Vol.2021-OS-153 No.4
2021/7/20

バイト単位でアクセス可能な永続メモリに最適化したキーバリューストア

大山　泰弘[1]　オブラン　ピエールルイ[†1]　貴田　駿[1]　河野　健二[1]

概要：バイト単位でアクセス可能な永続メモリが利用されるようになっているものの，従来のストレージデバイスとの性能特性の違いから，ストレージデバイスを永続メモリに置き換えただけではその性能を生かすことはできない．本論文ではキーバリューストアを対象とし，永続メモリに最適化した実装手法の検討を行う．既存のキーバリューストアでは，I/O 処理は大きな遅延を伴い，ブロック単位でのみアクセス可能であると仮定している．永続メモリのアクセス遅延は DRAM のそれに近い上バイト単位でのアクセスが可能であり，旧来のストレージデバイスを想定した最適化の多くは無効であることが多い． 本論文では，Intel Optane Persistent Memory を対象にキーバリューストア向けの最適化手法を提案する．Intel Optane Persistent Memory の性能分析から，永続メモリへの書き込み性能が著しく低下するアクセスパターンがあることを指摘し，キーバリューストアにおいてそのようなアクセスパターンを回避する手法を示す．低遅延のストレージデバイスである NVMe 等に向けて最適化されたキーバリューストアである KVell をベースに実装を行い，レイテンシとスループットの改善が図れることを示す．

1. はじめに

今日キーバリューストアは，検索エンジン [1] や SNS[2]，オンラインショッピング [3]，クラウドフォトストレージ [4] など幅広く用いられている．

本論文では，Intel Optane Persistent Memory[5] などの永続メモリを対象としたキーバリューストアの最適化手法を示す．永続メモリとは，バイト単位でアクセス可能な不揮発性メモリのことである．従来の不揮発性メモリに比べ，レイテンシが低く，バンド幅が広いという特徴がある．具体的には，NVMe SSD では，読み出しと書き込みのバンド幅は共に 2.4GB/s である．また，読み出しのレイテンシは 22us，書き込みのレイテンシは 29us である [6]．これに対して，Intel Optane Persistent Memory では，6 枚の DIMM をインタリーブさせた場合の読み出しのバンド幅は 38.2GB/s，書き込みのバンド幅は 12.8GB/s である．また，読み出しのレイテンシは 200–300ns，書き込みのレイテンシは 100ns 未満である [7]．

従来のキーバリューストアでは，ブロックデバイスに対して最適化が行われている．例えば，LSM-tree を用いたキーバリューストアでは，ブロックデバイスへのアクセスはランダムアクセスよりもシーケンシャルアクセスの方が性能が良いという特徴を利用し，ブロックデバイスへのアクセスがシーケンシャルに行われるように最適化している．

本論文では，Intel Optane Persistent Memory を対象に，永続メモリに最適化したキーバリューストアの実現手法を示す．まず，永続メモリへはバイト単位でアクセス可能であることを利用し，ブロック単位ではなく，バイト単位でアクセスを行うようにする．次に，永続メモリではシーケンシャルアクセスとランダムアクセスの場合で性能はあまり変わらない．したがって，ランダムアクセスを避けるための旧来の最適化は不要である．さらに，Intel Optane Persistent Memory の実装に依存した最適化を行う．Intel Optane Persistent Memory では，アクセスサイズが 256 バイト未満の場合にランダムアクセスの性能が落ちる．したがって，アクセスサイズには 256 バイト以上を用いるようにする．また，アクセスサイズを 256 バイト以上にした場合でも，CPU キャッシュを介して Intel Optane Persistent Memory へ書き込みを行うと，アクセスサイズが 256 バイト未満になってしまう．これは，CPU キャッシュから書き戻される際に，キャッシュラインごとにランダムに書き戻しが発生するためである．CPU キャッシュからの書き戻しがランダムに発生すると，Write Amplification が増大する．Write Amplification とは，書き込もうとする量に対して実際に書き込まれた量の比をいう．Write Amplification が増大すると必要以上に書き込みが発生してしまうため，

[1] 慶應義塾大学
Keio University, Japan
[†1] 現在，株式会社 IIJ インスティテュート

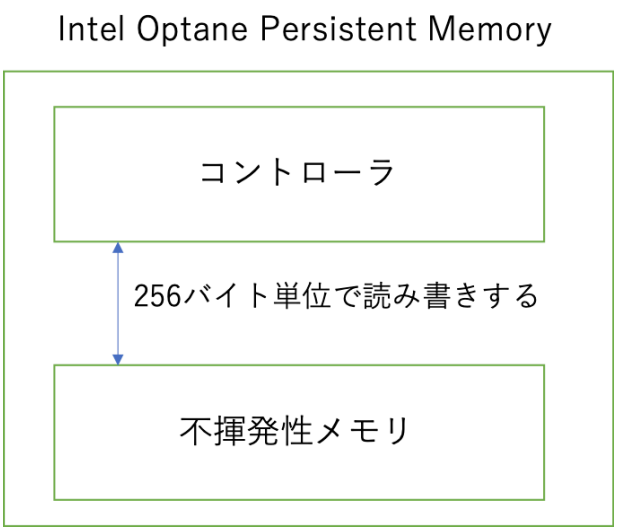

図 1: Intel Optane Persistent Memory の内部構造

ストレージデバイスの書き込み可能回数の減少やバンド幅の圧迫につながってしまう．したがって，CPU キャッシュからの書き戻しがランダムに発生しないように制御する必要がある．この点に関して，本論文では non-temporal store 命令を用いて CPU キャッシュを回避して直接 Intel Optane Persistent Memory へ書き込みを行うようにすることで，そもそも CPU キャッシュからの書き戻しが発生しないようにする．

上記の手法をキーバリューストアの一つである KVell[8] に対して実装し，性能測定を行った．KVell は，NVMe SSD などの低遅延・高スループットのブロックデバイスに最適化しており，旧来の LSM-tree や B-tree といったシーケンシャルアクセスのための最適化は行っていない．KVell と比較し，write-intensive なワークロードにおいて，データベースが空の状態から始めた場合の性能に関してはスループットが約 2 倍，レイテンシが約 1/2 倍となった．また，リカバリー後の性能に関しては，スループットが約 4 倍，レイテンシが約 1/7 倍となった．

2. Intel Optane Persistent Memory

2.1 内部構造

Intel Optane Persistent Memory の内部構造を**図 1** に示す．Intel Optane Persistent Memory では，不揮発性メモリを 256 バイトのブロックに分けて管理し，256 バイト単位で読み書きを実行する．したがって，Intel Optane Persistent Memory の内部的なアクセスサイズは 256 バイトである．

2.2 アクセスサイズの小さい書き込み

Intel Optane Persistent Memory では，アクセスサイズが 256 バイト未満の場合にランダムアクセスの性能が落ちる．既存の研究 [7] では，アクセスサイズを変化させた場合の DRAM と Intel Optane Persistent Memory の性能を比較している（**図 2**）．DRAM ではアクセスサイズによらずバンド幅は一定であるが，Intel Optane Persistent Memory ではアクセスサイズが 256 バイト未満の場合にバンド幅が狭くなっている．これは，上述の Intel Optane Persistent Memory の内部構造によるものである．そのため，Intel Optane Persistent Memory ではアクセスサイズが 256 バイト未満の書き込みについては，コントローラの内部でバッファリングし，256 バイトにまとめて書き込みを実行するようにしている．もっとも，アクセスサイズの小さい書き込みがバッファのサイズよりも広い範囲に対して行われる場合には，バッファ内のいずれかのブロックを書き戻した上で新たに 256 バイトのブロックを割り当て，そこに書き込みを行う．ブロックの書き戻しは 256 バイト単位で行われるため，ブロックに書き込まれたデータが 256 バイト未満の場合には Write Amplification が増大する．

2.3 Write Amplification

既存の研究では，書き込みの範囲を変化させてアクセスサイズの小さい書き込みを行った場合の Intel Optane Persistent Memory の Write Amplification の値について調査している（**図 3**）．書き込みの範囲が 16KB 以内の場合には，Write Amplification の値は 1 以下となっている．一方で，書き込みの範囲が 16KB を超える場合には，Write Amplification の値は 1 よりも大きくなっている．したがって，内部のコントローラのバッファのサイズは 16KB であると推定される．

CPU キャッシュを介して Intel Optane Persistent Memory へ書き込みを行うと，Write Amplification が発生する．まず, Intel Optane Persistent Memory への書き込みはメモリ参照命令を用いて行われるため，CPU キャッシュに対して行われる．CPU キャッシュから Intel Optane Persistent Memory へデータが書き戻される際に，キャッシュラインごとにランダムに書き戻しが発生する．書き込みが局所的な場合には上述の通りコントローラ内で書き込みを 256 バイトにまとめて実行するため，Write Amplification は発生しない．しかし，書き込みが広範にわたる場合にはコントローラ内のバッファにヒットしづらくなり，Write Amplification が増大する．したがって，CPU キャッシュを介して書き込みを行う場合には，CPU キャッシュからの書き戻しがランダムに発生しないように制御する必要がある．

CPU キャッシュからの書き込みがランダムに発生しないように制御する方法として，キャッシュラインのサイズごとに clwb 命令を発行する方法がある．この方法を用いれば，キャッシュに書き込まれた順序でキャッシュラインが書き戻されていくため，キャッシュラインは連続してコントローラ内のバッファに格納される．一方で，non-temporal

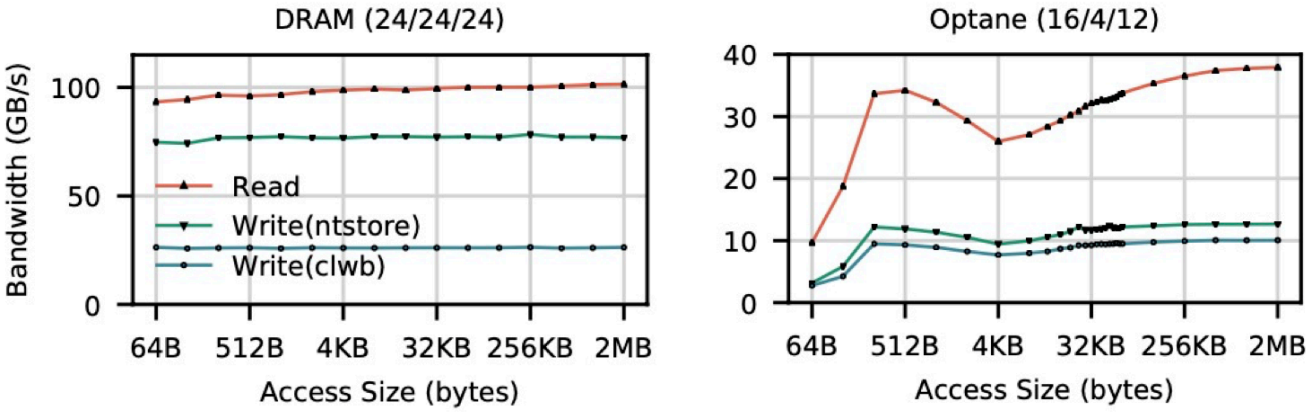

図 2: アクセスサイズを変化させた場合の性能
（文献 [7] より引用）

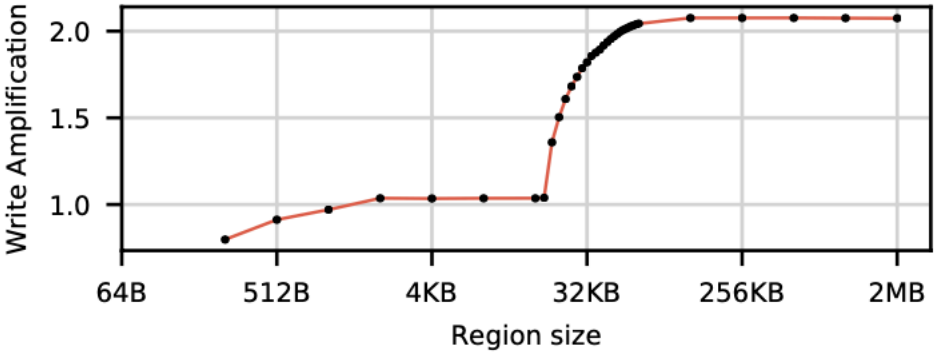

図 3: 書き込みの範囲を変化させた場合の Write Amplification の値
（文献 [7] より引用）

store 命令を用いて CPU キャッシュを回避して直接 Intel Optane Persistent Memory に書き込む方法もある．既存の研究 [7] では，アクセスサイズを変化させた場合の両者の性能について調査を行なっている（**図 4**）．アクセスサイズが 256 バイト以上の場合には，store 命令と clwb 命令を用いるよりも non-temporal store 命令を用いた方が性能が良い．前者では書き込みの前に一度データを CPU キャッシュに読み出す必要があるが，後者では読み出しを行わず直接 Intel Optane Persistent Memory へ書き込みを行う．したがって，アクセスサイズが大きい場合には読み出しにかかるオーバヘッドが大きくなるため，non-temporal store 命令を用いて書き込みを行った方が性能が良くなる．

3. 提案

本論文では，CPU ボトルネックを解消しつつ，Write Amplification を削減し，永続メモリに最適化したキーバリューストアの実装手法を提案する．

3.1 CPU オーバヘッドの削減

CPU 側の処理を減らすため，永続メモリにキーとバリューを書き込む際にはキーによるソートは行わず，永続メモリ上のランダムな場所にキーとバリューを書き込む．KVell[8] では，NVMe SSD のような高速に動作するデバイスにおいては CPU 側の処理がボトルネックとなることを指摘している．同様のことが永続メモリについても言える．永続メモリではシーケンシャルアクセスとランダムアクセスの性能はほとんど変わらないため，シーケンシャルアクセスを行うための最適化は不要である．また，キーとバリューの読み出しにかかるオーバヘッドを削減するため，永続メモリ上のキーとバリューの位置を DRAM 上のインデックスで管理する．

3.2 Write Amplification の削減

永続メモリへ書き込みを行う際には，アクセスサイズは 256 バイト以上とし，non-temporal store 命令を用いて書き込みを行う．Intel Optane Persistent Memory ではランダムアクセスの際にアクセスサイズが 256 バイト未満であると性能が落ちるため，アクセスサイズには 256 バイト以上を用いる．また，CPU キャッシュからランダムに書き戻しが発生し，Write Amplification が増大するのを避ける

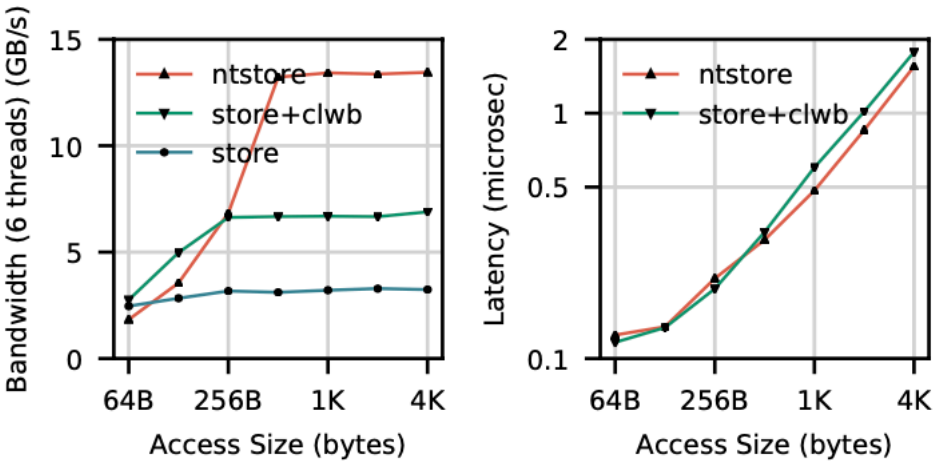

図 4: アクセスサイズを変化させた場合の永続命令の性能比較
（文献 [7] より引用）

ため，non-temporal store 命令を用いて書き込みを行う．store 命令と clwb 命令ではなく，non-temporal store 命令を用いる理由については，アクセスサイズが 256 バイト以上の場合には non-temporal store 命令を用いた方が性能が良いためである．

3.3 キーとバリューの更新

キーとバリューの更新を行う際には毎回同じ場所に対して更新を行うのではなく，永続メモリ上の別の場所に対して更新を行うようにする．non-temporal store 命令を用いて書き込みを行った場合，書き込み先のアドレスが CPU キャッシュに存在しない場合には CPU キャッシュを回避して直接 Intel Optane Persistent Memory へ書き込みを行うが，そうではない場合には CPU キャッシュからの書き戻しを行う．すなわち，clwb 命令を用いた場合と同様の動作になる．キーとバリューの更新前に既にキーとバリューの読み出しを行っていた場合，キーとバリューが CPU キャッシュに存在することになる．そのため，キーとバリューの更新時には毎回別の場所に対して更新を行うようにする．

3.4 フリーされたキーとバリューの管理

フリーされたキーとバリューに関しては，ビットマップを用いて DRAM 上で管理を行う．直接永続メモリ上でフリーされたキーとバリューをリストで管理する方法も考えられるが，Write Amplification 削減の観点から全て DRAM 上で管理を行うようにする．

3.5 リカバリーの処理

リカバリー時には，データベース全体のキーとバリューを走査し，データベース内に存在するキーとバリューのインデックスやキーとバリューの個数，データベース内の空き領域の情報などを復元する．

4. 実装

上記の提案手法を NVMe SSD に最適化したキーバリューストアである KVell に対して実装した．KVell ではブロックデバイスへのアクセスを想定しているため，ページキャッシュを用いてページ単位で読み書きを行なっている．そのため，キーとバリューの読み出しの際には，まずページキャッシュを調べ，該当するページが存在しなかった場合にはストレージからページを読み出し，キーとバリューを返している．また，キーとバリューの更新の際には，まずページキャッシュを調べ，該当するページが存在する場合にはそのページに対して更新を行ったのち，ページをストレージに書き戻している．該当するページがページキャッシュに存在しない場合には，ストレージからページを読み出した上で上記の操作を行っている．本論文の実装では，キーとバリューの読み出しに関してはページキャッシュを用いて行うようにしている．これは，毎回永続メモリから読み出すよりも DRAM から読み出した方が速いためである．一方で，キーとバリューの更新に関してはページキャッシュを用いず，直接永続メモリに対して更新を行うようにしている．その際，CPU キャッシュを介して書き込みを行わないように non-temporal store 命令を用いている．

KVell では，アイテムサイズごとに slab という領域をストレージ上に設け，アイテムサイズが同じキーとバリューに関しては毎回同じ slab 内の同じ場所に対してキーとバリューが書き込まれる．本論文の実装では，一度キーとバリューを読み出した後に同じ場所に対してキーとバリューの更新を行うと CPU キャッシュに対して書き込みが行われ，CPU キャッシュからランダムに書き戻される際に

Write Amplification が発生するため，キーとバリューの更新は毎回 slab 内の別の場所に対して行うようにしている．slab 内の空いている場所をビットマップで管理し，キーとバリューを更新する際にはビットマップから空いている場所を選択し，そこにキーとバリューを書き込む．キーとバリューを書き込んだ後は，キーとバリューの元の場所を空いている場所としてビットマップに追加する．さらに，キーとバリューの位置が更新前と更新後で変化しているため，インデックスの更新も行う．

KVell では CPU のオーバーヘッドを削減するため，I/O 要求をバッチして非同期に I/O 処理を行い，システムコールを呼び出す回数を減らしている．本論文では永続メモリをユーザ空間にマップし，ユーザ空間から直接永続メモリに対して読み書きを行っているため，同期的に I/O を実行する．

永続メモリへの読み書きには，インテルが提供する永続メモリ向けのライブラリである PMDK[9] を用いる．永続メモリへ読み書きを行う際にはまず，永続メモリ上の領域をファイルとしてオープンし，ユーザ空間へとマップする．そして，ユーザ空間から直接読み書きを行う．ファイルのオープン，マップには pmem_mmap() を使用し，読み書きには pmem_memcpy() を用いる．non-temporal store 命令を用いて書き込みを行うためには，pmem_memcpy() の引数に PMEM_F_MEM_NONTEMPORAL を指定する．CPU キャッシュから永続メモリへの書き戻しは，pmem_persist() を用いて行う．

5. 実験

5.1 実験環境

実験に用いたマシンの仕様は以下のとおりである．CPU は，Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz を用いる．CPU のソケット数は 2 で，ソケットごとのコア数は 16 である．DRAM のサイズは 188GB である．永続メモリには，500GB の Intel Optane Persistent Memory Module を 1 枚用いる．OS は Ubuntu 20.04 LTS を用いる．

5.2 性能評価

提案手法によるキーバリューストアの性能向上を調べるために実験を行い，スループットとレイテンシを計測する．ベンチマークには，キーバリューストアの性能を測る際の標準的なベンチマークである YCSB[10] を用いる．ワークロードには YCSB_A を用い，キーとバリューの読み出しと書き込みの比率は 1 対 1 とする．データセットに関しては，アイテムサイズは 1KB とし，アイテム数は 100,000,000 とする．また，スレッド数に関しては，クライアントからのリクエストをキューに入れるスレッドの数が 16，クライアントからのリクエストを処理するスレッドの数が 16 である．また，リクエストは全てのキーに対して均等に行うようにする．ベースラインには，KVell を永続メモリ向けに最小限変更したものを用いている．具体的には，メモリ参照命令を用いて永続メモリへの読み書きを行うようにし，書き込みを行った後に pmem_persist() を実行して書き込みが永続メモリに到達するようにしている．

実験結果は以下のとおりである．まず，データベースが空の状態からアイテムを 100,000,000 個作成し，リクエストを 100,000,000 回実行した場合の結果を**図 5** に示す．既存のキーバリューストアに対し，スループットは約 2 倍，レイテンシは約 1/2 倍となった．次に，キーバリューストアを一度終了させ，再度実行し，リカバリーを行ってからリクエストを 100,000,000 回実行した場合の結果を**図 6** に示す．既存のキーバリューストアに対し，スループットは約 4 倍，レイテンシは約 1/7 倍となった．データベースが空の状態から始めた場合に対し，リカバリー後の方が性能が高くなっている理由については，現時点でははっきりとは分かっていないが，おそらくリカバリー時には一度データベースの全てのキーとバリューを走査するため，必要な情報が上手くキャッシュに乗るためではないかと推測される．

6. 関連研究

ChameleonDB[11] は，LSM-tree をベースに高い読み出し性能と書き込み性能を実現し，永続メモリに最適化したキーバリューストアである．ChameleonDB では，Intel Optane Persistent Memory は 256 バイト単位でアクセス可能なブロックデバイスであることを指摘し，書き込み性能を向上させるために LSM-tree を用いている．また，読み出し性能を向上させるために LSM-tree の最後のレベル以外のテーブルを DRAM 上にコピーし，ハッシュテーブルで管理している．したがって，読み出しの際には，DRAM 上のバッファとハッシュテーブル，LSM-tree の最後のレベルのみを探索することになり，高速に読み出しを行うことができる．

SLM-DB[12] は，永続メモリをインデックスとバッファに用い，高い読み出し性能と書き込み性能を実現したキーバリューストアである．従来通りキーとバリューは HDD や SSD などのディスクへと書き込み，ディスク上の構造には LSM-tree を用いている．SLM-DB では，LSM-tree で複数のレベルを維持することにより発生する Read Amplification と Write Amplification を削減するため，LSM-tree を一つのレベルで管理している．さらに，キーの範囲検索を効率的に行える範囲でコンパクションを限定的に行う．また，キーとバリューの読み出しを高速に行うため，永続メモリ上に B+-tree のインデックスを用意している．

KVell[8] は，NVMe SSD に最適化したキーバリューストアである．KVell では，従来のブロックデバイスへのアクセスでは最適化の方法として有効であった LSM-tree と

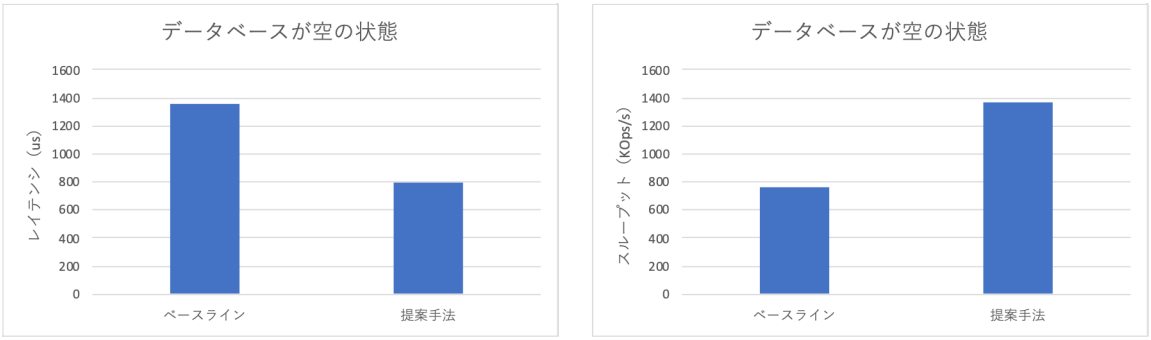

図 5: データベースが空の状態から始めた場合の実行結果

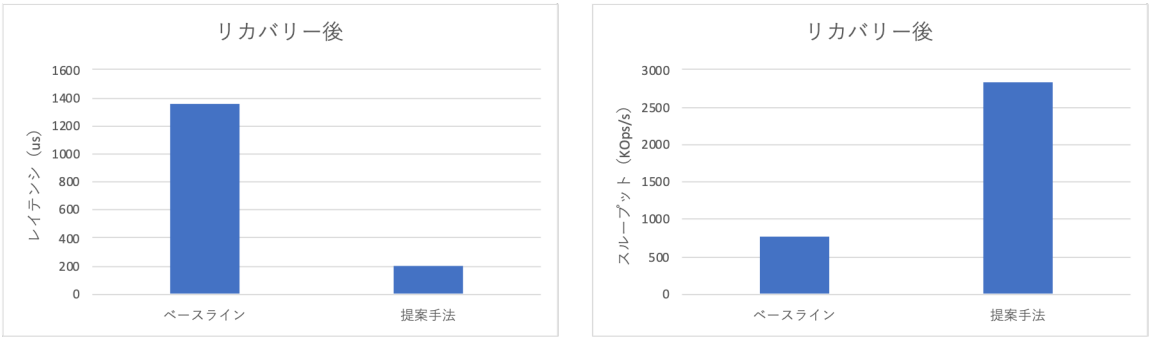

図 6: リカバリー後の実行結果

B-tree のアプローチが，NVMe SSD のような高速に動作するブロックデバイスでは逆に CPU ボトルネックとなることを示している．KVell では CPU ボトルネックを解消するために，キーによるソートは行わず，ディスク上のランダムな位置にキーとバリューを書き込むようにしている．また，キーのハッシュ値によってデータ構造を分割し，複数のスレッドが同期をとらずに操作を行えるようにしている．さらに，複数の I/O 要求をバッチして非同期に処理を行うことでシステムコールを呼び出す回数を減らしている．そして，コミットログを排除し，書き込みにかかるオーバヘッドを削減している．

7. 結論

本論文では，バイト単位でアクセス可能な永続メモリに最適化したキーバリューストアの実装手法について提案した．従来のブロックデバイスに最適化したキーバリューストアでは，シーケンシャルアクセスのための最適化を行い，ページキャッシュを用いてページ単位で読み書きを行っている．本論文の提案手法では，永続メモリへはブロック単位ではなくバイト単位でアクセスするようにし，キーとバリューの更新の際には CPU キャッシュを介さずに直接永続メモリに対して更新を行うようにした．

上記の提案手法を NVMe SSD に最適化したキーバリューストアの一つである KVell に対して実装し，実験を行った．実験結果では，write-intensive なワークロードにおいて既存のキーバリューストアに対し，スループットは約 2–4 倍，レイテンシは約 1/2–1/7 倍となり，上記の提案手法の有効性が示された．

参考文献

[1] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A. and Gruber, R. E.: Bigtable: A Distributed Storage System for Structured Data, *ACM Trans. Comput. Syst.*, Vol. 26, No. 2, pp. 1–26 (2008).

[2] Facebook: RocksDB, Facebook (online), available from ⟨https://rocksdb.org⟩ (accessed 2021-06-14).

[3] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P. and Vogels, W.: Dynamo: amazon's highly available key-value store, *Oper. Syst. Rev.*, Vol. 41, No. 6, pp. 205–220 (2007).

[4] Beaver, D., Kumar, S., Li, H. C., Sobel, J., Vajgel, P. and Others: Finding a Needle in Haystack: Facebook's Photo Storage, *OSDI*, Vol. 10, pp. 1–8 (2010).

[5] Intel: Intel Optane Persistent Memory, Intel (online), available from ⟨https://www.intel.co.jp/content/www/jp/ja/architecture-and-technology/optane-dc-persistent-memory.html⟩ (accessed 2021-06-20).

[6] Intel: Intel Optane SSD DC P4800X Series, Intel (online), available from ⟨https://www.intel.co.jp/content/www/jp/ja/products/sku/129969/intel-optane-ssd-dc-d4800x-series-375gb-2-5in-pcie-2x2-3d-xpoint/specifications.html⟩ (accessed 2021-06-17).

[7] Yang, J., Kim, J., Hoseinzadeh, M., Izraelevitz, J. and Swanson, S.: An empirical guide to the behavior and use of scalable persistent memory, *18th USENIX Conference on File and Storage Technologies (FAST 20)*, pp. 169–182 (2020).

[8] Lepers, B., Balmau, O., Gupta, K. and Zwaenepoel, W.:

KVell: the design and implementation of a fast persistent key-value store, *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, SOSP '19, New York, NY, USA, Association for Computing Machinery, pp. 447–461 (2019).

[9] Intel: Persistent Memory Development Kit, Intel (online), available from ⟨https://pmem.io/pmdk/⟩ (accessed 2021-06-24).

[10] Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R. and Sears, R.: Benchmarking cloud serving systems with YCSB, *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, New York, NY, USA, Association for Computing Machinery, pp. 143–154 (2010).

[11] Zhang, W., Zhao, X., Jiang, S. and Jiang, H.: ChameleonDB: a key-value store for optane persistent memory, *Proceedings of the Sixteenth European Conference on Computer Systems*, Association for Computing Machinery, New York, NY, USA, pp. 194–209 (2021).

[12] Kaiyrakhmet, O., Lee, S., Nam, B., Noh, S. H. and Choi, Y.-R.: SLM-DB: single-level key-value store with persistent memory, *17th USENIX Conference on File and Storage Technologies (FAST 19)*, pp. 191–205 (2019).