Today’s Reminders / Announcements

• No class on Friday
  – Can review for midterm or think about projects

• Midterm #1 on Monday
  – Limited to the 14 papers for which you wrote summaries
  – May be helpful to review lecture slides
  – Only need to answer 7 of 9 questions
  – 80 minutes. No notes or papers
  – Added a 2015 sample midterm to the course web page

• Projects
  – Feb 24: Day of meetings to discuss project ideas
  – Will be posting sample projects
“Concurrency Control and Recovery”
Michael J. Franklin 1997

- Mike Franklin (U. Maryland, UC Berkeley, U. Chicago)
  - ACM Fellow
  - Sigmod Test-of-Time award 2004 & 2013

Chapter in the Computer Science & Engineering Handbook
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
Concurrent 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 locking)
  - READ UNCOMMITTED: no read locks (can read roll-backed updates)
  - READ COMMITTED: short-duration read locks (non-repeatable reads)
  - REPEATABLE READ: strict read locks (phantom tuples)
  - SERIALIZABLE: strict read locks on predicates (harder to achieve)
Concurrency Control (cont.)

- Hierarchical Locking (DB, table, page, tuple)
  - Intention Shared (IS), Intention Exclusive (IX), Shared with Intension Exclusive (SIX)
  - Automatic lock escalation (when locking many children, grab parent lock instead)

<table>
<thead>
<tr>
<th></th>
<th>IS</th>
<th>IX</th>
<th>S</th>
<th>SIX</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>IX</td>
<td>y</td>
<td>y</td>
<td>n</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>S</td>
<td>y</td>
<td>n</td>
<td>y</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>SIX</td>
<td>y</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>X</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
</tr>
</tbody>
</table>

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

• Recovery: UNDO vs. REDO
  – UNDO: remove effects of incomplete/aborted transactions
  – REDO: re-instating effects of committed transactions

• Buffer Management: STEAL & FORCE
  – 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)
  - May be many physical changes for a single high-level op

- **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 (including its commit record) are in non-volatile storage
  - Each page has Log Sequence Number (pageLSN) of latest update
Discussion: Summary Question #1

- **State the 3 most important things the paper says.** These could be some combination of their motivations, observations, interesting parts of the design, or clever parts of their implementation.
ARIES Recovery Method

- Uses: Write Ahead Logging / STEAL (undo) / NO FORCE (redo)

Figure 3: The Three Passes of ARIES Restart
ARIES Recovery Method

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

- **Dirty Page Table: pages with updates NOT reflected in NV storage**
  - recoveryLSN: earliest LSN that made page dirty (for Redo)

- **prevLSN**: backwards linking of transaction’s log records

---

### Analysis Pass

- Processes log, updating Transaction & Dirty Page tables
- $\text{firstLSN} = \text{Earliest recoveryLSN}$ (start of REDO)
ARIES: Redo Pass

REDO updates for all Transactions 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

**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
Discussion: Summary Question #2

• Describe the paper's single most glaring deficiency. Every paper has some fault. Perhaps an experiment was poorly designed or the main idea had a narrow scope or applicability.
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 1983

- **Phil Bernstein (Harvard, Microsoft Research)**
  - NAE, ACM Fellow
  - SIGMOD Edgar F. Codd Innovations Award

- **Nathan Goodman (Harvard, bioinformatics industry)**
  - Founded non-profit organization providing education & support for people with Huntington’s disease
Multiversion Concurrency Control

• 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 (1-SR): equivalent to serial schedule on single-version DB

• Proves correctness of 3 MVCC algorithms
  – Timestamp-based [Reed78]
  – Locking-based [generalization of Bayer80, Stearns81]
  – Mixed Method using Lamport clocks for consistent timestamps [generalization of Dubourdieu82]
• Describe what conclusion you draw from the paper as to how to build systems in the future. Most of the assigned papers are significant to the systems community and have had some lasting impact on the area.
No Class Friday

Midterm Monday