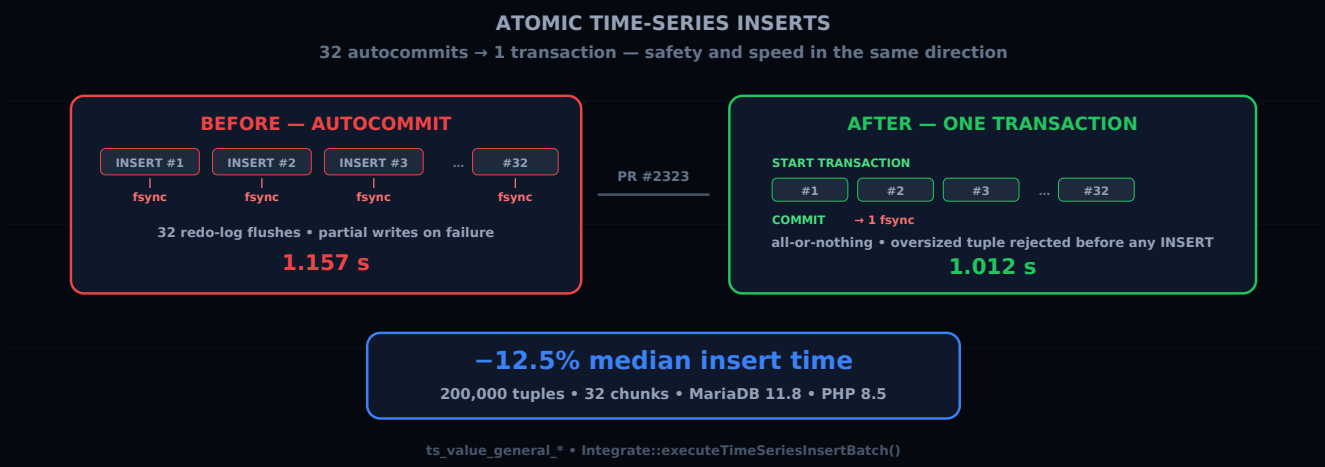
When an Atomicity Fix Makes Ingestion 12% Faster

Aurélien LEQUOY · July 25, 2026

MARIADB MYSQL INNODB TIME-SERIES PERFORMANCE TRANSACTIONS PMACONTROL



The Context: PmaControl's Hottest Write Path

Every few seconds, each PmaControl agent collects hundreds of metrics from your MariaDB / MySQL servers: status variables, InnoDB counters, replication state, query digests. All of it converges on `Integrate::insert_value()`, which writes those measurements into the `ts_value_general_*` tables — the busiest write path in the application.

To stay safely below `max_allowed_packet`, tuples are grouped into **chunks**: the SQL budget is computed dynamically from the server (with a safety margin), and a batch of 200,000 measurements becomes, say, 32 `INSERT INTO ... VALUES (...), (...), ...` statements of roughly 256 KiB each.

Until now, those 32 statements ran in **autocommit** mode: 32 INSERTs, 32 implicit COMMITs.

The Bug: Silent Partial Writes

What happens if chunk 17 fails — table full, deadlock, lost connection?

Before the fix: chunks 1 through 16 stayed written, chunks 17 through 32 were lost, and the downstream bookkeeping (server/variable linking, ingestion checkpoint) could run as if everything had succeeded. A logical batch turned into a silent partial write — the worst possible scenario for monitoring data: charts with holes that nothing ever reports.

Three distinct issues converged on this same root cause:

- **#1467 / #1471** — partial writes when a failure hits mid-batch;
- **#1468 / #1472** — a single tuple larger than the budget was still sent to the server, guaranteed to fail on `max_allowed_packet`;
- **#1469** — a silent fallback could disable dynamic budget detection.

The Fix: One Batch = One Transaction

The fix (PR #2323) introduces `executeTimeSeriesInsertBatch()`: all chunks of a given metric type are now wrapped in **a single transaction**:

```
START TRANSACTION;
INSERT INTO ts_value_general_int (...) VALUES (...), (...), ...;  -- chunk 1
INSERT INTO ts_value_general_int (...) VALUES (...), (...), ...;  -- chunk 2
-- ... 30 more chunks ...
COMMIT;
```

Any failure — falsy result, non-benign warning, exception — triggers a `ROLLBACK` and raises a typed exception carrying safe diagnostic metadata (table, chunk, estimated bytes, budget). The measurement file is kept for replay: either the whole batch is persisted, or none of it is.

As a bonus, an isolated tuple whose estimated size already exceeds the budget is detected **before** the transaction even opens: no INSERT that is doomed to fail is ever sent to the server.

The 12% Question: What Does It Cost?

A transaction spanning 32 statements means extra accounting on the InnoDB side: a longer undo log, locks held longer. We expected to pay a small overhead, which we considered well worth the atomicity guarantee.

We benchmarked before merging. Protocol:

- an **exact** replay of the `insert_value()` hot loop (chunking → SQL build → execution), not a synthetic micro-benchmark;
- 200,000 deterministic tuples into `ts_value_general_int` (unique primary keys), 256 KiB budget → 32 chunks;
- MariaDB 11.8, PHP 8.5.8, InnoDB table, `TRUNCATE` between runs, 1 warmup + 5 measured runs;
- same LXC container, same data, only the application code changes between the two measurements.

Results:

Run	Before (autocommit ×32)	After (1 transaction)
1	1.127 s	0.927 s
2	1.199 s	0.947 s
3	1.155 s	1.012 s
4	1.168 s	1.019 s
5	1.157 s	1.026 s
Median	**1.157 s**	**1.012 s**

12.5% faster on the median. The atomicity fix costs nothing: it saves time. Ingestion throughput goes from roughly 173,000 to 198,000 measurements per second.

Why It's Faster: The Hidden Price of COMMIT

The answer lies in what a `COMMIT` actually does under InnoDB.

On every commit, InnoDB must make the transaction **durable**: write the redo log pages and — with `innodb_flush_log_at_trx_commit = 1`, the default and the only truly safe value — force an `fsync()` of the log file. That flush is the most expensive operation in a transaction's life cycle: it physically waits for storage.

In autocommit mode, every `INSERT` is its own transaction:

- **before**: 32 chunks = 32 COMMITs = 32 redo log flushes;
- **after**: 32 chunks = 1 COMMIT = 1 redo log flush.

The 31 saved synchronizations outweigh the slightly longer undo log by a wide margin. It's the same mechanism that makes a SQL import dramatically faster when wrapped in a transaction, or that lets MariaDB's *group commit* amortize flushes across concurrent transactions — applied here inside a single logical batch.

Note that the gain depends on hardware: on storage where `fsync()` is very slow (spinning disks, virtualization without a safe write cache), the gap widens further. On fast NVMe it narrows. But the sign never flips: the single transaction is always at least as fast.

Takeaways

- **Atomicity is not a luxury you pay for; it's often an optimization.** Grouping related writes into one transaction eliminates redo log flushes — safety and performance point in the same direction.
- **Benchmark the real code path.** Replaying the actual hot loop against a real database is what turned "we accept the overhead" into "there is no overhead."
- **Failure must be loud and total.** A half-written metrics batch is worse than a replayed one: since this fix, PmaControl guarantees all-or-nothing time-series ingestion.

This fix has been available on PmaControl's `master` branch since July 25, 2026.