PostgreSQL インサイド

論理レプリケーションにおけるコンフリクトの対処方法 (ALTER SUBSCRIPTION ... SKIP コマンド) PostgreSQL 15 でコミットされた機能の先行紹介

技術者 Blog

大墨 昂道
富士通株式会社
ソフトウェアプロダクト事業本部 データマネジメント事業部

はじめに

私の前回のブログ「論理レプリケーションにおけるコンフリクトの対処方法」では、論理レプリケーションのコンフリクトの説明と、この問題に対する対処方法として、pg_replication_origin_advance 関数を使用した手法について書きました。このブログでは、別の対処方法として、 PostgreSQL 15 でコミットされた新機能である ALTER SUBSCRIPTION ... SKIP コマンドを使用する方法について説明します。

背景

私の前回のブログでは、論理レプリケーションのコンフリクトとは何か、どのようにして起こるのか、そして PostgreSQL 15 でコミュニティーが開発した新しい機能が、解決にどのように役立つのかを説明しました。それと並行して、コンフリクトの対処について別の議論が行われました。OSS コミュニティーはプロジェクトの達成に積極的に取り組み、PostgreSQL のコードベースに ALTER SUBSCRIPTION ... SKIP コマンドを実装しました。

- ALTER SUBSCRIPTION（PostgreSQL オフィシャルのページへ）
  https://www.postgresql.org/docs/15/sql-altersubscription.html

参照先のドキュメントで説明されているように、この機能はリモートトランザクションのすべての変更適用をスキップするように動作します。これは pg_replication_origin_advance 関数の機能に似ているように見えるかもしれません。そこで、このブログでは ALTER SUBSCRIPTION ... SKIP コマンドに焦点を当てて、相違点を明らかにしたいと思います。
私の前回のブログでは、他の背景や関連する新機能について説明しました。まだ読まれていないのなら、まずそちらのブログを読むことをお勧めします。ALTER SUBSCRIPTION ... SKIP コマンドは、私の前回のブログで pg_replication_origin_advance 関数に使用したものと同様のシナリオで説明していきます。

- 免責事項　PostgreSQL 15 の公式リリース前のため、この機能はまだリリースされておらず、コミュニティーはこのブログに関連する設計を変更したり、完全に元に戻したりする可能性があることをご承知おきください。

ALTER SUBSCRIPTION ... SKIP コマンド

このコマンドの目的は、サブスクライバーにおいて、指定された LSN でエラー終了し、コンフリクトが発生しているトランザクションをスキップすることです。ユーザーは、 SKIP オプション（'lsn' skip_option）によって finish LSN を指定することで、失敗したトランザクションを指定できます。適用ワーカーはパブリッシャーから変更を受け取り、ノード上でそれをスキップするかどうかを判断します。このコマンドの入力パラメーターには、エラーコンテキストに出力された finish LSN をそのまま指定できます。そのため、pg_replication_origin_advance 関数の引数とは異なり、サーバーログでエクスポートされた LSN の編集は必要ありません。その意味で、ALTER SUBSCRIPTION ... SKIP コマンドを使用するほうがシンプルです。では以下のデモを見てみましょう。

トランザクションをスキップすることによるコンフリクトの解決（ALTER SUBSCRIPTION ... SKIP コマンドを使う場合）

サブスクライバー側をコンフリクトの状態にしてから、ALTER SUBSCRIPTION ... SKIP コマンドによって解決するシナリオを見てみましょう。

1. パブリッシャー側：1 つのテーブル（tab）とパブリケーション（mypub）を作成

```
postgres=# CREATE TABLE tab (id integer);
CREATE TABLE
postgres=# INSERT INTO tab VALUES (5);
INSERT 0 1
postgres=# CREATE PUBLICATION mypub FOR TABLE tab;
CREATE PUBLICATION
```

   最初に、初期テーブル同期を行うために 1 レコードを挿入しています。

2. サブスクライバー側：一意性制約を持つテーブル（tab）とサブスクリプション（mysub）を作成

```
postgres=# CREATE TABLE tab (id integer UNIQUE);
CREATE TABLE
postgres=# CREATE SUBSCRIPTION mysub CONNECTION '…' PUBLICATION mypub WITH (disable_on_error = true);
NOTICE:  created replication slot "mysub" on publisher
CREATE SUBSCRIPTION
```

   disable_on_error オプションが有効なサブスクリプションを作成しました。このサブスクリプション定義により、バックグラウンドで初期テーブル同期が行われ、問題なく成功します。その結果、値 5 がサブスクライバーに挿入されます。

3. パブリッシャー側：テーブルの同期後に 3 つのトランザクション（Txn1、Txn2、Txn3）を順次実行

```
postgres=# BEGIN; -- Txn1
BEGIN
postgres=*# INSERT INTO tab VALUES (1);
INSERT 0 1
postgres=*# COMMIT;
COMMIT
postgres=# BEGIN; -- Txn2
BEGIN
postgres=*# INSERT INTO tab VALUES (generate_series(2, 8));
INSERT 0 7
postgres=*# COMMIT;
COMMIT
postgres=# BEGIN; -- Txn3
BEGIN
postgres=*# INSERT INTO tab VALUES (9);
INSERT 0 1
postgres=*# COMMIT;
COMMIT
postgres=# SELECT * FROM tab;
 id
```



```
----
  5
  1
  2
  3
  4
  5
  6
  7
  8
  9
(10 rows)
```

パブリッシャー側では、これらのトランザクションを正常に実行できました。ただし、Txn2 には初期テーブル同期時にすでに挿入されていたデータと重複するデータ（5）が含まれます。したがってサブスクライバー側では、これがテーブルの一意性制約に違反してエラーになります。

4. サブスクライバー側：現在のステータスを確認

```
postgres=# SELECT subname, subenabled FROM pg_subscription;
 subname | subenabled
---------+------------
 mysub   | f
(1 row)

postgres=# SELECT * FROM tab;
 id
----
  5
  1
(2 rows)
```

では、サブスクライバーの現在のステータスを確認してみましょう。「disable_on_error」オプション の動作に従い、サブスクリプション（mysub）はエラーによって自動的に無効（f）になりました。また、Txn1（および初期テーブル同期によってレプリケートされたデータ）だけがサブスクライバーにレプリケートされています。ここでは、コンフリクトのために Txn2 と Txn3 の結果は見られません。Txn3 は、コンフリクトを解決した後で再生されます。

参照　disable_on_error オプションの詳細については CREATE SUBSCRIPTION を参照してください。

- CREATE SUBSCRIPTION（PostgreSQL オフィシャルのページへ）
  https://www.postgresql.org/docs/15/sql-createsubscription.html

5. サブスクライバー側のログ：このコンフリクトのエラーメッセージと disable_on_error オプションのログを確認

```
ERROR:  duplicate key value violates unique constraint "tab_id_key"
DETAIL:  Key (id)=(5) already exists.
CONTEXT:  processing remote data for replication origin "pg_16389" during "INSERT" for replication target relation "public.tab" in transaction 730 finished at 0/1566D10
LOG:  logical replication subscription "mysub" has been disabled due to an error
```

サブスクライバーのサーバーログで、ALTER SUBSCRIPTION ... SKIP コマンドの引数となる finish LSN（0/1566D10）を取得できます。これを利用して、以下のように Txn2 をスキップします。

6. サブスクライバー側：ALTER SUBSCRIPTION ... SKIP コマンドを実行してから、サブスクリプションを有効化

```
postgres=# ALTER SUBSCRIPTION mysub SKIP (lsn = '0/1566D10');
ALTER SUBSCRIPTION
postgres=# SELECT subname, subskiplsn, subenabled FROM pg_subscription;
 subname | subskiplsn | subenabled
---------+------------+------------
 mysub   | 0/1566D10  | f
(1 row)

postgres=# ALTER SUBSCRIPTION mysub ENABLE;
ALTER SUBSCRIPTION
postgres=# SELECT * FROM tab;
 id
----
  5
  1
  9
(3 rows)

postgres=# SELECT subname, subskiplsn, subenabled FROM pg_subscription;
 subname | subskiplsn | subenabled
---------+------------+------------
 mysub   | 0/0        | t
(1 row)
```

ここで、skip LSN（0/1566D10）を設定してから、サブスクリプションを再アクティブ化しました。すぐに Txn2 のトランザクション全体がスキップされて、Txn2 の値が無いことと、Txn3 のレプリケートされた値が確認できました。これに伴い、変更のスキップに成功した後、システムカタログ pg_subscription の subskiplsn 列に格納されていた skip LSN（0/1566D10）がクリアされました。

7. サブスクライバー側のログ：正常終了したコマンドのログメッセージを確認

```
LOG:  start skipping logical replication transaction finished at 0/1566D10
CONTEXT:  processing remote data for replication origin "pg_16389" during "BEGIN" in transaction 730 finished at 0/1566D10
LOG:  done skipping logical replication transaction finished at 0/1566D10
CONTEXT:  processing remote data for replication origin "pg_16389" during "COMMIT" in transaction 730 finished at 0/1566D10
```

サーバーログからもコマンドの正常終了を確認できます。

ALTER SUBSCRIPTION ... SKIP コマンドの高度な保護

この機能には、コンフリクト処理に固有の内部的な保護と検査の機構があります。その結果、コンフリクト時に失敗していないトランザクションをスキップするリスクがなくなります。
これを、私の前回のブログ「論理レプリケーションにおけるコンフリクトの対処方法」で説明した pg_replication_origin_advance 関数の誤った使い方の例と比較してみてください。そこで私は、この関数がコンフリクトと関係のない成功したトランザクションをスキップすることがある点に触れました。しかし、この ALTER SUBSCRIPTION ... SKIP コマンドでは、それは起きません。これは、以下の内部チェックにより実現されます。

まず、指定された LSN は、サブスクライバー上でデータが既にレプリケートされた場所を指す起点 LSN よりも、大きな値である必要があります。

さらに、この機能では、引数の finish LSN と、サブスクライバーがパブリッシャーから取得する最初のトランザクションの finish LSN が完全に一致する必要があります。そうでなければ、（サブスクリプションを有効にした後）PostgreSQL は失敗したトランザクションを再度適用しようとし、同じコンフリクトを即座に引き起こします。PostgreSQL 15 では、このような失敗は起こりません。エラーメッセージ中に finish LSN が出力される改善が行われているため、ユーザーは正確な finish LSN を取得でき、それを指定するだけでよいからです。

また、コンフリクトのない skip LSN を設定し、トランザクションの適用に成功すると、警告メッセージの出力と共に skip LSN がクリアされるだけです。実際にはスキップされません。この動作は、受信するトランザクションが問題のないものであることが判明し、ユーザーが誤って LSN を設定した可能性があるため、理にかなっています。コンフリクトが発生していない場合に、ユーザーが誤って LSN を設定しても対処すべきことはありません。新たなコンフリクトが生じた際に、この手段を採用すると決めるなどの適切なタイミングで再度 skip LSN を設定すればよいです。
これらの内部処理の詳細に興味を持たれた場合は、以下のコミットログを参照してください。

- Add ALTER SUBSCRIPTION ... SKIP.（GitHub のページへ）
  https://git.postgresql.org/gitweb/?p=postgresql.git;a=commitdiff;h=208c5d65bbd60e33e272964578cb74182ac726a8

まとめ

論理レプリケーションのコンフリクトに関して、コミュニティーはこの新しい便利な方法を実現するために多くの時間と労力を費やしてきました。よりシンプルで安全な方法を享受する選択があることは素晴らしいことです。
加えて、既にコミュニティー内において様々な角度から、この機能を強化するためのいくつかのアイデアが提案されています。例えば、リレーションやスキップするコマンド（INSERT、DELETE、・・・）を指定するなど、他のタイプの skip_option を ALTER SUBSCRIPTION ... SKIP コマンドに追加するというアイデアがあります。さらに、finish LSN のようなエラー関連情報をシステムカタログに格納したり、サブスクライバーがスキップしたデータをサーバーログや一部のテーブルに記録したりするアイデアも提案されています。
これらの機能強化はどれも魅力的であり、PostgreSQL の今後のバージョンでどのような新機能が追加されるのか楽しみです。

2022 年 7 月 22 日