Lecture 16:
Recovery

Wednesday, February 15, 2006

Outline

• Checkpointing
• CRedo logging 17.3
• Redo/undo 17.4
Checkpointing

Checkpoint the database periodically
• Stop accepting new transactions
• Wait until all current transactions complete
• Flush log to disk
• Write a <CKPT> log record, flush
• Resume transactions

Undo Recovery with Checkpointing

During recovery, Can stop at first <CKPT>

other transactions

transactions T2,T3,T4,T5

<T9,X9,v9>
<T5,X5,v5>
<T4,X4,v4>
<T3,X3,v3>
<T2,X2,v2>
Nonquiescent Checkpointing

- Problem with checkpointing: database freezes during checkpoint
- Would like to checkpoint while database is operational
- Idea: nonquiescent checkpointing

Quiescent = being quiet, still, or at rest; inactive
Non-quiescent = allowing transactions to be active

Nonquiescent Checkpointing

- Write a `<START CKPT(T1,…,Tk)>` where T1,…,Tk are all active transactions
- Continue normal operation
- When all of T1,…,Tk have completed, write `<END CKPT>`
Undo Recovery with Nonquiescent Checkpointing

During recovery, Can stop at first <CKPT>

Q: why do we need <END CKPT> ?

Redo Logging

Log records
- <START T> = transaction T has begun
- <COMMIT T> = T has committed
- <ABORT T> = T has aborted
- <T,X,v> = T has updated element X, and its new value is v
Redo-Logging Rules

R1: If T modifies X, then both \(<T,X,v>\) and \(<\text{COMMIT } T>\) must be written to disk before OUTPUT(X)

- Hence: OUTPUTs are done *late*
## Recovery with Redo Log

After system’s crash, run recovery manager

- **Step 1.** Decide for each transaction \( T \) whether it is completed or not
  - \(<\text{START } T>\ldots<\text{COMMIT } T>\ldots\) = yes
  - \(<\text{START } T>\ldots<\text{ABORT } T>\ldots\ldots\) = yes
  - \(<\text{START } T>\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\) = no

- **Step 2.** Read log from the beginning, redo all updates of *committed* transactions

### Table Example

<table>
<thead>
<tr>
<th>Action</th>
<th>Mem A</th>
<th>Mem B</th>
<th>Disk A</th>
<th>Disk B</th>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRITE(A,i)</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>(&lt;\text{START } T&gt;)</td>
</tr>
<tr>
<td>( t:=t+2 )</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>(&lt;T,A,16&gt;)</td>
</tr>
<tr>
<td>WRITE(B,i)</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>(&lt;T,B,16&gt;)</td>
</tr>
<tr>
<td>( t:=t+2 )</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>(&lt;\text{COMMIT } T&gt;)</td>
</tr>
</tbody>
</table>

**Disks:**
- Disks A and B

**Actions:**
- READ
- WRITE
- COMMIT

**Memories:**
- Mem A
- Mem B

**Log Entries:**
- \(<\text{START } T>\)
- \(<T,A,16>\)
- \(<T,B,16>\)
- \(<\text{COMMIT } T>\)
Recovery with Redo Log

- \(<\text{START T1}>\)
- \(<T1,X1,v1>\)
- \(<\text{START T2}>\)
- \(<T2,X2,v2>\)
- \(<\text{START T3}>\)
- \(<T1,X3,v3>\)
- \(<\text{COMMIT T2}>\)
- \(<T3,X4,v4>\)
- \(<T1,X5,v5>\)
- ...

Nonquiescent Checkpointing

- Write a \(<\text{START CKPT(T1,...,Tk)>\)}
  where T1,...,Tk are all active transactions
- Flush to disk all blocks of committed transactions (dirty blocks), while continuing normal operation
- When all blocks have been written, write \(<\text{END CKPT}>\)
Redo Recovery with 
Nonquiescent Checkpointing

Step 1: look for 
The last 
<END CKPT>

All OUTPUTs of T1 are 
known to be on disk

Step 2: redo 
from the 
earliest 
start of 
T4, T5, T6 
ignoring 
transactions 
committed 
earlier

Comparison Undo/Redo

- Undo logging:
  - OUTPUT must be done early
  - If <COMMIT T> is seen, T definitely has written all its data to 
    disk (hence, don’t need to redo) – inefficient

- Redo logging
  - OUTPUT must be done late
  - If <COMMIT T> is not seen, T definitely has not written any of its 
    data to disk (hence there is not dirty data on disk, no need to undo) 
    – inflexible

- Would like more flexibility on when to OUTPUT: 
  undo/redo logging (next)
Undo/Redo Logging

Log records, only one change
• \(<T,X,u,v>\) = T has updated element X, its \textit{old} value was u, and its \textit{new} value is v

Undo/Redo-Logging Rule

UR1: If T modifies X, then \(<T,X,u,v>\) must be written to disk before OUTPUT(X)

Note: we are free to OUTPUT early or late relative to \(<\text{COMMIT T}>\)
### Recovery with Undo/Redo Log

After system’s crash, run recovery manager
- Redo all committed transaction, top-down
- Undo all uncommitted transactions, bottom-up

<table>
<thead>
<tr>
<th>Action</th>
<th>T</th>
<th>Mem A</th>
<th>Mem B</th>
<th>Disk A</th>
<th>Disk B</th>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ(A, t)</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>&lt;START T&gt;</td>
</tr>
<tr>
<td>t:=t*2</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>WRITE(A, t)</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>&lt;T,A,8,16&gt;</td>
</tr>
<tr>
<td>READ(B, t)</td>
<td>8</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>t:=t*2</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>WRITE(B, t)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>&lt;T,B,8,16&gt;</td>
</tr>
<tr>
<td>OUTPUT(A)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>&lt;COMMIT T&gt;</td>
</tr>
<tr>
<td>OUTPUT(B)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

Can OUTPUT whenever we want: before/after COMMIT
Recovery with Undo/Redo Log

<START T1>
<T1,X1,v1>
<START T2>
<T2,X2,v2>
<START T3>
<T1,X3,v3>
<COMMIT T2>
<T3,X4,v4>
<T1,X5,v5>
...
...