Vol.2017-DBS-165 No.13
Vol.2017-IFAT-128 No.13
2017/9/18

PostgreSQLにおける複数外部データソース並列スキャン機能と即時結果取得機能の開発

片山大河[†1]　嶋村誠[†1]　金松基孝[†1]

概要：PostgreSQLにはForeign Data Wrapperという機能があり，MySQLやcsvファイルのような異なるデータソースと連携できる．しかし，システムで使用する外部データソースが多いと，データの収集に時間を要する．このようなケースには例えば，異種分散データベースや，IoTゲートウェイでのセンサデータの保管といった用途が考えられる．そこで，複数のデータソース間でスキーマが同一であるという特徴を利用した並列スキャンにより高速に検索する機能を開発した．また，時間をかけて正確な結果を算出するよりも，時間をかけずに曖昧な結果を算出することが望まれることがある．この用途のために，データソースからの収集が未完了でも，収集済みのデータのみを使用して算出した結果をアプリケーションに応答できる機能を開発した．

Development of parallel scan feature from multiple of foreign data sources and immediate result acquisition feature on PostgreSQL

TAIGA KATAYAMA[†1]　MAKOTO SHIMAMURA[†1]
MOTOTAKA KANEMATSU[†1]

1. はじめに

IoT（Internet of Things）機器の開発が活発になり，あらゆるものがインターネットに接続されるようになってきている．図1のように，IoT機器は，　IoTゲートウェイを通してクラウドと接続している．IoTゲートウェイはルータとして配置され，クラウドとIoT機器のメッセージの中継を行う．そのような機器から発生するデータは日々増大していて,データをクラウドシステムへ保管するだけでなく，IoTゲートウェイでデータを保管することがある．ここで，IoTゲートウェイ上でデータを格納するデータベース管理システム（DBMS）は必ずしも同一の種類のDBMSとは限らない．

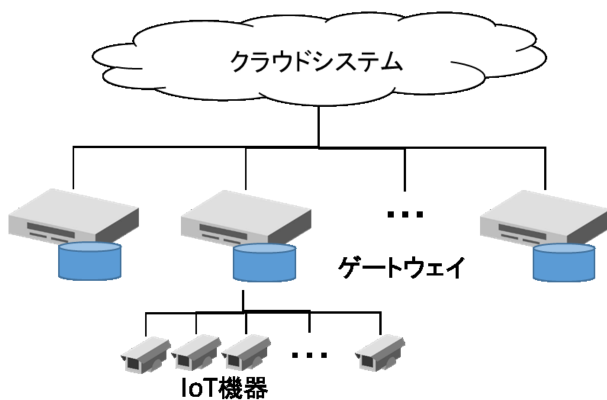

図1　システム構成
Figure 1　The system architecture.

このような背景から, 我々はForeign Data Wrapper（FDW）という機能があるPostgreSQL[1]をクラウド上に配置し，ゲートウェイ上にある多数のデータソースからデータを収集しようと考えている（図2）．FDWとは，外部データソースであるMySQLやcsvファイルなどの様々な種類のデータにアクセスするための仕組みである．しかし，アクセスする外部データソースが多いと，データの収集に時間を要する．

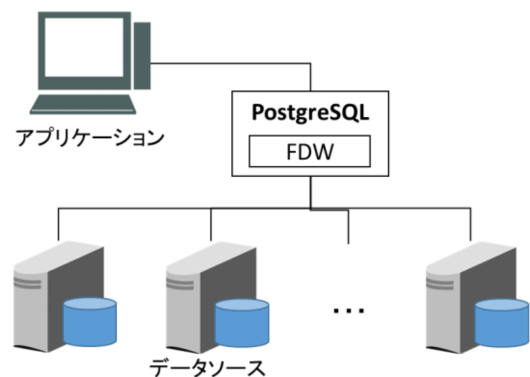

図2　PostgreSQLとデータソースの関係
Figure 2　The relationship between PostgreSQL and Datasources.

IoTの分野では，分散データベースのように複数のデータソース間でデータの構造（スキーマ）が同一であることが多い．この特徴を利用して，並列スキャンにより高速に検索する機能を開発した．本機能ではユーザがPostgreSQLに対して問い合わせた検索クエリを，複数のデータソースに対して並列に実行させてデータを収集する．

また，大量のデータを扱う場合，時間をかけて正確なデータを表示するよりも，時間をかけずにデータの概要を表

†1 株式会社東芝
TOSHIBA CORPORATION

ⓒ 2017 Information Processing Society of Japan

示することが望まれることがある．この用途のために，即時結果取得機能も併せて開発した．この機能は，データソースからの収集が未完了でも，ユーザによる結果取得要求を受け付けた時点で収集済みのデータのみを使用して，結果を算出してアプリケーションに返答する．

本論文では，2 章で並列スキャン機能，3 章で即時結果取得機能についてそれぞれ設計を説明する．4 章ではそれぞれの機能について動作確認として実施した実験について説明し，最後に 5 章でまとめる．

2. 並列スキャン機能の設計

この機能は，PostgreSQL は他のデータソースのテーブルを仮想的にひとつのテーブルとみなして，複数データソースに対して並列に問い合わせる．

例えば，2 つのデータソース A と B があり，それぞれテーブル table1 を持つとする．通常，ユーザは PostgreSQL を介してこれらのデータを収集したい場合，「SELECT * FROM A.table1 UNION ALL SELECT * FROM B.table1」を実行することになる，これでは，それぞれのデータソースへの問い合わせと結果フェッチ処理がシーケンシャルになり問題である．

このシナリオを高速に行うために，ユーザは「SELECT * FROM table1」として仮想的なテーブル table1 を用いて問い合わせられるようにした．本機能は複数データソース上にあるテーブルの検索を並列に実行する機能であり，既存のパラレルクエリ機能[2]とは異なるものである．パラレルクエリ機能は，自 PostgreSQL 上の一つのテーブルに対する検索を並列に行う機能で，複数テーブル間での検索は並列化されない．以降の節では，仮想テーブルを複数の実テーブルに展開して並列にアクセスする仕組みの実現方法について述べる．

なお，PostgreSQL では VIEW を用いて複数のデータソースを仮想的なテーブルにまとめて問い合わせることができる，しかし，UNION ALL を用いて同様のことを実現することが可能であるが，UNION ALL を用いて実現することになるため，各データソースへのアクセスは並列処理されない．

2.1 モジュール構成

FDW は，データソースと PostgreSQL エンジンとを仲介するモジュールである．FDW では実装すべき API ルーチンの仕様が定められていて，それに基づいて任意のデータソース向けに実装することで，連携可能なデータソースの種類を増やすことができる[3].

今回，仮想テーブルを複数の実テーブルに変換して複数のデータソースにアクセスする FDW を新しく開発した．以降では，従来の FDW をデータソース FDW，今回開発した FDW を仮想化 FDW と呼ぶことにする．図 3 に示すように，仮想化 FDW がデータソースに問い合わせるにはデータソース FDW を使用する．

仮想化 FDW は主に次の 2 つの機能を持つ．

- エンジンからの要求を複数のデータソース要求に変換し，並列にデータソースに問い合わせてデータを収集する．
- データソースからの収集データをひとつにまとめてエンジンに返却する．

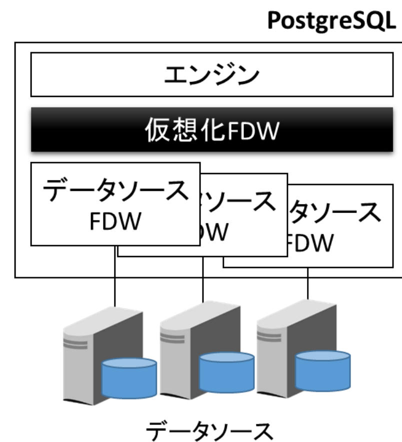

図 3　モジュール構成

Figure 3　The modular construction.

2.2 複数データソースへの問い合わせ

メインスレッドは，データソースへの問い合わせを開始する仮想化 FDW の BeginForeignScan において，データソースの数に応じてスレッドを生成する（図 4）．以降ではこれを FDW スレッドと呼ぶ．FDW スレッドは，データソース FDW を用いてデータソースにアクセスする．PostgreSQL 上で複数のスレッドを動作させるには，メモリアクセスが衝突しないようにメモリコンテキストを各スレッド向けに用意する必要がある．

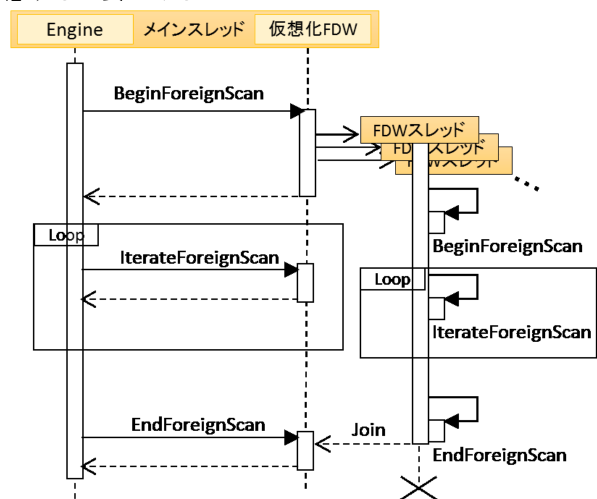

図 4　スレッド化による並列問い合わせ

Figure 4　Parallel access by multiple threads.

今回，各データソースは同一の機能を有する前提とした．つまり，仮想化 FDW は各データソースに対して同一内容の問い合わせを実施することができる．仮想化 FDW はエンジンからひとつの問い合わせ要求を受け付けると，生成した複数のスキャンスレッドに対して問い合わせ要求を与える．その後，各スレッドはデータソース FDW を用いてデータソースにアクセスして，その結果を収集する．

2.3 データソースからの収集データのマージ

FDW スレッドがデータソースから取得したデータは，仮想化 FDW を経由してエンジンに渡す必要がある．

エンジンが問い合わせ結果を取り込む仕組みは，IterateForeignScan という FDW ルーチンを繰り返し実行することで，FDW からレコードを 1 行ずつ取り込むようになっている．具体的には，まずデータソースへの問い合わせ開始として BeginForeignScan を実行し，次に IterateForeignScan を実行して NULL が返されるまで繰り返す．最後に EndForeignScan を呼び出す．IterateForeignScan の返り値としてスロットという型のデータに 1 行分のレコード情報を格納することで，エンジンと FDW とのデータを受け渡す．仮想化 FDW がエンジンにレコード情報を渡す方法は，上記の仕組みで実施する．

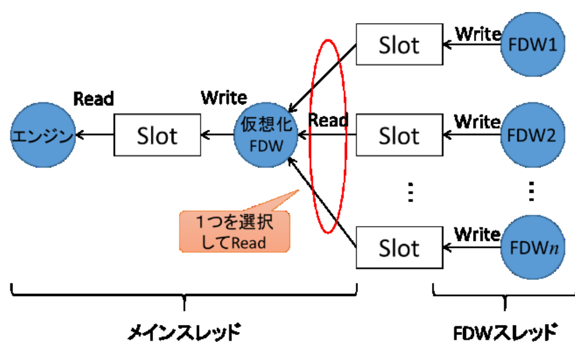

図 5 FDW とエンジン間のデータの受け渡し
Figure 5 The data flow between FDWs and engine.

一方で，仮想化 FDW とデータソース FDW 間でレコード情報を受け渡しするには，それらのモジュール間で共有する領域を使用した．そのスロットはスレッドの数だけ用意した（図 5）．仮想化 FDW がエンジンにはいずれかひとつのスロットを選択し，エンジンに返却することになる．

関数の返り値による受け渡しをしない理由としては，メインスレッドと FDW スレッドを独立して処理させるためである．つまり，エンジンが仮想化 FDW の IterateForeignScan を呼び出す前に，FDW スレッドがデータソース FDW の IterateForeignScan を実行して事前にデータをスロット型に変換しておくようにする．

3. 即時結果取得機能の設計

PQgetCurrentResult というクライアント API を開発した．この機能は，非同期問い合わせ PQSendQuery によって実行したクエリに対して，即時結果取得するものである．非同期問い合わせとは，クエリの実行の完了を待たずに制御をアプリケーションに返す機能である．

使用方法は PQgetResult と同様である．挙動の違いは，PQgetResult は PostgreSQL サーバから最終結果がクライアントに到着するまで待つ．一方で，今回開発した PQgetCurrentResult は，サーバに結果取得要求を出し，それを受け取ったサーバはクエリ処理を中止し，その時点で生成できていた最終結果をクライアントに返す．サーバが処理中止を確認するタイミングについては 3.1 で説明する．

本機能は 2 章で説明した並列スキャン機能の下で動作させる．本機能により，複数のデータソースからデータを収集するケースにおいて，データソースの処理性能やネットワーク性能にばらつきがあっても，低性能なデータソースに引きずられることなく，ユーザが必要なときに結果を取得できるようになる．

3.1 クエリ処理の打ち切りと最終結果生成

処理を中止するか継続するかの判定は，随所で実施することが望ましい. サーバ上で繰り返し実行されるところに，判定ポイントを設けた．サーバはクライアントから中止要求を受け付けたら中止フラグを立てて，判定ポイントでそのフラグを確認する．以降では，判定ポイントの設置場所について説明する．

一つ目に，仮想化 FDW の IterateForeignScan の冒頭に判定処理を導入した（図 6）. IterateForeignScan はデータソースから取得したデータの件数回繰り返し実行される．処理の中止タイミングの場合，すべてのデータがエンジンに取り込まれていなくても，IterateForeignScan が NULL を返すことで，エンジンにデータソースからのデータ取り込み終了とする．

二つ目として，並列スキャン機能との併用のために，仮想化 FDW 内の FDW スレッドがデータソース FDW の IterateForeignScan を繰り返し実行するループ内に判定処理を導入した. FDW スレッドはエンジンのクエリ処理とは独立して動作していて，それを停止させるためである．中止タイミングと判定した場合そのループを抜けることで，仮想化 FDW に渡すデータソースのデータを止める．

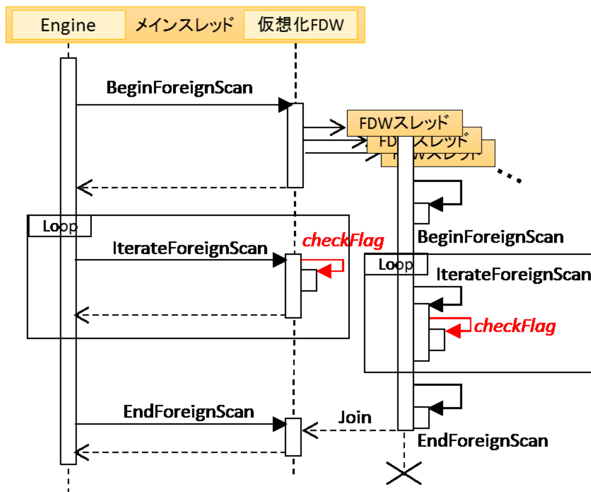

図 6　中止フラグの判定箇所

Figure 6　The check point of stop flag.

3.2　即時結果取得 API の仕組み

先記のとおり，本機能は非同期問い合わせ後に，即時結果取得要求としてサーバに新たな要求を発行することになる．サーバがクエリ処理中に，クライアントから他の要求を受け付ける方法については，文献[4]を参照されたい．この文献で実現しているクエリの進捗度取得要求の送受信の仕組みをそのまま利用した．つまり，クエリ処理開始時に別のスレッドを生成し，そのスレッドがクライアントからの要求に答える．その要求種別として新たに即時結果取得要求'R'を定義した．

クライアント側の動きとしては，まず要求種別'R'のメッセージをサーバに送信する．サーバ側は結果を生成してクライアントに送信するので，PQgetResult と同様の仕組みでそれを受信する．これを図示すると図 7 のようになる．

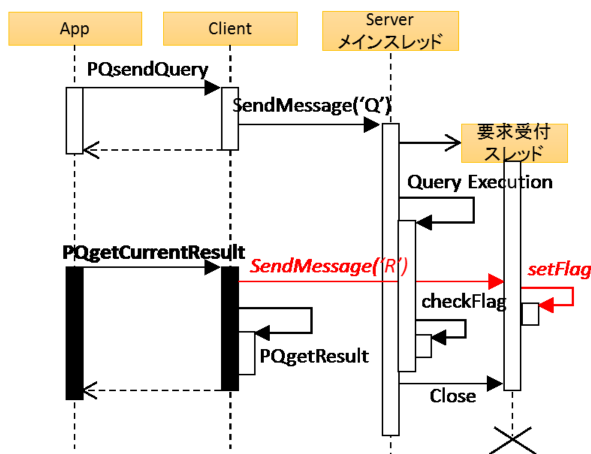

図 7　クライアント上での API のシーケンス

Figure 7　The sequence of API in client.

4.　実験

機能の動作を確認するために実験を行った．2 つの機能それぞれについて実験内容を説明する．環境は両実験とも表 2 の通りである．

表 1　実験環境

Table 1　Experiment environment.

CPU	Intel Core i7-4800MQ 2.70GHz
RAM	16GB
SSD	256MB
OS	Ubuntu 16.04 (VirtualBox 5.1.18)

4.1　並列スキャン機能の実験内容

並列スキャン機能の動作を確認するために，複数台のデータソースからデータを収集するクエリ"SELECT * FROM table"の処理時間を測定した．比較対象としては，UNION ALL でそれぞれのデータソースへの検索を集合させる処理とした．

データソースは docker を用いて仮想マシン上に 1 台から 4 台まで用意した．データソースの DBMS として PostgreSQL を使用し，それぞれ 5 万件のレコードを格納している．

4.2　並列スキャン機能の実験結果と考察

図 8 に実行結果を示す．左側の棒グラフは従来手法（UNION ALL），右側は提案手法（並列スキャン）の処理時間を表している．値は 5 回の試行の平均値である．

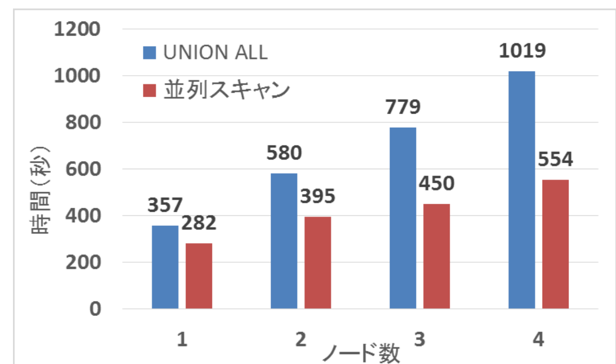

図 8　処理時間（秒）

Figure 8　Processing time in second.

図のように，台数の増加に連れて高速化できることを確認した． 1 台の場合でも，本機能を使用すると高速になっていた．これは，メインスレッドによるエンジンの処理と FDW スレッドによるデータ取り込み処理が並列に実行されているからだと考えている．

4.3　即時結果取得機能の実験内容

PQgetCurrentResult の動作を確認するために，処理時間と取得結果件数を確認し，PQgetResult と比較した．処理時間は，PQsendQuery の実行前を開始として，PQgetResult あるいは PQgetCurrentResult の実行後を終了とした．PQgetCurrentResult を呼び出すタイミングは，開始から 10 秒後，30 秒後および 60 秒後の 3 パターンで実施した．実

ⓒ 2017 Information Processing Society of Japan

験に使用したデータとシナリオとしては，1 台のデータソースに 1,048,576（=2^20）件のデータが格納されている．それに対して全取得するクエリを発行して結果を取得した．

4.4 即時結果取得機能の実験結果と考察

表 2に実行結果を示す. 値は5回の試行の平均値である．16MB/208KB は送受信バッファの設定値である．詳細は後で説明する．まずは 16MB の列を参照されたい．

PQgetRresult の処理時間を見てわかるように，10 秒後および 30 秒後はクエリ処理中である．その時に実行した PQgetCurrentResult（16MB の列）では，すぐに処理が終了できて，途中までの結果が取得できたことが確認できた．

表 2 処理時間と取得データ数

Table 2 Time and the record count.

API	時間（秒）	件数：16MB	件数：208KB
PQgetCurrentResult 10 秒後	10.03	244,447	16,516
PQgetCurrentResult 30 秒後	30.06	545,064	16,275
PQgetCurrentResult 60 秒後	60.11	985,367	16,516
PQgetResult	37.52	1,048,576	1,048,576

60 秒後はクエリ処理が終わっているケースで，その場合も取得結果行数が全件となることを確認した．表 2 において，60 秒後の件数が全件と一致しないのは，処理が完了していないことがまれにあり，それが平均値として現れたからである．なお，PQgetResult の場合でも 60 秒以上かかることがあった．

PQgetCurrentResult の結果件数は送受信バッファサイズに依存することが分かった．バッファサイズが 208KB（デフォルト値）の場合，表 2 の 208KB の列のような結果となった．つまり，十分時間が経過しているにも関わらず，一定のデータ数しか取得できなかった．

サーバの動きとして，結果を生成できたタイミングでクライアントに送信する．このとき，バッファサイズが一杯になるまでしか送信できない. PQgetResult を実行した場合，クライアントでデータを受信すると，サーバで処理が再開される．しかし，PQgetCurrentResult の場合，そこで処理が中断されてしまうため，いくら PQgetCurrentResult を実行するタイミングを遅くしても，取得できる件数がバッファサイズに依存してしまうようである．

このように送受信バッファのサイズに依存しないように途中結果を取得できるようにするには，さらなる改造が必要で，今後の課題として捉えている．

5. おわりに

PostgreSQL における外部データソースに対する検索を並列化する機能と，処理を中止して即時結果を取得できる機能を提案し，それが動作することを確認した．即時結果取得機能については，送受信バッファの設定値が小さいと期待通り動作しないことが分かった. その設定値に依らず，動作できるようにすることが今後の課題と考えている．

参考文献

[1] “PostgreSQL”. https://www.postgresql.org/, (参照 2017-08-03).
[2] “PostgreSQL のパラレルクエリ”. https://www.postgresql.jp/document/9.6/html/when-can-parallel-query-be-used.html, (参照 2017-07-22).
[3] “Foreign data wrappers“. https://wiki.postgresql.org/wiki/Foreign_data_wrappers, (参照 2017-08-03).
[4] 片山大河, 廣瀬繁雄, 嶋村誠, 金松基孝. PostgreSQL への進捗度取得機能の導入. 情報処理学会第 79 回全国大会, 2017.

ⓒ 2017 Information Processing Society of Japan