CS4432: Database Systems II

Lecture #20
Failure Recovery

Professor Elke A. Rundensteiner
Transaction: a collection of actions that preserve consistency
Big assumption:

If T starts with *consistent state* AND T executes in *isolation*

⇒ T leaves consistent state
Our failure model

- CPU
- Memory (M)
- Disk (D)
- Processor

Diagram showing the components of a computer system and their failure model.
Operations re Storage Hierarchy:

- **Input (x):** block containing $x \rightarrow$ memory
- **Output (x):** block containing $x \rightarrow$ disk
- **Read (x,t):** do input(x) if necessary
  - $t \leftarrow$ value of x in block
- **Write (x,t):** do input(x) if necessary
  - value of x in block $\leftarrow$ t
Key problem: Unfinished transaction

Example: Constraint: $A = B$

$T_1$: $A \leftarrow A \times 2$

$B \leftarrow B \times 2$
T1: Read (A,t); \( t \leftarrow t \times 2 \)
Write (A,t);
Read (B,t); \( t \leftarrow t \times 2 \)
Write (B,t);
Output-to-Disk (A); Output-to-Disk (B);

A: 8 16
B: 8 16

memory

disk

failure!
• Need **atomicity**: execute all actions of a transaction or none at all
One solution: undo logging (immediate modification)

A la Hansel and Gretel recording their navigation through forest via bread crumbs ...

Must have **durable undo logging** !!!
Undo logging  (Immediate modification)

T₁:  Read (A,t);  t ← t×2  \hspace{1cm} A = B
    Write (A,t);
    Read (B,t);  t ← t×2
    Write (B,t);
    Output-to-disk (A);
    Output-to-disk (B);

\[\begin{array}{c}
A:8 16 \\
B:8 16
\end{array}\]  memory

\[\begin{array}{c}
A:8 16 \\
B:8 16
\end{array}\]  disk

\[\begin{array}{c}
<T₁, start>
<T₁, A, 8>
<T₁, B, 8>
<T₁, commit>
\end{array}\]  log
One “complication”

- Log is first written in memory
- Not written to disk on every action

```
memory
A: 8 16
B: 8 16
Log:
<T₁,start>
<T₁,A,8>
<T₁,B,8>
```

```
DB
BAD STATE
# 1
```
One “complication”

- Log is first written in memory
- Not written to disk on every action
Undo logging rules

(1) For every action generate undo log record (containing old value)
(2) Before $x$ is modified on disk, log records pertaining to $x$ must be on disk (write ahead logging)
(3) Before commit is written to log on disk, all writes of transaction must be reflected on disk
Recovery rules: Undo logging

- For every Ti with <Ti, start> in log:
  - If <Ti,commit> or <Ti,abort> in log, do nothing
  - Else For all <Ti, X, ν> in log:
    - write (X, ν)
    - output (X)
    - Write <Ti, abort> to log

—is this correct??
Recovery rules: Undo logging

(1) Let $S =$ set of transactions with $<T_i, \text{start}>$ in log, but no $<T_i, \text{commit}>$ (or $<T_i, \text{abort}>$) record in log

(2) For each $<T_i, X, v>$ in log, in reverse order (latest $\rightarrow$ earliest) do:
   - write old value from log back to disk:
     - if $T_i \in S$ then
       - write $(X, v)$
       - output $(X)$

(3) For each $T_i \in S$ do
   - write $<T_i, \text{abort}>$ to log
What if failure during recovery?

No problem!  ⇨ Undo idempotent
To discuss next:

- Redo logging
- Undo/redo logging, why both?
- Checkpoints
Redo logging (deferred modification)

$T_1$: Read(A, t); $t \leftarrow t \times 2$; write (A, t);
Read(B, t); $t \leftarrow t \times 2$; write (B, t);
Output(A); Output(B)

memory

A: 8 16
B: 8 16

DB

A: 8 16
B: 8

LOG

<T1, start>
<T1, A, 16>
<T1, B, 16>
<T1, commit>

output
Redo logging rules

1. For every action, generate redo log record (containing new value)

2. Before X is modified on disk (DB), all log records for transaction that modified X (including commit) must be on disk

3. Flush log at commit
Recovery rules: Redo logging

- Idea: Need to redo if transaction commit is in log because we don’t know if transaction failed or not.
- For every Ti with \(<Ti, commit>\) in log:
  - For all \(<Ti, X, v>\) in log:
    - Write\((X, v)\)
    - Output\((X)\)

**IS THIS CORRECT??**
Recovery rules: Redo logging

(1) Let $S =$ set of transactions with $<Ti, \text{commit}>$ in log

(2) For each $<Ti, X, v>$ in log, in forward order (earliest $\rightarrow$ latest) do:
   - if $Ti \in S$ then
     \[
     \begin{aligned}
     &\text{Write}(X, v) \\
     &\text{Output}(X) \quad \text{optional}
     \end{aligned}
     \]}
Recovery is very, very **SLOW**!

Redo log:

First Record (1 year ago)

T1 wrote A,B Committed a year ago

--- STILL, Need to redo after crash!!

Last Record

Crash
Solution: Checkpoint (simple version)

Periodically:
1. Do not accept new transactions
2. Wait until all transactions finish
3. Flush all log records to disk (log)
4. Flush all buffers to disk (DB) (do not discard buffers)
5. Write “checkpoint” record on disk (log)
6. Resume transaction processing
Example: what to do at recovery?

Redo log (disk):

<table>
<thead>
<tr>
<th>...</th>
<th>&lt;T1,A,16&gt;</th>
<th>...</th>
<th>&lt;T1,commit&gt;</th>
<th>...</th>
<th>Checkpoint</th>
<th>...</th>
<th>&lt;T2,B,17&gt;</th>
<th>...</th>
<th>&lt;T2,commit&gt;</th>
<th>...</th>
<th>&lt;T3,C,21&gt;</th>
<th>Crash</th>
</tr>
</thead>
</table>

Key drawbacks:

- *Undo logging*: cannot bring backup DB copies up to date
- *Redo logging*: need to keep all modified blocks in memory until commit
Solution: undo/redo logging!

Combine Undo/Redo Logging, namely:

Update $\Rightarrow$ $<T_i, Xid, \text{New X val}, \text{Old X val}>$
page X
Rules

• Page X can be flushed before or after Ti commit
• Log record flushed before corresponding updated page (WAL)
• Flush at commit (log only)
Non-quiesce checkpoint

LOG

... Start-ckpt active TR: Ti,T2,... ...

LOG

: for undo

for undo

dirty buffer pool pages flushed
Examples  what to do at recovery time?

\[ \text{LOG} \]

\[
\begin{array}{ccccccc}
\cdots & T_1,-a & \cdots & \text{Ckpt} & T_1 & \cdots & \text{Ckpt} & \text{end} & \cdots & T_1-b \\
\end{array}
\]

\[ \Rightarrow \text{Undo } T_1 \ (\text{undo } a, b) \]

no T1 commit
Example

LOG

... T₁ a ... ckpt-s T₁ ... T₁ b ... ckpt-end ... T₁ c ... T₁ cmt ...

■ Redo T₁: (redo b,c)
Recovery process:

- **Backwards pass** (end of log ➔ latest checkpoint start)
  - construct set $S$ of committed transactions
  - undo actions of transactions not in $S$

- **Undo pending transactions**
  - follow undo chains for transactions in (checkpoint active list) - $S$

- **Forward pass** (latest checkpoint start ➔ end of log)
  - redo actions of $S$ transactions
Summary

- Consistency of data
- One source of problems: failures
  - Logging
  - Redundancy
- Another source of problems: Data Sharing..... next