DEFINITION

two-phase commit (2PC)

Posted by: Margaret Rouse WhatIs.com

Contributor(s): Crystal Bedell

Two-phase commit (2PC) is a standardized protocol that ensures atomicity, consistency, isolation and durability (ACID) of a transaction; it is an atomic commitment protocol for distributed systems.

In a distributed system, transactions involve altering data on multiple databases or resource managers, causing the processing to be more complicated since the database has to coordinate the committing or rolling back of changes in a transaction as a self-contained unit; either the entire transaction commits or the entire transaction rolls back.

A transaction manager uses 2PC to ensure data integrity as well as the integrity of the global database -- the collection of databases participating in the transaction -- as well as monitor the commitment or rollback of the distributed transactions. This protocol is entirely transparent and requires no programming by the user or application developer.

How does two-phase commit work?

In order for a distributed transaction to take place, a special object, known as a coordinator, is required. The coordinator is in charge or arranging activities and synchronizations between distributed servers.

As the name implies, 2PC consists of two phases:

Phase 1 (the prepare phase) - The protocol ensures all resource managers have saved the transaction’s updates to stable storage. Every server that is required to commit writes its data records in a log. If a server is unsuccessful in doing so, then it responds with a failure message; if it is successful, then it sends an OK message.

In this first phase, the initiating node requests all other participating nodes to promise to either commit or roll back the transaction.

There are three types of responses that the responding node can send back:

- Prepared - A prepared response is given when data in the node has been revised by a statement in the distributed transaction and the node has successfully composed itself for commitment or rollback. The prepared response also ensures that locks held for the transaction can survive a failure.
- Read-only - A read-only response means that data on the node has been queried, but it cannot be modified. Therefore, no preparation is necessary.
- Abort - An abort response indicates that the node cannot successfully prepare itself for commitment.

In order for the prepare phase to reach completion and one of the three messages to be sent, each node, except for the commit point site, must perform several steps. First, the node must request that the following referenced nodes are ready to commit. Then the node checks if the transaction changes data on itself or the subsequent nodes. If the data does not change, then the node skips the rest of the steps and replies with the read-only response.

If the data does change, then the node assigns the resources it needs to commit the transaction. The node will save redo records matching the changes made by the transaction to its redo log. A lock is then placed on the modified tables to prevent them from being read.

Next, the node ensures that locks held for the transaction can survive a failure. If all steps go according to plan, then the node issues a prepared response. However, if the attempts of the node, or one of its subsequent nodes, are unsuccessful in preparing to commit, then it issues the abort response.

Prepared nodes then wait for either a commit or rollback response from the global coordinator. The prepared nodes are considered to be in-doubt until all changes are either committed or rolled back.

Phase 2 (the commit phase) - If phase one is successful and all participants send an OK response, then phase two tells all resource managers to commit. After committing, each node logs its commit in a record and sends the coordinator a message indicating that its commit was successful. If phase one fails, then phase two tells the resource managers to abort, all servers roll back and each node sends feedback that the rollback has been successfully accomplished.

The commit phase can be broken down into the following steps:

- The global coordinator prompts the commit point site to commit and the action is performed.
- The commit point site records its commitment and sends a response back to the global coordinator, informing that it has successfully committed.
- The global and local coordinators instruct all other nodes to commit to the transaction.
- Each node's database releases its locks and commits its local portion of the distributed transaction.
- Each node's database registers an additional redo entry in its local log to show that is has committed the transaction.
- All participating nodes alert the global coordinator to the status of their successful commitment.

Once the commit phase is complete, all nodes in the distributed system possess consistent data.

In a distributed system, databases can independently fail and recover. As a result, it’s possible for a transaction to successfully commit its updates on one database system, but not on another due to a system failure. When the failed database recovers, it must be able to commit the transaction. To do so, the system must have a copy of the transaction’s updates that were executed there. However, when a system fails, it can lose the contents of its main memory. The database must therefore store a copy of the transaction’s updates before a failure occurs. 2PC ensures that each system accessed by a transaction durably stores its portion of the transaction’s updates before the transaction commits anywhere.

2PC is usually implemented by a transaction manager. The transaction manager tracks which resource managers are accessed by each transaction and runs the 2PC protocol.

Two-phase commit vs. Saga

Sagas and 2PC have the same goal: to coordinate resources while overlaying operations form a coherent unit of work. As a result, both protocols will produce a consistent system state at the end. However, the two protocols utilize different approaches to reach this goal. Specifically, Saga uses units of work that can be unfinished; a commitment protocol is not included.

The Saga pattern is a sequence of local transactions in which each transaction modifies data within a single service. Unlike 2PC, which waits for all nodes to be ready to commit or rollback before performing the action, Sagas individually respond to an external request matching the system operation and then triggers each subsequent step with the completion of the proceeding one.

Also, changes made by Saga operations are immediately visible to the outside world. This is because Saga instantly commits resource-located transactions after each step in the business process ends. On the other hand, 2PC resource-located transactions carry through just about the whole global transaction lifetime.

As a result, 2PC allows programmers to commit the entire transaction in one request with this request spanning over various systems and networks. If each participating system and network abides by the protocol, then the entire transaction can easily commit or rollback.

Saga allows programmers to split the transaction into multiple steps, allowing the protocol to span extensive periods or time, but not necessarily over systems and networks. Consequently, 2PC is used for more immediate transactions while Saga is utilized in long running transactions.

2PC is also easier for the application programmer to use since the responsibility of managing all the transaction troubleshooting falls on the transaction manager. This means the programmer only has to be concerned with their business logic -- such as inserting data into the database or sending a message to the queue. Sagas require an extra step for the programmer because the protocol requires a compensating action to be created and defined for each specific Saga pattern.

Margaret Rouse asks:

What types of database management systems does your organization use, and for what purposes?

Join the Discussion