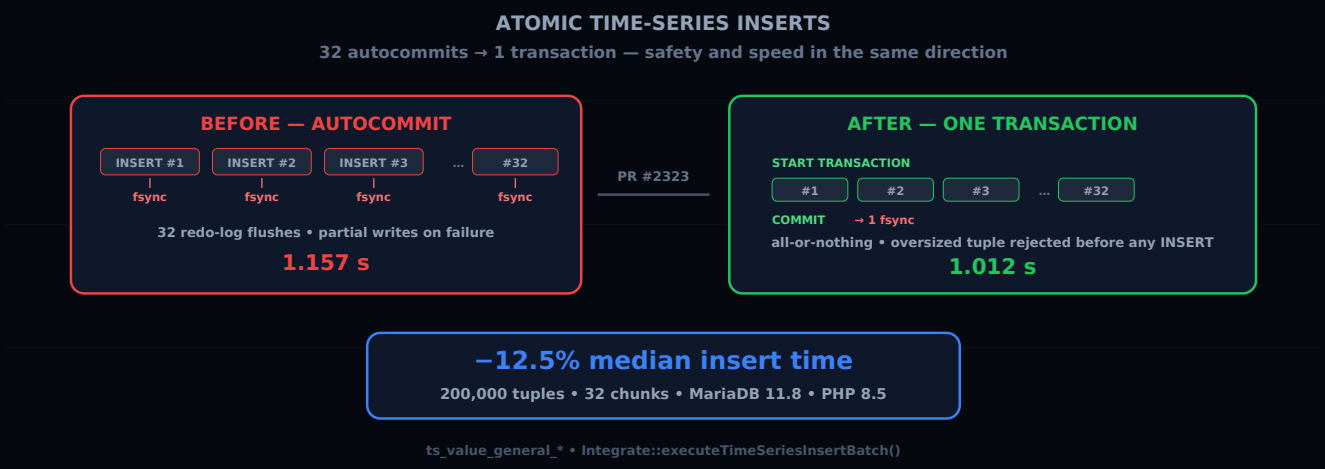
Quand un fix d'atomicité rend l'ingestion 12 % plus rapide

Aurélien LEQUOY · 25 juillet 2026

MARIADB MYSQL INNODB TIME-SERIES PERFORMANCE TRANSACTIONS PMACONTROL



Le contexte : le chemin d'écriture le plus chaud de PmaControl

Toutes les quelques secondes, chaque agent PmaControl collecte des centaines de métriques sur vos serveurs MariaDB / MySQL : variables de statut, compteurs InnoDB, état de réplication, digests de requêtes. Tout cela converge vers `Integrate::insert_value()`, qui écrit ces mesures dans les tables `ts_value_general_*` — le chemin d'écriture le plus sollicité de l'application.

Pour ne jamais dépasser `max_allowed_packet`, les tuples sont regroupés en **chunks** : le budget SQL est calculé dynamiquement à partir du serveur (avec une marge de sécurité), et un lot de 200 000 mesures devient par exemple 32 requêtes `INSERT INTO ... VALUES (...), (...), ...` d'environ 256 Kio chacune.

Jusqu'ici, ces 32 requêtes partaient en **autocommit** : 32 INSERT, 32 COMMIT implicites.

Le bug : des écritures partielles silencieuses

Que se passe-t-il si le chunk 17 échoue — table pleine, deadlock, connexion perdue ?

Avant le correctif : les chunks 1 à 16 restaient écrits, les chunks 17 à 32 étaient perdus, et le bookkeeping aval (liaison serveur/variable, checkpoint d'ingestion) pouvait s'exécuter comme si tout avait réussi. Un lot logique devenait une écriture partielle silencieuse — le pire des scénarios pour des données de supervision : des graphiques avec des trous que rien ne signale.

Trois issues distinctes convergeaient vers cette même racine :

- **#1467 / #1471** — écritures partielles en cas d'échec au milieu d'un lot ;
- **#1468 / #1472** — un tuple unique plus gros que le budget était quand même envoyé au serveur, pour échouer à coup sûr sur `max_allowed_packet` ;
- **#1469** — un fallback silencieux pouvait désactiver la détection dynamique du budget.

Le fix : un lot = une transaction

La correction (PR #2323) introduit `executeTimeSeriesInsertBatch()` : tous les chunks d'un même type de métrique sont désormais enveloppés dans **une seule transaction** :

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

Le moindre échec — résultat falsy, warning non bénin, exception — déclenche un `ROLLBACK` et remonte une exception typée avec des métadonnées de diagnostic sûres (table, chunk, octets estimés, budget). Le fichier de mesures est conservé pour rejeu : soit tout le lot est persisté, soit rien.

En prime, un tuple isolé dont la taille estimée dépasse déjà le budget est détecté **avant** d'ouvrir la transaction : plus aucun INSERT voué à l'échec n'est envoyé au serveur.

La question à 12 % : combien ça coûte ?

Une transaction qui englobe 32 requêtes, c'est de la comptabilité supplémentaire côté InnoDB : undo log plus long, verrous conservés plus longtemps. On s'attendait à payer un léger surcoût, qu'on jugeait largement justifié par la garantie d'atomicité.

On a benché avant de merger. Protocole :

- rejeu **exact** du hot loop d'`insert_value()` (chunking → construction SQL → exécution), pas un micro-bench synthétique ;
- 200 000 tuples déterministes dans `ts_value_general_int` (clés primaires uniques), budget 256 Kio → 32 chunks ;
- MariaDB 11.8, PHP 8.5.8, table InnoDB, `TRUNCATE` entre les runs, 1 warmup + 5 runs mesurés ;
- même conteneur LXC, mêmes données, seul le code applicatif change entre les deux mesures.

Résultats :

Run	Avant (autocommit ×32)	Après (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
Médiane	**1,157 s**	**1,012 s**

−12,5 % sur la médiane. Le fix d'atomicité ne coûte rien : il fait gagner du temps. Le débit d'ingestion passe d'environ 173 000 à 198 000 mesures par seconde.

Pourquoi c'est plus rapide : le prix caché du COMMIT

La réponse tient dans ce que fait réellement un `COMMIT` sous InnoDB.

À chaque validation, InnoDB doit rendre la transaction **durable** : écrire les pages de redo log et — avec `innodb_flush_log_at_trx_commit = 1`, la valeur par défaut et la seule vraiment sûre — forcer un `fsync()` du fichier de log. Ce flush est l'opération la plus chère du cycle de vie d'une transaction : il attend physiquement le stockage.

En autocommit, chaque `INSERT` est sa propre transaction :

- **avant** : 32 chunks = 32 COMMIT = 32 flushes de redo log ;
- **après** : 32 chunks = 1 COMMIT = 1 flush de redo log.

Les 31 synchronisations économisées pèsent bien plus lourd que l'undo log un peu plus long. C'est le même mécanisme qui fait qu'un import SQL est dramatiquement plus rapide enveloppé dans une transaction, ou que le *group commit* de MariaDB amortit les flushes entre transactions concurrentes — appliqué ici à l'intérieur d'un seul lot logique.

Notons que le gain dépend du matériel : sur un stockage où `fsync()` est très lent (disques mécaniques, virtualisation sans cache d'écriture sécurisé), l'écart se creuse encore. Sur NVMe rapide, il se resserre. Mais le signe ne change pas : la transaction unique est toujours au moins aussi rapide.

Ce qu'il faut retenir

- **L'atomicité n'est pas un luxe qu'on paie, c'est souvent une optimisation.** Regrouper des écritures liées dans une transaction élimine des flushes de redo log — la sûreté et la performance vont dans le même sens.
- **Benchez le vrai chemin de code.** C'est le rejeu du hot loop réel, contre une vraie base, qui a transformé « on accepte le surcoût » en « il n'y a pas de surcoût ».
- **Un échec doit être bruyant et total.** Un lot de métriques à moitié écrit est pire qu'un lot rejoué : depuis ce correctif, PmaControl garantit le tout-ou-rien sur son ingestion time-series.

Ce correctif est disponible sur la branche `master` de PmaControl depuis le 25 juillet 2026.