Transactions and Failure Recovery 2

Instructor: Matei Zaharia
CustomTable Showcase
Outline

Recap from last time
Redo logging
Undo/redo logging
External actions
Media failures
Outline

Recap from last time

Redo logging

Undo/redo logging

External actions

Media failures
Defining Correctness

**Constraint:** Boolean predicate about DB state (both logical & physical data structures)

**Consistent DB:** satisfies all constraints
Transaction: Collection of Actions that Preserve Consistency
Our Failure Model

Fail-stop failures of CPU & memory, but disk survives
Undo Logging  (Immediate modification)

$T_1$: Read $(A,t)$;  $t \leftarrow t \times 2$  \hspace{1cm} A=B
Write $(A,t)$;
Read $(B,t)$;  $t \leftarrow t \times 2$
Write $(B,t)$;
Output $(A)$;
Output $(B)$;

A:8  \hspace{1cm} A:8
B:8  \hspace{1cm} B:8

memory  \hspace{2cm} disk  \hspace{2cm} log
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  B:8
A:8  B:8

memory  disk  log

\(<T₁, \text{start}> \)
\(<T₁, A, 8> \)
**Undo Logging** (Immediate modification)

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

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

     disk
   A:8   16
   B:8

     log
  <T1, start>
  <T1, A, 8>
  <T1, B, 8>
```
Undo Logging  
(Immediate modification)

$T_1$: Read $(A,t)$; $t \leftarrow t \times 2$ \hspace{1cm} $A=B$
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  16
B: 8  16
```
```
log
<T1, start>
<T1, A, 8>
<T1, B, 8>
```
**Undo Logging**  (Immediate modification)

\( T_1: \) 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  

memory  

A:8  
B:8  

disk  

log  

\( <T_1, \text{start}> \)
\( <T_1, \text{A, 8}> \)
\( <T_1, \text{B, 8}> \)
\( <T_1, \text{commit}> \)
Redo Logging (deferred modification)

T1: Read(A,t); t ← t×2; write (A,t);
    Read(B,t); t ← t×2; write (B,t);
    Output(A); Output(B)
Redo Logging (deferred modification)

T1: Read(A,t); t ← t×2; write (A,t); Read(B,t); t ← t×2; write (B,t); Output(A); Output(B)
Redo Logging (deferred modification)

T1: Read(A,t); t ← t×2; write (A,t);
    Read(B,t); t ← t×2; write (B,t);
    Output(A); Output(B)
Redo Logging (deferred modification)

T1: Read(A,t); t ← t×2; write (A,t);
   Read(B,t); 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 (in DB), all log records for transaction that modified X (including commit) must be on disk

3. Flush log at commit

4. Write END record after DB updates are flushed to disk
Recovery Rules: Redo Logging

(1) Let $S$ = set of transactions with $<Ti, \text{commit}>$ and no $<Ti, \text{end}>$ in log.

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

(3) For each $Ti \in S$, write $<Ti, \text{end}>$.
Combining <Ti, end> Records

Want to delay DB flushes for hot objects

Say X is branch balance:
T1: ... update X...
T2: ... update X...
T3: ... update X...
T4: ... update X...

Actions:
write X
output X
write X
output X
write X
output X
write X
output X
Combining \(<Ti, \text{end}>\) Records

Want to delay DB flushes for hot objects

Say X is branch balance:

T1: ... update X...
T2: ... update X...
T3: ... update X...
T4: ... update X...

Actions:
- write X
- output X
- write X
- output X
- write X
- output X
- write X
- output X

combined \(<\text{end}>\) record (checkpoint)
Solution: Checkpoints

Simple, naïve checkpoint algorithm:
1. Stop accepting 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
Redo Logging: What To Do at Recovery?

Redo log (disk):

\[
\begin{array}{cccccccc}
\text{...} & \langle T_1, A, 16 \rangle & \ldots & \langle T_1, \text{commit} \rangle & \ldots & \langle \text{checkpoint} \rangle & \ldots & \langle T_2, B, 17 \rangle & \ldots & \langle T_2, \text{commit} \rangle & \ldots & \langle T_3, C, 21 \rangle & \text{Crash} \\
\end{array}
\]
Redo Logging:
What To Do at Recovery?

Redo log (disk):

T2 committed, so
REDO all its updates
Redo Logging: What To Do at Recovery?

Redo log (disk):

| ... | <T1,A,16> | ... | <T1,commit> | ... | <checkpoint> | ... | <T2,B,17> | ... | <T2,commit> | ... | <T3,C,21> | ... | Crash |

- T2 committed, so REDO all its updates
- T3 didn’t commit, so ignore it
Problems with Ideas So Far

**Undo logging:** need to wait for lots of I/O to commit; can’t easily have backup copies of DB

**Redo logging:** need to keep all modified blocks in memory until commit
Solution: Undo/Redo Logging!

Update = <Ti, X, new X val, old X val>

(X is the object updated)
Undo/Redo Logging Rules

Object X can be flushed before or after Ti commits

Log record (with undo/redo info) must be flushed before corresponding data (WAL)

Flush log up to commit record at Ti commit
## Undo/Redo Logging: What to Do at Recovery?

### Undo/redo log (disk):

<p>| | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>&lt;commit&gt;</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>&lt;T2, C, 30, 38&gt;</td>
<td>...</td>
<td>&lt;T2, D, 40, 41&gt;</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>&lt;T1, A, 10, 15&gt;</td>
<td>...</td>
<td>&lt;T1, B, 20, 23&gt;</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Crash
**Undo/Redo Logging: What to Do at Recovery?**

**Undo/redo log (disk):**

| ... | <checkpoint> | ... | <T1, A, 10, 15> | ... | <T1, B, 20, 23> | ... | <T1, commit> | ... | <T2, C, 30, 38> | ... | <T2, D, 40, 41> | ... | Crash |

T1 committed, so REDO all its updates
Undo/Redo Logging: What to Do at Recovery?

Undo/redo log (disk):

T1 committed, so REDO all its updates

T2 didn’t commit, so UNDO all its updates
Non-Quiescent Checkpoints

LOG

\[ \text{Start-ckpt} \]
\[ \text{active txs: T1, T2,...} \]

\[ \text{end ckpt} \]

\[ \text{for dirty memory undo} \]
\[ \text{pages flushed} \]
Non-Quiescent Checkpoints

checkpoint process:

for i := 1 to M do
  Output(buffer i)

[transactions run concurrently]
Example 1: How to Recover?

| LOG | ... | T1,-a | ... | Ckpt T1 | ... | Ckpt end | ... | T1,-b |

no T1 commit
Example 1: How to Recover?

<table>
<thead>
<tr>
<th>LOG</th>
<th>...</th>
<th>T1, a</th>
<th>...</th>
<th>Ckpt T1</th>
<th>...</th>
<th>Ckpt end</th>
<th>...</th>
<th>T1, b</th>
</tr>
</thead>
</table>

no T1 commit

Undo T1  (undo a,b)
Example 2: How to Recover?

<table>
<thead>
<tr>
<th>LOG</th>
<th>T1a</th>
<th>ckpt-s</th>
<th>T1b</th>
<th>ckpt-end</th>
<th>T1c</th>
<th>T1cmt</th>
</tr>
</thead>
</table>

...
Example 2: How to Recover?

Redo T1 (redo b,c)
**What if a Checkpoint Did Not Complete?**

| LOG | ... | ckpt start | ... | ckpt end | ... | T1 b | ... | ckpt-start | ... | T1 c | ... |

Start recovery from last complete checkpoint
Undo/Redo Recovery Algorithm

Backward pass (end of log → latest valid 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’s active list) - S

Forward pass (latest checkpoint start → end of log)
  » redo actions of all transactions in S
Outline

Recap from last time

Redo logging

Undo/redo logging

External actions

Media failures
External Actions

E.g., dispense cash at ATM

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

(1) Execute real-world actions after commit

(2) Try to make idempotent
Solution

(1) Execute real-world actions after commit

(2) Try to make idempotent

![Diagram]

```
Give $$(\text{amt}, \text{Tid}, \text{time})$$
```

```
ATM

| lastTid: |  
| time:   |  

give(\text{amt})
```

```
$\$
```

CS 245
How Would You Handle These Other External Actions?

- Charge a customer’s credit card
- Cancel someone’s hotel room
- Send data into a streaming system
Outline

Recap from last time

Undo/redo logging

External actions

Media failures
Media Failure
(Loss of Nonvolatile Storage)

A: 16
Media Failure
(Loss of Nonvolatile Storage)

Solution: Make copies of data!
**Naïve Way: Redundant Storage**

Example: keep 3 copies on separate disks

Output(X) → three outputs

Input(X) → three inputs + vote
Better Way: Log-Based Backup

If active database is lost,
- restore active database from backup
- bring up-to-date using redo entries in log
Backup Database

Just like a checkpoint, except that we write the full database

create backup database:

for i := 1 to DB_Size do
  [read DB block i; write to backup]

[transactions run concurrently]
Backup Database

Just like a checkpoint, except that we write the full database

create backup database:

for i := 1 to DB_Size do
    [read DB block i; write to backup]

[transactions run concurrently]

Restore from backup DB and log:
Similar to recovery from checkpoint and log
When Can Logs Be Discarded?

- **log**
  - last needed undo
  - DB dump
  - last needed undo
  - check-point

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

Consistency of data: maintain constraints

One source of problems: failures
  » Logging
  » Redundancy

Another source of problems: data sharing
  » We’ll cover this next!