Crash Recovery

Sunday, April 17, 2016 9:59 PM

Recovery manager:

Atomicity: undoing the actions of transactions that do not commit
Durability: making sure that all actions of committed transactions survive system crashes

Intro to ARIES

Steal + no force approach

Three phases:

- ANALYSIS: identify dirty pages in the buffer pool and active transactions at the time of crash
- REDO: repeat all actions and restores the database state to what it was at the time of the crash
- UNDO: undo actions of transactions that did not commit, so that the database reflects only the actions of committed transactions

Three main principles:

- Write-ahead logging: any change to a database object is first recorded in the log. The record in the log must be written to stable storage before the change to the database object is written to disk
- Repeating history during REDO: On restarting, ARIES retraces all actions of the DBMS before the crash and brings the system back to the exact state that it was in at the time of the crash. Then it UNDOes the actions of transactions still active at the time of the crash (effectively aborting them)
- Logging changes during UNDO: changes made to the database while undoing a transaction are logged to ensure such an action is not repeated in the event of repeated restarts

The log

Log tail

The most recent portion of the log. Kept in main memory. Periodically forced to stable storage

LSN

For recovery purposes, every page in the database contains the LSN of the most recent log record that describes a change to this page -> **pageLSN**

Log

- Updating a page: pageLSN = LSN
- Commit: append commit type log record to log tail. Log tail written to stable storage (up to and including the commit record)
- Abort: append abort record to log tail
- End: end type appended to log record
- Undoing an update: CLR written when an update log record is undone

Update Log records

- PrevLSN: link each transaction
- TransID
- Type
- PageID
- Length
- Offset
- Before-image: value before change
- After-image: value after change

Compensation Log Records

Written just before the change recorded in an update log record U is undone

- Abort
- Recovery from crash

Appended to the log tail
Fields:

- UndoNextLSN: the LSN of the next log record that is to be undone for the transaction that wrote update record U. set to the value of prevLSN in U

CLR describes an action that will never be undone
Bound: # of CLRs that can be written during undo is no more than the number of update log records for active transactions at the time of crash

Transaction table

One entry for each active transaction.

- Transaction id
- Status: in progress/ committed / aborted
- LastLSN: the most recent log record for this transaction

Dirty page table

One entry for each dirty page in the buffer pool: each page with changes not yet reflected on disk.

- RecLSN: LSN of the first log record that dirties the page (identifies the earliest log record that might have to be redone for this page during restart from a crash) (the first update to this page that may not have been written to disk)

Write-ahead log protocol

Committed transaction: effectively a transaction of whose log records, including a committed record, have been written to stable storage
When a transaction is commited, the log tail is forced to stable storage, even if no-force approach is being used

Checkpoint

Snapshot of the DBMS state

- Begin_checkpoint: written to indicate when the checkpoint starts
- End_checkpoint: constructed, including the current contents of the transaction table and the dirty page table

- Guarantees that: the transaction table and the dirty page table are accurate as of the time of the being_checkpoint record

Recovery Procedure

Analysis phase

Analysis phase begins by examine the most recent begin_checkpoint record, and proceeds forward in the log until the last log record.

Tasks:

1. The point to start redo
2. The pages in the buffer pool that were dirty at the time of the crash
3. Transactions that are dirty at the time of the crash

Steps:

1. Init the transaction table and the dirty page table to the most recent version included in the next end_checkpoint log record starting from the most recent begin_checkpoint log record
2. Scans the log in the forward direction until it reaches the end of the log:
    a. If END log record for transaction T is encountered, T is removed from the transaction table (no longer active)
    b. If a record for T other than END is encountered, an entry for T is added to the transaction table if it's not already there. This entry for T is modified as:
        i. The lastLSN filed is set to the LSN of this log record
        ii. If the log record is a COMMIT record, the status of is set to C, o/w set to U (to be undone)
    c. If a redoable log record affecting page P is encountered, and P is not in the dirty page table, an entry is inserted into this table with pageID P and recLSN equal to the LSN of this redoable log record. This LSN identifies the oldest change affecting page P that may not have been written to disk.

Redo phase

Redo phase

Follows the analysis phase and redos all changes to any page that might have been dirty at the time of the crash. Set of pages for redo is determined in the analysis phase.

REDO order: same as the original order!

Reapplies the updates of all transactions, committed / otherwise. If a transaction was aborted before the crash and its updates were undone, as indicated by CLRs, the actions described in the CLRS are also re-applied.

The redo phase begins with the log record that has the smallest recLSN of all pages in the dirty page table constructed by the Analysis pass because this log record identifies the odest update that may not have been written to disk prior to the crash. Starting from this record, REDO scans forward until the end of the log. For each REDOABLE log record (update / CLR) encountered, REDO checks whether the logged action must be redone. The action MUST BE REDONE unless one of the following conditions holds:

- The affected page is NOT in the dirty page table (ENSURES THAT ALL CHANGES TO THIS PAGE HAVE BEEN WRITTEN TO DISK)
- The affected page is in the dirty page table, but the recLSN for the entry is greater than the LSN of the log record being checked (UPDATE BEING CHECKED WAS INDEED PROPAGATED TO DISK)
- The pageLSN (stored on page, which must be retrieved to check this condition) is greater than or equal to the LSN of the log record begin checked.

If the logged action must be redone:

1. The log action is re-applied
2. The pageLSN on the page is set to the LSN of the redone log record. No additional log record is written at this time

At the end of the REDO phase, END type records are written for all transactions with status C, which are removed from the transaction table.

Undo phase

Undoes the changes of all transactions active at the time of the crash. Set of transactions identified in the analysis phase.

UNDO order: reverses changes in the opposite order, reversing the most recent changes first.

The UNDO phase scan backward from the end of the log.

UNDO algorithm

-> Loser transactions: all transactions active at the time of the crash, includes the LSN of the most recent log record (lastLSN field) for each such transaction. All loser transactions must be undone, in the reverse order in which they appear in the log.

-> **ToUndo**: set of lastLSN values for all loser transactions.

UNDO repeatedly chooses the largest LSN value in this set and processes it, until ToUndo is empty. To process a log record:

1. If it is a CLR and the undoNextLSN value is not null, the undoNextLSN value is added to the set ToUndo. If the undoNextLSN is null, an END record is written for the transaction because it is completely undone, and the CLR, is discarded.
2. If it is an UPDATE record, a CLR is written and the corresponding action is undone, and the prevLSN value in the update log record is added to the set ToUndo.

When the set ToUndo is empty, the UNDO phase is complete. Restart is now complete, and the system can proceed with normal operations.

Aborting a transaction

Upon ABORT, an ABORT type record is first written to the log record. Then the UNDO

part is called solely to make necessary undo procedures. This will undo relative actions and write corresponding CLRs to the log record. END type log record is automatically taken care of.

? ~~Upon ABORT, CLR are written to the log, and corresponding actions are undone. If end is reached, an END type log record is written to the log~~