UNIT I DISTRIBUTED DATABASES 9

Distributed Systems – Introduction – Architecture – Distributed Database Concepts – Distributed Data Storage – Distributed Transactions – Commit Protocols – Concurrency Control – Distributed Query Processing

---

DISTRIBUTED TRANSACTION

A distributed transaction is a set of operations on data that is performed across two or more data repositories (especially databases). It is typically coordinated across separate nodes connected by a network, but may also span multiple databases on a single server.

There are two possible outcomes:

1) all operations successfully complete, or

2) none of the operations are performed at all due to a failure somewhere in the system.

In the latter case, if some work was completed prior to the failure, that work will be reversed to ensure no net work was done. This type of operation is in compliance with the "ACID" (atomicity-consistency-isolation-durability) principles of databases that ensure data integrity. ACID is most commonly associated with transactions on a single database server, but distributed transactions extend that guarantee across multiple databases.

The operation known as a "two-phase commit" (2PC) is a form of a distributed transaction. "XA transactions" are transactions using the XA protocol, which is one implementation of a two-phase commit operation. A distributed transaction spans multiple databases and guarantees data integrity.

How Do Distributed Transactions Work?

- Distributed transactions have the same processing completion requirements as regular database transactions, but they must be managed across multiple resources, making them more challenging to implement for database developers. The multiple resources add more points of failure, such as the separate software systems that run the resources (e.g., the database software), the extra hardware servers, and network failures. This makes distributed transactions susceptible to failures, which is why safeguards must be put in place to retain data integrity.
- For a distributed transaction to occur, transaction managers coordinate the resources (either multiple databases or multiple nodes of a single database). The transaction manager can be one of the data repositories that will be updated as part of the transaction, or it can be a completely independent separate resource that is only responsible for coordination. The transaction manager decides whether to commit a successful transaction or rollback an unsuccessful transaction, the latter of which leaves the database unchanged.

- First, an application requests the distributed transaction to the transaction manager. The transaction manager then branches to each resource, which will have its own "resource manager" to help it participate in distributed transactions. Distributed transactions are often done in two phases to safeguard against partial updates that might occur when a failure is encountered. The first phase involves acknowledging intent to commit, or a "prepare-to-commit" phase. After all resources are acknowledged, they are then asked to run a final commit, and then the transaction is completed.

COMMIT PROTOCOLS

In a local database system, for committing a transaction, the transaction manager has to only convey the decision to commit to the recovery manager. However, in a distributed system, the transaction manager should convey the decision to commit to all the servers in the various sites where the transaction is being executed and uniformly enforce the decision. When processing is complete at each site, it reaches the partially committed transaction state and waits for all other transactions to reach their partially committed states. When it receives the message that all the sites are ready to commit, it starts to commit. In a distributed system, either all sites commit or none of them does.

The different distributed commit protocols are −

- One-phase commit
- Two-phase commit
- Three-phase commit

Distributed One-phase Commit

Distributed one-phase commit is the simplest commit protocol. Let us consider that there is a controlling site and a number of slave sites where the transaction is being executed. The steps in distributed commit are −

- After each slave has locally completed its transaction, it sends a "DONE" message to the controlling site.
- The slaves wait for "Commit" or "Abort" messages from the controlling site. This waiting time is called a window of vulnerability.
- When the controlling site receives a "DONE" message from each slave, it makes a decision to commit or abort. This is called the commit point. Then, it sends this message to all the slaves.
- On receiving this message, a slave either commits or aborts and then sends an acknowledgement message to the controlling site.

Distributed Two-phase Commit

Distributed two-phase commit reduces the vulnerability of one-phase commit protocols. The steps performed in the two phases are as follows −

Phase 1: Prepare Phase

- After each slave has locally completed its transaction, it sends a “DONE” message to the controlling site. When the controlling site has received a “DONE” message from all slaves, it sends a “Prepare” message to the slaves.
- The slaves vote on whether they still want to commit or not. If a slave wants to commit, it sends a “Ready” message.
- A slave that does not want to commit sends a “Not Ready” message. This may happen when the slave has conflicting concurrent transactions or there is a timeout.

Phase 2: Commit/Abort Phase

- After the controlling site has received “Ready” message from all the slaves − o The controlling site sends a “Global Commit” message to the slaves. o The slaves apply the transaction and send a “Commit ACK” message to the controlling site.
- When the controlling site receives “Commit ACK” message from all the slaves, it considers the transaction as committed.
- After the controlling site has received the first “Not Ready” message from any slave
- The controlling site sends a “Global Abort” message to the slaves. o The slaves abort the transaction and send a “Abort ACK” message to the controlling site.
- When the controlling site receives “Abort ACK” message from all the slaves, it considers the transaction has aborted.

Distributed Three-phase Commit

The steps in distributed three-phase commit are as follows −

Phase 1: Prepare Phase

- The steps are same as in distributed two-phase commit.

Phase 2: Prepare to Commit Phase

- The controlling site issues an “Enter Prepared State” broadcast message. The slave sites vote “OK” in response.

Phase 3: Commit / Abort Phase

- The steps are same as two-phase commit except that “Commit ACK”/”Abort ACK” message is not required.

CONCURRENCY CONTROL

Concurrency control in distributed systems is achieved by a program which is called scheduler. Schedulers help to order the operations of transactions in such a way that the resulting logs are serializable. There are two types of the concurrency control that are locking approach and non-locking approach.

Various Approaches For Concurrency Control.

1. Locking Based Concurrency Control Protocols

Locking-based concurrency control protocols use the concept of locking data items. A lock is a variable associated with a data item that determines whether read/write operations can be performed on that data item. Generally, a lock compatibility matrix is used which states whether a data item can be locked by two transactions at the same time. Locking-based concurrency control systems can use either one-phase or two-phase locking protocols.

1. **One-phase Locking Protocol:** In this method, each transaction locks an item before use and releases the lock as soon as it has finished using it. This locking method provides for maximum concurrency but does not always enforce serializability.
2. **Two-phase Locking Protocol**: In this method, all locking operations precede the first lock-release or unlock operation. The transaction comprise of two phases. In the first phase, a transaction only acquires all the locks it needs and do not release any lock. This is called the expanding or the **growing phase**. In the second phase, the transaction releases the locks and cannot request any new locks. This is called the **shrinking phase**.

Every transaction that follows a two-phase locking protocol is guaranteed to be serializable. However, this approach provides low parallelism between two conflicting transactions.

2. Timestamp Concurrency Control Algorithms:

Timestamp-based concurrency control algorithms use a transaction's timestamp to coordinate concurrent access to a data item to ensure serializability. A timestamp is a unique identifier given by DBMS to a transaction that represents the transaction's start time.

These algorithms ensure that transactions are committed in the order dictated by their timestamps. An older transaction should commit before a younger transaction, since the older transaction enters the system before the younger one.

Timestamp-based concurrency control techniques generate serializable schedules such that the equivalent serial schedule is arranged in order of the age of the participating transactions.

Optimistic Concurrency Control Algorithm:

In systems with low conflict rates, the task of validating every transaction for serializability may lower performance. In these cases, the test for serializability is postponed to just before commit. Since the conflict rate is low, the probability of aborting transactions which are not serializable is also low. This approach is called optimistic concurrency control technique.

In this approach, a transaction's life cycle is divided into the following three phases –

- Execution Phase − A transaction fetches data items to memory and performs operations upon them.
- Validation Phase − A transaction performs checks to ensure that committing its changes to the database passes serializability test.
- Commit Phase − A transaction writes back modified data item in memory to the disk.

QUERY PROCESSING IN DISTRIBUTED DBMS

A Query processing in a distributed database management system requires the transmission of data between the computers in a network. A distribution strategy for a query is the ordering of data transmissions and local data processing in a database system.

Distributed query processing is the procedure of answering queries (which means mainly read operations on large data sets) in a distributed environment where data is managed at multiple sites in a computer network.

Query processing involves the transformation of a high-level query (e.g., formulated in SQL) into a query execution plan (consisting of lower-level query operators in some variation of relational algebra) as well as the execution of this plan. The goal of the transformation is to produce a plan which is equivalent to the original query (returning the same result) and efficient, i.e., to minimize resource consumption like total costs or response time.

1. Costs (Transfer of data) of Distributed Query processing:

In Distributed Query processing, the data transfer cost of distributed query processing means the cost of transferring intermediate files to other sites for processing and therefore the cost of transferring the ultimate result files to the location where that results are required.

Commonly, the data transfer cost is calculated in terms of the size of the messages. By using the below formula, we can calculate the data transfer cost:

Data transfer cost = C * Size

C refers to the cost per byte of data transferring and Size is the no. of bytes transmitted.