Three-Phase Commit

Chapter · January 2009

DOI: 10.1007/978-0-387-39940-9_714

CITATIONS
2

READS
3,311

2 authors:

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

George Samaras
University of Cyprus
286 PUBLICATIONS **3,315** CITATIONS
SEE PROFILE

3D models is essential for the geometric 3D reconstruction of geological structures and geological processes [19].

Cross-references

▶ Digital Elevation Models
▶ Geography Markup Language
▶ Simplicial Complex
▶ Spatial Network Databases

Recommended Reading

1. Abdul-Rahman A., Zlatanova S., and Coors V. (eds.). Innovations in 3D Geoinformation Systems, Lecture Notes in Geoinformation and Cartography, Springer, Heidelberg, 2006.
2. Balovnev O., Bode T., Breunig M., Cremers A.B., Müller W., Pogodaev G., Shumilov S., Siebeck J., Siehl A., and Thomsen A. The story of the GeoToolKit – an object-oriented geodatabase kernel system. Geoinformatica, 8(1):5–47, 2004.
3. Beckmann N., Kriegel H.-P., Schneider R., and Seeger B. The R*-tree: an efficient and robust access method for points and rectangles. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1990, pp. 322–331.
4. Brisson E. Representing geometric structures in d dimensions: topology and order. In Proc. 5th Annual Symp. on Computational Geometry, 1989, pp. 218–227.
5. Coors V. and Zipf A (eds.). 3D-Geoinformationssysteme, Grundlagen und Anwendungen. Wichmann – Hüthig, Heidelberg, 2004.
6. GOCAD. http://www.gocad.org.
7. Götze H.J. and Lahmeyer B. Application of three-dimensional interactive modelling in gravity and magnetics. Geophysics, 53(8), 1988, pp. 1096–1108.
8. Güting R.H. An introduction to spatial database systems. VLDB J., 3(4):357–399, 1994.
9. Guttman A. R-Trees: a dynamic index structure for spatial searching. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1984, pp. 47–57.
10. Lienhardt P. Subdivision of n-dimensional spaces and n-dimensional generalized maps. In Proc. 5th Annual Symp. on Computational Geometry, 1989, pp. 228–236.
11. Lienhardt P. N-dimensional generalized combinatorial maps and cellular quasi-manifolds. J. Comp. Geom. App., 4(3): 275–324, 1994.
12. Lévy B. and Mallet J.-L. Discrete Smooth Interpolation: Constrained Discrete Fairing for Arbitrary Meshes, ISA-GOCAD (Inria Lorraine/CNRS), ENSG, Vandoeuvre Nancy, http://www.earthdecision.com/news/white_papers/DSI.pdf.
13. Mallet J.L. Geomodelling. Oxford University Press, New York, NY, 2002.
14. OGC. http://www.opengeospatial.org.
15. Pigot S. A topological model for a 3D spatial information system. In Proc. 5th Int. Symp. on Spatial Data Handling, 1992, pp. 344–360.
16. Raper J. (Ed.). Three dimensional applications in geographical information systems. Taylor & Francis, London, 1989.
17. Samet H. The design and analysis of spatial data structures, Addison-Wesley, Reading, 1990.
18. Schaeben H., Apel M., v.d. Boogart G., Kroner U. GIS 2D, 3D, 4D, nD. Informatik-Spektrum, 26(3), 2003, pp. 173–179.
19. Siehl A. Construction of geological maps based on digital spatial models. Geol. Jb. A., 104:253–261, 1988.
20. Turner A.K. (ed.) Three-dimensional modeling with geoscientific information systems, Kluwer Academic, Dordrecht, 1991.
21. van Oosterom P., Zlatanova S., Penninga F., and Fendel E. (eds.) Advances in 3D geoinformation systems, Lecture Notes in Geoinformation and Cartography. Springer, Heidelberg, 2007.
22. Vinken R. Digital geoscientific maps – a research project of the DFG. In Proc. Int. Colloquium at Dinkelsbühl, Geolog. Jahrbuch A104, 1988, pp. 7–20.

Three-Dimensional Similarity Search

▶ Feature-Based 3D Object Retrieval

Three-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

Three-phase commit (3PC) is a synchronization protocol that ensures *global atomicity* of distributed transactions while alleviating the *blocking* aspect of 2PC (Two-Phase Commit) in the events of *site* failures. That is, 3PC never requires operational sites to wait (i.e., block) until a failed site has recovered.

Historical Background

3PC was one of the first attempts to resolve the blocking aspects of 2PC [6]. The main purpose of the protocol is to allow operational sites to continue transaction processing and reach agreement about the final status of transactions in spite of the presence of site failures. 3PC can tolerate any number of site failures (except for total sites' failures), assuming a highly reliable network (i.e., a network that never causes operational sites to be partitioned into more than one set of communicating sites, implying a network that never fails).

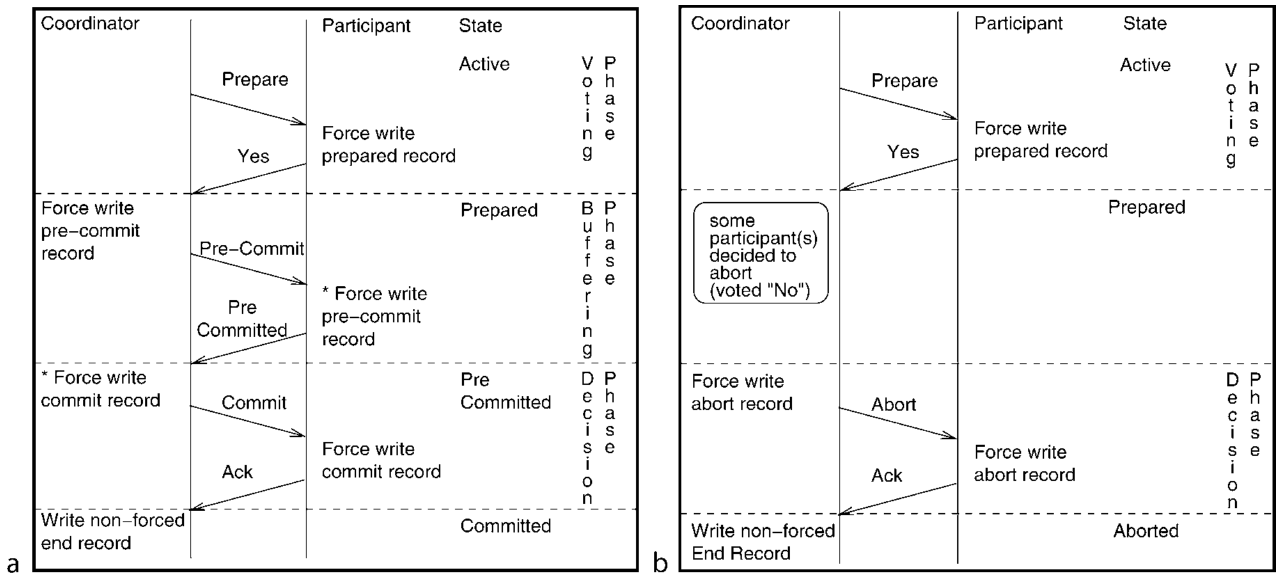
Foundations

In 2PC, a participant is *blocked* if it fails to communicate with the coordinator of a transaction while in a prepared-to-commit state. Blocking means that the participant cannot determine the final status of the transaction in the presence of a failure, rendering all resources held by the prepared-to-commit transaction at its site unusable by any other transaction until the final status of the transaction is resolved, i.e., the transaction is either committed or aborted. Blocking is inevitable in 2PC and it may occur either because of (i) a communication (link) failure or (ii) a coordinator's system crash. These two types of failures lead to blocking, in 2PC, even under the assumption that all participants remain operational and can communicate collaboratively to resolve the status of the prepared-to-commit transaction. For example, if a coordinator fails after all participating sites in a transaction's execution have entered their prepared-to-commit states; the participants (collectively) can neither commit nor abort the transaction. This is because the operational participants cannot be sure whether the coordinator had received all their votes and made a commit final decision just before it failed or it had received only some votes and did not have the chance to make the final decision before its failure and the transaction will be aborted by the coordinator when it recovers. Thus, the participants are blocked until the coordinator recovers.

The negative impact of blocking on the (i) overall system performance and (ii) availability of critical data on other transactions motivated the design of *non-blocking* atomic commit protocols (ACPs). In 3PC, an *extra (buffering) phase* is inserted between the two phases of 2PC to capture all the participants' intentions to commit, as shown in Fig. 1.

Dynamics of Three-Phase Commit

The basic idea behind the insertion of the buffering phase is to place the two (possible) reachable final states (i.e., the commit and the abort states) for a transaction apart from each other such that they cannot be reached from the *same* state. That is, if a final state can be reached from the current state of a site, then, the reachable final state can be either the commit or the abort state but not both. In 2PC, when a participant is in a prepared-to-commit state, both final states can be reached. Hence, if the coordinator fails, the participant cannot determine the final state for the transaction without any possible conflict in its decision with the coordinator. For this reason, the protocol is blocking. On the contrary and as shown in Fig. 1, the commit final state for a site (whether it is the coordinator or a participant), in 3PC, cannot be reached from the same


Coordinator
Participant
State
Active
Prepare
Yes
Force write prepared record
Voting Phase
Force write pre-commit record
Prepared
Pre-Commit
Pre Committed
* Force write pre-commit record
Buffering Phase
* Force write commit record
Commit
Ack
Force write commit record
Pre Committed
Decision Phase
Write non-forced end record
Committed
a
some participant(s) decided to abort (voted "No")
Force write abort record
Abort
Force write abort record
Write non-forced End Record
Aborted
b

* This record could be written in a non-forced manner without affecting the correctness of the protocol.

Three-Phase Commit. Figure 1. The three-phase commit protocol.

state as the abort final state. In the former case, the commit state can be reached from the *pre-commit* state whereas, in the latter case, the abort state can be reached from the *prepared* state.

When a site is in a pre-commit state, it means that each of the other sites is at least in a prepared-to-commit state (Notice that the other sites might lag in their states because of system's delays such as queuing and network delays.). Thus, the pre-commit state is called a *committable state* since it implies that all participants have voted "yes" and the coordinator agreed on the commitment of the transaction. In 3PC, a *non-committable* state, i.e., a state that does not imply that all the participants have voted "yes," is not placed adjacent to a commit state. This is not the case in 2PC as the prepared state, which non-committable, is placed adjacent to a commit state.

The insertion of the buffering state makes the structure of 3PC to satisfy the two necessary and sufficient conditions for the construction of synchronous non-blocking ACPs within one state transaction, i.e., a structure where neither the coordinator nor any participant leads each other by more than one state transition during its execution of the protocol. That is, 3PC is synchronous within one state transaction that (i) does not contain a state that is adjacent to both a commit and an abort state, and (ii) it does not contain a non-committable state that is adjacent to a commit state.

Based on the above, if the coordinator of a transaction fails at any point during the execution of the protocol, the operational participants can collectively and deterministically decide the final status of the transaction. The decision is commit if any of the participants is in *at least* a pre-commit state (because it is not possible for the coordinator to have decided to abort). Otherwise, the decision is abort (because it could be possible for the coordinator to have decided to abort but not to commit). To reach an agreement on the final status of a transaction, there is a need for a termination protocol which is invoked when the coordinator fails.

Recovery in Three-Phase Commit

When a participant times out while waiting for a message from the coordinator, it means that the coordinator must have failed (or it is perceived as a coordinator failure). In this case, the participant initiates an election protocol to determine a *new* coordinator. One way to determine the new coordinator is based on sites' identification numbers such that the participant with the highest (or the lowest) number becomes the new coordinator. Once a new coordinator is elected, the participants exchange status information about the transaction. If the new coordinator finds the transaction in at least a pre-commit state at any participant, it commits (in its local log) the transaction; otherwise, it aborts the transaction. Then, this new coordinator proceeds to complete the 3PC for the transaction in all the other participants. If the new coordinator fails, the election process is repeated again.

When a participant starts recovering from a failure, it needs to determine the status of each prepared or pre-committed transaction as recorded in its log. Notice that a recovering participant cannot commit a transaction even if the participant is in a pre-commit state with respect to the transaction. This is because the operational sites might have decided to abort the transaction after the participant had failed if none of them was in a pre-commit state. In this case, the participant must ask the other sites about the final status of the transaction.

A final note on 3PC is about total sites' failure where there is a need to determine the last participant to have failed. This is because such participant is the only one which can decide the status of the transaction for the other participants. Determining the last participant that has failed could be implemented by maintaining an "UP" list at each participating site. This list contains the identities of operational participants as seen by the participant that is maintaining the list and is stored (in a non-forced or asynchronous manner) onto the stable log of the participant. Thus, the "UP" lists allow a set of participants to determine, upon their recovery from the total failure, whether they contain among themselves the last participant to have failed, reducing the number of participants that needs to recover before the transaction status can be resolved. Alternatively, all participants should recover and become operational again before the status of the transaction can be resolved.

Non-Blocking Commit Protocol Variants

As discussed above, blocking occurs in 2PC when the coordinator of a transaction crashes while the transaction is in its *prepared-to-commit* state at a participating site. In such a case, the participant is blocked until the coordinator of the transaction recovers. In general, all ACPs are susceptible to blocking. They just differ in the size of the window during which a site might be blocked and the type of failures that cause their blocking. Several ACPs have been designed to eliminate some of the blocking aspects of 2PC, besides 3PC, by adding extra coordination messages and forced log writes. These protocols can be classified into whether they preserve the prepared-to-commit state, such as cooperative 2PC, or allow *unilateral or heuristic* decisions in the presence of unbearable delays, such as IBM's presumed nothing (IBM-PrN).

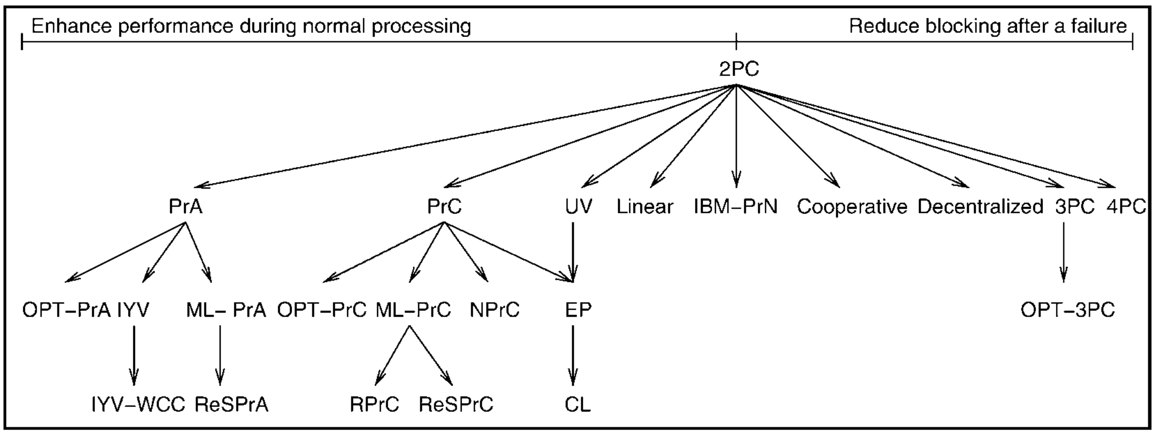
The *cooperative* 2PC (1981) reduces the *likelihood* of blocking in case of a coordinator's failure. In the cooperative 2PC, the identities of all participants are included in the prepare-to-commit message so that each participant becomes aware of the other participants. In the case of a coordinator's or a communication's failure, a participant does not block waiting until it reestablishes communication with the coordinator. Instead, it inquires the other operational participants in the transaction's execution about the final decision and if any of them has already received the final decision prior to the failure, it informs the inquiring participant accordingly.

The IBM-PrN (1990) is a 2PC variant that allows blocked participants to unilaterally commit or abort a transaction and detects atomicity violations due to conflicting heuristic decisions. In the event of atomicity violations, it reports any damage on transactions and data, simplifying the task of identifying problems that must be fixed. *Generalized presumed abort* (1994) is another IBM protocol that behaves like IBM-PrN when complete confidence in the final outcome and recognition of heuristic decisions is required and behaves like PrA during normal processing. Recent efforts to enhance commit protocols with heuristic decision processing resulted in the *allow-heuristics presumed nothing* (1996) commit protocol.

Other Atomic Commit Protocol Variants and Optimizations

Figure 2 shows some of the significant steps in the evolution of ACPs including the two most notable 2PC variants which are *presumed abort* (PrA) and *presumed commit* (PrC) [2]. The *new PrC* (NPrC) (1993) and *rooted PrC* (RPrC) (1997) protocols were proposed to reduce the log complexity of PrC further at the cost of slower recovery in the presence of failures. NPrC eliminates the initiation log record at the coordinator's site whereas RPrC eliminates the initiation log record at each cascaded coordinator when the *tree-of-processes* (or *multi-level transaction execution*) model is used.

In contrast to PrA and PrC variants, other 2PC variants have been proposed for specific environments. The common characteristic of these protocols is that they exploit the semantics of the communication networks, the database management systems and/or the transactions to enhance the performance of 2PC. For example, the *linear* 2PC (L2PC) reduces message complexity at the expense of time complexity compared to 2PC by assuming token-ring like networks. In L2PC, the participants are linearly ordered with the coordinator being the first in the linear order. The coordinator initiates the voting and each participant sends its "yes" vote to its successor in the linear order. The last participant in the order makes the decision and sends it to its predecessor and so on. In this way, L2PC maintains the same log complexity as 2PC, reduces



Three-Phase Commit. Figure 2. Some significant steps in the evolution of ACPs.

the message complexity of 2PC from “3” to “2*n*” while increasing the time complexity of 2PC from “3” to “2*n*” rounds, where *n* is the number of participants. In contrast to L2PC, *decentralized* 2PC (D2PC) reduces time complexity at the expense of message complexity which is $n^2 + n$ messages. In D2PC, the interconnecting communication network is assumed to be fully connected and efficiently supports the broadcasting of messages. In D2PC, two rounds of messages are required for each individual participant to make a final decision. During the first round, the coordinator broadcasts its vote (implicitly initiating commit processing) whereas, during the second one, all the participants broadcast their votes. Thus, each participant receives the votes of all the other participants, as well as the coordinator, and thereby, is able to independently conclude the final decision. By reducing the time complexity to two rounds, it becomes less likely for a participant, in D2PL, to be blocked during commit processing in the case of a coordinator’s failure.

There are four transaction type specific 2PC protocols, all of which, when applicable, improve both the message and time complexities of 2PC by eliminating the explicit voting phase of 2PC. The *unsolicited-vote* protocol (UV) (1979) shortens the voting phase of 2PC assuming that each participant knows when it has executed the last operation for a transaction. In this way, a participant sends its vote on its own initiative once it recognizes that it has executed the last operation for the transaction. When the coordinator receives the votes of the participants, it proceeds with the decision phase. The *early prepare* protocol (EP) (1990) combines UV with PrC without assuming that a participant can recognize the last operation of a transaction. Every operation is, therefore, treated as if it is the last operation executing at the participant and its acknowledgment is interpreted as a “yes” vote. This means that a participant has to force write its log each time it executes an operation so that it can preserve the global atomicity of the transaction after a system crash. In contrast to EP which reduces time and message complexities at the expense of log complexity, the *coordinator log* (CL) and *implicit yes-vote* (IYV) protocols do not require force writing the log records, at the participants’ sites, after the execution of each operation. Instead, they replicate the participants’ logs at the coordinators’ site. Hence, reducing log complexity compared to EP at the expense, however, of slower recovery. In CL, a participant does not maintain a local stable log and, therefore, it has to contact all coordinators in the system in order to recover after a failure. Moreover, a participant may need to contact the coordinator of a transaction during normal processing to maintain *write-ahead logging* (WAL) or to undo the effects of an aborting transaction. This is because the log of a participant is scattered across the coordinators’ sites. In contrast, the log of a participant in IYV is partially replicated across the coordinators’ sites. That is, only the *redo* records are replicated at the coordinators’ sites while the *undo* records are stored locally. Thus, a participant, in IYV, never communicates with any coordinator to maintain WAL or to undo the effects of aborting transactions. Thus, in IYV, the replicated records are used only to recover a participant after a system’s crash. Furthermore, IYV is based on PrA while CL is derived from PrC.

Existing 2PC variants are incompatible with each other and need to be made to interoperate in order to be integrated in (heterogeneous) multidatabase systems and the Internet. Thus, the continued research for more efficient ACPs has expanded to include the investigation of integrated ACPs. Efforts in this direction include the Harmony prototype system that integrates *centralized* participants that use centralized (asymmetric) ACPs with *centralized* participants that use decentralized (symmetric) ACPs (1991), the integration of *distributed* participants that use symmetric ACPs with *distributed* participants that use asymmetric ACPs (1994), and the *presumed any* protocol (1996) that integrates participants that use 2PC, PrA, or PrC. Besides that, recent efforts are targeted towards understanding the sources of incompatibilities [1] and the integration of ACPs in an adaptive manner to achieve higher system performance [8].

Several optimizations have been proposed that can reduce the costs associated with ACPs [5,2]. These include the *read-only, last agent, group commit, sharing the log, flattening the transaction tree* and *optimistic* optimizations. The read-only optimizations can be considered as the most significant ones, given that read-only transactions are the majority in any general database system. In fact, the performance gains allowed by the *traditional read-only* optimization provided the argument in favor of PrA to become the current choice of ACPs in the ISO OSI-TP (1998) and X/Open DTP (1996) distributed transaction processing standards, and commercial systems. The basic idea behind the read-only optimizations is that a read-only participant, a participant that has not performed any updates on behalf of a transaction, can be excluded from the decision phase of the transaction. This is because it does not matter whether the transaction is finally committed or aborted at a read-only participant to ensure the transaction's atomicity. In the traditional read-only optimization [4], a read-only participant votes "read-only" instead of a "yes" and immediately releases all the resources held by the transaction without writing any log records. A "read-only" vote allows a coordinator to recognize and discard the read-only participant from the rest of the protocol. The *unsolicited update-vote* (1997) is another read-only optimization that further reduces the costs associated with read-only participants. Not only that, but it incurs the same costs when used with both PrA and PrC, supporting the arguments for PrC to be also included in the standards.

The *last agent* optimization has been implemented by a number of commercial systems to reduce the cost of commit processing in the presence of a *single remote* participant. In this optimization, a coordinator first prepares itself and the nearby participants for commitment (fast first phase), and then delegates the responsibility of making the final decision to the remote participant. This eliminates the voting phase involving the remote participant. This same idea of delegating part of commitment (i.e., transferring the commit responsibilities) from one site to another has been also used to reduce blocking, for example, in open commit protocols (1990) and IYV with a commit coordinator (1996).

The *group commit* optimization has been also implemented by a number of commercial products to reduce log complexity. In the context of centralized database systems, a commit record pertaining to a transaction is not forced on an individual basis. Instead, a single force write to the log is performed when a number of transactions are to be committed or when a timer has expired. In the context of distributed database systems, this technique is used at the participants' sites *only* for the commit records of transactions during commit processing. The *lazy commit* optimization is a generalization of the *group commit* in which not only the commit records at the participants are forced in a group fashion, but *all* log records are lazily forced written onto stable storage during commit processing. Thus, the cost of a single access to the stable log is amortized among several transactions. The s*haring of log* between the transaction manager and data managers [5] at a site is another optimization that takes advantage of the sequential nature of the log to eliminate the need of force writing the log by the data managers.

The *flattening of the transaction tree* optimization is targeted for the tree-of-processes transaction model and is a big performance winner in distributed transactions that contain deep trees. It can reduce both the message and log complexities of an ACP by transforming the transaction execution tree of *any* depth into a two-level commit tree at commit initiation time. In this way, the root coordinator sends coordination messages directly to, and receives messages directly from, any participant. Thus, avoiding propagation delays and sequential forcing of log records. *Restructuring-the-commit-tree-around-update-participants* (RCT-UP) is an enhancement to the flattening technique that flattens only update participants (participants that have executed update operations on behave of the transaction), thereby, connecting them directly to the coordinator while leaving read-only participants connected in a multi-level manner. This is to reduce the effects of the communication delays on the overall system performance in systems that do not support simultaneous message multicasting to all participants.

Another optimization is *optimistic* (OPT) (1997) which can enhance the overall system performance by reducing blocking arising out of locks held by prepared transactions. OPT shares the same assumption as PrC, that is, transactions tend to commit when they reach their commit points. Under this assumption, OPT allows a transaction to *borrow* data that have been modified by another transaction that has entered a prepared-to-commit state and has not committed. A borrower is aborted if the lending transaction is finally aborted.

Key Applications

3PC has never been implemented in any commercial database system due to its cost, during normal transaction processing, compared to the other ACPs. This is besides its implementation complexity and, especially, the complexity of its termination protocol. Even with the added implementation complexity and cost, 3PC does not *completely* eliminate blocking since it is still susceptible to blocking in case of network partitioning. In fact there is no ACP that is non-blocking in the case of site as well as communication failures. This is an inherent characteristic of the *Byzantine Generals Problem*, the more general problem of atomic commitment. However, the protocol remains an instrumental theoretical result for understanding the behavior of ACPs and the limitations in solving the atomic commitment problem.

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. Chrysanthis P.K., Samaras G., and Al-Houmaily Y. Recovery and performance of atomic commit processing in distributed database systems, Chapter 13. In Recovery Mechanisms in Database Systems, V. Kumar, M. Hsu (eds.). Prentice Hall, Upper Saddle River, NJ, 1998, pp. 370–416.
3. Lamport L., Shostak R., and Pease M. The byzantine generals problem. ACM Trans. Programming Lang. Syst., 4(3):382–401, 1982.
4. 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.
5. 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.
6. Skeen D. Non-blocking Commit Protocols. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1981, pp. 133–142.
7. Skeen D. and Stonebraker M. A Formal model of crash recovery in a distributed system. IEEE Trans. Softw. Eng., 9(3):219–228, 1983.
8. Yu W. and Pu C. A Dynamic two-phase commit protocol for adaptive composite services. Int. J. Web Serv. Res., 4(1), 2007.

Thresholding

▶ Image Segmentation

Tight Coupling

SERGUEI MANKOVSKII
CA Labs, CA Inc., Thornhill, ON, Canada

Synonyms

Strong coupling

Definition

Tight coupling indicates strong dependency between software components. The dependency is strong in the sense that two or more components have multiple and complex dependencies between internal states, data and functions of the components.

Key Points

Tight coupling often achieves high performance characteristics at the expense of flexibility and ease of maintenance. It is often justified for systems that are going to be used as a black box, with no expectation of ongoing maintenance or upgrade. The smaller a systems is, the more congruent its function is, the more likely it would justify tight coupling as a design principle. This is why one can find tight coupling in the components of larger systems. In this situation a number of tightly coupled components might be interacting using loosely coupled approach. This type of systems leads to robust designs where maintainability and flexibility is achieved by simply replacing one or more components. At the same time, each component performs in the best possible way because of tight coupling inside of it. This is the idea behind Service Oriented Architectures (SOA). This way SOA achieves balance between performance and flexibility. However it is not of the case for other architectures and because of that it is often beneficial to evaluate trade-offs of both approaches during system design.

Cross-references

▶ Loose Coupling
▶ SOA
▶ Tight Coupling

Time Aggregated Graphs

BETSY GEORGE, SHASHI SHEKHAR
University of Minnesota, Minnesota, Minneapolis, MN, USA

Synonyms

Spatio-temporal graphs; Time-dependent graphs; Time-dependent networks

Definition

A time aggregated graph [1,2] is a model that can be used to represent a spatio-temporal network. The topology and the attributes in a spatio-temporal