

複数の一貫性レベルを保証可能な バックエンドベースデータレプリケーション

太田 篤^{1,a)} 松野 雅也^{1,b)} 川島 龍太^{1,c)} 松尾 啓志^{1,d)}

受付日 2015年6月29日, 採録日 2015年12月7日

概要：システムの可用性, 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, 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）が性能のボトルネックとなるという問題がある。本研究では, McRep と同様のレプリケーション制御を実現しつつ, この問題を解決する手法を提案する。具体的には, レプリケータをレプリカのみと通信を行うバックエンド部に配置し, 各レプリカも Read-only トランザクションの一貫性制御を行う。評価から, バックエンド部で制御を行うことでレプリケータがボトルネックとなることを回避し, 提案手法では 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, 最大スループットが McRep の約 2 倍向上することを確認した。

キーワード：データベースレプリケーション, 一貫性レベル, バックエンドベース, ミドルウェア

Back-end Based Data Replication Supporting Multiple Consistency Models

ATSUSHI OHTA^{1,a)} MASAYA MATSUNO^{1,b)} RYOTA KAWASHIMA^{1,c)} HIROSHI MATSUO^{1,d)}

Received: June 29, 2015, Accepted: December 7, 2015

Abstract: Replication is widely used in parallel and distributed systems for reliability and availability. On the other hand, developers have to consider minimum consistency requirement for each application. Therefore, novel replication protocol that ensure multiple consistency models is required. Multi-Consistency Data Replication (McRep) is a middleware-based replication protocol and can support multiple consistency models. However, McRep has a potential problem that a replicator (a server controlling replications) acting as a middleware can be a performance bottleneck. We propose a backend based replication protocol to solve this problem, while ensuring same consistency models. More precisely, we place the replicator on the backend area where the replicator communicates with only replica servers, and extend the replica's role to control the consistency for read-only transactions. We implemented and evaluated both the proposal protocol and the McRep. The results showed that our protocol improve up to approximately twice throughput of transactions at a read-heavy workload in one-copy serializability as McRep's.

Keywords: replication, consistency, back-end, middleware

¹ 名古屋工業大学
Nagoya Institute of Technology, Nagoya, Aichi 466–8555,
Japan

a) a-ohta@matlab.nitech.ac.jp

b) matsuno@matlab.nitech.ac.jp

c) kawa1983@nitech.ac.jp

d) matsuo@nitech.ac.jp

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 サービスが登場し, 広く普及している。これにともなって, 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

ベース層における性能向上が重要な課題となっている。しかし、データベースサーバ単体の性能向上には限界がある。うえ、単一障害点となるという問題がある。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。レプリケーションを実現する代表的な方式には、マスタスレーブ方式 [1] とマルチマスタ方式 [2] が存在する。しかし、マスタスレーブ方式ではマスタレプリカに負荷が集中する場合があります。マルチマスタ方式では一貫性制御が複雑になり、性能向上が困難になってしまうという問題がある。これらに対して、クライアントとレプリカ間に専用のミドルウェア（以降、レプリケータと呼ぶ）を配置し、レプリケータがレプリケーションの制御を行うミドルウェア方式のレプリケーションが研究されている [3], [4]。この方式では、レプリケータがレプリケーション制御を行うことで特定のレプリカに負荷が集中することを回避し、一貫性制御も容易にすることができる。しかし、全リクエストがレプリケータを経由するために、レプリケータがシステム全体の性能ボトルネックとなる場合がある。

また一方で、いずれの方式においても、レプリケーション時にはレプリカ間におけるデータの一貫性をどの程度保証すべきかという共通の課題が存在する。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。したがって、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。Multi-Consistency Data Replication [5] (McRep) 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だが、ミドルウェア方式を採用しているため、先述した性能ボトルネックの問題をかかえている。

本研究では、既存手法である McRep を拡張し、ミドルウェア部で行っていたレプリケーション制御を、バックエンド部で実現することでレプリケータがボトルネックとなることを回避し、よりスケールアウトしやすいレプリケーション手法を提案する。本研究で規定するバックエンド部とはクライアントからのリクエストを中継せず、レプリカのみと通信するサーバが配置される領域を指す。提案手法では、レプリケータがバックエンド部で制御を行うことに加え、各レプリカが Read-only トランザクションに関して一貫性制

御を行う。これにより、レプリケータが全リクエストを処理する必要がなくなるため、既存手法と比較してレプリケータの処理量を軽減することができる。評価から、McRep と比較して、提案手法では 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、レプリケータがボトルネックとなることを回避し、最大スループットが McRep の約 2 倍向上することを確認した。

本論文の構成は以下のとおりである。まず、2 章で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。3 章では、既存手法である McRep について紹介し、4 章で本研究での提案について述べる。そして、5 章で評価および考察を行い、6 章で関連研究を示し、最後に 7 章で本研究のまとめと今後の課題を述べる。

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 [6], [7] 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、このとき、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 [8] 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。したがって、各クライアントからは一貫性レベルが線形化可能性として観測される。

- 直列化可能性 (One-Copy Serializability)

直列化可能性 [9] では、実行されたすべ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。したがって、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場合がある。

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タ

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 [10], Generalized Snapshot Isolation [1], Causal Consistency [11]) 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 SRCA [12] 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、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) の上限値および下限値である 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およ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 (reQuired Version Number) 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 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、自身のバージョン番号である RVN が ($RVN \geq QVN$) 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 一貫性レベルの判定

レプリケータは現在設定されている一貫性レベルごとに指定される条件を満たすレプリカを選択し、そのうちの 1 つのレプリカへとトランザクション操作を転送することで一貫性レベルを満たすレプリカでの実行を保証する。一貫性レベルごとに選択されるレプリカの条件を表 1 に示す。

また、レプリカは、トランザクション実行後、トランザクションがアクセスしたデータ項目の情報とそのトランザクション実行時のレプリカのバージョンをレプリケータへ返す。レプリケータでは、そのデータ項目とキュー内の更新結果のうち、トランザクション実行時のバージョン以上のものと競合検出を行う。競合する場合は、トランザクション実行時のレプリカのバージョンより大きい値の RVNhi を持つレプリカを選択し、再度トランザクションを送信する。このトランザクションの再送信回数が一定の値を超えたら、トランザクションの競合が起こりコミットに失敗したことをクライアントへ伝える。競合検出の際、Linearizability, Sequential Consistency, そして One-Copy Serializability では、応答のデータ項目すべてについて競合を検査するが、Session Snapshot Isolation と Generalized Snapshot Isolation では、応答のデータ項目のうち更新操作によるもののみに競合を検査する。Causal Consistency の場合は、競合検出そのものを行わない。

3.3 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

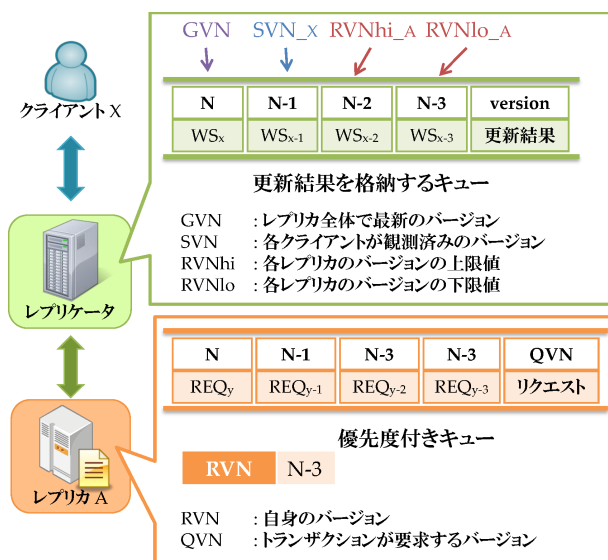


図 1 McRep における構成要素
Fig. 1 Components of the McRep protocol.

表 1 一貫性レベルごとの選択されるレプリカの条件

Table 1 Conditions of a replica to be selected from each level of consistency.

一貫性レベル	選択されるレプリカの条件
Linearizability	$RVNhi = GVN$
Sequential consistency	$RVNhi \geq SVN$
One-Copy Serializability	$RVNhi \geq 0$
Session Snapshot Isolation	$RVNhi \geq SVN$
Generalized Snapshot Isolation	$RVNhi \geq 0$
Causal Consistency	$RVNhi \geq SVN$

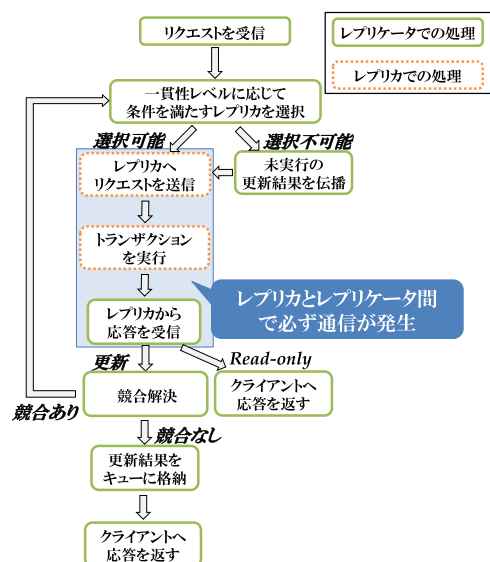


図 2 McRep における動作フロー

Fig. 2 A processing flow of the McRep.

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うち、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搬する。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。このとき、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。

3.4 問題点

McRep は、トランザクション処理を各レプリカにおい

表 2 各バージョン番号の更新動作

Table 2 Update timing of each version number.

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	$RVN_{lo} = RVN$
更新結果の伝搬時	$RVN_{hi} = GVN$

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。そのうえ、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となり、性能向上が困難になるという問題がある。

4. 提案手法

本章では、ミドルウェアベースではなくバックエンドベースで McRep と同様のレプリケーション制御を実現する手法を提案する。

4.1 概要

既存手法である McRep は、クライアントの全リクエストがレプリケータを経由するため、レプリケータが性能のボトルネックとなるという問題点が存在した。これを解決するため、提案手法では図 3 に示すように、レプリケータがミドルウェア部ではなくバックエンド部で動作するように拡張する。McRep では、レプリケータがクライアントからの全リクエストに関して一貫性制御を集中的に行っており、レプリカは単にレプリケータの要求どおりに処理を行う方式となっていた。これに対し、提案手法ではレプリケータだけでなくレプリカにおいても SVN と RVNhi のバージョン管理を行い、一貫性制御を行うように拡張する。これにより、レプリケータが集中的に行っていた一貫性制御を分担して行うことが可能となり、レプリケータの処理量が軽減される。そのうえ、レプリケータがバックエンド部で動作することで、設定されている一貫性レベルを満たすレプリカが存在するときはレプリケータが Read-only リクエストを処理する必要がないため、レプリケータの処理量を軽減することができる。

4.2 一貫性制御時の動作の違い

本節では、McRep で行っていた一貫性制御時の動作と

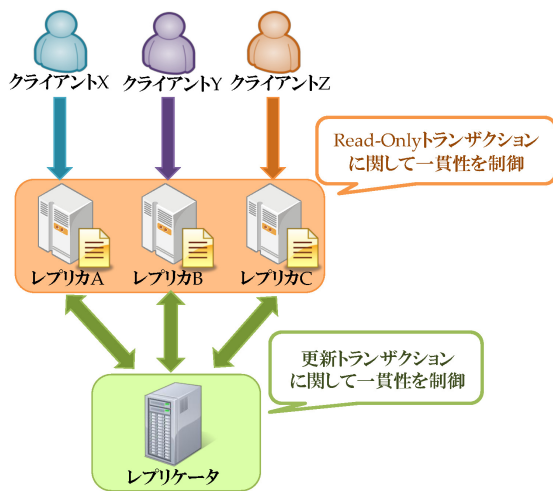


図3 バックエンドベースのレプリケーション

Fig. 3 An architecture of the proposed method.

提案手法における一貫性制御時の動作を比較し、その違いについて説明する。

- 既存手法 (McRep) での一貫性制御

既存手法である McRep において一貫性を制御する際には、レプリケーターは能動的に動作する。具体的には、レプリケーターはクライアントからトランザクション要求を受け取ると、まず一貫性レベルに応じて条件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しない場合は、レプリケーター主導で未実行の更新結果をレプリカへ伝搬することで、一貫性レベルを保証する。

- 提案手法での一貫性制御

一方、提案手法において一貫性を制御する際には、レプリケーターは受動的に動作する。具体的には、レプリカはクライアントからトランザクション要求を受け取ると、まず自身が一貫性レベルを満たしているかを判断する。一貫性レベルを満たしていない場合は、未実行の更新結果をレプリケーターに要求することで一貫性レベルを保証する。すなわち、レプリカ主導で一貫性制御を行い、レプリケーターはレプリカからの要求に応じて受動的に動作する。クライアントからトランザクション要求を受けたレプリカは、一貫性レベルを満たしている場合はレプリケーターを介さずに処理を行うため、レプリケーターの処理量を軽減することができる。

以上のように、提案手法ではレプリカとレプリケーターで分担して一貫性制御を行う。以降、この一貫性制御時の動作を実現するための、レプリカおよびレプリケーター内における具体的な制御について述べる。

4.3 レプリカにおける制御

提案手法においてレプリカが行うのは、クライアントからのリクエスト処理とレプリケーターからのリクエスト処理および応答処理である。

4.3.1 クライアントからのリクエスト処理

レプリカはクライアントからリクエストを受け取ると、まず自身が一貫性レベルを満たすかどうか判断する。このために、McRep がレプリケーターで管理していたバージョン番号 (GVN, SVN, RVNhi) のうち、SVN と RVNhi の管理をレプリカが担当する。各バージョン番号は次のように管理する。

- GVN

GVN はレプリカ全体で最新のバージョン番号を示す必要がある。これをレプリカで管理するためには同期をとる必要があり、通信コストが大きくなってしまう。そのため、GVN に関してはレプリカで管理は行わず、McRep と同様にレプリケーターが管理を行う。

- 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とる必要がなく、各レプリカごとに独立して管理可能である。SVN は、表 2 に示すように、各トランザクションの応答時に更新され、それぞれ更新時には RVN および GVN が必要となる。RVN に関しては元々各レプリカごとに管理しているため、特に問題は発生しない。一方で、GVN に関してはレプリカ内で管理していないため、提案手法では各更新トランザクションにおいて、レプリケーターが更新結果をキューへ格納し、レプリカへ応答を返す際に GVN の値を付与するように拡張する。つまり、レプリカは自身が管理する SVN の値を、更新トランザクションの応答時に受信した GVN の値に更新することで管理する。そして、レプリカはトランザクション応答に SVN を付与することで、クライアントはこれを自身の SVN として記憶することができる。

- RVNhi

RVNhi は、各レプリカごとのバージョンの上限値を示すものであり、レプリカごとに独立して管理可能である。RVNhi は、表 2 に示すように、更新結果の伝搬時に更新されるが、GVN が必要となるため、このままでは管理できない。そのため、提案手法ではレプリカへの更新結果の伝搬時においても、レプリケーターが GVN の値を付与するように拡張する。

以上のようにレプリカで SVN と RVNhi を管理することで、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。Linearizability の場合はレプリカ単独での判断はできないため、レプリカはトランザクションを受信次第レプリケーターへ転送し、レプリケーターにおいて GVN と等しい RVNhi を持つレプリカを選択しトランザクションを転送する。GVN 以上のレプリカが存在しない場合は、任意のレプリカにキュー内の更新結果を伝搬しレプリカ

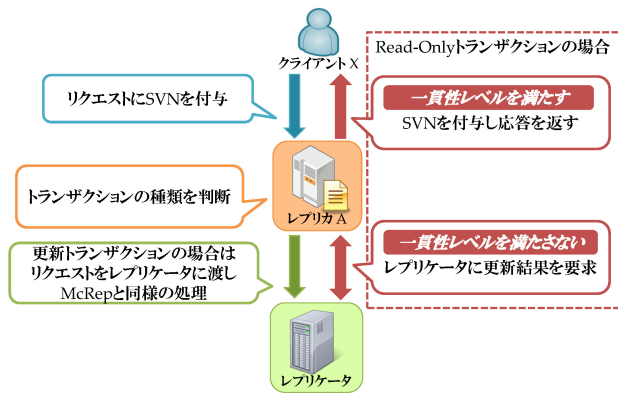


図4 レプリカにおけるクライアントからのリクエスト処理

Fig. 4 A process of the request from a client.

のバージョンを更新したうえで、そのレプリカにトランザクションを転送する。また、レプリカで一貫性制御を行う際、トランザクションの種類が更新トランザクションの場合は、たとえ一貫性レベルが判断できたとしても、必ずレプリケートによる競合解決およびトランザクションの更新結果のキューへの格納が必要になる。そのため、レプリケートを経由するリクエスト数の削減は期待できない。一方で、Read-only トランザクションの場合は、一貫性レベルを満たしていれば、レプリケートを介さずに処理することができる。まとめると、図4のように、クライアントはSVNを付与してレプリカにリクエストを送信し、レプリカでは受信したトランザクションの種類を判断する。Read-only トランザクションの場合は、自身のレプリカが一貫性レベルを満たしていれば、トランザクションを実行し即座にクライアントへ応答を返す。そうでなければ、レプリケートに未実行の更新結果を要求する。更新トランザクションの場合は、リクエストをそのままレプリケートへ渡し、McRepと同様の処理をする。

4.3.2 レプリケートからのメッセージ処理

レプリカはレプリケートから更新トランザクションリクエスト、更新トランザクションリクエストの応答、そして伝搬された更新結果を受信する。各メッセージは以下のよう

● 更新トランザクションリクエストの処理

レプリカはレプリケートから更新トランザクションのリクエストを受信すると、McRepと同様に処理し、レプリケートへ応答を返す。すなわち、Causal Consistencyの場合はトランザクション実行後、コミットを行い、RVNをインクリメントした後、トランザクション実行時に書き込まれたデータ項目であるwriteセットとRVNを応答に付与する。Causal Consistency以外の場合はコミットを行わず、トランザクション実行時に読み書きされたデータ項目であるread/writeセットとRVNを応答に付与する。

● 更新トランザクションリクエストの応答の処理

この場合も、受け取った応答にはGVNが付与されているため、自身が管理するSVNを受け取ったGVNに更新する。その後、リクエスト元クライアントへ応答を返す。

● 伝搬された更新結果の処理

この場合もMcRepと同様に、レプリケートから受け取った更新結果をコミットし、RVNをインクリメントした後、その値を付与してレプリケートへ応答を返す。ただし、前項で述べたように、この応答には更新結果に加えてGVNの値も付与されているため、更新結果をコミットする前に、自身が管理するRVN_{hi}をGVNの値に更新する。

4.4 レプリケートにおける制御

提案手法においてレプリケートは、レプリカから更新トランザクションのリクエストおよび更新結果の伝搬要求を受け取る。各リクエストは以下のように処理される。

● 更新トランザクションの場合

レプリケートはレプリカから更新トランザクションのリクエストを受信すると、McRepと同様に動作する。つまり、一貫性レベルを満たすレプリカを選択し、そのレプリカへリクエストを送信する。レプリカ選択時には、4.3.1項で述べたように、リクエストに付与されているSVNを用いる。そして、レプリカから応答を受け取り、競合解決を行い、トランザクションの更新結果をキューへ格納し、要求元レプリカへ応答を返す。また、応答を返す際にも4.3.1項で述べたように、レプリカ内でSVNを管理するためGVNの値を付与する。

● 更新結果の伝搬要求の場合

この場合は、RVN_{hi}およびRVN_{lo}を用いて、要求元レプリカにおいて実行されていない更新結果をすべて伝搬する。ただし、このときも、4.3.1項で述べたように、レプリカ内でRVN_{hi}を管理するため、更新結果に加えてGVNの値も付与する。そして各バージョン番号はMcRepと同様に更新される。つまり、表2に示すように、更新結果を伝搬した後、要求元レプリカのRVN_{hi}をGVNと同じエントリを指すように更新し、レプリカからの応答を受信後、RVN_{lo}をレプリカからの応答に含まれるRVNと同じバージョン番号を指すように更新する。

またこれらの処理に加え、レプリケートはMcRepと同様に、キュー内の更新結果を反映していないレプリカへ定期的に伝搬し、全レプリカへ伝搬済みとなった更新結果をキューから削除する。

4.5 動作フロー

以上で述べた提案手法の具体的な動作フローを図5に示

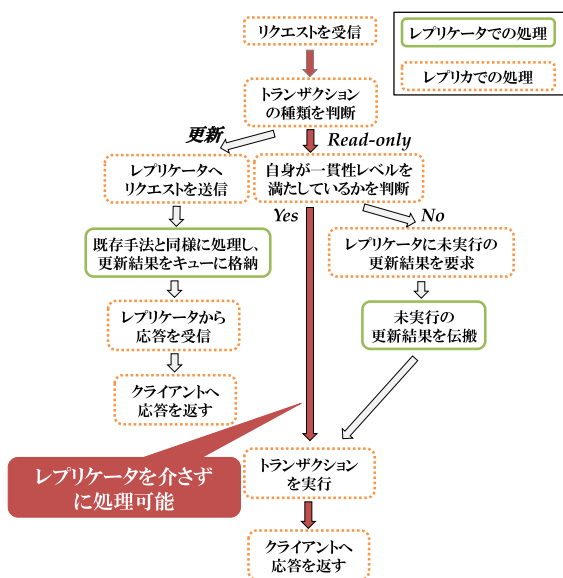


図 5 提案手法における動作フロー

Fig. 5 The processing flow of the proposal method.

す。提案手法では、クライアントからリクエストを受信したレプリカはまず、そのトランザクションの種類を判別し、更新トランザクションである場合は、レプリカでは一貫性制御は行わず、そのままレプリケータへリクエストを渡す。一方、Read-only トランザクションである場合は、レプリカで一貫性制御を行うため、まず要求されている一貫性レベルを満たしているかどうかを判断する。このとき、Linearizability の場合はつねに条件を満たさないと判断し、それ以外の一貫性レベルであれば SVN と RVNhi から判断する。そして、一貫性レベルを満たしている場合は、受け取ったトランザクション処理を実行し、即時にクライアントへ応答を返すことができる。一方で、一貫性レベルを満たしていない場合は、レプリケータに対して自身の RVN が ($RVN \geq GVN$) を満たすように更新結果の伝搬を要求する。レプリケータから受信した更新結果をコミットした後、受け取ったトランザクションを実行し、クライアントへ応答を返す。図 2 の動作フローと比較すると、更新トランザクションに関しては同等の処理を行うが、Read-only トランザクションに関しては、レプリカが一貫性レベルを満たしている限り、レプリケータを介することなく処理可能である点が大きく異なる。つまり、提案手法では McRep と比較して、Read-only トランザクション実行時におけるレプリケータとの通信回数が削減され、レプリケータの処理量を軽減することができる。

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 とバックエンドベースのレプリケーションである提案手法を比較し、提案手法ではレプリケータがボトルネックとなることを回避し、性能がスケールアウ

表 3 評価環境

Table 3 The specification of PostgreSQL, pgpool-II and client nodes.

	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 @ 3.2 GHz	
Memory	16 GB	
Network	1000BASE-T	

トすることを示す。実装において、レプリカとして PostgreSQL [13] を用い、レプリケータとして pgpool-II [14] を用いた。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。そのため、McRep およ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。したがって、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こととなる。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。

5.1 評価環境

評価環境を表 3 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 つ選択し読み出す 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。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 および 9 対 1 の場合において、クライアント数およ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 300 まで変化した際の評価結果を図 6 上部に示す。一貫性レベルが Sequential Consistency および Linearizability の場合は、SVN または QVN が更新されるたびに、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、レプリケータから更新結果が送信され、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

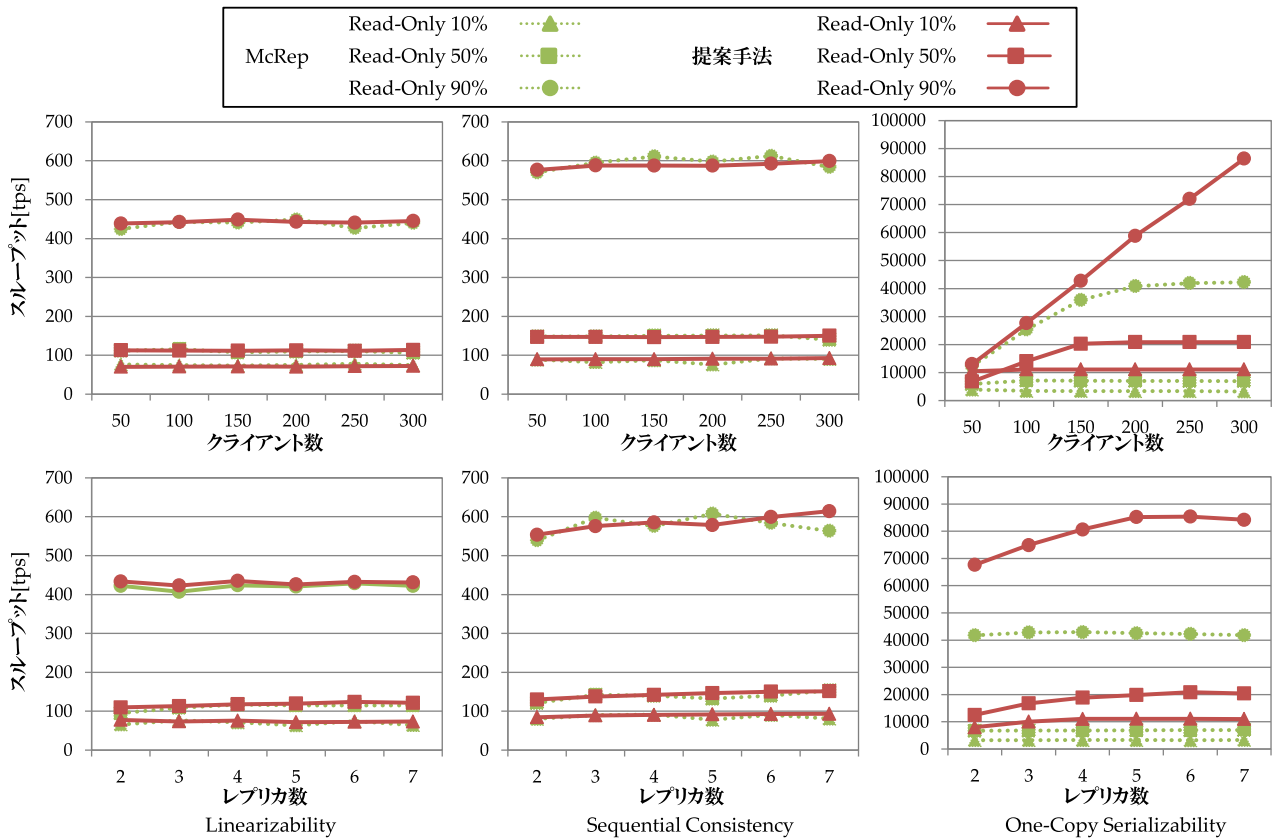


図 6 クライアント数およびレプリカ数に応じたスループットの推移

Fig. 6 Throughputs of different number of clients and replicas.

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ともないスループットが向上している。しかし、McRep では 200 クライアントで性能が頭打ちとなっている一方で、提案手法では 300 クライアントにおいても性能が向上している。さらに、最大スループットも提案手法では McRep の約 2 倍に向上している。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、pgpool-II がボトルネックとなっているのに対し、提案手法ではバックエンドベースで制御を行うことでボトルネックとなることを回避できたためである。

一貫性レベルが One-Copy Serializability かつ Read-heavy なワークロード (Read-only トランザクション 90%) でのレプリケータおよび各レプリカにおける CPU 使用率とネットワーク I/O バンド幅の平均値を図 7 に示す。他の条件はスループット測定時と同様である。この条件下において提案手法では更新操作のみがレプリケータを経由するため、レプリケータにおける CPU 使用率およびネットワーク I/O バンド幅は、McRep と比較し、それぞれ最大で約 1/4, 1/20 まで削減されている。一方、レプリカにおいては、クライアント数が増加すると CPU 使用率、ネットワークバンド幅ともに、提案手法のほうが McRep よりも増加している。これは、レプリケータがボトルネックと

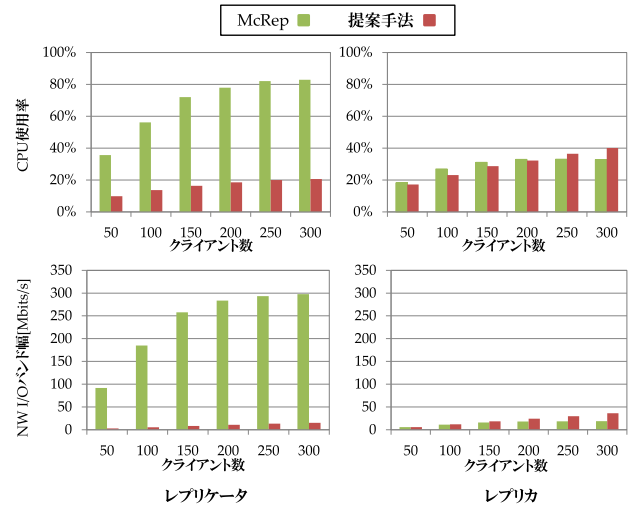


図 7 Read-heavy なワークロードにおける CPU 使用率およびネットワークバンド幅 (One-Copy Serializability)

Fig. 7 CPU usages and network bandwidths in a read-heavy workload (One-Copy Serializability).

なることを回避したことにより、レプリカにおけるトランザクションの処理数が増加したためである。

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 300 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 6 下部に示す。一貫性レベルが Sequential Consistency および Linearizability の場合は、レプリカ数の増加にともなう顕著なスループットの向上は見られない。これは 5.2 節と同様の理由による。一方、一貫性レベルが One-Copy Serializability の場合は、提案手法ではレプリカ数の増加にともないスループットの向上が見られるが、McRep では見られない。これは、バックエンドベースでレプリケーション制御を行うことで、レプリケータがボトルネックとなることを回避できているためだと考えられる。

以上から、クラウド上に展開された多くのアプリケーションにとって望ましい一貫性レベルである One-Copy Serializability [15] において、提案手法ではレプリケータがボトルネックとなることを回避し、Read-heavy なワークロードにおいてレプリカ数に応じて性能がスケールアウトすることが確認できる。

5.4 提案手法における遅延オーバーヘッドの評価

提案手法はクライアントが直接レプリカと通信するため、McRep と比較してレプリカを選択する自由度がない。McRep ではどのクライアントからのトランザクション要求もレプリケータを中継するため、一貫性レベルを満たすレプリカが 1 つでも存在すれば、即座にトランザクション要求を送信できる。一方、提案手法ではクライアントが通信するレプリカのバージョンが一貫性レベルを満たさない場合は、レプリケータを経由した後、一貫性レベルを満たす他のレプリカにトランザクション要求が送信され、その後、またレプリケータを経由し、元のレプリカに返される。この場合のトランザクション要求の通信経路は図 8 に示すように提案手法のほうが長くなる。

上記の場合において、McRep、提案手法での通信遅延の

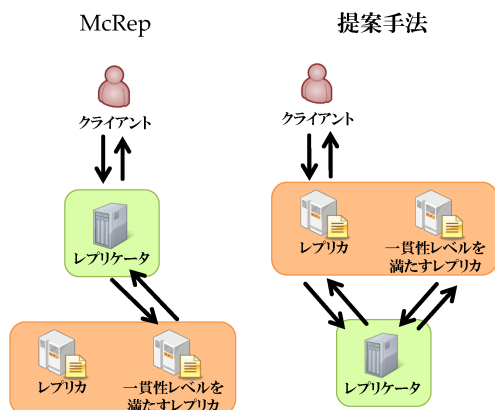


図 8 McRep と提案手法の間で通信経路に差が現れる状況

Fig. 8 A situation in which a difference appears in a communication path between McRep and proposed protocol.

オーバーヘッドを評価するために、図 8 に示す通信経路を通る Read-only/更新トランザクション要求を 100 個発行し、その遅延の平均を計測した。評価環境は 5.1 節と同様である。評価結果は、応答遅延の平均は、Read-only トランザクションでは、提案手法で 3.3 ミリ秒、McRep で 2.3 ミリ秒、更新トランザクションでは、提案手法で 3.7 ミリ秒、McRep で 2.8 ミリ秒となった。どちらの場合でも遅延の差はたかだか 1 ミリ秒程度であるため、その影響は小さい。さらに、レプリケータが更新結果を伝搬する間隔を調整することで、このような状況を減らすことが可能である。

6. 関連研究

開発コストの削減やアプリケーションの性能が保証する一貫性レベルにより受ける影響を最小化するため、複数の一貫性を制御可能なレプリケーションプロトコルが研究されている。PRACTI [16] はデータの一貫性レベルを柔軟に設定可能な部分レプリケーションシステムで、データに付与されたバージョン番号や更新時間を指標として一貫性を保証する。しかし、この手法は、McRep や提案手法のようにレプリケーション制御を行うサーバが存在しないため、一貫性制御のためにロックによる同期が必要となる。Garcia-Recuero ら [17] は利用者が、データ項目ごとにレプリケーションを作成するタイミングを指定することで利用者の要求に応じたデータの一貫性を保証する手法を提案している。ただし、この手法は、データの更新回数もしくは時間のみをレプリケーションを作成するタイミングの指標として用いるため、McRep や提案手法のように更新操作の時間的順序関係を考慮していない。

またミドルウェアを用いた手法として、AQuA [18] はレプリカ間で更新操作を全順序かもしくは FIFO 順序で順序付けするかを選択できる。Ganymed [4] は、ANSI SQL で定められた分離レベル [19] のうち 2 種類を保証できる。また、BaseCON [20] は Linearizability, Sequential Consistency および One-Copy Serializability の一貫性が保証可能な手法だが、McRep や提案手法とは違い、レプリカ間でバージョン番号の差異がある状態を許可しない。さらに、提案手法および McRep は、ミドルウェアを用いた上記の 4 つの手法よりも多くの一貫性レベルを保証することができる。

7. おわりに

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 を拡張し、ミドルウェアベースではなく、バックエンドベースで同様のレプリケーション制御を実現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レプリケータを経由する必要がある、レプリケータがシステム全体のボトルネックとなるという問題が存在した。これに対し、提案手法ではレプリケータがバックエンドで

制御を行うことで、全リクエストがレプリケータを経由する必要がなくなるうえ、レプリケータが集中的に行っていた一貫性制御を各レプリカにおいても行うように拡張することで、レプリケータにおける処理量を軽減することができる。評価から、提案手法は既存手法と比較して、レプリケータがボトルネックとなることを回避し、Read-heavyなワークロードにおいては性能がスケールアウトすることを確認した。

また、McRep, 提案手法とともに、レプリケータが単一障害点となるという問題があるため、今後の課題として、特定のサーバが単一障害点とならないレプリケーションモデルによる複数一貫性制御手法の実現があげられる。他にも、5.4 節で述べたような状況に対応するため、より現実的なベンチマークでの評価による、提案手法および既存手法における最適な更新結果の伝搬間隔の調査も必要である。

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

参考文献

- [1] Elnikety, S., Zwaenepoel, W. and Pedone, F.: Database Replication Using Generalized Snapshot Isolation, *Proc. IEEE 24th Symp. on Reliable Distributed Systems, SRDS '05*, pp.73–84 (2005).
- [2] Kemme, B. and Alonso, G.: Don't Be Lazy, Be Consistent: Postgres-R, A New Way to Implement Database Replication, *Proc. 26th Int'l Conf. on Very Large Data Bases, VLDB '00*, pp.134–143 (2000).
- [3] Krishnamurthy, S., Sanders, W.H. and Cukier, M.: An Adaptive Quality of Service Aware Middleware for Replicated Services, *IEEE Trans. Parallel Distrib. Syst.*, Vol.14, No.11, pp.1112–1125 (2003).
- [4] Plattner, C. and Alonso, G.: Ganymed: Scalable Replication for Transactional Web Applications, *Proc. 5th ACM/IFIP/USENIX Int'l Conf. on Middleware, Middleware '04*, pp.155–174 (2004).
- [5] Al-Ekram, R. and Holt, R.: Multi-consistency Data Replication, *Proc. IEEE 16th Int'l Conf. on Parallel and Distributed Systems (ICPADS 2010)*, pp.568–577 (2010).
- [6] Herlihy, M.P. and Wing, J.M.: Linearizability: A Correctness Condition for Concurrent Objects, *ACM Trans. Program. Lang. Syst.*, Vol.12, No.3, pp.463–492 (1990).
- [7] Riegel, T., Fetzer, C., Sturzhelm, H. and Felber, P.: From Causal to Z-linearizable Transactional Memory, *Proc. 26th Annual ACM Symposium on Principles of Distributed Computing, PODC '07*, pp.340–341 (2007).
- [8] Lamport, L.: How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs, *IEEE Trans. Comput.*, Vol.28, No.9, pp.690–691 (1979).
- [9] Bernstein, P. and Goodman, N.: A proof technique for concurrency control and recovery algorithms for replicated databases, *Distributed Computing*, Vol.2, No.1, pp.32–44 (1987).
- [10] Daudjee, K. and Salem, K.: Lazy Database Replication with Snapshot Isolation, *Proc. 32nd Int'l Conf. on Very Large Data Bases, VLDB '06*, pp.715–726 (2006).
- [11] Raynal, M., Thia-Kime, G. and Ahamad, M.: From serializable to causal transactions for collaborative applications, *Proc. 23rd Conf. EUROMICRO 97, New Frontiers of Information Technology*, pp.314–321 (1997).
- [12] Lin, Y., Kemme, B., Patiño Martínez, M. and Jiménez-Peris, R.: Middleware Based Data Replication Providing Snapshot Isolation, *Proc. 2005 ACM SIGMOD Int'l Conf. on Management of Data, SIGMOD '05*, pp.419–430 (2005).
- [13] Momjian, B.: *PostgreSQL: Introduction and concepts*, Vol.192, Addison-Wesley, New York (2001).
- [14] pgpool-II, available from (<http://www.pgpool.net/>).
- [15] Fetai, I. and Schultdt, H.: SO-ISR: Towards a Self-optimizing One-copy Serializability Protocol for Data Management in the Cloud, *Proc. 5th Int'l Workshop on Cloud Data Management, CloudDB '13*, pp.11–18 (2013).
- [16] Dahlin, M., Gao, L., Nayate, A., Venkataramana, A., Yalagandula, P. and Zheng, J.: PRACTI replication, *NSDI* (May 2006), Vol.12, p.80 (2006).
- [17] Garcia-Recuero, A., Esteves, S. and Veiga, L.: Quality-of-data for consistency levels in geo-replicated cloud data stores, *Proc. IEEE 5th Int'l Conf. on Cloud Computing Technology and Science (CloudCom)*, Vol.1, pp.164–170 (2013).
- [18] Krishnamurthy, S., Sanders, W.H. and Cukier, M.: An adaptive quality of service aware middleware for replicated services, *IEEE Trans. Parallel and Distributed Systems*, Vol.14, No.11, pp.1112–1125 (2003).
- [19] Berenson, H., Bernstein, P., Gray, J., Melton, J., O'Neil, E. and O'Neil, P.: A Critique of ANSI SQL Isolation Levels, *SIGMOD Rec.*, Vol.24, No.2, pp.1–10 (1995).
- [20] Zuikevičiūtė, V. and Pedone, F.: Correctness criteria for database replication: Theoretical and practical aspects, *On the Move to Meaningful Internet Systems: OTM 2008*, pp.639–656, Springer (2008).



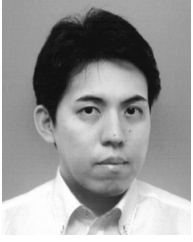
太田 篤 (学生会員)

1991 年生。2014 年名古屋工業大学工学部情報工学科卒業。現在、同大学大学院工学研究科創成シミュレーション工学専攻博士前期課程在籍。



松野 雅也

1990 年生。2014 年名古屋工業大学大学院工学研究科創成シミュレーション工学専攻博士前期課程修了。現在、日興システムソリューションズ。



川島 龍太 (正会員)

2007年岩手県立大学大学院ソフトウェア情報学研究科博士前期課程修了。2010年総合研究大学院大学複合科学研究科修了。博士(情報学)。同年株式会社ACCESS入社。2013年名古屋工業大学大学院工学研究科助教、現在に至る。主にSDN, システムソフトウェアに関する研究に従事。IEEE 会員。



松尾 啓志 (正会員)

1983年名古屋工業大学工学部情報工学科卒業。1985年同大学大学院修士課程修了。同年松下電器産業(株)入社。1989年名古屋工業大学大学院博士課程修了。同年名古屋工業大学工学部電気情報工学科助手。講師、助教授を経て、2003年同大学大学院教授、現在に至る。分散システム、画像認識、分散協調処理に関する研究に従事。工学博士。人工知能学会, IEEE 各会員。