1-2PC: The One-Two Phase Atomic Commit Protocol

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

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

ABSTRACT

This paper proposes a one-phase, two-phase commit (1-2PC) protocol that can be used to atomically commit Internet transactions distributed across sites in a wide area network. The 1-2PC protocol is characterized by its ability to dynamically select between one-phase and two-phase atomic commit protocols depending on the behavior of transactions and the system requirements. Thus, it offers the performance advantages of the one-phase atomic commit protocol whenever possible, while still providing the wide applicability of the two-phase commit protocol. This is achieved in spite of the incompatibilities between one-phase and two-phase commit protocols that lead to the general practice of having to adopt a single atomic commit protocol in any distributed database system.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Distributed databases, Transaction processing*

General Terms

Algorithms, Design, Experimentation, Performance

Keywords

Internet Transactions, Atomic Commit Protocol

1. INTRODUCTION

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

The *two-phase commit* (2PC) protocol [6, 8] is one of the most widely used and optimized ACP. It ensures atomicity and independent recovery but at a substantial cost during normal transaction execution which adversely affects the performance of the system. This is due to the cost 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 amount of information that needs to be stored in the stable logs of the participating sites). For this reason, there has been a re-newed interest in developing more efficient ACPs and optimizations [7, 3, 5, 2]. This is especially important in modern electronic services and electronic commerce environments that are characterized by high volume of transactions. Most notable results are *one-phase commit* (1PC) protocols [12, 5, 2, 11].

1PC protocols reduce both message and log complexities at the expense of placing assumptions on transactions or the database management systems. Whereas some of these assumptions are realistic (i.e., reflect how the database management systems are usually implemented), other assumptions can be considered very restrictive in some applications [1, 5]. 1PC protocols, including *implicit yes-vote* (IYV) [5] and *coordinator log* (CL) [12] assume that each transaction's operation is acknowledged after its execution at the participant's 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 the evaluation of integrity constraints during commit time at the end of the execution of a transaction rather than at the end of each operation. Thus, the evaluation of deferred constraints needs to be synchronized across all participating database sites.

For the above reason, we propose a new ACP called *one-phase, two-phase commit* (1-2PC) protocol which is essentially a combination of 1PC and 2PC and that starts as 1PC and switches to 2PC only when necessary. Thus, 1-2PC achieves the performance advantages of 1PC protocols and the wide applicability of 2PC protocols. In other words, 1-2PC supports deferred constraints without penalizing those transactions that do not require them. Furthermore, 1-2PC achieves this advantage on a participant basis within the same transaction in spite of the incompatibilities between the 1PC and 2PC protocols.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SAC'04, March 14-17, 2004, Nicosia, Cyprus.
Copyright 2004 ACM 1-58113-812-1/03/04 ...$5.00.

In the next section, 1PC and 2PC protocols are briefly reviewed. The discussion is limited to protocols that support atomicity and does not include any protocol such as Optimistic 2PC [9] which ensures weaker notions of atomicity, e.g., semantic atomicity. Section 3 first introduces the basic 1-2PC and then describes an advanced 1-2PC variant that (1) exploits read-only transactions to enhance its performance during normal processing and (2) supports forward recovery to enhance its performance after a site failure. It also provides an informal proof of correctness. Section 4 compares 1-2PC to the best known APCs with respect to message, log and time complexities and shows its performance advantages. Section 5 concludes the paper.

2. BACKGROUND

The 2PC protocol consists of two phases, namely a *voting phase* and a *decision phase*. During the voting phase, the coordinator requests all participating sites to *prepare to commit* whereas, during the decision phase, the coordinator either commits the transaction if *all* participants are prepared-to-commit (voted "yes"), or aborts the transaction if any participant has decided to abort (voted "no"). While in a prepared-to-commit state, a participant can neither commit nor abort the transaction until it receives the coordinator's final decision. When a participant receives the final decision, it complies with the decision and then sends back an acknowledgment (ACK). Once the coordinator receives ACKs from all the participants, it knows that all participants have received the decision and none of them will inquire about the status of the transaction in the future. Therefore, the coordinator discards all information pertaining to the transaction from its *protocol table* that is kept in main memory, and forgets the transaction.

Since the objective of 2PC is to achieve atomicity of transactions in the presence of possible system and communication failures, the protocol requires that a coordinator and each participant to record sufficient information about the progress of the protocol in their logs. Specifically, the coordinator is required to force write a decision record prior to sending out the final decision. Similarly, each participant is required to force write a prepared record before sending its "yes" vote and a decision record before sending an ACK. Since a forced write of a log record ensures that the record is written into a stable storage that survives system failures, the coordinator and the participants can recover a transaction to a consistent termination state in the case of a failure. When the coordinator completes the protocol, it writes a non-forced end log record in its log buffer.

The *presumed abort* protocol (PrA) is designed to reduce the cost associated with aborting transactions [10]. Specifically, in PrA, when a coordinator decides to abort a transaction, it does not force write the abort decision in its log as in 2PC. It just sends abort messages to all participants that have voted "yes" and discards all information about the transaction from its protocol table. That is, the coordinator of an aborted transaction does not write any log records or wait for ACKs. Since the participants do not acknowledge 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 or a participant failure, if the participant inquires about a transaction that has been aborted, the coordinator, not remembering the transaction, will direct the participant to abort it (by presumption).

As opposed to PrA, the *presumed commit* protocol (PrC) is designed to reduce the cost associated with committing transactions [10]. This is achieved by interpreting missing information about transactions as commit decisions. However, in PrC, a coordinator force writes an *initiation* record for each transaction before sending prepare to commit messages to the participants. This record ensures that missing information about a transaction will not be misinterpreted as a commit after a coordinator's failure. 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 the messages (i.e., their absence vs. their presence), but also because of their contradicting presumptions about the outcome of terminated transactions in the absence of information about them [4].

The 2PC variants are usually criticized on the grounds of their performance drawback especially for short living transactions (i.e., transactions that access few data objects at each participating database site), 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) protocols that consist of only a single phase which is the decision phase of 2PC. The (explicit) voting phase is eliminated by overlapping it with the acknowledgments of the database operations. This principle is used in both *coordinator log* (CL) [12] and *implicit yes-vote* (IYV) [5] protocols, but the mechanisms that the two protocols use for recovery are different. These differences are due to the assumptions made in the two protocols about the database sites. IYV, on which 1-2PC is based, 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.

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 perspective 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, in IYV, the explicit voting phase of 2PC is eliminated by overlapping it with the execution of operations 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* [5]. After a system 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 resume their execution without having to abort them after a failure. 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.

3. THE 1-2PC PROTOCOL

1PC protocols are more efficient than 2PC and its variants, but some of 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. As opposed to immediate constraints that are evaluated at the end of each operation, deferred constraints are evaluated during commit time when a transaction finishes its execution and are triggered by the prepare message of the voting phase. In order to support deferred constraints without penalizing those transactions that they do not require them, we developed 1-2PC that operates as a 1PC protocol as long as the voting phase of 2PC is not necessary.

3.1 Description of the Basic 1-2PC Protocol

As in all the other commit protocols, a coordinator in 1-2PC records information pertaining to the execution of a transaction in a protocol table which is kept the coordinator's main memory. Specifically, a coordinator keeps for each transaction the identities of the participants and any pending request at a participant. It also keeps track of the used protocol with each participant (i.e., whether the protocol is one-phase or two-phase).

When a coordinator submits the first operation to be executed at a participant's site for a particular transaction, the coordinator registers the participant in its protocol table and marks the participant as 1PC participant for the transaction. Thus, in 1-2PC, each transaction starts as a 1PC transaction at each participating site.

As in IYV, each participant 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 participant after a failure. In order to survive failures, an RCL is kept in the stable log. When a participant receives the first operation of a transaction, if 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 in its 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 AAF once it force writes an RCL containing the identities of *all* the coordinators and does not consider the RCL as long as the AAF is set.

Once an operation is executed successfully, the participant 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 participant sends a NACK message. In either case, the participant does not force its log into 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 participant follows the IYV protocol. 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 either for 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 the update operation causes deferred validation of consistency constraint(s), the participant indicates this to the coordinator and switches to 2PC by sending an *unsolicited deferred consistency constraint* (UDCC) vote. UDCC is a flag that is set as part of the operation's ACK. Once a UDCC is set, no redo records are included in the ACK for this or any subsequent operation(s) for the 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 2PC protocols. Further, if the transaction is the last active transaction submitted by its coordinator at the site, the participant resets its AAF if it is set, deletes the transaction's coordinator from the RCL, and force writes the updated list in its log.

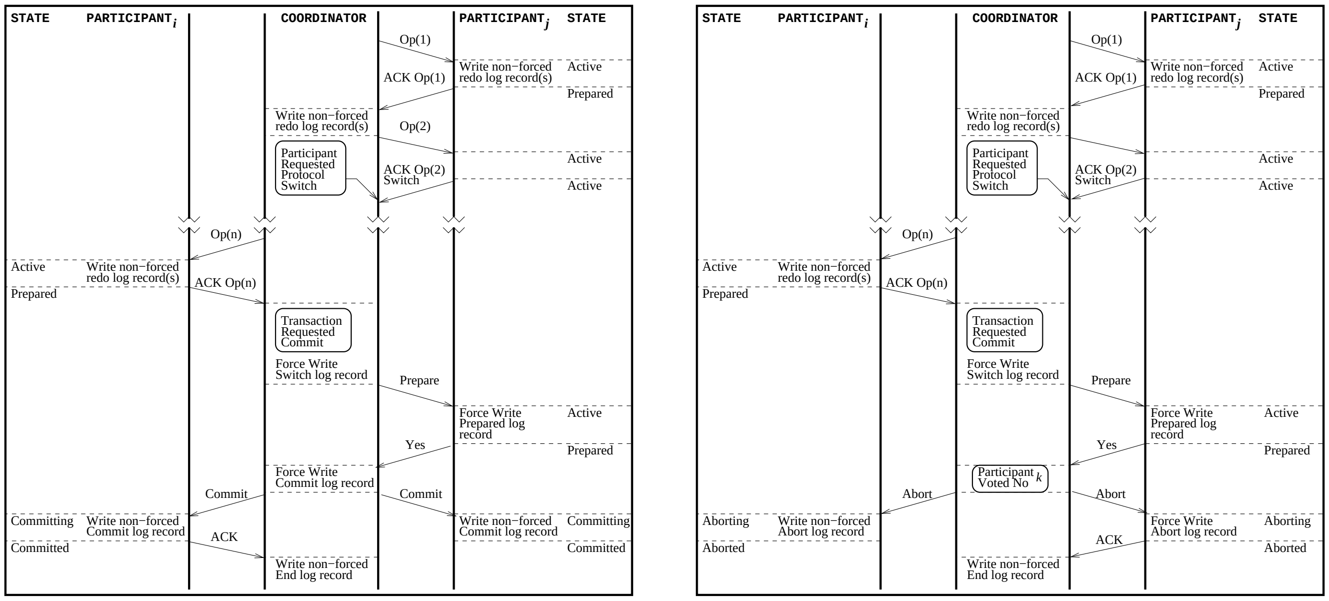
When a coordinator receives an ACK from a participant with the UDCC flag set, it updates its protocol table to reflect the protocol switch for the participant. On the other hand, if the flag is not set, the coordinator checks its protocol table to determine if the ACK is from a 1PC participant and extracts any redo log records from 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 a participant's log in case it is corrupted due to a system's failure.

3.1.1 Terminating a Transaction in 1-2PC

If the coordinator receives either an abort request from the transaction or a NACK regarding the transaction from a participant, it aborts the transaction. On an abort decision, 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 participant (i.e., each participant 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 participant has switched to 2PC. If no participant has switched protocol, the coordinator commits the transaction as in 1PC protocols. It force-writes a commit log record which includes the identities of all the participants, and then sends commit messages to the participants.

If any participant has switched protocol, the coordinator force-writes a *switch* log record which includes the identities of all the participants, indicating the participant(s) that


STATE
PARTICIPANT$_i$
COORDINATOR
PARTICIPANT$_j$
STATE
Op(1)
ACK Op(1)
Write non-forced redo log record(s)
Active
Prepared
Write non-forced redo log record(s)
Op(2)
Participant Requested Protocol Switch
ACK Op(2) Switch
Active
Active
Op(n)
Active
Write non-forced redo log record(s)
ACK Op(n)
Prepared
Transaction Requested Commit
Force Write Switch log record
Prepare
Force Write Prepared log record
Active
Yes
Prepared
Force Write Commit log record
Commit
Commit
Committing
Write non-forced Commit log record
ACK
Write non-forced Commit log record
Committing
Committed
Committed
Write non-forced End log record
STATE
PARTICIPANT$_i$
COORDINATOR
PARTICIPANT$_j$
STATE
Op(1)
ACK Op(1)
Write non-forced redo log record(s)
Active
Prepared
Write non-forced redo log record(s)
Participant Requested Protocol Switch
ACK Op(2) Switch
Active
Active
Op(n)
Active
Write non-forced redo log record(s)
ACK Op(n)
Prepared
Transaction Requested Commit
Force Write Switch log record
Prepare
Force Write Prepared log record
Active
Yes
Prepared
Participant$_k$ Voted No
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) Commit Case **(b) Abort Case**

Figure 1: The 1-2PC coordination messages and log writes.

caused protocol switch (Figure 1). Then, for 2PC participants, the coordinator sends prepare to commit messages.

When a 2PC participant receives a prepare to commit message, it validates the transaction and then sends back its vote. As in all 2PC variants, a participant in 1-2PC votes "yes" only if all consistency constraints are validated and the participant can comply with a commit final 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 (if any), the coordinator makes its final decision. The decision is commit if all 1PC participants are in an implicit prepared-to-commit state and all 2PC participants are in an explicit prepared-to-commit state. Otherwise, the decision is abort. On a commit decision (Figure 1 (a)), the coordinator force writes a commit log record, sends its decision to all participants, and waits for the ACKs of 1PC participants. On an abort decision (Figure 1 (b)), on the other hand, the coordinator sends the decision to the participants that are in their prepared-to-commit states (whether explicit or implicit) and waits for the acknowledgments of the prepared-to-commit 2PC participants, without writing an abort decision record.

When a 1PC participant receives a commit (abort) decision regarding a transaction, it enforces the decision, writes a non-forced commit (abort) log record and releases all the transaction's resources. A participant acknowledges a commit decision, but not an abort decision, only after the corresponding commit log record is placed into stable storage as a result of a subsequent force-write or flush of the log onto stable storage. 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, force writes the updated list in its log and then, acknowledges the decision if it is a commit decision. Thus, a participant in 1-2PC behaves as an IYV participant as long as it did not switch protocol and regardless of whether the other participants in the transaction's execution have switched protocol or not.

When a 2PC participant receives a commit decision, it writes a non-forced commit log record and complies with the decision, without sending an acknowledgment message back to the coordinator. If the decision is an abort, the participant force writes an abort log record, complies with the decision and then, acknowledges the decision. Thus, in 1-2PC, a participant behaves exactly as if it is using PrC from the moment it has switched protocol.

Finally, when the coordinator receives ACKs for a commit decision from all 1PC participants, it writes a non-forced end log record and discards all information pertaining to the transaction from its protocol table, knowing that no 1PC participant will inquire about the transaction's status in the future. Since only a 2PC participant (if any) might inquire about the outcome of a committed transaction, the coordinator, not remembering the transaction, it will reply with a commit message using the presumption of PrC protocol used by the participant. Similarly, when the coordinator receives ACKs for an abort decision from all 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 2PC participant will inquire about the transaction's outcome in the future. Since only a 1PC participant (if any) might inquire about the outcome of an aborted transaction, the coordinator, not remembering the transaction, it will reply with an abort message using the presumption of IYV protocol used by the participant.

3.2 The Advanced 1-2PC Protocol

The advanced 1-2PC protocol exploits read-only transactions to enhance its performance during normal processing, and the option of forward recovery [5] to enhance the performance of 1-2PC protocol after a site or communication failure.

3.2.1 1-2PC and Read-Only Transactions

In the traditional read-only optimization, during the voting phase, a participant votes *read-only* if it has executed only read operations [10]. Using this optimization, a participant can release all the locks held by the transaction once it votes. Furthermore, a read-only participant does not participate in the second phase of these protocols and hence it does not need to know the final outcome of the transaction. Since a read-only transaction does not update any data item, there is no logging associated with such transactions.

The cost associated with read-only participants can be reduced further if the coordinator of a transaction knows, before the initiation of the commit protocol, which participants are read-only in the execution of the transaction. In this way, if all participants are read-only, the coordinator can avoid writing any log records (in the case of PrC) and the read-only votes of the participants can be eliminated. This is the essence of the *unsolicited update-vote* (UUV) optimization [3], the principle of which is used in 1-2PC.

In 1-2PC, each transaction starts as a read-only transaction. The coordinator of a transaction marks in its protocol table a participant as an update one when it receives from the participant an ACK with redo log records or with the UDCC flag set. (Only update operations generate log records and cause consistency constraint validation.)

To commit a completely read-only transaction, the coordinator sends a *read-only* message to each participant without writing any log records for the transaction, and forgets the transaction. If the transaction is partially read-only, the coordinator sends a read-only message to each read-only participant and removes the participant from its protocol table, without waiting for the final decision to be made. This is especially important in the presence of 2PC (update) participants because it allows for an earlier release of resources at read-only participants. In any case, when a read-only participant receives the message, it release the resources held by the transaction without writing any log records or acknowledging the message. For update participants, the coordinator follows the basic 1-2PC discussed above.

3.2.2 Forward Recovery in 1-2PC

We define forward recovery as 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 has been fixed. In 1-2PC, forward recovery is an option that is not necessary for the correctness of the protocol and it is applicable to transactions that are context-free at the participants.

In 1-2PC with forward recovery, a transaction indicates to its coordinator, when it is initiated, whether or not it wishes to use the option of forward recovery. For a *forward recoverable* transaction, the coordinator notifies each participating site of this option by setting a *forward recovery flag* (FRF) as part of the first operation submitted to a participant.

For a forward recoverable transaction, a 1PC participant includes in an operation's acknowledgment message, in addition to the redo log records (if any), all the read locks that have been acquired during the execution of the operation. Unlike the basic 1-2PC, each 2PC participant of a forward recoverable transaction must also include both the redo log records and read locks in the ACK of each operation that it executes. When the coordinator receives an ACK, 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.

In this way, 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 a participant's log in case it is corrupted due to a system's failure. At the same time, the coordinator's PLT contains a partial image of each 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's failure. As a result, after a participant's site failure, the participant can recover its state *exactly* as it was prior to a failure with the help of the coordinators, reacquiring both read and write locks, thereby allowing partially executed forward recoverable transactions that are still active in the system to forward recover and resume their execution after the participant has recovered.

3.3 Recovery in 1-2PC Protocol

In this section, we discuss the recovery aspects of the advanced 1-2PC. 1-2PC is resilient to both communication and site failures. As in all ACPs, failures are detected by timeouts.

3.3.1 Communication Failures

There are four points during the execution of 1-2PC where a communication failure might occur while a site is waiting for a message. The first point is when a participant has no pending acknowledgments. If the transaction is 1PC or a forward recoverable 2PC transaction, the participant is blocked until the communication with the coordinator is re-established. Then, the participant inquires the coordinator about the transaction's status. The coordinator replies with either a final decision or a *still active* message if the transaction is 1PC at the participant's site. In the former case, the participant enforces the final decision and then acknowledges it, only if the decision is commit, while in the latter case, the participant waits for further operations. Similarly, if the transaction is a forward recoverable 2PC transaction, the participant is blocked until the communication with the coordinator is re-established. Then, the participant inquires the coordinator about the transaction's status, indicating that it is a 2PC participant which is *implicitly* in a prepared-to-commit state. The coordinator replies with either an abort final decision if does not remember the transaction, or a *still active* message if the transaction is still active in the system, or a prepare to commit message if the transaction is in the voting phase of 2PC and no decision has been made yet. In the first case, the participant enforces the abort decision without acknowledging it, while in the second case, the participant waits for further operations. In the third case, the participant validates the transaction and sends back its vote, as it is the case in 1-2PC during normal processing.

The second point is when the coordinator of a transaction is waiting for an operation acknowledgment 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 if it is 1PC or is 2PC but not forward recoverable, and the communication failure has occurred while the participant has a pending acknowledgment. Notice that the coordinator of a transaction may commit the transaction despite communication failures with some participants as long as these participants are 1PC participants and have no pending acknowledgments.

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 acknowledgments.

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, it re-submits the decision to the appropriate participants once these communication failures are fixed. If the decision is a commit, the coordinator re-submits a commit message to each inaccessible 1PC participant. If the decision is an abort, the coordinator re-submits an abort message to each inaccessible 2PC participant. When a 1PC 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 participant receives an abort decision, it either acknowledges the decision if it has already enforced the decision prior to the failure, or enforces the decision and then acknowledges it.

3.3.2 Site Failures

As mentioned above, we assume that each site employs physical logging and uses a Undo/Redo crash recovery protocol in which the undo phase *precedes* the redo phase. It should be pointed out that 1-2PC can also be combined with *logical* or *physiological* write-ahead logging schemes.

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 consider only those transactions that have switch or decision records without corresponding end records. For each of these transactions, the coordinator creates an entry in its protocol table that includes the identities of the participants as recorded in the transaction's switch or decision record. When the coordinator finds a transaction with a switch record but without a corresponding commit record, the coordinator considers the transaction as an aborted transaction. The fate of the other transactions depend on the decisions recorded in their corresponding log records. Once the coordinator has identified incomplete transactions (with respect to the commit protocol), it restarts the decision phase for each of these transactions by re-submitting its decision to all the participants recorded in the switch or decision record and resumes normal protocol operation.

As in the case of a communication failure, if a participant has already received and enforced a final decision prior to the failure, the participant simply responds with an ACK as required by 1-2PC. If the participant has not received the decision, it must have been waiting for the decision and once it receives the decision, it writes the required decision record and then sends back an ACK (according to the protocol specification) when the decision record is in the stable log.

For those transactions without final decision records (i.e., those transactions that were active prior to the failure or their non-forced abort records did not make it to the stable log before the failure), the coordinator can safely forget them and consider them as aborted transactions. If a participant in the execution of one of these transactions has a pending acknowledgment, when it times out due to the coordinator's site failure, it will abort the transaction, as in the case of a communication failure that we discussed above. On the other hand, if the participant is left blocked (i.e., the participant has acknowledged all a transaction's operations and is in the implicit prepared-to-commit state), when the coordinator recovers, the participant will inquire about the status of the transaction. If the participant is a 1PC participant, the coordinator, not remembering the transaction after its recovery, will respond with an abort message (using the implicit presumption of IYV protocol employed by the participant). If the transaction is forward recoverable 2PC at the participant, the coordinator will respond with an abort message, utilizing the implicit prepared-to-commit state information included in the participant's inquiry message instead of using the presumption of 2PC employed by the participant. For those transactions that are associated with decision and end records, the coordinator can safely discard all information about these transactions, knowing that all required participants have received their decisions and will not inquire about their outcome in the future.

Participant's Failure

Since the entire log might not be written into 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 participant. 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 in 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 participant which are replicated in their logs and are needed by the participant to fully recover. If the RCL is empty, the participant recovers the state of its database locally using its own log without sending a recovering message to any coordinator, and then resumes normal processing. On the the other hand, if the RCL is not empty, the participant waits for reply messages to arrive from the coordinators.

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 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, 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 participant's stable log and never replicated at the coordinators' sites.

When a coordinator receives a recovering message from a participant, it will know that the participant has failed

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

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

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

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

and is recovering from the failure. Based on this knowledge, the coordinator checks its protocol table to determine each transaction that the 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 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.

For each active 1PC transaction and forward recoverable 2PC transaction that is still in progress in other sites, the coordinator has the option to either abort or forward recover the transaction. If the coordinator decides to abort the transaction, it sends abort messages to all participants to rollback the transaction. If the coordinator decides to forward recover the transaction, it responds with a *still-active* status containing, as in the case of a committed transaction, a list of the redo records associated with LSNs greater than the one included in the recovering message of the participant. The message also contains all the read locks that were held by the transaction at the participant's site prior to its failure.

All these responses and redo log records are packaged with the read locks acquired by active transactions 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 1-2PC 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 as far as this coordinator is concerned.

Once the participant has received reply messages from all the coordinators in its RCL, the participant repairs its log and completes the redo phase. The participant also re-builds its lock table by re-acquiring the update locks during the redo phase in conjunction with the read locks received from the coordinators. Once the redo phase is completed, the participant sends back the required decision ACKs as in the case of normal processing. Then the participant resumes its normal processing. Thus, in 1-2PC, a long-executing transaction is not necessarily aborted as a result of a participant failure as would be the case in all other ACPs.

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

4. ANALYTICAL EVALUATION

Tables 1 and 2 compare the costs of the different protocols for the commit case and abort case on a per transaction basis, respectively. The column titled "1-2PC (1PC)" denotes the 1-2PC protocol when *all* participants are 1PC, whereas the column titled "1-2PC (2PC)" denotes the 1-2PC protocol when *all* participants are 2PC. The column titled "1-2PC (MIX)" denotes the 1-2PC protocol in the presence of a mixture of both 1PC and 2PC participants. In the table, n denotes the total number of sites participating in a transaction's execution (excluding the coordinator's site), whereas p denotes the number of 1PC participants (in the case of 1-2PC protocol). 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 participants' sites. For example, to commit a transaction, 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 the 2PC involves two sequential messages in order for a coordinator to make its final commit decision regarding a transaction (i.e., the first phase), and three sequential messages to release all the resources (e.g., locks) held by the transaction at the participants. In the row labeled "Total messages with piggy-

			PrC	PrA	IYV	1-2PC
Log force delays	R	O	1	0	0	0
Total forced log writes	E	N	1	0	0	0
Message delays (Decision)	A	L	2	2	0	0
Message delays (Locks)	D	Y	1	1	1	1
Total messages			2n	2n	n	n

Table 3: The costs for *read-only* transactions.

backing", we apply *piggybacking* of the acknowledgments of decision messages to eliminate the final round of messages.

It is clear from Tables 1 and 2 that 1-2PC performs as IYV when all participants are 1PC participants, 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 number of log force writes. For the commit case, the two protocols require only one forced log write whereas for the abort case neither 1-2PC nor IYV force write any log records. When all participants are 2PC, 1-2PC performs as PrC in all performance measures (since we adopt the PrC variant in 1-2PC protocol), for the commit case as well as the abort case. When there is participants' mix, the performance of 1-2PC exhibits the behavior of the PrC with respect to the sequential performance metrics. That is, 1-2PC has the same number of "log force delays", "Message delays (Decision)" and "Message delays (Locks)" as PrC. The performance of 1-2PC with respect to the total number of messages and forced log writes depends on the participants' mix, which is less than that of PrC.

Piggybacking can be used to eliminate the final round of messages for the commit case in 2PC, PrA, IYV and 1-2PC (1PC). That is not the case for PrC, and 1-2PC (2PC) because commit decisions are never acknowledged in these protocols. Similarly, this optimization can be used for the abort case in 2PC, PrC and 1-2PC (2PC) but not in PrA, IYV or 1-2PC (1PC). This is because participants in the latter set of protocols never acknowledge abort decisions. 1-2PC (MIX) benefits from this optimization in both the commit case as well as the abort case. This is because, in a commit case, a 1PC participant acknowledges the commit decision, whereas in an abort case, a 2PC participant acknowledges the abort decision, which can be both piggybacked.

Table 3 shows the costs of the different ACPs for read-only transactions. The costs in the table are evaluated assuming the use of the standard read-only optimization 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. In the table, all the protocols have the same costs as far as the number of sequential messages to release the resources held by read-only transactions at the participants after the decision point is reached (i.e., only a single message). However, 1-2PC and IYV, which have exactly the same costs for read-only transactions, dominate PrC and PrA with respect to the number of sequential message delays to reach the decision point. 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 far as the "log force delays" and "Total forced log writes" are concerned, only PrC suffers from these delays and log writes due to the forced initiation log record at the coordinator's site. Finally, both 1-2PC and IYV require n total messages instead of $2n$, which is the case in PrC and PrA.

5. CONCLUSIONS

To achieve the performance of 1PC protocols and the wide applicability of 2PC protocols, we proposed 1-2PC protocol. 1-2PC starts as 1PC and switches to 2PC only when necessary. Furthermore, 1-2PC supports the option of forward recovery. Thus, our new protocol alleviates the applicability shortcomings of 1PC protocols in the presence of (1) deferred consistency constraints, or (2) limited network bandwidth. 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 1-2PC make it a especially important protocol in the context of environments that are characterized by high volume of short transactions such as the Internet.

6. REFERENCES

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

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

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

[5] Al-Houmaily, Y. J. 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.

[6] Gray, J. Notes on Data Base Operating Systems. In Bayer R., R.M. Graham, and G. Seegmuller (Eds.), *Operating Systems: An Advanced Course*, LNCS Vol. 60, pp. 393–481, 1978.

[7] Gupta, R., J. Haritsa and K. Ramamritham. Revisiting Commit Processing in Distributed Database Systems. *Proc. of the 1997 ACM SIGMOD*, pp. 486–497, 1997.

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

[9] Levy, E., H. F. Korth, and A. Silberschatz. An optimistic commit protocol for distributed transaction management. *Proc. of the ACM SIGMOD Conf.*, pp. 88–97, 1991.

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

[11] Samaras, G., G. Kyrou, 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.

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