Buffer Pool Performance Improvements in the InnoDB Storage Engine of MariaDB Server

Marko Mäkelä
Lead Developer InnoDB
MariaDB Corporation

Scalability in Databases

- A database management system implements ACID transactions
- Users need concurrent access to the same tables, records, or data pages
  - **Transactional locks** on records will be held until `COMMIT` or `ROLLBACK`.
- MVCC reads are non-locking but still involve **latches**
  - Mini-transactions (atomic modifications of multiple pages) hold page latches
  - Buffer pool (requesting, reading, flushing, evicting pages), redo log writes, …

A Layered Implementation of Transactions

Low Layers in the OSI Model

- **Transport:** TCP/IP
- **Network:** IP, ICMP, UDP, BGP, DNS, … (router/switch)
- **Data link:** Packet framing, checksums
- **Physical:** Ethernet (CSMA/CD), WLAN (CSMA/CA), …

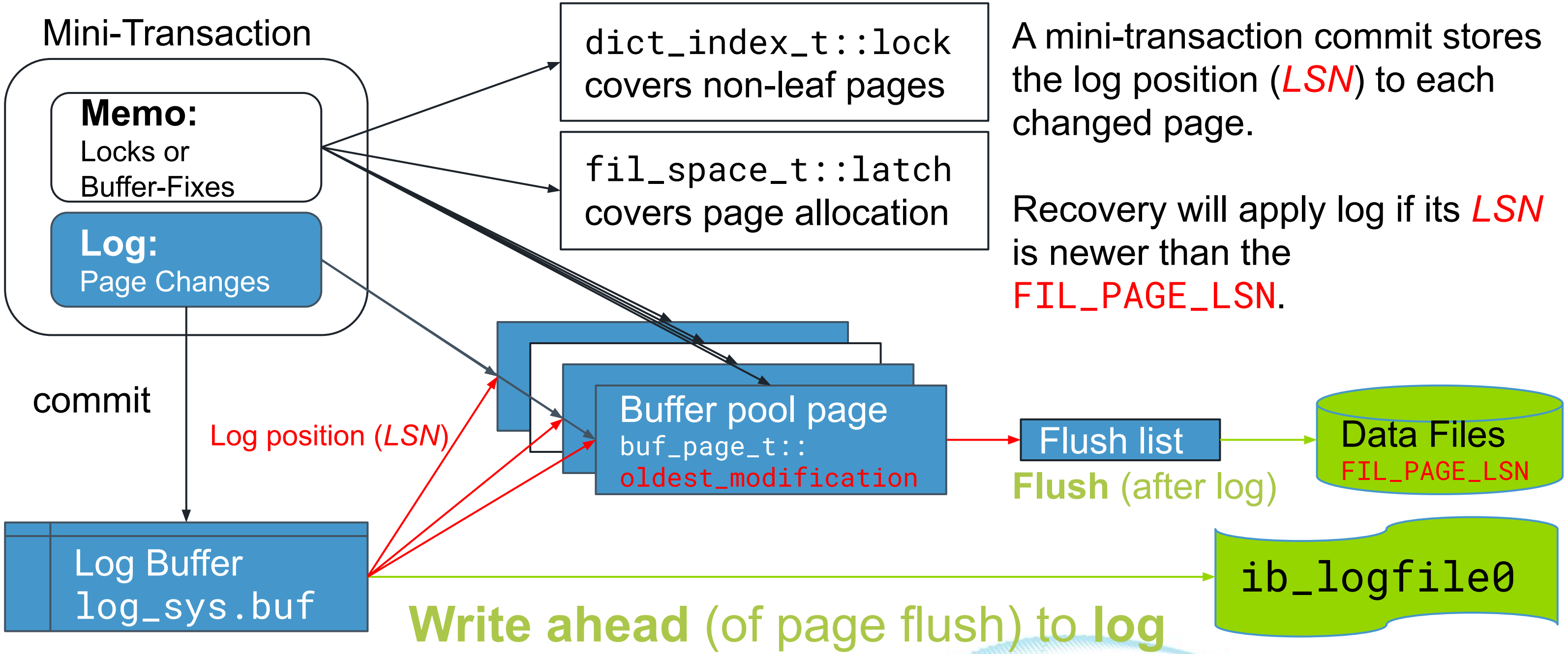
A Storage Engine in a DBMS

- **Transaction:** ACID, MVCC
- **Mini-transaction (+buffer pool):** Atomic, Durable writes (+recovery)
- **File system (+cache):** ext4, XFS, ZFS, NTFS, NFS, …
- **Storage:** HDD, SSD, PMEM, …

Write Dependencies and ACID

- 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



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

Recovery will apply log if its *LSN* is newer than the `FIL_PAGE_LSN`.

MariaDB 10.5 Avoids Unnecessary Writes

- Freed pages will be discarded: Useful in massive `DROP` (or rebuild) operations
- Doublewrite buffer will be skipped for newly (re)initialized pages
  - Crash recovery will avoid reading pages that are fully initialized by redo log.
- Change buffer merge is only executed on demand, not in the background
- `TEMPORARY TABLE` pages will only be written on LRU eviction (since 10.5.9)
- `innodb_flush_neighbors` is ignored on SSD (since 10.4)

Tackling the Root Causes of Bottlenecks

MariaDB Server 10.5

MariaDB

Some Changes to the InnoDB Buffer Pool

- In 2006, MySQL 5.0.30 introduced `buf_block_t::mutex` to reduce some contention on `buf_pool->mutex`
- In 2010, MySQL 5.5.7 partitioned the buffer pool by hash on page identifier
- MySQL 5.6: multiple page cleaner threads (complicated further in MySQL 5.7)
- In 2020, MariaDB Server 10.5 reverted to a single `buf_pool` and page cleaner
    - November 2020: MariaDB Server 10.5.7 reduced latency in the page cleaner

Low-Level Contention is Expensive

- The contention of `buf_pool.mutex` was reduced by the following:
    - The access rules for `buf_pool.page_hash` were simplified, and a `std::atomic` based cache-friendly rw-lock is now interleaved with the hash array.
    - The `buf_block_t::mutex` was eliminated, thanks to more use of `std::atomic`.
    - Code refactoring removed unnecessary pairs of mutex unlock/lock operations.
- MariaDB 10.5 seems to scale to thousands of concurrent connections
    - The work-around `innodb_thread_concurrency` was removed.

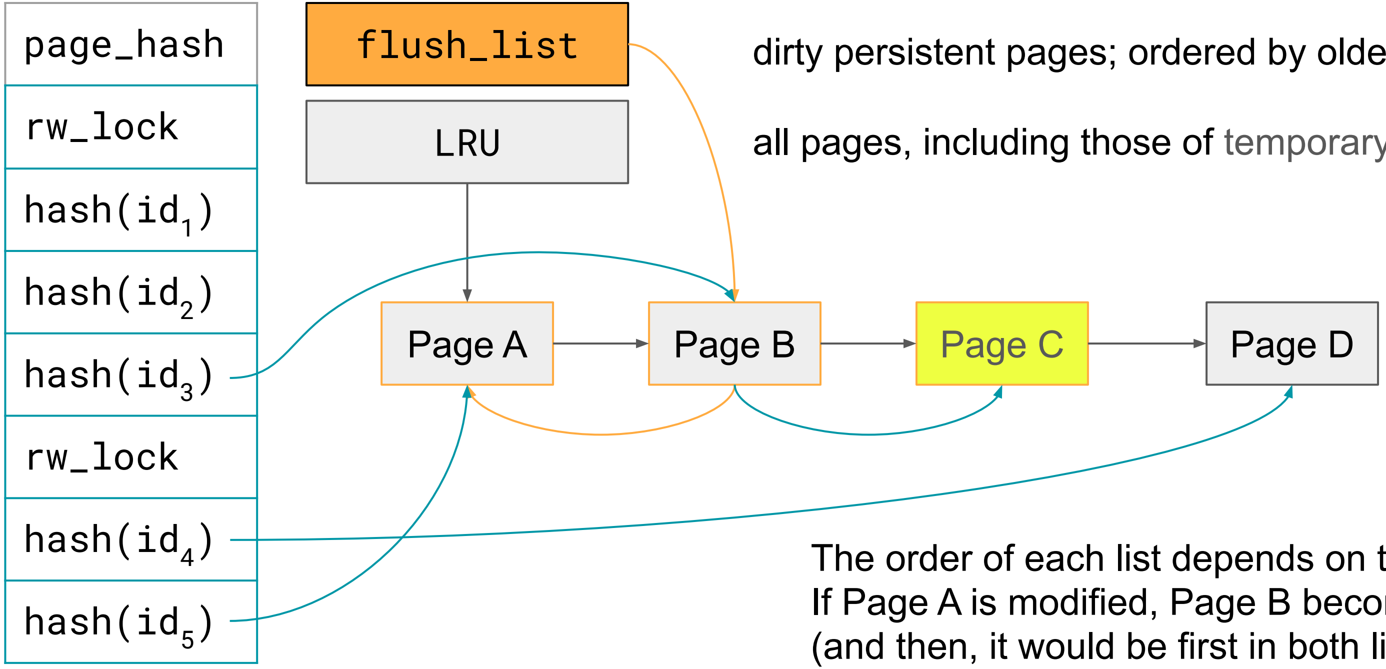
Partition mutexes, not the data structures

- MySQL 5.5 split not only `buf_pool->mutex` but the entire `buf_pool`
- MySQL 5.6: multiple page cleaner threads (complicated further in MySQL 5.7)
- Einstein: “Make things as simple as possible, but not any simpler”
    - Can we actually achieve scalability with a single `buf_pool.mutex`? Yes!
    - A regression was observed for some cases of write-heavy workloads.
    - Page flushing was not as simple as possible!

Removing Bottlenecks in Page Writes

- The `fil_system.mutex` was acquired several times per page write
  - Use a single `std::atomic` field in `fil_space_t` for reference-counting and flags
- The synchronous writes of the doublewrite buffer conflicted with `fsync()`
  - 10.5.7 initiates a single asynchronous write for 128 pages (while filling another 128-page buffer); on write completion initiates writes for the data pages
- Thanks to Microsoft tools and Linux `sudo perf record -t`⟨*page cleaner*⟩

An Overview of buf_pool Data Structures



dirty persistent pages; ordered by oldest *LSN*

all pages, including those of temporary tables

The order of each list depends on the workload.
If Page A is modified, Page B becomes LRU
(and then, it would be first in both lists!)

MariaDB

What is Eviction Flushing (LRU Flushing)?

- If the buffer pool is full and a page is going to be read or created, something must be thrown out (evicted) to free up storage space
    - `buf_pool.LRU` keeps track of all pages, following least-recently-used policy
- If none of the 100 least-recently-used pages are clean, flushing kicks in
    - MDEV-23399 (MariaDB Server 10.5.7) removed "single-page flushing", and instead makes the user thread initiate an asynchronous eviction flushing batch.
    - Write completion callbacks will instantly free the buffer block for future use.

What is Checkpoint Flushing?

- 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
  - Recovery in MariaDB Server 10.5 is faster and will not run out of memory.

Simplifying the Page Cleaner

- The page cleaner threads had multiple modes and coordination with each other
- With LRU flushing moved to user threads, and with a “recovery coordinator” thread removed, we dedicate the page cleaner to checkpoint flushing activity
  - Log checkpoints are cheapest to initiate at the end of page write batch completion!
  - Each batch skips locked or “too new” pages.
  - At the start of each batch, a concurrent log write is initiated to ensure progress.
  - Use normal mutexes and condition variables for inter-thread communication.

Lower-Latency Emergency Flushing in 10.5.7

- Cause of performance stalls: Ensuring that the log will not overwrite itself
  - The page cleaner tries to advance the checkpoint after every `innodb_io_capacity` pages, reducing the wait time in the user threads.
  - Common workaround: `SET GLOBAL innodb_dirty_pages_pct_lwm=10;`
- A new `buf_flush_ahead()` interface was added to give “early warning” to the page cleaner thread, initiating the “furious flushing” mode earlier
  - `mtr_t::commit()` may initiate it, avoiding a wait in a future `log_free_check()`

Future Improvements

- MariaDB 10.6 replaces `buf_block_t::lock` and the old homebrew `rw_lock_t` with a leaner implementation
- MariaDB 10.6 also replaces homebrew mutexes and events with normal mutexes and condition variables
  - The only remaining case of the homebrew spin-loop seems to be working around contention on `lock_sys.mutex`, which will be tackled separately.
- Upcoming changes to file formats will enable even more improvements

Concurrency is Hard, Performance is Harder

- Testing is overwhelmed by a combinatorial explosion of parameters
- The performance of a database server depends on many factors
  - Bad configuration parameters; sometimes poorly documented: `innodb_max_dirty_pages_pct_lwm=0` (MDEV-24537)
  - Particular hardware, firmware, operating system or file system version
- Performance testing introduces one more factor: time to reach steady state