Performance Oriented InnoDB Log Format Changes in MariaDB

Marko Mäkelä
Lead Developer InnoDB
MariaDB Corporation

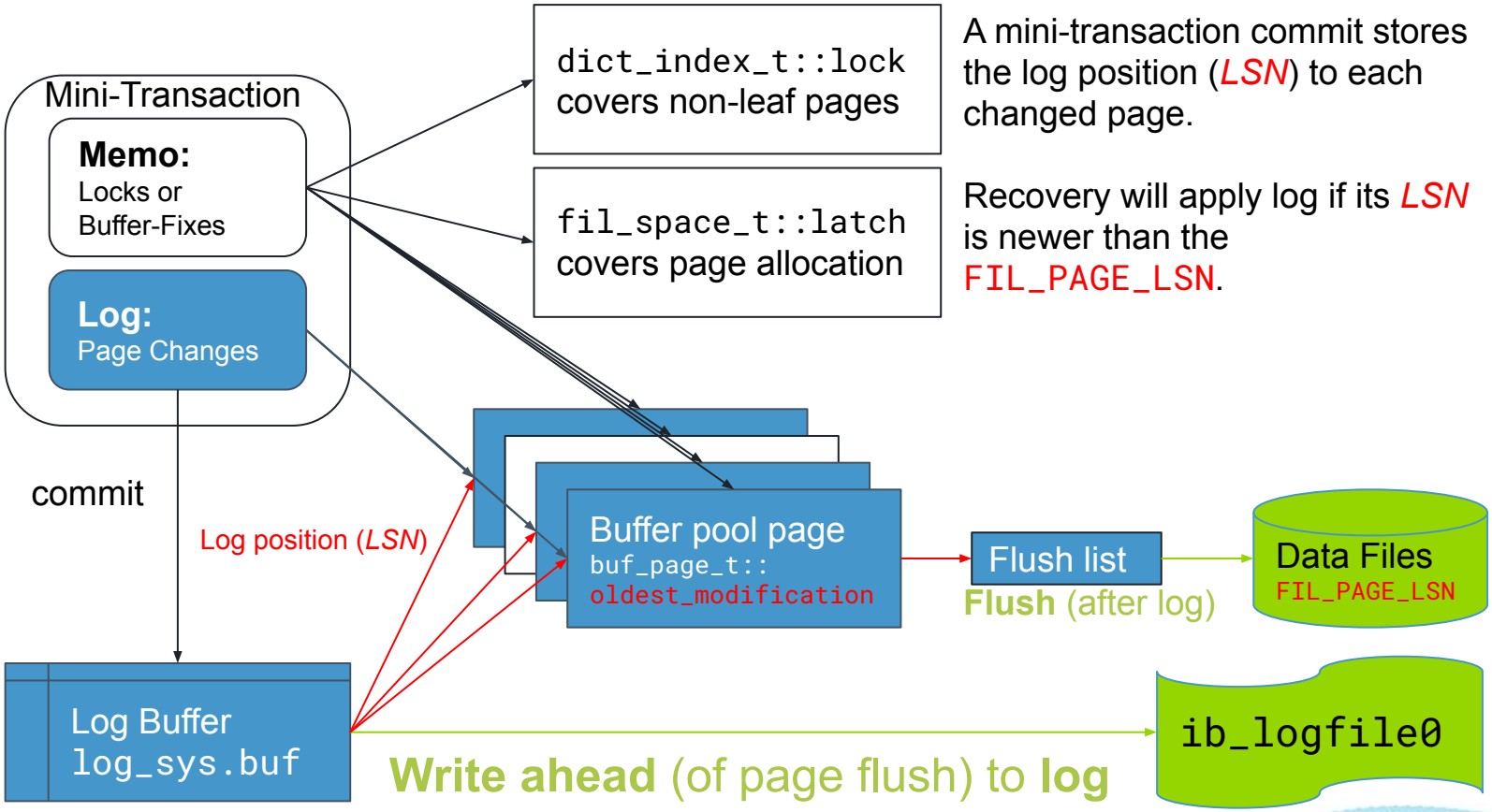
ACID Transactions for InnoDB in MariaDB

- MariaDB: Every access is covered by metadata **locks** (MDL) on the table name
- InnoDB modifications: table locks, index record locks and page latches
- `DELETE` (or `DROP`) will only schedule data for future removal, after `COMMIT`
- Any important state changes will be *durably* written to the redo **log** first
  - A transaction may consist of several mini-transactions
  - For a `COMMIT` to be durable, everything up to the commit *LSN* must be written

InnoDB ACID Basics: Locks and Log

- A log sequence number (*LSN*) totally orders the output of ***mini-transactions***
  - An **atomic** change to pages is **durable** if all log up to the end *LSN* has been written
- Undo log pages implement ACID ***transactions*** (implicit locks, rollback, MVCC)
- Write-ahead logging: The `FIL_PAGE_LSN` of a changed page must be durable
- Log checkpoint: **write all changed pages older than the checkpoint** *LSN*
- Recovery will have to process log from the checkpoint *LSN* to last durable *LSN*

Atomic Mini-Transactions: Latches and Log



How InnoDB Crash Recovery Works

Recovery Processes Log from Checkpoint

- The checkpoint *LSN* defines the logical point of time for starting recovery
- The logical end of the circular `ib_logfile0` must never overwrite the start!
- The start is logically discarded by advancing the checkpoint *LSN*
  - Checkpoint *LSN* must not be ahead of `MIN(oldest_modification)` in `buf_pool`
- Use `innodb_log_file_size` ≫ `innodb_buffer_pool_size` to optimize
  - MariaDB Server 10.5 improved the efficiency of memory usage on recovery

A Simplified View of the `ib_logfile0`

- Header block (512 bytes): Identifies the log file format and stores `first_lsn`: the *LSN* when the file was created, at `START_OFFSET`
- 2 checkpoint blocks (overwritten alternatively), containing
  - The checkpoint *LSN* (start of the log for recovery)
  - An “end” *LSN* pointing to records that identify names of files that were modified since the previous checkpoint (at the end of the log at the time of the checkpoint)
- Log records: `capacity()` bytes from `START_OFFSET` to `file_size`
- The byte offset of an *LSN* is given by the formula:
  `START_OFFSET + (lsn - first_lsn) % capacity()`

A Simplified View of Recovery

1. Determine the latest checkpoint *LSN*, and jump to the “end” *LSN*
    - We expect to find any number of `FILE_MODIFY` records and a `FILE_CHECKPOINT` record pointing to the checkpoint *LSN*
2. Start processing records from the checkpoint *LSN* to the very end
    - After the last complete mini-transaction, we will encounter checksum or sequence number mismatch
    - Construct a mapping from numeric tablespace identifiers to file names
    - Store page-level log in a hash table: (tablespace_id,page_number)↦(records)

Memory Management During Recovery

- For applying changes, we must allocate pages in the buffer pool
  - Typically for reading an old version of the page, to apply log on
  - MariaDB 10.2+ avoids read if the page was (re)initialized since the checkpoint
  - MariaDB 10.5+ discards log if the page was freed since the checkpoint
- Memory for the hash table of records is allocated from the buffer pool
- Multiple **apply batches** may be needed to make memory available
- During the final batch, we can allow concurrent access to the database

Format Changes for Performance

`ib_logfile0` Format Changes in MariaDB

- Before MDEV-12353 in MariaDB Server 10.5, log records had an irregular structure with no explicit length information
  - Parsing invoked “dry run” of the “apply” function of each log record type
- Redo log was stored in 512-byte blocks with some header and a footer
  - Validate and decrypt log blocks, copy the payload to `recv_sys.buf`
- MDEV-14425 (10.8): Remove the block structure
  - Process records directly from `log_sys.buf` (`innodb_log_buffer_size`)
  - Optionally, with `mmap()` of the entire log file

The MDEV-12353 Log Record Format

- 4 bits of type and 4 bits of length
  - If this byte is 0, this is the end of a mini-transaction (or a padding byte)
  - If the length bits are 0, the record will be longer than 16 bytes, and the remaining length will be written using variable-length encoding
- Tablespace ID and page number with variable-length encoding (1 to 5 bytes)
  - Omitted if the “same page” bit of the type is set (never for the first record)
- Any remaining bytes are interpreted according to the type bits
- If the “same page” bit is set in the first record, the mini-transaction only contains file-level records or the special `FILE_CHECKPOINT` record

The MDEV-14425 Mini-Transaction Format

- An end byte `0x00` or `0x01` marks the end of a mini-transaction
  - An INIT_PAGE record would always start with a byte `0x02` to `0x0a`
- If `innodb_encrypt_log=ON`, an 8-byte nonce will follow the end byte
- Last, a 4-byte checksum of the mini-transaction (excluding the end byte)
- The end byte contains a sequence bit: number of times the circular redo log wrapped around from the end, modulo 2
- For padding log blocks, dummy mini-transactions could be written
  - Parser support is present, but we are not padding anything right now

Example: A MDEV-14425 Mini-Transaction

- **35 00 08** 81 e5 20 (the non-**bold** bytes may be encrypted)
  - WRITE(3), 5 bytes follow, tablespace 0, page 8
  - Offset 613 (0x81e5 decoded as 0x80+0x1e5), 1 byte to write: 0x20
- **b9** 1e 0e 07 00 00 01 38 02 ff
  - WRITE(3), same page, offset 644 (613+1+0x1e), 8 bytes to write
- **01** (end of mini-transaction, and the value of the sequence bit at this point)
- **97 41 0a 2d**
  - HEX(CRC32C(x'35000881e520b91e0e0000102ff'))

The MDEV-14425 Encrypted Log Format

- We never encrypt file names, *LSN*, tablespace id, page number
  - They were always available even in encrypted data files anyway
  - Decryption is only needed for applying log, not for backup
  - No mutex is held while encrypting or calculating checksums
- The record payload (excluding type, length, tablespace identifier, page number) is encrypted with an initialization vector that consists of:
  - the tablespace identifier and the page number of the current record
  - the 8-byte nonce that precedes the mini-transaction checksum

Changes to File System Interface

- On Linux and Windows: Detect and use the physical block size; on Linux, allow `O_DIRECT` on the `ib_logfile0`
- When built with `libpmem` and the log is in a `mount -o dax` filesystem, we make `log_sys.buf` point directly to the persistent memory
- On Linux, we also allow “fake PMEM” when the log is in `/dev/shm`
  - A little faster CI runs (Linux regression tests run on `/dev/shm`)
  - More convenient rr debugging: the entire log is in `log_sys.buf` at all times
  - `innodb_log_group_home_dir=/dev/shm` gives PMEM performance estimate

Thank you for using MariaDB!