並行性制御法 Cicada の評価

田辺　敬之†　　川島　英之††　　建部　修見††

† 筑波大学 情報学群情報科学類　〒 305–8571 茨城県つくば市天王台 1 丁目 1–1
†† 筑波大学 計算科学研究センター　〒 305–8571 茨城県つくば市天王台 1 丁目 1–1
E-mail: †tanabe@hpcs.cs.tsukuba.ac.jp, ††{kawashima,tatebe}@cs.tsukuba.ac.jp

あらまし　国際会議 ACM SIGMOD において昨年，並行性制御法 Cicada が提案された．発表論文において Cicada は，並行性制御については高性能であることが示されている．本論文では，それが報告と一貫した性能を示すのか評価を行なった．また，Cicada の論文では障害対策について方針は示されているものの詳細は論じられておらず，性能評価もロギングを無効にしたものである．そこで Cicada に並列ログ先行書込み法 P-WAL を実装し，性能を評価する．

キーワード　トランザクション，並行性制御，データベースシステム，ロギング

1. 序　　論

トランザクション処理は至る所で用いられている．経済活動の基盤であるクレジットカード処理においてトランザクション処理は必須であるし，コンピュータで何かデータを保存する時，オペレーティングシステムの深部ではトランザクション処理が行われている．人類が処理するトランザクションは年々増加の一途を辿っており，それを効率よく処理することが要求される．

このトランザクション処理 [1] [2] の中核にあるのが並行性制御である．並行性制御 [3] とは複数のトランザクションがデータベースに同時並行的にアクセスしても，トランザクションの挙動を制御することで隔離性を保証する手法のことである．

この並行性制御に関して，我々は SIGMOD'17 にて提案された Cicada [4] に注目する．Cicada は本論文執筆時点において最も優れていると考えられる並行性制御法である．本論文は Cicada を再実装し，それが報告と一貫した性能を示すのか評価を行なった．また，発表論文において障害対策については方針は示されているものの詳細は論じられていない．そこで並列ログ先行書込み法 P-WAL を Cicada に実装し，その性能を評価した．

本論文の構成は以下の通りである．2 章では Cicada について述べる．3 章ではログ先行書込みについて述べる．4 章では Cicada の再実装について述べる．5 章では評価方法と結果を述べる．6 章では結論を述べる．

2. Cicada

SIGMOD'17 において並行性制御法 Cicada が提案された．Cicada で用いられる技術は三つに大別できる．多版法 [5]，楽観法 [6]，分散タイムスタンプ法 [7] である．多版法とはデータ更新の度に新しい版を作成することで，読込みと書込みの衝突を減らす技法を指す．楽観法とは，トランザクションが衝突を起こさないことを期待した上で効率を重視した技法を指す．この技法は，読込みは制限されない代わりに書込みに強く制限をかける技法である．分散タイムスタンプ法とは版に付与するタイムスタンプをある一つのスレッドが集中的に発行するのではなく，複数のスレッドが分散的に発行する技法を指す．

2.1 分散タイムスタンプ法

Cicada はトランザクションの開始時に各ワーカースレッドへタイムスタンプを割り当てる．

タイムスタンプは以下の三つの要素で構成される．

- Current local clock
- Clock boost
- Thread ID

各ワーカースレッドは 64 bit のローカルクロックを保持しており，その現在の値が current local clock である．Clock boost は直前のトランザクションがコミットされた時は 0, アボートされた時は 1 マイクロ秒に相当するクロック数となる．次のローカルクロックを仮に adjusted clock [4] とした時，adjusted clock は current local clock，最後にローカルクロックがインクリメントされてからの経過時間，clock boost の三つを足し合わせたものとなる．最後にローカルクロックがインクリメントされてからの経過時間は Time Stamp Counter [8] を利用して測定する．

タイムスタンプは 64 bit で表現され，上位 56 bit はローカルクロックの下位 56 bit，タイムスタンプの下位 8 bit は thread ID として構成される．

タイムスタンプには二種類ある．Read only のトランザクションに用いるものを thread.rts とし，読込みと書込み トランザクションに用いるものを thread.wts とする．thread.wts は上述した方法で生成される通常のタイムスタンプである．thread.rts は，ある時点の全スレッド間において最小の thread.wts を min_wts として，min_wts - 1 となる．このように生成された thread.rts が read only のトランザクションに用いられることによって，必ずそのトランザクションがコミットされるようになる．また，read set を集めることと，validation phase を省くことができるため性能向上に寄与する．min_wts と同様に min_rts も定義でき，それぞれを算出するのはリーダースレッドが定期的 (我々の実装では 100 マイクロ秒ごと) に行う．生成方法をふまえるとタイムスタンプは単調増加かつユニークとなる．

Time Stamp Counter [8] はコアごとに違いがある．それを

修正するために Cicada では，始まって間もないトランザクションを優先させ，長時間続いているトランザクションを劣後させる方針であり，二種類の同期がサポートされている．

2.1.1 One-sided synchronization

ワーカースレッドは他ワーカースレッドのトランザクション処理を中断させることなく他ワーカースレッドのローカルクロックをラウンドロビンで確認し，自身のローカルクロックより新しければそれを自身の新しいローカルクロックとする．これは 100 マイクロ秒ごとに行われる．

2.1.2 Temporary clock boosting

アボートした時に clock boost に値を設定する．値は想定されるコア間のクロックのずれ以上 (我々の実装では 1 マイクロ秒) の大きさにする．トランザクションがコミットし次第，clock boost に 0 秒を設定する．

2.2 多 版 法

Cicada におけるレコードのバージョンは単方向連結リストとなっていて，それぞれのヘッドは配列として並んでいる．これは後述する図 1 における Head node array にて示される．バージョンは生成された時間において，新しいものから古いものへ順に並んでいる．

バージョンは以下のメンバを有している．

- Write timestamp (wts) ... バージョンが生成された時間.
- Read timestamp (rts) ... バージョンが読み込まれた最大の時間.
- Record data
- Commit status (status) ... バージョンの状態を示す. pending, committed, aborted, unused, deleted の 5 種類ある.
- Allocation information ... NUMA に対応した割り当てのための NUMA ノード ID およびバージョンサイズ等の割り当て情報.

生成される全てのバージョンは validation phase で一度到達可能となる．その時にバージョンのステータスは pending で初期化されている．コミットかアボートされた時，ステータスはそれぞれに変更される．一度グローバルにインストールされたバージョンにおいて変更されうるメンバは read timestamp とバージョンステータスのみである．

トランザクションはワーカースレッドへ割り当てられたタイムスタンプを利用して，オペレーションの対象とするレコードのバージョンリストから，適切なバージョンを検索する．検索は新しいバージョンから古いバージョンへリストを辿ることによって行う．v.wts > tx.ts となる全てのバージョンは無視され，次のバージョンを確認しに行く．そうでなかった場合は v.status を確認して，pending であればステータスが変更されるまで spin-wait する．Aborted であれば，そのバージョンは無視されて次のバージョンを確認しに行く．Committed であれば，そのバージョンを利用する．v.status が committed であるバージョンは，全てのワーカースレッドから観測可能 (visible version) であるとする．

Pending 状態であれば spin-wait する理由を説明する．Cicada のプロトコルでは，pending 状態となったバージョンは間も無く commit されるか abort される．これを推測で判断してオペレーションを実行することはカスケードアボートのリスクがある．これらの理由から，pending 状態であれば spin-wait を行う．

バージョンを検索する時，性能向上のために eary abort を行う．これは abort される可能性の高いトランザクションを早期に abort させる技術である．書込みオペレーションを visible version(v) に適用する時，v.rts ≦ tx.ts でなければ early abort を行う．ここで early abort を行わなかったとしても，validation phase においてこの条件で abort されることが分かっているからである．

Cicada は書込みによって生成されるバージョンは，バージョンリストにおける最新バージョンのみと定めている．書込みオペレーションを実行する時，対象とするレコードの最新バージョン (v) が committed もしくは pending であり，かつ v.wts > tx.ts であれば early abort を行う．Committed であれば最新バージョンが生成できないので abort される．pending であったとしても，プロトコルにおいて pending version は commit される可能性が高いため，abort をしてしまう．

Cicada は read-own-writes をサポートしている．トランザクションが再度同じバージョンへアクセスした時，スレッドローカルバージョンが存在すればそれを読み込む．トランザクション内において，同じレコードに対して実行した最後の書込みオペレーションが古いものであった場合，もし対象としたバージョンへのポインターを覚えていたとしても，他のワーカースレッドで実行されるトランザクションによってポインターが書き換えられているかもしれない．それを防ぐための，トランザクション内で一貫性を保証する技術である．

トランザクション内の過去の書込みオペレーションは軽量のスレッドローカルなハッシュテーブルで検索することができる．ハッシュ値には record ID を利用する．

2.3 プロトコル

Cicada のプロトコルは楽観法 [6] と同様に，read phase, validation phase, write phase の 3 pahse に分類される．

2.3.1 Read phase

トランザクションが始まると，ワーカースレッドは最初に自身が保持するタイムスタンプを計算する．トランザクションはオペレーションに利用すべきバージョンを探索するために，そのタイムスタンプを利用してレコードのバージョンリストを検索する．

2.3.2 Validation phase

Validation phase には以下の 3 つのステップがある．

- Pending version installation

Write set にあるバージョンをバージョンリストに仮バージョンとして挿入する．

- Read timestamp update

必要があれば read timestamp を更新する．Read set に含まれる全てのバージョンは v.rts ≧ tx.ts が保証される．このタイムスタンプの更新は，他のトランザクションへ，更新するタイムスタンプの値とほぼ同じ時刻にバージョンが読み込まれたこ

とを通知する意味合いを兼ねる．

- Version consistency check

– (a) Read set のレコードにおける，以前コミット済みであった全てのバージョンは依然としてコミット済みである．

– (b) Write set のレコードにおける最新バージョンは v.rts ≦ tx.ts である．

(a) によって，このトランザクションによって読み込まれたバージョンのステータスを変更する新しいオペレーションを行った他のトランザクションが無いことを保証する．(b) によって，既にコミット済みのトランザクションを無効にするような古すぎるバージョンをコミットしないことを保証する．

2.3.3 Write phase

Write set のログを取り，インストールされた仮バージョンのステータスを pending から committed に変更する．

3. ログ先行書込み

3.1 ログ先行書込み手法 WAL

ログ先行書込み (Write-Ahead Logging) [9] [10] [11] とは，障害対策とし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生成されたトランザクションログはメモリ中の WAL バッファに溜められていく．トランザクションのコミット処理によってそれまでに溜められていた WAL バッファ中のログをまとめて，ストレージの WAL ファイルに書き込む．障害復旧時にトランザクションマネージャはログの有無によって，あるトランザクションが成功したのか失敗したのかを判断し，ロールフォワード/ロールバックを行う．

WAL はトランザクションの ACID 特性のうち，atomicity と durability を保証する技術である．

3.2 並列ログ先行書込み手法 P-WAL

従来の WAL において，一つのバッファに対するロックの競合と，一つの WAL ファイルへの IO が性能のボトルネックとなった．それを改善する並列ログ先行書込み手法 P-WAL [12] が提案された．

P-WAL が従来の WAL と異なる点は，従来一つであった 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ことである．ワーカースレッドの数だけ WAL バッファを分割することで，ログをバッファに挿入する際にロックを獲得する必要が無くなる．また，WAL ファイルも同様に分割することで，WAL バッファ中のログをまとめてストレージの WAL ファイルに書込む際にロックを獲得する必要が無くなる．

4. 並行性制御法 Cicada の再実装

4.1 データ構造

我々が実装した Cicada のデータ構造を 1 に示す．

Cicada のタプルはキー，バージョンを保持している．キーはタプルを特定するために利用され，バージョンはデータベースの現在の状態を表す．

バージョンは read timestamp，write timestamp，バージョンステータス，キー，バリュー，次のバージョンへのポインタを持つ．Read timestamp はそのバージョンが読み込まれた最大の時間を示す．Write timestamp はそのバージョンが生成された時間を示す．バージョンステータスは仮バージョンか，コミット済みか，アボートされたかを示す．キー，バリューは Integer を利用した．

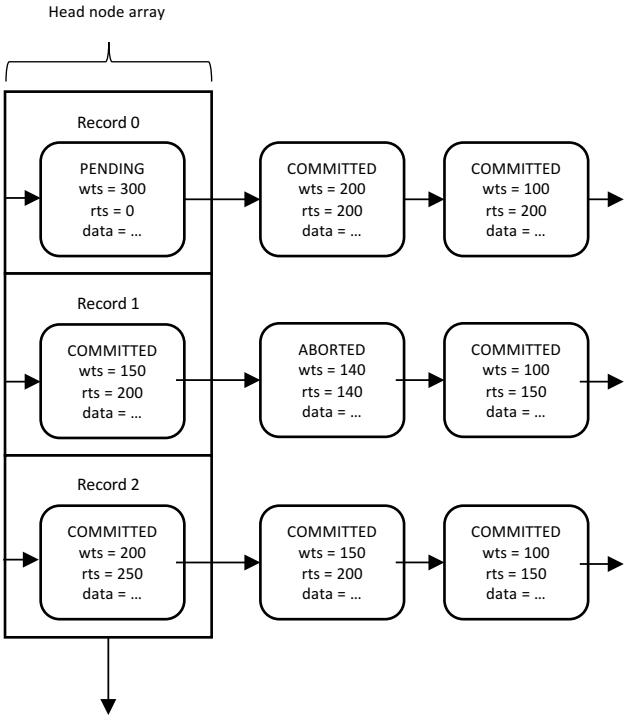

図 1 Multi-version data structures in Cicada ([4] より引用したものを改変)

表 1 評価環境

プロセッサ	Intel(R) Xeon(R) CPU E5-2695 v2 @ 2.40GHz
コア数	24
メモリ	66GB
OS	CentOS release 6.9 (Final)

4.2 プロトコル

我々が実装した Cicada におけるトランザクション処理の流れを図 2 に示す．

4.3 ロギング

Cicada においてログとして記録したものは write set である．これらはインメモリに書込まれ，ストレージに書き出す代用として一定時間の遅延を挟んだ．こうすることで様々なデバイスの IO 速度を想定できるようにした．

5. 評価と結果

5.1 並行性制御法の評価

我々が再実装した Cicada は適切に機能するかどうか検証するために，多版法として MVCC2PL [5]，楽観法として OCC [6] を実装し，自身の Cicada と性能を比較する．

評価環境は表 1 とする．ワークロードはトランザクション数 10 万件，tuple 数 100 万，トランザクションあたりのオペレーション数 20 とした．それぞれ 5 回実行した平均値をプロット

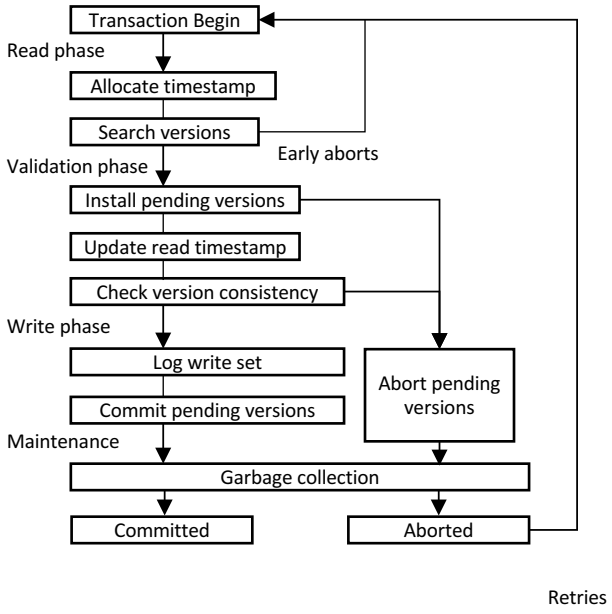

図 2　Workflow in cicada ([4] より引用したものを改変)

点として採用している．グラフのラベルに記載されている read 20%，read 50%，read 80%というのはオペレーション全体において，読込みオペレーションの占める割合を，それぞれ 20%，50%，80%としていることを表している．すなわちオペレーションの read / write 配分が read intensive，write intensive，even となるよう設定した．オペレーションは事前に C++の STL<random>を利用して生成した非決定乱数によりインメモリに生成し，それをロードする形で利用している．データベースの初期状態も同様の生成方法である．ロギングは無効にして並行性制御法のみの性能評価を行う．

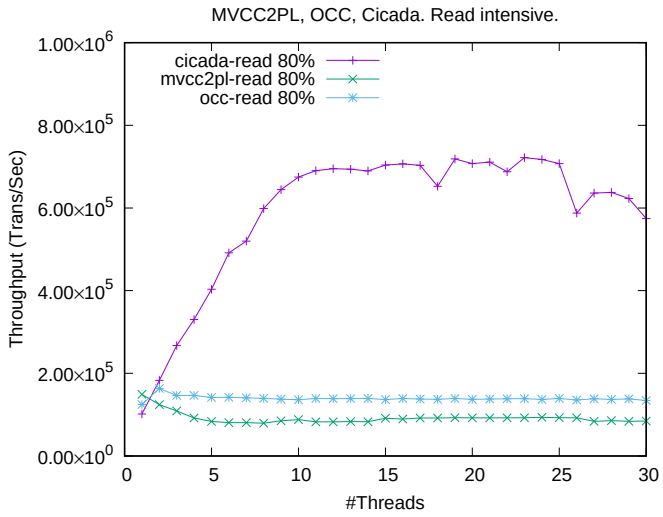

図 3　Read intensive における性能評価

実験の結果を図 3，図 4，図 5 に示した．ワークロードの設定

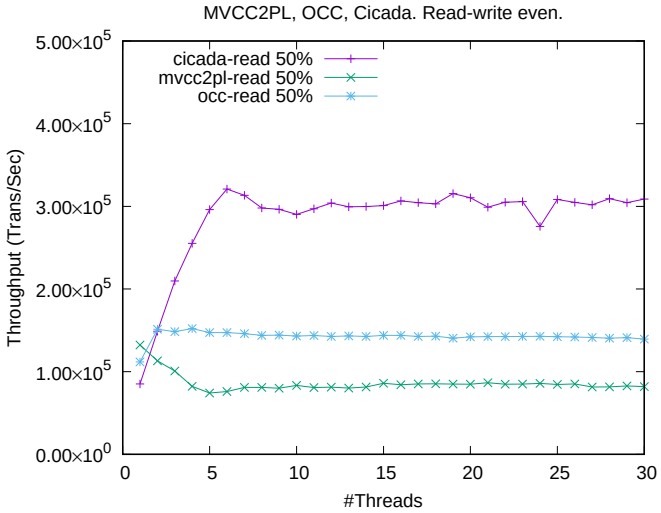

図 4　Read / write even における性能評価

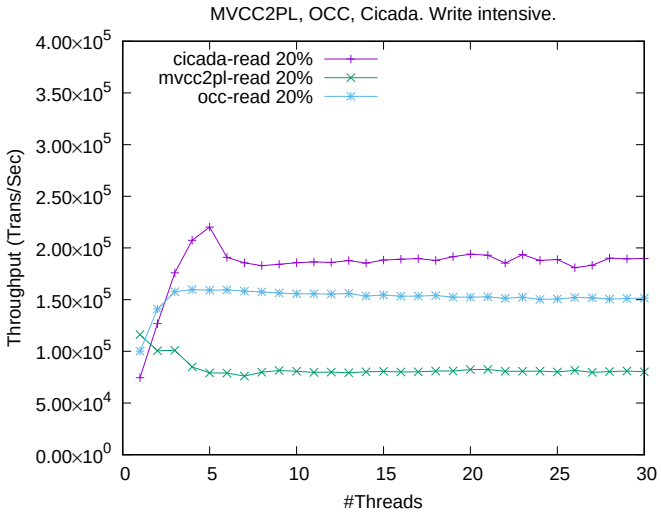

図 5　Write intensive における性能評価

上，実験は競合が少ない環境下にあるため OCC が MVCC2PL より優れた性能を示すと考えられる．

実験の結果より，Cicada が OCC より優れた性能を示す．前提として，OCC はタイムスタンプを集中発行していることがボトルネックの一つであると考えられる．Cicada は分散タイムスタンプ法を採用していることで，そのボトルネックを排除できている．よってこのように OCC より優れた性能を示すことができたと考えられる．

結論として，Cicada は MVCC2PL，OCC と比較して優れた性能を示した．

5.2 Cicada + P-WAL の評価

Cicada の原論文 [4] にてロギングの方針は示されているものの，詳細は論じられておらず，性能評価もロギングを無効にされているのでロギングが実装されているか，またロギングを加味した時に性能がどうなるのかは定かでは無い．そこで，Cicada に並列ログ先行書き込み P-WAL [12] を実装して性能評価を行なう．

評価環境は表 1 に示すとおりである．ワークロードは予備評価とほぼ同じであり，ロギングを有効にしている点が差分である．なお，将来の NVRAM IO を想定し，インメモリにログを書き込んだ後の flush をせず，代用として一律 5 ナノ秒の遅延を挟む．これは読込み，書込みの割合によらず一定である．

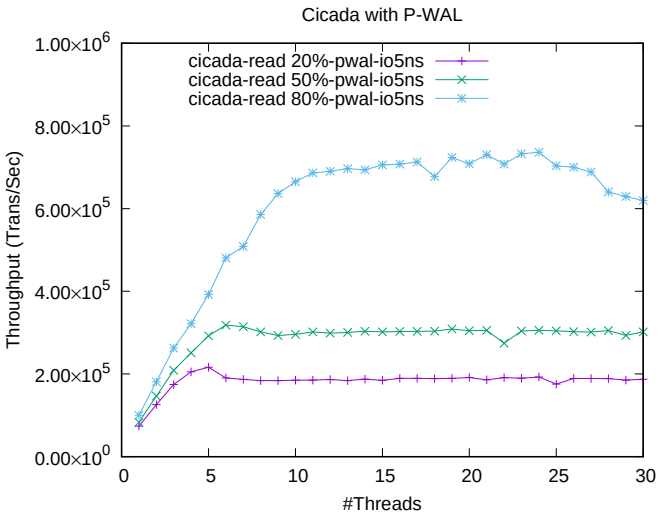

図 6　Cicada + P-WAL の性能評価

実験の結果を図 6 に示した．Cicada へ並列ログ先行書込み法 P-WAL を結合させることで，Cicada の性能劣化を引き起こさないことが示された．

6. 結　　論

本論文では，まず並行性制御法 Cicada を再実装し，それが優れた性能を示すことを検証した．本論文では次に Cicada に並列ログ先行書込み法 P-WAL を実装し，P-WAL が Cicada に性能劣化を引き起こさないことを示した．

謝　　辞

本研究の一部は，JST CREST JPMJCR1413，JST CREST JPMJCR1303，科研費＃ 16K00150，科研費＃ JP17H01748 による．

文　　献

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

[2] Jim Gray, Andreas Reuter, 優, 喜連川. トランザクション処理: 概念と技法. 日経 BP 社, 2001.

[3] Philip A Bernstein, Vassos Hadzilacos, and Nathan Goodman. Concurrency control and recovery in database systems. 1987.

[4] Hyeontaek Lim, Michael Kaminsky, and David G Andersen. Cicada: Dependably fast multi-core in-memory transactions. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pp. 21–35. ACM, 2017.

[5] David P Reed. Implementing atomic actions on decentralized data. *ACM Transactions on Computer Systems (TOCS)*, Vol. 1, No. 1, pp. 3–23, 1983.

[6] Hsiang-Tsung Kung and John T Robinson. On optimistic methods for concurrency control. *ACM Transactions on Database Systems (TODS)*, Vol. 6, No. 2, pp. 213–226, 1981.

[7] Xiangyao Yu, Andrew Pavlo, Daniel Sanchez, and Srinivas Devadas. Tictoc: Time traveling optimistic concurrency control. In *Proceedings of the 2016 International Conference on Management of Data*, pp. 1629–1642. ACM, 2016.

[8] Part Guide. Intel® 64 and ia-32 architectures software developer's manual. *Volume 3B: System programming Guide, Part*, Vol. 2, , 2011.

[9] C Mohan, Don Haderle, Bruce Lindsay, Hamid Pirahesh, and Peter Schwarz. Aries: a transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. *ACM Transactions on Database Systems (TODS)*, Vol. 17, No. 1, pp. 94–162, 1992.

[10] Jian Huang, Karsten Schwan, and Moinuddin K Qureshi. Nvram-aware logging in transaction systems. *Proceedings of the VLDB Endowment*, Vol. 8, No. 4, pp. 389–400, 2014.

[11] Wook-Hee Kim, Jinwoong Kim, Woongki Baek, Beomseok Nam, and Youjip Won. Nvwal: exploiting nvram in write-ahead logging. In *ACM SIGPLAN Notices*, Vol. 51, pp. 385–398. ACM, 2016.

[12] 神谷孝明, 川島英之, 星野喬, 建部修見ほか. 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, 2017.