Scalability Improvements in the InnoDB Storage Engine in MariaDB

Marko Mäkelä
Lead Developer InnoDB
MariaDB Corporation
Scalability in Databases

- A database management system implements concurrent transactions
  - Transactions must be ACID (Atomic, Consistent, Isolated, and Durable).
- Users need concurrent access to the same tables, records, or data pages
  - Concurrency may be limited due to locking conflicts or contention.
  - Transactional locks will be held until COMMIT or ROLLBACK.
- READ UNCOMMITTED, READ COMMITTED, and REPEATABLE READ bypass transactional locks but not any (hopefully short-duration) internal latches
  - Mini-transactions (atomic modifications of multiple pages) hold page latches
  - Buffer pool (requesting, flushing, or evicting pages), redo log writes, ...
A Layered Implementation of Transactions

**Low Layers in the OSI Model**
- **Transport**: Retransmission, flow control (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 changes to multiple files, Durable (with 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*.
  - The mini-transaction’s *atomic* change to one or multiple pages is *durable* if all log up to the end LSN has been written.

- Undo log pages implement ACID *transactions* (implicit locks, rollback, MVCC)
  - A user transaction COMMIT is durable if its undo page change is durable.

- Write-ahead logging: Must *write log before changed pages*, at least up to the FIL_PAGE_LSN of the changed page that is about to be written

- 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
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.
Log Format Changes for More Write Speed and Faster Recovery
Improvements to the Redo Log

- Fewer writes and reads of data pages thanks to new log records
  - We now avoid writes of freed pages after `DROP` (or rebuild) operations.
  - The doublewrite buffer is not used for newly (re)initialized pages.
- An improved group commit reduces contention and improves scalability
- We can write log without system calls to persistent memory module
- The physical format is easy to parse, thanks to explicitly encoded lengths
  - Optimized memory management on recovery (or `mariabackup --prepare`).
Fewer Writes and Reads of Data Pages

- Page (re)initialization will write an INIT_PAGE record
  - Recovery will avoid reading the page and reconstruct it based on log records.
  - Page flushing can safely skip the doublewrite buffer.

- Freeing a page will write a FREE_PAGE record to log, and
  - Freed pages will not be written back, nor read by crash recovery!
  - If scrubbing is enabled, flushing will overwrite freed pages with zeroes.
  - Short-lived pages may avoid being written completely.
Faster InnoDB Redo Log Writes

- Vladislav Vaintroub introduced a `group_commit_lock` for more efficient synchronization of redo log writing and flushing.
  - The goal was to reduce CPU consumption on `log_write_up_to()`, to reduce spurious wakeups, and improve the throughput in write-intensive benchmarks.
  - Benchmarks highlighted that performance is very sensitive to redo log volume. Logical `UNDO_APPEND`, `INSERT`, `DELETE` records are more compact than purely physical log covering changes to many header or pointer fields.

- Sergey Vojtovich and Eugene Kosov wrote an optional `libpmem` interface to improve performance on Intel® Optane™ DC Persistent Memory Module
  - Write to a memory-mapped file, and execute CLFLUSH to make it durable.
Improved Backup and Recovery

- Recovery (and mariabackup) must parse and buffer all log records that were durably written since the last completed log checkpoint LSN.

- The new log record format in MariaDB Server 10.5 makes this faster:
  - Explicitly encoded lengths simplify parsing.
  - Simpler memory management: A record can never exceed innodb_page_size.

- The recovery of logical INSERT, DELETE includes validation of page contents:
  - Corrupted data can be detected more reliably.
Code Cleanup in MariaDB Server 10.5
Cleanup of Background Threads and Tasks

- InnoDB used to have a single “master thread”

- MySQL 5.5, 5.6, 5.7, MariaDB 10.1: more and more threads for simple tasks
  - Most threads would be idle for much of the time, consuming OS resources.

- MariaDB Server 10.5: Most background tasks are run in a thread pool

- MariaDB Server 10.5: Purge tasks sort work by table_id
  - Reduces look-up of non-existent tables and contention between purge tasks.
  - Acquire MDL, process several records for the same table, release MDL.

- Future work: Scale background activity based on foreground workload
Removal of InnoDB thread throttling

- Back in the MySQL 5.1 times, throughput would collapse when exceeding 8 concurrent connections, due to kernel_mutex, buf_pool->mutex, ...
  - Workaround: innodb_thread_concurrency, innodb_commit_concurrency

- But, we test MariaDB with ‘insane’ number of connections without seeing a dramatic drop of total throughput
  - MariaDB Server 10.3 significantly reduced trx_sys.mutex contention
  - MariaDB Server 10.5 reduced some contention in buf_pool and dict_sys
  - MariaDB Server 10.5.5 removes the throttling code that has become useless, and deprecating and ignoring the parameters. MariaDB Server 10.6 will remove them.
More Predictable Change Buffer

- InnoDB aims to avoid read-before-write when it needs to modify a secondary index B-tree leaf page that is not in the buffer pool.
  - Insert, delete-mark and purge (delete) operations can be written to a change buffer in the system tablespace, to be merged to the final location later.

- MariaDB Server 10.5 no longer merges buffered changes in the background
  - Change buffer merges can no longer cause hard-to-predict I/O spikes.
  - A corrupted index can only cause trouble when it is being accessed.

- This was joint work with Thirunarayanan Balathandayuthapani

- Future work: Simpler, logical format; use it also on ROLLBACK
InnoDB Data Dictionary Cleanup

- Thirunarayanan Balathandayuthapani extended the use of metadata locks (MDL)
  - Background operations must ensure that the table not be dropped.
  - This used to be covered by `dict_operation_lock` (or `dict_sys.latch`), which covers any InnoDB table!
  - It suffices to acquire MDL on the table name.

- In a future release, we hope to remove `dict_sys.latch` altogether, and to replace internal transactional table locks with MDL.
Some Changes to the InnoDB Buffer Pool

- The InnoDB buffer pool is a page cache (user tables, indexes, or undo logs).

- 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.

- In 2020, MariaDB Server 10.5 reverted back to a single buffer pool.
  - Some unnecessarily global data was removed (e.g., `buf_page_t::flush_type`).
  - Some remaining contention was addressed by making more use of C++11 `std::atomic` in data structures, and `buf_block_t::mutex` was removed.
  - Simpler `buf_pool.page_hash` with cache-friendly latching improves concurrency.
Ideas for Faster Writes and Startup

- Asynchronous COMMIT: send OK packet on write completion
  - Execute next statement(s) without waiting for COMMIT. (Idea: Vladislav Vaintroub)

- Complete the InnoDB recovery in the background, while allowing connections
  - Basically, just remove a special ‘recovery mode’ from page flushing.
  - The rollback of recovered incomplete transactions was always performed in the background.
  - We could also allow read-only startup on a data directory when recovery is needed (so that you can look what is inside, without modifying anything).
Limitations in Current File Formats

● Secondary indexes are missing a per-record transaction ID
  ○ MVCC, purge, and checks for implicit locks could be much simpler and faster.

● DB_ROLL_PTR and the undo log format limit us to 128 rollback segments
  ○ Cannot possibly scale beyond 128 concurrently starting write transactions.

● Redo log: 512-byte block size causes copying and mutex contention
  ○ Block framing forces log records to be split or padded.
  ○ A mutex must be held while copying, padding, encrypting, computing checksums.
Flash-Friendly Log Format

- Write information about checkpoints and file operations into separate file
  - That file can be written to without affecting the LSN.
  - Instead of writing `.delta` files, `mariabackup` could append to this file!
  - No need for `mariabackup --prepare` before server startup!

- For the circular file, allow arbitrary block size (e.g., 1 to 16,384 bytes)
  - Write special ‘ignore next N bytes’ records when writing an incomplete block, observing the block size of the underlying storage. Avoids initializing pad bytes!
  - Encrypt records and compute checksums before acquiring mutex for copying!
  - `mtr_t::commit()` could copy directly to a memory-mapped file?
Conclusion

- MariaDB Server 10.5 makes better use of the available hardware resources
  - Useless or harmful parameters were removed, others made dynamic.
  - Performance and scalability were improved for various types of workloads.
- **Performance** must never come at the cost of **reliability** or **compatibility**
  - Our stress tests are based on some formal methods and state-of-the-art tools.
  - We also test in-place upgrades of existing data files.
- Watch out for more improvements in future releases
Thoughts on Testing
Concurrency is Hard

- Global locks around entire subsystems will easily guarantee correctness
  - It is easy to read and write sequential (single-threaded) algorithms.
  - But, a coarse lock or mutex will destroy any concurrency!
  - Multi-core CPUs demand fine-grained locking and multi-threaded execution.

- We need a **machine-readable specification** to catch errors
  - Assertions in debug builds
  - AddressSanitizer (ASAN) and MemorySanitizer (MSAN) with custom instrumentation

- Regression test (**mtr**) on CI systems; manual testing with random input
Repeatable Execution Traces of Failures

- [https://rr-project.org](https://rr-project.org) by the Mozilla Foundation records an execution trace that can be used for deterministic debugging with `rr replay`
  - Breakpoints and watchpoints will work and can catch data races!
  - Much smaller than core dumps, even though all intermediate states are included.
- Even the most nondeterministic bugs become tractable and fixable
  - Recovery bugs: need a trace of the killed server and the recovering server.
  - We recently found and fixed several elusive 10-year-old bugs.
- Best of all, this can be combined with ASAN and Random Query Generator
Performance Testing

- Performance regressions can be hard to catch due to huge variation of types of workload and hardware
  - Read-only vs. read-mostly vs. write-heavy
  - Small buffer pool vs. large buffer pool (in-memory workload)
  - Different storage characteristics: HDD, SSD, NAS, PMEM

- MariaDB Server 10.5 generally improves performance
  - We have identified some bottlenecks.
  - This is work in progress.