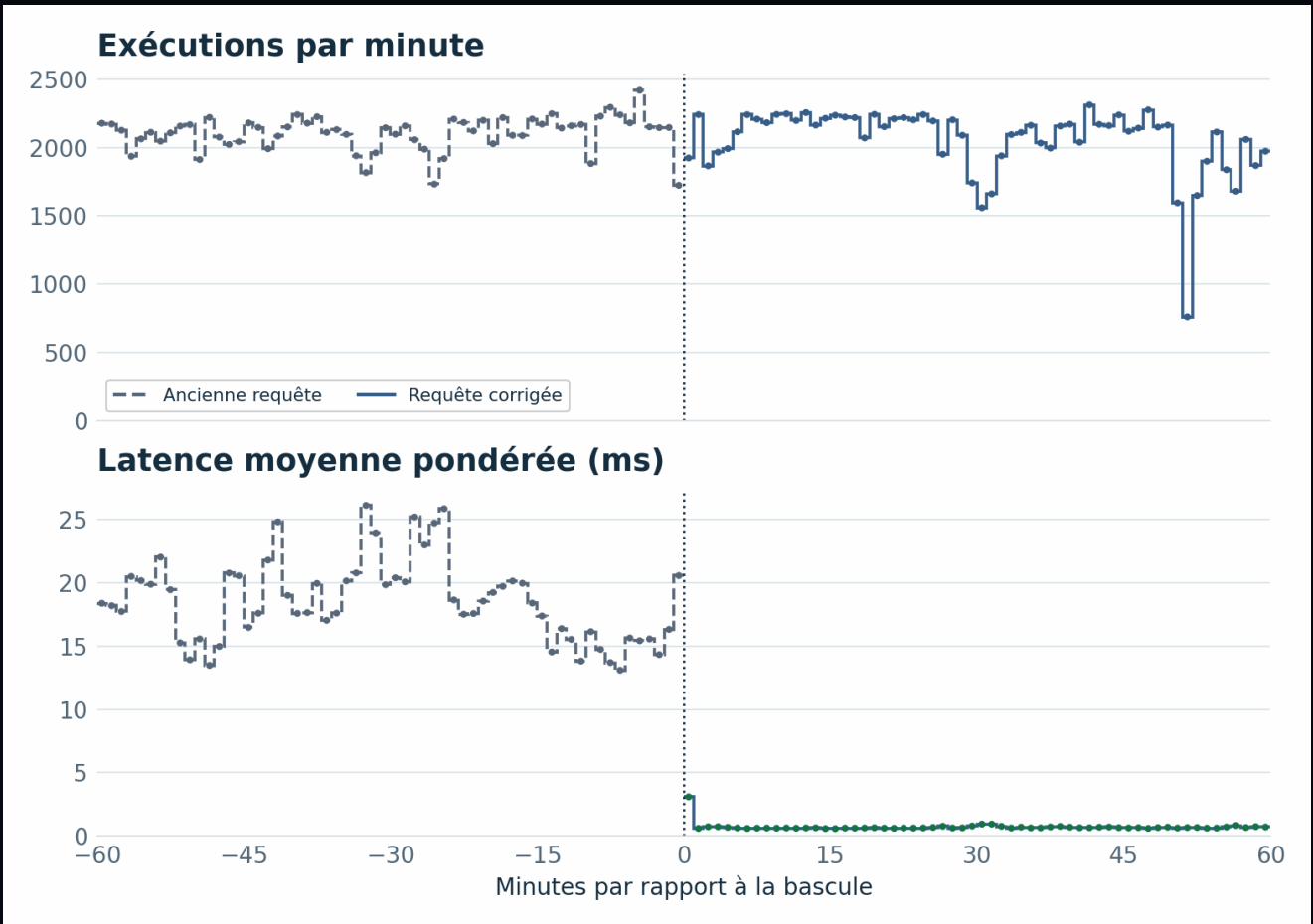
Pourquoi la latence MariaDB chute de 18,43 à 0,70 ms alors que le nom trouve déjà une seule ligne

Aurélien LEQUOY · 21 novembre 2026

PMACONTROL MARIADB PERFORMANCE SQL BENCHMARK OBSERVABILITY



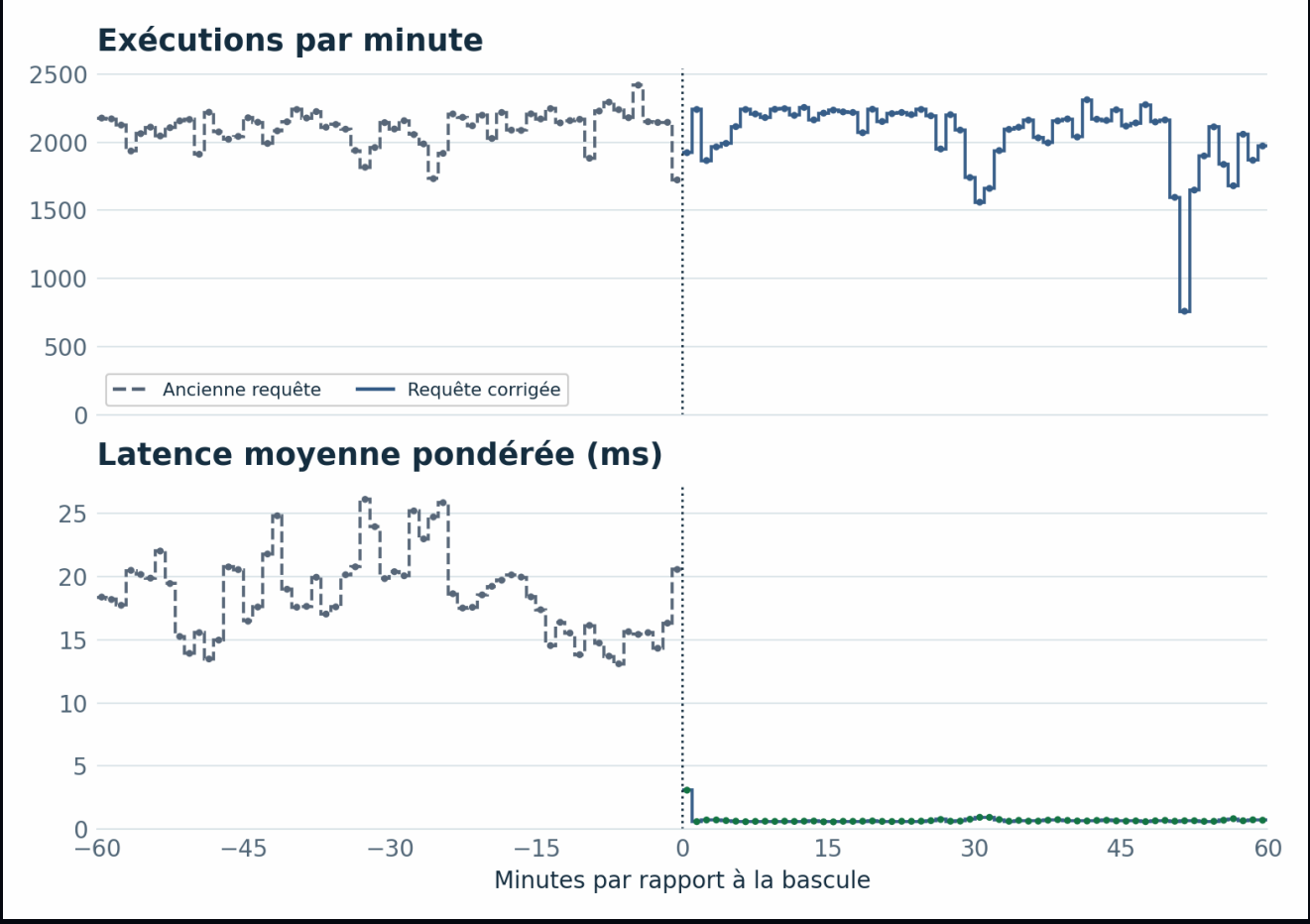
Avec **3 759 noms distincts pour 3 791 lignes**, le champ `name` de `ts_variable` faisait déjà presque tout le tri. Pour `mysql_available`, il trouvait exactement une ligne. Avec `v.`from` = 'mysql_server'`, on retrouvait la même ligne. On pouvait donc s'attendre à un gain négligeable. Pourtant, à environ 2 100 appels par minute, **la latence moyenne en production est passée de 18,43 à 0,70 ms**. Qu'est-ce que MariaDB a gagné si le second filtre n'a écarté aucune ligne de plus pour cette métrique ?

Avant d'enregistrer la disponibilité d'un service, PmaControl relit sa dernière valeur pour déterminer si son état a changé. Sur l'heure qui a précédé le déploiement, cette petite lecture a

mobilisé **2 326 secondes de temps SQL cumulé**. L'heure suivante, elle en a consommé **87**, pour un volume d'appels comparable. Une seule condition de jointure a changé.

Deux heures autour de la bascule

Le 21 novembre 2026 à **17 h 26 min 47 s, heure de Paris**, le collecteur a commencé à utiliser la requête corrigée. Modifier le texte SQL a créé une nouvelle empreinte dans Performance Schema. Si on suit uniquement l'ancienne, la courbe tombe à zéro. Pour suivre le service rendu, nous additionnons les variations des compteurs des deux empreintes sur des tranches d'une minute.



Le graphique comporte 120 points par série, un par minute. En haut figure le nombre d'exécutions de chaque minute. En bas, la latence moyenne de ces mêmes appels. Le pointillé vertical marque le déploiement. Les quelques appels de l'autre version autour de la bascule restent comptés.

Dans les tableaux, le vert (✓) signale un gain et le rouge (✗) une régression lorsque le rapport entre les mesures dépasse strictement 2 dans l'un ou l'autre sens. Pour une durée, l'écart doit

aussi atteindre 1 ms. Les volumes d’appels décrivent la charge. Le seuil sert à la lecture des résultats. Sur les graphes, les deux périodes sont distinguées par la ligne. Après la bascule, chaque point de latence est comparé à la moyenne pondérée de l’heure précédente. Le cumul utilise sa référence au même volume, et chaque paire du benchmark son propre temps initial.

Indicateur	Heure précédente	Heure suivante
Exécutions estimées	126 229	123 241
Exécutions par minute	2 104	2 054
Latence moyenne pondérée	18,43 ms	✓ **0,70 ms (÷26,25)**
Secondes SQL cumulées	2 326,5	✓ **86,5 (÷26,88)**
Exécutions par seconde SQL	54	✓ **1 424 (×26,25)**
Lignes examinées par appel	2,99	✓ **1,00 (÷2,98)**

Le rendement est le nombre de lectures réalisées par seconde de temps SQL cumulé. Il passe de **54 à 1 424**. La latence baisse de **96,2 %**, alors que le débit observé recule de seulement **2,4 %**. Ce rendement mesure le coût de la lecture. Le débit réel dépend aussi de la cadence des sondes et du parallélisme.

Ce que cherche la requête

Pour le serveur demandé, `ts_max_date` donne la dernière date du fichier de mesures. `ts_file` identifie ce fichier et `ts_variable` la métrique. `mysql_server` écarte les serveurs supprimés. La jointure externe cherche ensuite la valeur entière au même instant pour la même variable et le même serveur. Si la mesure manque, cette dernière jointure peut renvoyer `NULL`.

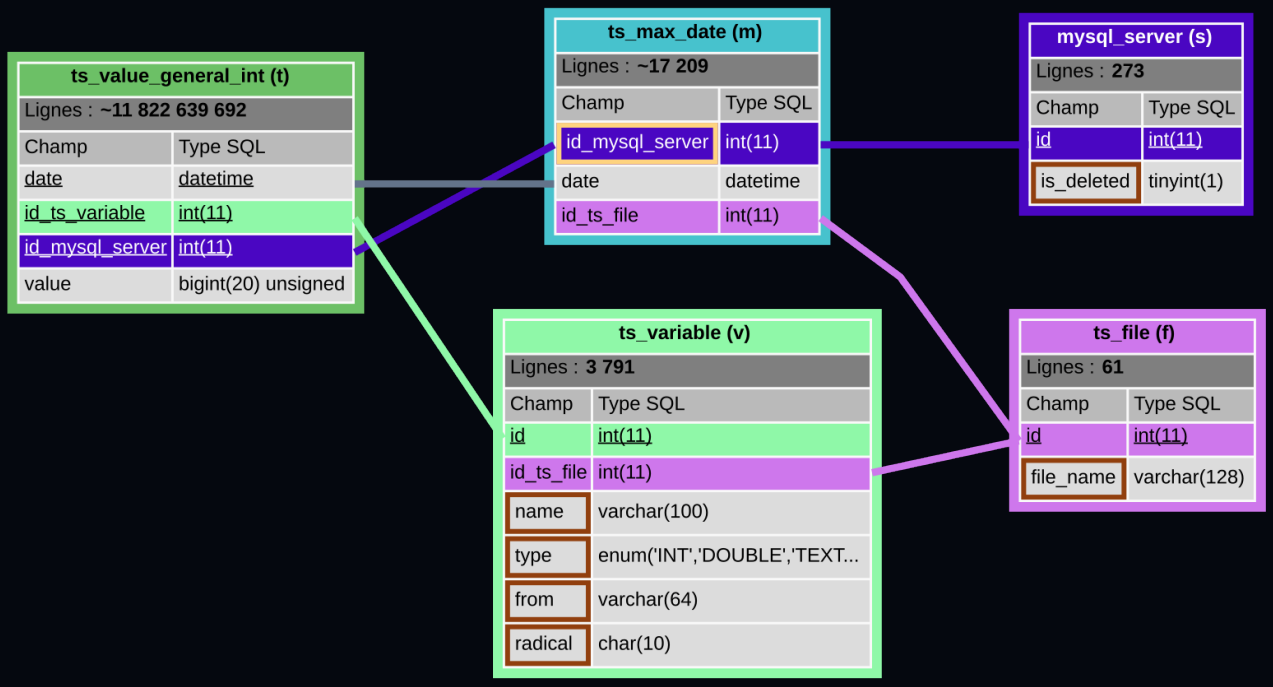


Schéma de la requête corrigée, extrait de PmaControl. Les flèches relient les colonnes de jointure. Le contour brun repère les champs filtrés, dont `ts_variable.from`. Le signe `~` indique un nombre de lignes estimé au moment du relevé.

Définitions des cinq tables

Ces définitions proviennent de `SHOW CREATE TABLE` sur l'instance MariaDB étudiée. Dans `ts_variable`, la ligne `UNIQUE KEY` donne les deux colonnes utilisées par la requête corrigée, dans l'ordre de l'index.

`SHOW CREATE TABLE ts_variable;`

```
CREATE TABLE `ts_variable` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `id_ts_file` int(11) NOT NULL DEFAULT 3,
  `name` varchar(100) NOT NULL,
  `type` enum('INT','DOUBLE','TEXT','JSON','') NOT NULL COMMENT '1 => int, 2 => double, 3 =>
text, 4=> json',
  `from` varchar(64) NOT NULL,
  `radical` char(10) NOT NULL,
  `is_derived` int(11) DEFAULT 1,
  `is_dynamic` int(11) DEFAULT 1 COMMENT 'only for global variables',
  PRIMARY KEY (`id`),
  UNIQUE KEY `name` (`name`,`from`),
  KEY `ts_variable_ibfk_1` (`id_ts_file`),
```

```
  KEY `from` (`from`),
  CONSTRAINT `ts_variable_ibfk_1` FOREIGN KEY (`id_ts_file`) REFERENCES `ts_file` (`id`) ON
DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=39387 DEFAULT CHARSET=latin1 COLLATE=latin1_swedish_ci;
```

SHOW CREATE TABLE ts_file;

```
CREATE TABLE `ts_file` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `file_name` varchar(128) NOT NULL,
  `each` int(11) NOT NULL DEFAULT 1,
  PRIMARY KEY (`id`),
  UNIQUE KEY `file_name` (`file_name`)
) ENGINE=InnoDB AUTO_INCREMENT=25508473 DEFAULT CHARSET=latin1 COLLATE=latin1_swedish_ci;
```

SHOW CREATE TABLE ts_max_date;

```
CREATE TABLE `ts_max_date` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `id_mysql_server` int(11) NOT NULL,
  `date` datetime DEFAULT NULL,
  `sample_epoch` bigint(20) unsigned NOT NULL DEFAULT 0,
  `date_p1` datetime DEFAULT NULL,
  `sample_epoch_p1` bigint(20) unsigned NOT NULL DEFAULT 0,
  `date_p2` datetime DEFAULT NULL,
  `sample_epoch_p2` bigint(20) unsigned NOT NULL DEFAULT 0,
  `date_p3` datetime DEFAULT NULL,
  `sample_epoch_p3` bigint(20) unsigned NOT NULL DEFAULT 0,
  `date_p4` datetime DEFAULT NULL,
  `sample_epoch_p4` bigint(20) unsigned NOT NULL DEFAULT 0,
  `id_ts_file` int(11) NOT NULL,
  `last_date_listener` datetime DEFAULT current_timestamp(),
  `last_sample_epoch_listener` bigint(20) unsigned NOT NULL DEFAULT 0,
  PRIMARY KEY (`id`),
  UNIQUE KEY `id_mysql_server_2` (`id_mysql_server`,`id_ts_file`),
  UNIQUE KEY `id_mysql_server` (`id_mysql_server`,`id_ts_file`,`date`),
  UNIQUE KEY `date` (`date`,`id_ts_file`,`id_mysql_server`),
  KEY `id_ts_file` (`id_ts_file`),
  KEY `id_mysql_server_4` (`id_mysql_server`,`date`),
  KEY `date_2` (`date`,`last_date_listener`),
  CONSTRAINT `ts_max_date_ibfk_3` FOREIGN KEY (`id_ts_file`) REFERENCES `ts_file` (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=22859 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
```

`SHOW CREATE TABLE mysql_server;`

```
CREATE TABLE `mysql_server` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `is_deleted` tinyint(1) NOT NULL DEFAULT 0,
  `created_at` datetime NOT NULL DEFAULT current_timestamp(),
  `updated_at` datetime NOT NULL DEFAULT current_timestamp() ON UPDATE current_timestamp(),
  `id_client` int(11) NOT NULL,
  `id_environment` int(11) NOT NULL,
  `name` varchar(100) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_ci NOT NULL,
  `display_name` varchar(100) NOT NULL,
  `ip` varchar(255) CHARACTER SET latin1 COLLATE latin1_bin NOT NULL,
  `hostname` varchar(200) NOT NULL DEFAULT '',
  `login` varchar(32) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_ci NOT NULL,
  `passwd` varchar(255) CHARACTER SET latin1 COLLATE latin1_bin NOT NULL,
  `database` varchar(64) NOT NULL,
  `is_password_crypted` int(11) NOT NULL,
  `is_ssl` int(11) NOT NULL DEFAULT 0,
  `port` int(11) NOT NULL,
  `ssh_nat` varchar(64) CHARACTER SET latin1 COLLATE latin1_bin NOT NULL DEFAULT '',
  `ssh_port` int(11) NOT NULL DEFAULT 22,
  `ssh_login` text CHARACTER SET latin1 COLLATE latin1_bin NOT NULL DEFAULT '\'\'',
  `is_sudo` int(11) NOT NULL DEFAULT 0,
  `is_root` int(11) NOT NULL DEFAULT 1,
  `is_monitored` int(11) NOT NULL DEFAULT 1,
  `monitoring_disabled_reason` enum('manual','license_quota') DEFAULT NULL,
  `is_proxy` int(11) NOT NULL DEFAULT 0,
  `is_vip` tinyint(1) NOT NULL DEFAULT 0,
  `is_binlog_relay` tinyint(1) NOT NULL DEFAULT 0,
  `is_clickhouse` tinyint(1) NOT NULL DEFAULT 0,
  `errant_gtid_set` text DEFAULT NULL COMMENT '#1280 P0.1 last computed errant GTID set
(canonical form, empty string = no errant)',
  `is_acknowledged` int(11) NOT NULL DEFAULT 0,
  `timeout` int(11) GENERATED ALWAYS AS (case when `is_proxy` = 1 then 11 else 1 end)
VIRTUAL,
  `errant_gtid_sampled_at` datetime DEFAULT NULL COMMENT '#1280 P0.1 wall-clock of the last
successful errant-set computation',
  `errant_gtid_ignore` tinyint(1) NOT NULL DEFAULT 0,
  `replica_lag_sla_seconds` int(11) NOT NULL DEFAULT 30,
  `is_out_of_service` tinyint(1) NOT NULL DEFAULT 0 COMMENT 'operator-flagged HS for reload
from master',
  PRIMARY KEY (`id`),
```

```
  UNIQUE KEY `name` (`name`),
  UNIQUE KEY `ip` (`ip`,`port`),
  KEY `id_client` (`id_client`),
  KEY `id_environment` (`id_environment`),
  KEY `is_monitored` (`id_client`,`is_monitored`,`is_deleted`) USING BTREE,
  KEY `is_deleted` (`is_deleted`)
) ENGINE=InnoDB AUTO_INCREMENT=337 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci WITH
SYSTEM VERSIONING
 PARTITION BY SYSTEM_TIME
PARTITIONS 2;
```

SHOW CREATE TABLE ts_value_general_int;

```
CREATE TABLE `ts_value_general_int` (
  `date` datetime NOT NULL,
  `id_ts_variable` int(11) NOT NULL,
  `id_mysql_server` int(11) NOT NULL,
  `value` bigint(20) unsigned DEFAULT NULL,
  PRIMARY KEY (`date`,`id_ts_variable`,`id_mysql_server`),
  KEY `id_mysql_server` (`id_mysql_server`,`id_ts_variable`,`date`)
) ENGINE=ROCKSDB DEFAULT CHARSET=latin1 COLLATE=latin1_swedish_ci
 PARTITION BY RANGE (to_days(`date`))
(PARTITION `p740279` VALUES LESS THAN (740279) ENGINE = ROCKSDB,
 PARTITION `p740280` VALUES LESS THAN (740280) ENGINE = ROCKSDB,
 PARTITION `p740281` VALUES LESS THAN (740281) ENGINE = ROCKSDB,
 PARTITION `p740282` VALUES LESS THAN (740282) ENGINE = ROCKSDB,
 PARTITION `p740283` VALUES LESS THAN (740283) ENGINE = ROCKSDB,
 PARTITION `p740284` VALUES LESS THAN (740284) ENGINE = ROCKSDB,
 PARTITION `p740285` VALUES LESS THAN (740285) ENGINE = ROCKSDB,
 PARTITION `p740286` VALUES LESS THAN (740286) ENGINE = ROCKSDB,
 PARTITION `p740287` VALUES LESS THAN (740287) ENGINE = ROCKSDB,
 PARTITION `p740288` VALUES LESS THAN (740288) ENGINE = ROCKSDB,
 PARTITION `p740289` VALUES LESS THAN (740289) ENGINE = ROCKSDB,
 PARTITION `p740290` VALUES LESS THAN (740290) ENGINE = ROCKSDB,
 PARTITION `p740291` VALUES LESS THAN (740291) ENGINE = ROCKSDB,
 PARTITION `p740292` VALUES LESS THAN (740292) ENGINE = ROCKSDB,
 PARTITION `p740293` VALUES LESS THAN (740293) ENGINE = ROCKSDB,
 PARTITION `p740294` VALUES LESS THAN (740294) ENGINE = ROCKSDB,
 PARTITION `p740295` VALUES LESS THAN (740295) ENGINE = ROCKSDB,
 PARTITION `p740296` VALUES LESS THAN (740296) ENGINE = ROCKSDB,
 PARTITION `p740297` VALUES LESS THAN (740297) ENGINE = ROCKSDB,
```

```
 PARTITION `p740298` VALUES LESS THAN (740298) ENGINE = ROCKSDB,
 PARTITION `p740299` VALUES LESS THAN (740299) ENGINE = ROCKSDB,
 PARTITION `p740300` VALUES LESS THAN (740300) ENGINE = ROCKSDB,
 PARTITION `p740301` VALUES LESS THAN (740301) ENGINE = ROCKSDB,
 PARTITION `p740302` VALUES LESS THAN (740302) ENGINE = ROCKSDB,
 PARTITION `p740303` VALUES LESS THAN (740303) ENGINE = ROCKSDB,
 PARTITION `p740304` VALUES LESS THAN (740304) ENGINE = ROCKSDB,
 PARTITION `p740305` VALUES LESS THAN (740305) ENGINE = ROCKSDB,
 PARTITION `p740306` VALUES LESS THAN (740306) ENGINE = ROCKSDB,
 PARTITION `p740307` VALUES LESS THAN (740307) ENGINE = ROCKSDB,
 PARTITION `p740308` VALUES LESS THAN (740308) ENGINE = ROCKSDB,
 PARTITION `p740309` VALUES LESS THAN (740309) ENGINE = ROCKSDB,
 PARTITION `p740310` VALUES LESS THAN (740310) ENGINE = ROCKSDB);
```

La correction dans le collecteur

L'index unique de `ts_variable`, nommé `name`, porte sur **(`name`, `from`)**. Il autorise plusieurs sources pour un nom de métrique, tout en garantissant une seule ligne pour un couple donné ici. Le code connaissait déjà la source à l'écriture. À la lecture, il filtrait le nom, le type et le radical, mais omettait la source. La version déployée réutilise cette source. Voici le chemin de disponibilité MySQL avec l'identifiant de serveur paramétré.

```
SELECT t.value AS previous_value
FROM ts_max_date AS m
JOIN ts_file AS f ON f.id = m.id_ts_file
JOIN ts_variable AS v ON v.id_ts_file = f.id
  AND v.name = 'mysql_available'
  AND v.`from` = 'mysql_server'  /* + */
  AND v.type = 'INT' AND v.radical = 'general'
JOIN mysql_server AS s ON s.id = m.id_mysql_server
  AND s.is_deleted = 0
LEFT JOIN ts_value_general_int AS t
  ON t.id_mysql_server = m.id_mysql_server
 AND t.id_ts_variable = v.id
 AND t.date = m.date
WHERE m.id_mysql_server = ? AND f.file_name = 'mysql_server'
LIMIT 1;
```

La ligne marquée « + » est la seule addition. `from` est ici un nom de colonne, protégé par des accents graves. Sur ce chemin, sa valeur est `mysql_server`. Les autres collecteurs emploient leur propre source. Aucun index n'a été créé pendant la bascule.

La source distingue seulement 32 groupes de plus sur 3 791 lignes

Sur les 3 791 lignes de `ts_variable`, nous avons compté les groupes dans un même instantané en lecture seule, le 21 novembre 2026 à 21 h 25, heure de Paris. La proportion ci-dessous rapporte le nombre de groupes distincts au nombre de lignes de la table.

Colonnes regroupées	Groupes exacts	Groupes / lignes	`SHOW INDEX` avant → après
(`name`)	3 759	99,16 %	3 841 → 3 840
(`name` , `from`)	3 791	100 %	3 841 → 3 840
(`from`)	42	1,11 %	42 → 42

Les deux premières lignes correspondent aux préfixes successifs de l'index unique `name`. La troisième décrit l'index séparé sur `from`. Au premier relevé, MariaDB affichait **3 841 aux deux positions** de l'index unique. Cette statistique est une estimation et peut dépasser le nombre réel de lignes. Documentation MariaDB sur la cardinalité.

Le comptage explique l'intuition de départ. **3 729 noms** apparaissent une fois, **28** deux fois et **2** trois fois. Regrouper par (`name` , `from`) ajoute seulement **32 groupes** sur toute la table. Le rapport entre groupes distincts et lignes passe de **99,16 % à 100 %**, soit **0,84 point**. Pour la métrique de la sonde, même ce petit écart disparaît. `mysql_available` renvoie **une ligne** avec le nom seul et **la même ligne** avec la source.

```
SELECT rows_per_name, COUNT(*) AS number_of_names
FROM (
    SELECT name, COUNT(*) AS rows_per_name
    FROM ts_variable
    GROUP BY name
) AS names
GROUP BY rows_per_name
ORDER BY rows_per_name;
```

Après actualisation des statistiques de `ts_variable`

À 21 h 30, heure de Paris, `ANALYZE NO_WRITE_TO_BINLOG TABLE pmacontrol.ts_variable` a renvoyé `OK` sur l'instance étudiée. `SHOW INDEX` est passé de 3 841 à **3 840 pour les deux préfixes**. Le `GROUP BY` refait juste après donne toujours les mêmes cardinalités exactes.

Les statistiques persistantes d'InnoDB détaillent les préfixes dans `mysql.innodb_index_stats`. Pour l'index `name`, `n_diff_pfx01` est passé de 3 757 à **3 759** et `n_diff_pfx02` de 3 789 à **3 791**. Ces deux estimations concordent désormais avec le comptage exact. Avant comme après l'actualisation, les deux lectures isolées de `ts_variable` estiment une ligne dans `EXPLAIN` et conservent leurs accès respectifs, `ref` par nom et `const` par couple.

Requêtes de comptage et résultats bruts (ZIP)

Comparaison des plans d'exécution

Avant, MariaDB utilise déjà l'index `name` avec un accès `ref` limité à sa première colonne. Après, les deux égalités renseignent toute la clé unique et l'accès à `v` devient `const`. Le plan mesuré utilise 102 puis 168 octets de clé. La lecture de la valeur `t` passe également de `eq_ref` à `const` sur sa clé primaire. Les lignes examinées par appel, environ trois puis une, confirment l'effet sur l'ensemble de la lecture. Le moteur peut reconnaître l'unicité avant de poursuivre les jointures.

Chaque cellule donne le rang de lecture, le type d'accès, l'index choisi et la longueur de clé en octets. Les lignes sont alignées par table.

Table	Avant	Après
`ts_file` (f)	1 · `const` · `file_name` · 130 B	1 · `const` · `file_name` · 130 B
`ts_max_date` (m)	2 · `const` · `id_mysql_server_2` · 8 B	3 · `const` · `id_mysql_server_2` · 8 B
`ts_variable` (v)	3 · `ref` · `name` · 102 B	2 · `const` · `name` · 168 B
`mysql_server` (s)	4 · `ref` · `PRIMARY` · 4 B	5 · `ref` · `PRIMARY` · 4 B
`ts_value_general_int` (t)	5 · `eq_ref` · `PRIMARY` · 13 B	4 · `const` · `PRIMARY` · 13 B

Dans les deux plans, MariaDB estime une ligne pour chacune des cinq tables et liste les mêmes 32 partitions pour `ts_value_general_int`.

Plan complet avant (CSV) · Plan complet après (CSV)

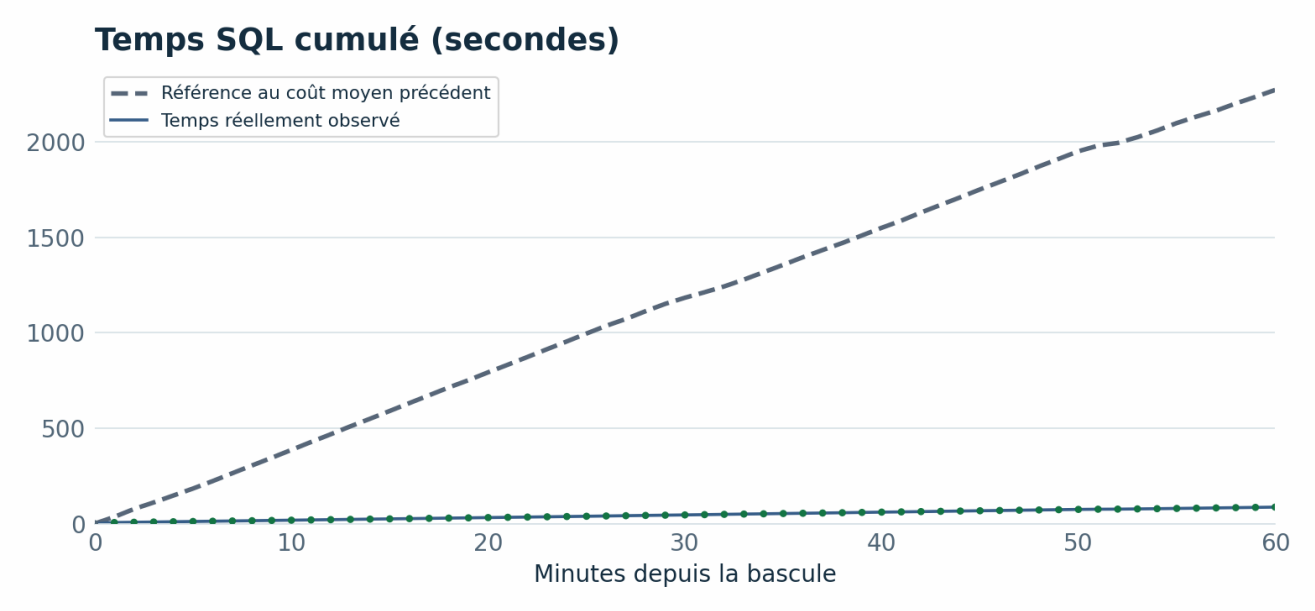
Où le temps a changé

Phase · médiane sur 50 serveurs	Avant	Après
Optimisation	35,085 ms	✓ **0,261 ms (÷134,27)**
Exécution après optimisation	0,107 ms	0,011 ms
Étape `Statistics`	54,983 ms	✓ **0,224 ms (÷245,46)**

Les deux premières lignes proviennent d'`ANALYZE FORMAT=JSON`. La ligne `Statistics` vient d'une autre série mesurée avec `SHOW PROFILE`. Ces durées ont été relevées séparément.

Avec `name` seul, l'optimiseur de MariaDB **estime** qu'il lira une ligne. Avec les deux égalités, l'index unique (`name`, `from`) lui **garantit** qu'il y en aura au plus une. Le moteur peut donc traiter `v.id` comme une constante lorsqu'elle prépare la jointure avec la table des valeurs. Sur 50 serveurs, la médiane tombe de **35,085 à 0,261 ms pour préparer le plan**, et de **0,107 à 0,011 ms** pour l'exécuter ensuite. C'est dans la préparation du plan que se trouve l'essentiel du gain mesuré.

Le temps SQL cumulé depuis le déploiement



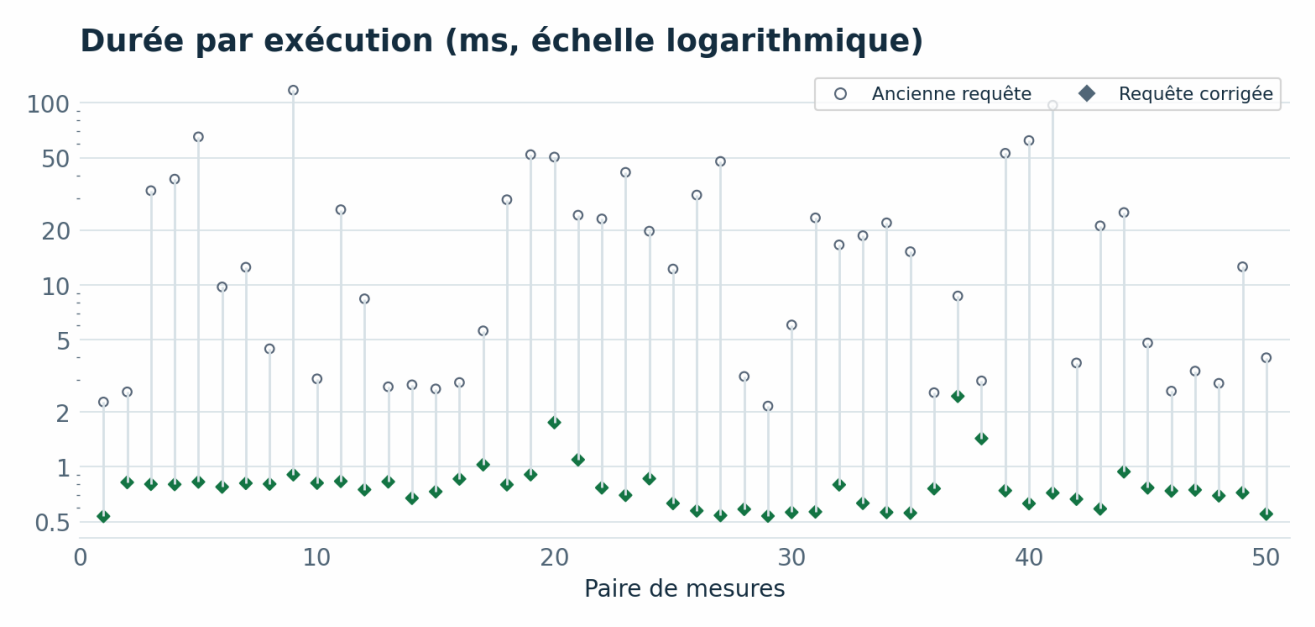
Les deux courbes partent de zéro à la bascule et suivent exactement le même volume d'appels après celle-ci. La courbe pleine cumule le temps mesuré. La référence applique aux appels

réellement reçus le coût moyen de l'heure précédente. À une heure, cela donne **86,5 secondes observées** contre **2 271,4 secondes estimées** au coût précédent, soit **36 min 25 s de temps SQL économisées selon cette estimation**. Les temps SQL peuvent se chevaucher entre connexions et incluent les attentes. Cette estimation ne mesure pas des minutes de CPU.

Le contrôle sur 50 serveurs

Sous MariaDB 11.8.10, cache de requêtes désactivé, nous avons tiré au hasard 50 identifiants distincts parmi 111 donnant une valeur. Chaque paire a lu le même serveur dans une transaction en lecture seule. L'ancienne requête est passée en premier 25 fois, la nouvelle 25 fois. Les durées proviennent de `SHOW PROFILES` côté serveur. Les **50 résultats sont identiques** et la requête corrigée a été plus rapide dans **les 50 paires**.

Variante	Minimum (ms)	Médiane (ms)	Moyenne (ms)	Maximum (ms)
Avant	2,159	12,543	21,686	117,530
Après	✓ **0,535 (÷4,03)**	✓ **0,755 (÷16,62)**	✓ **0,804 (÷26,98)**	✓ **2,446 (÷48,05)**



Chaque colonne relie les deux mesures d'un même serveur. L'échelle logarithmique permet de lire à la fois les valeurs inférieures à 1 ms et le maximum de 117,53 ms. Sur cet échantillon, la moyenne est divisée par 26,98. En production, le rapport des moyennes pondérées des deux heures est de 26,25.

Variantes comparées à la requête déployée

Chaque ligne compare les moyennes de deux requêtes sur les mêmes serveurs, dans le même essai. Chaque série porte sur 50 serveurs. Le nombre entre parenthèses indique les exécutions par variante.

Variante	Référence (ms)	Variante (ms)
Nom seul (150)	0,651	✗ **36,782 (×56,50)**
Nom seul, clé primaire imposée sur les valeurs (150)	0,651	✗ **1,674 (×2,57)**
Clé complète, clé primaire imposée sur les valeurs (150)	0,651	0,630
Index couvrant imposé sur la date (50)	0,692	✗ **20,232 (×29,25)**
Serveur actif vérifié avec `EXISTS` (150)	0,651	0,663
Valeur lue par sous-requête scalaire (150)	0,617	0,546
Ordre des jointures imposé (50)	0,692	✗ **41,737 (×60,34)**
Sous-requête scalaire et `EXISTS` (150)	0,617	0,563
Sous-requête scalaire sans `LIMIT` (150)	0,617	0,560
Jointures déployées sans `LIMIT` (150)	0,617	0,602
Sous-requête scalaire avec le nom seul (150)	0,617	1,218

Mesures anonymisées des variantes et des phases (ZIP)

Comment reproduire la lecture des graphes

La fenêtre va de 16 h 26 min 47 s à 18 h 26 min 47 s, en heure de Paris. Aux bornes des tranches, les compteurs sont interpolés entre relevés. Le compteur d’appels relevé directement à la bascule fixe le raccord. Les 496 appels de la nouvelle empreinte pendant les essais et les 224 derniers appels de l’ancienne après la bascule restent inclus. Le redémarrage des collecteurs vers 18 h 17 affecte aussi la fin de la courbe de volume.

Pour chaque minute, la différence de `COUNT_STAR` donne directement le nombre d’appels. Pour la latence, on divise la différence de `SUM_TIMER_WAIT` par la différence de `COUNT_STAR`, puis par un milliard pour passer des picosecondes aux millisecondes. La moyenne horaire utilise la somme des durées et la somme des appels. Le test apparié isole mieux la modification SQL que les deux

heures d’observation en charge.

Données et plans en téléchargement

- Relevés des compteurs (CSV)
- Séries à la minute et cumul (CSV)
- 50 paires anonymisées (CSV)
- Plans détaillés (CSV)
- Documentation MariaDB sur EXPLAIN