DEALING WITH INCOMPATIBLE PRESUMPTIONS OF COMMIT PROTOCOLS IN MULTIDATABASE SYSTEMS

Yousef J. Al-Houmaily
Dept. of Electrical Engineering
University of Pittsburgh
Pittsburgh, PA 15261
yjast1+@pitt.edu

Panos K. Chrysanthis
Dept. of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260
panos@cs.pitt.edu

Keywords: Atomic Commit Protocols, Two-Phase Commit, Distributed Transaction Processing, Multidatabase Systems.

ABSTRACT

A *multidatabase system* (MDBS) is a software system that is built on top of multiple pre-existing and heterogeneous database systems to facilitate their interoperation. This paper discusses the issue of compatibility among *atomic commit protocols* (ACPs) in a MDBS environment. Specifically, it shows that supporting a visible *prepare to commit* state is not enough for a successful integration of ACPs in an operational fashion because the outcome of some transactions might have to be remembered forever. Therefore, we define an *operational* correctness criterion that allows terminated transactions to be forgotten and propose *Presumed Any*, a *Two-Phase Commit* protocol variant that successfully integrates *Presumed Nothing*, *Presumed Abort* and *Presumed Commit* in a MDBS. We also show how the behavior of a local database system that employs presumed abort can be made to look as if it employs presumed commit and vice versa, allowing the MDBS to dynamically adapt to the most appropriate two-phase commit variant at any given time.

INTRODUCTION

A *multidatabase system* (MDBS) is a software system that facilitates interoperability across multiple pre-existing and heterogeneous database systems (Figure 1). A MDBS allows each database system to continue to operate in an independent fashion and (ideally) does not require any changes to existing databases, applications, and the local database management systems (LDBMSs).

Two types of transactions execute in a MDBS:

- *local transactions* that access data located at only a single database under the control of the LDBMS and whose existence the MDBS is not aware of.
- *global transactions* that access data located at multiple databases under the control of the *global transaction manager* (GTM) of the MDBS.

A global transaction is decomposed by the GTM into several *subtransactions*, each of which executes as a local transaction at some site. An *agent* which resides above each LDBMS, is responsible for the different aspects of the execution of subtransactions at its site and in particular, of the termination protocol needed to commit the subtransactions [1].

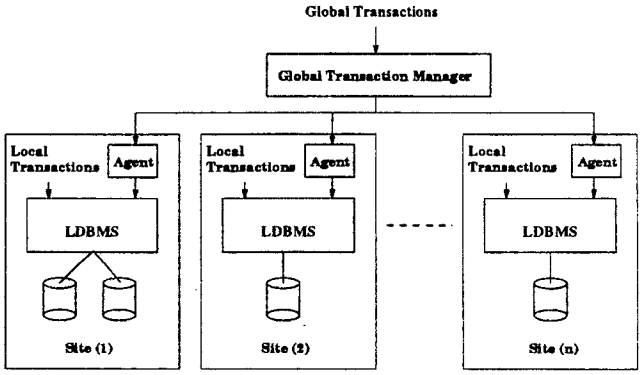


Figure 1: The MDBS model.

Termination protocols in MDBSs that ensure the atomicity of global transactions can be classified based on whether or not they assume a visible *prepare to commit* state. That is, whether each LDBMS externalizes its commit protocol by making it public to the outside world through its interface. *Atomic commit protocols* (ACPs) that do not assume externalized *prepare to commit* state are typically designed to emulate the *two-phase commit* (2PC) protocol [5, 6, 10]. In emulated 2PC protocols, the agents act as the participants in the execution of 2PC whereas the *prepare to commit* state is emulated through restrictions on either the data access pattern or the initiation of global and local transactions [2, 7, 12, 14, 3]. On the other hand, ACPs that assume externalized *prepare to commit* state are usually designed to resolve the incompatibilities between the ACPs used by the different LDBMSs [13].

The incompatibility of ACPs means that the semantics of the coordination messages and the actions that are taken by a LDBMS that employs one ACP might be completely different than their counterparts in another ACP. Interoperat-

This material is based upon work partially supported by the National Science Foundation under grants IRI-9210588 and IRI-95020091.

ing different ACPs is not a trivial task as it was previously believed [1, 13], i.e., that once a LDBMS supports a visible *prepare to commit* state, it can be integrated in a MDBS regardless of the ACP variants that are employed by the other LDBMSs [11, 4].

In this paper, we examine the compatibility of the basic 2PC and its two most common variants, namely *presumed abort* (PrA) and *presumed commit* (PrC) [9], and propose a new 2PC protocol, called *presumed any* (PrAny) that effectively integrates these three 2PC protocols and allows them to interoperate in a MDBS environment despite their conflicting presumptions about the outcome of transactions. Then, we show how an agent can alter the behavior of a LDBMS that employs PrA to look as if it uses PrC and vice versa, allowing the MDBS to dynamically adapt the most appropriate 2PC variant in a particular situation, hence achieving the best performance during normal transaction processing.

TWO-PHASE COMMIT VARIANTS

The basic *two-phase commit* protocol (2PC) consists of a *voting phase* during which the coordinator of a distributed transaction requests all the sites participating in the transaction's execution to *prepare to commit*, and of a *decision phase* during which the coordinator either decides to commit the transaction if *all* the participants are *prepared to commit* (voted `Yes`), or to abort if any participant has *decided to abort* (voted `No`). If a participant has voted `Yes`, it can neither commit nor abort the transaction until it receives the final decision. When a participant receives the final decision, it complies and acknowledges the decision. The coordinator completes the protocol and discards any information in its *protocol table* in main memory regarding the transaction when it receives acknowledgments from all the participants.

The resilience of 2PC to system and communication failures is achieved by recording the progress of the protocol in the logs of the coordinator and the participants. The coordinator is required to force-write a `decision` record prior to sending out the final decision. Since a *force-write* ensures that a log record is written into a stable storage that sustains system failures, the final decision is not lost in the case of a coordinator failure. Similarly, each participant force-writes a `prepared` record before sending its vote and a `decision` record before acknowledging a final decision. When the coordinator completes the protocol, it writes a non-forced `end` record indicating that the log records pertaining to the transaction can be garbage collected when necessary.

The basic 2PC protocol is also referred to as the *presumed nothing* 2PC (PrN) because it treats all transactions uniformly, whether they are to be committed or aborted, requiring information to be explicitly exchanged and logged at all times. However, in case of a coordinator's failure, there is a hidden presumption in PrN by which the coordinator considers all active transactions at the time of the failure as aborted ones. This presumption allows a coordinator not to force-write any log records prior to the decision phase. If a participant inquires the coordinator about an active transaction after the coordinator has failed and recovered, the coordinator, not remembering the transaction, will direct the participant to abort it (by presumption).

The *presumed abort 2PC* protocol (PrA) [8, 9] operates in a manner similar to PrN for the commit case but makes the abort presumption more explicit. When a coordinator decides to abort a transaction, it does not force-write the abort decision in its log. Instead, the coordinator sends an abort message to all the participants and discards all information about the transaction from its protocol table. Thus, the coordinator of an aborted transaction does not have to write any log records or wait for acknowledgments. Since the participants do not have to acknowledge abort decisions, they are also not required to force-write such decisions. In case of a coordinator failure, if a participant inquires about a transaction that has been aborted, the coordinator will not remember the transaction and will direct the participant to abort it (by presumption).

As opposed to PrA, the *presumed commit 2PC* protocol (PrC) [9] is designed to reduce the cost of committing transactions. Instead of recording commit decisions in the stable log and interpreting missing information about transactions as abort decisions (which is the case in PrA), in PrC, coordinators record abort decisions and interpret missing information about transactions as commit decisions. However, in PrC, a coordinator has to force write an `initiation` record for each transaction before sending *prepare to commit* messages to the participants. This record contains the identities of the participants and ensures that missing information about a transaction will not be mis-interpreted as a commit case after a coordinator failure.

To commit a transaction, the coordinator force writes a `commit` record to logically eliminate the `initiation` record of the transaction and then sends out the commit decision. The coordinator also discards all information pertaining to the transaction from its protocol table. When a participant receives the decision, it writes a non-forced `commit` record and commits the transaction without having to acknowledge the decision. If a participant fails before the `commit` record is in its stable log, the participant will inquire the coordinator about its final decision during its recovery. The coordinator, not remembering the transaction, will direct the participant to commit it (by presumption). Similarly, if the coordinator of a committed transaction fails before it submits its decision to the participants, the coordinator, after its recovery, will direct any participant that inquires about the transaction to commit the transaction if it does not have any recollection about the transaction.

To abort a transaction, on the other hand, the coordinator does not have to write the abort decision in its log. Instead, the coordinator, sends out the abort decision and waits for the acknowledgments. When a participant receives the decision, it force writes an `abort` record and then acknowledges it. Once the coordinator receives acknowledgments from all the participants, it discards all information pertaining to the transaction from its protocol table and writes a non-forced `end` record.

COMPATIBILITY OF 2PC VARIANTS

Let us examine in this section the compatibility of PrN, PrA and PrC discussed above.

Consider the case where a transaction has executed at two participants. Further, assume that the coordinator and one of the participants employ PrC while the other participant employs PrA. The voting phase is the same in both variants. The only difference between the two variants, as far as the coordination messages are concerned, occurs in the decision

phase. In the event that the coordinator of the transaction makes a commit final decision, the PrA participant will acknowledge the commit which is not recognized by the coordinator and is ignored. With respect to the logging activities at the coordinator, it will be able to forget about the transaction and discard all information pertaining to the transaction from its protocol table once it makes the commit final decision and can garbage collect the transaction's log records when necessary. Since the coordinator employs PrC, it will always be able to respond to the inquiries of the participants in case of a failure with a commit final decision (using the PrC presumption).

Now, consider another transaction that has finished its execution at the same two participants and the coordinator has decided to abort the transaction. In this case, the coordinator can never garbage collect the records pertaining to the transaction from its stable log nor can it discard the information from its protocol table. This is because the PrA participant never acknowledges an abort decision. If, on the other hand, the coordinator attempts to forget the outcome of the transaction once it receives the acknowledgment from the PrC participant, the atomicity of the transaction might be violated. If the PrA participant fails after it receives the final outcome but before writing it into the stable log, the participant will inquire about the outcome of the transaction as part of its recovery. If the coordinator has already received the acknowledgment from the PrC participant and forgotten about the transaction, the coordinator will wrongly respond with a commit final decision (using the PrC presumption) which clearly violates the atomicity of the transaction. Therefore, the coordinator has to remember the abort decision forever in order to prohibit the occurrence of such a situation.

A similar situation occurs if the coordinator employs PrN or PrA. In the case that the coordinator employs PrN and some participants employ PrC while the others employ PrA, the coordinator needs to remember the outcome of both committed as well as aborted transactions forever. This is because commit decisions will not be acknowledged by PrC participants while abort decisions will not be acknowledged by PrA participants. If the coordinator employs PrA, the coordinator has to remember all committed transactions forever because PrC participants never acknowledge commit decisions.

The above scenarios show that it is possible from a consistency point of view to integrate PrN, PrA and PrC in a MDBS by ignoring any unnecessary messages. However, this result is impractical because the coordinator has to remember the committed or aborted transactions *forever*. Thus, guaranteeing the consistency of final decisions regarding the outcome of transactions is not enough for an operationally correct integration of ACPs.

Definition 1: The integration of different ACPs is *operationally correct* if and only if

1. The coordinator and all the participants reach consistent decisions regarding the outcome of transactions and regardless of failures.
2. The coordinator can, eventually, discard all the information pertaining to terminated transactions from its protocol table and garbage collect its log.
3. All participants can, eventually, forget about transactions and garbage collect their logs.

PRESUMED ANY (PrAny)

To maintain operational correctness in a MDBS, a coordinator should be able to, eventually, reach a *safe* state in which it can forget about the outcome of transactions without violating the consistency of its decisions.

Definition 2: The coordinator is in a *safe* state if and only if it can reply to the inquiry messages of the participants about the status of a transaction based on a single presumption which is consistent with the transaction's final outcome.

That is, the safety criterion implies that some information including the outcome of transactions has to be remembered as long as more than one presumption is possible. In PrAny, a coordinator knows the protocols used by the different LDBMSs and uses this knowledge to decide when to discard the information about a transaction.

According to the behavior of PrN, PrA and PrC, the coordinator expects those participants that employ PrN and PrA to acknowledge commit final decisions but not those participants that employ PrC (Figure 2(a)). The coordinator can forget about the outcome of a committed transaction once the PrN and PrA participants acknowledge the commit decision knowing that only a participant that employs PrC might inquire about the decision in the future. If a PrC participant inquires about a (commit) final decision after the coordinator has forgotten the transaction, the coordinator, knowing that the participant uses PrC, will direct the participant to commit the transaction (by the presumption of PrC).

Similarly, if a coordinator makes an abort final decision, it expects only those participants that employ PrN and PrC to acknowledge the decision but not those employing PrA (Figure 2(b)). Hence, the coordinator can forget about the outcome of an aborted transaction once the PrN and PrC participants acknowledge the abort decision. If a PrA participant inquires about an (abort) final decision after the coordinator has forgotten the transaction, the coordinator, knowing that the participant uses PrA, will direct the participant to abort the transaction (by the presumption of PrA).

In PrAny, a coordinator records the 2PC protocol employed by each participant in a table called *participants' commit protocol* (PCP). The PCP is kept on stable storage and updated when a new site joins or leaves the MDBS. Only a portion of the PCP, called *active participant protocols* (APP) table, is maintained in main memory, containing the identities (IDs) of the participants with active transactions. APP can be structured as a hash table on the ID numbers of the participants.

A coordinator refers to its APP to decide which protocol to use with the participants in the execution of a transaction. The coordinator selects PrN if all the participants use PrN. Similarly, it selects PrA if all the participants use PrA whereas it decides to use PrC if all the participants use PrC. By using PrN, PrA or PrC with all the participants, the coordinator will always be in a safe state if it does not remember the final outcome of a transaction.

In the event that some of the participants employ PrA while the others employ PrN or PrC, the coordinator selects PrAny. From the coordinator's perspective, PrAny consists of the same two phases, i.e., the voting phase and the decision phase, as in PrN, PrA and PrC. The only distinction between

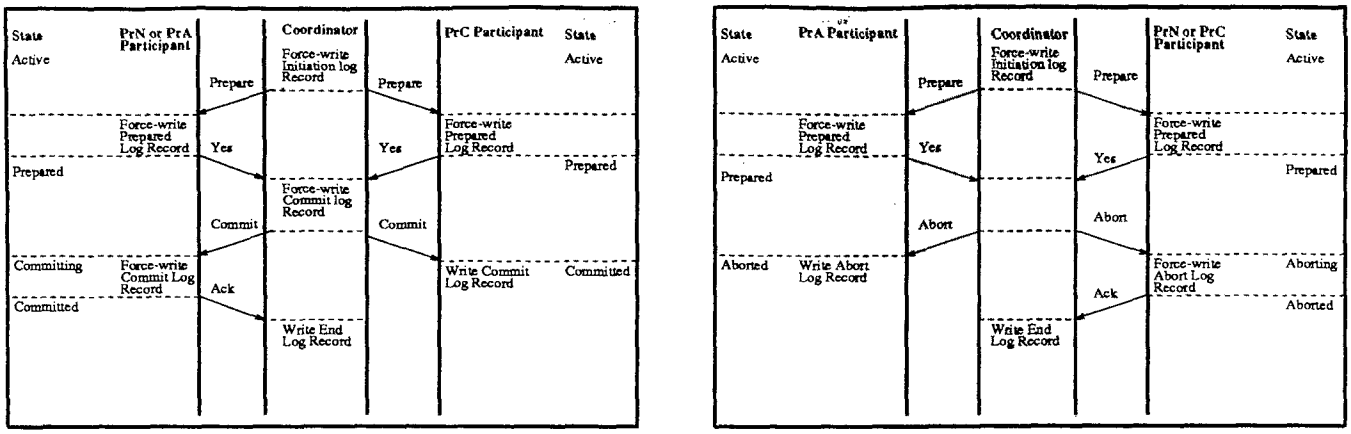


Figure 2: The presumed any protocol.

PrAny and the other variants is in the logging activities at the coordinator's site and the timing at which the coordinator can safely forget about the outcome of transactions.

In PrAny, the coordinator starts the voting phase by force writing an `initiation` record which includes the identities of the participants as it is the case in the PrC variant. Then, it sends to each participant a prepare to commit request. Once the coordinator receives the votes from all the participants, it force writes a `commit` record if the decision is commit. If the decision is abort, no decision record is written into the log. Then, the coordinator sends its final decision to all the participants. On a commit final decision, the coordinator writes a non-forced `end` record once all the PrN and PrA participants acknowledge the decision. On an abort final decision, on the other hand, the coordinator writes an `end` record once all the PrN and PrC participants acknowledge the decision. After the coordinator writes `end` record in its log, it discards all information pertaining to the transaction from its protocol table and can garbage collect the log records regarding the transaction.

Recovery in PrAny

As in all other commit protocols, communication and site failures are detected by timeouts. The recovery procedure in case of communication and participants' failures are handled in a manner similar to the way they are handled in PrN, PrA and PrC protocols. The only difference between PrAny and the other 2PC variants is in the way a coordinator recovers after a site failure. To reduce the cost of recovery, in all protocols, the coordinator is required to record the protocol of each participant along with its ID (in the `initiation` record for PrC and PrAny, and in the `decision` record for PrA and PrN). That is, without accessing the PCP on stable storage, the recovery procedure can determine the commit protocol used for each transaction from the information in its stable log.

After a failure, at the beginning of its recovery process, the coordinator re-builds its protocol table by analyzing its stable log. For each transaction that has a `decision` log record without an `initiation` record, it means that PrN or PrA has been used for its commitment. For each such transaction without an `end` record, the coordinator adds the transaction in its protocol table and re-initiates the decision phase with the recorded decision in the log. In the case of PrA, the decision is always commit since PrA requires only commit decisions to be recorded in the log. In the case of PrN, the decision could be either commit or abort.

For each transaction that has an `initiation` record, it means that PrC or PrAny has been used for its commitment. For each such transaction that PrC has been used for its commitment and has no `commit` or `end` log record, the coordinator adds the transaction in its protocol table and re-initiates the decision phase with an abort decision in accordance to PrC.

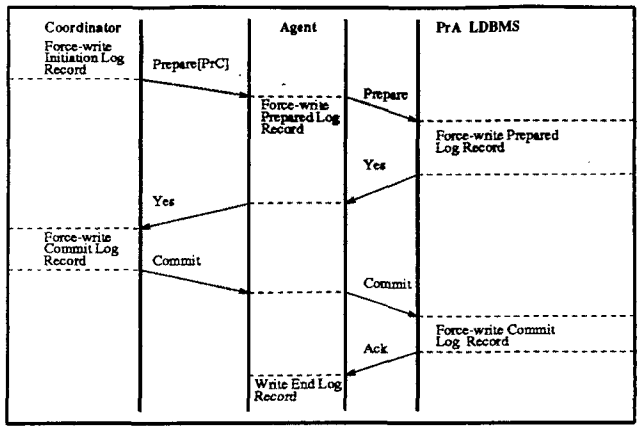
Finally, for each transaction that PrAny has been used for its commitment and has only an `initiation` record, or has `initiation` and `commit` records but no `end` record, the coordinator adds the transaction in its protocol table. In the former case, since either no decision was made or abort was decided before the failure, the coordinator submits an abort decision to the PrN and PrC participants. It does not include the PrA participants in accordance to PrA[1]. In the latter case, since a commit decision record is found, the coordinator submits a commit decision to the PrN and PrA participants but, in accordance to PrC, not to PrC participants.

As during normal processing, after sending out a decision, the coordinator waits for acknowledgments from PrN and PrC participants in the case of an abort decision and from PrN and PrA participants in the case of a commit decision. When a participant receives a final decision, it enforces and acknowledges the decision if it has not already enforced the decision. Otherwise, the participant simply acknowledges the decision[2]. When all the expected acknowledgments arrive, the coordinator writes an `end` log record and forgets about the transaction.

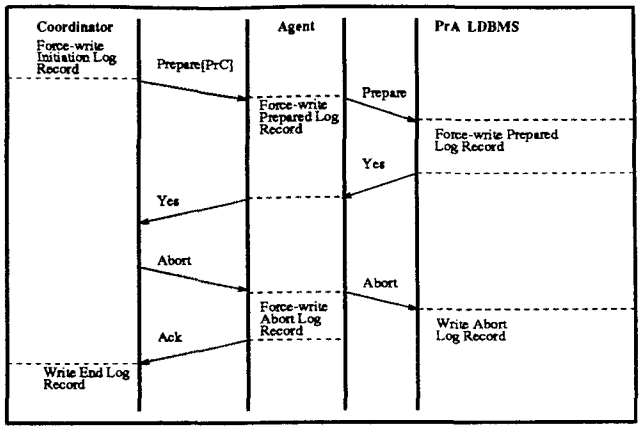
If a participant inquires about the outcome of a transaction

[1] A coordinator in PrA never re-submits an abort decision to the participants after its failure because it will not have any recollection about aborted transactions. It is the responsibility of the participants to inquire about the outcome of such transactions. Similarly, a coordinator in PrC never re-submits commit decisions to the participants after its failure.

[2] A participant without any memory regarding a transaction is assumed to have already received and enforced the decision and discarded all information pertaining to the transaction.



(a) Commit case



(b) Abort case

Figure 3: Transaction commit on PrA LDBMS using PrC.

after the coordinator has failed and recovered, not remembering the transaction, the coordinator replies with a commit message if the participant employs PrC or an abort message if the participant employs PrA.

SWITCHING BETWEEN PrA and PrC

During normal processing, PrA exhibits better performance than PrC if a transaction is most probably going to be aborted while PrC exhibits better performance than PrA if the transaction is most probably going to be committed. Usually, the behavior of transactions and systems changes over the time depending on several factors including the system load and the degree of conflicts between transactions. To be adaptive to the changes in a MDBS environment, it is necessary to be able to switch from one commit protocol to another, hence, at any given time, using the most appropriate 2PC protocol. The issue of using either PrA or PrC on a per transaction basis has been previously addressed in the context of traditional distributed database systems [9].

The cost of communication between a coordinator and the participants as well as of forcing the logs in a MDBS might vary from one site to another. Thus, rather than changing the used commit protocol on a per transaction basis, it would be more efficient to change the used protocol on a *per participant* and a *per coordinator* basis (GTM is a logical entity that can be distributed among different sites in a MDBS). That is, in a per participant basis, a coordinator might prefer to use one protocol with a participant during a certain period of time and to switch to another protocol during other periods of time. In a per coordinator basis, a participant might choose to use one protocol with some coordinators during certain periods of time while using the other protocol with the other coordinators.

PrAny facilitates all the above three options, allowing the coordinators and agents to cooperate in selecting the most efficient commit strategy at any given time. In these optimizations, the agents act as the participants in the case that the selected commit protocol is different than the one of their corresponding LDBMSs, while at the same time, they act as coordinators for their corresponding LDBMSs. Otherwise, they act as gateways for message passing between the GTM and the LDBMSs. That is, the agents are considered as *active* components with logging capabilities participating in the selection and the progress of the commit protocol rather than *passive* ones as they were assumed in the previous section. An agent is required to record the used protocol after a switch in a `prepared` log record to achieve operational correctness. That is, the information on switched participants is kept either on the coordinator or the agents as long as it is needed to ensure that the coordinator is in a safe state.

A. Switching on a Per Transaction Basis

Based on the behavior of transactions and the participants, a coordinator might instruct an agent to use PrC even though the underlying LDBMS uses PrA or visa versa. This is in order to reduce the number of coordination messages at the expense of an extra forced log write which *is desirable if communication is more expensive than the write of the log on disk.*

Figure 3 shows how a coordinator and an agent cooperate to commit a subtransaction executing on a PrA LDBMS using PrC. In this optimization, the coordinator force writes an `initiation` record which includes the identities of all participating agents and of those that are to be switched from PrA to PrC. Then, the coordinator sends to all the agents a *prepare to commit* message which is augmented with a used protocol field. When an agent receives a *prepare to commit* that directs it to switch, the agent forwards the message to the underlying LDBMS and force writes a `prepared` log record indicating that PrC is used for the transaction. When the underlying LDBMS sends its vote to the agent, the agent forwards the vote to the coordinator, only after the `prepared` record the agent has written is in the stable log. Thus, when a coordinator receives a vote, it knows that the agent has the required information for recovery in case of failure, and may forget about the transaction after making a decision.

According to PrC, on a commit decision (Figure 3(a)), the coordinator force writes a `commit` record, submits its decision to all the agents, and forgets about the transaction. When the agent receives the commit final decision, it passes the decision to the LDBMS. Since the LDBMS uses PrA, the agent expects an acknowledgment from the LDBMS. Once the LDBMS acknowledges the decision, the agent writes an `end` record and forgets about the transaction as well, knowing that the LDBMS will not inquire about the

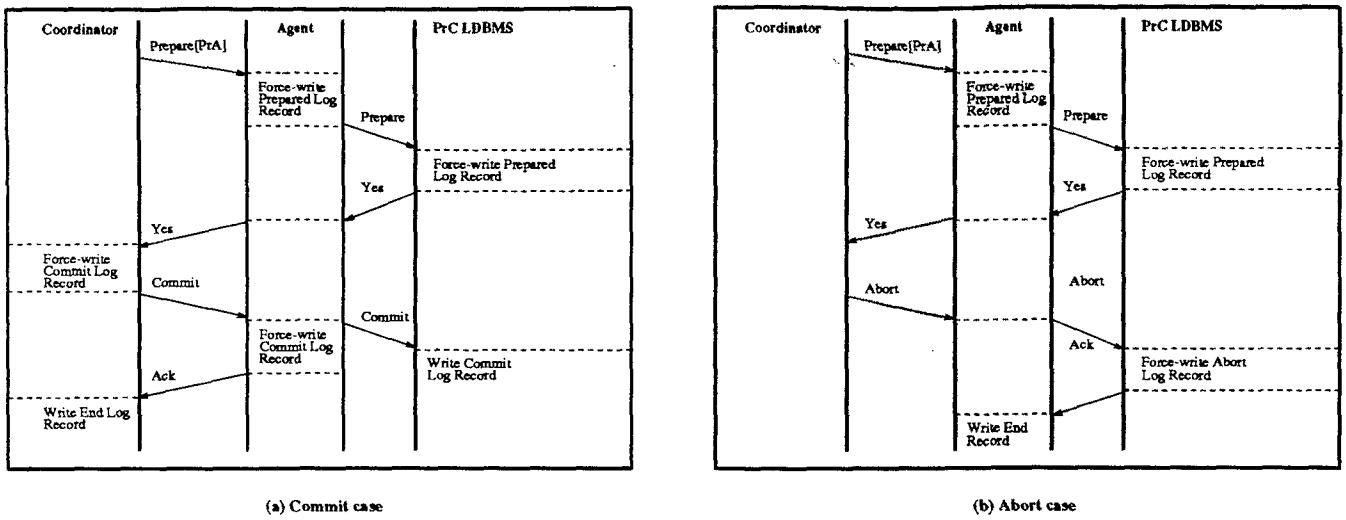


Figure 4: Transaction commit on PrC LDBMS using PrA.

transaction in the future.

On an abort final decision (Figure 3(b)), the coordinator sends abort to the agents without logging the decision. When an agent receives an abort decision, it forwards it to the LDBMS, force writes an abort record, and then sends an acknowledgment to the coordinator. Once the coordinator receives the acknowledgments from all the agents, it writes an `end` record and forgets about the transaction.

Figure 4 shows how a coordinator cooperates with an agent to commit a subtransaction executing on a PrC LDBMS using PrA. The coordinator begins by sending out a `prepare to commit` message to all participating agents, specifying PrA as the protocol to be used. When an agent receives the *prepare to commit* message, it force writes a `prepared` record indicating that PrA protocol is used for the transaction. Since the agent is acting as a PrC coordinator for the LDBMS, the `prepared` record is also used as an `initiation` that needs to be force written at the beginning of PrC to ensure atomicity. Once the `prepared` record is in the stable log, the agent forwards the message to the LDBMS[3]. When the agent receives the vote of the LDBMS, it passes the vote to the coordinator.

Based on PrA, on a commit final decision (Figure 4(a)), the coordinator force writes a `commit` record which includes the identities of the participants and then submits its decision to the agents. When an agent receives the commit decision, it passes the decision to the LDBMS and force writes a `commit` record. Since the agent is using PrA with the coordinator, it finally acknowledges the decision when the `commit` record is in the stable log. Once the coordinator receives the acknowledgment messages, it writes an `end` record and forgets about the transaction.

[3]Notice that sending the *prepare to commit* message to the LDBMS and force writing the `prepared` record cannot be overlapped, as in the case of switching a PrA LDBMS to use PrC, for the commit presumption to hold. This is because the coordinator does not force write an `initiation` record according to PrA, and if a failure occurs after the LDBMS has prepared the transaction and before the agent has forced the `prepared` record, the recovery procedure, by not finding any information about the transaction on either the coordinator or the agent, might wrongly decide to commit the transaction based on the PrC presumption.

On an abort final decision (Figure 4(b)), on the other hand, the coordinator submits its decision to all the agents and forgets about the transaction without having to write any log records. Once an agent receives the abort decision, it passes the decision to the LDBMS and waits for its acknowledgment according to PrC used by the LDBMS. When the LDBMS acknowledges the abort decision, the agent writes an `end` record and forgets about the transaction.

Recovery of a Coordinator

After a failure, as in all 2PC protocols, during its recovery, the coordinator identifies all the transactions whose commitment was interrupted by the failure, adds them in its protocol table and re-initiates their decision phase.

For each transaction associated with an `initiation` record, indicating the use of PrC, but without either a `commit` or `end` record, the coordinator sends out to all the participant agents an abort decision and waits for their acknowledgments in accordance to PrC. It also specifies in the abort message that PrC is to be used. When the coordinator receives all the acknowledgments, it writes an `end` record and forgets about the transaction.

When an agent of PrC LDBMS that was not supposed to have switched, receives the abort decision, it forwards the decision to the LDBMS which has either received and enforced the abort decision prior to the coordinator failure or not. In the former case, the LDBMS, not remembering the transaction, simply acknowledges the abort decision. In the latter case, the LDBMS first enforces the decision and then acknowledges it. In either case, when the agent receives the acknowledgment from the LDBMS, it forwards the acknowledgment to the coordinator.

When an agent of a PrA LDBMS that was supposed to have switched receives the abort message, it replies with an acknowledgment message, if it does not remember the transaction, i.e., the agent has finished the protocol prior to the failure. If the agent remembers the transaction, i.e., it has not received the final decision prior to the coordinator's failure and has to finish the protocol, it forwards the abort

decision to the LDBMS, force writes an `abort` record and acknowledges the coordinator.

For each transaction that is associated with a `commit` record but without either an `initiation` or an `end` record, the coordinator knows that PrA has been used with the transaction and sends a commit decision to all the participant agents, specifying that PrA is the used protocol. When an agent of a PrA LDBMS receives the commit message, it passes the message to the LDBMS. As above, the LDBMS either replies to the agent with an acknowledgment, if it has already received the decision prior to the coordinator's failure, or enforces the decision and then sends an acknowledgment to the agent. In either case, the agent forwards the acknowledgments to the coordinator.

When an agent of a PrC LDBMS that was supposed to have switched, receives the commit decision, it either replies with an acknowledgment, if it has no recollection about the transaction, or forwards the message to the LDBMS, force writes a `commit` record and then acknowledges the coordinator. When the coordinator receives the acknowledgment from all the agents, it writes an `end` record and safely forgets about the transaction.

Recovery of a Participant

After a failure, the agent and the LDBMS at a site recover simultaneously. The agent which acts as a coordinator for the LDBMS, re-builds its protocol table by considering only those transactions associated with a `prepared` record but without any other record in its stable log. The commitment of these transactions has not been completed before the failure. Furthermore, the existence of the `prepared` record indicates that the agent has switched protocols, using with the coordinator a protocol that differs from the one used by its LDBMS. Once the agent re-builds its protocol table, it starts accepting the inquiries of the LDBMS and final decisions from the coordinators.

When an agent receives a final decision from the coordinator for a transaction not in its protocol table, it forwards the decision to the LDBMS. If the transaction is in its protocol table, the agent forwards the decision to the LDBMS only if the decision is abort and the LDBMS uses PrC or the decision is commit and the LDBMS uses PrA. (Recall that, after a failure, a coordinator in PrA does not resubmit abort decisions to the participants and a coordinator in PrC does not resubmit commit decisions.) In case that the agent does not forward the decision to the LDBMS, it completes the protocol with the coordinator by force writing the appropriate decision record and then acknowledging the coordinator. In either case, the agent forgets about the transaction once the decision record is in the stable log.

During its recovery, the LDBMS will inquire about each transaction with a `prepared` record but without a corresponding final `decision` record. When the agent receives an inquiry message about a transaction from the LDBMS, the agent checks its protocol table. If the transaction is not in its protocol table, then it means that either (1) the agent has used a different protocol with the coordinator than with the LDBMS but finished the commitment of the transaction prior the failure, or (2) there was no switch in protocols and the agent was a passive component in the commitment of the transaction. Since the agent is in doubt about its role in the commitment of the transaction, it inquires the coordinator. If the coordinator remembers the transaction, it sends the final decision to the agent which in turn passes it to the LDBMS.

If the coordinator does not remember the transaction and the agent has not switched protocols (i.e., coordinator has completed the protocol with the LDBMS), it means that the coordinator has received all the expected acknowledgements from the agents and is in a safe state in which it can use the presumption of the protocol used by the LDBMS. In the case that the agent has switched protocols, then it is only possible for a LDBMS to send an inquiry message and for the coordinator and the agent not to remember the transaction, if either (1) the decision is abort, the agent has switched to PrC, and the LDBMS uses PrA, or (2) the decision is commit, the agent has switched to PrA and the LDBMS uses PrC (see Figures 3 and 4). Thus, in both cases, the coordinator refers to its PCP to determine which variant is used by the LDBMS and replies with an abort decision if the LDBMS employs PrA or a commit decision if the LDBMS employs PrC.

If the transaction is in the protocol table of the agent, the agent knows that it has switched protocols for this transaction but it has not finished the protocol with LDBMS by the time of the failure. Therefore, the agent inquires the coordinator with a message that includes the used protocol as specified in the `prepared` record. If the coordinator remembers the transaction, it responds with a final decision message and the agent completes the used protocol (as described in the previous section). On the other hand, if the coordinator does not remember the transaction, it uses the presumption of the protocol specified in the agent's inquiry message in its reply rather than the presumption of the protocol used by the LDBMS as it was the case above. The reason is that it is only possible for a LDBMS to send an inquiry message and for the coordinator not to remember the transaction while the agent remembers it, if either (1) the decision is commit, the agent has switched to PrC and the LDBMS uses PrA, or (2) the decision is abort, the agent has switched to PrA and the LDBMS uses PrC (see Figures 3 and 4)[4]. Thus, the coordinator replies with an abort decision if the agent has switched to PrA or a commit decision if the agent has switched to PrC.

B. Switching on a Per Participant Basis

Switching on a per transaction basis trades-off a forced log write at an agent with an acknowledgment message. Switching on a per participant basis, on the other hand, reduces the number of forced log writes at the agents. Instead of force writing a prepared record and an end/decision record for each transaction in a sequence of transactions that are to be processed using a protocol different than that of the LDBMS, the agent, in switching on a per participant basis, force writes a single `switch` log record for the entire sequence of transactions. In the `switch` record, the agent records in addition to the used protocol, the transaction identifier at the beginning of the sequence. Thus, assuming that transactions have unique, monotonically increasing identification numbers, a sequence of transactions can be subsequently determined by comparing the transaction IDs in two consecutive switch records.

[4] In both of these cases, the agent remembers the transaction because it has not received the expected acknowledgement from the LDBMS before the failure.

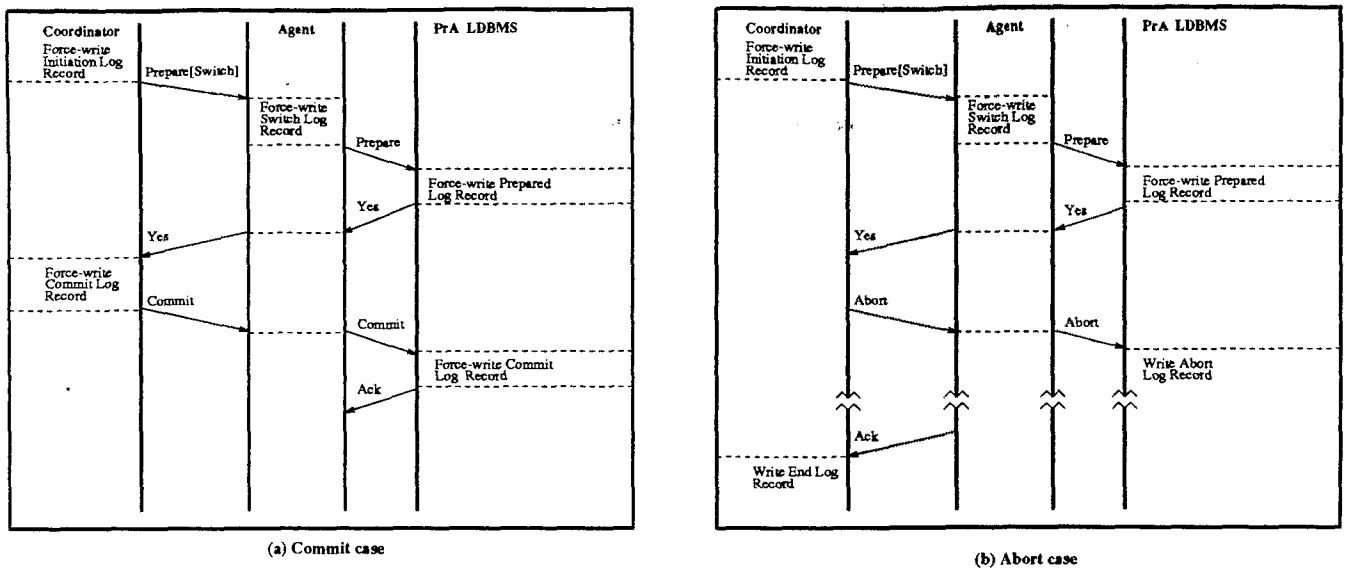


Figure 5: Switch an agent of a PrA LDBMS to support PrC.

In principle, this strategy is similar to the first one, namely, switching per transaction basis. However, an important issue in this strategy is when an agent can safely acknowledge a final decision that is not acknowledged by the LDBMS and without the force write of a decision log record at the agent. Recall that if a decision on a transaction is acknowledged prematurely by an agent (i.e., before it is logged at the LDBMS), allowing the coordinator to forget the transaction, after a failure, the coordinator might wrongly respond to an inquiry based on the presumption of the protocol. We prevent this from happening, i.e., acknowledging a decision only when it is safe, by treating a LDBMS as a cascaded coordinator and utilizing the sequential behavior of log. Specifically, each LDBMS is forced to act as coordinator with its agent as its leaf participant. Thus, any decision send to the LDBMS is forwarded back to the agent after the LDBMS force writes the appropriate decision record.

In this strategy, each coordinator maintains a look-up table in main memory, called *agents' protocol table* (APT) in which it records which agents have switched to a different protocol than the one used by their LDBMSs. Similarly, in order to be able to know the protocol to be used with each coordinator, each agent maintains a list, called *coordinators' commit protocol* (CCP). The behavior of the coordinator is similar to that of switching per transaction basis strategy (Figure 5). The only difference is that it has to update APT when it directs an agent to switch protocols with the *prepare to commit* message.

When the agent receives a *prepare to commit* message with a switch directive, the agent updates its CCP, force writes a `switch` record which also includes the CCP and then forwards the *prepare to commit* message to the LDBMS. If no switching is specified, the agent simply forwards the *prepare to commit* message to the LDBMS. Subsequently, the agent forwards the vote received from the LDBMS to the coordinator.

In the case that the agent uses PrC with the coordinator and PrA with the LDBMS, when the agent receives a commit decision (Figure 5(a)), it forwards the decision to the LDBMS and ignores the decision and acknowledgment messages received from the LDBMS. Also, the agent does not have to acknowledge the final decision in accordance to the PrC protocol used with the coordinator.

On the other hand, the agent has to acknowledge abort final decisions (Figure 5(b)). The agent determines whether an abort message has been received by the LDBMS, if the message is forwarded back to the agent by the LDBMS acting as cascaded coordinator. Furthermore, assuming sequential logs, an abort decision record pertaining to the transaction is guaranteed to be in the stable log of the LDBMS, if the agent subsequent to the received of the forwarded abort decision, sends to the LDBMS a commit decision of another transactions and receives an acknowledgement for the commit decision from the LDBMS. Thus, an agent can safely acknowledge an abort decision received from a coordinator, once the agent (as a leaf participant) receives an abort message from the LDBMS and a latter transaction has committed and acknowledges by the LDBMS, and knows that the LDBMS will not send any inquiry message after a failure in the future.

In the case that the agent uses PrA with the coordinator and PrC with the LDBMS, on a commit final decision (Figure 6(a)), when the agent receives the decision, it passes the decision to the LDBMS. Since the LDBMS is employing PrC, it will not acknowledge commit final decisions, a situation similar to the one discussed above where LDBMSs employing PrA will not acknowledge abort decisions. We use the same technique and observation as above to resolve this situation. That is, the LDBMS is treated as a cascaded coordinator with the agent as a leaf participant and utilizing the sequential nature of logs. In this way, an agent acknowledges a commit decision once it receives a commit message from the LDBMS and a subsequent transaction is aborted and acknowledged by the LDBMS. On an abort final decision (Figure 6(b)), on the other hand, when the agent receives the decision, it passes the decision to the LDBMS and ignores any acknowledgements of an abort decision from the LDBMS.

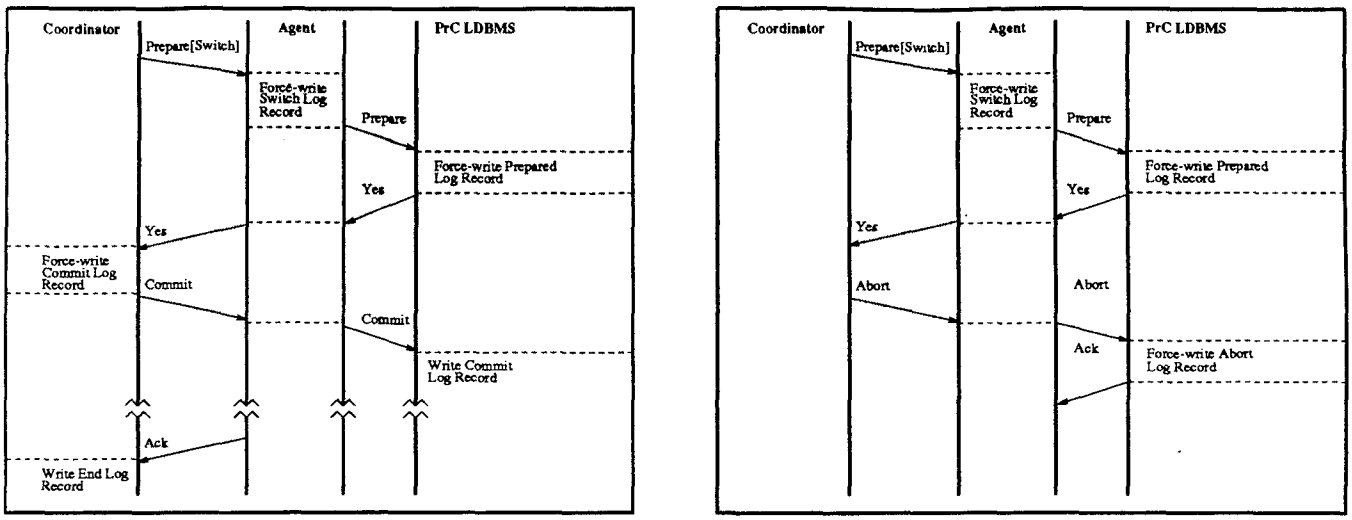


Figure 6: Switch an agent of a PrC LDBMS to support PrA.

When an agent is directed by a coordinator to switch back to the protocol used by the LDBMS, the agent updates its CCP and force writes it in a `switch` record. Then, the agent resumes its passive behavior during the commitment of a transaction. Even though the `switch` records are forced rarely and should not consume a large amount of stable storage, the agents should be able to garbage collect them. Therefore, an agent writes a non-forced `end-switch` record for each transaction sequence when the agent has switched to the protocol used by the LDBMS and each final decision pertaining to a transaction in the sequence has been either acknowledged by the LDBMS or its decision record is guaranteed to be in the stable log of the LDBMS.

Recovery in the Switch on a Per Participant Basis

The recovery of the coordinator after a failure is the same as in the switch on a per participant basis strategy discussed above. In addition, in this strategy, the recovery has to restore the consistency of the APT. This is achieved by initializing APT to PCP and directing all the agents to use the same commit protocol as their LDBMS. When an agent receives such a directive, it updates its CCP, force writes a `switch` log record and sends an acknowledgement back to the coordinator.

As in the case of the coordinator, the recovery of a participant after a failure is along the same lines as the recovery of the participant in the switch on a per transaction basis strategy. The only difference is that, in this strategy, an agent rebuilds its protocol table and determines the protocol used with each transaction from the information in the `switch` log records that have not a corresponding `end-switch` record.

C. Switching on a Per Coordinator Basis

This strategy works as the previous one except that an agent, in this strategy, initiates the protocol switch. Specifically, an agent suggests to a coordinator to use the most appropriate protocol once the agent determines that the transactions submitted by the coordinator have a higher probability of being aborted while the coordinator is using PrC with the agent (or vice versa). Once a coordinator receives such a suggestion, it directs the agent to switch to the suggested protocol as part of the next prepared-to-commit message sent to the agent.

CONCLUSION

In this paper, we showed that it is possible to integrate incompatible atomic commit protocols in a multidatabase system from a functional point of view as long as these protocols support a visible *prepare to commit* state. However, this result is not enough for a successful integration because the outcome of some transactions might have to be remembered forever. Therefore, we defined an operational correctness criterion for integration that allows transactions to be forgotten.

Based on the proposed operational correctness criterion, we developed a multidatabase two-phase commit (2PC) protocol, called *Presumed Any* (PrAny), that integrates the *presumed nothing*, *presumed abort* and *presumed commit* 2PC variants despite their conflicting presumptions about the outcome of transactions and without violating the autonomy of the local database systems.

Furthermore, based on the same principle, we have proposed three strategies that allow a multidatabase system to dynamically adapt to the most appropriate two-phase commit variant at any given time. This is achieved by structuring the agents to act as participants in the case that the selected commit protocol is different than the one of their corresponding LDBMSs, while at the same time, they act as coordinators for their corresponding LDBMSs. Depending on the strategy, agents may switch protocols on a per transaction, per participant and on per coordinator basis.

References

[1] Breitbart, Y., H. Garcia-Molina and A. Silberschatz. Overview of Multidatabase Transaction Management. *VLDB journal*, 1(2):181–239, 1992.

[2] Breitbart, Y., A. Silberschatz and G. Thompson. Reliable Transaction Management in a Multidatabase System. *Proc. of ACM SIGMOND International Conference on Management of Data*, pp. 215–224, 1990.

[3] Chrysanthis, P. K. and K. Ramamritham. Autonomy Requirements in Heterogeneous Distributed Database Systems. *Proc. of the Conference on the Advances on Data Management*, pp. 283-302, 1994.

[4] Gligor, V. D. and G. L. Lunckenaugh. Interconnecting Heterogeneous Database Management Systems. *IEEE Computer*. 17(1): 33–43, 1984.

[5] Gray, J. Notes on Data Base Operating Systems. In Bayer R., R.M. Graham, and G. Seegmuller (Eds.), *Operating Systems: An Advanced Course*, Lecture Notes in Computer Science, Vol. 60, pp. 393–481, Springer-Verlag, 1978.

[6] Lampson, B.W. Atomic Transactions. In *Distributed Systems: Architecture and Implementation - An Advanced Course*, B.W. Lampson (Ed.), Lecture Notes in Computer Science, Vol. 105, pp. 246–265, Springer-Verlag, 1981.

[7] Levy, E., H. Korth and A. Silberschatz. An Optimistic Commit Protocol for Distributed Transaction Management. *Proc. of the ACM SIGMOD International Conference on Management of Data*, pp. 88–97, 1991.

[8] Mohan, C. and B. Lindsay. Efficient Commit Protocols for the Tree of Processes Model of Distributed Transactions. *Proc. of the 2nd ACM Symposium on Principles of Distributed Computing*, 1983.

[9] Mohan, C., B. Lindsay and R. Obermarck. Transaction Management in the R^* Distributed Data Base Management System. *ACM Transactions on Database Systems*, 11(4):378–396, 1986.

[10] Samaras, G., K. Britton, A. Citron and C. Mohan. Two-Phase Commit Optimizations and Tradeoffs in the Commercial Environment. *Proc. of the 9th International Conference on Data Engineering*, pp. 520–529, 1993.

[11] Skeen, D. Non-blocking Commit Protocols. *Proc. of ACM SIGMOD Conference on the Management of Data*, pp. 133–142, 1982.

[12] Soparkar, N., H. Korth and A. Silberschatz. Failure-Resilient Transaction Management in Multidatabases, *IEEE Computer*, 24(12):28–36, 1991.

[13] Tal, A. and R. Alonso. Integration of Commit Protocols in Heterogeneous Databases. *Technical Report TR-375-92, Princeton University*, 1992.

[14] Veijalainen, J. and A. Wolski. Prepare and Commit Certification for Decentralized Transaction Management in Rigorous Heterogeneous Multidatabases, *Proc. of the 8th International Conference on Data Engineering*, pp. 470–479, 1992.