PostgreSQL の外部連携機能におけるレコード変換処理の高速化

柿村　直拓†　　立床　雅司†　　柴田　秀哉†

†三菱電機株式会社　情報技術総合研究所　〒247-0056 神奈川県鎌倉市大船５丁目１－１

E-mail:　†{ Kakimura.Naohiro@ak,　Tatedoko.Masashi@ap,　Shibata.Hideya@cb}.MitsubishiElectric.co.jp

あらまし　様々なデータソースを統一的に扱うために，PostgreSQL は Foreign Data Wrapper（FDW）と呼ばれる外部データ連携機能を備えている．FDW は PostgreSQL からのレコード要求に応じ，外部データソースからレコードを取得し，PostgreSQL の内部形式に変換する．この一連の処理において，レコード変換処理の比重が大きい場合，変換処理を並列化することで高速化が期待できるが，単一のデータソースに対する並列化の事例は知られていない．そこで本論文では，単一データソースに対するレコード変換処理の並列化機構を，商用データベース管理システム AQL 向けの FDW に対して実装し，評価により，変換処理の比重が大きい問合せが高速化されることを確認した．

キーワード　PostgreSQL，FDW，並列処理，データ変換

1. はじめに

IoT 関連技術が発展したことにより，大規模なデータを蓄積し，分析することがより身近になった．例えば，製造現場では設置されたセンサからデータを蓄積し，分析することで生産効率向上に努めている．これに伴い，データの取得，蓄積，参照方法は複雑化しており，データソースも多様化傾向にある．一般的にデータ蓄積に使われる RDBMS (Relational Database Management System) は，多くのユーザが同時に少量データを更新，検索するような OLTP (Online Transaction Processing) 処理を得意とする反面，少ないユーザがデータ全体を参照するような OLAP (Online Analytical Processing) 処理は不得手である．このように，用途に応じてデータベースを使い分ける必要があるが，異なるデータソースに対しデータを参照，分析するのは手間がかかるため，標準的な SQL でデータを統一的にアクセスできるインターフェースが求められる．

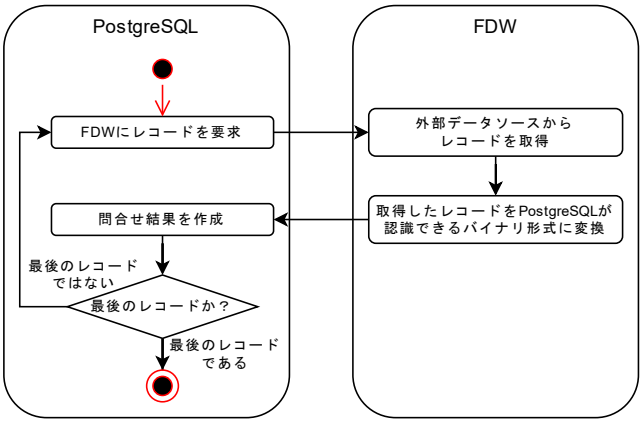

図　1 PostgreSQL における FDW を活用した外部データソースのレコード取得処理の流れ

OSS の RDBMS である PostgreSQL[1]は，標準的な SQL のインターフェースを持ち様々な分野で利用されている．PostgreSQL は OLTP 処理を実現するためにマルチプロセスで複数の問合せを受け付け，各問合せはシングルプロセスかつシングルスレッドで実行される．また，PostgreSQL は外部データソースからレコードを取得できる機能である Foreign Data Wrapper (FDW)[2]を備えている．

PostgreSQL が外部データソースにアクセスする問合せを受け付けた時，PostgreSQL は FDW にレコードを要求する（図 1）．FDW は外部データソースからレコードを取得して，PostgreSQL が処理できるバイナリ形式に変換して，PostgreSQL にレコードを渡す．その後，PostgreSQL は問合せ結果の作成などを直列に行う．そのため，FDW は変換処理完了後に再度 PostgreSQL からレコード要求を受けるまで，次のレコード取得を行うことができず待ち時間が発生する．特に，レコード変換に文字コード変換などの時間を要する処理を含む場合，この問題は顕著となる．この問題に対して複数データソースを用いた並列化の試みはあるが、単一の外部データソースに対する事例は知られていない．

そこで本論文では，単一の外部データソースに対するデータ変換を効率化するため，データ変換処理を並列化する方式を提案する．また，提案方式を商用データベース管理システム AQL[3]向けの FDW に対して実装し，評価を実施した．本論文では，2 章で関連研究，3 章で提案方式，4 章で提案方式の評価について説明し，5 章で総括を行う．

2. 関連研究

2.1. FDW の非同期実行

PostgreSQL サーバを外部データソースとして扱う FDW モジュール postgres_fdw には，非同期実行機能が

ある[2]．この機能では，シャーディング等で外部データソースを分割可能な場合に外部データソースアクセスを非同期化し，外部データソースからのレコード取得を並列化することで，処理を高速化している．

2.2. パラレルクエリ

PostgreSQL には，パラレルクエリ[4]と呼ばれるマルチプロセスを活用した実行計画の作成・実行機能がある．この機能では，PostgreSQL のテーブルに対して，レコード取得の並列処理が可能である．

2.3. 複数データソースの並列レコード取得

文献[5]では，PostgreSQL から異なる種類の複数外部データソースに連携してアクセス可能とするための FDW が提案されている．ここでは，各データソースに対してスレッドを立ち上げ，レコード取得・変換処理を並列化している．また，各スレッドは PostgreSQL とは独立に動作し，PostgreSQL からレコードが要求される前に FDW で事前にレコード取得・変換処理を実施することで高速化している．

2.4. 課題

本章で説明した既存の方式では，以下の 2 点の性質を持つ単一データソースを PostgreSQL と連携する場合の高速化手段として不十分である．

1. レコード取得処理が並列化できない．
2. レコード変換処理の時間が，PostgreSQL 内部の処理やレコード取得時間と比較し長い．

postgres_fdw による非同期実行は，単一の外部データソースに対しては直列的な処理となる．パラレルクエリは，外部データソースの性質 1 により適用できない．文献[5]の方法では，PostgreSQL からのレコード要求前にレコード取得・変換処理を開始するため，一定の高速化効果は見込めるが，単一の外部データソースに対してはレコード変換処理が並列化されないため，性質 2 によりボトルネックが解消されない問題がある．

3. 提案方式

本章では，単一の外部データソースであっても，PostgreSQL のレコード要求に対しての待ち時間を削減可能な方式を提案する．提案方式では，レコード変換処理を並列化するため，レコード変換処理とレコード取得処理を非同期に実行する．

図 1 において，PostgreSQL が FDW にレコードを要求し，受け取ったレコードを使用して問合せ結果を作成するまでの一連の処理，FDW のレコード取得処理，FDW のレコード変換処理の 3 つの処理は非同期に実行可能である．非同期処理はマルチスレッドにより実現する．各処理を実行するスレッドを，以下では順にメインスレッド，取得スレッド，変換スレッドと呼ぶ．

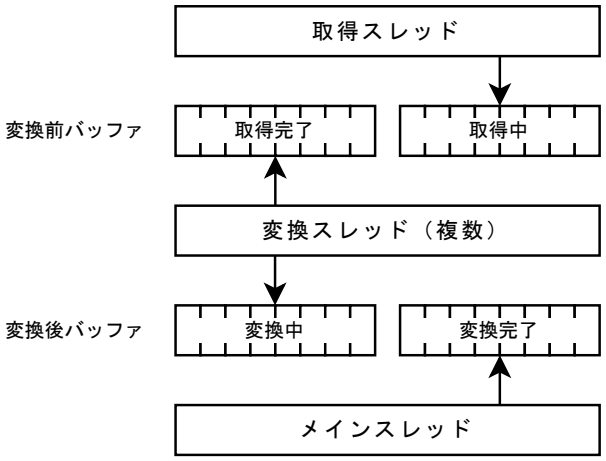

図 2 各スレッドとバッファの関係

提案方式では，変換処理におけるボトルネックを解消するために，変換スレッドを複数作成し並列化する．変換処理の並列化に伴い，複数レコードを同時に変換することになるため，取得処理では，複数レコードを一括取得する必要がある．スレッド間でデータをやり取りするために，取得されたレコードは変換前バッファに一時保管され，変換されたレコードは変換後バッファに保管される（図 2）．各スレッドを切れ目なく動作させるため，各バッファを 2 つずつ用意し，一方を書込み用，他方を読込み用とする．書込み用と読込み用は交互に役割を入れ替える．

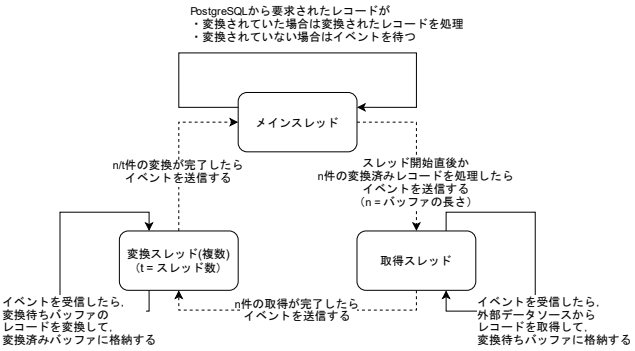

図 3 スレッド間のイベント完了通知

各処理は非同期に実行可能であるが，各スレッドは互いの処理結果に依存している．例えば，変換スレッドは取得スレッドがレコード取得を完了しなければ処理を開始できず，メインスレッドは変換スレッドがレコード変換を完了するまで問合せ結果の作成ができない．そこで，スレッド間の処理完了通知を実現するため，提案方式ではイベント駆動型アーキテクチャで各スレッドを連携させる（図 3）．各スレッドはイベントを受信したら処理を開始し，処理が完了したらイベン

トを送信して待機する．この方式では，イベントの送受信や待機開始･解除のオーバーヘッドがかかるため，一括取得するレコード数は変換スレッド数よりも多くなければ効果が見込めない．

4. 評価

本章では提案方式の有効性を評価するため，非同期化・並列化を実施していない素朴な FDW を比較方式として，問合せ実行時間を比較する．

4.1. 評価環境

評価環境を表 1 に記載する．なお，PostgreSQL と外部データソースは同じサーバに存在するとする．外部データソースとして，商用のデータ分析フレームワーク AnalyticMart[6]で使用されているデータベース管理システム AQL を対象とする．

表 1 評価環境

CPU	Intel Corei5-10500 6 コア/12 スレッド/3.1GHz/
OS	Windows Server 2016 standard
RAM	16GB (8GBx2) DDR4 DIMM 2666MT/s
SSD	512GB SSD (M.2 NVMe PCIe TLC)
PostgreSQL	11.12 版，UTF-8
外部データソース	商用データベース AQL

4.2. 評価データ

評価に使用するデータは，外部データソース保管された 100 万件のレコードデータである．外部データのスキーマを図 4 に示す．

```
CREATE FOREIGN TABLE foreign_table (
      c_integer           INTEGER,
      c_bigint            BIGINT,
      c_numeric1809       NUMERIC(18, 9),
      c_char0004          CHAR(4),
      c_char0032          CHAR(32),
      c_char0128          CHAR(128),
      c_varchar0004       VARCHAR(4),
      c_varchar0032       VARCHAR(32),
      c_varchar0128       VARCHAR(128),
      c_date              DATE,
      c_timestamp6        TIMESTAMP(6)
) SERVER xx_FDW;
```

図 4 評価用データのスキーマ

各データ型の変換方法について説明する．これらの方法はレコード変換処理時間に大きく影響する．INTEGER, BIGINT, CHAR, VARCHAR 型については，特別な形式変換を行わず，外部データソースから取得したデータをそのままコピーする．但し，対象の外部データソースにおける文字列型のエンコーディングは Shift_JIS 又は UTF-8 が選択可能であるが，Shift_JIS が選択されていた場合は，UTF-8 への変換処理を追加で行う．NUMERIC，DATE，TIMESTAMP 型は文字列として外部データソースから出力され，それらの文字列を解析して，PostgreSQL で扱えるバイナリ形式に変換する．

4.3. 評価方法

評価では，比較方式と提案方式との問合せ実行時間を，データ型による違いとデータ変換処理の並列度による違いの観点から比較する．データ型による違いでは，図 4 のテーブルにおける各データ型の問合せ実行時間を提案方式（変換スレッド数＝4）と比較方式で比較する．並列度による違いでは，変換処理が短いデータ型（INTEGER 型）と長いデータ型（Shift_JIS の VARCHAR 型）で，並列度（変換スレッド数＝1, 2, 4）を変えて比較する．両評価において，変換前，変換後バッファの長さはそれぞれ 1,000 レコード分とする．

測定は PostgreSQL のクライアントツール psql の ¥timing コマンドを使用し，null 出力する．また，問合せは外部データソースから全件参照するようなクエリを使用する（図 5）．このクエリの count 関数は，問合せ結果として，大量のデータ書出しが発生しないよう意図したものである．また，count 関数内の null 確認は,外部データソースで実行できない処理であるため，プッシュダウンができず，全レコードを取得しなければならない処理となっている．このように外部データソースからのレコードを全件取得しつつ，問合せ結果の書き出し時間を抑制する．

```
SELECT count(c_integer is null) FROM foreign_table;
```

図 5 評価に使用するクエリ例

4.4. 評価結果

図 6 に各データ型を取得した際の，比較方式に対する提案方式の問合せ実行速度を相対速度として示す．結果から INTEGER 型, BIGINT 型以外は高速化されていることが確認できる．特に，CHAR 型(Shift_JIS), VARCHAR 型(Shift_JIS), TIMESTAMP 型の変換において，2 倍以上の高速化効果が確認できる．この結果から文字コード変換や文字列を解析するような変換を行う場合は，高速化されることが確認できた．一方，INTEGER 型，BIGINT 型のように変換処理の比重が小さい（4, 8 バイトのデータのコピー）場合は，並列化のオーバーヘッドが大きく，高速化されないことを確認した．

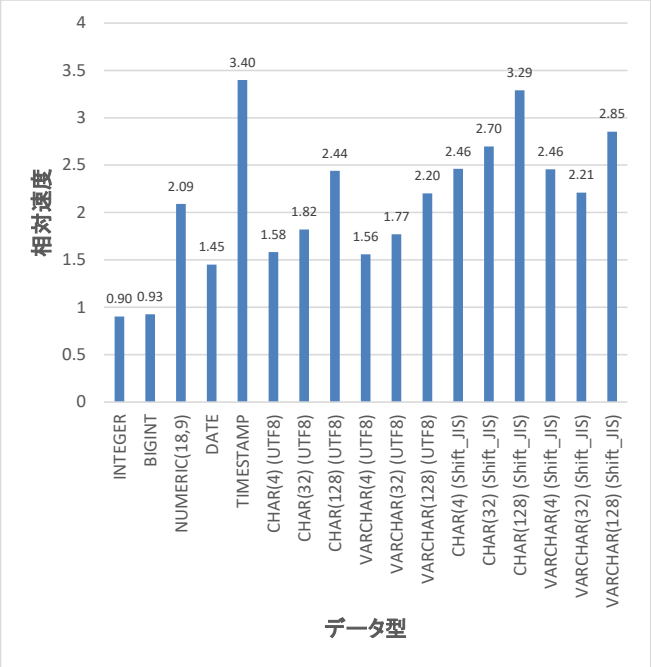

図 6 比較方式に対する提案方式の相対速度
（データ型毎，並列度 4）

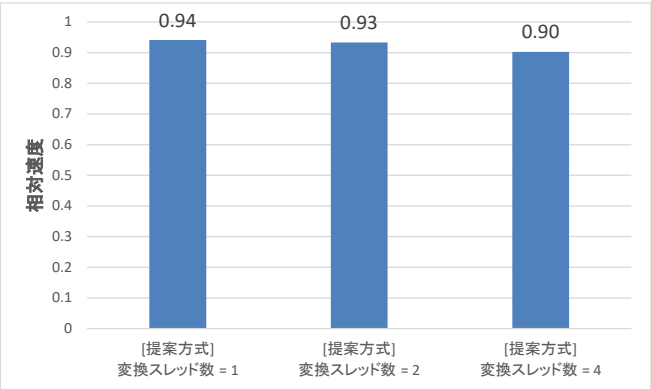

図 7 比較方式に対する提案方式の相対速度
（並列度毎，INTEGER 型参照時）

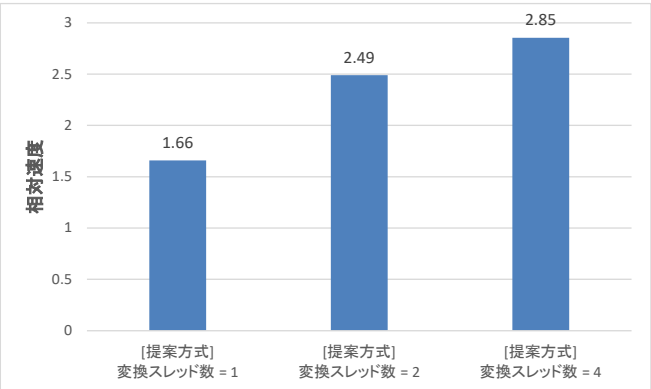

図 8 比較方式に対する提案方式の相対速度
（並列度毎，VARCHAR(128)型 (Shift_JIS)参照時）

図 7 と図 8 に，並列度を変更した時の問合せ実行時間の相対速度を示す．図 7 では，変換処理の比重が小さい INTEGER 型，図 8 では，変換処理の比重が大きい文字コード変換ありの VARCHAR(128)型 (Shift_JIS)の実行時間を示す．INTEGER 型に関しては，並列度が高くなるにつれて提案方式が遅くなっている．これは，前の評価と同じく並列化のオーバーヘッドに起因するものと考えられる．VARCHAR 型に関しては，並列度が高くなるほど高速化されていることが確認できる．但し，並列度が高くなるほどスレッドあたりの高速化効果は減少している．これは，レコード変換処理時間が並列化によりレコード取得処理時間やメインスレッドの処理時間と近い時間になったためであると考えられる．

5. おわりに

本論文では，FDW を介した PostgreSQL と外部データソースとの連携において，外部データソースからのレコード取得処理とレコード変換処理を非同期に動作させ，更に変換処理を並列化する方式を提案し，評価を実施した．その結果，レコード変換処理の比重が大きい問合せに関しては，並列化による高速化の効果を確認した．一方，レコード変換処理の比重が小さいものに関しては，並列化のオーバーヘッドが顕在化し，高速化効果は得られず，若干処理時間が増大する結果となった．双方のケースにおいて，内部で最適な処理方法に自動的に切り替えられるようにすることが，今後の課題である．

参 考 文 献

[1] PostgreSQL: The world's most advanced open source database
https://www.postgresql.org/

[2] F.38. postgres_fdw (postgresql.jp)
https://www.postgresql.jp/document/15/html/postgres-fdw.html

[3] 佐藤重雄，塚本純子，清水英弘，石井篤，藤原聡子，“多次元明細データベース DIAPRISM/AQL の概要”，情報処理学会全国大会講演論文集 55.3(1997): 501-502

[4] PostgreSQL: Documentation: 16: Chapter 15. Parallel Query
https://www.postgresql.org/docs/current/parallel-query.html

[5] 片山大河，嶋村誠，金松基孝，“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”，電子情報通信学会技術研究報告 117.212 (2017): 75-79.

[6] データ分析フレームワーク AnalyticMart | プラットフォームソリューション | サービス・製品 | 三菱電機インフォメーションネットワーク株式会社（MIND）
https://www.mind.co.jp/service/idc_platform/platform_products/analyticmart/index.html