Chapter 17: Coping with System Failures
Failure Recovery

Introduction
Undo Logging
Redo Logging
Integrity or correctness of data

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

- EMPLOYEE

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

- There are constraints data must satisfy
- Examples:
  - $x$ is key of relation $R$
  - $x \rightarrow y$ holds in $R$
  - $\text{Domain}(x) = \{\text{Red, Blue, Green}\}$
  - Other constraints
Definition:

- **Consistent state**: satisfies all constraints
- **Consistent DB**: DB in consistent state
Observation: DB cannot be consistent always!

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

Deposit $100$ in $a_2$:

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

Deposit $100$ in $a_2$:  
\[ a_2 \leftarrow a_2 + 100 \]

\[ \text{TOT} \leftarrow \text{TOT} + 100 \]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>50</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOT</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ \text{TOT} \leftarrow 1000 \]

\[ \text{TOT} \leftarrow 1100 \]
Transaction: collection of actions that preserve consistency
Big assumption:

If $T$ starts with consistent state + $T$ executes in isolation

$\Rightarrow T$ remains in consistent state
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 $\Rightarrow$ 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

- Processor
- Memory
- Disk

Diagram:
- CPU connected to Memory (M) and Disk (D)
- Memory (M) connected to Disk (D)
Desired events: see product manuals....

Undesired expected events:

System crash
- memory lost
- cpu halts, resets
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 works incorrectly and corrupts data
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

Memory                  Disk
Operations:

- **Input \((x)\):** block containing \(x\) → memory
- **Output \((x)\):** block containing \(x\) → disk

- **Read \((x, t)\):** do input\((x)\) if necessary
  \(t ←\) value of \(x\) in block

- **Write \((x, t)\):** do input\((x)\) if necessary
  value of \(x\) in block ← \(t\)
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 ← t×2
Write (A,t);
Read (B,t); t ← t×2
Write (B,t);
Output (A);
Output (B);

memory

A: 8
B: 8

disk

A: 8
B: 8
T1: Read (A,t); t ← t×2
    Write (A,t);
    Read (B,t); t ← t×2
    Write (B,t);
    Output (A);
    Output (B);

A: 8 16
B: 8 16

memory

disk
T1: Read (A, t); t ← t × 2
   Write (A, t);
   Read (B, t); t ← t × 2
   Write (B, t);
   Output (A);
   Output (B);

failure!

A: 8 16
B: 8 16

memory

A: 8 16
B: 8

disk
Need **atomicity**: execute all actions of a transaction or none at all
One solution: undo logging (immediate modification)
Undo logging  (Immediate modification)

T:  Read (A,t);  t ← t×2       A=B
    Write (A,t);
    Read (B,t);  t ← t×2
    Write (B,t);
    Output (A);
    Output (B);

A:8
B:8

A:8
B:8

memory
disk
log
Undo logging (Immediate modification)

T: Read (A,t); t ← t×2  A=B
   Write (A,t);
   Read (B,t);  t ← t×2
   Write (B,t);
   Output (A);
   Output (B);

A:8
B:8
memory

A:8
B:8
disk

<T, start>
log
Undo logging  (Immediate modification)

T:  Read (A,t);  t ← t×2   A=B
    Write (A,t);
    Read (B,t);  t ← t×2
    Write (B,t);
    Output (A);
    Output (B);

Unid logging (Immediate modification)
Undo logging (Immediate modification)

T: 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  B:8  
A:8  B:8  

memory  disk  log

\(<T, \text{start}>\)  
\(<T, A, 8>\)
Undologging (Immediate modification)

T: 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);

memory

A: 8 16
B: 8 16

disk

A: 8
B: 8

log

<T, start>
<T, A, 8>
Undo logging (Immediate modification)

T: Read (A,t); t ← t×2 A=B
  Write (A,t);
  Read (B,t); t ← t×2
  Write (B,t);
  Output (A);
  Output (B);

A:8  B:8
A:8  B:8

memory

disk

log

<T, start>
<T, A, 8>
<T, B, 8>
Undo logging  (Immediate modification)

T:
- 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  disk  log

\(<T, \text{start}>\)
\(<T, A, 8>\)
\(<T, B, 8>\)
Undo logging (Immediate modification)

T: Read (A,t);  t ← t×2  A=B
Write (A,t);
Read (B,t);  t ← t×2
Write (B,t);
Output (A);
Output (B);

A:8 16
B:8 16

memory

A:8 16
B:8 16

disk

<T, start>
<T, A, 8>
<T, B, 8>

log
Undo logging  (Immediate modification)

T:  Read (A,t);  t ← t×2       A=B
Write (A,t);
Read (B,t);  t ← t×2
Write (B,t);
Output (A);
Output (B);

<table>
<thead>
<tr>
<th></th>
<th>memory</th>
<th>disk</th>
<th>log</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>8</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>8</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

<T, start>
<T, A, 8>
<T, B, 8>
<T, commit>
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: 16</td>
</tr>
<tr>
<td>B: 16</td>
</tr>
<tr>
<td>Log:</td>
</tr>
<tr>
<td>&lt;T,start&gt;</td>
</tr>
<tr>
<td>&lt;T, A, 8&gt;</td>
</tr>
<tr>
<td>&lt;T, B, 8&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: 8</td>
</tr>
<tr>
<td>B: 8</td>
</tr>
</tbody>
</table>

| Log |
One “complication”

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

log entry:

<table>
<thead>
<tr>
<th>Memory</th>
<th>DB</th>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: $16</td>
<td></td>
<td>&lt;T,start&gt;</td>
</tr>
<tr>
<td>B: $16</td>
<td></td>
<td>&lt;T, A, 8&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;T, B, 8&gt;</td>
</tr>
</tbody>
</table>
One “complication”

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

memory

A: 8
B: 8

Log:
<T₁,start>
<T₁, A, 8>
<T₁, B, 8>
<T₁, commit>

A: 8
B: 8

DB

Log

BAD STATE

# 2
**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
    $< T, \text{start} >$ in log, but no
    $< T, \text{commit} >$ (or $< T, \text{abort} >$) record in log
(2) For each $< T, X, v >$ in log,
    in reverse order (latest $\rightarrow$ earliest) do:
    - if $T \in S$ then
      - write $(X, v)$
      - output $(X)$
(3) For each $T \in S$ do
    - write $< T, \text{abort} >$ 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)

\[ T:\ \text{Read}(A,t); \ t \leftarrow t \times 2; \ \text{write} \ (A,t); \]
\[ \text{Read}(B,t); \ t \leftarrow t \times 2; \ \text{write} \ (B,t); \]
\[ \text{Output}(A); \ \text{Output}(B) \]

memory

A: 8
B: 8

DB

A: 8
B: 8

LOG
Redo logging (deferred modification)

T: Read(A,t); t ← t × 2; write (A,t); Read(B,t); t ← t × 2; write (B,t); Output(A); Output(B)

A: 8
B: 8

memory

A: 8
B: 8

DB

LOG

<T, start>
Redo logging  (deferred modification)

T:  Read(A,t); t ← t×2; write (A,t);
    Read(B,t); t ← t×2; write (B,t);
    Output(A); Output(B)

memory

A: 8
B: 8

DB

LOG

<T, start>
Redo logging (deferred modification)

T: Read(A,t); t ← t×2; write (A,t);
   Read(B,t); t ← t×2; write (B,t);
   Output(A); Output(B)

memory

A: 8
B: 8

DB

A: 8
B: 8

LOG

<T, start>
<T, A, 16>
Redo logging (deferred modification)

T: Read(A,t); t ← t × 2; write (A,t);
Read(B,t); t ← t × 2; write (B,t);
Output(A); Output(B)

A: 8
B: 8

memory

A: 8
B: 8

DB

<T, start>
<T, A, 16>

LOG
Redo logging (deferred modification)

T:

Read(A,t); t ← t×2; write (A,t);
Read(B,t); t ← t×2; write (B,t);
Output(A); Output(B)

A: 8  16
B: 8  16

memory

A: 8
B: 8

DB

LOG

<T, start>
<T, A, 16>
<T, B, 16>
Redo logging (deferred modification)

T: \[\text{Read}(A,t); \ t \leftarrow t \times 2; \ \text{write} \ (A,t); \]
\[\text{Read}(B,t); \ t \leftarrow t \times 2; \ \text{write} \ (B,t); \]
\[\text{Output}(A); \ \text{Output}(B)\]
Redo logging (deferred modification)

T: 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 to Disk
Recovery rules: Redo logging

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

(2) For each $<T, X, v>$ in log, in forward order (earliest $\rightarrow$ latest) do:
   - if $T \in S$ then
     \[
     \begin{cases} 
     \text{Write}(X, v) \\
     \text{Output}(X) 
     \end{cases}
     \]
Failure Recovery

Checkpointing
Undo/Redo Logging
Undo/Redo Logging
Key drawbacks of Undo logging and Redo logging:

- **Undo logging**: data must be written to disk immediately after a transaction finishes, which can increase number of disk I/O's

- **Redo logging**: need to keep all modified blocks in memory until transaction commits and log is flushed, which can increase the number of buffers required
Solution: undo/redo logging!

Update record in the log has the format

\(<T, X, \text{new } X \text{ val}, \text{old } X \text{ val}>\)
Rules

- Buffer containing X can be flushed to disk either before or after T commits
- Log record must be flushed to disk before corresponding updated buffer is (WAL, or “write after logging”)
Recovery with Undo/Redo Logging

1. Redo all committed transactions in order from earliest to latest
   - handles committed transactions with some changes not yet on disk
2. Undo all incomplete transactions in order from latest to earliest
   - handles uncommitted transactions with some changes already on disk
Checkpoint
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):**

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T1,A,16</td>
<td>T1,commit</td>
<td>Checkpoint</td>
<td>T2,B,17</td>
<td>T2,commit</td>
<td>T3,C,21</td>
</tr>
</tbody>
</table>

Crash

Start from last checkpoint and move forward in the log file redoing updates for committed transactions. (Similarly for Undo log: move backward to last checkpoint.)
Non-quiescent Checkpoint

- Simple checkpointing scheme requires system to "quiesce" (reach a point with no active transactions), ensured by preventing new transactions from starting for a while.
- Avoid this behavior with non-quiescent checkpointing:
  - write a "start checkpoint" record to the log
  - later write an "end checkpoint" record to the log
- Details vary depending on whether undo, redo, or undo/redo logging.
Non-quiescent Checkpoint for Undo/Redo

- write "start checkpoint" listing all active transactions to log
- flush log to disk
- write to disk all dirty buffers (contain a changed DB element), whether or not transaction has committed (this way we move data from memory to disk as much as we can)
- write "end checkpoint" to log
Non-quiescent checkpoint for undo/redo

LOG

... start ckpt active T's: T1, T2, ...

... end ckpt ...

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

LOG

...  T₁₋a  ...  Ckpt-s  T₁  ...  Ckpt  end  ...  T₁₋b

➽ Undo T₁ (undo a,b)

no T₁ commit
Example

Redo T1: (redo b,c)
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?

- log
- db dump
- last needed undo
- checkpoint

- not needed for media recovery
- not needed for undo after system failure
- not needed for redo after system failure

(time)
Summary

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