Recovery

CS 1270
Why Recovery?

- Atomicity: All actions in the Xaction happen, or none happen.
- Consistency: If each Xaction is consistent, and the DB starts consistent, it ends up consistent.
- Isolation: Execution of one Xaction is isolated from that of other Xacts.
- Durability: If a Xaction commits, its effects persist.

- CC guarantees Isolation and Consistency.
- The Recovery Manager guarantees Atomicity & Durability.

(not committed tx should be undone, committed tx should be redone)
Recovery algorithms have two parts

- Actions taken during normal transaction processing to ensure enough information exists to recover from failures
- Actions taken after a failure to recover the database contents to a state that ensures atomicity and durability
Log-based Recovery

Log records for transaction Ti:
<Ti start >
<Ti , X, V1, V2>
<Ti commit >

Two approaches using logs
Deferred database modification / Immediate database modification
Log Examples

<table>
<thead>
<tr>
<th>Transaction T1</th>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read(A)</td>
<td>&lt;T1, start&gt;</td>
</tr>
<tr>
<td>A = A - 50</td>
<td>&lt;T1, A, 1000, 950&gt;</td>
</tr>
<tr>
<td>Write(A)</td>
<td>&lt;T1, B, 2000, 2050&gt;</td>
</tr>
<tr>
<td>Read(B)</td>
<td>&lt;T1, commit&gt;</td>
</tr>
<tr>
<td>B = B + 50</td>
<td></td>
</tr>
<tr>
<td>Write(B)</td>
<td></td>
</tr>
</tbody>
</table>
Deferred Database Modification

- All logs are written on to the stable storage and the database is updated only when a transaction commits
- `<Ti start> ...... < Ti commit >`
- Redo: if both `<Ti start >` and `< Ti commit>` are there in the log.
Immediate Database Modification

- Database updates of an uncommitted transaction are allowed
- Log records must be of the form: \(<Ti, X, Vold, Vnew >\)
- Log record must be written before database item is written
- Output of DB blocks can occur before or after commit in any order
Immediate Database Modification

Assume serial transaction (no different tx interleaving)

Undo: if <Ti, start> is in the log but <Ti commit> is not.  
How: restore the value of all data items updated by Ti to their old values,
going backwards from the last log record for Ti

Redo: if <Ti start> and <Ti commit> are both in the log.  
How: sets the value of all data items updated by Ti to the new values, going 
forward from the first log record for Ti

TL;DR => not committed tx should be undone; committed tx should be redone
## Practice

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;T0, <strong>start</strong>&gt;</td>
<td>&lt;T0, <strong>start</strong>&gt;</td>
<td>&lt;T0, <strong>start</strong>&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;T0, A, 1000, 950&gt;</td>
<td>&lt;T0, A, 1000, 950&gt;</td>
<td>&lt;T0, A, 1000, 950&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;T0, B, 2000, 2050&gt;</td>
<td>&lt;T0, B, 2000, 2050&gt;</td>
<td>&lt;T0, B, 2000, 2050&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;T0, <strong>commit</strong>&gt;</td>
<td>&lt;T0, <strong>commit</strong>&gt;</td>
<td>&lt;T0, <strong>commit</strong>&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;T1, <strong>start</strong>&gt;</td>
<td>&lt;T1, <strong>start</strong>&gt;</td>
<td>&lt;T1, <strong>start</strong>&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;T1, C, 700, 600&gt;</td>
<td>&lt;T1, C, 700, 600&gt;</td>
<td>&lt;T1, C, 700, 600&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;T1, <strong>commit</strong>&gt;</td>
<td>&lt;T1, <strong>commit</strong>&gt;</td>
<td>&lt;T1, <strong>commit</strong>&gt;</td>
<td></td>
</tr>
</tbody>
</table>
### Practice

<table>
<thead>
<tr>
<th>T0, start</th>
<th>T0, start</th>
<th>T0, start</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;T0, start&gt;</td>
<td>&lt;T0, start&gt;</td>
<td>&lt;T0, start&gt;</td>
</tr>
<tr>
<td>&lt;T0, A, 1000, 950&gt;</td>
<td>&lt;T0, A, 1000, 950&gt;</td>
<td>&lt;T0, A, 1000, 950&gt;</td>
</tr>
<tr>
<td>&lt;T0, B, 2000, 2050&gt;</td>
<td>&lt;T0, B, 2000, 2050&gt;</td>
<td>&lt;T0, B, 2000, 2050&gt;</td>
</tr>
<tr>
<td>&lt;T0, commit&gt;</td>
<td>&lt;T0, commit&gt;</td>
<td>&lt;T0, commit&gt;</td>
</tr>
<tr>
<td>T1, start</td>
<td>T1, start</td>
<td>T1, start</td>
</tr>
<tr>
<td>&lt;T1, C, 700, 600&gt;</td>
<td>&lt;T1, C, 700, 600&gt;</td>
<td>&lt;T1, C, 700, 600&gt;</td>
</tr>
<tr>
<td>&lt;T1, commit&gt;</td>
<td>&lt;T1, commit&gt;</td>
<td>&lt;T1, commit&gt;</td>
</tr>
</tbody>
</table>

(a) **undo** (T0): B is restored to 2000 and A to 1000.

(b) **undo** (T1) and **redo** (T0): C is restored to 700, and then A and B are set to 950 and 2050 respectively.

(c) **redo** (T0) and **redo** (T1): A and B are set to 950 and 2050 respectively. Then C is set to 600.
Checkpoint

- Objective: avoid redundant redo operations
- How: Put marks in the log indicating that at that point DB and log are consistent.

- \( T_1 \) can be ignored (updates already output to disk due to checkpoint)
- \( T_2 \) and \( T_3 \) redone. (already committed)
- \( T_4 \) undone
Recovery With Concurrent Transactions

<checkpoint L> L: the list of transactions active at the time of the checkpoint

Recovery for concurrent transactions, 3 phases:
First phase:
1. Initialize undo-list and redo-list to empty
2. Scan the log backwards from the end, stopping when the first <checkpoint L> record is found.
3. For each record found during the backward scan:
   if the record is <Ti commit>, add Ti to redo-list
   if the record is <Ti start>, then if Ti is not in redo-list, add Ti to undo-list
4. For every Ti in L, if Ti is not in redo-list, add Ti to undo-list
Recovery With Concurrent Transactions

Recovery for concurrent transactions, 3 phases:
Second phase: **Undo** phase
  Scan log backwards
  - Perform undo(T) for every transaction in undo-list
  - Stop when you have seen <T, start> for every T in **undo-list**.

Third phase: **Redo** phase
  Locate the most recent <checkpoint L> record.
  - Scan log forwards from the <checkpoint L> record till the end of the log.
  - Perform redo for each log record that belongs to a transaction on the **redo-list**
Practice

LOG:
<T0 start>
<T0, A, 0, 10>
<T0 commit>
<T1 start>
<T1, B, 0, 10>
<T2 start>
<T2, C, 0, 10>
<T2, C, 10, 20>
<checkpoint {T1, T2}>
<T3 start>
<T3, A, 10, 20>
<T3, D, 0, 10>
<T3 commit>
<table>
<thead>
<tr>
<th>LOG:</th>
<th>Undo-list: T1, T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;T0 start&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;T0, A, 0, 10&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;T0 commit&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;T1 start&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;T1, B, 0, 10&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;T2 start&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;T2, C, 0, 10&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;T2, C, 10, 20&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;checkpoint {T1, T2}&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;T3 start&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;T3, A, 10, 20&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;T3, D, 0, 10&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;T3 commit&gt;</td>
<td></td>
</tr>
</tbody>
</table>

Redo-list: T3
Practice

LOG:
<T0 start>
<T0, A, 0, 10>
<T0 commit>
<T1 start>
<T1, B, 0, 10>
<T2 start>
<T2, C, 0, 10>
<T2, C, 10, 20>
<checkpoint {T1, T2}>
<T3 start>
<T3, A, 10, 20>
<T3, D, 0, 10>
<T3 commit>

Undo-list: T1, T2
Redo-list: T3

Undo:
Set C to 10
Set C to 0
Set B to 0
Practice

LOG:
<T0 start>
<T0, A, 0, 10>
<T0 commit>
<T1 start>
<T1, B, 0, 10>
<T2 start>
<T2, C, 0, 10>
<T2, C, 10, 20>
<checkpoint {T1, T2}>
<T3 start>
<T3, A, 10, 20>
<T3, D, 0, 10>
<T3 commit>

Undo-list: T1, T2
Redo-list: T3

Undo:
Set C to 10
Set C to 0
Set B to 0

Redo:
Set A to 20
Set D to 10
Practice

LOG:
<T0 start>
<T0, A, 0, 10>
<T0 commit>
<T1 start>
<T1, B, 0, 10>
<T2 start>
<T2, C, 0, 10>
<T2, C, 10, 20>
<checkpoint {T1, T2}>
<T3 start>
<T3, A, 10, 20>
<T3, D, 0, 10>
<T3 commit>

Undo-list: T1, T2
Redo-list: T3

Undo:
Set C to 10
Set C to 0
Set B to 0

Redo:
Set A to 20
Set D to 10

After recovery:
A: 20, B: 0, C: 0, D: 10