Chapter 17: 
Coping with System Failures

(Slides by Hector Garcia-Molina, 
http://www-db.stanford.edu/~hector/cs245/notes.htm)

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 → y holds in R
  - Domain(x) = {Red, Blue, Green}
  - α 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

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 \( a_2 \):
\[
\begin{align*}
a_2 &\leftarrow a_2 + 100 \\
\text{TOT} &\leftarrow \text{TOT} + 100
\end{align*}
\]

\[
\begin{array}{c|c|c}
\hline
\text{a}_2 & \text{50} & \text{150} \\
\hline
\text{TOT} & \text{1000} & \text{1100} \\
\hline
\end{array}
\]
**Transaction:** collection of actions that preserve consistency

![Diagram](image)

**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

We 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

Exercise

- Given a consistency constraint
  - 0 <= A <= B
- Which of the transactions preserves consistency?
  - A := A+B; B := A+B
  - B := A+B; A := A+B
  - A := B+1; B := A+1
Chapter 17  Recovery Management

• First order of business:
  Failure Model

Events — Desired
  — Undesired — Expected
  — Unexpected

Our failure model

CPU — processor
memory —— M

D — disk
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 $\rightarrow$ memory
- Output (x): block with x $\rightarrow$ disk
- Read (x,t): do input(x) if necessary
- Write (x,t): do input(x) if necessary

Exercise

- Consider transaction
  - $A := A+B; B := A+B$
- Assume initially $A=5$ and $B=10$
- Add read- and write-actions to the computation
- Show the effects of the steps on main memory and disk
Key problem: Unfinished transaction

Example: Constraint: $A = B$

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

\[ T_1: \quad \text{Read (A, t); t} \leftarrow t \times 2 \]
\[ \text{Write (A, t);} \]
\[ \text{Read (B, t); t} \leftarrow t \times 2 \]
\[ \text{Write (B, t);} \]
\[ \text{Output (A);} \]
\[ \text{Output (B); failure!} \]

\[ A: 8'16 \]
\[ B: 8'16 \]

memory

disk

- 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)

$T_1$:  
Read $(A, t)$; $t ← t \times 2$  
$A = B$  
Write $(A, t)$;  
Read $(B, t)$; $t ← t \times 2$  
Write $(B, t)$;  
Output $(A)$;  
Output $(B)$;

---

One “complication”

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

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 → earliest) do:
   - 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
Exercise

• Consider the following undo-log records by two transactions T and U:
  – <START T>; <T, A, 10>; <START U>, <U, B, 20>; <T, C, 30>; <U, D, 40>; <COMMIT U>; <T, E, 50>; <COMMIT T>
  – Suppose there is a crash and the last log record on disk is <T, E, 50>.
  – Describe the actions of the recovery manager.
  – What if the last log record on disk is <COMMIT, U>?

What if failure during recovery?
No problem! ⇐ Undo idempotent

Redo logging (deferred modification)

T1: Read(A,t); t×2; write (A,t);
    Read(B,t); t×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

(1) Let S = set of transactions with <Ti, commit> in log
(2) For each <Ti, X, v> in log, in forward order (earliest → latest) do:
   - if Ti ∈ S then Write(X, v)
   - Output(X) ← optional

Exercise

- Consider the following redo-log records by two transactions T and U:
  - <START T>; <T, A, 10>; <START U>, <U, B, 20>; <T, C, 30>; <U, D, 40>; <COMMIT U>; <T, E, 50>; <COMMIT T>
  - Suppose there is a crash and the last log record on disk is <T, E, 50>.
  - Describe the actions of the recovery manager.
  - What if the last log record on disk is <COMMIT, U>?
Recovery is very, very **SLOW**!

Redo log:

![Diagram of redo log](image)

First Record

(1 year ago)  

\( \rightarrow STIL, \text{ Need to redo after crash}!! \)

Last Record

T1 wrote A,B

Committed a year ago

\( \rightarrow \text{STILL, Need to redo after crash}!! \)

---

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):

![Diagram of redo log](image)
Comparison of undo and redo logging

Key drawbacks:

- **Undo logging:** must flush data to disk immediately after a transaction finishes
  
  - Why is it bad?

- **Redo logging:** need to keep all modified blocks in memory until commit
  
  - Why is it bad?

Solution: undo/redo logging!

Update ⇒ <Ti, Xid, Old X val, New 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)
Recovery process:

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

Exercise

- Consider undo/redo log records by two transactions T and U:
  - <START T>; <T, A, 10, 11>; <START U>; <U, B, 20, 21>; <T, C, 30, 31>; <U, D, 40, 41>; <COMMIT U>; <T, E, 50, 51>; <COMMIT T>
  - Suppose there is a crash and the last log record on disk is <T, E, 50, 51>.
  - Describe the actions of the recovery manager.
Examples  what to do at recovery time?

no T1 commit

LOG

... T1 - a ... Ckpt T1 ... Ckpt end ... T1 - b

➽ Undo T1 (undo a,b)

Example

LOG

... T1 ... skpt-s T1 ... T1 - b ... skpt-end ... T1 - c ... T1 - cmn ...

➽ Redo T1: (redo b,c)

Nonquiescent checkpoint

• The rules: section 17.4.3
• Example 17.12 p. 906
Real world actions

E.g., dispense cash at ATM
\[ T_i = a_1, a_2, \ldots, a_j, \ldots, a_n \]
\[ \Downarrow \]
\[ $ \]

Solution

execute real-world actions after commit

ATM

Give$$
(amt, Tid, time)

lastTid:

\[ \text{time:} \]

give(amt)

\[ $ \]

\[ $ \]
Chapter 17

Media failure (loss of non-volatile storage)

A: 16

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?

Summary

- Consistency of data
- One source of problems: failures
  - Logging
  - Redundancy
- Another source of problems:
  Data Sharing..... Next: Chapter 18