12. Concurrency Control Techniques - Short

Most of content here is borrowed from book Elmasri/Navathe

Concurrency Control Techniques

Objectives Concurrency Control is
to ensure "serializable" and "recoverable" schedules

Concurrency control and Recovery system go hand in hand; here, we plan to have abstract understanding of following techniques and algorithms

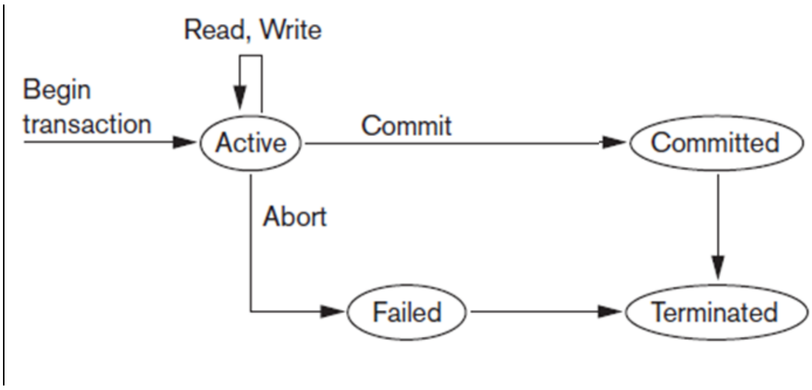
Concurrency Control Techniques (Algorithms)

- 2PL protocol
- Timestamp ordering based Protocols
- Multi-version Concurrency Control
- Snapshot Isolation

Recovery (Algorithms)

- Write Ahead Logging Protocol
- ARIES recovery algorithm

Life cycle of a transaction



Recall: Schedule, Serializable Schedule, Recoverable Schedule!

Two Phase Locking protocol (2PL)

Concept of Locking:

Before reading and writing any data item "appropriate locks" are acquired. For writing, exclusive lock is required, whereas for reading shared lock is fine. A transaction cannot proceed its execution unless it has appropriate lock for reading/writing.

Below is Lock Compatibility Matrix.

Lock Compatibility Matrix		
	Shared	e**X**lusive
If No Lock	YES	YES
If shared	YES	NO
If exclusive	NO	NO

Once transaction is done with read/write should release the locks that it acquire while progression.

Two phase locking protocol:

A transaction follows two phases of locking: first locking phase and then unlocking phase. In this strategy, a two phases - Locking phase and unlocking phase. Two phase locking protocol requires that once a transaction starts unlocking its locks, it can no more acquire newer locks.

A most commonly used variation of 2PL protocol is **Rigorous 2PL** – this does not allow a transaction to release any of lock (read/write) until it commits or aborts.

2PL helps in not having cycles in precedence graphs and hence generating "serializable schedules".

Main drawback of 2PL protocol is it can have Deadlocks.

Concurrency control based on Time-Stamp Ordering

Deadlock is the main problem with 2PL. Concurrency control with timestamp ordering doesn't use locks, therefore deadlocks cannot occur.

What is Timestamps?

- Unique identifiers generated by DBMS to identify transactions
- Also indicate start order of transactions
- Let Timestamps of transaction T, is denoted by TS(T)
- Time stamps can be generated in several ways: Starting "timestamp" of a transaction; Transaction counter, increased every time new transaction is started, and so.

Timestamp Ordering – the intuition

- Operations from multiple transactions are included in schedule in their timestamp order: this leads to making schedule equivalent a "serial schedule" as "$T_{older};T_{younger}$".

- Intuition of Time stamp based concurrency control is that conflicting operations in a schedule occur in the order of their transaction's timestamp. The time ordering algorithm ensures this "serialization rules".
- Based on this principle, and request from a participating transaction; decision for transaction is taken - if its operation is immediately included in schedule, or should wait, or no chance of inclusion, therefore it should abort, etc.

Multi-version Concurrency Control Schemes

- Basic Time Stamp ordering algorithm may leaded to many aborts.
- Multi-version TO schemes help in increasing concurrency.
- Multi-version schemes keep old versions of data items, and instead of asking a requester to wait/abort, "old copies" of data item is handed over.
- When read request comes, an "appropriate version" is returned immediately - typically version created by youngest among older (including self)
- Each successful write results in the creation of a new version of the data item written.
- Both type of MVCC algorithms are available
    - Timestamp ordering – MV Timestamp Ordering Schemes
    - Locking – Multi-version 2PL protocol

Snapshot Isolation [1]

Motivation: Decision support queries that read large amounts of data have concurrency conflicts with OLTP transactions that update a few rows - results poor performance

Snapshot Isolation is primarily a optimistic variation of Multi Version Concurrency Control (MVCC), and proposed by Berenson et al, SIGMOD 1995. Many DBMS like Oracle, PostgreSQL, SQL Server use its variations.

The technique in nutshell through a simple example

- A transaction T2 executing with Snapshot Isolation takes snapshot of committed data at start;
- A transaction always reads/modifies data in its own snapshot

T1	T2	T3
W(Y, 1) Commit		
	Start R(X) → 0 R(Y)→ 1	
		W(X,2) W(Z,3) Commit
	R(Z) → 0 R(Y) → 1 W(X, 3) **Commit**	

[1] Source: Lecture slides of Silberschatz-Korth-Sudarshan, 6th ed

- updates of concurrent transactions are not visible to T2
- writes of T2 complete when it commits
- First-committer-wins rule:
  Commits only if, no other concurrent transaction has already written data that T2 intends to write.
- In the case here T2 cannot be committed, therefore aborts.

In snapshot isolation, reading is *never* blocked, and also doesn't block other transaction activities, even updates.

Problem with Snapshot Isolation

Has a noted problem called "skew write" – explained through example below

T1: Read Y; X=Y; Write(X)

T2: Read X; Y=X; Write(Y)

Initially x = 3 and y = 17

Serial execution should: x = ?, y = ?

If both transactions start at the same time, with snapshot isolation: x = ? , y = ?

More recent variation of snapshot isolation is Serializable Snapshot Isolation (SSI) is implemented in newer versions of RDBMS, and you may not experience skew write.