Graph based protocol

Graph-Based Protocol for Concurrency Control

The **Graph-Based Protocol** is a type of concurrency control mechanism that ensures serializability by constructing and enforcing a **directed acyclic graph (DAG)** of transactions and the data items they access. This protocol assigns a **partial order** to the data items and ensures that transactions access these items in that order, thus avoiding cycles (and preventing deadlocks).

Time stamp based

Timestamp-Based Concurrency Control (TCC) is a mechanism that ensures serializability by assigning a unique **timestamp** to each transaction when it starts. Transactions are then ordered based on their timestamps. The goal is to enforce that transactions execute in the same order as their timestamps to ensure serializability.

In TCC, two main rules govern how transactions interact with data:

1. **Read Rule**:
    - A transaction can **read** a data item only if no other transaction that has written the item has a **later** timestamp than the transaction's timestamp.
2. **Write Rule**:
    - A transaction can **write** a data item only if no other transaction has already **read** or **written** the data item, and has a **later** timestamp than the current transaction.

If a transaction violates these rules, it is aborted to maintain serializability.

Multiversion Concurrency control

① Based on Time stamp.

Key Concepts of MVTO

1. **Multiple Versions of Data Items**:
   - Each data item has multiple versions, each associated with a **timestamp** (when the transaction wrote that version).
2. **Read Rule**:
   - A transaction reads the **most recent version** of a data item that has a timestamp less than or equal to the transaction's timestamp.
3. **Write Rule**:
   - A transaction writes a new version of the data item only if there is no transaction that has **already read** or **written** the item with a later timestamp.
4. **Versioning**:
   - Each time a transaction writes to a data item, a new version of the data item is created.
   - Each version has a **write timestamp** and a **read timestamp**.
5. **Commit and Abort**:
   - Transactions that violate the MVTO rules are **aborted** and may need to be retried.

② Based on 2 PL

Key Concepts of MV2PL

1. **Two-Phase Locking (2PL)**:
   - Transactions must obtain locks on the data items before accessing them. The transaction is in the **growing phase** when it is acquiring locks and in the **shrinking phase** when it releases them. The transaction must release locks only after it has acquired all the necessary ones.
2. **Multiversioning**:
   - Each data item can have multiple versions, and the versions are stored with timestamps or transaction identifiers to identify when they were created. The system can maintain multiple versions of a data item to provide **read consistency** for transactions that access data concurrently.
3. **Read and Write Rules**:
   - **Read Rule**: A transaction can read the **latest valid version** of a data item that it is authorized to access (this is determined by timestamps or transaction IDs).
   - **Write Rule**: A transaction can only write to a data item if no other transaction has modified that data item in the current transaction phase.
4. **Locking Mechanism**:
   - **Shared Locks**: Used for read access, allowing other transactions to read the same data item but not modify it.
   - **Exclusive Locks**: Used for write access, preventing any other transaction from accessing the item for reading or writing.

[Optimistic]

(1) Validation based

Key Concepts in Optimistic Validation-Based Concurrency Control

1. **Transaction Phases**:
   - **Read Phase**: Transactions read data without locking it.
   - **Validation Phase**: Transactions are validated to ensure they do not conflict with other concurrent transactions.
   - **Write Phase**: If validation succeeds, the transaction writes its changes to the database. If validation fails, the transaction is aborted.
2. **Validation**:
   - A transaction is validated to check for conflicts with other transactions. There are several types of validation strategies, including:
     - **Global Validation**: Checks whether the transaction conflicts with any other transaction that is attempting to modify the same data.
     - **Local Validation**: Ensures that a transaction only conflicts with other transactions it interacts with directly.
3. **Conflict Detection**:
   - Transactions are checked for conflicting reads or writes. If two transactions try to modify the same data at the same time, this is considered a conflict.
   - Conflicts can happen in **write-write**, **write-read**, or **read-write** situations.
4. **Transaction Retry**:
   - If validation fails (i.e., the transaction conflicts with others), it is aborted, and the transaction may be retried.

(2) Snapshot isolation

Key Concepts in Snapshot Isolation with Optimistic Concurrency Control (SI-OCC)

1. **Snapshot of the Database**:
    - When a transaction starts, it is given a **snapshot** of the database. This snapshot is a **consistent view** of the database at the time the transaction started, meaning that the transaction sees all data as it was at the moment of its start, even if other transactions modify the data concurrently.
2. **Optimistic Execution**:
    - Transactions execute optimistically by reading data and performing operations without acquiring locks. They continue to work as though there will be no conflicts with other transactions.
3. **Validation**:
    - Before committing, a transaction checks if any conflicts have occurred with other transactions. Specifically, it checks if any other transaction has modified data that this transaction has read or written. If there are conflicts, the transaction is **aborted** and might be retried.
4. **Conflict Detection**:
    - Conflicts are typically detected using the following rules:
        - **Write-write conflict**: If two transactions try to write to the same data item.
        - **Read-write conflict**: If one transaction reads a data item that another transaction writes to after the transaction has started.
5. **Transaction Rollback**:
    - If conflicts are detected during the validation phase, the transaction is **rolled back** and must be retried, as it is no longer guaranteed to produce a serializable result.