PostgreSQL インサイド

論理レプリケーションにおける行フィルターと列リストのパフォーマンスへの影響
PostgreSQL 15 でコミットされた機能の紹介
技術者 Blog

Yu Shi
南京富士通南大軟件技術有限公司
開発事業本部デジタルソリューション事業部

はじめに

PostgreSQL におけるストリーミングレプリケーションと論理レプリケーションの違いの 1 つは、ストリーミングレプリケーションはインスタンス全体をレプリケートするのに対し、論理レプリケーションでは特定のテーブルと操作のみをレプリケートできることです。

PostgreSQL 14 以前では、テーブルがパブリッシュされると、そのすべての行と列がレプリケートされました。しかし、PostgreSQL 15 では、パブリケーションに対して行フィルターと列リストを指定して、指定した条件に一致する行と列のみをレプリケートできます。これらの 2 つの新機能について詳しく知りたい方は、以下のブログ記事をお読みください。

- 論理レプリケーションにおけるパブリケーションの行フィルター
- 論理レプリケーションにおける列リスト

これらの 2 つの新機能は、ユーザーの機能的なニーズを満たすだけでなく、論理レプリケーションのパフォーマンスにも影響を与えます。フィルターを使用する場合、データは送信前にフィルタリングされるため、オーバーヘッドが生じます。一方、送信されるデータが少なくなるため、データ転送に費やされる帯域幅と時間が節約されます。また、サブスクライバーがトランザクションの再実行に費やす時間が短縮されます。

それでは、いくつかのテストにより、それら全体のパフォーマンス上の影響を見てみましょう。

パフォーマンステストの概要

2 つの異なる方法によって、フィルター機能（行フィルターと列リスト）が論理レプリケーションのパフォーマンスに与える影響を確認しました。

テスト方法

以下の 2 つの方法を使用して、異なる側面からテストを行いました。

テスト方法 1） パブリケーションとサブスクリプションを作成する
論理レプリケーションでのデータ転送は、2 つのパートで構成されます。レプリケーション開始時にテーブルデータを初期同期するパートと、その後の増分を同期するパートです。行フィルターと列リストはそれら両方ともに機能するので、両方をテストしました。増分の同期は同期論理レプリケーションを用い、パブリッシャー側の SQL の実行時間を比較しました。

テスト方法 2） パブリケーションを作成し、データの受信に pg_recvlogical を使用する
pg_recvlogical の実行時間を比較しました。

最初の方法は、現実のシナリオに近いものです。2 番目の方法では、論理レプリケーションスロットで変更を受信し、指定したパブリケーションでの変更をファイルにリダイレクトします。これはデコード、フィルタリング、データ転送以外の影響を受けることはありません。

このテストでは、フィルター機能を使用するレプリケーションと、フィルター機能を使用しないレプリケーションを比較します。行フィルターについては、さまざまな量の行のフィルタリングをテストしました。

テスト手順

各テスト方法について、次の手順を実行しました。

- パブリケーションとサブスクリプションを作成する方法
    - 初期同期
        1. パブリッシャーとサブスクライバーとして 2 つのインスタンスを作成します。
        2. パブリッシャー側に 200 万行のデータを挿入し、パブリケーションを作成します。
        3. サブスクライバー側でサブスクリプションを作成します。
        4. テーブルの同期が完了してから、初期同期の時間を取得します。

        テーブル同期の開始時刻と終了時刻は、サブスクライバーのログから取得しました。テストは 10 回行い、平均時間を取りました。
    - 増分の同期
        1. パブリッシャーとサブスクライバーとして 2 つのインスタンスを作成します。
        2. パブリケーションとサブスクリプションを作成します。
        3. 同期論理レプリケーションをセットアップします。
        4. パブリッシャー側で SQL を実行し、実行時間または tps を取得します。

        2 つのシナリオをテストしました：一括挿入を行うトランザクションの実行（1 つのトランザクション内で 200 万行のデータを挿入）と、更新量の少ないトランザクションの実行（テーブル内に 200 万行のデータがあり、1 行が更新される）です。pgbench を使用して、前者は SQL 実行時間を、後者は tps を取得しました。どちらもテスト時間は 10 分でした。
- pg_recvlogical を使用する方法
    1. パブリッシャーとしてインスタンスを作成します。
    2. パブリケーションを作成します。
    3. 論理レプリケーションスロットを作成します。
    4. SQL を実行します。
    5. pg_recvlogical を使用してパブリッシャーからデータを受け取り、実行時間を取得します。

    ここでも、2 つのシナリオをテストしました：一括挿入を行うトランザクションの実行（1 つのトランザクション内で 200 万行のデータを挿入）と、更新量の少ないトランザクションの実行（1 つのトランザクション内で 1 行を更新し、合計 2 万トランザクション）です。time コマンドを使用して実行時間を記録し、10 回実行して平均を取りました。

指定した GUC パラメーター

パフォーマンステストにおいて、他の要因による干渉を防ぐために GUC パラメーターを以下のように指定しました。

```
shared_buffers = 8GB
checkpoint_timeout = 30min
max_wal_size = 20GB
min_wal_size = 10GB
```



```
autovacuum = off
```

ハードウェア環境

このテストでは 2 台のマシンを使用しました。1 台はパブリッシャー用、もう 1 台はサブスクライバー用（サブスクリプションを作成、または pg_recvlogical を使用）です。2 台のマシンはイントラネットにあり、ハードウェアのスペックは以下のとおりです。

項目	パブリッシャー	サブスクライバー
CPU(s)	16	16
モデル名	Intel(R) Xeon(R) Silver 4210 CPU @ 2.20GHz	Intel(R) Xeon(R) Silver 4210 CPU @ 2.20GHz
メモリ	50 GB	50 GB

両マシン間のネットワーク帯域幅は約 20.0 Gbps でした。

テーブルの構造

まず、以下の構造を持つテーブルを作成しました。

```
postgres=# ¥d tbl
                        Table "public.tbl"
 Column |            Type             | Collation | Nullable | Default
--------+-----------------------------+-----------+----------+---------
 id     | integer                     |           | not null |
 kind   | integer                     |           |          |
 key    | integer                     |           |          |
 time   | timestamp without time zone |           |          | now()
 item1  | integer                     |           |          |
 item2  | integer                     |           |          |
 item3  | integer                     |           |          |
 item4  | integer                     |           |          |
Indexes:
    "tbl_pkey" PRIMARY KEY, btree (id)
```

その後、行フィルターでは kind 列の値によってフィルタリングされるデータの比率を変えるようにしました。
（なお、行フィルター機能にはレプリカアイデンティティが設定されているカラムの指定が必要な場合があるため、本テストではレプリカアイデンティティ列として id および kind を設定しました。）

列リストのテストでは、id、kind、key、および time の 4 つの列のみをコピーするようにしました。

テスト結果

検証環境は上述したとおりです。では、結果を見てみましょう。

行フィルター

最初に、フィルタリングした行の比率を変えた場合と、フィルタリングしない場合で、初期同期にかかった時間を比較しました。フィルタリングされたデータの割合は、基本的に時間の割合と等しいことがわかります。

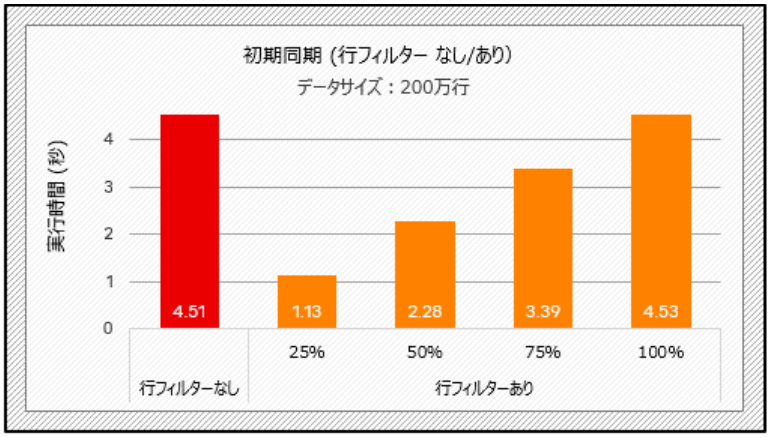

図 1：行フィルターの性能比較（テスト 1）

次の 2 つの図は、論理レプリケーションの増分同期のテスト結果です。一括挿入シナリオ（下記のテスト 2）では、フィルタリング後にデータの 25%が送信されると、フィルタリングなしの場合に比べて約 33%の時間が短縮されます。また、更新量の少ないトランザクションを実行する場合（下記のテスト 3)、データの 25%が送信されると tps が約 70%向上し、大幅な改善が見られます。

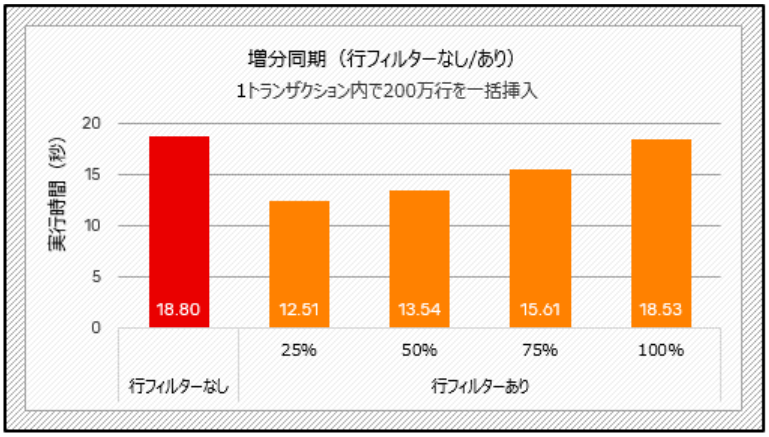

図 2：行フィルターの性能比較（テスト 2）

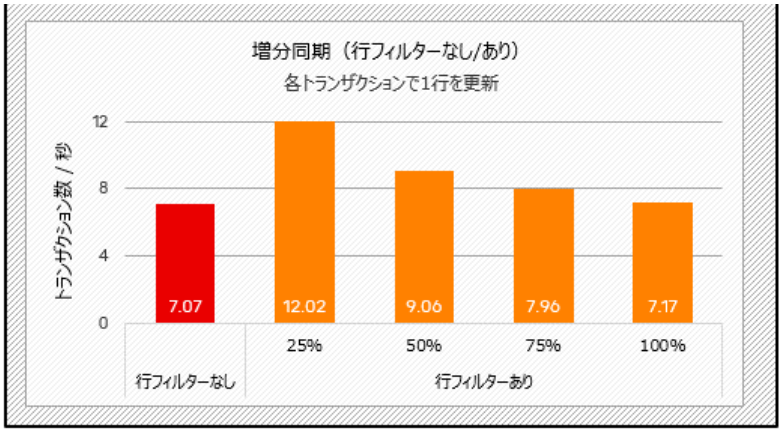

図 3：行フィルターの性能比較（テスト 3）

以下は、pg_recvlogical を使用した結果です。フィルタリング後にデータの 25%が送信される場合、一括挿入シナリオ（下記のテスト 4）では、フィルタリングなしの場合と比較して約 43%の時間が短縮され、更新量の少ないトランザクションのシナリオ（下記のテスト 5）では、約 59%の時間が短縮されます。後者のシナリオで、より顕著に改善されていることがわかります。

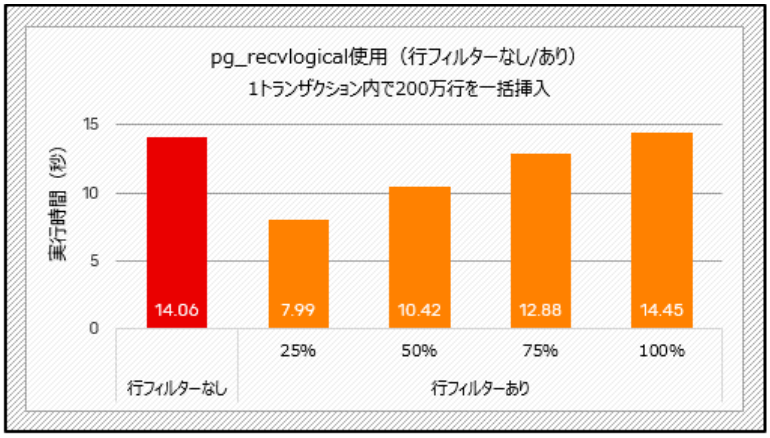

図 4：行フィルターの性能比較（テスト 4）

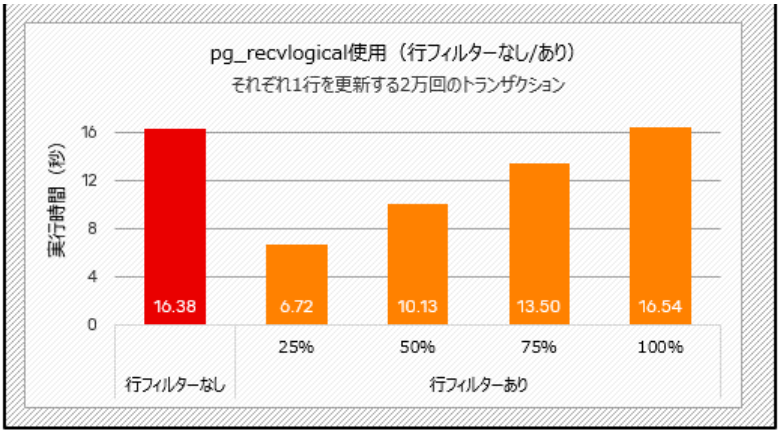

図 5：行フィルターの性能比較（テスト 5）

結論として、初期同期、一括挿入、および少数の更新はすべて、行フィルターを使用することで大幅に改善できます。フィルターによって送信されるデータが削減されない場合でも、劣化は顕著ではありません（3%未満）。

列リスト

次の図に示すように、初期同期シナリオでは、列リストを使用することでフィルターなしの場合と比較して約 11%の時間が短縮されます。

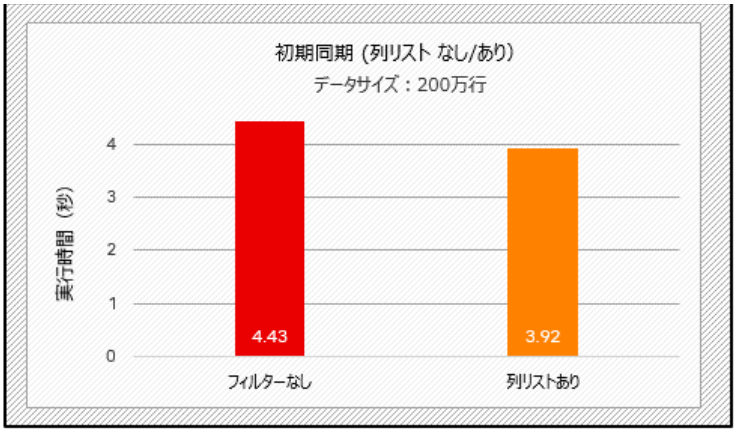

図 6：列リストの性能比較（テスト 1）

増分のレプリケーションのテストでは、一括挿入のシナリオで列リストを使用すると、約 14%の時間が短縮されます（下記のテスト 2）。更新量の少ないトランザクションを実行すると、tps が 2.5%増加しますが（下記のテスト 3）、この場合は顕著な改善とは言えません。

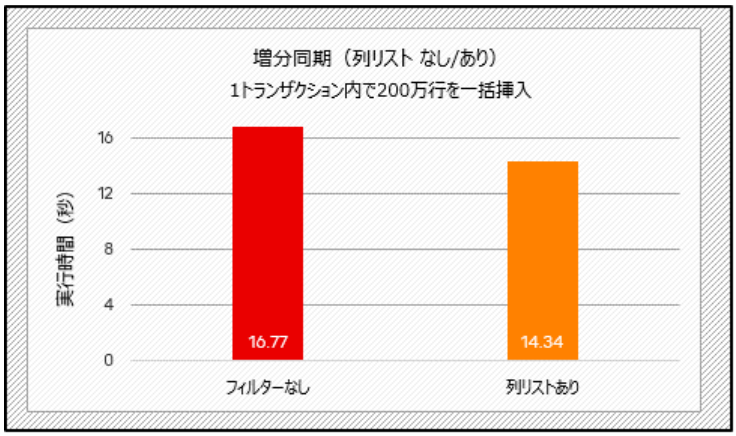

図 7：列リストの性能比較（テスト 2）

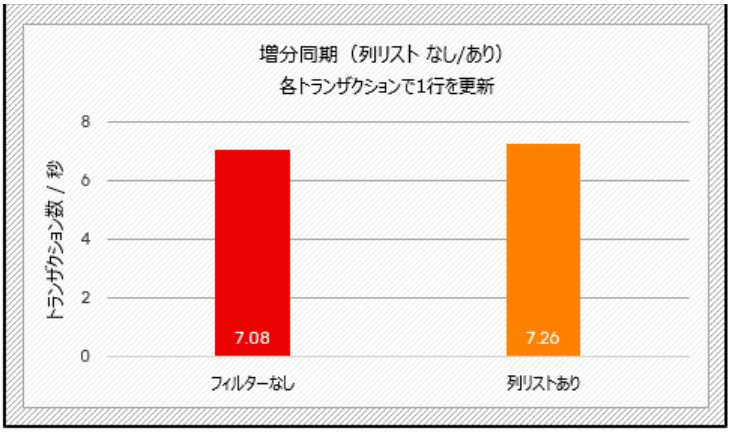

図 8：列リストの性能比較（テスト 3）

pg_recvlogical のテスト結果は増分のレプリケーションと同様で、一括挿入のシナリオ（以下のテスト 4）では約 12%、更新量の少ないシナリオ（下記のテスト 5）では 2.7%改善されました。

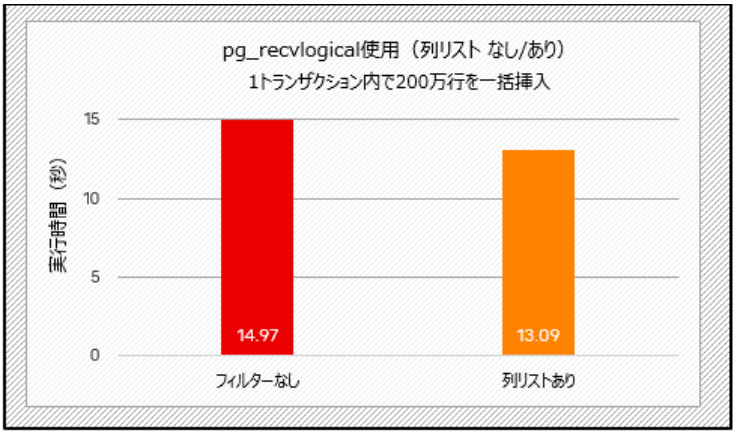

図 9：列リストの性能比較（テスト 4）

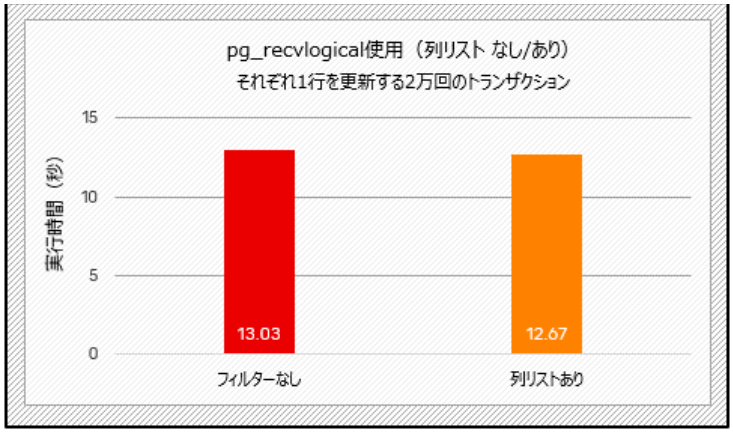

図 10：列リストの性能比較（テスト 5）

全体として、列フィルターを使用すると、初期のテーブルレプリケーションと一括挿入のパフォーマンスが向上しますが、更新量の少ないトランザクションでは、パフォーマンスの向上は明らかではありません。ただし、フィルタリングされた列に格納されるデータサイズが大きい（たとえば、写真や記事のコンテンツを格納する列）場合、改善がより明白になる可能性があります。

効果

上記のテスト結果から、論理レプリケーションの行フィルターと列リストによってパフォーマンスが向上することがわかります。これら 2 つの機能を使用することで、パブリッシャーが送信しサブスクライバーが処理するデータの量を削減でき、ネットワーク使用量と CPU のオーバーヘッドを削減できます。同時に、サブスクライバー側のディスク容量も節約することができます。もしユーザーがサブスクライバー側にテーブルの行または列のサブセットのみを必要とする場合は、フィルターの使用を検討し、パフォーマンスを向上させることができます。

今後に向けて

論理レプリケーションの行フィルターと列リストの使用によりパフォーマンスを向上させることができます。今後の PostgreSQL のバージョンアップで、論理レプリケーションに関連する、さらなる新機能とパフォーマンスの向上に期待しましょう。

詳細情報

本ブログに関連する情報の詳細については、PostgreSQL 文書をご覧ください。

- ストリーミングレプリケーションについて
    - PostgreSQL: Documentation: 15: 27.2.5. Streaming Replication（PostgreSQL オフィシャルのページへ）
      https://www.postgresql.org/docs/15/warm-standby.html#STREAMING-REPLICATION
- 論理レプリケーションについて
    - PostgreSQL: Documentation: 15: Chapter 31. Logical replication（PostgreSQL オフィシャルのページへ）
      https://www.postgresql.org/docs/15/logical-replication.html
- 同期レプリケーションについて
    - PostgreSQL: Documentation: 15: 27.2.8. Synchronous Replication（PostgreSQL オフィシャルのページへ）
      https://www.postgresql.org/docs/15/warm-standby.html#SYNCHRONOUS-REPLICATION
- pg_recvlogical について
    - PostgreSQL: Documentation: 15: pg_recvlogical（PostgreSQL オフィシャルのページへ）
      https://www.postgresql.org/docs/15/app-pgrecvlogical.html

2023 年 3 月 6 日