Vol.2015-HPC-150 No.37
2015/8/6

並列ログ先行書き込みの評価

神谷 孝明[1,3,a)]　川島 英之[2,3]　建部 修見[2,3]

概要：CPU コア数の増加に伴い，従来の WAL プロトコルにおいて，ログ挿入時の競合の増加がトランザクション処理性能を劣化させる原因の一つとなっている．そこで，我々は ioDrive をストレージデバイスとする際にふさわしい WAL プロトコルとして P-WAL を提案した．I/O アクセスと排他制御処理による性能劣化の問題に対処するために，P-WAL はそれぞれのログライタが専用の領域にログを書き込む，並列ログ書き込み方式を採用している．LSN のアクセス性能を向上させるため，従来の CAS（Compare And Swap）によるアクセスではなく，fetch-and-add 命令を用いて性能向上を試みる．P-WAL の性能評価には，TPC-C ベンチマークの New-Order トランザクションとその他の特性の異なる 3 種類のトランザクションを用いる．UPDATE を 1 回行うトランザクションにおいて，P-WAL は 425,531（tps）の性能を達成し，従来方式の WAL に比べて 2.42 倍の性能向上を示した．また，TPC-C ベンチマークの New-Order トランザクションにおいては P-WAL は 17,391（tps）の性能を達成し，従来方式の WAL に比べて 1.18 倍の性能向上を示した．

1. はじめに

ニューヨーク証券取引所が提供する応答時間は 10 マイクロ秒以下 [15] であり，MasterCard に求められる処理性能は 4 万件/秒 [13] である．これらの処理は，処理の途中で障害が起こったとしても，それまでに完了された操作の情報は確実に保存され，システムのリスタート時に適切に障害回復処理を行う必要がある．このような不可分な一連の操作群はトランザクション [5] と呼ばれる．同時実行制御を行いながら複数のトランザクションを管理する機構はトランザクションマネージャと呼ばれる．トランザクション処理の高性能化は様々な研究で行われている [1][2][6][12]．トランザクションの実行中に障害が起こった場合は，トランザクションマネージャはデータベースの状態をトランザクション前の状態に戻さなければならない．トランザクションマネージャが保証しなければならないトランザクションの特性は，原子性（Atomicity），一貫性（Consistency），独立性（Isolation），永続性（Durability）であり，この中で障害復旧に関する特性は AD である．

AD 特性を保証するため，トランザクションマネージャはデータベース操作（例：SQL 問合せ，SQL 更新処理）に加えて特別な処理を行う必要がある．それはログ先行書き込み（WAL: Write Ahead Logging）[4] である．WAL とは，ストレージ中のデータを書き換える前にその更新内容をストレージにログとして記録する処理である．WAL はストレージアクセスを要するためにトランザクション処理の性能を劣化させる．現在の WAL アルゴリズムはストレージデバイスとして HDD を前提としている．そのため，現在の WAL は高い I/O アクセスコストを極小化することを目的関数としており，複数のトランザクションログをメモリ中でまとめてストレージデバイスに一括して書き込む方式を採用している．この方式は PostgreSQL[17] や Oracle[16] などのデータベースシステムで用いられているに留まらず，現在でも技術革新が行われている [7]．

高性能ストレージとして知られる ioDrive[3] は HDD とは I/O アクセスコストが異なるため，現在の WAL アルゴリズムがそのストレージデバイスの性能を極大化できるか明らかではない．そこで我々は ioDrive をストレージデバイスとする時にふさわしい WAL プロトコルとして P-WAL を提案した [22]．*P*-WAL はそれぞれのワーカースレッドに専用の WAL バッファを持たせることで，ロックを必要とせずに，ログレコードを WAL バッファに挿入できる．また，不揮発ストレージ上に WAL バッファと同じ数の WAL ファイルを作成し，各 WAL ファイルを WAL バッファと一対一で対応させる．これにより，ログレコードの

[1] 筑波大学大学院 システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba
[2] 筑波大学 システム情報系
Faculty of Engineering, Information and Systems, University of Tsukuba
[3] 国立研究開発法人 科学技術振興機構 CREST
JST CREST
[a)] kamiya@hpcs.cs.tsukuba.ac.jp

挿入処理と同様に，それぞれのワーカースレッドはロックを必要とせずに，WAL バッファの内容を WAL ファイルへ書き込むことができる． ioDrive における並列ランダムライトの高性能性を活用することで，P-WAL は高い性能を達成する．

我々は以前の研究 [22] では LSN アクセスを高速化するために CAS を利用していた．CAS の LSN への適用は [10] 等でも用いられるなど標準的ではあるが，あまり性能が高いとは考えられない．そこで本研究では fetch-and-add 命令を用いて性能向上を試みる．また，我々は以前の研究 [22] では自作ベンチマークを用いて評価を行った．自作ベンチマークは提案機構の性能を詳細に分析するには有益である一方，実用的観点からは標準的ベンチマークの利用が望ましい．そこで本研究では多くの論文が標準的に用いている TPC-C ベンチマークを利用して性能評価を行う．すなわち本研究の貢献は fetch-and-add の導入と TPC-C ベンチマークを用いた性能評価である．

本研究は ARIES[14] スキームに基づき，WAL バッファへのログ挿入処理の競合緩和と WAL バッファの WAL ファイルへの移送の並列化を行う．ARIES スキームに基づき WAL を高速化する既存研究には Aether[7]，Deuteronomy[11][10]，そして分散ロギング [20] がある．Aether や Deuteronomy は本研究とは異なり WAL バッファの WAL ファイルへの移送を並列化しない．分散ロギングは高いスケーラビリティを示すが，オペレーションの依存関係を調べるために，ログだけでなくページやトランザクションにも頻繁に番号を発行する必要があるため，競合が深刻でない場合に性能が劣化する．本研究は従来の WAL プロトコルと比較してもオーバーヘッドはほとんど存在しない．

一方，シーケンス番号を使わずに WAL を再設計する研究には Silo[19][21] と FOEDUS[8] がある．これらの研究はいずれも ARIES スキームを大幅に修正する必要があるため，本研究とは異なり，成果を既存システムへ導入することは困難だと考えられる．

本稿の構成は以下の通りである．2 節ではトランザクション処理で用いられる WAL について述べる．3 節では高性能ランダムライトを活用した高性能並列 WAL として P-WAL を提案し，そのプロトコルを述べる．4 節では自作ベンチマークと TPC-C ベンチマークの New-Order トランザクションによる P-WAL の性能評価結果を示す．5 節ではロギングにまつわるその他の関連研究を述べる．6 節では結論と今後の課題を述べる．

2. WAL

ログ先行書き込み（Write-Ahead Logging），通称 WAL とはシステム障害に備えてデータの更新前にログを書き込む手法である．トランザクションによって作成されたログはメモリ中の WAL バッファに溜められる．トランザクションのコミット処理で，それまでに溜められた WAL バッファ中のログとコミットログをまとめ，ストレージの WAL ファイルに一括で書き込む（**図 1**（[22] より引用））．トランザクションマネージャはコミットログの有無によってトランザクションが成功したか失敗したかを判断する．コミットされたトランザクションによる更新はデータベースに適用し，コミットされていないトランザクションによる更新を書き戻す．このようにして，WAL はトランザクションの ACID 特性の Atomicity と Durability を保証する．WAL はログを永続化するため，ストレージへの書き込みを必要とする．書き込み待ち時間や WAL バッファへログを挿入する際の排他制御処理にかかる時間が大きいと，WAL がトランザクション処理のボトルネックになる可能性がある．

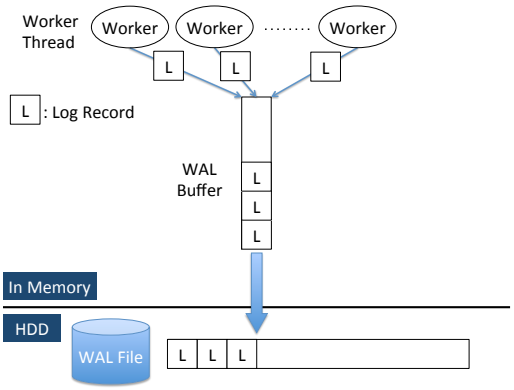

図 1 WAL のアーキテクチャ

Algorithm 1 log_insert_with_lock(log)

```
1: WALbuffer.lock() #WAL バッファをロック
2: LSN ← WALbuffer.insert(log)
3: if log.Type == 'COMMIT' then
4:     #コミットログが挿入された場合，カウントアップ
5:     WALbuffer.ncommit ← WALbuffer.ncommit +1
6:     if WALbuffer.ncommit == NGROUP or WALbuffer.full() then
7:         WALbuffer.flush() #WAL バッファの内容を WAL ファイルに書き込む
8:     end if
9: end if
10: WALbuffer.unlock() #WAL バッファをアンロックする
11: return LSN
```

2.1 ログレコードの挿入アルゴリズム

トランザクションのログレコードを WAL バッファに挿入する手続き log_insert_with_lock を Algorithm 1 に示す．まず，WAL バッファにロックをかける（1 行目），引数として渡されたログレコードを WAL バッファに挿入する（2 行目）．その際にはログレコードの ID となる LSN（Log Sequence Number）が決定される．挿入処理が終わ

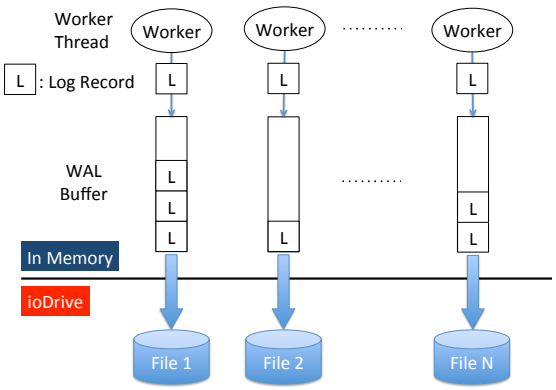

図 2 P-WAL のアーキテクチャ

ると WAL バッファをアンロックする（10 行目）．WAL バッファがグループコミットのグループ数に達したか，あるいは WAL バッファが一杯になったかをチェックして（6 行目），そうであれば WAL バッファのログレコード群を WAL ファイルに書き込む（7 行目）．ログレコードの WAL バッファへの挿入，及び WAL ファイルへの書き込みは，WAL バッファのロックを保持した状態で行われる．そのため，あるスレッドが WAL バッファへ挿入処理をしている間は，他のスレッドはログレコードを挿入することができない．特に WAL ファイルへの書き込みが発生すると，I/O 待ちのために遅延時間が大きくなる．

3. *P*-WAL：高性能ランダムライトを活用した高性能並列 WAL

3.1 *P*-WAL

ioDrive の並列ランダムライトの高性能性を活用すべく，ワーカースレッド毎に WAL バッファと WAL ファイルを割り当てる WAL プロトコル-P-WAL を提案する．**図 2**（[22] より引用）に P-WAL 方式の全体像を示す．図 1 と異なる点は，従来一つだけであった WAL バッファと WAL ファイルをワーカースレッドの数だけ分割した点である．

3.1.1 WAL バッファの分割

従来方式における WAL バッファは一つである．そのため，ログレコードを WAL バッファに挿入する際に衝突が頻発する．そこで衝突を緩和させるべく，ワーカースレッド毎に専用の WAL バッファを用意する．各 WAL バッファへの挿入は対応するワーカースレッドのみが行うので，それぞれのワーカースレッドはロックを必要とせずに，ログレコードを WAL バッファに挿入できる．

3.1.2 WAL ファイルの分割

ログレコードの書き込み先である WAL ファイルは WAL バッファと一対一で対応させる．そのため，P-WAL における WAL ファイルの分割数は WAL バッファの数と等しくなる．これにより，ログレコードの挿入処理と同様に，それぞれのワーカースレッドはロックを必要とせずに，WAL バッファの内容を WAL ファイルへ書き込むことができる．

Algorithm 2 log_insert(log,buffer_id)

```
1: #buffer_id で挿入する WAL バッファを指定
2: LSN ← WALbuffer[buffer_id].insert(log)
3: if log.Type == 'COMMIT' then
4:     #コミットログが挿入された場合，カウントアップ
5:     WALbuffer[buffer_id].ncommit ← WALbuffer[buffer_id].
       ncommit +1
6:     if WALbuffer[buffer_id].ncommit == NGROUP or WAL-
       buffer[buffer_id].full() then
7:         WALbuffer[buffer_id].flush() #WAL バッファの内容を
           WAL ファイルに書き込む
8:     end if
9: end if
10: return LSN
```

3.1.3 ログレコードの挿入アルゴリズム (log_insert)

P-WAL における，トランザクションのログレコードを WAL バッファに挿入する手続き log_insert を Algorithm 2 に示す．従来の log_insert_with_lock（Algorithm 1）と異なる点は，ロックを必要とせずにログレコードを並列に WAL バッファに挿入する点である．新たな引数 buffer_id で，ログの挿入先の WAL バッファの番号を指定する．buffer_id で指定された WAL バッファにログを挿入する（2 行目）．グループコミットのグループ数に達したか，あるいは WAL バッファが一杯になったかをチェックして（6 行目），もしそうであれば，WAL バッファのログレコード群を WAL ファイルに書き込む（7 行目）．

3.2 提案するリカバリプロトコル

P-WAL は WAL ファイルを分割する．これにより，従来のリカバリプロトコルが使用不能になる．ナイーブな解決策は全ファイル中の N 件のログレコードの整列だが，これには O（NlogN）の高いコストを要する．そこでマージ方式を用いる．

3.2.1 各ログの順序の決定

従来の WAL 方式では共通の WAL バッファにログレコードを挿入し，それをそのまま WAL ファイルにシーケンシャルに追記する．そのため，リカバリ時には WAL ファイルの先頭からログを読んでいけば，時系列順にログを処理できる．一方，P-WAL 方式では，ワーカースレッドの数だけ WAL ファイルが生成される．各 WAL ファイル内のログレコードの順序関係は，末尾に近い方が新しい一方，複数の WAL ファイル間でのログレコードの順序関係は不明である．そこで LSN（Log Sequence Number）により，WAL ファイル間でのログの順序関係を解決する．LSN とは単調増加するログの ID[5] である．

リカバリの際は，LSN の値が小さい順にログを処理する．従来方式の WAL では，この LSN の値が WAL ファ

表 1　実験環境 (1)

CPU	Intel（R） Xeon（R） CPU E5-2665 × 2
コア数	8 × 2
メモリ	64GB
ioDrive	SLC, 160GB, VRG5T VSL v3.3.3, Low-Level Formatting

イル上の位置を示す役割も兼ねる．P-WAL で発行される LSN はログのファイル上の位置を示すものではないため，ワーカースレッドは格納先の WAL ファイルの番号とファイル中のオフセットを LSN の付属情報としてログに記録する．

3.2.2　fetch-and-add による LSN アクセスの高性能化

LSN を取得する際に，共有変数 global_LSN にアクセスする．この global_LSN へアクセスのため，スレッド間で衝突が発生する．global_LSN への最も単純なアクセス方法は，ロックの利用だが，この方法だとロックの競合による性能劣化が大きい．そこで，以前の研究 [22] はロックを使わずに CAS（Compare And Swap）を使用していた．CAS の LSN への適用は [10] 等でも用いられるなど標準的ではあるが，あまり性能が高いとは考えられない．そこで本研究では，更なる性能向上を目指すために fetch-and-add 命令を使用する．

4.　評価実験

実験環境を**表 1** に示す．

4.1　自作ベンチマーク

4.1.1　実験内容

この実験では，UPDATE の回数や比率が異なる 3 種類のトランザクション (UPDATE-1, UPDATE-5&READ-5, UPDATE-10）を用いて，ワークロード毎に従来 WAL（従来方式の WAL）と P-WAL の性能を比較する．一つのログレコードのサイズは固定長の 512（Bytes）である．グループコミットによって，16 件のトランザクションのログを一括してストレージに書き込む．各オペレーションは，メモリ上にある総数 65536 のページの中からランダムに一つのページを選択して操作を適用する．各ページには int 型オブジェクト一つが含まれており，オペレーションに応じて READ（読み込み）や UPDATE（更新）を行う．READ は読み込みロック，UPDATE は書き込みロックを用いてページアクセスの排他制御を行う．

UPDATE-1 は UPDATE を 1 回（[BEGIN,UPDATE, END] の 3 個のログレコードを生成），UPDATE-5&READ-5 は UPDATE を 5 回，READ を 5 回（[BEGIN,UPDATE×5,END] の 7 個のログレコードを生成），UPDATE-10 は UPDATE を 10 回行う（[BEGIN,UPDATE×10,END] の 12 個のログレコードを生成）．

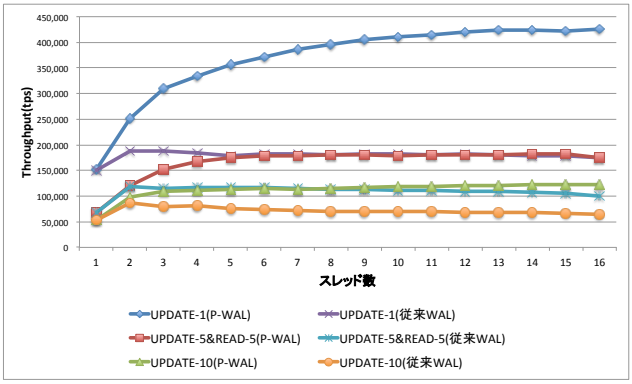

図 3　3 種類のトランザクションの性能評価

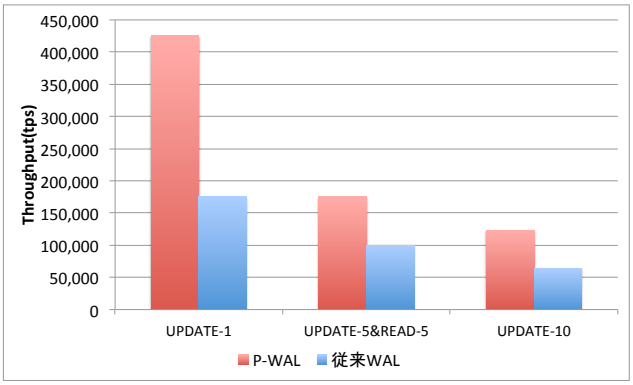

図 4　16 スレッド時の 3 種類のトランザクションの比較

4.1.2　実験結果

P-WAL と従来 WAL でトランザクションの種類とスレッド数を変えた時の実験結果を**図 3** に示す．スレッド数 16 の時の結果について注目して，トランザクションの種類毎に P-WAL と従来 WAL を比較したものを**図 4** に示す．

スレッド数が 16 の時（図 4），UPDATE-1 トランザクションで P-WAL は 425,531（tps）を達成し，従来 WAL の 175,131（tps）の 2.42 倍の性能向上を示した．また，UPDATE-5&READ-5 トランザクションにおいて P-WAL は従来 WAL の 1.75 倍の性能向上を示し，UPDATE-10 のトランザクションにおいて P-WAL は従来 WAL の 1.91 倍の性能向上を示した．

4.2　TPC-C ベンチマーク

4.2.1　実験内容

この実験では，TPC-C ベンチマークの New-Order トランザクションを用いて，従来 WAL（従来方式の WAL）と P-WAL の性能を比較，評価する．New-Order トランザクションは，商品の注文処理を模擬している．TPC-C で使われるテーブルのページをログに含めるために，一つのログレコードのサイズは先ほどの 4.1 節の実験の 2 倍の 1024（Bytes）になっている．グループコミットによって，16 件のトランザクションのログを一括してストレージに書き

Algorithm 3 New-Order トランザクション

```
 1: BEGIN;
 2: SELECT FROM Warehouse;
 3: SELECT FROM District;
 4: UPDATE District;
 5: INSERT INTO Order;
 6: INSERT INTO NewOrder;
 7:
 8: LOOP
 9: SELECT FROM Item;
10: SELECT FROM Stock;
11: UPDATE Stock;
12: INSERT INTO OrderLine;
13: END LOOP
14:
15: COMMIT;
```

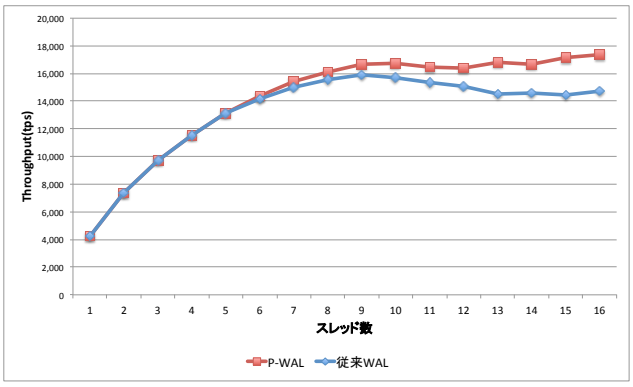

図 5 TPC-C ベンチマーク（New-Order）の性能評価

込む．一回の注文で購入する商品の種類の数（ol_cnt）は，ベンチマークアプリケーションによってランダムに生成される．また，1%の割合で rollback トランザクションが含まれる．通常のトランザクション内のオペレーションの内訳は，SELECT が 2+2*ol_cnt 回，UPDATE が 1+ol_cnt 回，INSERT が 2+ol_cnt 回発生する．この New-Order を SQL 文で示すと，Alogorithm 3 のようになる．

4.2.2 実験結果

実験結果を**図 5** に示す．スレッド数が 1〜6 の時, P-WAL と従来 WAL の性能差はほとんどない．これは SELECT や INSERT などの処理が大半を占めており，ログの挿入時の競合があまり深刻でないと考えられる．しかし，スレッド数をさらに増やしていくと，従来 WAL ではログ挿入時の競合の増加に伴い，性能が徐々に劣化している．一方，P-WAL では性能は僅かながら向上している．スレッド数が 16 の時，P-WAL のスループットは 17,391（tps）で，従来 WAL のスループット 14,705（tps） の 1.18 倍の性能向上を示した．

4.1 節の実験では，単一ページへの読み書きオペレーションのみであったため，同一ページへのアクセスと，LSN アクセス以外の競合は存在しなかった．一方，New-Order に存在する INSERT オペレーションは並列に実行できないため，トランザクション全体のボトルネックとなっており，P-WAL の性能向上率が低下したものと考えられる．

5. 関連研究

WAL を高性能化する研究は 2 種類に分類される．ARIES スキームに基づきシーケンス番号を使いながら高性能化を図る研究と，シーケンス番号を使わずに WAL を再設計する研究である．

ARIES スキームに基づく研究には Aether[7] , Deuteronomy[11][10]，そして分散ロギング [20] がある．Aether はロック解放タイミングの早期化，グループコミット待ち処理の非同期化，そして WAL バッファへのログ挿入処理の競合緩和を行う．Deuteronomy はトランザクションコンポーネントとデータコンポーネントを分離して非モノリシックな設計を用いる．この柔軟な設計によりトランザクショナル KVS，アトミック KVS，ページストレージエンジン等の様々なシステムが容易に構成される．これらの既存研究と本研究との違いは並列 I/O の活用にある．既存研究は本研究とは異なり WAL バッファの WAL ファイルへの移送を並列化しない．本研究では各ワーカースレッドに WAL バッファと WAL ファイルを専用的に持たせる設計により，ログ書込み処理の並列性を高めている．また，分散ロギングは，LSN だけでなくグローバルシーケンス番号（GSN）という論理クロックを用いて，依存関係のあるオペレーションが発生する場合にのみ同期を取ることで，スケーラビリティの高い分散ロギングを実現する．しかし，依存関係を調べるために，ログだけでなくページやトランザクションにも頻繁に GSN を発行する必要があるため，競合が深刻でない場合に性能が劣化する．一方，本研究は従来の WAL プロトコルと比較しても，各 WAL ファイル上の位置計算以外のオーバーヘッドは存在しない．ARIES の WAL スキーム [14] に基づいた研究ではいずれもシーケンス番号を利用する．多くの既存システムは ARIES を採用しているため，これらの成果を既存システムへ導入することは本研究成果同様に容易だと考えられる．

WAL を再設計する研究には Silo[19][21] と FOEDUS[8] がある．Silo と FOEDUS はシーケンス番号を使わない．その代わりに時区間（epoch）と楽観的実行制御 [9] を組み合わせる．この方式は同一時区間におけるログの順序を非決定とする．この問題はページアクセス衝突に伴うアボートにより防がれている．ただし，それにより平均遅延が悪化する可能性がある．これらの研究はいずれも ARIES スキームを大幅に修正する必要がある．従って本研究成果とは異なり，これらの成果を既存システムへ導入することは困難だと考えられる．

6. 結論と今後の課題

本研究では，ioDrive の高並列度におけるランダムライト

の高性能性を活用するべく，ログ先行書き込みを並列化した方式の P-WAL において，fetch-and-add を用いた LSN アクセスの高性能化，及び，自作ベンチマークと TPC-C ベンチマークによる評価を行った．UPDATE を 1 回行うトランザクションにおいて，P-WAL は 425,531（tps）の性能を達成し，従来方式の WAL に比べて 2.42 倍の性能向上を示した．TPC-C ベンチマークの New-Order トランザクションにおいては P-WAL は 17,391（tps）の性能を達成し，従来方式の WAL の 1.18 倍の性能向上を示した．

今後の課題は，よりスケーラビリティを向上させるための LSN アクセスの効率化，S2PL よりも効率的な並行実行制御プロトコルの導入，現実のアプリケーション上での P-WAL の実装と評価である．

謝辞 本研究の一部は，JST CREST「ポストペタスケールデータインテンシブサイエンスのためのシステムソフトウェア」，JST CREST「EBD：次世代の年ヨッタバイト処理に向けたエクストリームビッグデータの基盤技術」，JST CREST「広域撮像探査観測のビッグデータ分析による統計計算宇宙物理学」，科研費「#25280043HA」による．

参考文献

[1] Chatzistergiou, A., Cintra, M. and Viglas, S. D.: REWIND: Recovery Write-Ahead System for In-Memory Non-Volatile Data-Structures, PVLDB, Vol. 8, No. 5, pp. 497–508 (2015).
[2] Coburn, J., Bunker, T., Schwarz, M., Gupta, R. and Swanson, S.: From ARIES to MARS: transaction support for next-generation, solid-state drives, SOSP, pp. 197–212 (2013).
[3] Fusion-io: Application Acceleration Enterprise Flash Memory Platform — Fusion-io:, http://www.fusionio.com/.（アクセス日: 2015-04-15）.
[4] Gawlick, D., Gray, J., Iimura, W. and Obermarck, R.: Method and apparatus for logging journal data using a log write ahead data set (1985). US Patent 4,507,751.
[5] Gray, J.: The Transaction Concept: Virtues and Limitations, VLDB, pp. 144–154 (1981).
[6] Huang, J., Schwan, K. and Qureshi, M. K.: NVRAM-aware Logging in Transaction Systems, PVLDB, Vol. 8, No. 4, pp. 389–400 (2014).
[7] Johnson, R., Pandis, I., Stoica, R., Athanassoulis, M. and Ailamaki, A.: Aether: A Scalable Approach to Logging, PVLDB, Vol. 3, No. 1, pp. 681–692 (2010).
[8] Kimura, H.: FOEDUS: OLTP Engine for a Thousand Cores and NVRAM, SIGMOD Conference (2015).
[9] Kung, H. T. and Robinson, J. T.: On Optimistic Methods for Concurrency Control, ACM Trans. Database Syst., Vol. 6, No. 2, pp. 213–226 (1981).
[10] Levandoski, J., Lomet, D., Sengupta, S., Stutsman, R. and Wang, R.: High Performance Transactions in Deuteronomy, CIDR (2015).
[11] Levandoski, J. J., Lomet, D., Mokbel, M. F. and Zhao, K.: Deuteronomy: Transaction Support for Cloud Data, CIDR, pp. 123–133 (2011).
[12] Malviya, N., Weisberg, A., Madden, S. and Stonebraker, M.: Rethinking main memory OLTP recovery, ICDE, pp. 604–615 (2014).
[13] MasterCard: Processing: Brilliance Behind the Scenes of Commerce — MasterCard, http://www.mastercard.com/us/company/en/whatwedo/processing_brilliance_behind_commerce.html.（アクセス日: 2015-04-15）.
[14] Mohan, C., Haderle, D. J., Lindsay, B. G., Pirahesh, H. and Schwarz, P. M.: ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging, ACM Trans. Database Syst, Vol. 17, No. 1, pp. 94–162 (1992).
[15] NYSE: NYSE, New York Stock Exchange > About Us > News & Events > News Releases > Press Release 06-03-2009:, http://www1.nyse.com/press/1244024115279.html.（アクセス日: 2015-04-15）.
[16] Oracle: Oracle — Hardware and Software, Engineered to Work Together:, http://www.oracle.com/index.html.（アクセス日: 2015-05-01）.
[17] PostgreSQL: PostgreSQL: The world's most advanced open source database:, http://www.postgresql.org/.（アクセス日: 2015-05-01）.
[18] Ramakrishnan, R. and Gehrke, J.: Database Management Systems, McGraw Hill Higher Education (2002).
[19] Tu, S., Zheng, W., Kohler, E., Liskov, B. and Madden, S.: Speedy transactions in multicore in-memory databases, SOSP, pp. 18–32 (2013).
[20] Wang, T. and Johnson, R.: Scalable Logging through Emerging Non-Volatile Memory, PVLDB, Vol. 7, No. 10, pp. 865–876 (2014).
[21] Zheng, W., Tu, S., Kohler, E. and Liskov, B.: Fast Databases with Fast Durability and Recovery Through Multicore Parallelism, OSDI, pp. 465–477 (2014).
[22] 神谷 孝明, 川島 英之, 建部 修見: P-WAL: 並列ログ先行書き込みの提案, 研究報告システムソフトウェアとオペレーティング・システム（OS）Vol. 2015-OS-133, No. 18, pp. 1–10 (2015).