Vol.2020-OS-150 No.10
2020/7/31

集約ログ転送法を用いた外部整合的トランザクション機構の評価

秋田佳紀[1]　小倉拓人[1]　宮澤勇貴[1]　川島英之[2]

概要：分散トランザクション処理を実行する際，外部整合性が求められる．これを実現するためにシングルマスタ方式ではマスタノードでトランザクション処理を行い，その結果をバックアップノードに複製する．この方式ではトランザクションを 1 つずつ実行するために性能が犠牲になる．この問題に対処する方法として集約ログ転送法がある．集約ログ転送法によりマスタノードにおいて，複数のクライアントから到着するトランザクションをバックアップノードへ一括転送することを可能にする．さらにクライアントにおいて複数のトランザクションをまとめて 1 つのトランザクションとしてマスタノード に送信し合意形成を行うことを可能にする．本研究では提案手法を分散合意手法 Raft と並行性制御法 S2PL を用いた Key-Value Store として設計・実装を行う．その上で，複製数を増やした場合や，遅延を与えた場合の性能特性について評価と分析を行う．

キーワード：分散トランザクション，Raft，S2PL

1. はじめに

1.1 動機

金融取引，中でも決済取引を扱うシステムにおいて，データベースには一貫性と信頼性が求められる．高い信頼性を保つために，金融の根幹を支える基幹系システムは，メインフレームを用いて構築されることが一般的である．メインフレームはレガシーシステムと呼ばれ，近い将来なくなり，Linux OS といったオープン系システムに移行すると言われている．しかし，未だその移行は進んでいない．その主たる理由は，基幹システムの安定稼働を支えているメインフレームに置き換えられるほどオープン系システムには信頼が置かれていないためである．一方，メインフレームには，維持・管理コストが高い，システム変更に要する時間がかかる，技術者が減っている，等の問題がある．サービスに急速な変化が求められ，それに対応せねばならない昨今において，メインフレームは足枷となる面が大きいため，オープン系システムを用いて高い信頼性と一貫性に関する要求を満たすことが望まれる．更に，電子決済の流行や IoT 機器の普及に伴って，モノ同士が通信を行い，モノの利用時間によって自動決済を行うような世界が来ることを我々は考えている．こうした世界では，少額大量決済といった社会需要が高まると考えられる．そのような需要に応えるには，より多くのトランザクションを処理するための高い性能が求められる．

そこで本研究では，オープン系システムであっても高い信頼性と一貫性を保ちつつ，高い性能を実現する手法について議論する．高信頼性を実現するために，複数のノードでデータの複製を保有する分散データベースという仕組みを用いることとする．そして分散データベースにおけるトランザクション一貫性レベルとしては外部整合的であることを前提とする．これは複製一貫性では最高度の一貫性の指標である線形化可能一貫性レベルと，トランザクション分離性では逐次化可能分離レベルを満たすことを表す．この前提において効率的な分散トランザクション処理を実現し，高い性能を実現することを目指す．

1.2 研究課題

分散データベースを実現する手法は，大別するとシングルマスタ方式とマルチマスタ方式に分けられる．シングルマスタ方式はクライアントからの要求を受け付けるノードが一つである一方,マルチマスタ方式はそれが複数となる．前者の例には Google Spanner [1]があり，後者の例には SLOG [4]，OceanVista [3] がある．マルチマスタ方式は複数のマスタを用いることで原理的に高性能化が可能だが，それに基づくプロダクトは実装が容易ではなく，外部整合性を保証するものは筆者が知る限り存在しない．シングルマスタ方式を用いたプロダクトの実装はマルチマスタ方式に比較すれば外部整合性を保証することは容易である．しかし，我々が必要とする下記の 3 条件を満たすシステムは筆者が知る限り存在しない．

① バックアップノードが日本全国に分散配置
② 処理性能が 100 万トランザクション毎秒（TPS）
③ 各トランザクションの応答時間(Round Trip Time (RTT))が 0.4 秒以内

本研究では，シングルマスタ方式を用いて外部整合性を保証した状態で，上記 3 つの条件を満たすことを目指す．そのために堀江等［20］によって提案された集約ログ転送法を拡張する方式を提案する．堀江等［20］の方式は，線形化可能一貫性レベルを保証するために分散合意アルゴリズム Raft [18]を用いており，逐次化可能分離レベルを保証

1 三菱 UFJ インフォメーションテクノロジー株式会社
2 慶應義塾大学

するために Strict Two Phase Locking (S2PL) [19]を用いている．また，堀江等［20］の方式はクライアントからまとめて送られた複数のトランザクションリクエストをマスタノードで集約し，それらを一括してバックアップノードに転送することで効率を上げている．

一方，堀江等［20］の論文には性能評価が表面的だという問題がある．例えば同論文ではクライアントがマスタに一括して転送するトランザクション数や，マスタノードでの集約数の変化に伴う性能変動が評価されていない．また，3.1 節で述べるように，彼らの実験システムはシステムを単純化する仮定を置いて実装されているため，一部の実験データにおいて理想的な値が示されている．

1.3 提案

本研究では，集約ログ転送法を詳細評価可能にすべく，同手法が実際の決済システムで採用されるトランザクションに近しい形でも成立するような精密な実装をおこなう．まず，ノード間の通信インターフェースやログ形式を刷新する．次に，通信パケットサイズに可変長を利用可能とする．これにより，(1) クライアントからマスタノードへ一括して送信するトランザクション数，(2) マスタノードからバックアップノードへ一括転送するログのサイズ，によってパケットサイズが適切に設定可能になる．

実装後，詳細な性能評価を実施する．第一に，上述の(1), (2)の調整による性能評価を行う．第二に，我々の想定する大規模地域災害下においても可用性を保つか調査するために，マスタノードとバックアップノードが遠隔地に存在するものと仮定してネットワーク遅延を加味した上での性能評価を行う．

1.4 論文構成

本論文の構成は以下の通りである．2 章では本論文で使う概念や用語の定義を説明する．3 章では，研究課題に対する提案を述べ，評価実験に用いたシステム，評価手法や評価項目について述べる．4 章では評価実験について述べる．実験環境について述べた後，3 章で述べた評価実験の結果を述べ，そして結果に対する考察を述べる．5 章では，高信頼性・一貫性を保ちつつ高性能を実現するための手法を扱う関連研究について述べる．最後に 6 章では集約ログ転送法の評価結果や今後の展望についてまとめる．

2. 準備

本研究では，シングルマスタ方式を採用して外部整合性を実現するシステム構成として，線形化可能一貫性レベルを保証するために Raft [18]を用い，逐次化可能分離レベルを保証するために S2PL [19]を用いる．また，効率的にトランザクションを処理する仕組みとして堀江等 [20]によって提案された集約ログ転送法を採用する．本章では，Raft, S2PL，そして集約ログ転送法について述べる．

2.1 Raft

Raft とは備えるべき性質である高信頼性と複製データ一貫性を備えた，シングルマスタ方式で採用される分散合意手法の一つである．Paxos [22] や Viewstamped Replication [23] など分散合意手法の多くはモデルが複雑であるが，Raft はモデルがシンプルで理解しやすいという特徴があり，多数の実装がある [24].

2.1.1 用語定義

Raft を説明する準備として用語の定義を行う．リーダとは，マスタとなるノードであり，フォロワとは，リーダから情報を受け取り，複製を保持することでバックアップとなるノードである．リーダは交代することが想定される．コマンドとは，クライアントから発行され，最終的にステートマシンに適用される一連の更新操作である．本研究においては，決済トランザクションがコマンドとなる．ログとは，コマンドの適用履歴である．ログに書き込まれたコマンドをエントリと呼ぶ．すなわちログはエントリの集合である．障害時の再起動に備えて，ログにはすべてのコマンドに関する適用履歴が書き込まれている．インデックスとは，ログ内の各エントリの添字である．古いエントリのインデックスの値は新しいエントリのインデックスの値以下である．タームとは，Raft の論理時間である．単調増加し，リーダ が選出開始された時から任期が終わるまでが一つのタームである．{インデックス，ターム}の組み合わせにより，エントリを一意に特定できる．

2.1.2 Raft の性質

Raft は以下の性質を保証する．

(1) 選出安全性：一つのタームでは，最大で一つのリーダが選出される．選出されるリーダ数は，0 はあっても 2 以上となることはない．
(2) リーダは追記のみ：リーダは自身のログに新しいエントリを追加するのみであり，過去のエントリの上書きや削除は行わない．
(3) ログマッチング：二つのログが同じ{インデックス，ターム}のペアを保持しているのであれば，それらのログにおいて，そのインデックスに至るまでのすべてのエントリは等しい．
(4) リーダ完全性：あるタームでエントリがコミットされたとすると，そのエントリは，そのタームより大きなタームのリーダのログに必ず含まれる．つまり一度コミットされたエントリは消えない．
(5) ステートマシン安全性：あるノードが，あるインデックスのエントリを，ステートマシンに適用したとする．別のノードでインデックスが同じだが内容が異なるエントリをステートマシンに適用することはない．Raft は全ノードのステートマシンの状態が等しくなることを保証する．

Raft はリーダ選出，ログレプリケーションという 2 つの

要素によって上記 5 つの性質を保証している．Raft のログレプリケーションをアルゴリズム 1 に示す．

1. リーダがクライアントからコマンドを受信.
2. リーダはログをロック.
3. リーダはログに 1. のコマンドをエントリとして追加.
4. リーダはログをアンロック.
5. リーダはエントリをフォロワにブロードキャスト.
6. 過半数以上のフォロワからエントリ永続化の RPC 応答があれば 7. へ移動.
7. リーダはクライアントに応答.

アルゴリズム 1. Raft のログレプリケーション

2.2 S2PL (Strict Two Phase Locking)

S2PL は逐次化可能分離レベルを保証するためのトランザクション制御手法である．データアクセス時にロックを順次とっていき，トランザクション内のすべての処理を完了して，全てのロックを解除してからコミットする．S2PL のアルゴリズムを Raft のアルゴリズムに統合するとアルゴリズム 2 となる．

1. リーダはクライアントからトランザクションを受信.
2. リーダはトランザクション中のデータアイテムをロック.
3. リーダはデータアイテムを更新.
4. リーダはログをロック.
5. リーダがログにエントリを追加.
6. リーダはログをアンロック.
7. リーダはエントリをフォロワにブロードキャスト.
8. 過半数以上のフォロワからエントリ永続化の RPC 応答があれば 9. へ移動.
9. リーダはデータアイテムをアンロック.
10. リーダはコミットをクライアントに通知.

アルゴリズム 2. Raft と S2PL の統合

アルゴリズム 2 の要点は，フォロワはこの処理の中でデータアイテムの更新を行っていない点である．ログが正しく伝搬されることを保証することで，リーダとフォロワのデータの一貫性は担保されるため，フォロワのデータアイテム更新はアルゴリズム 2 の中ではなく非同期的に別の処理で実施される．

2.3 集約ログ転送法

集約ログ転送法は，複数のトランザクションをまとめて処理することで，通信回数を削減し，効率的にトランザクションを処理する手法である．この手法はクライアントからリーダへのトランザクションのグループ（以後，**TxGrp** と表記）の転送と，リーダからフォロワへのログのグループ（以後，**LogGrp** と表記)の転送，という 2 つの要素から構成される．これをアルゴリズム 2 に適用した手法をアルゴリズム 3 と 4 に示す．

1. クライアントは複数のトランザクションから TxGrp を生成してリーダへ送付.
2. リーダにおいて，受信スレッドは TxGrp を受信し，それをワーカスレッドへ転送.
3. ワーカスレッドは TxGrp 内トランザクションからデータアイテムのリストを作成・整列.
4. ワーカスレッドはデータアイテムをロック.
5. ワーカスレッドはデータアイテムを操作.
6. ワーカスレッドはログをロック.
7. ワーカスレッドはログに TxGrp 内の全エントリを追加.
8. ワーカスレッドはログをアンロック.

アルゴリズム 3. ワーカスレッドの挙動

1. ログ転送スレッドは一定時間毎にログを確認し，追加されたエントリから LogGrp を構成してフォロワにブロードキャスト.
2. 過半数以上のフォロワから LogGrp をログに保存したとの RPC 応答があれば，3. へ移動.
3. ログ転送スレッドは LogGrp に該当するデータアイテムをアンロック.
4. リーダはコミットをクライアントに通知.

アルゴリズム 4. ログ転送スレッドの挙動

TxGrp に含まれるトランザクション数（以後 **TxGrpSize** と表記）をn, LogGrp に含まれる TxGrp 数（以後 **LogGrpSize** と表記）をmとすると，Append Entries RPC は $n \times m$ 件のトランザクションを一括してフォロワへ転送することとなる．これにより処理を効率化する．

3. 提案

堀江等［20］は性能評価が表面的だという問題がある．例えば同論文ではクライアントがリーダに一括して転送するトランザクション数や，リーダでの集約数の変化に伴う性能変動が評価されていない．彼らの実験システムはシス

テムを単純化するという仮定を置いて実装されているため,一部の実験データにおいて理想的な値が示されている.本章では集約ログ転送法による性能への効果を評価する手法を提案する.3.1 節では正しく集約ログ転送法を評価するためのプログラムの改善点について提案し,3.2 節以降で評価手法を提案する.

3.1 システムの再実装

堀江等 [20] による集約ログ転送法の実装では，ログ転送に用いるパケットが固定長であり，トランザクションの内容に関わらず一定の文字列が記録されるよう，不適切なエミュレートがされている．さらに，通信パケットサイズは固定であり，集約数の変更に合わせて変える事ができないなど,やはり不適切なエミュレートがされている.また,我々のシナリオにおいては実際の金融システムを想定して「送金元アカウント ID，送金先アカウント ID，送金額」からなる決済トランザクションを扱う必要があるが，そのようなトランザクションを取り扱うことができない．そのため以下の観点でシステムを再実装する.

3.1.1 クライアントとリーダ間のプロトコルの変更

堀江等 [20]はクライアントからリーダへ送信するパケットが固定長かつ長大な文字列であるため，パケットサイズが大きく，送受信負荷が高いという問題がある．本論文ではそのような負荷を軽減すべく，決済トランザクションを「送金元アカウント ID， 送金先アカウント ID，送金額」で表現し，この情報をバイナリ化してパケットサイズを縮小したものを単一のトランザクションとする.

3.1.2 ログ形式の変更

堀江等 [20]の研究では，初期プロトタイピング実装であった為に，エントリのフォーマットは固定文字列となっていた．これでは TxGrpSize を増やしてもエントリのサイズが増えることがない．実際には TxGrpSize の値によってエントリのサイズは大きくなるため，実際と比較して TxGrpSize を高めた際のデータ容量が小さくなり，実際よりも高い性能が示されてしまうという負の影響があると考えられる.そこで,今回は 3.1.1 節で述べた決済トランザクションを用いて TxGrpSize 分のエントリをログに記録するよう，1 つの TxGrp に格納されたトランザクションのバイナリ文字列をつなげたものを 1 つのエントリとしてログに記録する.

3.1.3 Append Entries RPC のパケットサイズを可変長化

堀江等 [20] では，Append Entries RPC のパケットサイズは固定長であり，TxGrpSize や LogGrpSize に関わらずある一定の領域を確保し通信を行っていた．TxGrpSize や LogGrpSize を増やすと通常 1 つの RPC に含められるデータサイズは大きくなるが，そのような仕様とはなっていなかったため，TxGrpSize や LogGrpSize を変えた場合の効果を正しく測定できなかった．そこで本研究では TxGrpSize や LogGrpSize に応じてパケットサイズを変えられるよう，パケットサイズを可変長とし，TxGrpSize や LogGrpSize に応じた領域を確保可能とする.

3.2 評価の前提

3.1 節の再実装を行ったうえで,性能測定を行い、評価する。本研究では，性能を単一時間当たりの処理トランザクション数（TPS）と，あるトランザクションの応答時間（トランザクションがクライアントから送信されてクライアントが応答を受け取るまでの時間（RTT））の 2 つの値によって表す．集約ログ転送法の最大効率を評価するために，クライアントから送信される各トランザクションは，異なるデータアイテムの変更を行うよう設定し，決済トランザクションの ID の重複はないものとする.また,クライアントは十分な台数を準備し，各条件に合わせて，最高性能が得られる台数を用いて評価を行う.

3.3 評価観点と評価手法

3 章冒頭で述べた堀江等 [20]の論文における課題について，本研究では次の 4 つの実験を行うことで集約ログ転送法を評価する．その観点と手法について説明する.

実験 1. LogGrp 転送の評価

実験 1 では，LogGrpSize の変化による性能変化を調べることで，最も効率的な LogGrpSize を調査する．そのために，LogGrpSize の上限をパラメータとし，この値を変更し繰り返し性能を測定する．LogGrpSize を大きくすることでリーダとフォロワの間の通信回数を削減できる．一方，単一通信に含められるエントリ数が増えパケットサイズが大きくなるため、LogGrpSize には適切な値が存在すると考えられる.

実験 2. TxGrp 転送の評価

実験 2 では，TxGrpSize の変化による性能変化を調べることで,最も効率的な TxGrpSize を調査する.そのために,TxGrpSize を変更して繰り返し性能を測定する．TxGrpSize を大きくすることで，クライアントとリーダの間の通信回数を削減できる一方で，TxGrp 当たりのデータサイズが大きくなるため，効率化には上限があり TxGrpSize には適切な値が存在すると考えられる．なお，実験 1 で用いた LogGrpSize との相関はないものとして,LogGrpSize は実験 1 で得られた最適な値を用いて本実験をおこなう.

実験 3.フォロワ数の変化による性能評価

実験 3 ではフォロワ数の増加と性能劣化の相関性を評価する．フォロワ数の増加に伴って，ログ転送スレッドは，フォロワの数だけ Append Entries RPC を発行する必要があるため，処理の負荷が高くなり性能が劣化することが予測される．そのためフォロワ数を変えて性能を繰り返し測定する．なお，TxGrpSize 及び LogGrpSize は実験 1 と実験 2 で得られた最適な値を用いる.

実験 4. 遠隔地分散を想定した遅延を発生させた性能評価

1 章で説明した通り，我々は下記を条件としている．

① バックアップノードが日本全国に分散配置
② 処理性能が 100 万トランザクション毎秒（TPS）
③ 各トランザクションの応答時間(Round Trip Time (RTT))が 0.4 秒以内

そのため，各フォロワの Append Entries RPC の送受信に待機時間を挿入することで，遠隔地にノードが存在するかのようにエミュレートし、遅延時間を発生させて性能測定を行う．本来リーダはランダムに選出されるが，本実験では必ず東京で起動する設定とし，フォロワを配置する都市を増やしていく．この実験では実験 3 の結果と比較することで，遅延による性能影響を評価する．TxGrpSize と LogGrpSize は実験 3 と同値を用いる．

4. 評価

4.1 実験環境

本研究では，Amazon Web Service(AWS)の Elastic Compute Cloud(EC2)を用いて実験を行った．AWS とは Amazon の提供するクラウドサービスであり，EC2 とはクラウド上に自由に OS インスタンスを構築することができる IaaS サービスである．実験に用いる EC2 の設定は表 1 の通りである．

表 1. 実験で用いた EC2 の設定

リージョン	Tokyo
Availability Zone	全インスタンス同一
インスタンスタイプ	m5.8Xlarge
OS	RedHat Enterprise Linux 8
VCPU	32Core
メモリ	128GiB
ストレージ	100GB

全てのインスタンスは同一データセンタに位置し，極力ネットワーク遅延の小さい環境としており，恣意的に遅延を発生させない限りはネットワーク遅延 0 と仮定している．実験で調整するパラメータは，TxGrpSize，LogGrpSize，フォロワ数，インスタンス間の遅延時間 の 4 つである．

4.2 実験結果

実験 1. LogGrp 転送の評価

本実験において，LogGrpSize 以外のパラメータは表 2 のように設定した．LogGrpSize と性能の関係は図 1，2 のようになった．図 1 から，LogGrpSize 10 までは LogGrpSize に比例して TPS が向上したが，それ以降は TPS に変化が見られなかった．図 2 から，LogGrpSize と RTT の相関は見られなかった．

表 2. 実験 1 で用いるパラメータ

フォロワ数	4
TxGrpSize	100
インスタンス間のネットワーク遅延(ms)	0

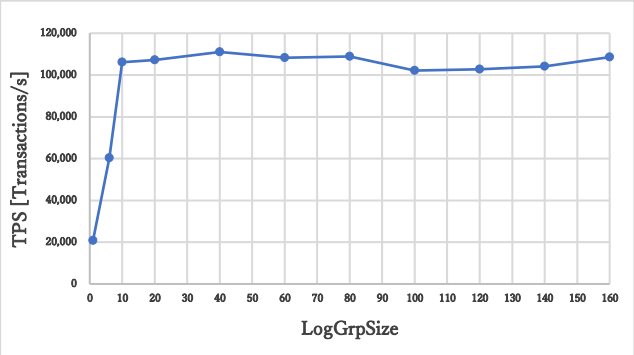

図 1. LogGrpSize と TPS の関係

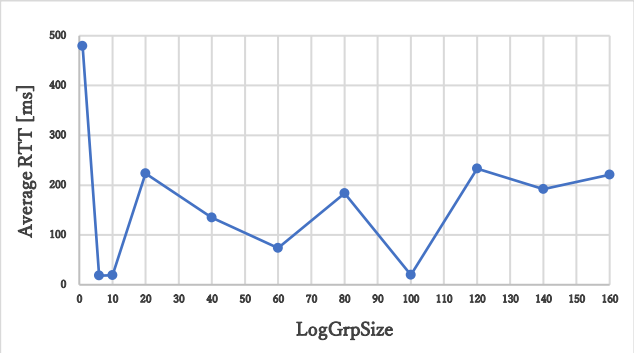

図 2. LogGrpSize と RTT の関係

実験 2. TxGrp 転送の評価

本実験において，TxGrpSize 以外のパラメータは表 3 のように設定した．TxGrpSize と性能の関係は図 3，4 のようになった．

表 3. 実験 2 で用いるパラメータ

フォロワ数	4
LogGrpSize	10
インスタンス間のネットワーク遅延(ms)	0

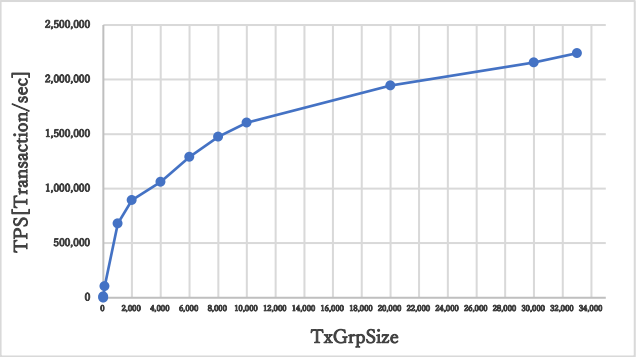

図 3. TxGrpSize と TPS の関係

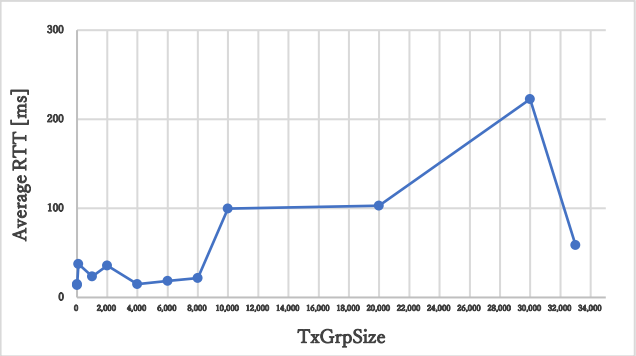

図 4. TxGrpSize と RTT の関係

図 3 より，TxGrpSize を上げることで TPS が向上することを確認した．実験では TPS がスケールする傾向が見られたが，最大 TxGrpSize を 33,000 とした．この理由は，これ以上の値を採用した場合 OS の仕様によりメモリ空間に保持できるオブジェクトのスタックサイズが上限を超えてしまいクラッシュしてしまったためである．

図 4 より，TxGrpSize をあげると RTT の値が大きくなり，性能が劣化する傾向が巨視的には見られたが，微視的には例外も存在した．RTT と TxGrpSize の関係を見るには追加実験が必要と考えられるが，現状高々200ms 程度であり目標値を満たすため，本研究では追加実験は実施しなかった．

実験 3.フォロワ数の変化による性能評価

本実験においてフォロワ数以外のパラメータを表 4 のように設定した．フォロワ数と性能の関係を図 5，6 に示す．

表 4. 実験 3 で用いるパラメータ

LogGrpSize	10
TxGrpSize	33,000
インスタンス間のネットワーク遅延(ms)	0

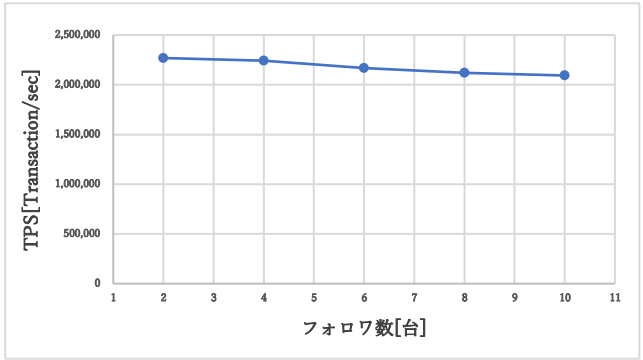

図 5.フォロワ数と TPS の関係

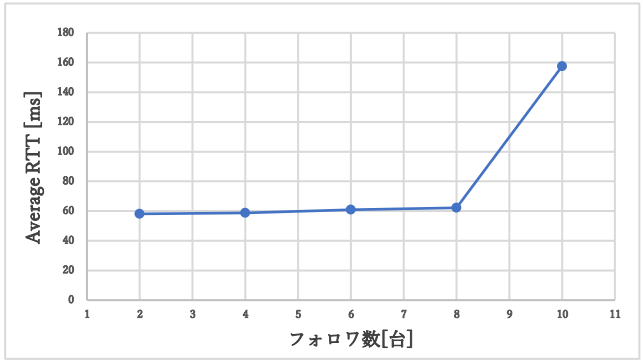

図 6. フォロワ数と RTT の関係

図 5 より，フォロワ数を増やすと微量ではあるが TPS の減少がみられた．図 6 より，RTT についてもノード数の増大によって性能が劣化した．なかでもフォロワ数を 10 に増やした際の影響が大きく，約 100 ms 遅くなるという結果が観察された．

実験 4. 遠隔地分散を想定した遅延による性能評価

本実験においてフォロワ数以外のパラメータを表 5 のように設定した．

表 5. 実験 4 で用いるパラメータ

LogGrpSize	10
TxGrpSize	33,000
インスタンス間のネットワーク遅延(ms)	表 6 参照

インスタンス間のネットワーク遅延については，リーダが必ず東京で起動すると仮定し，フォロワが日本国内の都市に分散した状態をエミュレートした．各都市の東京からのネットワーク遅延は，Broadband Acceleration Project [21]を参考にして設定した．ここに情報のない都市は距離から算出した．フォロワを増やす際には表 6 の上から順に都市を追加していき，実際にフォロワを順番に増やしていった際の性能の変化は図 7，8 のようになった．

表 6. 東京からのネットワーク遅延仮定

大阪	15 ms
福岡	20 ms
札幌	22 ms
名古屋	8 ms
広島	26 ms
仙台	12 ms
浜松	8 ms
沖縄	37 ms

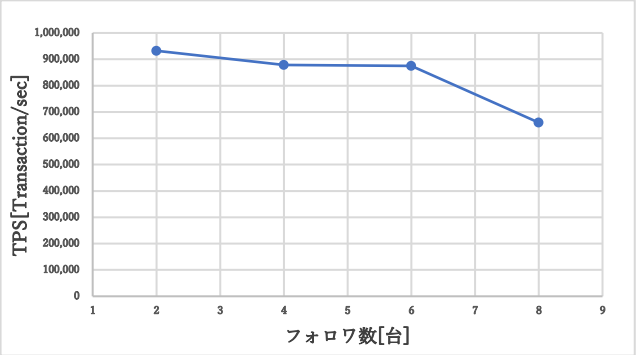

図 7. 遅延を想定したフォロワ数と TPS の関係

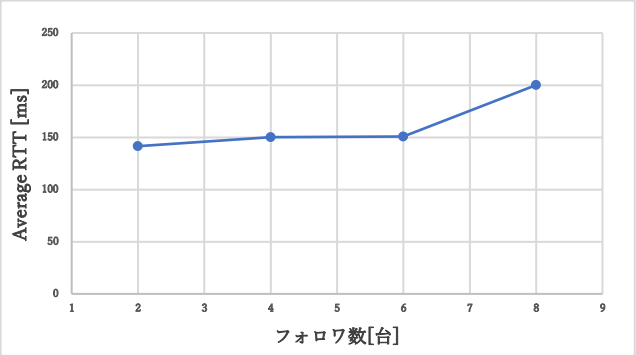

図 8. 遅延を想定したフォロワ数と RTT の関係

図 7 より，実験 3 と同様に，フォロワ数を増やすと TPS が減少する結果となった．遅延を発生させるとその減少の割合は大きくなった．遅延発生前は 200 万 TPS を超えていたが，遅延の発生によりフォロワ数が 2 の場合でも 90 万程度の TPS まで減少しており，遅延の影響は大きいものとなった．図 8 については実験 3 と同様フォロワ数を増やすと性能が劣化する結果となった．遅延時間は 8～40ms といった値だが，RTT は 100ms 前後遅くなる結果となっており，距離遅延の影響が大きいことが分かった．

4.3 考察

実験 1 と実験 2 の結果から，集約ログ転送法によって，TPS 向上の傾向が確かめられた．特に TxGrpSize は効率化への寄与が大きく，集約しない場合と最大まで集約した場合で，約 200 万の TPS 差があった．RTT については分散が大きく，相関が見られなかったが，クライアントからリーダへのトランザクション集約数を大きくすると，やや値が大きくなる傾向が見られた．ログやトランザクションのグループ化により書き込み処理や通信処理などの待ち時間が発生するため，これは合理的な結果といえる．実験 3 及び実験 4 の結果から，遅延のない環境下と比較して遅延を発生させた場合はフォロワー台数を増やした時の性能への影響度合いが大きいものとなった．これは Raft の性質上，最低でもフォロワの過半数から応答を待たなければならず，フォロワ数を増やすほど遅延時間の大きいノードが全体の処理を待たせる形となってしまうため，遅延の影響が大きくなったと考えられる．本研究で掲げていた 3 つの満たすべき条件については，ネットワーク遅延を想定しない環境下では TPS，RTT 共に目標値を達成したが，ネットワーク遅延を発生させた結果，TPS が目標値を下回る結果となった．ネットワーク遅延を想定した環境下でいかに性能を下げずに遅延を隠蔽するかは今後の課題である．また，遅延による性能劣化は免れないものであるため，さらなる性能の向上が必要である．

5. 関連研究

5.1 シングルマスタシステムに関する研究

Cloud Spanner [1] は本研究と同じくシングルマスタ方式を採用しており，地球規模でデータを地理的に分散管理し，外部整合性のある分散トランザクションを実現する．データの複製には Paxos ベースのプロトコルが用いられており，高可用性と水平スケーラビリティが特徴である．本研究は Spanner とは異なり，効率化に資するリーダ多重化とグループログ転送を実現している．

APUS [2] は Remote Direct Memory Access (RDMA)を用いた Paxos ベースの分散合意プロトコルである．APUS では複数クライアントからの命令をまとめて処理し，RDMA を用いて通信を行うことで高速化を図っている．本研究ではこの手法を拡張し，グループログ通信を実現している．

5.2 マルチマスタシステムに関する研究

OceanVista [3] はマルチマスタ 方式を採用したトランザクション処理プロトコルである．OceanVista ではデータベース間で下限水準を定期的に交換し合うことで非同期トランザクション処理を可能にしており，地理的に分散化したデータベースでも通信の遅延を隠蔽し，同時実行制御とデータの複製のコストを抑える．

SLOG [4] は Spanner と同等の高可用性と水平スケーラビリティを持ちながら，低レイテンシ書き込みを実現する．SLOG はアプリケーションの物理的な局所性を活用することで，逐次化可能分離レベルならびにトランザクション処理の低遅延化と高スループット化を実現している．

STAR [5] では，トランザクションがアクセスする領域が

単一か複数かにより，トランザクション処理の実行を変えることで通信コストを削減する．場合によっては分散合意を用いることなくトランザクション処理を行えるため，トランザクション処理が効率化される．

これらマルチマスタシステムは原理的には高性能化を達成可能なことが上記研究より導かれている．一方，それを実現したプロダクトは筆者が知る限り存在せず，更なる研究の進展が期待される分野である．

5.3 単一ノードにおけるトランザクション処理に関する研究

Silo [6] は楽観的並行実行制御(OCC) に基づいてトランザクション処理を行う手法である．Silo のコミットプロトコルにはロック，検証，書き込み，の 3 相がある．古典的 OCC が単一ロックを用いるのに対して，ロック対象を S2PL 同様に非集中化させることで高性能化を達成している． FOEDUS [7] や MOCC [8] 等の設計は Silo に影響を受けている．近年は多版並行性制御法に基づく Cicada [9], 時刻印順序法 (T/O) に基づく TicToc [10]， 逐次グラフ検定に基づく M-SGT [11] 等の並行性制御法が提案されている．ただし Silo のみ並行性制御法とリカバリ法を提案しており, 他はいずれも並行性制御法の提案にとどまっている．本研究の手法は S2PL であり，これらの近代的手法の導入により更なる高性能化を達成できる可能性がある．

トランザクション処理の構成要素には並行性制御法とリカバリ法がある. 従来のリカバリ法は Aether [12] のように，複数のワーカスレッドの生成するログをメモリに集約した後，ストレージへ一括転送していた．その設計は HDD のようにランダムアクセスが不利になるデバイスを前提にしており, SSD や NVRAM などの近代的デバイスにはそぐわない．近代的なリカバリ法は複数のワーカスレッドが個別にログをストレージに並列転送することで高性能化を達成する．それらの例には epoch を利用する SiloR [13], DRAM に一切ログを書かない passive group commit [14], そして DRAM を利用する P-WAL [15, 16, 17]がある．

6. 結論

本論文では集約ログ転送法を用いることで TPS が向上することを確認した．一方，同手法では遅延を発生させた際に 100 万 TPS という目標性能を達成できなかった．それゆえ, さらなる性能向上の工夫が必要である. そのためには，今回用いた Raft と S2PL の統合よりも処理効率の高い手法を用いて外部整合性を満たす手段が考えられる．また，現状のアルゴリズムではリーダのログアクセス前後でロックを取得する関係で，追加処理が逐次化されるため，並列性が失われる．エントリの追加処理を並列化できれば，それに伴う性能向上が期待されるだろう. 本研究においては，実験データとしてキー重複のないトランザクションを準備したが，実際の金融システムの利用状況では，義援金口座への集金や小売店での決済など、キー重複は十分に想定される．キー重複を含む際の性能劣化についても実験を通じて評価する必要があると考える．

7. 参考文献

[1] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson C. Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, Dale Woodford: Spanner: Google's Globally-Distributed Database. OSDI 2012: 251-264

[2] Cheng Wang, Jianyu Jiang, Xusheng Chen, Ning Yi, Heming Cui: APUS: fast and scalable paxos on RDMA. SoCC 2017: 94-107

[3] Hua Fan, Wojciech Golab:Ocean Vista: Gossip-Based Visibility Control for Speedy Geo-Distributed Transactions. Proc. VLDB Endow. 12(11): 1471-1484 (2019)

[4] Kun Ren, Dennis Li, Daniel J. Abadi: SLOG: Serializable, Low-latency, Geo-replicated Transactions. Proc. VLDB Endow. 12(11): 1747-1761 (2019)

[5] Yi Lu, Xiangyao Yu, Samuel Madden: STAR: Scaling Transactions through Asymmetric Replication. Proc. VLDB Endow. 12(11): 1316-1329 (2019)

[6] Stephen Tu, Wenting Zheng, Eddie Kohler, Barbara Liskov, Samuel Madden: Speedy transactions in multicore in-memory databases. SOSP 2013: 18-32

[7] Hideaki Kimura:
FOEDUS: OLTP Engine for a Thousand Cores and NVRAM. SIGMOD Conference 2015: 691-706

[8] Tianzheng Wang, Hideaki Kimura: Mostly-Optimistic Concurrency Control for Highly Contended Dynamic Workloads on a Thousand Cores. Proc. VLDB Endow. 10(2): 49-60 (2016)

[9] Hyeontaek Lim, Michael Kaminsky, David G. Andersen: Cicada: Dependably Fast Multi-Core In-Memory Transactions. SIGMOD Conference 2017: 21-35

[10] Xiangyao Yu, Andrew Pavlo, Daniel Sánchez, Srinivas Devadas: TicToc: Time Traveling Optimistic Concurrency Control. SIGMOD Conference 2016: 1629-1642

[11] Dominik Durner, Thomas Neumann: No False Negatives: Accepting All Useful Schedules in a Fast Serializable Many-Core System. ICDE 2019: 734-745

[12] Ryan Johnson, Ippokratis Pandis, Radu Stoica, Manos Athanassoulis, Anastasia Ailamaki: Aether: A Scalable Approach

to Logging. Proc. VLDB Endow. 3(1): 681-692 (2010)
[13] Wenting Zheng, Stephen Tu, Eddie Kohler, Barbara Liskov: Fast Databases with Fast Durability and Recovery Through Multicore Parallelism. OSDI 2014: 465-477
[14] Tianzheng Wang, Ryan Johnson: Scalable Logging through Emerging Non-Volatile Memory. Proc. VLDB Endow. 7(10): 865-876 (2014)
[15] 神谷 孝明, 川島 英之, 星野 喬, 建部 修見, 並列ログ先行書き込み手法 P-WAL, 情報処理学会論文誌データベース（TOD）, Vol. 10, No. 1, pp. 24-39.
[16] Yasuhiro Nakamura, Hideyuki Kawashima, Osamu Tatebe: Integration of TicToc Concurrency Control Protocol with Parallel Write Ahead Logging Protocol. Int. J. Netw. Comput. , Vol. 9, No. 2, pp. 339-353, 2019.
[17] Takayuki Tanabe, Hideyuki Kawashima, Osamu Tatebe: Integration of Parallel Write Ahead Logging and Cicada Concurrency Control Method, SMARTCOMP, pp. 291-296, 2018.
[18] Diego Ongaro, John K. Ousterhout: In Search of an Understandable Consensus Algorithm. USENIX Annual Technical Conference 2014: 305-319
[19] Weikum, Gerhard, and Gottfried Vossen. Transactional information systems: theory, algorithms, and the practice of concurrency control and recovery. Elsevier, 2001.
[20] 堀江 悠樹, 梶原 顕伍, 川島 英之, 建部 修見, ログ転送グループ化による外部整合的トランザクション処理の高性能化, 情報処理学会研究報告, 2020-OS-148.
[21] Broadband Acceleration Project https://bbzero.jp/background
[22] Leslie Lamport: The Part-Time Parliament. ACM Trans. Comput. Syst. 16(2): 133-169 (1998)
[23] Brian M. Oki, Barbara Liskov: Viewstamped Replication: A General Primary Copy. PODC 1988: 8-17
[24] Raft Consensus Algorithm and its Implementations, https://raft.github.io/#implementations.