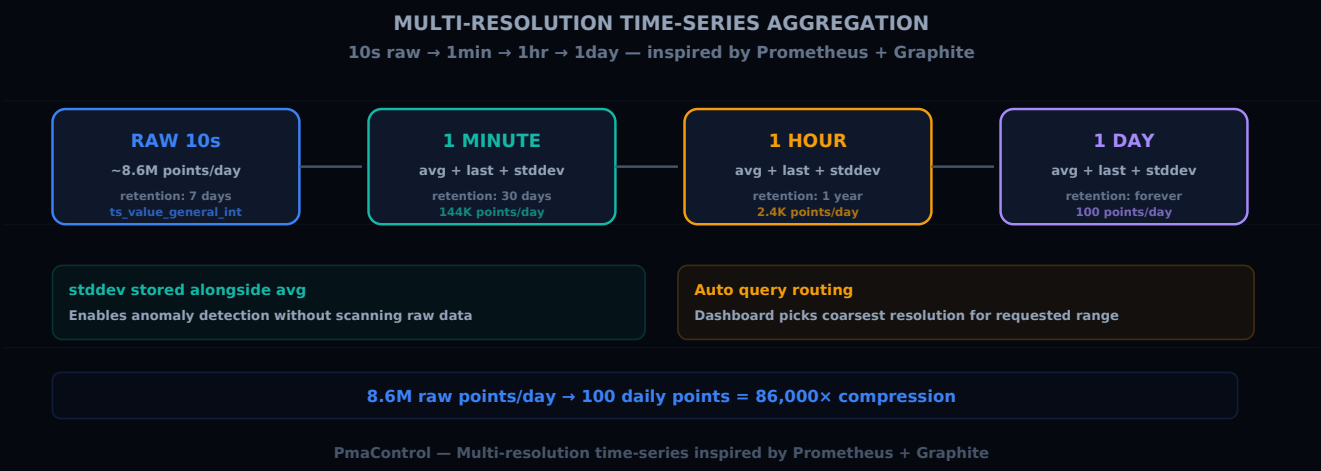
Agrégation Time-Series : De Millions de Points Bruts à des Requêtes Rapides

Aurélien LEQUOY · 21 mars 2026

PMACONTROL TIME-SERIES AGGREGATION MONITORING ARCHITECTURE



Le volume brut

PmaControl collecte des métriques depuis chaque instance MariaDB / MySQL supervisée toutes les **10 secondes**. Pour chaque serveur, cela représente :

- 6 points par minute
- 360 points par heure
- 8 640 points par jour
- **60 480 points par semaine**

Avec 100 serveurs et 50 métriques par serveur :

```
100 serveurs × 50 métriques × 8 640 points/jour = 43 200 000 points/jour
```

43 millions de points par jour. 302 millions par semaine. Plus d'un milliard par mois.

Stocker tout cela en résolution brute (10 secondes) indéfiniment est techniquement possible mais pratiquement inutile. Personne ne regarde un graphique avec une résolution de 10 secondes sur les données d'il y a 6 mois. Et les requêtes qui balaient des millions de lignes pour afficher un

graphique d'un an sont lentes et coûteuses.

L'inspiration : Prometheus et Graphite

Le problème n'est pas nouveau. Deux systèmes l'ont résolu de manière élégante :

- **Prometheus** avec ses **recording rules** : des requêtes PromQL pré-calculées qui agrègent les données brutes en métriques dérivées à intervalles réguliers
- **Graphite** avec son format **Whisper** : un système de rétention multi-résolution où les données sont automatiquement agrégées quand elles vieillissent

PmaControl s'inspire des deux approches pour concevoir son propre système d'agrégation.

Le schéma multi-résolution

Quatre niveaux de résolution :

Niveau	Intervalle	Rétention	Volume estimé (100 srv)
Raw	10 secondes	7 jours	302M points/semaine
1 minute	1 minute	30 jours	216M points/mois
1 heure	1 heure	1 an	43.8M points/an
1 jour	1 jour	Indéfini	1.8M points/an

Le volume total stocké à tout instant (avec 100 serveurs) :

```
Raw (7 jours) :    302M points
1min (30 jours) :  216M points
1hr (1 an) :        43.8M points
1day (tout) :        ~2M points
Total :            ~564M points
```

Sans agrégation, garder un an de données brutes représenterait **15.8 milliards de points**. L'agrégation réduit le stockage d'un facteur 28.

Ce qu'on stocke à chaque niveau agrégé

Pour chaque point agrégé, trois valeurs sont stockées :

```
CREATE TABLE ts_aggregated_1min (
    server_id     INT,
    metric_id     INT,
    timestamp     DATETIME,
    last_value    DOUBLE,    -- dernière valeur de l'intervalle
    avg_value     DOUBLE,    -- moyenne de l'intervalle
    stddev_value  DOUBLE,    -- écart-type de l'intervalle
    PRIMARY KEY (server_id, metric_id, timestamp)
);
```

Pourquoi `last_value` ?

Pour les métriques de type compteur (nombre de requêtes, bytes envoyés), la dernière valeur de l'intervalle est souvent plus pertinente que la moyenne. Elle représente l'état le plus récent.

Pourquoi `avg_value` ?

Pour les métriques de type gauge (utilisation CPU, mémoire, threads actifs), la moyenne est la représentation la plus fidèle du comportement sur l'intervalle.

Pourquoi `stddev_value` ? L'insight clé

C'est l'innovation principale de cette conception. **Stocker l'écart-type aux côtés de la moyenne permet la détection d'anomalies sans les données brutes.**

Considérons deux heures avec la même moyenne CPU de 45% :

- **Heure A** : CPU stable entre 42% et 48%. `avg=45%, stddev=2%`
- **Heure B** : CPU oscillant entre 5% et 85%. `avg=45%, stddev=28%`

Sans le stddev, ces deux heures sont indistinguables dans les données agrégées. Avec le stddev, l'heure B est immédiatement identifiable comme anormale.

Cela permet de construire des alertes basées sur le stddev historique :

```
SI stddev_actuel > 3 × stddev_moyen_30_derniers_jours
ALORS alerte : comportement anormal détecté
```

Le processus d'agrégation

L'agrégation fonctionne en cascade, pilotée par un cron job :

Étape 1 : Raw → 1 minute

Toutes les minutes, un worker lit les 6 derniers points raw pour chaque paire (serveur, métrique) et calcule :

```
INSERT INTO ts_aggregated_1min (server_id, metric_id, timestamp, last_value, avg_value,
stddev_value)
SELECT
    server_id,
    metric_id,
    DATE_FORMAT(timestamp, '%Y-%m-%d %H:%i:00') AS minute,
    -- last_value : sous-requête pour le dernier point
    (SELECT value FROM ts_raw r2
     WHERE r2.server_id = ts_raw.server_id
       AND r2.metric_id = ts_raw.metric_id
       AND r2.timestamp >= DATE_FORMAT(ts_raw.timestamp, '%Y-%m-%d %H:%i:00')
       AND r2.timestamp < DATE_FORMAT(ts_raw.timestamp, '%Y-%m-%d %H:%i:00') + INTERVAL 1
MINUTE
     ORDER BY r2.timestamp DESC LIMIT 1),
    AVG(value),
    STDDEV(value)
FROM ts_raw
WHERE timestamp >= NOW() - INTERVAL 1 MINUTE
GROUP BY server_id, metric_id, minute;
```

Étape 2 : 1 minute → 1 heure

Toutes les heures, un worker agrège les 60 points 1-minute en un point 1-heure. Le calcul du stddev combiné utilise la formule de variance poolée :

```
σ_combiné = sqrt( mean(σ²_i) + var(μ_i) )
```

Où `σ_i` sont les stddevs des sous-intervalles et `μ_i` leurs moyennes. Cette formule est mathématiquement exacte et n'a pas besoin des données brutes.

Étape 3 : 1 heure → 1 jour

Même principe, une fois par jour, 24 points 1-heure deviennent un point 1-jour.

Étape 4 : Purge des anciennes données

Après chaque agrégation, les données au-delà de la rétention sont supprimées :

```
DELETE FROM ts_raw WHERE timestamp < NOW() - INTERVAL 7 DAY;
DELETE FROM ts_aggregated_1min WHERE timestamp < NOW() - INTERVAL 30 DAY;
DELETE FROM ts_aggregated_1hr WHERE timestamp < NOW() - INTERVAL 1 YEAR;
-- ts_aggregated_1day : jamais purgé
```

Le routage de requêtes

Quand le dashboard PmaControl affiche un graphique, il doit choisir la bonne résolution. Le principe est simple : **utiliser la résolution la plus grossière qui couvre la plage demandée**.

```
function selectResolution(int $timeRangeSeconds): string {
    if ($timeRangeSeconds <= 3600) {       // <= 1 heure
        return 'ts_raw';                   // 10s resolution
    } elseif ($timeRangeSeconds <= 86400 * 2) {  // <= 2 jours
        return 'ts_aggregated_1min';       // 1min resolution
    } elseif ($timeRangeSeconds <= 86400 * 90) { // <= 90 jours
        return 'ts_aggregated_1hr';        // 1hr resolution
    } else {
        return 'ts_aggregated_1day';       // 1day resolution
    }
}
```

Résultat : un graphique sur 1 an ne charge que **365 points** (résolution 1 jour) au lieu de 3.1 millions (résolution 10 secondes). La requête passe de plusieurs secondes à quelques millisecondes.

Impact sur les requêtes

Plage demandée	Résolution	Points chargés	Temps de requête
1 heure	10s (raw)	360	< 10 ms
24 heures	1 min	1 440	< 20 ms
30 jours	1 heure	720	< 15 ms

Plage demandée	Résolution	Points chargés	Temps de requête
1 an	1 jour	365	< 10 ms

Les temps de requête deviennent **indépendants de la plage temporelle**. Un graphique d'un an est aussi rapide qu'un graphique d'une heure.

Détection d'anomalies avec le stddev stocké

Grâce au stddev pré-calculé, PmaControl peut détecter des anomalies sur les données agrégées sans revenir aux données brutes :

1. **Calcul du baseline** : moyenne et stddev du stddev sur les 30 derniers jours pour chaque métrique
2. **Comparaison** : le stddev de l'heure courante est comparé au baseline
3. **Alerte** : si le stddev dépasse 3 fois le baseline, comportement anormal

Exemple concret :

- Baseline threads_running : `avg_stddev = 2.1, stddev_stddev = 0.8`
- Heure courante : `stddev = 14.3`
- Score : `(14.3 - 2.1) / 0.8 = 15.25` sigmas — **anomalie certaine**

Ce mécanisme détecte des anomalies que la simple moyenne manquerait : un serveur dont le CPU oscille violemment mais revient toujours à une moyenne correcte.

Conclusion

L'agrégation multi-résolution est la clé pour gérer les données time-series à grande échelle. Le stockage du stddev aux côtés de la moyenne est un choix de conception peu commun mais puissant : il préserve l'information sur la variabilité, permettant la détection d'anomalies même sur des données agrégées.

Avec ce système, PmaControl peut superviser 100+ serveurs MariaDB / MySQL sur un an complet tout en garantissant des requêtes dashboard en moins de 20 millisecondes.