Lecture #21: ARIES Database Crash Recovery Algorithms

15-445/645 Database Systems (Fall 2019)
`https://15445.courses.cs.cmu.edu/fall2019/`
Carnegie Mellon University
Prof. Andy Pavlo

1 ARIES

Algorithms for **R**ecovery and **I**solation **E**xploiting **S**emantics. Developed at IBM research in early 1990s. Not all systems implement ARIES exactly as defined in the original paper, but they are similar enough.

Main ideas of the ARIES recovery protocol:

- **Write Ahead Logging:** Any change is recorded in log on stable storage before the database change is written to disk (STEAL + NO-FORCE).
- **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.

2 WAL Records

We need to extend the DBMS's log record format to include additional info. Every log record now includes a globally unique *log sequence number* (LSN). Various components in the system keep track of **LSNs** that pertain to them:

- Each data page contains a *pageLSN*: The LSN of the most recent update to that page.
- System keeps track of *flushedLSN*: The max *LSN* flushed so far
- Before page i can be written to disk, we must flush log at least to the point where $pageLSN_i \leq flushedLSN$

3 Normal Execution

We first discuss the steps that the DBMS takes at runtime while it executes transactions.

Transaction Commit

When a transaction goes to commit, the DBMS first writes `COMMIT` record to log buffer in memory. Then the DBMS flushes all log records up to and including the transaction's `COMMIT` record to disk. Note that these log flushes are sequential, synchronous writes to disk.

Once the `COMMIT` record is safely stored on disk, the DBMS returns an acknowledgment back to the application that the transaction has committed. At some later point, the DBMS will write a special `TXN-END` record to log. This indicates that the transaction is completely finished in the system and there will not be anymore log records for it. These `TXN-END` records are used for internal bookkeeping and do not need to be flushed immediately.

Transaction Abort

Aborting a transaction is a special case of the ARIES undo operation applied to only one transaction.

We need to add another field to our log records called the *prevLSN*. This corresponds to the previous LSN for the transaction. The DBMS uses these *prevLSNs* to maintain a linked-list for each transaction that makes it easier to walk through the log to find its records.

We also need to introduce a new type of record called the *compensation log record* (CLR). A CLR describes the actions taken to undo the actions of a previous update record. It has all the fields of an update log record plus the *undoNext* pointer (i.e., the next-to-be-undone LSN). The DBMS adds CLRs to the log like any other record but they never need to be undone.

To abort a transaction, the DBMS first appends a `ABORT` record to the log buffer in memory. It then undoes the transaction's updates in reverse order to remove their effects from the database. For each undone update, the DBMS creates **CLR** entry in the log and restore old value. After all of the aborted transaction's updates are reversed, the DBMS then writes a `TXN-END` log record.

4 Checkpointing

The DBMS periodically takes *checkpoints* where it writes the dirty pages in its buffer pool out to disk. This is used to minimize how much of the log it has to replay upon recovery.

We first discuss two blocking checkpoint methods where the DBMS pauses transactions during the checkpoint process. This pausing is necessary to ensure that the DBMS does not miss updates to pages during the checkpoint. We then present a better approach that allows transactions to continue to execute during the checkpoint but requires the DBMS to record additional information to determine what updates it may have missed.

Blocking Checkpoints

The DBMS halts everything when it takes a checkpoint to ensure that it writes a consistent snapshot of the database to disk. The is the same approach discussed in previous lecture:

- Halt the start of any new transactions.
- Wait until all active transactions finish executing.
- Flush dirty pages on disk.

Slightly Better Blocking Checkpoints

Like previous checkpoint scheme except that you the DBMS does not have to wait for active transactions to finish executing. We have to now record internal system state as of the beginning of the checkpoint.

- Halt the start of any new transactions.
- Pause transactions while the DBMS takes the checkpoint.

Active Transaction Table (ATT): The ATT represents the state of transactions that are actively running in the DBMS. A transaction's entry is removed after the DBMS completes the commit/abort process for that transaction. For each transaction entry, the ATT contains the following information:

- *transactionId*: Unique transaction identifier
- *status*: the current "mode" of the transaction (**R**unning, **C**ommitting, **U**ndo candidate)
- *lastLSN*: Most recent LSN written by transaction

Dirty Page Table (DPT): The DPT contains information about the pages in the buffer pool that were modified by uncommitted transactions. There is one entry per dirty page containing the *recLSN* (i.e., the LSN of the log record that first caused the page to be dirty).

Fuzzy Checkpoints

A *fuzzy checkpoint* is where the DBMS allows other transactions to continue to run. This is what ARIES uses in its protocol.

The DBMS uses additional log records to track checkpoint boundaries:

- `<CHECKPOINT-BEGIN>`: Indicates the start of the checkpoint.
- `<CHECKPOINT-END>`: When the checkpoint has completed. It contains the ATT + DPT.

5 ARIES Recovery

The ARIES protocol is comprised of three phases. Upon start-up after a crash, the DBMS will execute the following phases:

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

Analysis Phase

Start from last checkpoint found via the database's MasterRecord *LSN*.

- Scan log forward from the checkpoint.
- If the DBMS finds a `TXN-END` record, remove its transaction from ATT.
- All other records, add transaction to ATT with status **UNDO**, and on commit, change transaction status to **COMMIT**.
- For `UPDATE` log records, if page P is not in the DPT, then add P to DPT and set P's*recLSN* to the log record's *LSN*.

Redo Phase

The goal is to repeat history to reconstruct state at the moment of the crash. Reapply all updates (even aborted transactions) and redo **CLRs**.

The DBMS scans forward from log record containing smallest *recLSN* in the DPT. For each update log record or CLR with a given *LSN*, the DBMS re-applies the update unless:

- Affected page is not in the DPT, or
- Affected page is in DPT but that record's *LSN* is greater than smallest *recLSN*, or
- Affected *pageLSN* (on disk) $\geq$ *LSN*.

To redo an action, the DBMS re-applies the change in the log record and then sets the affected page's *pageLSN* to that log record's *LSN*.

At the end of the redo phase, write `TXN-END` log records for all transactions with status COMMIT and remove them from the ATT.

Undo Phase

In the last phase, the DBMS reverses all transactions that were active at the time of crash. These are all transactions with UNDO status in the ATT after the Analysis phase.

The DBMS processes transactions in reverse *LSN* order using the *lastLSN* to speed up traversal. As it reverses the updates of a transaction, the DBMS writes a CLR entry to the log for each modification.

Once the last transaction has been successfully aborted, the DBMS flushes out the log and then is ready to start processing new transactions.