Atomicity with Incompatible Presumptions

Yousef J. Al-Houmaily
Dept. of Computer Programs
Institute of Public Administration
Riyadh 11141, Saudi Arabia

E-mail: houmaily@ipa.edu.sa

Panos K. Chrysanthis*
Dept. of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260, USA

E-mail: panos@cs.pitt.edu

ABSTRACT

We identify one of the incompatibility problems associated with atomic commit protocols that prevents them from being used together and we derive a correctness criterion that captures the correctness of their integration. We also present a new atomic commit protocol, called *Presumed Any*, that integrates the three commonly known two-phase commit protocols and prove its correctness.

Keywords

Two-Phase Commit, Atomic Commit Protocols, Distributed Transaction Processing, Multi-database Systems, Distributed Systems.

1. INTRODUCTION

An *atomic commit protocol* (ACP) is the only mean to ensure the traditional atomicity property of transactions in any distributed database system. This is to guarantee, in spite of communication and site failures, that all sites participating in a transaction's execution agree on the final outcome of the transaction, i.e., to either commit or abort the transaction. Since commit processing consumes a substantial amount of a transaction's execution time and ACPs are blocking in the case of failures, a variety of ACPs and optimizations have been proposed in the literature. Some of which enhance the *performance* of the basic *two-phase commit* (2PC) protocol (which is also known as *presumed nothing* protocol (PrN)), during normal processing, while the others reduce the cost of *recovery* processing after failures [5, 16, 9].

With the recent advance of intranet and internet technologies, there is a greater need than ever before to integrate different database environments in a practical and efficient manner. Such integration is absolutely necessary to support the *interoperability characteristic* of advanced future database applications such as electronic commerce, multi-organizational workflows and web-based transactions (to name just a few). A key requirement of these applications is the ability to support universal transactional access and, in particular, to guarantee the atomicity of transactions in the presence of *incompatible* ACPs.

In this paper, we address the issue of compatibility among ACPs in a distributed database environment, such as a *multidatabase system* (MDBS), in which not all sites use the same ACP. As we show in section 2, the incompatibility of the various ACPs may be due to the differences in the semantics of their coordination messages or actions. In the case of the three commonly known two-phase commit variants (namely, the presumed nothing (PrN), *presumed abort* (PrA) and *presumed commit* (PrC) protocols[1]), the incompatibility arises because of their conflicting presumptions about the outcome of (terminated) transactions in the presence of failures.

In section 3, in contrast to what it was previously believed [6, 18], we show that supporting a *visible prepare-to-commit* state is not enough for a practical integration of ACPs because the outcome of some transactions might have to be remembered forever. This leads us to distinguish between *functional* correctness and *operational* correctness and to define both the notion of a *safe* state in ACPs and an *operational* correctness criterion that, besides requiring functional correctness, allows terminated transactions to be forgotten.

Finally, in section 4, we present *presumed any* (PrAny), a two-phase commit protocol variant that successfully integrates PrN, PrA and PrC protocols, and prove its correctness with respect to our operational correctness criterion. We choose to integrate PrN and PrA because they have been widely implemented in commercial systems, and PrC be-

*This work is supported in part by National Science Foundation under grant IRI-95020091.

PODS '99 Philadelphia PA

[1] In the appendix, we provide a brief review of related work including PrN, PrA and PrC protocols.

cause it is expected to become part of the standards as we argued in [4].

We use ACTA [7], a first order predicate logic formalism, to express the safe state in PrAny. All ACPs can be specified and all theorems can be proven using ACTA, by modeling log operations and system crashes as transactions' significant events.[2] Here, however, for the sake of brevity and simplicity, the proofs are structured in a manner similar to the proofs of ACPs in [5]. In all the proofs, we assume that (1) each site is *sane* and (2) each site can cause only *omission* failures. That is, each site is assumed to be *fail stop* where it never deviates from the specification of the protocol that is using, and when it fails, it will, eventually, recover.

2. COMPATIBILITY OF 2PC VARIANTS

In this section, we examine the compatibility of PrN, PrA and PrC by assuming that they can coexist in a system and can be used together to commit a distributed transaction. Further, we assume that a coordinator follows its own protocol, knows what messages to expect from each participant and handles any violations of its protocol with respect to messages by ignoring such messages. We call this type of integrated protocol used by a coordinator as *union two-phase commit* (U2PC) protocol [1]. In our examples below, a site will follow U2PC when acting as a coordinator and its original ACP when acting as a participant.

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, in accordance to PrC, the coordinator does not expect any commit acknowledgment messages. However, the PrA participant will acknowledge the commit decision. By knowing that this participant will send an acknowledgment, the coordinator will not consider this message since this message is a violation of its protocol. 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 PrA participant never acknowledges an abort decision. This means that the coordinator which expects acknowledgment messages from all participants can never garbage collect the records pertaining to the transaction from its stable log nor can it discard the information from its protocol table. To alleviate this situation, in U2PC, the coordinator forgets the outcome of the transaction once it has received the acknowledgment of the PrC participant, knowing that the PrA will never acknowledge such a decision. In this case, the atomicity of the transaction might be violated. For example, if the PrA participant fails after it has received the final outcome but before writing it in its stable log, the participant will inquire about the outcome of the transaction as part of its recovery procedure. 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.

A similar situation may occur 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 atomicity of both committed and aborted transactions might be violated. On the other hand, if the coordinator is using PrA, the atomicity of committed transactions might be violated.

The above scenarios can be generalized with the following theorem.

Theorem 1: It is impossible to ensure *global atomicity* of distributed transactions executed at both PrA and PrC participants with a coordinator using U2PC.

Proof: The proof proceeds by contradiction and consists of three parts. The first is when the coordinator is using PrN. The second is when the coordinator is using PrA. The third is when the coordinator is using PrC.

Part I: Consider a coordinator that uses PrN and a transaction that has executed at two participants one of which is using PrA whereas the other is using PrC. Assume that the coordinator decides to commit the transaction while ensuring its atomicity. In this case, the PrA participant will acknowledge the commit decision but the PrC participant will not. Now, it is possible for the PrC participant to fail before receiving the commit decision and for the inquiring message of the PrC participant to arrive after the coordinator has received the acknowledgment of the PrA participant and forgotten the transaction. In this case, the coordinator will respond with an abort decision (using the PrN presumption[3]) which violates the

[2] In [8], PrN was specified in ACTA and the important aspect of its functional correctness was shown.

[3] This is due to the hidden presumption in PrN that we discuss in the appendix.

atomicity of the transaction, thus contradicting the assumption.

Part II: Consider a transaction that has executed at two participants as above but the coordinator is using PrA instead of PrN. Assume that the coordinator decides to commit the transaction while ensuring its atomicity. In this case, the PrA participant will acknowledge the decision but the PrC one will not, as above. Now, it is possible for the PrC participant to fail before receiving the commit decision and for inquiring message to arrive after the coordinator has received the acknowledgment of the PrA participant and forgotten the transaction. In this case, the coordinator will respond with an abort decision (using the PrA presumption) which violates the atomicity of the transaction, thus, contradicting the assumption.

Part III: We have proven this part in our motivating example at the beginning of this section.

□

3. OPERATIONAL CORRECTNESS

Clearly, the obvious solution of a U2PC coordinator "talking" PrA to those participants implementing PrA and "talking" PrC to those implementing PrC does not work. The U2PC protocol might violate transaction atomicity because the coordinator forgets about transactions prematurely. Let us consider an alternative integrated protocol, called *coordinator two-phase commit* (C2PC) protocol which behaves similar to U2PC, but unlike U2PC, C2PC never forgets a transaction until it has received all necessary acknowledgments.

As we have discussed above, some participants will never acknowledge either commit or abort decisions. This means that the coordinator will never be able to discard information from either its protocol table or stable log pertaining to some terminated transactions. Since these terminated transactions when they are forgotten might lead to a wrong presumption (as seen in U2PC), C2PC does not lead to atomicity violations by requiring that a coordinator always remembers the outcome of these terminated transactions and never uses its presumption after a failure. Thus, even though C2PC guarantees *functional correctness* in which it ensures the atomicity of all distributed transactions, it fails to guarantee *operational correctness* which requires that the coordinator should be able to eventually forget about the outcome of terminated transactions, as the following definition states.

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.

Since C2PC has to remember the outcome of some transactions forever, we generalize this result with the following theorem.

Theorem 2: It is impossible to achieve *operational correctness* if the coordinator is using C2PC and distributed transactions execute at both PrA and PrC participants.

Proof: The proof proceeds by contradiction and consists of three parts. The first is when the coordinator is using PrN. The second is when the coordinator is using PrA. The third is when the coordinator is using PrC.

Part I: Consider a coordinator that uses PrN and a transaction that has executed at two participants one of which is using PrA whereas the other is using PrC. Assume that the coordinator decides to commit the transaction and can eventually forget the transaction. In this case, the PrA participant will acknowledge the commit decision but the PrC participant will not. Hence, the coordinator will not be able to write an end log record and has to remember the transaction forever which contradicts the assumption.

Part II: Consider a coordinator that uses PrC and a transaction that has executed at two participants one of which is using PrA whereas the other is using PrC. Assume that the coordinator decides to commit the transaction and can eventually forget the transaction. In this case, the PrA participant will acknowledge the decision but the PrC one will not, as above. Hence, the coordinator will not be able to write an end log record and has to remember the transaction forever which contradicts the assumption.

Part III: Consider a transaction that has executed at two participants as above but the coordinator is using PrC. Assume that the coordinator decides to abort the transaction and can eventually forget the transaction. In this case, the PrC participant will acknowledge the decision but the PrA one will not. Hence, the coordinator will not be able to write an end log record and has to remember the transaction forever which contradicts the assumption.

□

To maintain operational correctness in an ACP, a coordinator should be able to, eventually, forget the outcome of transactions without violating the consistency of its decisions. We call this a *safe state*. Intuitively, a coordinator is in a safe state with respect to a transaction if

(1) it forgets a transaction after all participants have acknowledged its decision (as in PrN), or

(2) it can use a single presumption that is consistent with the transaction's final outcome.

Thus, in order to integrate PrA and PrC in spite of their conflicting presumptions, we need a safety criterion that will allow a coordinator to reach a safe state in which only a single presumption will hold. In order to be consistent with both PrA and PrC, we propose the following safety criterion for their integration. The safety criterion is expressed using ACTA [7], a first order predicate logic with a precedence relation ($\rightarrow$) in H. H represents the complete history of the execution of the transaction until it is either committed or aborted at all sites. C denotes the coordinator of the transaction. The predicate $\epsilon \rightarrow \epsilon'$ is true if event ϵ *precedes* event ϵ' in H. It is false, otherwise. Here, $Decide_C(Abort)$ denotes that the coordinator decides to abort a transaction T and $Decide_C(Commit)$ denotes that the coordinator decides to commit a transaction. $DeletePT_C(T)$ denotes that the information pertaining to T is deleted from the protocol table of the coordinator. INQ_{t_i} denotes an inquiry message from a participant regarding a subtransaction t_i that it has executed at its site on behalf of T. $Respond_C(Outcome_{t_i})$ denotes the reply of the coordinator to the inquiry message.

Definition 2: (The definition of safe state)

$$
\begin{aligned}
&SafeState_C(T) \Rightarrow \\
&\quad ((Decide_C(Abort_T) \in H \wedge \\
&\qquad \forall t_i \in T\ (DeletePT_C(T)) \rightarrow INQ_{t_i}) \Rightarrow \\
&\qquad\qquad Respond_c(Abort_{t_i}) \in H) \vee \\
&\quad ((Decide_C(Commit_T) \in H \wedge \\
&\qquad \forall t_i \in T\ (DeletePT_C(T)) \rightarrow INQ_{t_i}) \Rightarrow \\
&\qquad\qquad Respond_C(Commit_{t_i}) \in H)
\end{aligned}
$$

The above definition states that a coordinator is in a safe state with respect to the outcome of a transaction T, if T has been aborted and only the presumed abort presumption holds (the first clause of the safe state implication), or T has been committed and only the presumed commit presumption holds (the second clause).

This safety criterion implies that some information including the outcome of transactions has to be remembered as long as more than one presumption is possible.

4. PRESUMED ANY (PrAny)

In this section, we describe the PrAny protocol that integrates PrN, PrA and PrC according to the operational correctness criterion that we have defined above. First, we describe PrAny during normal processing. Then, in Section 4.2, we discuss the recovery aspects of PrAny in case of failures. In Section 4.3, we prove the correctness of PrAny protocol.

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 is updated when a new site joins or leaves the distributed environment. Only a portion of the PCP, called *active participants' protocols* (APP) table, is maintained in main memory, containing the identities (IDs) of the participants with active transactions.

4.1 PrAny During Normal Processing

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, as shown in Figure 1. The only distinction between 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 in PrC. The initiation record also includes the protocol used by each participant. Then, the coordinator 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 (Figure 1 (a)). If the decision is abort, no decision record is written into the log (Figure 1 (b)). 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 writing the end record in its

stable log, the coordinator discards all information pertaining to the transaction from its protocol table.

4.2 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. 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 1 (a)). The coordinator forgets 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 and without examining its log. Note that, as opposed to PrA and PrC, PrAny does not make any a priori presumption but a PrAny coordinator dynamically adopts the presumption of an inquiring participant's protocol.

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 1(b)). Hence, the coordinator forgets 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.

After a failure, at the beginning of its recovery procedure, 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. Depending on the identities of the participants recorded in the initiation record and the protocols that they use, the coordinator determines which of the two protocols was used for the commitment of the transaction. 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.[4] 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.[5] When all the expected acknowledgments arrive, the coordinator writes an end log record and forgets about the transaction.

4.3 Proof of Correctness

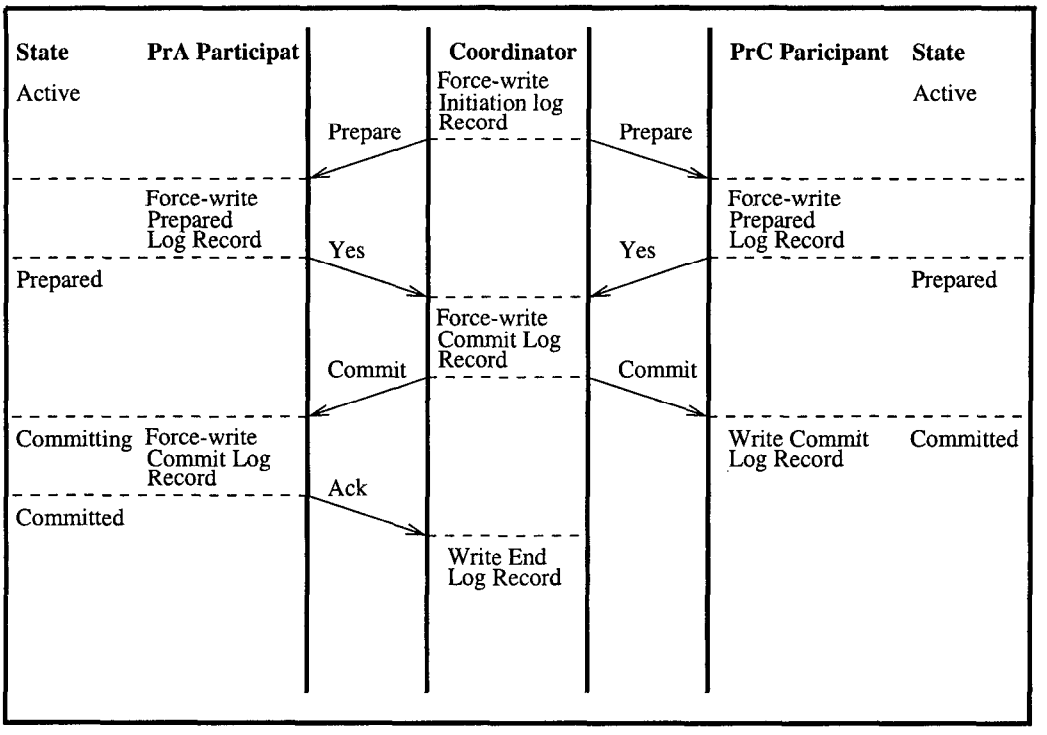
In [5], the behavior of PrN and how it recovers after failures is thoroughly discussed. That discussion provides an iterative method that prove the correctness of the protocol. That is, what would happen if a failure occurs and at what point during the course of protocol. We use below the same strategy to show the correctness of PrAny.

Theorem 3: The PrAny protocol satisfies the operational correctness criterion.

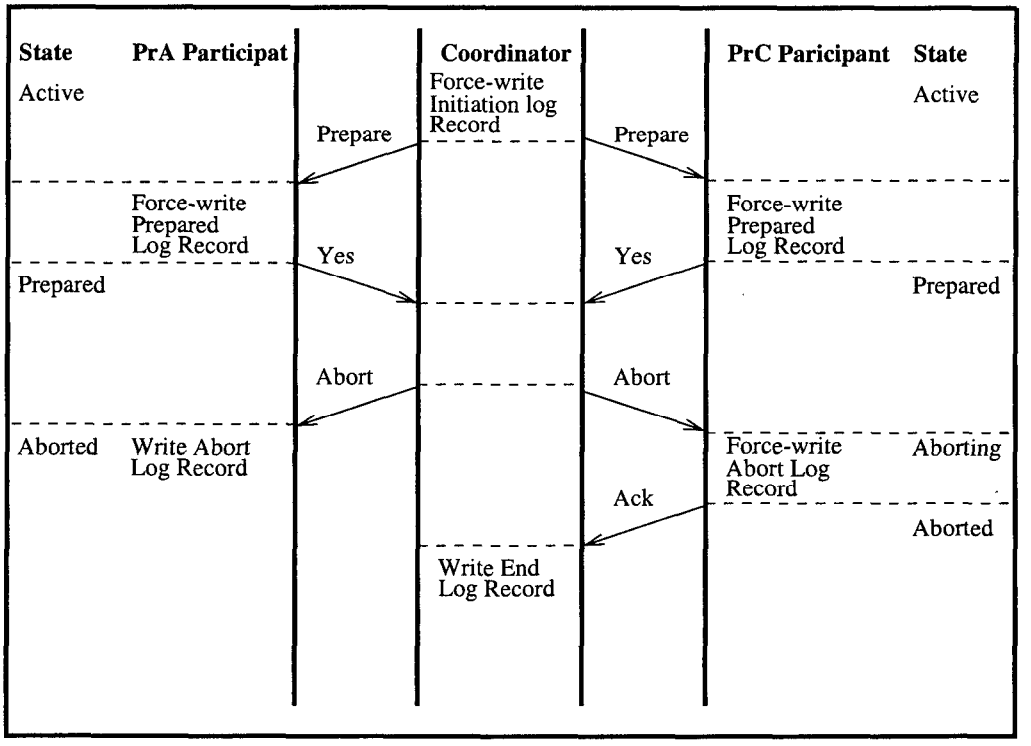
Proof: To show the correctness of PrAny, we need to show that all the three requirements of operational correctness are satisfied. PrAny consists of the same two phases as PrN. Hence, the first and the third requirements of the operational correctness criterion are satisfied since all participants in a transaction's execution will reach an agreement and forget about the transaction. The only remaining requirement that needs to be proven

[4] 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.

[5] 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 1: The presumed any protocol.

is the second one which requires that the coordinator should eventually be able to forget about the outcome of transactions. We prove the second requirement by considering the two possible outcome of transactions. The proof proceeds by contradiction, showing that a PrAny coordinator is always in a safe state.

Commit Case: Assume that the coordinator has made a commit decision and after forgetting the outcome of the transaction, it replies to an inquiry message with an abort decision.
If the inquiring participant is PrC, then the coordinator will use the commit presumption of PrC and will respond with a commit decision which contradicts the initial assumption.
In order to reply with an abort, it means that coordinator has used the abort presumption. This means that the message is from a PrA participant, but this is impossible since all PrA and PrN participants must have acknowledged the commit decision in order for the coordinator to forget the outcome of the transaction. Similarly, it is impossible for the inquiry message to be from a PrN participant.

Abort Case: Assume that the coordinator has made an abort decision and after forgetting the outcome of the transaction, it replies to an inquiry message with a commit decision.
If the inquiring participant is PrA, then the coordinator will use the presumption of PrA and will respond with an abort decision which contradicts the initial assumption.
In order to reply with an commit, it means that the coordinator has used the commit presumption. This means that the message is from a PrC participant, but this is impossible since all PrC and PrN participants must have acknowledged the abort decision in order for the coordinator to forget the outcome of the transaction. Similarly, it is impossible for the inquiry message to be from a PrN participant.

□

5. SUMMARY AND CONCLUSION

Our two contributions in this paper, one theoretical and the other more practical, are:

1. We showed that although it is possible from a functional point of view to integrate incompatible ACPs in a distributed database system as long as these protocols support a visible prepare to commit state, it is not enough for a practical integration because the outcome of some transactions might have to be remembered forever.
2. We defined an operational correctness criterion for ACP integration based upon which we developed *Presumed Any* (PrAny) that integrates the *presumed nothing*, *presumed abort* and *presumed commit* 2PC variants despite their conflicting presumptions about the outcome of terminated transactions and we proved its correctness.

The same operational correctness criterion can be used as a basis for the integration of a variety of ACPs, such as *coordinator log* [17] and *implicit yes-vote* [3], as well as atomic commit optimizations, such as read-only optimizations [15, 1, 4]. Currently, we are developing a coordinator protocol that dynamically recognizes the ACPs and optimizations used by participants and constructing the condition that expresses the operational criterion that allows for a practical interoperation in future dynamic database environments.

References

[1] Al-Houmaily, Y. J. Commit Processing in Distributed Database Systems and in Heterogeneous Multidatabase Systems. Ph.D. Thesis, Department of Electrical Engineering, University of Pittsburgh, Apr. 1997.

[2] Al-Houmaily, Y. J. and P. K. Chrysanthis. Dealing with Incompatible Presumptions of Commit Protocols in Multidatabase Systems. *Proc. of the 11th ACM Annual Symp. on Applied Computing*, pp. 186–195, Feb. 1996.

[3] Al-Houmaily, Y. and P. Chrysanthis. An Atomic Commit Protocol for Gigabit-Networked Distributed Databases, *J. of Systems Architecture, The EUROMICRO J.*, (To Appear).

[4] Al-Houmaily, Y., P. Chrysanthis and S. Levitan. An Argument in Favor of the Presumed Commit Protocol. *Proc. of the 13th Int'l Conf. on Data Engineering*, pp. 255–265, April 1997.

[5] Bernstein P. A., V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, Reading, MA, 1987.

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

[7] Chrysanthis P. K. and K. Ramamritham. Synthesis of Extended Transaction Models using ACTA. *ACM TODS*, 19(3):450–491, Sep. 1994.

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

[9] Chrysanthis, P. K. , G. Samaras and Y. Al-Houmaily. Recovery and Performance of Atomic Commit Processing in Distributed Database Systems (Ch. 13). *Recovery Mechanisms in Database Systems*, V. Kumar and M. Hsu, Eds., Prentice Hall, 1998.

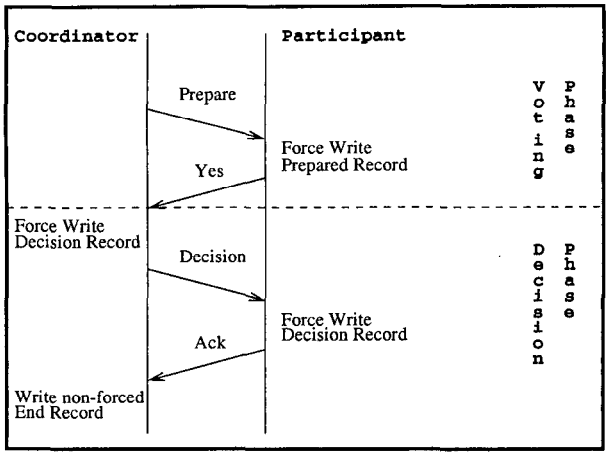


Figure 2: The basic 2PC protocol.

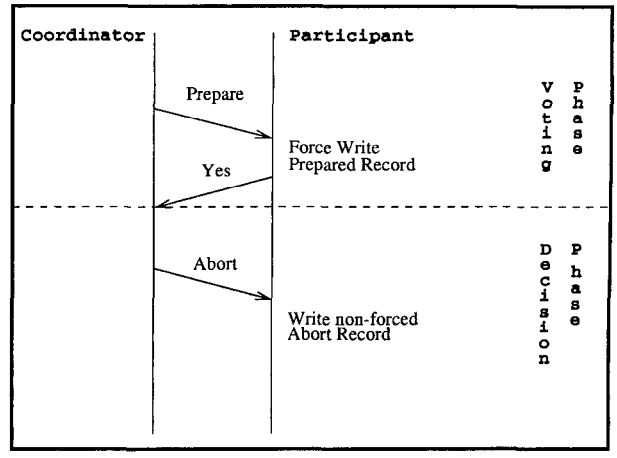


Figure 3: The presumed abort protocol.

[10] 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*, Volume 60, pp. 393–481, Springer-Verlag, 1978.

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

[12] Lampson, B. and D. Lomet. A New Presumed Commit Optimization for Two Phase Commit. *Proc. of the 19th Int'l Conf. on Very Large Data Bases*, pp. 630-640, Aug. 1993.

[13] Mullen, J. Atomic Commitment in Multidatabase Systems. Ph.D. Thesis, Department of Computer Science, Purdue University, West Lafayette, Indiana, Dec. 1993.

[14] Mohan, C. and B. Lindsay. Efficient Commit Protocols for the Tree of Processes Model of Distributed Transactions. *Proc. of the 2nd ACM SIGACT/SICOPS Symp. on Principles of Distributed Computing*, Aug. 1983.

[15] Mohan, C., B. Lindsay and R. Obermarck. Transaction Management in the R^* Distributed Data Base Management System. *ACM TODS*, 11(4):378–396, Dec. 1986.

[16] Samaras, G., K. Britton, A. Citron and C. Mohan. Two-Phase Commit Optimizations in a Commercial Distributed Environment. *Distributed and Parallel Databases*, 3(4):325–360, Oct. 1995.

[17] Stamos, J. and F. Cristian. Coordinator Log Transaction Execution Protocol. *Distributed and Parallel Databases*, 1(4):383–408, 1993.

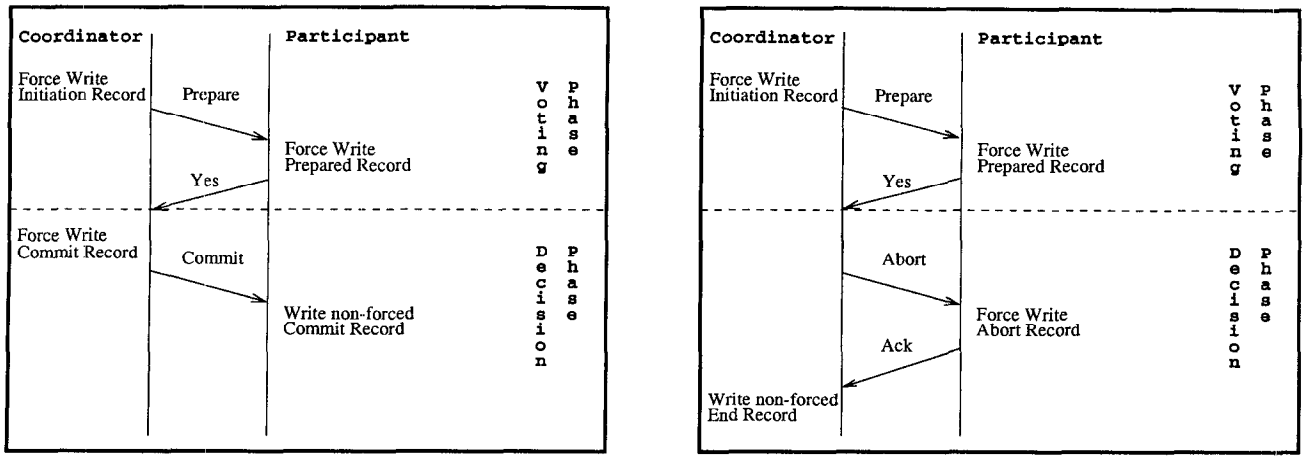
[18] Tal, A. and R. Alonso. Integration of Commit Protocols in Heterogeneous Databases. *Distributed and Parallel Databases*, 2(2):209–234, Apr. 1994.

APPENDIX: BRIEF OVERVIEW OF RELATED WORK

In distributed databases, a distributed transaction is decomposed into a set of *subtransactions*, each of which executes at a single participant site. Without loss of generality, the *transaction manager* at the site where the transaction has been initiated is responsible to coordinate the different aspects of the execution of the transaction and in particular, its commitment. When a transaction finishes its execution and submits its commit request, its coordinator initiates an *atomic commit protocol*, such as the two-phase commit protocol.

The basic *two-phase commit* protocol (2PC) [10, 11], as the name implies, consists of two phases, namely a *voting phase* and a *decision phase* (Figure 2). During the voting phase, the coordinator of a distributed transaction requests all the participating sites to *prepare to commit* whereas, during the decision phase, 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 discards any information in its *protocol table* in main memory regarding the transaction when it receives acknowledgments from all the participants and forgets the transaction.

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 force-writes 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 survives system failures, the final decision is not lost if the coordinator fails. Similarly, each participant force-writes a *prepared* record before send-



(a) Commit case.

(b) Abort case.

Figure 4: The presumed commit protocol.

ing its "Yes" vote and a *decision* record before acknowledging the final decision.[6] 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 is also referred to as the *presumed nothing* 2PC protocol (PrN) [12] 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 the 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. The *presumed abort* protocol (PrA) makes this abort presumption explicit [15].

Specifically, in PrA, when a coordinator decides to abort a transaction, it does not force-write the abort decision in its log as in PrN (Figure 3). It just sends abort messages to all the participants that have voted "Yes" and discards all information about the transaction from its protocol table. That is, 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. After a coordinator or a participant failure, if the participant *inquires* about a transaction that has been aborted, the coordinator, not remembering the transaction, will direct the participant to abort it (by presumption).

As opposed to PrA, the *presumed commit* protocol (PrC) is designed to reduce the cost of committing transactions [15, 12]. Instead of interpreting missing information about transactions as abort decisions, in PrC, coordinators interpret missing information about transactions as commit decisions. However, in PrC, a coordinator has to force write an *initiation* (which is also called *collecting* in [15]) record for each transaction before sending prepare to commit messages to the participants. This record ensures that missing information about a transaction will not be misinterpreted as a commit after a coordinator failure.

To commit a transaction (Figure 4 (a)), 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. After a coordinator or a participant failure, if the participant inquires about a transaction that has been committed, the coordinator, not remembering the transaction, will direct the participant to commit it (by presumption).

To abort a transaction (Figure 4 (b)), on the other hand, the coordinator does not write the abort decision in its log. Instead, the coordinator, sends out the abort decision and waits for the acknowledgments before discarding all information pertaining to the transaction. When a participant receives the decision, it force writes an abort record and then acknowledges the decision, as in PrN.

Unlike (homogeneous) distributed database systems, the constituent database sites in future distributed environments might use different atomic commit protocols such as the *basic two-phase commit* protocol, that we discussed above, or one of its variants (see [1] for a survey of the most commonly known 2PC variants). Furthermore, some sites might not support any form of ACPs. For this reason, a database site

[6] Writing the decision at the participants and acknowledging it in a lazy fashion is an optimization that is not considered here.

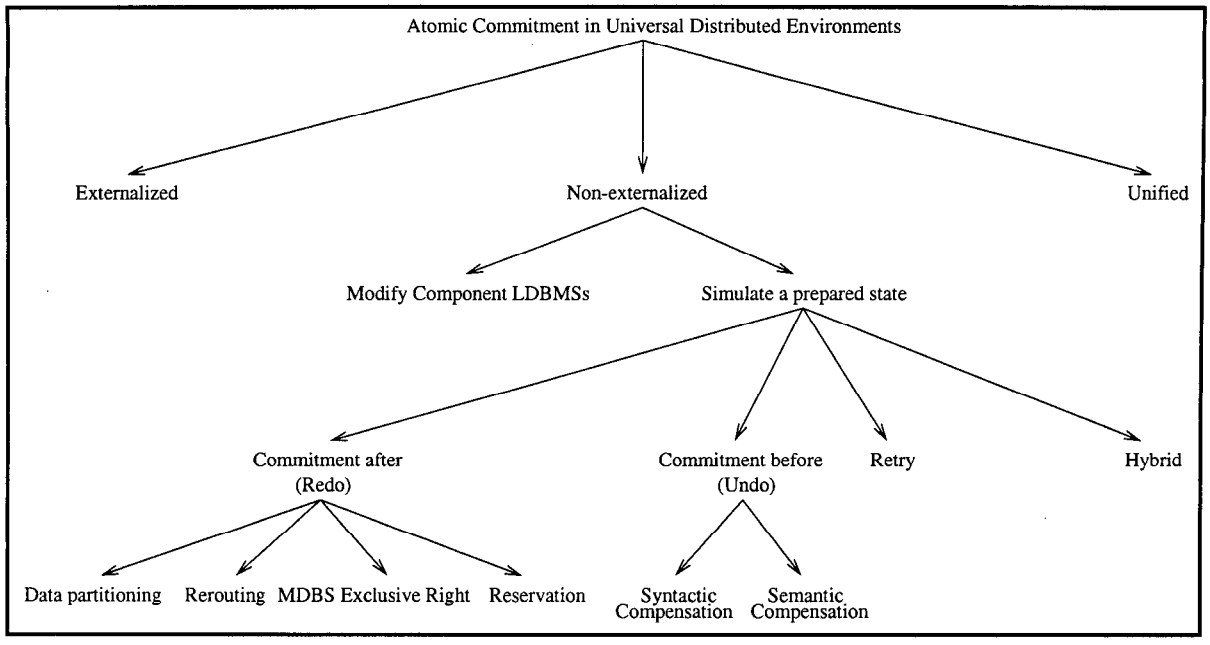


Figure 5: Taxonomy of atomic commitment in universal distributed environments.

can be classified as either *externalized* or *non-externalized* site [18]. An externalized site: (1) implements an ACP and (2) makes the system calls pertaining to its commit protocol, called *commit operators*, available to the outside word through its interface. Otherwise, the site is called non-externalized.

Figure 5 depicts three approaches that ensure the atomicity of global transactions in a form of a taxonomy. The taxonomy is based on the two categories of sites that we mentioned above. In what follows, we briefly discuss the motivation behind each of the three categories.

The motivation behind the works in the externalized approach is based on the assumption that future database sites will implement ACPs, which is well supported by the current standardization efforts. The challenge in this direction is to integrate database sites that use different and incompatible ACPs. The incompatibility of ACPs means that the semantics of the coordination messages and the actions of one ACP might be completely different than their counterparts in another ACP. Integrating incompatible ACPs is not a trivial task as it was previously believed [6, 18]. That is, it is not simply the case that once a database site supports an externalized ACP, it can be integrated with other database sites regardless of the used ACPs. The work reported in this paper fits in this research direction, highlighting one of these difficulties and proposing a practical solution.

Some researches have concentrated in resolving the incompatibility of ACPs with respect to the semantics of the coordination messages without considering their practical implications (e.g., [18]). Hence, this group of researchers were interested in achieving *functional* correctness. In the work reported in this paper, we looked at the incompatibility issue from a more pragmatic point of view. That is, achieving *operational* correctness in which, besides achieving functional correctness, the outcome of terminated transactions can be, eventually, forgotten without sacrificing consistency.

On the other hand, the motivation behind the work in the non-externalized approach is based on the fact that most existing database systems are legacy systems that do not externalize their ACPs. Thus, the challenge in this direction is to ensure the atomicity of transactions despite the fact that each site does not externalize an ACP. The methods reported in the literature can be classified into two categories, as shown in Figure 5. In the first category, it is suggested to modify each database to incorporate an ACP into it and to externalize the ACP to the outside world, while the in second category it is suggested to simulate a *prepared to commit* state. Some of the methods under simulated ACPs guarantee the traditional notion of atomicity while the others achieve a weaker correctness notion, called *semantic atomicity*. In semantic atomicity, the state of the database is not necessarily equivalent to the state of the database after some transaction is executed and finally aborted, whereas, in the traditional atomicity, the two states are equivalent.

The unified approach combines the other two categories since they complement each other and this category ensures the atomicity of transactions despite the diversity of the semantics of transactions and data.