Dejima アーキテクチャにおけるサーバ配置の提案と性能評価

2019SC011　橋本大輔　2019SC012　檜木和幸
指導教員：石原靖哲

1　はじめに

近年, 企業や大学など多くの組織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ればなるほど, その組織が共有したいデータやそれらデータを共有する方法も複雑になっていく．そのためある組織におけるデータの更新が, 整合性を保証しながら他の組織が保持しているデータに自動的に伝播される仕組みが必要である．

我々の所属する研究室では, 関係データを柔軟に共有するためのアーキテクチャである Dejima アーキテクチャ [1, 2] の研究が行われている．また, 他の研究室ではプロトタイプ [3] も実装されている．Dejima アーキテクチャは，データの双方向変換と分散トランザクション管理を利用し, 更新伝搬を用いて適応性に富んだデータ統合を可能としたアーキテクチャである．しかし，遠隔地を結んだデータ共有については，今のところ実証実験が行われておらず，得られた知見も十分ではない．

また近年，データ処理のアーキテクチャとして, データを集中処理するクラウドコンピューティングとデータを分散処理するエッジコンピューティングが導入されている．特にエッジコンピューティングでは，負荷を分散することができ通信の遅延も起こりにくため, リアルタイムかつ低負荷なデータ処理の実現にしばしば用いられる．

そこで本研究では，エッジコンピューティングとクラウドコンピューティングの 2 つの観点から, Dejima アーキテクチャについて 2 つのパターンのサーバ配置を行う．そして，それらの処理時間について比較し評価を行う．

結果として, Dejima アーキテクチャは PostgreSQL の INSERT 文のみの問合せを 2 人だけの情報共有で行った場合，1000 行のときパターン 1 に対するパターン 2 の処理時間の比は，更新伝播の有無に関わらず 10 倍以上の差があったため, エッジコンピューティングの観点からサーバ配置を行った方が適していることが分かった．

2　関連研究

2.1　非集中型データ統合アーキテクチャにおける分散トランザクション制御手法に関する研究

三宅の研究 [5] は，Dejima アーキテクチャを分散トランザクション制御の観点から分析することで，非集中型データ統合アーキテクチャにおける効率的な分散トランザクション制御手法を 2 つ提案した．Dejima システムを実装し評価実験を実施した結果，競争率の高いワークロードにおいて保守的な 2 相ロック，適応的 2 相ロックともに通常の 2 相ロックの性能を上回る性能を達成している．

2.2　Tor を介した Dejima アーキテクチャに関する研究

中西らの研究 [6] は，Tor を介した Dejima アーキテクチャの実現を行った．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を行うための通信をしている．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した．そして，Tor を介すことによる Dejima アーキテクチャと FDW の通信速度の比較をして評価を行った．本研究では，Dejima アーキテクチャのサーバ配置をクラウドコンピューティングとエッジコンピューティングの概念を取り入れ変化させ，比較することで性能評価を行う．

2.3　エッジコンピューティングの性能評価に関する研究

今金らの研究 [4] は，エッジコンピューティングを活用したマルチメディアを処理したときの実行速度を実際のクラウドサービスを用いることで, 比較し評価した．結果としてユーザの近くであるネットワーク周縁にあるエッジサーバの方がコア (中心) に位置するコアサーバよりも実行速度を短縮できることがわかった．

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, 更新伝搬を用いて, 適応性に富んだデータ統合を可能としたアーキテクチャである．Dejima アーキテクチャは以下の 4 つの構成要素からなる．

- ピア
- ベーステーブル
- Dejima テーブル
- Dejima グループ

各ピアごとに独自のスキーマを持つベーステーブルと，各ピアのベーステーブルから導出されるビューである Dejima テーブルを保有している．この Dejima テーブルは双方向変換を利用した更新可能なビューである．従って, このビューに対して直接更新をすることができ，その更新した内容はベーステーブルに適用される．ピアはデータを共有したい他のピア同士と Dejima グループを形成し, Dejima グループに属しているピア間で, Dejima テーブルを同期している．図 1 に Dejima アーキテクチャにおける更新手順の概要を示し, 以下にその詳細を示す．

1. ピア P1 においてユーザがベーステーブル BT1 内のデータベースを更新する．

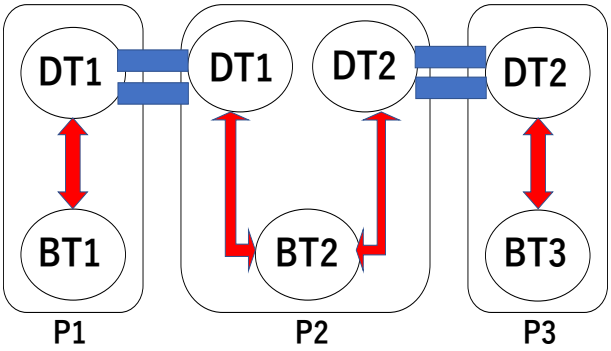

図 1　Dejima アーキテクチャの概要

2. 同じ P1 内に存在する Dejima テーブル DT1 にその更新が反映される.
3. Dejima テーブル DT1 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 P2 に通知する.
4. 3 で通知された更新内容を元に, P2 での DT1 の更新がまず BT2 に伝わり，それが DT2 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　分散トランザクション制御

分散トランザクションとはトランザクション処理における処理形態の 1 つである．ネットワーク上における 2 つ以上のホストが関連する, 一連の操作 (トランザクション) のことを示す. データベースにアクセスするようなトランザクションのことはグローバルトランザクションと呼ばれる．一方 1 つのデータベースにのみアクセスするようなトランザクションのことはローカルトランザクションと呼ばれる.

3.3　双方向変換

双方向変換は, ソースと呼ばれる元のデータをビューと呼ばれる別のデータに変換した後, その変換後のデータ更新を元のデータに反映させることができる計算の仕組みのことである. データの変換はデータ共有において必要不可欠な操作であるが, 双方向変換を用いることで, 一貫性を保ちながら双方向にその更新を伝播させることができる.

4　実現環境

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

4.1　Docker

Docker は OS 環境にコンテナと呼ばれる隔離された空間を生成し，アプリケーションを開発, 配置, 実行するためのコンテナ仮想化ツールである．コンテナはサーバのカーネルを利用することで, プロセスやユーザなどを他のプロセスから分離することができる．表面上全く別のマシンが OS 上に動いているかのように行うことが可能である.

4.2　PostgreSQL

PostgreSQL はオープンソースタイプの ORDBMS (オブジェクトリレーションデータベース管理システム) の 1 つであり，Windows や Linux，MacOS など様々な種類の OS に対応している．大規模もしくは複雑なデータの管理ができ，Excel のような表形式で保存したデータを 1 つのまとまりとし，別のまとまりと関連付けることで大量のデータを扱える．本研究では外部データラッパ (FDW) を使用するため，拡張機能が優れている PostgreSQL を使用する.

4.3　SSH

SSH は「Secure Shell」の略であり，ネットワークを介することで別のコンピュータを遠隔操作するためのソフトウェアである．SSH は，ポートフォワーディング機能を利用することができる．ポートフォワーディングとは，IP アドレスの TCP や UDP といった特定のポート番号への通信を, 別のアドレスの特定のポートへ自動的に転送することである．その経路に SSH を使用することで通信を暗号化することができる.

5　SSH を介した Dejima アーキテクチャの実現

本研究の最終目標はクラウドコンピューティングとエッジコンピューティングの観点から 2 つのパターンにサーバ配置をした Dejima アーキテクチャを実装し，処理速度について性能評価することである．図 2 にエッジコンピューティングの観点からサーバ配置を行った Dejima アーキテクチャのデータ共有の詳細を示す．また図 3 にクラウドコンピューティングの観点からサーバ配置を行った Dejima アーキテクチャのデータ共有の詳細を示す.

5.1　実現の概要

上述したように, エッジコンピューティングとクラウドコンピューティングの 2 つの観点から Dejima アーキテクチャのサーバ配置を行った．エッジコンピューティングは, IoT 端末などのデバイス自体や，その近辺に設置されたサーバでデータの処理・分析を行う分散コンピューティング手法である．一方, クラウドコンピューティングは, ユーザからの物理的な距離が長いクラウド環境でデータ処理を行うコンピューティング手法である．しかし, 我々が今回利用した三宅の Dejima アーキテクチャのプロトタイプは，すべてのコンテナが物理的に 1 台の計算機上で動作するように作られている.

そこで本研究では, このプロトタイプを改変し, PeerA と Mediator のコンテナ間の距離を長くするために PeerA-proxy と Mediator-proxy の間に研究室のレンタルサーバを介した, エッジコンピューティングの観点からサーバ配置を行った図 2 の Dejima アーキテクチャと, 図 3 のように一方のユーザと db との間にレンタルサーバを介した,

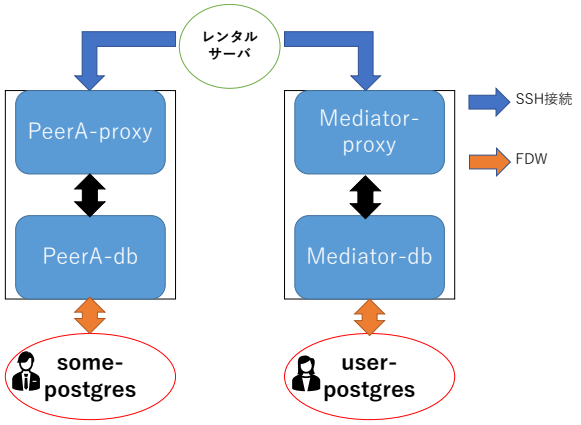

図 2　パターン 1 のサーバ配置

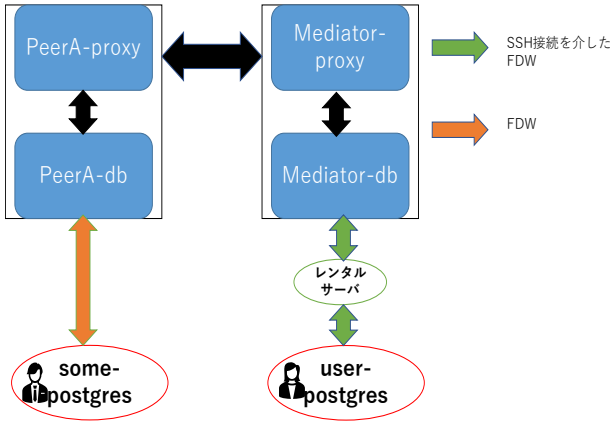

図 3　パターン 2 のサーバ配置

クラウドコンピューティングの観点からサーバ配置を行った Dejima アーキテクチャの実現を目指す.

5.2　実現の手順

図 2 で示したパターン 1 のサーバ配置の手順を示す. まず，三宅の作成した Dejima アーキテクチャのプロトタイプから Docker のコンテナを作成した．レンタルサーバを介した接続は SSH を用いて行うため，PeerA-proxy と Mediator-proxy のコンテナとレンタルサーバで公開鍵認証を行い SSH 接続の設定を行った．次に，PeerA-proxy と Mediator-proxy からレンタルサーバに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の設定を行った. そして，some-postgres と PeerA-db, user-postgres と Mediator-db の間は SSH を介していない FDW 接続を行った．最後に，some-postgres のコンテナで PostgreSQL を用いてベーステーブルにデータを挿入することで user-postgres の PostgreSQL でデータ共有を行えていると確認するとができたため, エッジコンピューティングの観点からサーバ配置を行った Dejima アーキテクチャの実現が出来た.

次に図 3 で示したパターン 2 のサーバ配置の手順を示す．ユーザのコンテナが必要なので some-postgres と user-postgres を作った．パターン 1 のサーバ配置と同様

表 1　PC のスペック

ノート PC	
OS	Microsoft Windows 11 pro
CPU	Intel(R) Core(TM)i7-10750H
メモリ	32.0GB

に user-postgres と Mediator-db の間でレンタルサーバを通すために SSH 接続を行い，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の設定を行った. some-postgres と PeerA-db の間はパターン 1 と公平にするために FDW を接続した．最後に，some-postgres のコンテナで PostgreSQL を用いてベーステーブルにデータを挿入することで user-postgres の PostgreSQL でデータの共有を行えていると確認するとができたため, クラウドコンピューティングの観点からサーバ配置を行った Dejima アーキテクチャの実現が出来た.

6　評価

本節では, 本研究の評価環境と評価結果について説明をする.

6.1　評価環境

処理速度の計測では, 研究室から貸された PC, 南山大学の Wi-Fi(00axia) と研究室のレンタルサーバ (「さくらの VPS」の「CPU 仮想 2Core メモリ 1GB 東京」) を使用した．PC 本体のスペックを表 1 に示す．また, PostgreSQL に初期内蔵されている\timing を使った．timing コマンドを使うことで実行してから結果が返るまでの時間を測ることができる.

次に，実験で用いたデータについて述べる．ベーステーブルのスキーマである dejima_id, vid, location, rid, sharable に情報を書き込む. その後，Dejima テーブルで sharable が Ture の場合，dejima_id, vid, location, rid の情報が伝播する．しかし，False の場合，情報は書き込まれないため更新されない．更新された情報は proxy でトランザクション処理を行い，他のピアに更新伝播される.

2 通りのサーバ配置に INSERT するデータが 1 行，10 行，100 行，1000 行のそれぞれについて，更新伝播のある場合とない場合の計 16 通り実行し，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．以下に使用した問合せを示す.

- 更新伝播がありでデータ挿入が 1000 行の場合

```
INSERT INTO bt (dejima_id,vid,location,rid,sharable)
VALUES (101,101,'A',100,'True'),
(102,102,'B',100,'True'),
        ・
        ・
```

.
(1099,1099,'C',100,'True'),
(1100,1100,'D',100,'True');

- 更新伝播がなしでデータ挿入が 1000 行の場合

INSERT INTO bt (dejima_id,vid,location,rid,sharable)
VALUES (101,101,'A',100,'False'),
(102,102,'B',100,'False'),
.
.
.
(1099,1099,'C',100,'False'),
(1100,1100,'D',100,'False');

6.2 評価結果

16 通りそれぞれについて 5 回ずつ試行した平均の処理時間を求め，パターン 1 に対するパターン 2 の処理時間の比について比較を行った．その処理時間の比について表 2 に示す．表 2 より，更新伝播がある場合は 1 行のとき 1.3 倍，10 行のとき 2.8 倍，100 行のとき 9.0 倍，1000 倍のとき 13.0 倍になった．更新伝播がない場合は 1 行のとき 5.4 倍，10 行のとき 9.6 倍，100 行のとき 17.0 倍，1000 行のとき 17.6 倍になった．更新伝播の有無に関わらず，パターン 1 に対するパターン 2 の処理時間の比は行数が増えると大きくなり，10 行から 100 行になったときが最も比は増加したことが分かった．また，1000 行のとき更新伝播の有無に関わらず 10 倍以上の差があることを確認した．Dejima アーキテクチャの proxy の情報のやり取りは，ユーザとベーステーブル間のやり取りに比べてデータ量がコンパクトであるためと推測できる．上記の実験から，Dejima アーキテクチャは PostgreSQL の INSERT 文のみの問合せを 2 人だけの情報共有で行った場合は，エッジコンピューティングの観点からサーバ配置を行った方が適していることが分かった．

7 まとめ

本研究では，エッジコンピューティングとクラウドコンピューティングの観点から, Dejima アーキテクチャのサーバ配置を 2 つのパターンに分けて行った．そして，それらの処理時間について比較し評価を行った．その結果, 更新伝播の有無に関わらず，パターン 1 に対するパターン 2 の処理時間の比は行数が増えていくたびに大きくなった．また, PostgreSQL の INSERT 文のみの問合せを 2 人だけの情報共有で行った場合，1000 行のときパターン 1 に対するパターン 2 の処理時間の比は，更新伝播の有無に関わらず 10 倍以上の差があったため, エッジコンピューティングの観点からサーバ配置を行った方が適していることが分かった．

しかし，情報共有を行う人数が 3 人，4 人となったとき評価結果が大きく変わる可能性がある．その理由として，パターン 2 のサーバ配置において両ユーザとベーステーブルの間にレンタルサーバを接続したサーバ配置で実験を行った場合，パターン 2 のサーバ配置の処理時間とほとんど変わらない結果になったことが挙げられる．このことから, 人数が 3 人，4 人になる場合もパターン 2 の処理時間は 2 人の場合と比べてあまり変化しないと予想される．一方，パターン 1 のサーバ配置は，proxy の間にレンタルサーバを接続するので人数が増えた分だけ処理時間が長くなることが予想される．以上のことから，情報共有する人数が 3 人，4 人と増えていったときにデータの更新伝播をする場合, 評価結果が大きく変わる可能性があるので，パターン 1 とパターン 2 のどちらのサーバ配置がどれくらい処理時間に差があるのか実験する必要がある．また，レンタルサーバ上で遅延時間を制御できるようにし，遅延時間と処理時間についてまとめる必要がある．

表 2 パターン 1 に対するパターン 2 の処理時間の比

	1 行	10 行	100 行	1000 行
更新伝播がある	1.3 倍	2.8 倍	9.0 倍	13.0 倍
更新伝播がない	5.4 倍	9.6 倍	17.0 倍	17.6 倍

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] 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.

[3] Kouta Miyake. ekayim/dejima-prototype. `https://github.com/ekayim/dejima-prototype.git`.

[4] 今金健太郎, 金井謙治, 甲藤二郎. マルチメディア処理におけるエッジコンピューティングの特性評価. 映像情報メディア学会冬季大会講演予稿集, 2015. B-3 21.

[5] 三宅康太, 佐々木勇和, 肖川, 鬼塚真. 統合型データベースにおける適応的 2 相ロックに基づく分散トランザクション制御. データ工学と情報マネジメントに関するフォーラム, 2022. J34-3.

[6] 中西遼, 中尾瑠衣. 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. 南山大学理工学部機械電子制御工学科卒業論文, 2022.