



vmagent 數據處理與優化



How vmagent Collects and Shi...
PDF

參考 "How vmagent Collects and Ships Metrics Fast with Aggregation, Deduplication, and More.pdf"，以繁體中文詳細說明



vmagent 如何透過聚合、去重與其他技術高效收集與傳輸指標數據

1. vmagent 簡介

VictoriaMetrics Agent (vmagent) 是一款輕量級的數據收集工具，設計用來從多個來源收集監控指標數據。收集後，vmagent 可以透過 **重標籤 (Relabeling)**、**數據篩選 (Reducing Cardinality)**、**流式聚合 (Stream Aggregation)** 和 **去重 (Deduplication)** 來處理數據，並將其傳輸到儲存系統，如 **VictoriaMetrics**，或其他支援 Prometheus Remote Write 協議的存儲服務。

2. 數據接收與處理

vmagent 透過 **API 接收數據** 或 **定期從目標抓取 (Scraping)** 數據。

2.1 API 接收數據

- 支援 HTTP API，允許使用者透過 **查詢參數** 或 **Pushgateway 格式 URL** 添加額外標籤。
- 採用 **並發限制 (Concurrency Limiter)**，以防止過多請求佔用資源：
 - 預設允許的並發寫入數量為 **CPU 核心數的 2 倍**。
 - 若請求排隊時間超過 **1 分鐘**，vmagent 會回應 **503 錯誤**。

2.2 抓取指標數據 (Scraping)





- vmagent 可從 Prometheus 兼容的端點 **定期抓取數據**。
- 預設抓取間隔 **1 分鐘**，超時 **10 秒**。
- 若回應超過 **16 MB**，vmagent 會丟棄超過該大小的數據。

數據處理模式

- **單次處理模式 (One-shot mode)**：適用於小型請求，處理整個抓取回應。
- **流式處理模式 (Stream mode)**：適用於大型請求，按 **64 KB** 塊逐步處理。

3. 數據標籤處理與降維 (Relabeling & Cardinality Reduction)

- **重標籤 (Relabeling)**：
 - vmagent 可透過 **metric_relabel_configs** 根據條件篩選、修改或刪除標籤，以降低數據量。
 - 例如，將 `{job=~"my-app-.*", env!="dev"}` 的 `foo` 標籤改為 `bar`。
- **降維 (Reducing Cardinality)**：
 - 避免高標籤基數 (High Cardinality) 問題，例如 `user_id` 或 `IP` 產生過多時序數據。
 - 設定 `-remoteWrite.maxHourlySeries` 和 `-remoteWrite.maxDailySeries` 限制小時或每天允許的唯一時序數量，超過則丟棄。

4. 去重 (Deduplication) 與流式聚合 (Stream Aggregation)

- **去重 (Deduplication)**
 - vmagent 透過 `-streamAggr.dedupInterval` 設定 **去重時間窗口**，僅保留時間窗口內最新或最大值的數據。
 - 例如，設定 `30s`，則只保留該 30 秒內最新的指標數據。



- **流式聚合 (Stream Aggregation)**

- vmagent 可按 **固定間隔** 聚合數據，以減少存儲佔用。
- 例如，設定 `5m`，則所有 `_total` 結尾的指標將被聚合為 `5 分鐘總計`。
- 透過 `-streamAggr.keepInput` 和 `-streamAggr.dropInput` 來控制原始數據是否保留或丟棄。

去重與聚合的順序

1. vmagent **先執行去重**，確保每個時間窗口內只保留最重要的數據。
2. **剩餘數據再進行流式聚合**，壓縮數據量。

5. 分片 (Sharding) 與複製 (Replication)

- **數據複製 (Replication)**

- 若配置了多個遠端存儲 (`-remoteWrite.url`)，則 vmagent 會向每個存儲系統 **發送相同的數據**。

- **數據分片 (Sharding)**

- 透過 `-remoteWrite.shardByURL`，vmagent 可根據時序數據的標籤計算哈希值，將數據 **均勻分配** 到不同存儲節點，避免單點過載。
- 也可透過 `-remoteWrite.shardByURLReplicas` **增加副本數量**，確保存儲系統容錯能力。

6. 遠端存儲優化與樣本值處理

- vmagent 支援：
 - **重新標籤遠端數據** (`-remoteWrite.urlRelabelConfig`)。
 - **遠端存儲級別的流式聚合** (`-remoteWrite.streamAggr.config`)。



- **數據標籤統一化** (``-remoteWrite.label``)。
- **樣本值四捨五入**：
 - ``-remoteWrite.significantFigures`` 控制**有效數字**（如四捨五入至 2 位數）。
 - ``-remoteWrite.roundDigits`` 控制 **小數點位數**（如保留 2 位小數）。

這些處理有助於提高 **數據壓縮率**，減少存儲佔用。

7. 數據快取與寫入機制

vmagent 透過 **內存快取 (In-Memory Queue)** 和 **磁碟快取 (File-based Queue)** 來避免數據丟失。

- **內存快取**
 - 使用 Go **FIFO** 佇列，若 5 秒內未能寫入遠端存儲，則數據轉存磁碟快取。
 - 默認最多允許 60% 記憶體儲存數據 (``-memory.allowedPercent``)。
- **磁碟快取**
 - 預設存儲路徑：``/vmagent-remotewrite-data``，可透過 ``-remoteWrite.tmpDataPath`` 調整。
 - 若存儲空間達上限，vmagent 會 **刪除最舊的數據塊** 來騰出空間 (``-remoteWrite.maxDiskUsagePerURL``)。

8. 最終數據發送 (Flush & Remote Write)

- vmagent 會根據 ``-remoteWrite.queues`` 配置的 **寫入佇列數量**，並行發送數據到遠端存儲。
- 透過 **速率限制 (Rate Limiting)** (``-remoteWrite.rateLimit``) 防止存儲過載。
- 若遠端存儲回應 **409 或 400 錯誤**（如請求衝突或格式錯誤），vmagent 會 **跳過該數據塊**。



- 其他錯誤則採用 **指數退避機制 (Exponential Backoff)** 重試。

總結

vmagent 透過 **並發控制、數據標籤處理、去重、聚合、分片、複製與快取**，確保監控數據的 **高效傳輸與存儲**。這些機制大幅降低網路流量、存儲需求，同時保持數據的完整性與可用性。

這使得 vmagent 成為 **高效能、低資源佔用** 的監控指標收集代理，適用於高並發、大規模監控環境。



vmagent如何快速收集和轉發Metrics...
PDF

參考"How vmagent Collects and Ships Metrics Fast with Aggregation, Deduplication, and More.pdf"、"vmagent 如何快速收集和轉發Metrics - charlieroro - 博客園.pdf"，以繁體中文詳細說明



vmagent 快速收集與傳輸 Metrics 的運作原理

參考資料來源：

1. 《How vmagent Collects and Ships Metrics Fast with Aggregation, Deduplication, and More》
2. 《vmagent如何快速收集和轉發Metrics - charlieroro - 博客園》

1. vmagent 簡介





VictoriaMetrics Agent (**vmagent**) 是一款輕量級的指標數據收集代理，可從不同來源擷取數據，並在發送前進行過濾、降維、聚合、去重、重新標籤 (**Relabeling**) 等處理，以減少數據量，提高傳輸效率。其主要特點包括：

- 支持 **HTTP API** 及 **Scrape** (拉取) 方式接收數據
- 數據處理 (**Relabeling**、去重、降維、聚合)
- 高效存儲數據，並發送到遠端存儲
- 支持 **Prometheus Remote Write** 及 **VictoriaMetrics Remote Write** 協議
- 內建高效快取與佇列 (**Memory Queue & Persistent Queue**)，確保高吞吐量及可靠性

2. vmagent 如何收集數據

vmagent 可以透過 **HTTP API 推送數據** 或 **Scrape 指標數據** 來獲取指標。

2.1 HTTP API 方式

- vmagent 提供 **HTTP API** 以接受 Prometheus 兼容的指標數據。
- 支援 附加標籤 (**Extra Labels**)，例如：

```
http://vmagent-host:8429/api/v1/write?  
extra_label=foo=bar&extra_label=baz=aaa
```

- 並發限制 (**Concurrency Limiter**)：
 - 預設最多允許 `2 × CPU 核心數` 的請求 (`-maxConcurrentInserts`)。
 - 若請求超過 `1 分鐘` (`-insert.maxQueueDuration`)，vmagent 會回傳 **503 錯誤**。

2.2 Scrape 指標

- 類似 Prometheus 的 **Pull** 模式，可定期抓取目標指標。
- 配置示例：



```
global:
  scrape_interval: 10s
scrape_configs:
  - job_name: "example"
    scrape_interval: 5m
    scrape_timeout: 30s
```

- 數據大小限制：
 - 預設最大抓取回應大小為 `16MB` (`-promscrape.maxScrapeSize`)。
 - 若 `scrape_interval` 小於 `scrape_timeout`，則 `scrape_timeout` 優先。

3. 數據預處理：Relabeling & 降維

3.1 Relabeling（重新標籤）

- 用於修改、篩選或刪除標籤，以降低指標數據量。
- 示例：將 `job=my-app-*` 但 `env!=dev` 的 `foo` 標籤改為 `bar`：

```
metric_relabel_configs:
  - source_labels: [foo]
    target_label: bar
    regex: "my-app-.*"
    action: replace
```

3.2 降維（Reducing Cardinality）

- 限制高標籤基數（High Cardinality），減少存儲壓力。
- 設定 `-remoteWrite.maxHourlySeries` 與 `-remoteWrite.maxDailySeries`，超過則丟棄。



4. 去重 (Deduplication) 與流聚合 (Stream Aggregation)

4.1 去重 (Deduplication)

- vmagent 會 在一定時間窗口內刪除重複數據，只保留最新或最大值。
- 配置方式：

```
-streamAggr.dedupInterval=30s
```

- 30s 內的重複數據僅保留最新的數據點。

4.2 流聚合 (Stream Aggregation)

- 聚合類似指標，以減少存儲佔用量，提高查詢效率。
- 示例：將 `_total` 結尾的指標數據每 `5 分鐘` 聚合一次：

```
- match: '{__name__=~".+_total"}'  
  interval: 5m  
  outputs: [total]
```

- 去重 + 聚合時的規則：
 - 流聚合間隔須為去重間隔的倍數（如 `dedup_interval = 30s`，則 `aggregation_interval = 1m`）。

5. 分片 (Sharding) 與複製 (Replication)

5.1 複製 (Replication)

- 若有多個遠端存儲，vmagent 會將相同數據發送至每個存儲：

```
-remoteWrite.url=http://storage1,http://storage2
```




5.2 分片 (Sharding)

- vmagent 將數據均勻分配到不同存儲，避免負載不均：

```
-remoteWrite.shardByURL=true
```

- 透過 Hash 計算分片，可指定：

```
-remoteWrite.shardByURL.labels=instance,job
```

6. vmagent 內建快取與數據佇列

6.1 內存快取 (Memory Queue)

- 快取數據，避免磁碟 I/O 開銷
- 若 5s 內無法處理，則寫入磁碟快取
- 記憶體佔用限制：
 - `-memory.allowedPercent=60%`
 - `-memory.allowedBytes=2GB`

6.2 磁碟快取 (File-based Queue)

- 數據持久化，避免數據丟失
- 預設最大佔用空間：無限制（可調整 `-remoteWrite.maxDiskUsagePerURL`）。
- 快取數據存放在 `/vmagent-remotewrite-data`。

7. 最終數據發送

7.1 傳輸與速率限制



- vmagent 從快取佇列讀取數據並傳輸至遠端存儲。
- 控制發送速率（預設無限制）：

```
-remoteWrite.rateLimit=10MB/s
```

7.2 錯誤處理

- 若發送失敗：
 - `409/400` 狀態碼：直接丟棄數據。
 - 其他錯誤：使用指數退避機制（Exponential Backoff）進行重試。

8. 總結

功能	描述
數據收集	HTTP API / Scrape
預處理	Relabeling、降維
去重	移除重複數據
流聚合	降低數據佔用
分片 & 複製	提高存儲可用性
快取機制	內存 & 磁碟快取
最終發送	限速控制 & 錯誤重試

vmagent 透過 **數據壓縮、流聚合、去重、分片與快取機制**，大幅減少網路流量與存儲開銷，是 **高效能、低資源消耗** 的指標收集代理，適用於大規模監控環境。





詳細說明:流式聚合 (Stream Aggregation)

vmagent 可按 固定間隔 聚合數據，以減少存儲佔用。
例如，設定 5m，則所有 _total 結尾的指標將被聚合為 5 分鐘總計。
透過 -streamAggr.keepInput 和 -streamAggr.dropInput 來控制原始數據是否保留或丟棄。



流式聚合 (Stream Aggregation) 詳細解析

vmagent **流式聚合 (Stream Aggregation)** 的主要目標是**減少指標數據存儲佔用**，提升查詢效率，並降低數據傳輸的頻寬需求。這項技術主要應用於 Prometheus 監控環境，以**長時間間隔聚合數據**，避免存儲高頻率的細粒度數據。

1. 為何需要流式聚合？

在大規模監控系統中，指標數據的**標籤組合 (label cardinality)** 可能會導致存儲壓力過大。例如：

- 如果有 **1000 台伺服器**，每台每 **10 秒** 上報一次數據，每次包含 **100 條指標**，那麼：

$1000 \text{ 台} * 100 \text{ 指標} * 6 \text{ (每分鐘上報 6 次)} * 60 \text{ (1 小時)} = 3,600,000 \text{ 條數據 / 小時}$

- 若採用 **1 分鐘聚合 (5m interval)**，則數據量可大幅減少：

$1000 \text{ 台} * 100 \text{ 指標} * 1 \text{ (每分鐘聚合 1 次)} * 60 \text{ (1 小時)} = 600,000 \text{ 條數據 / 小時}$

這樣存儲量減少 **6 倍**，且查詢效率更高。

2. 流式聚合的運作機制



2.1 配置方式

流式聚合允許按固定時間間隔（**aggregation interval**）對指標數據進行聚合，例如：

- 每 5 分鐘（5m）聚合 `_total` 結尾的計數器指標，合併為 `5m_total`：

```
- match: '{__name__=~".+_total"}'  
interval: 5m  
outputs: [total]
```

- 假設原始指標是 `http_requests_total`，則聚合後的指標為 `http_requests_total:5m_total`。
- 這樣可以減少高頻率指標存儲的數據量，且不影響統計。

3. 流式聚合的步驟

1. 定義匹配規則：

- 透過 `match` 選擇需要聚合的指標（例如 `_total`）。

2. 設定聚合間隔：

- 透過 `interval` 定義時間範圍，例如 5 分鐘、10 分鐘、1 小時。

3. 選擇輸出格式：

- 透過 `outputs` 指定聚合結果格式（如 `total`、`sum`、`avg`）。

4. 原始數據的保留與丟棄

在執行聚合時，可以透過以下選項控制原始數據的去留：

- `-streamAggr.keepInput=true`：保留原始數據，即聚合後仍然存儲原始數據。
- `-streamAggr.dropInput=true`：丟棄原始數據，只保留聚合後的結果。



示例

```
- match: '{__name__=~".+_total"}'  
  interval: 5m  
  outputs: [total]  
  keepInput: false
```

- 若 `keepInput=false`，則 5 分鐘內的原始 `_total` 數據將不再存儲，只保留 `5m_total` 的結果。
- 若 `keepInput=true`，則 5 分鐘內的 `_total` 數據仍然存儲，額外產生 `5m_total` 聚合數據。

5. 流式聚合與去重（Deduplication）的組合

去重（Deduplication）是指在一定時間窗口內只保留最新的數據點，而流式聚合則是在指定時間範圍內合併數據。這兩者可以結合使用，以最大限度減少數據量。

去重與流聚合的組合示例

```
- match: '{__name__=~".+_total"}'  
  interval: 1m  
  outputs: [total]  
  dedup_interval: 30s
```

- **dedup_interval=30s**：表示在 30 秒內僅保留最新的數據。
- **interval=1m**：表示每分鐘進行一次聚合，將 1 分鐘內的數據匯總。

去重與流式聚合的順序

1. 先進行去重（避免相同數據存儲多次）。
2. 再執行流聚合（合併固定時間範圍內的數據）。



“**注意：**流式聚合的 `interval` 應該是 `dedup_interval` 的整數倍，例如：

- `dedup_interval=30s, interval=1m` (正確 ✅)
- `dedup_interval=45s, interval=1m` (錯誤 ❌, 應為 `1m30s`)

6. 不同聚合類型的應用

根據不同的監控需求，流式聚合可應用於不同的統計類型：

聚合類型	描述	適用場景
<code>`total`</code>	統計總數	HTTP 請求數量
<code>`sum`</code>	取時間範圍內總和	記憶體用量總計
<code>`avg`</code>	計算平均值	CPU 使用率
<code>`max`</code>	取最大值	網路請求峰值
<code>`min`</code>	取最小值	服務延遲最低值

示例：針對不同指標類型的聚合

```
- match: '{__name__=~".+_bytes"}'
  interval: 5m
  outputs: [sum]
- match: '{__name__=~".+_latency"}'
  interval: 1m
  outputs: [avg]
```

- ``_bytes`` 結尾的指標 使用 ``sum``，適合統計網路流量。
- ``_latency`` 結尾的指標 使用 ``avg``，適合統計平均響應時間。



7. 流式聚合的影響

優點

- ✓ **減少存儲佔用**：長時間間隔聚合數據，降低存儲需求。
- ✓ **提高查詢效率**：減少細粒度數據，查詢時需處理的數據量更少。
- ✓ **降低網路傳輸成本**：減少頻繁傳輸高頻率數據的開銷。
- ✓ **適用於長期趨勢分析**：例如 ``5m_total``、``1h_avg``，適合可視化展示。

可能的缺點

- ✗ **可能丟失細節數據**：若只存儲聚合結果，將無法回溯細粒度數據。
- ✗ **不適用於短時間監測**：如需監測秒級變化，長時間聚合可能導致異常偵測不敏感。

8. 總結

流式聚合 (Stream Aggregation) 是 vmagent 內建的一種優化技術，可顯著減少數據存儲量，提升查詢性能。其核心概念為：

- **按固定間隔聚合數據**（如 ``5m``、``1h``）。
- **可選擇不同的聚合類型**（如 ``total``、``sum``、``avg``）。
- **支持保留或丟棄原始數據**（``keepInput`` vs. ``dropInput``）。
- **與去重 (Deduplication) 結合** 可進一步優化數據量。

這項技術適用於 **大規模監控場景**，特別是在 **Prometheus + VictoriaMetrics** 環境中，可幫助用戶有效管理數據存儲與查詢性能。



詳細說明:遠端存儲優化與樣本值處理、數據快取與寫入機制、



遠端存儲優化與樣本值處理、數據快取與寫入機制、最終數據發送 (Flush & Remote Write) 詳細解析

在 **vmagent** 的數據處理流程中，數據會經過 **遠端存儲優化**、**快取與寫入機制**，最終發送到遠端存儲 (Remote Storage)。這個過程確保了數據的可靠性、高效傳輸，以及最佳的存儲效能。

1. 遠端存儲優化與樣本值處理

vmagent 透過一系列的 **數據優化與標準化技術**，減少遠端存儲的負擔，並確保數據的一致性。主要優化包括：

1.1 標籤標準化 (Relabeling for Remote Storage)

- 遠端存儲的數據可能需要額外的標籤 (Labels)，以確保查詢時能夠一致。
- vmagent 支援在數據寫入遠端存儲之前執行 **標籤重寫 (Relabeling)**：

```
-remoteWrite.urlRelabelConfig=/path/to/relabel.yml
```

- 這可用於**刪除不必要的標籤**，避免存儲高標籤基數數據，減少存儲壓力。

1.2 樣本值處理 (Sample Value Rounding)

vmagent 可透過數值優化，減少數據存儲量，提升查詢效率，並降低數據傳輸大小。

- 有效數字控制 (Significant Figures)**
 - 限制數值的有效位數，例如：



```
-remoteWrite.significantFigures=3
```

- 原數據：`12345.6789`
- 轉換後：`12300`

- **小數點精度 (Round Digits)**

- 限制小數點後的精度：

```
-remoteWrite.roundDigits=2
```

- 原數據：`123.4567`
- 轉換後：`123.46`

這些設定有助於 **減少存儲數據的大小**，提高計算效率，特別適用於高頻率數據寫入場景。

2. 數據快取與寫入機制

為了避免數據丟失、提升效能，vmagent 採用了 **內存快取 (Memory Queue)** 和 **磁碟快取 (Persistent Queue)** 來處理數據。

2.1 內存快取 (Memory Queue)

vmagent 會**先將數據存儲在內存中**，以確保低延遲、高效能數據傳輸。

2.1.1 內存快取機制

- 使用 **FIFO 佇列** 存放數據，並設置超時：
 - 若數據 5 秒內未能發送，則寫入 **磁碟快取**。
- **內存限制 (Memory Limit)**：





- ``-memory.allowedPercent=60%``（預設最多佔用 60% 的記憶體）
- ``-memory.allowedBytes=2GB``（也可直接設定最大記憶體使用量）

2.1.2 內存快取的溢出策略

當內存快取超過限制時，vmagent 採取以下策略：

1. 將最舊的數據移動到磁碟快取
2. 若磁碟快取也超過限制，則丟棄最舊的數據

2.2 磁碟快取（Persistent Queue）

當遠端存儲 **無法即時接收數據** 或發生網絡故障時，vmagent **會將數據暫存到磁碟**，以確保數據不會丟失。

2.2.1 磁碟快取機制

- **快取存儲位置**
 - 預設存儲在 ``/vmagent-remotewrite-data``。
 - 可透過 ``-remoteWrite.tmpDataPath=/custom/path`` 指定存儲目錄。
- **快取文件結構**

```
/vmagent-remotewrite-data/persistent-queue/  
├─ <remote_storage_id>_<hash>/  
│   ├─ 000000000000000000    # 512MB chunk file  
│   ├─ 000000000020000000    # 512MB chunk file  
│   ├─ metainfo.json          # 儲存快取資訊  
│   └─ flock.lock             # 檔案鎖，確保多執行緒安全性
```

每個快取文件最大 512MB，當佇列達到存儲上限時，最舊的數據會被刪除。





2.2.2 磁碟快取的限制

- 可透過 `-remoteWrite.maxDiskUsagePerURL=10GB` 限制單個遠端存儲的磁碟快取上限。
- 若超過上限，vmagent 會刪除最舊的數據，以釋放空間。

3. 最終數據發送（Flush & Remote Write）

當數據準備就緒後，vmagent 會透過異步多佇列發送數據到遠端存儲。

3.1 數據發送佇列

vmagent 會根據配置的遠端存儲數量與 CPU 核心數啟動多個 worker 進行數據發送：

```
-remoteWrite.queues=4
```

- 預設值為 $2 \times \text{CPU 核心數}$ 。
- 若佇列數量過低，可能會導致數據積壓。
- 若過高，可能會導致遠端存儲負載過重。

3.2 發送速率控制（Rate Limiting）

若不進行限速，vmagent 可能會對遠端存儲造成過大負擔。可透過 `-remoteWrite.rateLimit` 設定：

```
-remoteWrite.rateLimit=10MB/s
```



- 限制數據傳輸速率，確保網路資源不被過度佔用。

3.3 錯誤處理與重試機制

vmagent 會根據遠端存儲的回應來決定如何處理錯誤：

HTTP 狀態碼	處理策略
`409` / `400`	直接丟棄數據
`500` / `503`	指數退避（Exponential Backoff）機制進行重試
`429`（Too Many Requests）	降低發送速率，進行重試

指數退避機制

如果發送失敗，vmagent 會：

1. 等待 **1 秒**後重試。
2. 每次重試等待時間加倍（最多 **1 分鐘**）。
3. 若持續失敗，將數據存入磁碟快取，等待恢復後再發送。

4. 總結

步驟	描述
標籤標準化	重新標籤數據，確保存儲一致性
樣本值處理	限制有效數字與小數點，減少存儲佔用
內存快取	確保數據高效發送，避免丟失



步驟	描述
磁碟快取	避免數據丟失，確保可靠性
多佇列發送	使用多個 worker 並行發送數據
發送速率控制	避免過載，保護遠端存儲
錯誤處理	使用指數退避機制，確保數據可靠發送

透過這些機制，vmagent **確保了數據的可靠性、傳輸效率與遠端存儲優化**，使其成為大規模監控環境的理想選擇。