15-645 Database System

Siqi Guo (siqiguo) June 23, 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.

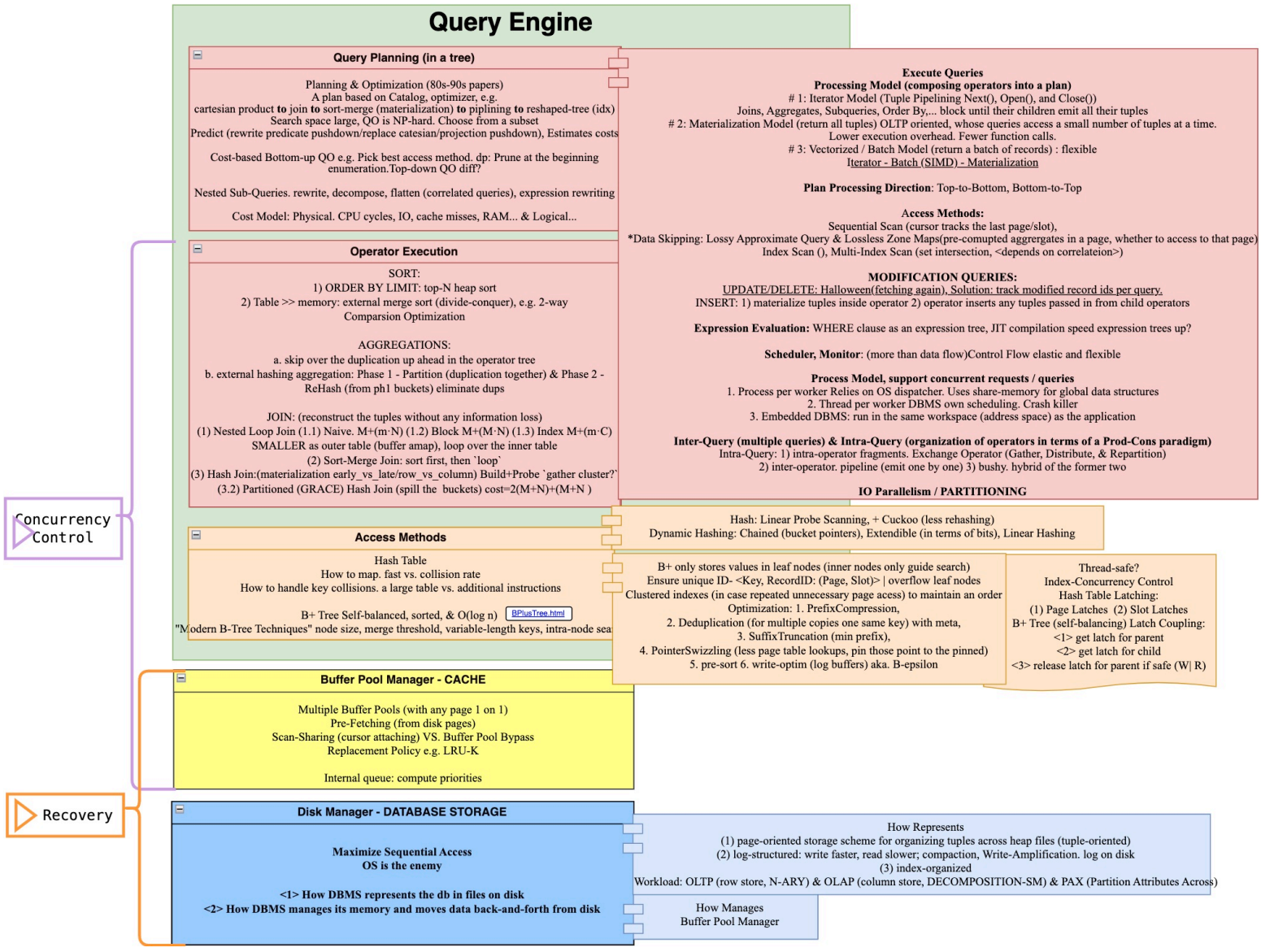

Figure 1: Course lec#1 - lec#14 Overview

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

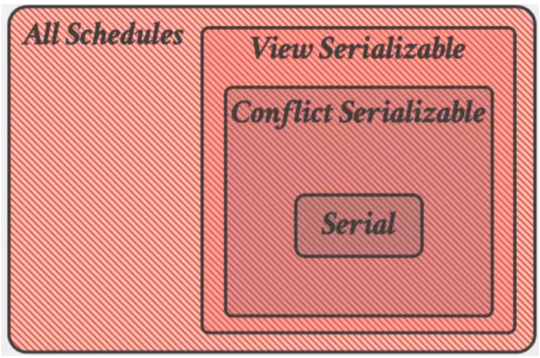

Figure 2: Serializable

Concurrency Control Theory & Recovery

lec15: **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.

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