# 不揮発メモリを用いたSiloロギング法の性能評価

田中　昌宏†　　川島　英之†

† 慶應義塾大学　〒 252–0882 神奈川県藤沢市遠藤 5322
E-mail: †masa16.tanaka@keio.jp

**あらまし**　トランザクション処理技法 Silo のログ永続化法を設計・実装し，スレッド数・バッファサイズなどの実装パラメータがトランザクション性能に及ぼす影響について，Optane DCPMM を用いて評価を行う．また，リアルタイム性が求められる使用を想定して，Silo ロギングにおける低遅延安定化の手法について提案する．Silo ロギングの実装では，NUMA アーキテクチャのためのスレッド・メモリ割り当てに対応した．評価実験では，SSD に対する DCPMM の書き込み性能の優位性，ロガースレッドを DCPMM モジュール毎に複数割り当てによる性能向上，ログバッファ数の増加による性能低下の傾向などが観察された．低遅延安定化の提案手法の実証実験では，性能を 17%抑えることにより，永続化遅延時間を 9.1 ミリ秒から 3.3 ミリ秒に短縮できることを確認した．

**キーワード**　Silo，トランザクション，ロギング，不揮発メモリ

## 1　はじめに

メニーコアマシンの性能を活用するためのインメモリ DB 高性能トランザクション技術として，Silo [1]，Tictoc [2]，Cicada [3] を始め，様々な手法が提案されている．複数のトランザクション手法が検証可能なベンチマークとして DBx1000 [4]，CCBench [5] がある．

一方，障害からのリカバリのため，インメモリ DB にはロギングが必要である．我々はロギングの性能比較が可能なベンチマークが必要と考え，その実装を計画している．Silo のロギング法を実装した SiloR [6] では，ログ発生速度よりストレージの書き込み性能が高いケースについてのみ検証している．また，ロギング実装において考えられるパラメータを変えた場合にどのようなケースで性能が最適であるかについての検証はされていない．本研究では，そのような実装パラメータのサーベイが可能なロギングベンチマークを実装し，性能について検証することを目的とする．今回は Silo ロギング法を実装し，ログバッファなどのパラメータについて調査する．

トランザクションの性能は，ログを保存するストレージの性能にも依存する．高速かつ低遅延な永続化メモリとして期待されているのが，Intel Optane Data Center Persistent Memory Module (DCPMM) である．本稿では，DCPMM を用いた Silo ロギング性能の初期調査について報告する．

さらに，ロボティクス等のリアルタイム処理が必要な分野からは，クライアントからの要求が来てから永続化されて応答が帰るまでの永続化遅延が重要になる．そこで，遅延が小さい DCPMM を用いた Silo ロギング法において永続化遅延を短縮する方法について探る．

## 2　Silo ロギング法の概要

Silo [1] は，メニーコアを活用して高性能するために設計されたインメモリトランザクションプロトコルである．Silo の並行性制御プロトコルは，write set についてはロックを取り，read set についてはロックを取らない楽観的並行性制御法を採用している．

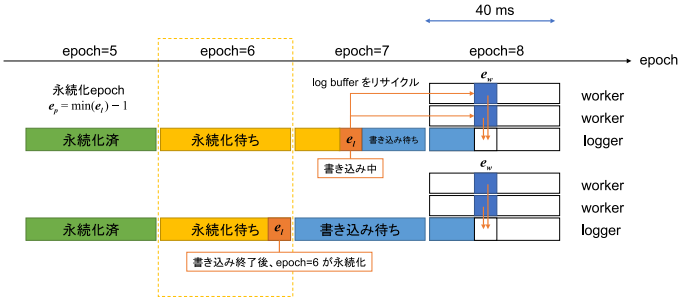


図 1　Silo ロギング法におけるログ永続化までの流れ．

Silo は，epoch と呼ばれる 40 ms 毎にカウントアップするグローバルなカウンタを持つ．この epoch は，トランザクションの serial order やログ永続化など色々な局面で用いられる．発行コストの高い Log Sequence Number (LSN) の代わりに epoch を用いることで，性能向上が期待ができる．

Silo のロギングは Silo [1] および SiloR [6] の文献に記述されている．Value-logging の採用により，リカバリ時には redo のみ行い，undo は不要である．Silo ロギング法の epoch に基づく永続化の仕組みを図 1 に示す．Silo ではロガースレッドをストレージの数に応じて複数起動し，各ロガースレッドは複数のワーカースレッドからログを受け取る．各ロガーは，担当ワーカーの epoch 及び未永続化ログの epoch の最小値 $e_l$ をチェックして保持する．全ロガーの $e_l$ から永続化エポック (persistent epoch) $e_p$ を $e_p = \min(e_l) - 1$ から求め，それ以下のエポックについては永続化が完了しているとして，クライアントに完了通知を行う．

## 3　Silo ロギング法の実装

今回の実装は，CCBench [5] を拡張する形で行った．ロギン

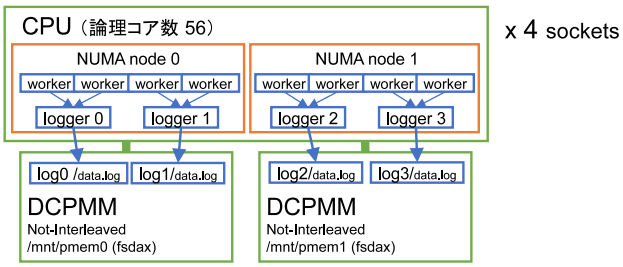


図 2　ソケット内のスレッド配置.

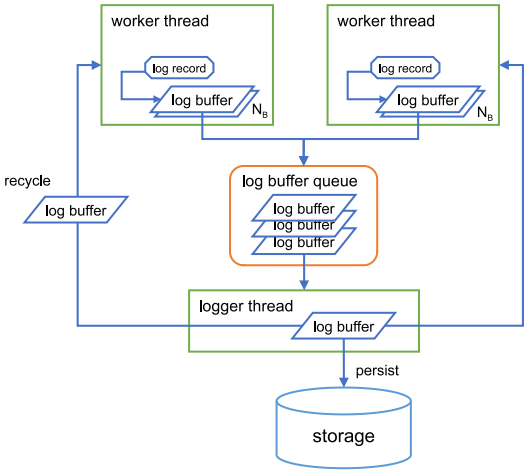


図 3　ログバッファの概要.

グのみ実装し，チェクポイント，リカバリは実装していない.

### 3.1　ロガースレッド

Silo では，1 つのロガースレッドが複数のワーカースレッドを担当してログを受け取る．メモリアクセス性能が不均一な NUMA (Non-Uniform Memory Access) アーキテクチャにおいて，図 2 のようにワーカースレッドとロガースレッドを同一の NUMA ノードのコアで実行するような配置にする必要がある.

今回の実装では，各ワーカースレッドとロガースレッドが動作する論理コアの ID を次のようにコマンドラインオプションで指定する.

```
silo.exe -affinity 0:1,2,3+4:5,6,10
```

この例では，最初のロガースレッドを論理コア#0 に割当て，論理コア#1, #2, #3 に割り当てられたワーカースレッドからログを受け取る．+以降は次のロガースレッドである．スレッドに対する論理コア ID の指定は，`pthread_setaffinity_np` 関数で行った．最初の (0 番目の) ロガースレッド#0 は，実行ディレクトリ内の `log0` というディレクトリの下に `data.log` というログファイルを作成してログを保存する.

### 3.2　ログバッファ

ログバッファの概要を図 3 に示す．ログバッファは，ワーカー毎に $N_B$ 個用意しておく．ワーカーがログをログバッファに書き込み，バッファが一杯になるか，epoch が変わるタイミングでキューを通してロガーに渡す．ロガーはログバッファの内容を永続化ストレージに書き込んだ後，再びワーカーに戻してリサイクルする.

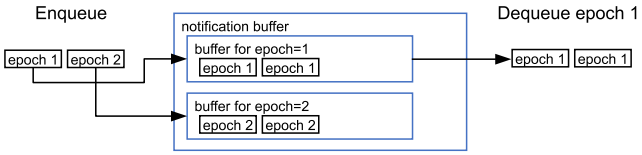


図 4　永続化通知バッファの概要.

ログ発生速度がストレージへのログ書き込み性能を超える場合，ログバッファはキューに溜まっていく．そしてワーカーの側は書き込めるログバッファのストックがなくなると，ロガーからログバッファが返されるを待つという状態になる．このような backpressure をかけてトランザクションの進行を遅らせている.

SiloR の論文 [6] によると，ログバッファのサイズは 512KB としているが，ログバッファ数に関しては具体的な記述がない．本研究ではロガー数，ログバッファサイズ，ログバッファ数のパラメタと，Silo ロギングの性能の関係について調査する.

YCSB ワークロードでは，データベースを保管するメモリをインターリーブさせるほうが性能が良いため，`numactl --interleave=[nodes]` を用いてベンチマークを実行する．しかし，ログバッファに関しては CPU 及び DCPMM と同じ NUMA node の DRAM に確保するため，`numa_set_localalloc` 関数を用いた.

### 3.3　永続化通知バッファ

前述のように，Silo ロギング法では，ストレージに書き込み永続化が完了したトランザクションのうち，$\min(e_l) - 1$ 以下の epoch に属するもののみクライアントに永続化完了通知を行うことができる．そのため，書き込み同期が完了した後，通知を行うまで，トランザクションの識別情報を永続化通知バッファに一時保管する必要がある．我々は，永続化通知バッファを効率よく動作するため，図 4 のように epoch 毎にバッファを分け，挿入時に epoch 毎に分類するように実装した．$\min(e_l)$ が繰り上がるタイミングでバッファの内部を検査し，$\min(e_l) - 1$ 以下についてはクライアントに通知する．通知が終わったバッファは空にして次の epoch のためにリサイクルする.

## 4　Silo ロギング低遅延安定化手法の提案

ロボティクス等のリアルタイム処理が必要なシステムに適用可能なデータベースを実現するには，クライアントからの要求から応答までの時間が短いことが求められる．トランザクションシステムにおいて，この応答時間は，ログに書き込んでから永続化するまでの永続化遅延時間に相当する．そこで，DCPMM を用いた Silo ロギング法において永続化遅延を短縮する方法について探る.

Silo ロギング法では，標準で 40 ms 毎に 1 epoch 進むとされており，永続化遅延時間はこのエポック間隔に依存す

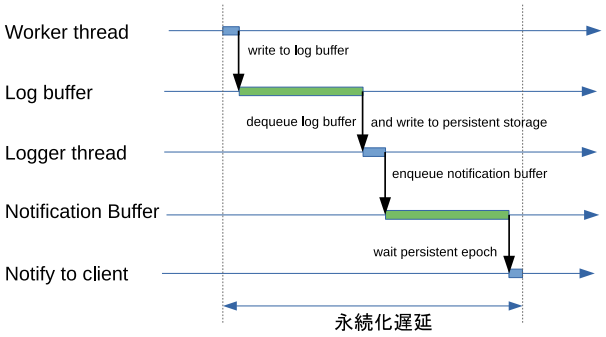


図 5　永続化遅延時間の内訳.

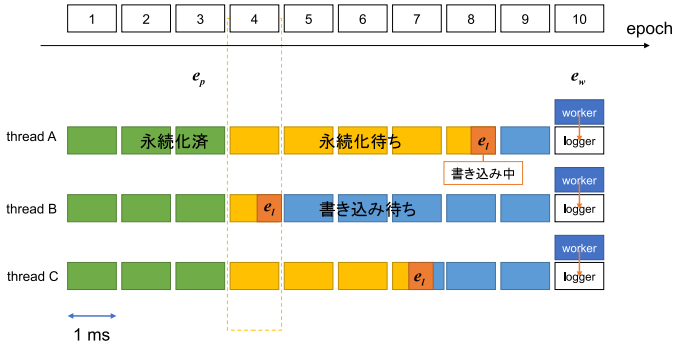


図 6　一部のロガースレッドの進行の遅れによる永続化の遅れ.

る．単純に考えれば，このエポックが進む時間 `epoch_time` を 40ms から短縮すれば永続化遅延が短縮されると考えられる．`epoch_time`=1 ms で行った予備実験では，ワーカー数を増やすと永続化遅延時間が約 10 ms と予想より大きくなるケースが見られた．この原因を探るため，図 5 に示した永続化遅延時間の内訳を調べると，notification buffer に滞在する時間が長いことがわかった．notification buffer の滞在中は，永続化エポック (persistent epoch) が通知待ちのエポックと同じになるまで待っている状態であり，永続化エポックの進行が遅れていることを示す．永続化エポックが進まない原因を探ると，図 6 に示すように，ロガー毎に処理の進行のばらつきがあり，進行が遅いロガーのために全体の永続化が遅れることがわかった．そこで，本研究では，

> ワーカーの epoch $e_w$ と ロガーの epoch $e_l$ の差が閾値 (`max_epoch_diff`) 以上のとき，ロガーが担当するワーカーの進行を止める．

という方法を提案する．アルゴリズムを図 7 に示す．このアルゴリズムによってログバッファへのログ追加を止めてロガーの処理を進めることにより，persistent epoch を進めることが狙いである．実装上の注意点としては，ワーカーが止まることによってそのトリガーで動くロガーも止まり，デッドロックが発生することがある．それを防ぐためにログ待ちの間もロガーの epoch を定期的に更新する必要がある．本提案手法の有効性について 6.3 節で検証する．

**Algorithm 1** Low latency algorithm for Silo logging

**Require:** global epoch $E$, logger epoch $e_l$, `max_epoch_diff`
  **loop**
    **do** Silo Phase 1
    $e_w \leftarrow E$
    **do** Silo Phase 2
    **do** Silo Phase 3
    **wait while** $e_w - e_l \geqq$ `max_epoch_diff`
  **end loop**

図 7　Silo ロギング低遅延安定化手法のアルゴリズム

表 1　使用マシン

| | | |
|---|---|---|
| CPU | Intel Xeon Platinum 8276 2.20GHz (1GHz 動作) | |
| | 1 チップの物理コア数 | 28 |
| | ソケット数/NUMA ノード数 | 4/8 |
| | 全物理/論理コア数 | 112/224 |
| | L1/L2/L3 キャッシュ | 32 KiB/1 MiB/38.5 MiB |
| DRAM | 512 GiB | |
| DCPMM | 128 GiB × 8 モジュール | |
| SSD | Micron 5200 Series 480 GB (RAID1) | |

## 5　測 定 環 境

### 5.1　測定マシン

測定に用いたマシンは表 1 の通りである．CPU は 4 ソケット，物理コア数は 112 であるが，Hyper-Threading により 224 の論理コアまで使用する．2.2 GHz で動作する CPU であるが，今回の測定では 1.0 GHz で動作させている．DCPMM は 8 モジュールあり，1 ソケットあたり 2 モジュール装着されている．今回の測定では，Not-Interleaved，File System DAX (fsdax) の設定で行い，ext4 ファイルシステムを構築してマウントした．

### 5.2　DCPMM と SSD の書き込み性能

まず今回ロギングに使用するストレージに対して，書き込み性能を測定した．この測定では，1 回の書き込みサイズを Silo のログバッファサイズと同じ 512 KiB とし，sync size まで繰り返して同期を行う．スレッド数 1 から 4 までにより並列に書き込む．

SSD の書き込み性能の測定結果を図 8(a) に示す．sync size が約 4 MB までは書き込み性能が上昇し，それ以上では 300 前後–400 MB/s の性能を示す．スレッド数を増やすと性能が向上する傾向もみられる．

次に，DCPMM に対して，同一の NUMA ノードの CPU からの書き込み性能を測定した．DCPMM への書き込み性能については，(1) warm up の有無，(2) 2 種類の書き込み方法 (`write` システムコールまたは `libpmem`) について調査した．warm up とは，測定の前にダミーデータを一度ファイルに書き込むことである．本番の測定ではそれを上書きする形で行う．warm up なしの測定結果を図 8(b) に，warm up ありの測定結果を図 8(c)

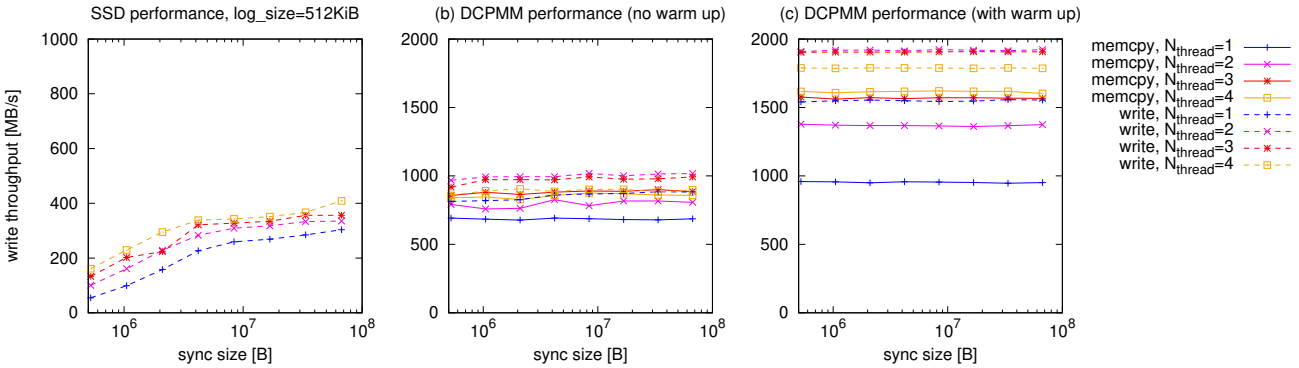


図 8　ストレージへの書き込み性能測定結果．(a) SSD，(b) DCPMM(warm up なし)，(c) DCPMM(warm up あり)．DPCMM は Not-Interleaved，CPU と同一 NUMA ノードの書き込み．

に示す．それぞれ，`write` システムコールまたは `libpmem` の結果をプロットしている．`libpmem` の場合は，`pmem_map_file` でマップしたアドレスへ `pmem_memcpy_nodrain` で書き込みを行った．測定結果から，warm up ありが warm up なしよりも約 2 倍性能が向上していることがわかる．これは warm up なしの場合に，[7] に記述されているページ初期化のオーバーヘッドが発生していることが考えられる．また，`libpmem` より `write` の方が全般的に性能が高いことがわかる．fsdax ではページキャッシュが回避されるため，`write` でも性能が出ることが期待できるが，性能に差が出る理由は不明である．並列書き込みに関しては，2 または 3 スレッドがスループットが高く，1 スレッドまたは 4 スレッドはそれらより低いという傾向もみられる．

以上の結果から，以下で述べるロギング性能の測定では，実装にシステムコールの `write` を用いる．warm up については，実装への対応が間に合わなかったため，warm up なしの状態で測定した．

### 5.3　測定方法

DIMM スロットに装着する DCPMM は，CPU との affinity が性能に大きく影響する．ロガースレッドの CPU とログ保存先の DCPMM を同一の NUMA ノードにする設定は次のように行う．まず，NUMA ノード`#0` に属する DCPMM モジュール `/dev/pmem0p1` に作成したファイルシステム内のディレクトリに `log0` という名前でシンボリックリンクを張っておく．次に，`-affinity` オプションによりロガースレッド`#0` に対して NUMA ノード`#0` に属する論理コア ID を指定してベンチマークを実行する．

## 6　測定結果

今回測定するワークロードは YCSB である．共通するパラメータはトランザクション中命令数 10，RMW 命令無し，レコード数 100 万，Skew 0 である．これらは CCBench のオプションで指定する．CCBench の内部 DB には Masstree [8] を使用する．同じ測定を 5 回行い，中央値を採用する．

### 6.1　1 ソケットでの SSD と DCPMM のロギング性能

SSD と DCPMM へのログを書き出した場合の性能を，YCSB-A (read : write = 50% : 50%) のワークロードにより比較した．使用した CPU は 1 ソケットであり，DCPMM はそのソケットに接続する 2 モジュールである．図 9(a) にトランザクションのスループットの測定結果を Mega transactions per second (Mtps) の単位で示す．SSD のスループットは 15 ワーカー を超えると頭打ちになり，最大で 4.2 Mtps (50 ワーカー + 6 ロガー) である．また，ロガースレッドの増加にる性能向上がわずかにみられる．一方，DCPMM のスループットは，ロギングなしの場合に近い性能が得られており，最大で 9.2 Mtps (54 ワーカー + 2 ロガー) である．ロガースレッド数による違いはみられない．

図 9(b) に本実験における書き込み速度を示す．ここでの書き込み速度は，書き込んだログのバイト数を，最初書き込み開始から最後の同期までの経過時間で割った値とする．図 9(a) のスループットとほぼ同じ形なのは，書き込み量がトランザクション数に比例し，経過時間がトランザクションの実行時間とほぼ同じであるためである．SSD の測定結果は約 300 MB/s で頭打ちになっており，これは図 8(a) で測定した SSD の書き込み性能とほぼ同じである．したがってこの場合は SSD の書き込み性能がトランザクションの性能を制限していることになる．一方，DCPMM の書き込み速度は最高で約 700 MB/s である．図 8(b) の DCPMM の性能測定結果から 1 モジュールあたり約 1 GB/s の性能があるとすると，2 モジュールでは約 2 GB/s の性能が期待できる．DCPMM の書き込み性能は，write=50% のワークロードのロギングには十分であると言える．

図 9(c) に永続化遅延時間の測定結果を示す．永続化遅延時間は，トランザクションの開始から永続化完了通知までの時間の平均である．epoch の単位が 40 ms であるため，永続化遅延時間も 40 ms が目安となる．DCPMM の場合は　ワーカー数の増加に伴い 45 ms (2 ワーカー + 2 ロガー) から最大 81 ms (54 ワーカー + 2 ロガー) である．SSD の場合は 60 ms (2 ワーカー + 2 ロガー) から最大約 1 秒 (50 ワーカー + 6 ロガー) まで増加している．SSD の遅延の大きさは，書き込み性能によ

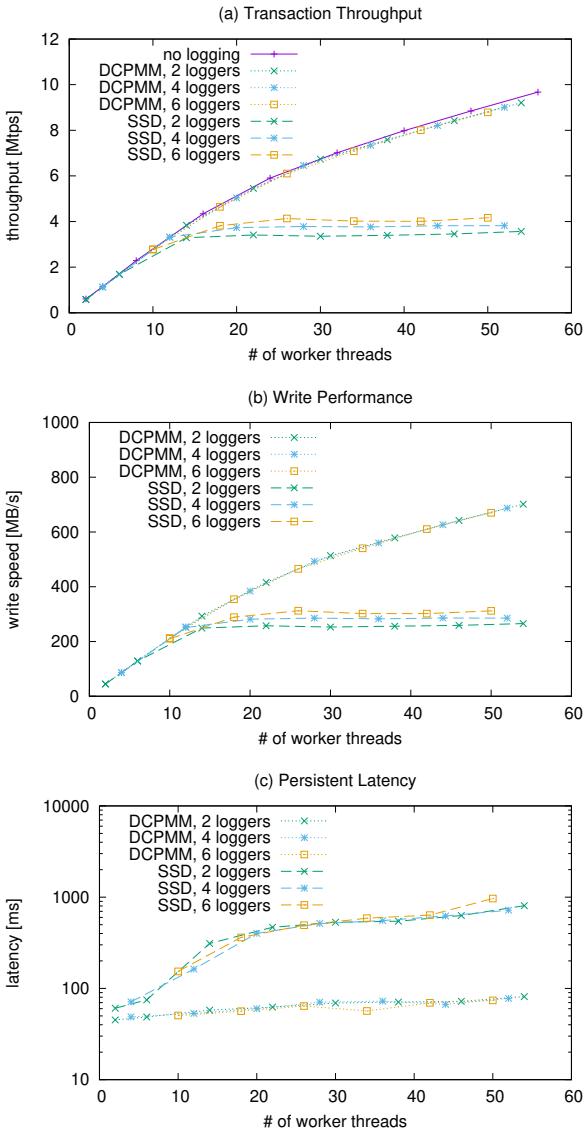


図 9　1 ソケット使用時の SSD と DCPMM(2 モジュール) へのロギング性能．Silo YCSB write=50%，ログバッファ数 $N_B = 4$．(a) トランザクション性能，(b) 書き込み性能，(c) 永続化遅延時間．

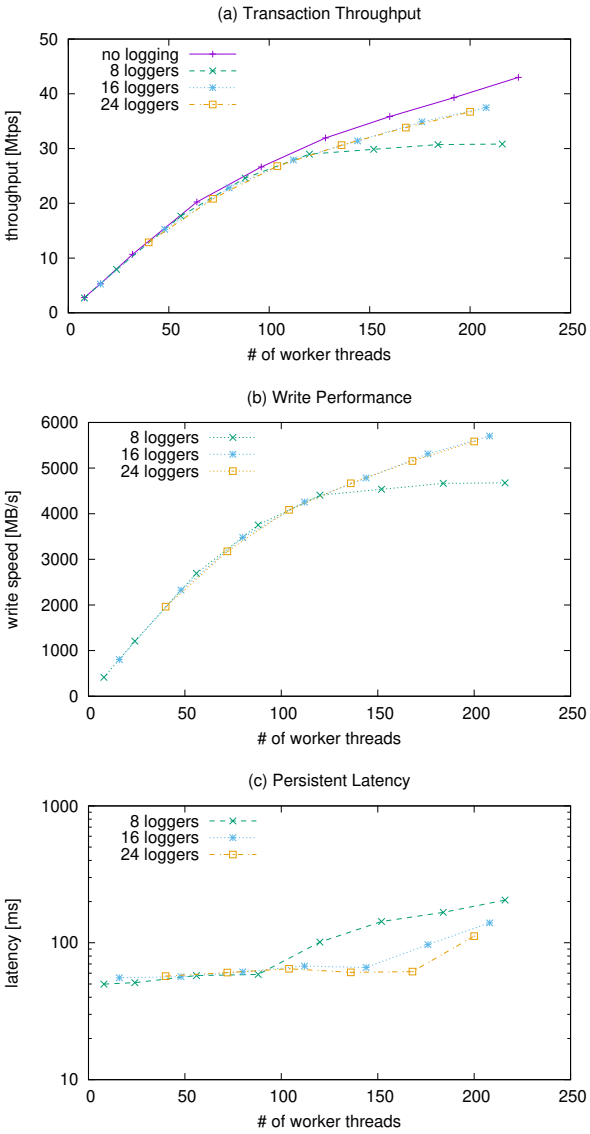


図 10　4 ソケット使用時の SSD と DCPMM(8 モジュール) へのロギング性能．Silo YCSB write=100%，ログバッファ数 $N_B = 4$．(a) トランザクション性能，(b) 書き込み性能，(c) 永続化遅延時間．

るものと考えられる．

### 6.2　4 ソケットでの DCPMM ロギング性能

次に，write が 100%の YCSB ワークロードを全 4 ソケットを使用して実行し，8 モジュール DCPMM に対して並列に書き込んだ場合の性能を測定した．このワークロードは，ログの発生量が最も多いケースである．測定結果は，ワーカー数に対するトランザクションのスループット及び永続化遅延時間のグラフとして示す．

#### 6.2.1　ロガースレッド数による性能

ワーカー毎のログバッファ数 $N_B$ を 4 に設定し，全ロガースレッド数を 8, 16, 24 (DCPMM モジュール毎に 1, 2, 3) として測定した．

図 10(a) にトランザクション性能の測定結果を示す．100 ワーカーを超えると，ロギングによる性能低下がみられる．8 ロガーの場合は約 30 Mtps で頭打ちになるのに対し，16, 24 ロガーの場合は ワーカー数とともに性能が向上している．16, 24 ロガーは同じ線上にあり，性能差はみられない．16 ロガーの最高性能は 37.5 Mtps (208 ワーカー)，24 ロガーは 36.5 Mtps (200 ワーカー) である．16 ロガーの方が ワーカー数が多いため，高い性能が得られた．

図 10(b) はこの実験における書き込み速度である．この結果も図 10(a) のスループットとほぼ同じ形である．書き込み速度は最高で 5.6 GB/s (208 ワーカー + 16 ロガー) である．一方 8 ロガーのケースでは，4.7 GB/s までで頭打ちとなっている．図 8(b) の 1 スレッドの DCPMM の結果から 1 モジュールあたり約 800B/s の性能があるとすると，8 モジュールで約 6.4 GB/s が期待できる．それより低い性能で頭打ちとなるのは，書き込み性能の他の原因があることが考えられる．

図 10(c) に永続化遅延時間の測定結果を示す．永続化遅延時

間が 50–70 ms に収まる部分がある一方，あるワーカー数以上ではワーカー数とともに永続化遅延が増えるという傾向がみられる．また，ロガー数が少ないほど少ないワーカー数で永続化遅延の増加が始まり，同じワーカー数で比較した遅延時間も大きいことがわかる．この理由として，1 つのロガーが担当するログのサイズがロガー数に反比例し，サイズが小さいほど早く永続化できることが考えられる．

### 6.2.2 ログバッファ数による性能

Silo ロギング法におけるログバッファ数 $N_B$ の性能への影響を調査するため，ワーカー毎のログバッファ数を 2 から 32 まで変えて測定した．

スループットの測定結果を図 11(a) に示す．次のような傾向が見られる．(1) $N_B = 2$ では，$N_B = 4$ 以上の場合に比べて全体的にスループットがやや低い．この理由として，バッファ数が少ないために受け渡しの時に待ちが発生することが考えられる．(2) $N_B = 16$ のとき 208 ワーカー，$N_B = 32$ のとき 176 ワーカー以上でスループットが下がっている．この理由として，バッファ数増加によるメインメモリの使用率の増加とが考えられるが，詳細は未解明である．

永続化遅延時間の測定結果を図 11(b) に示す．ワーカー数が 144 以上では，バッファ数の増加による永続化遅延の増加がみられる．同じワーカー数で比較すると，バッファ数が多いほど永続化遅延が大きい．この理由として，ログバッファ数が増えると，キューに積まれるログバッファの数が増え，キューに滞在する時間が長くなることが考えられる．

バッファ数が 2 のときにスループットがやや低く，バッファ数を増やすと永続化遅延が増加するため，バッファ数は 4 程度が良いと言える．

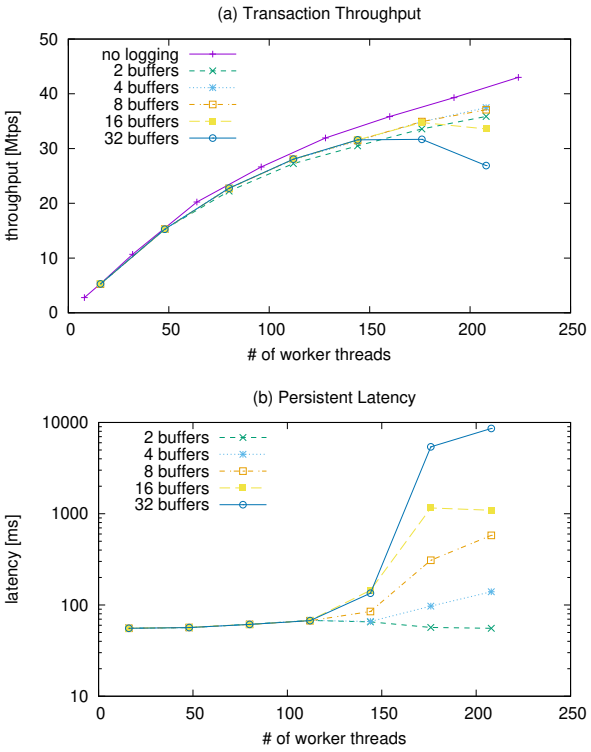


図 11　ログバッファ数 $N_B$ を変えた場合の DCPMM へのロギング性能．Silo YCSB write=100%，4 ソケット，8 DCPMM，ロガースレッド数 16．(a) トランザクション性能，(b) 永続化遅延時間．

## 6.3 低遅延安定化手法の検証

4 節で提案した Silo ロギング法低遅延安定化手法の検証のため，1 epoch が進む時間 `epoch_time` を Silo 標準の 40 ms から 1 ms, 0.5 ms, 2 ms に変更して実験を行った．この実験における共通の条件は，1 ソケットで 2 つの DCPMM モジュールを使用，バッファサイズ 512KiB，バッファ数 4，ワークロードは YCSB の write only である．5 回測定した中央値を採用し，最小最大をエラーバーで示す．永続化遅延時間とトランザクション性能の測定についてそれぞれ，`epoch_time`=1 ms の結果を図 12 と図 13 に，`epoch_time`=0.5 ms の結果を図 14 と図 15 に，`epoch_time`=2 ms の結果を図 16 と図 17 に示す．これらの図では，(a) は従来手法 (`max_epoch_diff` によるトランザクション進行の制限なし) の場合，(b), (c), (d) はそれぞれ `max_epoch_diff`=2, 3, 4 の結果を示す．

まず `epoch_time`=1 ms の測定結果について述べる．従来手法の永続化遅延時間 (図 12(a)) は，ワーカー数の増加とともに 10 ms 近くまで増加しているのに対し，`max_epoch_diff`=2 の永続化遅延時間 (図 12(b)) は 2.7 ms 未満と増加は見られない．これは，persistent epoch とワーカーの epoch が離れないようにトランザクションの進行を止めることが，永続化遅延の短縮に有効であることを示している．一方，図 13 のスループットに着目すると，従来手法 (図 13(a)) と比較して，`max_epoch_diff`=2 (図 13(b)) ではスループットが大きく低下している．このように，トランザクションの進行を止めることが，性能の低下を招くことがわかる．次に `max_epoch_diff`=3 のケースを見ると，永続化遅延時間 (図 12(c)) が `max_epoch_diff`=2 (図 12(b)) よりわずかに増えているものの 3.3 ms 以下であり従来手法 (図 12(a)) のような遅延増加は見られない．一方，スループット (図 13(c)) は `max_epoch_diff`=2 (図 13(b)) より大幅に向上している．このように `epoch_time`=1 ms の設定では，低遅延のままスループット低下を抑えているという点で，`max_epoch_diff` が 2 より 3 のほうが望ましいと言える．さらに `max_epoch_diff` を 4 に増やすと，スループット (図 13(d)) はわずかに改善するものの，永続化遅延時間 (図 12(d)) は増加するという傾向が見える．

次に `epoch_time`=0.5 ms の測定結果を見ると，`max_epoch_diff`=2, 3 については，永続化遅延時間 (図 14(b), (c)) は 2.7 ms 以下と小さいが，スループット (図 15(b), (c)) は大きく低下している．`max_epoch_diff`=4 については，永続化遅延時間 (図 14(d)) の傾向は，`epoch_time`=1 ms かつ `max_epoch_diff`=3 のケース (図 12(c)) と似ているが，スループット (図 15(d)) は図 13(c) より劣っている．このように，`epoch_time`=0.5 ms の場合は，性能に関して不利であると言える．

最後に `epoch_time`=2 ms の測定結果について述べる．ここで遅延時間のプロット (図 16) は図 12, 14 に比べて縦軸が 2 倍に

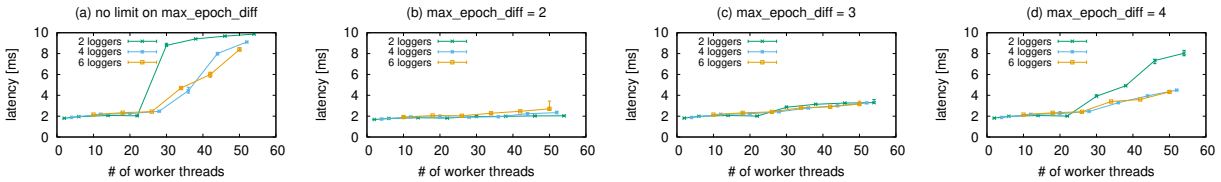


図 12　epoch_time = 1 ms の永続化遅延時間 (Silo YCSB write=100%，1 ソケット，2 DPCMM，$N_B = 4$)

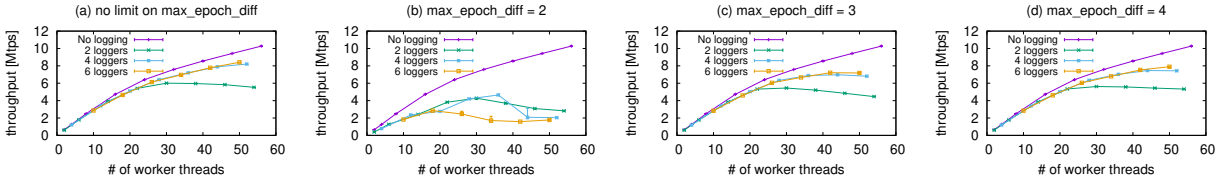


図 13　epoch_time = 1 ms のトランザクション性能 (Silo YCSB write=100%，1 ソケット，2 DPCMM，$N_B = 4$)

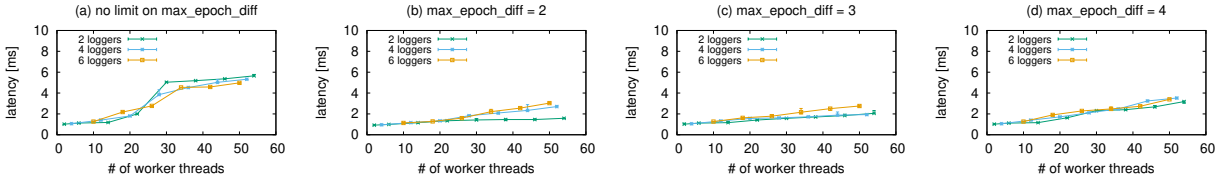


図 14　epoch_time = 0.5 ms の永続化遅延時間 (Silo YCSB write=100%，1 ソケット，2 DPCMM，$N_B = 4$)

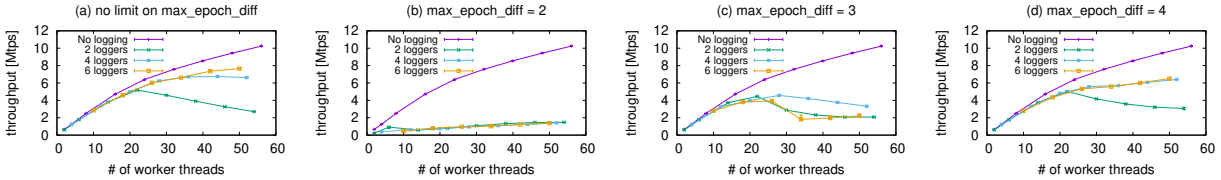


図 15　epoch_time = 0.5 ms のトランザクション性能 (Silo YCSB write=100%，1 ソケット，2 DPCMM，$N_B = 4$)

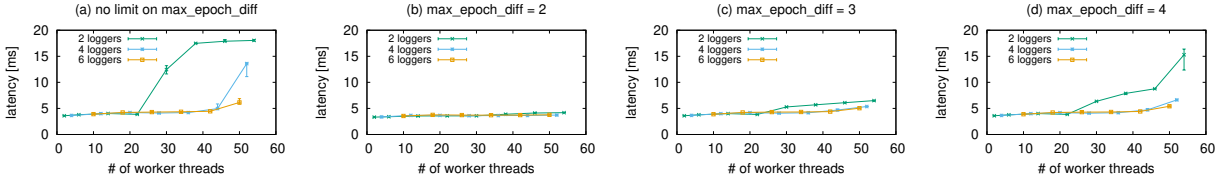


図 16　epoch_time = 2 ms の永続化遅延時間 (Silo YCSB write=100%，1 ソケット，2 DPCMM，$N_B = 4$)

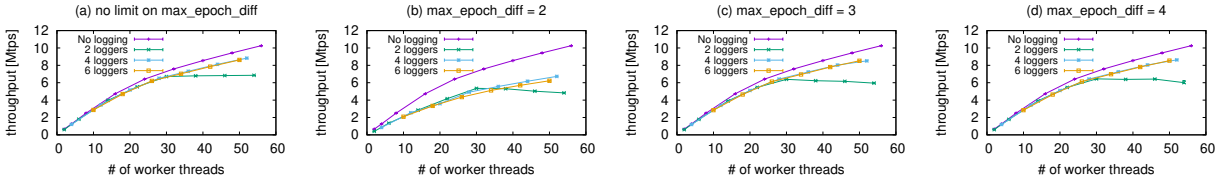


図 17　epoch_time = 2 ms のトランザクション性能 (Silo YCSB write=100%，1 ソケット，2 DPCMM，$N_B = 4$)

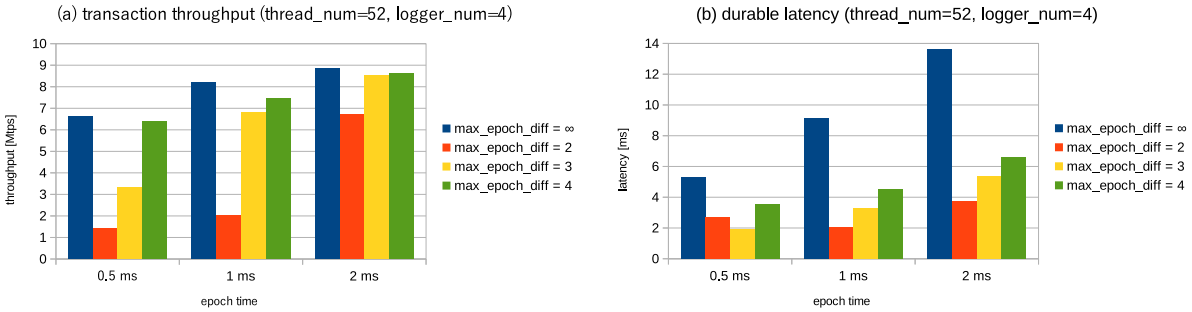


図 18　ワーカースレッド数=52，ロガースレッド数=4 におけるトランザクション性能と永続化遅延時間の比較

なっていることに注意していただきたい．`max_epoch_diff`=3, 4 に関しては，スループット (図 17(c), (d)) は `epoch_time`=1 ms の場合より改善しロギングなしに近づく一方で，永続化遅延時間 (図 16(c), (d)) は `epoch_time`=1 ms の場合の 2 倍ほどである．永続化遅延時間は `epoch_time` に比例する傾向がわかる．

以上のように，トランザクションの性能を抑えることによって，永続化遅延を小さくすることができることが実証できた．しかし，トランザクション性能と永続化遅延はトレードオフであり，どのパラメータがベストであるかはケースバイケースとなる．比較のため，性能が出るケースとしてワーカースレッド数が 52，ロガースレッド数が 4 のケースについて，`epoch_time` と `max_epoch_diff` のパラメータによるトランザクションスループットと永続化遅延時間の棒グラフを図 18 に示す．トランザクションスループットが低いという観点から，`epoch_time`=0.5 ms のケース全般と，`epoch_time`=1 ms かつ `max_epoch_diff` = 2 のケース (75%のスループット低下) は望ましくない．その他のケースで永続化遅延が最も小さいのは，`epoch_time`=1 ms かつ `max_epoch_diff` = 3 のケースの 3.3 ms であり，従来手法 (`max_epoch_diff`=∞) の 9.1 ms から 2.8 倍の短縮となる．この場合のスループットの低下は 17%である．こうした結果に基づき，実際に要請される永続化遅延とスループットに照らし合わせてパラメータを選択すればよい．

## 7 ま と め

本研究では，Silo ロギング法を CCBench を拡張して実装し，Optane DCPMM を用いた評価を行った．実装において，NUMA アーキテクチャへの対応として，ワーカースレッド・ロガースレッド・ログバッファ・DCPMM モジュールが同一の NUMA ノードにする設定を導入した．また，DCPMM の書き込み性能評価に基づき，書き込みにはシステムコールの `write` を利用した．評価では，YCSB-A のワークロードにおいて，SSD の書き込み性能がトランザクションの性能を制限する一方，DCPMM はほとんど性能低下がないことを実証した．また，書き込み量が最も多い write only のワークロードについて，スレッド数・ログバッファ数のパラメータについて調査した結果，DCPMM モジュール毎のロガースレッド数が 1 のケースや，ログバッファ数が 2, 16, 32 のケースで 4, 8 のときより性能が下がる傾向などが観察された．永続化遅延短縮のため，ワーカーのエポックと永続化エポックの差が `max_epoch_diff` を超えたときトランザクションの進行を止める方法を提案した．実験の結果，1 ミリ秒で epoch が進む設定のとき，`max_epoch_diff=2` では性能が 75%低下する一方，`max_epoch_diff=3` では 17%の性能低下と引き換えに永続化遅延時間を 9.1 ミリ秒から 3.3 ミリ秒へ短縮した．

## 謝　　辞

本研究は，NEDO「実社会の事象をリアルタイム処理可能な次世代データ処理基盤技術の研究開発」の支援により行った．

## 文　　献

[1] Tu, S., Zheng, W., Kohler, E., Liskov, B. and Madden, S.: Speedy transactions in multicore in-memory databases, *SOSP* (2013).
[2] Yu, X., Pavlo, A., Sanchez, D. and Devadas, S.: TicToc: Time traveling optimistic concurrency control, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Vol. 26-June-20, New York, New York, USA, ACM Press, pp. 1629–1642 (2016).
[3] Lim, H., Kaminsky, M. and Andersen, D. G.: Cicada: Dependably fast multi-core in-memory transactions, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Vol. Part F1277, New York, New York, USA, ACM Press, pp. 21–35 (2017).
[4] Yu, X.: DBx1000, https://github.com/yxymit/DBx1000.
[5] Tanabe, T., Hoshino, T., Kawashima, H. and Tatebe, O.: An Analysis of Concurrency Control Protocols for In-Memory Databases with CCBench, *PVLDB* (2020).
[6] Zheng, W., Tu, S., Kohler, E. and Liskov, B.: Fast databases with fast durability and recovery through multicore parallelism, *OSDI* (2014).
[7] : Speeding Up I/O Workloads with Intel® Optane™ Persistent Memory Modules, `https://software.intel.com/content/www/us/en/develop/articles/speeding-up-io-workloads-with-intel-optane-dc-persistent-memory-modules.html` (2019).
[8] Mao, Y., Kohler, E. and Morris, R.: Cache craftiness for fast multicore key-value storage, *EuroSys'12*, ACM Press, pp. 183–196 (2012).