



VictoriaMetrics 刪除重複數據



詳細說明Victoria metrics 刪除重複資料



在使用 **VictoriaMetrics** 時，有時可能需要刪除重複的資料（例如由於誤操作或錯誤配置導致重複數據的寫入）。VictoriaMetrics 是一款高效的時間序列數據庫，提供了一些工具和方法來處理重複數據。以下是刪除重複數據的詳細說明：

VictoriaMetrics 的數據存儲特性

- Write-Once 特性**：VictoriaMetrics 內部數據存儲的特性是寫入後無法直接修改或刪除具體的數據點。
- 去重功能**：VictoriaMetrics 在查詢階段對重複數據提供內建的處理能力，但它不會自動刪除重複數據。
- 刪除方式**：需要通過覆蓋策略或使用過濾條件間接實現刪除重複數據的效果。

如何刪除重複數據？

VictoriaMetrics 提供的解決方案分為以下幾種情境：

1. 使用過濾器處理重複數據

在數據查詢階段，您可以利用 VictoriaMetrics 的內建過濾器來忽略重複數據。例如，您可以通過 ``max_over_time`` 或 ``last_over_time`` 等聚合函數，選取每個時間點的唯一值。

```
max_over_time(metric_name[時間範圍])
```





這種方式適用於避免重複數據影響查詢結果，但數據仍然會保留在存儲中。

2. 清理重複數據的方式

VictoriaMetrics 不支持直接刪除特定數據點，但您可以通過以下方式間接實現：

1. **刪除整個時間序列**：使用 `/api/v1/admin/tsdb/delete_series` API 刪除包含重複數據的整個時間序列。刪除後，重新寫入正確的數據。

示例請求：

```
POST http://<VictoriaMetrics 地址>:8428/api/v1/admin/tsdb/delete_series
```

參數：

- `match[]`：用於指定要刪除的時間序列，格式類似 PromQL 的查詢。例如：

```
match[]=metric_name{label="value"}
```

注意事項：

- 這會刪除匹配條件的所有數據，請務必確認匹配條件正確無誤。
2. **刪除特定時間範圍的數據**：如果重複數據只出現在特定時間範圍內，可以使用 `/api/v1/admin/tsdb/delete_series` API，結合時間範圍標籤（如 `time>=` 和 `time<=`），精確刪除該範圍內的數據。

示例請求：

```
POST http://<VictoriaMetrics 地址>:8428/api/v1/admin/tsdb/delete_series
```

參數：



```
match[]=metric_name{label="value", __name__="time"}
```

3. 啟用 VictoriaMetrics 的去重功能

如果數據重複是由於多個寫入點造成的，您可以啟用 VictoriaMetrics 的內建去重功能來緩解問題。啟用方式如下：

1. 在 VictoriaMetrics 配置中使用 `-dedup.minScrapeInterval` 標誌設置最小抓取間隔。
例如：

```
-dedup.minScrapeInterval=1ms
```

2. 當時間範圍內有多個數據點時，VictoriaMetrics 將僅保留最新的數據點。

刪除重複數據的最佳實踐

1. 預防重複數據的產生：

- 確保數據源配置正確，避免重複寫入。
- 使用唯一標籤組合來唯一標識時間序列。

2. 查詢時進行數據清理：

- 利用 VictoriaMetrics 的查詢聚合函數過濾數據。

3. 定期刪除過期數據：

- 配置數據過期時間（`retentionPeriod`），自動清除超過保留時間的舊數據。

4. 備份與測試：



- 在進行刪除操作前，做好數據的備份，並在測試環境驗證操作效果。

如有更進一步需求，可以提供具體的場景或數據範例，幫助設計解決方案。



詳細說明:-dedup.minScrapeInterval



`-dedup.minScrapeInterval`` 是 **VictoriaMetrics** 的一個重要參數，用於控制數據的去重行為，特別是在處理來自多個重複來源的數據時。以下是對此參數的詳細說明：

1. 參數的作用

VictoriaMetrics 支持從多個數據來源或重複抓取的情境中合併數據。當數據中有多個時間戳相近的數據點時，`-dedup.minScrapeInterval`` 定義了在多短的抓取間隔內，VictoriaMetrics 將啟用去重機制，只保留最新的數據點。

這個參數的核心功能是：

- 過濾掉多餘的數據點，降低存儲需求。
- 提高查詢效率，避免數據重複導致查詢結果不準確。

2. 配置方式

`-dedup.minScrapeInterval`` 通常在啟動 **VictoriaMetrics** 時作為命令行參數配置。



```
./victoria-metrics -dedup.minScrapeInterval=1s
```

3. 參數詳解

- 值的單位：
 - 可以是毫秒 (ms)、秒 (s)、分鐘 (m) 或小時 (h)。
 - 示例：
 - ``1ms``：最小抓取間隔為 1 毫秒。
 - ``10s``：最小抓取間隔為 10 秒。
- 默認值：如果未明確設置，VictoriaMetrics 默認值為 ``0``，即不啟用去重機制。
- 作用範圍：
 - 同一時間序列（由 metric 名稱和標籤組合唯一標識）的數據。
 - 去重邏輯僅在兩個數據點的時間戳間隔小於 ``-dedup.minScrapeInterval`` 時發生。

4. 工作原理

1. 無去重時的情況：

- 如果兩個數據點具有相同的時間戳和標籤，這些數據會被認為是獨立的並存儲起來。
- 查詢時，重複數據可能導致結果不準確（例如平均值偏高）。

2. 啟用去重後的情況：

- 當數據點的時間戳差異小於 ``-dedup.minScrapeInterval`` 時，VictoriaMetrics 只保留最近寫入的數據點，忽略其他數據。



- 例如，設置 `-dedup.minScrapeInterval=1s``，則在 1 秒內的多個數據點中，僅保留最後一次寫入的數據。

5. 配置場景

場景 1：多數據來源或重複抓取

如果從同一數據源抓取多次（例如 Prometheus 實例高可用配置中），會導致數據重複寫入，這種情況下應啟用 `-dedup.minScrapeInterval``。

示例：

- **問題：** Prometheus HA 設置中，兩個 Prometheus 節點抓取相同目標並將數據發送到 VictoriaMetrics。
- **解決方案：** 設置 `-dedup.minScrapeInterval=15s``，VictoriaMetrics 將在 15 秒內自動去重，僅保留最新的數據點。

場景 2：數據源時間戳不精確

某些數據源可能生成的時間戳有微小的偏差，導致數據點看似重複。啟用去重可以解決此問題。

示例：

- **問題：** 溫度傳感器每秒發送一次數據，可能由於網絡延遲導致時間戳相近的數據多次被寫入。
- **解決方案：** 設置 `-dedup.minScrapeInterval=1s``，VictoriaMetrics 僅保留每秒的最新數據。

6. 優化建議

1. 選擇合理的間隔：



- 設置的間隔應小於數據的抓取頻率。例如，如果數據每 10 秒抓取一次，設置 ``-dedup.minScrapeInterval=15s`` 是合理的。

2. 監控效果：

- 配置完成後，觀察數據庫的存儲需求和查詢性能是否改善。

3. 避免數據丟失：

- 如果數據的每個數據點都很重要（如事件數據），請不要啟用過短的去重間隔，否則可能無法存儲關鍵數據。

7. 注意事項

- **僅影響存儲：**此參數僅影響寫入的數據，對已存儲的數據不生效。
- **數據不可恢復：**被去重過濾掉的數據無法恢復，確保數據源抓取頻率合理。
- **不影響查詢去重：**查詢中仍可使用 PromQL 聚合函數進行數據去重。

8. 小結

``-dedup.minScrapeInterval`` 是 VictoriaMetrics 提供的一種高效工具，通過定義最小抓取間隔，自動去重存儲的重複數據。正確使用此參數可以減少存儲空間浪費並提升查詢效率。建議根據具體的數據來源特性和使用需求合理配置此參數。



詳細說明1-7



以下是針對 **1-7** 的詳細說明，完整解釋每一點：



1. 值的單位

VictoriaMetrics 的 `-dedup.minScrapeInterval` 參數支持多種時間單位，可靈活調整。這些單位的格式和意義如下：

- **支持的單位：**
 - **ms**：毫秒（1 秒的千分之一）。
 - **s**：秒。
 - **m**：分鐘。
 - **h**：小時。
- **示例：**
 - `10ms`：最小抓取間隔為 10 毫秒，適合非常高頻數據的場景。
 - `1s`：最小抓取間隔為 1 秒，適用於常見的秒級抓取配置。
 - `5m`：最小抓取間隔為 5 分鐘，用於低頻數據（如設備狀態檢查）。
 - `1h`：最小抓取間隔為 1 小時，適合非常低頻或批次數據。

設定適當的單位對於優化存儲空間和查詢性能非常重要。

2. 默認值

- 當 **未設置** `-dedup.minScrapeInterval` 時，VictoriaMetrics 使用默認值 `0`。
 - **效果**：禁用去重機制，所有寫入的數據都將存儲。
 - **風險**：
 - 如果數據源重複抓取，將導致存儲大量重複數據。
 - 重複數據會增加存儲需求，並可能在查詢結果中引入誤差。
- **建議**：



- 對於有高可用（HA）配置或高頻數據抓取的場景，應設置非零值以啟用去重功能。

3. 作用範圍

- **範圍的定義：** ``-dedup.minScrapeInterval`` 影響的是同一時間序列的數據。
 - **時間序列：**由 **metric 名稱** 和 **標籤組合** 唯一標識。
 - 例如：``cpu_usage{host="server1", region="us-east"}``。
 - 不同標籤組合的數據點會被認為是不同的時間序列，去重邏輯不會影響它們。
- **去重邏輯：**
 - 如果兩個數據點的時間戳差異小於 ``-dedup.minScrapeInterval``，VictoriaMetrics 會保留最新的數據點，丟棄較舊的。

4. 工作原理

- **正常數據存儲流程：**
 1. 數據從數據源（如 Prometheus、Telegraf）寫入 VictoriaMetrics。
 2. 每條數據包括：時間戳、metric 名稱、標籤和數據值。
- **啟用去重後的存儲行為：**
 1. VictoriaMetrics 會檢查數據點的時間戳。
 2. 如果時間戳之間的間隔小於 ``-dedup.minScrapeInterval``，舊的數據點會被丟棄。
 3. 最終，僅保留最新的數據點。

示例：

- 設置 ``-dedup.minScrapeInterval=10s``。



- 數據源在 10 秒內重複寫入以下數據：

```
cpu_usage{host="server1"} 0.75 @ t1
cpu_usage{host="server1"} 0.80 @ t1+5s
cpu_usage{host="server1"} 0.85 @ t1+10s
```

- 去重後結果：
 - 保留：`0.85 @ t1+10s`。
 - 丟棄：`0.75 @ t1` 和 `0.80 @ t1+5s`。

5. 配置場景

- 典型應用場景：

1. 高可用（HA）環境：

- Prometheus 集群配置多個節點抓取相同的目標，數據重複寫入 VictoriaMetrics。
- 配置 `-dedup.minScrapeInterval` 可有效去除重複數據。

2. 多數據來源：

- 不同系統將相同的數據推送到 VictoriaMetrics，造成重複寫入。
- 啟用去重可節省存儲空間，減少無意義的數據。

3. 時間戳誤差的數據源：

- 某些數據源時間戳不精確，可能出現多個相近時間的數據。
- `-dedup.minScrapeInterval` 可幫助解決這種問題。

6. 優化建議



- **設置合理值：**
 - 選擇的間隔應與數據的抓取頻率匹配。
 - 過短的間隔可能丟失有效數據；過長的間隔可能無法有效去重。
 - **數據預測：**
 - 根據歷史數據的重複情況來調整去重參數。
 - 如果數據重複率較高，可適當增加間隔。
 - **查詢驗證：**
 - 去重後檢查數據查詢結果，確保數據完整性。
-

7. 注意事項

1. **不可恢復的數據丟失：**
 - 被過濾掉的數據無法恢復，因此在設置該參數前需確認數據抓取頻率和重要性。
 2. **只影響存儲，查詢不受影響：**
 - 此參數影響數據寫入存儲的過程，查詢時仍可以基於已存儲的數據進行聚合和過濾。
 3. **監控影響：**
 - 在修改此參數後，需持續監控存儲使用情況和查詢性能，確保優化效果。
 4. **對已存儲數據無效：**
 - ``-dedup.minScrapeInterval`` 只能影響新的數據寫入，對於已存儲的數據無法去重。
-



小結

`-dedup.minScrapeInterval` 是 VictoriaMetrics 中一個強大的參數，用於控制數據存儲過程中的去重行為。合理設置此參數可有效減少存儲需求、提高查詢性能，但需根據數據抓取頻率和應用場景謹慎設置，避免重要數據被過濾掉。



機制跟thanos做詳細比較



VictoriaMetrics 和 Thanos 都是用於處理時序數據的高效解決方案，但它們在 **數據去重機制** 以及整體架構設計上存在明顯差異。以下是對兩者在去重機制和相關設計方面的詳細比較：

1. 去重機制設計對比

VictoriaMetrics

- **核心機制：**基於 `-dedup.minScrapeInterval`。
 - 當多個數據點的時間戳間隔小於該參數時，VictoriaMetrics 會保留最新的數據點，丟棄其他數據。
 - **去重範圍：**同一時間序列（由 metric 名稱和標籤唯一標識）。
- **優點：**
 - **低資源消耗：**去重在寫入階段完成，減少了存儲和查詢的壓力。
 - **簡單配置：**僅需設置一個參數即可實現自動去重。
 - **快速寫入：**在數據進入存儲之前完成去重，對存儲壓力影響較小。
- **缺點：**
 - **有限的靈活性：**僅能基於時間間隔進行去重，對於需要更加複雜規則的場景可能不夠靈活。



- **無法處理已存儲數據**：如果數據已經進入存儲，無法後續再進行去重操作。
-

Thanos

- **核心機制**：基於 **Queriers** 和 **StoreAPI** 的去重。
 - Thanos 的去重主要發生在查詢階段，而非寫入階段。
 - 當數據來自多個 Prometheus 實例時，Thanos Query 根據唯一標籤（如 `__replica__`）進行去重。
 - **去重工作流**：
 1. **多源數據合併**：
 - Thanos 聚合來自不同 Prometheus 實例的數據，將它們識別為相同的時間序列。
 2. **標籤去重**：
 - 添加 `--query.replica-label`（默認是 `__replica__`），在查詢時選擇性去重。
 3. **數據對比與合併**：
 - 基於數據的時間戳和標籤組合，保留唯一的數據點。
 - **優點**：
 - **高靈活性**：支持查詢時動態去重，適用於多數據源的高可用配置。
 - **不影響原始數據**：原始數據完整保留，適合需要審計或分析的場景。
 - **缺點**：
 - **查詢性能損耗**：去重在查詢階段完成，對查詢性能有一定影響，特別是大規模數據時。
 - **高存儲需求**：由於重複數據被保留，對存儲空間的需求較大。
-



2. 去重邏輯與場景對比

功能/特性	VictoriaMetrics	Thanos
去重階段	寫入階段（`dedup.minScrapeInterval`）。	查詢階段（通過標籤或時間戳進行去重）。
去重依據	時間間隔（小於配置值的數據點）。	標籤（如 `__replica__`）和時間戳。
影響範圍	僅影響新寫入數據，已存儲數據不受影響。	原始數據保留，查詢時可動態應用去重邏輯。
性能影響	對存儲有明顯優化，查詢性能不受影響。	查詢性能會受到影響，特別是大規模數據場景。
存儲效率	重複數據在寫入時已過濾，存儲效率高。	原始數據完整保留，存儲需求較高。
靈活性	靠配置參數控制，靈活性有限。	查詢時動態控制，適合多變的查詢需求。
應用場景	寫入頻率高且需要高效存儲的場景。	多 Prometheus 節點高可用（HA）配置的場景。

3. 架構層面的差異

層面	VictoriaMetrics	Thanos
設計理念	單節點高效性能和存儲優化。	分布式存儲與高可用設計。
高可用支持	無需額外配置，內建支持 HA 寫入（通過去重實現）。	通過 Sidecar 和 Queriers 集成多 Prometheus 實例。
存儲模型	自定義存儲引擎，針對高壓縮優化。	使用對象存儲（如 S3、GCS）進行長期存儲。
查詢模型	單節點查詢，適合小型部署。	分布式查詢，適合大規模環境。
資源需求	低資源消耗，適合資源受限環境。	高資源需求，適合資源充裕的集群環境。



4. 適用場景分析

VictoriaMetrics

- 適合需要高效存儲和快速查詢的小型或中型部署。
- 適合資源有限的場景，例如單機或嵌入式環境。
- 適合數據重複率高但不需要保留所有原始數據的情境。

Thanos

- 適合需要跨多 Prometheus 實例進行聚合和查詢的大型分佈式環境。
- 適合需要長期存儲（對象存儲）和數據審計的場景。
- 適合查詢需求複雜且頻繁改變的業務場景。

5. 小結

選擇指南

南	VictoriaMetrics	Thanos
優勢	高效存儲、低查詢延遲、簡單配置。	高可用、多源整合、靈活查詢。
缺點	不保留原始數據、缺乏分布式特性。	存儲需求高、查詢性能可能下降。
推薦使用	資源受限的小型部署、需高效存儲的場景。	分佈式查詢、大型集群、多 Prometheus 節點場景。

根據需求選擇合適的工具，VictoriaMetrics 和 Thanos 各自針對不同的應用場景提供最佳解決方案。