CS 277: Database System Implementation

Notes 08: Failure Recovery

Arthur Keller
PART II

- Crash recovery (2 lectures) Ch.17
- Concurrency control (3 lectures) Ch.18
- Transaction processing (2 lects) Ch.19
- Information integration (1 lect) Ch.20
Integrity or correctness of data

- Would like data to be “accurate” or “correct” at all times

<table>
<thead>
<tr>
<th>EMP</th>
<th>Name</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>White</td>
<td>52</td>
</tr>
<tr>
<td></td>
<td>Green</td>
<td>3421</td>
</tr>
<tr>
<td></td>
<td>Gray</td>
<td>1</td>
</tr>
</tbody>
</table>
Integrity or consistency constraints

• Predicates data must satisfy

• Examples:
  - $x$ is key of relation $R$
  - $x \rightarrow y$ holds in $R$
  - Domain($x$) = \{Red, Blue, Green\}
  - $\alpha$ is valid index for attribute $x$ of $R$
  - no employee should make more than twice the average salary
Definition:

- **Consistent state**: satisfies all constraints
- **Consistent DB**: DB in consistent state
Constraints (as we use here) may not capture “full correctness”

Example 1  Transaction constraints
• When salary is updated,
  new salary > old salary
• When account record is deleted,
  balance = 0
Note: could be “emulated” by simple constraints, e.g.,

| account | Acct # | .... | balance | deleted? |
Constraints (as we use here) may not capture “full correctness”

Example 2: Database should reflect real world
in any case, continue with constraints...

**Observation:** DB cannot be consistent always!

**Example:** $a_1 + a_2 + \ldots + a_n = \text{TOT}$ (constraint)

Deposit $100$ in $a_2$:

$$
\begin{align*}
    a_2 &\leftarrow a_2 + 100 \\
    \text{TOT} &\leftarrow \text{TOT} + 100
\end{align*}
$$
Example: \( a_1 + a_2 + \ldots + a_n = \text{TOT} \) (constraint)

Deposit $100 in \( a_2 \):  
\[
\begin{align*}
\text{a}_2 & \leftarrow a_2 + 100 \\
\text{TOT} & \leftarrow \text{TOT} + 100
\end{align*}
\]
Transaction: collection of actions that preserve consistency

Consistent DB \( \rightarrow T \rightarrow \) Consistent DB′
Big assumption:

If T starts with consistent state + T executes in isolation
⇒ T leaves consistent state
Correctness (informally)

• If we stop running transactions, DB left consistent
• Each transaction sees a consistent DB
How can constraints be violated?

- Transaction bug
- DBMS bug
- Hardware failure
  
e.g., disk crash alters balance of account

- Data sharing
  
e.g.: T1: give 10% raise to programmers
  
  T2: change programmers ⇒ systems analysts
How can we prevent/fix violations?

• Chapter 17: due to failures only
• Chapter 18: due to data sharing only
• Chapter 19: due to failures and sharing
Will not consider:

- How to write correct transactions
- How to write correct DBMS
- Constraint checking & repair

That is, solutions studied here do not need to know constraints
Chapter 17  Recovery

• First order of business:  
  Failure Model
Events  ─  Desired
       └── Undesired  ─  Expected
              └── Unexpected
Our failure model

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

Diagram:

```
CPU ← processor

memory [M] ← disk [D]
```
Desired events: see product manuals....

Undesired expected events:
System crash
- memory lost
- cpu halts, resets

that’s it!!

Undesired Unexpected: Everything else!
Undesired Unexpected:  Everything else!

Examples:
- Disk data is lost
- Memory lost without CPU halt
- CPU implodes wiping out universe....
Is this model reasonable?

**Approach:** Add low level checks + redundancy to increase probability model holds

E.g.,
- Replicate disk storage (stable store)
- Memory parity
- CPU checks
Second order of business:

Storage hierarchy
Operations:

- **Input (x):** block with x → memory
- **Output (x):** block with x → disk
- **Read (x,t):** do input(x) if necessary
  \[ t \leftarrow \text{value of } x \text{ in block} \]
- **Write (x,t):** do input(x) if necessary
  \[ \text{value of } x \text{ in block} \leftarrow t \]
Key problem  Unfinished transaction

Example  Constraint: \( A = B \)

\[ T_1: \quad A \leftarrow A \times 2 \]

\[ B \leftarrow B \times 2 \]
T₁:  Read (A,t);  t ← t×2
     Write (A,t);
     Read (B,t);  t ← t×2
     Write (B,t);
     Output (A);
     Output (B);  

memory

disk

A: ⚡ 16
B: ⚡ 16

A: ⚡ 16
B: 8

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

due to: Hansel and Gretel, 782 AD

• Improved in 784 AD to durable undo logging
**Undo logging**  (Immediate modification)

T₁: Read (A,t);  \( t \leftarrow t \times 2 \)  \( A=B \)
Write (A,t);
Read (B,t);  \( t \leftarrow t \times 2 \)
Write (B,t);
Output (A);
Output (B);

```
A: 8 16
B: 8 16
```

```
memory
```

```
A: 16
B: 16
```

```
disk
```

```
<T1, start>
<T1, A, 8>
<T1, B, 8>
<T1, commit>
```

log
One “complication”

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

<table>
<thead>
<tr>
<th>memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: 8 16</td>
</tr>
<tr>
<td>B: 8 16</td>
</tr>
<tr>
<td>Log:</td>
</tr>
<tr>
<td>&lt;T₁,start&gt;</td>
</tr>
<tr>
<td>&lt;T₁, A, 8&gt;</td>
</tr>
<tr>
<td>&lt;T₁, B, 8&gt;</td>
</tr>
</tbody>
</table>

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: WAL)

(3) Before commit is flushed to log, 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, v> in log:
    \[\begin{align*}
    &\text{write } (X, v) \\
    &\text{output } (X) \\
    \text{Write } <Ti, \text{ abort}> \text{ to log}
    \end{align*}\]

\[\text{IS THIS CORRECT??}\]
Recovery rules: Undo logging

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

(2) For each $<Ti, X, v> \text{ in log,}$
    in reverse order (latest $\rightarrow$ earliest) do:
    - if $Ti \in S$ then
      - write $(X, v)$
      - output $(X)$

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

No problem! / Undo idempotent
To discuss:

- Redo logging
- Undo/redo logging, why both?
- Real world actions
- Checkpoints
- Media failures
Redo logging (deferred modification)

T₁: 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)
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

- For every Ti with <Ti, commit> in log:
  - For all <Ti, X, v> in log:
    \[
    \begin{align*}
    \text{Write}(X, v) \\
    \text{Output}(X)
    \end{align*}
    \]

\[\text{IS THIS CORRECT??}\]
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
     \[
     \begin{align*}
     \text{Write}(X, v) \\
     \text{Output}(X) \quad \text{optional}
     \end{align*}
     \]
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!!
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></th>
<th>&lt;T1, commit&gt;</th>
<th></th>
<th>&lt;T2, commit&gt;</th>
<th></th>
<th>&lt;T3, C, 21&gt;</th>
<th>Crash</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>&lt;T1, A, 16&gt;</td>
<td>...</td>
<td>Checkpoint</td>
<td>...</td>
<td>...</td>
<td>&lt;T2, B, 17&gt;</td>
<td>...</td>
</tr>
</tbody>
</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!

Update ⇒ <Ti, Xid, New X val, 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,...

... end ckpt ...

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

LOG

\[ \cdots \quad T_1, -a \quad \cdots \quad \text{Ckpt} \quad T_1 \quad \cdots \quad \text{Ckpt} \quad \text{end} \quad \cdots \quad T_1-b \]

\( \Rightarrow \) Undo \( T_1 \) (undo \( a, b \))

no \( T_1 \) commit
Example

\[ \text{Redo } T_1: (\text{redo } b,c) \]
Recovery process:

- **Backwards pass** (end of log $\Leftarrow$ 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 $\Leftarrow$ end of log)
  - redo actions of $S$ transactions
Real world actions

E.g., dispense cash at ATM

\[ T_i = a_1 a_2 \ldots a_j \ldots a_n \]

\[ \downarrow \]

\$
Solution

(1) execute real-world actions after commit
(2) try to make idempotent
ATM

Give$(amt, Tid, time)$

give$(amt)$

lastTid:

time:

give$(amt)$

$
Media failure  (loss of non-volatile storage)

Solution: Make copies of data!
Example 1  Triple modular redundancy

- Keep 3 copies on separate disks
- Output(X) --> three outputs
- Input(X) --> three inputs + vote
Example #2    Redundant writes, Single reads

• Keep N copies on separate disks
• Output(X) --> N outputs
• Input(X) --> Input one copy
  - if ok, done
  - else try another one

Assumes bad data can be detected
Example #3: DB Dump + Log

- If active database is lost,
  - restore active database from backup
  - bring up-to-date using redo entries in log
### When can log be discarded?

<table>
<thead>
<tr>
<th>log</th>
<th>db dump</th>
<th>last needed undo</th>
<th>checkpoint</th>
</tr>
</thead>
</table>

- Not needed for media recovery
- Not needed for undo after system failure
- Not needed for redo after system failure
Summary

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