D-008

連合学習がデータ転送の無駄なく実現できるデータ仮想化エンジン PGSpider の分散データ処理機能

Distributed data processing feature enabling to realize Federated Learning with efficient data transfer on data virtualization engine PGSpider

片山 大河† 廣瀬 繁雄† 熊谷 宏樹†
Taiga Katayama Shigeo Hirose Hiroki Kumagai

1. はじめに

様々な場所でデータが生成されるようになり、そのデータの活用技術の重要性が高まっている。この分散データの活用の課題として、まずデータを管理するシステムの異種性への対応がある。データはその特性に合わせて適したシステムで管理されるため、異なるシステム上にデータが格納されることがある。これらのデータの取得を容易にする技術としてデータ仮想化がある。データを格納するノード（以降データソースと呼ぶ）とデータを利用するプログラムとの間にデータ仮想化エンジンを配置し、このエンジンが異種性を吸収する。これにより、統一インターフェイスで異種システムに容易にアクセス可能になる。

別の課題としては、データ移動時の転送量削減やプライバシーの考慮がある。データソースから処理を行うノードへ大量の生データの転送を避けることが望ましい。この問題を解決するために、データ仮想化エンジンにおける分散ストアドファンクション機能を提案する。データ処理をデータソース上で行い、集約処理をデータ仮想化エンジンで行う。これにより生データを送受信せずに必要最小限のデータ通信で済むようになる。

今回、データ仮想化エンジン PGSpider™[1][2]（PGSpider は東芝の登録商標です）にこの機能を実現し、連合学習を適用例として実現性および効果を確認した。第 2 章では提案機能実現のための基礎技術を紹介し、第 3 章で提案機能および第 4 章でその効果を説明し、第 5 章でまとめる。

2. 提案手法実現のための基礎技術

2.1 データ仮想化エンジン PGSpider

データ仮想化エンジン PGSpider はオープンソースで、リレーショナルデータベース管理システム（RDBMS）である PostgreSQL®[3]（PostgreSQL は PostgreSQL の登録商標または商標です）をベースに作られている。PostgreSQL には外部データラッパー（ForeignDataWrapper：FDW）という機能があり、PostgreSQL 外のシステムと連携できる。データソースの異種性はこの機能が吸収するため、種類の異なるデータソースが複数あっても、それらのデータを横断アクセスしたいプログラムは、PGSpider を介すことで共通の I/F でデータ取得が行える。これから PGSpider の特徴であるマルチテナント機能とプッシュダウン機能について説明する。

2.1.1 マルチテナント機能

この機能は、複数のデータソース上のテーブル群を仮想的にひとつのテーブルとみなせる機能である（図 1）。この仮想テーブルはマルチテナントテーブルと呼び、複数の

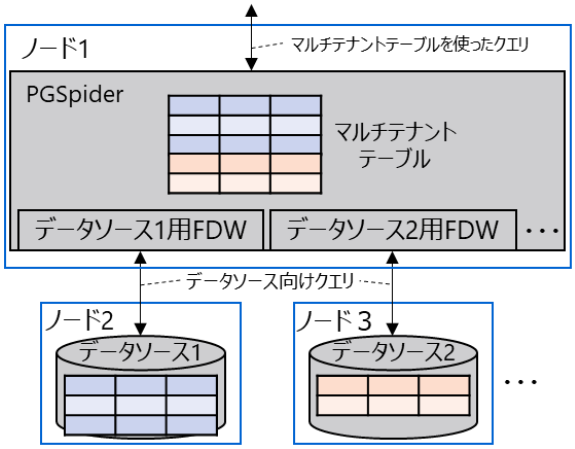

図 1 マルチテナントテーブル

テーブル上のレコードを UNION ALL で統合したテーブルに相当する動作を行う。PGSpider はこのマルチテナントテーブルに対するデータ取得を高速化していて、個々のテーブルへのアクセスを並列化している。また、ユーザがデータの所在地を把握できるように、各レコードに情報を付与する機能も備えている。

2.1.2 プッシュダウン機能

分散データ活用における転送量削減のアプローチとして、データ仮想化の分野ではプッシュダウンが挙げられる。プッシュダウンとは、データソースに処理を任せる機能である。例えば WHERE 条件式や集約関数などのプッシュダウンがある。WHERE 条件式をプッシュダウンすると、条件に一致しない不要なデータをデータ仮想化エンジンに転送しなくて済む。また、集約関数をプッシュダウンすると、データソース上で集約計算を行うため生データを仮想化エンジンに転送しなくて済む。

マルチテナントテーブルに対して（複数のテーブルをまたがって）集約関数を実行する場合、実行したい関数をそのままデータソースにプッシュダウンできるものとできないものがあり、PGSpider は関数に応じて特別な対応を行っている。例えば、データ件数を計算する count は、n 個の各データソース i でデータ件数$count_i$を計算し、PGSpider 上でそれらを合算（$\sum_{i=1}^{n} count_i$）する。一方で、平均値を計算する avg の場合、各データソース i 上で合計値sum_iとデータ数$count_i$を計算し、PGSpider 上で$\sum_{i=1}^{n} sum_i / \sum_{i=1}^{n} count_i$を計算するようにしている。

2.2 ストアドファンクション

PostgreSQL を含む RDBMS にはストアドファンクションという機能を備えることが多く、データ転送量削減のアプ

†株式会社東芝 TOSHIBA CORPORATION

ローチのひとつに挙げられる。PGSpider もこの機能を備える。この機能は、データベースに対する複数の手続き処理をひとつの関数としてまとめられる機能である。ユーザは任意に関数を定義しておき、SQL 文の中から呼び出して実行できる。各手続き間のデータのやり取りは RDBMS 内で完結するため、アプリケーションと RDBMS 間のデータ転送量の削除に繋がる。

2.3 連合学習とそのフレームワーク

分散データ活用における転送量削減の別のアプローチとして、機械学習の分野では連合学習が挙げられる。一般的な機械学習では学習データを一箇所に集めてから学習を実施する。連合学習ではまず複数の分散したノードで各々学習し、個々の学習結果として重み（パラメータ）を中央の計算ノードに転送して集約することで最終的にひとつのモデルを作り上げる手法である。

連合学習を行うソフトウェアの開発には、Flower[4]あるいは TensorFlow® Federated[5]（TensorFlow は Google LLC の登録商標または商標です）などのフレームワーク（FW）が利用できる。これらの FW はクライアント-サーバモデルのアーキテクチャとなっていて、分散した各データに対して学習を行うクライアント群と、その学習結果を統合するひとつのサーバから構成される。クライアント-サーバ間の通信は FW が担っていて、FW 利用者はモデル定義、学習データおよび学習結果の統合方法を FW に与えるだけで簡単に連合学習できるようになっている（図 2）。

このような手法により転送量削減だけでなく学習データをデータソース外に出さないためプライバシーの保護が必要な場面でも適用できる。例えば、NVIDIA 社は 20 の病院が持つ患者データを用いて連合学習したところ、COVID-19 患者の酸素必要量を高い精度で予測できたとの報告がある[4]。また、別の記事[7]では、複数の銀行が持つデータを使った不正取引の検知や、介護の分野で各被介護者のプライベートなデータの学習に連合学習が適用されていると紹介されている。このように連合学習は注目の技術であり、世の中で適用され始めている[8]。

3. 分散ストアドファンクションの提案

3.1 現状の問題点

2.3 節の通り、連合学習はデータ転送量削減やデータのプライベシー保護に有効で、連合学習 FW を活用すると学習処理の実現は容易になり便利である。しかし、データ取得あるいは処理内容の管理の面ではまだ改善の余地があると考えている。データソースの種類が混在する場合、種類に応じてデータ取得処理を実装しなければならない。また、複数のノード上でそれぞれプログラムを配置し実行させる必要があるため、それらのプログラムを一元管理するには一工夫必要な状況である。

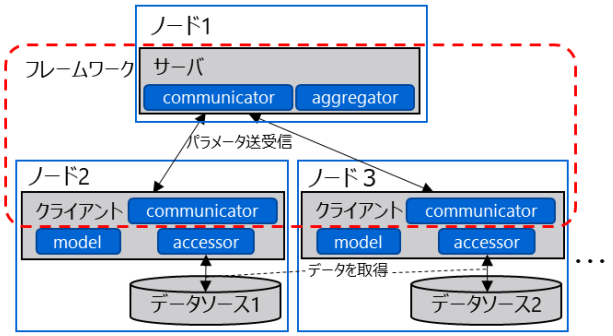

図 2 連合学習フレームワーク

一方データ仮想化の分野では、集約関数のプッシュダウンによりデータ転送を効率化できる（2.1.2 節）が特定の関数に限られていて、任意のユーザ定義処理には対応していない。これは、データソース上で実行する処理と仮想化エンジン上で実行する処理は集約関数ごとに異なるため、あらかじめ仮想化エンジンに集約関数個別の対処を組み込む必要があるからである。

そこで、ユーザが定義した任意の処理をプッシュダウンできる仕組みを考案した。これを分散ストアドファンクション機能（以降、分散関数と記載する）と名付けた。仮想化エンジンに分散処理の仕組みを搭載させることで、データソースの異種性を吸収しつつ、処理内容を一元管理できることを狙っている。

3.2 提案機能の概要とアーキテクチャ

3.3 提案機能の実現

分散関数とはマルチテナントテーブルを対象に計算を行う集約関数でありストアドファンクションである。親関数と子関数から構成される。子関数はマルチテナントテーブルに所属する子テーブルのデータを入力として、各データソース上で実行される関数である。親関数は子関数の結果を入力として、仮想化エンジン上で実行される集約関数である。

今回、この機能を PGSpider 上に実現した。本節ではその使い方および PGSpider 内部での実現方法を説明する。

3.3.1 外部仕様

PGSpider は分散関数を作成するための SQL クエリとして CREATE DISTRIBUTED_FUNC コマンドを提供する。あらかじめ、親関数および子関数を仮想化定義した上でこのコマンドを実行することで分散関数を作成する。具体的な仕様は以下の通りである。

```
CREATE DISTRIBUTED_FUNC 分散関数名 ( 引数 1, [ ... ] )
    PARENT 親関数名 ( [ 引数 2 ] )
    CHILD 子関数名 ( [ 引数 3, [ ... ] ] );
```

引数 1 には分散関数の引数（データ型と名前）を指定する。引数 2 は 親関数の引数で、子関数の返り値と同一でなければならない。引数 3 は 子関数の引数で、分散関数の引数と同一でなければならない。

このようにして定義した分散関数は、従来の集約関数の実行と同様に、以下のような SELECT 文の中で呼び出すことができる。

```
SELECT 分散関数名 (引数 1...)
                    FROM マルチテナントテーブル名;
```

このクエリの実行要求を受けた PGSpider はまずマルチテナントテーブルに対する子テーブル群を見つけ、子テーブルが所属する各データソースに対して子関数を定義し実行する。

```
SELECT 子関数名 (引数 1 ...) FROM 子テーブル名;
```

その後、各子関数の結果を受け取り、親関数を実行する（次の SQL 文相当の処理）。

```
SELECT 親関数名 (引数 2) FROM 子関数の結果;
```

3.3.2 内部の実現の仕組み

PGSpider のベースとしている PostgreSQL の集約関数の実行処理を拡張して実現している（図 4）。単純な集約関数は、transition 関数と final 関数で構成される。処理の流れは、（1）まず関数の入力となるレコード群を得る。（2）次にレコードをひとつ選び transition 関数を実行し、中間結果を得る。別のレコードと中間結果を入力として transition 関数を実行し中間結果を得る。これをすべての入力レコードに対して実行し、（3）最後に中間結果を入力として final 関数を実行する。

分散関数の場合、親関数を従来の集約関数とみなして実行し、その入力データとして子関数の結果を利用する。少し具体化すると、（1）の処理としてまず分散関数から親関数および子関数を特定する。また、分散関数の実行対象のマルチテナントテーブルを元に対象の子テーブル群を見つける。その子テーブルに対して子関数をデータソース上で実行することでレコード群を取得する。（2）では（1）で得られた子関数の結果を入力として親関数の transition 関数を実行する。（3）では親関数の final 関数を実行する。

3.4 適用例

今回提案した分散ストアドファンクション機能を連合学習に適用してみた。連合学習 FW には Flower を利用した。連合学習のクライアントを子関数、サーバを親関数として PGSpider 上に分散関数を作成して実行した。この FW は Python のパッケージなため、PL/Python という手続き言語を使用して親・子関数の処理内容を記述した。

この適用に際し、子関数の実行完了を待たずに親関数を実行可能にする機能を導入した。Flower 側でサーバ・クライアント間の通信を行うため（図 3）、PGSpider 上での親・子関数間のデータ授受は不要で、その代わりに親・子関数を同時に起動させることが必要となったからである。親関数が入力を必要としない（引数がない）とき、本機能が動作するようにした。

このように、Flower のクライアントおよびサーバプログラムを分散関数として定義することで、ユーザは個々のデータソースに接続することなく PGSpider に接続するだけで、学習のモデルを変更したり学習の開始を命令したりできるようになる。

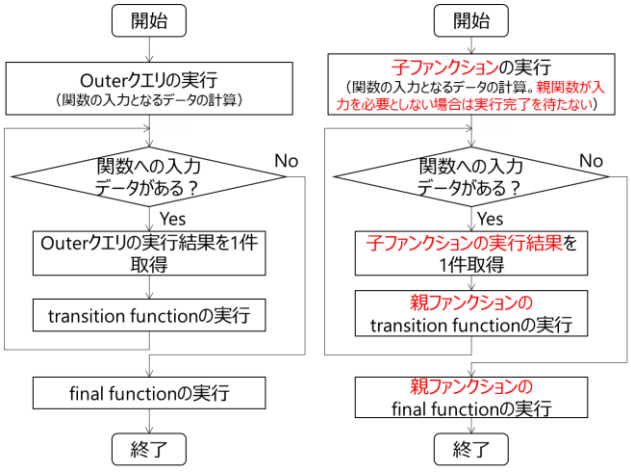

図 4 内部の処理フロー
（左：通常の集約関数、右：分散関数）

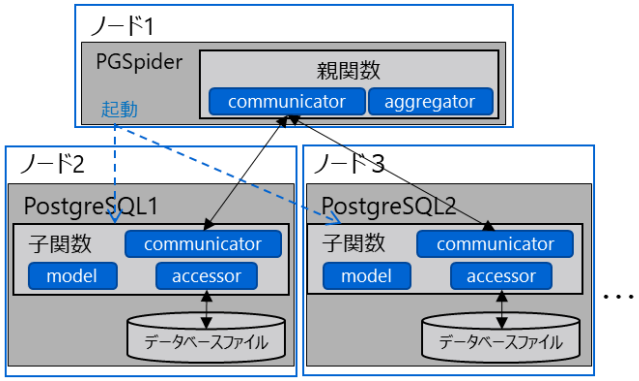

図 3 提案機能の連合学習への適用

4. 評価と考察

4.1 定性的評価

既存のシステムを変更することなく PGSpider を導入するだけで、複数のデータソースに対する分散処理を定義できるようになり、データ処理内容の自由度が向上した。また、処理内容を仮想化エンジン上で一元管理できるようになることも提案機能によるメリットである。この技術は、連合学習以外の計算に対しても適用でき、プライバシーを保護しつつデータ転送量の削減が行える汎用的な仕組みとして分散データ処理基盤の機能と考えている。

コンセプトとしては、データ仮想化エンジンに分散処理を定義できる仕組みを持たせることで、データソースの異種性を吸収する狙いがある。今回は 4.1 節で述べるように、データソースとして PGSpider と親和性の高い（同じ手続き言語が使用できる）PostgreSQL を対象に動作確認した。もし他の種類のデータソースが混在した場合、その種類に依存しない手続き言語が必要になってくるため、処理内容の仮想化をどう実現するかは今後の課題と考えている。

4.2 PGSpider による学習結果への影響

PGSpider を介して連合学習した場合でも PGSpider を介さず連合学習した場合と学習結果は変わらないはずである。まずはこのことを確認した。

実験環境は AWS 上に構築した。10 台の仮想マシンを作成し、8 台をデータソース用（m5.large）、1 台を統合エンジン用（m5.2xlarge）、1 台をアプリケーション用（m5.xlarge）とした。データソースとして、8 つのノードに PostgreSQL を動作させた。学習データは MNIST を使用し、画像 1 枚を 1 レコードとして 6 万枚の画像を 8 分割してそれぞれの PostgreSQL に格納した。連合学習 FW には Flower を使用し、実装は[9]を参考にした。

学習回数に対する評価値を表 1 に示す。学習回数を重ねる毎に精度が向上し、両者で同じような傾向となった。このことから、PGSpider を介すことによる学習結果への影響はないことが確認できた。

4.3 性能評価

学習にかかる時間を比較した。PGSpider を利用した場合、オーバヘッドによる性能劣化がある一方で、データ転送を抑えられるため性能改善も期待できる。ノード構成は 4.2

表 1 学習結果

学習回数	accuracy		loss	
	Flower 単独	PGSpider 経由	Flower 単独	PGSpider 経由
1	0.8062	0.8110	2.8677	2.7695
2	0.8815	0.8732	1.1955	1.2920
3	0.9010	0.8992	0.7914	0.8224
4	0.9086	0.9100	0.6498	0.6715
5	0.9141	0.9143	0.5561	0.5837
6	0.9204	0.9181	0.5099	0.5215
7	0.9219	0.9220	0.4738	0.4805
8	0.9255	0.9255	0.4448	0.4498
9	0.9286	0.9270	0.4253	0.4278
10	0.9304	0.9287	0.4073	0.4060

節と同じく、各データソースが 7,500 枚のデータを保有し、学習で使用するノード数を変化させて測定した。さらに、データ量を増やした場合の傾向を調査するために、各データソースが 6 万枚のデータを保有するケースでも実験した。

図 6 に示すように、データ量が少ない場合は PGSpider の方が少し低速であった。そのオーバヘッドは 5%未満であり、ユースケースによって許容される範囲であると考えている。一方で、データ量が多い場合、PGSpider の方が 6～8%高速になった。データソース数に着目すると、データソース数によらず同じ傾向となった。これは PGSpider がデータソース数の影響がほとんどないことを示している。

データ量を多くすると性能が向上する理由について調査した。従来手法と提案手法で異なる処理は 2 箇所ある。ひとつはデータの取得処理である。従来は PostgreSQL の I/F を使用した方法であり、提案手法では PL/Python による方法である。取得したデータは機械学習処理の I/F に合うよ

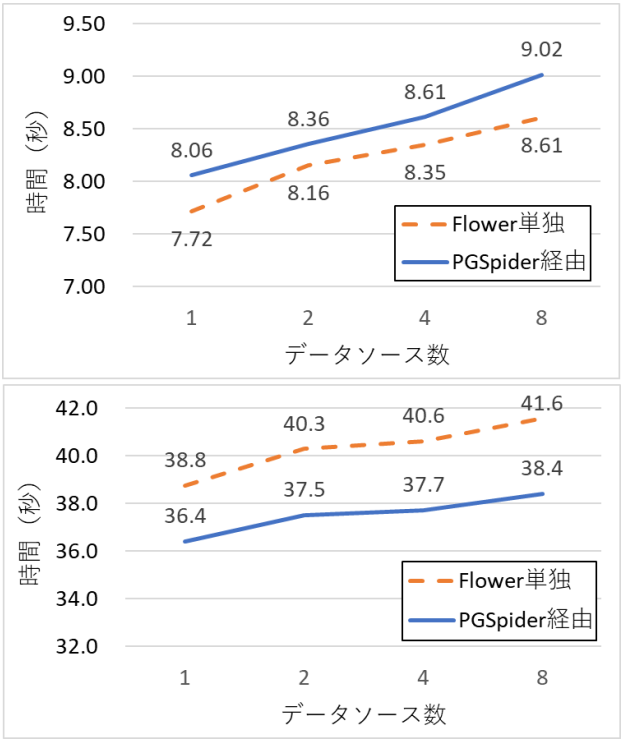

図 6 学習に要した時間（各データソースのデータ数：上 7,500 件ずつ、下 60,000 件ずつ）

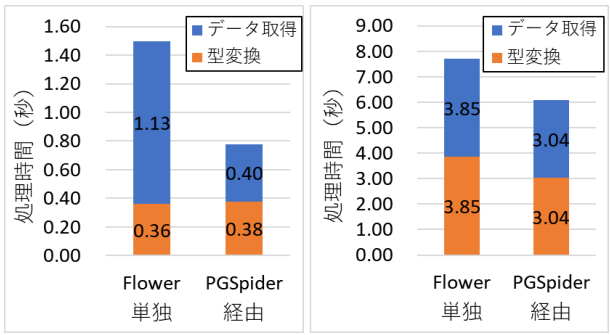

図 5 データ取得と型変換処理の時間（各データソースのデータ数：左 7,500 件ずつ、右 60,000 件ずつ）

うに型変換を行う必要がある。異なるデータ取得方法によりそれで得られるデータの型も異なるため、型変換処理の内容も両者で異なる。これが 2 つ目の違いである。

これらのデータ取得時間および型変換時間を測定した結果を図 5 に示す。この結果からわかるとおり、型変換時間はそれほど差がなく、データ取得時間の差が性能差に影響していた。従来手法の場合、クライアントプログラムは PostgreSQL からデータを取得する必要があり、一旦 PostgreSQL 外に出ることになる。一方で PGSpider を利用した場合、データを格納する PostgreSQL 内にデータを留めたまま学習処理を行うことができる。同一ノード内の処理とはいえデータ移動を抑えることにより、データ量が大きくなるとオーバヘッドが無視できるほど改善効果が出たことが性能向の理由だと考えている。

5. おわりに

無駄なデータ転送を抑えデータを活用する技術として、データ仮想化エンジンによる分散データ処理機能を提案した。分散したデータを一箇所に集めることなくユーザが任意の処理を実行可能になることに加え、その処理の管理がデータ仮想化エンジンのみで行えるようになる。また、データをデータソースのノード外もっと言えばデータを管理するプログラム外に出すことなく処理が行えるため、プライバシー保護や性能の点でも有効な手法となり得る。日々世の中ではデータが様々な場所で生成されているので、このような技術を実システムで利用しデータ活用の促進に繋げたいと考えている。

参考文献

[1] PGSpider, https://github.com/pgspider
[2] Taiga Katayama, PGSPIDER - HIGH-PERFORMANCE SQL CLUSTER ENGINE., FOSDEM PGDAY 2020
[3] PostgreSQL, https://www.postgresql.org/
[4] Flower, https://flower.dev/
[5] TensorFlow Federated, https://www.tensorflow.org/federated
[6] Triaging COVID-19 Patients: 20 Hospitals in 20 Days Build AI Model that Predicts Oxygen Needs, https://blogs.nvidia.com/blog/2020/10/05/federated-learning-covid-oxygen-needs/
[7] 連合学習（フェデレーテッドラーニング）とは。仕組みや活用例を解説, https://www.eaglys.co.jp/news/column/secure-computing/encryptfederatedlearning/
[8] 清藤武暢, プライバシー保護技術としての 連合学習の仕組みと最新動向, IEICE Fundamentals Review Vol.16, No.3 (2022)
[9] Akira Shirahama, Federated Learning Tutorials, https://github.com/AkiraShirahama/federated_learning_tutorials.git