Database Systems

CHAPTER 22

Database Recovery Techniques

Introduction

- Recovery algorithms
- Recovery concepts
  - Write-ahead logging
  - In-place versus shadow updates
  - Rollback
  - Deferred update
  - Immediate update
- Certain recovery techniques best used with specific concurrency control methods

Recovery Concepts

- Recovery process restores database to most recent consistent state before time of failure
- Information kept in system log
- Typical recovery strategies
  - Restore backed-up copy of database
    - Best in cases of extensive damage
  - Identify any changes that may cause inconsistency
    - Best in cases of noncatastrophic failure
    - Some operations may require redo

Recovery Concepts (cont’d.)

- **Deferred update techniques**
  - Do not physically update the database until after transaction commits
  - Undo is not needed; redo may be needed

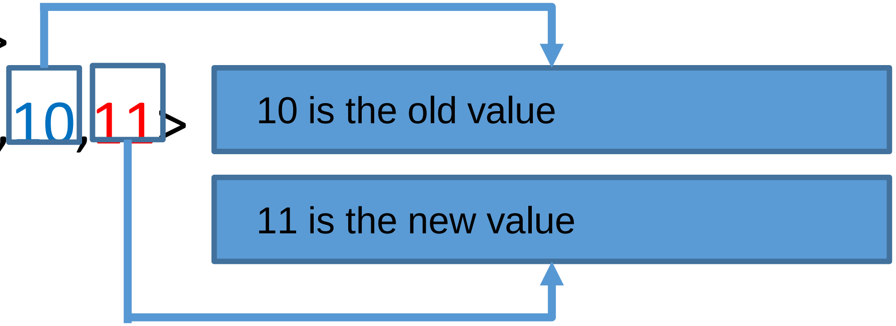
- **Example: (Log For Deffered Update)**
  - <start_Transaction,T1>
  - <read_item,T1,A>
  - <write_item,T1,A,10>
  - <commit, T1>

10 is the new value

- **Note:** in deffered update we don’t need to store outdated value of data item because in this case we never undo any operation.

Recovery Concepts (cont’d.)

- Immediate update techniques
  - Database may be updated by some operations of a transaction before it reaches commit point
  - Operations also recorded in log
  - Recovery still possible.
- Example: (Log For ImmediateUpdate)
  - <start_Transaction,T1>
  - <read_item,T1,A>
  - <write_item,T1,A,10,11>
  - <commit, T1>



10 is the old value

11 is the new value

Recovery Concepts (cont’d.)

- **Note:** in immediate update we store outdated value first so that we can rollback to the initial value of data item and updated value is stored after the outdated value.

Recovery Concepts (cont’d.)

- Undo and redo operations required to be idempotent
  - Executing operations multiple times equivalent to executing just once
  - Entire recovery process should be idempotent
- Caching (buffering) of disk blocks
  - DBMS cache: a collection of in-memory buffers
  - Cache directory keeps track of which database items are in the buffers

Recovery Concepts (cont’d.)

- Cache buffers replaced (flushed) to make space for new items
- Dirty bit associated with each buffer in the cache
  - Indicates whether the buffer has been modified
- Contents written back to disk before flush if dirty bit equals one
- Pin-unpin bit
  - Page is pinned(1) if it cannot be written back to disk yet

Recovery Concepts (cont'd.)

- Main strategies
  - **In-place updating**
    - Writes the buffer to the same original disk location
    - Overwrites old values of any changed data items
  - **Shadowing**
    - Writes an updated buffer at a different disk location, to maintain multiple versions of data items
    - Not typically used in practice
- Before-image: old value of data item
- After-image: new value of data item

Recovery Concepts (cont’d.)

- **Write-ahead logging**
  - Ensure the before-image (BFIM) is recorded
  - Appropriate log entry flushed to disk
  - Necessary for UNDO operation if needed
- **UNDO-type log entries:** include the old value (BFIM) of the item written by the operation since this is needed to undo the effects of the operations from the log.
- **REDO-type log entries:** include the new value(AFIM) of the item written by the operation since this is needed to redo the effects of the operations from the log.

Recovery Concepts (cont'd.)

Steal/no-steal and force/no-force

- Rules that govern when a page from the database cache can be written to disk
- **No-steal approach**
  - Cache buffer page updated by a transaction cannot be written to disk before the transaction commits
- **Steal approach**
  - Recovery protocol allows writing an updated buffer before the transaction commits

Recovery Concepts (cont'd.)

- **Force approach**
  - All pages updated by a transaction are immediately written to disk before the transaction commits
  - **Otherwise, no-force**
- **Typical database systems employ a steal/no-force strategy**
  - **Advantage of "steal"** is that it avoids need for very large buffer space to store all updated pages in memory
  - **Advantage of "no-force"** is that an updated page of a committed transaction may still be in the buffer when another transaction needs to update it, thus eliminating the I/O cost to write that page multiple times to disk and possible having to read it again from disk

Recovery Concepts (cont'd.)

- **Write-Ahead Logging (WAL)** protocol for recovery algorithm requiring both UNDO and REDO
  - BFIM of an item cannot be overwritten by its after image until all UNDO-type log entries have been force-written to disk
  - Commit operation of a transaction cannot be completed until all REDO-type and UNDO-type log records for that transaction have been force-written to disk

Checkpoints in the System Log and Fuzzy Checkpointing

- **Taking a checkpoint**
  - Suspend execution of all transactions temporarily
  - Force-write all main memory buffers that have been modified to disk
  - Write a checkpoint record to the log, and force-write the log to the disk
  - Resume executing transactions
- **DBMS recovery manager decides on checkpoint interval**
  - It may be measured in time – say, every ‘m’ minutes – or in the number ‘t’ of committed transactions since the last checkpoint

Checkpoints in the System Log and Fuzzy Checkpointing (cont'd.)

- **Fuzzy checkpointing**
  - System can resume transaction processing after a begin_checkpoint record is written to the log
  - Previous checkpoint record maintained until end_checkpoint record is written

Transaction Rollback

- Transaction failure after update but before commit
  - Necessary to roll back the transaction
  - Old data values restored using undo-type log entries
- Cascading rollback
  - If transaction T is rolled back, any transaction S that has read value of item written by T, similarly any transaction R that has read value of item written by S, and so on…, must also be rolled back
  - possible only when recovery protocol ensures *recoverable* schedules but does not ensure *strict* or *cascade less* schedule
  - Almost all recovery mechanisms designed to avoid this

Figure 22.1 Illustrating cascading rollback (a process that never occurs in strict or cascadeless schedules) (a) The read and write operations of three transactions (b) System log at point of crash (c) Operations before the crash

(a)

T_1	T_2	T_3
read_item(*A*)	read_item(*B*)	read_item(*C*)
read_item(*D*)	write_item(*B*)	write_item(*B*)
write_item(*D*)	read_item(*D*)	read_item(*A*)
	write_item(*D*)	write_item(*A*)

(b)

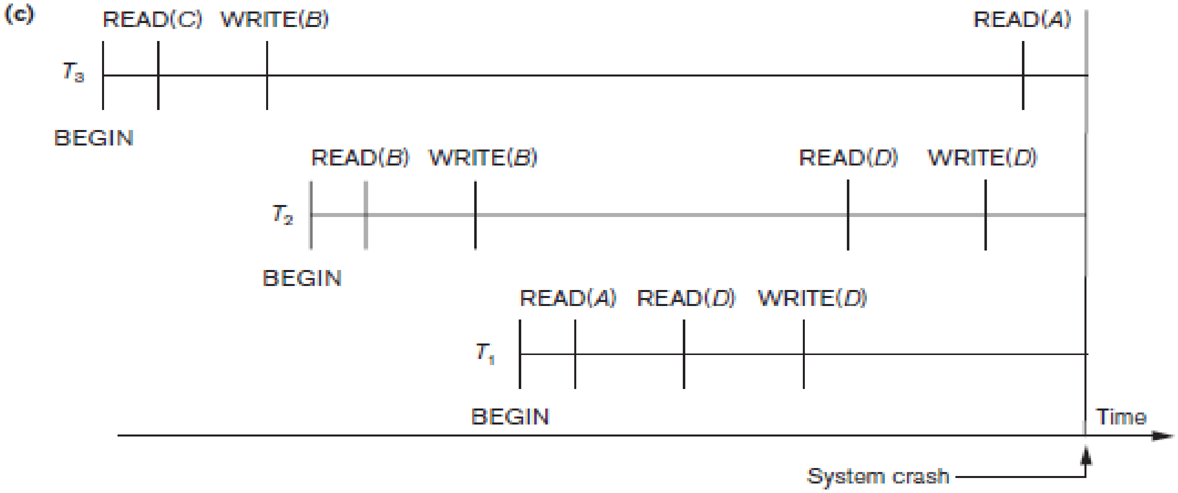
		A	*B*	*C*	*D*
		30	15	40	20
	[start_transaction,T_3]				
	[read_item,T_3,*C*]				
*	[write_item,T_3,*B*,15,12]		12		
	[start_transaction,T_2]				
	[read_item,T_2,*B*]				
**	[write_item,T_2,*B*,12,18]		18		
	[start_transaction,T_1]				
	[read_item,T_1,*A*]				
	[read_item,T_1,*D*]				
	[write_item,T_1,*D*,20,25]				25
	[read_item,T_2,*D*]				
**	[write_item,T_2,*D*,25,26]				26
	[read_item,T_3,*A*]				

← System crash

* T_3 is rolled back because it did not reach its commit point.

** T_2 is rolled back because it reads the value of item *B* written by T_3.

(c)



Transactions that Do Not Affect the Database

- **Example actions:** generating and printing messages and reports
- If transaction fails before completion, may not want user to get these reports
  - Reports should be generated only after transaction reaches commit point
- Commands that generate reports issued as batch jobs executed only after transaction reaches commit point
  - Batch jobs canceled if transaction fails