D-006

PostgreSQL によるデータ仮想化エンジンを実現する拡張機能 PGSpider
PGSpider extension on PostgreSQL to construct Data Virtualization engine

片山 大河†　廣瀬 繁雄†　金松 基孝†
Taiga Katayama　Shigeo Hirose　Mototaka Kanematsu

1. はじめに

CPS（Cyber Physical Systems：実世界のデータを活用しサイバー世界で分析を行い、実世界にフィードバックして付加価値を創造する仕組み）の分野では点在するデータを一箇所に集めることなく、データソースからリアルタイムに取得するデータ仮想化技術が注目されている。これら分散したデータはスキーマが同じであるという特徴があり、それらを仮想的なひとつのテーブルとみなしてアクセスしたいという一方で、各レコードがどこのデータソースに格納されているかが分かることも求められる。

そこで我々は OSS データベース PostgreSQL[1]の外部データラッパ（Foreign Data Wrapper：FDW）という機能拡張の仕組みを利用して、これらの機能を PGSpider extension として実現した（以降 PGSpider と呼ぶ）。これをパラレルクエリおよび宣言的パーティショニングと組み合わせることで、仮想テーブルを実現するとともにデータソースへのアクセスを並列化し、PostgreSQL をデータ仮想化エンジンとして活用できるようにした。本稿では、PGSpider の機能と仕組みについて述べる。

2. PostgreSQL の機能

本章では、今回使用する外部データラッパと宣言的パーティショニング機能について説明する。

2.1 外部データラッパ

PostgreSQL には動的にパッケージを読み込み、機能を拡張する仕組みが備わっている。外部データラッパはそのひとつで、PostgreSQL から他のデータソースへのアクセスを可能とする機能拡張である。PostgreSQL core が外部データラッパを介してデータソースに接続する構成になる（図 1）。以降これらをデータソース FDW と呼ぶ。

データソース FDW はデータソースに依存する処理を実装したライブラリであり、様々なデータソース FDW が公開されていている[2]。PostgreSQL core から呼ばれる関数の I/F が定義されていて、その仕様に基づいてデータソース FDW を独自に開発すれば、所望のデータソースにアクセスできるように拡張可能である。

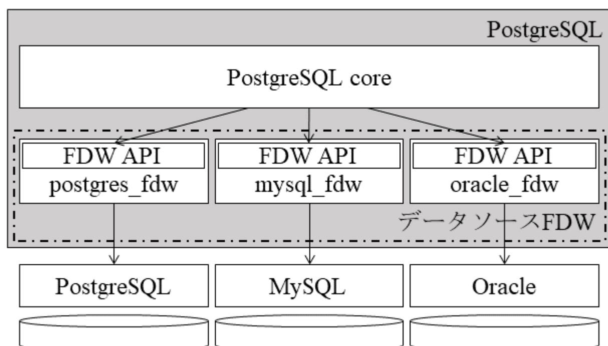

図 1 外部データラッパのモジュール関係

データソース FDW のメカニズムは、まずプラン構築フェーズでは、PostgreSQL core から与えられる問い合わせ情報を解析しプランを構築する。ここでは、どのカラムをデータソースから取得するか、WHERE 条件式の評価をデータソース上で行うか、といったプッシュダウン判定を行う。次に、データソースの仕様にしたがってデータソース向けの問い合わせ情報を構築する。例えばデータソースがリレーショナルデータソースの場合は SQL 文を作成する。その後、プラン情報を PostgreSQL core に返す。これによって、PostgreSQL core はデータソース上で行う処理を把握し、PostgreSQL core 上で行うべき処理を決定できる。

次に、クエリ実行フェーズでは、データソースの API を使用して問い合わせを実行して結果セットを受け取る。それを PostgreSQL のレコード形式に変換して、PostgreSQL core に返す。

2.2 宣言的パーティショニング

ひとつのテーブルをある規則にしたがって複数の子テーブル群に分割して管理する機能である。詳細は[3]を参照されたい。現在 PostgreSQL では、パーティション分割の規則は 3 つの方式（範囲、リスト、ハッシュ）をサポートしている。今回利用するリストパーティショニングについて簡単に説明する。

この方式は、パーティションキーにテーブルのカラムを指定する。レコードは、このキーカラムの値に応じてテーブルが決められる。例えば、このキーカラムの取り得る値のバリエーションが 5 つの場合、各レコードは 5 つの子テーブルで管理される。

3. PGSpider extension の提案

PostgreSQL をデータ仮想化エンジンとして活用するためには、（1）点在するデータソース上のデータに対して、ユーザは個別にアクセスすることなく仮想的化によってまとめて横断アクセスできること、および（2）取得した各データの所在地が分かることが必要と考える。

3.1 現状の課題と解決方法

3.1.1 データの所在地の管理

まず（2）について、あらかじめこのような用途を想定してデータソースのシステムを設計していれば、各データに所在地を表す情報を持たせるなどして対応可能であり問題にはならない。しかし、既設システムがデータソースの場合、各データがその情報を持っているとは限らなく、既設システムを変更しなければならない。

そこで、既設システムを変更することなくこの課題を解決するために、PGSpider を考案した。PGSpider は、データ

†株式会社東芝　TOSHIBA CORPORATION

Copyright © 2021 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

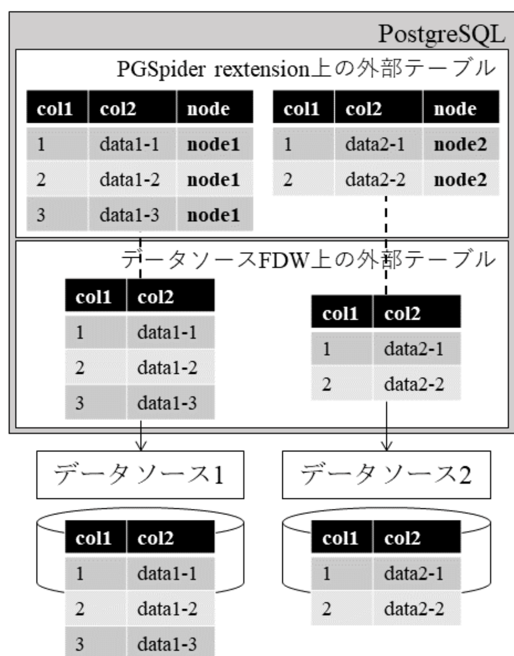

図 2 各テーブルのカラム構成

ソース FDW を使ってデータベースにアクセスし、取得した各データにデータソース情報を付加する機能を持つ。つまり PostgreSQL 上では、データソース情報を格納するカラム（node カラム）を追加で用意することになる。例えば、データソース上に 2 カラムあるテーブルがあるとすると、PostgreSQL 上では 3 カラムになる（図 2）。

3.1.2 仮想化によるデータソース横断アクセス

次に（1）の仮想化方法でまず思いつく手段は VIEW を使う方法である。各外部テーブルに対する SELECT クエリを UNION ALL でまとめた VIEW を作成する。しかし、この方法では集約関数のプッシュダウンや並列処理（詳細は 4.3 節）を行うことができない。PostgreSQL core は VIEW 内の SELECT クエリ間で連携したプランを作成しないため、これを実現しようと思うと PostgreSQL core 側を修正しなければならない。

そこで今回は宣言的パーティショニングと FDW の機能を組み合わせて課題を解決した。データソース上のテーブルに対応する各外部テーブルがひとつのパーティション（子テーブル）で、親テーブルが仮想化テーブルとして統合されたデータに横断アクセスできるようにする。

しかし、ここで問題となるのがデータソースでパーティションを切るために、パーティションキーを設定する方法である。データソース上のデータにはデータソースを識別する情報がないからである。そこで、課題（2）の解決策で導入した node カラムを利用することとした。パーティション方式にはリストを使い、node カラムをパーティションキーとする。

4. PGSpider extension の実現方法

PGSpider も FDW の仕組みを利用して拡張機能化を実現した。テーブル構成を図 3 に示す。

4.1 テーブル間の連携方法

PGSpider は PostgreSQL core とデータソース FDW を仲介していて、パーティションの子テーブルは PGSpider 上のテーブルとして作成する。各子テーブルはデータソース FDW 上の外部テーブルに 1 対 1 で対応付けていて、データソース FDW を介してデータソースにアクセスする。この対応付けは、外部テーブル名のネーミングルールに基づいて決定する仕様とした。具体的には、子テーブル名＋"_child"を連携先の外部テーブル名とした。

なお、データソース FDW での外部テーブルとデータソース上のテーブルの対応付け方法はデータソース FDW によって異なる。postgres_fdw の場合、外部テーブル名と同じ名前としていて、ユーザがテーブルオプションでデータソース上のテーブル名を指定することも可能になっている。

4.2 処理の内容

PGSpider は以下で説明する動作を行うように FDW の関数を実装する。まず、プラン構築フェーズ（図 4）：

1. PostgreSQL core から子テーブルの問い合わせ情報を受ける。
2. 4.1 節で説明した対応付けルールに従って、データソース FDW 上の外部テーブルを特定する。
3. 外部テーブル向けの問い合わせ情報を作成する。処理 1 で受け取った情報は子テーブル向けの問い合わせ情報でありオブジェクトの識別子（テーブル OID）などが外部テーブル用と異なるため、この処理が必要になる。
   また、もし node 列に関する情報が含まれていれば、外部テーブルはそれを認識できないためを除外する必要もある。
4. 処理 3 で作成した情報をデータソース FDW に与え、外部テーブルのプランを構築する。
5. 外部テーブルのプランを元にして子テーブルのプランを構築し、プラン情報を PostgreSQL core に返す。データソース上での処理と PostgreSQL core 上での処理で整合性を図るためである。
   もし処理 3 で node 列に関する情報を除外していたら、ここで復元する。

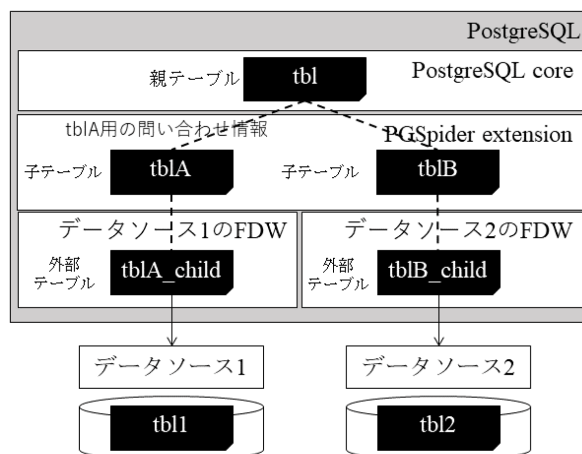

図 3 テーブル構成

Copyright © 2021 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

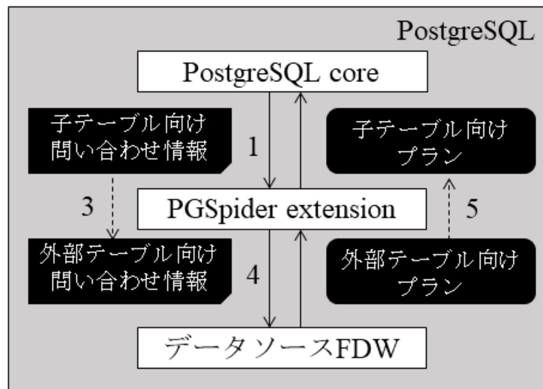

図 4 外部データラッパ間での情報の流れ

次にクエリ実行フェーズ：

6. 処理 2 で特定したデータソース FDW を使ってデータソースにアクセスし、結果セットを受け取る。データソース FDW 側で PostgreSQL のレコード形式にしているため、PGSpider ではデータ形式を変換する必要はない。

7. データソース FDW が取得した結果セットに対し、必要に応じて node カラムの値を設定し、PostgreSQL core に結果セットを返す。ここで設定する値は、子テーブルのパーティションキーの値を使用する。

4.3 高速化手法

高速化のために集約関数をプッシュダウンする。集約関数をデータソース上で計算させることにより、PostgreSQL が取得するデータ量を削減して高速化する。この機能はデータソース FDW に備わっているものを利用して実現する。つまりデータソース FDW がプッシュダウン判定を行い、プランを構築する。通常のクエリ（パーティション化されていないテーブルに対するクエリ）の場合、特別な対応を行う必要はない。しかし、パーティションテーブルに関連するクエリの場合は対策が必要になる。

データソース FDW は集約関数プッシュダウン可否の判定で集約関数の属性を参照する。その属性が通常クエリとパーティション関連クエリとで異なり、データソース FDW は通常クエリ時の属性の場合のみプッシュダウンする可能と判定するように実装されている場合がある。例えば、属性のひとつに AggSplit という関数の種別を表現する属性がある。通常クエリでは AGGSPLIT_SIMPLE が設定されているが、パーティション関連クエリでは AGGSPLIT_INITIAL_SERIAL が設定されていて、postgres_fdw では AGGSPLIT_SIMPLE がプッシュダウン可能の必要条件となっている。

理論的にはいずれの設定値でも関数はプッシュダウン可能である。そこで、PGSpider が一時的にその設定値を変更し、データソース FDW にとっては通常クエリ上の集約関数と認識させるようにした。具体的には、外部テーブル向けの問い合わせ情報の作成時（4.2 節の処理 3）に通常クエリにみなすように設定値を書き換え、子テーブルのプランを構築時（4.2 節の処理 5）に元に戻す。このように PGSpider 側に手を加えることで、各データソース FDW のソースコードを都度変更する必要なく実現することができる。

4.4 動作例

動作確認に使用したサーバおよびテーブル構成は図 3 のとおりである。2 台のデータソースは、一方が PostgreSQL（バージョン 13.2）で、同一マシン上にメインの PostgreSQL とは異なるポートを使用して立ち上げた。もう一方は MySQL（バージョン 8.0）を使用した。また、postgres_fdw は PostgreSQL に付属のものを、mysql_fdw は [4]を使用した。

コードは公開予定である。

4.4.1 データの準備

データソース上には、それぞれデータベース test1 および test2 を作成し、テーブル tbl1 および tbl2 にはランダムなデータを 65536 件登録した。65536 件とした理由は、並列プランが構築されることを確認するためである。PostgreSQL はコストベースでプランを構築していて、見積もりコストが小さいと並列プランを作成しない仕様となっている。実行した SQL は以下の通りである。

データソース 1（PostgreSQL）：

```
CREATE TABLE tbl1(col1 integer, col2 text);
INSERT INTO tbl1 VALUES (generate_series(1,65536), 'data1-'
    || generate_series(1,65536));
```

データソース 2（MySQL）：

```
CREATE TABLE tbl2(col1 integer, col2 text);
INSERT INTO tbl2 VALUES(1, 'data1-1');
INSERT INTO tbl2 SELECT col1 + c.num,
  CONCAT('data1-', (col1 + c.num)) FROM tbl2
  (SELECT count(col1) AS num FROM tbl2) AS c; --16 回実行
```

4.4.2 仮想テーブルの準備

仮想テーブルを作成する PostgreSQL 上で以下の設定を行う。以下の SQL を実行した。

postgres_fdw の外部テーブルの設定：

```
CREATE EXTENSION postgres_fdw;
CREATE SERVER datasource1 FOREIGN DATA WRAPPER
    postgres_fdw OPTIONS (port '5433', dbname 'test1',
    use_remote_estimate 'true');
CREATE USER MAPPING FOR public SERVER datasource1;
CREATE FOREIGN TABLE tblA_child(col1 integer, col2 text)
    SERVER datasource1 OPTIONS (table_name 'tbl1');
```

mysql_fdw の外部テーブルの設定：

```
CREATE EXTENSION mysql_fdw;
CREATE SERVER datasource2 FOREIGN DATA WRAPPER
    mysql_fdw OPTIONS (host '127.0.0.1', port '3306',
    use_remote_estimate 'true');
CREATE USER MAPPING FOR public SERVER datasource2
    OPTIONS(username 'user', password 'Pwd123');
CREATE FOREIGN TABLE tblB_child(col1 integer, col2 text)
    SERVER datasource2 OPTIONS (dbname 'test2',
    table_name 'tbl2');
```

PGSpider を使用してパーティションテーブルの作成：

```
CREATE EXTENSION pgspider_ext;
CREATE SERVER pgspider FOREIGN DATA WRAPPER
    pgspider_ext;
CREATE USER MAPPING FOR public SERVER pgspider;
CREATE TABLE tbl(col1 int, col2 text, node text) PARTITION
    BY LIST (node);
```


Copyright © 2021 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

```
CREATE FOREIGN TABLE tblA PARTITION OF tbl FOR
    VALUES IN ('node1') SERVER pgspider;
CREATE FOREIGN TABLE tblB PARTITION OF tbl FOR
    VALUES IN ('node2') SERVER pgspider;
```

集約関数の計算を並列処理させるための設定：

```
SET enable_partitionwise_aggregate TO on;
```

4.4.3 動作確認

仮想テーブル tbl を使ってクエリを実行できる。以下のように、各レコードの node 列にはデータソースの名前が格納できていることが分かる。

```
SELECT * FROM tbl LIMIT 5 OFFSET 100000;
 col1  |     col2      | node
-------+---------------+-------
 51902 | data1-51902 | node1
 48100 | data1-48100 | node2
 48101 | data1-48101 | node2
 51903 | data1-51903 | node1
 51904 | data1-51904 | node1
```

また、仮想テーブル tbl は、各データソース上のテーブルをまとめた合計 131,072 件を持つテーブルであることが確認できる。

```
SELECT count(col1) FROM tbl;
 count
--------
 131072
```

次にクエリプランを確認した。なお、表示の都合上、実行結果のレイアウトは修正している。

```
EXPLAIN (VERBOSE, COSTS OFF) SELECT col1 FROM tbl;
                        QUERY PLAN
---------------------------------------------------
 Gather
  Output: tbl.col1
  Workers Planned: 2
  -> Parallel Append
    -> Foreign Scan on public.tbla tbl_1
      Output: tbl_1.col1
       Remote SQL: SELECT col1 FROM public.tbl1
    -> Foreign Scan on public.tblb tbl_2
      Output: tbl_2.col1
       Remote SQL: SELECT `col1` FROM `test2`.`tbl2`
```

上記のように、Parallel Append が使用された並列化プランになっていることが確認できる。また、集約関数の場合は下記のように、Parallel Append による並列化プランになり、count がデータソースにプッシュダウンできている。

```
EXPLAIN (VERBOSE, COSTS OFF)
    SELECT count(col1) FROM tbl;
                       QUERY PLAN
---------------------------------------------------
 Finalize Aggregate
  Output: count(tbl_1.col1)
  -> Gather
      Output: (PARTIAL count(tbl_1.col1))
      Workers Planned: 2
      -> Parallel Append
          -> Foreign Scan
                    Output: (PARTIAL count(tbl_1.col1))
                      Relations: Aggregate on
                                  (public.tbla tbl_1)
                      Remote SQL: SELECT count(col1)
                                   FROM public.tbl1
               -> Foreign Scan
                    Output: (PARTIAL count(tbl.col1))
                      Local server startup cost: 10
                      Remote SQL: SELECT count(`col1`)
                                  FROM `test2`.`tbl2`
```

5. おわりに

PostgreSQL をデータ仮想化エンジンとして活用できるように PGSpider extension を提案した。

パラレルクエリおよび宣言的パーティショニングと組み合わせることで、複数のデータソース上のデータを仮想的なひとつのテーブルとみなして並列にアクセス可能になった。また、集約関数をプッシュダウンさせて高速化も図った。

本機能は、PostgreSQL の Extension として実現した。従来版と異なり PostgreSQL 本体のソースコードに手を入れる必要がないため、PostgreSQL 本体のバージョンアップに容易に追従できるようになっている。なお、ソースコードは[5]で公開予定である。

データ仮想化によるデータベース横断アクセスは今後重要な技術であると捉えている。PostgreSQL は、様々なデータソースにアクセスが可能になる FDW を備え、様々なエコシステムが OSS で公開されているため、非常に強力なツールと考えている。今後も PostgreSQL をベースに、データ仮想化エンジンに求められそうな機能を検討し実現したい。

参考文献

[1] PostgreSQL https://www.postgresql.org/
[2] Foreign data wrappers
https://wiki.postgresql.org/wiki/Foreign_data_wrappers
[3] PostgreSQL Table Partitioning
https://www.postgresql.org/docs/13/ddl-partitioning.html
[4] mysql_fdw https://github.com/pgspider/mysql_fdw
[5] PGSpider extension https://github.com/pgspider/pgspider_ext

Copyright © 2021 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.