COL362-632: Recovery System

Garima Gaur

April 12, 2023

Recovery System

- Recovery system is responsible for atomicity and durability of transaction.

Recovery System

- Recovery system is responsible for atomicity and durability of transaction.
- Two tasks of a recovery system:
  - Record information about the transaction updates that will help in recovery after a crash.
  - After the system crashes, recover and ensure the durability of committed transactions.

Recovery System

- Recovery system is responsible for atomicity and durability of transaction.
- Two tasks of a recovery system:
  - Record information about the transaction updates that will help in recovery after a crash.
  - After the system crashes, recover and ensure the durability of committed transactions.
- Log-based recovery: Maintain a log that has a sequence of all the update operations of the transactions.
- Log is stored in stable store – information *never* gets lost.

Database Modification

- DB modification: Copying updated local variables to either the buffer blocks or the disk.

Database Modification

- DB modification: Copying updated local variables to either the buffer blocks or the disk.
- Two approaches of DB modification:
  - Deferred modification: Update DB only after the partial commit. (Validation protocol)
  - Immediate modification: Update DB along the instruction execution. (Lock-based protocol)

Database Modification

- DB modification: Copying updated local variables to either the buffer blocks or the disk.
- Two approaches of DB modification:
  - Deferred modification: Update DB only after the partial commit. (Validation protocol)
  - Immediate modification: Update DB along the instruction execution. (Lock-based protocol)
- Extra information collection prior DB modification by a recovery system.

Example

$\mathbf{T_1}$
read(A);
A := A -50;
write(A);
read(B);
B := B + 50;
write(B);

$\mathbf{T_2}$
read(C);
$C = C - 100$;
write(C);

Log	**DB**
$\langle T_1, Start \rangle$	
$\langle T_1, A, 1000, 950 \rangle$	
$\langle T_1, B, 2000, 2050 \rangle$	
	$A = 950$
	$B = 2050$
$\langle T_1, Commit \rangle$	
$\langle T_2, Start \rangle$	
$\langle T_2, C, 700, 600 \rangle$	
	$C = 600$
$\langle T_2, Commit \rangle$	

Example

$\mathbf{T_1}$	$\mathbf{T_2}$
read(A);	
A := A -50;	read(C);
write(A);	$C = C - 100$;
read(B);	write(C);
B := B + 50;	
write(B);	

Log	DB
$\langle T_1, Start \rangle$	
$\langle T_1, A, 1000, 950 \rangle$	
$\langle T_1, B, 2000, 2050 \rangle$	
	$A = 950$
	$B = 2050$
$\langle T_1, Commit \rangle$	
$\langle T_2, Start \rangle$	
$\langle T_2, C, 700, 600 \rangle$	
	$C = 600$
$\langle T_2, Commit \rangle$	

- Each transaction T_i generates 2 types of log records:
  - For each write(Q) operation
    $$\langle T_i, Q, Q_{old}, Q_{new} \rangle$$
  - Special records to denote the state of the transaction:
    $$\langle T_i, Start \rangle; \langle T_i, Abort \rangle; \langle T_i, Commit \rangle$$
- A transaction commits after $\langle T_i, Commit \rangle$ is output to the log.

Redo and Undo Operation

- Redo(T_i): Set the value of all the data items specified in the log record to the new updated value assigned by T_i.
- Undo(T_i): Restore the value of all the data items updated by T_i to their old value.

Redo and Undo Operation

- Redo(T_i): Set the value of all the data items specified in the log record to the new updated value assigned by T_i.
- Undo(T_i): Restore the value of all the data items updated by T_i to their old value.

Log
$\langle T_1, Start \rangle$
$\langle T_1, A, 1000, 950 \rangle$
$\langle T_1, B, 2000, 2050 \rangle$
$\langle T_1, Commit \rangle$
$\langle T_2, Start \rangle$
$\langle T_2, C, 700, 600 \rangle$
$\langle T_2, Commit \rangle$

- Crash after write(A)

Redo and Undo Operation

- Redo(T_i): Set the value of all the data items specified in the log record to the new updated value assigned by T_i.
- Undo(T_i): Restore the value of all the data items updated by T_i to their old value.

Log
$\langle T_1, Start \rangle$
$\langle T_1, A, 1000, 950 \rangle$
$\langle T_1, B, 2000, 2050 \rangle$
$\langle T_1, Commit \rangle$
$\langle T_2, Start \rangle$
$\langle T_2, C, 700, 600 \rangle$
$\langle T_2, Commit \rangle$

- Crash after write(A) – undo(T_1).
- Crash after write(C) –

Redo and Undo Operation

- Redo(T_i): Set the value of all the data items specified in the log record to the new updated value assigned by T_i.
- Undo(T_i): Restore the value of all the data items updated by T_i to their old value.

Log
$\langle T_1, Start \rangle$
$\langle T_1, A, 1000, 950 \rangle$
$\langle T_1, B, 2000, 2050 \rangle$
$\langle T_1, Commit \rangle$
$\langle T_2, Start \rangle$
$\langle T_2, C, 700, 600 \rangle$
$\langle T_2, Commit \rangle$

- Crash after write(A) – undo(T_1).
- Crash after write(C) – redo(T_i) and undo(T_2).

Redo and Undo Operation

Log
$\langle T_1, \mathit{Start} \rangle$
$\langle T_1, A, 1000, 950 \rangle$
$\langle T_1, B, 2000, 2050 \rangle$
$\langle T_1, \mathit{Commit} \rangle$
$\langle T_2, \mathit{Start} \rangle$
$\langle T_2, C, 700, 600 \rangle$
$\langle T_2, \mathit{Commit} \rangle$

- Crash after write(B) operation – redo(T_1) or undo(T_1)?

Redo and Undo Operation

Log
$\langle T_1, \mathit{Start} \rangle$
$\langle T_1, A, 1000, 950 \rangle$
$\langle T_1, B, 2000, 2050 \rangle$
$\langle T_1, \mathit{Commit} \rangle$
$\langle T_2, \mathit{Start} \rangle$
$\langle T_2, C, 700, 600 \rangle$
$\langle T_2, \mathit{Commit} \rangle$

- Crash after write(B) operation – redo(T_1) or undo(T_1)?
- No commit record implies no persistence of change.

Checkpointing

- Recovery involves redoing and undoing transactions based on if $\langle T_\bullet, Commit \rangle$ or $\langle T_\bullet, Abort \rangle$ record is in the log.
- Logs can be enormous – longer search time and unnecessary Redo operations.

Checkpointing

- Recovery involves redoing and undoing transactions based on if $\langle T_\bullet, Commit \rangle$ or $\langle T_\bullet, Abort \rangle$ record is in the log.
- Logs can be enormous – longer search time and unnecessary Redo operations.
- Checkpointing is performed as:
  - Output all log records to stable storage.
  - Output all modified buffer blocks to the disk.
  - Make an entry $\langle checkpoint, L \rangle$ where L is the list of active transactions at checkpointing.
- No transaction performs any update actions during checkpointing.

Checkpointing

- Recovery involves redoing and undoing transactions based on if $\langle T_\bullet, Commit \rangle$ or $\langle T_\bullet, Abort \rangle$ record is in the log.
- Logs can be enormous – longer search time and unnecessary Redo operations.
- Checkpointing is performed as:
  - Output all log records to stable storage.
  - Output all modified buffer blocks to the disk.
  - Make an entry $\langle checkpoint, L \rangle$ where L is the list of active transactions at checkpointing.
- No transaction performs any update actions during checkpointing.
- All the transactions T_i with $\langle T_i, Commit \rangle$ after the checkpoint are redone.
- All the transactions without commit or abort record after the checkpoint are undone.

Recovery Algorithm

- Recovery takes place in 2 phases: redo phase and undo phase.
- Redo Phase: Scan the log forward from the last checkpoint
  - Initialize undo-list with L.
  - Whenever a non-special log record is encountered, the operation is redone.
  - If $\langle T_i, Start \rangle$ is found, add T_i to undo-list.
  - If either $\langle T_i, Commit \rangle$ or $\langle T_i, Abort \rangle$ is found, remove T_i from the undo-list.
- Undo-list is the list of all incomplete transactions.

Recovery Algorithm

- Recovery takes place in 2 phases: redo phase and undo phase.
- Redo Phase: Scan the log forward from the last checkpoint
  - Initialize undo-list with L.
  - Whenever a non-special log record is encountered, the operation is redone.
  - If $\langle T_i, Start \rangle$ is found, add T_i to undo-list.
  - If either $\langle T_i, Commit \rangle$ or $\langle T_i, Abort \rangle$ is found, remove T_i from the undo-list.
- Undo-list is the list of all incomplete transactions.
- Undo Phase: Scan the log backward from the end and focus on only undo-list transaction records.
  - When a non-special log record is found of undo list transaction T_i, perform undo operation as if T_i has failed and been rolled back.
  - If $\langle T_i, Start \rangle$ is found, writes $\langle T_i, Abort \rangle$ record and remove T_i from undo-list.
- Undo phase ends when undo list becomes empty.

Transaction Rollback

- T_i is rolled back during normal execution:
- Scan the log backward and for each record $\langle T_i, X, X_{old}, X_{new} \rangle$:
  - Update value of X to X_{old}.
  - Write a special redo-only record $\langle T_i, X, X_{old} \rangle$ to the log.
- When $\langle T_i, Start \rangle$ is found, stop scanning and write $\langle T_i, Abort \rangle$ to the log.

Example

$\langle T_0, \mathit{Start} \rangle$

$\langle T_0, B, 2000, 2050 \rangle$

$\langle T_1, \mathit{Start} \rangle$

$\langle \mathit{checkpoint}, \{T_0, T_1\} \rangle$

$\langle T_1, C, 700, 600 \rangle$

$\langle T_1, \mathit{Commit} \rangle$

$\langle T_2, \mathit{Start} \rangle$

$\langle T_2, A, 500, 400 \rangle$

$\langle T_0, B, 2000 \rangle$

$\langle T_0, \mathit{Abort} \rangle$

crash

$\vdots$

$\langle T_2, A, 500 \rangle$

$\langle T_2, \mathit{Abort} \rangle$