DEIM2022 J34-3

統合型データベースにおける適応的2相ロックに基づく分散トランザクション制御

三宅 康太† 佐々木勇和† 肖 川† 鬼塚 真†

† 大阪大学大学院情報科学研究科 〒 565–0871 大阪府吹田市山田丘 1-5
E-mail: †{miyake.kouta,sasaki,chuanx,onizuka}@ist.osaka-u.ac.jp

あらまし データベースが分散する環境において各データベースが自律分散的にデータを共有する非集中型データ統合が提案されている．特に，複数のデータベースにおける更新が相互に伝搬する非集中型のアーキテクチャでは，データの一貫性は重要な課題である．既存技術では 2 相ロックを用いた分散トランザクション制御を行うが，更新が伝搬する環境では分散デッドロックが発生しやすく性能が低下する問題がある．そこで，非集中型データ統合アーキテクチャの特性を分散トランザクション制御の観点から分析し，大域的な一貫性を保証する効率的な分散トランザクション制御手法を提案する．提案手法は分散ロックの機構を導入し，全てのデータベースにわたって更新される可能性のあるレコードをトランザクション実行前にロックする保守的な 2 相ロックによる制御を実現する．競合率の高いワークロードにおいて頻発しうる分散デッドロックによるスループットの低下を保守的な 2 相ロックによって防止し，競合率の低いワークロードにおいては保守的な 2 相ロックによる制御のオーバヘッドを削減するため通常の 2 相ロックによる制御に適応的に切り替えることでスループットを向上させる．評価実験により，競合率の高いワークロードにおいて，提案手法が通常の 2 相ロックの性能を上回り，また競合率の低いワークロードにおいても制御の適応的な切り替えにより性能低下を抑制できることがわかった．

キーワード データ統合，分散トランザクション処理

1 はじめに

データ統合とは，複数のデータベースに分散するデータの統合的な利活用を可能にする技術である．分散する各データベースはスキーマ設計が異なるため，従来のデータ統合技術の多くは，各データベースからデータを収集し，単一のグローバルスキーマを利用して統合ビューを作成する集中型のアプローチでデータ統合を実現している [1]．しかしながら近年，生成されるデータ量の爆発的増大を受けて，データの移行コストが大きくなり，この中央集権的なアーキテクチャでは不適切な状況が見られる．こういった背景を受けて，非集中型データ統合アーキテクチャが提案されている [2]．このアーキテクチャでは，サービスプロバイダなどがピアとしてデータ統合に参加し，各ピアが自立分散的に他のピアとデータを共有することでデータの統合的な利活用を実現する．複数のピア間でデータを共有しているため，あるピアでのレコードの更新が他のピアへ連鎖的に伝搬していくことが大きな特徴である．

非集中型データ統合アーキテクチャの多くは，各ピア内のデータの一貫性を保証するものの複数のピアをまたがるデータの大域的な一貫性は保証しない．これは既存技術が，様々な科学分野における膨大な科学データの共有を必要とするバイオインフォマティクスにおける応用をユースケースとして想定していたためである．科学者が他分野のデータについて信頼できるものだけを選択的に共有したいという要望と，データを信頼する基準が科学者によって異なるという側面から，既存技術の想定するユースケースでは大域的な一貫性を保証する必要がなかった．しかしながら，例えば旅行代理店によるホテルや航空券などの予約管理への応用など，より一般的なユースケースを想定する場合，これは重要な課題となる．

より一般的な応用を実現するため，Dejima アーキテクチャ [3], [4] が提案されている．既存技術と比べて更に一般的なユースケースを想定したアーキテクチャであり，非集中型データ統合において大域的な一貫性を保証することに重点を置いている．そのため，Dejima において分散トランザクション制御は不可欠である．そこで本研究では，Dejima のような非集中型データ統合アーキテクチャにおいてデータの大域的な一貫性を保証する効率的な分散トランザクション制御を提案する．

非集中型データ統合アーキテクチャにおいて大域的一貫性を保証する分散トランザクション制御の設計にあたり，このアーキテクチャの特性に注意する必要がある．そこで本論文では，非集中型データ統合アーキテクチャ Dejima [3], [4] を対象に非集中型アーキテクチャを分散トランザクション制御の観点から分析し，効率的な分散トランザクション制御を提案する．提案手法は Family Record Set の概念を導入することで，あるレコードの更新が伝搬されうる全てのレコードをトランザクション実行前に特定し，保守的な 2 相ロックを実現する．競合率の高いワークロードにおいては保守的な 2 相ロックによる制御によって分散デッドロックによるスループットの低下を防止する．一方，競合率の低いワークロードでは通常の 2 相ロックによる制御に適応的に切り替えることで，保守的な 2 相ロックによる制御のオーバヘッドを削減しスループットを向上させる．

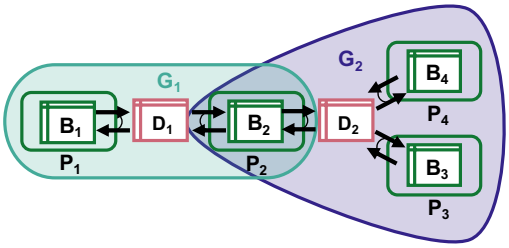

図 1　Dejima アーキテクチャの例．各ピアがベーステーブルと Dejima テーブルを保持し，他のピアと共に Dejima グループを形成し Dejima テーブルを共有することでデータ共有を行う．

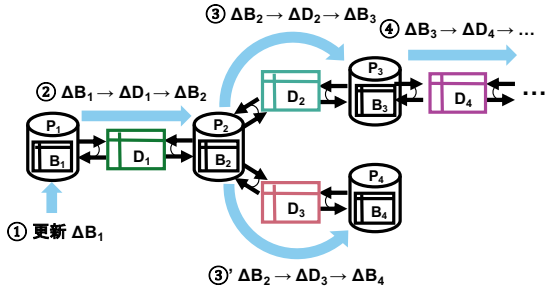

図 2　Dejima における更新伝搬の例

本論文の構成は以下の通りである．2 章において非集中型データ統合アーキテクチャである Dejima の概要を説明し，分散トランザクション制御を設計する際に考慮すべき点について分析し整理する．3 章では分析した点に基づき Dejima に適したトランザクション制御手法を提案する．4 章にて，本研究で実施した評価実験について説明する．5 章で関連研究について説明し，6 章にて本論文をまとめ，今後の課題を論ずる．

2　Dejima

Dejima アーキテクチャとは，双方向変換を利用した更新可能ビューを用いた柔軟なデータ統合を可能とするアーキテクチャである [3]．本節では，このアーキテクチャの概要及び分散トランザクション制御の必要性について説明し，応用を紹介する．

2.1　概　　要

Dejima は以下の 4 つの構成要素から成る．

- ピア：サービスプロバイダなど，固有のデータベースを有しデータ統合に参加する主体．
- ベーステーブル：各ピアがデータベース内に保有する各ピア固有のテーブル．
- Dejima グループ：データを共有するピア同士で構成するグループ．
- Dejima テーブル：Dejima グループ内で共有する，双方向変換を用いた更新可能ビュー．各ピアのベーステーブルから導出される．

Dejima のアーキテクチャの例を図 1 に示す．P_1，P_2，P_3，P_4 がピアにあたり，B_1，B_2，B_3，B_4 が各ピアの保有するベーステーブルにあたる．P_1 と P_2 が Dejima グループ G_1 を，P_2 と P_3 と P_4 が Dejima グループ G_2 を構成している．G_1 内で Dejima テーブル D_1 を，G_2 内で Dejima テーブル D_2 を共有している．Dejima テーブルは各ピアのベーステーブルから導出されるビューであり，双方向変換技術を利用した更新可能ビューである [4], [5]．すなわち，このビューに更新が発生した場合，このビューのソースとなっているベーステーブル全てにおいて，更新後のビューが導出できるデータとなるように更新が行われる．各ピアにおけるベーステーブルのスキーマやデータはピアごとに異なるが，これらのベーステーブルから計算されるビューである Dejima テーブルは常に同一となる．したがって，Dejima テーブルはその Dejima グループにおけるグローバルスキーマとしての役割を担う．なお，本研究では SPJU クエリのみで定義された Dejima テーブルのみ扱う．

Dejima における更新伝搬の例を図 2 に示す．あるテーブルやビュー R における更新を ΔR と表すと，図 2 において更新伝搬は次のような過程で行われる．

(1) P_1 の B_1 に対し，ユーザによって更新 ΔB_1 が実行される．
(2) ΔB_1 によって，B_1 のビューである D_1 において更新 ΔD_1 が発生する．
(3) この Dejima テーブル D_1 を共有している P_2 に対して ΔD_1 の情報を送信する．
(4) これを受け取った P_2 において，双方向変換を用いて ΔD_1 の情報をベーステーブル B_2 における更新 ΔB_2 に変換し，これを適用する．
(5) D_1 以外に保有する Dejima テーブルがある場合，これらにおいて更新が発生する．ここでは ΔD_2 と ΔD_3 が発生している．
(6) 手順 (3) と同様に，これらの Dejima テーブルを共有するピアに対してこの更新情報を送信する．
(7) 以降，Dejima テーブルの更新が発生しなくなるまで手順 (4)〜(6) を繰り返す．

以上のような仕組みで，双方向変換による更新可能ビューを用いて，スキーマの異なるピアの間でデータ共有が行われる．

2.2　分散トランザクション制御

2.2.1　分散トランザクション制御の必要性

Dejima アーキテクチャにおける分散トランザクション制御の必要性について説明する．このアーキテクチャにおいて，大きく 2 つの要因から分散トランザクション制御が必要とされている．

第一に，ユースケースの一般性が挙げられる．従来の非集中型データ統合アーキテクチャはそのユースケースの特性から大域的なデータの一貫性を保証する必要がない．一方，Dejima アーキテクチャはより一般的なユースケースを想定しているため，大域的なデータの一貫性の保証は重要な課題となる．例えば，旅行代理店が飛行機の座席予約とホテルの部屋の予約を一括に行いたい場合など，大域的なデータの一貫性を保証しなけ

ればならない状況が想定され，分散トランザクション制御が必要とされている．

第二に，このアーキテクチャは各ピアの自律性を重視しているため，各ピアにおいて独自の整合性制約などの設定を許容している．これを考慮すると，あるピアでの更新が他のピアに伝搬される時，伝搬先のデータベースにおいてその更新が整合性制約に違反する場合がある．この時，大域的なデータの一貫性を保証するためには，問題が生じたピアだけでなくこの更新の影響を受けた全てのピアにおいてアボート処理を実行する必要がある．したがって分散コミットプロトコルが必要であり，分散トランザクション制御はこのアーキテクチャにおいて必要不可欠なものとなっている．

2.2.2 Dejima における分散トランザクション制御

非集中型データ統合アーキテクチャはピアの自律性を重視している．分散トランザクション制御手法の多くは, グローバルトランザクションマネージャのような大域トランザクション管理コンポーネントに依存しているが, 各ピアの自律性を実現するためには, このような大域的なコンポーネントを利用しない手法が望ましい．また同様の観点から，各ピアは任意の RDBMS を運用できることが望ましく，分散トランザクション制御によってピアの自律性を損なわせてはならない．したがって，各ピアの RDBMS に共通した最低限の機能を用いた分散トランザクション制御である必要がある．

Dejima は 2 相ロックによる分散トランザクション制御を採用している．更新伝搬が発生する非集中型データ統合において 2 相ロックによる制御を行う場合，実際にトランザクションが実行され更新が伝搬するまで，他のピアのどのレコードが更新されるか特定できない．そのため，レコードのロック，更新，その更新の伝搬を繰り返すことで 2 相ロックによる分散トランザクション制御を行う．2 相ロックの実行において，各ピアのデータベースに要求される機能はレコードのロック機能のみであるため，非常に多くの種類の RDBMS がデータ統合システムに参加可能となる．そのため，ピアの自律性の観点において，2 相ロックをはじめとするロッキングプロトコルは Dejima における分散トランザクション制御として望ましい性質を持つ．

しかしながら，Dejima において 2 相ロックによる分散トランザクション制御を行う場合，分散デッドロックの増加が問題点として挙げられる．更新伝搬が発生する非集中型アーキテクチャでは，通常の分散データベースにおける分散トランザクション制御に比べ分散デッドロックが発生しやすい．通常の分散データベースシステムにおける分散デッドロックは，複数のトランザクションが同じレコードに対して同時にロック要求を行った結果，互い違いにロックを取得してしまい，互いのトランザクションの進行を妨げてしまう現象である．これに対し，更新伝搬の発生する非集中型データ統合においては，ネットワークの両端からトランザクションによる更新が伝搬した結果として分散デッドロックが発生したり，あるいは互いのロック要求が閉路を形成する場合など，通常の分散データベースシステムでは発生しない分散デッドロックが発生する．分散デッドロックが発生した場合，原因となっている一部のトランザクションをアボートすることによって分散デッドロックを解消する．しかしながら更新の連鎖的な伝搬を含む非集中型データ統合におけるトランザクションは，実行に多くの通信や処理を含んでおり，進行中のトランザクションのアボートはスループットの低下につながる．

また，同一のトランザクションがあるピアに対し複数の経路で伝搬する場合，データベースが不整合な状態に陥りうる．あるピアから伝搬したトランザクションが閉路を形成しそのピアに再び伝搬してくる場合や，あるピアから分岐したトランザクションが他のピアで合流する場合などがこれにあたる．複数の経路で更新が伝搬する場合，たとえば経路ごとに取得される Dejima テーブルへの更新内容が異なったり，あるいは閉路が形成され永遠に更新が伝搬され続ける場合がある．その結果，システム全体で大域的なデータの一貫性が保証できなくなったり，永遠に終了しないトランザクションが発生する．したがって，トランザクションの複数経路での更新伝搬が発生した際には，これを検知しアボートしなければならない．

3 提案手法

本章では非集中型データ統合アーキテクチャにおける分散トランザクション制御を提案する．提案手法は，Family Record Set という新たな概念を導入し，あるレコードの更新が伝搬しうる全てのレコードをトランザクション実行前に特定可能にする．この機構により，更新されうるレコードをトランザクション実行前にロックでき，保守的な 2 相ロックによる制御が実現できる．提案手法は，分散デッドロックが頻発しうる競合率の高いワークロードにおいて，保守的な 2 相ロックによって分散デッドロックによる実行中のトランザクションのアボートを防止し，スループットの低下を防ぐ．一方で，競合率の低いワークロードにおいては，保守的な 2 相ロックの実行にかかるオーバヘッドにより，スループットが低下することが予想される．そこで保守的な 2 相ロックの提案に加え，競合率の低いワークロードにおける保守的な 2 相ロックの制御によるオーバヘッドを削減するため，ワークロードに応じて通常の 2 相ロックによる制御に適応的に切り替えながら 2 手法を併用する適応的 2 相ロックを提案する．

3.1 Family Record Set

Family Record Set は，更新が伝搬し合うレコード群を表現する概念である．ある Family Record Set に含まれるレコードが更新された場合，この更新はこの Family Record Set に含まれる全てのレコードに伝搬される可能性がある．

非集中型アーキテクチャにおいて新しいレコードが挿入されると，この更新は連鎖的に他のピアへ伝搬され新しいレコードを生成する．そこで本研究では非集中型データ統合アーキテクチャにおけるレコードを 2 種類に分類した．ピアに直接挿入されたレコードを Original Record, 更新伝搬によって他のピアで生成されたレコードを Derived Record と定義する．ここで，Original Record とその挿入の更新伝搬によって他のピ

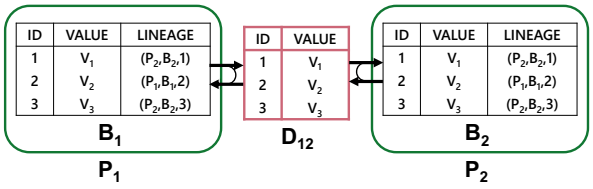

図 3　Family Record Set を導入した Dejima の例.

アで生成された Derived Record の集合を Family Record Set と定義する．すなわち，ある Original Record とこれに対応する Derived Record は共に同一の Family Record Set に属する．ある Family Record Set において，含まれる Original Record を Lineage と呼ぶ．Original Record はピア ID, ベーステーブル ID, レコード ID の組み合わせで表現される．Lineage は Family Record Set の識別子として使用される．新たな Original Record の挿入時には，更新伝搬と共にそのレコードの Lineage も通知し，システム内の全てのレコードに Lineage を付加する．

あるレコードが更新されるとき，その更新対象となるレコードの属する Family Record Set に含まれるレコード全てに更新が伝搬する可能性がある．一方で，その更新対象となるレコードの属する Family Record Set に含まれないレコードには更新が伝搬し得ない．すなわち，Family Record Set は更新伝搬における更新単位である．したがって，Family Record Set を導入することによって，あるレコードの更新に際してロックすべきレコードをシステム全体にわたって特定できる．

Family Record Set を導入した Dejima アーキテクチャの例を図 3 に示す．各レコードはベーステーブルのスキーマに加え Lineage カラムを持ち，このカラムに Lineage が記録される．Lineage は Family Record Set の識別子であるため，これを用いて各レコードがどの Family Record Set に属しているかを区別できる．たとえば，B_1 において ID が 1 であるレコードの Lineage は $(P_2, B_2, 1)$ となっており，同じ Lineage を持つ B_2 の ID が 1 であるレコードと同じ Family Record Set に属していることがわかる．また，この Family Record Set における Original Record は P_2 の B_2 に挿入された ID が 1 のレコードであることがわかる．このように，各レコードの属する Family Record Set の識別子として Lineage を利用し，トランザクション実行前にアクセスするレコードと同じ Lineage を持つレコードにロックを取得することで，同じ Family Record Set に属するレコード全てにロックを取得でき，保守的な 2 相ロックによる制御を実現できる．

また，双方向変換には JOIN クエリも含まれるため，一般に Lineage は Original Record の集合となる．JOIN クエリにより定義された Dejima テーブルを含む Dejima において Family Record Set を導入した例を図 4 に示す．P_1 と P_2 が共有する Dejima テーブル D_1 について，P_2 における定義に JOIN クエリが含まれている．このため，この JOIN クエリによって結合されたレコードを共有する P_1 のベーステーブル B_1 において，Lineage が複数の Original Record を用いて表され

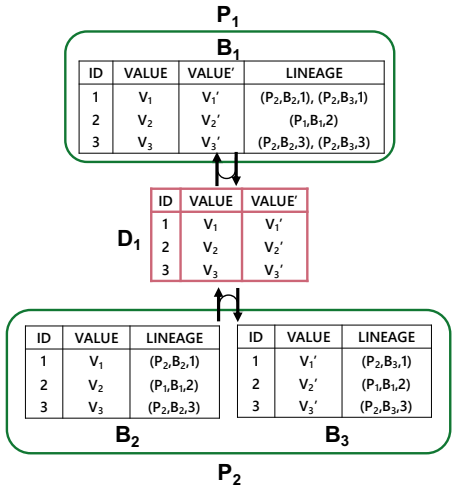

図 4　JOIN クエリにより定義された Dejima テーブルを含む Dejima において Family Record Set を導入した Dejima の例.

るレコードがある．このレコードを更新した際，この Lineage で表される Family Record Set のうち少なくとも一方に含まれるレコードにこの更新が伝搬する場合がある．すなわち，複数の Family Record Set がこの結合レコードによって結合したと考えることができる．このように Family Record Set は，JOIN クエリを含むような Dejima テーブルの定義がある場合でも，ロックすべきレコードを柔軟に特定できる概念である．

3.2　保守的な 2 相ロック

3.1 節で述べたように，Family Record Set を導入することによりトランザクション実行前に更新されうる全てのレコードを特定できるため，保守的な 2 相ロックによる制御を実現できる．本研究で実現する保守的な 2 相ロックのアルゴリズムは以下の通りである．

(1) 発行されたトランザクションの実行に必要な共有ロックをそのピアのデータベースにおいて取得する．
(2) 発行されたトランザクションの実行に必要な排斥ロックをそのピアのデータベースにおいて取得する．
(3) 排斥ロックを取得したレコードの Lineage を取得する．
(4) Dejima システム内の全ピアに対し，排斥ロックを取得したレコードの Lineage を通知し，各ピアにおいて同じ Lineage を持つレコードに 排斥ロックを取得する．
  (a) 全ての排斥ロック が取得できた場合トランザクションを実行し，木構造 2 相コミットによりトランザクションを終了する．
  (b) 全ての排斥ロックが取得できなかった場合，取得した排斥ロックを解放し，トランザクションをアボートする．

本手法は Family Record Set の導入によって保守的な 2 相ロックによる制御を実現する手法である．このため，保守的な 2 相ロックと同じ特性をもつ．本手法では進行中のトランザクションが分散デッドロックを引き起こさない．これにより，2 章で述べた分散デッドロックの問題を解消できる．しかしなが

ら，トランザクションが発行されたピアにおいてアクセスするレコードを事前に特定する必要がある．したがって，トランザクションの処理に条件分岐が含まれるような，実行時に動的にアクセスするレコードが決定するトランザクションにおいては，実行前にアクセスする全てのレコードを特定できないため実行できないという制約がある．また，本手法は大域的なコンポーネントに依存することなく分散トランザクション制御が可能であり，また各ピアのデータベースに要求する機能はレコードのロック機能のみであるため，非常に多くの種類のリレーショナルデータベースがデータ統合システムに参加可能となる．したがって，非集中型データ統合アーキテクチャにおけるピアの自律性を大きく損なわない手法である．

2 章で述べた通り，非集中型アーキテクチャでは同一のトランザクションがあるピアに対し複数の経路で伝搬することがあり，その場合データベースが不整合な状態に陥りうる．これを防ぐため本手法では，同一のトランザクションが複数の経路で同じピアに到達した場合，トランザクションをアボートする．

3.3 適応的 2 相ロック

3.2 節で述べた保守的な 2 相ロックの制御によって，分散デッドロックによる実行中のトランザクションのアボートが削減でき，スループットの向上が見込める．しかしながら，分散デッドロックが発生しにくい競合率の低いワークロードなどにおいては，トランザクション実行前の全ピアに対するロック要求にかかる処理や通信などがオーバヘッドとなり，スループットが低下する．そこで，本研究では保守的な 2 相ロックによる制御に加え，ワークロードに合わせて Dejima で元々採用されている通常の 2 相ロックによる制御に適応的に切り替えながら 2 手法を併用する適応的 2 相ロックを提案する．保守的な 2 相ロックは通常の 2 相ロックに従うプロトコルであり，両プロトコルを併用しても正しく直列化可能スケジュールを保証できる．

本手法では，各ピアが通常の 2 相ロックによる制御と保守的な 2 相ロックによる制御の 2 つの手法のうちどちらかの制御で動作する．あるピアにおいて発行されたトランザクションの実行に用いられる制御は，発行されたピアのその時点における手法に従う．本手法ではより性能の良い手法を選択するため，定期的に現在の手法と異なる手法での制御に切り替え，一定時間トランザクション処理を行いスループットを計測する．この時，現在の手法に従う制御におけるスループットよりも高いスループットが計測された場合，異なる手法による制御に切り替える．定期的にこのスループットの計測と手法の選択を実施することで，各ピアにおいてワークロードに適した手法を選択する．

4 評価実験

本章では提案手法を評価するために実施した実験について説明する．本実験では Amazon EC2 において東京リージョン上の m4.2xlarge インスタンスタイプのサーバを複数台利用し分散環境を再現した．各サーバは 8 コアの仮想 CPU と 32 GB のメモリを搭載している．各サーバは実験における 1 ピアに対応している．

4.1 実験設定

4.1.1 ネットワークトポロジ

本実験において，ピアによって構成されるネットワークトポロジと Dejima グループについて説明する．本実験では，ピアをノードとするランダムな全域木を生成し，これに従ってネットワークトポロジを決定する．この時，ピア間のエッジは Dejima グループに対応する．すなわち，ピア P_1 と P_2 の間にエッジがある場合，P_1 と P_2 は一つの Dejima グループを構成し，同じ Dejima テーブルを共有する．

本実験では，ピア間での更新伝搬の起こりやすさを制御するため，ベーステーブルのスキーマに 10 個の特別なカラムを追加する．これらのカラム名はそれぞれ，COND_1, COND_2, ..., COND_10 とする．これらのカラムの値は，各レコードごとに 0 から 100 の値からランダムに決定される．本実験では Dejima グループで共有する Dejima テーブルを，各ピアのベーステーブル B を用いて，`SELECT * FROM B WHERE COND_X < Y` と定義する．ここで，X は各 Dejima グループ毎に 1 から 10 の間の乱数によって決定される．また，本実験において Y の値を伝搬率と呼び，ピア間の更新伝搬の起こりやすさを制御するためのパラメータとして調整される．例えば Y の値が 0 のとき，ベーステーブルのあらゆるレコードはビュー定義に合致せず，すべてのレコードが隣接のピアに共有されない．Y の値が 100 のとき，ベーステーブルのあらゆるレコードはビュー定義に合致し，すべてのレコードが隣接のピアに共有される．

4.2 ベンチマーク

本節では評価実験に用いたベンチマークについて説明する．

4.2.1 YCSB

Yahoo! Cloud Serving Benchmark (YCSB) [6] はデータベースの性能評価によく利用されているベンチマークの一つである．実世界のアプリケーションを模倣したベンチマークではなく，システムの性能特性を分析するために用いられる．各ピアは一つのベーステーブルを保持し，このテーブルは ID カラムと COL1 から COL10 までの 10 カラムから構成される．また，4.1.1 節で述べたとおり，伝搬率を調整するための 10 カラムを追加する．YCSB を Dejima における分散トランザクション処理の性能評価に用いるため，5 つの読み込み命令と 5 つの書き込み命令を一つのトランザクションとして扱う．このとき，各命令の操作するレコードは次に示されるジップ分布に従って選択される．

$$f(k; s, N) = \frac{1/k^s}{\Sigma_{n=1}^{N} 1/n^s}$$

ここで k はレコード ID, N はレコード数, s は分布の偏りを調整するハイパーパラメータである．本論文において，このハイパーパラメータをスキューファクタと呼ぶ．スキューファクタが 0 のとき，この分布は一様分布と等価である．スキューファクタが 1 に近づくほど分布が偏り，同じレコード ID を選択しやすくなる．読み込み命令はベーステーブルのうち 1 つのレコードをジップ分布に従って参照する．書き込み命令はこの

テーブルのうち 1 つのレコードをジップ分布に従って選択し，COL1 から COL10 のうち一つのカラムをランダムに選択し，その値を変更する．また，各テーブルの初期レコードを生成する際，各ピアにおけるベーステーブルへのレコードの挿入は，Dejima のデータ共有によって様々なピアに伝搬する．本実験ではシステム全体で 10 万レコードが挿入されるように，各ピアで均等に初期レコードの生成が実行される．

4.2.2 TPC-C

TPC-C ベンチマーク [7] は実世界における卸売業務をモデルとしたベンチマークであり，OLTP システムの性能評価によく利用されている．本実験では TPC-C ベンチマークが規定する 9 つのテーブルのうち，customer テーブルをベーステーブルとし，このテーブルに含まれるレコードを他のピアと共有する．4.1.1 節で述べたとおり，customer テーブルのスキーマは，TPC-C ベンチマークの規定するカラムに加え，伝搬率を調整するための 10 カラムを追加する．また，本実験では 1 ピアが 1 つの district に対応する．したがって，実験において各テーブルの初期レコードを生成する際，customer テーブルに挿入するレコードやこれに対応して挿入される各テーブルのレコードは，対応する district に属するレコードのみとなる．この初期レコードの生成の際，customer テーブルにレコードが挿入されると，Dejima のデータ共有によって様々なピアにこのレコードが伝搬する．また，本実験では TPC-C ベンチマークが規定する 5 つのトランザクションのうち， New-Order トランザクションと Payment トランザクションの 2 種のトランザクションを 1 : 1 の割合で実行する．

4.3 実験結果

本節では得られた実験結果について説明する．各実験設定において，通常の 2 相ロック，保守的な 2 相ロック，適応的 2 相ロックの 3 手法について，それぞれ各ピア 1 スレッドで 1200 秒間ベンチマークを実行した．また，適応的 2 相ロックについては，スループット計測前に 600 秒実行することで各ピアにおいてワークロードに合わせた手法の切り替えを行い，続く 1200 秒で手法の切り替えを無効にしスループットの計測を行う．適応的 2 相ロックにおける定期的な各手法のスループット計測は，300 秒間に一度，各手法ごとに 30 秒間ずつ実施される．本実験では，得られたスループットについて通常の 2 相ロックで得られたスループットの値を 1 とした相対値を用いて評価する．全ての手法において，分散コミットプロトコルには木構造 2 相コミットを用いた．また，いずれの手法についても分散デッドロックを解消する方式として NO WAIT 方式を採用した．

4.3.1 YCSB

本節では YCSB ベンチマークによって得られた実験結果について説明する．

全ての Dejima テーブルにおける伝搬率を 100%, スキューファクタを 0.8 とし，伝搬率とスキューファクタを高く設定することで，競合の起こりやすいワークロードを再現した．この条件下でピア数を 20, 40, 60, 80, 100 と変更した際のスループットを計測した．結果を図 5 に示す．全てのピア数において，

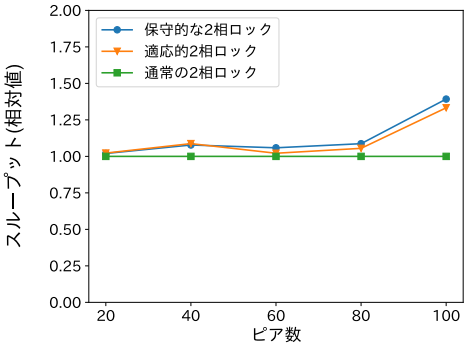

図 5　YCSB ベンチマークの各ピア数におけるスループット

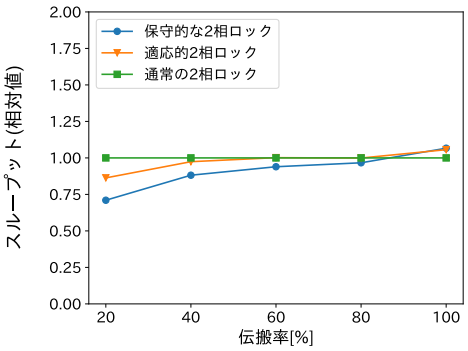

図 6　YCSB ベンチマークの各伝搬率におけるスループット

提案手法である保守的な 2 相ロックおよび適応的 2 相ロックが通常の 2 相ロックを上回る性能を達成している．これは，競合率の高いワークロードであるため分散デッドロックが頻発する実験設定であり，通常の 2 相ロックの性能が低下する一方，提案手法は分散デッドロックによる性能低下を抑制する手法であるためである．また，このワークロードにおいて，システムを構成するピア数が多いほど，提案手法である保守的な 2 相ロックと適応的 2 相ロックの性能が通常の 2 相ロックに比べて高くなった．これはピア数が多いほど 1 トランザクションに必要な更新伝搬の回数が増え，実行時間が増加するためである．通常の 2 相ロックでは分散デッドロックによる進行中のトランザクションのアボートが多発しスループットが低下するが，このトランザクションの実行時間が長いほど，アボートによるスループットの低下も大きくなる．このため，通常の 2 相ロックの性能低下が大きく低下し，保守的な 2 相ロックの性能が大きく上回る結果となった．また，適応的 2 相ロックについても，各ピアが適応的に保守的な 2 相ロックによる制御を選択することでスループットの低下を防ぎ，通常の 2 相ロックを上回る性能を達成している．

次に，ピア数を 60, スキューファクタを 0.8 とし，この条件下で伝搬率を 20%, 40%, 60%, 80%, 100% と変更することで，競合率の異なるワークロードを再現し，各伝搬率におけるスループットを計測した．結果を図 6 に示す．このワークロードにおいて，伝搬率が低い場合，提案手法が通常の 2 相ロックに比べて大きく劣っている．この原因は大きく 2 つあると考えられる．まず，伝播率が低い場合，レコードを他のピアと共有

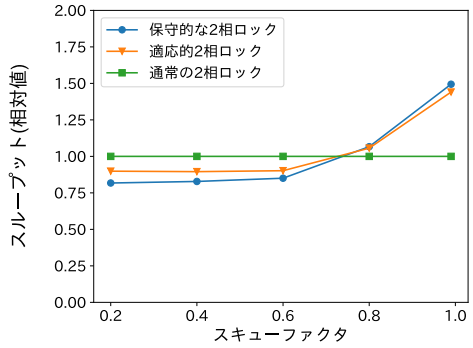

図 7　YCSB ベンチマークの各ハイパーパラメータにおけるスループット

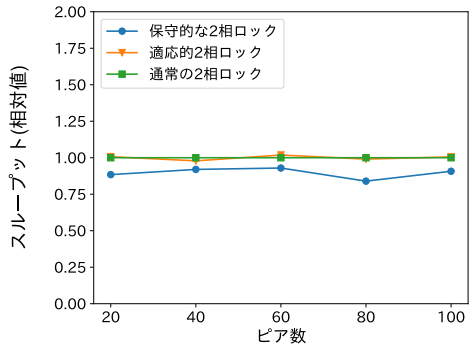

図 8　TPC-C ベンチマークの各ピア数におけるスループット

する可能性が低いため競合率も低く，分散デッドロックの発生回数が減少するため，結果として通常の 2 相ロックにおける分散デッドロックによる性能低下の影響が小さくなったためである．また，伝播率が低い場合は 1 トランザクションの実行時間が減少するため，アボート自体のコストが小さくなったことも原因として考えられる．

次に，ピア数を 60, 伝搬率を 100% とし，この条件下でスキューファクタを 0.2, 0.4, 0.6, 0.8, 0.99 と変更することで，競合率の異なるワークロードを再現し，各伝搬率におけるスループットを計測した．結果を図 7 に示す．スキューファクタが小さいほど競合率が低く，高いほど競合率が高くなるワークロードである．スキューファクタが 0.8 以上となる実験において，保守的な 2 相ロックと適応的 2 相ロックが通常の 2 相ロックの性能を上回った．一方でスキューファクタが 0.6 以下となる実験において，保守的な 2 相ロックと適応的 2 相ロックが通常の 2 相ロックの性能を下回った．ここから，提案手法は設計方針の通り競合率が高いワークロードにおいて効率的な分散トランザクション制御を達成することがわかる．

4.3.2　TPC-C

本節では YCSB ベンチマークによって得られた実験結果について説明する．

全ての Dejima グループにおける伝搬率を 80% とし，ピア数を 20, 40, 60, 80, 100 と変更した際のスループットを計測した．結果を図 8 に示す．TPC-C ベンチマークにおいては，YCSB ベンチマークにおけるスキューファクタのようなアクセスするレコードの偏りを直接的に調整するパラメータが存在しない．そのため，TPC-C ベンチマークによるワークロードは YCSB ベンチマークによるワークロードに比べ競合率が低く，分散デッドロックが発生しにくい．このワークロードにおいて，保守的な 2 相ロックの性能が通常の 2 相ロックを下回っている．これは通常の 2 相ロックにおける分散デッドロックによる性能低下に比べ，保守的な 2 相ロックにおけるロック要求のオーバヘッドによる性能低下の方が大きくなったためである．一方で適応的 2 相ロックは，ピアの多くが通常の 2 相ロックを選択することにより保守的な 2 相ロックのような性能低下を抑制している．

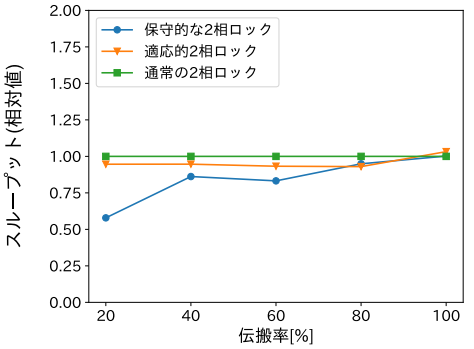

図 9　TPC-C ベンチマークの各伝搬率におけるスループット

次に，ピア数を 60 とし, Dejima グループにおける伝搬率を 20%, 40%, 60%, 80%, 100% と変更した際のスループットを計測した．結果を図 9 に示す．このワークロードにおいても，ピア数を変化させた際の実験結果と同様に保守的な 2 相ロックの性能が通常の 2 相ロックを下回っており，さらに伝搬率が低い場合において保守的な 2 相ロックの性能がより低下している．伝搬率が低い場合，1 トランザクションに含まれる更新伝搬が少なくなり実行時間が短くなるだけでなく，競合率も低くなる．この結果，通常の 2 相ロックにおける分散デッドロックによる性能低下の影響が小さくなったため，保守的な 2 相ロックとの性能差がより大きくなったと考えられる．一方でピア数を変化させた際の実験結果と同じく，適応的 2 相ロックでは適切な手法の選択の結果，保守的な 2 相ロックほどの性能低下を抑制している．

これらの実験結果から，競合率の高いワークロードにおいては保守的な 2 相ロックが効率的に分散トランザクション制御を行っていることがわかる．また，競合率の低いワークロードや，1 トランザクションの実行にかかる時間が短く分散デッドロックによる性能低下が大きくならない場合，保守的な 2 相ロックの実行にかかるオーバヘッドによる性能低下の方が大きくなる．また，そのような場合においても適応的 2 相ロックでは適切な手法の選択により，性能低下を抑制できることがわかる．

5　関 連 研 究

5.1　非集中型データ統合

従来のデータ統合アーキテクチャの多くが中央集権型として

設計されている．しかしながら近年，各ピアが自立分散的にデータを共有する非集中型アーキテクチャが提案されている．非集中型データ統合アーキテクチャを最初に提案したプロジェクトとして Piazza [8] がある．本システムはグローバルスキーマを用いず，複数のデータベースに分散した XML データをピア間で相互にやりとりする．

Piazza の後継プロジェクトとして ORCHESTRA [9] がある．ORCHESTRA はバイオインフォマティクスの分野における膨大な量のデータの効率的な共有を目的としている．各ピア内のデータの一貫性は保証されるが，大域的なデータの一貫性は保証されない．

本研究で扱った Dejima [3], [4] は ORCHESTRA に強く影響を受け提案されたアーキテクチャであり，双方向変換技術を用いた柔軟なデータ共有を実現する．従来の非集中型データ統合アーキテクチャと異なり，より一般的な応用のため大域的なデータの一貫性の保証を重視する．

5.2 トランザクション制御

分散トランザクションの同時実行制御としてさまざまな手法が提案されている [10]．Spanner [11] は多版 2 相ロックを分散環境に拡張する手法である．True Time API を利用し正確なタイムスタンプを取得することで分散環境における多版化を実現している．VoltDB [12] はテーブルをパーティショニングし，1 パーティションにつき 1 実行スレッドを割り当て，トランザクションを実際に直列に実行する．トランザクションのアクセスするレコードが 1 パーティション内に収まる場合，排他制御が必要なく，高いスループットを達成できる．

決定論的アプローチ [13] に基づく手法も提案されている．各データベースにおいて決定論的な動作を強制することで，トランザクションの入力順のみの複製によって実行後の状態が全てのレプリカサーバで等しくなることを保証できる手法である．Calvin [14] は特殊なロッキングプロトコルによりデータベースの決定論的な動作を実現し，入力のみを複製する形で多くの通信を削減する．Aria [15] はトランザクションを並行実行した後にコミットの可否を確認する形で処理を行うことで，データベースの決定論的な動作を実現しつつ排他処理を削減する．

6 おわりに

本研究では非集中型データ統合アーキテクチャ Dejima を分散トランザクション制御の観点から分析することで，非集中型データ統合アーキテクチャにおける効率的な分散トランザクション制御手法を 2 つ提案した．Family Record Set を導入することで実現した保守的な 2 相ロックと，ワークロードに応じて通常の 2 相ロックと保守的な 2 相ロックを適応的に切り替え併用する適応的 2 相ロックである．評価実験により，競合率の高いワークロードにおいて保守的な 2 相ロックが通常の 2 相ロックの性能を上回り，また競合率の低いワークロードにおいても適応的 2 相ロックによって性能の低下を防げることがわかった．

今後の課題として，非集中型データ統合アーキテクチャの実世界における応用をモデルにしたベンチマークの設計と，そのベンチマークによる分散トランザクション処理性能の評価が挙げられる．本研究では伝播率を調整するカラムを追加して直接伝搬率を操作したが，実世界の応用では実用的なビュー定義によってより複雑な伝搬条件となる．また，各ピアのデータベースサーバが地理的に離れており，通信に大きな遅延が発生する場合もある．したがって，こういった要素を包括的にモデリングしたベンチマークを設計する必要がある．

また，本研究では Dejima テーブルが SPJU クエリによって定義される場合のみを扱った．しかしながら，応用によっては集約処理など，SPJU クエリに含まれないクエリを含む Dejima テーブル定義を用いる場合もある．こういった処理を含む場合においてもデータの大域的な一貫性を保証する j 分散トランザクション制御手法の提案も今後の課題として挙げられる．

文　献

[1] Maurizio Lenzerini. Data integration: A theoretical perspective. In *ACM SIGMOD PODS*, pages 233–246, 2002.

[2] Grigoris Karvounarakis et al. Collaborative data sharing via update exchange and provenance. *ACM TODS*, 38(3):19, 2013.

[3] Yasuhito Asano et al. Flexible framework for data integration and update propagation: system aspect. In *IEEE BigComp*, pages 1–5, 2019.

[4] Yasuhito Asano et al. Making view update strategies programmable-toward controlling and sharing distributed data. *arXiv preprint arXiv:1809.10357*, 2018.

[5] Van-Dang Tran, Hiroyuki Kato, and Zhenjiang Hu. Programmable view update strategies on relations. *VLDB Endow.*, 13(5):726–739, 2020.

[6] Brian F Cooper et al. Benchmarking cloud serving systems with ycsb. In *ACM symposium on Cloud computing*.

[7] TPC-C Overview. `http://www.tpc.org/tpcc/detail5.asp`.

[8] Alon Y Halevy et al. Piazza: data management infrastructure for semantic web applications. In *WWW*, pages 556–567, 2003.

[9] Zachary G Ives et al. Orchestra: Rapid, collaborative sharing of dynamic data. In *Innovative Data Systems Research*, pages 107–118, 2005.

[10] Rachael Harding et al. An evaluation of distributed concurrency control. *VLDB Endow.*, 10(5):553–564, jan 2017.

[11] James C Corbett et al. Spanner: Google ' s globally distributed database. *ACM TOCS*, 31(3):1–22, 2013.

[12] Michael Stonebraker and Ariel Weisberg. The voltdb main memory dbms. *IEEE Data Engineering Bulletin*, 36(2):21–27, 2013.

[13] Daniel J Abadi and Jose M Faleiro. An overview of deterministic database systems. *Communications of the ACM*, 61(9):78–88, 2018.

[14] Alexander Thomson et al. Calvin: fast distributed transactions for partitioned database systems. In *SIGMOD, pages=1–12, year=2012*.

[15] Yi Lu et al. Aria: A fast and practical deterministic oltp database. *VLDB Endow.*, 13(11), 2020.