Logging & Recovery

April 18, 2017
Announcements

• CSE-662 **Wait List** created

• I will **force reg** up to 10 students for CSE-662
  
  • Required: B+ in 562

  • If >10 eligible, selection will be based on weighted avg of project/exam grades.

• **In-Class Final Exam:** May 11

  • If this is a problem, contact me directly.
What does it mean for a transaction to be committed?
If commit \textit{returns successfully}, the transaction…

• … is recorded completely (atomicity)
• … left the database in a stable state (consistency)
• …’s effects are independent of other xacts (isolation)
• … will survive failures (durability)
commit returns successfully = the xact’s effects are visible forever
commit returns successfully = the xact’s effects are visible forever

commit called but doesn’t return = the xact’s effects may be visible
Motivation

Committed Transactions.
These should be present when the DB restarts.

Uncommitted Transactions.
These should leave no trace

Image copyright: Wikimedia Commons
ACID

• **Isolation**: Already addressed.
• **Atomicity**: Need writes to get *flushed* in a single step.
  • IOs are only atomic at the page level.

• **Durability**: Need to *buffer* some writes until commit.
  • May need to free up memory for another xact.

• **Consistency**: Need to roll back incomplete xacts.
  • May have already paged back to disk.
Atomicity

- **Problem**: IOs are only atomic for 1 page.
  - What if we crash in between writes?
- **Solution**: Logging (e.g., Journaling Filesystem)
  - Log everything first before you do it.
Durability / Consistency

- **Problem**: Buffer memory is limited
  - What if we need to ‘page out’ some data?

- **Solution**: Use log (or similar) to recover buffer
  - *Problem*: Commits more expensive

- **Solution**: Modify DB in place, use log to ‘undo’ on abort
  - *Problem*: Aborts more expensive
Problem 1: Providing durability under failures.
Simplified Model
When a write succeeds, the data is completely written
Problems

- A crash occurs part-way through the write.
- A crash occurs before buffered data is written.
Write-Ahead Logging

Before writing to the database, first write what you plan to write to a log file...

<table>
<thead>
<tr>
<th>Log</th>
<th>W(A:10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>8</td>
</tr>
<tr>
<td>B</td>
<td>12</td>
</tr>
<tr>
<td>C</td>
<td>5</td>
</tr>
<tr>
<td>D</td>
<td>18</td>
</tr>
<tr>
<td>E</td>
<td>16</td>
</tr>
</tbody>
</table>

Image copyright: OpenClipart (rg1024)
Write-Ahead Logging

Once the log is safely on disk you can write the database

Log

\[ W(A:10) \]

Image copyright: OpenClipart (rg1024)
Write-Ahead Logging

Log is append-only, so writes are always efficient

Log
W(A:10)
W(C:8)
W(E:9)
Write-Ahead Logging

...allowing random writes to be safely batched

*Log*

\[ W(A:10) \]
\[ W(C:8) \]
\[ W(E:9) \]
Anatomy of a log entry

Last entry for this Xact (forms a Linked List)

What was written, where, prior value, etc…

Which Xact Triggered This Entry

Write, Commit, etc…

Entry Metadata
Problem 2: Providing rollback.
Single DB Model

**Txn 1**
- A = 20
- B = 14
- COMMIT

**Txn 2**
- E = 19
- B = 15
- ABORT

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8</td>
<td>12</td>
<td>5</td>
<td>18</td>
<td>16</td>
</tr>
</tbody>
</table>

Image copyright: OpenClipart (rg1024)
Single DB Model

**Txn 1**
A = 20
B = 14
COMMIT

**Txn 2**
E = 19
B = 15
ABORT

Image copyright: OpenClipart (rg1024)
Single DB Model

**Txn 1**
- A = 20
- B = 14
- COMMIT

**Txn 2**
- E = 19
- B = 15
- ABORT

### Database
- A: 8
- B: 12
- C: 5
- D: 18
- E: 16
Single DB Model

**Txn 1**
- A = 20
- B = 14
- COMMIT

**Txn 2**
- E = 19
- B = 15
- ABORT
Single DB Model

**Txn 1**
A = 20
B = 14
COMMIT

**Txn 2**
E = 19
B = 15
ABORT

![Database Diagram]

A = 20
B = 15
C = 5
D = 18
E = 16

Image copyright: OpenClipart (rg1024)
Staged DB Model

**Txn 1**
A = 20
B = 14
COMMIT

**Txn 2**
E = 19
B = 15
ABORT
Staged DB Model

**Txn 1**
A = 20  
B = 14  
COMMIT

**Txn 2**
E = 19  
B = 15  
ABORT

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>12</td>
<td>14</td>
<td>18</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>12</td>
<td>15</td>
<td>18</td>
<td>19</td>
</tr>
</tbody>
</table>

Image copyright: OpenClipart (rg1024)
Staged DB Model

**Txn 1**
- A = 20
- B = 14
- COMMIT

**Txn 2**
- E = 19
- B = 15
- ABORT

![Database diagram](image)
Is staging always possible?
• Staging takes up more memory.

• Merging after-the-fact can be harder.

• Merging after-the-fact introduces more latency!
Problem 2: Providing rollback for the single database model
UNDO Logging

Store both the “old” and the “new” values of the record being replaced.

\[ W(A:8 \rightarrow 10) \]
\[ W(C:5 \rightarrow 8) \]
\[ W(E:16 \rightarrow 9) \]
UNDO Logging

**Active Xacts**
- Xact: 1, Log: 45
- Xact: 2, Log: 32

**Log**
- 43: \( W(A:8 \rightarrow 10) \)
- 44: \( W(C:5 \rightarrow 8) \)
- 45: \( W(E:16 \rightarrow 9) \)
UNDO Logging

Active Xacts
Xact: 1, Log: 45
Xact: 2, Log: 32

ABORT

Log
43: W(A: 8→10)
44: W(C: 5→8)
45: W(E: 16→9)
UNDO Logging

Active Xacts
Xact: ABORT, Log: 45
Xact: 2, Log: 32

Log
43: W(A: 8 → 10)
44: W(C: 5 → 8)
45: W(E: 16 → 9)

Image copyright: OpenClipart (rg1024)
UNDO Logging

Active Xacts
Xact: 1, Log: 45
Xact: 2, Log: 32

Log:
43: W(A: 8 → 10)
44: W(C: 5 → 8)
45: W(E: 16 → 9)
UNDO Logging

Active Xacts
- Xact: 1, Log: 45
- Xact: 2, Log: 32

Log
- 43: W(A: 8 → 10)
- 44: W(C: 5 → 8)
- 45: W(E: 16 → 9)

Image copyright: OpenClipart (rg1024)
Problem 3: Providing atomicity.
**Goal**: Be able to reconstruct all state at the time of the DB’s crash (minus all running xacts)
## Transaction Table

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Status</th>
<th>Last Log Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction 24</td>
<td>VALIDATING</td>
<td>99</td>
</tr>
<tr>
<td>Transaction 38</td>
<td>COMMITTING</td>
<td>85</td>
</tr>
<tr>
<td>Transaction 42</td>
<td>ABORTING</td>
<td>87</td>
</tr>
<tr>
<td>Transaction 56</td>
<td>ACTIVE</td>
<td>100</td>
</tr>
</tbody>
</table>
## Buffer Manager

<table>
<thead>
<tr>
<th>Page</th>
<th>Status</th>
<th>First Log Entry</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>DIRTY</td>
<td>47</td>
<td>01011010...</td>
</tr>
<tr>
<td>30</td>
<td>CLEAN</td>
<td>n/a</td>
<td>11001101...</td>
</tr>
<tr>
<td>52</td>
<td>DIRTY</td>
<td>107</td>
<td>10100010...</td>
</tr>
<tr>
<td>57</td>
<td>DIRTY</td>
<td>87</td>
<td>01001101...</td>
</tr>
<tr>
<td>66</td>
<td>CLEAN</td>
<td>n/a</td>
<td>01001011...</td>
</tr>
</tbody>
</table>
DB State

On-Disk (or rebuildable)

In-Memory Only!

Active Xacts
Xact:1, Log: 45
Xact:2, Log: 32

On-Disk

Log
43: W(A: 8→10)
44: W(C: 5→8)
45: W(E: 16→9)

Image copyright: OpenClipart (rg1024)
ARIES Recovery

1. Rebuild Transaction Table
2. Rebuild Buffer Manager State
3. ABORT Crashed Transactions
Transaction Table

**Step 1:** Rebuild Transaction Table

- Log all state changes
- Replay state change log entries
Required Log Entries

Log every COMMIT
(replay triggers commit process)

Log every ABORT
(replay triggers abort process)

New message: END
(replay removes Xact from Xact Table)

What about BEGIN?
(when does an Xact get added to the Table?)
Transaction Commit

• Write **Commit** Record to Log

• All Log records up to the transaction’s LastLSN are flushed.

  • Note that Log Flushes are Sequential, Synchronous Writes to Disk

• Commit() returns.

• Write **End** record to log.
Speeding Up Recovery

• **Problem**: We might need to scan to the very beginning of the log to recover the full state of the Xact table (& Buffer Manager)

• **Solution**: Periodically save (checkpoint) the Xact table to the log.
  
  • Only need to scan the log up to the last (successful) checkpoint.
Checkpointing

- **begin_checkpoint** record indicates when the checkpoint began.

- Checkpoint covers all log entries before this entry.

- **end_checkpoint** record contains the current transaction table and the dirty page table.

- Signifies that the checkpoint is now stable.
Buffer Manager

**Step 2:** Recover Buffered Data

- Where do we get the buffered data from?

Save Dirty Page Table with Checkpoint
Consistency

**Step 3**: Undo incomplete xacts

- Record *previous values* with log entries
- Replay log in reverse (linked list of entries)
  - Which Xacts do we undo?
  - Which log entries do we undo?
  - How far in the log do we need to go?
Compensation Log Records

• **Problem**: Step 3 is expensive!
  • What if we crash during step 3?

• **Optimization**: Log undos as writes as they are performed (CLR).s.
  • Less repeat computation if we crash during recovery
  • Shifts effort to step 2 (replay)
  • CLR.s don’t need to be undone!
ARIES Crash Recovery

- Start from checkpoint stored in master record.
- **Analysis**: Rebuild the Xact Table
- **Redo**: Replay operations from all live Xacts (even uncommitted ones).
- **Undo**: Revert operations from all uncommitted/aborted Xacts.

**Oldest log record of transaction active at crash**

**Smallest recLSN in dirty page table after Analysis**

**Last Checkpoint**

**CRASH**
## Recovery Example

<table>
<thead>
<tr>
<th>LSN</th>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>begin_checkpoint</td>
</tr>
<tr>
<td>05</td>
<td>end_checkpoint</td>
</tr>
<tr>
<td>10</td>
<td>update: T1 writes P5</td>
</tr>
<tr>
<td>20</td>
<td>update: T2 writes P3</td>
</tr>
<tr>
<td>30</td>
<td>T1 Abort</td>
</tr>
<tr>
<td>40</td>
<td>CLR Undo T1 LSN 10</td>
</tr>
<tr>
<td>45</td>
<td>T1 End</td>
</tr>
<tr>
<td>50</td>
<td>update: T3 writes P1</td>
</tr>
<tr>
<td>60</td>
<td>update: T2 writes P5</td>
</tr>
</tbody>
</table>

**PrevLSNs**

**CRASH! Restart!**
Analysis

<table>
<thead>
<tr>
<th>LSN</th>
<th>Log</th>
<th>Xact Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>begin_checkpoint</td>
<td>T1; &lt;0</td>
</tr>
<tr>
<td>05</td>
<td>end_checkpoint</td>
<td>T2; 20</td>
</tr>
<tr>
<td>10</td>
<td>update: T1 writes P5</td>
<td>T3; 50</td>
</tr>
<tr>
<td>20</td>
<td>update: T2 writes P3</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>T1 Abort</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>CLR Undo T1 LSN 10</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>T1 End</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>update: T3 writes P1</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>update: T2 writes P5</td>
<td></td>
</tr>
</tbody>
</table>
Redo

<table>
<thead>
<tr>
<th>LSN</th>
<th>Log</th>
<th>Xact Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>begin_checkpoint</td>
<td></td>
</tr>
<tr>
<td>05</td>
<td>end_checkpoint</td>
<td>T2; 60</td>
</tr>
<tr>
<td>10</td>
<td>update: T1 writes P5</td>
<td>T3; 50</td>
</tr>
<tr>
<td>20</td>
<td>update: T2 writes P3</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>T1 Abort</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>CLR Undo T1 LSN 10</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>T1 End</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>update: T3 writes P1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>update: T2 writes P5</td>
<td></td>
</tr>
</tbody>
</table>
Undo

<table>
<thead>
<tr>
<th>LSN</th>
<th>Log</th>
<th>Xact Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>00,05</td>
<td>begin_checkpoint, end_checkpoint</td>
<td>T2; 60</td>
</tr>
<tr>
<td>10</td>
<td>update: T1 writes P5</td>
<td>T3; 50</td>
</tr>
<tr>
<td>20</td>
<td>update: T2 writes P3</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>T1 Abort</td>
<td></td>
</tr>
<tr>
<td>40, 45</td>
<td>CLR Undo T1 LSN 10; T1 End</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>update: T3 writes P1</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>update: T2 writes P5</td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>CRASH</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>CLR: Undo T2, LSN 60</td>
<td></td>
</tr>
<tr>
<td>90,95</td>
<td>CLR: Undo T3, LSN 50; T3 End</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>CRASH</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>CLR: Undo T2, LSN 20; T2 End</td>
<td></td>
</tr>
</tbody>
</table>

CRASH!