Concurrency Control and Recovery

Phil Gibbons

15-712 F15

Lecture 20

Today's Reminders

- **Sign up for Interim Project Report slots (11/20)**

Concurrency Control and Recovery
[Computer Science & Engineering Handbook 1997]

- **Mike Franklin** (UC Berkeley)
  - ACM Fellow
  - Sigmod Test-of-Time award 2004 & 2013

ACID Properties

- **Atomicity: all-or-nothing**
- **Consistency: preserves integrity constraints**
- **Isolation: as-if running alone**
- **Durability: effects of committed transactions survive failures**

Serializability

- **Equivalent to some serial schedule (i.e., a one transaction at-a-time schedule)**
  - Conflict serializability: Preserves order of conflicting operations in non-aborting transactions
  - View serializability: Preserves semantics of operations in non-aborting transactions

$w_0[A, 100], w_1[A, 200], r_0[A, 100]$?

Is view serializable but not conflict serializable

Concurrency Control

- **Two-phase Locking: Each transaction acquires all its locks before releasing any**
  - Guarantees serializability
  - OCC not used because consumes more resources than locking
- **Deadlock avoidance/detection**
  - Avoidance: Impose order on locks
  - Detection: Timeouts or cycles in waits-for graphs
- **Isolation levels: Write locks held to commit/abort (strict)**
  - READ UNCOMMITTED: no read locks
  - READ COMMITTED: short-duration read locks
  - REPEATABLE READ: strict read locks
  - SERIALIZABLE: strict read locks on predicates

Concurrency Control (cont.)

- **Hierarchical Locking**
  - Intention Shared (IS), Intention Exclusive (IX), Shared with Intension Exclusive (SIX)
  - Automatic lock escalation

	IS	IX	S	SIX	X
IS	y	y	y	y	n
IX	y	y	n	n	n
S	y	n	y	n	n
SIX	y	n	n	n	n
X	n	n	n	n	n

Table 2: Compatibility Matrix for Regular and Intention Locks

To Get	Must Have on all Ancestors
IS or S	IS or IX
IX,SIX, or X	IX or SIX

Table 3: Hierarchical Locking Rules

Recovery

- **Failure Types**
  - Transaction Failure, System Failure, Media Failure
- **Buffer Management**
  - STEAL: uncommitted transaction can overwrite most recent committed value on non-volatile storage
    - Must be able to UNDO
  - NO-FORCE: can commit before updates in non-volatile storage
    - Must be able to REDO

Logging

- **Physical Logging: location on particular page, old value (for UNDO) & new value (for REDO)**
- **Logical/Operation Logging: record high-level op info**
  - Less info to log, but hard to get recovery correct
- **Physiological Logging: for each page, record logical ops**
- **Write Ahead Logging**
  - Write log record to non-volatile storage before update data
  - Transaction committed iff all its log records (incl. commit record) in non-volatile storage
  - Each page has Log Sequence Number (LSN) of latest update

ARIES Recovery Method

- **Write ahead logging; STEAL (UNDO); NO FORCE (REDO)**

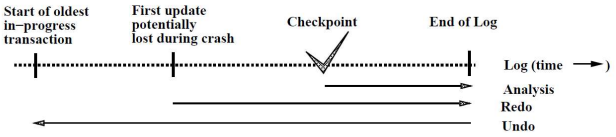

Figure 3: The Three Passes of ARIES Restart

- **Transaction Table of currently running transactions**
  - lastLSN: most recent log record written by transaction

ARIES Recovery Method

- **Dirty Page Table: pages with updates NOT reflected in non-volatile storage**
  - recoveryLSN: earliest LSN that made page dirty
  - prevLSN: backwards linking of transaction's log records

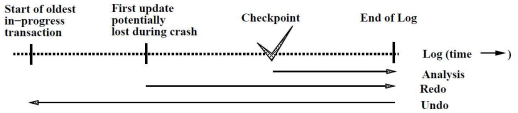

Figure 3: The Three Passes of ARIES Restart

- **Analysis Pass**
  - Processes log, updating Transaction & Dirty Page tables
  - firstLSN = Earliest recoveryLSN (start of REDO)

ARIES: Redo Pass

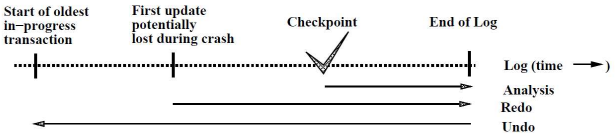

Figure 3: The Three Passes of ARIES Restart

REDO updates for all Trans. from firstLSN, committed or not

- If affected page is NOT in Dirty Page Table (DPT), ignore
- If IS in DPT but recoveryLSN > LSN of record being checked, ignore [Flush occurred after this record (& prior to checkpoint)]
- Otherwise, fetch the page to get the pageLSN.
  If pageLSN ≥ LSN of record being checked, ignore
  [since flush occurred after this record]
  Else apply logged action to page & set pageLSN (but no logging)

ARIES: Undo Pass

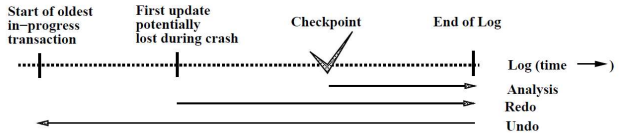

Figure 3: The Three Passes of ARIES Restart

UNDO all transactions not committed at time of crash

- Follow backward chains (prevLSN) for each transaction
- For crash recovery during UNDO: Log Compensation Log Record containing next log record to be undone for transaction

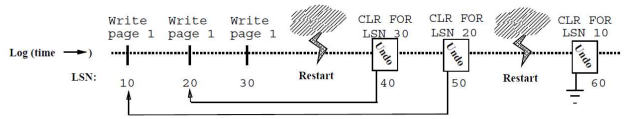

Figure 4: The Use of CLRs for UNDO

Distributed Transactions?

- **Need two-phase commit to ensure agreement on whether or not to commit**
- **Many issues to address: E.g.,**
  - Need to handle failures during two-phase commit
  - Lack of global LSN ordering
  - Node needs to log additional state in order to reply to other recovering nodes
  - Nodes may have inconsistent views on database state
- **One approach: Each data item has an owning node, and only the owner can update the data item**

Multiversion Concurrency Control: Theory and Algorithms

Philip Bernstein & Nathan Goodman [TODS 1983]

- **Theory for analyzing the correctness of MVCC algorithms**
  - Need to map reads/writes to versioned reads/writes
  - Transactions ordered by "write before its reads"
  - One-copy serializability: equivalent to serial schedule on single-version DB
- **Show correctness of 3 MVCC algorithms**
  - Locking-based; Timestamp-based; Mixed Method that uses Lamport clocks for consistent timestamps

Coordination Avoidance in Database Systems

Bailis, Fekete, Franklin, Ghodsi, Hellerstein, Stoica [VLDB'15]

- **Coordination in Distributed Systems limits:**
  - Scalability
  - Throughput
  - Low latency
  - Availability
- **Invariant Confluence: Coordination can be avoided iff all local commit decisions are globally valid when merged**

Invariant Confluence Tests

Constraint	*Operation*	*Passes ICT?*
Equality, Inequality	Any	Y
Generate unique ID	Any	Y
Specify unique ID	Insert	N
>	Increment	Y
>	Decrement	N
<	Decrement	Y
<	Increment	N
Foreign Key	Insert	Y
Foreign Key	Delete	Y*
Secondary Indexing	Any	Y
Materialized Views	Any	Y
AUTO_INCREMENT	Insert	N

Monday's Class

Implementing Fault-Tolerant Services using the State Machine Approach: A Tutorial

Fred Schneider

[ACM Computing Surveys 1990]