Incompatibility Dimensions and Integration of Atomic Commit Protocols

Yousef J. Al-Houmaily
Department of Computer and Information Programs
Institute of Public Administration, Riyadh 11141, Saudi Arabia
email: {houmaily@ipa.edu.sa}

Abstract: *Advanced software application systems contain transactions that tend to traverse incompatible database sites belonging to different human organizations. One key requirement of these application systems is universal transactional support and, in particular, guaranteeing the atomicity property of transactions in the presence of incompatible atomic commit protocols (ACPs). Detailed analysis show that incompatibilities among ACPs could be due to the semantics of coordination messages or the presumptions about the outcome of terminated transactions. This leads to the definition of "operational correctness", a criterion that captures the practical integration of incompatible ACPs. It also leads to the definition of "safe state", a notion that determines the conditions under which all information pertaining to distributed transactions can be discarded without sacrificing their consistent termination across all participating sites. The significance of the analytical results is demonstrated through the development of a new ACP called "integrated two-phase commit" that integrates the most commonly known ACPs, with respect to applicability and performance, in a practical manner and in spite of their incompatibilities.*

Keywords: *Two-Phase Commit, Voting Protocols, Distributed Transaction Processing, Integrated Database Systems, Internet Transactions, Electronic Services and Electronic Commerce.*

1. Introduction

With the recent advances of Intranet and Internet technologies, there is a greater need than ever before to inter-operate different database sites in a practical and efficient manner. Such inter-operation is absolutely necessary towards supporting the *interoperability characteristic* of advanced database applications such as electronic services and 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, the *atomicity* property of transactions.

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 possible site and communication failures, that all sites participating in a transaction's execution reach the same final outcome for the transaction, i.e., to either commit or abort the transaction. Since commit processing consumes a substantial amount of a transaction's execution time [10] and ACPs are known to be blocking in case of failures [18], a variety of ACPs and optimizations have been proposed in the literature. Although the search for efficient ACPs has received much attention in the past decade and continue to be an important research topic for many environments including main memory databases (e.g., [15]), mobile database systems (e.g., [17]) and real-time databases (e.g., [12]), besides traditional (homogenous) distributed databases (e.g., [1, 21]); the issue of compatibility among ACPs did not receive as much attention in spite of its importance in advanced applications.

For the above reason, it is imperative to focus on the compatibility of ACPs in distributed database environments where the different database sites do not unanimously adopt the same ACP, such as multidatabase systems and the Internet. Section 2 presents the choice of protocols that are used to demonstrate the incompatibly issues while Section 3 shows that incompatibilities among ACPs could be due to (1) the semantics of the coordination messages (which include both their meanings as well as their existence), or (2) the presumptions about the outcome of terminated transactions in case of failures. Thus, in contrast to what was previously believed [7, 19], supporting a *visible prepared-to-commit* state is not sufficient for a practical integration of ACPs. This is because the outcome of some terminated transactions might have to be remembered forever, curtailing the system's operation on the long run. This leads to the definition of *operational correctness*, a criterion that captures, unlike *functional correctness*, the practical integration of incompatible ACPs. It also leads to the definition of *safe state*, a notion that determines the conditions under which all information pertaining to distributed transactions can be discarded without sacrificing their consistent termination across all participating sites.

The notion of safe state is expressed using ACTA [8], a first order predicate logic formalism. Although all ACPs can be specified and all theorems can be proven using ACTA by modeling log operations and system crashes as transactions' significant events[1], we choose to structure the proofs of the theorems along the lines of the proofs of ACPs in [5] for the sake of simplicity and ease of exposition. 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 it is using and, when it fails, it will, eventually, recover.

The significance of the analytical results is demonstrated through the development of a new ACP called *integrated two-phase commit* (I-2PC) which is presented in Section 4. I-2PC integrates the most commonly known APCs, with respect to performance and applicability, according to the operational correctness criterion. Section 4 also provides a prove of correctness to the new protocol. Section 5 summarizes the contributions of this paper with some concluding remarks.

2. Choice of ACPs

A distributed/Internet transaction accesses data located at different database sites. When the transaction finishes its execution and submits a "Commit" request, the *transaction manager* at the site where the transaction was initiated acts as the *coordinator* for the termination of the transaction across all participating sites. This is achieved by initiating an ACP such as the *basic two-phase commit* (2PC) protocol [11, 13], which is also called *presumed nothing* (PrN) [14]. In this paper, it is assumed that each database site implements an ACP that is not necessarily the same as the ACPs adopted by the other sites. Furthermore, it is assumed that the ACP adopted by a site can be either PrN, *presumed abort* (PrA) [16], *presumed commit* (PrC) [16] or *implicit yes-vote* (IYV)[2] [3]. The choice of these four protocols is because they are the best to demonstrate the dimensions of incompatibilities among ACPs that seem, at first glance, to be straight forward to interoperate and also because of the importance of these protocols which is as follows:

- PrN for historical reason since it is the first known and published ACP.
- PrA because it is currently part of the database standards [6, 20].
- PrC because of its performance advantage for committing transactions and the argument that favors it to become also part of the database protocol standards [4].
- IYV because of its performance advantages in high-speed networks that characterize today's computing environments.

3. Incompatibility Dimensions of ACPs

This section examines the compatibility of PrN, PrA, PrC and IYV by assuming that they can co-exist in a system and can be used together to commit a distributed/Internet transaction. As it shows, the incompatibilities of ACPs could be due to the semantics of their coordination messages or the presumptions that they make about the outcome of terminated transactions. The analysis of both of these dimensions is presented in the next two sections.

3.1. Message Semantics Incompatibilities

The incompatibilities that are due to the semantics of messages arise in two forms. The first one is due to the *meaning* of messages whereas, the second one, is due to the *existence* of messages. The differences between the two forms of incompatibilities are presented through two example protocols.

3.1.1. Meaning Incompatibilities

Assume that a coordinator follows its own protocol and does not realize any message out of its protocol. That is, it simply ignores any message that violates its protocol and interprets any message that it recognizes according to its own protocol. We call this type of integrated protocol used by a coordinator as *strict atomic commit* (SAC) protocol. In the examples below, a site follows SAC 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. Furthermore, assume that the coordinator and one of the participants employ PrA while the other participant employs IYV. Following IYV, when the participant executes an update operation, it acknowledges the operation with a message that contains the redo log records that were generated during the execution of the operation and enters an *implicit* prepared-to-commit state. The coordinator, following SAC, will recognize and interpret the message as only an acknowledgment for the successful execution of the operation without extracting the redo records contained in the message since this is not part of its protocol. At commit time of the transaction, the coordinator initiates the voting phase that will be recognized by the PrA participant but not the IYV participant. Based on that, the IYV participant will never send an explicit vote back to the coordinator since it employs a *one-phase commit* (1PC) protocol. In this scenario, the coordinator will timeout awaiting for the vote of the IYV participant and will abort the transaction. Thus, using SAC, no

[1] The basic two-phase commit protocol was specified and its important functional correctness aspects was shown using ACTA in [9].

[2] Autonomy implications on the constituent database sites are not discussed in this paper as it has been shown to be violated in one form or another in [9].

transaction that executes at an IYV participant will ever commit. Similar scenarios occur if the coordinator is using PrN or PrC and there is at least one IYV participant.

Now, assume that, instead of using a 2PC variant, the coordinator and one of the participants are using IYV while the other participant is using PrA. Furthermore, assume that the transaction has finished its execution and submitted its final commit primitive. Following IYV, the coordinator will commit the transaction since all the operations pertaining to the transaction have been executed at both participants and acknowledged. In this case, it force writes a commit record and sends commit messages to both participants. The IYV participant will recognize the message and commits the transaction whereas, the PrA participant will not recognize the commit message since it is out of its protocol and will ignore it. In this case, the coordinator will keep sending the final commit message to the PrA participant, according to IYV, forever, without getting an acknowledgment. On the other hand, the participant will keep ignoring these messages awaiting a prepare to commit message from the coordinator. Eventually, the PrA participant will timeout and abort the transaction, according to PrA. Thus, in this scenario, the atomicity of the transaction has been violated because it ended up committing at one site and aborting at the other. Similar scenarios occur if any participant in a transaction's execution uses PrN or PrC.

We reached the above two scenarios because the coordinator misinterpreted the meaning of the operations' acknowledgment messages. In the first scenario, the coordinator interpreted the meaning of an operation's acknowledgment received from the 1PC participant to *only* mean that the operation has been executed successfully without interpreting it to also mean that the participant has entered an *implicit* prepared-to-commit state. In the second scenario, the opposite happened. That is, the coordinator misinterpreted the meaning of an operation's acknowledgment received from the 2PC participant to mean that the participant has entered an implicit prepared-to-commit state while the participant is still in an active state. The above two scenarios can be generalized with the following theorem.

Theorem 1: It is impossible to achieve *global atomicity* if the coordinator is using SAC in the presence of transactions that execute at both 1PC and 2PC participants.

Proof: The proof proceeds by example and consists of two parts. The first is when the coordinator is using a 2PC variant while the second is when the coordinator is using 1PC.

Part I: Assume that the coordinator is using a 2PC variant and a transaction has executed at a 1PC participant. Furthermore, assume that all the transaction's operations have been executed successfully across all participants and acknowledged, and the coordinator decided to commit the transaction. In this case, the 1PC participant will not recognize the prepare to commit message of the voting phase and, consequently, will never send back an explicit vote in response to the prepare to commit message of the coordinator. Eventually, the coordinator will timeout and abort the transaction. Thus, no transaction will ever commit when the coordinator is using 2PC in the presence of 1PC participants.

Part II: Assume that the coordinator is using 1PC and a transaction has executed at a 2PC participant. Furthermore, assume that all the transaction's operations have been executed successfully across all participants and acknowledged, and the coordinator decided to commit the transaction. In this case, the 2PC participant will not recognize the commit message of the coordinator since it precedes the voting phase of the participant's protocol. Eventually, the participant will timeout awaiting the prepare to commit message and will abort the transaction. Thus, the atomicity of the transaction is violated since it ended up committing by its coordinator (and 1PC participants if any) and aborting at the 2PC participant. □

3.1.2. Existence Incompatibilities

This section demonstrates the incompatibilities that are due to the existence (i.e., absence vs. presence) of messages rather than their meaning. Assume that a coordinator follows its own protocol, "knows" and "understands" what messages to send and what messages to expect from each participant. Furthermore, assume that the coordinator handles any violations of its protocol with respect to extra or missing messages by simply ignoring such messages. We call this protocol used by a coordinator *participants' integrated protocol* (PIP). In the examples below, a site will follow PIP 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. Furthermore, assume that the coordinator and one of the participants are using PrC while the other participant is using IYV. Assuming that the coordinator knows the used protocol by each of the two participants and understands the meaning of their coordination messages, it will extract any redo log records contained in an acknowledgment form the IYV participant and record them in its log. The coordinator will also interpret the message to mean that the participant is in an implicit prepared-to-commit state. At the end of the transaction, in accordance to PrC, the coordinator will force write an *initiation* record and sends a prepare to commit message to *only* the PrC participant. This is because such a message is not within the IYV protocol. When the coordinator receives the vote of the PrC participant, the coordinator makes the final decision. Assuming a commit final decision, the coordinator will force write a commit final decision and then sends commit messages to both participants. However, the IYV 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, the coordinator will be able to forget about the transaction and discard all information pertaining to the transaction from its protocol table once it has written the commit final decision onto its stable log. The coordinator will be also able to garbage collect the transaction's log records when necessary. Since the coordinator employs PrC, it will respond to the inquiries of the participants in case of a failure with a commit final decision, using the PrC presumption.

Now, let us 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 IYV participant will never acknowledge the 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 it can discard the information from its protocol table that is kept in main memory. To alleviate this situation, knowing that the IYV participant will never acknowledge an abort decision, in PIP, the coordinator forgets the outcome of the transaction once it has received the acknowledgment of the PrC participant. In this case, the atomicity of the transaction might be violated. For example, if a failure occurs before the IYV participant has received the abort decision, the participant is left blocked and 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, before the failure, and forgotten about the transaction, it will wrongly respond with a commit final decision (using the PrC presumption) which clearly violates the atomicity of the transaction.

Similar situations occur if the coordinator employs PrN, PrA or IYV and some participants employ PrC while the others employ PrN, PrA or IYV. In these situations, the atomicity of committed transactions might be violated.

The above scenarios can be generalized with the following theorem.

Theorem 2: It is impossible to achieve *global atomicity* if the coordinator is using PIP in the presence of transactions that execute at participants that acknowledge only abort decisions and participants that acknowledge only commit decisions.

Proof: The proof proceeds by example and consists of four 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. The fourth is when the coordinator is using IYV.

Part I: Assume that the coordinator is using PrN and a transaction has executed at two participants one of which is using PrA whereas the other is using PrC. Furthermore, assume that coordinator decides to commit the transaction. 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) which violates the atomicity of the transaction.

Part II: Assume that a transaction 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. In this case, the PrA participant will acknowledge the decision but the PrC participant will not, as above. Now, it is possible for the PrC participant to fail before receiving the commit decision and for the 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.

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

Part IV: Assume that a transaction has executed at two participants one of which is using IYV whereas the other one is using PrC. Assume that the coordinator is using IYV. Furthermore, assume that the transaction has finished it execution at both participants successfully and the coordinator has received a "Yes" vote from the PrC participant. If the coordinator makes a commit final decision, the IYV participant will acknowledge the 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 to arrive after the coordinator has received the acknowledgment of the IYV participant and forgotten the transaction. In this case, the coordinator will respond with an abort decision (using the IYV presumption) which violates the atomicity of the transaction. □

3.2. Presumptions' Incompatibilities

Clearly, the PIP solution in which a coordinator "knows" and "understands" (i.e., "talks") the language of the protocols implemented by the different participants does not work. The PIP protocol might violate transaction atomicity because the coordinator forgets about transactions prematurely due to missing messages from some participants. Let us consider an alternative integrated protocol, called *coordinator integrated protocol* (CIP) which behaves similar to PIP. However, unlike PIP, a coordinator in CIP never forgets a transaction until it has received all necessary messages.

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 pertaining to some terminated transactions from both its protocol table and stable log. Since these terminated transactions when they are forgotten might lead to a wrong presumption (as seen in PIP), CIP does not lead to atomicity violations by requiring a coordinator to always remember the outcome of these transactions and never uses its presumption after a failure. Thus, even though CIP 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 [2]:

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

Theorem 3: It is impossible to achieve *operational correctness* if the coordinator is using CIP in the presence of transactions that execute at participants that adopt ACPs with contradicting presumptions about terminated transactions.

Proof: The proof proceeds by example and consists of four 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. The fourth is when the coordinator is using IYV.

Part I: Assume that the coordinator is using PrN and that a transaction has executed at two participants one of which is using PrA whereas the other is using PrC. Furthermore, assume that coordinator decides to commit 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.

Part II: Assume that a transaction 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. In this case, the PrA participant will acknowledge the decision but the PrC participant will not, as above. Hence, the coordinator will not be able to write an end log record and has to remember the transaction forever.

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

Part IV: Assume that a transaction has executed at two participants one of which is using IYV whereas the other one is using PrC. Assume that the coordinator is using IYV. Furthermore, assume that the transaction has finished it execution at both participants successfully and the coordinator has received a “Yes” vote from the PrC participant. If the coordinator makes a commit final decision, the IYV participant will acknowledge the 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. □

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. This is called a *safe state* [2]. 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 (as in PrA, PrC and IYV).

Thus, in order to integrate protocols that adopt contradicting presumptions in a practical manner, we need a safety criterion that determines the conditions under which a coordinator can reach a safe state in which only a single presumption that is consistent with a transaction’s final outcome holds. The following safety criterion satisfies this requirement. It is expressed using ACTA [8], a first order predicate logic with a precedence relation ($\rightarrow$) in the *execution history* (H). *H* represents the complete history of the execution of a transaction until it is either committed or aborted across all participating sites. In the definition below, *C* denotes the coordinator of the transaction. The predicate $\alpha \rightarrow \beta$ is true if event α *precedes* event β in *H*. It is false, otherwise. Here, $Decide_C(Abort_T)$ denotes that the coordinator decides to abort a transaction T and $Decide_C(Commit_T)$ denotes that the coordinator decides to commit T. $DeletePT_C(T)$ 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)

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

The above definition states that a coordinator is in a safe state with respect to 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). Thus, 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.

4. The Integrated Two-Phase Commit

This section presents I-2PC that integrates PrN, PrA, PrC and IYV according to the operational correctness criterion that is defined above. The basic philosophy behind the design of I-2PC is to resolve the incompatibilities that are due to the semantics of messages as in CIP and, at the same time, to allow a coordinator to reach a safe state with respect to the outcome of terminated transactions without having to remember them forever.

According to the behavior of PrN, PrA, PrC and IYV, a coordinator expects those participants that employ PrN, PrA and IYV to acknowledge commit final decisions but not those participants that employ PrC. Similarly, a coordinator expects those participants that employ PrN and PrC to acknowledge abort final decisions but not those participants that employ PrA and IYV. Based on the behavior of the four protocols, a coordinator, in I-2PC, forgets a committed transaction when PrN, PrA and IYV participants acknowledge the commit decision. For an abort decision, a coordinator forgets an aborted transaction when PrC participants acknowledge the abort decision[3]. Thus, I-2PC behaves similar to PIP with respect to the timing at which it forgets about the outcome of terminated transactions.

However, unlike PIP, a coordinator in I-2PC, instead of using a *single* presumption for all terminated transactions, which is the case in all presumption-based ACPs, the presumption used by the coordinator (in the absence of information) depends on the protocol used by the inquiring participant. That is, if the inquiring participant is abort-based, the presumption of the coordinator is abort. On the other hand, if the inquiring participant is commit-based, the presumption is commit. In this way, the presumption of the coordinator always matches the actual final outcome of a forgotten terminated transaction.

The next section presents the details of I-2PC during normal processing. Then, Section 4.2 discusses the recovery aspects of I-2PC in case of failures and prove its correctness.

4.1. I-2PC During Normal Processing

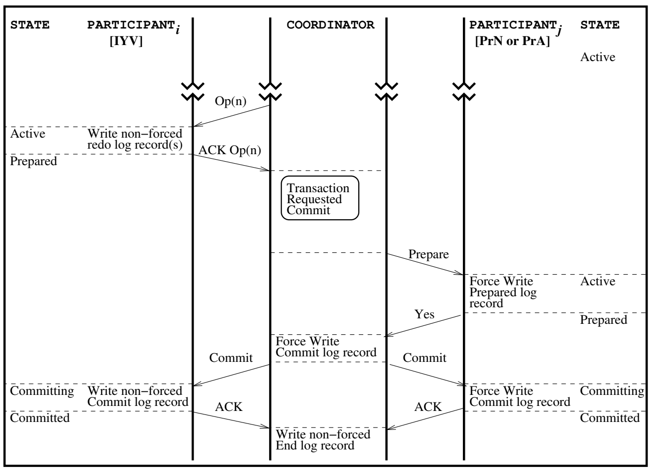
In I-2PC, a coordinator records the 2PC protocol employed by each participant in a table called *participants' commit protocol* (PCP). The PCP table is kept onto stable storage and is updated when a new site joins or leaves the distributed environment. Only a portion of the PCP table, called *active participants' protocols* (APP), is maintained in main memory, containing the identities of the participants with active transactions.

Once the coordinator of a transaction has identified a participating site for the execution of the transaction, it checks its protocol table. If the identity of the participant is not in the protocol table, the coordinator adds the identity of the participant into the table. Then, it forwards the operation to the participant for execution.

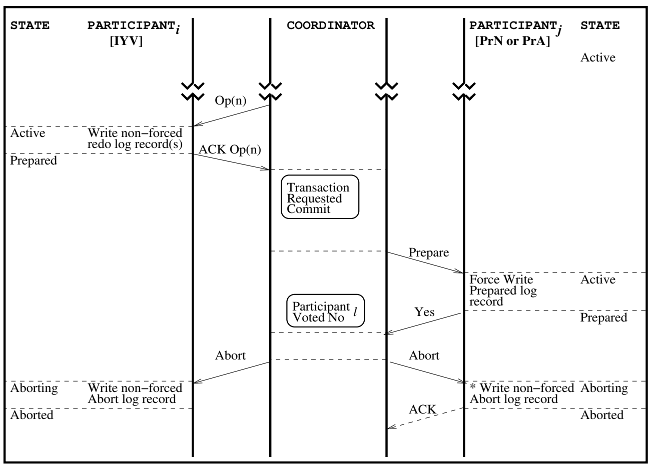
If the coordinator receives either an abort request from a transaction or a negative acknowledgment from any participant, it aborts the transaction. In this case, the coordinator discards all information pertaining to the transaction from its protocol table without writing a decision log record for the transaction. Then, the coordinator sends an abort message to each participant that has acknowledged the processing of all the transaction's operations successfully.

On the other hand, when the coordinator of a transaction receives a commit primitive from the transaction, it waits for the acknowledgments of the transaction's pending operations and then checks its APP to determine which protocol to use for the termination of the transaction. The coordinator selects PrN if all the participants are using PrN. Similarly, it selects PrA if all the participants are using PrA whereas, it selects PrC if all the participants are using PrC. If all participants are using IYV, the coordinator selects IYV.

[3] Although PrN treats transactions uniformly during normal processing regardless of whether they are to be finally committed or aborted, there is a hidden presumption in PrN by which it considers all active transactions as aborted in case of a failure. For this reason, there is no need for an abort acknowledgment from a PrN participant in I-2PC.


STATE
PARTICIPANT$_i$ [IYV]
COORDINATOR
PARTICIPANT$_j$ [PrN or PrA]
STATE
Active
Op(n)
Active
Write non-forced redo log record(s)
ACK Op(n)
Prepared
Transaction Requested Commit
Prepare
Force Write Prepared log record
Active
Yes
Prepared
Force Write Commit log record
Commit
Commit
Committing
Write non-forced Commit log record
Force Write Commit log record
Committing
ACK
ACK
Committed
Committed
Write non-forced End log record

(a) Commit Case.


STATE
PARTICIPANT$_i$ [IYV]
COORDINATOR
PARTICIPANT$_j$ [PrN or PrA]
STATE
Active
Op(n)
Active
Write non-forced redo log record(s)
ACK Op(n)
Prepared
Transaction Requested Commit
Prepare
Force Write Prepared log record
Active
Participant Voted No l
Yes
Prepared
Abort
Abort
Aborting
Write non-forced Abort log record
* Write non-forced Abort log record
Aborting
ACK
Aborted
Aborted

* This record is forced written in the case of a PrN participant.

⇠ - - - An ACK message is sent back by a PrN participant which is ignored by the coordinator.

(b) Abort Case.

Figure 1: I-2PC in the absence of contradicting presumptions.

In the event of protocols' mix, the coordinator selects I-2PC. By using I-2PC, there are two cases to consider. The first one is when the protocols used by the participants have the same presumptions about the outcome of terminated transactions. This case occurs when the participants are mixed PrN, PrA and IYV. These three protocols adopt the abort presumption of terminated transactions. The second case is when the used protocols' mix has contradicting presumptions about the outcome of terminated transactions. This case occurs when the participants' mix contains a PrC participant.

4.1.1. Absence of Contradicting Presumptions

When the used protocols by the participants have the same presumption about the outcome of terminated transactions, the coordinator sends a prepare to commit message to each 2PC participant (i.e., each PrN and PrA participant), as shown in Figure 1. When a 2PC participant receives a prepare to commit message, it validates the transaction and then sends back its vote. If the transaction can be committed, the participant force writes a prepared log record and then sends back its "Yes" vote, following either PrN or PrA used by the participant. Otherwise, the participant aborts the transaction and sends back a "No" vote without writing any log records.

When the coordinator receives the votes of 2PC participants, the coordinator makes the final decision. The decision is commit if each IYV participant is in an implicit prepared-to-commit state and each 2PC participant is in an explicit prepared-to-commit state. Otherwise, the decision is abort.

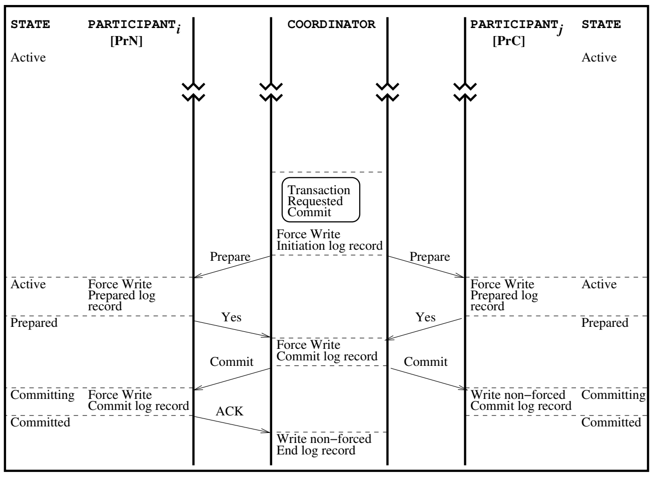
On a commit decision (Figure 1 (a)), the coordinator force writes a commit log record, that includes the identities of all participating sites, and sends out commit messages. When a 2PC participant receives a commit message, it commits the transaction, force writes a commit log record and then, acknowledges the commit decision. When a 1PC participant receives a commit message, it commits the transaction, writes a non-forced commit log record and, when the commit record is flushed onto the stable log, it sends back an acknowledgment. Once the coordinator has received acknowledgments from all participating sites, it writes a non-forced end log record and forgets the transaction.

On an abort decision (Figure 1 (b)), assuming that some 2PC participant (*l*) has voted "No", the coordinator sends out abort messages to IYV participants and each 2PC participant that has voted "Yes" and forgets the transaction without writing any log records. When an IYV or PrA participant receives an abort message, it complies with the decision and writes a non-forced abort log record. On the other hand, when a PrN participant receives an abort message, following PrN, it complies with the decision, force writes an abort log record and sends back an acknowledgment. When the coordinator receives an acknowledgment from a PrN, it simply ignores the message, knowing that it has no effect on the protocol correctness (as we show in Section 4.2).

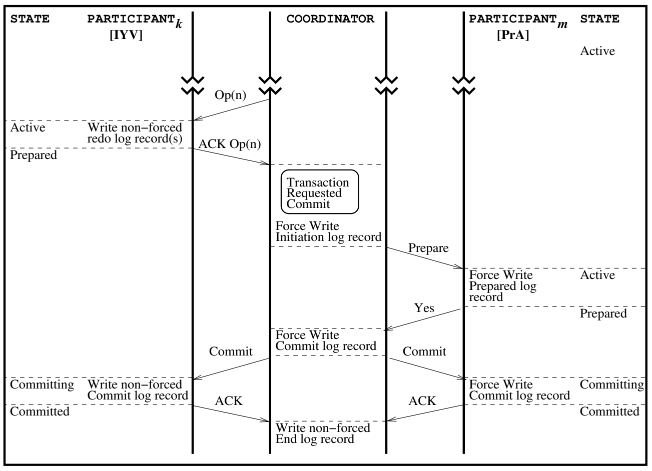
4.1.2. Presence of Contradicting Presumptions

When the used protocols by the participants have contradicting presumptions about the outcome of terminated transactions (i.e., there is at least one PrC participant), the coordinator force writes an *initiation* log record, that includes the identities of all participants, and then, sends a prepare to commit message to each 2PC participant (i.e., each PrN, PrA and PrC participant), as shown in Figure 2. When a 2PC participant receives a prepare to commit message, it validates the transaction and then sends back its vote. If the transaction can be committed, the participant force writes a prepared log record and then sends back its "Yes" vote, following either PrN, PrA or PrC used by the participant. Otherwise, the participant aborts the transaction and sends back a "No" vote without writing any log records.

When the coordinator receives the votes of 2PC participants, the coordinator makes the final decision.


STATE
PARTICIPANT$_i$ [PrN]
COORDINATOR
PARTICIPANT$_j$ [PrC]
STATE
Active
Active
Transaction Requested Commit
Force Write Initiation log record
Prepare
Prepare
Active
Force Write Prepared log record
Force Write Prepared log record
Active
Prepared
Yes
Yes
Prepared
Force Write Commit log record
Commit
Commit
Committing
Force Write Commit log record
Write non-forced Commit log record
Committing
Committed
ACK
Committed
Write non-forced End log record

(a) PrN and PrC participants in I-2PC.


STATE
PARTICIPANT$_k$ [IYV]
COORDINATOR
PARTICIPANT$_m$ [PrA]
STATE
Active
Op(n)
Active
Write non-forced redo log record(s)
ACK Op(n)
Prepared
Transaction Requested Commit
Force Write Initiation log record
Prepare
Force Write Prepared log record
Active
Yes
Prepared
Force Write Commit log record
Commit
Commit
Committing
Write non-forced Commit log record
Force Write Commit log record
Committing
ACK
ACK
Committed
Committed
Write non-forced End log record

(b) IYV and PrA participants in I-2PC.

Figure 2: Committing in presence of contradicting presumptions.

The decision is commit if each IYV participant is in an implicit prepared-to-commit state and each 2PC participant is in an explicit prepared-to-commit state. Otherwise, the decision is abort.

On a commit decision (Figure 2), the coordinator force writes a commit log record and then sends out commit messages. When a PrN or PrA participant receives a commit message, it commits the transaction, force writes a commit log record and then, sends back an acknowledgment. When a PrC participant receives the commit decision, it commits the transaction, writes a non-forced commit log record without sending an acknowledgment back to the coordinator (following PrC protocol). When an IYV participant receives a commit message, it commits the transaction, writes a non-forced commit log record and, when the commit record is flushed onto the stable log, it sends back an acknowledgment. Once the coordinator receives "commit" acknowledgments from all sites employing abort-based presumption protocols, the coordinator writes a non-forced end log record and forgets the transaction.

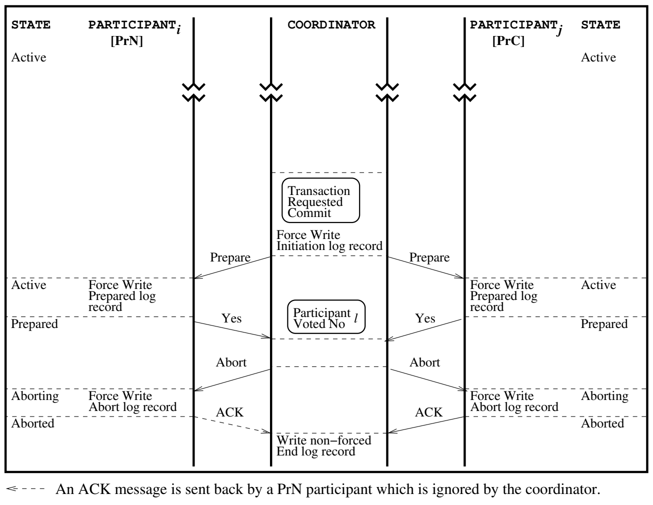
On an abort decision (Figure 3), once again assuming that some 2PC participant (*l*) has voted "No" during the voting phase, the coordinator sends out an abort message to each prepared-to-commit participant (whether implicitly or explicitly) without writing an abort log record. When an IYV or PrA participant receives an abort message (Figure 3 (b)), it complies with the decision and writes a non-forced abort log record. On the other hand, when a PrN participant receives an abort message (Figure 3 (a)), following PrN, it complies with the decision, force writes an abort log record and sends back an acknowledgment. When a PrC participant receives the abort decision, it aborts the transaction, force writes an abort log record and then, sends back an acknowledgment.

When the coordinator receives an acknowledgment from a PrN, it simply ignores the message, knowing that it has no effect on the protocol correctness (as we show in the next section). On the other hand, when the coordinator receives "abort" acknowledgments from all PrC participants, it writes a non-forced end log record and forgets the transaction.

4.2. Recovery and Correctness of I-2PC

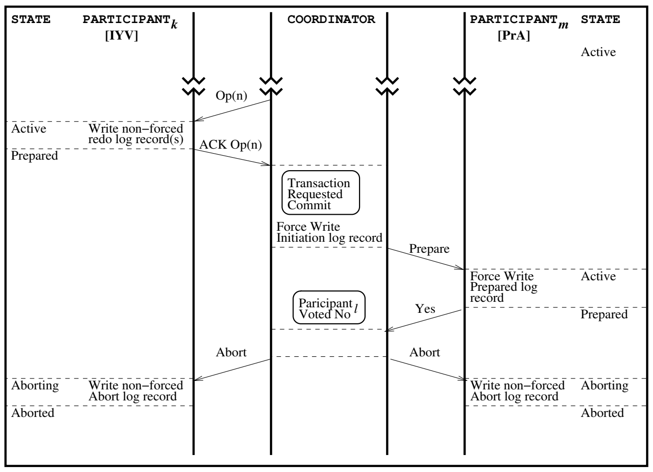
As in all other commit protocols, communication and site failures are detected by timeouts. The recovery procedure in case of communication and participants' failures are handled in a manner similar to the way they are handled in PrN, PrA, PrC and IYV protocols. According to the behavior of PrN, PrA, PrC and IYV, the coordinator expects those participants that employ PrN, PrA and IYV to acknowledge commit final decisions but not those participants that employ PrC. Based on the that, the coordinator forgets about the outcome of a committed transaction once the PrN, PrA and IYV participants acknowledge the commit decision, knowing that only a participant that uses PrC might inquire about the decision in the future. If a PrC participant inquires about a *forgotten* commit decision, the coordinator, knowing that the participant uses PrC, will direct the participant to commit the transaction using the presumption of PrC employed by the participant. This is accomplished by the coordinator even without examining its stable log.

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 and IYV. Since the coordinator does not wait for, or even consider, the acknowledgments of PrN participants when writing the end log record for an aborted transaction, the coordinator forgets about the outcome of such a transaction once the PrC participants acknowledge the abort decision. Hence, besides PrA and IYV participants, PrN participants might inquire about a *forgotten* abort decision. In this case, the coordinator, knowing that only a participant that uses an abort-based protocol (i.e., PrN, PrA or IYV) might inquire about the decision, it will direct the participant to abort the transaction using the abort presumption of these three protocols. Again, this is



STATE
PARTICIPANT$_i$ [PrN]
COORDINATOR
PARTICIPANT$_j$ [PrC]
STATE
Active
Active
Transaction Requested Commit
Force Write Initiation log record
Prepare
Prepare
Active
Force Write Prepared log record
Force Write Prepared log record
Active
Prepared
Yes
Participant$_l$ Voted No
Yes
Prepared
Abort
Abort
Aborting
Force Write Abort log record
Force Write Abort log record
Aborting
Aborted
ACK
ACK
Aborted
Write non−forced End log record
An ACK message is sent back by a PrN participant which is ignored by the coordinator.

(a) PrN and PrC participants in I-2PC.



STATE
PARTICIPANT$_k$ [IYV]
COORDINATOR
PARTICIPANT$_m$ [PrA]
STATE
Active
Op(n)
Active
Write non−forced redo log record(s)
ACK Op(n)
Prepared
Transaction Requested Commit
Force Write Initiation log record
Prepare
Force Write Prepared log record
Active
Participant$_l$ Voted No
Yes
Prepared
Abort
Abort
Aborting
Write non−forced Abort log record
Write non−forced Abort log record
Aborting
Aborted
Aborted

(b) IYV and PrA participants in I-2PC.

Figure 3: Aborting in presence of contradicting presumptions.

accomplished by the coordinator even without having to examine its stable log.

Thus, in I-2PC, when a participant inquires a coordinator about the final outcome of a forgotten transaction, the coordinator, not remembering the transaction, it *infers* the transaction's outcome from the presumption used in the inquiring participant's protocol. This inference of decisions is always consistent with the actual final outcome of forgotten transactions.

The next section thoroughly analyzes all possible scenarios of communication failures whereas Section 4.2.2 analyzes the recovery aspects of a coordinator's site failure. On the other hand, participants' site failures are not discussed since they are handled in a manner similar to the way they are handled in PrN, PrA, PrC and IYV, depending on the protocol adopted by each participant

4.2.1. Communication Failures

There are four points during the execution of I-2PC where a communication failure might occur while a site is waiting for a message. The first point is when the coordinator of a transaction has sent an operation for execution at a participant's site and is waiting for an operation acknowledgment from the participant. In this case, the coordinator aborts the transaction and sends out abort messages to the rest of the participants. Similarly, a participant aborts a transaction when a communication failure occurs and the participant has a pending operation's acknowledgment. Notice that the coordinator of a transaction may commit the transaction in spite of communication failures with some participants as long as these participants are IYV participants and have no pending operations' acknowledgments.

The second point is when a participant has no pending operation acknowledgment. If the participant is using a 2PC variant, it aborts the transaction. On the other hand, if the participant is using IYV, in accordance to IYV, the participant is left blocked until communication is re-established with the coordinator. Then, the participant inquires the coordinator about the transaction's status. If the coordinator has already committed the transaction, it must have been waiting for the commit acknowledgment of the participant. Based on that, the coordinator replies with a commit message and waits for an acknowledgment. If the coordinator has aborted the transaction and still remembers it (i.e, the transaction is still in the protocol table), the coordinator replies with an abort decision. If the coordinator does not remember the transaction, it means that the coordinator must have aborted the transaction. In this case, it replies with an abort message using the presumption of IYV, which is the presumption used in the protocol of the inquiring participant. If the transaction is still active in the system, the coordinator replies with a *still active* message, following IYV protocol. When the participant receives a final decision, the participant enforces the decision and writes a non-forced decision (i.e., commit or abort) log record. Then, if the decision is commit, the participant also acknowledges the decision (after the decision is written onto the stable log). If the participant receives a still active message, the participant waits for further operations.

The third point is when a coordinator is waiting for the votes of 2PC participants. In this case, the coordinator treats communication failures as "No" votes and aborts the transaction. As during normal processing, once the coordinator has aborted a transaction, it sends out abort messages to all accessible participants and waits for the required acknowledgments. For an inaccessible participant, the participant is left blocked if has voted "Yes" before the communication failure and it is the responsibility of the participant to inquire about the transaction's status after the failure is fixed. If the coordinator receives an inquiry message after the failure has been fixed, the coordinator either still remembers the aborted transaction (because the transaction has an initiation record in its protocol table and some participants are using PrC) or it has aborted and forgotten the transaction. In the former case, the coordinator sends back an abort message. It also waits for an acknowledgment if the participant is using PrC. Once the participant has received the abort message, it aborts the transaction and sends back an acknowledgment only if it uses PrC. In the latter case, since the coordinator does not remember the transaction and the transaction has been aborted, it means that the inquiring participant must be a PrN or PrA participant. Based on that, the coordinator replies with an abort message using the presumption of PrN or PrA, which is the presumption used in the protocol of the inquiring participant.

The fourth point is when the coordinator of a transaction is waiting for the acknowledgments of a final decision. Since the coordinator needs the acknowledgments in order to discard the information pertaining to the transaction from its protocol table and its log (during the garbage collection procedure), it re-sends the decision to the appropriate participants once communication failures are fixed. That is, if the decision is commit, the coordinator re-sends the decision to each inaccessible PrN, PrA and IYV participant. On the other hand, if the decision is abort, the coordinator re-sends the decision to each inaccessible PrC participant. When an IYV participant receives a commit decision after a failure, it either acknowledges the decision if it has already received and enforced the decision prior to the failure (i.e., the participant has no recollection about the transaction), or enforces the decision, writes a non-forced commit log record and then sends back an acknowledgment (after the decision is written onto the stable log). Similarly, when a 2PC participant receives a decision, it either acknowledges the decision if it has already received and enforced the decision prior to the failure, or enforces the decision, force writes a decision record and then acknowledges the decision. Once the coordinator has received the required acknowledgments, it writes an end log record, as during normal processing, and forgets the transaction.

4.2.2. Coordinator's Failure

Upon a coordinator's restart after a failure, the coordinator re-builds its protocol table by scanning its stable log. The coordinator needs to complete the commit protocol for each incomplete transaction. Hence, it needs to consider only the following transactions during its recovery procedure:

1. Each transaction with an initiation log record but without a corresponding commit and end records - the coordinator knows that either PrC or I-2PC (with contradicting presumptions) was used for the commit processing of the transaction. In either case, the coordinator considers the transaction aborted and sends an abort message to each PrC participant recorded in the initiation record and waits for acknowledgments.
2. Each transaction with an initiation log record and a commit record but without an end record - the coordinator knows that either PrC or I-2PC (with contradicting presumptions) was used for the commit processing of the transaction. Based on the identities of the participants recorded in the initiation record, if I-2PC was used, the coordinator sends commit messages to all participants recorded in the initiation record except those using PrC and waits for acknowledgments.
3. Each transaction with a commit record but without an initiation and an end record - the coordinator knows that either PrN, PrA, IYV or I-2PC (without contradicting presumptions) was used for the commit processing of the transaction. In either case, the coordinator sends a commit message to each participant recorded in the commit decision record and waits for acknowledgments.

In all the three cases above, when a participant receives a decision message, it either acknowledges the message if it has already received and enforced the decision prior to the failure, or enforces the decision, writes the required log record and then sends back an acknowledgment. Once the coordinator receives the required acknowledgments for a transaction, it writes an end log record an forgets the transaction.

For all other transactions, the coordinator can safely ignore them during its recovery procedure and considers them completed transactions. If a participant inquires about a transaction that has been considered completed by the coordinator, *regardless of the protocol used for the termination of the transaction*, the coordinator, not remembering the transaction, it replies with a decision that matches the presumption used in the protocol of the inquiring participant (as recorded in the PCP).

4.2.3. Proof of Correctness

The above discussion provides an iterative method that prove the correctness of I-2PC in the presence of site and communication failures. That is, it enumerates all possible points of site and communication failures during the course of the protocol and shows how to deal with them. This leads to the following theorem.

Theorem 4: The I-2PC protocol satisfies the operational correctness criterion.

Proof: To show the correctness of I-2PC according to operational correctness, we need to show that all the three requirements of operational correctness are satisfied. 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, as we iteratively proven in the previous two sections. The only remaining requirement that needs to be proven is the second one which requires that the coordinator should eventually be able to forget about the outcome of transactions. I-2PC also satisfies this requirement because a transaction is forgotten once all required acknowledgments arrive from the participants. What we need to prove is that a coordinator never sacrifices the consistency of its decisions even though it might be using different protocols for the termination of different transactions (i.e., I-2PC, PrN, PrA, PrC or IYV). We prove this part by considering the two possible outcome of transactions. For the prove of this part, we recall that, in the absence of information, a coordinator of a transaction always uses the presumption adopted by the protocol of the inquiring participant. This is regardless of the actual protocol that has been used for the termination of the transaction. The prove proceeds by contradiction.

Commit Case: Assume that the coordinator has made a commit decision and after forgetting the transaction, it replies to an inquiry message with an abort decision.

If the inquiring participant is PrC, 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 either a PrN, PrA or IYV participant which is impossible since all PrN, PrA and IYV participants must have acknowledged the commit decision in order for the coordinator to forget the outcome of the transaction.

Abort Case: Assume that the coordinator has made an abort decision and after forgetting the transaction, it replies to an inquiry message with a commit decision.

If the inquiring participant is PrN, PrA or IYV, the coordinator will use the abort presumption and will respond with an abort decision which contradicts the initial assumption.

In order to reply with a commit, it means that the coordinator has used the commit presumption. This means that the message is from a PrC participant which is impossible since all PrC participants must have acknowledged the abort decision in order for the coordinator to forget the outcome of the transaction. □

5. Conclusion

With the current advances in Internet applications, it is imperative to support universal transactional access and, in particular, guaranteeing the atomicity property of transactions in the presence of incompatible *atomic commit protocols* (ACPs). Detailed analysis showed the dimensions of incompatibilities among ACPs. Then, the significance of the analytical results was demonstrated through the development of a new ACP called "integrated two-phase commit" (I-2PC) that integrates the most commonly known ACPs, with respect to applicability and performance, in a practical manner and in spite of their incompatibilities.

The results of this work should help in a better understanding to atomicity in heterogeneous environments where the different database sites do not unanimously adopt the same ACP. It should also stimulate the development of new and more flexible methods that support the interoperability characteristic of today's software application systems especially for those emerging environments such as mobile database systems and e-government.

References

[1] Al-Houmaily, Y., "On Interoperating Incompatible Atomic Commit Protocols in Distributed Databases," *in Proc. of the 1st IEEE Int'l Conf. on Computers, Communications, and Signal Processing*, Nov. 2005.

[2] Al-Houmaily, Y. and P. Chrysanthis, "Atomicity with Incompatible Presumptions," *in Proc. of the 18th ACM PODS*, May 1999.

[3] Al-Houmaily, Y. and P. Chrysanthis, "An Atomic Commit Protocol for Gigabit-Networked Distributed Database systems," *Journal of Systems Architecture*, vol. 46, pp. 809-833, 2000.

[4] Al-Houmaily, Y., P. Chrysanthis and S. Levitan, "An Argument in Favor of the Presumed Commit Protocol," *in Proc. of the 13th ICDE*, April 1997.

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

[6] Braginski, E., "The X/Open DTP Effort," *in Proc. of the 4th Int'l Workshop on HPTS*, Asilomar, California, 1991.

[7] Breitbart, Y., H. Garcia-Molina, and A. Silberschatz, "Overview of Multidatabase Transaction Management," *VLDB Journal*, vol. 1, no. 2, October 1992.

[8] Chrysanthis, P. and K. Ramamritham, "Synthesis of Extended Transaction Models using ACTA," *ACM TODS*, vol. 19, no. 3, pp. 450-491, September 1994.

[9] Chrysanthis, P. and K. Ramamritham, "Autonomy Requirements in Heterogeneous Distributed Database Systems," *in Proc. of the 6th Int'l Conf. on Management of Data*, pp. 283-302, December 1994.

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

[11] Gray, J., "Notes on Data Base Operating Systems," in Bayer R., R.M. Graham, and G. Seegmuller (Eds.), *Operating Systems: An Advanced Course*, LNCS, vol. 60, pp. 393-481, Springer-Verlag, 1978.

[12] Haritsa, J., K. Ramamritham and R. Gupta, "The PROMPT Real-Time Commit Protocol," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no, 2, 2000.

[13] Lampson, B. "Atomic Transactions," in B. Lampson (Ed.), *Distributed Systems: Architecture and Implementation - An Advanced Course*, LNCS, vol. 105, pp. 246-265, Springer-Verlag, 1981.

[14] Lampson, B. and D. Lomet, "A New Presumed Commit Optimization for Two Phase Commit," *in Proc. of the 19th Int'l Conf. on VLDB*, pp. 630-640, Aug. 1993.

[15] Lee, I. and H. Yeom. "A Single Phase Distributed Commit Protocol for Main Memory Database Systems," *6th Int'l Parallel and Distributed Processing Symposium* (IPDPS), April 2002, Fort Lauderdale, FL, USA, 2002.

[16] Mohan, C., B. Lindsay and R. Obermarck, "Transaction Management in the R^* Distributed Data Base Management System," *ACM TODS*, vol. 11, no. 4, pp. 378-396, Dec. 1986.

[17] Nouali, N., H. Drias and A. Doucet, "A Mobility-Aware Two-Phase Commit Protocol," *Int'l Arab Journal of Information Technology,* vol. 3, no. 1, 2006.

[18] Skeen, D. and M. Stonebraker, "A Formal Model of Crash Recovery in a Distributed System," *IEEE TSE*, vol. SE-9, no. 3, May 1983.

[19] Tal, A. and R. Alonso, "Integration of Commit Protocols in Heterogeneous Databases," *Distributed and Parallel Databases*, vol. 2, no. 2, pp. 209-234, Apr. 1994.

[20] Upton IV, F., "OSI Distributed Transaction Processing, An Overview," *in Proc. of the 4th Int'l Workshop on HPTS*, Asilomar, CA, Sep. 1991.

[21] Yu, W. and C. Pu, "A Dynamic Two-Phase Commit Protocol for Adaptive Composite Services," *Int'l Journal of Web Services Research*, vol. 4, no. 1, 2007.

Yousef J. Al-Houmaily received a B.S. in Computer Engineering from King Saud University, Riyadh, Saudi Arabia in 1986, a M.S. in Computer Science from George Washington University, Washington D.C. in 1990, and a Ph.D. in Computer Engineering from the University of Pittsburgh, Pittsburgh, Pennsylvania in 1997.

Currently, Dr. Al-Houmaily is an Assistant Professor at the Department of Computer and Information Programs, Institute of Public Administration, Riyadh, Saudi Arabia. Dr. Al-Houmaily was the Director of Computer and Information Programs at the Institute of Public Administration from November 1997 till June 2002. He also served (or currently serving) on a number of important councils and committees at the Institute of Public Administration including the Academic Council, the Permanent Academic Promotion Committee, the Educational Staff Recruitment Committee and the Permanent Committee for Information and Technology. On the Saudi national level, he served on a number of committees of which his membership in the General Commission for the Ministerial Committee for Administrative Organization. Dr. Al-Houmaily worked as a consultant to a number of government agencies in Saudi Arabia including the Deputy Prime Minister's Office, the Control and Investigation Board and the General Presidency for Girls Education. From July, 2005 till June, 2006 he was a Visiting Assistant Professor at the David R. Cheriton School of Computer Science, University of Waterloo, Ontario, Canada, where he taught database courses at both the undergraduate and graduate levels.

Dr. Al-Houmaily's current research interests are in the areas of database systems, mobile distributed computing systems and sensor networks. His publication record contains over twenty scientific articles and two books (that are written in the Arabic language). Dr. Al-Houmaily's professional activities include his membership to a number of professional organizations and his services as a referee and a reviewer to several research centers, international conferences and international journals. Besides that, he frequently serves on the Program Committees of international conferences.