An intelligent adaptive participant's presumption protocol for atomic commitment in distributed databases

Yousef J. Al-Houmaily

Department of Computer and Information Programs,
Institute of Public Administration,
Riyadh 11141, Saudi Arabia
E-mail: houmaily@ipa.edu.sa
E-mail: ysf_al@yahoo.com

Abstract: The objective of the original 'adaptive participant's presumption protocol' (AP^3) was to achieve the performance advantages of one-phase commit protocols, whenever possible, and, at the same time, the wide applicability of two-phase commit protocols. Specifically, AP^3 operates as a one-phase commit protocol for the commit processing of a transaction so long as the transaction is not associated with the validation of any deferred consistency constraints. But, when a transaction is associated with the validation of deferred consistency constraints that need to be synchronised at commit time of the transaction, AP^3 dynamically switches to the most appropriate two-phase commit variant. However, providing further intelligence to enhance the performance and applicability of the original AP^3 is expected to make it a highly appealing choice for adoption in the design of future distributed database management systems. This is the essence of the 'intelligent AP^3' that extends the basic AP^3 in two directions: *efficiency and applicability*.

Keywords: atomic commit protocols; atomicity; database management systems; database recovery; integrated database systems; one-phase commit; 1PC.

Reference to this paper should be made as follows: Al-Houmaily, Y.J. (2013) 'An intelligent adaptive participant's presumption protocol for atomic commitment in distributed databases', *Int. J. Intelligent Information and Database Systems*, Vol. 7, No. 3, pp.242–277.

Biographical notes: Yousef J. Al-Houmaily received his BSc in Computer Engineering from King Saud University, Saudi Arabia in 1986, MSc in Computer Science from George Washington University, Washington DC in 1990, and PhD in Computer Engineering from the University of Pittsburgh in 1997. He is currently an Associate Professor in the Department of Computer and Information Programs at the Institute of Public Administration, Riyadh, Saudi Arabia, where he was the Director from November 1997 until June 2002. He provided consultation services 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 until June 2006, he was a Visiting Assistant Professor at the School of Computer Science, University of Waterloo, Canada. His current research interests are in the areas of database management systems, mobile distributed computing systems and sensor networks.

1 Introduction

An *atomic commit protocol* (ACP) is a synchronisation algorithm used in distributed database systems to ensure global atomicity of distributed transactions in spite of possible site and communication failures. That is, an ACP ensures that all sites participating in the execution of a distributed transaction reach the same final outcome whereby the transaction either commits, reflecting all its effects on the state of the distributed database; or aborts, nullifying any of its propagated effects from the state of the database.

The *two-phase commit* (2PC) (Gray, 1978; Lampson, 1981) is the first known ACP (Al-Houmaily and Samaras, 2009). It ensures atomicity of distributed transactions but at a substantial cost during normal transaction execution. This is due to the costs associated with its message complexity (i.e., the number of messages used for coordinating the actions of the different sites) and log complexity (i.e., the frequency at which information is stored onto the stable logs of the participating sites). For this reason, there is a continuing interest in developing more efficient ACPs and optimisations (Al-Houmaily, 2010). Most notable results are *one-phase commit* (1PC) protocols (Stamos and Cristian, 1993; Al-Houmaily and Chrysanthis, 2000; Abdallah et al., 2002; Lee and Yeom, 2002).

1PC protocols reduce both message and log complexities of 2PC by eliminating its (explicit) *voting phase* which is the first phase of the protocol. This is, however, at the expense of placing certain assumptions on transactions, the database management systems or the communication networks. Whereas some of these assumptions are realistic, reflecting how database management systems are usually implemented, other assumptions can be considered very restrictive for some software application systems (Abdallah et al., 1998; Al-Houmaily and Chrysanthis, 2000). One common assumption in 1PC protocols, including *implicit yes-vote* (IYV) (Al-Houmaily and Chrysanthis, 2000), is that each transaction's operation is acknowledged after its execution at a participating site. Thus, an operation acknowledgement (ACK) in these protocols does not only mean that the transaction preserves the *isolation* and *cascadless* properties, but it also means that the transaction is not in violation of any existing consistency constraints at the acknowledging participating site. Although this assumption is not too restrictive with respect to the most commonly implemented database management mechanisms in commercial systems as they implement *rigorous* schedulers (that preserves both the isolation and cascadless properties of database transactions), it clearly restricts the implementation of software application systems that wish to utilise the option of *deferred consistency constraints validation*. This option is currently part of the SQL standards (ISO, 2008) and allows for the evaluation of integrity constraints at the end of the execution of a transaction rather than at the end of each of its operations. Thus, when this option is used, the evaluation of deferred constraints needs to be synchronised across all participating database sites at commit time of the transaction, a clear limitation to the applicability scope of 1PC protocols as they eliminate such synchronisations.

The *adaptive participant's presumption protocol* (AP^3) was proposed to specifically alleviate the above limitation (Al-Houmaily, 2005). That is, AP^3 supports software application systems that use deferred consistency constraints without penalising those that do not require them. This is achieved by operating as a 1PC protocol for the commit processing of a transaction so long as the transaction is not associated with the validation of any deferred consistency constraints. But, when a transaction is associated with the validation of deferred consistency constraints, AP^3 dynamically switches to the most appropriate 2PC variant, depending on the anticipated results of the consistency constraints. Thus, even when AP^3 switches to 2PC, it selects the cheapest 2PC for the commit processing of the transaction. Furthermore, switching from one-phase to 2PC in AP^3 is accomplished on a *per participant* basis rather than on a *per transaction* basis. This is to achieve the highest attainable commit processing performance of the integrated component protocols in spite of the incompatibilities among them.

Although the (basic) AP^3 addresses and resolves a serious limitation of 1PC protocols in a cost-effective manner, it needs further extensions in order to make it a highly appealing ACP, among existing ones, for adoption in the design of future generations' database management systems. This is the essence of the *intelligent adaptive participant's presumption protocol* (intelligent AP^3). The new protocol encompasses significant extensions to the previously known AP^3. The new extensions are based on a comprehensive and effective use of *piggybacking* of control information among database sites regarding transactions. These extensions address and resolve four important issues in the design of ACPs. These issues can be classified into two categories: those that enhance *efficiency*, and those that enhance *applicability*.

The efficiency issues are concerned with the following:

1 exploiting read-only transactions for enhanced commit processing performance during normal processing

2 incorporating the option of forward recovery with transactions that are context-free for enhanced performance after failures while still able to identify transactions that create run-time contexts and prohibiting them from utilising this option.

The applicability issues, on the other hand, are concerned with the following:

1 identifying transactions that update large amounts of data (LAD) and switching them to a protocol that has the ability to allow them to continue their execution without having to propagate LAD from one site to another

2 resolving incompatibilities with pre-existing database sites that adopt the standardised 2PC protocol variant.

The remainder of this paper is structured as follows: first, Section 2 presents some background material about distributed transactions and the well known 2PC protocol that is used for their termination. For the sake of completeness, Section 2 also discusses the most pronounced 2PC protocol variants and IYV. After that, Section 3 reviews the basic AP^3 being the base for the design of the new protocol. Next, Section 4 presents and discusses the proposed intelligent AP^3 including its failure recovery aspects. Following that, Section 5 provides analytical performance evaluations and comparisons between the intelligent AP^3 and other previously known protocols based on the customary used performance metrics. Then, Section 6 shades some light on the most closely related previous works that were performed in the area of integrated ACPs. Finally, Section 7 makes some concluding remarks and suggests directions for future works in this field.

2 Background

A distributed/internet transaction accesses data located at different database sites that are interconnected via a communication network. Regardless of the location of data, each transaction accesses its required data by submitting operations to its *coordinator* which is assumed, without the loss of generality, to be the *transaction manager* at the site where the transaction is first initiated. Depending on the data distribution, the coordinator decomposes the transaction into a set of *subtransactions*, each of which executes at a single participating database site. When the transaction finishes its execution and submits its final commit request, the coordinator initiates an ACP such as the basic 2PC or one of its variants. This is done in order to synchronise a consistent termination of the transaction, across all participating sites, in spite of possible failures.

2.1 The 2PC and its two most common variants

The basic 2PC consists of two phases, namely a *voting phase* and a *decision phase*. During the voting phase, a coordinator requests all the participants in a transaction's execution 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'). On a commit decision, the coordinator sends out commit messages to *all* participants whereas, on an abort decision, it sends out abort messages to *only* those (required) participants that are prepared to commit (voted 'yes'). When a participant receives a decision, it enforces the decision and sends back an ACK.

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. Specifically, the coordinator of a transaction force-writes a *decision* record for the transaction prior to sending out its final decision to the participants. Since a *forced write* of a log record causes a flush of the log onto 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 sending its 'yes' vote and a *decision* record before acknowledging a final decision. When the coordinator completes the protocol, it writes a non-forced *end* record in the volatile portion of its log that is kept in the main memory. This record indicates that all (required) participants have received the final decision and none of them will inquire about the transaction's status in the future. Hence, allowing the coordinator to (permanently) forget the transaction, with respect to the 2PC protocol, and to garbage collect the log records of the transaction when necessary.

The basic 2PC is also referred to as the *presumed nothing* (PrN) protocol (Lampson and Lomet, 1993). This is 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. Thus, in PrN, missing commit processing information about an active transaction at its coordinating site after a failure is interpreted to mean that the transaction was aborted.

Recognising the above behaviour of PrN in the absence of information about active transactions after a failure and utilising it for the enhancement of its performance led to the design of the two most commonly pronounced 2PC variants, namely *presumed abort* (PrA) (Mohan et al., 1986) and *presumed commit* (PrC) (Mohan et al., 1986). In both variants, the cost of commit processing is reduced in comparison to PrN. This is accomplished by making an explicit and general interpretation to the meaning of missing information at a coordinator's site rather than making an implicit and special case interpretation to the meaning of such missing information, albeit for different final decisions. That is, PrA was designed to reduce the costs associated with aborting transactions by interpreting missing information to mean abort decisions; whereas PrC was designed to reduce the costs associated with committing transactions by interpreting missing information to mean commit decisions. For the sake of completeness, some further details about these two protocol variants are provided next.

2.1.1 Presumed abort

PrA reduces the costs associated with aborting transactions by generalising the special case whereby a coordinator, in PrN, makes an abort presumption about unremembered transactions. That is, PrA is designed such that the coordinator *intentionally* forgets aborting transactions, during normal transaction processing, and uses the abort presumption when replying to the inquiry messages of the participants while keeping its behaviour the same as in PrN for committing transactions. This is in contrast with PrN in which the coordinator uses the abort presumption *only* with undecided transactions and after it has recovered from a system failure.

Specifically, when a coordinator, in PrA, decides to abort a transaction, it does not force-write an abort decision in its log as in PrN. Instead, it just sends abort messages to all the participants that have voted 'yes' and discards all information about the transaction from its protocol table. Hence, the coordinator of an aborted transaction does not write any log records for the transaction or needs to wait for ACKs from the participants in order to forget the transaction. Since the coordinator of an aborting transaction does not require ACKs from the participants in order to be able to forget the transaction, the participants do not have to acknowledge abort decisions. Consequently, the participants do not need to force-write the abort decision onto their logs. Based on this design to PrA, if a participant inquires about a transaction, *at any time*, and the coordinator does not remember the transaction, it means that the transaction must have been aborted. Hence, the coordinator replies with an abort message. Thus, as the name implies, if no information is found about a transaction, the transaction is presumed aborted.

2.1.2 Presumed commit

PrA and PrC are both based on the same principle but serve different purposes. PrA reduces the costs associated with aborting transactions, while PrC reduces the costs associated with committing transactions. That is, PrC is designed such that missing information about transactions at a coordinator's site is interpreted as commit decisions. However, in PrC, a coordinator has to force-write a commit *initiation* record for each transaction before sending out 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's site failure without an actual commit decision being made.

To commit a transaction, the coordinator force-writes a commit record to logically eliminate the initiation record of the transaction and then send out the commit decision messages. 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. Based on this design to PrC, if a participant inquires about a transaction, *at any time*, and the coordinator does not remember the transaction, it means that the transaction must have been committed. Hence, the coordinator replies with a commit message. Thus, as the name implies, if no information is found about a transaction, the transaction is presumed committed.

To abort a transaction, 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 ACKs 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 the basic 2PC. Once the required ACKs arrive, the coordinator writes a non-forced end record and forgets the transaction. In the case of a coordinator's failure, the initiation record of an interrupted transaction contains all needed information for recovering the transaction and completing its commit processing.

2.2 One-phase commit

The basic 2PC and its common variants are usually criticised on the grounds of their performance drawbacks especially for short transactions, which is a common characteristic in many distributed database application systems including internet applications. For this reason and given the high reliability of today's database servers in addition to the increasing bandwidth of communication networks, *1PC* protocols were developed. These protocols consist of only a single phase which is the decision phase of 2PC. The (explicit) voting phase of 2PC is eliminated by overlapping it with the ACKs of the database operations (Stamos and Cristian, 1993; Al-Houmaily and Chrysanthis, 2000). This principle is used in *IYV*, on which AP^3 is based. In IYV, it is assumed that each database site deploys:

1 *strict two-phase locking* (S2PL) for concurrency control

2 *physical page-level replicated-write-ahead logging* (RWAL) with the undo phase *preceding* the redo phase for recovery.

These two assumptions allow IYV to atomically commit distributed transactions without the need for the (explicit) voting phase of 2PC (Al-Houmaily and Chrysanthis, 2000).

In IYV, when the coordinator of a transaction receives an ACK from a participant pertaining to a transaction's operation, the ACK is *implicitly* interpreted to mean that the transaction is in a prepared-to-commit state at the participant. When the participant receives a new operation for execution, the transaction becomes active again at the participant and can be aborted, for example, if it causes a deadlock or violation to any of the site's database consistency constraints. If the transaction is aborted, the participant responds with a *negative ACK* message (NACK). Only when *all* the operations pertaining to the transaction are executed and acknowledged by their respective participants, the coordinator commits the transaction. Otherwise, the coordinator aborts the transaction. In either case, the coordinator propagates its decision to all the participants and waits for their ACKs, as in 2PC. Thus, the explicit voting phase of 2PC, which pulls the votes of the participants, is eliminated by overlapping it with the execution of operations in IYV, while the *decision* phase remains the same as in 2PC.

IYV handles participant failures by partially replicating its log rather than force writing the log before each ACK. Each participant includes the *redo* log records that are generated during the execution of an operation with their corresponding *log sequence numbers* (LSNs) in the operation's ACK. Each participant also includes the *read* locks acquired during the execution of an operation in the ACK in order to support the option of *forward recovery* (Al-Houmaily and Chrysanthis, 2000). After a crash, a participant reconstructs the state of its database, which includes its log and lock table as it was just prior to the failure with the help of the coordinators. Hence, a participant in IYV is not only able to ensure the consistency of its database by applying the effects of committed transactions and rolling-back the effects of aborted transactions, but it is also able to allow transactions that are still active in the system, using the option of forward recovery, to continue their execution without having to abort them. On the other hand, by maintaining a local log and using WAL, each participant is able to undo the effects of aborted transactions locally using only its own log.

To limit the number of coordinators that need to be contacted after a site failure, each participant maintains a *recovery-coordinators' list* (RCL). Although this list is not necessary for the correct operation of the protocol, it is an optimisation in which the RCL contains the identities of the coordinators that have active transactions at the participant's site and must be contacted during the recovery of the participant after a failure. In order to survive failures, an RCL is kept in the stable log. Thus, when a participant receives the first operation of a transaction and the identity of the coordinator of the transaction is not already in its RCL, the participant adds it to its RCL, force writes the RCL onto its stable log and then executes the operation. In order to avoid searching the entire RCL in the case that all the coordinators in the system are active at a participant, an *all-active flag* (AAF) is used. A participant sets its AAF once it force-writes an RCL containing the identities of *all* the coordinators and does not consider the RCL so long as the AAF is set.

To reduce the costs associated with aborting transactions, the *IYV presumed abort* (IYV-PrA) was developed (Al-Houmaily and Chrysanthis, 2000). IYV-PrA is based on the same principles as PrA. That is, making the abort presumption explicit and more general. Specifically, when a transaction is aborted by its coordinator, the coordinator does not write the final decision in its log. It just sends abort messages to the participants and forgets the transaction. As the coordinator can forget an aborting transaction without any ACKs from the participants, the participants are not required to acknowledge abort decisions. As a result, IYV-PrA reduces the costs associated with the base IYV protocol for each aborting transaction by one forced log write (at the coordinator's site of the transaction) and a message from each participating site.

3 The AP^3

Although 1PC protocols are more efficient than 2PC and its commonly known variants (Chrysanthis et al., 1998), they place assumptions that make them less appropriate than 2PC to be adopted in commercial systems. Some of these assumptions can be relaxed (Abdallah et al., 2002) while others cannot, leading to applicability limitations. One of the biggest limitations in 1PC protocols is their inability to support deferred consistency constraints. This is because they eliminate the (explicit) voting phase of 2PC which triggers the validation of any deferred constraints at the participating sites. AP^3 was designed to specifically address this limitation and to support application systems that wish to utilise the option of deferred constraints that is specified in the SQL standards without penalising those that do not require them. To achieve this, AP^3 operates as a 1PC protocol so long as the voting phase is not necessary. But, when the voting phase is needed with some participants (to validate deferred consistency constraints), the coordinator switches to either PrA or PrC with these participants. Switching to either protocol depends on the tendency of the evaluation of consistency constraints. That is, if deferred consistency constraints tend to be fulfilled, AP^3 switches to PrC. Otherwise, it switches to PrA. Thus, even when AP^3 switches to 2PC, it selects the cheapest 2PC variant. In this way, AP^3 achieves the performance of IYV whenever possible and, at the same time, the applicability of 2PC with respect to its ability to handle deferred consistency constraints.

3.1 The general dynamics of the AP^3

In AP^3, a coordinator records information pertaining to the execution of a transaction in a protocol table which is kept in main memory. Specifically, a coordinator keeps for each transaction the identities of the participants that join in the execution of the transaction and any operation request submitted to a participant and still pending for execution at the participant. The coordinator also keeps track of the used protocol with each participant (i.e., IYV, PrA or PrC). When a coordinator submits the *first* operation to be executed at a participant for a particular transaction, the coordinator registers the participant in its protocol table and marks the participant as 1PC. Thus, the default ACP with each participant in AP^3 is 1PC.

Following IYV, each participant keeps a *RCL* that contains the identities of the coordinators that have active transactions at its site and must be contacted during recovery after a site's failure. In order to survive failures, an RCL is kept onto the stable log. When a participant receives the first operation of a transaction and the identity of the coordinator of the transaction is not already in its RCL, the participant adds the identity of the coordinator to its RCL, force writes the RCL onto its stable log, and then executes the operation. In order to avoid searching the entire RCL in the case that all the coordinators in the system are active, an *AAF* is used. A participant sets AAF once it force-writes an RCL that contains the identities of *all* the coordinators and does not consult its RCL so long as the AAF is set.

When an operation is executed successfully at a participating site, the participant sends back an ACK that contains the results of the operation to the coordinator. On the other hand, if the execution of an operation fails, the participant sends back a NACK message. In either case, the participant does not force its log onto stable storage prior to acknowledging an operation. For a successful update operation that is not associated with any deferred validation of consistency constraints, the participant follows IYV. Hence, it includes in each ACK all the *redo* log records that have been generated during the execution of the operation and implicitly enters a prepared-to-commit state with respect to the invoking transaction, waiting for either the final decision or another operation from the transaction. If a new operation arrives, the participant returns to an active state to execute the operation.

If an update operation requires the validation of some deferred consistency constraint(s), the participant indicates this to the coordinator by requesting to *switch* to a 2PC variant. This is achieved by sending an *unsolicited deferred consistency constraint* (UDCC) flag that is set as part of the operation's ACK. The participant also indicates, in the UDCC, the most appropriate 2PC variant depending on the tendency of the validation results of the consistency constraint(s) at its site. That is, if a deferred constraint tends to be violated by transactions and consequently cause them to abort at commit processing time, the participant selects PrA. Otherwise, the participant selects PrC. The reason behind the participant's choice between PrA and PrC is due to the fact that it is cheaper to abort a transaction using PrA compared to PrC, while it is cheaper to commit a transaction using PrC compared to PrA (Mohan et al., 1986).

Once UDCC is set as part of an ACK, the participant does not include any redo log records for the executed operation or any subsequent update operation(s) pertaining to the same transaction. Also, the participant does not enter a prepared-to-commit state with respect to the transaction until it receives an *explicit* prepare-to-commit message from the coordinator, as in all 2PC protocol variants. Furthermore, if the transaction is the last active transaction submitted by its coordinator to the participant, the participant resets its AAF if it is set, deletes the transaction's coordinator from the RCL, and force writes the updated list onto its stable log. This is because the participant does not need to consult the transaction's coordinator after a site's failure as the transaction has switched to a 2PC protocol and consulting such a coordinator is necessary only for the recovery of 1PC transactions.

When a coordinator receives an ACK with a UDCC flag set, it updates its protocol table to reflect the protocol switch for the participant and the preferred 2PC variant. On the other hand, if the flag is not set, the coordinator checks its protocol table to determine if the ACK is from a 1PC participant and extracts any redo log records contained in the message. Then, it writes a non-forced log record containing the received redo records along with the participant's identity. Hence, for 1PC participants, the coordinator's log contains a partial image of the redo part of each participant's log which can be used to reconstruct the redo part of the participant's log in case it is corrupted due to a system failure.

3.2 Dynamics of terminating a transaction in the AP^3

If the coordinator receives either an abort request from a transaction or a NACK in response to the execution of one of its operations 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 successful processing of all the operations that were assigned to it.

On the other hand, if the coordinator of a transaction receives a commit primitive from the transaction after all its operations have been executed successfully, the coordinator checks its protocol table to determine whether any participant has switched to 2PC. If no participant has switched protocol, the coordinator commits the transaction as in IYV. That is, it force writes a commit log record which includes the identities of all participants, and then sends commit messages to the participants. Once the coordinator receives ACKs from all participants, it writes a non-forced end log record and forgets the transaction.

If any participant has switched protocol, the coordinator decides which 2PC variant to use for the commitment of the transaction with 2PC participant(s). The coordinator selects PrA if *any* 2PC participant has requested the use of PrA. Otherwise, the coordinator selects PrC. The preference of using PrA instead of PrC with all 2PC participants in the presence of even one PrA participant among them in AP^3 is because it is cheaper to use PrA than PrC. That is, when a participant requests to use PrA, it means that the transaction is more likely going to violate the deferred constraint(s) at the PrA participant and, consequently, the transaction will end up aborting regardless of the number of participants that have requested to use PrC.

After deciding on the 2PC variant, the coordinator force-writes a *switch* log record only if the protocol to be used with the 2PC participants is PrC. The switch record includes the identities of all participants, indicating the participants that have switched protocol. Then, for each 2PC participant, the coordinator sends a prepare-to-commit message, indicating the decided 2PC protocol variant (i.e., PrA or PrC), as shown in Figure 1.

When a 2PC participant receives a prepare-to-commit message, it validates the transaction with respect to the deferred consistency constraints. If the transaction can be committed, the participant force writes a prepared log record which includes the used 2PC variant. Then, the participant sends back its vote to the coordinator. As in all 2PC variants, a 2PC participant in AP^3 sends a 'yes' vote only if all consistency constraints are validated and the participant can comply with a final commit decision. Otherwise, it aborts the transaction and sends back a 'no' vote. When a participant votes 'yes', it enters an *explicit* prepared to commit state.

When the coordinator receives the votes of 2PC participants, the coordinator makes the final decision. The decision is commit if each 1PC 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.

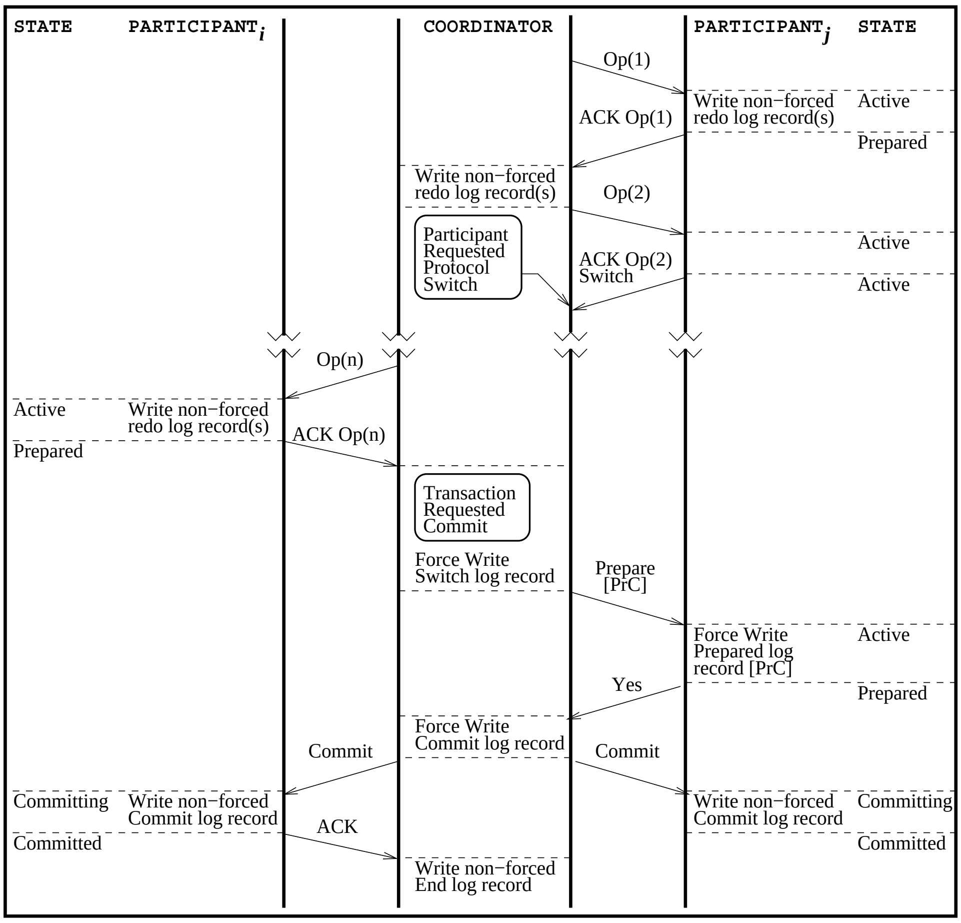
3.2.1 Committing a transaction in presence of 2PC participants

If the decision is commit and the used protocol with 2PC participants is PrC [Figure 1(a)], the coordinator force writes a commit log record, sends the decision to all participants, and waits for the ACKs of 1PC participants only. When a 2PC participant receives the decision, following PrC, it writes a non-forced commit log record and complies with the decision without sending an ACK back to the coordinator.

On the other hand, if the used protocol with 2PC participants is PrA [Figure 1(b)], the coordinator force writes a commit log record, sends the decision to all participants, and waits for the ACKs of both 1PC participants as well as 2PC participants. When a 2PC participant receives the commit decision, following PrA, it force writes a commit log record and complies with the decision. Then, it sends back an ACK to the coordinator.

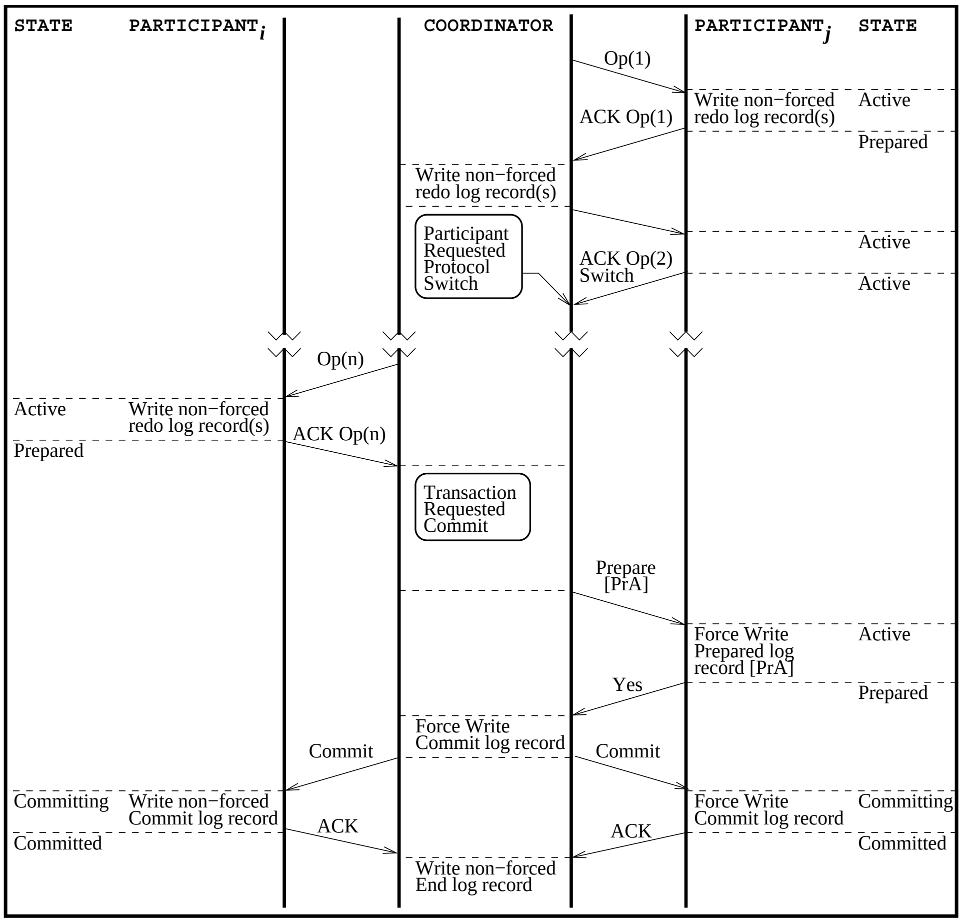
When a 1PC participant receives a commit decision for a transaction, it enforces the decision, writes a non-forced commit log record and releases all the resources held by the transaction. When the commit record is, eventually, forced onto the stable storage, due to a subsequent flush to the log buffer possibly on behalf of some other transaction, the participant resets its AAF if the transaction was the last active transaction submitted by its coordinator and the AAF is set, deletes the transaction's coordinator from the RCL and force writes the updated RCL list onto the stable log. Then, the participant acknowledges the decision. Thus, a participant in AP^3 follows IYV so long as it did not switch protocol and regardless of the used protocols with the other participants.

Figure 1 Committing a transaction in AP^3, (a) switching to PrC (b) switching to PrA


STATE
PARTICIPANT$_i$
COORDINATOR
PARTICIPANT$_j$
STATE
Op(1)
Write non−forced redo log record(s)
Active
ACK Op(1)
Prepared
Write non−forced redo log record(s)
Op(2)
Participant Requested Protocol Switch
Active
ACK Op(2) Switch
Active
Op(n)
Active
Write non−forced redo log record(s)
ACK Op(n)
Prepared
Transaction Requested Commit
Force Write Switch log record
Prepare [PrC]
Force Write Prepared log record [PrC]
Active
Yes
Prepared
Force Write Commit log record
Commit
Commit
Committing
Write non−forced Commit log record
Write non−forced Commit log record
Committing
ACK
Committed
Committed
Write non−forced End log record

(a)

Figure 1 Committing a transaction in AP^3, (a) switching to PrC (b) switching to PrA (continued)


STATE
PARTICIPANT$_i$
COORDINATOR
PARTICIPANT$_j$
STATE
Op(1)
Write non−forced redo log record(s)
Active
ACK Op(1)
Prepared
Write non−forced redo log record(s)
Active
Participant Requested Protocol Switch
ACK Op(2) Switch
Active
Op(n)
Active
Write non−forced redo log record(s)
ACK Op(n)
Prepared
Transaction Requested Commit
Prepare [PrA]
Force Write Prepared log record [PrA]
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)

When the coordinator receives ACKs for a commit decision from 1PC participants, in the case of using PrC with 2PC participants, it writes a non-forced end log record and discards all information pertaining to the transaction from its protocol table, knowing that only PrC participants might inquire about the transaction's status in the future. On the other hand, when the coordinator receives ACKs for a commit decision from *all* participants, in the case of using PrA with 2PC participants, it writes a non-forced end log record and discards all information pertaining to the transaction from its protocol table, knowing that no participant will inquire about the transaction's status in the future.

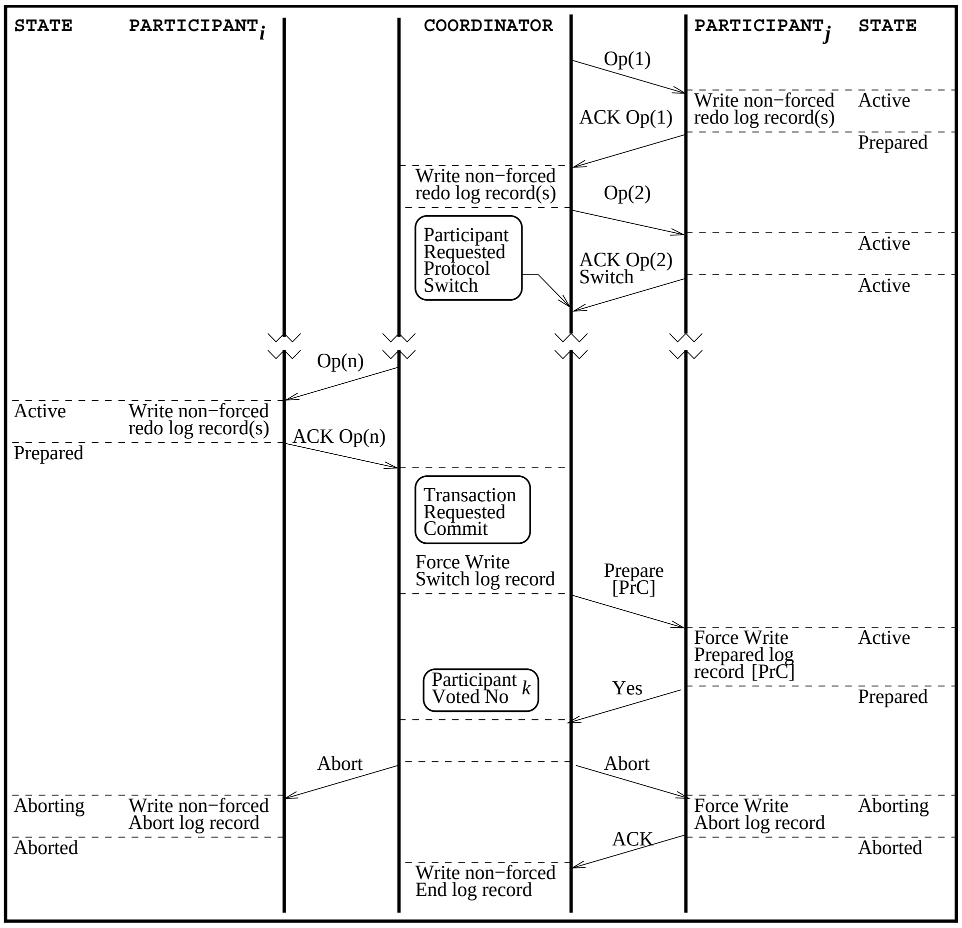
Since for the commit case only a PrC participant (if any) might inquire about the outcome of a forgotten transaction, the coordinator, not remembering the transaction, will reply with a commit message using the presumption of PrC protocol that was used by the participant. Recall that each 2PC participant records the used 2PC variant for a transaction in the prepared log record. This information is used in the inquiry message of the participant to guide the coordinator in its response in the absence of information about the transaction.

3.2.2 Aborting a transaction in presence of 2PC participants

If the decision is abort and the used protocol with 2PC participants is PrC [Figure 2(a)], the coordinator sends the decision to all participants, except those 2PC participants that voted ‘no’, without writing a decision record. Then, the coordinator waits for the ACKs of 2PC participants only. When a 2PC participant receives the decision, it force writes an abort log record, complies with the decision and then, sends back an ACK.

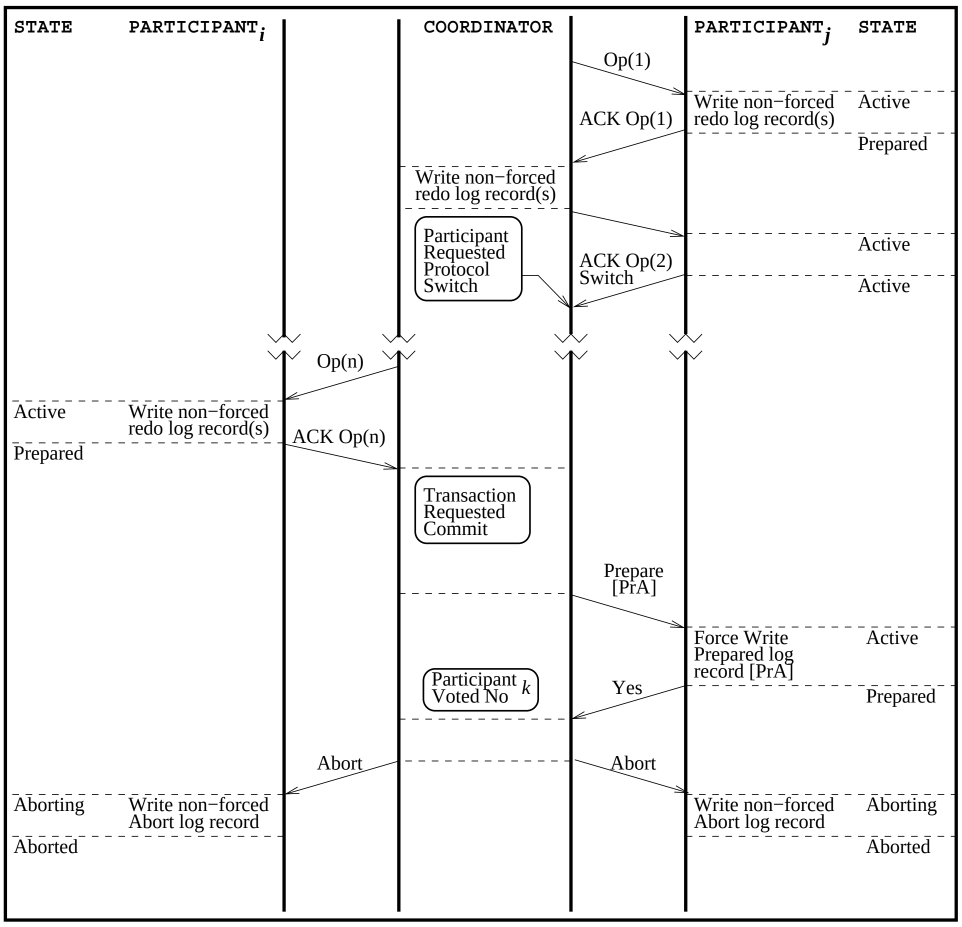
On the other hand, if the used protocol with 2PC participants is PrA [Figure 2(b)], the coordinator sends the decision to all participants, except those 2PC participants that voted ‘no’, and then, discards all information pertaining to the transaction from its protocol table. Thus, in this case, the coordinator does not write any log records for the aborted transaction onto its log or waits for ACKs. When a 2PC participant receives the decision, it writes a non-forced abort log record and complies with the decision without sending an ACK back to the coordinator.

Figure 2 Aborting a transaction in AP^3, (a) switching to PrC (b) switching to PrA


STATE
PARTICIPANT$_i$
COORDINATOR
PARTICIPANT$_j$
STATE
Op(1)
Write non−forced redo log record(s)
Active
ACK Op(1)
Prepared
Write non−forced redo log record(s)
Op(2)
Active
Participant Requested Protocol Switch
ACK Op(2) Switch
Active
Op(n)
Active
Write non−forced redo log record(s)
ACK Op(n)
Prepared
Transaction Requested Commit
Force Write Switch log record
Prepare [PrC]
Force Write Prepared log record [PrC]
Active
Participant Voted No k
Yes
Prepared
Abort
Abort
Aborting
Write non−forced Abort log record
Force Write Abort log record
Aborting
ACK
Aborted
Aborted
Write non−forced End log record

(a)

Figure 2 Aborting a transaction in AP^3, (a) switching to PrC (b) switching to PrA (continued)


STATE
PARTICIPANT$_i$
COORDINATOR
PARTICIPANT$_j$
STATE
Op(1)
Write non−forced redo log record(s)
Active
ACK Op(1)
Prepared
Write non−forced redo log record(s)
Participant Requested Protocol Switch
Active
ACK Op(2) Switch
Active
Op(n)
Active
Write non−forced redo log record(s)
ACK Op(n)
Prepared
Transaction Requested Commit
Prepare [PrA]
Force Write Prepared log record [PrA]
Active
Participant Voted No k
Yes
Prepared
Abort
Abort
Aborting
Write non−forced Abort log record
Write non−forced Abort log record
Aborting
Aborted
Aborted

(b)

When a 1PC participant receives an abort decision, it enforces the decision, writes a non-forced abort log record and releases all the resources held by the transaction without sending back an ACK. This is because AP^3 incorporates the PrA variant of IYV in which only commit decisions are acknowledged but not abort decisions. Then, if the transaction was the last active transaction submitted by its coordinator, the participant resets its AAF if it is set, deletes the transaction's coordinator from the RCL and force writes the updated RCL list onto the stable log. Again, a participant in AP^3 follows IYV so long as it did not switch protocol and regardless of the used protocols with the other participants.

When the coordinator receives ACKs of an abort decision from the required 2PC participants, in the case of using PrC with 2PC participants [Figure 2(a)], it writes a non-forced end log record and discards all information pertaining to the transaction from its protocol table, knowing that only 1PC participants might inquire about the transaction's status in the future.

Since for the abort case only an IYV or a PrA participant might inquire about the outcome of a forgotten transaction, the coordinator, not remembering the transaction, will reply with an abort message using the abort presumption of IYV and PrA. Again, recall that each 2PC participant records the used 2PC variant for a transaction in the prepared record. This information is used in the inquiry message of the participant to guide the coordinator in its response in the absence of information about the transaction. For an inquiry message from a 1PC participant, the message does not include any indication about the used protocol. Such an inquiry message from a participant is implicitly interpreted by the coordinator to mean that the used protocol with the forgotten transaction is IYV and, consequently, the coordinator responds with an abort message as the transaction must have been aborted.

4 The intelligent AP^3

This section presents the intelligent AP^3 that extends the basic AP^3 in two directions: *efficiency* and *applicability*. The efficiency extensions are enhancing:

1 the performance of the protocol in the presence of read-only participants

2 the performance of the protocol after a participant's site failure by supporting the option of forward recovery.

On the other hand, the applicability extensions are:

1 overcoming the limitations of the basic AP^3 from its inability to handle transactions that update LAD at some participants in an efficient manner

2 providing a solution to the problem of compatibility with pre-existing PrA participants.

4.1 The intelligent AP^3 and read-only transactions

In the traditional read-only optimisation (Mohan et al., 1986), a participant votes *read-only* on behalf of a transaction, during the voting phase of 2PC, if the participant has executed only read operations for the transaction. Using this optimisation, a read-only participant releases all the resources held by the transaction once it votes and does not participate in the second phase of the protocol. This is because the participant can terminate the transaction at its site and release all the resources held by the transaction without needing to know the final decision. In this way, a read-only participant does not write any log records and the cost of messages with such a participant are reduced by half (i.e., one round of messages). Besides that, the early release of resources, at read-only participants, significantly enhances the overall concurrency of transactions in the system.

The cost associated with read-only participants can be reduced further if the coordinator of a transaction knows which participants are read-only in the execution of the transaction before the initiation of commit processing. If such prior knowledge can be made available to the coordinator, the coordinator can inform each read-only participant that the transaction has terminated, during the voting phase, without having to explicitly ask the participant about its (read-only) vote. By eliminating the read-only votes, the cost of messages with read-only participants is reduced further. In addition, for an *exclusively* read-only transaction, using this prior knowledge, the coordinator can avoid writing any log records for the transaction when PrC is used. This represents the essence of the *unsolicited update-vote* (UUV) optimisation (Al-Houmaily et al., 1997a, 1997b), the principle of which is used in the intelligent AP^3.

In the intelligent AP^3, each transaction starts as a read-only transaction and is marked as such in its coordinator's protocol table. The transaction remains read-only so long as its coordinator does not receive any ACK that contains redo log records and any ACK with UDCC flag that is set. This is because only update operations generate redo log records at a participant and are sent to the coordinator in an ACK; or are associated with deferred consistency constraints and, consequently, the setting of the UDCC flag in a *switch* ACK. If the coordinator receives an ACK that contains redo log records from a participant, it means that the participant has updated some data on behalf of the transaction and the transaction is not read-only at the participant anymore. Similarly, if the coordinator receives an ACK from a participant with the UDCC flag set, it means that the participant has updated some data on behalf of the transaction and the updated data is associated with some deferred consistency constraints that need to be validated at commit time of the transaction. In this latter case, the participant also indicates its preferred 2PC variant in the message as in the basic AP^3. As discussed earlier, this message contains the set UDCC flag without including any redo log records for the executed operation. In either of the two cases above, the coordinator marks the transaction as an update transaction instead of a read-only at this participant.

By following the above strategy, once a transaction finishes its execution and submits its final commit primitive, the coordinator refers to its protocol table to determine not only which protocols to use with the different participants, which would be the case in the basic AP^3, but also which participants are read-only. To commit an *exclusively* read-only transaction, the coordinator sends a *read-only* message to each participant without writing any log records for the transaction, and then forgets the transaction. If the transaction is *partially* read-only (i.e., it is an update transaction at some sites and read-only at others), the coordinator initiates the voting phase with 2PC participants (if any) and, at the same time, sends a read-only message to each read-only participant. Then, the coordinator removes the read-only participants from its protocol table without waiting until the final decision is made. When a read-only participant receives the message, it releases all the resources held by the transaction without writing any log records or acknowledging the message. Thus, this read-only optimisation allows for an early release of resources at read-only participants, as in the traditional read-only optimisation, while reducing the number of messages with such participants to the half. For update participants, whether they are 1PC or 2PC participants, the coordinator completes the protocol as in the basic AP^3.

4.2 The intelligent AP^3 and the option of forward recovery

The intelligent AP^3 adopts the option of forward recovery as defined and used in IYV. Specifically, the adopted definition is concerned with the ability of the system to allow a partially executed transaction that was interrupted during its execution by a (site or communication) failure to resume its execution after the failure is fixed (Al-Houmaily and Chrysanthis, 2000). Thus, the adopted definition of forward recovery is applicable to transactions that are *context-free* (Gray and Reuter, 1993) at the participants whereby there is no need for the preservation of any of a transaction's intermediate run-time results at a participant in order for the transaction to forward recover.

In the intelligent AP^3, a transaction indicates to its coordinator, when it is first initiated, whether it wishes to use the option of forward recovery or not. When this option is used, it means that the transaction is willing to wait on any delays that it may encounter during its execution due to a failure. This is an especially important option for long living transactions so that they have the ability to finish their execution after the failure is fixed instead of aborting them once a failure is detected. For a *forward recoverable* transaction, the coordinator marks the transaction as such in its protocol table and notifies each participating site of this option by setting a *forward recovery flag* (FRF) as part of the first operation submitted to the participant.

For a forward recoverable transaction, a 1PC participant includes in each operation's ACK, in addition to the redo log records (if any), all the read locks that have been acquired during the execution of the operation. When the coordinator receives an ACK from a 1PC participant, in addition to the redo log records, it extracts the read locks from the message and keeps them in a *participants' lock table* (PLT) which is part of its protocol table.

Using the option of forward recovery, the coordinator's log contains a partial image of the redo part of each 1PC participant's log which can be used to reconstruct the redo part of the participant's log in the case it is corrupted due to a system failure. At the same time, the coordinator's PLT contains a partial image of each 1PC participant's lock table which can be used to reconstruct the participant's lock table in the case it is corrupted due to a system failure. As a result, after a participant's failure, the participant can recover its state, with respect to any prematurely aborted 1PC forward recoverable transaction, as it was prior to the failure with the help of the coordinators. This is accomplished by reconstructing its log to reflect missing redo log records and reacquiring both read and write locks of the interrupted forward recoverable transaction. This allows partially executed forward recoverable 1PC transactions at a participant that are still active in the system to forward recover and resume their execution after the participant recovers.

4.2.1 Forward recovery and transactions' contexts in the intelligent AP^3

In general, it is not possible for a transaction to determine whether it will create a run-time context at a participating site or not. In fact, a distributed transaction does not know about data distribution or even about its decomposition to a number of subtransactions let alone about creating a context at a participant. For this reason and for the applicability of any ACP that chooses to adopt the option of forward recovery, the ACP should be able to handle transactions that create contexts. However, as run-time *context preservation* and *context propagation* are beyond the scope of this article, the intelligent AP^3 provides a simple mechanism for handling such transactions. The solution is based on identifying such transactions and prohibiting them from utilising the option of forward recovery.

In the intelligent AP^3, when a participant executes the first operation that creates a context for a 1PC forward recoverable transaction, the participant becomes aware that the transaction state cannot be recovered if the participant fails. Based on that, the participant switches the transaction to become non-forward recoverable (NFR) and marks it as such in its protocol table. Then, the participant indicates this state switch in the ACK of the operation (that first created the context for the transaction). After that, the participant refrains from sending any read locks that the transaction acquire to the transaction's coordinator. However, the participant continues to send any generated redo log records for the transaction as these records are still needed for recovery purposes after a failure even though the transaction is not forward recoverable.

When the coordinator receives an ACK indicating that a 1PC transaction has become NFR at a participant, the coordinator marks the transaction as NFR in its protocol table and informs the other 1PC participants accordingly. To inform one of the other 1PC participants that the transaction is not forward recoverable anymore, the coordinator sets a *NFR* flag in the next operation that it submits to the participant. Once a 1PC participant receives a new operation for execution and the operation message contains an NFR flag set, it behaves as if it is the one which has switched the transaction to be NFR. That is, the participant refrains from sending any acquired read locks for the transaction to its coordinator. At the end of the transaction, the coordinator and each participant all proceed with the commit processing of the transaction as in the basic AP^3.

It should be pointed out that forward recovery in the intelligent AP^3 is applied on each transaction as a whole but not on its individual subtransactions. Thus, a transaction that chooses to use the option of forward recovery may be forward recoverable so long as all the sites participating in its execution are 1PC participants. This is because a 2PC transaction at a participant, in the intelligent AP^3, is associated with the validation of deferred consistency constraints and these constraints represent run-time contexts that need to be preserved in order for the transaction to be validated at commit time of the transaction. For this reason, it is not possible for a 2PC transaction at a participant, in the intelligent AP^3, to forward recover as the run-time context is lost in the event of a system failure causing the whole transaction to be NFR. Hence, when a transaction switches to a 2PC variant, it becomes a NFR transaction across all participating sites.

4.3 The intelligent AP^3 and updating LAD

In some database applications, a transaction may cause updates to LAD. For example, a single SQL 'UPDATE' statement may cause updates to the tuples of an entire relation. The size of such a relation might be prohibitively large to be propagated to the coordinator of a transaction. This is not because of the communication link bandwidth, that one may pragmatically assume to be in the order of gigabit per second given today's technology, but because of the limited capacity of the log buffer (in the main memory) of the coordinator. This limited size might become a bottleneck in the system due to the need for its flushing, possibly several times, to handle such LAD.

The intelligent AP^3 provides a solution to this applicability limitation problem in the basic AP^3. The solution is based on using a flag called *LAD* that a participant sets as part of the ACK of an operation that causes updates to LAD. This flag informs the coordinator that the participant has switched protocol because the transaction has updated LAD at the participant's site. More specifically, when a 1PC participant updates LAD during the execution of an operation and the updated data is not associated with deferred consistency constraints, it sets this flag in the ACK of the operation and switches to PrC. On the other hand, if the updated data is associated with deferred constraints, the participant chooses the appropriate 2PC variant depending on the tendency of the evaluation of these constraints at commit time of the transaction. Once a participant has switched its protocol, it refrains from sending any redo log records in the ACKs of update operations. Furthermore, if the transaction was set as forward recoverable, the participant marks the transaction as NFR in its protocol table and refrains from sending any read locks that the transaction acquires during its execution to the coordinator. However, if a participant switches to PrC because the transaction has updated LAD at its site and later on it executed an operation that is associated with deferred consistency constraints that tend to be violated, the participant request to change its previously selected protocol to PrA in the ACK of the operation. Thus, when a participant in the execution of a transaction switches to a 2PC variant, it chooses PrA only when the transaction is associated with deferred constraints that tend to be violated at commit time of the transaction.

When the coordinator of a transaction receives an ACK with a set LAD flag from a participant, it marks the participant as either PrC or PrA in its protocol table, depending on the chosen protocol by the participant. Furthermore, if the transaction is forward recoverable, the coordinator changes the state of the transaction in its protocol table to be NFR. Once the coordinator has changed the state of a transaction to NFR because of a LAD flag that it has received from a participant, it needs to inform all the other 1PC participants about this new state of the transaction. This is accomplished by indicating that the transaction is not forward recoverable in the first operation the coordinator sends to each of the other 1PC participants. When a participant receives such a message, it refrains from sending read locks in the ACK of each operation it executes on behalf of the transaction. At the end of the transaction, the coordinator and each participant all proceed with the commit processing of the transaction as in the basic AP^3.

4.4 *The intelligent AP^3 and backward compatibility*

The basic AP^3 interoperates three ACPs (i.e., IYV, PrA and PrC) assuming a *homogeneous* environment in which each participant uses the same protocol, i.e., the basic AP^3. In this way, each participant understands the language of its coordinator with respect to messages and has the ability to respond to them accordingly, and vice versa. However, for practical reasons, the basic AP^3 still needs to resolve the problem of compatibility with any *pre-existing* database sites that do not use AP^3. In this article, we concentrate on the compatibility of AP^3 with database sites that adopt PrA. This is because PrA is the current choice of database standards (X/Open Company Limited, 1996; ISO, 1998), and commercial systems that adopt this protocol are likely to exist for many years to come. For this reason, it is imperative to cope with the existence of sites that deploy PrA and do not recognise the mechanisms used in the basic AP^3.

To resolve the problem of compatibility of AP^3 with pre-existing PrA database sites, the intelligent AP^3 addresses the two possible cases for the presence of sites that deploy PrA:

1 the presence of a PrA participant with an AP^3 coordinator

2 the presence of a PrA coordinator with an AP^3 participant.

The intelligent AP^3 addresses the first case because in the presence of a PrA participant, an AP^3 coordinator cannot recognise such a participant and, consequently, will act with it inappropriately, jeopardising transactions' atomicity. Similarly, the intelligent AP^3 addresses the second case because in the presence of a PrA coordinator, an AP^3 participant cannot recognise such a coordinator and, consequently, will act with it inappropriately, also jeopardising transactions' atomicity.

In the next section, the compatibility problems that arise in the presence of only PrA participants are discussed and it is shown how the intelligent AP^3 overcomes these problems. Then, in Section 4.4.2, the compatibility problems that arise in the presence of both PrA participants as well as PrA coordinators are discussed and it is shown how the intelligent AP^3 resolves these problems.

4.4.1 The intelligent AP^3 in the presence of only PrA participants

Assume that coordinators are AP^3 whereas participants are mixed AP^3 and pre-existing PrA. In this scenario, a PrA participant will not declare its used protocol during the course of the execution of transactions. Thus, a coordinator of a transaction, in accordance to AP^3, will presume that any pre-existing PrA participant is an IYV participant. This is because, in AP^3, the default protocol with a participant is IYV unless the participant switches its protocol using a switch message. Based on this presumption, the coordinator will not request from the participant to prepare-to-commit and will make its decision without the (explicit) vote of this participant. If the coordinator decides to commit the transaction after the transaction has executed to completion, the coordinator will send its final decision without ever requesting the participant to prepare-to-commit. This constitutes a protocol violation and the PrA participant is not equipped with the mechanisms to handle this type of violations. Based on that, the participant has no alternative way other than ignoring such a message. By doing so, the participant will eventually abort the transaction either because it is left inactive for a long period of time which causes the transaction to time-out, or because of a subsequent participant's site failure that will cause the transaction to be also aborted as it has never entered a prepared to commit state. Thus, the transaction will terminate inconsistently across different sites, violating atomicity.

The intelligent AP^3 resolves the above problem by having each AP^3 participant, but, of course, not any pre-existing PrA participant, to declare itself to the coordinator as an AP^3 participant. This is accomplished at the beginning of the execution of each transaction at an AP^3 participant. The rest of the participants are assumed, by the coordinator, to be PrA participants. More specifically, an AP^3 participant declares its used protocol in the ACK of the first operation that it executes for each transaction. That is, if the first operation that the participant has executed is associated with the validation of deferred consistency constraint(s), the participant includes its preferred protocol among PrA and PrC in the ACK of the operation, as it would be the case in the basic AP^3. Otherwise, the participant *explicitly* declares itself as an IYV participant. This is accomplished by using a *participant's used protocol* (PUP) flag in the ACK of the first operation that a participant executes at its site for a transaction. In this way, the coordinator of a transaction can infer pre-existing PrA participants. That is, if the ACK of the first operation executed at a participant does not include the PUP flag, it means that the participant is a pre-existing PrA participant.

At the end of a transaction, an intelligent AP^3 coordinator precisely knows which participant is using which protocol. Based on that, if any AP^3 participant has chosen PrA, the coordinator uses PrA and informs all AP^3 participants that requested to switch to PrC about the decided protocol in the prepare-to-commit message. Then, it follows the basic AP^3 for the rest of the protocol. On the other hand, if all AP^3 participants that have switched to 2PC had all chosen PrC as their preferred protocol in the presence of pre-existing PrA participants, the coordinator could still decide to use PrA with such AP^3

participants as in the previous case. But, this is not the best strategy from performance point of view. This is because the use of PrA by the pre-existing participants does not imply, by any means, that transactions will most likely go to abort rather than to commit or even imply that transactions are associated with the validation of deferred consistency constraints. For this reason, the intelligent AP^3 uses the chosen PrC protocol with AP^3 participants and PrA with the pre-existing PrA participants.

In the intelligent AP^3, when 2PC participants use mixed PrA and PrC protocols, the coordinator force-writes an initiation record. Then, it initiates the voting phase with all 2PC participants indicating the decided protocol to 2PC AP^3 participants without requesting AP^3 participants to use PrA if they all had chosen to use PrC. Once the participants' votes arrive, the coordinator makes the final decision. If the decision is commit, the coordinator force-writes a commit record and then sends out commit messages to all participants (including IYV participants). As PrC participants never acknowledge commit decisions, the coordinator writes an end record once IYV and pre-existing PrA participants acknowledge the commit decision. Then, the coordinator forgets the transaction, knowing that only a PrC might inquire about the status of the transaction in the future. On the other hand, if the decision is abort, the coordinator sends out its decision to all participants and writes an end record when PrC participants acknowledge the decision, knowing that only an IYV or a pre-existing PrA might inquire about the status of the transaction in the future.

For a read-only transaction at a pre-existing PrA participant, the participant will respond with a 'read-only' message instead of a 'yes' vote if the participant employs the traditional read-only optimisation. In this case, the coordinator, following the traditional read-only optimisation, removes the participant from the decision phase of the protocol.

4.4.2 *The intelligent AP^3 in presence of both PrA participants and PrA coordinators*

Assume the general scenario in which both AP^3 and PrA coordinators are present besides the presence of mixed AP^3 and PrA participants. In this general scenario, the above strategy does not work as it was designed assuming that all coordinators deploy AP^3. This is because any message that includes a participant's declaration to its used protocol, which is the case with the intelligent AP^3 participants in the above strategy, will constitute a PrA protocol violation from a PrA coordinator's perspective. For example, a PrA coordinator does not understand any message from an AP^3 participant that includes redo log records and will ignore such a message. Similarly, a PrA coordinator does not understand any switch message from an AP^3 participant and will ignore such a message. By ignoring such violating messages, the coordinator will either reach a wrong decision that might violate atomicity across the different participants or a decision cannot be reached for some transactions that execute at certain mixes of participants.

To provide a general and complete solution to the problem of compatibility with pre-existing PrA participants as well as pre-existing PrA coordinators, in the intelligent AP^3, each AP^3 participant records in its protocol table, at system installation time, the identities of all coordinators that use PrA. These identities are kept in a list called *presumed-abort coordinators* (PAC). This list of coordinators is stored onto the stable storage of each AP^3 participant and is updated when a new PrA coordinator joins or when an already existing PrA coordinator leaves the system. Thus, this list is kept at an AP^3 participant so long as there is some PrA coordinator(s) in the system.

By using the PAC lists, when an intelligent AP^3 participant receives the first operation for a transaction from a coordinator, it refers to its PAC list to identify the protocol used by the coordinator. If the coordinator is included in the list, it means that the coordinator is PrA and the participant has to deal with it as a PrA participant. Based on that, the participant marks the transaction as a PrA transaction in its protocol table and follows PrA with the transaction. That is, the AP^3 participant does not include any redo log records or read locks in the ACK of any operation that it executes for the transaction. Furthermore, at commit processing time of the transaction, the participant deals with the coordinator of the transaction as if it were a PrA participant with respect to messages and logging activities at its site. This includes the use of the traditional read-only optimisation if this optimisation is supported by the coordinator.

On the other hand, if the coordinator is not in the PAC list, it means that the coordinator is an intelligent AP^3. Based on that, the participant declares itself to the coordinator as an AP^3 participant in the ACK of the first operation that it executes for the transaction and proceeds with the transaction as an AP^3. An AP^3 participant has to declare itself in this case because the coordinator needs to know which participants are pre-existing PrA participants and which ones are AP^3. Otherwise, the coordinator cannot deal with both types of participants in a correct manner. Once the coordinator knows which participant is using which protocol, it proceeds with the transaction as discussed in the previous section.

The above solution works so long as the option of forward recovery is not used in the protocol. But, when this option is used, protocol violations are still possible. This is because the first operation submitted to a participant includes an FRF for a forward recoverable transaction which is not recognised if the participant is a pre-existing PrA participant. For this reason, each intelligent AP^3 coordinator keeps a list called *presumed-abort participants* (PAP) that includes the identities of pre-existing PrA participants. In this way, an AP^3 coordinator can determine whether a participant in a transaction's execution is a pre-existing PrA participant or not. If the participant is pre-existing PrA, the coordinator converts the transaction to become NFR before sending the first operation of the transaction to the participant and informs the other participants accordingly, as previously discussed.

It should be noted that the above choices to solving the problem of compatibility with pre-existing PrA was specially designed taking into consideration the sizes of the PAC and PAP lists and the possibility of eliminating both of them from the protocol in the long run. That is, as more database sites start to switch to the intelligent AP^3, both lists will shrink and eventually be eliminated from the intelligent AP^3.

4.5 Recovery in the intelligent AP^3

The intelligent AP^3 is resilient to both communication and site failures which are detected by timeouts, as in all other ACPs. In this section, the recovery aspects of the protocol in the presence of communication failures are first discussed. Then, the recovery aspects of the protocol in the presence of site failures are discussed.

4.5.1 Communication failures

There are four points during the execution of the intelligent AP^3 where a communication failure may occur while a site is waiting for a message. The first point is when a participant has no pending operation ACK for a transaction. If the transaction is 1PC at the participant, the participant is blocked until communication is re-established with the coordinator. Then, the participant inquires the coordinator about the transaction's status. The coordinator replies with either a final decision or a *still active* message. In the former case, the participant enforces the final decision. Then, if the decision is commit, the participant also acknowledges it. In the latter case, the participant waits for further operations. On the other hand, if the participant is 2PC, whether it is an AP^3 or a pre-existing PrA, it aborts the transaction.

The second point is when the coordinator of a transaction is waiting for an operation ACK from a participant. In this case, the coordinator aborts the transaction and submits a final abort decision to the rest of the participants. Similarly, a participant aborts a transaction when a communication failure occurs and the participant has a pending operation ACK. Notice that the coordinator of a transaction may commit the transaction in spite of communication failures with some participants so long as these participants are 1PC participants and have no pending operations' ACKs (as all the necessary redo log records that are needed for recovery are replicated at the coordinator's log).

The third point is when the 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 the transaction, it submits abort messages to all accessible participants and waits for the required ACKs (i.e., the ACKs of PrC participants when PrC is used with 2PC AP^3 participants). These ACKs are necessary for the coordinator in order to write an end log record and to forget the transaction. For an inaccessible participant, the participant is left blocked if has voted 'yes' before the communication failure and it is its responsibility to inquire about the transaction's status after the failure is fixed. If a participant inquires about the transaction after the failure is fixed, it includes the used protocol with the transaction in the inquiry message. If the transaction has already been forgotten by the coordinator, the inquiry message indicates either an IYV, PrA, or no-indication (for pre-existing PrA participant). The outcome of all the three choices implies a correct response which is an abort decision. Notice that if the protocol used with a blocked participant is PrC, it means that the coordinator could not have forgotten the transaction without the ACK of such a participant (as shown previously in Figure 2).

The fourth point is when the coordinator of a transaction is waiting for the ACKs of a final decision. Since the coordinator needs the ACKs in order to discard the information pertaining to the transaction from its protocol table and its log (during the garbage collection procedure), it re-submits the decision to the appropriate participants once communication failures are fixed. If the decision is commit, the coordinator re-submits a commit message to each inaccessible 1PC participant and pre-existing PrA. It also re-submits the commit decision to each inaccessible AP^3 participant when PrA is used with AP^3 participants. On the other hand, if the decision is an abort and PrC is used with AP^3 participants, the coordinator re-submits an abort message to each inaccessible one of them. When a 1PC or a pre-existing PrA 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, or enforces the decision and then sends back an ACK. Similarly, when a 2PC AP^3 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 and then acknowledges it.

4.5.2 Site failures

As implied above, when IYV was discussed, each database site is assumed to employ physical logging and uses an Undo/Redo crash recovery protocol in which the undo phase *precedes* the redo phase. Although this assumption can be relaxed whereby the intelligent AP^3 can be also combined with *logical* or *physiological* write-ahead logging (WAL) schemes, the details of such combinations are beyond the scope of this paper.

Based on the above assumption, a coordinator's site failure recovery aspects are first discussed; then, a participant's site failure recovery aspects are discussed.

4.5.2.1 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. If the coordinator is a pre-existing PrA coordinator, it will correctly handle intelligent AP^3 participants using its own failure recovery mechanisms. This is because each AP^3 participant knows that the recovering coordinator is a pre-existing PrA coordinator, using its own (PAC) list of coordinators, and will deal with the recovering coordinator accordingly. On the other hand, if the coordinator is an intelligent AP^3, it needs to consider the following types of transactions during its failure recovery:

- Each transaction with only a switch record: the coordinator knows that PrC was used with 2PC AP^3 participants as a switch record exists only when PrC is used with such participants. The coordinator also knows that the commit processing for such a transaction was interrupted before the decision was made. Based on that, it aborts the transaction and sends an abort message to each 2PC AP^3 participant recorded in the switch record and waits for its ACK. If there is any other participants (i.e., pre-existing PrA participants and AP^3 participants that did not request a protocol switch), it is the responsibility of these participants to inquire about the transaction after the coordinator has recovered. For a 1PC AP^3 participant, the inquiry message contains a flag that indicates the protocol used with the transaction (which is IYV) whereas for a pre-existing PrA participant, the inquiry message does not contain such a flag. In either case, the coordinator replies with an abort decision that is consistent with the presumption of the protocol used by the participant for the transaction (which is abort).
- Each transaction with a switch record and a commit record but without an end record: the coordinator knows that PrC was used with 2PC AP^3 participants and the transaction was committed. In this case, the coordinator, using the switch record that contains the set of participants, determines whether all participants are 2PC AP^3 or not. If all participants are 2PC AP^3, the coordinator considers the transaction complete and does not include it in its protocol table. Otherwise, the coordinator knows that not all the required ACKs from 1PC and pre-existing PrA participants were received before the failure. Based on that, it sends commit messages to all such participants and waits for their ACKs.

- Each transaction with a commit record but without a switch and end records: the coordinator knows that the protocol used with the transaction across all participants has to be a presumed-abort-based protocol (i.e., IYV or PrA) but not PrC with any of them. This is because the use of PrC with any participant requires a switch log record before the commit decision can be made. Based on that, the coordinator knows that IYV was used and no participant has switched protocol, IYV in conjunction with PrA were used with the transaction, or only PrA. All the three participants' mix possibilities require the ACK of the commit decision. Based on that, the coordinator sends a commit message to each participant recorded in the commit decision record (i.e., all participants) and waits for ACKs.

For all the three types of transactions above, when a participant receives a decision message, it means that the coordinator needs an ACK from the participant before it can forget the transaction. Based on that, the participant either replies with an ACK if it does not remember the transaction (because it has already received and enforced the decision prior to the failure), or enforces the decision and then sends back an ACK.

For all the other transactions, the coordinator can safely ignore them during its failure recovery and considers them as completed transactions. If a participant inquires about a transaction that the coordinator has already forgotten, the coordinator responds with a decision that matches the presumption of the protocol indicated in the inquiry message of the AP^3 participant (i.e., abort for IYV and PrA AP^3 participants; commit for PrC AP^3). For an inquiry message that does not include an indication about the used protocol with the transaction, it means that the inquiring participant has to be a pre-existing PrA participant. Based on that, the coordinator responds with an abort message.

4.5.2.2 Participant's failure

Since the stable log of a participant, in IYV, may not contain all the redo records of transactions committed by their perspective coordinators after a site failure, the participant needs to retrieve all missing records from such coordinators. Thus, at the beginning of the *analysis phase* of the restart procedure, the participant determines the largest LSN that is associated with the last record written onto its log and survived the failure. Then, the participant sends a *recovering* message that contains the largest LSN to all intelligent AP^3 coordinators recorded in its RCL. This LSN is used by intelligent AP^3 coordinators to determine missing redo log records at the participant which are replicated in their logs and are needed by the participant to fully recover. If the RCL is empty, it means that there were no intelligent AP^3 coordinators with active transactions at the participant's site before the participant's failure. In this case, there is no need for the participant to consult any AP^3 coordinator to recover its database as all the necessary information needed for recovery can be found in its own log without sending a recovering message to any coordinator. Otherwise, the participant waits for the reply messages to arrive from the intelligent AP^3 coordinators included in its RCL. Thus, the benefit of using the RCL is to limit the number of coordinators that need to be consulted during the recovery procedure of a participant after a failure.

While waiting for the reply messages to arrive from the coordinators, the *undo phase* can be performed, even potentially completed, and the *redo phase* can be initiated. That is, the participant recovers those aborted and committed transactions that have decision records already stored in its stable log while waiting for the reply messages to arrive from the coordinators. This ability of overlapping the undo phase with the resolution of the status of active transactions and the repairing of the redo part of the log partially masks the effects of dual logging and communication delays. Note that because of the use of WAL, all the required undo log records that are needed to eliminate the propagated effects of any transaction on the database are always available in the participant's stable log and never replicated at the intelligent AP^3 coordinators' sites.

When an intelligent AP^3 coordinator receives a recovering message from a participant, it means that the participant has failed and is recovering from a failure. Based on that, the coordinator checks its protocol table to determine each transaction that the failed participant has executed some of its operations and the transaction is either still active in the system (i.e., still executing at other sites and no decision has been made about its final status, yet) or has terminated but did not finish the protocol (i.e., a final decision has been made but the participant was not aware of the decision prior to its failure). For each transaction that is marked as forward recoverable and is still active in the system, the coordinator responds with the list of the redo log records that are stored in its log and have LSNs greater than the one that was included in the recovering message of the participant along with all the read-locks received from the participant for the transaction and stored in its PLT. On the other hand, for each transaction that is finally committed, the coordinator responds with a commit status along with a list of all the transaction's redo records that are stored in its log and have LSNs greater than the one that was included in the recovering message of the participant.

All these responses and redo log records are packaged in a single *repair* message and sent back to the participant. If a coordinator has no active transactions and all terminated transactions have been acknowledged (according to the intelligent AP^3 protocol) as far as the failed participant is concerned, the coordinator sends an ACK repair message, indicating to the participant that there are no transactions to be recovered. Notice that this latter situation may occur as a participant always force writes its RCL ahead of the execution of an operation received from a previously inactive coordinator. Thus, if the participant fails after it has forced its list into the stable log but before it has started the execution of the operation, the identity of the coordinator will be included in the RCL during the recovery procedure of the participant even though the coordinator may have already aborted and forgotten the transaction due to the failure.

When the participant receives the required reply messages from all the intelligent AP^3 coordinators recorded in its RCL, the participant repairs its log and lock table. Then, it completes the redo phase by replaying the effects of both committed as well as still-active transactions. For any unmentioned prepared-to-commit transaction in the received repair messages, the participant considers the transaction aborted. Once the redo phase is completed, the participant sends back the required decisions' ACKs as during normal processing and then, resumes normal processing. During the recovery procedure, the participant also resolves the states of any prepared-to-commit transactions that were coordinated by pre-existing PrA coordinators. This is accomplished by identifying such transactions during the analysis phase and inquiring their coordinators about their final states. Thus, instead of sending a single recovering message to each pre-existing PrA coordinator, which would not be understood by such a coordinator, the participant inquires the coordinator of each prepared-to-commit transaction about the final status of the transaction individually, in accordance with PrA.

The case of an overlapped coordinator and participant failure is handled using the same procedures discussed above. However, if the failed coordinator is in the RCL of a recovering participant, the coordinator needs to recover first before responding to the participant's recovering message. Meanwhile, the recovery procedure of the participant is suspended until the failed coordinator is recovered.

5 Analytical evaluations

To evaluate the performance of the intelligent AP^3 during normal processing and compare it with the performance of the other protocols, the customary used performance metrics (i.e., log, message and time complexities) are adopted. Furthermore, the adopted divisions to these metrics are as appeared in Al-Houmaily (2010), which are as follows:

- *Log overhead* corresponds to *log complexity* and is divided into '*Total*' number of log records and the portion of total log records that needs to be '*Forced*' written.
- *Log delays* represents the number of '*Sequential*' forced log writes that are required up to the point that the commit/abort decision is made.
- *Message overhead* corresponds to *message complexity* and is divided into '*Total*' number of messages and the number of messages required for a '*Decision*' to be made and propagated to the participants.
- *Message delays* corresponds to *time complexity* and is divided into the number of required sequential messages up to the '*Decision*' point and the number of required sequential messages for the release of all the '*Locks*'.

Table 1 compares the costs of the different protocols, on a per transaction basis, using the above divisions to the basic performance metrics. The first portion of the table compares the costs of the different protocols to commit a transaction whereas the second portion compares their costs to abort a transaction. The rows titled 'Basic AP^3 (1PC)' denotes the use of the basic AP^3 protocol when *all* participants are 1PC. The rows titled 'Basic AP^3 (PrC)' denotes the use of the basic AP^3 when *all* participants are 2PC and PrC is used. The rows titled 'Basic AP^3 (PrA)' denotes the use of the basic AP^3 when *all* participants are 2PC and PrA is used. The rows titled 'Basic AP^3-MIX (PrC)' denotes the use of the basic AP^3 in the presence of both 1PC and 2PC participants with PrC used with 2PC participants. The rows titled 'Basic AP^3-MIX (PrA)' denotes the use of the basic AP^3 in the presence of both 1PC and 2PC participants with PrA used with 2PC participants. The rows that begin with 'Intelligent AP^3' have the same meanings as the corresponding ones used with the basic AP^3 except that they denote the use of the intelligent AP^3 instead of the basic AP^3. In the table, '*n*' denotes the total number of sites participating in the execution of a transaction (excluding the coordinator's site) whereas '*p*' denotes the number of 1PC participants.

Table 1 shows, for example, that the cost of committing a transaction, in 2PC, is *exactly* the same as the cost of aborting a transaction. For either case, the '*Total Log Overhead*' is '$2n + 2$', out of which '$2n + 1$' are '*Forced*' log records. The '*Sequential Log Delays*' for a decision to be made, in 2PC, is '2': one at the participant(s) and one at the coordinator. The '*Total Message Overhead*' is '$4n$', out of which '$3n$' are required for a '*Decision*' to be made and propagated to all participants. The sequential '*Decision Message Delays*' is '2' as 2PC requires two sequential messages for a decision to be made whereas the '*Locks Message Delays*' is '3' as 2PC requires three sequential messages for a decision to reach a participant and releases the locks held by a transaction.

Table 1 Protocols' costs

A. Commit decision							
Metric / *ACP*	*Log overhead*		*Log delays*	*Message overhead*		*Message delays*	
	Total	*Forced*	*Sequential*	*Total*	*Decision*	*Decision*	*Locks*
2PC	$2n+2$	$2n+1$	2	$4n$	$3n$	2	3
PrC	$2n+2$	$n+2$	3	$3n$	$3n$	2	3
PrA	$2n+2$	$2n+1$	2	$4n$	$3n$	2	3
IYV	$n+2$	1	1	$2n$	n	0	1
Basic AP^3 (IPC)	$n+2$	1	1	$2n$	n	0	1
Basic AP^3 (PrC)	$2n+2$	$n+2$	3	$3n$	$3n$	2	3
Basic AP^3 (PrA)	$2n+2$	$2n+1$	2	$4n$	$3n$	2	3
Basic AP^3-Mix (PrC)	$2(n-p)+p+3$	$n+2$	3	$3(n-p)+2p$	$3(n-p)+p$	2	3
Basic AP^3-Mix (PrA)	$2(n-p)+p+2$	$2(n-p)+1$	2	$4(n-p)+2p$	$3(n-p)+p$	2	3
Intelligent AP^3 (IPC)	$n+2$	1	1	$2n$	n	0	1
Intelligent AP^3 (PrC)	$2n+2$	$n+2$	3	$3n$	$3n$	2	3
Intelligent AP^3 (PrA)	$2n+2$	$2n+1$	2	$4n$	$3n$	2	3
Intelligent AP^3-Mix (PrC)	$2(n-p)+p+3$	$n+2$	3	$3(n-p)+2p$	$3(n-p)+p$	2	3
Intelligent AP^3-Mix (PrA)	$2(n-p)+p+2$	$2(n-p)+1$	2	$4(n-p)+2p$	$3(n-p)+p$	2	3

Table 1 Protocols' costs (continued)

	B. Abort decision						
Metric	*Log overhead*		*Log delays*	*Message overhead*		*Message delays*	
ACP	*Total*	*Forced*	*Sequential*	*Total*	*Decision*	*Decision*	*Locks*
2PC	$2n+2$	$2n+1$	2	$4n$	$3n$	2	3
PrC	$2n+2$	$2n+1$	2	$4n$	$3n$	2	3
PrA	$2n$	n	1	$3n$	$3n$	2	3
IYV	n	0	0	n	n	0	1
Basic AP^3 (IPC)	n	0	0	n	n	0	1
Basic AP^3 (PrC)	$2n+2$	$2n+1$	2	$4n$	$3n$	2	3
Basic AP^3 (PrA)	$2n$	n	1	$3n$	$3n$	2	3
Basic AP^3-Mix (PrC)	$2(n-p)+p+2$	$2(n-p)+1$	2	$4(n-p)+p$	$3(n-p)+p$	2	3
Basic AP^3-Mix (PrA)	$2(n-p)+p$	$n-p$	1	$3(n-p)+p$	$3(n-p)+p$	2	3
Intelligent AP^3 (IPC)	n	0	0	n	n	0	1
Intelligent AP^3 (PrC)	$2n+2$	$2n+1$	2	$4n$	$3n$	2	3
Intelligent AP^3 (PrA)	$2n$	n	1	$3n$	$3n$	2	3
Intelligent AP^3-Mix (PrC)	$2(n-p)+p+2$	$2(n-p)+1$	2	$4(n-p)+p$	$3(n-p)+p$	2	3
Intelligent AP^3-Mix (PrA)	$2(n-p)+p$	$n-p$	1	$3(n-p)+p$	$3(n-p)+p$	2	3

Table 1 also confirms that the costs of both the basic AP^3 and the intelligent AP^3 protocols are identical, given the same participants' mix, for committing and aborting transactions, respectively. That is, the cost of the 'Basic AP^3 (1PC)' to commit a transaction, for example, is the same as the cost of the 'Intelligent AP^3 (1PC)' to commit the same transaction. It is clear from Table 1 that the two AP^3 variants perform as IYV when all participants are 1PC, outperforming 2PC and its variants in all performance measures including the number of '*Sequential Log delays*' to reach a decision as well as the '*Forced Log Overhead*'. For the commit case, IYV and the two AP^3 variants require only one forced log write whereas, for the abort case, neither of the two AP^3 variants nor IYV force write any log records. When all participants are 2PC and the used protocol variant is PrC, the AP^3 variants perform as PrC in all performance measures for the commit case as well as the abort case. Similarly, when all participants are 2PC and the used protocol is PrA, the AP^3 variants perform as PrA in all performance measures for the commit case as well as the abort case. When there is participants' mix, which is the general case, the performance of the AP^3 variants exhibit the behaviour of either PrC or PrA, depending on the protocol variant used, with respect to the sequential performance metrics. That is, the AP^3 variants have the same number of '*Sequential Log Delays*', '*Decision Message Delays*' and '*Locks Message Delays*' as either PrC or PrA. The performance of the AP^3 variants with respect to the '*Total Log Overhead*' and '*Total Message Overhead*' depends on the participants' mix, which is, in general, less than that of PrC and PrA. Thus, in spite of the additional applicability features in the intelligent AP^3, there is no added overhead on its performance, using the common performance metrics, compared to the basic AP^3.

Table 2 Protocols' costs for read-only transactions

Metric / *ACP*	*Log overload*		*Log delays*	*Message overload*		*Message delays*	
	Total	*Forced*	*Sequential*	*Total*	*Decision*	*Decision*	*Locks*
PrC	2	1	1	2*n*	2*n*	2	1
PrA	0	0	0	2*n*	2*n*	2	1
IYV	0	0	0	*n*	0	0	1
Intelligent AP^3	0	0	0	*n*	0	0	1

Table 2 shows the costs of the different ACPs for read-only transactions. The costs in the table are calculated assuming the use of the standard read-only optimisation with PrC and PrA. Furthermore, it is assumed that read-only transactions are finally aborted since it is cheaper to abort read-only transactions than to commit them in both PrC and PrA (Al-Houmaily et al., 1997b). In the table, all the protocols have the same costs with respect to the sequential '*Locks Message Delays*' to release the resources held by read-only transactions at the participants (i.e., only a single message). However, the intelligent AP^3 and IYV, which have exactly the same costs for read-only transactions, dominate PrC and PrA with respect to the number of sequential '*Decision Message Delays*' to reach the decision point (at a coordinator's site). This is because they eliminate the (explicit) voting phase of 2PC which pulls the read-only votes of the participants in both PrC and PrA. As for the '*Total Log Overhead*', '*Forced Log Overhead*' and '*Sequential Log Delays*' are concerned, only PrC suffers from these log writes and delays due to the forced initiation log record at the coordinator's site. Lastly, both of the intelligent AP^3 and IYV require '*n*' in the '*Total Message Overhead*' instead of '2*n*', which is the case in PrC and PrA.

6 Related works

The *presumed any* (PrAny) protocol (Al-Houmaily and Chrysanthis, 1996, 1999) was the first protocol to recognise the problem of presumption incompatibilities. It interoperates PrN, PrA and PrC in the context of *multidatabase systems*, a special case of heterogeneous distributed databases in which each database site is pre-existing, supporting its own local applications and users, and has some degree of local autonomy. Specifically, the protocol was used to demonstrate the difficulties that arise when one attempts to interoperate participants that adopt ACPs with different presumptions about the outcome of terminated transactions in the same environment and, more importantly, to introduce the *'operational correctness criterion'* and the notion of *'safe state'*. Operational correctness means that *all* participating sites should be able, not only to reach an agreement but also, to (eventually) forget the outcome of terminated transactions. On the other hand, the safe state means that, for any operationally correct ACP, the coordinator should be able to reach a state in which it can reply to the inquiry messages of the participants, in a consistent manner, without having to remember the outcome of terminated transactions forever. Formal definitions of operational correctness and safe state can be found in Al-Houmaily and Chrysanthis, (1999) and Al-Houmaily, (2008).

Systematic analyses to the sources of incompatibilities when interoperating 2PC protocols with 1PC protocols, such as IYV, led to the design of the *integrated two-phase commit* (I-2PC) protocol (Al-Houmaily, 2008). That is, besides identifying the sources of incompatibilities when one attempts to interoperate 2PC with 1PC protocols, I-2PC generalises PrAny by incorporating IYV, which is a 1PC protocol, besides PrN, PrA and PrC.

Whereas the motivation behind PrAny and I-2PC was mainly on achieving atomicity of transactions, in accordance to the operational correctness criterion and in spite of the heterogeneity of the used ACPs by the participants, the main motivation behind the design of *presumed-either* (PE) (Attaluri and Salem, 2002), *dynamic presumption two-phase commit* (DPr-2PC) (Yu and Pu, 2007), *one-two phase commit* (1-2PC) (Al-Houmaily and Chrysanthis, 2004a, 2004b) and AP^3 (Al-Houmaily, 2005) was for performance reasons. That is, enhancing the performance of atomic commit processing through the use of adaptive integrated ACPs rather than using only a single ACP at all times and with all transactions. For this reason, these protocols were designed implicitly assuming *homogenous* distributed database systems in which all sites deploy the same (integrated) ACP.

PE interoperates both PrA and PrC side-by-side in the same environment with the protocol choice being made on a *per transaction* basis and at the beginning of the commit processing stage of the transaction. It was designed motivated by the fact that transactions tend to commit when they reach their commit processing stage and that forcing the initiation record in PrC represents its main performance efficiency obstacle for committing transactions when compared with the performance of PrA for aborting transactions. Thus, it was designed to eliminate the *forcing* of the initiation record of PrC *whenever possible*. In PE, this is accomplished by having the coordinator of a transaction to write a non-forced initiation (or participants') record into the log buffer (in main memory) each time a new participant joins in the execution of the transaction. If the last initiation record that was written for the transaction is forced written (i.e., piggybacked) onto the stable log due to a subsequent forced log write, possibly on behalf of some other transaction, or a flush to the log buffer due to an overflow, PE behaves much like PrC for the rest of the protocol. Otherwise, PE behaves much like PrA.

Instead of having to use the same ACP by *all participants* for the commit processing of a transaction, DPr-2PC provides each participant with the flexibility to select its preferred protocol among PrA and PrC on a dynamic basis. That is, each participant has the freedom to select the most appropriate ACP variant with respect to performance, from its own point of view, before the beginning of the commit processing stage at the coordinator's site. In a manner similar to PE, this is accomplished using a flag, but the flag is used as part of the ACK of an operation that a participant executes on behalf of the transaction. This is in contrast to being part of the prepare-to-commit message that a coordinator sends to each participant, which is the case in PE. Of course, the chosen ACP by a participant, in DPr-2PC, can be overridden by the coordinator with a different flag value which is sent to the participant as part of the prepare-to-commit message (Yu and Pu, 2007).

Rather than interoperating only 2PC protocol variants, the1-2PC interoperates 1PC besides 2PC protocols. It was designed to achieve the performance of 1PC protocols in the absence of deferred consistency constraints and the wide applicability of 2PC protocols in their presence. Specifically, 1-2PC is a combination of IYV and PrC. The 1-2PC protocol integrates these two protocols in a dynamic fashion, depending on the behaviour of transactions and system requirements, in spite of their incompatibilities.

The basic AP^3 extends 1-2PC by incorporating PrA besides PrC. That is, it integrates IYV, PrC and PrA. As in 1-2PC, AP^3 starts as 1PC at each participant and, only when necessary, it dynamically switches to 2PC. Thus, it achieves the performance advantages of 1PC whenever possible and, at the same time, the wide applicability of 2PC. However, when 2PC is to be used, unlike 1-2PC which has the ability to switch to only PrC, AP^3 has the ability to switch to the most appropriate 2PC variant to further enhance the performance of commit processing. In AP^3, the choice of 2PC depends on the anticipated results of the evaluation of the deferred consistency constraints. That is, if consistency constraints tend to be violated and consequently transactions tend to abort, PrA is used with 2PC participants. Otherwise, PrC is used. In this way, AP^3 supports deferred constraints without penalising those transactions that do not require them. Furthermore, as in 1-2PC, AP^3 achieves this advantage on a per participant basis within the same transaction in spite of the incompatibilities between 1PC and 2PC protocols.

The intelligent AP^3 builds on the basic AP^3 by addressing four issues, representing the focus of this article. Read-only transactions were not mentioned in the above previous works, except for 1-2PC, assuming that the traditional read-only optimisation can be incorporated in the above protocols in a straight forward manner. Forward recovery, as defined in this article, was not discussed except for IYV in which transactions have the option of using forward recovery without addressing how to handle transactions that create contexts at the participants' sites. Handling large amounts of updated data in 1PC protocols or any protocol that operates as1PC at certain times were never addressed in any previous work, according to our knowledge. Backward compatibility was addressed in the *generalised presumed abort* (GPrA) (Samaras et al., 1993; Mohan et al., 1993) and DPr-2PC (Yu and Pu, 2007).

GPrA is an IBM protocol that was designed with the aim of incorporating the standard PrA protocol (X/Open Company Limited, 1996; ISO, 1998) in the (System Network Architecture) SNA LU6.2 (IBM, 1990, 1993). This architecture is based on the *peer-to-peer* paradigm and defines the distributed transaction environment, the commit protocols and their synchronisation. The architecture initially defined a form of PrN that supports heuristic decisions. This is in order to handle real-world applications that cannot sustain long delays, presumably due to failures. Thus, a transaction in this architecture can be heuristically committed or aborted and later on fixing any damages to the consistency of the database in the event that the heuristic decision taken did not conform to the transaction's final outcome. To incorporate PrA in the architecture, it had to be modified in order to also support heuristic decisions and to be compatible with the previously defined PrN. This modification to PrA led to GPrA and made it a superset to the originally used PrN variant (Samaras and Nikolopoulos, 1995).

DPr-2PC addresses backward compatibility in the presence of only pre-existing PrA participants but not in the presence of pre-existing PrA coordinators. Thus, resolving this problem was much simpler as it is assumed, at commit initiation time, that a participant in a transaction's execution is pre-existing PrA so long as the participant did not declare itself otherwise during the course of the execution of the transaction. However, as shown in this article, achieving backward compatibility is not as simple when both pre-existing PrA participants and pre-existing PrA coordinators are present in the same environment.

The above research works represent the most closely related works to the presented one. However, other works have been performed in the context of other distributed database environments that have different constraints/characteristics such as *multidatabase systems*, *mobile and ad hoc networks*, and *real-time database systems*. In multidatabase systems, each database site is assumed to be pre-existing, supporting its own local applications and users. The challenge in this type of environments is to ensure the atomicity of transactions that execute across the different database sites in the environment without sacrificing the autonomy of any site. One study to Al-Houmaily, (2009) overviews and classifies the different proposals of ACPs for this type of environments.

In mobile and ad hoc networks, the limited energy available in mobile devices and their vulnerability to loss and damage in addition to the limited bandwidth of the communication network represent constraints on the design of efficient and reliable ACPs (Obermeier et al., 2009; Nouali-Taboudjemat et al., 2010; Obermeier and Böttcher 2010). Ayari et al. (2011) provides a comprehensive treatment to the sources of disturbances/perturbations that affect the atomic termination of transactions in an efficient manner and classifies these disturbances/perturbations into different categories. Furthermore, a representative ACP that can cope with the disturbances/perturbations of each category is proposed.

In real-time database systems, the challenge is to design an ACP that allows for the minimum number of transactions that miss their deadlines thereby, reducing the amount of transactions' abort and enhancing the overall efficiency of the system. Gupta et al. (1997), Qin and Liu (2003), and Shanker et al. (2008) are examples to research works that deal with this type of environments.

7 Conclusions and future works

Piggybacking control information among database sites regarding transactions is a powerful technique when intelligently used in the design of ACPs. This technique was comprehensively and effectively used in the design of the *intelligent* AP^3. Although the new protocol can be viewed as an extension to the previously known (basic) AP^3, the new extensions represent significant contributions in the design of highly efficient and practical ACPs. Specifically, through piggybacking, the intelligent AP^3 addresses and resolves four important issues in the design of the basic AP^3. These issues can be classified into two categories: those that enhance *efficiency*, and those that enhance *applicability*. The efficiency issues are concerned with:

1 efficiently handling read-only transactions

2 utilising the option of forward recovery with transactions that are context-free while still able to identify those that create run-time contexts and prohibit them from utilising this option.

On the other hand, the applicability issues are concerned with:

1 identifying transactions that update LAD and switching them to a protocol that has the ability to allow them to continue their execution without having to propagate LAD from one site to another

2 resolving incompatibilities with pre-existing database sites that adopt the standard ACP.

Addressing the above four issues and resolving them in the intelligent AP^3 makes the protocol a highly appealing choice for adoption, among existing ones, in the design of future generations' database management systems. This is supported by the fact that the mechanisms deployed in the intelligent AP^3 do not add significant complications in the design of the protocol compared to existing ones. Thus, the protocol can be easily realised and implemented as part of any database management system. However, to further confirm the performance superiority of the protocol in comparison to previously existing ones, more performance examinations, preferably, through empirical simulation studies, are required besides the analytical evaluations presented in this article. Such studies are left as part of future works. In addition, extending the protocol to the multi-level transaction execution model, the most widely used model in practice, and examining the possible usage of a form of context preservation/propagation techniques to enhance the option of forward recovery that is incorporated in the protocol remain open areas for research.

References

Abdallah, M., Guerraoui, R. and Pucheral, P. (1998) 'One-phase commit: does it make sense?', *Proc. of the Int'l Conference on Parallel and Distributed Systems*.

Abdallah, M., Guerraoui, R. and Pucheral, P. (2002) 'Dictatorial transaction processing: atomic commitment without veto right', *Distributed and Parallel Databases*, Vol. 11, No. 3, pp.239–268.

Al-Houmaily, Y. (2005) 'On interoperating incompatible atomic commit protocols in distributed databases', *Proc. of the 1st IEEE Int'l Conf. on Computers, Comm., and Signal Processing*, Kuala Lumpur, Malaysia.

Al-Houmaily, Y. (2008) 'Incompatibility dimensions and integration of atomic commit protocols', *Int'l Arab J. of Inf. Tech.*, Vol. 5, No. 4, pp.381–392.

Al-Houmaily, Y. (2009) 'Schemes for atomicity in heterogeneous database environments', *Applied Computing & Informatics – The Saudi Computer Journal*, Vol. 7, No. 2, pp.31–62.

Al-Houmaily, Y. (2010) 'Atomic commit protocols, their integration, and their optimisations in distributed database systems', *Int'l J. of Intelligent Information and Database Systems*, Vol. 4, No. 4, pp.373–412.

Al-Houmaily, Y. and Chrysanthis, P. (1996) 'Dealing with incompatible presumptions of commit protocols in multidatabase systems', *Proc. of the ACM SAC*, Philadelphia, USA.

Al-Houmaily, Y. and Chrysanthis, P. (1999) 'Atomicity with incompatible presumptions', *Proc. of the 18th ACM PODS*.

Al-Houmaily, Y. and Chrysanthis, P. (2000) 'An atomic commit protocol for gigabit-networked distributed database systems', *J. of Systems Architecture*, Vol. 46, No. 9, pp.809–833.

Al-Houmaily, Y. and Chrysanthis, P. (2004a) '1-2PC: the one-two phase atomic commit protocol', *Proc. of the ACM SAC*, Nicosia, Cyprus.

Al-Houmaily, Y. and Chrysanthis, P. (2004b) 'ML-1-2PC: an adaptive multi-level atomic commit protocol', *Proc. of the 8th East European Conf. on the Advances in Databases and Inf. Systems*, Budapest, Hungary.

Al-Houmaily, Y. and Samaras, G. (2009) 'Two-phase commit', in Liu, L. and Tamer Özsu, M. (Eds.): *Encyclopedia of Database Systems*, Springer.

Al-Houmaily, Y., Chrysanthis, P. and Levitan, S. (1997a) 'Enhancing the performance of presumed commit protocol', *Proc. of the 12th ACM Annual Symposium on Applied Computing*, San Jose, California.

Al-Houmaily, Y., Chrysanthis, P. and Levitan, S. (1997b) 'An argument in favor of the presumed commit protocol', *Proc. of the 13th ICDE*.

Attaluri, G. and Salem, K. (2002) 'The presumed-either two-phase commit protocol', *IEEE TKDE*, Vol. 14, No. 5, pp.1190–1196.

Ayari, B., Khelil, A. and Suri, N. (2011) 'On the design of perturbation-resilient atomic commit protocols for mobile transactions', *ACM Transactions on Computer Systems*, Vol. 29, No. 3, pp.7:1–7:36.

Chrysanthis, P., Samaras, G. and Al-Houmaily, Y. (1998) 'Recovery and performance of atomic commit processing in distributed database systems', in Kumar, V. and Hsu, M. (Eds.): *Recovery Mechanisms in Database Systems*, pp.370–416, Prentice Hall.

Gray, J. (1978) 'Notes on database operating systems', in Bayer, R., Graham, R.M. and Seegmuller, G. (Eds.): *Operating Systems: An Advanced Course*, LNCS, Vol. 60, pp.393–481, Springer-Verlag.

Gray, J. and Reuter, A. (1993) *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann Inc., USA.

Gupta, R., Haritsa, J. and Ramamritham, K. (1997) 'Revisiting commit processing in distributed database systems', *Proc. of the ACM SIGMOD Int'l Conference on Management of Data*, pp.486–497.

IBM (1990) *Systems Network Architecture LU 6.2 Reference: Peer Protocols*, Document No. SC31-6808-1.

IBM (1993) *Systems Network Architecture Transaction Programmer's Reference Manual for LU Type 6.2*, Document No. SC30-3084-5.

ISO (1998) *Open Systems Interconnection – Distributed Transaction Processing – Part 1: OSI TP Model*, ISO/IEC 10026-1.

ISO (2008) *Information Technology – Database Languages – SQL – Part 2: Foundation (SQL/Foundation)*, ISO/IEC 9075-2.

Lampson, B. (1981) 'Atomic transactions', in Lampson, B., Paul, M. and Siegert, H.J. (Eds.): *Distributed Systems: Architecture and Implementation – An Advanced Course*, LNCS, Vol. 105, pp.246–265, Springer-Verlag.

Lampson, B. and Lomet, D. (1993) 'A new presumed commit optimization for two phase commit', *Proc. of the 19th Int'l Conf. on VLDB*, pp.630–640.

Lee, I. and Yeom, H. (2002) 'A single phase distributed commit protocol for main memory database systems', *Proc. of the 6th Int'l Parallel and Distributed Processing Symposium*.

Mohan, C., Britton, K., Citron, A. and Samaras, G. (1993) 'Generalized presumed abort: marrying presumed abort and SNA's LU 6.2 commit protocols', *Proc. of the 5th Int'l Workshop on High Performance Transaction Systems*.

Mohan, C., Lindsay, B. and Obermarck, R. (1986) 'Transaction management in the R^* distributed data base management system', *ACM TODS*, Vol. 11, No. 4, pp.378–396.

Nouali-Taboudjemat, N., Chehbour, F. and Drias, H. (2010) 'On performance evaluation and design of atomic commit protocols for mobile transactions', *Distributed and Parallel Databases*, Vol. 27, No. 1, pp.53–94.

Obermeier, S. and Böttcher, S. (2010) 'A DBMS for mobile transactions using bi-state-termination', *Int'l Journal of Database Management Systems*, Vol. 2, No. 2, pp.141–159.

Obermeier, S., Böttcher, S., Hett, M., Chrysanthis, P. and Samaras, G. (2009) 'Blocking reduction for distributed transaction processing within MANETs', *Distributed and Parallel Databases*, Vol. 25, No. 3, pp.165–192.

Qin, B. and Liu, Y. (2003) 'High performance distributed real-time commit protocol', *Journal of Systems and Software*, Vol. 68, No. 2, pp.145–152

Samaras, G. and Nikolopoulos, S. (1995) 'Algorithmic techniques incorporating heuristic decisions to commit protocols', *Proc. of the 21st Euromicro Conf.*, Como, Italy.

Samaras, G., Britton, K., Citron, A. and Mohan, C. (1993) 'Enhancing SNA's LU6.2 sync point to include presumed abort protocol', IBM Technical Report TR29.1751, IBM Research Triangle Park.

Shanker, U., Mesra, M. and Sarje, A. (2008) 'Distributed real time database systems: background and literature review', *Distributed and Parallel Databases*, Vol. 23, No. 2, pp.127–149.

Stamos, J. and Cristian, F. (1993) 'Coordinator log transaction execution protocol', *Distributed and Parallel Databases*, Vol. 1, No. 4, pp.383–408.

X/Open Company Limited (1996) *Distributed Transaction Processing: Reference Model*, Version 3 (X/Open Doc. No. 504).

Yu, W. and Pu, C. (2007) 'A dynamic two-phase commit protocol for adaptive composite services', *Int'l J. of Web Services Research*, Vol. 4, No. 1, pp.80–100.