Vol.2015-OS-134 No.7
2015/8/4

複数の一貫性レベルを保証可能なマルチマスタデータレプリケーション

太田 篤[1]　松野 雅也[1]　川島 龍太[1]　松尾 啓志[1]

概要：システムの高可用性，信頼性を実現するためにデータベースの複製（レプリカ）を作成する技術であるレプリケーションが広く利用されている．しかし，アプリケーションごとに要求される最低限の一貫性を保証するレプリケーションプロトコルを個別に実装することは，管理や運用の面でユーザの負担となる．したがって，複数の一貫性レベルを保証可能な，汎用性のあるレプリケーションプロトコルが求められる．既存研究である Multi-Consistency Data Replication（McRep）は，ミドルウェアベースのレプリケーションプロトコルであり，かつ複数の一貫性レベルが保証可能である．しかし，クライアント数の増加につれてミドルウェア部に配置されたレプリケーション制御を行うサーバ（レプリケータ）が単一障害点かつ性能ボトルネックとなる問題がある．これらを回避する方法として，Deferred-update によるマルチマスタレプリケーションが研究されている．本研究では，Deferred-update を用いたマルチマスタレプリケーションにおける複数一貫性制御手法を提案する．提案手法では，McRep がレプリケータで行っていた一貫性制御を各レプリカにて行う．マルチマスタレプリケーションで一貫性制御を行うことにより，特定のサーバが単一障害点となる問題を解決し，システムの可用性の向上を実現した．評価から Read-heavy なワークロードにおいて一貫性レベルが One-copy serializability の場合，McRep では 200 クライアントでスループットが頭打ちしていたところ，提案手法では 400 クライアントまでスループットが向上し，性能ボトルネックを解消した．さらに，Sequential Consistency の場合は McRep と同等のスループットが得られることを確認した．

キーワード：分散データベース，データレプリケーション，一貫性レベル，Deferred-update，ミドルウェア

1. はじめに

近年では Google や Amazon，Facebook に代表されるように多様な Web サービスが存在し，広く普及している．これに伴って，サービス利用者も増加し続け，システムも大規模化している．このような状況において，データベース層で障害が発生した場合はシステム全体に影響が及ぶ．そこで，あるデータベースの複製を別サーバ上にリアルタイムに作成する技術であるレプリケーションが広く用いられている．以降ではデータベースの複製が存在するサーバをレプリカと呼ぶ．複数のレプリカで同一データを管理することで，負荷分散が可能となり，高可用性，耐故障性を実現することができる．しかし，レプリケーション時には，各レプリカ間でどの程度の一貫性を保証するべきかという課題がある．一般的には，より強い一貫性を保証するほど性能が犠牲になるため，アプリケーションが要求する最低限の一貫性を保証することが望ましい．そのため，レプリケーション時にレプリカ間でデータの一貫性をどの程度保証するかによって，線形化可能性（Linearizability）や順序一貫性（Sequential Consistency）など様々な一貫性レベルが提案されている．しかし，特定の一貫性レベルしか保証できない場合は利用可能なアプリケーションが限られ，汎用性に欠けてしまう．従って，複数の一貫性レベルを保証可能なレプリケーションプロトコルが望ましい．既存研究である Multi-Consistency Data Replication（McRep）[1] は，6 つの一貫性レベル [2], [3], [4], [5], [6], [7] を保証可能な手法である．McRep では，クライアントからのリクエストとレプリカからの応答を中継するミドルウェア部にレプリケータと呼ばれる一貫性制御を行うサーバを配置したミドルウェアベースのレプリケーション [8], [9] である．レプリケータではレプリカからの更新トランザクション応答を受信した順にトランザクションの更新結果をキューに格納することで，更新結果の直列化を行う．またレプリケータでは，キュー内の更新結果に固有のバージョン番号を付

[1] 名古屋工業大学
Nagoya Institute of Technology

与し，このバージョン番号と対応するように与えられた，システム全体の最新バージョン番号，各クライアントが観測した最新バージョン番号及び各レプリカのバージョン番号を管理する．そして，レプリケータがキューに格納された更新結果を各レプリカに伝播しデータベースを更新することで，レプリカ間の一貫性を制御する．しかし，この手法はミドルウェアベースのレプリケーションであるため，クライアント数の増加につれてレプリケータがボトルネックとなってしまう恐れがある他，単一障害点となる問題がある．

ボトルネックや単一障害点を回避する方法として，Deferred-update[10] によるマルチマスタレプリケーションが研究されている．この手法では，[11] の考え方に基づいてトランザクションの並行処理を行う．具体的には，各レプリカはトランザクションの実行中に他のレプリカとの調停を行わず，クライアントがコミットリクエストを発行した時に，全順序マルチキャスト [12] を用いて必要なデータ項目を全てのレプリカに送信する．全順序マルチキャストにより送信された更新結果は全てのレプリカに同一の順序で受信されることが保証されるため，更新結果を受信した際に，並行に実行され，コミットされた他の更新結果と競合解決することで直列化可能性を保つ．従って，Deferred-update によるレプリケーションでは耐障害性のある全順序マルチキャストを用いることでシステムの可用性の向上を見込める他，Read-only トランザクションならば，全順序マルチキャストによる直列化可能性の検査の必要が無く，レプリカの台数に伴う性能の向上が期待できる．

本研究では，Deferred-update にマルチマスタレプリケーションに，McRep の一貫性制御手法の考え方を導入し，複数一貫性制御可能なレプリケーションプロトコルを提案する．マルチマスタレプリケーションで一貫性制御を行うことにより，特定のサーバが単一障害点となる問題を解決し，システムの可用性の向上を実現する．提案手法では，全てのリクエストを中継するようなレプリケータは存在せず，各レプリカが直接クライアントからリクエストを受信するため，McRep がレプリケータで行っていた各バージョン番号の管理を，各レプリカが独立して行う．

本論文の構成は以下の通りである．まず，第 2 章では研究背景として，レプリケーションにおける課題であるレプリカ間の一貫性について説明する．第 3 章では，既存手法である McRep について紹介し，第 4 章で本研究での提案とその実装について述べる．そして，第 5 章で評価，考察を行い，第 6 章で本研究のまとめを述べる．

2. レプリケーションにおける一貫性

レプリケーションを行う際には，レプリカ間の一貫性をどの程度保証するべきかという点も考慮する必要がある．一般的には，より強い一貫性を保証するほど性能が低下するため，アプリケーションにとって最低限必要な一貫性を保証することが望ましい．以下に代表的な一貫性レベルについて説明する．

- 線形化可能性（Linearizability）
  線形化可能性 [2], [13] では，実行される一連のトランザクションが実時間で順序付けられる必要があるが，この時，並行実行されたトランザクション間の順序については実時間順になる必要はなく，レプリカ間で同一の順序となることを保証すればよい．
- 順序一貫性（Sequential Consistency）
  順序一貫性 [3] では，同一クライアントが実行した一連のトランザクションは実時間で順序付けられる必要がある．従って，各クライアントからは一貫性レベルが線形化可能性であるように観測される．
- 直列化可能性（One-Copy Serializability）
  直列化可能性 [4] では，実行された全てのトランザクションが全レプリカにおいて同一順序で逐次実行された結果と同じであることのみを保証すればよく，一連のトランザクションの処理順序に関して制約はない．従って，各クライアントは古いデータを読み出す可能性があるため，一貫性のない状態を観測する恐れがある．

上記以外にも様々な一貫性レベルが提案されているが，一般にはアプリケーションごとに適切な一貫性レベルは異なるため，複数の一貫性レベルを保証可能な汎用性のあるレプリケーションプロトコルが望ましいといえる．

3. ミドルウェアベースによる複数一貫性制御手法とその問題点

本章では，複数の一貫性レベルを保証可能なレプリケーション手法である McRep について紹介し，その問題点を指摘する．

3.1 McRep の概要

McRep は，ミドルウェア方式の非同期レプリケーションであり，6 つの一貫性レベル（Linearizability，Sequential Consistency，One-copy serializability，Session Snapshot Isolation[6]，Generalized Snapshot Isolation[7]，Causal Consistency[5]）を保証可能とする手法である．McRep の基本的な構造はミドルウェアベースのプロトコルである Simple Replica Control Algorithm（SRCA）[14] に基づく．McRep では，図 1 に示すように，各クライアントからのリクエストを仲介するレプリケータにおいて，トランザクションの更新結果（トランザクション実行時に書き込まれたデータ項目）を格納するキューを保持しており，各更新結果には順にバージョン番号を付与する．そして，このキュー内の特定のエントリを示す 4 種類のバージョン番号を用い

表 1　一貫性レベルに対応する QVN の値

一貫性レベル	QVN
Linearizability	GVN
Sequential consistency	SVN
One-copy serializability	0
Session Snapshot Isolation	SVN
Generalized Snapshot Isolation	0
Causal Consistency	SVN

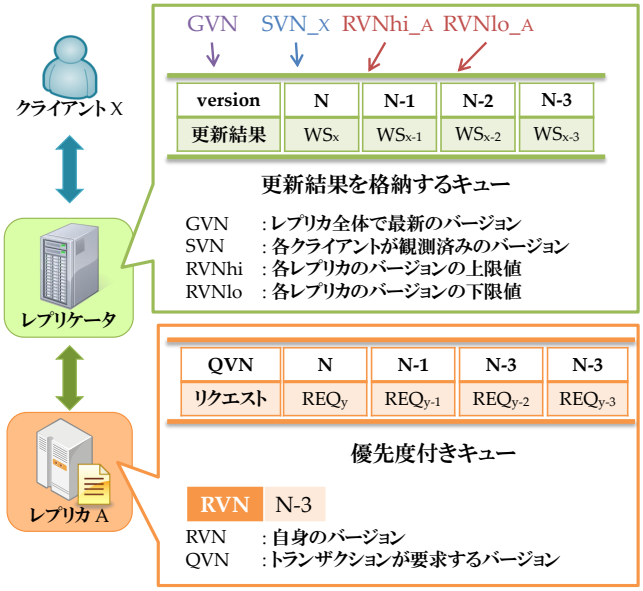

図 1　McRep における構成要素

て複数の一貫性レベルを制御する．GVN（Global Version Number）はレプリカ全体での最新のバージョンを示し，SVN（Session Version Number）は各クライアントごとの観測済みのバージョンを示す．また，レプリカのバージョン（Replica Version Number，RVN）の上限値及び下限値である RVNhi，RVNlo はレプリカにおいてキュー内の更新結果がどこまで反映されているかを管理するために用いる．このように，McRep ではレプリケータにおいて各クライアント及び各レプリカに対応するバージョン番号を管理する．一方，レプリカではレプリケータからのリクエストが格納される優先度付きキューを保持する．そして，各リクエストに付与されているバージョン番号である QVN と自身のバージョン番号である RVN を用いてトランザクション処理の制御を行う．QVN（reQuired Version Number）はそのトランザクション操作が要求するバージョン番号を示し，表 1 のように求める．自身のバージョン番号である RVN が（RVN≥QVN）を満たす場合にキューからエントリを取り出し，対応するトランザクション操作を排他的に実行する． 以上のように McRep では，レプリケータにおいて各トランザクションの更新結果をバージョン番号を付

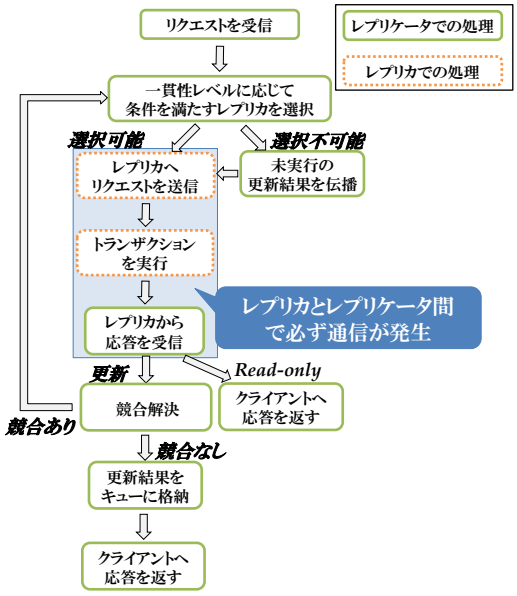

図 2　McRep における動作フロー

与してバッファリングし，各レプリカがそのバージョン番号に基づいて実行制御を行うことで複数の一貫性レベルを保証可能とする．

3.2　動作フロー

McRep における動作フローを図 2 に示す．レプリケータはクライアントからリクエストを受け取ると，GVN，SVN，RVNhi の 3 つのバージョン番号を用いて，設定されている一貫性レベルを満たすレプリカを選択する．条件を満たすレプリカが存在する場合は，そのレプリカへクライアントのリクエストを送信する．条件を満たすレプリカが存在しない場合は，任意のレプリカを選択し，レプリケータ内のキューに格納されている更新結果の内，選択したレプリカのデータベースに書き込まれていないもの，つまり，RVNhi より大きいバージョンを持つ更新結果を伝播する．更新結果を受け取ったレプリカは，その更新結果をコミットし，自身のバージョン番号である RVN をインクリメントし，（RVN=GVN）を満たすようになる．その後，レプリケータはそのレプリカへクライアントのリクエストを送信する．そして，レプリカは，そのトランザクション処理を実行し，レプリケータへ応答を返す．この時，一貫性レベルが Causal Consistency であれば実行したトランザクションの更新結果を即時にデータベースへ書き込み，コミットを行うが，それ以外であればこの時点では更新結果をデータベースへは書き込まず，コミットは行わない．そして，レプリケータはレプリカから応答を受け取ると，トランザクションの種類が Read-Only トランザクションである場合は単純にクライアントへ応答を返す．一方，更新トランザクションである場合は，一貫性レベルが Causal Consistency であれば受け取った更新結果を即時にキューへ格納した後，クライアントへ応答を返す．それ以外であ

表 2　各バージョン番号の更新動作

タイミング	更新動作
Read-Only トランザクション に対する応答時	SVN = max(SVN，RVN)
更新トランザクション に対する応答時	SVN = ++GVN
更新結果の伝搬に対する応答時	RVNlo = RVN
更新結果の伝搬時	RVNhi = GVN

れば，競合解決を行った後で，更新結果をキューへ格納し，クライアントへ応答を返す．また，この一連の処理において，レプリケータ内で管理している 4 種類のバージョン番号は表 2 に示すように更新される．

3.3 問題点

McRep は，トランザクション処理を各レプリカにおいて並行実行可能であり，かつレプリカ間で同期的に実行する必要がない非同期レプリケーションである．そのため，レプリカ数に応じて性能がスケールアウトすることが期待される．その上，6 つの一貫性レベルをサポートしており，汎用性のあるレプリケーションプロトコルであるといえる．しかし，McRep はミドルウェア方式のレプリケーションであるため，クライアントからの全リクエストがレプリケータを経由する必要がある．その結果，クライアント数の増加につれてレプリケータがシステム全体のボトルネックかつ単一障害点となるという問題がある．

4. 提案手法

本章では，Deferred-update によるマルチマスタレプリケーションで複数一貫性制御手法を実現する手法を提案する．

4.1 提案手法の処理概要

McRep では，ミドルウェアが全てのリクエストを中継してレプリケーション制御を行うため，ミドルウェアが単一障害点となるという問題が存在した．これを解決するために，Deferred-update によるマルチマスタレプリケーションでの複数一貫性制御手法を提案する．この手法では各レプリカが，McRep がレプリケータで管理していた更新結果を保持するキュー及び各バージョン番号の一部（RVN，RVNlo，SVN，RVNhi）をレプリカとクライアントで管理する．

この手法におけるレプリカでのトランザクション処理は，Deferred-update でのトランザクション処理（トランザクションを投機実行した後，トランザクションをコミットする順序を全順序マルチキャストで決定し，競合検出によりコミット可能かどうか判定）に加え，各バージョン番号を用いて，自身のレプリカが一貫性レベルを満たしているか判断する．従って，あるレプリカが，クライアントからトランザクションを受信してから全順序マルチキャストによりコミット順序を決定するまでの大まかな処理の流れを以下に示す．

(1) クライアントからトランザクション開始リクエストを受信
(2) リクエストを受信したレプリカが一貫性レベルを満たしているか判断
(3) トランザクションの投機実行
(4) 全順序マルチキャストによるトランザクションのコミット順序決定

また，全順序マルチキャストによるコミット順序の決定からクライアントに応答を返すまでの処理を以下に示す．

(1) トランザクションのコミット順序決定
(2) トランザクションの競合検出
(3) トランザクションのコミット
(4) 更新結果のキューへの格納
(5) 全順序マルチキャストによる更新結果の順序決定

以下の項では，まず各バージョン番号の管理と更新動作について述べ，次に，処理の詳細について説明する．

4.2 各バージョン番号の管理

提案手法では，McRep がレプリケータで管理していたバージョン番号（RVN，RVNhi，RVNlo，GVN，SVN）の内，RVN，RVNlo，RVNhi の管理をレプリカで行い，SVN はクライントが管理する．また，GVN をレプリカで管理するためには各トランザクションを投機実行する前に調停が必要となるため，ステートマシーンレプリケーション [15] と同様の動作となり，Deferred-update の利点が失われてしまう．そのため，提案手法では，Linearizability 一貫性レベルをサポートせず，従って GVN を使用しない．GVN 以外のバージョン番号（RVN，RVNhi，RVNlo，SVN）は次のように管理する．

- RVN
  RVN は自レプリカのバージョン番号を表す．提案手法では，McRep と同様に更新結果のデータベースへの書き込みが完了した時点で RVN を更新する．
- RVNhi
  RVNhi は，各レプリカごとのバージョンの上限値を示し，レプリカごとに独立して管理可能である．提案手法では全順序マルチキャストによりコミット順序が決定したトランザクションに競合が存在せずコミット成功とみなされた時，自 RVNhi を更新する．また McRep において，他レプリカの RVNhi が必要となるのはレプリケータが更新結果を伝播するとき，あるいは一貫性レベルを満たすレプリカを選択するときのみである．提案手法では，特定のサーバがこのような集中的な一貫性制御を行わないため，他レプリカの RVNhi を使用する必要はない．

表 3　提案手法での各バージョン番号の更新動作

タイミング	更新動作
Read-only トランザクションに対する応答時	SVN = max(SVN，RVN)
全順序マルチキャストによるトランザクション終了処理メッセージ受信時	RVNlo = RVN
トランザクションのコミット成功時	++RVNhi
更新トランザクションに対する応答時	SVN = RVNhi
更新結果の書き込み時	++RVN

- RVNlo
  RVNlo は，各レプリカごとのバージョンの下限値を示す．提案手法において，自レプリカの RVNlo は RVN と等価となるため，管理しない．他レプリカの RVNlo は，McRep と同様に，更新結果を格納するキューから，全てのレプリカに伝播されている要素を削除する際に使用されるため，管理が必要である．そこで，各レプリカは自身の RVN を全順序マルチキャストによる後述するトランザクション終了処理メッセージ送信時に付与し，受信時に RVNlo を更新する．
- SVN
  SVN は各クライアントが観測済みのバージョン番号であるため，クライアントが自身の SVN を記憶し，それをリクエストに付与することで，レプリカ間で同期を取る必要がなく，各レプリカごとに独立して管理可能である．また，レプリカが各トランザクションの応答に SVN を付与することで，クライアントは SVN を更新することができる．SVN は Read-only トランザクションの応答時と更新トランザクションの応答時で異なる．Read-only トランザクションの応答時には，McRep と同様に，RVN を SVN として応答に付与する．一方，更新トランザクションの応答時には，クライアントは，自身が発行した更新トランザクションをレプリカがコミットした際の RVNhi を観測するため，RVNhi を SVN として応答に付与する．

まとめると，以上の各バージョン番号は，表 3 に示すように更新される．また，クライアントが SVN を保持し，それをリクエストに付与することで，Linearizability 以外の一貫性レベルにおいて，レプリカがトランザクションの QVN を求めることができる．従って，レプリカは，QVN と RVNhi を用いて，自身が一貫性レベルを満たしているかを判断することが可能となる．

4.3　クライアントからのトランザクション受信時の処理

提案手法におけるクライアントからのリクエスト処理は全てレプリカで行われる．この処理の過程を図 3 に示す．

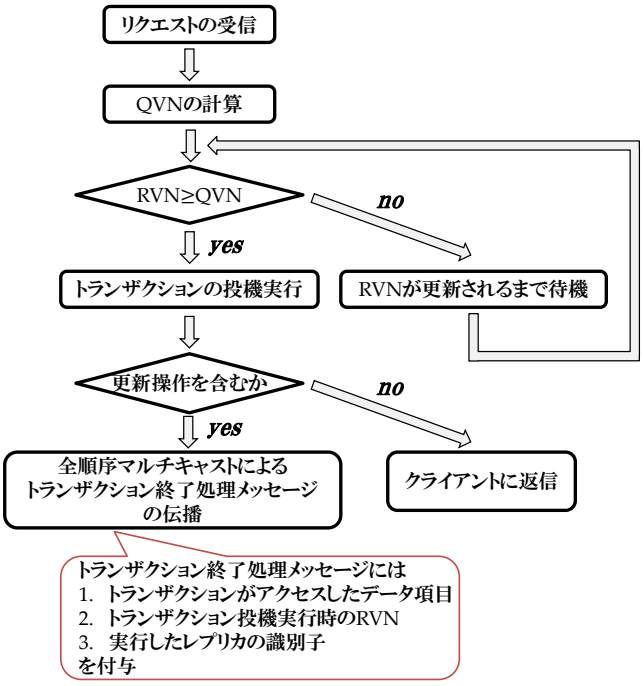

図 3　提案手法におけるクライアントからのリクエスト処理

クライアントからリクエストを受信したレプリカは，リクエストに対応するトランザクションの QVN を McRep と同様に求め，自身のレプリカのバージョン番号（RVN）が一貫性レベルを満たすかどうか判断する．具体的には，RVN<QVN であれば，RVN≥QVN となるまで，トランザクションの実行を待機する．RVN≥QVN となるとき，トランザクションの実行を開始する．

Read-only トランザクションの場合は，トランザクションを実行し，クライアントへ応答を返す．このとき，要求元クライアントは実行時のバージョンである RVN を観測するため，応答に RVN を付与する．更新トランザクションの場合は，まず，トランザクションを投機実行する．続いて，このトランザクションによる実行順序を決定するために，全順序マルチキャストプロトコルにより，トランザクション終了処理メッセージを全てのレプリカへ伝播する．トランザクション終了処理メッセージには，トランザクションがアクセスしたデータ項目，トランザクションを投機実行したレプリカ ID（T_R）及び実行時のバージョン（T_{ver}）を付与する．T_R，T_{ver} として，それぞれ自レプリカの ID と RVN を設定する．

4.3.1　全順序マルチキャストによるトランザクション終了処理メッセージ受信時の処理

全順序マルチキャストプロトコルで受信したトランザクション終了処理メッセージのレプリカでの処理過程を図 4 に示す．まず，レプリカでは，メッセージに付与されたトランザクションがアクセスしたデータ項目（T_{WS}）とキュー内の更新結果との競合検出を行う，競合検出の方法は McRep における処理に基づく．メッセージに付与され

たトランザクション投機実行時のバージョン（T_{ver}）より大きいキュー内の更新結果と T_{WS} との間に重複があった場合，先にコミットされたトランザクションとの間に競合が発生するため，トランザクション間の直列化可能性を保証できない．従って，キュー内の更新結果の内，実行時の RVN 以上のバージョンを持つ更新結果と競合が発生していた場合は，トランザクションのコミットを許可しない．一方，競合が検出されない場合は，トランザクションの直列化可能性が保証されるため，トランザクションをコミットする，

トランザクションのコミットに成功した場合は，T_{WS} をキューに格納し，RVNhi を更新する．キューに格納された更新結果は並行してデータベースに書き込まれる．また，メッセージに付与された，トランザクションを投機実行したレプリカ ID（T_R）に対応する RVNlo を，T_{ver} へと更新する．そして，T_R が自身のレプリカ ID（M_R）と一致した場合は，自レプリカで受信したトランザクションであるため，クライアントへの応答に，RVNhi を SVN として付与し，返信する．

競合が検出され，トランザクションのコミットに失敗した場合は，まず，T_R が M_R と一致するか確認する．一致しなかった場合は，自レプリカで投機実行したトランザクションでないため，メッセージを破棄し，処理を終了する．一致した場合は，自レプリカで投機実行したトランザクションであるため，トランザクションのロールバックを行う．次に，レプリカは，新しいバージョンにおいてトランザクションを再実行することで，競合の回避を試みる．そのために，トランザクションの QVN を $T_{ver}+1$ として処理を再開する．また，この再実行の回数が設定された回数を超えた場合は，トランザクションをアボートし，要求元クライアントに競合が発生したことを伝える．

5. 評価実験

本章では，ミドルウェアベースのレプリケーションである McRep とマルチマスタレプリケーションを用いることで単一障害点を回避した提案手法を比較し，提案手法において，レプリカが分散してリクエストの処理を行うことによるスループットへの影響を調査した．実装において，レプリカとして PostgreSQL[16] を用い，レプリケータとして pgpool-II[17] を用いた．また，提案手法では，各レプリカ上に，Paxos[18] による全順序マルチキャストを遂行するプロセスを LibPaxos[19] を用いて実装した．PostgreSQL/pgpool-II はマルチプロセス構成であり，クライアントごとに別々のプロセスで処理されるため，McRep 及び提案手法におけるレプリケータ内のキューは，全プロセスから観測できるように共有メモリ上にリングバッファとして実装した．従って，バッファサイズに制限があるため，バッファが一杯になった際には更新トランザクション

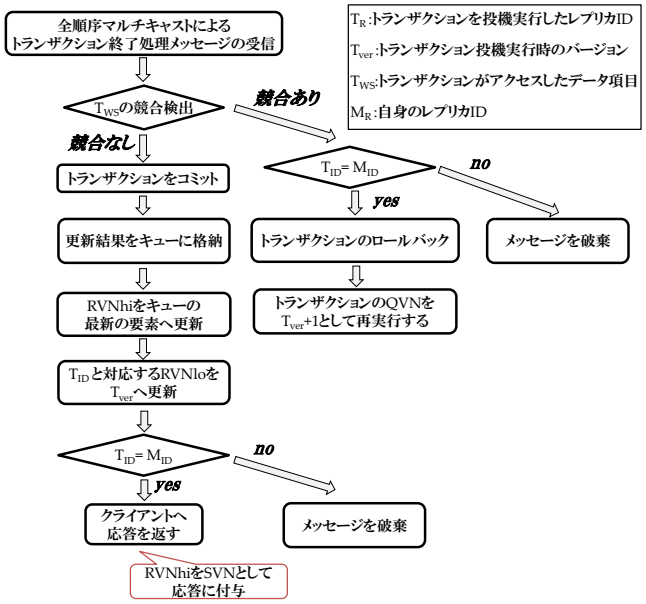

図 4 提案手法における全順序マルチキャストによるトランザクション終了処理メッセージ受信時の処理

が待たされることとなり，バッファサイズによって更新トランザクションの性能が大きく左右される．そのため，以降の評価においては待機が発生しない十分なバッファサイズで行った．また，Paxos の最適化として，コーディネータがフェーズ 1 のインスタンスを事前実行する手法 [20] を用いた．

5.1 評価環境

評価環境を表 4 に示す．ベンチマークには PostgreSQL 付属の pgbench を用いた．この際，100,000 エントリを持つテーブルを用意し，更新トランザクションではこのテーブルから 1 つのエントリを選択する UPDATE 文を，Read-only トランザクションでは同一のテーブルから 1 つ選択し読みだす SELECT 文を使用した．以降の評価は，McRep，提案手法についてそれぞれ行った．また，提案手法における Paxos の最適化であるフェーズ 1 の事前実行サイズは 128 とした．評価内容として，更新トランザクションと Read-only トランザクションの比率が 1 対 9，5 対 5 及び 9 対 1 の場合において，クライアント数及びレプリカ数を変化させた場合におけるスループットを評価した．また，以降の評価結果はいずれも pgbench で 60 秒間トランザクションを実行し，それを 10 回繰り返した際のスループットの平均を測定した．

5.2 クライアント数による影響

まず，レプリカ数を固定してクライアント数を変化させた場合のスループットを評価した．レプリカ数を 6 で固定し，pgbench のクライアント数を 50 から 400 まで変化させた際の評価結果を図 5 上部に示す．一貫性レベルが Sequential Consistency の場合は，SVN または QVN が

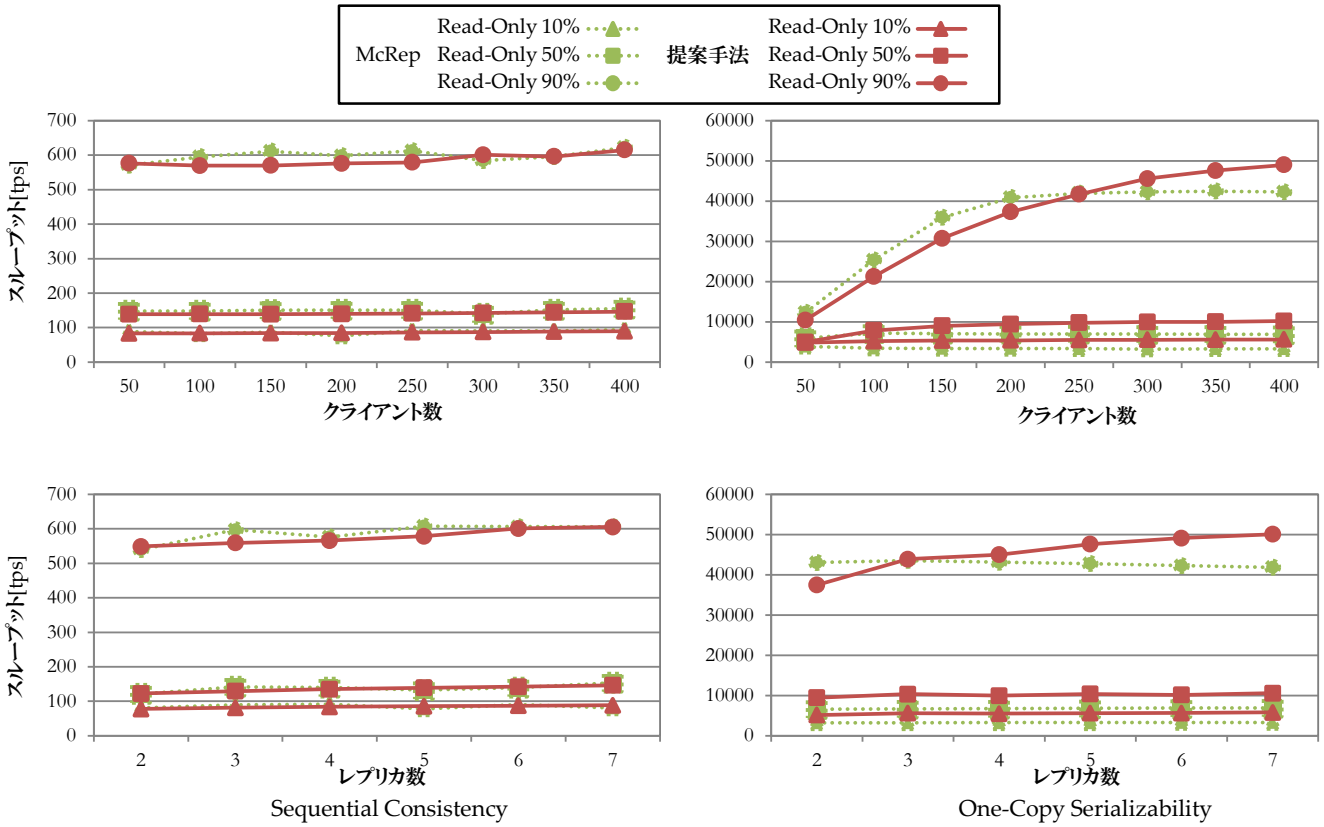

図 5　クライアント数及びレプリカ数に応じたスループットの推移

表 4　評価環境

	PostgreSQL	pgpool-II
version	9.3.5	3.3.3
OS	Linux 3.5.0-23-amd64 Ubuntu 12.04 Server	
CPU	Intel(R) Core(TM) i5 3470 @ 3.2GHz	
Memory	16GB	
Network	1000BASE-T	

更新されるたびに，レプリカの RVN を更新する必要がある．このためトランザクションを受信してから実行するまでの間，まだデータベースに書き込まれていない更新結果を受信するまでの時間と，それらをデータベースへ書き込むまでの待機時間が発生する．この待機時間のために McRep，提案手法ともにクライアント数が増加してもスループットが向上しない．一方，一貫性レベルが One-copy serializability の場合は，この待機時間は発生しないため，両手法ともにクライアント数の増加に伴いスループットが向上している．しかし，McRep では，レプリケータがボトルネックとなり，200 クライアントで性能が頭打ちしている一方で，提案手法では 400 クライアントまで性能が向上している．また，両手法において，更新トランザクションの割合が増大するほど，スループットが低下している．この性能低下は，McRep では，レプリケータがクライアントからのリクエストを全て処理することに加え，更新トランザクションの処理も行うことによる処理量の増加に起因しているが，提案手法では，クライアントからの処理を分散しているため，この性能低下は主に全順序マルチキャストによる通信遅延の増加に起因している．従って，McRep に比べ，性能低下を抑制できたのは，フェーズ 1 の事前実行により，通信遅延が削減されたためである．

5.3　レプリカ数による影響

次に，クライアント数を固定し，レプリカ数を変化させた場合のスループットを評価した．評価では，十分な負荷を与えるために，pgbench でクライアントを 400 とし，レプリカ数を 2 台から 7 台まで変化させた場合のスループットの推移を測定した．評価結果を図 5 下部に示す．一貫性レベルが Sequential Consistency の場合は，レプリカ数の増加に伴う顕著なスループットの向上は見られない．これは 5.2 項と同様の理由による．次に，一貫性レベルが One-copy serializability の場合に注目すると，McRep ではレプリカ数が増加してもスループットが向上していないことがわかる．これは，McRep がミドルウェア方式のレプリケーションであるために，レプリケータがボトルネックとなって，性能がスケールアウトしなかったと考えられる．一方，マルチマスタ方式である提案手法でもレプリカ数の増加に伴うスループットの向上は，Read-only トランザクションの割合が 9 割を占めるワークロードにおいて，レプリカ数 2 から 7 まで，スループットは 1.3 倍ほどしか向上

していない．これは，レプリカ数が増えるほど全順序マルチキャストにかかる通信コストが増加するためである．

6. まとめ

本研究では，既存のミドルウェア方式のレプリケーション手法である McRep のプロトコルをミドルウェアベースではなく，マルチマスタレプリケーションに適応する手法を提案した．ミドルウェア方式のレプリケーションにおいては，クライアントからの全リクエストがミドルウェアを経由する必要があり，ミドルウェアにおいて障害が発生するとクライアントがシステムを利用できなくなる問題が存在した．これに対し，保証できる一貫性に制限はあるが，提案手法ではマルチマスタレプリケーションで制御を行うことで，特定のサーバが単一障害点になる問題を解決し，システムの可用性の向上を実現した．評価から Read-heavy なワークロードで，一貫性レベルが One-copy serializability の場合は，McRep では 200 クライアントでスループットが頭打ちしていたのに対し，提案手法では 400 クライアントまでスループットが向上し，性能ボトルネックの解消を確認した．さらに，Sequential Consistency の場合は同等のスループットが得られることを確認した．

現状は，フルレプリケーションのみを想定したプロトコルとなっているが，今後の課題としては，一貫性レベルごとに部分レプリケーション作成できるようなプロトコルを設計することで，システム全体の性能向上を目指す．また，その際に，全順序の決定が必要無い一貫性レベルには，通信コストの高い全順序マルチキャストを用いないように変更することで，システム全体の通信コストを削減する．

謝辞 本研究の一部は，科研費基盤研究（C)15K00168，(C)24500113 による．

参考文献

[1] R. Al-Ekram and R. Holt. Multi-consistency Data Replication. In *Proc. IEEE 16th Int'l Conf. on Parallel and Distributed Systems (ICPADS)*, pp. 568–577, Dec. 2010.

[2] Torval Riegel, Christof Fetzer, Heiko Sturzrehm, and Pascal Felber. From Causal to Z-linearizable Transactional Memory. In *Proc. the 26th Annual ACM Symp. on Principles of Distributed Computing*, pp. 340–341, Aug. 2007.

[3] L. Lamport. How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs. *IEEE Trans. Comput.*, Vol. 28, No. 9, pp. 690–691, Sep. 1979.

[4] Philip A Bernstein and Nathan Goodman. A proof technique for concurrency control and recovery algorithms for replicated databases. *Distributed Computing*, Vol. 2, No. 1, pp. 32–44, 1987.

[5] M. Raynal, G. Thia-Kime, and M. Ahamad. From serializable to causal transactions for collaborative applications. In *Proc. EUROMICRO 23rd Conf. EUROMICRO 97. New Frontiers of Information Technology.*, pp. 314–321, Sep. 1997.

[6] Khuzaima Daudjee and Kenneth Salem. Lazy Database Replication with Snapshot Isolation. In *Proc. 32nd Int'l Conf. on Very Large Data Bases*, pp. 715–726, 2006.

[7] Sameh Elnikety, Willy Zwaenepoel, and Fernando Pedone. Database Replication Using Generalized Snapshot Isolation. In *Proc. IEEE 24th Symp. on Reliable Distributed Systems*, pp. 73–84, 2005.

[8] Sudha Krishnamurthy, W. H. Sanders, and M. Cukier. An Adaptive Quality of Service Aware Middleware for Replicated Services. *IEEE Trans. Parallel Distrib. Syst.*, Vol. 14, No. 11, pp. 1112–1125, Nov. 2003.

[9] Christian Plattner and Gustavo Alonso. Ganymed: Scalable Replication for Transactional Web Applications. In *Proc. the 5th ACM/IFIP/USENIX Int'l Conf. on Middleware*, pp. 155–174, 2004.

[10] D. Sciascia and F. Pedone. Geo-Replicated Storage with Scalable Deferred Update Replication. In *Proc. IEEE 33rd Int'l Symp. on Reliable Distributed Systems Workshops (SRDSW)*, pp. 26–29, Oct 2014.

[11] H. T. Kung and John T. Robinson. On Optimistic Methods for Concurrency Control. *ACM Trans. Database Syst.*, Vol. 6, No. 2, pp. 213–226, June 1981.

[12] Vassos Hadzilacos and Sam Toueg. Distributed Systems (2Nd Ed.). chapter Fault-tolerant Broadcasts and Related Problems, pp. 97–145. 1993.

[13] Maurice P. Herlihy and Jeannette M. Wing. Linearizability: A Correctness Condition for Concurrent Objects. *ACM Trans. Program. Lang. Syst.*, Vol. 12, No. 3, pp. 463–492, Jul. 1990.

[14] Yi Lin, Bettina Kemme, Marta Patiño Martínez, and Ricardo Jiménez-Peris. Middleware Based Data Replication Providing Snapshot Isolation. In *Proc. the ACM SIGMOD Int'l Conf. on Management of Data*, pp. 419–430, 2005.

[15] Fred B. Schneider. Implementing Fault-tolerant Services Using the State Machine Approach: A Tutorial. *ACM Comput. Surv.*, Vol. 22, No. 4, pp. 299–319, December 1990.

[16] Bruce Momjian. *PostgreSQL: introduction and concepts*, Vol. 192. Addison-Wesley New York, 2001.

[17] pgpool-II. `http://www.pgpool.net/`.

[18] Leslie Lamport. Paxos made simple. *ACM Sigact News*, Vol. 32, No. 4, pp. 18–25, 2001.

[19] Daniele Sciascia. LibPaxos. `http://libpaxos.sourceforge.net/`.

[20] P.J. Marandi, M. Primi, and F. Pedone. High performance state-machine replication. In *Proc. IEEE/IFIP 41st Int'l Conf. on Dependable Systems Networks (DSN)*, pp. 454–465, June 2011.