Write-ahead Logging

1. Goals

(a) Provide high-availability (fast repair) and consistency in the presence of failures

(b) Transaction failures
   i. Logical errors: internal error condition, bad input, data not found, resource unavailable, etc. These correspond to software faults
   ii. System errors: deadlock or other system problem that prevents the execution of a transaction
   iii. rollback: explicit call to fail a transaction by an application. We will often use the term rollback to describe all of these conditions, which all cause the log to “rollback”

(c) System crash
   i. bug in the OS or the DB
   ii. hardware problem
   iii. environmental fault (power)

(d) We will be taking a single transaction view of recovery at this point.
   i. While I will discuss multiple transactions, we will not consider isolation.
   ii. So WAL is a protocol for atomicity and durability
   iii. What about consistency?

2. The Database Log

(a) What’s a log? A sequential file that contains a record of actions taken by an entity
   i. entity is the DB actions are writes to pages
   ii. Sequential refers to the fact that the log is written sequentially (at consecutive increasing offsets)
   iii. Items are written to the tail of the log

(b) What's a log look like? A contiguous region of storage (preferably a whole disk)
   i. Why contiguous? So that sequential log writes are written sequentially to storage.
      A. Sidebar on disk drive properties.
      B. Sequential layout allows writes to occur without seeks
   ii. Why a whole disk?
      A. The storage is not important, i.e. we do not need a whole disks worth of storage.
      B. The disk arm is the valuable resource.
      C. If the disk is shared with other data, two workloads are presented to the disk at the same time. The workload of the log and the workload of the other data.
      D. The combined workload is not sequential and seeking must occur when switching between workloads.

3. The what’s what of writes and updates

(a) The log is non-volatile storage, log writes go to disk

(b) The DB is non-volatile storage, db writes go to disk

(c) Pages are the in-memory representation of DB disk block(s)

(d) Updates are made to the in-memory copy of the database

4. What does the log contain?

(a) Log records (Duh!). In ARIES, an update record describes a single database write and consists of:
i. transaction ID
ii. data-item identifier typically physical (disk location), rather than logical (tuple)
iii. Old value
iv. New value
v. ⟨ TID, Disk block, Old, New ⟩

(b) This is a very general form (we will look at more specific examples) that allows operations to be undone and redone
(c) There are other types of log records (non-update).
   i. begin transaction, end transaction, checkpoint, rollback

5. The Logging Principle
   (a) Logically, the log only grows, nothing is delete
      i. If you want to undo an update ⟨ TID, Disk block, Old, New ⟩, place an equivalent undo forward in the log ⟨ TID, Disk block, New, Old ⟩

6. Implementing the Log on Disk
   (a) Important to remember, the log is not the database. The log exists beside the primary copy of the database.
   (b) How do we implement a log in a fixed amount of space?
   (c) remove transactions that are resolved (aborted or committed) from the head of the log
      i. Apply all outstanding updates (in memory) to the database image (disk)
      ii. Truncate the log from the head up to the first update record for an unresolved transaction
      iii. never delete from the tail of the log
   (d) Only throwing out old records that contain un-needed entries
   (e) What’s the problem?
      i. Long-running transactions can overflow the log
      ii. even if they don’t write a lot of data
      iii. the only thing that matters is the distance between the first update record and the current head of the log
      iv. These transactions cannot even be rolled-back. Why?
      v. Rollback requires writing undo update records

Logging Protocols

1. Deferred Modification (Redo Logging)
   (a) Log contains entries that allow an update to be re-done
   (b) ⟨ TID, Disk Block, New ⟩
   (c) Updates are written to the log only!
   (d) Updates are made to the in-memory version of the database
   (e) Once a transaction:
      i. commits: i.e. a commit record is written to the database, all pages used in the transaction can be written to the database
      ii. aborts: i.e. all pages used in the transaction are discarded in favor of the on-disk, pre-transaction versions
(f) the abort/commit semantics mean that a page needs to be hardened (written to disk) after a transaction prior to the next transaction that uses the page

(g) *Why? What is the problem and its performance implications?*

(h) this can be a hotspot

(i) actually, the system could allow a new transaction to use the page

(j) *How?* if the new transaction aborts, we would need to redo the committed transaction

(k) this does not tend to be done (too complex) makes it harder to truncate the log, i.e. determine completed/resolved transactions

(l) there is a better protocol for dealing with this situation

(m) Deferred modification refers to commit time

(n) Updates *must* be deferred until this point.

2. Immediate modification (Undo Logging)

   (a) Log contains entries that allow an update to be un-done

   (b) ⟨TID, Disk Block, Old⟩

   (c) Updates are written to the log and database

   (d) When a transaction:

      i. commits: all updates must be written to disk at commit time

         ii. aborts: all work must be undone

   (e) Immediate modification refers to commit time, i.e. all updates must be on disk prior to committing the transaction

   (f) Updates during the transaction may be written to disk

3. *Any initial thoughts on the relative advantages/disadvantages of these two protocols?*

   (a) Memory usage

   (b) I/O performance

4. Steal and Force

   (a) *Steal:* stealing a page is writing an uncommitted update to disk

      i. Used to free memory resources

      ii. Allows the system to run even when outstanding transactions have oversubscribed memory

      iii. Without steal, a database consumes all memory and fails

      iv. *What kind of logging supports steal?*

         v. Undo logging makes it possible to steal pages

   (b) *Force:* force is the requirement that all pages dirtied by a transaction are on disk when that transaction commits

      i. No-force refers to system that can commit a transaction without writing all updates to disk

         ii. No-force requires redo logging

   (c) Redo is a no-steal/no-force policy

   (d) Undo is a steal/force

   (e) Redo-Undo logging is steal/no-force at the expense of some space and complexity

      i. Flexible and powerful WAL that is used in most production databases
5. For what applications would you pick undo versus redo versus undo/redo?

(a) Undo – when you need to steal pages, but don’t need force
   i. Long-running transactions
   ii. Performance insensitive at commit
   iii. Space-constrained transactional environments (handhelds?)

(b) Redo – when you don’t need to steal pages
   i. short transactions (OLTP systems)
   ii. Fast commit – throughput and latency sensitive applications

6. Restart Recovery

(a) What we do when the DB has failed.
(b) Does not apply to transaction abort or rollback of individual tran
(c) Systems starts with unresolved transactions in the log
   i. Either, committed updates that are not to the disk
   ii. Dirty pages on disk for uncommitted transactions
(d) What happens?
   i. Incomplete transactions get rolled back
   ii. DB is brought into a consistent state with the log for complete transactions

7. Redo Recovery

(a) Take a forward pass over the log to redo the operations of committed transactions that are not in the database.
(b) With analysis – two passes
   i. First pass resolves the outcomes (commit/abort/incomplete—abort) of all transactions in the log
   ii. Second pass redoes updates for committed transactions
(c) without analysis – one pass
   i. Forward pass perform updates to in-memory pages (only)
   ii. Write pages to DB when a commit record is found
   iii. Discard pages for transactions that rollback/incomplete
(d) With and w/out analysis is a tradeoff between memory usage, I/O, and # of passes
   i. w/out analysis can run out of memory and needs to fall back on with analysis
   ii. fewer passes better

8. Undo Recovery

(a) Backward pass over the log to undo the operations of uncommitted transactions that are not in the database.
(b) Requires no separate analysis pass.
   i. In a backward pass, if the recovery manager sees a TID for which no commit record was already read, the operation need to be undone.

9. Undo and Redo compared

(a) Redo
   i. Advantages: fast updates (write to log only), fast commit (write to log only)
ii. Disadvantages: slower restart (not significant) and can run out of memory when the system has many uncommitted updates.

(b) Undo
   i. Advantages: never runs out of memory, easy and efficient recovery
   ii. Disadvantages: requires all writes to be made to the DB at commit time
   iii. Why is this a problem? Writes to the DB are much slower than the log (not sequential). Consider what happens when many transactions repeatedly update the same page.

10. Undo/redo logging
   (a) The real solution, best of both worlds
   (b) Fast commit and space management
   (c) Steal/no-force
   (d) Redo recovery – analysis pass, redo pass, undo pass

11. Recovery concepts
   (a) Why it might seem desirable to do the undo pass first, it turns out that the undo pass is actually forward processing (i.e. requires log records to be written) whereas the redo pass is restart processing (no log records)
   (b) ARIES recovery ethic: bring the DB system memory into the state that it was in prior to a crash (this includes recovering things like the lock table) and then undo any transactions that did not complete
   (c) Also allows for concurrency between new transactions and undo workload.
   (d) Note: undo can be the highest overhead portion of recovery. Parallelism is very desirable.

12. Revisiting some issues
   (a) Running out of memory. We had discussed previously that a system can get into a deadlock situation in which it cannot rollback transactions because it is out of memory. What do the recovery protocols we have discussed do to address this?
      i. Steal policy
      ii. Analysis pass on restart
   (b) What are we logging and why?
      i. Specifically page updates.
      ii. Log that we have made a change to a page.
      iii. Not tuple/record oriented logging. Why not?
      iv. Makes more sense when we discuss the mechanics of logging.