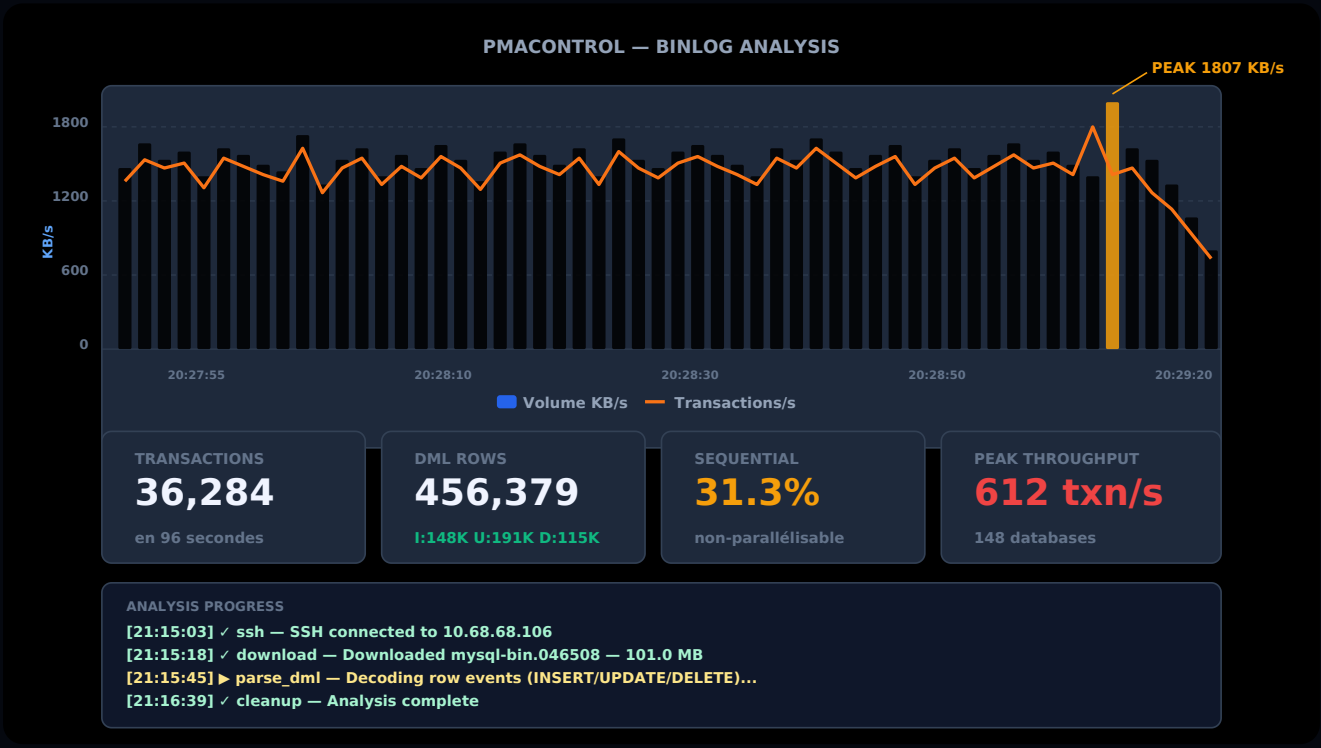
Analyser les binlogs pour comprendre le lag de réplication

Aurélien LEQUOY · 13 avril 2026

PMACONTROL BINLOG REPLICATION MYSQL MARIADB PERCONA-SERVER PERFORMANCE LAG MYSQLBINLOG



Le lag de réplication, ce fléau silencieux

Tout DBA a vécu ce moment : le monitoring affiche 300 secondes de retard sur le replica, les alertes pleuvent, et la question est toujours la même — **pourquoi ?**

Le `Seconds_Behind_Master` (ou `Seconds_Behind_Source` en MySQL 8) est un indicateur binaire : ça lag ou ça lag pas. Il ne dit rien sur la **cause**. Est-ce un batch de DELETE massif ? Un manque de parallélisme ? Une transaction de 800 KB qui bloque le SQL thread ? Pour le savoir, il faut aller disséquer les binlogs du master — ce qui demande un accès SSH, le bon binaire `mysqlbinlog`, du temps et du grep.

PmaControl automatise désormais tout ça.

La nouvelle fonctionnalité : Binlog Analysis

Le principe

Depuis la page **slave/show** de n'importe quel replica, vous pouvez maintenant :

1. **Sélectionner une plage horaire** directement sur le graphe de lag (clic-glissé à la souris) ou via des champs datetime
2. **Lancer l'analyse** en un clic — tout se fait en arrière-plan
3. **Suivre la progression en temps réel** étape par étape
4. **Consulter le rapport** avec graphe interactif et métriques détaillées
5. **Recevoir une notification Telegram** avec le résumé

Ce qui se passe en coulisse

Quand vous lancez une analyse, PmaControl exécute un pipeline en 15 phases :

```
[21:15:02] ✓ init — Analysis #42 — slave=85 master=106 range=[20:27:51 → 20:29:27]
[21:15:02] ✓ master_info — Master: prod-db-01 — 10.68.68.106:3306
[21:15:02] ✓ version — Version: 8.0.44
[21:15:02] ✓ binary — Using: mysqlbinlog-8.0 — Ver 8.0.42
[21:15:03] ✓ credentials — MySQL user: pmacontrol@10.68.68.106:3306
[21:15:04] ✓ find_binlogs — Found 1 binlog file(s): mysql-bin.046508
[21:15:18] ✓ fetch — Fetched mysql-bin.046508 — 101.0 MB (via --read-from-remote-server,
cached)
[21:15:19] ✓ file_ranges — 1 file(s) probed
[21:15:45] ✓ parse_gtid — Parsed 36284 transactions, total 97.2 MB, 123 >100KB, 3 >500KB
[21:16:12] ✓ parse_dml — I:148,659 U:191,867 D:115,853 = 456,379 rows across 148 databases
[21:16:38] ✓ parse_volume — 97 seconds of data, peak 612 txn/s, peak 1807 KB/s
[21:16:39] ✓ parse_ddl — No DDL — 100% DML row-based
[21:16:39] ✓ metadata — 1 metadata file(s) written
[21:16:39] ✓ store — Report stored successfully
[21:16:39] ✓ cleanup — Cleanup done — analysis complete
```

Chaque étape est visible en temps réel dans l'interface, avec le détail de ce qui se passe. Plus besoin de deviner si le transfert est en cours ou si le parsing bloque. À noter : la récupération passe par le protocole MySQL (`--read-from-remote-server`), pas par SSH — aucune clé à déployer sur le master.

Le bon binaire pour la bonne version

Un des pièges classiques : utiliser un `mysqlbinlog` MariaDB pour lire un binlog MySQL 8, ou vice-versa. Les événements sont marqués "Ignorable" et les données row-based deviennent illisibles.

PmaControl embarque **8 binaires mysqlbinlog** statiques (x86_64 et aarch64) :

Binaire	Versions couvertes
`mysqlbinlog-5.6`	MySQL 5.5, 5.6, Percona 5.6
`mysqlbinlog-5.7`	MySQL 5.7, Percona 5.7
`mysqlbinlog-8.0`	MySQL 8.0
`mysqlbinlog-8.4`	MySQL 8.4 LTS
`mysqlbinlog-9.7`	MySQL 9.7 LTS
`mysqlbinlog-mariadb`	MariaDB 10.x, 11.x, 12.x (fallback générique)

Le choix est automatique : PmaControl lit la variable `version` du master dans ses métriques et sélectionne le binaire compatible le plus proche (cf. `MysqlbinlogBinaryResolver`). Pour MySQL Oracle, la stratégie consiste à prendre la **plus petite version ≥ celle du master** (la compatibilité ascendante est plus fiable que la rétro-compatibilité). Pour MariaDB, le resolver supporte aussi des binaires plus précis nommés `mysqlbinlog-mariadb-10.11`, `mysqlbinlog-mariadb-11.8`, etc. — déployés à la demande quand un host expose un format de binlog spécifique à sa minor.

Le rapport d'analyse

Graphe volume seconde par seconde

Le graphe principal affiche deux séries superposées :

- **Barres bleues** : volume en KB/s (somme des `transaction_length` par seconde)
- **Ligne orange** : nombre de transactions par seconde

C'est le premier réflexe du DBA : est-ce que le lag vient d'un pic ponctuel ou d'un débit soutenu ? Le graphe répond immédiatement.

Métriques de parallélisme MTS

C'est là que l'analyse devient vraiment utile. PmaControl parse les champs `last_committed` et `sequence_number` de chaque GTID event pour calculer :

- **% de transactions séquentielles** — celles où `sequence_number = last_committed + 1`, qui ne peuvent pas être parallélisées par le Multi-Threaded Slave
- **Taille maximale des groupes de parallélisme** — combien de transactions peuvent réellement s'exécuter en parallèle
- **Distribution des groupes** — combien de groupes de 1, 2, 3... transactions

Exemple concret : si 31% des transactions sont séquentielles et le max de parallélisme est de 7, alors même avec 16 `replica_parallel_workers`, la plupart resteront idle. La recommandation est claire : activer `binlog_transaction_dependency_tracking = WRITESET` sur le master.

Détection des grosses transactions

Les transactions volumineuses sont le tueur de réplication n°1. Une seule transaction de 834 KB qui touche 6171 rows bloque le SQL applier thread pendant toute sa durée — les autres transactions attendent derrière.

PmaControl compte :

- Les transactions > 100 KB
- Les transactions > 500 KB
- La plus grosse transaction (avec son contenu : quelles tables, combien de rows)

Top tables

Le rapport liste les 30 tables les plus modifiées avec le détail INSERT/UPDATE/DELETE. C'est souvent là qu'on trouve le pattern toxique : des batches qui font `DELETE FROM table WHERE ...;` `INSERT INTO table ...` pour "rafraîchir" des données au lieu d'utiliser `INSERT ... ON DUPLICATE KEY UPDATE` ou des transactions plus petites.

Recommandations automatiques

En fonction des métriques, PmaControl génère des recommandations concrètes :

- "Sequential transaction ratio is 31.3%. Consider `binlog_transaction_dependency_tracking = WRITESET`."
- "3 transaction(s) > 500 KB. Large transactions block the SQL applier thread."

- "148 databases modified. Multi-tenant pattern may cause replication hot-spots."

Ce ne sont pas des conseils génériques — ils sont calculés à partir des données réelles de vos binlogs.

Notification Telegram

Quand l'analyse est terminée, un résumé est envoyé automatiquement via Telegram :

```
Binlog Analysis Complete
Slave: replica-prod-01
Master: master-prod-01 (8.0.44)
Range: 2026-04-13 20:27:51 → 2026-04-13 20:29:27
━━━━━━━━━━━━━━━━━━━
Size: 101.0 MB | Txn: 36,284 | Duration: 96s
DML: I:148,659 U:191,867 D:115,853 = 456,379 rows
Peak: 612 txn/s | Avg: 378.0 txn/s
Sequential: 31.3% | Max parallel: 7
Large txn: 123 >100K, 3 >500K (max 815 KB)
Databases: 148
━━━━━━━━━━━━━━━━━━━
• High write throughput (peak 612 txn/s). Ensure replica_parallel_workers >= 8.
• Sequential transaction ratio is 31.3%. Consider WRITESET.
• 3 transaction(s) > 500 KB. Large transactions block the SQL applier thread.
```

Le DBA d'astreinte a tout ce qu'il faut pour agir, directement dans sa poche.

Architecture technique

Récupération des binlogs : comme un IO thread

PmaControl ne passe pas par SSH pour récupérer les binlogs. Il utilise `mysqlbinlog --read-from-remote-server` qui se connecte au master via le protocole MySQL, exactement comme le IO thread d'un replica. Les credentials MySQL de PmaControl suffisent — pas besoin de clé SSH sur le master.

```
mysqlbinlog --read-from-remote-server \
  --host=10.105.1.11 --port=3306 \
  --user=pmacontrol --password=*** \
```

```
    --raw --result-file=/tmp/analysis/ \
    mysql-bin.1054495
```

Cela signifie que si PmaControl peut se connecter au MySQL du master (ce qu'il fait déjà pour le monitoring), il peut récupérer les binlogs. Aucune configuration supplémentaire.

Découverte intelligente des fichiers binlog

Un master de production peut avoir **des milliers** de fichiers binlog (on a rencontré 2712 fichiers lors du développement). Scanner chaque fichier pour trouver la plage horaire serait prohibitif.

La stratégie en 2 étapes :

1. **Ancrage via la position du slave** — on lit `Master_Log_File` du `SHOW SLAVE STATUS` pour connaître le binlog courant. On lit aussi `SHOW MASTER STATUS` sur le master pour tenir compte du lag. Cela donne une fenêtre étroite au lieu de scanner les 2712 fichiers.
2. **Recherche binaire** — dans cette fenêtre, on probe le premier timestamp de chaque fichier via `mysqlbinlog --stop-position=8192` (ne lit que l'en-tête FORMAT_DESCRIPTION + premier event). Avec une recherche binaire, trouver le bon fichier parmi 100 candidats ne prend que 7 probes au lieu de 100.

Résultat : la découverte de binlogs sur un master avec 2712 fichiers prend **2 secondes** au lieu de plusieurs minutes.

Cache persistant + sidecar JSON pour IA

Les binlogs téléchargés sont stockés dans `data/binlog_analysis/<id_master>/` avec une **TTL de 30 jours**. Conséquence directe : ré-analyser la même plage horaire (ou une plage adjacente qui retape les mêmes fichiers) est gratuit côté réseau et durée — on retombe immédiatement sur `Cache hit: mysql-bin.046508`.

Pour chaque binlog mis en cache, PmaControl écrit aussi un sidecar `mysql-bin.046508.meta.json` contenant un résumé structuré : transactions totales, parallélisme MTS, top tables DML, DDL, pic txn/s. Ce JSON est conçu pour être consommé par un agent LLM (cf. la page Agents IA du site) qui doit raisonner sur un incident sans avoir besoin de re-parser les binlogs lui-même.

Les 6 binaires statiques, et pourquoi

PmaControl embarque des binaires `mysqlbinlog` pré-compilés pour chaque version majeure. Ce choix découle d'un constat technique : **un `mysqlbinlog` incompatible ne produit pas une erreur, il produit des résultats silencieusement faux**.

Exemple vécu : un `mysqlbinlog` MariaDB qui lit un binlog MySQL 8.0 marque les événements row-based comme "Ignorable" et les saute. Le décompte des transactions passe de 36284 à 0. Aucune erreur n'est affichée.

Les binaires sont extraits des tarballs officiels :

- MySQL 5.6 et 5.7 : depuis `dev.mysql.com/get/Downloads/MySQL-5.x/`
- MySQL 8.0 et 8.4 : depuis les minimaux glibc2.17
- MySQL 9.2 : release Innovation
- MariaDB : le `mariadb-binlog` de la distribution système

Bug mysqlbinlog 5.6/5.7 : `--raw` ignore `--stop-datetime`

Un piège découvert pendant le développement : en mode `--raw --read-from-remote-server`, les versions 5.6 et 5.7 de `mysqlbinlog` **ignorent silencieusement** les options `--start-datetime` et `--stop-datetime`. Au lieu de s'arrêter à la date demandée, le processus se comporte comme un IO thread et **attend indéfiniment** les nouveaux événements.

La solution dans PmaControl : utiliser `--raw` sans filtre temporel (télécharge le fichier complet, puis s'arrête), et appliquer le filtrage par plage horaire uniquement lors du parsing local. Un timeout de 120 secondes par fichier est ajouté en sécurité.

Pipeline asynchrone avec progress en temps réel

L'analyse tourne en arrière-plan via le CLI Glial (`php App/Webroot/index.php slave runBinlogAnalysisCli <id>`). Le frontend poll le statut toutes les 2 secondes et affiche la progression dans un terminal stylisé. Le champ `progress` en JSON stocke chaque étape avec timestamp, phase, message et statut — 15 phases de l'initialisation au cleanup.

Stockage des résultats

Tout est persisté dans la table `binlog_analysis` :

- Volume par seconde (JSON)
- Top tables (JSON)

- Distribution du parallélisme (JSON)
- Recommandations (JSON)

Les résultats sont consultables à tout moment depuis l'historique des analyses.

Cas d'usage concrets

"Le replica lag à 3h du matin"

Le monitoring montre un pic de lag à 3h. Le lendemain matin, le DBA sélectionne la plage 02:55-03:10 sur le graphe et lance l'analyse. Résultat : un batch de refresh `port_flux` qui DELETE + INSERT 15000 rows en une seule transaction sur 148 bases. Solution : découper en transactions de 500 rows.

"Pourquoi le replica ne rattrape pas ?"

Le lag augmente progressivement malgré 8 parallel workers. L'analyse montre 35% de transactions séquentielles et un max de parallélisme de 5. Solution : activer WRITESET sur le master et passer à 16 workers.

"Quel est l'impact de ce batch ?"

Avant de lancer un batch de migration en production, on peut analyser un binlog similaire d'un environnement de staging pour estimer l'impact sur la réplication.

Comment l'utiliser

1. Ouvrez PmaControl → Slave → Show sur votre replica
2. Cliquez-glissez sur le graphe de lag pour sélectionner une plage
3. Cliquez "Analyze Binlogs"
4. Regardez les étapes défiler en temps réel
5. Consultez le rapport, vérifiez les recommandations
6. Recevez le résumé sur Telegram

C'est tout. Pas de SSH manuel, pas de `mysqlbinlog | grep | awk | sort`, pas de script à maintenir. Le diagnostic de lag de réplication en un clic.