Recovery

Aries – C. Mohan, ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging, ACM Transactions on Database Systems, Vol. 17, No. 1, March 1992, pp. 94–162

Designed to work with a STEAL, NO-FORCE approach.

STEAL – covered in storage (basically says that a trans can steal a buffer-pool frame from uncommitted trans)
NO-FORCE – Changes made to the actual object are not required to be made in-place (i.e. forced to disk).

Works in 3 phases:

1) Analysis phase – identifies the dirty pages in the buffer pool (i.e. changes that have not been written to disk) and active transactions at the time of the crash.

2) Redo phase – repeats all actions, starting from an appropriate point in the log and restores the database state to what it was at the time of the crash. (This includes the transactions that did not finish)

3) Undo phase – undoes the actions of transactions of transactions that did not commit so that the database reflects only the actions of committed transactions.

Recovery system is also responsible for executing the ROLLBACK command – which aborts a single transaction.

T1:W(P5); T2:W(P3); T2:commit; T2:end; T3:W(P1); T3:W(P3); CRASH

Analysis identifies T1 and T3 as active transactions (must be undone), T2 is a committed transaction (i.e. all it’s actions written to disk), and P1, P3, P5 as potentially dirty pages.

3 main principals behind Airies:

1) Write ahead logging – any change to the database 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.
2) Repeating history during redo: On restart following a crash, Aries retraces all actions of the DBMS before the crash and brings the system back to the exact state it was at the time of the crash. Then it undoes the actions of transactions still active at the time of the crash.
3) 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.

Log entries are identified by LSN (log sequence number). Each log record has certain fields – prevLSN, transID, type. The log entries for a transaction are kept as a linked list going back in time through the prevLSN field – updated each time a log record is added.

Compensation Log Record (CLR) is written just before the change recorded in an update log record U is undone. Such an undo can happen during normal execution when a transaction is aborted or during recovery from a crash. This record describes the action taken to undo the actions recorded in the corresponding update log record and is appended to the log tail. A CLR contains a field called undoNextLSN which is the LSN of the next log record that is to be undone for the transaction that wrote the original update record – this is set to the value of prevLSN from the U record.

Transaction table – contains one entry for each active transaction. Contains lastLSN – the LSN of the most recent log record for this transaction and the status of the transaction in progress (in progress, committed, aborted).

Dirty page table – contains one entry for each dirty page in the buffer pool, i.e. each page with changes that are not yet written to disk. Contains recLSN which is the LSN of the first log record that caused the page to be dirty. This identifies the earliest log record that might have to be redone for this page during restart from a crash.

Write ahead log protocol – fundamental rule that ensures that a record of every change to the database is available while attempting to recover from a crash. No-force says that that some of the changes may not have written to disk at the time of a crash.

1) Must force the log record for an update before the corresponding data page gets to disk (i.e. the flushed LSN (the LSN of the last log entry flushed to disk) must be greater than the pageLSN (most recent log record for an update to this page).
2) When a transaction is committed, the log tail is forced to stable storage. (as opposed in a force situation, all pages modified by a trans would be forced) Much smaller write. Log kept as a sequential, cost of appending cheaper.

Checkpointing – snapshot of the DBMS state, reduces the amount of work to be done during a restart.

1) Begin_checkpoint record is written to indicate when the checkpoint starts
2) End_checkpoint record is constructed, including in it the current contents of the transaction table and the dirty page table, and appended to the log
3) After the End_checkpoint record is written to stable storage, a master record containing the LSN of the begin_checkpoint record is written to a known place on stable storage.

This is called a fuzzy checkpoint and is inexpensive because it does not require quiescing the system or writing the pages in the buffer pool.

Analysis phase

1) Determines the point in the log at which to start the redo phase.
2) Determines (a conservative superset of the) pages in the buffer pool that were dirty at the time of the crash.
3) Identifies the transactions that were active at the time of the crash and must be undone.

Begins by examining the most recent begin_checkpoint record and initializing the dirty page table and transaction table to the copies of those structures in the next end_checkpoint record. If additional log records are between the begin_checkpoint and the end_checkpoint records, the tables are adjusted to reflect the info in these records.

Scanning the log forward from this point:

1) If an end log record for a transaction is encountered, the transaction is removed from the transaction table because it is no longer active. (End record is written after the commit() returns).
2) If a log record other than an end record for a transaction is encountered, an entry for T is added to the transaction table if it is not already there, the lastLSN of that transaction is set to the current log entry LSN, and if the transaction is a commit, the status of the transaction is set to C. (default is U meaning it must be undone).
3) If a redoable log record affecting page P is encountered and P is not in the dirty page table – and entry is inserted into this table with page id P and recLSN equal to the LSN of the redoable log record.

At the end of the Analysis phase, the trans table has been rebuild, and the dirty page table contains a superset.

Redo phase

During the redo phase, ARIES reapplies the updates of all transactions – committed or otherwise. If a transaction was aborted before the crash and its updates were undone (indicated by the presence of CLR records), the actions described in the CLRs is also reapplied. This is called repeating history – and causes the database to be brought to the same point it was at the time of the crash.

We start by inspecting the dirty page table and finding the smallest recLSN of all the pages (this is why we stored the first transaction to dirty the page and not the last). This record identifies the oldest update that may not have been written to disk prior to the crash. For each redoable record (an update or a CLR), redo checks to see if the logged action must be redone. The action is done unless of these conditions holds:

1) The affected page is not in the dirty page table
2) The affected page is in the dirty page table, but the recLSN for the entry is greater than the LSN of the log entry being checked (i.e. the first record to dirty this page after the checkpoint is greater than the current log entry)
3) The pageLSN (stored on the page which is retrieved from disk) is greater than or equal to the LSN of the log record being checked (i.e. the last log entry to change the page before it was written to disk came after or was this log entry).

If the logged action must be redone

1) The logged action is reapplied
2) The pageLSN on the page is set to the LSN of the redone log record.

Undo phase

The undo phase then scans backward from the end of the log. It's goal is to undo the actions of all transactions active at the time of the crash – effectively aborting them. This set of transactions is identified in the transaction table constructed during the Analysis phase.

The transaction table contains the LastLSN – the LSN of the most recent log record this transaction wrote. These transactions – called loser transactions – must be undone and further, these actions must be done in reverse order in which they appear in the log (i.e. you want to return to the intial state not a partial state). We consider the set of lastLSN values for all loser transactions – called ToUndo.

Undo continually chooses the largest (i.e. most recent) LSN value in this set and processes it until the ToUndo set is empty. Processes the log record is accomplished by:

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, and end record is written for the transaction because it is completely undone and the CLR discarded.
2) If it is an update record, a CLR is written and the corresponding action is undone. The prevLSN value in the update log is added to the set ToUndo.