Database recovery

CS348 Spring 2023
Instructor: Sujaya Maiyya
Sections: 002 & 004 only
Announcements

• Assignment 3 due July 20th

• Final demo for projects:
  • Option 1: Online live demo with the TA
  • Option 2: Send a recording to the TA

• Send your choice to your TA by July 24th
  • Lose 2 points otherwise
Review

• ACID
  • Atomicity: TX’s are either completely done or not done at all
  • Consistency: TX’s should leave the database in a consistent state
  • Isolation: TX’s must behave as if they are executed in isolation
  • Durability: Effects of committed TX’s are resilient against failures

• SQL transactions
  -- Begins implicitly
  SELECT ...;
  UPDATE ...;
  ROLLBACK | COMMIT;
Outline

• Recovery – atomicity and durability
  • Naïve approaches
  • Logging for undo and redo
Execution model

To read/write X

• The disk block containing X must be first brought into memory

• X is read/written in memory

• The memory block containing X, if modified, must be written back (flushed) to disk eventually
Failures

• System crashes right after a transaction $T_1$ commits; but not all effects of $T_1$ were written to disk
  • How do we complete/redo $T_1$ (durability)?

• System crashes in the middle of a transaction $T_2$; partial effects of $T_2$ were written to disk
  • How do we undo $T_2$ (atomicity)?
Naïve approach: Force -- durability

\[ T1 \text{ (balance transfer of $100 from A to B)} \]
read(A, a); \( a = a - 100 \);
write(A, a);
read(B, b); \( b = b + 100 \);
write(B, b);
commit;

**Force:** all writes must be reflected on disk when a transaction commits
Naïve approach: Force -- durability

\( T1 \) (balance transfer of $100 from A to B)

read\((A, a)\); \( a = a - 100 \);
write\((A, a)\);
read\((B, b)\); \( b = b + 100 \);
write\((B, b)\);
commit;

**Force:** all writes must be reflected on disk when a transaction commits

Without force: not all writes are on disk when \( T1 \) commits
If system crashes right after \( T1 \) commits, effects of \( T1 \) will be lost
Naïve approach: No steal -- atomicity

**T1** (balance transfer of $100 from A to B)

read(A, a); a = a – 100;
write(A, a);
read(B, b); b = b + 100;
write(B, b);
commit;

**No steal**: Writes of a transaction can only be flushed to disk at commit time:
- e.g. A=700 cannot be flushed to disk before commit.

With steal: some writes are on disk before T commits
If system crashes before T1 commits, there is no way to undo the changes
Naïve approach

- **Force**: When a transaction commits, all writes of this transaction must be reflected on disk
  - Ensures durability
  - Problem of force: Lots of *random writes* hurt performance

- **No steal**: Writes of a transaction can only be flushed to disk at commit time
  - Ensures atomicity
  - Problem of no steal: Holding on to all dirty blocks requires lots of memory
Logging

- **Database log:** sequence of log records, recording all changes made to the database, written to stable storage (e.g., disk) during normal operation

- Hey, one change turns into two—bad for performance?
  - But writes to log are ***sequential*** (append to the end of log)
Log format

• When a transaction $T_i$ starts
  • $\langle T_i, \text{start} \rangle$

• Record values before and after each modification:
  • $\langle T_i, X, \text{old	extunderscore value	extunderscore of	extunderscore X}, \text{new	extunderscore value	extunderscore of	extunderscore X} \rangle$
  • $T_i$ is transaction id
  • $X$ identifies the data item

• A transaction $T_i$ is committed when its commit log record is written to disk
  • $\langle T_i, \text{commit} \rangle$
When to write log records into stable store?

• **Write-ahead logging (WAL):** Before $X$ is modified on disk, the log record pertaining to $X$ must be flushed.

• Without WAL, the system might crash after $X$ is modified on disk but before its log record is written to disk—no way to undo.
Undo/redo logging example

\(T_1\) (balance transfer of $100 from A to B)

\[
\begin{align*}
\text{read}(A, a); & \quad a = a - 100; \\
\text{write}(A, a); & \\
\text{read}(B, b); & \quad b = b + 100; \\
\text{write}(B, b);
\end{align*}
\]

WAL: Before A,B are modified on disk, their log info must be flushed
Undo/redo logging example cont.

**T1** (balance transfer of $100 from A to B)

read(A, a); a = a - 100;
write(A, a);
read(B, b); b = b + 100;
write(B, b);

Steal: can flush before commit

If system crashes before T1 commits, we have the old value of A stored on the log to undo T1
Undo/redo logging example cont.

$T_1$ (balance transfer of $100 from A to B)

read($A, a$); $a = a - 100$;
write($A, a$);
read($B, b$); $b = b + 100$;
write($B, b$);
commit;

Memory buffer

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>800</td>
</tr>
<tr>
<td>$B$</td>
<td>400</td>
</tr>
</tbody>
</table>

Disk

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>800</td>
</tr>
<tr>
<td>$B$</td>
<td>400</td>
</tr>
</tbody>
</table>

Log

$\langle T_1, \text{start} \rangle$
$\langle T_1, A, 800, 700 \rangle$
$\langle T_1, B, 400, 500 \rangle$
$\langle T_1, \text{commit} \rangle$

No force: can flush after commit

If system crashes before we flush the changes of $A, B$ to the disk, we have their new committed values on the log to redo $T_1$
Log example - redo

- Redo phase:

List of active transactions at crash:
T1  T2  T3

Log

Start of log
redo
redo
redo
redo
redo
redo
redo
redo
End of log

T1, start
T1, x, 99, 100
T2, start
T2, y, 199, 200
T3, start
T3, z, 51, 50
T2, w, 1000, 10
T2, commit
T4, start
T3, z, 51
T3, abort
T4, y, 200, 50
Log example

- Redo phase:

  
<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>100</td>
</tr>
<tr>
<td>y</td>
<td>200</td>
</tr>
<tr>
<td>z</td>
<td>50</td>
</tr>
<tr>
<td>w</td>
<td>1000</td>
</tr>
</tbody>
</table>

List of active transactions at crash:
T1, T2, T3

Log

- Start of log
- Redo
- T1, start
- T1, x, 99, 100
- Redo
- T2, start
- T2, y, 199, 200
- Redo
- T3, start
- T3, z, 51, 50
- Redo
- T2, w, 1000, 10
- Redo
- T2, commit
- T4, start
- T3, z, 51
- Redo
- T3, abort
- T4, y, 200, 50

End of log
Log example

• Redo phase:

List of active transactions at crash: T1 T2 T3 T4
Log example

- **Redo phase:**

  - List of active transactions at crash: T1 T2 T3 T4

  - Start of log:
    - redo T1, start
    - redo T1, x, 99, 100
    - redo T2, start
    - redo T2, y, 199, 200
    - redo T3, start
    - redo T3, z, 51, 50
    - redo T2, w, 1000, 10
    - redo T2, commit
    - redo T4, start
    - redo T3, z, 51
    - redo T3, abort
    - redo T4, y, 200, 50

  - When txn manager receives abort, it logs reverse operations before abort:
    - redo x: 99
    - redo y: 199
    - redo z: 51
    - redo w: 1000

  - End of log:

- Log example: CRASH!
Log example

- Redo phase:

List of active transactions at crash:
T1 T2 T3 T4

End of log

Start of log

Log

T1, start
T1, x, 99, 100
T2, start
T2, y, 199, 200
T3, start
T3, z, 51, 50
T4, start
T4, y, 200, 50

Redo:
T1, 100
T2, 200, 50
T3, 51, 50
T4, 10
T2, commit
Log example - Undo

• Undo phase: T1, T4

List of active transactions at crash:
T1 T2 T3 T4

Start of log

Log

T1, start
T1, x, 99, 100
T2, start
T2, y, 199, 200
T3, start
T3, z, 51, 50
T2, w, 1000, 10
T2, commit

End of log

T4, start
T3, z, 51
T3, abort
T4, abort
T4, y, 200, 50
T4, y, 200
T4, abort
T1, x, 99
T1, abort
Undo/redo logging

• U: used to track the set of active transactions at crash

• Redo phase: scan **forward** to end of the log
  • For a log record \( \langle T, \text{start} \rangle \), add \( T \) to \( U \)
  • For a log record \( \langle T, X, \text{old}, \text{new} \rangle \), issue \( \text{write}(X, \text{new}) \)
  • For a log record \( \langle T, \text{commit | abort} \rangle \), remove \( T \) from \( U \)
    • If abort, undo changes of \( T \) i.e., add \( \langle T, X, \text{old} \rangle \) before logging abort

  ❖ **Basically repeats history!**

• Undo phase: scan log **backward**
  • Undo the effects of transactions in \( U \)
  • That is, for each log record \( \langle T, X, \text{old}, \text{new} \rangle \) where \( T \) is in \( U \), issue \( \text{write}(X, \text{old}) \), and log this operation too, i.e., add \( \langle T, X, \text{old} \rangle \)
  • Log \( \langle T, \text{abort} \rangle \) when all effects of \( T \) have been undone
Checkpointing

• Shortens the amount of log that needs to be undone or redone when a failure occurs

• Assumption: Txns cannot perform any update actions, such as writing to a buffer block or writing a log record, while a checkpoint is in progress

• Steps:
  • Output to the disk all modified buffer blocks
  • Add to log: <checkpoint \(L\)>, where \(L\) is a list of txns active at the time of the checkpoint

• After a system crash has occurred, the system examines the log to find the last <checkpoint \(L\)> record
  • The redo operations will start from the checkpoint record
  • The undo operations will start from the end of the log until the list of active transactions is empty
Summary

• Recovery: undo/redo logging
  • Normal operation: write-ahead logging, no force, steal
  • Recovery: first redo (forward), and then undo (backward)

• Next lecture:
  • Other forms of durability: data replication
  • Atomicity when data is stored on different machines
  • Data privacy