Two-Phase Commit

Chapter · January 2009

DOI: 10.1007/978-1-4899-7993-3_713-2

2 authors:

Yousef J. Al-houmaily
Institute of Public Administration
26 PUBLICATIONS **333** CITATIONS
SEE PROFILE

George Samaras
University of Cyprus
286 PUBLICATIONS **3,314** CITATIONS
SEE PROFILE

Two-Phase Commit

Yousef J. Al-Houmaily[1], George Samaras[2]
[1]Institute of Public Administration, Riyadh, Saudi Arabia
[2]University of Cyprus, Nicosia, Cyprus

Definition

Two-phase commit (2PC) is a synchronization protocol that solves the *atomic commitment problem*, a special case of the *Byzantine Generals* problem. Essentially, it is used in distributed database systems to ensure *global atomicity* of transactions in spite of site and communication failures, assuming that a failure will be, *eventually*, repaired and each site guarantees atomicity of transactions at its local level.

Historical Background

2PC is the simplest and most studied *atomic commit protocol* (ACP). It was first published in [9] and [4]. Since then, the protocol has received much attention from the academia and industry due to its importance in distributed database systems, and the research has resulted in numerous variants and optimizations for different distributed database environments. These environments include main memory databases (e.g., [10]), real-time databases (e.g., [5]), mobile database systems (e.g., [12]), heterogeneous database systems (e.g., [1]), Web databases (e.g., [15]), besides traditional (homogeneous) distributed database systems (e.g., [13,3]).

Foundations

In a distributed database system, a transaction is decomposed into a set of *subtransactions*, each of which executes at a single participating database site. Assuming that each database site preserves atomicity of (sub)transactions at its local level, global atomicity cannot be guaranteed without taking additional measures. This is because without global synchronization a distributed transaction might end-up committing at some participating sites and aborting at others due to a site or a communication failure. Thus, jeopardizing *global* atomicity and, consequently, the consistency of the (distributed) database.

To achieve atomicity at the global level, there is a need for a synchronization protocol that ensures a unanimous final outcome for each distributed transaction and regardless of failures. Such a protocol is referred to as an *atomic commit protocol* (ACP). An ACP ensures that a distributed transaction is either *committed* and all its effects become persistent across all participating sites, or *aborted* and all its effects are obliterated as if the transaction had never executed at any site. This is the essence of the two-phase commit (2PC) protocol.

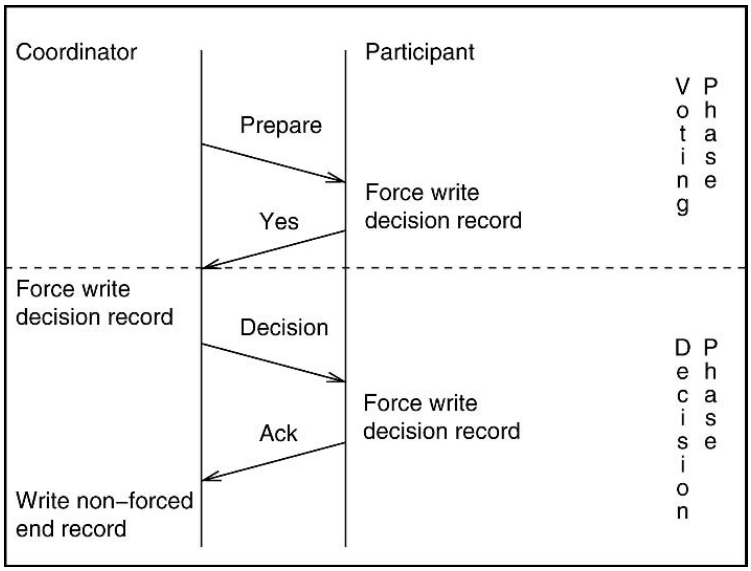
Dynamics of Two-Phase Commit

In 2PC, each transaction is associated with a designated site called the *coordinator* (or *master*). Although the coordinator of a transaction could be any of the sites participating in the transaction's execution, it is commonly the originating site of the transaction (i.e., the site where the transaction is first initiated). The rest of the sites are called *participants*, *subordinates*, *cohorts* or *slaves*. Once a transaction finishes its execution and indicates its termination point, through a commit primitive, to its coordinator, the coordinator initiates 2PC.

As the name implies, 2PC consists of two phases, namely a *voting phase* and a *decision phase*, as shown Fig. 1. During the voting phase, the coordinator requests all the sites participating in the 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 prepare-to-commit message for a transaction, it validates the transaction with respect to data consistency. If the transaction can be committed (i.e., it passed the validation process), the participant responds with a "yes" vote. Otherwise, it responds with a "no" vote and aborts the transaction, releasing all the resources held by the transaction.

If a participant had voted "yes", it can neither commit nor abort the transaction unilaterally and has to wait until it receives a final decision from the coordinator. In this case, the participant is said to be *blocked* for an indefinite period of time called *window of uncertainty* (or *window of vulnerability*) awaiting the coordinator's decision. When a participant receives the final decision, it complies with the decision, sends back an acknowledgement message (Ack) to the coordinator and releases all the resources held by the transaction.



Two-Phase Commit. Figure 1. The two-phase commit protocol.

When the coordinator receives Acks from all the participants that had voted "yes," it *forgets* the transaction by discarding all information pertaining to the transaction from its protocol table that is kept in main memory.

The resilience of 2PC to failures is achieved by recording the progress of the protocol in the logs of the coordinator and the participants. The coordinator force writes a *decision* record prior to sending out its 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 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 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 main memory. This record indicates that all (required) participants have received the decision and none of them will inquire about the transaction's status in the future. This allows the coordinator to (permanently) forget the transaction, with respect to 2PC, and garbage collect the log records of the transaction when necessary.

Recovery in Two-Phase Commit

Site and communication failures are detected by *timeouts*. When an operational site detects a failure, it invokes a recovery manager to handle the failure. In 2PC, there are four places where a communication failure might occur. The first place is when a participant is waiting for a prepare-to-commit message from the coordinator. This occurs before the participant has voted. In this case, the participant may unilaterally decide to abort the transaction. The second place is when the coordinator is waiting for the votes of the participants. Since the coordinator has not made a final decision yet and no participant could have decided to commit, the coordinator can decide to abort. The third place is when a participant had voted "yes" but has not received a commit or an abort final decision. In this case, the participant cannot make any unilateral decision because it is uncertain about the coordinator's final decision. The participant, in this case, is *blocked* until it re-establishes communication with the coordinator and, once re-established, the participant inquires the coordinator about the final decision and resumes the protocol by enforcing and, then, acknowledging the coordinator's decision. The fourth place is when the coordinator is waiting for the Acks of the participants. In this case, the coordinator re-submits its final decision to those participants that have not acknowledged the decision once it re-establishes communication with them. Notice that the coordinator cannot simply discard the information pertaining to a transaction from its protocol table or its stable log until it receives Acks from all the (required) participants.

To recover from site failures, there are two cases to consider: coordinator's failure and participant's failure.

For a coordinator's failure, the coordinator, upon its restart, scans its stable log and re-builds its protocol table to reflect the progress of 2PC for all the pending transactions prior to the failure. The coordinator has to consider only those transactions that have started 2PC and have not finished it prior to the failure (i.e., transactions that have decision log records without corresponding end log records in the stable log). For other transactions, i.e., transactions that were active at the coordinator's site prior to its failure without a decision record, the coordinator considers them as aborted transactions. Once the coordinator re-builds its protocol table, it completes the protocol for each of these transactions by re-submitting its final decision to all (required) participants whose identities are recorded in the decision record and waiting for their Acks. Since some of the participants might have already received the decision prior to the failure and enforced it, these participants might have already forgotten that the transaction had ever existed. Such participants simply reply with *blind* Acks, indicating that they have already received and enforced the decision prior to the failure.

For a participant's failure, the participant, as part of its recovery procedure, checks its log for the existence of any transaction that is in a prepared-to-commit state (i.e., has a prepared log record without a corresponding final decision one). For each such transaction, the participant inquires the transaction's coordinator about the final decision. Once the participant receives the decision from the coordinator, it completes the protocol by enforcing and, then, acknowledging the decision. Notice that a coordinator will be always able to respond to such inquires because it cannot forget a transaction before it has received the Acks of all (required) participants. However, there is a case where a participant might be in a prepared-to-commit state and the coordinator does not remember the transaction. This occurs if the coordinator fails after it has sent prepare-to-commit messages and just before it has made its decision. In this case, the coordinator will not remember the transaction after it has recovered. If a prepared-to-commit participant inquires about the transaction's status, the coordinator will *presume* that the transaction was aborted and responds with an abort message. This special case where an abort presumption is made about unremembered transactions in 2PC motivated the design of the *presumed abort* 2PC.

Underlying Assumptions

ACPs solve a special case of the problem of consensus in the presence of faults, a problem that is known in distributed systems as the *Byzantine Generals* problem [7]. This problem is, in its most general case, not solvable without some simplifying assumptions. In distributed database systems, ACPs solve the problem under the following general assumptions (among others that are sometimes ACP specific):

1. *Each site is sane*: A site is fail stop where it never deviates from its prescribed protocol. That is, a site is either operational or not but never behaves abnormally causing *commission* failures.
2. *Eventual recovery*: A failure (whether site or communication) will be, eventually, repaired.
3. *Binary outcome*: All sites unanimously agree on a single binary outcome, either commit or abort.

Performance Issues

There are three important performance issues that are associated with ACPs, which are as follows [3]:

1. *Efficiency During Normal Processing*: This refers to the cost of an ACP to provide atomicity in the absence of failures. Traditionally, this is measured using three metrics. The first metric is *message complexity* which deals with the number of messages that are needed to be exchanged between the systems participating in the execution of a transaction to reach a consistent decision regarding the final status of the transaction. The second metric is *log complexity* which accounts for the frequency at which information needs to be recorded at each participating site in order to achieve resiliency to failures. Typically, log complexity is expressed in terms of the required number of non-forced log records which are written into the log buffer (in main memory) and, more importantly, the number of forced log records which are written onto the stable log (on the disk). The third metric is *time complexity* which corresponds to the required number of rounds or sequential exchanges of messages in order for a decision to be made and propagated to the participants.
2. *Resilience to Failures*: This refers to the types of failures that an ACP can tolerate and the effects of failures on operational sites. An ACP is considered *non-blocking* if it never requires operational sites to

wait (i.e., block) until a failed site has recovered. One such protocol is 3PC.

3. *Independent Recovery*: This refers to the speed of recovery. That is, the time required for a site to recover its database and become operational, accepting new transactions after a system crash. A site can *independently* recover if it has all the necessary information needed for recovery stored locally (in its log) without requiring any communication with any other site in order to fully recover.

Most Common Two-Phase Commit Variants

Due to the costs associated with 2PC during normal transaction processing and the reliability drawbacks in the events of failures, a variety of ACPs have been proposed in the literature. These proposals can be, generally, classified as to enhance either (i) the efficiency of 2PC during normal processing or (ii) the reliability of 2PC by either reducing 2PC's blocking aspects or enhancing the degree of independent recovery. The most commonly pronounced 2PC variants are *presumed abort* (PrA) [11] and *presumed commit* (PrC) [11]. Both variants reduce the cost of 2PC during normal transaction processing, albeit for different final decisions. That is, PrA is designed to reduce the costs associated with aborting transactions whereas PrC is designed to reduce the costs associated with committing transactions.

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 the participants that have voted "yes" and discards all information about the transaction from its protocol table. That is, the coordinator of an aborted transaction does not have to write any log records or wait for Acks. Since the participants do not have to Ack abort decisions, they are also not required to force-write such decisions. After a coordinator's or a participant's failure, if the participant *inquires* about a transaction that has been aborted, the coordinator, not remembering the transaction, will direct the participant to abort the transaction (by presumption). Thus, as the name implies, if no information is found in the log of the coordinator of a transaction, the transaction is presumed aborted.

As opposed to PrA, in which missing information about transactions at a coordinator's site is interpreted as abort decisions, in PrC, a coordinator interprets missing information about transactions as commit decisions when replying to inquiry messages. 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 is made.

To commit a transaction, the coordinator force writes a commit record to logically eliminate the initiation record of the transaction and then sends out commit 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 Ack the decision. After a coordinator's or a participant's failure, if the participant *inquires* about a transaction that has been committed, the coordinator, not remembering the transaction, will direct the participant to commit the transaction (by presumption).

To abort a transaction, on the other hand, the coordinator does not write the abort decision in its log. Instead, the coordinator sends out abort messages 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 2PC. In the case of a coordinator's failure, the initiation record of an interrupted transaction contains all needed information for its recovery.

Table 1 summarizes the costs associated with the three 2PC variants for the commit as well as the abort case assuming a "yes" vote from each participant: "m" is the total number of log records, "n" is the number of forced log writes, "p" is the number of messages sent from the coordinator to each participant and "q" is the number of messages sent back to the coordinator. For simplicity, these costs are calculated for the flat (two-level) execution model in which, unlike the multi-level execution model, a participant never initiates (i.e., spawns) new (sub)transactions that execute at other participants, forming a tree of communicating participants.

Compatibility of 2PC Variants

ACPs are incompatible in the sense that they cannot be used (directly) in the same environment without conflicts. This is true even for the simplest and most

Two-Phase Commit. Table 1. The costs for update transactions in 2PC and its most commonly known two variants

2PC Variant	Commit decision						Abort decision					
	Coordinator			Participant			Coordinator			Participant		
	m	n	p	m	n	q	m	n	p	m	n	q
Basic 2PC	2	1	2	2	2	2	2	1	2	2	2	2
Presumed abort	2	1	2	2	2	2	0	0	2	2	1	1
Presumed commit	2	2	2	2	1	1	2	1	2	2	2	2

closely related variants such as the basic 2PC, PrA and PrC. The analysis of ACPs shows that incompatibilities among ACPs could be due to (i) the semantics of the coordination messages (which include both their meanings as well as their existence), or (ii) the presumptions about the outcome of terminated transactions in case of failures [1].

The *presumed any* (PrAny) protocol [2] interoperates the basic 2PC, PrA, and PrC. It was proposed in the context of multidatabase systems, a special case of heterogeneous distributed databases, to demonstrate the difficulties that arise when one attempts to interoperate different ACPs in the same environment and, more importantly, to introduce the "*operational correctness criterion*" and the notion of "*safe state*." Operational correctness means that all sites should be able, not only to reach an agreement but also, to 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.

In PrAny, a coordinator talks the language of the three 2PC variants and knows which variant is used by which participant. Based on that, it forgets a committed transaction once all PrA and 2PC participants Ack the commit decision, and forgets an aborted transaction once all PrC and 2PC participants Ack the abort decision. This is because only commit decisions are acknowledged in PrA whereas, in PrC, only abort decisions are acknowledged. However, unlike the other 2PC variants, in PrAny, a coordinator does not adopt a single presumption about the outcome of *all* terminated transactions. This is because, if it does so, the global atomicity of some transactions might be violated. For example, if the coordinator adopts for recovering purposes the abort presumption, it will respond with an abort message to a recovering PrC participant that inquires about a forgotten committed transaction. Similarly, if the coordinator adopts the commit presumption, it will respond with a commit message to a recovering PrA participant that inquires about a forgotten aborted transaction. Instead of using a single presumption, a coordinator in PrAny adopts the presumption of the protocol used by the inquiring participant. That is, if a participant inquires about a forgotten committed transaction, the participant has to be a PrC participant. This is because only PrC participants do not acknowledge commit decisions. Thus, the coordinator will reply with a commit message in accordance with PrC adopted by the participant. On the other hand, if a participant inquires about a forgotten aborted transaction, the participant has to be a PrA participant. This is because only PrA participants do not acknowledge abort decisions. Thus, the coordinator will reply with an abort message in accordance with PrA adopted by the participant. Knowledge about the used protocols by the participants could be recorded statically at the coordinator's site [2] or inferred dynamically by having a participant declares its used protocol in each inquiry message [15].

Key Applications

The use of 2PC (or one of its variants) is mandatory in any distributed database system in which the traditional atomicity property of transactions is to be preserved. However, the basic 2PC has never been implemented in any commercial database system due to its unnecessary costs compared to its two other most commonly known variants. Instead, PrA is considered the *de facto* standard in the industry and has been incorporated as part of the current X/Open DTP [14] and ISO OSI-TP [6] distributed transaction processing standards. PrA is chosen instead of PrC because (i) the cost of PrC for committing transactions is not symmetric with the

cost of PrA for aborting transactions (which is highlighted in Table 1) and, more importantly, (ii) PrA is much cheaper to use with read-only transactions when complementing it with the traditional *read-only* optimization [13,3] (which is also part of the current database standards).

The above two reasons that favor PrA have been nullified with new 2PC variants and a read-only optimization called *unsolicited update-vote* (UUV). Thus, PrC is expected to become also part of future database standards, especially that the two variants can be incorporated in the same environment without any conflicts [1,15].

Cross-references

- ▶ Atomicity
- ▶ Distributed Database Systems
- ▶ Distributed Recovery
- ▶ Distributed Transaction Management

Recommended Reading

1. Al-Houmaily Y. Incompatibility dimensions and integration of atomic commit protocols. Int. Arab J. Inf. Technol., 5(4):2008.
2. Al-Houmaily Y. and Chrysanthis P. Atomicity with incompatible presumptions. In Proc. 18th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1999, pp. 306–315.
3. Chrysanthis P.K., Samaras G., and Al-Houmaily Y. Recovery and performance of atomic commit processing in distributed database systems, In Recovery Mechanisms in Database Systems, V. Kumar, M. Hsu (eds.). Prentice Hall, Uppersaddle River, NJ, 1998, pp. 370–416.
4. Gray J.N. Notes on data base operating systems. In Operating Systems – An Advanced Course. M.J. Flynn et al. (eds.), LNCS, Vol. 60, Springer, London, 1978, pp. 393–481.
5. Haritsa J., Ramamritham K., and Gupta R. The PROMPT real-time commit protocol. IEEE Trans. Parallel Distributed Syst., 11(2):160–181, 2000.
6. ISO. Open systems interconnection – Distributed transaction processing – Part 1: OSI TP Model. ISO/IEC, 10026–1, 1998.
7. Lamport L., Shostak R., and Pease M. The Byzantine generals problem. ACM Trans. Programming Lang. Syst., 4(3):382–401, 1982.
8. Lampson B. and Lomet D. A new presumed commit optimization for two phase commit. In Proc. 19th Int. Conf. on Very Large Data Bases, 1993, pp. 630–640.
9. Lampson B. and Sturgis H. Crash recovery in a distributed data storage system. Technical report, Computer Science Laboratory, Xerox Palo Alto Research Center, CA, 1976.
10. Lee I. and Yeom H. A single phase distributed commit protocol for main memory database systems. In Proc. 16th Int. Parallel and Distributed Processing Symp., 2002, pp. 14–21.
11. Mohan C., Lindsay B., and Obermarck R. Transaction management in the R* distributed data base management system. ACM Trans. Database Syst., 11(4):378–396, 1986.
12. Nouali N., Drias H., and Doucet A. A mobility-aware two-phase commit protocol. Int. Arab J. Inf. Technol., 3(1):2006.
13. Samaras G., Britton K., Citron A., and Mohan C. Two-phase commit optimizations in a commercial distributed environment. Distrib. Parall. Databases, 3(4):325–361, 1995.
14. X/Open Company Limited. Distributed Transaction Processing: Reference Model. Version 3 (X/Open Document No. 504), 1996.
15. Yu W. and Pu C. A Dynamic Two-phase commit protocol for adaptive composite services. Int. J. Web Serv. Res., 4(1):2007.

Two-Phase Commit Protocol

Jens Lechtenbörger
University of Münster, Münster, Germany

Synonyms

XA standard

Definition

The *Two-phase commit (2PC)* protocol is a distributed algorithm to ensure the consistent termination of a transaction in a distributed environment. Thus, via 2PC a unanimous decision is reached and enforced among multiple participating servers whether to commit or abort a given transaction, thereby guaranteeing atomicity. The protocol proceeds in two phases, namely the prepare (or voting) and the commit (or decision) phase, which explains the protocol's name.

The protocol is executed by a *coordinator* process, while the participating servers are called *participants*. When the transaction's initiator issues a request to commit the transaction, the coordinator starts the first phase of the 2PC protocol by querying – *via prepare messages* – all participants whether to abort or to commit the transaction. If all participants vote to commit then in the second phase the coordinator informs all participants to commit their share of the transaction by sending a *commit message*. Otherwise, the coordinator instructs all participants to abort their share of the transaction by sending an *abort message*. Appropriate log entries are written by coordinator as well as participants to enable restart procedures in case of failures.

Historical Background

Essentially, the 2PC protocol is modeled after general contract law, where a contract among two or more