P2P 型データ統合アーキテクチャにおけるチケットベース手法を用いた分散トランザクション制御

三宅 康太† 涌田 悠佑† 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,yusuke.wakuta,sasaki,chuanx,onizuka}@ist.osaka-u.ac.jp

あらまし 近年，生成されるデータ量の爆発的な増大を受けてデータ統合はますます重要な技術となっており，様々なアーキテクチャが提案されている．複数のデータベースが更新情報を相互に伝搬する P2P 型のアーキテクチャにおいて，分散するデータの一貫性を保証するために分散トランザクションの同時実行制御が必要となる．分散トランザクションの同時実行制御手法の多くは，グローバルトランザクションマネージャのような大域的な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，各データベースサーバ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．そこで本論文では，大域的なコンポーネントを必要としない手法として知られるチケットベース手法と，各データベースサーバの自律性を高める木構造 2 相コミットプロトコルを組み合わせた分散トランザクション制御を提案する．また，その性能を評価する．

キーワード データ統合，分散トランザクション，同時実行制御

1 はじめに

データ統合とは，複数の異なるデータベースに分散した関連するデータを統合し，利活用の簡易性を向上させる技術である．例えば企業内において，各部門が各自の性能や利便性を意識してデータベースのスキーマ設計をした結果，部門間で共有すべきデータのスキーマが合致せず連携が十分に取れない状況がそのユースケースとして挙げられる．また，旅行代理店などのように，複数のサービスからの情報を統合し顧客に対し一括に情報を提供したい場合なども典型的な事例となる．IT 産業の発達に加え IoT の急速な普及を受けて，発生するデータ量はますます増大している．このような中で，個別のサービスに蓄積されたデータを活用するだけでなく，他の様々なサービスに蓄積されたデータも一括して活用したいという需要が高まっている．そのため，データ統合はより一層重要な技術になっている．

従来のデータ統合技術は，単一のグローバルスキーマを用いて，分散するデータベースに蓄積された実データを統合したビューをユーザに提供することでデータ統合を実現している [1]．このグローバルスキーマに基づくアーキテクチャはその特性から，全てのデータベースサーバからの通信を受け付ける大域的なコンポーネントを必要とする傾向にある．このような従来のデータ統合の考え方に対して，P2P 型のデータ統合である Collaborative Data Sharing Systems (CDSS) と呼ばれるコンセプトが提案されている．このコンセプトは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からの強い要請を受けたものである．Piazza [2] やその後継プロジェクトの ORCHESTRA [3] など，P2P 型のデータ統合アーキテクチャがこれにあたる．CDSS は非常に柔軟な仕組みであり，従来の単一のグローバルスキーマを利用したデータ統合とは異なり，自立した対等なピア間でデータをやりとりすることでデータ統合を実現する．理想的な P2P 型アーキテクチャは大域的なコンポーネントに依存せず全てのピアが対等であるため，システムの一部に障害が発生してもシステム全体が運用不可能になるような障害にならない．よって各データベースサーバのより自律的な運用が可能になる．一方で，提案されている P2P 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，大域的なデータの一貫性は保証していない．これは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，大域的な一貫性を保証する必要がないためである．しかし，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，大域的な一貫性を保証しないことは大きな問題となりうる．

既存技術の問題解決のために，Dejima アーキテクチャ [4] が提案されている．このアーキテクチャは ORCHESTRA と比べて更に一般的なユースケースを想定しており，P2P 環境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．Dejima アーキテクチャにおいて分散トランザクション制御は不可欠である．また，P2P 型アーキテクチャには先述の通り大域的なコンポーネントに依存せず各ピアの自律性を高められるという利点がある．よって Dejima アーキテクチャに適用する分散トランザクション制御においても大域的なコンポーネントに依存しないような手法が望ましい．しかしながら提案されている分散トランザクション制御の多くはグローバルトランザクションマネージャ (GTM) と呼ばれる大域コンポーネントに依存している．

分散トランザクション制御における同時実行制御手法の 1 つとしてチケットベース手法が知られている．この手法は分散す

る各データベースにチケットと呼ばれるデータアイテムを用意し，分散トランザクションにチケット取得操作を組み込むことで，大域的な直列化を保証する手法である．この手法も GTM を利用した手法であるが，分散する各データベースが特定の条件を満たす場合，チケットベース手法は GTM を全く必要としない手法となる．詳しくは 2 章で述べる．そのためチケットベース手法は P2P 型アーキテクチャにおける制御手法に適している．しかしチケットベース手法はデータベース分野において Snapshot Isolation (SI) と呼ばれる同時実行制御手法が提案される以前に提案された手法であり，各データベースのローカルスケジューラが Strict 2-phase Locking (S2PL) のようなロッキングプロトコルを採用していることを前提としている．近年，SI をさらに改善した Serializable Snapshot Isolation (SSI) による SERIALIZABLE 隔離レベルの実現が標準的になっており，この手法を適用した際の直列化の振る舞いは，チケットベース手法が提案された当時のものとは大きく異なっている．

したがって本論文では，分散する各データベースが SSI によって動作することを前提とした場合においても，チケットベース手法が大域的な一貫性を保証する同時実行制御手法となることを説明し，ロッキングプロトコルを前提とする従来のチケットベース手法との差異について明確にする．さらに，SSI に拡張したチケットベース手法においてチケットの粒度を変更可能にした手法を提案し，スループットの改善を行う．また，P2P 型アーキテクチャの自律性を高めるため，分散するトランザクションのコミット・アボートの結果を一致させる手法として木構造 2 相コミットを採用する．

これらの提案の有効性を検証するため，評価実験を行う．4 つのピアからなる Dejima アーキテクチャを仮想環境によって再現し，このアーキテクチャで実行される複数のトランザクションの更新伝搬範囲重複率や競合率を調整したワークロードで実験を行う．この結果から更新伝搬重複率の高いワークロードにおいてチケット粒度を細かくすることでより効率的な分散トランザクション制御が可能になることを示す．

本論文の構成は以下の通りである．2 章にて事前知識を説明し，3 章では本研究で提案する分散トランザクション制御手法の概要を示す．4 章にて評価実験について説明し，5 章で関連研究について説明する．6 章にて本論文をまとめ，今後の課題を論ずる．

2 事前知識

本章においては，本論文において必要な事前知識について説明する．まず 2.1 節において，Dejima アーキテクチャの概要を説明する．続いて 2.2 節においてに分散トランザクション制御を構成する 2 種類の制御について説明し，2.3 節において S2PL 及び SSI の概要を説明する．2.4 節において従来のチケットベース手法の概要を説明し，最後に 2.5 節において，分散トランザクション制御に必要な分散合意プロトコルについて説明する．

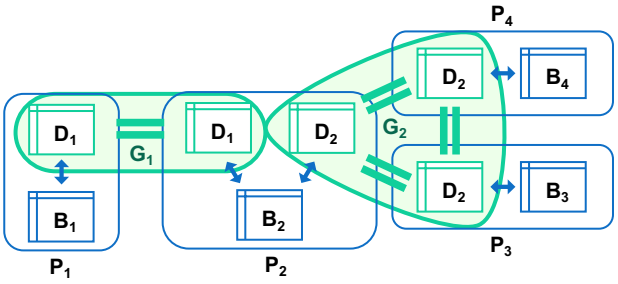

図 1 Dejima アーキテクチャの例．各ピアがベーステーブルと Dejima テーブルを保持しており，データを共有したいピアと共に Dejima グループを形成する．

2.1 Dejima アーキテクチャ

Dejima アーキテクチャとは，双方向変換を利用した更新伝搬を用いて柔軟なデータ統合を可能とするアーキテクチャである [4]．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する．

2.1.1 概　　要

Dejima アーキテクチャは以下の 4 つの構成要素から成る．

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

ピアはそれぞれ固有のデータベースを持っており，データベース内にベーステーブルと，ベーステーブルから導出されるビューである Dejima テーブル保持している．一般的にビューの多くは読み取り専用であるが，この Dejima テーブルは 双方向変換技術を利用することで更新可能なビューとなっている [5]．つまり，このビューに対して直接更新を実行でき，ビューの更新によって導出元のベーステーブルのデータも適切に更新される．

各ピアはデータを共有するピア同士で Dejima グループを形成する．同じ Dejima グループに属するピアは，そのグループに対応した同一の Dejima テーブルを同期して保持している．Dejima テーブルは 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．

Dejima アーキテクチャによってどのように更新伝搬がなされていくかを説明する．このアーキテクチャの一例を図 1 に示す．このアーキテクチャによる更新伝搬は次のような過程で行われる．

（1） あるピアにおいてユーザによりベーステーブルが更新される．

（2） (1) の更新によって，ベーステーブルのビューである Dejima テーブルが更新される．

（3） Dejima テーブルに行われた更新内容を，同じ Dejima テーブルを保持するピアに通知する．

（4） 同じ Dejima テーブルを保持するピアにおいて，(3) で通知された更新内容をもとに Dejima テーブルを同期する．

（5） Dejima テーブルが同期された各ピアにおいて，双方向変換によりベーステーブルが更新される．

（6） 以降，Dejima テーブルの更新が発生しなくなるまで

2〜5 を繰り返す．

2.1.2 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．ORCHESTRA 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を保証することも大きな目的の一つとなっている．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 P2P 型であり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．そのため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 分散トランザクション制御

分散トランザクション制御における重要な用語と概念 [6] について説明する．まずはじめに分散トランザクション制御におけるトランザクションの分類について説明する．2 つ以上のデータベースにアクセスするようなトランザクションのことをグローバルトランザクションと呼ぶ．一方 1 つのデータベースにのみアクセスするようなトランザクションのことはローカルトランザクションと呼ぶ．

GTM はグローバルトランザクションをユーザから受け付け，適切な実行を管理するコンポーネントである．グローバルトランザクションは GTM を介して実行されるのに対し，ローカルトランザクションは GTM を介さず，ユーザから直接データベースに入力される．また，分散トランザクション制御手法の多くは GTM に依存している．

分散トランザクション制御は分散合意プロトコルと同時実行制御の 2 つを組み合わせることによって達成される．分散合意プロトコルは，分散トランザクションがアクセスした各データベースにおいて，コミット・アボートの結果を一致させるために用いられる．一方同時実行制御はグローバルトランザクション，ローカルトランザクション問わず全てのトランザクションの実行結果がデータベースに不整合な状態を生み出さないよう制御する．

分散トランザクション制御の同時実行制御を考えるにあたり，同時実行制御がなされない場合に分散システムで発生する 2 つの問題について整理する．1 つ目の問題は，データベース間で矛盾した直列化が引き起こされることである．トランザクション t_1，t_2 とデータベースサーバ S_1，S_2 があるとき，S_1 においては t_1t_2，S_2 においては t_2t_1 の順に直列化される場合などがこれにあたる．2 つ目の問題は，ローカルトランザクションによってグローバルトランザクション間に依存関係が引き起こされることである．例えば，グローバルトランザクション t_1，t_2 があり，あるデータベースサーバ S においてそれぞれデータ x，y にアクセスする場合，このトランザクションは共通のデータにアクセスしていないため依存関係がない．しかし，このデータベースサーバ S にのみアクセスするローカルトランザクション t_3 がデータ x，y 両方にアクセスする場合，元々依存関係になかった t_1 と t_2 の間に t_3 を媒介とした間接的な依存関係が発生する可能性がある．GTM はグローバルトランザクションのみを受け付け，ローカルトランザクションの存在を知り得ないため，このような間接的な依存関係も同様に検知できず，これが矛盾した直列化を引き起こす可能性がある．

2.3 S2PL と SSI

データベースにおける SERIALIZABLE 隔離レベルを実現する手法として S2PL と呼ばれるロッキングプロトコルが提案されている．S2PL は以下のようなプロトコルである．トランザクション実行時，アクセスするデータに対しロックを取得していくが，ロックの取得されたデータへの処理を終えたとしても，トランザクション実行中はこのロックを解放しない．コミットかアボートによってトランザクションが完了したのち，取得したロックを一斉に解放する．S2PL が生み出すスケジュールは直列化可能であり，SERIALIZABLE 隔離レベルの実現方法の 1 つとして広く知られている．

しかしながら近年，SERIALIZABLE 隔離レベルを実現する手法として SSI と呼ばれるプロトコルを採用するデータベースが多くなっている．SSI は SI を改善した手法である．SI は以下のような手法である．トランザクション実行開始時，データベースがこのトランザクションに対しデータベースのスナップショットを発行し，トランザクションはこのスナップショットに対し処理を実行する．一連の処理が終わったのちコミット要求が行われ，スナップショットがデータベース本体に統合される．なお，同じデータに書き込みを行うトランザクションが複数存在し，それらが同時実行されている場合は，これらのトランザクションのうち 1 つのみを成功させ，他全てをアボートさせる．これによって，あるトランザクションの書き込みが同時実行されている他のトランザクションによって上書きされてしまい更新が遺失してしまうことを防ぐ．この制御の実現方法として，First-Committer-Win ルールを採用する方法がある．First-Committer-Win ルールとは，同じデータに書き込みを行う複数のトランザクションのうち，最速でコミット要求を行ったトランザクションを成功させ，他のトランザクションをアボートさせるというルールである．この制御の他の実現方法としては，データへの書き込み時にそのデータに対し排他ロックを取得するという方法がある．しかしながら SI による制御では，直列化可能でないようなスケジュールが生成されてしま

うことが指摘されている [7].

そこで SI を改善した手法として SSI が提案されている [8]. SSI はトランザクション間の依存関係を表現する依存関係グラフを生成する．SI において直列化可能でないようなスケジュールが生み出される場合，トランザクション間の依存関係を表現する依存関係グラフに閉路が形成される．この時，形成された閉路に Dangerous Structure と呼ばれる特殊な構造が現れる．SSI はこの Dangerous Structure の発生を実行時に検知し，この構造を形成している原因となっているトランザクションをアボートすることでこの構造を解消する．

2.4 従来のチケットベース手法

従来のチケットベース手法 [9] について説明する．この手法は，各データベースにチケットと呼ばれる整数型データを用意しておき，チケット取得操作と呼ばれる一連の操作をグローバルトランザクションに組み込むことで分散トランザクション制御を行うものである．チケット取得操作とは，このチケットの値をインクリメントする操作であり，チケットに対する読み取りと書き込みが必要となる．よって，本来依存関係の発生しないトランザクションに対しても，チケットを介した依存関係を必ず発生させる．

この手法では，GTM が各グローバルトランザクションの読み取ったチケットの値を元に大域的な依存関係グラフを作成することで大域的な直列化の制御を行う．依存関係グラフとはトランザクションの依存関係を表現するグラフであり，このグラフに閉路が形成されている場合，一貫性のないトランザクションの実行を許している可能性がある．そのため閉路が形成されている場合，GTM が必要な分だけトランザクションをアボートし閉路を解消する．よって 2.2 節で説明した 1 つ目の問題である矛盾した直列化を GTM によって防ぐことができる．さらに，本来依存関係の無い分散トランザクション間にもチケットを介した直接的な依存関係を発生させるため，2 つ目の問題である間接的な依存関係に対する制御を考える必要がなくなる．しかしこの手法は本来直接的な依存関係も間接的な依存関係も無いようなグローバルトランザクション間にすらチケットを介した依存関係を発生させる．そのためこの手法は不必要なアボートを引き起こす場合がある．

Georgakopoulos らの定理 [10] によると，各データベースが特定の条件を満たしている場合，グローバルトランザクションにチケット取得操作を組み込むだけで，GTM による閉路の検出なしに大域的な直列化が可能である．この場合 GTM はグローバルトランザクションを受け付け，チケット取得操作を組み込み各データベースにおいて実行するだけで良く，こういった前処理は GTM を介すことなく行うことができるため，GTM が不必要となる．例えばトランザクションが入力されたデータベースにおいて，データベースがそのトランザクションにチケット取得操作を組み込むだけで制御が可能になる．GTM を利用しない場合，複数のデータベースを介したデッドロックはタイムアウトによって解決する．

この手法が提案された当時の S2PL により制御されるデータベースはその多くが Georgakopoulos らによる定理の条件を満たしている．このため S2PL を採用するデータベースで構成された分散システムにおいては，チケットベース手法は非常に実用的な同時実行制御手法である．しかし近年のデータベースにおいて，SERIALIZABLE 隔離レベルは SSI による実現が主流となっている．多版同時実行制御はロッキングプロトコルによる制御と大きく異なる．そのため GTM を利用しないチケットベース手法を SSI に適用するといった提案はなされていない．

2.5 分散合意プトロコル

2.2 節において説明したように，分散する全てのデータベースでコミット・アボートの結果を同一に保つため，分散合意プロトコルが必要となる．分散合意プロトコルの 1 つに木構造 2 相コミットプロトコルがある [11]．調停者ノードから Prepare 命令が送信された時，コミット可能であれば OK として，コミット不可能であれば NG として調停者ノードに対し応答するという点に関しては通常の 2 相コミットと同様である．しかし木構造 2 相コミットは通常の 2 相コミットとは異なり木構造をしており，あるノードはその子ノードの調停者ノードとなる．ルートノード以外の全てのノードは親ノードから Prepare 命令が送られてきた際，子ノードに対して Prepare 命令を送る．全ての子ノードから OK の応答があり，そのノード自身もコミット可能であれば，親ノードに対し OK と応答する．ルートノードが全ての子ノードからコミット可能であるという応答を受け取った場合，自身のデータベースにおいてトランザクションをコミットし，子ノードに対してコミット命令を送る．親ノードからコミット命令を受け取ったノードは，同様に自身のデータベースにおいてコミットを行い，子ノードに対しコミット命令を送る．

3 提 案 手 法

本章の構成は次の通りである．まず 3.1 節において，分散する各データベースが SSI を採用する場合においても，チケットベース手法を適用することによって大域的なデータの一貫性を保証できることを説明する．続く 3.2 節では，さらにこのチケットの粒度を変更可能にした，より効率の良いチケットベース手法を提案する．また本研究で提案する分散トランザクション制御手法において，分散合意プロトコルとして木構造 2 相コミットを採用する．よって最後の 3.3 節で，木構造 2 相コミットを採用する理由について説明する．

3.1 SSI を前提としたチケットベース手法

本節では，分散する各データベースが SSI を採用する場合においても，チケットベース手法を適用することによって大域的なデータの一貫性を保証できることを説明する．従来と同様に各分散トランザクションに必要なチケット取得操作を組み込んだ場合を考える．複数のトランザクションが同じデータベースサーバにアクセスする際，チケット取得操作により同じデータベースサーバに用意されているチケットへの書き込みが強制される．よって同じデータに対して全てのトランザクションが同

時に書き込みを行おうとしているため，SI の制御によってこのうち 1 つのトランザクションのみコミットが許される．コミットを行うことができなかったトランザクションは，2 相コミットプロトコルなどの分散合意プロトコルによって，トランザクション処理を行った全てのデータベースサーバにおいてアボートが実行される．つまり，各データベースにおいてコミットされるトランザクションは同時実行されない．

2 章で述べた通り，SI において依存関係グラフに閉路が形成される場合，Dangerous Structure と呼ばれる特殊な構造が現れる．Dangerous Structure はその定義 [8] から同時実行されているトランザクションの間に依存関係が無い限り発生し得ない．チケット取得操作を組み込んだ場合，SSI においてはデータベースで同時実行されるトランザクションは 1 つしかコミットされることはないため，Dangerous Structure は発生しえない．よって依存関係グラフに閉路は発生せず，大域的なデータの一貫性を保持できる．

以上のことから，GTM を用いないチケットベース手法は，SSI による SERIALIZABLE 隔離レベルを前提とした場合においても分散トランザクション制御の手法となり得ることがわかる．なお SSI に適用する場合，複数のトランザクションのチケットに対する書き込みによって，これらのトランザクションのうち 1 つのみがコミットできるという特性が Dangerous Structure の発生を防いでいる．そのため，チケットのインクリメント操作は必要ではなく，チケットに何らかの値を書き込むだけでこの手法は機能する．よって，従来のチケットベース手法と違いチケットの読み込み命令を省略できる．

しかしながら SSI を前提としたチケットベース手法を Dejima アーキテクチャに適用する場合，大きな課題が存在する．SSI において同じデータに書き込みを行うトランザクションが同時実行される場合，コミットできるトランザクションは 1 つに制限されてしまう．チケットベース手法はチケットがデータベース単位で用意されており，あらゆるグローバルトランザクションがこのチケットに対して読み込みと書き込みを行う．よってあるデータベースにおいて同時実行できるトランザクションが 1 つに制限されてしまう．これは 1 章で述べたように，より一般的なユースケースを想定している Dejima アーキテクチャにおいて，スループットを大幅に下げてしまう要因となる．

3.2 チケット粒度を変更可能なチケットベース手法

3.1 節の最後に述べたように，従来のチケットベース手法を SSI を前提とした分散データベースに適用することは望ましくない．そこで本研究ではさらにこのチケットの粒度を変更可能にした手法を提案する．

従来のチケットベース手法では，ローカルトランザクションを介したグローバルトランザクション間の間接的な依存関係も考慮に入れるため，データベース単位でチケットを用意している．しかし，チケットがデータベース単位で用意されている限り，3.1 節の最後に述べた制約が発生する．そこで，ローカルトランザクションにもチケット取得操作を組み込みチケット粒度を変更可能とする手法を提案する．GTM を利用しないチケットベース手法において，トランザクションにチケット取得操作を組み込むのは GTM ではなく各データベースなどであるため，ローカルトランザクションにもチケット取得操作を組み込むことができる．これによりあらゆるトランザクションにチケット取得操作が組み込まれ依存関係が把握できる．このため，チケットの粒度をデータベース単位に限定する必要がなくなり，チケットの粒度が変更可能となる．例えばレコード単位でチケットを用意する場合，あるトランザクションがあるレコードにアクセスするのであれば，対応するレコードのチケット取得操作を行うということである．

また，この手法が同時実行制御になり得ることをより形式的に説明できる．3.1 節で述べた通り，Dangerous Structure は同時実行されているトランザクションの間に依存関係がない限り発生し得ない．依存関係は同じデータにアクセスするトランザクション間にのみ発生するが，このようなトランザクションは同じチケットにもアクセスしている．このトランザクションが同時実行されている場合 SSI の制御によってこのうち 1 つのみがコミットでき，他のトランザクションは全てアボートされる．よって同じデータにアクセスし同時実行されるトランザクションのうちコミットに成功するトランザクションは 1 つしかなく，Dangerous Structure を構成する依存関係は発生し得ない．よってこの手法は同時実行制御を達成する．

チケットがデータベース単位で用意されている場合，あるデータベースで同時実行されうるトランザクションは 1 つに制限される．一方で例えばチケットがレコード単位で用意されている場合，同じレコードにアクセスするような同時実行される複数のトランザクションについては SI の制御によりそのうち 1 つのみが成功となるが，データベースで同時実行できるトランザクションの数に制限はない．よって P2P 型データ統合アーキテクチャにおいて複数のトランザクションの更新伝搬範囲が重複していた場合でも，同じデータにアクセスしていない限り，全てのトランザクションをコミットすることができ，データベース単位のチケットベース手法よりも効率の良い同時実行制御手法となる．なお，チケットの粒度はデータベース単位やレコード単位だけでなく，テーブル単位やレコードの集合単位などに設定することもできる．

3.3 木構造 2 相コミット

分散する全てのデータベースでコミット・アボートの結果を同一に保つため，分散合意プロトコルが必要となる．提案する手法において，木構造 2 相コミットを採用する．

通常の 2 相コミットではデータ統合に参加する全てのピアのアドレスを事前に知っておく必要がある．一方で木構造 2 相コミットでは自分の子ノードのアドレスを知っておくだけで実行することができる．つまりデータ統合に参加するピアのうち自分が属している Dejima グループに属していないようなピアのアドレスを知っておく必要はない．このため木構造 2 相コミットは各ピアの自律性を重視する Dejima アーキテクチャに適している．

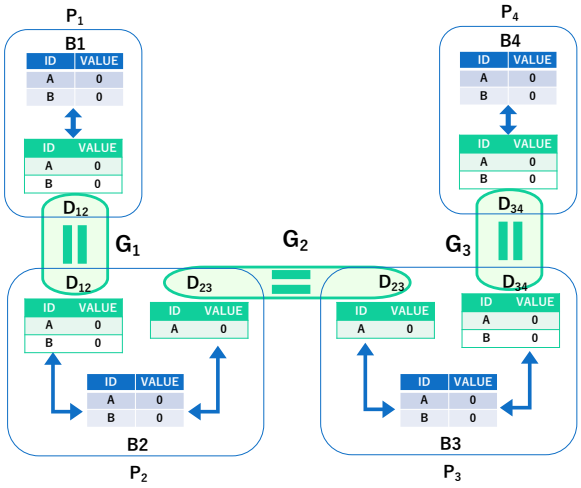

図 2　実 験 設 定

表 1　各ピアにおける Dejima テーブル定義

ピア	Dejima テーブル	定義
P_1	D_{12}	SELECT * FROM B1;
P_2	D_{12}	SELECT * FROM B2;
	D_{23}	SELECT * FROM B2 WHERE ID='A';
P_3	D_{23}	SELECT * FROM B3 WHERE ID='A';
	D_{34}	SELECT * FROM B3;
P_4	D_{34}	SELECT * FROM B4;

4　実　　験

本章ではチケットベース手法の有用性を評価するために実施した実験について説明する．

4.1　実 験 環 境

本実験には 64GB のメモリを搭載し CPU に Intel Xeon CPU E7-8880 v3 2.30GHz を 2 つ搭載した計算機を使用し，Ubuntu 18.04 上でデータベースの分散環境を再現した．Dejima アーキテクチャの各ピアで動作するデータベースは SSI による同時実行制御を実装している PostgreSQL を採用した．また，同時実行されているトランザクションによる同じデータへの書き込みに対する制御は，First-Committer-Win ルールではなく，書き込み対象となるデータへのロックによって実現している [12]．

4.2　実 験 設 定

図 2 に実験設定を示す．P_1，P_2，P_3，P_4 の 4 つのピアを用意し，それぞれのベーステーブルを B1，B2，B3, B4 とする．P_1 と P_2，P_2 と P_3, P_3 と P_4 はそれぞれ Dejima グループ G_1，G_2，G_3 を形成し Dejima テーブル D_{12}, D_{23}, D_{34} を共有する．各ベーステーブルは ID，VALUE の 2 つのカラムからなるスキーマで構成されており，ID は文字列型, VALUE は整数型のデータである．初期状態は図 2 に示すようになっている．なお，各ピアにおける Dejima テーブルの定義は表 1 の通りになっている．この構成では，ID が A であるようなレコードへの更新は全てのピアに伝搬するが，ID が A でないようなレコードへの更新は P_2 と P_3 の間では伝搬しないことが特徴である．このような構成のデータ統合システムにおいて，P_1及び P_4 に対し VALUE カラムのデータをインクリメントする操作を繰り返し行い，各手法の性能を評価する．

以降，ID が X であるようなレコードの VALUE をインクリメントすることを,「データ X をインクリメントする」と呼ぶことにする．本実験において，同時実行されるトランザクションについて次の 4 パターンに分類できる．

（1） P_1 のデータ A をインクリメントし，P_4 のデータ A をインクリメントする

（2） P_1 のデータ A をインクリメントし，P_4 のデータ B をインクリメントする

（3） P_1 のデータ B をインクリメントし，P_4 のデータ A をインクリメントする

（4） P_1 のデータ B をインクリメントし，P_4 のデータ B をインクリメントする

(1) においては，2 つのトランザクションが同じデータに対し読み書きを行う．(2)，(3) においては，2 つのトランザクションの更新伝搬範囲は重複するものの，同じデータに対する書き込みは発生しないため競合はしない．(4) においては，2 つのトランザクションがデータ B をインクリメントするが，P_1 の更新は P_2 まで，P_4 の更新は P_3 までしか伝搬しないため，1 つのデータベースサーバにおいて 2 つのトランザクションが同時実行されることはない．

本実験では 2 種の実験を行った．1 つ目の実験は P_4 において常にデータ B をインクリメントし続け，P_1 において一定割合でデータ A とデータ B をインクリメントする．この割合を変動させることで更新伝搬範囲を任意の割合で重複させ，この重複の割合に対しコミット成功数やコミット成功割合がどのように変化するかを調査する．先に述べたパターンで表現すると，このワークロードでは (2) と (4) のパターンが発生し，この割合を調整する．2 つ目の実験は P_4 において常にデータ A をインクリメントし続け，P_1 において一定割合でデータ A とデータ B をインクリメントする．この割合を変動させることでトランザクションを任意の割合で競合させ，この競合の割合に対しコミット成功数やコミット成功割合がどのように変化するかを調査する．このワークロードでは (1) と (3) のパターンが発生し，この割合を調整する．なお，実験は調整した各割合ごとに 1 分間トランザクションを入力し続け，この結果を計測する．トランザクションに含まれる SQL 文は 100 ミリ秒でタイムアウトするよう設定する．

また，提案手法と比較するベースラインとして以下に述べる手法を採用する．あるピアにおいてトランザクションが入力された際，このピアからデータ統合システムに参加する全てのピアに向けてデータベース全体に対するロックを要求する．全てのピアにおいてロックが取得できた場合のみトランザクションを実行する．ベースラインとなる手法は大域コンポーネントに依存していないものが望ましいため，このシンプルな手法を選択する．ベースラインにおける分散合意プロトコルは，提案手法と同じく木構造 2 相コミットを採用する．これは分散トラン

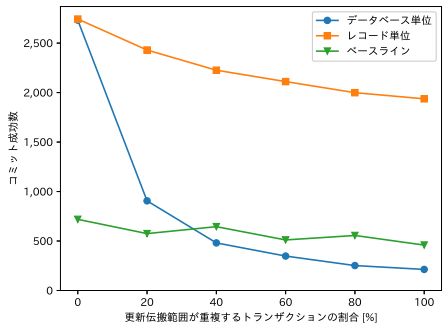

図 3　更新伝搬範囲の重複割合に対するコミット成功数の変化

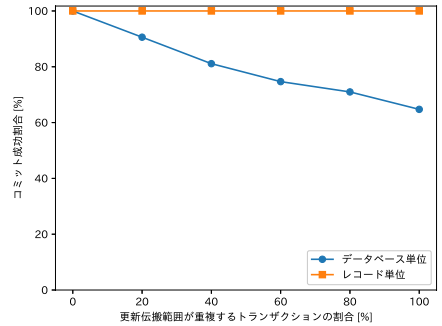

図 4　更新伝搬範囲の重複割合に対するコミット成功割合の変化

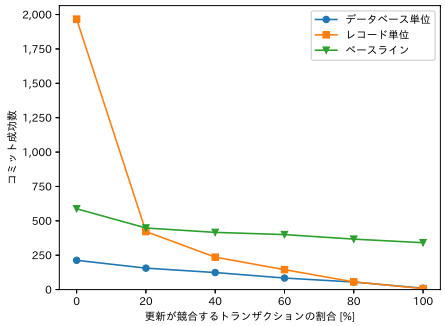

図 5　更新の競合割合に対するコミット成功数の変化

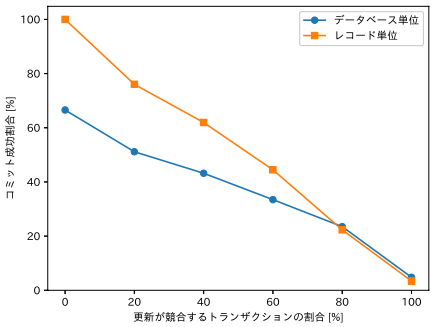

図 6　更新の競合割合に対するコミット成功割合の変化

ザクション制御のうち，同時実行制御手法の部分に注目した比較を行うためである．

4.3　実験結果

4.3.1　更新伝搬範囲の重複割合に対する性能

更新伝搬範囲の重複割合を調整したワークロードにおいてコミット成功数とコミット成功割合を計測する．図 3 及び図 4 に実験結果を示す．

実験結果から，チケット粒度をレコード単位とした提案手法が最も多くコミットを成功させていることがわかる．チケット粒度をデータベース単位とした提案手法は，更新伝搬範囲が全く重複しないワークロードにおいてチケット粒度をレコード単位とした提案手法と同程度トランザクションをコミットできているが，重複割合が大きくなるほどコミット成功数が減少している．これは 3 章で述べたように，あるデータベースで同時実行されるトランザクションは 1 つしかコミットできないという制限が発生するためである．なお，チケット粒度がレコード単位となる手法において更新伝搬範囲の重複割合が増加するほどコミット成功数が下がっている．これは同時実行制御手法による特性ではなく，ワークロードの特性であると考えられる．つまり，更新伝搬範囲の重複割合が高いほど，データ A をインクリイメントするトランザクションを多く実行するため，ネットワークの端まで更新が伝搬する．よってデータ A をインクリメントするトランザクションはデータ B をインクリメントするトランザクションよりも実行に時間を要するためである．一方でベースラインにおいては，同様の理由でコミット成功数が緩やかな減少を見せているが，どの割合においてもコミット数に大きな違いは見られない．

また，コミット成功割合について，チケット粒度をレコード単位とした提案手法においては，更新伝搬範囲の重複割合に全く影響を受けず，コミット成功割合が常に 100%になっていることがわかる．これはこのワークロードにおいて競合するトランザクションが全く無いため常にコミットを成功できるからである．一方でチケット粒度をデータベース単位としたトランザクションにおいては，成功割合が徐々に下がっている．これもコミット成功数減少の原因と同様に，各データベースにおいて同時実行されるトランザクションは 1 つしかコミットできないという制限が原因である．なお，図 4 にベースラインは図示していない．ベースラインは全データベースサーバにおいてロックが取得できない限りトランザクションが実行されることはなくアボートが発生し得ないため，コミット成功割合が常に 100%であった．

4.3.2　更新の競合割合に対する性能

更新の競合割合を調整したワークロードにおいてコミット成功数とコミット成功割合を計測する．図 5 及び図 6 に実験結果を示す．

実験結果から，全トランザクションの 20%が競合するようなトランザクションにおいてはベースラインと同程度，それ以上の競合率のワークロードにおいてはベースラインよりコミットを成功させることができず，チケット粒度をデータベース単位としたチケットベース手法のコミット成功数に近づいていることが分かる．

提案手法が競合率の高いワークロードにおいてベースラインに劣る原因は次のように考えられる．例えばデータ A をインクリメントするトランザクション T_1 が P_1 に入力され，同じくデータ A をインクリメントするトランザクション T_2 が P_4

に入力された場合を考える．これらが更新伝搬していく時，T_1 が P_1 と P_2 においてチケット取得命令によりチケットにロックを取得し，T_2 が P_4 と P_3 においてチケットにロックを取得したとする．それぞれのトランザクションがさらに隣のピアに伝搬した際，お互いがお互いのチケットに対するロックの開放を待つためデッドロック状態となる．この時，例えば T_1 が先にタイムアウトした場合，木構造 2 相コミットによって全ピアにおいて T_1 がアボートされる．しかし両端から更新が伝搬していくという実験設定の性質上，チケットに対するロックの開放待ちはほとんど同時に発生しているため，T_1 がタイムアウトしてからアボートするまでに T_2 がタイムアウトを起こしてしまい，結果として両方のトランザクションがアボートするといった状況が頻発する．これにより他のトランザクションを実行する時間が減ってしまい，コミット成功数の減少につながっていると考えられる．

いずれの粒度のチケットベース手法においても，競合のしやすいワークロードではコミット成功割合も同様に著しく低下していることがわかる．これもコミット成功数の減少の原因と同じく，両端から伝搬するトランザクションの両方がアボートする状況が頻発することが原因であると考えられる．

5 関 連 研 究

5.1 CDSS コンセプトに基づいたデータ統合アーキテクチャ

CDSS のコンセプトを採用したデータ統合アーキテクチャに関する研究について簡単に述べる．P2P 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 [2] がある．このシステムは Semantic web が膨大な量のデータを処理する際の問題を解決することを目的として提案された．グローバルスキーマを使わず，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．

この後継プロジェクトとして ORCHESTRA [3] がある．これは 1 章でも述べたとおり，バイオインフォマティクスの分野における問題を解決することを目的としている．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．

この ORCHESTRA に大きな影響を受け提案されたのが，本論文で用いた Dejima アーキテクチャ [4] [5] である．Dejima アーキテクチャは ORCHESTRA で重要視されなかった大域的なデータの一貫性を重視しつつ，複数のピア間でさらに効率的なデータ交換を行う．

5.2 分散トランザクションの同時実行制御

分散トランザクションの同時実行制御は，その多くが大域コンポーネントを利用する手法となっている．この場合，単一のデータベースにおけるトランザクション制御と同じような制御手法を容易に応用できる．一方で大域コンポーネントを利用しない同時実行制御はそのような手法の応用が困難である．

大域コンポーネントを利用しない同時実行制御として有名な手法に，本論文で扱ったチケットベース手法 [9] [10] がある．この詳細は 3 章で説明した．

また近年，分散トランザクションの同時実行制御として Calvin [13] などの決定論的アプローチ [14] が提案されている．これは主にレプリケーションの際，2 相コミットのような分散合意プロトコルを削減できるというメリットがある．他にもスケーラビリティの向上など様々なメリットが主張されている．

6 ま と め

本論文では，P2P 型のデータ統合アーキテクチャである Dejima アーキテクチャに対し，大域コンポーネントに依存しない分散トランザクション制御における同時実行制御手法として，SI を前提としたチケットベース手法を提案した．チケットの粒度を変更可能とすることで，その性能を改善できることを評価実験によって確認した．特に更新伝搬範囲の重複率が高いようなワークロードにおいて，チケットの粒度を細かくした手法による性能向上が顕著であった．

今後の課題としては，Dejima アーキテクチャに対しさらに効率の良い分散トランザクション制御手法を提案することが挙げられる．また，ネットワークトポロジによる制御手法の性能の変化などについて検証することも今後の課題である．

文 献

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

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

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

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

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

[6] Gerhard Weikum and Gottfried Vossen. *Transactional information systems: theory, algorithms, and the practice of concurrency control and recovery*. Elsevier, 2001.

[7] Hal Berenson et al. A critique of ansi sql isolation levels. *ACM SIGMOD Record*, 24(2):1–10, 1995.

[8] Alan Fekete et al. Making snapshot isolation serializable. *ACM TODS*, 30(2):492–528, 2005.

[9] Dimitrios Georgakopoulos et al. On serializability of multidatabase transactions through forced local conflicts. In *ICDE*, pages 314–323, 1991.

[10] Dimitrios Georgakopoulos et al. Using tickets to enforce the serializability of multidatabase transactions. *TKDE*, 6(1):166–180, 1994.

[11] George Samaras et al. Two-phase commit optimizations in a commercial distributed environment. *Distributed and Parallel Databases*, 3(4):325–360, 1995.

[12] Dan RK Ports et al. Serializable snapshot isolation in postgresql. *VLDB Endowment*, 5(12), 2012.

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

[14] Daniel J Abadi et al. An overview of deterministic database systems. *Communications of the ACM*, 61(9):78–88, 2018.