UNIT I DISTRIBUTED DATABASES 9

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

---

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

A lock is a variable associated with a data item that determines whether read/write operations can be performed on that data item. 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**: The transaction comprises two phases. In the first phase, a transaction only obtains all the locks it needs and does 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.

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.

3. 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 tests.
- Commit Phase − A transaction writes back modified data items in memory to the disk.

Problems arise in a distributed DBMS environment for concurrency control and recovery purposes that are not encountered in a centralized DBMS environment. These include the following:

- **Dealing with multiple copies of the data items**. The concurrency control method is responsible for maintaining consistency among these copies. The recovery method is responsible for making a copy consistent with other copies of the site on which fails and recovers later.
- **Failure of individual sites**. The DDBMS should continue to operate with its running sites, even when one or more individual sites fail. When a site recovers, its local database must be updated the same as the rest of the sites.
- **Failure of communication links**. The system must be able to deal with the failure of one or more communication links that connect the sites. An extreme case of this problem is that network partitioning may occur. This breaks up the sites into two or more partitions, where the sites within each partition can communicate only with one another and not with sites in other partitions.
- **Distributed commit**. Problems can arise with committing a transaction that is accessing databases stored on multiple sites if some sites fail during the commit process. The two-phase commit protocol is often used to deal with this problem.
- **Distributed deadlock**. Deadlock may occur among several sites, so techniques for dealing with deadlocks must be extended to take this into account.

Techniques to deal with recovery and concurrency control in DDBMSs:

1. Distributed Concurrency Control Based on a Distinguished Copy of a Data Item

The idea is to designate a particular copy of each data item as a distinguished copy. The locks for this data item are linked with the distinguished copy, and the locking and unlocking requests are sent to the site that contains that copy. The distinguished copies are chosen based on four methods. They are,

1. Primary Site Technique
2. Primary Site with Backup Site
3. Primary Copy Technique
4. Choosing a New Coordinator Site in Case of Failure

1. Primary Site Technique:

In this method, a single primary site is designated to be the coordinator site for all database items. All locks are kept at that site, and all requests for locking or unlocking are sent there. This method is thus an extension of the centralized locking approach. If all transactions follow the two-phase locking protocol, serializability is guaranteed.

The **advantage** is - it is a simple extension of the centralized approach and thus is not much complex.

Disadvantages are

- All locking requests are sent to a single site, which leads to overloading that site and causing a system bottleneck.
- Failure of the primary site paralyzes the system, since all locking information is kept at that site.

This can limit system reliability and availability. All locks are accessed at the primary site, the items can be accessed at any site at which they reside.

If a transaction obtains a Read_lock on a data item from the primary site, it can access any copy of that data item.

If a transaction obtains a Write_lock and updates a data item, the DDBMS is responsible for updating all copies of the data item before releasing the lock.

2. Primary Site with Backup Site:

This approach designates a second site to be a backup site. All locking information is maintained at both the primary and the backup sites. In case of primary site failure, the backup site takes over as the primary site, and a new backup site is chosen and the lock status information is copied to that site.

It simplifies the process of recovery of the primary site. It slows down the process of obtaining locks, because all lock requests and granting of locks must be recorded at both the primary and the backup sites before a response is sent to the requesting transaction. The primary and backup sites become overloaded with requests and slow down the system.

3. Primary Copy Technique:

This method attempts to distribute the load of lock coordination among various sites by having the distinguished copies of different data items stored at different sites. Failure of one site affects the transactions that are accessing locks on items whose primary copies reside at that site, but other transactions are not affected. This method can also use backup sites to enhance reliability and availability.

4. Choosing a New Coordinator Site in Case of Failure:

Whenever a coordinator site fails, the sites that are still running must choose a new coordinator. In the case of the primary site approach with no backup site, all executing transactions must be aborted and restarted in a recovery process. The recovery process involves choosing a new primary site and creating a lock manager process and a record of all lock information at that site.

For methods that use backup sites, transaction processing is suspended while the backup site is designated as the new primary site and a new backup site is chosen and the lock status information is copied to that site. If a backup site X is about to become the new primary site, X can choose the new backup site from among the system's running sites. However, if no backup site existed, or if both the primary and the backup sites are down, a process called election can be used to choose the new coordinator site.

In this process, any site Y that attempts to communicate with the coordinator site repeatedly and fails to do so, can assume that the coordinator is down and can start the election process by sending a message to all running sites proposing that Y become the new coordinator. As soon as Y receives a majority of yes votes, Y can declare that it is the new coordinator.

2. Distributed Concurrency Control Based on Voting

In the voting method, there is no distinguished copy; rather, a lock request is sent to all sites that includes a copy of the data item. Each copy maintains its own lock and can grant or deny the request for it. If a transaction that requests a lock is granted by a majority of the copies, it holds the lock and informs all copies that it has been granted the lock. If a transaction does not receive a majority of votes granting it a lock within a certain time-out period, it cancels its request and informs all sites of the cancellation.

The voting method is considered a truly distributed concurrency control method, since the responsibility for a decision resides with all the sites involved.

3. Distributed Recovery

In some cases it is difficult even to determine whether a site is down without exchanging numerous messages with other sites. For example, suppose that site X sends a message to site Y and expects a response from Y but does not receive it. There are several possible explanations:

- The message was not delivered to Y because of communication failure.

- Site Y is down and could not respond.
- Site Y is running and sent a response, but the response was not delivered.

Without additional information or the sending of additional messages, it is difficult to determine what actually happened.

Another problem with distributed recovery is distributed commit. When a transaction is updating data at several sites, it cannot commit until it is sure that the effect of the transaction on every site cannot be lost. This means that every site must first have recorded the local effects of the transactions permanently in the local site log on disk. The two-phase commit protocol is often used to ensure the correctness of distributed commits.

QUERY PROCESSING

A distributed database query is processed in stages as follows:

1. **Query Mapping:**
    - The input query on distributed data is specified using a query language.
    - It is then translated into an algebraic query on global relations.
    - This translation is referred to as a global conceptual schema. Hence, this translation is mostly identical to the one performed in a centralized DBMS.
    - It is first normalized, analyzed for semantic errors, simplified, and finally restructured into an algebraic query.
2. **Localization:**
    - In a distributed database, fragmentation results in fragments or relations being stored in separate sites, with some fragments replicated.
    - This stage maps the distributed query on the global schema to separate queries on individual fragments using data distribution and replication information.
3. **Global Query Optimization.**
    - Optimization consists of selecting a strategy from a list of candidates that is closest to optimal.
    - A list of candidate queries can be obtained by permuting the ordering of operations within a fragment query generated by the previous stage.
    - Time is the preferred unit for measuring cost.
    - The total cost is a weighted combination of costs such as CPU cost, I/O costs, and communication costs.
4. **Local Query Optimization.**
    - This stage is common to all sites in the DDB.
    - The techniques are similar to those used in centralized systems.

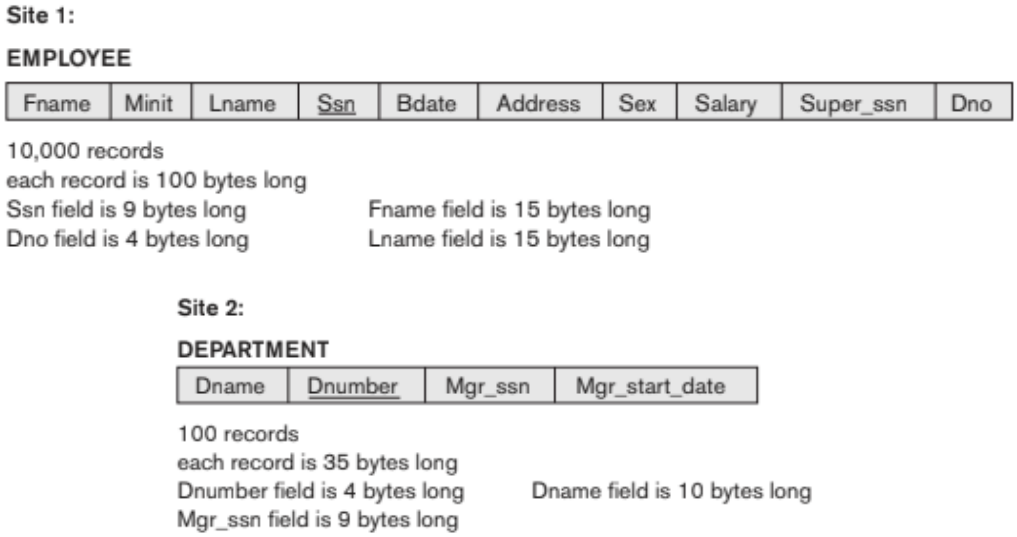
- The first three stages discussed above are performed at a central control site, whereas the last stage is performed locally.

Data Transfer Costs of Distributed Query Processing

In a distributed system, the complicating factors in query processing are,

- **The cost of transferring data over the network**.
- **The goal of reducing the amount of data transfer**

The EMPLOYEE and DEPARTMENT relations are distributed at two sites as shown in Figure.



We will assume in this example that neither relation is fragmented. According to Figure, the size of the EMPLOYEE relation is 100 * 10,000 = 106 bytes, and the size of the DEPARTMENT relation is 35 * 100 = 3,500 bytes.

Consider the query Q: For each employee, retrieve the employee name and the name of the department for which the employee works. This can be stated as follows in the relational algebra:

Q: $\pi_{Fname,Lname,Dname}$**(EMPLOYEE** $\bowtie_{Dno=Dnumber}$ **DEPARTMENT)**

The result of this query will include 10,000 records. The query is submitted at a distinct site 3. There are three simple strategies for executing this distributed query:

1. Transfer both the EMPLOYEE and the DEPARTMENT relations to the result site, and perform the join at site 3. In this case, a total of 1,000,000 + 3,500 = 1,003,500 bytes must be transferred.
2. Transfer the EMPLOYEE relation to site 2, execute the join at site 2, and send the result to site 3. The size of the query result is 40 * 10,000 = 400,000 bytes, so 400,000 + 1,000,000 = 1,400,000 bytes must be transferred.

3. Transfer the DEPARTMENT relation to site 1, execute the join at site 1, and send the result to site 3. In this case, 400,000 + 3,500 = 403,500 bytes must be transferred.

If minimizing the amount of data transfer is the optimization criterion, we should choose strategy 3.

Now consider another query Q′: For each department, retrieve the department name and the name of the department manager. This can be stated as follows in the relational algebra:

Q′: $\pi_{Fname,Lname,Dname}$(DEPARTMENT$\bowtie_{Mgr_ssn=Ssn}$ EMPLOYEE)

Again, suppose that the query is submitted at site 3. The same three strategies for executing query Q apply to Q′, except that the result of Q′ includes only 100 records, assuming that each department has a manager:

1. Transfer both the EMPLOYEE and the DEPARTMENT relations to the result site, and perform the join at site 3. In this case, a total of 1,000,000 + 3,500 = 1,003,500 bytes must be transferred.
2. Transfer the EMPLOYEE relation to site 2, execute the join at site 2, and send the result to site 3. The size of the query result is 40 * 100 = 4,000 bytes, so 4,000 + 1,000,000 = 1,004,000 bytes must be transferred.
3. Transfer the DEPARTMENT relation to site 1, execute the join at site 1, and send the result to site 3. In this case, 4,000 + 3,500 = 7,500 bytes must be transferred.

Again, we would choose strategy 3—this time by an overwhelming margin over strategies 1 and 2. The preceding three strategies are the most obvious ones for the case where the result site (site 3) is different from all the sites that contain files involved in the query (sites 1 and 2). However, suppose that the result site is site 2; then we have two simple strategies:

1. Transfer the EMPLOYEE relation to site 2, execute the query, and present the result to the user at site 2. Here, the same number of bytes—1,000,000—must be transferred for both Q and Q′.
2. Transfer the DEPARTMENT relation to site 1, execute the query at site 1, and send the result back to site 2. In this case 400,000 + 3,500 = 403,500 bytes must be transferred for Q and 4,000 + 3,500 = 7,500 bytes for Q′.

A more complex strategy, which sometimes works better than these simple strategies, uses an operation called semijoin.

Distributed Query Processing Using Semijoin

- Distributed query processing uses the semijoin operation to reduce the number of tuples in a relation before transferring it to another site.

- Joining is done by sending the column of one relation R to the site where the other relation S is located.
- The join attributes and the attributes required in the result, are projected out and shipped back to the original site and joined with R.
- Hence, only the joining column of R is transferred in one direction, and a subset of S with no irrelevant tuples or attributes is transferred in the other direction. This can be an efficient solution to minimizing data transfer.

Consider the following strategy for executing Q or Q′:

1. **Project the join attributes** of DEPARTMENT at site 2, and transfer them to site. For Q, we transfer $\mathbf{F = \pi_{Dnumber}(DEPARTMENT)}$, whose size is 4 * 100 = 400 bytes, whereas for Q′, we transfer $\mathbf{F' = \pi_{Mgr\_ssn}(DEPARTMENT)}$, whose size is 9 * 100 = 900 bytes.
2. **Join the transferred file** with the EMPLOYEE relation at site 1, and transfer the required attributes from the resulting file to site 2. For Q, we transfer
   $\mathbf{R = \pi_{Dno,\ Fname,\ Lname}(F \ltimes_{Dnumber=Dno} EMPLOYEE)}$,
   whose size is 34 * 10,000 = 340,000 bytes, whereas for Q′, we transfer
   $\mathbf{R' = \pi_{Mgr\_ssn,\ Fname,\ Lname}(F' \ltimes_{Mgr\_ssn=Ssn} EMPLOYEE)}$,
   whose size is 39 * 100 = 3,900 bytes.
3. **Execute the query** by joining the transferred file R or R′ with DEPARTMENT, and present the result to the user at site.

Using this strategy, we transfer 340,400 (340,000 + 400) bytes for Q and 4,800 (3,900 + 900) bytes for Q′.

A semijoin operation $\mathbf{R \ltimes_{A=B} S}$, where A and B are domain-compatible attributes of R and S, respectively, produces the same result as the relational algebra expression $\mathbf{\pi R(R \ltimes_{A=B} S)}$.

In a distributed environment where R and S reside at different sites, the semijoin is typically implemented by first transferring $\mathbf{F = \pi_B(S)}$ to the site where R resides and then joining F with R, thus leading to the strategy discussed here. The semijoin operation is not commutative.
