PostgreSQL インサイド

origin パラメーターによる双方向の論理レプリケーション PostgreSQL 16 でコミットされた機能の紹介

技術者 Blog

Vigneshwaran C
FUJITSU Limited
Software Open Innovation Business Unit Data Management Division Software Lead Developer
PostgreSQL Contributor

はじめに

このブログでは、PostgreSQL 16 の新機能である、origin パラメーターによるフィルタリングを使用した双方向の論理レプリケーションについて簡単に紹介します。

背景

通常、論理レプリケーションのパブリッシュとサブスクライブモデルを使用する場合、パブリッシュされたテーブルのすべての更新データは、データの origin（起点）に関係なく、適切なサブスクライバーにレプリケートされます。論理レプリケーションのデータをその origin でフィルタリングする新しい機能により、そのパブリッシャーで発生した更新データのみがレプリケートされるように、レプリケーションを制限できるようになりました。これは、レプリケーションのプライマリーノード間での再帰（同じデータの無限レプリケーション）を回避するために使用できます。この機能の重要な点は、WAL レコードにデータの origin が含まれていることです。これは、パブリッシャーがそれに従ってパブリッシュするのに役立ちます。

機能の概要

PostgreSQL の CREATE SUBSCRIPTION 文に新しいパラメーター「origin」 が追加されました。これは、サブスクリプションがパブリッシャーに対して、origin に関連付けられていない更新データのみを送るか、あるいは origin の関連付けに関係なく更新データを送るかを指定します。

origin に **NONE** を設定すると、サブスクリプションは origin を持たない更新データのみを送信するようにパブリッシャーに要求します。
origin に **ANY** を設定すると、パブリッシャーは origin に関係なく更新データを送信します。デフォルトは ANY です。

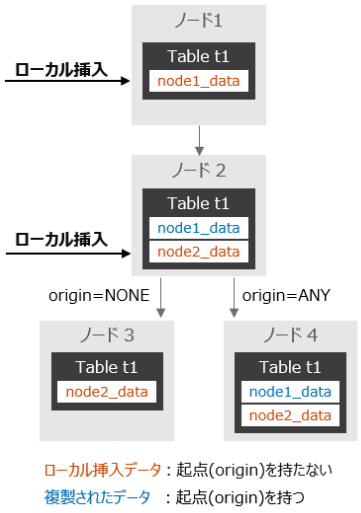

図 1：origin パラメーターによるフィルタリングの概要

注意）上記 origin=NONE の例では、ノード 2 は「node1_data」行のデータをノード 3 にレプリケートしません。このデータはノード 1 からのものであるためです。

CRATE SUBSCRIPTION 構文の改良

次に、簡略化した CREATE/ALTER SUBSCRIPTION 構文を示します。PostgreSQL 16 で追加された新しい origin パラメーターを強調表示しています。

```
CREATE SUBSCRIPTION <sub-name> CONNECTION 'conninfo' PUBLICATION <pub-list> [WITH (origin = NONE|ANY)]

ALTER SUBSCRIPTION <sub-name> SET (origin = NONE|ANY)
```

例 1

```
node1=# CREATE SUBSCRIPTION sub_host1_local_data
node1-#    CONNECTION 'host=192.168.1.50 port=5432 user=dba1 dbname=salesdb'
node1-#    PUBLICATION pub_host1 WITH (origin=NONE);
```

例 2

```
node2=# CREATE SUBSCRIPTION sub_host1_all_data
node2-#    CONNECTION 'host=192.168.1.50 port=5432 user=dba1 dbname=salesdb'
node2-#    PUBLICATION pub_host1 WITH (origin=ANY);
```

origin=NONE の場合のフィルタリング

パブリッシャー上の walsender プロセスは WAL ファイルをレコード単位で読み取り、デコードします。origin パラメーターに NONE が設定されている場合、WAL レコードのデコード中、出力プラグインは origin が関連付けられているすべてのデータを除外します。

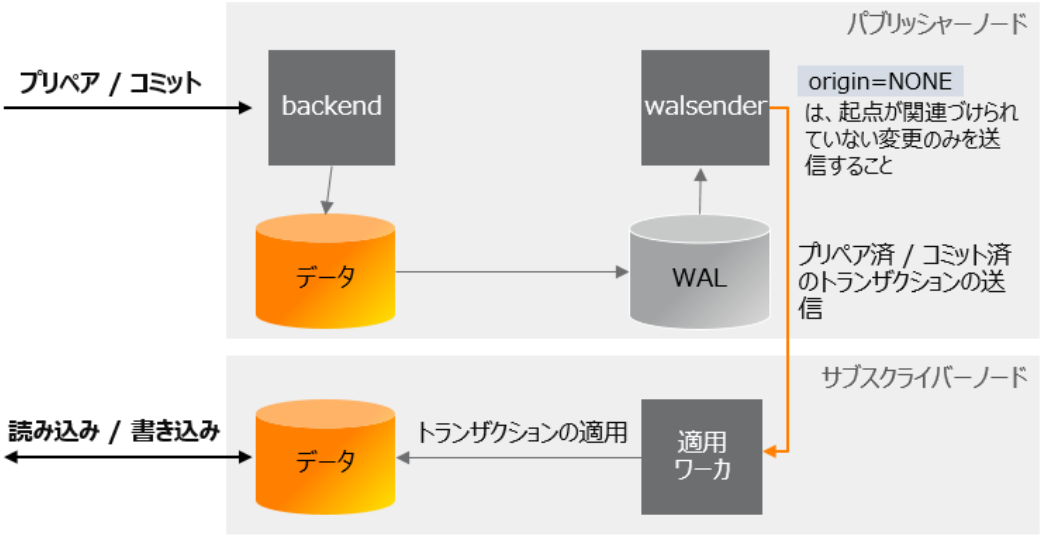

図 2：origin=NONE でフィルタリングされたトランザクション

プライマリーノード間のレプリケーション

プライマリーノード間のレプリケーションは、いずれのプライマリーノードによって実行された書込み操作もレプリケートするため、マルチマスターデータベース環境を構築するのに便利です。さまざまなシナリオでプライマリーノード間のレプリケーションを作成する手順を次に示します。

2 つのプライマリーノード間のレプリケーション設定

次の手順では、両方のプライマリーノードにテーブルデータが存在しないときに、2 つのプライマリーノード（primary1 と primary2）間でレプリケーションを設定する方法を示します。

注意）セットアップが完了するまで、ノード primary1 およびノード primary2 のテーブル t1 に対して、いずれの操作も実行されないようにしてください。

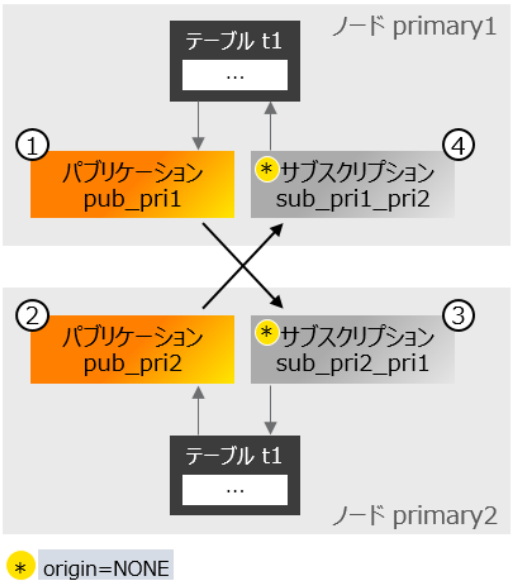

図 3：2 つのプライマリーノード間の双方向レプリケーション

1. ノード primary1 にパブリケーションを作成します。

```
primary1=# CREATE PUBLICATION pub_pri1 FOR TABLE t1;
CREATE PUBLICATION
```

2. ノード primary2 にパブリケーションを作成します。

```
primary2=# CREATE PUBLICATION pub_pri2 FOR TABLE t1;
CREATE PUBLICATION
```

3. primary2 から primary1 へのサブスクリプションを作成します。

```
primary2=# CREATE SUBSCRIPTION sub_pri2_pri1
primary2-# CONNECTION 'dbname=foo host=primary1 user=repuser'
primary2-# PUBLICATION pub_pri1 WITH (origin = NONE);
CREATE SUBSCRIPTION
```


4. primary1 から primary2 へのサブスクリプションを作成します。

```
primary1=# CREATE SUBSCRIPTION sub_pri1_pri2
primary1-# CONNECTION 'dbname=foo host=primary2 user=repuser'
primary1-# PUBLICATION pub_pri2 WITH (origin = NONE);
CREATE SUBSCRIPTION
```

この段階で、primary1 と primary2 のノード間のレプリケーションのセットアップは完了です。primary1 からの増分の更新データは primary2 にレプリケートされ、その逆も同様です。
以降の手順を実施する前に、以下の SQL 文を実行し、ここまでのセットアップが完了しているかを確認します。まず、ノード primary1 のテーブルにデータを挿入します。

```
primary1=# INSERT INTO t1 VALUES('node1_data1');
INSERT 0 1
```

次に、ノード primary2 のテーブルを読み取ります。挿入されたデータはレプリケートされます。

```
primary2=# SELECT * FROM t1;
     c1
-------------
 node1_data1
(1 row)
```

では、同じテストを primary2 から primary1 の逆方向に実行します。primary2 にデータを挿入してみましょう。テーブルには、上記の primary1 からレプリケートされたデータがすでに含まれていることに注意してください。

```
primary2=# INSERT INTO t1 VALUES('node2_data1');
INSERT 0 1
Primary2=# SELECT * FROM t1;
     c1
-------------
 node1_data1
 node2_data1
(2 rows)
```

最後に、primary1 のテーブルを確認してみましょう。primary2 からデータがレプリケートされていることがわかります。

```
primary1=# SELECT * FROM t1;
     c1
-------------
 node1_data1
 node2_data1
(2 rows)
```


3 つめのプライマリーノードの追加

次の手順では、どのプライマリーノードにもテーブル t1 にデータがないときに、既存のセットアップに新しいプライマリーノード（primary3）を追加する方法を示します。これには、ノード primary1 と primary2 にサブスクリプションを作成して primary3 からデータをレプリケートし、primary3 にサブスクリプションを作成して primary1 と primary2 からデータをレプリケートする必要があります。
これらの手順は、primary1 と primary2 間のレプリケーション設定がすでに完了していることを前提としています。

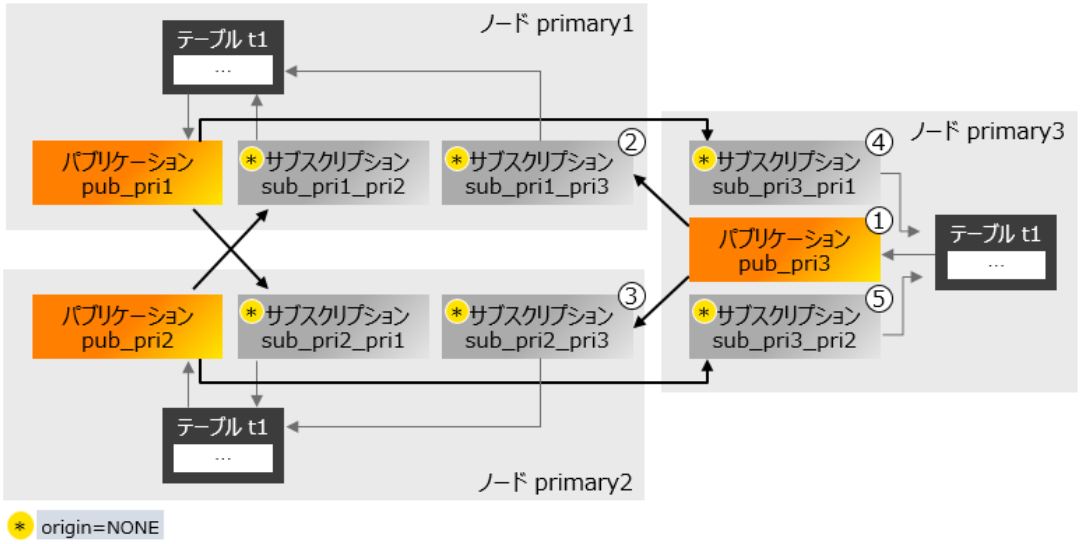

図 4：3 つのプライマリーノード間の双方向レプリケーション

1. ノード primary3 にパブリケーションを作成します。

```
primary3=# CREATE PUBLICATION pub_pri3 FOR TABLE t1;
CREATE PUBLICATION
```

2. primary1 から primary3 へのサブスクリプションを作成します。

```
primary1=# CREATE SUBSCRIPTION sub_pri1_pri3
primary1-# CONNECTION 'dbname=foo host=primary3 user=repuser'
primary1-# PUBLICATION pub_pri3 WITH (origin = NONE);
CREATE SUBSCRIPTION
```

3. primary2 から primary3 へのサブスクリプションを作成します。

```
primary2=# CREATE SUBSCRIPTION sub_pri2_pri3
primary2-# CONNECTION 'dbname=foo host=primary3 user=repuser'
primary2-# PUBLICATION pub_pri3 WITH (origin = NONE);
CREATE SUBSCRIPTION
```

4. primary3 から primary1 へのサブスクリプションを作成します。

```
primary3=# CREATE SUBSCRIPTION sub_pri3_pri1
```



```
primary3-# CONNECTION 'dbname=foo host=primary1 user=repuser'
primary3-# PUBLICATION pub_pri1 WITH (origin = NONE);
CREATE SUBSCRIPTION
```

5. primary3 から primary2 へのサブスクリプションを作成します。

```
primary3=# CREATE SUBSCRIPTION sub_pri3_pri2
primary3-# CONNECTION 'dbname=foo host=primary2 user=repuser'
primary3-# PUBLICATION pub_pri2 WITH (origin = NONE);
CREATE SUBSCRIPTION
```

これで、ノード primary1、primary2、および primary3 間のレプリケーションのセットアップが完了しました。プライマリーノードで行われた増分の更新データは、他の 2 つのノードにレプリケートされます。

制限事項

前述したレプリケーション設定の例には、いくつかの制限事項があります。

- 既存のテーブルデータを持つ新しいプライマリーノードの追加はサポートされていません。
- CREATE SUBSCRIPTION 文で copy_data=true と origin=NONE のパラメーターを組み合わせて使用する場合、初期同期テーブルデータはパブリッシャーから直接コピーされます。これは、そのデータの真の origin を知ることができないことを意味します。
  パブリッシャーがサブスクリプション機能も持つ場合、コピーされたテーブルデータはさらに上流 origin から送信された可能性があります。このシナリオでは検出されて、ユーザーに警告が記録されますが、警告は潜在的な問題を引き起こす可能性にすぎません。コピーされたデータの origin が本当に期待どおりであるかどうかを確認するのはユーザーの責任です。（パブリッシャーで作成された他のサブスクリプションが原因で）ローカル以外の origin を含む可能性のあるテーブルを特定するには、次の SQL クエリを実行してください。

```
#以下の<pub-names>を照会するパブリケーション名に置き換えてください。
SELECT DISTINCT PT.schemaname, PT.tablename
FROM pg_publication_tables PT,
     pg_subscription_rel PS
     JOIN pg_class C ON (C.oid = PS.srrelid)
     JOIN pg_namespace N ON (N.oid = C.relnamespace)
WHERE N.nspname = PT.schemaname AND
      C.relname = PT.tablename AND
      PT.pubname IN (<pub-names>);
```

注意事項

- システムの双方向レプリケーションを設定する場合、ユーザーは論理レプリケーションのコンフリクトのリスクを最小限に抑える方法でスキーマを設計する必要があります。この問題の詳細については、PostgreSQL 文書の「Conflicts」の節を参照してください。
- 現在、論理レプリケーションにはいくつかの制限、または見方によっては不足している機能があります。詳細については、PostgreSQL 文書の「Restrictions」の節を参照してください。
- プライマリーノード間のレプリケーションを設定するには、さまざまなプライマリーノードで複数の手順を実行する必要があります。すべての操作がトランザクション処理されるわけではないため、バックアップを取ることをお勧めします。PostgreSQL 文書の「Backup and Restore」の節で説明されているようにバックアップを取ることができます。

今後に向けて

「origin」パラメーターの追加は、マルチマスター論理レプリケーションをサポートするための第一歩です。この機能により、論理レプリケーションの柔軟性が向上します。富士通 OSS チームは PostgreSQL コミュニティと共に、PostgreSQL 論理レプリケーションの機能強化と追加を引き続き支援していきます。
この記事で説明したキーポイントの詳細については、以下を参照ください。

- PostgreSQL 文書の CREATE SUBSCRIPTION 文のページでは、「origin」の使用方法に関する構文と詳細な仕様が記載されています。
    - PostgreSQL:Documentaiton: 16: CREATE SUBSCRIPTION（PostgreSQL オフィィシャルのページへ）
      https://www.postgresql.org/docs/16/sql-createsubscription.html
- PostgreSQL 文書の「Replication Progress Tracking」のページでは、レプリケーションの origin について、この記事より詳細に説明しています。
    - PostgreSQL:Documentaiton: 16: Replication Progress Tracking（PostgreSQL オフィィシャルのページへ）
      https://www.postgresql.org/docs/16/replication-origins.html
- PostgreSQL の GitHub ソースコードのメインパッチは、origin を使って論理レプリケーションをフィルタリングできるようにします。
    - Allow users to skip logical replication of data having origin.（GitHub.com のページへ）
      https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=366283961ac0ed6d89014444c6090f3fd02fce0a

2023 年 11 月 8 日