WIKIPEDIA

Two-phase commit protocol

In transaction processing, databases, and computer networking, the **two-phase commit protocol** (**2PC**) is a type of atomic commitment protocol (ACP). It is a distributed algorithm that coordinates all the processes that participate in a distributed atomic transaction on whether to *commit* or *abort* (*roll back*) the transaction (it is a specialized type of consensus protocol). The protocol achieves its goal even in many cases of temporary system failure (involving either process, network node, communication, etc. failures), and is thus widely used.[1][2][3] However, it is not resilient to all possible failure configurations, and in rare cases, manual intervention is needed to remedy an outcome. To accommodate recovery from failure (automatic in most cases) the protocol's participants use logging of the protocol's states. Log records, which are typically slow to generate but survive failures, are used by the protocol's recovery procedures. Many protocol variants exist that primarily differ in logging strategies and recovery mechanisms. Though usually intended to be used infrequently, recovery procedures compose a substantial portion of the protocol, due to many possible failure scenarios to be considered and supported by the protocol.

In a "normal execution" of any single distributed transaction (i.e., when no failure occurs, which is typically the most frequent situation), the protocol consists of two phases:

1. The *commit-request phase* (or *voting phase*), in which a *coordinator* process attempts to prepare all the transaction's participating processes (named *participants*, *cohorts*, or *workers*) to take the necessary steps for either committing or aborting the transaction and to *vote*, either "Yes": commit (if the transaction participant's local portion execution has ended properly), or "No": abort (if a problem has been detected with the local portion), and
2. The *commit phase*, in which, based on *voting* of the participants, the coordinator decides whether to commit (only if *all* have voted "Yes") or abort the transaction (otherwise), and notifies the result to all the participants. The participants then follow with the needed actions (commit or abort) with their local transactional resources (also called *recoverable resources*; e.g., database data) and their respective portions in the transaction's other output (if applicable).

Note that the two-phase commit (2PC) protocol should not be confused with the two-phase locking (2PL) protocol, a concurrency control protocol.

Contents

Assumptions

The protocol works in the following manner: one node is a designated **coordinator**, which is the master site, and the rest of the nodes in the network are designated the **participants**. The protocol assumes that there is stable storage at each node with a write-ahead log, that no node crashes forever, that the data in the write-ahead log is never lost or corrupted in a crash, and that any two nodes can communicate with each other. The last assumption is not too restrictive, as network communication can typically be rerouted. The first two assumptions are much stronger; if a node is totally destroyed then data can be lost.

The protocol is initiated by the coordinator after the last step of the transaction has been reached. The participants then respond with an **agreement** message or an **abort** message depending on whether the transaction has been processed successfully at the participant.

Basic algorithm

Commit request (or voting) phase

1. The coordinator sends a **query to commit** message to all participants and waits until it has received a reply from all participants.
2. The participants execute the transaction up to the point where they will be asked to commit. They each write an entry to their *undo log* and an entry to their *redo log*.
3. Each participant replies with an **agreement** message (participant votes **Yes** to commit), if the participant's actions succeeded, or an **abort** message (participant votes **No**, not to commit), if the participant experiences a failure that will make it impossible to commit.

Commit (or completion) phase

Success

If the coordinator received an **agreement** message from *all* participants during the commit-request phase:

1. The coordinator sends a **commit** message to all the participants.
2. Each participant completes the operation, and releases all the locks and resources held during the transaction.

3. Each participant sends an **acknowledgment** to the coordinator.
4. The coordinator completes the transaction when all acknowledgments have been received.

Failure

If *any* participant votes **No** during the commit-request phase (or the coordinator's timeout **expires**):

1. The coordinator sends a **rollback** message to all the participants.
2. Each participant undoes the transaction using the undo log, and releases the resources and locks held during the transaction.
3. Each participant sends an **acknowledgement** to the coordinator.
4. The coordinator undoes the transaction when all acknowledgements have been received.

Message flow

```
Coordinator                                     Participant
                        QUERY TO COMMIT
                -------------------------------->
                        VOTE YES/NO             prepare*/abort*
                <--------------------------------
commit*/abort*          COMMIT/ROLLBACK
                -------------------------------->
                        ACKNOWLEDGMENT          commit*/abort*
                <--------------------------------
end
```

An * next to the record type means that the record is forced to stable storage.[4]

Disadvantages

The greatest disadvantage of the two-phase commit protocol is that it is a blocking protocol. If the coordinator fails permanently, some participants will never resolve their transactions: After a participant has sent an **agreement** message to the coordinator, it will block until a **commit** or **rollback** is received.

Implementing the two-phase commit protocol

Common architecture

In many cases the 2PC protocol is distributed in a computer network. It is easily distributed by implementing multiple dedicated 2PC components similar to each other, typically named *Transaction managers* (TMs; also referred to as *2PC agents* or Transaction Processing Monitors), that carry out the protocol's execution for each transaction (e.g., The Open Group's X/Open XA). The databases involved with a distributed transaction, the *participants*, both the coordinator and

participants, *register* to close TMs (typically residing on respective same network nodes as the participants) for terminating that transaction using 2PC. Each distributed transaction has an ad hoc set of TMs, the TMs to which the transaction participants register. A leader, the coordinator TM, exists for each transaction to coordinate 2PC for it, typically the TM of the coordinator database. However, the coordinator role can be transferred to another TM for performance or reliability reasons. Rather than exchanging 2PC messages among themselves, the participants exchange the messages with their respective TMs. The relevant TMs communicate among themselves to execute the 2PC protocol schema above, "representing" the respective participants, for terminating that transaction. With this architecture the protocol is fully distributed (does not need any central processing component or data structure), and scales up with number of network nodes (network size) effectively.

This common architecture is also effective for the distribution of other atomic commitment protocols besides 2PC, since all such protocols use the same voting mechanism and outcome propagation to protocol participants.[1][2]

Protocol optimizations

Database research has been done on ways to get most of the benefits of the two-phase commit protocol while reducing costs by *protocol optimizations*[1][2][3] and protocol operations saving under certain system's behavior assumptions.

Presumed Abort and Presumed Commit

Presumed abort or *Presumed commit* are common such optimizations.[2][3][5] An assumption about the outcome of transactions, either commit, or abort, can save both messages and logging operations by the participants during the 2PC protocol's execution. For example, when presumed abort, if during system recovery from failure no logged evidence for commit of some transaction is found by the recovery procedure, then it assumes that the transaction has been aborted, and acts accordingly. This means that it does not matter if aborts are logged at all, and such logging can be saved under this assumption. Typically a penalty of additional operations is paid during recovery from failure, depending on optimization type. Thus the best variant of optimization, if any, is chosen according to failure and transaction outcome statistics.

Tree two-phase commit protocol

The **Tree 2PC protocol**[2] (also called *Nested 2PC*, or *Recursive 2PC*) is a common variant of 2PC in a computer network, which better utilizes the underlying communication infrastructure. The participants in a distributed transaction are typically invoked in an order which defines a tree structure, the *invocation tree*, where the participants are the nodes and the edges are the invocations (communication links). The same tree is commonly utilized to complete the transaction by a 2PC protocol, but also another communication tree can be utilized for this, in principle. In a tree 2PC the coordinator is considered the root ("top") of a communication tree (inverted tree), while the participants are the other nodes. The coordinator can be the node that originated the transaction (invoked recursively (transitively) the other participants), but also another node in the same tree can take the coordinator role instead. 2PC messages from the coordinator are propagated "down" the tree, while messages to the coordinator are "collected" by a participant from all the participants below it, before it sends the appropriate message "up" the tree (except an **abort** message, which is propagated "up" immediately upon receiving it or if the current participant initiates the abort).

The **Dynamic two-phase commit** (Dynamic two-phase commitment, D2PC) **protocol**[2][6] is a variant of Tree 2PC with no predetermined coordinator. It subsumes several optimizations that have been proposed earlier. **Agreement** messages (**Yes** votes) start to propagate from all the leaves, each leaf when completing its tasks on behalf of the transaction (becoming *ready*). An intermediate (non leaf) node sends *ready* when an **agreement** message to the last (single) neighboring node from which **agreement** message has not yet been received. The coordinator is determined dynamically by racing **agreement** messages over the transaction tree, at the place where they collide. They collide either at a transaction tree node, to be the coordinator, or on a tree edge. In the latter case one of the two edge's nodes is elected as a coordinator (any node). D2PC is time optimal (among all the instances of a specific transaction tree, and any specific Tree 2PC protocol implementation; all instances have the same tree; each instance has a different node as coordinator): By choosing an optimal coordinator D2PC commits both the coordinator and each participant in minimum possible time, allowing the earliest possible release of locked resources in each transaction participant (tree node).

See also

- Atomic commit
- Commit (data management)
- Three-phase commit protocol
- XA
- Paxos algorithm
- Two Generals' Problem

References

1. Philip A. Bernstein, Vassos Hadzilacos, Nathan Goodman (1987): *Concurrency Control and Recovery in Database Systems* (http://research.microsoft.com/en-us/people/philbe/ccontrol.aspx), Chapter 7, Addison Wesley Publishing Company, ISBN 0-201-10715-5
2. Gerhard Weikum, Gottfried Vossen (2001): *Transactional Information Systems* (http://www.elsevier.com/wps/find/bookdescription.cws_home/677937/description#description), Chapter 19, Elsevier, ISBN 1-55860-508-8
3. Philip A. Bernstein, Eric Newcomer (2009): *Principles of Transaction Processing*, 2nd Edition (http://www.elsevierdirect.com/product.jsp?isbn=9781558606234), Chapter 8, Morgan Kaufmann (Elsevier), ISBN 978-1-55860-623-4
4. C. Mohan, Bruce Lindsay and R. Obermarck (1986): "Transaction management in the R* distributed database management system" (http://dl.acm.org/citation.cfm?id=7266),*ACM Transactions on Database Systems (TODS)*, Volume 11 Issue 4, Dec. 1986, Pages 378 - 396
5. C. Mohan, Bruce Lindsay (1985): "Efficient commit protocols for the tree of processes model of distributed transactions" (http://portal.acm.org/citation.cfm?id=850772),*ACM SIGOPS Operating Systems Review*, 19(2),pp. 40-52 (April 1985)
6. Yoav Raz (1995): "The Dynamic Two Phase Commitment (D2PC) protocol " (http://www.springerlink.com/content/pv12p828kk616258/),*Database Theory — ICDT '95*, *Lecture Notes in Computer Science*, Volume 893/1995, pp. 162-176, Springer, ISBN 978-3-540-58907-5

External links

- Two Phase Commit protocol explained in Pictures (http://exploredatabase.blogspot.in/2014/07/two-phase-commit-protocol-in-pictures.html) by exploreDatabase

---