匿名通信システムを介したデータ共有アーキテクチャの実現と評価

2018SC061 中西遼　2018SC062 中尾瑠以
指導教員：石原靖哲

1　はじめに

近年，大学や企業など複数の企業が参画し共同で研究開発を行うことが盛んになってきている．研究開発の対象が複雑になるにつれ，参画組織が共有するデータ自身はもちろんデータの共有の仕方も複雑化している．そのため，ある組織におけるデータの更新が，整合性を保証しつつ他の組織の所有するデータに自動的に伝播される仕組みを考える必要性がある．さらに，新製品の極秘開発などのようにどの組織と共同研究開発しているのかを第三者から秘匿したい場合も考えられる．

関係データを柔軟に共有するためのアーキテクチャとして Dejima アーキテクチャ [1, 3] が提案されており，プロトタイプ [4] も実装されている．Dejima アーキテクチャでは，双方向変換技術と分散トランザクション管理により，どの参画組織がデータを更新してもそれが他の組織に整合性をもって伝播することが保証されている．また，通信相手を秘匿する技術としては，Tor がよく知られている．Tor では通信を行う際に世界各地に点在しているノードを経由することによって IP アドレスを匿名化している．しかし，我々の知る限り，Dejima アーキテクチャが提供しているような複雑なデータ共有を匿名通信システム上で実現した例は存在しない．

そこで本研究では，Tor を介した Dejima アーキテクチャの実現をすることで，匿名通信システムを介したデータ共有アーキテクチャの実現を行う．Dejima アーキテクチャのプロトタイプでは，異なる参画組織間で分散トランザクション処理のための通信が行われる．本研究では，その通信が Tor を介して行われるよう，プロトタイプを改変する．そして，Tor を介すことによる通信速度の評価を行う．

結果として，Tor を介さない Dejima アーキテクチャにおいて一行のデータを挿入した場合，実行時間が約 0.1 秒であった．対して，Tor を介した Dejima アーキテクチャにおける 1 行のデータの挿入時間は約 3.2 秒であった．一方で，データを閲覧する際はトランザクション処理の通信が行われないため，Tor を介している状態でも遅延が発生しないことを確認した．

2　関連研究

2.1　Dejima アーキテクチャに関する研究

浅野らの研究 [1, 3] は，Dejima アーキテクチャの詳細を説明しており，Dejima アーキテクチャを用いたライドシェアリングアライアンスの実現を通して，Dejima アーキテクチャの有用性を示す研究である．浅野らの研究では，Dejima を実装することで満たすことのできる特性を複数挙げ，それに対して議論を行っている．浅野らの研究では位置情報の難読化技術を使用しているが，本研究では匿名通信システムである Tor を使用する．

2.2　Tor を介した FDW に関する研究

松山の研究 [2] は，Tor と SSH サーバを介したポートフォワーディングを行う際に，外部に存在するデータにアクセスすることが可能な PostgreSQL の拡張機能である外部データラッパ (Foreign Data Wrapper，以下 FDW と省略) を用いて行っている．この研究の課題として，Tor を介すことによる通信速度の低下が挙げられる．本研究では Tor を介した Dejima アーキテクチャを実現し，Tor を介したデータ共有を行う．

3　Dejima アーキテクチャ

3.1　概要

Dejima アーキテクチャは，データの双方向変換を利用した更新伝播を用いて，柔軟なデータ統合を可能とするアーキテクチャである．三宅 [4] によって Dejima のプロトタイプが公開されている．

Dejima アーキテクチャはピア, ベーステーブル, Dejima テーブル，Dejima グループの 4 つの構成要素から成る．ピアはそれぞれ固有のデータベースを持ち，このデータベース内にベーステーブルと，ベーステーブルから導出されるビューである Dejima テーブルを保持している．この Dejima テーブルは双方向変換技術を使用することで，更新可能なビューとなっている．また，このビューに対して直接更新することができ，更新した内容もベーステーブルに適用される．各ピアはデータを共有しているピア同士で Dejima グループを形成する．この時同じ Dejima グループに属しているピアは，Dejima グループに対応した同一の Dejima テーブルを同期して保持している．

図 1 に Dejima アーキテクチャの更新伝播の図を示し，以下にその詳細を示す．

1. ピア P_1 においてユーザがベーステーブル BT_1 内のデータベースを更新する．
2. 上記の更新により，P_1 内に存在する Dejima テーブル DT_1 も更新される．
3. Dejima テーブル DT_1 にて行われた更新内容を，同じ Dejima テーブルを所有するピア P_2 に対して通知を行う．

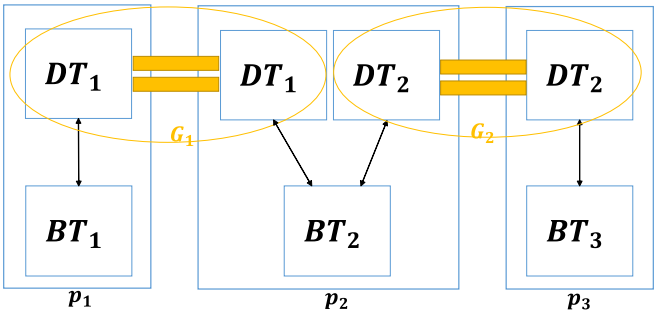

図 1　Dejima アーキテクチャの概要

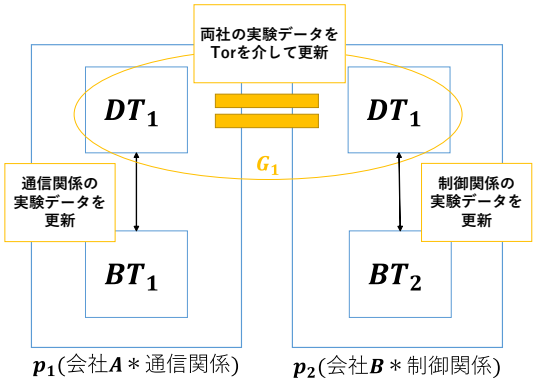

図 2　ユースケースの概要

4. P_2 での DT_1 の更新がまず BT_2 に伝わり，それが DT_2 に反映される．
5. これ以降は，Dejima テーブルが更新されなくなるまで上記の (2) から (4) を繰り返し実行する．

3.2　Dejima アーキテクチャと Tor を用いたユースケース

Dejima アーキテクチャと Tor を用いたユースケースとして，会社間である製品を共同で開発する際のデータ共有を提案する．図 2 にそのユースケースの概要を示す．背景として，会社 A と会社 B は製品を共同で開発・研究を行い，データ共有を実行する際 Tor を介して執り行う．このデータ共有システムは，ピア P_1 とピア P_2 の二つのピアを用いて行う．ピア P_1 を会社 A(通信関係)，ピア P_2 を会社 B(制御関係) とする．また，Dejima グループ G_1 で共有するデータは，製品を作成するために必要な通信項目のデータと，制御関係項目のデータとする．2 社とも新たに実験などで得られたデータがあった場合，図 2 のユースケースの概要のようにデータ共有を実行する．

これを使用することで，会社が物理上距離があったとしてもデータの内容が共有でき，かつ，閲覧・編集することが可能になるため，A 社と B 社が共同開発を簡易に，かつ，飛躍的に進めることができる．つまり，上記のシステムの実現が可能であれば，会社 A と会社 B の双方に Dejima アーキテクチャのメリットを発生させることができると考える．

次に，上記のシステムに Tor を用いた場合にメリットが発生するかを考える．これを匿名で実現させるメリットとして，共同研究を公にしなくて済むという点が挙げられる．例えば中小企業である会社 A と会社 B があり，大企業である会社 C が存在する場合を想定する．会社 A と会社 B の共同研究を匿名通信を介さず行う場合，会社 A と会社 B の共同研究の内容を盗聴をされてしまい，先に公表される恐れがある．しかし匿名通信を介して共同研究をすることで，新製品のプレスリリースまで安全に共同研究を行うことができる．以上のように会社 A と会社 B の双方に Tor を使用するメリットがあると考える．

3.3　一般的なファイル共有と Dejima アーキテクチャの違い

Google Drive などの一般的なファイル共有では，関係データのような複雑な依存関係を持つデータを扱うことはできない．しかし，Dejima アーキテクチャは双方向変換とトランザクションを用いた，関係データのための共有アーキテクチャであるため，所持している関係データから共有のために任意のデータを選択し，更新も含めた管理することができる．

また，松山の研究で用いられている PostgreSQL の拡張機能である FDW は postgres_fdw の更新機能オプションを用いることで，外部からテーブルのデータを直接更新するため，元のテーブルにも更新を反映させることが出来る．FDW と Dejima アーキテクチャの違いは，FDW では外部テーブルを作成し，更新機能オプションを用いて直接テーブルに接続しデータ共有を行うため，テーブルのすべてのデータを共有することになる．また，FDW ではテーブルのデータの一部を取り出し更新を行っても，ファイル共有と同様に更新することが出来ない．対して，Dejima アーキテクチャはベーステーブルに存在する任意のデータのみを Dejima テーブルに送ることで，ベーステーブルの特定のデータを秘匿した上で共有を行うことが出来る．

4　実現環境

本節では，本研究の実現環境である The Onion Router, Docker, PostgreSQL について説明をする．

4.1　The Onion Router

Tor (The Onion Router) は米海軍調査研究所で，インターネットでの匿名性の向上を目的として開発された匿名化ネットワークであり，現在世界で最も利用されている匿名通信システムである．

Web サイトなどへアクセスした際に残るユーザの IP アドレスを，インターネット上に置かれている多数あるノードから無作為に 3 つ以上選択し，ノード A からノード B，ノード C のように接続することで，ノードの壁を作成し IP アドレスを特定できなくするというものである．

4.2 Docker

Docker は OS 環境にコンテナと呼ばれる分離された空間を生成し，アプリケーションを開発，配置，実行するための仮想環境を構築するツールである．

コンテナは Docker エンジンの上で動作している仮想環境のことであり、ホスト OS のカーネルを利用して動作するという特徴を持つ．Docker はコンテナごとに OS が独立して動くのではなく，共有することによりホスト OS のカーネルを各コンテナが共有するので，ハードウェアの CPU やメモリ消費を抑制しながら動作することが可能になる．Docker で使用する Linux ディストリビューションには実装のし易さにおいて優位性を持つ Debian と動作速度に優位性をもつ Alpine が存在する．本研究では，実現の容易さを優先させることにし，実装面において優位性を持つ Debian を用いて実装を行う．以上の理由より本研究では Docker を使用する．また，三宅の GitHub 上で Dejima-prototype-master[4] をフォークし，そのクローンを元に編集しコンテナを作成することで実現を行う．

4.3 PostgreSQL

PostgreSQL はオープンソースのリレーショナルデータベースマネジメントシステム（RDBMS）と呼ばれるものの一つであり，Linux や MacOS，Windows など様々な OS に対応している．オープンソースの DBMS としては MySQL も有名であるが，本研究では FDW も使用するため，拡張機能が優れている PostgreSQL を使用する．

外部データラッパ (FDW) とは，外部に存在するデータにアクセスするための PostgreSQL の拡張機能である．PostgreSQL から接続可能な外部データは多数あり，PostgreSQL はもちろん，PostgreSQL 以外の RDB である MySQL や Oracle などにも接続可能である．実際に外部データ接続を行う際には，対応している外部データラッパを PostgreSQL の拡張機能としてインストールを行う．FDW を用いる際，外部サーバの設定を行い，外部データを外部テーブルとして定義することで，FDW を行い外部からサーバ側のデータに接続することが可能になる．

5 Tor を介した Dejima アーキテクチャの実現

本研究の最終目的は Tor を介した Dejima アーキテクチャの実現である．図 3 に実現したアーキテクチャによるデータ共有の詳細を示す．このアーキテクチャは会社 A-db が所持しているベーステーブル 1 のデータを更新することで，双方向変換を用いた通信と Tor を介した通信を行い，会社 B-db が所持するベーステーブル 2 のデータに更新内容が反映されるアーキテクチャである．

5.1 実現の概要

上述したように，このアーキテクチャは会社 A-db が所持するベーステーブル 1 でデータを更新した際に，会社

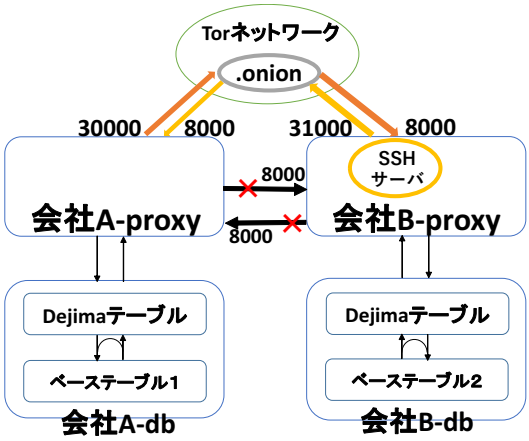

図 3 データ共有の詳細

B-db が所持するベーステーブル 2 に更新内容が反映されるアーキテクチャである．データの更新方法は，まずベーステーブル 1 と PeerA-db の Dejima テーブルで双方向変換を行いデータを共有する．次に，Dejima テーブルから会社 A-proxy に変更されたデータを送信し，会社 A-proxy から会社 B-proxy に 8000 番のポートを用いて更新データの送信をする．この時 proxy 同士の通信を，通信で用いている 8000 番とホストで指定する 30000 番のポート番号を用いて Tor と SSH サーバを介したポートフォワーディングを行い，匿名でのデータ共有を実現している．そして，更新データを受け取った会社 B-proxy から会社 B-db の Dejima テーブルに更新データを会社 B-db の PosgreSQL のポート番号を用いて送信し，Dejima テーブルとベーステーブル 2 で双方向変換を行うことで，ベーステーブル 1 で更新したデータをベーステーブル 2 に反映することが可能となる．

5.2 実現の手順

Tor を介した Dejima アーキテクチャの実現をするために行った手順を示す．

まず，三宅の作成した Dejima アーキテクチャのプロトタイプを用いて，Docker のコンテナを作成した．Tor を介した接続は SSH を用いて行うため，PeerA-proxy(クライアント) と Mediator-proxy(サーバ) のコンテナで公開鍵認証を用いた SSH 接続の設定を行った．次に，Dejima-prototype-master のトランザクション処理を行う通信の設定ファイルを編集した．その後，サーバ側のコンテナで SSH サーバを介した接続を行えるように Tor の設定ファイルを編集した．また，クライアント側のコンテナで Tor を介した SSH の接続を行うためのプロクシ設定を行った．そして，Tor を起動することで作成される hostname を用いて，クライアント側のコンテナからサーバ側の SSH サーバに Tor を介したローカルポートフォワーディングと，リモートポートフォワーディングを行った．最後に PeerA-db のコンテナで PostgreSQL を用いてベーステーブルに

表 1 SELECT の実行時間

	実行時間 [ms]	
	FDW	Dejima
何も介さない場合	14.498	0.472
SSH を介した場合	17.165	0.533
Tor を介した場合	4546.357	0.438

表 2 INSERT の実行時間

	実行時間 [ms]	
	FDW	Dejima
何も介さない場合	22.671	86.412
SSH を介した場合	23.757	99.985
Tor を介した場合	4603.780	3206.722

データを挿入し，Mediator-db の PostgreSQL でデータの共有を行えていることが確認出来たため，Tor を介した Dejima アーキテクチャの実現を行うことが出来た．

6 評価

以下の 6 通りの実現をし，SQL を実行してから結果が返るまでの時間の比較を行った．時間の計測は PostgreSQL の \timing を使用した．timing コマンドをオンにした状態で SELECT と INSERT の SQL を実行し，結果が返ってくるまでの時間 (10 回行った平均) をそれぞれ表 1 と表 2 に示す．この時，INSERT ではデータを一行のみ挿入して実行時間の計測を行った．また FDW を行うコンテナには，Dejima のコンテナに存在するベーステーブルと同じ定義を指定したテーブルを作成して計測を行う．

- 何も介さず FDW を行う場合
- SSH サーバを介して FDW を行う場合
- Tor を介して FDW を行う場合
- 何も介さない Dejima アーキテクチャの場合
- トランザクション処理を行う通信で SSH サーバを介した Dejima アーキテクチャの場合
- トランザクション処理を行う通信で Tor を介した Dejima アーキテクチャの場合

その結果，何も介さない Dejima アーキテクチャの場合，実行時間は約 0.1 秒であった．対して，Tor と SSH サーバを介した Dejima アーキテクチャの場合，実行時間は約 3.2 秒であった．一方で，データを閲覧する際はトランザクション処理の通信が行われないため，Tor を介している状態でも遅延が発生しないことを確認した．また，FDW と Dejima アーキテクチャを用いた際の両方で，何も介さない場合と SSH サーバを介した場合の SQL の実行にかかった時間の差はあまりなかった．しかし Tor を介した FDW の場合は，実行した SQL の両方で FDW のみや SSH サーバを介した FDW よりもかなりの遅延が発生していた．対して，Tor を介した Dejima アーキテクチャの場合は，データの挿入を行った場合のみ遅延が発生した．また，Dejima アーキテクチャを用いたデータの挿入時間を計測した結果，Dejima アーキテクチャは FDW より遅延が少なく，約 1.4 秒の差が発生するという結果になった．

7 まとめ

本研究では，Tor を介した Dejima アーキテクチャの実現を行った．何も介さない Dejima アーキテクチャを用いた場合，データの挿入時間は約 0.1 秒であった．対して，トランザクション処理を行う通信で Tor を介した Dejima アーキテクチャを用いた場合，データの挿入時間が約 3.2 秒であった．これは，Dejima アーキテクチャはデータの更新を行う場合のみトランザクション処理を行う通信をして，外部のテーブルに共有をする．本研究では，そのトランザクション処理を行う通信で Tor を介して Dejima アーキテクチャの実現を行ったため，更新を行う場合は実行時間に遅延が発生する．また，更新を行わないデータの閲覧をする場合は，トランザクション処理を行う通信をしないため実行時間に遅延は発生しない．また，FDW は外部のテーブルに接続する通信で Tor を介すため，データを閲覧する場合も遅延が発生した．対して，Dejima アーキテクチャでデータの閲覧をする場合は，トランザクション処理を行う通信をしないため遅延が発生しない．

本研究では，挿入するデータを増やすことでトランザクション処理のエラーが発生してしまうため，データの挿入は一行の挿入のみで行った．そのため，エラーを発生させず挿入数を増やし，評価を行うことが今後の課題である．

参考文献

[1] Yasuhito Asano, Dennis-Florian Herr, Yasunori Ishihara, Hiroyuki Kato, Keisuke Nakano, Makoto Onizuka, and Yuya Sasaki. Flexible framework for data integration and update propagation: system aspect. In *Second Workshop on Software Foundations for Data Interoperability*, 2019.

[2] 松山到生．PostgreSQL における匿名通信システムを介した外部データラッパ機能の実現と評価．南山大学理工学部機械電子制御工学科卒業論文, 2020.

[3] Yasuhito Asano, Zhenjiang Hu, Yasunori Ishihara, Hiroyuki Kato, Makoto Onizuka, and Masatoshi Yoshikawa. Controlling and sharing distributed data for implementing service alliance. In *Second Workshop on Software Foundations for Data Interoperability*, 2019.

[4] Kouta Miyake. ekayim/Dejima-prototype. https://github.com/ekayim/dejima-prototype.git.