Carnegie Mellon Univ.  
Dept. of Computer Science 
15-415/615 - DB Applications  

C. Faloutsos – A. Pavlo  
Lecture#23: Crash Recovery  
(R&G ch. 18)

Last Class

- Basic Timestamp Ordering
- Optimistic Concurrency Control
- Multi-Version Concurrency Control
- Multi-Version+2PL
- Partition-based T/O

Today’s Class

- Overview
- Write-Ahead Log
- Checkpoints
- Logging Schemes
- Recovery Protocol
- Shadow Paging

Motivation
### Crash Recovery

- Recovery algorithms are techniques to ensure database **consistency**, transaction **atomicity** and **durability** despite failures.
- Recovery algorithms have two parts:
  - Actions during normal txn processing to ensure that the DBMS can recover from a failure.
  - Actions after a failure to recover the database to a state that ensures atomicity, consistency, and durability.

### Storage Types

- **Volatile Storage:**
  - Data does not persist after power is cut.
  - Examples: DRAM, SRAM
- **Non-volatile Storage:**
  - Data persists after losing power.
  - Examples: HDD, SDD
- **Stable Storage:**
  - A non-existent form of non-volatile storage that survives all possible failures scenarios.

### Failure Classification

- Transaction Failures
- System Failures
- Storage Media Failures
Transaction Failures

- **Logical Errors:**
  - Transaction cannot complete due to some internal error condition (e.g., integrity constraint violation).

- **Internal State Errors:**
  - DBMS must terminate an active transaction due to an error condition (e.g., deadlock)

System Failures

- **Software Failure:**
  - Problem with the DBMS implementation (e.g., uncaught divide-by-zero exception).

- **Hardware Failure:**
  - The computer hosting the DBMS crashes (e.g., power plug gets pulled).
  - **Fail-stop Assumption:** Non-volatile storage contents are assumed to not be corrupted by system crash.

Storage Media Failure

- **Non-Repairable Hardware Failure:**
  - A head crash or similar disk failure destroys all or part of non-volatile storage.
  - Destruction is assumed to be detectable (e.g., disk controller use checksums to detect failures).

- No DBMS can recover from this. Database must be restored from archived version.

Problem Definition

- Primary storage location of records is on non-volatile storage, but this is much slower than volatile storage.

- Use volatile memory for faster access:
  - First copy target record into memory.
  - Perform the writes in memory.
  - Write dirty records back to disk.
Problem Definition

- Need to ensure:
  - The changes for any txn are durable once the DBMS has told somebody that it committed.
  - No changes are durable if the txn aborted.

Undo vs. Redo

- **Undo**: The process of removing the effects of an incomplete or aborted txn.
- **Redo**: The process of re-instating the effects of a committed txn for durability.

- How the DBMS supports this functionality depends on how it manages the buffer pool…

Buffer Pool Management

- Whether the DBMS allows an uncommitted txn to overwrite the most recent committed value of an object in non-volatile storage.
  - **STEAL**: Is allowed.
  - **NO-STEAL**: Is not allowed.
Buffer Pool – Force Policy

- Whether the DBMS ensures that all updates made by a txn are reflected on non-volatile storage before the txn is allowed to commit:
  - **FORCE**: Is enforced.
  - **NO-FORCE**: Is not enforced.

- Force writes makes it easier to recover but results in poor runtime performance.

NO-STEAL + FORCE

- This approach is the easiest to implement:
  - Never have to undo changes of an aborted txn because the changes were not written to disk.
  - Never have to redo changes of a committed txn because all the changes are guaranteed to be written to disk at commit time.

- But this will be slow…
- What if txn modifies the entire database?

Today’s Class

- Overview
- Write-Ahead Log
- Checkpoints
- Logging Schemes
- Recovery Protocol
- Shadow Paging
Write-Ahead Log

- Record the changes made to the database in a log **before** the change is made.
  - Assume that the log is on stable storage.
  - Log contains sufficient information to perform the necessary undo and redo actions to restore the database after a crash.
- Buffer Pool: **STEAL + NO-FORCE**

Write-Ahead Log Protocol

- All log records pertaining to an updated page are written to non-volatile storage before the page itself is allowed to be overwritten in non-volatile storage.
- A txn is not considered committed until all its log records have been written to stable storage.

Write-Ahead Log Protocol

- Log record format:
  - `<txnId, objectId, beforeValue, afterValue>`
  - Each transaction writes a log record first, before doing the change.
  - Write a `<BEGIN>` record to mark txn starting point.
- When a txn finishes, the DBMS will:
  - Write a `<COMMIT>` record on the log
  - Make sure that all log records are flushed before it returns an acknowledgement to application.
WAL – Implementation Details

• When should we write log entries to disk?
  – When the transaction commits.
  – Can use group commit to batch multiple log
    flushes together to amortize overhead.

• When should we write dirty records to disk?
  – Every time the txn executes an update?
  – Once when the txn commits?

WAL – Deferred Updates

• Observation: If we prevent the DBMS from
  writing dirty records to disk until the txn
  commits, then we don’t need to store their
  original values.

• This won’t work if the change set of a txn is
  larger than the amount of memory available.
  – Example: Update all salaries by 5%

• The DBMS cannot undo changes for an
  abortedtxnifitdoesn’thave theoriginal
  values in the log.

• We need to use the STEAL policy.

Faloutsos/Pavlo
CMU SCS 15-415/615
25

Faloutsos/Pavlo
CMU SCS 15-415/615
26

Faloutsos/Pavlo
CMU SCS 15-415/615
27

Faloutsos/Pavlo
CMU SCS 15-415/615
28
WAL – Buffer Pool Policies

<table>
<thead>
<tr>
<th>NO-FORCE</th>
<th>FORCE</th>
<th>NO-STEAL</th>
<th>STEAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slowest</td>
<td>Fastest</td>
<td>Fastest</td>
<td>Slowest</td>
</tr>
</tbody>
</table>

Runtime Performance

Recovery Performance

Almost every DBMS uses **NO-FORCE + STEAL**

Today’s Class

- Overview
- Write-Ahead Log
- Checkpoints
- Logging Schemes
- Recovery Protocol
- Shadow Paging

Checkpoints

- The WAL will grow forever.
- After a crash, the DBMS has to replay the entire log which will take a long time.
- The DBMS periodically takes a **checkpoint** where it flushes all buffers out to disk.

Checkpoints

- Output onto stable storage all log records currently residing in main memory.
- Output to the disk all modified blocks.
- Write a `<CHECKPOINT>` entry to the log and flush to stable storage.
Checkpoints

- Any txn that committed before the checkpoint is ignored (T1).
- T2 + T3 did not commit before the last checkpoint.
  - Need to redo T2 because it committed after checkpoint.
  - Need to undo T3 because it did not commit before the crash.

Checkpoints – Challenges

- We have to stall all txns when take a checkpoint to ensure a consistent snapshot.
- Scanning the log to find uncommitted can take a long time.
- Not obvious how often the DBMS should take a checkpoint.

Checkpoints – Frequency

- Checkpointing too often causes the runtime performance to degrade.
  - System spends too much time flushing buffers.
- But waiting a long time is just as bad:
  - The checkpoint will be large and slow.
  - Makes recovery time much longer.

Today’s Class

- Overview
- Write-Ahead Log
- Checkpoints
- Logging Schemes
- Recovery Protocol
- Shadow Paging
Logging Schemes

- **Physical Logging**: Record the changes made to a specific location in the database.
  - Example: Position of a record in a page.
- **Logical Logging**: Record the high-level operations executed by txns.
  - Example: The UPDATE, DELETE, and INSERT queries invoked by a txn.

Physical vs. Logical Logging

- Logical logging requires less data written in each log record than physical logging.
- Difficult to implement recovery with logical logging if you have concurrent txns.
  - Hard to determine which parts of the database may have been modified by a query before crash.
  - Also takes longer to recover because you must re-execute everytxn all over again.

Physiological Logging

- Hybrid approach where log records target a single page but do not specify data organization of the page.
- This is the most popular approach.

Logging Schemes

- **INSERT INTO X VALUES(1,2,3);**
  - **Physical**: `<T1, Table=X, Page=99, Offset=4, Record=(1,2,3)>`
  - **Logical**: `<T1, "INSERT INTO X VALUES(1,2,3)">`
  - **Physiological**: `<T1, Table=X, Page=99, Record=(1,2,3)>
    `<T1, Index=X_PKEY, Page=45, Offset=9, Key=(1,Record1)?>

Faloutsos/Pavlo   CMU SCS 15-415/615   37
Faloutsos/Pavlo   CMU SCS 15-415/615   38
Faloutsos/Pavlo   CMU SCS 15-415/615   39
Faloutsos/Pavlo   CMU SCS 15-415/615   40
Today’s Class

• Overview
• Write-Ahead Log
• Checkpoints
• Logging Schemes
• Recovery Protocol
• Shadow Paging

Crash Recovery

• Recovery algorithms are techniques to ensure database **consistency**, transaction **atomicity** and **durability** despite failures.
• Recovery algorithms have two parts:
  – Actions during normal txn processing to ensure that the DBMS can recover from a failure.
  – Actions after a failure to recover the database to a state that ensures atomicity, consistency, and durability.

Today's Class – ARIES

• Algorithms for **Recovery and Isolation**
  Exploiting Semantics
  – Write-ahead Logging
  – Repeating History during Redo
  – Logging Changes during Undo

ARIES

• Developed at IBM during the early 1990s.
• Considered the “gold standard” in database crash recovery.
  – Implemented in DB2.
  – Everybody else more or less implements a variant of it.
ARIES – Main Ideas

- **Write-Ahead Logging:**
  - Any change is recorded in log on stable storage before the database change is written to disk.

- **Repeating History During Redo:**
  - On restart, retrace actions and restore database to exact state before crash.

- **Logging Changes During Undo:**
  - Record undo actions to log to ensure action is not repeated in the event of repeated failures.

ARIES – Recovery Phases

- **Analysis:** Read the WAL to identify dirty pages in the buffer pool and active txns at the time of the crash.
- **Redo:** Repeat all actions starting from an appropriate point in the log.
- **Undo:** Reverse the actions of txns that did not commit before the crash.

ARIES - Overview

- Start from last checkpoint found via *Master Record*.
- Three phases.
  - **Analysis** - Figure out which txns committed or failed since checkpoint.
  - **Redo** all actions (repeat history)
  - **Undo** effects of failed txns.
Additional Crash Issues

- What happens if system crashes during the Analysis Phase? During the Redo Phase?
- How do you limit the amount of work in the Redo Phase?
  - Flush asynchronously in the background.
- How do you limit the amount of work in the Undo Phase?
  - Avoid long-running txns.

Today’s Class

- Overview
- Write-Ahead Log
- Checkpoints
- Logging Schemes
- Recovery Protocol
- Shadow Paging

Shadow Paging

- Maintain two separate copies of the database (master, shadow)
- Updates are only made in the shadow copy.
- When a txn commits, atomically switch the shadow to become the new master.
- Buffer Pool: NO-STEAL + FORCE

Shadow Paging

- Database is a tree whose root is a single disk block.
- There are two copies of the tree, the master and shadow
  - The root points to the master copy.
  - Updates are applied to the shadow copy.
Shadow Paging – Example

- To install the updates, overwrite the root so it points to the shadow, thereby swapping the master and shadow:
  - Before overwriting the root, none of the transaction’s updates are part of the disk-resident database.
  - After overwriting the root, all of the transaction’s updates are part of the disk-resident database.

Shadow Paging – Undo/Redo

- Supporting rollbacks and recovery is easy.
  - **Undo:**
    - Simply remove the shadow pages. Leave the master and the DB root pointer alone.
  - **Redo:**
    - Not needed at all.
Shadow Paging – Advantages

- No overhead of writing log records.
- Recovery is trivial.

Shadow Paging – Disadvantages

- Copying the entire page table is expensive:
  - Use a page table structured like a B+tree
  - No need to copy entire tree, only need to copy paths in the tree that lead to updated leaf nodes
- Commit overhead is high:
  - Flush every updated page, page table, & root.
  - Data gets fragmented.
  - Need garbage collection.

Summary

- Write-Ahead Log to handle loss of volatile storage.
- Use incremental updates (i.e., STEAL, NO-FORCE) with checkpoints.
- On recovery: undo uncommitted txns + redo committed txns.

Conclusion

- Recovery is really hard.
- Be thankful that you don’t have to write it yourself.