CS422 Principles of Database Systems
Failure Recovery

Chengyu Sun
California State University, Los Angeles

ACID Properties of DB Transaction
- Atomicity
- Consistency
- Isolation
- Durability

Failure Recovery
- Ensure atomicity and durability despite system failures

System crash
start transaction;
select balance from accounts where id=1;
update accounts set balance=balance-100
  where id=1;
update accounts set balance=balance+100
  where id=2;
commit;
System crash

Failure Model
- System crash
  - CPU halts
  - Data in memory is lost
  - Data on disk is OK
- Everything else

Logging
- Log
  - A sequence of log records
  - Append only

What Do We Log
- Transaction
  - start transaction;
  - select balance
    - from accounts
      - where id=1;
  - update accounts
    - set balance=balance+100
      - where id=1;
  - update accounts
    - set balance=balance+100
      - where id=2;
  - commit;
- Log
Log Records in SimpleDB

<table>
<thead>
<tr>
<th>Record Type</th>
<th>Transaction #</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;START, 27&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;SETINT, 27, accounts.tbl, 0, 38, 1000, 900&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;COMMIT, 27&gt;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>File Name</th>
<th>Block #</th>
<th>Position</th>
<th>Old Value</th>
<th>New Value</th>
</tr>
</thead>
</table>

General Notation for Log Records

- <START, T>
- <COMMIT, T>
- <ABORT, T>

Recover from System Crash

- Remove changes made by uncommitted transactions – Undo
- Reapply changes made by committed transactions – Redo

Recover with Undo Only

- Assumption: all changes made by committed transactions have been saved to disk

Example: Create Undo Logging Records

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Transaction;</td>
<td>&lt;START, T&gt;</td>
</tr>
<tr>
<td>Write(X, v)</td>
<td>&lt;UPDATE, T, X, v&gt;</td>
</tr>
<tr>
<td>Write(Y, v)</td>
<td>&lt;UPDATE, T, Y, v&gt;</td>
</tr>
<tr>
<td>Commit;</td>
<td>&lt;COMMIT, T&gt;</td>
</tr>
</tbody>
</table>

About Logging

- Undo logging records do not need to store the new values
- Why??
- The key of logging is to decide when to flush the changes made by the transaction and the log records to disk
Example: Flushing for Undo Recovery

- Order the actions, including $\text{Flush}(X)$ and $\text{Flush}(\text{log})$, into a sequence that allows Undo Recovery

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Transaction; Write($X, v$)_&lt;sub&gt;T&lt;/sub&gt;</td>
<td>$&lt;\text{START}, \text{T}&gt;$</td>
</tr>
<tr>
<td>Write($Y, v$)_&lt;sub&gt;T&lt;/sub&gt;</td>
<td>$&lt;\text{UPDATE}, \text{T}, X, v&gt;$</td>
</tr>
<tr>
<td>Commit;</td>
<td>$&lt;\text{UPDATE}, \text{T}, Y, v&gt;$</td>
</tr>
<tr>
<td></td>
<td>$&lt;\text{COMMIT}, \text{T}&gt;$</td>
</tr>
</tbody>
</table>

Order Flush(X) and Flush(<UPDATE,X>)) for Undo

- Consider the following cases
  - (a) Both $X$ and $<\text{UPDATE}, X>$ are written to disk
  - (b) $X$ is written to disk but not $<\text{UPDATE}, X>$
  - (c) $<\text{UPDATE}, X>$ is written to disk but not $X$
  - (d) Neither is written to disk

Write-Ahead Logging

- A modified buffer can be written to disk only after all of its update log records have been written to disk

Implement Write-Ahead Logging

- Each log record has a unique id called log sequence number (LSN)
- Each buffer page keeps the LSN of the log record corresponding to the latest change
- Before a buffer page is flushed, notify the log manager to flush the log up to the buffer’s LSN

Order Flush(<COMMIT,T>) for Undo

- $<\text{COMMIT, T}>$ cannot be written to disk before new value of $X$ is written to disk
- Commit statement cannot return before $<\text{COMMIT, T}>$ is written to disk

Undo Logging

- Write $<\text{UPDATE}, T, X, v'>_<$ to disk before writing new value of $X$ to disk
- Write $<\text{COMMIT, T}>$ after writing all new values to disk
- COMMIT returns after writing $<\text{COMMIT, T}>$ to disk
Undo Recovery

- Scan the log
  - *Forward or backward?*
- `<COMMIT,T>`: add T to a list of committed transactions
- `<ABORT,T>`: add T to a list of rolled-back transactions
- `<UPDATE,T,X,v,x>`: if T is not in the lists of committed or aborted transactions, restore X's value to $v_x$

About Undo Recovery

- No need to keep new value $v_1$
- Scan the log once for recovery
- COMMIT must wait until all changes are flushed
- Idempotent – recovery processes can be run multiple times with the same result

Recover with Redo Only

- Assumption: *none* of the changes made by *uncommitted* transactions have been saved to disk

Example: Flushing for Redo Recovery

- Order the actions, including `Flush(X)` and `Flush(<log>)`, into a sequence that allows Undo Recovery

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Transaction;</td>
<td><code>&lt;START,T&gt;</code></td>
</tr>
<tr>
<td>Write(X, $v'_x$)</td>
<td><code>&lt;UPDATE, T, X, v'_x&gt;</code></td>
</tr>
<tr>
<td>Write(Y, $v'_y$)</td>
<td><code>&lt;UPDATE, T, Y, v'_y&gt;</code></td>
</tr>
<tr>
<td>Commit</td>
<td><code>&lt;COMMIT, T&gt;</code></td>
</tr>
</tbody>
</table>

Order Flush(X) and `Flush(<UPDATE,X>)` for Redo

- Consider the following cases
  - (a) Both X and `<UPDATE,X>` are written to disk
  - (b) X is written to disk but not `<UPDATE,X>`
  - (c) `<UPDATE,X>` is written to disk but not X
  - (d) Neither is written to disk

Order `Flush(<COMMIT,T>)` for Redo

- Commit statement cannot return before `<COMMIT,T>` is written to disk
Redo Logging
- Write \(<\text{UPDATE}, T, X, v_x^*\) and \(<\text{COMMIT}, T\) to disk \textit{before} writing any new value of the transaction to disk
- COMMIT returns \textit{after} writing \(<\text{COMMIT}, T\) to disk

Redo Recovery
- Scan the log to create a list of committed transactions
- Scan the log again to replay the updates of the committed transactions
  - \textit{Forward or backward??}

About Redo Recovery
- A transaction must keep all the blocks it needs pinned until the transaction completes – increases buffer contention

Combined Undo and Redo – Undo/Redo Logging
- Write \(<\text{UPDATE}, T, X, v_x^*v_x^*\) to disk \textit{before} writing new value of X to disk
- COMMIT returns \textit{after} writing \(<\text{COMMIT}, T\) to disk

Undo/Redo Recovery
- Stage 1: undo recovery
- Stage 2: redo recovery

Advantages of Undo/Redo
- Vs. Undo??
- Vs. Redo??
Checkpoint

- Log can get very large
- A recovery algorithm can stop scanning the log if it knows
  - All the remaining records are for completed transactions
  - All the changes made by these transactions have been written to disk

Quiescent Checkpointing

- Stop accepting new transactions
- Wait for all existing transactions to finish
- Flush all dirty buffer pages
- Create a <CHECKPOINT> log record
- Flush the log
- Start accepting new transactions

Nonquiescent Checkpointing

- Stop accepting new transactions
- Let \( T_1, \ldots, T_k \) be the currently running transactions
- Flush all modified buffers
- Write the record \(<\text{NQCKPT}, T_1, \ldots, T_k>\) to the log
- Start accepting new transactions

About Nonquiescent Checkpointing

- Do not need to wait for existing transactions to complete
- Recovery algorithm does not need to look beyond the start record of the earliest uncommitted transaction in \( \{T_1, \ldots, T_k\} \)

Example: Nonquiescent Checkpoint

- Using Undo/Redo Recovery

\[
\begin{align*}
\langle \text{START}, 0 \rangle \\
\langle \text{WRITE}, 0, A, v_w, v_s \rangle \\
\langle \text{START}, 1 \rangle \\
\langle \text{START}, 2 \rangle \\
\langle \text{COMMIT}, 1 \rangle \\
\langle \text{WRITE}, 2, B, v_p, v_r \rangle \\
\langle \text{NQCKPT}, 0, 2 \rangle \\
\langle \text{WRITE}, 0, C, v_u, v_v \rangle \\
\langle \text{COMMIT}, 0 \rangle \\
\langle \text{START}, 3 \rangle \\
\langle \text{WRITE}, 2, D, v_q, v_r \rangle \\
\langle \text{WRITE}, 3, E, v_u, v_v \rangle
\end{align*}
\]

Failure Recovery in SimpleDB

- Log Manager
  - \texttt{simpledb.log}
- Recovery Manager
  - \texttt{simpledb.tx.recovery}
SimpleDB Log Manager
- Default log file: simpledb.log
- Grows the log one block at a time
- The last block is kept in memory (i.e. only needs one page)

Append()
- Records are treated as arrays of objects (String or int)
- A new block is created if the current block does not have enough room to hold the new record
- The LSN of a log record is the block number

Locate Records in a Block
Two records: <1, 'Hi'>, <2, 32>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>i</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

LogIterator
- LogIterator iterates through a log backwards
- BasicLogRecord is simply a page and the starting position of a record in the page – it’s up to the Recovery Manager to decide how to read the record

SimpleDB Recovery Manager
- Each transaction has its own recovery manager

LogRecord Interface
- Record types
  - Checkpoint (quiescent)
  - Start
  - Commit
  - Rollback
  - SetInt
  - SetString
- Record operations
  - Write to log
  - Get record type
  - Get transaction #
  - Undo
  - [ Redo ]
Log Record Format

- Array of Integer and String
  - Record type
  - Additional information (optional)
- See the `writeToLog()` method in each log record class

LogRecordIterator

- Built on top of LogIterator
- Convert each BasicLogRecord to an a LogRecord object

Example: LogViewer

- Display the log
  - Up to the last `<CHECKPOINT>`

Recovery Manager

- Each transaction operation (e.g. start, commit, setint, setstring, rollback) creates a log record
- Rollback: undo the changes made by this transaction
- Recovery: perform recovery for the whole database

Undo Recovery in SimpleDB

- Recovery is done inside a transaction
- Iterate through the log backward
  - EOF or `<Checkpoint>`: stop
  - `<Commit>` or `<Abort>`: add transaction number to a list of finished transactions
  - Other: if the transaction # is not in the list of finished transactions, call undo()
- Save the changes (i.e. flush buffers)
- Write a `<CHECKPOINT>` log record

Examples: TestLogWriter

- Write some records in the log for testing purpose
Readings

- Textbook
  - Chapter 13.1-13.3
  - Chapter 14.1-14.3
- SimpleDB source code
  - simpledb.log
  - simpledb.txt
  - simpledb.txt.recovery