Chapter 15: Recovery System

- Advanced Recovery Techniques
- Failure with Loss of Nonvolatile Storage
- Buffer Management
- Recovery with Concurrent Transactions
- Shadow Paging
- Log-Based Recovery
- Recovery and Atomicity
- Storage Structure
- Failure Classification
Failure Classification

- Disk failure: a head crash of similar failure destroys all or part of disk storage
- System crash: a power failure or other hardware failure causes the system to crash
- System crash: an error condition due to an error condition (e.g., deadlock)
- System error: the database system must terminate an active transaction due to an error condition
- Logical error: transaction cannot complete due to some
nonvolatile media

approximated by maintaining multiple copies on distinct

a mythical form of storage that survives all failures

Stable storage:

• examples: disk, tape

survives system crashes

Nonvolatile storage:

• examples: main memory, cache memory

does not survive system crashes

Volatile storage:

Storage Structure
1. Write the information onto the first physical block.
2. When the first write successfully completes, write the same information onto the second physical block.
3. The output is completed only after the second write successfully completes.

Execute output operation as follows:

- Protect storage medium from failure during data transfer, system block.
replaces the appropriate physical block there. The output
transfers the buffer block to the disk, and
input transfers the physical block to main memory.

through the following two operations:

Block movements between disk and main memory are initiated

blocks are the blocks residing temporarily in main memory.

physical blocks are those blocks residing on the disk’ buffer.

Data Access
Transfer of data between the database and program

**Data Transfer**

- The transfer of data between the database and program

Assign the value of x to X in buffer B.

\[ \text{assign}(X, B) \]

If block B is not in main memory, then issue X in the buffer block.

\[ \text{write}(X) \]

- Assign the value of local variable x to data item.

\[ \text{read}(X) \]

- Assigns the value of data item X to the local variable x.

Variables is accomplished using:

\[ \text{read}(X) \]
Example of Data Transfer and Access
• Comment: a data "item" can be a file, relation, record, physical page, etc. The choice is up to the designer.

• If the transaction reads a value from the data base and this value was previously modified, it always gets the latest value.

• Sequential order that the write instructions are issued into the global buffer are in the same transaction. It can be modified many times in the local buffer.

• Each data item can be read and written only once by one single transaction.

Assumptions
Recovery and Atomicity

To ensure atomicity, first output information describing the modifications to stable storage without modifying the database.

- To ensure atomicity, first output information describing the modifications to stable storage without modifying the database.

- If performed multiple database modifications, several output

- If performed multiple database modifications, several output

- If performed by $L$ or none at all.

- If performed $L$ to account B, goal is either to perform all database modifications

- If performed $L$ to account B, goal is either to perform all database modifications

- For transaction $L$ that transfers $50 from account A to account

- For transaction $L$ that transfers $50 from account A to account

- Will commit may leave the database in an inconsistent state.

- Will commit may leave the database in an inconsistent state.
are written sequentially on the log, and then the write (X) is executed. 

\[
\begin{align*}
\text{new value} & \quad (e.g., V_2) \\
\text{old value} & \quad (e.g., V_1) \\
\text{data item name} & \quad (i.e., X) \\
\text{transaction name} & \quad (i.e., T) 
\end{align*}
\]

Whenever \( T \) executes write \((X)\), the fields:

- \(<\text{start } T>\) writing

When transaction \( T \) starts, it registers itself on the log by

- \(\) a log file is kept on stable storage

Log-Based Recovery
be in stable storage

Before L is committed, all its corresponding log records must
actually written on the database
first actually written on the log (stable storage) and then
If X is modified then its corresponding log record is always

commit L is added to the Log
When L reaches its last statement, the record

Log-Based Recovery (Cont.)
\[(\forall) \text{write} \]
\[
\exists + \mathcal{B} =: \mathcal{B} \]
\[
(\forall) \text{write} \]
\[
\mathcal{B} + \exists =: \exists \]
\[
(\mathcal{B}) \text{read} \]
\[
(\exists) \text{read} \]
\[
10 - \mathcal{B} =: \mathcal{B} \]
\[
(\mathcal{B}) \text{read} \]
\[
(\exists) \text{write} \]
\[
10 + \mathcal{V} =: \mathcal{V} \]
\[
(\mathcal{V}) \text{read} \]
\[
\mathcal{V} \in \mathcal{L}^2 \]
\[
\mathcal{V} \text{write} \]
\[
\mathcal{C} + \mathcal{B} + \mathcal{V} =: \mathcal{V} \]
\[
(\mathcal{C}) \text{write} \]
\[
\mathcal{C} =: \mathcal{C} \]
\[
(\mathcal{C}) \text{read} \]
\[
\mathcal{B} \text{write} \]
\[
00 + \mathcal{B} =: \mathcal{B} \]
\[
(\mathcal{B}) \text{read} \]
\[
00 + \mathcal{V} =: \mathcal{V} \]
\[
(\mathcal{V}) \text{read} \]
\[
\mathcal{L}^1 \]

\[
0 = \exists, \mathcal{L} = 60, \mathcal{B} = 300, \mathcal{C} = 5, \mathcal{V} = 80 \]

Sequentially by the system, and with initial values of \( \mathcal{V} = 100, \)

Consider transactions \( \mathcal{L}^1 \) and \( \mathcal{L}^2 \) which are executed.

Example of Recovery
Example of Recovery (Cont.)
Example of Recovery (Cont.)

The order of actual writes to log and database might be:

<table>
<thead>
<tr>
<th>Time</th>
<th>Log</th>
<th>Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

IA IV III II I
Consequence: L\textsubscript{1} ran
\texttt{redo L}\textsubscript{2} has not run
Consequence: L\textsubscript{1} ran,
\texttt{undo L}\textsubscript{1} has not run
modified by L\textsubscript{1} to the values created by L\textsubscript{1}
\texttt{redo L}\textsubscript{1} set the values of the variables
modified by L\textsubscript{1} to old values
\texttt{undo L}\textsubscript{1} restore the values of the variables
nothing

<table>
<thead>
<tr>
<th>Action</th>
<th>Last instruction (I)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 (\geq) I (\geq) 3</td>
<td></td>
</tr>
<tr>
<td>4 (\geq) I (\geq) 1</td>
<td></td>
</tr>
<tr>
<td>0 = I</td>
<td></td>
</tr>
</tbody>
</table>

Example of Recovery (Cont.)
In this algorithm, a large number of transactions need to be undone since we do not know how far behind the database recovery must still give correct results. If a system crashes during the recovery stage, the new transaction may need to be undone.

Remarks:

- **commit**
  - undo all transactions for which the log has *start* but no
  - redo all transactions for which the log has both *start* and

**The Algorithm**
Redo all transactions that have committed after checkpoint.

- Undo all transactions that have not committed.

During recovery:

1. Output all log records currently residing in main memory.

2. Output all modified buffer blocks to the disk.

3. Output a log record <checkpoint> onto stable storage.

Checkpointing Streamlining recovery procedure by periodically performing checkpoints.
Example of Checkpoints

- $T_1$
- $T_2$
- $T_3$
- $T_4$

System failure

Checkpoint

- $T_1$
- $T_2$
- $T_3$
- $T_4$

undone

and redone

or

Additional notes:
- $T_L$
- $T_R$
- $T_F$

Time
Shadow Paging

- Alternative to log-based recovery; maintain *two* page tables during the life of a transaction
  
  - *current* page table
  
  - *shadow* page table

- Store the shadow page table in nonvolatile storage; state of the database prior to transaction execution may be recovered.
- Garbage collection
- Data fragmentation

Drawbacks:
- Crashes is faster.
  - Eliminates overhead of log-record output, recovery from table and the next transaction is allowed to begin execution.
- The current page table then becomes the new shadow page written to nonvolatile storage.
- When the transaction commits, the current page table is

Shadow Page (Cont.)
When the system recovers from a crash it constructs two lists:

- **redo-list** consists of transactions to be redone.
- **undo-list** consists of transactions to be undone.

where \( I \) is a list of transactions active at the time of the

\[ \text{checkpoint} \left\langle \right. \] checkpoint

\( \text{checkpoint} \log \) record be of the form

\[ \text{Concurrent transaction-processing} \] Concurrent transaction-processing system requires that the

transactions may have been active at the time of the Last

\[ \text{Log scanning – when transactions execute concurrently, several} \] Log scanning – when transactions execute concurrently, several
transactions on the redo-list are redone.

• After all transactions on undo-list have been undone,
  perform \texttt{redo} for each \texttt{L}\textsubscript{r} on the redo-list.

  3. Scan log forward from checkpoint until the undo-list has been located.
  2. Continue to scan log backward, performing undo for each \texttt{L}\textsubscript{r} on the undo-list.
  1. Rescan log from most recent record backward until the
    recovery proceeds as follows:

    Once the redo-list and undo-list have been constructed,
Buffer Management
operating system may result in extra output of data to disk.

- DB implements its buffer within the virtual memory of the database buffer.
  - database buffer

  - data block transfer; limits amount of main memory
  - reserves part of main memory as a buffer and manages

If the OS cannot enforce output of log records prior to output

- Input block $B_2$ from disk to main memory.
- Output block $B_1$ to disk.
- Output to stable storage all block $B_1$'s log records.

virtual memory)

has been modified, $B_1$ must be output prior to $B_2$'s input

when another block $B_2$ needs to be brought into memory; if $B_1$

Database buffering - Overwrite block $B_1$ in main memory

Buffer Management (Cont.)
Failure with Loss of Nonvolatile Storage

- Output a log record
- Copy the contents of the database to stable storage.
- Output all buffer blocks onto the disk.
- Output all log records currently residing in main memory onto stable storage.

Procedure similar to checkpointing must take place:
- No transaction may be active during the dump procedure;
- Periodically dump the entire content of the database to stable storage.
• Fuzzy checkpointer
• Restart recovery
• Checkpoints
• Transaction rollback
• Logical undo logging

Incomplete transactions using the log, followed by an undo pass on the log to roll back.

When recovering from system failure, perform a redo pass (operation) undo, and follow the principle of repeating history.

Support high-concurrency locking techniques, such as those used for B+ tree concurrency control; based on logical

Advanced Recovery Techniques