Oral Presentations | Track 2: Big Data Infrastructure Technologies, Security & Privacy

Wed. Feb 28, 2024 4:00 PM - 6:10 PM JST | Wed. Feb 28, 2024 7:00 AM - 9:10 AM UTC | T2-A オンライン（Zoom Events)

Parallel processing / distributed database

座長:合田 和生(東京大学)
コメンテータ:金政 泰彦(富士通株式会社)

4:00 PM - 4:05 PM JST | 7:00 AM - 7:05 AM UTC

Introduction

4:05 PM - 4:30 PM JST | 7:05 AM - 7:30 AM UTC

[T2-A-3-01] データ検証可能な分散DB実現手法の提案と実現に向けたHyperledger Irohaのコマンド開発

*堀 遥[1]、小口 正人[1] (1. お茶の水女子大学)

4:30 PM - 4:55 PM JST | 7:30 AM - 7:55 AM UTC

[T2-A-3-02] SparkSQLによる効率的な問合せ処理のためのワークロードに基づくRDFデータ分割手法

*山﨑 昂輔[1]、天笠 俊之[2] (1. 筑波大学大学院理工情報生命学術院システム情報工学研究群情報理工学位プログラム 知識・データ工学研究室、2. 筑波大学計算科学研究センター)

4:55 PM - 5:20 PM JST | 7:55 AM - 8:20 AM UTC

[T2-A-3-03] Fast Parallel Computation for Random Walk based t-SNE

*三木 武志[1]、藤原 靖宏[2]、川島 英之[1] (1. 慶應義塾大学、2. 日本電信電話株式会社)

5:20 PM - 5:45 PM JST | 8:20 AM - 8:45 AM UTC

[T2-A-3-04] ブロックチェーン技術を活用したデータ検証可能な分散データベースの性能に関する検討

*坂本 明穂[1]、小口 正人[2] (1. お茶の水女子大学理学部情報科学科小口研究室、2. お茶の水女子大学基幹研究院自然科学系)

5:45 PM - 6:10 PM JST | 8:45 AM - 9:10 AM UTC

[T2-A-3-05] 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[1]、立床 雅司[1]、柴田 秀哉[1] (1. 三菱電機株式会社 情報技術総合研究所 データマネジメント技術部 データ最適化技術グループ)

DEIM Forum 2024 XX-Y

データ検証可能な分散DB実現手法の提案と実現に向けたHyperledger Irohaのコマンド開発

堀　遥† 小口　正人†

† お茶の水女子大学　〒112–8610 東京都文京区大塚 2–1–1
E-mail: †haruka-h@ogl.is.ocha.ac.jp, ††oguchi@is.ocha.ac.jp

あらまし　データ駆動型社会の実現にむけ，信頼度などに異種性を持つデータを一元的に取り扱う分散データベースシステムの開発が求められる．分散環境において，格納データの安全性を保証するためには，全てのデータが検証可能であることを保証すれば良い．すなわち，データ検証機能を有し，かつ，ユーザが安全性に欠けると判断されたデータを利用する際には，その利用可否を直接問う仕組みを実装することが要求される．我々はこのようなデータ管理基盤の開発を目指し，少ないリソースでの高速動作と高い開発の自由度が特徴のブロックチェーン基盤ソフトウェアのHyperledger Irohaをプラットフォームとした分散データベースシステムを検討した．また，このシステムの実装の一部として，検証対象となるデータの生成・格納処理を行うHyperledger Irohaの新たな組み込みコマンドの開発に着手した．

キーワード　クラウド基盤、ブロックチェーン技術、データ検証技術

1 はじめに

1.1 研究背景

近年，スマートフォンやIoTデバイスの普及から膨大なデータが収集されるようになった．DXの進展に伴うデータ駆動型社会の実現に向け，このような実社会の多様なデータをその不確実性を考慮しながら活用するような新しいデータ管理基盤の実現が期待されている．このとき，信頼度や来歴，種類などに異種性のあるデータを一元的に保存し，データ利用の際にはその信頼性の保証に貢献するような分散データベースシステムが考えられる．一方で，そのようなデータ管理基盤では，複数のユーザがアクセスできるオープンな環境において，これらのヘテロなデータの健全性・安全性の保証が課題となる．これに対し，収集・保管された全ての有効なデータが検証可能であることを保証する機能を分散データベースシステムに取り入れるというアプローチが考えられる．

1.2 既存研究

嘉戸らの研究[16]では，クラウドストレージシステムに保存されたデータの完全性検証へのアプローチとして，Private PDPを採用している．PDPはデータを複数のブロックに分割し，その中からランダムに選んだブロックに対してデータの完全性検証を行う．特にPrivate PDPでは，データ保有者とクラウドサービスプロバイダのみで完全性検証を行う方式である．Private PDPにブロックチェーン技術の一つであるスマートコントラクトを応用し，データの完全性検証および完全性検証結果不一致時の合意形成を行う方法を提案している．

また，Neha Mishra らの研究[7]では，PDV：Personal Data Vaultと呼ばれる個人の生涯にわたるデジタルドキュメントを，検証可能かつ安全な方法で保管，保存，保護，共有するためのフレームワークの開発にブロックチェーンプラットフォームHyperledgr Irohaを採用している．各文書は暗号化，圧縮され，クラウドに安全に保存され，文書の保存先URLがHyperledger Irohaに入力される．また，開発システムはMarkov Treeを用いた予測プリフェッチ機能を備えており，次に発生する要求を予測，事前実行する．

いずれの場合においても，現代で有用なデータ管理基盤において，データの安全性を保証する際にはブロックチェーンの活用することが効果的であることがわかる．

1.3 研究目的

本研究では，任意のデータが検証可能であることを保証する機能を持つ分散データベースシステムの開発することを目的とする．この目的を達成するため，まずメタデータをブロックチェーンプラットフォームHyperledger Irohaに付属するオンチェーンデータベースにて管理する．オフチェーンデータベースへのデータの追加や更新などが発生し，新規メタデータが必要となる場合，一連の処理をHyperledger Irohaの組み込みコマンドで行う．コマンドの結果はブロックチェーンに記録され，これによりメタデータの安全性が確保される．そして，安全性の保証されたメタデータ，蓄積されたブロック，Hyperledger Irohaのブロック検証機能を利用し，データの検証可能性を保証する．最終的には，データ検証の結果，安全性が保証されないと判断されたデータを利用する際には，ユーザに対象のデータの利用可否を問うシステムの実現を目指す．

1.4 実証実験

脱炭素を実現する循環型社会への貢献を期待し，製造業におけるカーボンフットプリント管理を対象とした実証実験を行う．

表 1　ブロックチェーンの適用事例

業界	企業名	利用目的
金融	三菱 UFJ ファイナンシャルグループ	独自のデジタル通貨管理 [14]
不動産業	積水ハウス株式会社	物件情報と不動産賃貸契約の実行 [15]
インフラ (電力取引)	国立大学法人東京大学 トヨタ自動車株式会社 TRENDE 株式会社	分散型電源を活用した電力の個人間売買システムを検証 [11]
医療	Z.com Cloud ブロックチェーン	医療カルテの共有 [5]
コンテンツ業	ソニー株式会社 株式会社ソニー・ミュージックエンタテイメント 株式会社ソニー・グローバルエデュケーション	デジタルコンテンツに関わる権利情報を処理する機能 [12]

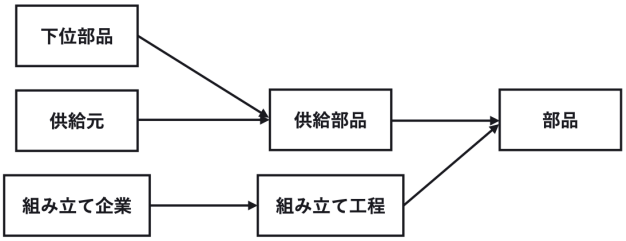


図 1　カーボンフットプリント応用のスキーマ

複数企業を接続する分散データベース上で製造工程全体のトレーサビリティを実現し，センサなどのクライアントデバイスから収集した CO_2 排出量データのメタデータを Hyperledger Iroha で管理する．これにより，収集した CO_2 排出量の検証可能性が保証され，カーボンフットプリント削減に向けた安全・安心なデータ分析・予測を支援する．

本稿では，自動車製造に関わる企業のうち部品製造・組み立て工場にフォーカスし，これらの工場間における製造時・組み立て時のカーボンフットプリント管理を想定する．図 1 は，カーボンフットプリント応用におけるスキーマであり，供給元から下位部品が供給され，組み立て工場による組み立て工程を経て，部品を構築する．本スキーマは各工場が管理しており，下位部品から上位部品を階層的に組み立てて，複数企業を横断して最上位部品の自動車を製造する．

一台の自動車には約 3 万点ほどの部品が使用されるとされている [9]．おおよその国内の自動車部品メーカーと呼ばれる企業数，過去 10 年間で発売された新車数，そしてこれらの重複などを考慮すると，1 車種あたり 100 社程度が扱う 3 万点もの部品を分散管理できるシステムが必要となることが考えられる．

ここで，スキーマに従い，部品の組み立て工場で排出された CO_2 排出量を *EMISSIONS* と呼ぶことにする．また，各部品は階層関係にあり，CO_2 排出量もまた階層関係にある．下位部品の *EMISSIONS* を再帰的に足し合わせ，部品自体の *EMISSIONS* と合計したものを *TotalEMISSIONS* と呼ぶ．本稿では各部品の *TotalEMISSIONS* を検証対象データであることを前提にシステムの実装手法を行う．

1.5　本稿の構成

本稿は以下の通り構成される．第 2 章では本研究の前提となるブロックチェーン及び Hyperledger Iroha の概要を説明する．第 3 章では本研究の提案手法の概要を，第 4 章で提案手法の実装に向けた計画を説明する．最後に第 5 章でまとめ，今後の課題を述べる．

2　関 連 研 究

2.1　ブロックチェーン

ブロックチェーンは，2008 年に Satoshi Nakamoto により投稿された論文 [8] に基づき，暗号通貨 Bitcoin [2] の公開取引台帳としての役割を果たすために発明された．Bitcoin の成功によりブロックチェーンは仮想通貨技術として強く注目されたが，一般社団法人日本ブロックチェーン協会は広義のブロックチェーンを「電子署名とハッシュポインタを使用し改竄検出が容易なデータ構造を持ち，且つ，当該データをネットワーク上に分散する多数のノードに保持させることで，高可用性及びデータ同一性等を実現する技術」と定義している [1]．

現在では仮想通貨以外での用途も促進されており，表 1 は書籍 [13] で紹介される例を一部抜粋したものである．例えば不動産業では，煩雑で膨大な量の手続きによって浪費されるコストと時間を，ブロックチェーン活用によって削減し，より効率的にさせている．また，医療では，ブロックチェーンに備わる権限機構と威厳性を活用し，医療機関ごとに管理されているカルテをブロックチェーンで一元的に管理して，より安全に利便性を高めるといった効果をもたらしている．

2.1.1　技術的概要

ブロックチェーンは，ブロックと呼ばれるトランザクションを記録する単位を生成し，これをチェーンのように連鎖するデータ構造である．仮想通貨の基礎技術として開発されたブロックチェーンの厳格性を担保する仕組みはチェーンにある．一つのブロックにはハッシュ値が付与される．このハッシュ値は，前のブロックのハッシュ値と新規のトランザクションの内容やタイムスタンプなどから算出される．すなわち，ハッシュ値によりブロックの関連が生まれ，これがチェーンとなる．よってブロックチェーンは高い改ざん耐性を持つと言われる．また，Peer to Peer 型のブロックチェーンネットワーク上に参加する各ノードがブロックチェーンを管理するるため，欠損ブロックを他のノードから補うことができ，耐障害性と可用性に優れるといった特徴を持つ．

表 2　ブロックチェーンと集中型データベースの比較

比較項目	ブロックチェーン	中央型データベース
A.　構築の信頼性	管理者が必要ない場合がある	中央管理者が必要
B.　データの機密性	全ノードがデータ参照可能	関係者のみにアクセスを制限
C.　堅牢性& 耐障害性	データはノード間で分散される	データは中央データベースで保管
D.　パフォーマンス	合意形成に時間を要す (Bitcoin では約 10 分)	即時に実行/アップデート
E.　冗長性	(デフォルトでは) 参加する各ノードが最新のコピーを保持	管理者のみがコピーを保持
F.　セキュリティ	(デフォルトでは) 暗号化メカニズム使用	従来のアクセスコントロール

2.1.2　従来のデータベース技術との比較

M. J. M. Chowdhury らの論文 [3] では，ブロックチェーンと集中型データベースの各アプローチを表 2 のように挙げている．また，これら 6 つの比較項目に対しする比較分析から，データの信頼性，堅牢性，実証性がシステムの優先事項であれば，ブロックチェーンがより優れており，機密性とパフォーマンスが優先であれば，従来のデータベースがより優れたソリューションであるとされている．

以上より，ブロックチェーンとデータベースの両方の機能を持つことで，アプリケーションは効率性と安全性を高めることができるため，ブロックチェーンプラットフォームの多くはデータベースと統合されている．

2.1.3　ブロックチェーンのモデル

ブロックチェーンは「パーミッションレス型」,「パーミッション型」,「コンソーシアム型」という 3 種類のモデルに分別される．表 3 はそれぞれのモデルの特徴である．

Bitcoin や Etherum [4] に代表されるパーミッションレス型は非中央集権でマイニングと呼ばれる膨大な計算の承認によって取引の正当性を担保する．透明性の高さを持つ反面，時間とリソースのコストが高く，データのプライバシーは保証されない．

パーミッション型は，中央集権型のネットワークで，透明性はないもののプライバシーが確保される上に，マイニングも行う必要がないため，高いパフォーマンスを持つ．こうした特徴から，プライベート型は単一の企業や組織内での運用に有効であり，特に銀行間の取引や証券取引での活用が促進されている．次の節で説明する Hyperledger の数々のプロジェクトはパーミッション型である．

最後のコンソーシアム型には複数の管理者が存在している．そのため，パーミッションレス型の分散性とパーミッション型の迅速な大量処理が可能という機能を備えている．また，管理者が複数存在しているため，運用ルールの変更についても一定数以上の合意が必要となる．これにより，高い公平性が保たれ，セキュリティや耐障害性もプライベート型に比べると強固であるとされる．こうした特徴から，コンソーシアム型は同業他社が協力して構築するブロックチェーンシステムに有効であり，こちらも銀行間の取引や証券取引での活用が促進されている．

2.1.4　コンセンサスアルゴリズム

ブロックチェーンを構成する技術の中に「コンセンサスアルゴリズム」がある．コンセンサスアルゴリズムとは，生成ブロックをブロックチェーンに追加する前に各ノード間で合意形成を行うメカニズムである．表 4 は代表的なコンセンサスアルゴリズムである．本研究で使用する Hyperledger Iroha では効率的に少ない Peer 数で承認を得る YAC 方式をとる．

表 3　ブロックチェーンの種類

名称	管理者	参加者
パーミッションレス型	なし	制限なし
パーミッション型	複数存在	個人や単一の組織
コンソーシアム型	複数存在	グループや複数の組織

表 4　ブロックチェーンの代表的なコンセンサスアルゴリズム

名称	採用システム	決定方法
PoW	Bitcoin Ethereum	正解を最も早く算出した Peer を採用
PoS	Ethereum で検討	資産の保有量に応じて 有利な条件で PoW を実施
PBFT	Hyperledger Fabric	検証を行った Peer の 過半数から承認を得る
YAC	Hyperledger Iroha	PBFT より効率的に少ない Peer 数で承認を得る

2.2　Hyperledger Iroha

Hyperledger [6] は 2015 年 12 月に The Linux Foundation よって開始され，ブロックチェーンベースの分散台帳をサポートするプロジェクトである．仮想通貨や金融，サプライチェーンに限らず，広範囲なビジネスシーンへのブロックチェーン基盤提供し，パフォーマンスや信頼性などの多方面においてサポート・改善ことを目指す．

Hyperledger では方向性の異なる複数のプロジェクトが推進されており，GA リリースされたプロジェクトは 2024 年 1 月時点で 4 つとなる．Hyperledger Iroha は 2019 年 5 月に GA リリースされたパーミッション型ブロックチェーンプラットフォームである．ソラミツ株式会社が初期開発者として継続的に開発に貢献している．

2.2.1　特　　徴

Hyperledger Iroha の主な特徴として「簡単な導入とメンテナンス」「開発に向けたさまざまなライブラリ」「役割に基づいたアクセス制限」「コマンドとクエリの分離によって行われるモジュール型設計」「資産とアイデンティティ管理」の 5 点が挙げられる．

- 簡単な導入とメンテナンス

Hyperledger Iroha の導入は非常に容易である．Docker 環境を利用したテスト環境であれば数十分で終了する．また，全ての Hyperledger プロジェクトはオープンソースとしてソースコー

ドが公開されている．

- 開発に向けたさまざまなライブラリ

Hyperledger Iroha は Java，JavaScript，Swift，Python の 4 つのプログラミング言語に対する API を備える．API は統一されておりどのプログラミング言語からでも同様の操作を実現できる．そのため，開発者のニーズに合わせた言語選択が可能となる．

- 役割に基づいたアクセス制限

仮想通貨では権限の概念が軽薄であり，これはネットワークにアクセス可能なユーザに対して全ての資源に対するアクセス権を付与する状況にある．Hyperledger Iroha では，ドメインと呼ばれる権限が及ぶ範囲を表す概念，ロールと呼ばれる複数の権限を 1 つに集約する概念が実装されている．これにより，広範囲な用途に対して，汎用的かつ柔軟的に対応することが可能である．

- コマンドとクエリの分離によって行われるモジュール型設計

Hyperledger Iroha の API 操作は，Hyperledger Iroha に対し変化をもたらすコマンドと，Hyperledger Iroha は普遍のまま現在持つ情報を表示するクエリの 2 種類に分けられる．コマンドの実行結果は，トランザクションとしてブロックチェーンに記録される．これらの 2 種類の API を組み合わせ，システムの開発が可能である．

- 資産とアイデンティティ管理

仮想通貨では通常，単一の仮想通貨を扱うが，Hyperledger Iroha では複数の通貨をアセットという概念で実装されている．アセットは作成時にドメインを指定する必要がある．

また，品質モデルにおいては次の 3 点が挙げられる．

- 信頼性

耐障害性，回復性．

- パフォーマンス効率

とりわけ時間挙動とリソースの使用効率．鍵生成アルゴリズムに安全性とパフォーマンスに優れた ED25519 を，コンセンサスアルゴリズムにより高速な YAC 方式を採用している点も効果的となっている．

- ユーザビリティ

学習可能性，ユーザエラー保護，妥当性の評価可能性．

以上の特徴を踏まえ，Hyperledger Iroha は Hyperledger プロジェクトの中でも，最も容易で短時間にブロックチェーンによる分散台帳を実現することができ，かつライトウェイトであるとされている．また，豊富な API や柔軟な概念構成から高い自由度の開発環境が提供されている．

2.2.2 ブロック検証機能

Hyperledger Iroha は起動時に全ブロックのハッシュ値の再計算を行い，ブロックを検証する．もし改ざんが発生した場合どのような挙動を示すか，ブロック欠損時，ブロックチェーン改ざん時について説明する．

まず過去ブロックの欠損時，起動時のハッシュ値再計算にてブロックの欠損を知らせるエラーメッセージが出力され，Hyperledger Iroha は起動しなくなる．複数 Peer 構成で運用する場合は，自動的に他の Peer から欠損ブロックが読み込まれる．

次にブロック情報の改ざん時，ブロック中のトランザクション処理内容であれば，ハッシュ値に変化が生じる．そのため，他の Peer とのハッシュ値の比較により改ざんが検知される．そしてブロックの作成者，すなわち電子署名が改ざんされれば，ブロック欠損時同様に Hyperledger Iroha は起動せず，改ざんが検知される．

本研究では，Hyperledger Iroha のブロック検証機能を利用し，データ検証メカニズムを実現することを検討している．

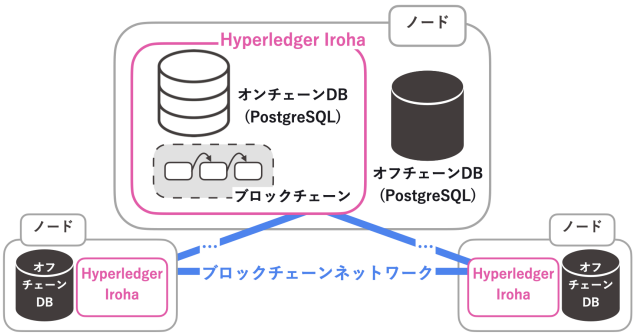


図 2　システム全体の構成

3 提案手法

本章では，提案手法を提案モデル，提案手法の手順，カーボンフットプリント応用における動作シナリオに分けて詳しく説明する．

3.1 提案モデル

まず本稿における提案モデルのシステム全体の構成を図 2 に示す．各ノードは Hyperledger Iroha とオフチェーンデータベースを備える．カーボンフットプリント応用の場合，ノードは企業と捉えてよく，全てのノードが Hyperledger Iroha によって構成されるブロックチェーンネットワークで接続される．オフチェーンデータベースと Hyperledger Iroha のオンチェーンデータベースはともに，オープンソースのリレーショナルデータベース管理システムである PostgreSQL [10] にて実装する．次の項で，各データベースの詳細を述べる．

3.1.1 オフチェーンデータベース

オフチェーンデータベースはノードに固有のデータベースである．主に各ノードで製造に関わっている部品のデータを管理する役割を持ち，検証対象データはオフチェーンデータベースに保管される．

表 5 は，オフチェーンデータベース上で管理するテーブルの例である．テーブル CO2Emissions は，カーボンフットプリント応用において現在想定している，検証対象データ *TotalEMISSIONS* を管理するテーブルであり，PartsID は部品の ID，部品の *EMISSIONS*，*TotalEMISSIONS*，時刻を表す TimeStamp を属性としてもつ．プライマリーキーとして PartsID と TimeStamp を指定している．ノードで新規 *EMISSIONS* が得られると，Hyperledger Iroha 上で *TotalEMISSIONS* が算出され，

表 5　テーブル CO2Emissions

CO2Emissions	
PK	PartsID
	EMISSIONS
	TotalEMISSIONS
PK	TimeStamp

表 6　テーブル Metadata

Metadata	
PK	PartsID
	Link
	ChildPartsID
	TimeStamp

これらの値がタイムスタンプとともにテーブル CO2Emissions に追加される．

上記の例以外にも，部品の基本情報や，ノードに関連する情報などもここで管理される．

3.1.2　オンチェーンデータベース

Hyperledger Iroha のオンチェーンデータベースは World State View とも呼ばれ，Hyperledger Iroha の最新情報が格納される．すなわち，ブロックチェーンネットワークに参加する全ノードでこの内容が同一となる．ここで，ブロックチェーン上のブロックそのものはオンチェーンデータベースには保存されず，json ファイルで生成されて別の場所で保管されている．

提案システムでは，各ノードのオフチェーンデータベースにて保管される全ての検証対象データに対し，メタデータを作成する．全部品のメタデータは，オンチェーンデータベース上に存在するテーブル Metadata に格納・管理される．表 6 は，カーボンフットプリント応用におけるテーブル Metadata である．PartsID は部品の ID，Link は表 5 上に格納される *TotalEMISSIONS* のリンク情報，ChildPartsID は下位部品の PartsID，TimeStamp は時刻を表す．プライマリーキーとして PartsID のみを指定しているため，新規メタデータ作成のたびにテーブル Metadata にデータを追加するのではなく，対象部品のカラムを更新する．1.4 節でも触れたように，開発システムで扱う部品数は約 3 万点と推測されるため，テーブル Metadata では全部品，おおよそ 3 万もの最新のメタデータが保管されることとなる．

このように，実データでなくメタデータをブロックチェーンで管理することで，実データのデータ形式に捉われず，また，その設計次第でデータのサイズを抑えることが可能となる．

3.2　提案手法の手順

本説では，提案手法の検証対象データの格納時，データの参照時の手順を説明する．その後，カーボンフットプリント応用における動作シナリオを説明する．

3.2.1　検証対象データの格納時

まず，検証対象データ格納時の手順を説明する．

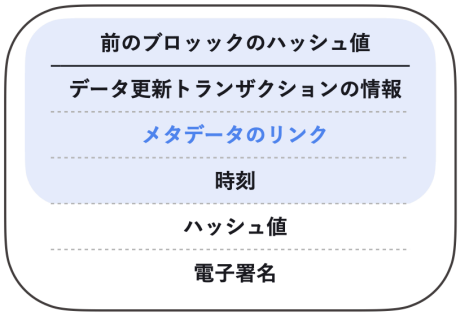


図 3　ブロック構成の概要

(1)　データ更新トランザクションがトリガーされる．
　　トリガーの条件は応用システムにより様々である．

(2)　データ更新トランザクションの実行．

　(2-1)　必要に応じたデータの加工処理などの実施．
　(2-2)　オフチェーンデータベースに格納．

(3)　(2-2) の格納場所情報 (Link) を含めて新規メタデータ作成．

(4)　(3) のメタデータをテーブル Metadata に格納．

(5)　ブロック作成，追加．

データ更新トランザクションでは，まず検証対象データ追加・更新のために必要な一連のデータ加工処理や演算を行う．これにより新しく得た検証対象データをオフチェーンデータベースに格納する．データ更新トランザクションの実装は，Hyperledger Iroha の組み込みコマンドとして開発を行う．

図 3 は (5) で作成，追加されるブロックの構成概要である．メタデータのリンクとは，テーブル Metadata のインデックスやプライマリーキーに該当するデータを表す．前のブロックのハッシュ値，更新トランザクションの情報，メタデータのリンク，時刻情報から，ブロックのハッシュ値は生成される．

以上の手順で検証対象データを格納することにより，メタデータの厳格性が担保される．

3.2.2　データ参照

データ参照時の手順は次の通りである．

(1)　メタデータを参照し，Link を取得．

(2)　(1) からオフチェーンデータベースにアクセスし，データを参照．

3.2.3　実証実験におけるデータ更新トランザクション

カーボンフットプリント応用では，*TotalEMISSIONS* を検証対象データとする．*TotalEMISSIONS* は，下位部品の *TotalEMISSIONS* の合計値と *EMISSIONS* を加算することで求まる．

本稿では，データ更新トランザクションのトリガー条件を新

規 *EMISSIONS* の取得とする．*EMISSIONS* の新しい値が得られた部品の *TotalEMISSIONS* を更新するため，データ更新トランザクションが実行される．データ更新トランザクションの手順は次の通りである．

(1) テーブル Metadata を参照し，対象部品の下位部品の PartsID を取得.

(2) テーブル Metadata で (1) の PartsID を検索し，Link を取得.

(3) Link からオフチェーンデータベース上の下位部品の *TotalEMISSIONS* を取得.
これを対象部品の全下位部品で行う.

(4) (3) で取得した *TotalEMISSIONS* らを合算する.

(5) (4) で算出した値に対象部品の新規 *EMISSIONS* を加算し，これが新規 *TotalEMISSIONS* となる.

(6) (5) をオフチェーンデータベースに格納.

データ更新トランザクションとその後のメタデータ生成・更新の一連の操作は，Hyperledger Iroha のコマンドにより実装する．また，ブロックに含まれるメタデータのリンクは常に対象部品の PartsID となる．

4 提案手法の実装に向けた計画

提案手法実装のため，第一にデータ検証トランザクションを実装する．具体的には，データ更新トランザクションの処理を提供する Hyperledger Iroha の組み込みコマンドの作成である．そのために Hyperledger Iroha の構成ファイルの開発が必要となる．

データ更新トランザクションを実現に向け，そのベースとなるコマンド *Update_testTable* の開発を行なう．開発環境，コマンド Insert の概要の二点について以降の節で説明する．

4.1 開 発 環 境

表 7 は実装環境の詳細である．仮想環境として Docker を使用し，Ubuntu22.04LST コンテナ上に Hyperledger Iroha の動作環境をビルドする．PostgreSQL コンテナでは，オンチェーン DB とオンチェーン DB としての機能を提供する．表 6 に示したテーブル Metadata は PostgreSQL コンテナ上で動作するデータベース上に存在する．

4.2 コマンド *Update_testTable* の概要

コマンド *Update_testTable* は，Hyperledger Iroha 上のオンチェーンデータベースに保管されるテーブル testTable に対して，UPDATE 命令を実行するコマンドである．表 8 にテーブル testTable の詳細を示す.

コマンド *Update_testTable* では，属性 Val に格納される値の更新を行う．実行の際には，プライマリーキーの ID と Val の更新値を与え，これらを持ってテーブル testTable に対して，UPDATE 命令を実行する.

表 7 実 装 環 境

コンテナ	Hyperledger Iroha ver.1	オン/オフチェーン DB
	Ubuntu イメージ：Ubuntu:22.04	PostgreSQL イメージ：postgres
仮想環境	Docker	
OS	Ubuntu20.04LTS	
サーバ	CPU：Intel(R) Xeon(R) Silver 4314 CPU @ 2.40GHz メモリ：192GB	

表 8 テーブル testTable

testTable	
PK	ID
	Val
	TimeStamp

コマンド *Update_testTable* 開発おける目標として，以下の 3 点が挙げられる.

- コマンド実行結果がテーブル testTable に反映されている.
- コンセンサスアルゴリズムを経てブロック作成がアクセプトされている.
- 作成されたブロックにコマンド実行情報が記録されている.

5 まとめと今後の課題

本稿では，信頼度などに異種性を持つヘテロなデータを一元的に保管・利用するための分散データベースシステムの実現を目標とし，提案システムの検討を行った．このような，分散環境においてデータの安全性を保証するため，本稿では任意のデータに対するデータ検証機能をシステムに組み込むというアプローチを採用した．データ検証機能実現に向け，システムのプラットフォームにはブロックチェーン基盤ソフトウェアの Hyperledger Iroha を用い，Hyperledger Iroha にて検証対象データのメタデータを管理するためのメタデータ設計も行った．また，実装の一部として，Hyperledger Iroha の組み込みコマンドの開発に着手した.

今後は，第一にデータ更新トランザクション実装に向けた，コマンド *Update_testTable* の開発を行う．コマンド *Update_testTable* はオンチェーンデータベース内に閉じた処理であるため，これの開発が完了次第，Hyperledger Iroha からオフチェーンデータベースへのアクセス手法の検討・および実装に取り掛かる．その後は，提案手法におけるブロック構成と検証対象データ格納シナリオを順に実装する予定である.

また，Hyperledger Iroha のブロック検証機能の有用性についても調査を行なっていく.

謝　　辞

本研究は一部，JST CREST JPMJCR22M2 の支援を受けたものである．

文　　献

[1] Japan Blockchain Association. ブロックチェーンの定義. https://jba-web.jp/news/642.
[2] Bitcoin. https://bitcoin.org/ja/.
[3] Mohammad Jabed Morshed Chowdhury, Alan Colman, Muhammad Ashad Kabir, Jun Han, and Paul Sarda. Blockchain versus database: A critical analysis. In *2018 IEEE TrustCom/BigDataSE*, pp. 1348–1353, 2018.
[4] Ethereum. https://ethereum.org/ja/.
[5] GMO インターネット株式会社. 医療機関カルテ共有システム. https://www.gmo.jp/news/article/5736/.
[6] Hyperledger foundation. https://www.hyperledger.org/.
[7] Neha Mishra and Dr. Haim Levkowitz. Pdv: Permissioned blockchain based personal data vault using predictive prefetching. In *BIOTC '21: Proceedings of the 2021 3rd Blockchain and Internet of Things Conference*, pp. 59–69.
[8] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.
[9] Toyota クルマ教室. https://global.toyota/jp/kids/faq/parts/001.html.
[10] Postgresql: The world's most advanced open source relational database. https://www.postgresql.org/.
[11] TOYOTA 自動車株式会社. 東京大学、トヨタ、trende が、次世代電力システムの共同実証実験を開始. https://global.toyota/jp/newsroom/corporate/28227543.html.
[12] ソニー株式会社, 株式会社ソニー・ミュージックエンタテインメント, 株式会社ソニー・グローバルエデュケーション. ブロックチェーン基盤を活用したデジタルコンテンツの権利情報処理システムを開発. https://www.sony.com/ja/SonyInfo/News/Press/201810/18-1015/.
[13] 佐藤栄一. Hyperledger Iroha 入門 -ブロックチェーンの導入と運用管理-. 株式会社 オーム社, 2020.
[14] 三菱 UFJ フィナンシャル・グループ. Mufg におけるブロックチェーンの取組み. https://www.boj.or.jp/paym/fintech/data/rel180214a6.pdf.
[15] 積水ハウス株式会社. 積水ハウス 業界の新たなスタンダード構築へ 業界初 ブロックチェーンで賃貸入居の煩雑なプロセスをワンストップ化. https://www.sekisuihouse.co.jp/library/company/topics/datail/__icsFiles/afieldfile/2020/06/08/20200608.pdf.
[16] 嘉戸裕一, 廣友雅徳, 瀧田愼, 掛井将平, 白石善明, 毛利公美, 森井昌克. ブロックチェーンを用いたクラウドストレージのプライベートな完全性検証方式. In *Computer Security Symposium 2023*, pp. 727–734, 2023.

Spark SQLによる効率的な問合せ処理のためのワークロードに基づくRDFデータ分割手法

山崎　昂輔†　　天笠　俊之††

† 筑波大学院 理工情報生命学術院 システム情報工学研究群 情報理工学位プログラム　〒305-8573 茨城県つくば市天王台1丁目1-1

†† 筑波大学 計算科学研究センター　〒305–8577 茨城県つくば市天王台1丁目1-1

E-mail: †kosuke.y@kde.cs.tsukuba.ac.jp, ††amagasa@cs.tsukuba.ac.jp

あらまし　近年，機械可読なデータとしてRDF（Resource Description Framework）が注目されている．RDFは主語，述語，目的語のトリプルの組み合わせで多くのデータの表現を可能とする単純で柔軟なフレームワークであり，さまざまな領域でRDFデータが生成され，そのデータ量は急速に増加している．膨大なRDFデータを効率的に処理するために多くの研究が行われており，その中でも並列分散処理による高速化が期待されている．Apache Sparkは，インメモリでデータを処理することで高速化を実現するフレームワークであり，Spark SQLを用いることでSQLを用いたRDFデータの処理が可能となる． 本研究では，ワークロードの共起性を用いたデータ分割手法と，Spark SQLを用いたRDFデータの効率的な問合せ処理を提案する．ワークロード情報を活用することにより問合せ時の関連が強いデータを同一パーティションに配置することが期待でき，処理対象のデータ量の削減を実現できる．実行時間の比較では，多くのクエリで従来手法を上回る結果を示した．

キーワード　RDF, パーティショニング, 分散処理, Apache Spark

1　はじめに

RDF（Resource Description Framework）[1]は，主語，述語，目的語からなるトリプルと呼ばれる基本構造を持ち，これによってデータ間の関係を簡潔に表現できるフレームワークである．このフレームワークはデータへのセマンティクスの付与やグラフ表現が可能なこと，機械可読であるなどの特徴があり，その有用性から近年RDF形式のデータが爆発的に増加している．RDFデータは基本的にSPARQL（SPARQL Protocol and RDF Query Language）という標準的な問合せ言語を用いてデータの問合せを行う．SPARQLでは，問合せたいデータをトリプル構造に基づいて記述することで問合せを実行することができる．問合せにかかるコストはデータ量の増加に伴って大きくなるが，一般に大規模なRDFデータは数百万から数億のトリプルを含んでいる．従って，そのようなデータに対していかに効率的で高速な問合せ処理を実現するかが昨今の課題となっている．

この課題を解決するために，大きく分けて集中型システムと分散型システムの2種類が多数発表されている．集中型システムは，データの保管や問合せ処理を1つのマシンで管理するものであり，Jena [2] やRDF-3X [3] などが挙げられる．一般に集中型システムは通信コストが低いという利点がある一方で，処理効率は単一マシンの性能に依存するためスケーラビリティが低く大規模データに対応することが難しい．一方，分散型システムは，データの処理を複数のマシンで行うため通信のオーバーヘッドが発生するという欠点を持つものの集中型システムの課題であった大規模データへの効率的な処理が可能であり，この特徴に注目した研究が昨今多く検討されている [4], [5].

Apache Spark [6] は大規模並列分散処理を行うフレームワークの1つであり，インメモリデータ構造を使用したデータ処理が可能であることから，分散型システムに活用した研究が増えてきている．SPARQLGX [7]，WORQ [8]，DIAERESIS [9] などがその例であり，これらはデータへの問合せ処理高速化に向けたデータ分割手法を提案している．

データ分割は分散並列処理技術を用いた処理における重要な課題の一つであり，そのデータレイアウトは問合せ時に必要な情報を取得するためにアクセスすべきデータ量を大きく左右する．単純な，あるいはランダムなデータ分割では問合せ時に不要なデータへのアクセスや探索が発生し，それに応じて処理コストが増加し問合せ応答の性能低下へとつながる．

ところで，多くのシステムではユーザが発行する典型的な問合せをワークロード情報として取得可能である．この情報を活用することで，問合せ処理を行う上で関連性の高いデータを同じパーティションに配置でき，データアクセスの抑制とそれによる性能の向上が実現できると期待される．

そこで本研究では，ワークロード中の主語，述語，目的語それぞれの共起性を利用したデータ分割とデータのインデックスを利用した効率的な問合せ処理を実現する手法を提案する．実際のワークロードを基にした共起性の導入により，同時に問合せられるデータを同一のパーティションにまとめられると期待できる．さらに，インデックスを用いることで問合せ時に必要なパーティションのみを選択して読み込むことを可能にし，処理するデータ量を抑制できると考えられる．

我々の提案手法の優位性を確かめるために，評価実験ではデータ分割時間，問合せ実行時間，問合せ時のデータ読み込み時間，読み込んだトリプル数，データ読み込み時間と問合せ実行時間の合計時間の 5 つの観点で既存手法と比較する．

本論文の構成は以下のとおりである．2 章では本研究の前提知識を述べ，3 章で関連研究を紹介する．4 章で提案手法を説明し，5 章で評価実験の結果を示す．最後に 6 章で本研究のまとめと今後の展望を述べる．

2 前提知識

本章では，本研究の基本事項を紹介する．まず 2.1 節で，本研究で扱うデータの対象である RDF について触れ，続く 2.2 節で RDF への標準的な問合せ言語である SPARQL に焦点を当てる．最後に，2.3 節で本研究で並列分散処理のために用いる Apache Spark について説明する．

2.1 RDF（Resource Description Framework）

2.1.1 RDF の基本知識

RDF（Resource Description Framework）は，データを主語 s，述語 p，目的語 o のトリプルと呼ばれる形式を基本パターンとして表現するフレームワークであり，トリプルを組み合わせることで複雑なデータの関係性を簡潔に表現することができる．各データは IRI（International Resource Identifier）を用いて示されており，あらゆる資源を一意に識別し，複数のデータベースにまたがるデータを統合することも可能である．また，リテラルやブランクノードも重要である．リテラルは文字列や数などの具体的なデータ値をそのまま扱うものであり，目的語としてのみ使用可能である．ブランクノードは IRI を持たない一時的なエンティティの表現などに使用されるものであり，主語と目的語としてのみ使用可能である．

以下はトリプルの例である．ただし，⟨_:Brank⟩ は空白ノード，⟨4.20000⟩ は数のリテラルによる表現である．

⟨dbr:University_of_Tsukuba⟩⟨dbo:city⟩⟨dbr:Tsukuba⟩

⟨dbr:Tsukuba⟩⟨db:Temperature⟩⟨_:Brank⟩

⟨_:Brank⟩⟨dbp:febMeanC⟩⟨4.200000⟩

2.1.2 RDF の視覚化

さらに，RDF には視覚化やそれを組み合わせたグラフ表現が可能であるという特徴もある．主語と目的語をノード，述語を有向エッジとすることで視覚化でき，それらを複数組み合わせることでグラフが形成できる．これは複雑なデータ間のネットワークの直感的理解や，グラフアプローチによる処理などに役立つ．図 1 はトリプルの基本パターンを視覚化したものであり，筑波大学がつくば市にあることを示している．そして図 **??** は複数の基本パターンを組み合わせてグラフにしたものである．なお，ここでは楕円が IRI ノード，長方形がリテラル，菱形がブランクノードを表している．

2.1.3 RDF の基本構文

RDF データの表現には，Turtle（Terse RDF Triple Language）や N-Triples などの構文が一般に用いられる．Turtle は，RDF データを簡潔かつ読みやすい形式で書くための構文であり，主に人間が読み書きするシナリオに適している．さらに，図 1 や図 2 で用いているように dbr や dbo などの接頭辞を使用して IRI を短縮し，データの表現を簡略化することが可能である．一方，N-Triples はよりシンプルな形式で，各トリプルを一行で表現する．これは機械による解析などに適しており，その単純さから大規模な RDF データの処理に用いられることも多い．本研究でも使用する RDF データは N-Triples 形式のものである．

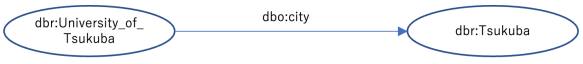


図 1: トリプルの視覚化の例

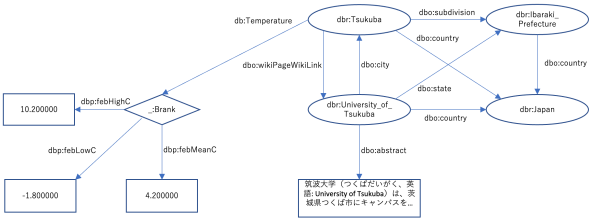


図 2: RDF のグラフ表現の例

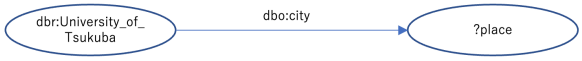


図 3: SPARQL クエリのグラフ表現の例

2.2 SPARQL（SPARQL Protocol and RDF Query Language）

SPARQL（SPARQL Protocol and RDF Query Language）は，RDF データへの標準的な問合せ言語である．基本パターンはリレーショナルデータベースへの標準的な問合せ言語である SQL と類似しており，SELECT 句と WHERE 句から構成される．SELECT 句は，問合せの結果として取得したい情報に対応する変数を列挙し，WHERE 句では問合せたい要素を”?”で始まる変数としたトリプルを記述する．例えば，図 2 の RDF グラフに対して筑波大学がある市町村を問合せたい場合，以下のように記述する．

```
SELECT ?place
WHERE{
  dbr:University_of_Tsukuba dbo:city ?place .
}
```

さらに，SPARQL は RDF と同様にグラフ表現が可能であり，問合せは主にグラフパターンマッチングにより行われる．例示した SPARQL クエリは，図 3 のように視覚化できる．また，これらの問合せ処理は SPARQL Endpoint と呼ばれるインターフェースを介して行われる．

2.3 Apache Spark

2.3.1 Apache Spark の概要

Apache Spark [6] は，大規模データの高速処理を目的とした分散処理フレームワークである．このシステムはスケールアウトアーキテクチャに基づいており，サーバー台数の増加に伴って処理能力を向上させることができる．この特徴により，ビッグデータの急激な増加に対して柔軟かつ効率的に対応することが可能になっている．

Spark 登場前には，Google が開発した GFS（Google File System）[10] と Google MapReduce [11] を基にしたオープンソースである Hadoop [12] が大規模データ処理分野の主流であったが，Hadoop はストレージ管理やデータ処理においていくつかの課題を抱えていた．Spark はこれらの課題を克服し，さらに RAM 上でのデータ処理を行うことで MapReduce よりも高速な処理を実現している．

また，Spark には構造化データや半構造化データの処理を容易にするための豊富なデータアクセスモデルと API が提供されている．これらを利用することで，様々なデータフォーマットを柔軟に扱いながら高速な並列データ処理を実現できる．

2.3.2 DataFrame

DataFrame は Spark で提供されている API の一つであり，大規模構造化データのための表形式データモデルとして分散処理に特化している．各列には名前とデータ型を保持することができ，SQL のような操作を容易にする設計となっている．また，JSON, CSV, Parquet など様々なデータフォーマットとの互換性があり，多くのデータソースからの読み込みが可能である．

2.3.3 Spark SQL

Spark SQL も Spark で提供されている API の一つであり，構造化データの処理に特化している．ユーザは SQL クエリを用いて DataFrame のデータを処理することが可能であり，様々なデータセットに対して統合的な解析操作を実現する．ただし，SPARQL には対応していないため，本研究では SPARQL から SQL への変換処理を行ったのち，DataFrame で読み込んだ RDF データに対して Spark SQL による問合せ処理を実行した．

3 関 連 研 究

本章では，本研究の関連研究を紹介する．3.1 節では，ワークロードを利用したデータ分割を行っている手法を説明し，3.2 節ではワークロードを使用しない手法として媒介中心性とインデックスを利用した手法を説明する．

3.1 ワークロードを用いた手法

ワークロードを利用したデータ分割手法として，Adnan らの研究 [13] がある．この研究では，SPARQL クエリのワークロードから共起している述語をカウントし，貪欲なクラスタリングを適用することで同時に問合せられることが多い述語ペアが同じクラスになるようにデータを分割する．Adnan らは，分割したデータを複数の SPARQL Endpoint 上に一つずつ格納し，フェデレーションエンジンを利用して処理しており，Spark 処理系を対象としている本研究とはこの点で異なっている．

Madkour らの研究 [8] もワークロードに基づいたデータ分割手法を提案している．このアプローチでは，複数クエリで頻繁に繰り返される結合パターンに着目し，ディスク I/O やネットワークシャッフリングのオーバーヘッドを最小化することを目指している．そのために，複数のテーブル間の結合が必要なクエリに対して，ブルームフィルタによる要素の削減を行う．そして，得られたデータに対して主語もしくは目的語に基づいて垂直分割を行い，その結果をキャッシュすることで処理対象のデータ量を抑えている．

Guo らの研究 [14] では，ワークロードクエリのグラフパターンに着目し，各サブグラフから頻出パターンを抽出することでそれに基づいたデータレイアウトを構築している．データ分割は動的に行われ，一定数のワークロードクエリが入力されるたびにデータレイアウトの再構築とそれに伴う移動コストを計算し，それを基に実際に再配置する．なお，この手法も Spark 系を対象とはしていない．

3.2 ワークロードを用いない手法

Georgia らの研究 [9] では，二段階のデータ分割を行う．第一段階では，スキーマグラフに対して媒介中心性を計算し，中心性が高いノードを重要なノードとして特定する．その後，残りのノードに対してスキーマノード同士の依存関係を計算する Dependence を定義し，重要なノードへ割り当てることでデータの分割を行う．第一段階の分割で得られた結果に対して，第二段階では述語ベースで垂直方向のサブパーティショニングを行う．各垂直パーティションには，一つの述語に対する主語と目的語が含まれており，各パーティションのデータサイズを小さくしている．さらに，これらのパーティションに対してクエリ実行時に必要なサブパーティションを直接探せるようにインデックスを生成している．この二段階の分割とインデックスによって，述語が束縛されていないクエリに対してその述語を含む第一段階のパーティションに含まれるデータすべてにアクセスする必要がなくなり，必要なパーティションのみのロードを可能にしている．

4 提 案 手 法

本章では，提案手法について説明する．本研究は大規模な RDF データに対する問合せを高速に行うことを目的としている．これを実現するために，ワークロード中の主語，述語，目的語それぞれに対する共起性を利用した二段階のデータ分割と，それぞれに対するデータへのインデックスを利用する手法を提案する．4.1 節では提案手法の概要を導入し，4.2 節ではデータ分割処理について，4.3 節では問合せ処理についてそれぞれ説明する．

4.1 提案手法の概要

本手法は SPARQL クエリのワークロード情報が与えられることを前提としており，ワークロードクエリ中のトリプルにお

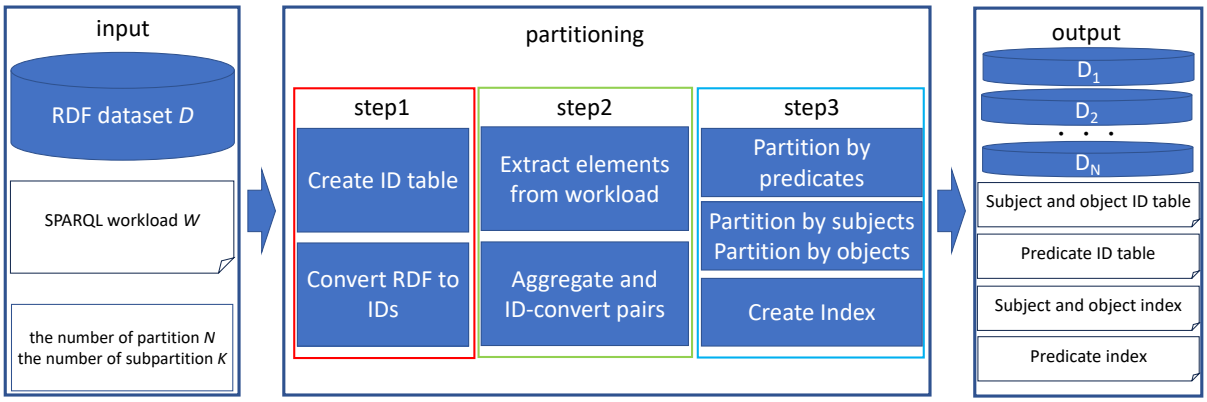


図 4: 提案手法のデータ分割処理の流れ

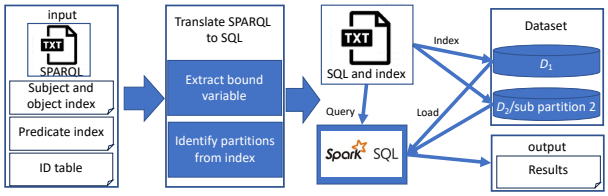


図 5: 提案手法の問合せ処理の流れ

いて共起性の高いトリプルを同一パーティションに配置することで SQL での効率的なクエリ処理を実現するという考えに基づく．図 4 は，提案手法のデータ分割処理の概要を示している．入力は，1) トリプルの集合からなる RDF データ D，2) ワークロードクエリ $W = q_1, \ldots, q_{|W|}$，3) パーティション数 N，およびサブパーティション数 K である．出力は RDF データのパーティション $D_1, \ldots, D_N$ （$D_1 \cap \cdots \cap D_N = \emptyset$），データと ID との関係を記録した ID テーブル，および各データのパーティションへのパスを記録するインデックスである．データ分割処理は大きく三つのステップで構成される．ステップ 1 では，データの冗長性を軽減するための ID 化を行う．ステップ 2 では，ワークロードから主語，述語，目的語をそれぞれ抽出し，共起する要素を ID に変換してペアを作成する．その後，ペアの出現回数を測定し，出現頻度の順にソートする．ステップ 3 では，述語の共起回数を基にしたクラスタリングによる第一段階の分割を行い，その結果に対して第二段階の分割として主語の共起回数に基づくクラスタリングと目的語の共起回数に基づくクラスタリングの二種類を行い，データを分割して出力する．同時に，主語，述語，目的語それぞれのパーティションへのインデックスも出力する．

図 5 は，提案手法の問合せ処理の概要を示している．問合せ処理は大きく二つのステップで構成される．ステップ 1 では，SPARQL の SQL への変換処理を行う．これは，通常 RDF データへの問合せは SPARQL によって行うが，Spark SQL は SPARQL に対応していないためである．また，その際 ID テーブルを利用した束縛変数の ID 化とインデックスを利用したパーティションの特定を同時に行う．ステップ 2 では，実際に SQL による問合せ処理を行う．変換されたクエリを受け取ると，ステップ 1 で特定していたパーティションを読み込み，DataFrame によりテーブルを作成し，それに対して Spark SQL による問合せを行うことで結果を取得する．

Algorithm 1 Workload Analysis.

Require: SPARQL workload W,Subject-object ID table ID_{so}, Predicate ID table ID_p
Ensure: Subject, predicate, and object co-occurrence counts denoted as $C_{sub}, C_{pre}, and C_{obj}$

1: Load ID_{so} and ID_p as HashMap
2: $L_{sub} := \emptyset; L_{pre} := \emptyset; L_{obj} := \emptyset$
3: **for all** $query \leftarrow W$ **do**
4: $where$ = extractInWhere($query$) ▷ Extract contents inside WHERE clause
5: **for all** $(s,t) \leftarrow$ extractBoundSubject($where$) **do** ▷ Create subject pairs from WHERE clause
6: $L_{sub} \leftarrow (\mathrm{ID}_{so}(s), \mathrm{ID}_{so}(t))$
7: **end for**
8: **for all** $(s,t) \leftarrow$ extractBoundPredicate($where$) **do** ▷ Create predicate pairs from WHERE clause
9: $L_{pre} \leftarrow (\mathrm{ID}_p(s), \mathrm{ID}_p(t))$
10: **end for**
11: **for all** $(s,t) \leftarrow$ extractBoundObject($where$) **do** ▷ Create object pairs from WHERE clause
12: $L_{obj} \leftarrow (\mathrm{ID}_{so}(s), \mathrm{ID}_{so}(t))$
13: **end for**
14: **end for**
15: Convert all elements in L_{sub}, L_{pre}, and L_{obj} into ID using HashMap
16: C_{sub} =countAllPairsByMapReduce(L_{sub})
17: C_{pre} =countAllPairsByMapReduce(L_{pre})
18: C_{obj} =countAllPairsByMapReduce(L_{obj})
19: Output C_{sub}, C_{pre}, and C_{obj}

4.2 データ分割処理

4.2.1 Step1: ID の割り当て

RDF データ中の IRI やリテラルは一般に長い文字列であり，問合せ処理における文字列マッチングのコストが高くなる．したがって，これらを事前に唯一性が保証された整数値の ID に変換しておくことで，冗長性を軽減する．

4.2.2 Step2: ワークロードの分析

続いて，ワークロードクエリを分析して主語，述語，目的語それぞれに対して共起しているペアを集計する．入力は SPARQL クエリのワークロード W と，ステップ 1 で作成したデータと ID の対応テーブルであり，出力は各ペアとその共起回数をまとめたテキストである．Algorithm 1 はこのステップのアルゴリズムを示している．まず，与えられたワークロード $W = q_1, \ldots, q_{|W|}$ に対して `WHERE` 句内を抽出し，そこから主語，述語，目的語それぞれの束縛変数をすべて取り出す．各束縛変数を ID に変換し，対応する共起リスト $L_{sub}, L_{pre}, Lobj$ に追加する (2-14 行目)．このリストの要素はタプル $p = \langle P_1, P_2 \rangle$ であり，ここから共起回数を集計する．

例えば，図 6(左) の 4 つのクエリに対して述語の共起回数

```
SELECT * WHERE          SELECT * WHERE          SELECT * WHERE          SELECT * WHERE
{                       {                       {                       {
 ?s p1 o1 .              s1 p1 ?o .              ?s p1 o1 .              ?s p1 ?o .
 ?s p2 o2                ?o p2 ?o2               ?s p3 o3                ?s p2 ?o .
}                       }                       }                       ?s p3 ?o .
                                                                        ?s p4 ?o .
                                                                        }
```

P1	P2	count
p1	p2	3
p1	p3	2
p1	p4	1
p2	p3	1
p2	p4	1
p3	p4	1

図 6: クエリの例 (左) とそれに対する述語の共起回数の例 (右)

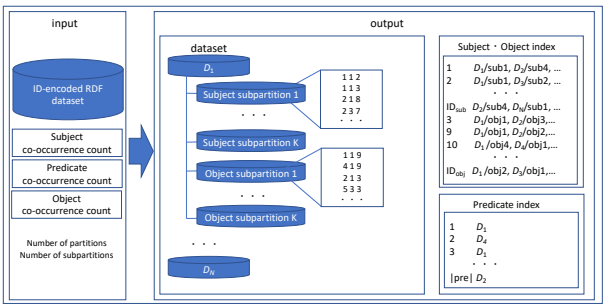


図 7: Step3 のデータ構成

を数える場合，$L_{pre} = \langle\langle p1, p2\rangle, \ldots, \langle p2, p4\rangle, \langle p3, p4\rangle\rangle$ となる．そして，これに対して集計処理を施すことで図 6(左) の結果が得られる．

4.2.3 Step3: データ分割

最後に，ステップ 2 で集計した情報を基にデータセットを分割する．図 7 は，ステップ 3 で入出力されるデータの具体的な構成を示したものである．

提案手法では，データは二段階に分割する．クラスタリングの手法は両段階で同じであり，そのアルゴリズムは Algorithm 2 に示す通りである．このアルゴリズムでは，まず，各クラスタに属する述語を記録する cluster， 述語が属するクラスタを記録する Index の二つのハッシュマップを用意する．ただし，Index は最初は空である．また，述語の種類数をパーティション数 N で割った値を一つのクラスタのサイズとしてあらかじめ計算しておく（1-4 行目）．共起している述語ペアを取り出し，ペアの一方がクラスタに属していない場合はそれをもう一方と同じクラスタに追加した場合のクラスタに含まれる述語数と t を比較する．サイズが t 以下である場合には同じクラスタに追加し，Index を更新する（9-15 行目）．述語ペアの両方がまだクラスタに属していない場合には，クラスターに含まれる述語数が最も小さいクラスタに両方の述語を追加し，Index を更新する（17-22）．最後に，9-15 行目の処理で追加できなかった述語やワークロードに出現していなかった述語を，述語数が最小のクラスタに順に追加していく．

Algorithm 3 にステップ 3 全体のアルゴリズムを示す．Index_s と Index_o は ID をキー，その ID が含まれるパーティションの番号の集合を値とする HashMap である（1-2 行目）．第一段

Algorithm 2 Clustering(D_{ID}, Co-occurrences$elem$, N)

Require: ID-encoded RDF dataset DID, Co-occurrence counts Co-occurrences$_{elem}$, ▷ elem is either subject, predicate, or object Number of partitions N
Ensure: Partitioned datasets $D_1, \ldots, D_N$, Index
1: cluster = $((0, \emptyset), \ldots, (N-1, \emptyset))$: HashMap; ▷ Record elements contained in each cluster
2: Index: HashMap; ▷ Record the cluster to which an element belongs
3: t = |distinct($elem$)|/N ▷ The number of types of elem in one cluster
4: **for all** $(p_1, p_2) \leftarrow$Co-occurrences **do**
5: **if** p_1 in Index AND p_2 in Index **then**
6: continue
7: **end if**
8: **if** One of them already belongs to a cluster **then**
9: Let p_1 already belong to a cluster
10: clusterNo = Index(p_1)
11: **if** |cluster(clusterNo)| + 1 ≤ t **then** ▷ Compare the number of elem types after addition with t
12: cluster(clusterNo) ← p_2
13: Index(p_2) ← clusterNo
14: **end if**
15: **end if**
16: **if** Neither belongs to a cluster **then**
17: clusterNo = cluster with the fewest number of elem
18: cluster(clusterNo) ← p_1, p_2
19: Index(p_1) ← clusterNo
20: Index(p_2) ← clusterNo
21: **end if**
22: **end for**
23: **for all** $p \leftarrow$ remaining elements **do**
24: clusterNo = cluster with the fewest number of elem
25: cluster(clusterNo) ← p
26: Index$_p$(p) ← clusterNo
27: **end for**
28: Output Index, cluster

階では ID 化された RDF データ全体と述語の共起回数，パーティション数を clustering 関数の入力として述語の共起に基づいたクラスタリングを行う（3 行目）．続いて第二段階では，その結果得られたクラスタを入力データセットとして，主語，目的語それぞれに対して clustering 関数を適用してクラスタリングを行う．得られたパーティションは parquet 形式で逐次出力し，Index を更新する（4-15 行目）．すべてのデータの分割が完了すると，Index を出力して処理が終了する．

4.3 問合せ処理

4.3.1 SPARQL から SQL への変換

SPARQL は Spark SQL での処理ができないため，SPARQL が与えられた時，まず SQL への変換を行う．変換処理は以下の流れになる．

1. SPARQL クエリ Q から `WHERE` 句内のトリプルを抽出
2. 各トリプル（$q_1, \ldots q_Q$）に対して，それぞれ次の処理を行う．
   (a) トリプル q_i から束縛変数を抽出する．
   (b) インデックスを基に読み込むべきファイルへのパス

Algorithm 3 Data Partitioning Algorithm.

Require: ID-encoded RDF dataset D_{ID}, Subject co-occurrence counts Co-occurrences$_{sub}$, Predicate co-occurrence counts Co-occurrences$_{pre}$, Object co-occurrence counts Co-occurrences$_{obj}$, Number of partitions N, Number of sub-partitions K
Ensure: N partitions $D_1, \ldots, D_N$,
Subject index Index$_s$, Predicate index Index$_p$, Object index Index$_o$

```
Index_s := ((subject_1, ∅), ..., (subject_|subject|, ∅)): HashMap;
Index_o := ((object_1, ∅), ..., (object_|object|, ∅)): HashMap;
firstPartition, Index_p = clustering(D_ID, Co-occurrences_pre, N)
for all partition ← firstPartition do
    secondPartition, Index = clustering(partition, Co-occurrences_sub, K)
    Output secondPartition
    for all (e, p) ← Index do
        Index_s(e) ← p
    end for
    secondPartition, Index = clustering(partition, Co-occurrences_obj, K)
    Output secondPartition
    for all (e, p) ← Index do
        Index_o(e) ← p
    end for
end for
Output Index_s, Index_p, Index_o
```

を特定し，テーブル名を決定する．

(c) 決定したテーブル名を `FROM` 句に，束縛変数を `WHERE` 句に，非束縛変数を `SELECT` 句に指定し，その結果を `AS` 句によって table_i とする SQL サブクエリを作成する．

3. 得られた $|Q|$ 個の SQL サブクエリを `FROM` 句に指定し，SPARQL クエリの `SELECT` 句に指定されていた変数を SQL の変数に指定する．SPARQL クエリの `WHERE` 句内の非束縛変数の結合を基に SQL の `WHERE` 句内を作成し，得られた SQL と読み込むファイルのパス，テーブル名をまとめて出力する．

上記の手順 2(b) のテーブル名はファイルへのパスから指定され，パスが同じ場合常にテーブル名は同じになる．なお，重複したパスとテーブル名は 1 つにまとめられる．したがって，複数の SQL サブクエリに同じパスの組み合わせが存在する場合，それらすべてのサブクエリに対して 1 つのテーブルを作成するだけでよくなり，データの読み込み時間と読み込むデータ量の削減が期待できる．一方で，クエリの `WHERE` 句が `s1 ?p1 ?o1` や `s2 ?p2 ?o2` のようなパターンで構成されている場合，各サブクエリの結果は複数のファイルに分散している可能性がある．あるサブクエリパターンの結果がファイル 1, 2, 4 に，別のサブクエリパターンの結果がファイル 4, 5, 6 にある場合，変換プロセスではファイル 1, 2, 4 を 1 つのテーブルに，ファイル 4, 5, 6 を別のテーブルにグループ化する．これにより，ファイル 4 からのデータが重複する可能性があり，この場合は読み込むデータ量が増加することになる．さらに，データセット内のすべてのデータにインデックスを作成しているため，サブクエリの束縛変数にインデックスがない場合は一致するデータが存在しないと判断できる．評価実験における速度評価ではこのような場合，あらかじめ用意しておいた空のテーブルを読み込むファイルとして指定して問合せを行った．

5 評価実験

本章では，提案手法の性能を確認するために合成データセットと実世界データセットを 1 つずつ使用し，複数の指標によって従来手法との比較を行った結果について述べる．従来手法としては従来手法を上回る性能を示している DIAERESIS を使用した．その際のパーティション数は，DIAERESIS の論文中のものと同様に 4 に設定した．なお，提案手法のパーティション数は第一段階で 5, 第二段階で２０とした．また，ID 化の優位性を確認するために，提案手法で ID 化を行わない場合の性能も検証した．

5.1 実験環境

実験は，Apache Spark（３．３．１）を Spark Standalone モードで動作させた 1 台の物理マシン上で実施した．提案手法では基本的にドライバーメモリとエグゼキューターメモリに 32GB を割り当てた．ただし，ID 化を行う時のみ 50GB を割り当てている．従来手法は，ドライバーメモリとエグゼキューターメモリのいずれも 10GB を割り当てている．マシンのスペックは表 1 の通りである．さらに，AWS EC2 の EMR クラスター

表 1: 実行マシンのスペック

OS	Ubuntu 18.04.5 LTS
CPU	AMD EPYC 7502P 32-Core Processor
メモリ	128GB

(emr-7.0.0) でインスタンスタイプ m5.xlarge，インスタンスサイズ 10 での実験も行った．提案手法は scala(2.12.7) で実装し，sbt（1.8.0）でコンパイルした．比較手法は scala(2.12.10) で実装されており，JDK1.6 でコンパイルした．

5.2 データセット

本実験では以下の 2 つのデータセットを使用した．なお，AWS 上では，比較手法において LUBM を用いた際にエラーが発生したため SWDF のみ実験を実施している．

5.2.1 LUBM（The Lehigh University Benchmark）

LUBM（The Lehigh University Benchmark）[15] は大学の数をパラメータとした合成データ生成ツールであり，RDF データベースシステムのベンチマークとして広く利用されている．実験では，パラメータを 100 としたデータセットを作成しており，これはデータサイズ 2.37GB，トリプル数 13.8M で構成されている．ワークロードと性能評価には，ベンチマークから提供されたクエリから 13 個を使用した．

5.2.2 SWDF（The Semantic Web Dog Food）

SWDF（The Semantic Web Dog Food）は，人，論文，講演に関するセマンティックウェブ会議のメタデータを含む実世界の RDF データセットである．このデータセットはデータサイズが 50MB であり，304,583 トリプルが含まれている．ワークロードと性能評価には，比較手法である DIAERESIS で使用されていたクエリから 207 個を使用した．なお，このクエリは実際のクエリログに基づいて FEASIBLE ベンチマークジェネレータ [16] によって生成されたものである．

5.3 評価指標

本実験では，性能評価に以下の指標を導入した．なお，比較手法の実験の条件に合わせるために，SPARQL から SQL への

表 2: 前処理時間の比較（括弧内は ID 化時間（左）と分割時間（右））

Method	Processing Time	Data Size
LUBM (13.4M Triples)		
DIAERESIS	16.38 ± 0.30 min	762MB
Proposed Method (with ID)	28.31 ± 0.39 min (14.46 + 13.85)	199MB
Proposed Method (without ID)	38.44 ± 0.87 min	406MB
SWDF (30.4K Triples)		
DIAERESIS	1.80 ± 0.02 min	14MB
Proposed Method (with ID)	1.27 ± 0.03 min (0.20 + 1.07)	6.6MB
Proposed Method (without ID)	3.06 ± 0.07 min	64MB

表 3: 比較手法を 1 とした時の問合せ実行時間の比の平均

Dataset	DIAERESIS	Proposed method (with ID)	Proposed method (without ID)
LUBM	1	0.80	0.95
SWDF	1	0.51	0.96

変換にかかる時間は計測していない．報告する時間は 5 回の実行の平均値である．

- 前処理時間: 分割したデータを出力するまでの時間
- 問合せ実行時間: SQL を実行してから結果を得るまでの時間
- テーブル作成コスト: 読み込んだデータ中のトリプル数と，テーブル作成までの時間
- 合計問合せ処理時間: 問合せ実行時間とテーブル作成時間の合計

5.4 実験結果（単一マシン）

5.4.1 前処理時間

前処理時間は元の RDF データセットを入力してから分割データが出力されるまでの時間を計測したものであり，その結果を表 2 に示す．提案手法の ID 化の有無による処理時間では，ID 化を行う場合の方が各データセットに対して大きく優れた結果を示しており，ID 化の優位性が確認できた．提案手法と比較手法では，LUBM では比較手法が，SWDF では提案手法が高速であり，ID 化にかかる時間がボトルネックであると考えられる．

5.4.2 問合せ実行時間

表 3 は SQL 実行時間の比較結果であり，いずれのデータセットにおいても提案手法が比較手法を上回る結果となっている．個々のクエリでの比較では，LUBM では 13 個中 12 個のクエリで，SWDF では 207 個中 155 個のクエリで提案手法が優れていた．特に SWDF では，平均 73%，最大 90%の処理時間削減となっており，一方で結果が劣っていたクエリについてもその処理時間増加量の平均は 23%，最大 46%であった．よって，一部劣る処理はあるもののほとんどの問合せ処理で性能向上が実証できていると考える．

5.4.3 テーブル作成コスト

提案手法と比較手法はいずれもインデックスを利用して必要なデータを事前に特定し，それらによって短時間で最低限のデータへアクセスしている． 表 4 は問合せ時に読み込んだトリプル数の比較であるが，いずれのデータセットにおいても提案手法は比較手法よりも平均的に多くのトリプルを読み込んでいる．これについて詳細な検証を行うと，特に LUBM のコンプレックスクエリで特に多くのトリプルを読み込んでいることが確認された．また，SWDF では半数以上の問合せでは読み込んだトリプル数は比較手法より少なかった．したがって，特定のパターンを含むクエリに対してはより多くのトリプルを読み込む傾向にあると考えられる．

表 4: 比較手法を 1 とした時のロードトリプル数の比の平均

Dataset	DIAERESIS	Proposed Method (with ID)	Proposed Method (without ID)
LUBM	1	2.01	2.59
SWDF	1	1.82	2.03

表 5: 比較手法を 1 とした時のテーブル作成時間の比の平均

Dataset	DIAERESIS	Proposed method (with ID)	Proposed method (without ID)
LUBM	1	0.66	1.48
SWDF	1	0.52	0.68

表 6: 比較手法を 1 とした時の合計時間の比の平均

Dataset	DIAERESIS	Proposed Method (with ID)	Proposed Method (without ID)
LUBM	1	0.71	1.26
SWDF	1	0.47	0.84

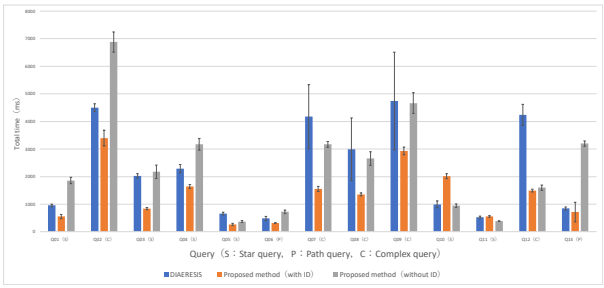


図 8: LUBM における合計時間の比較

一方，テーブルを作成する時間は表 5 に示す通りであり，提案手法はいずれのデータセットに対しても比較手法より 30%以上高速にテーブル作成を完了している．個別に LUBM のクエリを分析したところ，標準偏差も多くのクエリで比較手法より大幅に小さく，安定していることが確認できた．この傾向は SWDF でも同様であり，207 個中 205 個のクエリで比較手法より高速であった．これはワークロードの導入により複数のサブクエリ間でテーブルの共有が可能になり，作成すべきテーブル数が削減できたためではないかと考えられる．

5.4.4 合 計 時 間

最後にテーブル作成時間と問合せ実行時間の合計時間を問合せ全体にかかる時間とみなして比較を行う． 表 6 を見ると，提案手法は両データセットで比較手法を大幅に凌駕していることが確認できる．詳細な分析を行ったところ，LUBM においては図 8 に示すように，比較手法より 2 倍程度遅いクエリも 1 つ存在したものの，その他はほとんどのクエリで比較手法を大きく上回っており，最大約 75%の高速化を達成した．SWDF においても，198 のクエリで比較手法を上回っており，その差は最

Method	Processing Time
DIAERESIS	2.84 ± 0.06 min
Proposed Method（with ID）	20.18 ± 0.81 min（0.70+19.48）

図 9: 比較手法と提案手法の前処理時間（括弧内は ID 化時間（左）と分割時間（右））

method	Load Time	Query Time	Total Time
DIAERESIS	1	1	1
Proposed Method（with ID）	0.80	0.61	0.59

図 10: 比較手法を 1 とした時の各処理時間の比の平均

大約 95%であった．一方，残りの 9 個のクエリで比較手法に劣る結果となっていたが，その差は最大約 15%であり，許容できるものと考えられる．

5.5 実験結果（AWS）

5.5.1 前処理時間

AWS 上で RDF データを分割する際にかかった時間を計測した結果を表 9 に示す．両手法とも単一マシンでの実験よりも処理が遅くなっており，これは複数ノード間での通信にかかるコストによるものだと考える．また，提案手法の実装における分散処理の最適化が十分ではなく，その影響もより顕著に出たと考えている．

5.5.2 実 行 時 間

各処理にかかる時間を表 10 に示す．いずれも比較手法を大幅に上回っており，個別のクエリでも約 70%のクエリで提案手法が高速に処理されていた．さらに，標準偏差も多くのクエリで小さく安定した処理ができていると考えられる．

6 ま と め

本論文では，RDF データの効率的な分割と処理のための Spark ベースの方法を提案した．入力ワークロードクエリの共起を分析することにより関連するデータを同じパーティションに配置し，データ取得の効率向上を達成した．また，各データとそれが属するパーティションを記録したインデックスの導入によりさらなる効率化も可能にした．評価実験では，提案手法が単一ノードにおいて既存手法を上回ることを示した．これは主にデータアクセスの向上や共起性の活用による読み込むトリプル数の削減，ID 化による冗長性排除によるものであると考えられる．一方で，クエリの構成によっては重複したデータの読み込みが発生することがあり，それに伴って読み込むトリプルの数が極端に多くなる場合も確認した．したがって，今後の課題として重複したデータ読み込みの抑制が挙げられる．また，ワークロードの量や質による影響の検証，統計データ等の利用によるクエリ効率化，より大規模なデータセットにおける検証等も行う予定である．

謝　　辞

この成果は，国立研究開発法人新エネルギー・産業技術総合開発機構（NEDO）の 「ポスト 5G 情報通信システム基盤強化研究開発事業」（JPNP20017）の委託事業，JST CREST（JPMJCR22M2），科学研究費補助金（JP22H03694）に支援によるものです．

文　　献

[1] Graham KLYNE. Resource description framework (rdf) : Concepts and abstract syntax. *http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/*, 2004.
[2] B. McBride. Jena: a semantic web toolkit. *IEEE Internet Computing*, Vol. 6, No. 6, pp. 55–59, 2002.
[3] Thomas Neumann and Gerhard Weikum. Rdf-3x: A risc-style engine for rdf. *Proc. VLDB Endow.*, Vol. 1, No. 1, p. 647–659, aug 2008.
[4] Mohammad Hammoud, Dania Abed Rabbou, Reza Nouri, Seyed-Mehdi-Reza Beheshti, and Sherif Sakr. Dream: Distributed rdf engine with adaptive query planner and minimal communication. Vol. 8, No. 6, p. 654–665, feb 2015.
[5] Alexander Schätzle, Martin Przyjaciel-Zablocki, Simon Skilevic, and Georg Lausen. S2rdf: Rdf querying with sparql on spark. *arXiv preprint arXiv:1512.07021*, 2015.
[6] http://spark.apache.org.
[7] Damien Graux, Louis Jachiet, Pierre Genevès, and Nabil Layaïda. Sparqlgx: Efficient distributed evaluation of sparql with apache spark. In *The Semantic Web–ISWC 2016: 15th International Semantic Web Conference, Kobe, Japan, October 17–21, 2016, Proceedings, Part II 15*, pp. 80–87. Springer, 2016.
[8] Amgad Madkour, Ahmed M Aly, and Walid G Aref. Worq: Workload-driven rdf query processing. In *International Semantic Web Conference*, pp. 583–599. Springer, 2018.
[9] Georgia Troullinou, Giannis Agathangelos, Haridimos Kondylakis, Kostas Stefanidis, and Dimitris Plexousakis. Diaeresis: Rdf data partitioning and query processing on spark.
[10] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pp. 29–43, 2003.
[11] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. 2004.
[12] http://hadoop.apache.org.
[13] Adnan Akhter, Muhammad Saleem, Alexander Bigerl, and Axel-Cyrille Ngonga Ngomo. Efficient rdf knowledge graph partitioning using querying workload. K-CAP '21, 2021.
[14] Xintong Guo, Hong Gao, and Zhaonian Zou. Wise: Workload-aware partitioning for rdf systems. *Big Data Research*, Vol. 22, p. 100161, 12 2020.
[15] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. Lubm: A benchmark for owl knowledge base systems. *J. Web Semant.*, Vol. 3, No. 2-3, pp. 158–182, 2005.
[16] Muhammad Saleem, Qaiser Mehmood, and Axel-Cyrille Ngonga Ngomo. Feasible: A feature-based sparql benchmark generation framework. In *The Semantic Web-ISWC 2015: 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part I 14*, pp. 52–69. Springer, 2015.

Fast Parallel Computation for Random Walk based t-SNE

Takeshi MIKI†, Yasuhiro FUJIWARA††, and Hideyuki KAWASHIMA††

† Keio University
5322 Endō, Fujisawa-shi, Kanagawa 252–0882, Japan
†† NTT Communication Science Laboratories,
1–5–1 Ootemachi, Chiyoda-ku, Tokyo, 100–8116, Japan
E-mail: †t.miki@keio.jp, ††yasuhiro.fujiwara@ntt.com, †††river@sfc.keio.ac.jp

Abstract Effective dimension reduction techniques are crucial in the era of big data. In this paper, we introduce an optimized version of the renowned dimension reduction technique, random walk-based t-SNE, by addressing its traditionally significant computational cost. Our proposed method incorporates parallel computing and Modified Cholesky factorization to enhance processing efficiency, without sacrificing the quality of dimension reduction. The experimental results demonstrate a substantial reduction in execution time while maintaining the high standard of dimension reduction quality. This method has the potential to accelerate the adoption of t-SNE across various data-intensive domains.

Key words t-SNE, Parallization, Efficient

1 Introduction

In our data-driven world, the volume of high-dimensional data that we handle is continuously expanding [2]. Consequently, the importance of data visualization has been steadily increasing. The practice of visualizing high-dimensional data often necessitates a reduction of its dimensions to two or three. Numerous dimension reduction algorithms have been developed for this purpose, including UMAP [12], PCA [9], and PLS [13]. Among those methods, t-SNE (t-Distributed Stochastic Neighbor Embedding) is one of the most common and widely accepted approaches [15].

t-SNE is a non-linear dimension reduction algorithm that portrays the similarity between data points by conditional probabilities. Subsequently, it tries to minimize the Kullback-Leibler (KL) divergence of the conditional probabilities between the high-dimensional and lower-dimensional data points. In doing so, t-SNE effectively preserves both the global and local structural features. Due to its remarkable performance and simplicity, t-SNE finds the following diverse applications:

Cancer Detection: Mass Spectrometry Imaging (MSI) is a technique that simultaneously provides the spatial distribution of biomolecules. This information is necessary for identifying diagnostic and prognostic biomarkers in the context of cancer research [1]. While MSI offers rich insights into the intricate molecular landscape within cancer, its data complexity poses significant challenges. t-SNE, with its ability to reduce data dimensionality while preserving both global and local structures, is a suitable solution. In practice, the application of t-SNE involves the detection of distinct clusters. This has proven particularly beneficial for identifying gastric and primary breast cancer patients [1].

Playlist Generation: In the domain of playlist generation, t-SNE has been introduced as a modeling approach. Specifically, each song is represented as a 34-dimensional data, and t-SNE is used to transform it into a two-dimensional data. The two-dimensional output of the model will be used for projecting and rendering a song catalogue onto a plane. In this context, it has outperformed previous methodologies, such as PCA [10].

As shown in previous examples, t-SNE is a popular technique used in various fields. However, its main drawback is the significant computational cost. The problem arises from the need to calculate pairwise similarities between all data points to compute the Kullback-Leibler divergence between the original data points and the embedded data points. Additionally, t-SNE computes pairwise similarities for all embedded data points in each iteration when updating the embedded data using the gradient descent method. This leads to a quadratic increase in computational complexity [14].

In order to solve this computational bottleneck, random walk-based t-SNE has been introduced. This method selects a subset of landmark points in a randomized manner, reducing the number of data points that need to be visualized. Additionally, random walk-based t-SNE uses the k-nearest neighbor graph to approximate distances between

Table 1 Definition of each symbols

Symbol	Definition
N	Number of data-points
M	Number of dimensions in the original data
m	Number of dimensions in the embedded data
n	Number of landmark points
$\boldsymbol{X}$	Matrix of the original data
$\boldsymbol{Y}$	Matrix of the embedded data
x_i	i-th data-point of the original data
y_i	i-th data-point of the embedded data
p_{ij}	Similarity of the original data
q_{ij}	Similarity of the embedded data
$\mathbf{L}$	Graph Laplacian
k	Number of neighbors in *knn* graph

these points efficiently.

However, despite these optimizations, time complexity still remains a substantial challenge. This complexity stems from the need to construct and utilize the k-nearest neighbor graph, which is a critical step in the random walk-based approach for estimating pairwise distances and similarities among data points.

In this paper, we propose a multi-core version of the modified random walk-based t-SNE. The implementation was done in C++ and leveraged OpenMP to execute the code in parallel across multiple cores. The parallelization was done with thread counts ranging from 1 to 64.

The paper is structured as follows: Section 2 provides an overview of the foundational concepts, offering a brief explanation of t-SNE and random walk-based t-SNE. Section 3 introduces the methodology developed for this research. Section 4 discusses the experiments conducted and the results earned. Sequentially, section 5 focuses on related work. The conclusion and discussion drawn from the research is provided in sections 6 and 7.

2 Preliminaries

2.1 t-SNE

This section will explain the background and mathematical definitions related to t-SNE. First of all, t-SNE is a dimensional reduction technique used for data visualization. It aims to reduce the dimension of high-dimensional data points and create a lower-dimensional representation while preserving their pairwise similarities. Specifically, t-SNE reduces the dimension of the original data $\boldsymbol{X}$, represented as $N \times M$ matrix, into a lower-dimensional matrix $\boldsymbol{Y}$, represented as $N \times m$ embedded matrix, where N, M and m are the number of data-points, number of dimension in the original data and the number of dimension in the embedded data, respectively. This is achieved by minimizing the cost function F, defined as follows:

$$F = \sum_{i \neq j} p_{ij} log \frac{p_{ij}}{q_{ij}} \tag{1}$$

where p_{ij} is the pairwise similarity between $\mathbf{x}_i$ and $\mathbf{x}_j$, where $\mathbf{x}_i$ and $\mathbf{x}_j$ represents the ith and jth element of $\boldsymbol{X}$, respectively. Additionally, q_{ij} represents the pairwise similarity of embedded data, $\mathbf{y_i}$ and $\mathbf{y}_j$, where $\mathbf{y}_i$ and $\mathbf{y}_j$ are the ith and jth element of $\boldsymbol{Y}$, respectively. Pairwise similarity in the high dimensional data, p_{ij} is defined as follows:

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2},\ p_{i|i} = 0 \tag{2}$$

where $p_{i|j}$ is the similarity of x_i to x_j using conditional probability. $p_{i|j}$ is given as follows:

$$p_{j|i} = \frac{exp(-||x_i - x_j||^2/2\sigma^2)}{\sum_{k \neq i} exp(-||x_i - x_k||^2/2\sigma^2)} \tag{3}$$

where σ is the variance of the Gaussian distribution.

Since x_i will not choose x_i as its neighbour, $p_{i|i}$ becomes 0. On the other hand, $p_{i|j}$ may not equal to $p_{j|i}$, however, joint probability of Equation (2) p_{ij} takes the average. Therefore, the pairwise similarity between x_i and x_j becomes symmetrized. The pairwise similarity of y_i to y_j is defined as q_{ij} which is given as follows:

$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k \neq l}(1 + ||y_i - y_l||^2)^{-1}},\ q_{ii} = 0 \tag{4}$$

Unlike p_{ij} which uses a Gaussian distribution to convert distance into probability [15], q_{ij} uses student-t distribution with single degree of freedom to effectively visualize data. Since the distance from y_i to y_i is 0, $q_{ii} = 0$.

Embedding original data is done by minimization of the cost function using gradient descent method with momentum. The gradient is as follows:

$$\frac{\partial F}{\partial y_i} = 4 \sum_{j \neq i} (p_{ij} - q_{ij})(y_i - y_j)(1 + ||y_i - y_j||^2)^{-1} \tag{5}$$

After obtaining the gradient, $\mathbf{Y}$ should be updated. Gradient update with momentum is given as follows:

$$Y^{(t)} = Y^{(t-1)} + \eta \frac{\partial F}{\partial Y} + \alpha(t) \left(Y^{(t-1)} - Y^{(t-2)} \right) \tag{6}$$

where t is the number of iterations when undergoing gradient descent method, α represents the momentum and η is the learning rate.

Since t-SNE visualizes all embedded data-points, it is required to compute the value of q_{ij} for each data-point pair. Consequently, the computational and memory costs becomes $O(N^2)$. These expensive costs are a major issue, since it becomes infeasible for t-SNE to deal with a big data-set.

2.2 Random walk-based t-SNE

One common solution for high computational cost is to use Random walk-based t-SNE, where it tries to reduce the cost by selecting a certain number of landmark points and only visualizing them [15].

Additionally, unlike original t-SNE, random walk-based t-SNE uses *knn* graph to calculate pairwise similarity analytically. this method starts off by creating *knn* graph for all N data-points with k as the number of neighbours. This *knn* graph is represented using graph Laplacian, denoted as $N \times N$ matrix $\mathbf{L}$, which is defined as

$$\mathbf{L} = \mathbf{A} - \mathbf{W} \tag{7}$$

where $\mathbf{W}$ represents adjacency matrix and $\mathbf{A}$ represents diagonal matrix. When obtaining $\mathbf{W}$, an assumption that the distance between data points x_i and x_j is equivalent to $e^{-||x_i - x_j||^2}$ is made. Additionally, diagonal matrix $\mathbf{A}$ is obtained by diag $(\sum_j W_{1,j}, \sum_j W_{2,j}, \cdots \sum_j W_{N,j})$.

Pairwise similarity p_{ij} can be redefined as the fraction of random walks that starts at landmark x_i and terminates at landmark x_j. Pairwise similarity between all data-points and landmark points could be obtained accurately by solving the following linear system:

$$\mathbf{LP} = -\mathbf{B^T} \tag{8}$$

In this equation, $\mathbf{B^T}$ is $N \times n$ matrix containing columns from adjacency matrix $\mathbf{W}$ corresponding to landmark points, where n is the number of landmark points. It is important to note that $\mathbf{P}$ is a $N \times n$ matrix where pairwise similarity between ith data point and jth landmark point is represented as $\mathbf{P}_{ij}$.

In order to solve Equation (8), Cholesky factorization [8] on Graph Laplacian $\mathbf{L}$ is first conducted, resulting in $\mathbf{L} = \mathbf{CC^T}$, where $\mathbf{C}$ represents the upper-triangular matrix. Sequentially, $\mathbf{Cy} = -\mathbf{B^T}$ is solved using forward substitution and $\mathbf{CP} = \mathbf{y}$ is solved using backward substitution [15].

Once pairwise similarities of landmark points from data points are obtained from $\mathbf{P}$, random walk-based t-SNE would compute the pairwise similarities between embedded data using Equation (4). However, it is important to note that only landmark points are used to visualize, meaning that there are only n embedded data. One should note that embedded data is denoted as $n \times m$ matrix $\mathbf{Y}$, where m is the number of dimension in the embedded space. Gradient descent method is used to update $\mathbf{Y}$.

While the random walk-based t-SNE method offers a potential for acceleration, the challenge involving computational complexity still remains. Specifically, the computation of pairwise similarities q_{ij} among the landmarks scales quadratically with the number of landmarks. Additionally the computational cost for updating embedded matrix $\mathbf{Y}$ has complexity of $O(tn^2)$ where t is the number of iterations in the gradient descent. When using very large datasets, where both the number of landmarks and the iteration count can be enormous, these challenges makes this technology infeasible. To address these issues, our approach focuses on optimizing three key areas: the acceleration of the k-nearest neighbors (*knn*) computation, the efficient calculation of the matrix P representing pairwise similarities, and the updating process for the matrix Y. These enhancements are designed to reduce the time complexity inherent in the original method, making it more applicable for the practical use of large-scale data.

3 Method

3.1 Parallelization of Graph Laplacian

Graph Laplacian $\mathbf{L}$ is calculated using Equation (7). The computational complexity of creating *knn* graph as graph Laplacian is $O(N^2M)$, where M is the number of dimension in the original data. Although this step is only done once, depending on the number of its data and dimension, it could be computationally expensive. Thus, this could be a major issue when dealing with a large data-set.

To address this challenge, we used OpenMP, a standard parallel programming API for C and C++. OpenMP allows us to compute the weight of the edges in parallel. Therefore, the construction of *knn* graph can be divided among multiple processors. Simple implementation is shown in Algorithm 1.

Each row of $\mathbf{L}$ corresponds to a node in *knn* graph. Determining which nodes are connected is done by comparing the weight of the edges and selecting k nearest nodes. In Algorithm 1, since determining each row of $\mathbf{L}$ can be computed independently, it is done in a parallel loop by distributing computation across multiple threads. It is important to note that the weight of the edges are calculated using $e^{-||x_i - x_j||^2}$, which is also computed in parallel. Sequentially, by comparing the weights, we could identify whether a node is k nearest neighbor or not. If a node is not a neighbor then those nodes are not connected, thus the weight will be 0.

In order to efficiently compare the weights, we would create a simple buffer with k spaces and a lock to avoid race condition. The buffer represents the k nearest neighbors. An entry value, which would be the weight W_{ij} of each nodes, would be placed in the buffer if it is empty. If not, by comparing the smallest value of the buffer, each node could either be swapped and replaced in the buffer or rejected meaning that the node is not k nearest neighbor. It is important to note that the smallest value in the buffer always comes in the beginning so that the comparison can be done simply.

Additionally, when a entry value wants to read or write to the buffer, it must acquires the lock in order to avoid race condition. Finally, once each row of $\mathbf{L}$ is computed, they are merged together to create one large matrix $\mathbf{L^T}$.

Algorithm 1: Parallelization of graph Laplacian

Data: $\mathbf{X} = \{x_1, x_2, \ldots, x_N\}$, k= Number of neighbours
Result: Graph Laplacian $\mathbf{L}$
♯ pragma omp parallel for
for *each* x_i **do**
  ♯ pragma omp parallel for
  for *each* x_j **do**
    $W_{ij} = e^{-||x_i - x_j||^2}$
    if x_j is not k-nearest neighbour **then**
      $W_{ij} = 0$
    end
  end
  ♯ pragma omp parallel for
  for *each* x_j **do**
    $A_{ii} += W_{ij}$
  end
end
$L^T = \begin{bmatrix} L_{1,;} & L_{2,;} & \cdots & L_{N,;} \end{bmatrix}$
Return $\mathbf{L}$

3.2 Solving Linear Systems

In order to find pairwise similarity p_{ij} using graph Laplacian, Equation (8) is solved. This is done by decomposing L using Cholesky factorization then using forward and back substitution to get the value of P which is a matrix containing pairwise similarity. However, computational cost for Cholesky factorization is cubic in the number of data points [5]. One solution is to use Modified Cholesky factorization.

Cholesky factorization decomposes positive symmetric matrix $\mathbf{L}$ into lower triangular matrix $\mathbf{C}$ and upper-triangular matrix $\mathbf{C^T}$. Starting from C_{11}, by using Equation (9) and Equation (10), Cholesky factorization solves each value of $\mathbf{C}$. The computational cost for Cholesky factorization would be $\frac{n^3}{3}$ [6].

$$C_{jj} = \sqrt{L_{jj} - \sum_{k=1}^{j-1} C_{jk}^2} \tag{9}$$

$$C_{ij} = \frac{L_{ij} - \sum_{k=1}^{j-1} C_{ik} C_{jk}}{C_{jj}},\ i > j \tag{10}$$

The proposed method uses Modified Cholesky factorization to accelerate the process. The core idea of modified Cholesky factorization is to decompose $\mathbf{L}$ in such way that $\mathbf{L} = \mathbf{CDC^T}$ where $\mathbf{D}$ is an additional diagonal matrix. It is important to note that diagonal component of $\mathbf{C}$ is all 1. This decomposition would reduce the computational cost for factorization but also support solving Equation (8). Similar to Cholesky factorization, starting off from C_{11}, using Equation (11) and Equation (12), Modified Cholesky factorization tries to reduce the total computation. It should be pointed out that, D_{11} would be equal to A_{11}. The total computational complexity would be $\frac{n^3}{6}$ [6], which is faster than normal Cholesky factorization.

$$C_{ij} = \frac{L_{ij} - \sum_{k=1}^{j-1} C_{ik} D_{kk} C_{jk}}{D_{jj}},\ C_{ii} = 1 \tag{11}$$

$$D_{ii} = L_{ii} - \sum_{k=1}^{i-1} C_{ik}^2 D_{kk},\ D_{11} = A_{11} \tag{12}$$

Modified Cholesky factorization guarantees a speedup by using its symmetric features to avoid unnecessary square root calculations.

Furthermore, solving linear system presented in Equation (8) is achieved by solving $\mathbf{CDH} = -\mathbf{B^T}$ using forward substituting and solving $\mathbf{C^T P} = \mathbf{H}$ using backward substitution [7].

Fast computation of solving linear systems is achieved by replacing Cholesky factorization with Modified Cholesky factorization and parallelization as presented in Algorithm 2.

Algorithm 2: Solving Linear Systems

Data: $\mathbf{L}$= Graph Laplacian, $\mathbf{B}$= Matrix containing columns from adjacency matrix $\mathbf{W}$, n= Number of landmark points, N= Number of data points
Result: $\mathbf{P'}$= Matrix containing the pairwise similarity between landmark points
for $i = 1$ *to* N **do**
  Solve D_{ii} with Equation (12)
  ♯ pragma omp parallel for
  for $j = i + 1$ *to* N **do**
    Solve C_{ji} with Equation (11)
  end
end
♯ pragma omp parallel for
for $i = 1$ *to* n **do**
  Solve $\mathbf{p}_i$ with Equation (13)
  ♯ pragma omp parallel for
  for $k = 1$ *to* N **do**
    if Encounter landmark **then**
      Append p_{ik} to $\acute{p}_i$
    end
  end
  $\mathbf{P'} = \begin{bmatrix} p'_1 & p'_2 & \cdots & p'_n \end{bmatrix}$
end
Return $\mathbf{P'}$

The first loop in Algorithm 2 endeavors to decompose $\mathbf{L}$ into $\mathbf{CDC^T}$ by solving Equation (11) and Equation (10) in

turn. This is because once D_{ii} is known, ith column of $\mathbf{C}$ can be solved. Therefore, computation of each element in the ith column is done in parallel. It is important to note that since $\mathbf{C}$ is a lower triangular matrix with diagonal component being 1, C_{ij} should only be computed if $j > i$.

In order to achieve efficient parallelization by distributing computation across multiple threads, we solved Equation (8) in such way that $\mathbf{P}$ is solved per column. Therefore, in each thread, we compute the following equation:

$$\mathbf{L}\mathbf{p}_i = \mathbf{b}_i \tag{13}$$

where $\mathbf{p}_i$ represents ith column of matrix $\mathbf{P}$. Similarly, $\mathbf{b}_i$ represents ith column of $-\mathbf{B^T}$.

Equation (13) can be solved by computing $\mathbf{CDh} = \mathbf{b_i}$ and $\mathbf{C^T p_i} = \mathbf{h}$ using forward and back substitution.

Since Equation (13) is computed in parallel, $\mathbf{p}_i$ would be computed in each thread. Since we are not interested in elements of $\mathbf{p}_i$ that corresponds to non landmark points, our algorithm would only store elements that corresponds to landmark points to a column vector $\mathbf{p'_i}$. To avoid excess operations, when all landmark points are selected, we would terminate the process.

The output for Algorithm 2 is $n \times n$ matrix $\mathbf{P'}$ where P'_{ij} represents pairwise similarity between ith and jth landmark points. It is important to note that $\mathbf{P'}$ is equivalent to selecting rows that corresponds to landmark points in $N \times n$ matrix $\mathbf{P}$.

3.3 Parallelization of Updating Embedded Data

Computational complexity for obtaining $\mathbf{Y}$ is $O(tn^2)$, where t is the number of iterations done when applying gradient descent method and n is the number of landmarks. Depending on the number of iterations, calculating $\mathbf{Y}$ can also be very computationally expensive. Parallelization of updating embedded data is shown in Algorithm 3.

Embedded data $\mathbf{Y}$ would be initialized by random manner [15]. Pairwise similarity between embedded data q_{ij} is obtained using Equation (4). After computing pairwise similarity, the gradient of cost function $\frac{\delta \mathcal{F}}{\delta \mathbf{Y}}$ is acquired using Equation (5). Since computation of q_{ij} for each point is computationally heavy, we would calculate this in parallel. Once the gradient is calculated, $\mathbf{Y}$ is updated using gradient descent method with momentum according to Equation (6).

3.4 Summary of the Proposed Method

Algorithm 4 provides a summary of the process to compute the proposed version of random walk-based t-SNE. There are 3 main steps to the entire implementation. Firstly, we need to compute the adjacency matrix $\mathbf{W}$ and graph Laplacian $\mathbf{L}$ which is indicated in Algorithm 1. Secondly, we would want to obtain matrix $\mathbf{P}$, which involves pairwise similarity

Algorithm 3: Parallelization of Updating Embedded Data

Data: $\mathbf{Y} = \{y_1, y_2, \ldots, y_n\}$, $\mathbf{P'}$ = Matrix with pairwise similarity between landmark points, t = The number of iteration to conduct gradient descent method, α = momentum, η = learning rate
Result: $\mathbf{Y} = \{y_1, y_2, \ldots, y_n\}$
Initialize $\mathbf{Y}$
for *i=1 to t* **do**
  ♯ pragma omp parallel for
  for *each* y_i **do**
    ♯ pragma omp parallel for
    for *each* y_j **do**
      Calculate q_{ij} using Equation (4)
    end
  end
  Update $\mathbf{Y}$ using Equation (5) and Equation (6)
end
Return $\mathbf{Y}$

p_{ij}, using Algorithm 2. However, before running Algorithm 2, we need to compute $-\mathbf{B^T}$ in order to solve Equation (8). This is done by extracting columns of adjacency matrix $\mathbf{W}$, which corresponds to landmark points. $\mathbf{Y}$ is then updated using Algorithm 3 where the computation of gradient is done in parallel.

Our approach improves the efficiency of random walk-based t-SNE focuses on parallel computation and algorithmic optimizations. By leveraging multi-core processors, we hope to accelerate the construction of the graph Laplacian, calculation of pairwise similarities and updating $\mathbf{Y}$.

Algorithm 4: Summary of Parallel and fast computation for random waslk-based t-SNE

Data: $\mathbf{X} = \{x_1, x_2, \ldots, P\}$, t= Number of iteration
Result: $\mathbf{Y} = \{y_1, y_2, \ldots, y_n\}$
if Dimension of $\mathbf{X} > 30$ **then**
  Reduce the dimension of X to 30 by PCA
end
$\mathbf{W}$, $\mathbf{L} \leftarrow$ Algorithm 1
$\mathbf{B^T} \leftarrow$ Columns of $\mathbf{W}$ corresponding to landmark points
$\mathbf{P'} \leftarrow$ Algorithm 2
Update $\mathbf{Y}$ using Algorithm 3
Return $\mathbf{Y}$

4 Experiments and Results

4.1 Experimental Setup

In order to examine the results of the proposed methods, we have conducted multiple experiments. Implementation was done using gcc and OpenMP on a x86-64 computer with 64 CPU cores with frequency of 2.10GHz and 1.5 TiB memory.

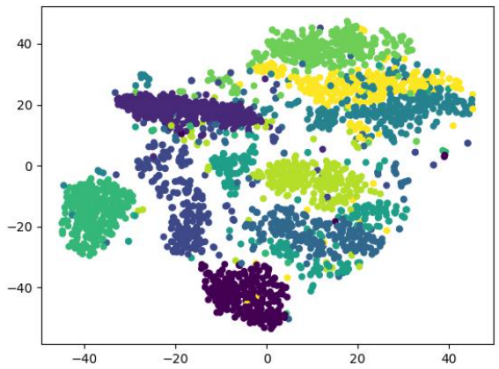


Figure 1 MNIST 2500 with Random Walk based t-SNE

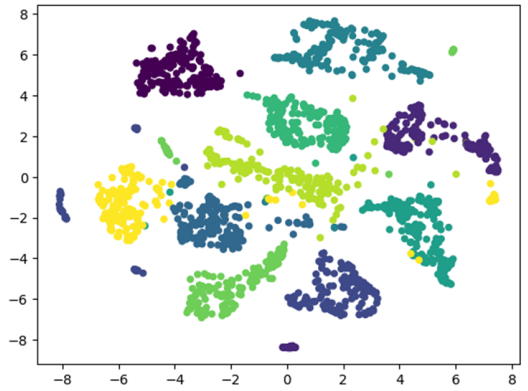


Figure 2 MNIST 2500 with proposed method

Following the original paper on t-SNE, if the number of dimensions of the original data was higher than 30, we conducted PCA to reduce it to 30 [15]. Sequentially, we used t-SNE in order to reduce the dimension into 2. Additionally, we limited the number of nearest neighbors to 20. The number of iteration t was set to 1000, the momentum was 0.5 if $t < 250$ and 0.8 if $t \geqq 250$ [15]. Referring to the original paper, we decided to use an adaptive learning rate, where the initial learning rate would be 100 but would get updated over time [15].

4.2 Visualization of datasets

In order to confirm the quality of this technique, we visualized multiple data using random walk-based t-SNE and the proposed method.

First dataset we used was MNIST 2500 [3] which has 2500 data with each containing 784 dimensions. For this common dataset, each dimension represents the pixel of a hand written number. There are 10 labels from 0 to 9. Figures 2 and 1 shows the result of applying random walk-based t-SNE and the proposed method. The separation of the clusters in Figure 2 indicates that the proposed method has effectively reduced the high-dimensional data into two dimensions while preserving the distinct features of each class. Although Figures 1 and 2 do not look exactly the same, they have both clustered different handwritten digits into a fairly distinct groups, meaning that the proposed method could conduct dimension reduction without the loss of quality. The reason why the two figures do not look exactly the same is due to randomness involving initialization of $\mathbf{Y}$.

The Iris dataset [4] was also used for visualization. The dataset contains 150 samples from three species of Iris flowers: setosa, Virginica and Versicolor. The four features in this dataset includes the length and the width of sepals and petals in centimeters. Figures 3 and 4 illustrate the results of visualization using random walk-based t-SNE and proposed method. The distinction between the three groups illustrates the success in visualization of the dataset. Both techniques achieve a clear visual separation of the species, which suggests that the proposed method maintains the quality of dimension reduction. The similarity in Figures 4 and 3 including the positioning of the embedded data points, is likely because of the straightforward structure of the Iris dataset. If we were using more complex datasets, it might be more challenging to achieve such consistency in results between different dimensional reduction methods.

In summary, when we visualized both the MNIST 2500 dataset and the Iris dataset, the outcomes suggested that the proposed method was capable of conducting dimension reduction without the loss of quality. Both methods were able to embed the data into a simpler form without losing the general structure of the data.

4.3 Processing Time

We have measure the execution time using MNIST 2500 [3] dataset. Figure 5 shows the execution time for varying numbers of threads. The experiment began with a single thread, and then additional threads were introduced in increments of 8, culminating in a total of 64 threads.

As shown in Figure 5, the parallelization was done perfectly as the execution time has dramatically dropped according to the number of threads. A significant decrease in processing time is observed as the number of threads increases. This steep decrease from a single thread to approximately 10 threads illustrates the initial benefits of parallel computation. After the 10 threads, the graph illustrates a fairly stable trend. This could originate from several factors in parallel computing, such as synchronization overhead, the communication time between threads etc.

Figure 6, 7, and 8 represents the execution time for Algorithm 1, 2, and 3, respectively,.

Figure 6 shows a steep decline in execution time as the number of threads increases, indicating significant gains from parallelization in the initial phase. However, it is important

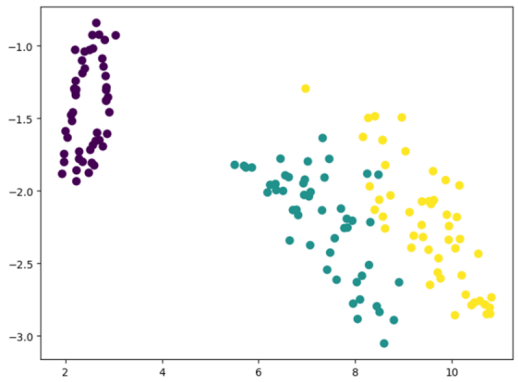


Figure 3 Iris dataset with Random Walk based t-SNE

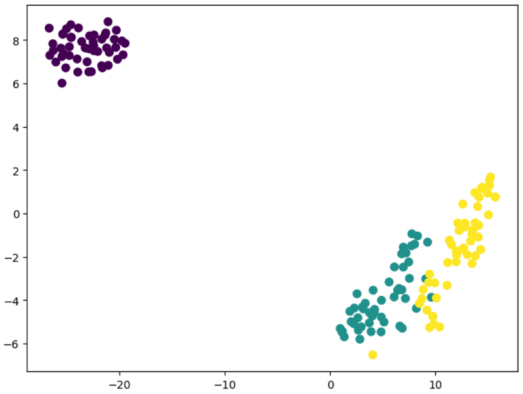


Figure 4 Iris dataset with proposed method

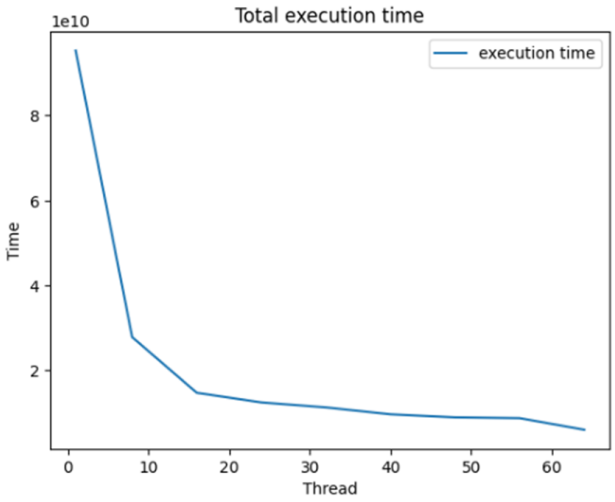


Figure 5 Execution time of Parallel t-SNE on MNIST 2500

to note that for this case, the actual impact of this step on the overall execution time was minimal. This implies that, although parallelization makes this step faster, it was not the critical factor in the total computation time. This maybe due to reduction in dimension in the prepossessing stage where the dimension of MNIST has been reduced to 30.

Figure 7 also shows a sharp reduction in execution time as more threads are utilized. This step has utilised parallel processing and modified choleky factorization. Given that this step is the primary contributor to the total execution time, the efficiency gains in Algorithm 2 are critical to the overall performance improvement of the method.

Figure 8 illustrates a less steep reduction in the execution time compared to the others. This could be due to factors like the iteration in gradient descent method that do not decrease linearly with the addition of more threads.

The substantial reduction in execution time was achieved mainly through the refinements in Algorithm 2. This was what decisively improved the method's efficiency. The other steps, while improved by parallelization, play secondary roles in the method's overall performance enhancement.

To sum up, by using modified cholesky factorization and parallel programming, we have successfully demonstrated that our method reduces computational time significantly, which has addressed one of the biggest disadvantages with original t-SNE.

5 Related Work

Previous efforts have explored enhancing t-SNE's efficiency through parallel processing. For example, Lopes et al. (2020) developed a parallel version of t-SNE to aid in visualizing smart city data [11]. Their work involved three main phases: prepossessing, joint probability calculations, and execution of t-SNE, all of which were parallelized using C and OpenMP.

In their prepossessing phase, parallel computing was applied to speed up the calculation of eigenvalues and eigenvectors as well as the computation of the distance matrix when undergoing PCA. For calculating joint probabilities, tasks such as Gaussian kernel calculations and entropy estimations were performed concurrently. Finally, in the execution phase, the joint probability calculations, the Student-t distribution computations, and the gradient calculations were all parallelized.

In contrast, our research advances the field by applying parallel computing specifically to random walk-based t-SNE for the first time, as far as the author is aware.

6 Conclusion

We have presented an advanced methodology for executing random walk-based t-SNE, utilizing parallel computing and Modified Cholesky factorization to enhance computational efficiency. Our approach has succeeded in significantly reducing processing times, with scalability that effectively corresponds with the addition of processing threads. The results demonstrate the feasibility of our method, as evidenced by the substantial decrease in execution time while maintaining the quality of data visualization. This approach not only offers a faster alternative to traditional random walk-based

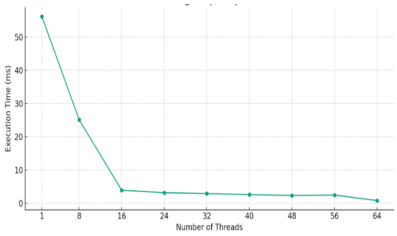


Figure 6 Creating graph Laplacian

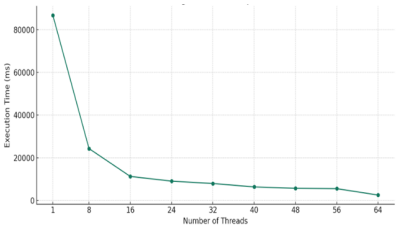


Figure 7 Computing pairwise similarity

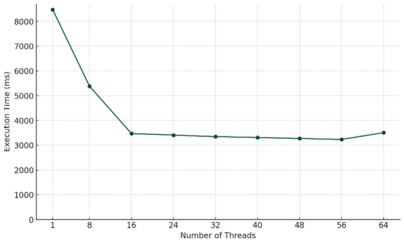


Figure 8 Updating embedded data

t-SNE methods but also provides a path for tackling larger and more complex datasets, which were previously impractical to process.

7 Discussion

A primary concern with the proposed method is its memory footprint, which expands proportionally with the dataset size, potentially limiting its application to extremely large datasets. With the proposed methodology, the computer stores at least three $N \times N$ matrices during Modified Cholesky factorization. As the number of data points grows significantly large, this requirement may become impractical. Future research will be directed towards optimizing memory usage, potentially through the refinement of the data structures employed. We aim to investigate techniques for partitioning the data processing, enabling our method to be both adaptable and memory-efficient. Such strategies might include mini-batching or streaming data processing, each of which can manage memory usage dynamically.

References

[1] Walid M Abdelmoula, Benjamin Balluff, Sonja Englert, Jouke Dijkstra, Marcel JT Reinders, Axel Walch, Liam A McDonnell, and Boudewijn PF Lelieveldt. Data-driven identification of prognostic tumor subpopulations using spatially mapped t-sne of mass spectrometry imaging data. *Proceedings of the National Academy of Sciences*, 113(43):12244–12249, 2016.

[2] Ejaz Ahmed, Ibrar Yaqoob, Ibrahim Abaker Targio Hashem, Imran Khan, Abdelmuttlib Ibrahim Abdalla Ahmed, Muhammad Imran, and Athanasios V Vasilakos. The role of big data analytics in internet of things. *Computer Networks*, 129:459–471, 2017.

[3] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[4] R. A. Fisher. Iris. UCI Machine Learning Repository, 1988. DOI: https://doi.org/10.24432/C56C76.

[5] Yasuhiro Fujiwara, Yasutoshi Ida, Sekitoshi Kanai, Atsutoshi Kumagai, and Naonori Ueda. Fast similarity computation for t-sne. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 1691–1702. IEEE, 2021.

[6] PE Gill. W. murray, and mh wright. *Practical Optimization*, pages 212–229, 1981.

[7] Leo Grady. Random walks for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 28(11):1768–1783, 2006.

[8] Nicholas J Higham. Cholesky factorization. *Wiley interdisciplinary reviews: computational statistics*, 1(2):251–254, 2009.

[9] H Hotelling. Analysis of a complex of statistical variables into principal components. warwick & york. *Inc, Baltimore, Maryland*, 1933.

[10] Matteo Lionello, Luca Pietrogrande, Hendrik Purwins, and Mohamed Abou-Zleikha. Interactive exploration of musical space with parametric t-sne. In *15th Sound and Music Computing Conference (SMC 2018)*, pages 200–208. Sound and Music Computing Network, 2018.

[11] Maximiliano Araújo Da Silva Lopes, Adrião D Dória Neto, and Allan De Medeiros Martins. Parallel t-sne applied to data visualization in smart cities. *IEEE Access*, 8:11482–11490, 2020.

[12] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[13] William Robson Schwartz, Aniruddha Kembhavi, David Harwood, and Larry S Davis. Human detection using partial least squares analysis. In *2009 IEEE 12th international conference on computer vision*, pages 24–31. IEEE, 2009.

[14] Laurens Van Der Maaten. Accelerating t-sne using tree-based algorithms. *The journal of machine learning research*, 15(1):3221–3245, 2014.

[15] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

ブロックチェーン技術を活用したデータ検証可能な分散データベースの性能に関する検討

坂本 明穂† 小口 正人†

† お茶の水女子大学 〒 112–8610 東京都文京区大塚 2-1-1
E-mail: †akiho@ogl.is.ocha.ac.jp, oguchi@is.ocha.ac.jp

あらまし 複数ユーザ間でのデータの共有方法として分散データベースの利用が考えられるが，共有されたデータを有効に活用するためにはデータベース上に保存されたデータが正しいことを保証する必要がある．そこでデータの保存時にブロックチェーンによる検証プロセスを行うことで健全なデータのみをデータベースに保存することができる．しかしブロックチェーンはデータ検証プロセスを持つために一般的なデータベースと比べて処理性能が劣ることが知られている．そこでブロックチェーンの 1 つである Hyperledger Iroha について，一般的なデータベースである PostgreSQL との処理性能の比較を行う．また Hyperledger Iroha のトランザクション処理における特徴について調査する．

キーワード 並列・分散処理，分散システム，分散台帳

A Study on the Performance of Data Verifiable Distributed Database Using Blockchain Technology

Akiho SAKAMOTO† and Masato OGUCHI†

† Ochanomizu University,
2–1–1 Otsuka, Bunkyou-ku, Tokyo 112–8610 Japan
E-mail: †akiho@ogl.is.ocha.ac.jp, oguchi@is.ocha.ac.jp

1. はじめに

分散データベース上でデータを管理することで，複数のユーザ間でのデータ共有が容易になる．例えば製造業における各部品製造段階での二酸化炭素排出量を分散データベース上で管理することで，複数の企業にまたがるデータを一元的に管理することができる．データの一元的な管理により，製品の製造時に各過程で排出される二酸化炭素量の追跡やカーボンフットプリントの管理が簡単になるため，製造業における二酸化炭素排出量の削減などが期待できる．しかし保存されたデータを有効に活用するためには，管理対象となるデータが正しいものであることを保証する必要がある．すなわち分散データベースには，保存される全てのデータに対してその健全性を検証してから保存するような機能が求められる．しかし分散データベース上に保存されるデータ量が増加すると，データベースに保存される全てのデータに対して実用的な時間で検証することが難しくなる．そこで検証対象となるデータをメタデータに限ることで検証対象となるデータ量を減らし，データ検証にかかる時間の削減を行う．その結果，実時間でのデータ検証が可能な分散データベースの構築が可能となる．

データ検証を行うプラットフォームとしてブロックチェーンの利用が考えられる．ブロックチェーンはネットワークに送信されたトランザクションが正しいかを検証するメカニズムを持っているため，この機能を利用して分散ストレージを作成することで目的のデータベースを構築することができる．

ブロックチェーンには Bitcoin などの不特定多数のユーザが参加できるパブリックブロックチェーンと，Hyperledger Fabric などの事前に認証されたユーザしか参加できないプライベートブロックチェーンがある．分散データベースによるデータの共有は，複数の企業間などの特定のユーザ間で行われることが想定されるため，プライベートブロックチェーンが適切と考えられる．

プライベートブロックチェーンにおけるデータ検証では，ブロックチェーンに参加しているノード間でメッセージをやり取りし，データが正しいという合意を形成することが必要となる．ノード間でのメッセージのやり取りはネットワークを通して複数回行われるため，ブロックチェーンでのデータの処理速度は通常のデータベースよりも遅くなる．本研究では，プライベー

トブロックチェーンの 1 つである Hyperledger Iroha の性能について通常のデータベースとの性能の比較を行う．またネットワークサイズを変化させて，処理するトランザクション数が増加することによるレイテンシやスループットへの影響を計測する．

2. Hyperledger Iroha

プライベートブロックチェーンの中では Hyperledger プロジェクトの 1 つである Hyperledger Fabric が有名であるが，同じ Hyperledger プロジェクトで直近にリリースされたものに Hyperledger Iroha がある．Hyperledger Iroha は Hyperledger Fabric よりも効率の良いコンセンサスアルゴリズムを採用しているという特徴がある．

2.1 Hyperledger Fabric

Hyperledger Fabric v0.6 ではコンセンサスアルゴリズムに PBFT(Practical Byzantine Fault Tolerance) が，Hyperledger Fabric v1.0 では BFT-SMaRT が用いられている．両者の違いは信頼性の向上と署名の評価のためのマルチコア計算である [1]．PBFT 方式のコンセンサスアルゴリズムでのトランザクション処理の流れは以下の通りである．まずクライアントは各ノードにトランザクションを送信し，ノードはそれをシミュレーションしてその結果であるエンドースメントをクライアントに送信する．各ノードから返ってきたエンドースメントが一致していれば，クライアントはそれらを当該トランザクションに含めてオーダリングサービスに送信する．オーダリングサービスは複数トランザクションから成るブロックを作成し，それをノードに配布する．ノードはブロックに含まれるトランザクションを検証して，検証を通ったものだけを台帳に反映する [5]．

2.2 Hyperledger Iroha

Hyperledger Iroha では YAC(Yet Another Consensus) と呼ばれるコンセンサスアルゴリズムを採用している．また YAC 方式ではネットワークを構成するノードの一部が悪意のあるユーザやオフラインになったとしても正常に動作することが保証されている．

図 1 に Hyperledger Iroha ネットワークにおけるトランザクションの処理の流れを，図 2 にコンセンサス形成におけるピア間でのメッセージのやり取りを示す．ただしクライアントはトランザクションを生成してオーダリングサービスに送信する役割をもち，ピアはプロポーザルに含まれるトランザクションを検証して合意し，トランザクションをブロックへ格納する役割をもつ．またオーダリングサービスはクライアントから送信されたトランザクションを受け取り，1 つ以上のトランザクションから成るプロポーザルを作成する役割を果たす．

1. クライアントがピアにトランザクションを送信する．
2. トランザクションを受け取ったピアはオーダリングサービスにトランザクションを転送する．
3. オーダリングサービスは，一定時間内に受け取ったトランザクションをまとめて順番に並べ，プロポーザルを作成する．
4. オーダリングサービスは各ピアにプロポーザルを送信する．
5. ピア間でコンセンサスを形成する．
   a. プロポーザルを受け取ったピアは，プロポーザルに含まれるトランザクションが正しいかを検証する．
   b. プロポーザルに含まれる正しいトランザクションのみからハッシュ値を計算する．
   c. ピアは正しいトランザクションとハッシュ値を含むブロックを生成する．
   d. ハッシュ値と初期のピア順から新しいピア順を計算する．
   e. 求めたピア順の先頭ピアにプロポーザルハッシュとブロックハッシュ，ピアの署名を含む投票を送る．
   f. 先頭ピアは 2/3 以上の投票を得ると，ネットワークを構成するピアにコミットメッセージをブロードキャストする．
6. 各ピアは合意されたトランザクションのブロックをチェーンに追記する．

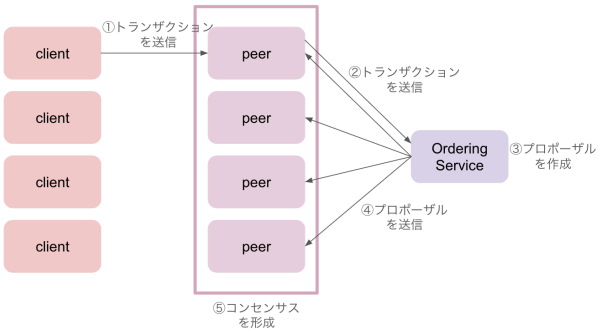


図 1: トランザクションの処理プロセス

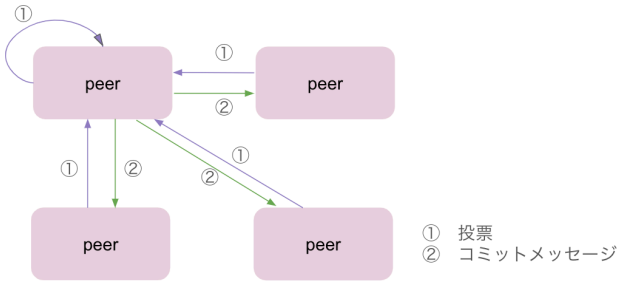


図 2: コンセンサス形成時のメッセージのやり取り

3. 関 連 研 究

Muratov ら (2018) は異なるネットワークサイズに対して，投票ステップ遅延を変化させたときのスループットの変化および，不安定な挙動を示したノードの数について測定を行っている [1]．ここで投票ステップ遅延とは，プロポーザルを検証し投票によって次のブロックを決定する期間を指す．4，16，28，64 ノードのネットワークに対して測定を行った結果，4 ピア程度の小さなネットワークサイズでは小さな投票遅延値を設定す

ることがスループットの向上に有効である一方，ネットワークサイズが大きくなると小さな投票遅延値ではタイムアウトが発生してコンセンサスに失敗する可能性があることが示された．したがってネットワークのスループットを最大にするためには，28 ノード以下では投票遅延値を 1ms に設定し，64 ノードでは 20ms に設定するとよい．

Woznica ら (2022) は，Hyperledger Fabric，Hyperledger Sawtooth，Hyperledger Iroha について，トランザクション送信レート，ブロックサイズ，ネットワークトラフィックモード，ネットワークサイズを変化させたときのレイテンシとスループットの変化を測定している [2]．Hyperledger Iroha について次のような結果が得られた．トランザクション送信レートを 10，20，30，40，50，60，70，80，90，100，150，200 と変化させて最小レイテンシを計測すると両者に厳密な相関関係は見られない．また同条件でトランザクション送信レートを変化させても平均レイテンシに影響はないが，ネットワークサイズを 5，10，20，30，40 と増加させると平均レイテンシは大きくなる．さらにトランザクション送信レートが小さい (20tps) ときスループットは概ね一定であるが，トランザクション送信レートが高い (100tps) とき，スループットはネットワークサイズが大きくなるにつれて減少する．またブロックサイズを 10 と 50 の 2 種類に対して平均レイテンシとスループットを計測すると，ほとんどのトランザクション送信レートにおいてブロックサイズが大きいほど平均レイテンシが短くなり，ブロックサイズが小さくなるほどスループットが低下する．

Anh Dinh ら (2017) はブロックチェーンのベンチマークツールである Blockench を用いて Ethereum，Parity，Hyperledger Fabric v0.6 の性能評価を行った [3]．Hyperledger Fabric は Ethereum や Parity よりも一貫して優れたパフォーマンスを示したが，16 ノード以上へのスケールアップには失敗した．また Hyperledger Fabric と Ethereum のボトルネックはコンセンサスプロトコルだが，Parity ではトランザクションの署名がボトルネックである．

Fan ら (2020) はブロックチェーンの性能評価に関する調査を行なっている [4]．ブロックチェーンの性能評価手法にはモニタリングやベンチマーキング，実験，シミュレーションなどがある．パブリックブロックチェーンとプライベートブロックチェーンでは適している手法が異なる．例えばモニタリングはベンチマーキングと比べてパブリックブロックチェーンの評価に適しているが，ベンチマーキングはプライベートブロックチェーンの評価に適している．またモニタリングよりもベンチマーキングの方が拡張性が高い．

4. 実　　験

4.1 実験概要

本研究では Hyperledger Iroha と一般的なデータベースのパフォーマンスについての比較および，Hyperledger Iroha のパフォーマンスやスケーラビリティについて評価を行う．図 3 のように，Hyperledger Iroha ネットワークの各ピアは PostgreSQL をデータベースに持ち，ピア同士はネットワークで接続されている．

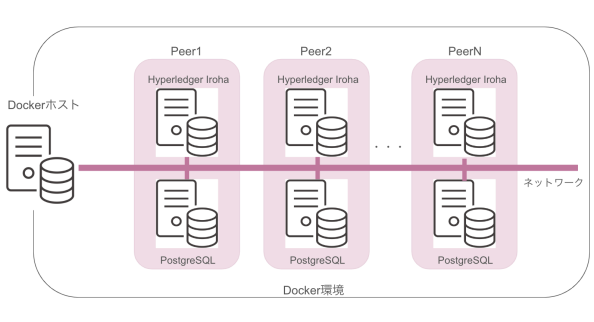


図 3: Hyperledger Iroha 実験環境

そこで PostgreSQL データベースにデータを追加する操作にかかる時間と Hyperledger Iroha の実行時間との比較を行う．PostgreSQL ではコネクションを確立し，1 つのクエリを処理した後，コネクションが切断されるまでにかかる時間を計測する．これは，Hyperledger Iroha の各ピアが各自のデータベースを書き換える際に行うことが想定される処理であり，この実験によって得られる PostgreSQL の実行時間と Hyperledger Iroha の実行時間の差は，Hyperledger Iroha のコンセンサス形成にかかる時間とすることができる．PostgreSQL データベースへの挿入操作にかかる時間の計測は，データベースにレコードが作成された時刻を記録するカラム created_at を作成し，挿入する際に現在時刻のタイムスタンプを取得し created_at に書き込むことで行った．また Hyperledger Iroha ネットワークを構成するピア数を 5，10，20，30 として，各ピア数のネットワークが 1 つのトランザクションを処理するのにかかる時間を計測する．Hyperledger Iroha の実行時間の計測にはアカウント A からアカウント B にアセットを転送するコマンド transferAsset を使用する．トランザクションが Hyperledger Iroha ネットワークに送信されてからトランザクションの処理が終わるまでの時間を JavaScript の performance.now() を用いて計測する．

次にブロックチェーン基盤の性能測定ツールである Hyperledger Caliper を用いて，Hyperledger Iroha への単位時間あたりの送信トランザクション数の変化およびネットワークサイズが変化したときのレイテンシとスループットの測定を行う．5，10，20，30 ピアから構成される Hyperledger Iroha ネットワークを対象とし，1 秒あたりに送信するトランザクション数を 10，20，30，40，50，60，70，80，90，100，150，200 と変化させて計測を行う．

4.2 実験環境

実験に使用したサーバの性能は Ubuntu20.04LTS の OS，Intel(R) Xeon(R) Silver 4314 CPU @ 2.40GHz の CPU，コア数 32，スレッド数 64，メモリ 192GB である．

また PostgreSQL 単体および Hyperledger Iroha ネットワークを構成するピアは Docker コンテナを用いて構成される．Hyperledger Iroha の実験環境は図 3 の通りである．Hyperledger Iroha の各ピアは Hyperledger Iroha プロセスを実行するコンテナと PostgreSQL データベースを実行するコンテナから成

る．ピア同士は仮想ネットワークによって接続される．

使用した Hyperledger Caliper のバージョンは 0.3.2 である．

4.3 実験結果

4.3.1 PostgreSQL と Hyperledger Iroha の性能比較

PostgreSQL と Hyperledger Iroha でトランザクション 1 つの処理にかかる時間は表 1 のようになった．PostgreSQL では 10ms 程度で 1 つのクエリを処理できたのに対し，Hyperledger Iroha では 1 つのトランザクションの処理に約 570 倍の 5700s 程度の時間が必要となった．したがって Hyperledger Iroha での PostgreSQL へのデータの書き込みにかかる時間の割合はトランザクションの処理にかかる時間全体の約 0.2%であり，Hyperledger Iroha ではトランザクション処理時間のほとんどをコンセンサス形成に費やしていることがわかる．また Hyperledger Iroha では構成ピア数が増加しても，処理時間に大きな差が見られなかった．

表 1: PostgreSQL と Hyperledger Iroha での処理時間

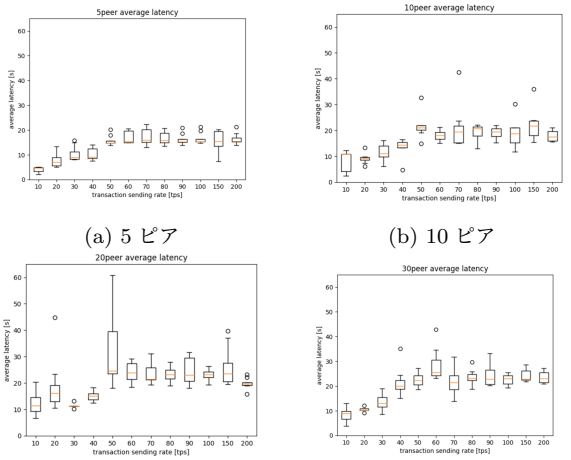
測定対象	測定結果 [ms]
PostgreSQL	10.3
Hyperledger Iroha 5peer	5696.27
Hyperledger Iroha 10peer	5641.27
Hyperledger Iroha 20peer	5777.56
Hyperledger Iroha 30peer	5804.01

4.3.2 トランザクション送信レートと平均レイテンシ

次に，Hyperledger Caliper を用いて Hyperledger Iroha ネットワークのレイテンシとスループットを計測した．

まずトランザクション送信レートを変化させたときの平均レイテンシについて評価を行う．図 4 に 5，10，20，30 の各ネットワークサイズに対してトランザクション送信レートを変化させたときの平均レイテンシの変化を示す．毎秒 50 トランザクションまではトランザクション送信レートの増加とともに平均レイテンシが増加する傾向が見られたが，50 トランザクション以降はトランザクション送信レートを増加させても平均レイテンシの値は横ばいとなった．

トランザクション送信レートが 50tps までのとき，トランザクション送信レートの増加に伴いネットワークが単位時間あたりに受け取るトランザクション数が増加したため，平均レイテンシが増加した．一方 50tps 以降ではトランザクション送信レートを増加させても平均レイテンシは増加せずに概ね一定の値となったため，単位時間あたりにネットワークに送信されるトランザクション数が一定になったと考えられる．よって単位時間あたりにオーダリングサービスからピアに送信されるプロポーザル数には上限が定められていると推測できる．オーダリングサービスが受け取るトランザクション数が一定値を超えると，超えた分のトランザクションはすぐにピアに送信されるのではなくオーダリングサービス内で管理され，時間をおいてピアに送信される．その結果送信されるトランザクション数が増加しても，単位時間あたりにピアが処理するプロポーザル数は一定になり，平均レイテンシの値が一定になったと考えられる．



(a) 5 ピア　(b) 10 ピア

(c) 20 ピア　(d) 30 ピア

図 4: トランザクション送信レートを変化させたときの平均レイテンシの変化

4.3.3 トランザクション送信レートとスループット

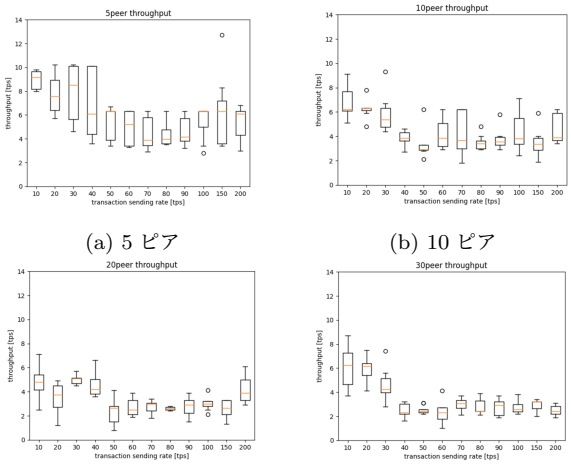
次にトランザクション送信レートを変化させたときのスループットの変化を図 5 に示す．ネットワークを構成するピアが 5 つのとき，単位時間あたりのスループット数は 70tps までは減少し，その後横ばいになるが 100tps 以降は増加した．10 ピア以降では，トランザクション送信レートが 30tps または 40tps まではスループットは減少し，30tps から 40tps 以降ではスループットの値に大きな変化は見られなかった．

トランザクション送信レートの値が小さいとき，トランザクション送信レートを増加させると単位時間あたりにネットワークに送信されるトランザクション数が増加する．したがって Hyperledger Iroha ネットワークにかかる負荷が大きくなり，スループットが低下したと考えられる．またトランザクション送信レートがある程度大きくなったとき，それ以上増加させてもスループットの値が横ばいとなった原因として，ピアに送信されるトランザクション数が一定になったことが考えられる．クライアントから送信されるトランザクション数が増加しても，単位時間あたりにピアに送信され処理されるトランザクション数が一定であったためスループットが一定になったと考えられる．

4.3.4 ネットワークサイズと平均レイテンシ

トランザクション送信レートを一定にして，ネットワークサイズを変化させたときの結果を図 6 に示す．トランザクション送信レートが 20tps 以下のときにネットワークサイズを大きくすると，20 ピアまでは平均レイテンシは増加したが 30 ピアでは減少した．トランザクション送信レートが 30tps 以上のときはネットワーク構成ピア数の増加とともに平均レイテンシも増加した．

ネットワークを構成するピア数が増えるとコンセンサス形成に参加するピア数が多くなるため，プロポーザルのコンセンサス形成時に必要なメッセージの総数が増加する．よってピア数が増えるにしたがって平均レイテンシも増加したと考えられる．



(a) 5 ピア　(b) 10 ピア

(c) 20 ピア　(d) 30 ピア

図 5: トランザクション送信レートを変化させたときのスループットの変化

4.3.5 ネットワークサイズとスループット

ネットワークを構成するピア数を変化させたときのスループットの変化を図 7 に示す．ネットワーク構成ピア数を増加させるとスループットは概ね低下した．ただしトランザクション送信レートが 10tps，20tps のときは 20 ピアのときスループットが最低となった．

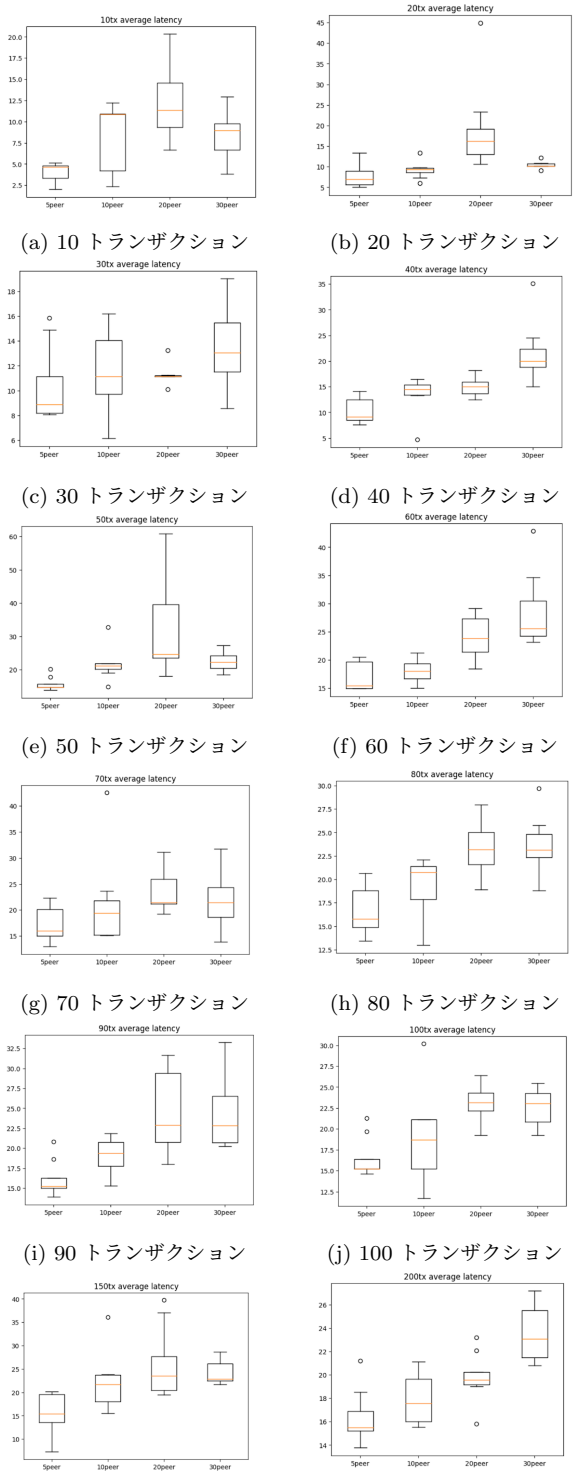
ネットワークサイズが大きくなると，コンセンサス形成時に送信されるメッセージ数が増加する．よって 1 つのプロポーザルの処理にかかる時間が増加し，スループットが低下すると考えられる．

また測定ではトランザクション送信レートが 10tps および 20tps のとき，30 ピアよりも 20 ピアの方が性能が低いという結果が得られた．Hyperledger Iroha を構成する各コンテナは CPU の各スレッドに割り当てられるためコンテナ内の処理は他のコンテナの影響を受けない．しかしネットワークや I/O は共有しているため，他のコアの処理のためにそれらに高い負荷がかかっている場合がある．したがって共有部分において他のコアを原因とする遅延が発生する可能性があり，測定された性能にある程度ランダム性が存在すると考えられる．

5. まとめと今後の課題

分散データベースのデータ検証プラットフォームとして導入が期待されるブロックチェーンについて，プライベートブロックチェーンの 1 つである Hyperledger Iroha を取り上げてその性能の評価を行った．

PostgreSQL とのデータ処理速度の比較により，Hyperledger Iroha では PostgreSQL よりもトランザクションの処理にかかる時間が非常に長いことがわかり，トランザクション処理のほとんど全ての時間がコンセンサスの形成に費やされているという結果が得られた．またトランザクション送信レートをある程度まで増加させていくと．平均レイテンシは高くなりスループットは低下することがわかった．トランザクション送信レートが一定値を超えると，それ以上単位時間あたりに送信するトランザクション数を増加させても平均レイテンシとスループットに大きな変化は見られなくなった．ネットワークサイズを変化させたとき，サイズを大きくすると平均レイテンシは増加し，スループットは低下することがわかった．



(a) 10 トランザクション　(b) 20 トランザクション

(c) 30 トランザクション　(d) 40 トランザクション

(e) 50 トランザクション　(f) 60 トランザクション

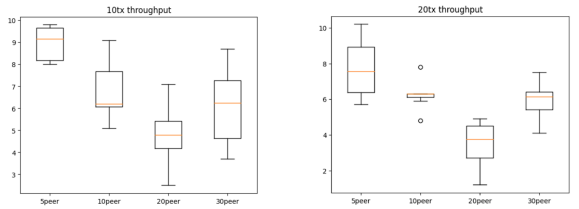
(g) 70 トランザクション　(h) 80 トランザクション

(i) 90 トランザクション　(j) 100 トランザクション

(k) 150 トランザクション　(l) 200 トランザクション

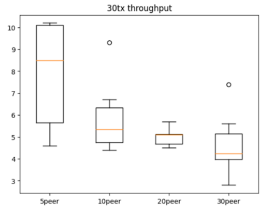
図 6: ネットワークサイズを変化させたときの平均レイテンシの変化

今後は仮想ネットワークではなく実ネットワークでピア同士を繋いで実験を行うことを検討している．また実験で使用した Hyperledger Iroha ネットワークを構成するピアは全て同等の

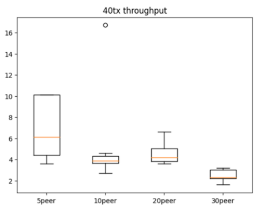


(a) 10 トランザクション

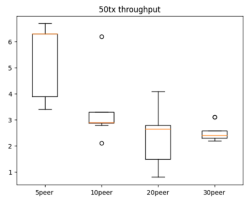
(b) 20 トランザクション



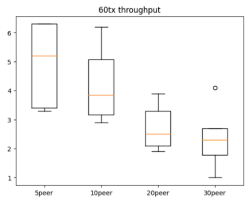
(c) 30 トランザクション



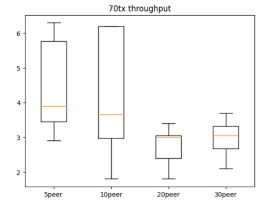
(d) 40 トランザクション



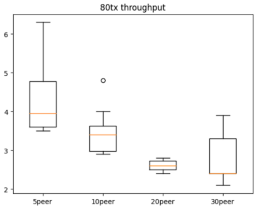
(e) 50 トランザクション



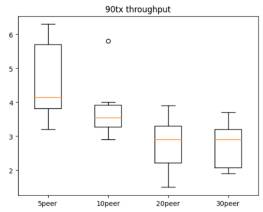
(f) 60 トランザクション



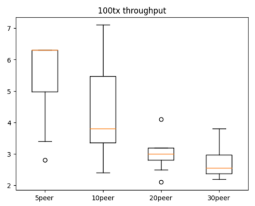
(g) 70 トランザクション



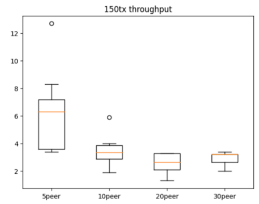
(h) 80 トランザクション



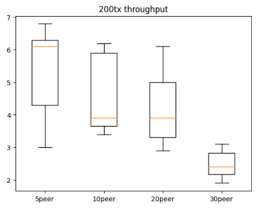
(i) 90 トランザクション



(j) 100 トランザクション



(k) 150 トランザクション



(l) 200 トランザクション

図 7: トランザクション送信レートを変化させたときのスループットの変化

性能を有していたが，同じネットワーク内に異なる性能を持つピアが存在したときの性能の変化についても評価を行いたい．

謝　　辞

本研究は一部，JST CREST JPMJCR22M2 の支援を受けたものである．

文　　献

[1] Fedor Muratov, Andrei Lebedev, Nikolai Iushkevich, Bulat Nasrulin, Makoto Takemiya, “YAC: BFT Consensus Algorithm for Blockchain,” September 3, 2018.

[2] Arnold Woznica, Michal Kedziora, “Performance and Scalability Evaluation of a Permissioned Blockchain Based on the Hyperledger Fabric, Sawtooth and Iroha,” Computer Science and Information Systems, Vol 19, Issue 2, pp. 659-678, 2022.

[3] Tien Tuan Anh Dinh, Ji Wang, Gang Chen, Rui Liu, Beng Chin Ooi, Kian-Lee Tan, “BLOCKBENCH: A Framework for Analyzing Private Blockchainsi,” SIGMOD ’17: Proceedings of the 2017 ACM International Conference on Management of Data, pp. 1085–1100, May 2017.

[4] Caixiang Fan, Sara Ghaemi, Hamzeh Khazaei, Petr Musilek, “Performance Evaluation of BlockchainSystems: A Systematic Survey,” June, 2020.

[5] 齋藤 新, “4. 分散台帳技術におけるコンセンサス・メカニズム,” 情報処理, Vol 61, No. 2, pp. 165-175, 2022.

[6] 佐藤 栄一, “Hyperledger Iroha 入門ーブロックチェーンの導入と運営管理ー,” オーム社, 2020.

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．

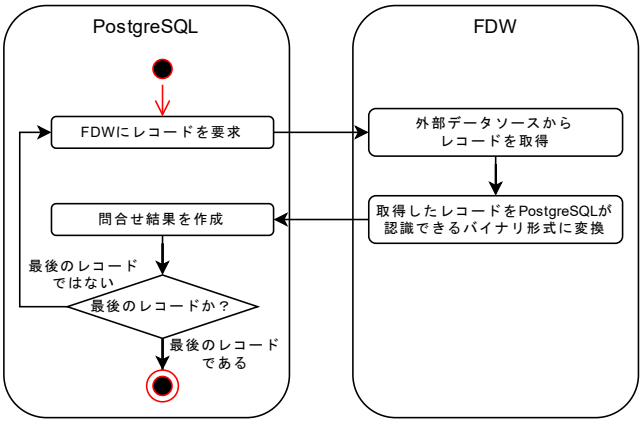


図　1 PostgreSQL における FDW を活用した外部データソースのレコード取得処理の流れ

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．

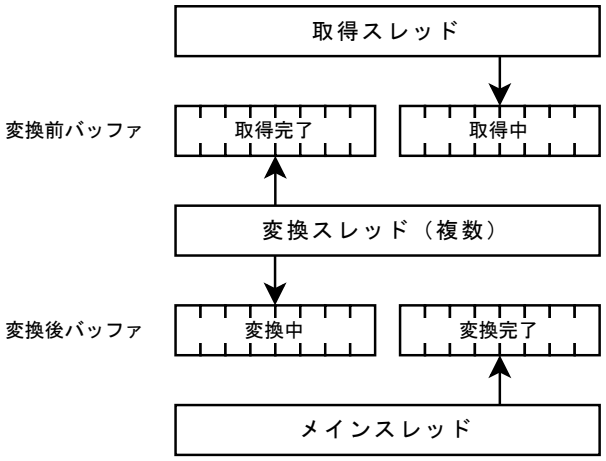


図 2 各スレッドとバッファの関係

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．

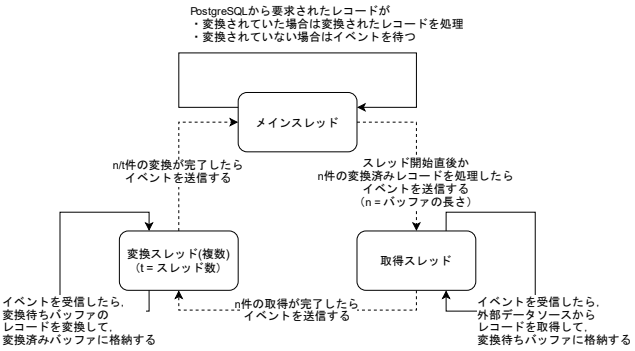


図 3 スレッド間のイベント完了通知

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．

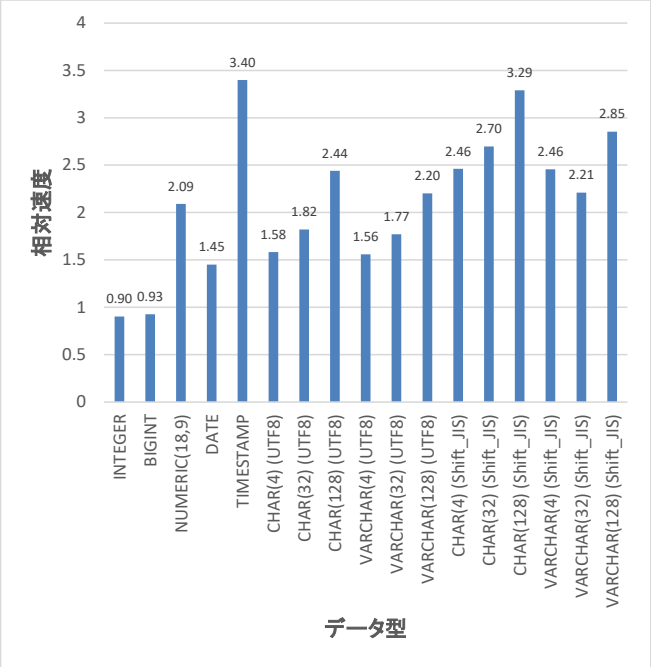


図 6 比較方式に対する提案方式の相対速度
（データ型毎，並列度 4）

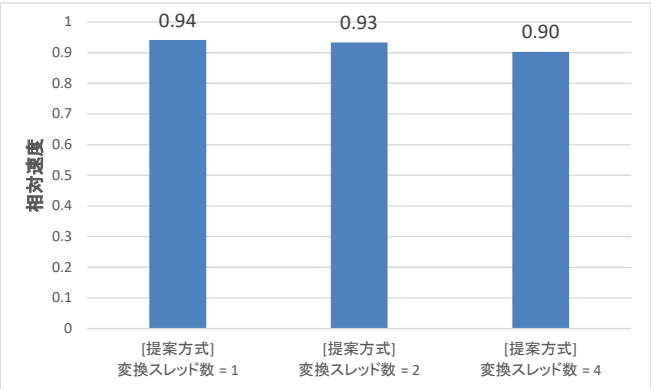


図 7 比較方式に対する提案方式の相対速度
（並列度毎，INTEGER 型参照時）

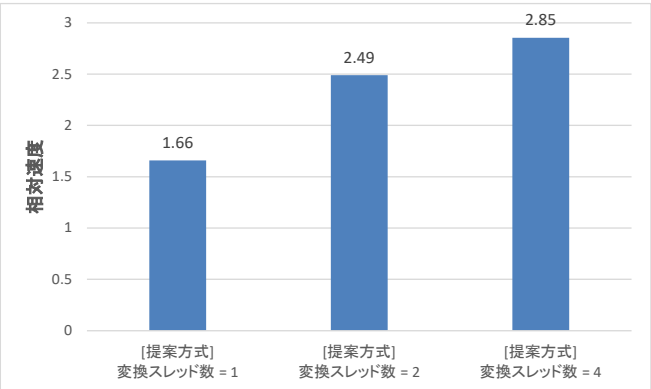


図 8 比較方式に対する提案方式の相対速度
（並列度毎，VARCHAR(128)型 (Shift_JIS)参照時）

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