Logging and conflict-serializability

CSE 444 section
October 28, 2010
Today

• Logging and recovery review

• Identifying conflict-serializable schedules
Why use logs to recover from crashes?

Helps satisfy 2 of the ACID constraints:

• Atomicity
  – How does log-based recovery keep TXen atomic?
  – How is this done in an undo log?
  – In a redo log?

• Durability
  – How does logging ensure that TXen persist?
Our undo log notation

- `<START T>`
  - Transaction T has begun
- `<COMMIT T>`
  - T has committed
- `<ABORT T>`
  - T has aborted
- `<T, X, v>` - Update record
  - T has updated element X, and its *old* value was v
An undo logging problem

Given this undo log, when can each data item be output to disk?

- **A**: after 2
- **B**: after 3
- **C**: after 5, before 12
- **D**: after 7
- **E**: after 8, before 12
- **F**: after 10
- **G**: after 11

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&lt;START T1&gt;</td>
</tr>
<tr>
<td>2</td>
<td>&lt;T1, A, a&gt;</td>
</tr>
<tr>
<td>3</td>
<td>&lt;T1, B, b&gt;</td>
</tr>
<tr>
<td>4</td>
<td>&lt;START T2&gt;</td>
</tr>
<tr>
<td>5</td>
<td>&lt;T2, C, c&gt;</td>
</tr>
<tr>
<td>6</td>
<td>&lt;START T3&gt;</td>
</tr>
<tr>
<td>7</td>
<td>&lt;T3, D, d&gt;</td>
</tr>
<tr>
<td>8</td>
<td>&lt;T2, E, e&gt;</td>
</tr>
<tr>
<td>9</td>
<td>&lt;START T4&gt;</td>
</tr>
<tr>
<td>10</td>
<td>&lt;T4, F, f&gt;</td>
</tr>
<tr>
<td>11</td>
<td>&lt;T3, G, g&gt;</td>
</tr>
<tr>
<td>12</td>
<td>&lt;COMMIT T2&gt;</td>
</tr>
</tbody>
</table>
Undo logging problem, continued

After writing these log entries, the DBMS crashes. What does it do when it restarts?

• Scan for transactions to undo: T1, T3, T4
• G, F, D, B, A reverted (in that order)
• <ABORT> written for T1, T3, T4

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&lt;START T1&gt;</td>
</tr>
<tr>
<td>2</td>
<td>&lt;T1, A, a&gt;</td>
</tr>
<tr>
<td>3</td>
<td>&lt;T1, B, b&gt;</td>
</tr>
<tr>
<td>4</td>
<td>&lt;START T2&gt;</td>
</tr>
<tr>
<td>5</td>
<td>&lt;T2, C, c&gt;</td>
</tr>
<tr>
<td>6</td>
<td>&lt;START T3&gt;</td>
</tr>
<tr>
<td>7</td>
<td>&lt;T3, D, d&gt;</td>
</tr>
<tr>
<td>8</td>
<td>&lt;T2, E, e&gt;</td>
</tr>
<tr>
<td>9</td>
<td>&lt;START T4&gt;</td>
</tr>
<tr>
<td>10</td>
<td>&lt;T4, F, f&gt;</td>
</tr>
<tr>
<td>11</td>
<td>&lt;T3, G, g&gt;</td>
</tr>
<tr>
<td>12</td>
<td>&lt;COMMIT T2&gt;</td>
</tr>
</tbody>
</table>
What if it was a redo log?

Now, \(<T,X,v>\) means \(X\)'s new value is \(v\)!

... so now when can we output each item?

- **C, E:** after 12
- **Others:** never

(given log available)
Redo log problem, continued

How do we recover from this redo log?

• Scan for transactions to redo: only T2
• C and E rewritten

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&lt;START T1&gt;</td>
</tr>
<tr>
<td>2</td>
<td>&lt;T1, A, a&gt;</td>
</tr>
<tr>
<td>3</td>
<td>&lt;T1, B, b&gt;</td>
</tr>
<tr>
<td>4</td>
<td>&lt;START T2&gt;</td>
</tr>
<tr>
<td>5</td>
<td>&lt;T2, C, c&gt;</td>
</tr>
<tr>
<td>6</td>
<td>&lt;START T3&gt;</td>
</tr>
<tr>
<td>7</td>
<td>&lt;T3, D, d&gt;</td>
</tr>
<tr>
<td>8</td>
<td>&lt;T2, E, e&gt;</td>
</tr>
<tr>
<td>9</td>
<td>&lt;START T4&gt;</td>
</tr>
<tr>
<td>10</td>
<td>&lt;T4, F, f&gt;</td>
</tr>
<tr>
<td>11</td>
<td>&lt;T3, G, g&gt;</td>
</tr>
<tr>
<td>12</td>
<td>&lt;COMMIT T2&gt;</td>
</tr>
</tbody>
</table>
Why add (non-quiescent) checkpoints?
Undo log recovery with checkpoints

The DBMS crashes with this undo log.

What do we do to recover?

– Which log entries are read?
  From end to 9: <START CKPT>

– Which transactions are undone?
  None; all have committed

– Which data do we change?
  None; no transactions to undo
This similar log is a REDO log.

How do we recover this one?
- Which log entries are read?
  - From end to 9: <START CKPT>
  - Then from 4: <START T2> down to end
- Which transactions are redone?
  - T2, T3, T4
- Which data do we change?
  - C ← c, D ← d, E ← e, F ← f, G ← g

Lines 15, 16 swapped
Today

• Logging and recovery review
• Identifying conflict-serializable schedules
Schedules and conflicts

For some transaction $T_1$:

- $r_1(X)$ means "$T_1$ reads the data element $X$"
- $w_1(X)$ means "$T_1$ writes the data element $X$"

Two actions from $T_1$, $T_2$ conflict iff:

- one or both is a write, and
- they act on the same element

Two actions both from $T_1$ also conflict
Example 1: find all conflicts

\[ \begin{align*}
& w_3(A) \\
& r_1(A) \\
& w_1(B) \\
& r_2(B) \\
& w_3(C) \\
& r_2(C)
\end{align*} \]
The precedence graph

• Recall: $T_1$ must *precede* $T_2$ iff an action from $T_1$ conflicts with a later action from $T_2$
  – Ignore conflicting actions from the same transaction

• Precedence graph shows the precedence relations
Example 1: precedence graph
Is it conflict serializable?

• **YES:** if no cycles in the precedence graph
  – Any transaction order which follows the precedences shown is an equivalent serial schedule

• **NO:** if there are cycles in the precedence graph
Example 1: conflict serializable?

No cycles: **YES**, conflict serializable

Only serial equivalent schedule: T₃, T₁, T₂
Example 1: serial equivalent

Only serial equivalent schedule: \( T_3, T_1, T_2 \)
Example 2: find non-self conflicts

\[ r_1(A) \]
\[ r_2(A) \]
\[ r_1(B) \]
\[ r_2(B) \]
\[ r_3(A) \]
\[ r_4(B) \]
\[ w_1(A) \]
\[ w_2(B) \]
Example 2: precedence graph
Example 2: conflict serializable?

Cycle between $T_1$ and $T_2$: **NO**, not conflict serializable