15-645 Database System

Siqi Guo (siqiguo) June 24, 2024

Course Overview

In the context of the Distributed System 15-640, we want to give the illusion that we operate the file system in local machine, even if it's actually operated on the remote server.
While, here, DBMS wants to give the illusion that we are operating with the database entirely in memory.

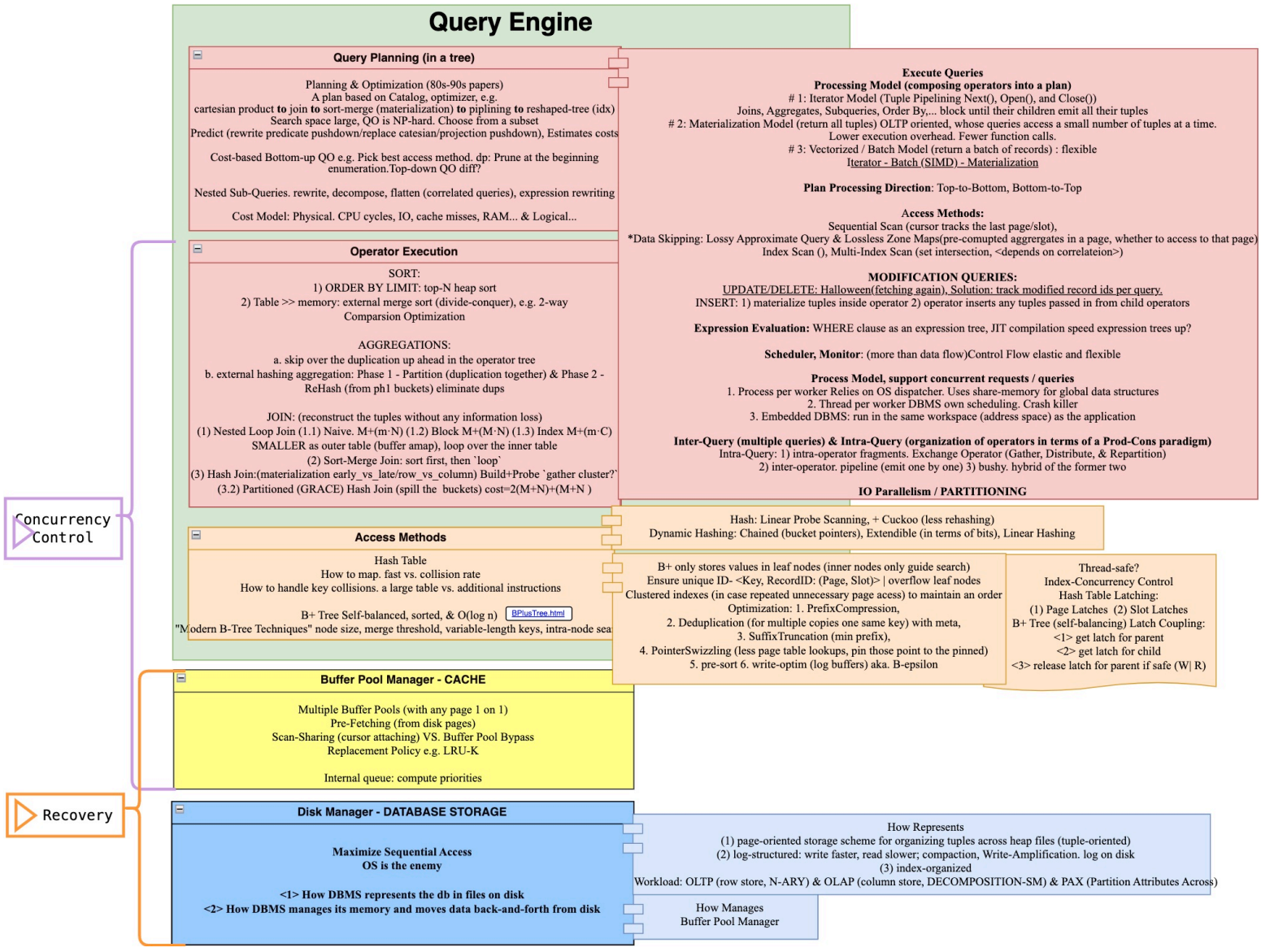

Figure 1: Course lec#1 - lec#14 Overview

The above figure shows the overview of the course from lec#01 to lec#14.

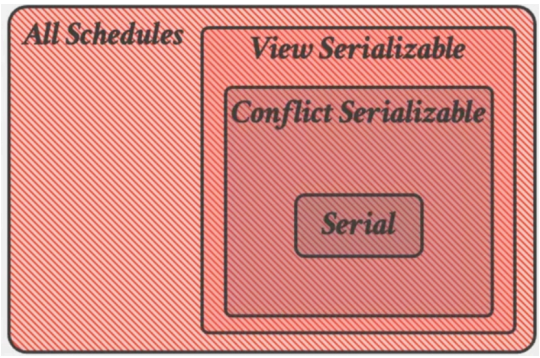

Figure 2: Serializable

Concurrency Control Theory & Recovery

lec15: ACID & **Logging** & **Shadow Paging**

lec16: **Two-Phase Locking** (Locks during entire transactions are kept in Lock Manager to protect Database Contents.)

Phase #1: Growing

→ Each txn requests the locks that it needs from the DBMS's lock manager.
→ The lock manager grants/denies lock requests.

Phase #2: Shrinking

→ The txn is allowed to only release/ downgrade locks that it previously acquired. It cannot acquire new locks.

2PL guarantees conflict serializability because it generates schedules whose precedence graph is acyclic. However, it is subject to cascading aborts (one txn holds the lock on one object which was hold by another txn, then this txn also has to be aborted once that txn was aborted.)

To solve this, **strong strict 2PL** ensure the txn releases all locks after is has ended.

Based on the DATABASE LOCK HIERARCHY, we have extra **Intention Locks** for higher parallelism (as we traverse down the hierarchy and know what we intend to do). IS, IX, SIX

lec17: **A serialization mechanism: Timestamp Ordering**

(I) Do not read stuff from the future.

(II) Can't write if a future transaction has read or written to the object.

lec17: **Optimistic Concurrency Control (OCC)**

#1 - Read Phase:
→ Track the read / write sets of txns and store their writes in a private workspace.
Read and write objects, making local copies.

#2 - Validation Phase:
→ When a txn commits, check whether it conflicts with other txns.
Check for serializable schedule-related anomalies.

if Validation (Ti < Tj)
Case 1: Ti completes its write phase before Tj starts its read phase.
Case 2: Ti completes its write phase before Tj starts its write phase.
Check: WriteSet(T_i) ∩ ReadSet (T_j) = ∅
Case 3: Ti completes its read phase right after Tj ends its read phase.
Check: WriteSet(T_i) ∩ ReadSet (T_j) = ∅ && WriteSet(T_i) ∩ WriteSet(T_j) = ∅

#3 - Write Phase:
→ If validation succeeds, apply private changes to database. Otherwise abort and restart the txn.
Propagate changes in the tn's write set to database to make them visible to other txns.

Serial Commits:
⋆ Use a global latch to limit a single txn to be in the Validation/Write phases at a time.

Parallel Commits:
⋆ Use fine-grained write latches to support parallel Validation/Write phases
⋆ Ixns acquire latches in primary key order to avoid deadlocks.

Problem:
→ OCC is not suitable for works well high # of conflicts.
→ High overhead for copying data locally.
→ Validation/ Write phase bottlenecks.

⋆ For Dynamic Databases,
(1) Re-Execute Scans, (2) Predicate Locking (rarely), and (3) Index Locking.

⋆ Unrepeatable Read vs. Phantom

lec18: **Multi-Version Concurrency Control (MVCC)**

Similar to what is included in 15640 Distributed System Project,

→ When a txn writes to an object, the DBMS creates a new version of that object.

→ When a txn reads an object, it reads the newest version that existed when the txn started.

Writers do not block readers.

Readers do not block writers.

Read-only txns do not need to acquire locks, and use timestamps to determine visibility.

lec19: **MVCC & Logging**

In addition to concurrency control mechanism like 2PL, (Timestamp Ordering) OCC, we also need the version control mechanism like MVCC in lec.18-19.

All these are for better balance between serializability and parallelism.

⋆Concurrency Control Protocol

⋆Version Storage

Approach 1: Append-Only Storage → New versions are appended to the same table space.

Approach 2: Time-Travel Storage → Old versions are copied to separate table space. Update the pointers.

Approach 3: Delta Storage → The original values of the modified attributes are copied into a separate delta record space.

⋆Garbage Collection (cleaning up the versions)

Approach 1: Tuple-level → Find old versions by examining tuples directly.

Approach 2: Transaction-level → Txns keep track of their old versions, so the DBMS does not have to scan tuples to determine visibility.

⋆Index Management

Primary Key indexes point to version chain head.

Secondary indexes: Logical Pointers or Physical Pointers. (converter: global record ID)

⋆Delete

lec19-20: **Recovery**

<I> Actions during txn processing to ensure recovery from a failure:

⋆ **Shadow Paging (Master & Shadow)**'s Buffer Pool policy:

NO-STEAL (not allowed uncommitted txn to overwrite the committed in non-volatile) +
FORCE (requires all updates made by a txn are reflected on non-volatile before the txn can commit)

⋆ **Write-Ahead Logging (WAL)**'s Buffer Pool Policy: STEAL + NO-FORCE. low-latency & high throughput

Maintain a log file separate from data files that contains the changes that txns make to database.
DBMS must write to disk the log file records that correspond to changes made to a database object before it can flush that object to disk.
A txn is not considered committed until all its log records have been written to stable storage.

Checkpointing.

<II> Actions after a failure to recover.

Here, WAL records are used (guess, e.g. ARIES), and log sequence number (LSN) tracks the log records.

Update the pageLSN every time a txn modifies a record in the page.
Update the flushedLSN in memory every time the DBMS writes the WAL buffer to disk.
ABORT needs another two pointes previousLSN to walk through faster (with time passing, a bunch of records between two continuous ones), [and optional with UndoNextLSN, to undo. (in CLR operation)]

Checkpoints: Active Transaction Table + Dirty Page Table
A point here is Fuzzy Checkpoint allows other active transactions to continue while the checkpoint is in progress.

1. Checkpoint in WAL should be a range (CHECKPOINT-BEGIN, CHECKPOINT-END).
2. Txn after the CHECKPOINT-BEGIN will not be in the ATT in this CHECKPOINT-END; LSN after CHECKPOINT-BEGIN will not be into DB Master until CHECKPOINT-END.

ARIES: Analysis, Redo, Undo.
Redo everything since the earliest dirty page.
Undo txns that never committed.
Write CLRs when undoing, to survive failures during restarts.
reference

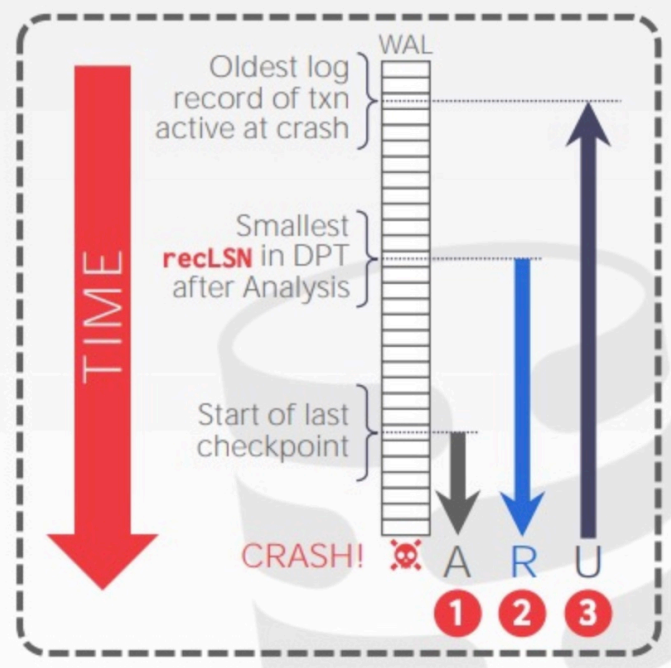

Figure 3: Serializable

lec21-END: **Distributed Databases**