum*, *abort quorum*>. A quorum is a collection of subsets of V; in a quorum agreement, each element of the commit quorum intersects each element of the abort quorum. Since their primary concern is the mutual exclusion, they do not specify how the state information is used: their only reference to states is "(of course, if any of the sites in the group is already in commit or abort state, the transaction is committed or aborted)". If we take this on the face value then this approach misses even trivial cases such as a site being in an initial state when any TP can move it into an Ab state. Thus, we interpret the quorum agreements as follows: a group satisfies the commit requirement if it is a superset of one of the sets in the commit quorum and is in CS state; similarly a superset of a set in the abort quorum satisfies the abort requirement if a) it contains a delay site and is in one of I, N, or CA states or b) it has no delay sites but is in I state. Chin and Ramarao in [3] define the termination protocols as mappings f from the power set of V X Q to {0,1,∞} where Q is the state set of an automaton and ∞ refers to the decision to wait, subject to the following constraints: a) f(X) = 0 if (i,s)∈X where s∈Ab⋃I⋃CA for some s, b) f(X) = 1 if (i,s)∈X where s∈Com for some s, and c) {f(X), f(X')} ≠ {0, 1} for X,X' such that X,X' can be concurrent. The notion of concurrency is defined such that in the present terminology, X⋃X' represents a reachable configuration. For any subset X of V X Q let sites(X) = {i | (i,s)∈X for some s} and states(X) = {s | (i,s)∈X for some i}. We now show that our model of TPs is more general than both of the above mentioned models, which we call respectively the quorum model and the mappings model.

Proposition 5. Let A be a (delayed-)commit protocol such that any configuration in which some sites are in N states and the rest are in CS states is reachable. The sets of TPs of A obtained by the mappings model and the quorum model are identical.

Proof. Let q = $<q_c, q_a>$ be a quorum agreement. We define a mapping from the power set of V X Q to {0,1,∞} as follows: f(X) = 0 if either (i,s)∈X where s∈Ab⋃I or sites(X)⊃S for some S∈q_a AND (i,s)∈X where s∈N for some delay site i; f(X) = 1 if (i,s)∈X where s∈Com or sites(X)⊃S for some S∈q_c AND (i,s)∈X where s∈CS for some i∈sites(X); and f(X) = ∞ otherwise. Correctness of this mapping follows from the correctness of the quorum agreement. Conversely, let f be a TP given by a mapping. Let S_C = {sites(X)| there is no (i,s)∈X such that s∈*Com*} and let S_A = {sites(X)| there is no (i,s)∈X such that s∈I⋃Ab⋃CA}. Remove all sets S from S_C such that S⊃T∈S_C. Process S_A the same way. We claim that $<S_C, S_A>$ is a quorum agreement. For this we must show that each set in S_C intersects each set in S_A. For S∈S_C and T∈S_A, let X,X' respectively be the sets from which they are derived. X,X' must be concurrent for all commit protocols A such that any configuration in which some sites are in N states and some sites are in CS states is reachable. But f(X) = 1 and f(X') = 0 by construction of S_C and S_A. Hence S and T must intersect. □

Even though it is possible to show that the mappings model generates more TPs than the quorum model for commit protocols A not satisfying the hypothesis of the above result, a slight modification of the interpretation of the quorum agreements for such protocols makes them equivalent. We shall omit these details here since they require more complex terminology while not shedding any new light.

Proposition 6. Let A be a delayed-commit protocol. Let M and T respectively denote the classes of CTPs of A described by the mappings and the termination predicates. Then, $T \supset M$.
Proof. Let f be a mapping TP. It can be easily shown that $f(X) \neq 1$ if there is no $(i,s) \in X$ such that $s \in Com \bigcup CS$. Similarly it can be shown that $f(X) = 0$ if there is no $(i,s) \in X$ such that $s \in I \bigcup CA \bigcup Ab \bigcup N$. Thus if we interpret each X as a subconfiguration on sites(X), $f(X) = 0$ only if sites(X) satisfies the weakest abort predicate and $f(X) = 1$ only if sites(X) satisfies the weakest commit predicate. Now, given f, we construct the commit predicate by simply listing all subconfigurations that correspond to X such that $f(X) = 1$. Abort predicate can also be described similarly. (Clearly this is a very inefficient transformation but we are only interested here in its existence.) We now prove that this relationship is proper. Consider the decentralized version of A. Let g be the progressive CTP of A. Consider a subconfiguration X on a set S such that in the information graph G of X, there are paths from $<p,0>$ to $<r,t_q>$ for all $p,r \in S$ (so that all sites in S are in N states) and there is a site $p \in S$ such that there is no path from $<p,0>$ to $<s,t>$ for any $s \in V \backslash S$, for all t. (Informally, the value of p is not known to any site outside S.) Clearly X is mapped to 0 under g. No mapping TP can distinguish between X and a subconfiguration in which there is a path from $<p,0>$ to $<r,t_r>$ for some t_r for all $r \in V \backslash S$ and hence must map the subset of $V \times Q$ corresponding to X to ∞. □

The Proposition 3 which says that fast-commit protocols have a unique termination protocol remains valid for CTPs also:

Proposition 7. Let A be a fast-commit protocol. Then there is exactly one CTP for A, given by <weakest commit predicate, strongest abort predicate>. □

4.2.1 Implementation of CTPs - An example.

Let us consider the progressive CTP of the three-phase commit protocol. Let S be a group formed due to a partition. Assume that S chooses to implement the CTP via a star VCV. Thus a site in S is elected to be the *leader* and all other sites report to the leader. The central site relays any decisions it makes to all of the other sites. (A spanning tree may be the actual structure used by the group to achieve this effect.) If S can determine that its state is among I, CS, CA, Ab, or Com, then it can move into a final state immediately. For this purpose, each site in S forwards the following information to the leader: <id, state, ids of the sites known to have initial values of 1 if the state is from N>. The leader can now determine if S is in one of the above-mentioned states. It directs all sites in S to move into the appropriate final state if possible. Otherwise, it sends the following information to the sites in S and waits for a reconfiguration : <list of sites in S, state of S, list of sites known to have initial values of 1>.

When a site detects a merging (that is, it can communicate with a site from a different group), its behaviour depends on whether or not it has received a message from the leader of its group regarding the completion of the transaction. If it has received a message directing it to move into a final state, then it simply passes on this decision to the new site also. If it knows that the leader has not been able to decide to move into a final state, then it first attempts to communicate with the leader of its group. If it can, then it asks the other site to participate into this group, thus starting a new cycle. (If it cannot communicate with the leader but the other site can communicate with its leader then it participates in the other group.) If none of them can communicate with their leaders, then they start the election process and form a new group. The algorithm used by the leader each time is exactly the same - it tries to find if the group is in one of the favorable states from which the group can move into a final state. It is not difficult to show that no two concurrently existing groups can move into conflicting final states while using this procedure.

As in the case of UTPs, the returned-message scenario can only make a group to "roll back" its state. A site communicates with the leader whenever it receives a returned message and the leader checks if this can cause a "roll back" of the group's state. If the leader has already decided to move into a final state, then such "roll backs" do not effect it. Otherwise the leader may be able to move into an abort state due to certain returned messages. The details in this case are very similar to those in the case of the UTPs and are omitted.

4.3 A Mixed Approach.

We have considered above two classes of termination protocols in the presence of a partition - UTPs and CTPs. Among these, the UTPs impose no communication overhead after the detection of a partition while the CTPs require communication among all sites of a group. Depending on the VCV used for the implementation of a CTP, the number of messages generated can vary from $O(|S|)$ to $O(|S|^2)$ for a group S. On the other hand, a CTP may be able to make a group move into a final state when no UTP can move all sites of that group. Thus it seems natural that each site first attempts a UTP and then joins a CTP only if it cannot move into a final state by itself. The UTP and the CTP must be **compatible** in such an approach since otherwise the mixed protocol may not be a termination protocol: the commit (abort) predicate for a site q and the abort (commit) predicate for $V \backslash \{q\}$ must be mutually exclusive. This is because it is possible that at some instant (of the real time) a site is using the UTP while some groups are using the CTP.

Using the mixed approach, a site behaves as follows after a partition is detected: first it attempts to complete an incomplete transaction using the UTP. If it can move into a final state, then it does so and informs all of its neighbors about it. If it cannot move into a final state, then it initiates the CTP by informing its neighbors that it is willing to form a group. A site receiving a message indicating the movement into a final state itself moves into a final state (from the same class) and informs its neighbors. If all messages it receives are about forming a group then it also participates in the group formation and the subsequent use of the CTP.

4.4 Performance Analysis.

Cooper in [5] analyzes the performance of certain commit protocols in the presence of partitions. In his model, a site waits if it is in its "window of uncertainty" at the partition time. For the centralized versions of basic two-phase commit, back-up based two-phase commit, and the two-phase commit with a straight-line VCV, the window of uncertainty coincides with being in an N state and it coincides with being in a CS state for the centralized three-phase commit. The expected number of waiting sites for each of these commit protocols is computed there and the analysis can be extended in a straight-forward manner to the other protocols also. As an example, we shall analyze the progressive protocol for the centralized three-phase commit. The terminology, assumptions, and the approach are from [5]. The interested reader is referred to the original source for more details.

Assumptions.

The probabilistic model used to analyze the protocols makes the following assumptions about the network and the transaction processing times [5]: "a) the time required to send a fixed-length message from one site to any number of others is δ, b) if m sites attempt to send a message to the same site simultaneously then the time required for all messages to arrive is mδ, c) the processing time required by a site i is described by a random variable T_i (T_i is the time from the receipt of the transaction by i to the time i makes a local decision, and T_i is taken into an absolute time for convenience), and d) T_i are independent and are identically distributed with cumulative distribution function F(t)." As remarked in [5], the results are valid for networks for which δ is small compared to the local processing times (the communication medium is fast and the processing power is not very high - local area networks of micros for instance). The broadcast facility assumed above is not essential and the results can be extended to point-to-point communication. The following result is very useful to us, as it was to Cooper.

Lemma 1 [5]. Let X, Y be independent random variables with cumulative distribution functions $F_X(z)$ and $F_Y(z)$ respectively. Let P(z,ρ) be the probability of the event X $\leqslant$ z $\leqslant$ max(X,Y)+ρ. Then P(z,ρ) = $F_X(z) - F_X(z-\rho)\, F_Y(z-\rho)$. □

The centralized three-phase commit.

We observe that a site waits under the progressive protocol only if it is in an N state at the time of partitioning. Thus, the partition must occur after the site sends its value but before it receives the "prepare to commit" message. In our model, i sends its value at T_i and the central site receives the last value and sends the "prepare to commit" message at time max(T_j)+δ which is received by all other sites at max(T_j)+2δ. Hence, if a partition occurs at τ and i is in an N state at that time, τ must satisfy the inequality

$$T_i \leqslant \tau < \max(T_j) + 2\delta.$$

It is easy to compute the probability of this event by taking X = T_i, Y = max{T_j | j≠i} so that max(T_k) = max(X,Y) and applying Lemma 1. Clearly $F_Y(z)$ = $F(x)^{n-1}$ and $F_X(z)$ = F(x). Thus, the required probability is $F(\tau)-F(\tau-2\delta)^n$. Since the central site need not wait even if it is in an N state, the expected number of waiting sites under the progressive protocol is given by $(n-1)[F(\tau)-F(\tau-2\delta)^n]$. By contrast, the probability that the site i waits while using the conservative protocol is shown to be Expwait = $F(\tau-2\delta)^n - F(\tau-(n+2)\delta))^n$ in [5]. Thus the expected number of waiting sites under the conservative protocol is (n-1)*Expwait.

This implies that for large n, the conservative protocol performs better than the progressive protocol. This can primarily be attributed to the fact that a site is likely to remain in an N state longer than in a CS state. Thus there is a trade-off between the number of waiting sites and the number of committing transactions. Since the abort of a transaction may be far more expensive than a commit (the aborted transaction may be resubmitted later for execution), a weight may have to be assigned to each aborted transaction and correspondingly a desirable UTP is required to minimize the weighted expected number of waiting sites.

Extension of this analysis to CTPs is also not hard. Cooper analyzes the group version of the SDD-1 protocol in [5]. The same principles apply to the other protocols and we shall not give the details here.

5. Related Work.

Gray has informally shown the impossibility of implementing atomic actions when arbitrary partitions occur in [13]. The same result is formally proved by Skeen and Stonebraker in [23]. They model the algorithm run by each site participating in a commit protocol by an indeterministic finite state automaton. They do not study the properties of the termination protocols in the presence of partitions. Skeen describes a commit protocol based on quorums in [22]. This work does not assume the detectability of partitions. Davidson [6] has explored a different approach known as the *optimistic approach* where the transactions are allowed to proceed in all groups and any inconsistencies arising from this are detected and resolved after a reconfiguration. Garcia-Molina et al [12] extend these ideas to cases where it may not be possible to resolve inconsistencies. Sarin et al [20] present a solution based on compensating transactions to resolve inconsistencies resulting from unrestricted transaction processing. Parker et al present a simple technique for the detection of inconsistencies when a transaction accesses a single file [17]. This technique is recently generalized to arbitrary transactions by Ramarao [18]. Chin and Ramarao investigate the properties of the termination protocols and develop a theory for the design of termination protocols optimizing the expected number of groups/sites that can continue processing, for the case of a complete graph topology in [3]. They use an information-based model in [4] to give an alternate proof for the non-existence of atomic commit protocols. The model considered in the present paper is different from that information-based model. Davidson et al survey most of the work on handling partitioned databases in [7]. Barbara and Garcia-Molina address the more general problem of mutual exclusion in [1] and report a class of termination protocols (quorum agreements) which properly includes the voting-based protocols.

Our work differs from all of the existing literature dealing with partition failures in the following ways: a) our formalism for termination protocols is more general than the existing ones and is intuitively more appealing, b) we explore different kinds of termination protocols - Unilateral, Cooperative, and Mixed, c) we study the effects of the communication patterns of the commit protocols and of returned messages, on the termination protocols and d) we have produced a CTP strictly better than all known termination protocols (in the proof of Proposition 6).

There has been an extensive amount of work reported on fault-tolerant concurrency control mechanisms and replica control algorithms. We shall not consider this aspect here. The interested reader is referred to the recent works by El Abbadi and Toueg [10] for concurrency control of databases and by Herlihy [14] for concurrency control of abstract data types.

6. Conclusions.

The problem of implementing atomic transactions in the presence of network partitions is studied in this paper. The approach taken is to seperate the essence of a protocol from its means. It is shown that this approach not only leads to a deeper understanding of the principles of commit protocols, but also to better protocols.

An important assumption made in this paper is that the failures are detectable. The approach based on termination protocols becomes inapplicable when this assumption is invalid.

References

[1] Barbara, D. and Garcia-Molina, H., "Mutual exclusion in partitioned distributed systems," Distributed Computing, 1986, pp.

[2] Bernstein, P.A. et al, Concurrency control and recovery in database systems, Addison-Wesley, 1987.

[3] Chin, F.Y. and Ramarao, K.V.S., "Optimal termination protocols for network partitioning," SIAM J. on Computing, Vol. 15, No. 1, Feb. 1986, pp. 131-144.

[4] Chin, F.Y. and Ramarao,K.V.S., "An information-based model for failure-handling in distributed database systems," IEEE Tr. on Software Engg., Vol. SE-13, No. 4, April 1987, pp. 420-431.

[5] Cooper, E.C., "Analysis of distributed commit protocols," Proc. ACM SIGMOD Conf. on Management of Data, 1982, pp. 175-183.

[6] Davidson, S.B., "Optimism and consistency in partitioned distributed database systems," ACM TODS, Vol. 9, No. 3, 1984, pp. 456-481.

[7] Davidson et al, "Consistency in partitioned networks," ACM Computing Surveys, Vol. 17, No. 3, 1985, pp. 341-370.

[8] Dolev, D. et al, "On the minimal synchronism needed for distributed consensus," JACM, Vol. 34, No. 1, 1987, pp. 77-97.

[9] Dwork, C. and Skeen, D. "The inherent cost of non-blocking commitment," Proc. 2nd ACM PODC, 1983, pp. 1-11.

[10] El Abbadi, A. and Toueg, S., "Availability in partitioned replicated database," Proc. 5th ACM PODS, 1986, pp. 240-251.

[11] Fisher, M.J. et al, "Impossibility of distributed consensus with one faulty process," JACM, Vol. 32, No. 3, pp. 374-382.

[12] Garcia-Molina, H. et al, "Datapatch: integrating inconsistent copies of a database after a partition," Proc. 3rd IEEE Symp. on Reliability in Distributed Software and Database Systems, 1983, pp. 38-48.

[13] Gray, J.N., Notes on Database Operating Systems, in Operating Systems: An Advanced Course, Lecture Notes in Computer Science 60, Springer-Verlag, 1978, pp. 393-481.

[14] Herlihy, M., "A quorum-consensus replication method for abstract data types," ACM TOCS, Vol. 4, No. 1, 1986, pp. 32-53.

[15] Ma, A.V. and Belford, G.G., "A failure and recovery detection protocol for optimistic partitioned operation in distributed database systems," Proc. 6th IEEE Int'l Conf. on Distributed Computing Systems, 1986, pp. 540-547.

[16] Mohan, C. and Lindsay, B., "Efficient commit protocols for the tree of processors model of distributed transactions," Proc. 2nd ACM PODC, 1983, pp. 76-88.

[17] Parker, D.S. et al, "Detection of mutual inconsistency in distributed systems," IEEE Tr. on Software Engg., Vol. SE-9, No. 3, 1983, pp. 240-247.

[18] Ramarao, K.V.S., "Detection of mutual inconsistency in distributed database systems," To appear in J. of Parallel and Distributed Computing. Also a preliminary version has appeared in Proc. 3rd IEEE Int'l Conf. on Data Engineering, 1987, pp. 405-411.

[19] Ramarao, K.V.S., "Transaction atomicity in the presence of network partitions," Proc. 4th IEEE Int'l Conf. on Data Engineering, 1988, pp. 512-519.

[20] Sarin, S.K. et al, "System architecture for partition-tolerant distributed databases," IEEE Tr. on Computers, Vol. C-34, No. 12, 1985, pp. 1158-1163.

[21] Skeen, D., "Nonblocking commit protocols," Proc. ACM SIGMOD Conf. on Management of Data, 1982, pp. 133-147.

[22] Skeen, D., "A quorum-based commit protocol," Proc. 6th Berkeley workshop on distributed data management and computer networks, 1982, pp. 69-80.

[23] Skeen, D. and Stonebraker, M., "A formal model of crash recovery in a distributed system," IEEE Tr. on Software Engg., Vol. SE-9, No. 3, 1983.