On Interoperating Incompatible Atomic Commit Protocols in Distributed Databases

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

Abstract

This paper proposes an adaptive participant's presumption protocol (AP^3) that can be used to atomically commit Internet transactions. AP^3 interoperates a one-phase commit protocol, namely, implicit-yes vote, and two-phase commit protocols, namely, presumed abort and presumed commit, in a dynamic fashion. Thus, it offers the performance advantage of the combined protocols, whenever possible, while still providing the wide applicability of two-phase commit protocols. This is achieved in spite of the incompatibilities between atomic commit protocols that lead to the general practice of adopting a single atomic commit protocol in any distributed database system.

Keywords: Two-Phase Commit, Voting Protocols, Transaction Processing, Database Systems, Internet Computing.

1. Introduction

Atomicity ensures that a transaction executes as a single, indivisible atomic unit of work. The transaction either commits and its effects become permanent on the states of all the database sites that it has visited, or aborts and its effects are obliterated from them as if it had never existed. This "all-or-nothing" property is achieved by implementing an atomic commit protocol (ACP) as part of any distributed database management system.

The *two-phase commit* (2PC) [9, 11] is the most widely used and optimized ACP [8]. It ensures atomicity 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 continuous interest in developing more efficient ACPs and optimizations (e.g., [10, 7, 4, 2, 13, 5, 6]), especially in the context of modern electronic services and electronic commerce environments that are characterized by high volume of transactions. Most notable developments are *one-phase commit* (1PC) protocols [14, 4, 2].

1PC protocols reduce both message and log complexities of 2PC at the expense of placing assumptions on transactions or the database management systems. Whereas some of these assumptions are realistic, other assumptions can be considered very restrictive for some applications [1, 4]. 1PC protocols, including *implicit yes-vote* (IYV) [4], assume that each transaction's operation is acknowledged after its execution at a participating site. An operation acknowledgment 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 participating site. Although this assumption is not too restrictive since commercial systems implement rigorous schedulers and database standards specify operation acknowledgment, it clearly restricts the implementation of applications that wish to utilize the option of *deferred consistency constraints validation*. This option is part of the SQL standards 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. When this option is used, the evaluation of deferred constraints needs to be synchronized across all participating database sites at commit time of the transaction.

For the above reason, this paper proposes a new ACP called *adaptive participant's presumption protocol* (AP^3) that interoperates IYV and the best known two-phase commit variants, namely, presumed abort and presumed commit [12]. In this respect, AP^3 can be viewed as an extension to the *one-two phase commit protocol* (1-2PC) [5] that incorporates presumed abort besides presumed commit. As in 1-2PC, AP^3 starts as 1PC and dynamically switches to 2PC only when necessary. Thus, it achieves the performance advantages of 1PC 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 presumed commit, AP^3 switches to the most appropriate 2PC variant to enhance the performance of commit processing, depending on the anticipated results of the evaluation of consistency constraints. That is, if consistency constraints tend to be vio-

lated and consequently transactions tend to abort, presumed abort is used with 2PC participants. Otherwise, presumed commit is used. In this way, AP^3 supports deferred constraints without penalizing those transactions that do not require them. Furthermore, as in 1-2PC, AP^3 achieves this advantage on a participant (i.e., cohort) basis within the same transaction in spite of the incompatibilities between 1PC and 2PC protocols.

In the next section, 1PC and 2PC protocols are briefly reviewed. Section 3 introduces AP^3 while Section 4 evaluates the performance of AP^3 analytically by comparing it to the best known ACPs. Section 5 concludes the paper.

2. Background

A distributed/Internet transaction accesses data by submitting operations to its *coordinator* which is assumed to be the *transaction manager* at the site where the transaction is 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 (cohort). When the transaction finishes its execution and submits its final commit request, the coordinator initiates an ACP such as the 2PC or one of its variants.

The *presumed abort* protocol (PrA) is the best known 2PC variant and is designed to reduce the cost for aborting transactions [12]. As in the basic 2PC protocol, PrA consists of two phases, namely a *voting phase* and a *decision phase*. During the voting phase, the coordinator requests all cohorts to *prepare to commit* whereas, during the decision phase, the coordinator either commits the transaction if *all* cohorts are prepared-to-commit (voted "yes"), or aborts the transaction if any cohort has decided to abort (voted "no"). While in a prepared-to-commit state, a cohort can neither commit nor abort the transaction until it receives the coordinator's final decision.

However, unlike the basic 2PC, when a coordinator in PrA decides to abort a transaction, it does not force write an abort decision onto its stable log, which is the case for a commit decision. It just sends abort messages to all cohorts that have voted "yes" and discards all information about the transaction from its protocol table. That is, the coordinator of an aborted transaction does not write any log records or wait for acknowledgments (ACKs). Since cohorts do not ACK abort decisions, they also do not force write such decisions. They only write abort decisions in the log buffer without forcing it onto the stable log. After a coordinator's or a cohort's failure, if the cohort inquires about a transaction that has been aborted, the coordinator, not remembering the transaction, will direct the cohort to abort the transaction (by presumption).

As opposed to PrA, the *presumed commit* protocol (PrC) is designed to reduce the cost for committing transactions [12]. This is achieved by interpreting missing information about transactions as commit decisions. Thus, unlike PrA, the absence of information in PrC means commitment. For this reason, 2PC, PrA and PrC are incompatible protocols. The three protocols are incompatible not only because of the semantics of messages but also because of their contradicting presumptions about the outcome of terminated transactions in the absence of information about them [3].

The 2PC variants are usually criticized on the grounds of their performance drawback especially for short transactions, which is a common characteristic of Internet transactions. For this reason and given the high reliability of today's database servers besides the increasing bandwidth of communication networks, a new class of ACPs have been recently proposed. The new set of protocols are one-phase commit (1PC). These protocols consist of only a single phase which is the decision phase of 2PC. The (explicit) voting phase is eliminated by overlapping it with the ACKs of the database operations [14, 4]. This principle is used in *implicit yes-vote* (IYV), on which AP^3 is based. IYV assumes that each site deploys (1) a *strict two-phase locking* (S2PL) for concurrency control and (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 transaction without the need for the voting phase of 2PC [4].

3. The Adaptive Participant's Presumption Protocol (AP^3)

1PC protocols are more efficient than 2PC and its variants but the assumptions that they place on transaction management make them less appropriate than 2PC to be adopted in commercial systems. Although some of these assumptions can be relaxed [1], their biggest limitation is their inability to support deferred consistency constraints. AP^3 supports application systems that use deferred constraints without penalizing those that they do not require them.

AP^3 operates as a 1PC protocol as long as the voting phase is not necessary. When the voting phase is needed with some cohorts, the coordinator switches to either PrA or PrC with these cohorts, depending on the tendency of the results of the deferred constraints with respect to validation. Thus, even when AP^3 switches to 2PC, it selects the cheapest 2PC variant.

3.1. Description of AP^3

A coordinator in AP^3 records information pertaining to the execution of a transaction in a protocol table which is kept in the coordinator's main memory. Specifically, a coordinator keeps for each transaction the identities of the cohorts and any pending request at a cohort. It also keeps track of the used protocol with each cohort (i.e., IYV, PrA or PrC). When a coordinator submits the first operation to be executed at a cohort for a particular transaction, the coordinator registers the cohort in its protocol table and marks

the cohort as 1PC. Thus, in AP^3, each transaction starts as 1PC at each cohort.

Following IYV, each cohort keeps a *recovery-coordinators' list* (RCL) that contains the identities of the coordinators that have active transactions at its site and must be contacted during the recovery of the cohort after a failure. In order to survive failures, an RCL is kept onto the stable log. When a cohort receives the first operation of a transaction and the identity of the coordinator of the transaction is not already in its RCL, the cohort 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 *all-active flag* (AAF) is used. A cohort sets AAF once it force writes an RCL that contains the identities of *all* the coordinators and does not consult its RCL as long as the AAF is set.

Once an operation is executed successfully, the cohort ACK the coordinator with a message that contains the results of the operation, as shown in Figure 1. On the other hand, if the operation fails, the cohort sends a negative acknowledgment (NACK) message. In either case, the cohort does not force its log onto stable storage prior to acknowledging an operation. For a successful update operation, as long as it does not cause deferred validation of consistency constraints, the cohort 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 cohort returns to an active state to execute the operation.

If an update operation causes deferred validation of consistency constraint(s), the cohort indicates this to the coordinator by requesting to switch to 2PC variant by sending an *unsolicited deferred consistency constraint* (UDCC) vote. UDCC is a flag that is set as part of the operation's ACK. The cohort also indicates, in the UDCC, the most appropriate 2PC variant depending on the tendency of the validation results of the consistency constraint(s). That is, if a deferred constraint tend to be violated by transactions and consequently aborting them at commit time, the cohort selects PrA. Otherwise, the cohort selects PrC. This is because 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 [7].

Once UDCC is set as part of an ACK, the cohort does not include any redo log records for the executed operation or any subsequent operation(s) pertaining to the same the transaction. Also, the cohort 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 2PC protocols. Furthermore, if the transaction is the last active transaction submitted by its coordinator to the cohort, the cohort 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.

When a coordinator receives an ACK with a UDCC flag set, it updates its protocol table to reflect the protocol switch for the cohort 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 cohort 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 cohort's identity. Hence, for 1PC cohorts, the coordinator's log contains a partial image of the redo part of each cohort's log which can be used to reconstruct the redo part of a cohort's log in case it is corrupted due to a system's failure.

3.1.1. Terminating a Transaction in AP^3

If the coordinator receives either an abort request from a transaction or a NACK from any cohort, 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 prepared-to-commit cohort (i.e., each cohort that has acknowledged the processing of all the transaction's operations successfully).

When the coordinator of a transaction receives a commit primitive from the transaction, it waits for the ACKs of the transaction's pending operations and then checks its protocol table to determine whether any cohort has switched to 2PC. If no cohort 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 cohorts, and then sends commit messages to the cohorts. Once the coordinator receives ACKs from all cohorts, it writes a non-forced end log record and forgets the transaction.

If any cohort has switched protocol, the coordinator decides which 2PC variant to use for the commitment of the transaction. The coordinator selects PrA if any of the 2PC cohorts has requested the use of PrA. Otherwise, the coordinator selects PrC. This is because it is more probable that the transaction will violate the deferred constraints at the cohorts that have requested to switch to PrA and, consequently, the transaction will end-up aborting. After deciding on the 2PC variant, the coordinator force-writes a *switch* log record only if the used protocol with the 2PC cohorts is PrC. The switch record includes the identities of all cohorts, indicating the cohorts that have switched protocol. Then, for 2PC cohorts, the coordinator sends prepare to commit messages, indicating the decided 2PC variant (i.e., PrA or PrC), as shown in Figure 1.

When a 2PC cohort receives a prepare to commit message, it validates the transaction. If the transaction can be committed, the cohort force writes a prepared log record

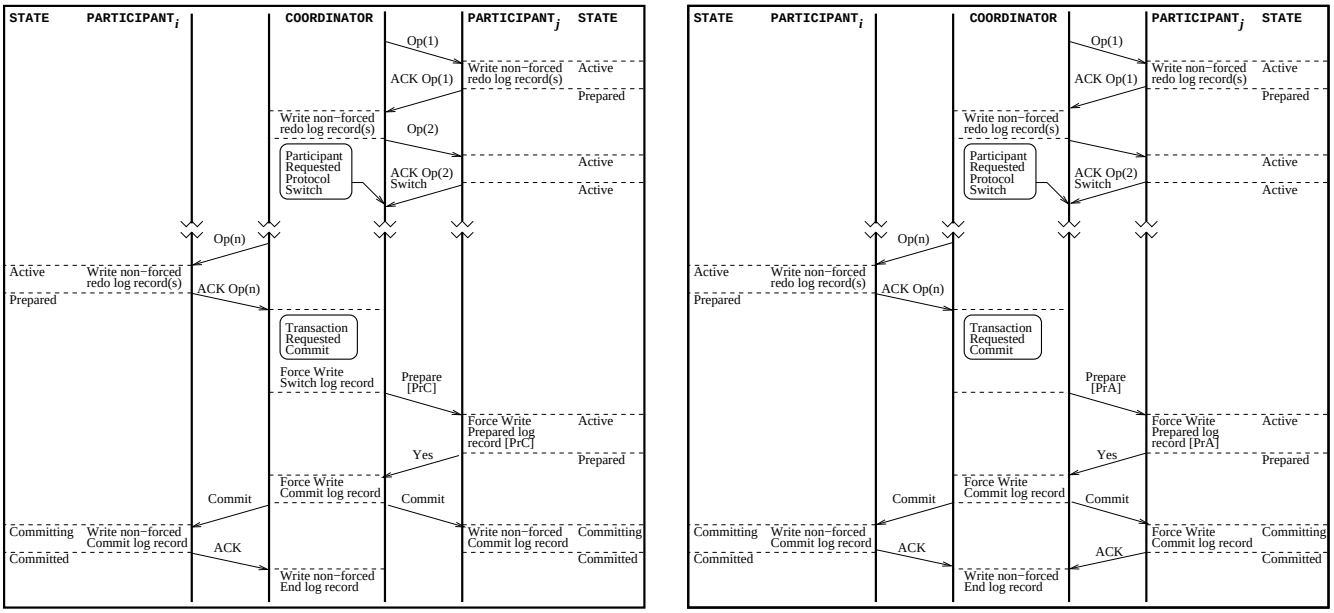

Figure 1. Committing a transaction in AP^3.

which includes the used 2PC variant. Then, the cohort sends back its vote to the coordinator. As in all 2PC variants, a 2PC cohort in AP^3 votes "yes" only if all consistency constraints are validated and the cohort can comply with a commit final decision. Otherwise, it aborts the transaction and sends back a "no" vote. When a cohort votes "yes", it enters an *explicit* prepared-to-commit state.

When the coordinator receives the votes of 2PC cohorts, the coordinator makes the final decision. The decision is commit if each 1PC cohort is in an implicit prepared-to-commit state and each 2PC cohort is in an explicit prepared-to-commit state. Otherwise, the decision is abort.

Committing a Transaction in Presence of 2PC Cohorts

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

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

When a 1PC cohort receives a commit decision regarding a transaction, it enforces the decision, writes a non-forced commit log record and releases all the transaction's resources. If the transaction was the last active transaction submitted by its coordinator, the cohort 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. Then, the cohort acknowledges the decision. Thus, a cohort in AP^3 follows IYV as long as it did not switch protocol and regardless of the used protocol with the other cohorts.

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

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

Aborting a Transaction in Presence of 2PC Cohorts

If the decision is abort and the used protocol with 2PC cohorts is PrC (Figure 2 (a)), the coordinator sends the decision to all cohorts, without writing a decision record, and

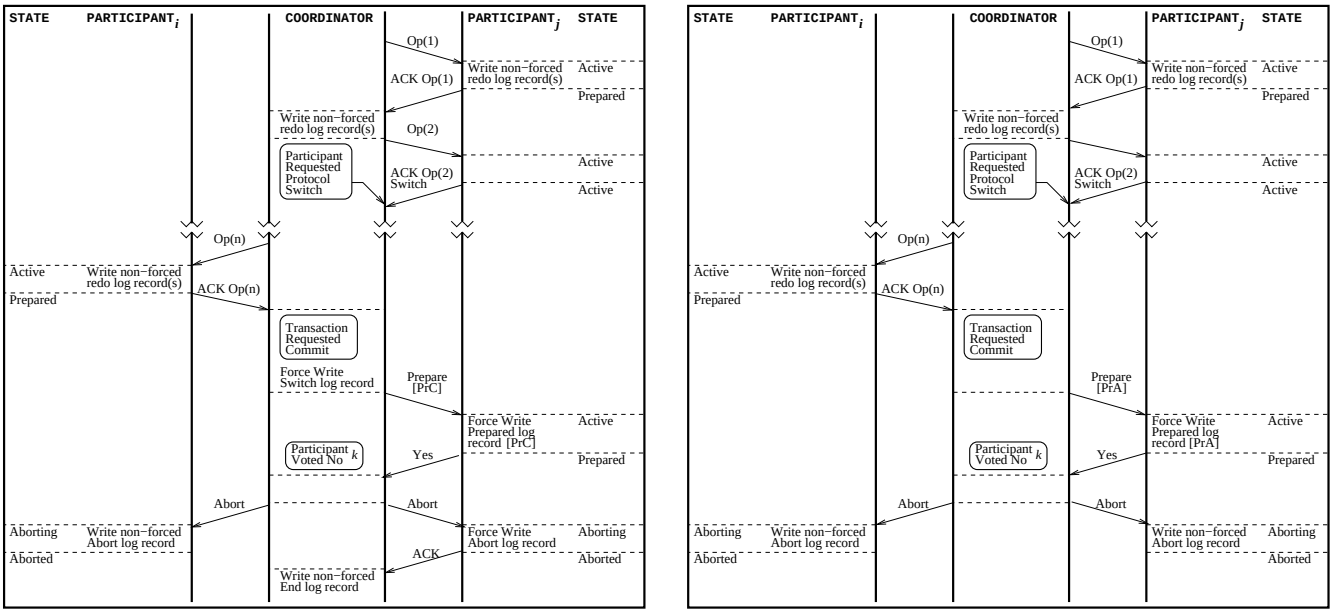

Figure 2. Aborting a transaction in AP^3.

waits for the ACKs of only 2PC cohorts. When a 2PC cohort 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 cohorts is PrA (Figure 2 (b)), the coordinator sends the decision to all cohorts 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 cohort receives the decision, it writes a non-forced abort log record and complies with the decision without sending an ACK back to the coordinator.

When a 1PC cohort receives an abort decision, it enforces the decision, writes a non-forced abort log record and releases all the transaction's resources. If the transaction was the last active transaction submitted by its coordinator, the cohort 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. Again, a cohort in AP^3 follows IYV as long as it did not switch protocol and regardless of the used protocol with the other cohorts.

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

Since for the abort case only an IYV or a PrA cohort might inquire about a transaction, the coordinator, not remembering the transaction, it will reply with an abort message using the abort presumption of IYV or PrA that was used by the cohort. Again, recall that each 2PC cohort records the used 2PC variant for a transaction in the prepared record. This information is used in the inquiry message of the cohort to guide the coordinator in its response in the absence of information about the transaction.

3.2. Recovery in AP^3

This section discusses the recovery aspects of AP^3. The protocol is resilient to both communication and site failures which are detected by timeouts, as in all ACPs.

3.2.1. Communication Failures

There are four points during the execution of AP^3 where a communication failure might occur while a site is waiting for a message. The first point is when a cohort has no pending operation ACKs. If the transaction is 1PC at a cohort, the cohort is blocked until communication is re-established with the coordinator. Then, the cohort 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 cohort enforces the final decision. Then, if the decision is commit, the cohort also acknowledges it. In the latter case, the cohort waits for further operations. On the other hand, if the cohort is 2PC, it aborts the transaction.

The second point is when the coordinator of a transaction is waiting for an operation ACK from a cohort. In this case, the coordinator aborts the transaction and submits a final abort decision to the rest of the cohorts. Similarly, a cohort aborts a transaction when a communication failure occurs and the cohort has a pending operation ACK. Notice that the coordinator of a transaction may commit the transaction in spite of communication failures with some cohorts as long as these cohorts are 1PC cohorts and have no pending operations' ACKs.

The third point is when the coordinator is waiting for the votes of 2PC cohorts. 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 cohorts and waits for the required ACKs (when PrC is used with 2PC cohorts). For an inaccessible cohort, the cohort 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.

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 cohorts once communication failures are fixed. If the decision is commit, the coordinator re-submits a commit message to each inaccessible 1PC cohort. It also re-submits the commit decision to each inaccessible 2PC cohort when PrA is used. On the other hand, if the decision is an abort and PrC is used with 2PC cohorts, the coordinator re-submits an abort message to each inaccessible 2PC cohort. When a 1PC cohort 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 cohort 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.

3.2.2. Site Failures

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

Coordinator's Failure

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

1. Each transaction with only a switch record: the coordinator knows that PrC was used with 2PC cohorts. Based on that, it aborts the transaction and sends an abort message to each 2PC cohort recorded in the switch record and waits for the ACKs.

2. Each transaction with a switch record and a commit record but without an end record: the coordinator knows that PrC was used with 2PC cohorts but not all the required ACKs from 1PC cohorts were received before the failure. Based on that, it sends commit messages to all 1PC cohorts and waits for their ACKs. On the other hand, if all cohorts are 2PC, the coordinator considers the transaction complete and does not include it in its protocol table.

3. Each transaction with a commit record but without a switch and an end record: the coordinator knows that either IYV was used and no cohort has switched protocol, IYV in conjunction with PrA were used, or only PrA was used because all cohorts have switched protocol. In either case, the coordinator sends a commit message to each cohort recorded in the commit decision record (i.e. all cohorts) and waits for ACKs.

In all the three cases above, when a cohort receives a decision message, it either ACK the message if it has already received and enforced the decision prior to the failure, or enforces the decision and then sends back an ACK.

For all other transactions, the coordinator can safely ignore them during its recovery procedure and considers them as completed transactions. If a cohort inquires about a transaction that the coordinator has forgotten, the coordinator responds with a decision that matches the presumption of the protocol indicated in the inquiry message of the cohort (i.e., abort for IYV and PrA, commit for PrC).

Cohort's Failure

Since the entire log might not be written onto a stable storage until after the log buffer overflows, the log may not contain all the redo records of the transactions committed by their perspective coordinators after a failure of a cohort. Thus, at the beginning of the *analysis phase* of the restart procedure, the cohort determines the largest LSN that is associated with the last record written onto its log that survived the failure and sends a *recovering* message that contains the largest LSN to all coordinators in its RCL. This LSN is used by the coordinators to determine missing redo log records at the cohort which are replicated in their logs and are needed by the cohort to fully recover. If the RCL is empty, the cohort recovers the state of its database locally using its own log without sending a recovering message to any coordinator, and then resumes normal processing. Otherwise, the cohort waits for the reply messages to arrive from the coordinators included in the RCL.

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 cohort recovers those aborted and committed transactions that have decision records pertaining to them 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,

Table 1. The costs of the different protocols to "commit" a transaction.

	2PC	PrC	PrA	IYV	AP^3 (1PC)	AP^3 (PrC)	AP^3 (PrA)	AP^3-MIX (PrC)	AP^3-MIX (PrA)
Log force delays	2	3	2	1	1	3	2	3	2
Total forced log writes	2n+1	n+2	2n+1	1	1	n+2	2n+1	(n-p)+2	2(n-p)+1
Message delays (Commit)	2	2	2	0	0	2	2	2	2
Message delays (Locks)	3	3	3	1	1	3	3	3	3
Total messages	4n	3n	4n	2n	2n	3n	4n	3(n-p)+2p	4(n-p)+2p

Table 2. The costs of the different protocols to "abort" a transaction.

	2PC	PrC	PrA	IYV	AP^3 (1PC)	AP^3 (PrC)	AP^3 (PrA)	AP^3-MIX (PrC)	AP^3-MIX (PrA)
Log force delays	2	2	1	0	0	2	1	2	1
Total forced log writes	2n+1	2n+1	n	0	0	2n+1	n	2(n-p)+1	n-p
Message delays (Abort)	2	2	2	0	0	2	2	2	2
Message delays (Locks)	3	3	3	1	1	3	3	3	3
Total messages	4n	4n	3n	n	n	4n	3n	4(n-p)+p	3(n-p)+p

partially masks the effects of dual logging and communication delays. Note that because of the use of write-ahead logging (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 cohort's stable log and never replicated at the coordinators' sites.

When a coordinator receives a recovering message from a cohort, it will know that the cohort has failed and is recovering from a failure. Based on this knowledge, the coordinator checks its protocol table to determine each transaction that the cohort 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 cohort was not aware of the decision prior to its failure). 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 cohort. For all other transactions, the coordinator responds with abort decisions.

All these responses and redo log records are packaged in a single *repair* message and sent back to the cohort. If a coordinator has no active transactions and all terminated transactions have been acknowledged (according to AP^3 protocol) as far as the failed cohort is concerned, the coordinator sends an ACK repair message, indicating to the cohort that there are no transactions to be recovered.

When the cohort has received reply messages from all the coordinators in its RCL, the cohort repairs its log and completes the redo phase. Once the redo phase is completed, the cohort sends back the required decision ACKs as during normal processing and then, resumes normal processing.

The case of an overlapped coordinator and cohort failure is handled using the same procedures discussed above. However, if the failed coordinator is in the RCL of a recovering cohort, the coordinator needs to recover first before responding to the cohort's pending repair message.

4. Analytical Evaluation

Tables 1 and 2 compare the costs of the different protocols for the commit case and the abort case on a per transaction basis, respectively. The column titled "AP^3(1PC)" denotes AP^3 when *all* cohorts are 1PC. The column titled "AP^3(PrC)" denotes AP^3 when *all* cohorts are 2PC and PrC is used. The column titled "AP^3(PrA)" denotes AP^3 when *all* cohorts are 2PC and PrA is used. The column titled "AP^3-MIX(PrC)" denotes AP^3 in the presence of both 1PC and 2PC cohorts when PrC is used with 2PC cohorts. The column titled "AP^3-MIX(PrA)" denotes AP^3 in the presence of both 1PC when 2PC cohorts and PrA is used with 2PC cohorts. In the tables, *n* denotes the total number of cohorts in a transaction's execution (excluding the coordinator), whereas *p* denotes the number of 1PC cohorts.

The row labeled "Log force delays" contains the sequence of forced log writes that are required by the different protocols up to the point that the commit/abort decision is made. The row labeled "Message delays (Decision)" contains the number of sequential messages up to the commit/abort point, and the row labeled "Message delays (Locks)" contains the number of sequential messages that are involved in order to release all the locks held by a committing/aborting transaction at the cohorts. For example, to commit a transaction (Table 1), the "Log force delays" for 2PC is "2" because there are two sequential forced log writes between the beginning of the protocol and the time a commit decision is made by the transaction's coordinator. Also, "Message delays (Decision)" and "Message delays (Locks)" are "2" and "3", respectively, because 2PC involves two sequential messages in order for a coordinator to make the final commit decision (i.e., the first phase), and three sequential messages to release all the resources (e.g., locks) held by the transaction at the cohorts.

It is clear from Tables 1 and 2 that AP^3 performs as IYV

when all cohorts are 1PC, outperforming 2PC and its variants in all performance measures including the number of "Log force delays" to reach a decision as well as the "Total forced log writes". For the commit case, the two protocols require only one forced log write whereas, for the abort case, neither AP^3 nor IYV force write any log records. When all cohorts are 2PC and the used protocol variant is PrC, AP^3 performs as PrC in all performance measures for the commit case as well as the abort case. Similarly, when all cohorts are 2PC and the used protocol variant is PrA, AP^3 performs as PrA in all performance measures for the commit case as well as the abort case. When there is cohorts' mix, which is the general case, the performance of AP^3 exhibits the behavior of either PrC or PrA, depending on the protocol variant used, with respect to the sequential performance metrics. That is, AP^3 has the same number of "Log force delays", "Message delays (Decision)" and "Message delays (Locks)" as either PrC or PrA. The performance of AP^3 with respect to the total number of messages and forced log writes depends on the cohorts' mix, which is, in general, less than that of PrC and PrA.

5. Conclusions

Modern Internet database applications such as those commonly found in electronic commerce and electronic services require coordination protocols with reduced overhead. This is in order to increase customer satisfaction through enhanced system's throughput. To this end, this paper proposed a new atomic commit protocol called *adaptive participant's presumption protocol* (AP^3) that interoperates one-phase commit (1PC) and two-phase commit (2PC) protocols. AP^3 starts as 1PC and switches to the most appropriate 2PC variant, on a participant's basis, only when necessary. This is achieved in spite of the incompatibilities between the combined protocols. Thus, AP^3 alleviates the applicability shortcomings of 1PC and at the same time, it keeps the overall protocol overhead below that of 2PC and its well known variants. This was highlighted by comparing the performance of the different protocols analytically with respect to log, message and time complexities. The performance and applicability of AP^3 make it a specially important protocol in the context of environments that are characterized by high volume of short transactions such as Internet database applications.

AP^3 needs further enhancements, such as reducing the cost of commit processing for read-only transactions, and extensions, such as extending it to the multi-level transaction execution model. These enhancements and extensions are left as part of our future work in this direction. Furthermore, there is a need to evaluate the performance of AP^3 and the other protocols empirically in order to reveal any hidden costs that cannot be captured analytically.

References

[1] M. Abdallah, R. Guerraoui and P. Pucheral. One-Phase Commit: Does it make sense? *Proc. of the Int'l Conf. on Parallel and Distributed Systems*, 1998.

[2] M. Abdallah, R. Guerraoui and P. Pucheral. Dictatorial Transaction Processing: Atomic Commitment without Veto Right. *Distributed and Parallel Databases*, 11(3):239-268, 2002.

[3] Y. J. Al-Houmaily and P. K. Chrysanthis. Atomicity with Incompatible Presumptions. *Proc. of the 18^{th} ACM PODS*, pp. 306–315, 1999.

[4] Y. J. Al-Houmaily and P. K. Chrysanthis. An Atomic Commit Protocol for Gigabit-Networked Distributed Database Systems. *Journal of Systems Architecture – The EUROMICRO Journal*, 46(9):809–833, 2000.

[5] Y. J. Al-Houmaily and P. K. Chrysanthis. 1-2PC: The One-Two Phase Atomic Commit Protocol. *Proc. of the ACM SAC*, 2004.

[6] Y. J. Al-Houmaily and P. K. Chrysanthis. ML-1-2PC: An Adaptive Multi-Level Atomic Commit Protocol. *ADBIS*, LNCS 3255, 2004.

[7] Y. Al-Houmaily, P. Chrysanthis and S. Levitan. An Argument in Favor of the Presumed Commit Protocol. *Proc. of the 13^{th} ICDE*, pp. 255–265, 1997.

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

[9] J. Gray. Notes on Data Base Operating Systems. In *Operating Systems: An Advanced Course*, LNCS Vol. 60, pp. 393–481, 1978.

[10] R. Gupta, J. Haritsa and K. Ramamritham. Revisiting Commit Processing in Distributed Database Systems. *Proc. of the ACM SIGMOD*, 1997.

[11] B. Lampson. Atomic Transactions. In *Distributed Systems: Architecture and Implementation - An Advanced Course*, LNCS Vol. 105, pp. 246-265, 1981.

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

[13] G. Samaras, G. Kyrou and P. K. Chrysanthis. Two-Phase Commit Processing with Restructured Commit Tree. *Proc. of the Panhellenic Conference on Informatics*, LNCS Vol. 2563, pp. 82-99, 2003.

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