PostgreSQL インサイド

二相コミットのロジカルデコーディング

PostgreSQL 14 でコミットされた機能の先行紹介

技術者 Blog

Ajin Cherian
Fujitsu Australia Software Technology
Senior Software Development Engineer
PostgreSQL Contributor

はじめに

この記事では、論理レプリケーション（ロジカルレプリケーション）における二相コミットのデコードを可能にするために、富士通 OSS チームが PostgreSQL オープンソースコミュニティーと協力して PostgreSQL 14 で機能追加したことについて、説明します。

背景

名前が示すように、二相コミットは、トランザクションが 2 つの相（フェーズ）でコミットされるメカニズムです。これは通常、分散データベースにおいて一貫性を保つために行われます。トランザクションの 2 つの相は、「PREPARE」相と「COMMIT / ROLLBACK」相です。
PostgreSQL で二相コミットを行うために使用されるコマンドは、以下のとおりです。

- PREPARE TRANSACTION
- COMMIT PREPARED
- ROLLBACK PREPARED

PostgreSQL 8.0 で二相コミットを、PostgreSQL 10.0 で論理レプリケーションをサポートしましたが、論理レプリケーションにおける二相コミットはサポートされていませんでした。コマンドの PREPARE TRANSACTION、COMMIT PREPARED、および ROLLBACK PREPARED はインスタンス内ではサポートされていましたが、これらのコマンドをスタンバイに論理的にレプリケーションする必要がある場合には、元の意味を保たなくなりました。

PREPARE TRANSACTION コマンドは NOP として扱われ、まったくデコードされませんでした。COMMIT PREPARED コマンドは COMMIT として扱われ、ROLLBACK PREPARED コマンドは ABORT として扱われました。

二相コミットとは？

二相コミットはアトミックコミットプロトコルの一種で、分散データベース間における一貫性の維持に役立ちます。データベース内で原子性を提供するプレーンなコミットでは、複数のデータベースにまたがるトランザクションの一貫性を維持できない場合があります。

この問題を説明するために、例として以下のような前提条件を持った 2 つの異なる銀行における口座間での送金を見てみましょう。

- John は銀行 A に残高 300 ドルの口座を持っています。
- Mark は銀行 B に残高 100 ドルの口座を持っています。
- John は Mark に 100 ドル送金したいと思っています。

取り引きを実行すると以下のようなトランザクションが実行され、取り引きの終了時に両方の口座残高が 200 ドルになる必要があります。もし、いずれかのトランザクションが失敗した場合、口座の状態は取り引き開始前の状態に戻るはずです。

1. 銀行 A にある John の口座残高から 100 ドルを差し引く
2. 銀行 B にある Mark の口座残高に 100 ドルを加算する

トランザクションは、複数の理由で失敗する可能性があります。データベースは、トランザクションがコミットされる前に障害が発生した場合、そのトランザクションがロールバックされるように構築されています。
前出の例では、John の口座から差し引かれている間に障害が発生した場合、障害発生後の John の口座には、トランザクションが失敗したために差し引かれた金額は反映されません。これが、単一のコミットがデータベース内の一貫性を維持する方法です。

しかし、銀行 A にある John の口座から 100 ドルを差し引く取り引きは 1 回のコミットで成功したが、銀行 B にある Mark の口座に 100 ドルを加える取り引きは失敗し、ロールバックされたというシナリオを考えてみましょう。
この場合、John の口座からは 100 ドルが差し引かれますが、Mark の口座には 100 ドルが加算されず、100 ドルが消えてしまいます。
単一のコミットでは、このような分散トランザクションを扱うときに失敗する可能性があります。二相コミットは、この問題を解決するためのメカニズムとして進化しました。

二相コミットの場合、データベースの 1 つ（または外部の裁定者）が分散トランザクションのコーディネーターとして機能します。

第 1 相

1 つのデータベースがトランザクションの適用を開始し、次に「プリペア」を行います。プリペアドトランザクションをプリペアメッセージで他のデータベースに送信します。第 2 のデータベースはプリペアメッセージを取得し、トランザクションの準備も行います。準備では、トランザクションに含まれる変更を行いますが、コミットは行いません。変更はディスクに書き込まれるため、障害に耐えることができます。両方のデータベースがトランザクションを「プリペア」し、トランザクションに関するすべての必要な情報がディスクに格納されると、プリペア相（第 1 相）が完了します。

第 2 相

次に、裁定者はコミット・フェーズを開始します。第 2 のデータベースが何らかの理由でトランザクションの「プリペア」に失敗した場合、裁定者はロールバック・フェーズを開始します。したがって、2 番目のデータベースで「プリペア」が完了したどうかによって、変更は両方ともコミットされるか、両方ともロールバックされます。最終コミット・フェーズで発生した障害は、必要なプリペアドトランザクションがディスクに書き込まれ、再適用できるため、リカバリー可能です。

二相コミットは、実際には単一インスタンスのデータベース・インストレーションには関係ありませんが、複数のデータベース・インスタンス間でデータがレプリケーションされる大規模なインストレーションには関係があります。

このことから PostgreSQL が、論理レプリケーションにおいて二相コミットをサポートすることが重要なのです。

機能概要

PostgreSQL 13 までは、論理レプリケーションのトランザクションは、トランザクションがコミットされた後にのみデコードおよびレプリケーションされました。これは、最終的に中断される可能性のあるトランザクションのレプリケーションを避けるためでした。

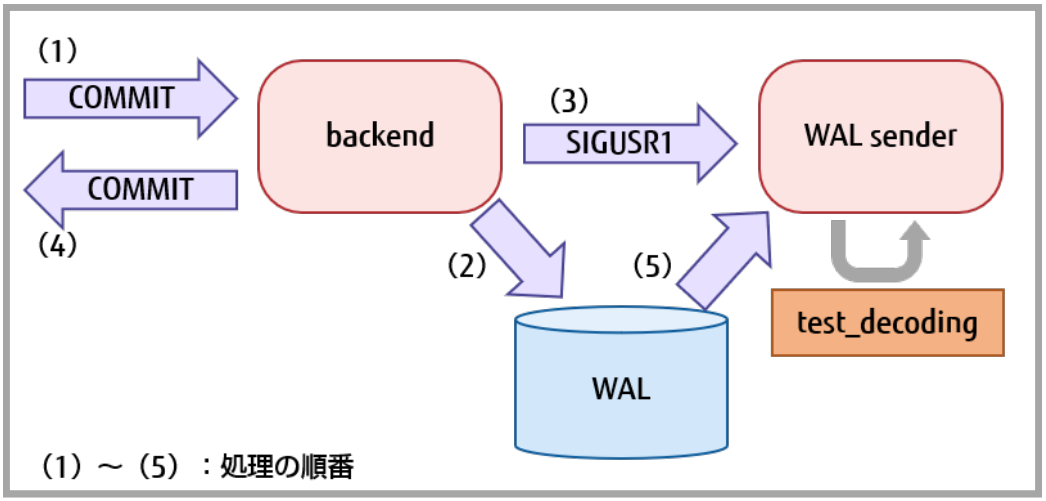

図 1：COMMIT によるトランザクションのデコード

富士通 OSS チームが PostgreSQL オープンソースコミュニティーのメンバーと協力して PostgreSQL 14 に実装した新機能により、PREPARE TRANSACTION、COMMIT PREPARED、ROLLBACK PREPARED の各コマンドが論理レプリケーションでサポートされるようになりました。PREPARE TRANSACTION コマンドのデコード時にトランザクションのデコードとレプリケーションが行われます。PREPARE TRANSACTION は、WAL sender の COMMIT コマンドと同様に、トランザクションの再実行とデコードを開始します。

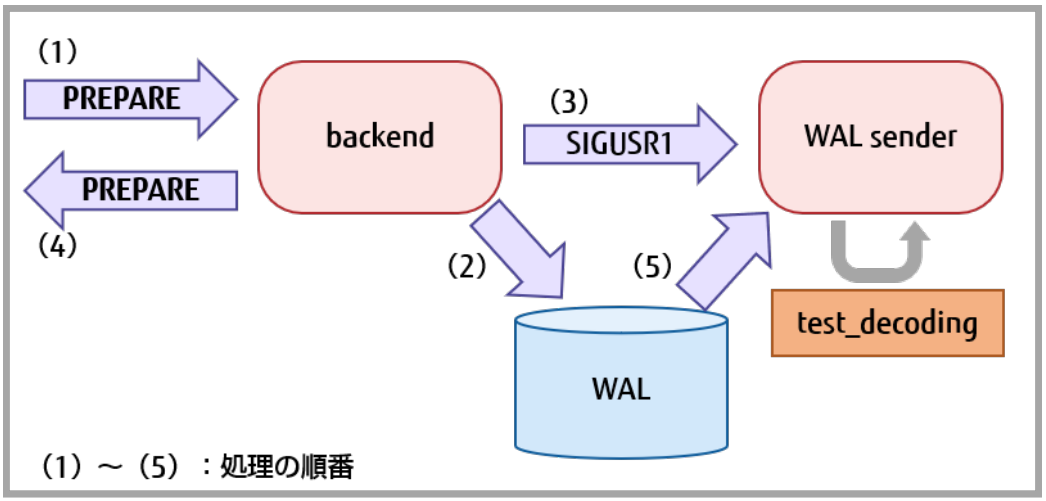

図 2：PREPARE によるトランザクションのデコード

また、ロジカルデコーディングのプラグインが二相コミットをサポートできるようにするための、新しいプラグインコールバックも定義しました。

新しい出力プラグインコールバックの一覧表

コールバック名	説明
filter_prepare_cb	PREPARE TRANSACTION コマンドで使用される GID に基づいて、プラグインが準備時にデコードする必要のないトランザクションをフィルタリングできるようにします。
begin_prepare_cb	プリペアドトランザクションの開始がデコードされる際に呼び出されます。

コールバック名	説明
prepare_cb	PREPARE TRANSACTION コマンドがデコードされる際に呼び出されます。
commit_prepared_cb	COMMIT PREPARED コマンドがデコードされる際に呼び出されます。
rollback_prepared_cb	ROLLBACK PREPARED コマンドがデコードされる際に呼び出されます。

実装の詳細は以下を参照してください。

- Extend the output plugin API to allow decoding of prepared xacts.（GitHub のページへ）
  https://github.com/postgres/postgres/commit/0aa8a01d04c8fe200b7a106878eebc3d0af9105c

プラグインの変更：test_decoding

test_decoding プラグインは、ユーザーが独自のロジカルデコーディング用プラグインを開発するための例となるロジカルデコーディング出力プラグインです。test_decoding は、ロジカルデコーディング機構を通して WAL を受け取り、実行された操作のテキスト表現にデコードします。

test_decoding プラグインは、準備時に新しい二相コールバックとデコードトランザクションを使用できるように変更されました。

関数の変更：pg_create_logical_replication_slot()

この関数では、スロットが二相コミットをサポートするかどうかを指定する新しいオプションが追加されました。two_phase オプションを指定して作成されたレプリケーションスロットは、二相コミットをサポートするために出力プラグインで使用できます。

```
pg_create_logical_replication_slot(slot_name name, plugin name [, temporary boolean, two_phase boolean ] )
```

詳細は以下を参照してください。

- Add option to enable two_phase commits via pg_create_logical_replicat...（GitHub のページへ）
  https://github.com/postgres/postgres/commit/19890a064ebf53dedcefed0d8339ed3d449b06e6

実行例

二相コミットにおいてトランザクションの出力をデコードしてみます。

1. レプリケーションスロットの作成

   「regression_slot」という名前のレプリケーションスロットを作成します。出力プラグインとして test_decoding を使用し、true を渡して、スロットが二相コミットのデコードをサポートするようにします。

```
postgres=# SELECT * FROM pg_create_logical_replication_slot('regression_slot', 'test_decoding', false, true);
    slot_name    |    lsn
-----------------+-----------
 regression_slot | 0/16B1970
(1 row)
```


2. テーブルの作成

```
postgres=# CREATE TABLE data(id serial primary key, data text);
CREATE TABLE
```

3. 二相コミットの実行とデコードされた結果の出力

   プリペアドトランザクションとコミットされたトランザクションのデコードされた変更情報を出力します。

```
postgres=# BEGIN;
postgres=*# INSERT INTO data(data) VALUES('5');
postgres=*# PREPARE TRANSACTION 'test_prepared1';

postgres=# SELECT * FROM pg_logical_slot_get_changes('regression_slot', NULL, NULL);
    lsn    | xid |                          data
-----------+-----+--------------------------------------------------------
 0/1689DC0 | 529 | BEGIN 529
 0/1689DC0 | 529 | table public.data: INSERT: id[integer]:3 data[text]:'5'
 0/1689FC0 | 529 | PREPARE TRANSACTION 'test_prepared1', txid 529
(3 rows)

postgres=# COMMIT PREPARED 'test_prepared1';
postgres=# select * from pg_logical_slot_get_changes('regression_slot', NULL, NULL);
    lsn    | xid |                    data
-----------+-----+-------------------------------------------
 0/168A060 | 529 | COMMIT PREPARED 'test_prepared1', txid 529
(4 row)

postgres=# select * from data;
id | data
----+------
  1 | 5
(1 row)
```

4. プリペアドトランザクションのロールバック実行とデコードされた結果の出力

   プリペアドトランザクションとロールバックされたトランザクションのデコードされた変更情報を出力します。

```
postgres=#-- you can also rollback a prepared transaction
postgres=# BEGIN;
postgres=*# INSERT INTO data(data) VALUES('6');
postgres=*# PREPARE TRANSACTION 'test_prepared2';
postgres=# select * from pg_logical_slot_get_changes('regression_slot', NULL, NULL);
    lsn    | xid |                          data
-----------+-----+--------------------------------------------------------
 0/168A180 | 530 | BEGIN 530
 0/168A1E8 | 530 | table public.data: INSERT: id[integer]:4 data[text]:'6'
 0/168A430 | 530 | PREPARE TRANSACTION 'test_prepared2', txid 530
(3 rows)
```



```
postgres=# ROLLBACK PREPARED 'test_prepared2';
postgres=# select * from pg_logical_slot_get_changes('regression_slot', NULL, NULL);
    lsn    | xid |                    data
-----------+-----+-------------------------------------------
 0/168A4B8 | 530 | ROLLBACK PREPARED 'test_prepared2', txid 530
(1 row)

postgres=# select * from data;
id | data
----+------
  1 | 5
(1 row)
```

今後に向けて

PostgreSQL 14 で加えられた機能変更により、準備時に二相コミットをデコードできるデコーダー側の基盤ができました。この基盤を利用するために test_decoding プラグインも変更しました。

次のステップは、PostgreSQL 内で最大のロジカルデコーディングプラグインである pgoutput プラグインに、二相コミットのサポートを実装することです。pgoutput プラグインは、PostgreSQL の論理レプリケーションの PUBLISHER / SUBSCRIBER モデルをサポートします。また、最も広く使用されている論理レプリケーション用のプラグインでもあります。富士通 OSS チームは、PostgreSQL 15 でこのサポートを追加するために、オープンソースコミュニティーと協力しています。

二相トランザクションが分散データベースで動作するためには、PostgreSQL は、失敗した「プリペア」をマスターに通知し、「ロールバック」を開始するようスタンバイ側をサポートする必要もあります。このタイプのスタンバイフィードバック機構は PostgreSQL には存在せず、将来の改善候補です。

2021 年 8 月 27 日