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

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

この記事では、論理レプリケーションのコンフリクトとは何か、また PostgreSQL 15 がコンフリクト解決のためにどのような優れたユーティリティーを提供するかについてお話しします。

背景

論理レプリケーションは、対象を選択してデータを複製する方法です。物理レプリケーションではクラスタ全体をコピーし、必要に応じてスタンバイ側で読み取り専用のクエリを受け付けますが、論理レプリケーションでは、データレプリケーションをより細かく、柔軟に制御することができます。

- 27.4. Hot Standby（PostgreSQL オフィシャルのページへ）
  https://www.postgresql.org/docs/15/hot-standby.html

論理レプリケーションで利用できる機能には、次のようなものがあります。

- 対象テーブルの選択と操作
- サブスクライバーへの直接的書き込み
- パブリケーションとサブスクリプション間の複雑なトポロジー

論理レプリケーションでは、データはサブスクリプションのワーカープロセスによってサブクスクライバーに適用されます。サブスクリプションのワーカープロセスは、ノード上で DML 操作を実行するのと同様に動作します。そのため、新たに受信したデータがサブスクライバーのいずれかの制約に違反した場合、レプリケーションはエラーで停止します。
これは"コンフリクト"と呼ばれ、処理を進めるためにはユーザーが手動で操作する必要があります。

- 31.4. Conflicts（PostgreSQL オフィシャルのページへ）
  https://www.postgresql.org/docs/15/logical-replication-conflicts.html

PostgreSQL 15 で何が変わったのか？

PostgreSQL 15 で、PostgreSQL コミュニティーは論理レプリケーションのコンフリクトに対処するための改良と新機能を導入しています。ここでは、これらの改良点と、それらをどのように適用してコンフリクトを処理するかを説明します。この記事では、既存データとコンフリクトするトランザクションをスキップすることで、問題を解決します。この記事で紹介する自動無効化機能（disable_on_error オプション）について、私もコミュニティーで開発者の 1 人として働いていました。
次の図 1 では、コンフリクトが発生する仕組みを説明します。

> 免責事項　この記事では、実機検証に開発版の PostgreSQL を使用しましたが、PostgreSQL 15 の公式リリース前のため、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。

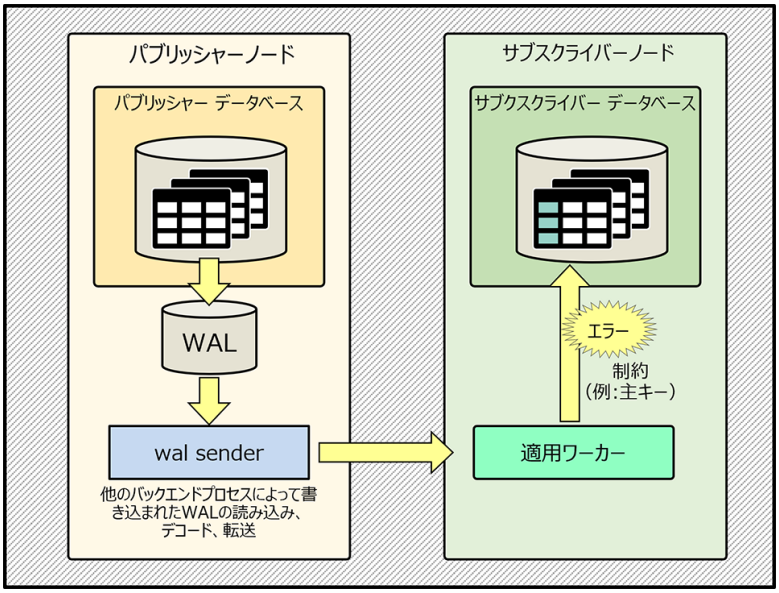

図 1：論理レプリケーション中のコンフリクト

論理レプリケーションの改良と新機能

コミュニティーは、PostgreSQL が論理レプリケーションに関して、信頼性が高く、効率的で、簡単な手段を提供できるように、懸命に努力してきました。この取り組みの一環として、以下の機能と改良がコミットされています。

サブスクリプションの統計情報のための新しいシステムビュー pg_stat_subscription_stats

このビューの各レコードは、サブスクリプションを参照します。このビューでは、2 種類のエラーカウンターが実装されています。1 つは初期テーブル同期のエラー用で、もう 1 つは変更適用のエラー用です。

- 28.2.9. pg_stat_subscription_stats（PostgreSQL オフィシャルのページへ）
  https://www.postgresql.org/docs/15/monitoring-stats.html#MONITORING-PG-STAT-SUBSCRIPTION-STATS

新しいサブスクリプションのオプション disable_on_error

何らかのコンフリクトが発生すると、論理レプリケーションワーカーはデフォルトでエラーループに陥ります。理由は、ワーカーが変更の適用に失敗したとき、エラーで終了し、再起動し、バックグラウンドで同じ変更を繰り返し適用しようとするためです。しかし、この新しいオプションを使用すると、サブスクリプションのワーカーは自動的にサブスクリプションを無効にし、エラー時のループから抜けることができます。その後、ユーザーは次に何をするかを選択できます。初期テーブル同期に失敗した場合も、サブスクリプションを無効にします。デフォルト値は false で、false であることは、コンフリクト時に同じエラーを繰り返すことを意味します。

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

サブスクリプションのワーカー・エラーについての拡張されたエラーコンテキスト情報

エラーコンテキストメッセージに、条件付きで 2 つの新しい情報が含まれるようになりました。

- **finish LSN（Log Sequence Number）**

一般的に、LSN は WAL 上の位置へのポインターです。finish LSN は、コミットされたトランザクションでは commit_lsn、プリペアドトランザクションでは prepare_lsn を意味します。

- **レプリケーション起点名**

レプリケーションの進捗状況を把握するレプリケーション起点の名前です。論理レプリケーションでは、サブスクリプションの定義とともに、対応する各レプリケーション起点が自動的に作成されます。

上記 2 つの情報は、ユーザーが pg_replication_origin_advance 関数を使用する場合、引数に使えて便利です。

- エラーコンテキストメッセージについては、PostgreSQL オフィシャルのページ「55.2. Reporting Errors Within the Server」を参照
  https://www.postgresql.org/docs/15/error-message-reporting.html
- レプリケーション起点名については、PostgreSQL オフィシャルのページ「Chapter 50. Replication Progress Tracking」を参照
  https://www.postgresql.org/docs/15/replication-origins.html
- pg_replication_origin_advance 関数については、PostgreSQL オフィシャルのページ「9.27. System Administration Functions」を参照
  https://www.postgresql.org/docs/15/functions-admin.html

失敗した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

ここまで本テーマの各拡張機能を確認しました。このセクションでは、1 つのコンフリクトシナリオをエミュレートしてみます。これを読む前に注意していただきたい点は、pg_replication_origin_advance 関数を呼び出してトランザクションをスキップすることは、ユーザーが選択できる解決策の 1 つに過ぎない、ということです。ユーザーは、サブスクライバーのデータまたは権限を変更してコンフリクトを解決することもできることを覚えておいてください。

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
```

   テーブルの初期同期を行うため、1 レコードを挿入しました。

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
```

   初期テーブル同期が行われますが、これは問題なく成功します。

3. パブリッシャー側：テーブル同期後に 3 つのトランザクションを順次実行

```
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, 4));
INSERT 0 3
postgres=*# INSERT INTO tab VALUES (5);
INSERT 0 1
postgres=*# INSERT INTO tab VALUES (generate_series(6, 8));
INSERT 0 3
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実行された Txn（トランザクション、以降 Txn）1 は、サブスクライバー側で正常に複製されます。しかし、上の青色で示した Txn2 の 2 回目の挿入では、初期テーブル同期のデータと重複するデータ（手順 1 でも青で示した箇所）が含まれています。サブスクライバー側で、これはテーブルの一意性制約に違反します。したがって、コンフリクトが発生し、サブスクリプションが無効となって、このサブスクリプションに対する複製処理が行われなくなります。次の手順でコンフリクトが解決されるまで、Txn3 は複製されません。

4. サブスクライバー側：現在の状況を確認

```
postgres=# SELECT * FROM pg_stat_subscription_stats;
 subid | subname | apply_error_count | sync_error_count | stats_reset
-------+---------+-------------------+------------------+-------------
 16389 | mysub   |                 1 |                0 |
(1 row)

postgres=# SELECT oid, subname, subenabled, subdisableonerr FROM pg_subscription;
  oid  | subname | subenabled | subdisableonerr
-------+---------+------------+-----------------
```



```
 16389 | mysub   | f          | t
(1 row)

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

トランザクションをスキップする前に、現在のサブスクライバー側の状態を確認します。
システムビューpg_stat_subscription_stats はこれまでの統計情報として、初期テーブル同期の失敗（sync_error_count）は無いが、変更適用フェーズで 1 つ失敗（apply_error_count）したことを表示しています。さらに、disable_on_error オプションに true を設定してサブスクリプションを作成したので、サブスクリプション"mysub"は失敗によりに無効（f）が設定されたことを表示しています。テーブル"tab"には、正常に複製された Txn1 の結果までのデータのみが格納されています。

5. サブスクライバー側のログ：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
```

上記では、レプリケーション起点名（pg_16389）と commit_lsn を示す LSN（0/1566D10）を確認できます。それらを活用して、次のように Txn2 をスキップしてみます。

6. サブスクライバー側：pg_replication_origin_advance 関数を実行して Txn2 をスキップし、サブスクリプション（mysub）を有効化

```
postgres=# SELECT pg_replication_origin_advance('pg_16389', '0/1566D11'::pg_lsn);
 pg_replication_origin_advance
-------------------------------

(1 row)

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

レプリケーション起点（pg_16389）と finish_LSN の次の LSN（0/1566D11）を指定して LSN を進めた後、サブスクリプションを有効にして再度活性化しました。すぐに、Txn3 の複製されたデータを確認できます。ここで、Txn2 のコンフリク

トの直接の原因とは無関係な他のいくつかのデータ（手順 3.の Txn2 で、赤色に示した別の挿入を行ったことを思い出してください）が、同じトランザクション内のタイミングにかかわらず、複製されていないことに注目してください。Txn2 トランザクション全体がスキップされました。
ここでの一連の流れは次のとおりです。

- pg_replication_origin_advance 関数を使うことで、サブスクリプションを有効にしました。
- サブスクリプションを有効にすると適用ワーカーが起動し、pg_replication_origin_advance 関数を介して渡された LSN をパブリッシャー上の wal sender プロセスに送信しました。
- この wal sender プロセスは、デコードコミット時に関連 LSN を比較して Txn2 を送信すべきかスキップすべきかを評価しました。
- wal sender プロセスは、Txn2 のトランザクションがスキップされるべきと判断しました。

最後に、適切な LSN を pg_replication_origin_advance 関数に渡すよう注意する必要があることを強調します。このブログで紹介しているコミュニティーの新しい改良により、誤用の可能性はかなり低くなっていますが、使い方を誤るとコンフリクトとは無関係の他のトランザクションを簡単にスキップできてしまいます。

間違ったパラメーターを指定するとどうなるのか？

参考までに、pg_replication_origin_advance 関数の誤使用例を示します。次の例では、上記のシナリオを再実行し、Txn3 の後に Txn4 を追加して 10 を挿入しました。そして、pg_replication_origin_advance 関数の引数に、Txn3 のコミットレコードよりも大きく、Txn4 のコミットレコード（pg_waldump コマンドで取得）よりも小さな LSN を設定しました。サブスクリプションを有効にした後、Txn3 の値が無いレプリケーションデータを取得してしまいました。

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

pg_replication_origin_advance 関数を間違って使用した場合のサブスクライバー側の結果

```
postgres=# SELECT * FROM tab;
 id
----
  5
  1
 10
(3 rows)
```

上に示したように、コンフリクトを解決するために手動でレプリケーションを操作する場合、絶対に間違えないように注意する必要があります。
この点において、コミュニティーはすでに別の機能(ALTER SUBSCRIPTION ... SKIP コマンド)を導入しています。この機能は、論理レプリケーションのコンフリクトの処理において、pg_replication_origin_advance 関数より 1 歩進んでいます。詳しくは私の次のブログ記事「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）」を参照ください。

まとめ

企業における論理レプリケーションの導入が進むにつれて、論理レプリケーションのコンフリクトなどの実務的な問題への対処がますます重要になります。このため、PostgreSQL に追加された改良は必要不可欠です。
PostgreSQL コミュニティーでは、データベースの強化を進めており、この記事では、論理レプリケーションのコンフリクトを簡単に処理する方法について説明しました。ただし、使用するツール（この場合は pg_replication_origin_advance 関数）に正しい情報を提供するように注意して利用していく必要があります。

もっと詳しく知りたい方は

PostgreSQL の論理レプリケーションとその仕組みについてもっと知りたい場合は、「pg_stat_replication_slots の基本的な内部構造」という私のブログ記事を参照ください。また、私の同僚である Ajin Cherian が、PostgreSQL 14 における「二相コミットのロジカルデコーディング」というブログ記事を書いているので、こちらも参考にしてください。

2022 年 7 月 8 日