
# InnoDB Status&Roadmap in MariaDB

Marko Mäkelä

Shanghai

November 2019

# InnoDB Improvements in MariaDB 10.5

- 10.5.0 MDEV-19514 Defer change buffer merges until pages requested
  - Prevents 'random' crashes due to change buffer corruption
- 10.5.0 MDEV-16264 Implement a work queue for InnoDB background tasks
  - Removes a large number of InnoDB background threads
- In progress: MDEV-18959 Engine transaction recovery through binlog
  - Only `fsync()` the binlog on transaction commit, not InnoDB redo log
- Planned: Remove `innodb_log_optimize_ddl` (write full `ALTER TABLE` log)
  - Enables MDEV-19738 Doublewrite buffer is unnecessarily used for newly (re)initialized pages

# I/O Scalability Improvements

- Not started: MDEV-16260 Scale the purge effort according to the workload
- In progress: MDEV-12353/MDEV-14425 Efficient redo log record format
- Early stages: MDEV-16526 Overhaul the InnoDB page flushing
  - Blocks: MDEV-15058 Remove multiple InnoDB buffer pool instances
  - In progress: MDEV-18115 Remove dummy tablespace for the redo log
- In progress: MDEV-15528 Punch holes when pages are freed
  - MDEV-12226 Avoid writes of freed (garbage) pages to InnoDB temporary tablespace
  - MDEV-12227 Defer writes to the InnoDB temporary tablespace
- Not started: MDEV-14481 Execute InnoDB crash recovery in the background

# Rewrite of I/O Subsystem

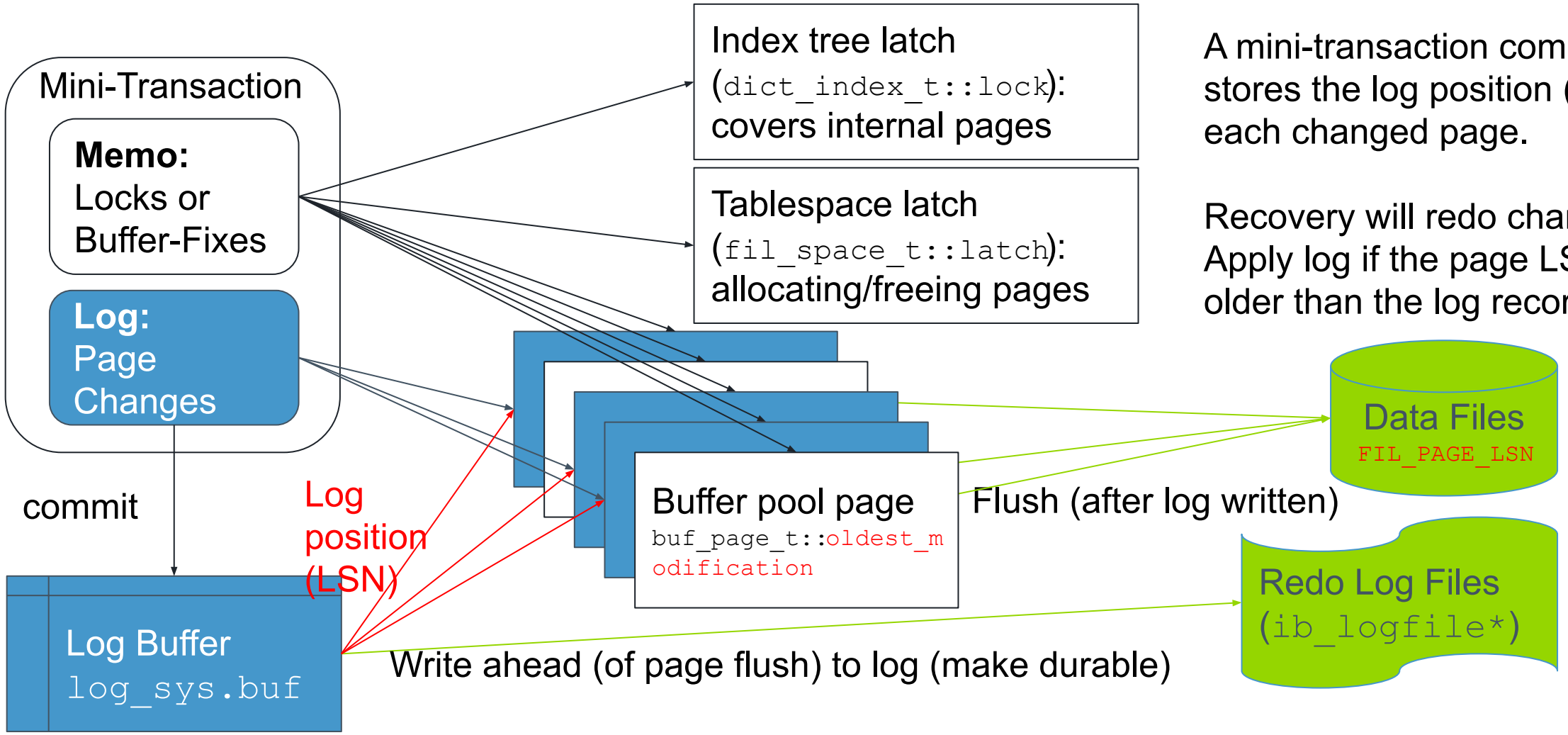
## Page Flushing and Log Checkpoints

MariaDB

# Write Dependencies and ACID

- Log is written by *mini-transactions*, to atomically update pages.
  - Transactional ACID (record locks, rollback, MVCC) builds upon this.
  - Mini-transactions are totally ordered by LSN (log sequence number)
  - A mini-transaction is durable if everything up to its LSN has been written to log
    - A user transaction `COMMIT` is durable if the mini-transaction of is durable
- Write-ahead logging: Must **write log before dirty pages**, at least up to the `FIL_PAGE_LSN` of the dirty page that is about to be written
- Log checkpoint: **write dirty pages older than the checkpoint** LSN
  - Recovery will have to process log from the checkpoint LSN to last written LSN
- MDEV-16264 Implement a common work queue… simplifies page flushing
  - `io_submit()` from only one thread, `io_getevents()` from another

# Mini-Transactions: RW-Latches and Redo Logs




A mini-transaction commit stores the log position (LSN) to each changed page.

Recovery will redo changes: Apply log if the page LSN is older than the log record LSN.

MariaDB

# Optimizing Log Writes

- Current situation: Mutex contention: Any thread that issues writes can:
  - write or fsync the log ⇒ contention on `log_sys.mutex` or `log_sys.write_mutex`
  - invoke `log_checkpoint()` by `log_free_check()`
  - Checkpoint is also initiated by master thread, and log writes by page writes!
- Goal: Have a **dedicated log writer task** that is signalled by other threads
  - Page flush skips "too new" pages instead of initiating&waiting for log flush
    - Avoid mutex: `log_sys.last_flushed_lsn.load()`
  - Remove `buf_page_t::newest_modification` and just use `FIL_PAGE_LSN`
  - **Dedicated log checkpoint task**
    - `log_free_check()` would submit a task (if needed) and wait for completion
- `mtr_t::commit()` returns immediately (just transfer the `mtr_t::m_log` ownership); user tasks can request a durable variant that waits

# Redo Log Format Redesign

Compact, extensible format, faster recovery

MariaDB

# Planned Redo Log Changes in 10.5+

- MDEV-12353 Efficient redo log record format
  - Done: Replace physio-logical log records with purely physical ones
  - Removed: `innodb_log_optimize_ddl` (write compact redo log for `ALTER TABLE`)
  - Missing: Implement compact encoding for the remaining (physical) log records
- Redo log apply will be completely rewritten (no GPL dependency!)
  - Opens possibility for "smart storage" à la Amazon Aurora or Alibaba PolarDB
    - InnoDB writes only log (no page flushing, no log checkpoints!)
    - InnoDB reads back pages as of a specified LSN. (Easy "flashback" to any time.)
- MDEV-14425 InnoDB redo log format for better performance
  - `ib_logfile0` will be a dummy file, or at most contain checkpoint information
  - Write file create/delete/rename and checkpoint information into a separate file
  - Two format options for the page-level log file:
    - circular in-place log (similar to the current format)
    - append-only log (periodically create new log files, allow log archiving)

# Redo Log File Format (1/2)

- Partitioning the log was considered and rejected in MDEV-14425
  - Forces `fsync()` of all log files at `COMMIT`, destroying any performance benefit
- Append-only, "stream of bytes" log file format to cover changes to pages
  - Checksum at the end of each durable snippet (after possible compression)
  - For more flexibility, make LSN count mini-transactions, not payload bytes
    - `mariabackup --incremental` can write records to the redo log!
    - `mariabackup --prepare` can be performed by normal server startup
- Checkpoint information file:
  - All files created, deleted, renamed, modified since the previous checkpoint
  - Checkpoint LSNs and corresponding log file names and byte offsets
  - Can contain multiple checkpoints, written sequentially
- Can use MDEV-17084 Optimize append only files for NVDIMM

# Redo Log File Format (2/2)

- `ib_logfile0` will just contain a special header that indicates new format
- Checkpoint files will follow the pattern `ib_files.%06u`
- Page-level data files will follow the pattern `ib_log.%06u`
  - Each file will start with a header that identifies the creator version, and whether the file is circular, or append-only
  - Circular log file does no rotation and will write blocks, with LSN in the header
- Checkpoint and log files may be rotated separately or in sync, upon reaching a configured maximum size
  - On rotation, a file with a “one bigger” suffix will be created. No renames!
  - Use the existing infrastructure for log file rotation (Aria log, binlog)

# Optimizing Write Performance

## Smarter Page Writes, Fewer `fsync()`

MariaDB

# Optimizing Dirty Page Flushing

- **Dedicated log checkpoint task** kicks in when the checkpoint is too old
  - Clustrix: **active page flushing** (concurrently with the normal page cleaners)
    i. `checkpoint_lsn=log_sys.lsn`; write and `fsync()` the log file
    ii. S-latch page, write (Clustrix: X-latch, copy to a **staging buffer** for writing), unlatch
    iii. Write all dirty pages and call `fsync()` or `fdatasync()` on the data files
    iv. Write and `fsync()` the checkpoint information
  - Clustrix: If right after completion, the circular log file is again too full, start another flushing thread to increase effort
  - Maybe active flushing is a bad idea (performance drop during checkpoint)
- Remove `BUF_FLUSH_SINGLE_PAGE`
- Do we need separate batches `BUF_FLUSH_LRU` (w/ evict) or `BUF_FLUSH_LIST`?
  - Can we always sort the `buf_pool->flush_list` like on recovery (`flush_rbt`)?

# Reducing `fsync()` Operations

- `fsync()` of redo log persists important state changes (and any older writes)
  - Binlog-driven transaction: Fake `XA PREPARE` in InnoDB (with `fsync()`), then `write();fsync()` binlog, and finally fake `XA COMMIT` without `fsync()`
    - After MDEV-18959: Do binlog `write();fsync()` and `COMMIT` without `fsync()`
  - Without binlog: `COMMIT`, `XA PREPARE`, `XA ROLLBACK`, (SQL-level) `XA COMMIT`
    - Set up a "`fsync()` completion" event that would send OK packet to client?
    - (Better throughput if the client connection submits multiple transactions.)
- `fsync()` is overkill for 'write barriers'. **Leverage `liburing` after 10.5?**
  - Before data page flush at *LSN*, we `fsync()` the write of log ≥*LSN*
  - Before completing log checkpoint, we `fsync()` all data files
  - Before binlog rotation (discarding the start of binlog), MDEV-18959 must `fsync()` the InnoDB redo log up to the LSN of the first remaining commit in the binlog

# Longer-Term Ideas

What to improve in InnoDB after 10.5

MariaDB

# More Performance and Flexibility (1/2)

- Leverage `liburing` to avoid `fsync()` for 'write barriers'
- Move things out of the system tablespace, to prepare for its removal
  - MDEV-11634 Logical change buffer, exploited also for `ROLLBACK`
  - MDEV-11659 Move the InnoDB doublewrite buffer to flat files
  - MDEV-19506 Remove the global sequence `DICT_HDR_ROW_ID` for `DB_ROW_ID`
  - MDEV-15020 Store persistent statistics in `.ibd` file (or remove the code?)
  - Note: InnoDB system tables will remain until MDEV-11655
- MDEV-18518 Atomic `CREATE` of partitioned table; crash-safe `DROP INDEX`
- MDEV-11658 Simpler, faster `IMPORT` of InnoDB tables
- Improve record locks: MDEV-10962, MDEV-16406, MDEV-16232, MDEV-11215, MDEV-20612; replace table locks with MDL?

# More Performance and Flexibility (2/2)

- Move `FOREIGN KEY` out of InnoDB: MDEV-12483, MDEV-10393, …
- Non-blocking `COMMIT`: Send OK packet after transaction is durable
  - Allow interleaved execution of the next transaction while log flush is pending
- MDEV-16232 Use fewer mini-transactions
  - Implicit record locks in `UPDATE`, `DELETE`, `INSERT…ODKU`, `REPLACE`
  - Remove the row prefetch buffer from InnoDB
- MDEV-515 Bulk insert into empty table or partition (`TRUNCATE` on `ROLLBACK`)
- MDEV-18746 Reduce the amount of `mem_heap_create()` or `malloc()`
- `ALTER TABLE`: MDEV-16356 `ADD CONSTRAINT`, `ALGORITHM=NOCOPY`, MDEV-16281 parallel `ADD INDEX`, MDEV-9260 Improve progress reporting


# Thank you