PostgreSQL インサイド

PostgreSQL 14 での論理レプリケーションの改善

技術者 Blog

Amit Kapila
FUJITSU Limited
Software Products Business Unit Data Management Division Senior Director
PostgreSQL Committer and Major Contributor

はじめに

PostgreSQL 14 では、論理レプリケーション（Logical Replication）_(https://www.postgresql.org/docs/14/logical-replication.html) に関する複数のエンハンスが行われました。これにより、論理レプリケーションがさらに活用されることを期待しています。このブログは主に、論理レプリケーションのすべての機能エンハンスについてまとめ、簡単に説明します。

大規模なトランザクションのデコーディング

実行中の大規模なトランザクションをサブスクライバーにストリーミングできるようにします。PostgreSQL 14 より前のバージョンでは、トランザクションはコミット時にのみストリームされていたため、大規模なトランザクションでは大きな適用遅延が発生していました。この機能により、適用遅延が軽減され、特定のシナリオでは性能が大幅に向上します。この機能については、前の blog「実行中のトランザクションの論理レプリケーション」で詳しく説明しています。

ロジカルデコーディングの性能

大量の DDL を実行するトランザクションの CPU 使用率を削減し、デコードの性能を向上させます。1000 パーティションのテーブルで全データを削除する処理（TRUNCATE）を含むトランザクションのデコードは、この改善の前は 4 分から 5 分かかっていましたが、1 秒で完了することが確認されています。この性能向上をどのように実現したかを説明する前に、この最適化を理解するために重要な PostgreSQL の無効化メッセージについて簡単に説明します。これらは、各バックエンドセッションで不可視のシステムキャッシュエントリーをフラッシュするためのメッセージです。通常は、これらを生成したバックエンドのコマンドの終了時に実行し、トランザクションの終了時に共有キューを介して他のバックエンドに送信して処理します。これらは通常、DDL 操作で発生するシステムカタログに対する挿入 / 削除 / 更新操作のために生成されます。
デコード中は、各コマンド終了時にトランザクション全体のすべての無効化を実行していました。これは、そのコマンドの前にどの無効化が行われたかを知る方法がなかったためです。このため、大量の DDL を含むトランザクションでは時間がかかり、CPU 使用率も高くなります。しかし現在では、各コマンドの終了時において明確に無効化が行われることがわかっているため、必要な無効化だけを実行します。これは、d7eb52d718：Execute invalidation messages for each XLOG_XACT_INVALIDATIONS message_ (https://git.postgresql.org/gitweb/?p=postgresql.git;a=commitdiff;h=d7eb52d7181d83cf2363570f7a205b8eb1008dbc) のコミットによって実現されました。

初期テーブル同期

初期テーブル同期には、テーブル同期ワーカによるテーブルの初期スナップショットのコピーが含まれます。その後、テーブルはメインの適用ワーカと同期された状態になります。この作業全体は、一時的なレプリケーションスロットを使用して単一のトランザクションで実行されるため、次のような大きな欠点があります。

- スロットは、すべての同期が完了するまで WAL（Write Ahead Log / ログ先行書き込み）を保持します。
- 同期フェーズ中にエラーが発生すると、コピーしたすべてのデータがロールバックされるため、大量データコピーの問題が発生します。
- CID 制限を超えるリスクがあります。

これらの欠点を克服するために PostgreSQL 14 では以下の改善を行いました。

- テーブル同期フェーズで複数のトランザクションを実行できるようにしました。
- テーブル同期の進行状況を追跡するために、永続的なスロットと起点を使用しました。

この作業については、blog「論理レプリケーションのテーブル同期ワーカ」で詳しく説明しています。これは、ce0fdbfe97：Allow multiple xacts during table sync in logical replication. (https://git.postgresql.org/gitweb/?p=postgresql.git;a=commitdiff;h=ce0fdbfe9722867b7fad4d3ede9b6a6bfc51fb4e)のコミットによって実現されました。

二相コミットのロジカルデコーディング

トランザクションのデコードをコミット時ではなく、プリペア時に行い、出力プラグインに送信できるようになります。これにより、プラグインはプリペア時にトランザクションを解読し、必要に応じて別のノードにルーティングできます。これには、次の 2 つの利点があります。

- 論理レプリケーションによって、複数のノードにまたがる二相の分散トランザクションが可能になります。
- プリペア時に別のノードでトランザクションを送信して再実行することで、適用遅延を軽減できます。

この作業については、blog「二相コミットのロジカルデコーディング」で詳しく説明しています。これは、0aa8a01d04：Extend the output plugin API to allow decoding of prepared xacts.(https://git.postgresql.org/gitweb/?p=postgresql.git;a=commitdiff;h=0aa8a01d04c8fe200b7a106878eebc3d0af9105c)、a271a1b50e：Allow decoding at prepare time in ReorderBuffer. (https://git.postgresql.org/gitweb/?p=postgresql.git;a=commitdiff;h=a271a1b50e9bec07e2ef3a05e38e7285113e4ce6)、19890a064e：Add option to enable two_phase commits via pg_create_logical_replication_slot. (https://git.postgresql.org/gitweb/?p=postgresql.git;a=commitdiff;h=19890a064ebf53dedcefed0d8339ed3d449b06e6)のコミットによって実現されました。

ロジカルデコーディングの監視

レプリケーションスロットは、クラスタからのレプリケーションストリームに関する状態を保持するために使用されます。その主な目的は、WAL の早期削除を防ぐことです。論理レプリケーションのコンテキストでは、スロットは、元のサーバーで行われた順序でクライアントに再生できるような変更のストリームを表します。各スロットは、単一のデータベースから一連の変更をストリームします。
PostgreSQL 14 では、レプリケーションスロットの活動を報告するシステムビュー（pg_stat_replication_slots）(https://www.postgresql.org/docs/14/monitoring-stats.html#MONITORING-PG-STAT-REPLICATION-SLOTS-VIEW) を追加しました。これは、出力プラグインまたはサブスクライバーにストリームされ、ディスクに書き出したデータの量を監視するために使用できます。さらに、ユーザーは、このスロットの WAL からの変更をデコードする間、デコーディングの出力プラグインにトランザクションを送信するためにデコードされた、トランザクションデータの総量を監視することができます。ここには、ストリーミングされたデータやオーバーフローしたデータが含まれることに注意してください。pg_stat_reset_replication_slot() 関数は、スロットの統計情報をリセットします。

例

```
CREATE TABLE stats_test(data text);
SET logical_decoding_work_mem to '64kB';
SELECT 'init' FROM pg_create_logical_replication_slot('slot_stats', 'test_decoding');
INSERT INTO stats_test SELECT 'serialize-topbig--1:'||g.i FROM generate_series(1, 5000) g(i);
SELECT count(*) FROM pg_logical_slot_peek_changes('slot_stats', NULL, NULL, 'skip-empty-xacts', '1');

SELECT slot_name, spill_txns, spill_count, spill_bytes, total_txns, total_bytes FROM pg_stat_replication_slots;
 slot_name  | spill_txns | spill_count | spill_bytes | total_txns | total_bytes
------------+------------+-------------+-------------+------------+-------------
 slot_stats |          1 |          12 |      763893 |          1 |      763893
(1 row)

DROP TABLE stats_test;

SELECT pg_drop_replication_slot('slot_stats');
```

パブリケーションの追加と削除の簡易化

PostgreSQL 13 までは、ユーザーがサブスクリプションに追加したパブリケーションを追加または削除する必要がある場合、既存のすべてのパブリケーションも一緒に指定する必要がありました。サブスクリプションが 2 つのパブリケーションにサブスクライブされていて、さらにパブリケーションを追加したい場合、ユーザーは Alter Subscription を実行するときに、3 つすべて（以前の 2 つと新規の 1 つ）を指定する必要があります。次に例を示します。

初期のサブスクリプション

```
CREATE SUBSCRIPTION mysub CONNECTION 'host=localhost port=5432 dbname=postgres' PUBLICATION mypub1, mypub2;
```

新規のパブリケーション mypub3 を追加

```
ALTER SUBSCRIPTION mysub SET PUBLICATION mypub1, mypub2, mypub3;
```

これは、特にサブスクリプションがサブスクライブされている既存のパブリケーションが多数ある場合に、ユーザーにとって不便になる可能性があります。PostgreSQL 14 では、ADD / DROP 個々のパブリケーションをサポートすることで、これを簡易にする新しい方法を追加しました。構文については PostgreSQL 文書「ALTER SUBSCRIPTION」(https://www.postgresql.org/docs/14/sql-altersubscription.html)を参照してください。新しい方法で、上記のケースにおいて新しいパブリケーションを追加するには、以下の操作を実行する必要があります。

```
ALTER SUBSCRIPTION mysub ADD mypub3;
```

これは、82ed7748b7：ALTER SUBSCRIPTION ... ADD/DROP PUBLICATION_
(https://git.postgresql.org/gitweb/?p=postgresql.git;a=commitdiff;h=82ed7748b710e3ddce3f7ebc74af80fe4869492f)のコミットによって実現されました。

バイナリー転送モード

この機能は、パブリッシャーからのデータをバイナリー形式で送信できるように、サブスクリプションの作成 / 変更時にオプションを提供します。このオプションのデフォルト値は false です。このオプションを有効にしても、バイナリーの送受信関数を持つデータ型のみがバイナリーで転送されます。クロスバージョンレプリケーションを実行する場合、サブスクライバーにそのタイプのバイナリー受信関数がないと、データ転送が失敗し、このオプションは使用できません。通常、このモードの方が高速です。

バイナリーモードを有効にする例

```
CREATE SUBSCRIPTION mysub CONNECTION 'host=localhost port=5432 dbname=postgres' PUBLICATION mypub1 WITH (binary = true);
```

これは、9de77b5453：Allow logical replication to transfer data in binary format.
(https://git.postgresql.org/gitweb/?p=postgresql.git;a=commitdiff;h=9de77b5453130242654ff0b30a551c9c862ed661) のコミットによって実現されました。

pgoutput 経由でメッセージを取得可能に

pgoutput プラグインに「messages」オプションを提供します。これにより、ロジカルデコーディングメッセージ（つまり、pg_logical_emit_message 経由で生成されたもの）をスロットの消費者に送信でき、変更データキャプチャー（CDC）にこれを使用する、pgoutput プラグインのユーザーに役立ちます。以下に例を示します。

```
SELECT pg_create_logical_replication_slot('pgout_slot', 'pgoutput');
SELECT pg_logical_emit_message(true, 'pgoutput', 'a transactional message');
SELECT get_byte(data, 1), encode(substr(data, 24, 23), 'escape')  FROM pg_logical_slot_peek_binary_changes('pgout_slot', NULL, NULL, 'proto_version', '1', 'publication_names', 'pgout_slot', 'messages', 'true') OFFSET 1 LIMIT 1;
 get_byte |         encode
----------+------------------------
        1 | a transactional message
(1 row)
SELECT pg_drop_replication_slot('pgout_slot');
```

変更を取得している間、publication_names はロジカルデコーディングメッセージには必要としていませんが、関数がエラーを返さないように指定されています。論理レプリケーションメッセージのフォーマットについては、PostgreSQL 文書「53.9. Logical Replication Message Formats」(https://www.postgresql.org/docs/14/protocol-logicalrep-message-formats.html) で参照できます。これは、ac4645c015：Allow pgoutput to send logical decoding messages
(https://git.postgresql.org/gitweb/?p=postgresql.git;a=commitdiff;h=ac4645c0157fc5fcef0af8ff571512aa284a2cec) .のコミットによって実現されました。

2021 年 11 月 26 日