3B-03

PostgreSQL への進捗度取得機能の導入

片山 大河† 廣瀬 繁雄† 嶋村 誠† 金松 基孝†
株式会社 東芝 インダストリアル ICT ソリューション社 IoT テクノロジーセンター†

1. はじめに

PostgreSQL[1]はエンタープライズ向けのデータベース管理システムとして広く適用されており、近年ではクラウド上の大規模システムにも利用されている。大量のデータが蓄積されていると、ひとつのクエリ処理時間が数十分を要することもある。ユーザはクエリ処理の完了時間を見積もることは難しい。そこで参照クエリの進捗状況を取得する機能を開発した。

アプリケーションは、サーバにクエリ実行要求を出した後に今回新しく開発した API を呼び出すと、進捗度がパーセンテージで取得できる。

2. PostgreSQL のアーキテクチャ

PostgreSQL は、クライアントの接続ごとに子プロセスを生成して 1 スレッドでクエリを処理するモデルである。そのプロセスはクライアントからのメッセージに応じて様々な処理を行う。

クエリの実行方法の中に PQsendQuery という非同期コマンドがある。図 1 のように、この API はクエリ実行要求をサーバに送ると、処理の完了を待たずにアプリに制御を返す。サーバのクエリ処理中に、アプリは他の処理を行うことができる。

この間、クライアントはサーバから送信される完了メッセージ’C’の受信待ちになる。サーバは処理中のコマンドを完了して、完了メッセージをクライアントへ返信するまで、クライアントから次のコマンドを受け付けることはできない。

3. 進捗度取得機能の設計

非同期コマンドに対する進捗度を取得する API を開発した。第一引数には非同期コマンドで使用した接続オブジェクトを与える。

```
int PQgetProgress(PGconn *conn, double *progress);
```

サーバが非同期コマンドの処理中でもメッセージを受付可能にするために、以下の点を改造した。

・進捗度取得命令の追加
・進捗度通信機能の導入
・要求受付スレッドの作成

Development of Progress Acquisition Feature on PostgreSQL
†Taiga Katayama, Shigeo Hirose, Makoto Shimamura, Mototaka Kanematsu
†Toshiba Corporation Industrial ICT Solutions Company

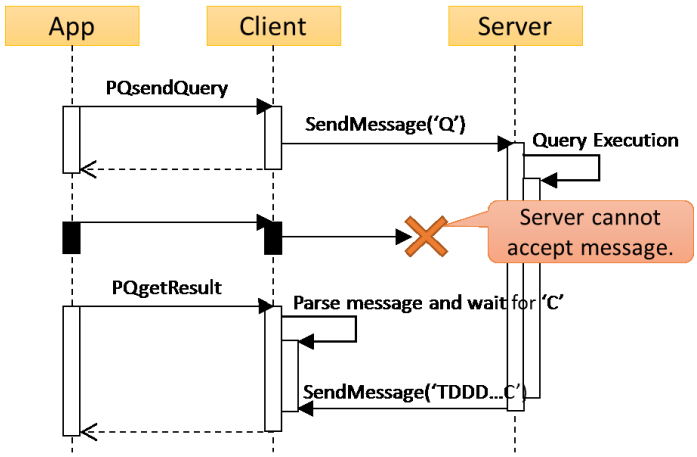

図 1 非同期コマンド実行と結果取得

・進捗度設定機能の導入

3.1. 進捗度取得命令の追加

クライアントからサーバに要求を出すとき、char 型のメッセージタイプにより要求内容を識別する。例えば、’Q’は非同期コマンドの実行要求、’P’はプリペアド文の作成要求、’E’はプリペアド文の実行要求などである。新たなメッセージタイプとして進捗度取得要求’G’を導入する。

3.2. 進捗度通信機能の導入

クライアントがサーバから進捗度を受信するための仕組みを導入する。クライアントーサーバ間の通信は、進捗度送受信用に新たに作成せず従来のものと共有する方針としたため、結果セットの受信プロトコルを拡張する。進捗度取得要求を出しても、タイミングによってはサーバから結果セットが返却される可能性がある。そのためにはクライアントは受信データが結果セットなのか進捗度なのかを区別する必要がある。その仕組みをこれから説明する。

PQgetResult がサーバからデータを受信し結果セットを構築する。その内部では、char 型のメッセージタイプによりデータ内容を識別する。例えば、’T’はフィールド情報、’D’はタプルのデータ、’C’は完了識別子などである。新たなメッセージタイプとして進捗度データを表す’P’を導入する。

3.3. 要求受付スレッドの作成

サーバはクライアントからの要求に応じて処理を行うスレッドが 1 つしかないため、その処理が完了してからでないと次の要求を実行でき

ない。この問題を解決するために、非同期コマンド実行要求を受け付けた際に、進捗度取得要求受け付け用の新たなスレッド（以降、進捗返却スレッドと呼ぶ）を作成することとした。

複数スレッドで動作させるためには、グローバル変数で管理されるモジュールがあるため注意が必要である。具体的には、PostgreSQL サーバ内部でメモリを管理するモジュールであるメモリコンテキストと通信のためのソケットハンドルを共有し、それらを排他制御して利用する。

進捗返却スレッドは、進捗度取得要求のみを受け付ける。受信したメッセージを解析し、進捗度取得要求だった場合、後述する（3.4 節）進捗度計算結果の値を読み込み、クライアントに送信する。進捗度取得要求でなかった場合、従来のシーケンスとみなす。具体的には、クエリ処理の完了を待ち、受信バッファをクエリ処理スレッドに渡す。

3.4. 進捗度設定機能の導入

サーバが進捗度の値を管理する方法について説明する。進捗度の値は、クエリの実行状態を管理する EState 構造体に新たなメンバー変数を作成して格納することとした。クエリ処理スレッドが進捗状況に応じての変数値を更新し、進捗返却スレッドが変数を読み込む。

今回、ユーザが進捗度を取得できる仕組みの構築に重点を置いたため、進捗度数値そのものの正確性は追求できていない。

4. 実験

進捗度取得機能が動作することを確認する実験を行った。実験に用いたシステム構成と進捗度の定義は以下のとおりである。

我々は、あるマシンの DB 上に大量のデータが

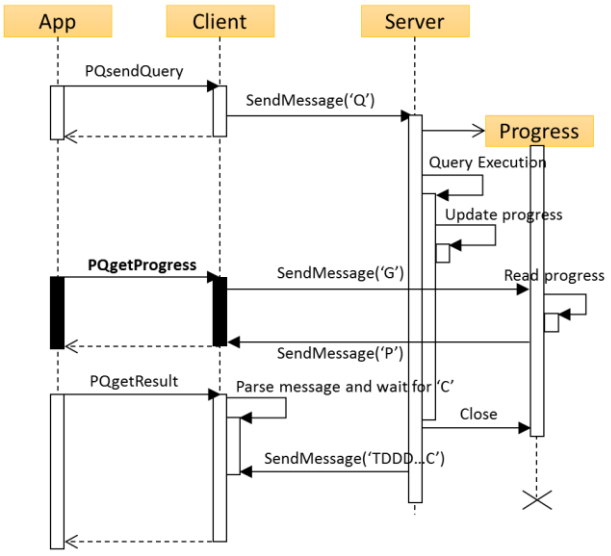

図 2 進捗度取得シーケンス

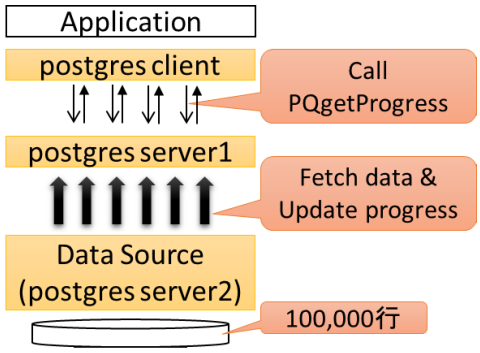

図 3 実験のシステム構成

蓄積されていて、PostgreSQL を介してそれらのデータを検索しようと考えている。これを実現するために、Foreign Data Wrapper（FDW）機能の利用を想定している。FDW とは、PostgreSQL 管理外のデータにアクセスするための仕組みである。以降では、データを格納するシステムをデータソースと呼ぶ。

実験では 100,000 行のレコードを格納した PostgreSQL をデータソースとして、進捗度機能を組み込んだ PostgreSQL が接続する構成とした（図 3）。進捗度は FDW がデータソースの問い合わせ結果を取得できた行数に応じて更新するようにして、進捗度取得機能を確認した。例えば、PostgreSQL が 1,000 行データを取得できたら進捗度は 1%である。アプリは、一定間隔で PQgetProgress を実行して進捗度を取得した。

```
PQsendQuery(db, "SELECT * FROM ftbl");
while (progress < 100) {
  PQgetProgress(db, &progress);
  printf("%.1f, ", progress);
  usleep(interval); /* 10000 or 30000 */
}
```

以下に示す実験結果のとおり、アプリから進捗度が取得できることが確認できた。

10 ミリ秒間隔の場合：

```
0.0, 0.5, 0.9, 1.6, 2.2, 2.8, 3.4, 4.0, 4.6, 5.2, （中略）
82.5, 84.8, 86.9, 89.3, 92.5, 94.7, 96.5, 98.4, 100.00
```

30 ミリ秒間隔の場合：

```
0.0, 1.2, 2.5, 4.0, 5.1, 7.6, 9.1, 10.5, 12.9, （中略）
95.8, 96.4, 96.9, 97.4, 98.0, 98.6, 99.2, 99.7, 100.00
```

5. おわりに

PostgreSQL の進捗度取得機能を開発した。サーバのクエリ処理スレッドの処理中に進捗度取得要求を受け付けられる仕組みを提案し、動作することを確認した。進捗度の値そのものの算出方法と急激な進捗度上昇については、今後の課題である。

参考文献

[1] PostgreSQL, https://www.postgresql.org/