Recovery Theory

Storage Types

- **Volatile storage**
  - main memory, which does not survive crashes.

- **Non-volatile storage**
  - tape, disk, which survive crashes.

- **Stable storage**
  - information in stable storage is "never" lost.
  - There is no such physical medium; it is an approximation that is implemented.

Failure Types

- **Program Failures**
  - logical errors, bad input, unavailable data, user cancellation
  - resource limits

- **System Failures**
  - computer hardware malfunction, power failures
  - bugs in O.S, operator error

- **Media Failures**
  - disk head crash, data transfer error,
  - disk controller failure

- **Unrecoverable errors**
  - failure to make archive dumps
  - destruction of archives

Theory of Recovery

The goals of the recovery system are:

- When a transaction $T$ **commits**
  - Make the updates permanent in the database so that they can survive subsequent failures.

- When a transaction $T$ **aborts**
  - Obliterate any updates on data items by aborted transactions in the database.
  - Obliterate the effects of $T$ on other transactions; i.e., transactions that read data items updated by $T$.

- When the system **crashes** after a system or media failure
  - Bring the database to its most recent consistent state.
Recovery Actions

- Recovery protocols implement two actions:
  - **Undo** action: required for atomicity.
    Undo all updates on the stable storage by an uncommitted transaction.
  - **Redo** action: required for durability
    Redoes the update (on the stable storage) of committed transaction.

Recovering from Failures

- **Program Failures**
  - **Transaction Undo**
    - Removes all the updates of the aborted transaction
    - Does not affect any other transaction

- **System Failures**
  - **Global Undo**
  - **Partial Redo**
    - Effects of committed transactions are reflected in the database

- **Media Failures**
  - **Global Redo**

Cascading Aborts

- Consider the execution:
  - \( w_1(x) \) \( r_2(x) \)
    - If \( T_1 \) aborts, \( T_2 \) must also abort.
    - \( T_2 \) has an **abort dependency** on \( T_1 \).
- In general, any transaction that reads data items updated (written) by a transaction that aborts must also be aborted.
- What will happen if \( T_2 \) is committed before \( T_1 \) is aborted?
  - \( w_1(x) \) \( r_2(x) \) \( c_2 \) \( \alpha_1 \)
    - The system cannot abort \( T_2 \) without violating the semantics of commit operations.

Recoverable Executions (RC)

- To prevent unrecoverable situations the TM must keep full track of read/write operations and delay commit requests of transactions.
- **Definition:**
  A transaction \( T_m \) **reads** \( x \) from transaction \( T_n \) in an execution if
  - \( \bowtie \ T_m \) reads \( x \) after \( T_n \) has written into it
  - \( \bowtie \ T_m \) does not abort before \( T_m \) reads \( x \) and
  - \( \forall T_k: W_{Tk}(x) \) occurred between \( W_{Tn}(x) \) and \( R_{Tm}(x) \),
    \( \alpha_{Tk} \) precedes \( R_{Tm}(x) \).
Recoverable Executions (RC) ...

- **Definition:**
  An execution is recoverable (RC) if for every transaction $T_n$ commits, $T_n$'s commit follows the commitment of every transaction $T_m$ from which $T_n$ reads.

- **RULE 0:**
  Delay the commit of a transaction that reads uncommitted data.

Effects of Cascading Aborts

- Significant bookkeeping of who updated what and who read what is required.
- Transactions may be forced to abort because some other transaction happened to abort and all the effects of the aborted transaction need to be undone (isolation ?).
- Significant amount of computation may be lost due to cascading aborts.
- In practice, most DBMS are designed to avoid cascading aborts.

Avoiding Cascading Aborts (ACA)

- **Definition:**
  An execution avoids cascading aborts (ACA) if whenever a transaction $T_n$ reads data updated by $T_m$, $T_m$ has already committed.

- That is it ensures that every transaction reads only those values there were written by committed transactions.
- This means the DBMS must delay each $r(x)$ until all transactions that previously issued a $w(x)$ have either aborted or committed.

- **RULE 1:** Do not permit reading of uncommitted data.
  - Note rule 1 is stronger than Rule 0 (the necessary condition for recoverability).

Undoing Writes

- **Assume**
  - Database = $\{ x, y \}$ with initial values $x = 1, y = 0$
  - Transactions:
    - T1: write($x$, 2); write($y$, 3); abort
    - T2: write($x$, 8); write($y$, 9); abort
An interleaved execution

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>before image of</td>
<td></td>
</tr>
<tr>
<td>write(x, 8)</td>
<td>x = 1</td>
</tr>
<tr>
<td>write(y, 9)</td>
<td>y = 0</td>
</tr>
<tr>
<td>write(x, 2)</td>
<td>x = 8</td>
</tr>
<tr>
<td>abort</td>
<td></td>
</tr>
<tr>
<td>write(y, 3)</td>
<td>y = 0</td>
</tr>
<tr>
<td>abort</td>
<td></td>
</tr>
</tbody>
</table>

- when T2 aborts
  - x = before image of write(x, 8) \( \Rightarrow x = 1 \)
  - y = before image of write(y, 9) \( \Rightarrow y = 0 \)

- when T1 aborts
  - x = before image of write(x, 2) \( \Rightarrow x = 8 \)
  - y = before image of write(y, 3) \( \Rightarrow y = 0 \)

The Lost Update Problem

Assume
Database = \{x, y\}
initially x = 1, y = 0

Transactions:
T1: write(x, 2); write(y, 3); abort
T2: write(x, 8); write(y, 9); commit

Consider the following execution
\( w_1(x, 2); w_2(x, 8); w_2(y, 9); c_2; w_1(y, 3); \alpha_1 \)

What is the state of the database after this execution?

Strict Executions

- To solve the undoing writes problem, we must delay the execution of a write(x, val) operation until the transaction that has previously written x terminates, i.e., commits or aborts.

  - **Definitions:**
    - An execution is strict (ST) if it avoids cascading aborts and overwriting of uncommitted data; i.e., it is ACA and RC.

  - That is, a transaction \( T_n \) can read or write a data item updated (written) by \( T_m \) only after \( T_m \) commits or aborts.

  - **RULE 2:** Do not permit overwriting of uncommitted data.

Recovery Correctness Criteria

RC ⊃ ACA ⊃ ST

All Histories

RC

ACA

ST
Reliability and Serializability

Diagram showing relationships between different serializability conditions.