不揮発メモリを対象とする空間索引のSSDとの性能比較

吉岡　弘隆†　　合田　和生††　　喜連川　優††

† 東京大学　大学院情報理工学系研究科
〒 153–8505 東京都目黒区駒場 4–6–1
†† 東京大学　生産技術研究所
〒 153–8505 東京都目黒区駒場 4–6–1
E-mail: †{hyoshiok,kgoda,kitsure}@tkl.iis.u-tokyo.ac.jp

あらまし
概要：不揮発メモリ (PMEM) は揮発性メモリ (DRAM) とほぼ同等のレイテンシーを持ちながら HDD/SSD 同様永続性を持つ．不揮発メモリへの書き込みは OS が非同期で行うが一貫性を担保したい場合は永続性を持たせるために明示的な flush および同期 (fence) が必要となる．空間索引の性能比較を従来デバイス (SSD) と不揮発メモリとで行った．不揮発メモリの基本的な動作特性 (レイテンシー，スループット等) や不揮発メモリ向けの基本的なデータ構造 (B+木やハッシュ索引) とアルゴリズムについては様々な研究がされてきている．しかしながら不揮発メモリを対象とする空間検索構造の実装方式の報告については FBR [11]，PMR [26] などに見られる程度で，十分とは言えない．そこで本研究では不揮発メモリを対象とする空間検索構造の実装方式の検討を行い，従来デバイス (SSD) と不揮発メモリで性能比較を行った．その結果，挿入については SSD より PMEM のほうが 22 倍以上のスループットが高かった．しかしメモリマッピングした場合の検索には顕著な差はみられなかった．

キーワード　データベース技術，不揮発メモリ，PMEM，SSD，空間索引構造

1　は じ め に

不揮発メモリと呼ばれるバイト単位アクセス可能なメモリが近年出荷されている [39], [41].

その特徴をまとめると，

- 性能（スループット，レイテンシなど）は SSD (NAND Flash)，HDD より高い，DRAM より低い
- DRAM と異なり永続性がある（電源が遮断されても情報は喪失しない）
- 記憶容量は DRAM より大きい
- バイト単位のアクセスが可能である（DRAM と同様）
- CPU cache coherent である（DRAM と同様）
- DMA （Direct Memory Access）と RDMA （Remote Direct Memory Access）をサポートする（DRAM と同様）
- ユーザ空間でアクセスできる．SSD や HDD のようにアクセスするために system call は必要ない．アクセスするためにカーネルコード，ページキャッシュ，割込などは必要ない

となる．

様々な興味深い特徴があるが，不揮発メモリの製品例である Intel Optane DCPMM も広く普及しているとは言い難く，研究段階での評価にとどまっているのが現状である．

ここでは不揮発メモリの製品例として Intel Optane DCPMM をとりあげ評価する．

不揮発メモリの特徴として SSD や HDD より高速だといわれている．そこで本研究ではまず PMEM と SSD の挿入時のスループットを計測しその性能を確認した．これによって PMEM が SSD より高性能な記憶デバイスとして利用できることを示した．

一方で，従来の揮発性メモリと違い，不揮発であるという特徴はプログラミングモデルの変更を余儀なくする．そのため，その動作特性の違いは単にスループットやレイテンシーだけでなく，アルゴリズムやデータ構造のあり方にも影響を与える．

不揮発メモリは従来のメモリ同様にバイト単位でアクセス可能であるが，CPU から見て不揮発メモリへのアクセスは通常キャッシュ経由でなされる．図 1 に示すようにキャッシュは揮発性なので永続性を必要とする場合は適宜キャッシュ内容を flush する必要がある．Intel x86 の場合，CLFLUSH 命令などによって明示的に行う．flush 関連の命令は CLFLUSH の他，CLWB，CLFLUSHOPT があり，キャッシュをバイパスする Non-temporal 命令もある．また，メモリへの書き込みを同期するために fence が必要になる．

また記憶の永続性を担保する動作についてはファイルシステムでは本質的な問題だが，メインメモリに於いては永続性は存在しなかったので一般的な問題ではなかった．例えばクラッシュコンシステンシーという概念は従来は主にハードディスクなど永続性を持つデバイスにおいて議論されてきたが，揮発性メモリではクラッシュした時点でそもそも永続性を保証しないので一般的には問題とならなかった．しかし不揮発メモリに於いてはストア命令によって記憶が永続化されるので，ディスクへの書き込みと同様な問題が生じる．ここで言うクラッシュコンシステンシーとは，情報の変更を首尾一貫した状態で行うことを指す．

不揮発メモリへのストア命令は必ずしも同期的にメディアに

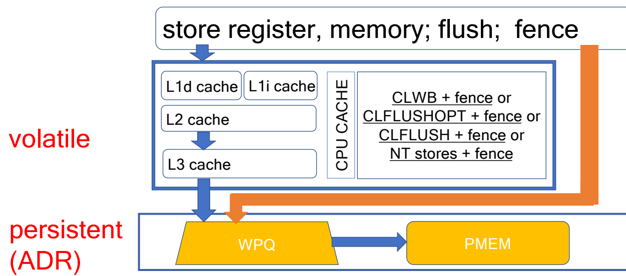

図 1　CPU とキャッシュ，メモリの関係

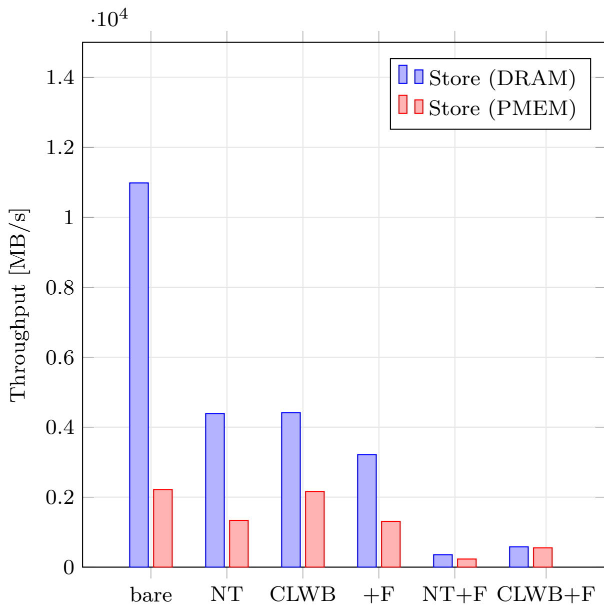

図 2　64 byte sequenial store, bare, NT, CLWB はデフォルトの store 命令および Non-temporal 命令，CLWB 命令を追加したもの，＋ F はさらに FENCE 命令を追加したものを示す [53]

書き込むことを保証しない，通常はキャッシュにのみ書き込み，非同期にメモリ (メディア) に書き込む．そのため，プログラムのストアの順序とメモリへの書き込み順は必ずしも等しくならず，書き込み順の一貫性を担保する仕組みが必要になる．

メモリレベルでの永続性を担保する命令（FLUSH 命令）や書き込み順の同期を取る命令（FENCE 命令）などをプログラマが意識しないと不揮発メモリを利用した正しいプログラム（クラッシュした時点での整合性を持つ）は構築できない．

またこのような各種の flush や fence が性能に与える影響は必ずしも自明ではない．そこでわれわれは，先行研究として pmmeter というマイクロベンチマークを作成し，flush 命令と同期命令が与えるオーバーヘッドを評価した [53]．64 バイトのシーケンシャルアクセスのスループットは DRAM で 10.9GB/s, PMEM で 2.2GB/s だが flush 命令を追加すると 4.4GB/s と 2.1GB/s にそれぞれ低下する．さらに fence 命令を追加すると 0.58GB/s と 0.55GB/s となる (図 2 参照)．flush と fence は大きなコストが発生する．

そのような特性を踏まえ，従来の広く利用されている索引技術を不揮発メモリへ適用する研究が近年活発に行われている．

例えば B+木索引はメモリアクセスとディスクアクセスのレイテンシの大きな違いを前提にファイルをディスクに格納した場合に最適化された構造で，検索時にディスクアクセス回数を削減するようなアルゴリズムになっている．

不揮発メモリ向けの B+木索引 [16], [18], [29] ，ハッシュ索引 [17]，ファイルシステム [38]，データベースエンジン，基本的な動作特性 [37], [46] などの報告はあるが，空間索引構造についてはまだ十分に検討がされていない．そこで本研究では，不揮発メモリを対象とする空間検索構造の実装方式を検討した．

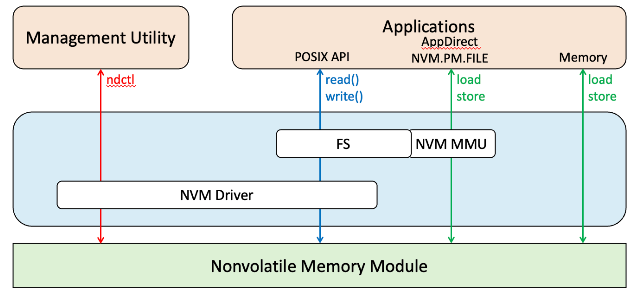

図 3　PMEM プログラミングモデル

1.1　PMEM プログラミングモデル

上述したように，PMEM のプログラミングは必ずしも自明ではない．例えば，メモリレベルでの永続性を担保する命令（FLUSH 命令）や書き込み順の同期を取る命令（FENCE 命令）などをプログラマが意識しないと不揮発メモリを利用した正しいプログラム（クラッシュした時点での整合性を持つ）は構築できない．

このように不揮発メモリは揮発性メモリとは異なったプログラミングモデルを持つために従来とは異なるプログラミング上の注意が必要となる．一般的には，上記のような煩雑な処理をアプリケーションプログラマに委ねるのはアプリケーションをアセンブリ言語で記述するようなものなので，動作の詳細を隠蔽するためにベンダーが提供する PMDK (Persistent Memory Developers Kit) [19] などのライブラリを利用することになる．

アプリケーション・プログラムないしシステム・ソフトウェアからみた NVM (Non Volatile Memory) のプログラミングモデルはストレージの業界団体である SNIA [43] が NVM Programming Model Specification として定義し公開している．図 3 で示した．

これらの機能は Linux ないし Windows など複数の OS によってすでに実装されている．Linux の XFS や ext4 ファイルシステムなどは DAX モード (AppDirect モード) をサポートしていて，mmap() と同様の機能を持つライブラリコールを利用することによって，ファイルの内容を直接ユーザーメモリ空

間にマッピングすることができる．一度メモリ空間にマップされると OS の提供する I/O 機能，すなわち read()/write() システムコールなどを利用しなくても直接 NVM を利用することができる．NVM への読み書きは load/store で行う．Intel x86 アーキテクチャの場合，通常の MOV 命令になる．この場合，システムコールが必要ないので，コンテキストスイッチ，割り込みなどが発生しないため，低いレイテンシーでメモリにアクセスできる．

一方で，従来の揮発性メモリと違い，不揮発であるという特徴はプログラミングモデルの変更を余儀なくする．そのため，その動作特性の違いは単にスループットやレイテンシーだけでなく，アルゴリズムやデータ構造のあり方にも影響を与える．

不揮発メモリは従来のメモリ同様にバイト単位でアクセス可能であるが，CPU から見て不揮発メモリへのアクセスは通常キャッシュ経由でなされる．図 1 に示すようにキャッシュは揮発性なので永続性を必要とする場合は適宜キャッシュ内容を flush する必要がある．Intel x86 の場合，CLFLUSH 命令などによって明示的に行う．flush 関連の命令は CLFLUSH の他，CLWB，CLFLUSHOPT があり，キャッシュをバイパスする Non-temporal 命令もある．また，メモリへの書き込みを同期するために fence が必要になる．

以下に論文の構成をしめす．第 2 章でまず，空間索引構造を紹介し，それを不揮発メモリに適応するときの課題を整理する．そして，第 3 章で今回行った実装方式の検討についてふれ，その実験で明らかになった問題を議論する．第 4 章で関連研究を紹介し，最後にまとめと今後の課題を述べる．

2 空間索引構造

2.1 R-Tree

多次元データを高速に検索する方法として空間索引構造が知られている．その例としてここでは R-Tree を取り上げる (図 4 参照)． R-Tree [14] は Guttman によって提案された空間索引構造であり，次のような特徴を持つ．

- R-tree の leaf ノードは (I, tuple 識別子) という形式で，I は多次元のオブジェクト，tuple 識別子は当該オブジェクトを一意に識別する．
- leaf ノードでない場合 (non-leaf ノード) は，(I, 子ノードのポインタ) という形式である．
- 子ノードは leaf ノードか non-leaf ノードである．
- 各 leaf ノードはルートノードでなければ m から M 個の索引レコードを持つ．
- leaf ノードの各索引レコードは n 次元の最小矩形である．
- ルートノードは leaf ノードでないかぎり 2 個の子ノードを持つ．
- 全ての leaf は等しいレベルである（leaf ノードまでの高さは等しい）

図 4 では，ルートノードが R1 と R2 の子ノードを持ち，R1 は R3,R4,R5 の子ノードを持つ，R3 は同様に R8,R9,R10 を持ち，それらは leaf ノードになる．leaf ノードは多次元オブジェクトデータと tuple 識別子を持つ．B+tree と同様に，leaf ノードにデータを格納していくと，leaf までの高さを一定に保つために適宜分割などが発生する．

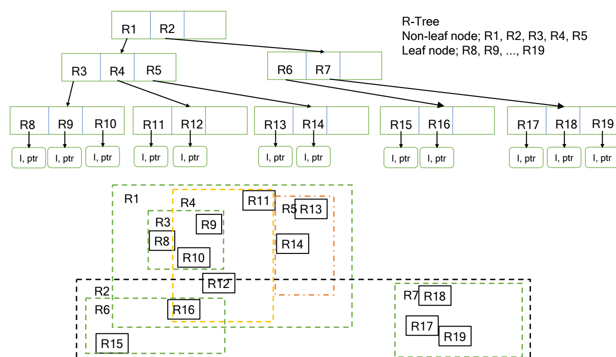

図 4　R-Tree の例

表 1　実 験 環 境

CPU model	Intel Xeon Silver, 2.5GHz
	8 core, 2 socket
No. of nodes	2
Cache	L1d 32KiB, L1i 32 KiB,
	L2 1 MiB, L3 11 MiB (shared)
DRAM	32 GiB * 12 (384 GiB)
DCPMM	128 GiB *12 (1536 GiB)
OS	CentOS 7.7.1908, linux kernel 3.10
PMDK	1.8

R-Tree そのものにはトランザクションの概念を含まないので同時実行制御等の機能は定義されていないが，データベースなどで利用する場合には検討する必要がある．

R-Tree そのものについては, [34] という参考書があり，Guttaman の提案以降の主要な提案が網羅されている．ただし出版年が 2006 年なので，近年のハードウェア動向については考慮されていない．

2.2 不揮発メモリのプログラミングモデル

不揮発メモリをストレージデバイスとして利用するときにはいくつかの注意すべき点がある．従来のプログラミングで永続性を確保したい場合は通常ファイルシステムないしデータベースなどを利用しておこなっているが，不揮発メモリの場合，ファイルシステムなどの利用は不必要となるが下記のようなプログラミングモデルとなるので従来型プログラムを変更する必要がある．

2.2.1 明示的な情報の永続化

不揮発メモリは情報をメモリにストアすると永続化されるが，そのタイミングは一般には非同期であり，通常はキャッシュに書き込まれるだけでメディアへの書き込みタイミングはプログラマは意識しない (図 1 参照)．また書き込む順も不定である．そのため，明示的にキャッシュからフラッシュし，書き込み順を同期する必要がある．書き込み順を指定しないと，一貫性を保てない状態が発生する場合がある．例えば，あるページへのポインタがあった場合，あるページへの書き込みが終了する前

に，そのページへのポインタを設定した場合，まだ情報が書き込まれていない場所への参照が発生してしまい，そのタイミングでシステムがクラッシュすると一貫性が保持されないままになる．最近のプロセッサは out of order で実行するために，プログラムを書いた順番に実行されるとは限らないので注意が必要である．Intel Optane DC Persistent Memory Module (以下 Intel DCPMM と称する) の場合，命令ごとの同期をとるために，明示的な FENCE 命令が必要になる．

2.2.2　アトミック書き込みは最大 8 バイト

Intel DCPMM がアトミックに書き込めるのは最大 8 バイトである．それ以上大きな塊での書き込みはアトミックに行われない．そのため書き込み途中でシステムがクラッシュすると一貫性が保持されない．R-Tree の木構造の変更はアトミックに行う必要がある．

上記のような問題に対処するため，不揮発メモリを対象とした B+Tree や Hash などで様々な提案がなされている．(主なサーベイとして B+木索引では [16], [18], [29] ，ハッシュ索引 [17] などがある)．

空間索引については, [11] (以下 FBR と称する) および [26] (以下 PMR と称する) などが上記の問題に対応している．

多次元空間索引はデータベース分野のみならずコンピュータグラフィックス，位置情報システム，高性能システム (High Performance Computing) など様々な分野での応用が期待される．科学技術計算分野でのデータ量の爆発は不揮発メモリの応用分野としても期待されている．

2.2.3　クラッシュコンシステンシの実装

前述したように不揮発メモリにおいては単なるメモリへの転送だけでは一貫した状態を保持できない．なんらかの方法でクラッシュコンシステンシを確保する必要がある．

R-Tree の場合，B+Tree と同様にページへのレコードの追加時にページ分割が発生する場合があり，それは leaf ノードから root ノードまで波及する場合がある．木構造を変更するためにアトミックに変更する必要があり，その一貫性制御が必要となる．

先行研究 FBR [11] ではクラッシュコンシステンシを確保するために，mutex lock を利用した実装を評価した．一方，PMR [26] では mutex lock ではなく不揮発メモリ向け lock free アルゴリズム (Persistent Multi-word Compare and Swap 以下 PMwCAS と称する) [45] を利用した実装を評価した．また複数バイトへのアトミックな CAS (Multi-word Compare and Swap 以下 MwCAS と称する) [15] を利用した．

FBR は木構造の変更は木構造の root ノードに対するジャイアントロックを mutex lock を排他的に取得することで行っている．PMR はそれを PMwCAS と MwCAS で実装した．

3　実　　　験

3.1　ハードウェア，ソフトウェア構成

次のような環境で実験した (表 1)．CPU は Intel Xeon Silver, 2.5Ghz, 8 core, 2 socket, Numa 2 ノード，OS は CentOS 7.7 Linux Kernel 3.10, PMDK 1.8 を利用した．

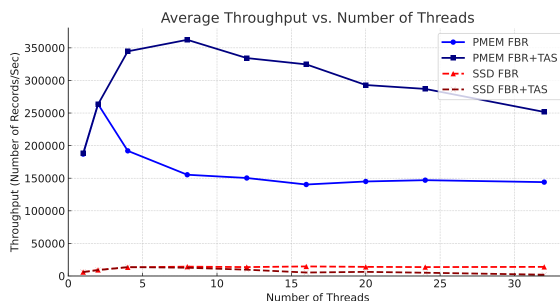

図 5　FBR，FBR+TAS，50 万件データの挿入，スレッド数を変化させる．縦軸は秒あたりの挿入件数 PMEM と SSD の差

3.2　実験方法と結果

FBR をベースラインとして，ランダムに生成した 50 万件の空間データを R-Tree に挿入する実験を行った．ソースコードは以下を利用した．`https://github.com/DICL/FBR-tree`

結果を図 5 に示した．データベースは PMDK を利用して不揮発メモリ上に生成して APP ダイレクトモードでメモリ上にマッピングしている．ランダムに生成した 50 万件の空間データをデータベースに挿入するスレッド数を 1, 2, 4, 8, 12, 16, 20, 24, 32 と変化させ，そのスループットを計測した．横軸にスレッド数，縦軸に秒あたりの挿入件数をとった．

まず，PMEM 領域にデータファイルを置きそこに挿入する時間を測定した．ベースラインの FBR アルゴリズム (図 5 の PMEM FBR の線，以下同様) は 1 スレッドのとき PMEM へ 1 秒あたり 19 万レコード弱挿入し，2 スレッドで 1 秒あたり 26 万レコード弱挿入しピークになる．その後スレッド数を増やしても 1 秒あたり 14 万レコード前後挿入する．次に SSD への挿入は (SSD FBR) で示される通り 1 スレッドのとき 6 千件/秒程度である．PMEM のほうが 30 倍以上速い．

FBR の場合，スレッド数を 2 より増やしても挿入件数は増えず，スケーラビリティがないことを確認した．

FBR の実装は mutex-lock を利用して R-Tree のノードの挿入の同期を取っている．R-Tree に対するジャイアントロックのため，レコードを挿入するとき，待ちが発生し，スケールしないと考えられる．そこで実験的に木全体を mutex-lock を利用してロックするのではなく Node に対する Test And Set (TAS) を利用して同期を取る方法を実装してみた (図 5PMEM FBR+TAS)．それによると 8 スレッドまで秒あたりの挿入件数が増えその後は徐々に減っていった．(188K 件/秒から 362K 件/秒)

FBR による検索の結果を図 6 に示した．予めレコードを 10 万件挿入したデータベースを用意して，それを PMDK の pmemobj_open()　API を利用してオープンしメモリにマッピングして，ランダムに 1 万件 (図 6 10K)，2 万件 (図 6 20K)，4 万件 (図 6 40K) 検索した，1 秒あたりの検索件数を計測した．検索スレッド数 32 まで増やした．スレッド数に比例して秒あたりの検索件数が増加した．

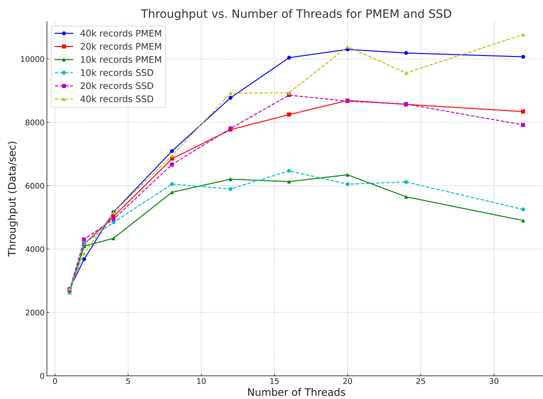

図 6　FBR 検索，PMEM と SSD の比較，横軸はスレッド数，縦軸は秒あたりの検索件数，データファイルへ 50 万件挿入後，1 万件 (10K)，2 万件 (20K)，4 万件 (40K) 検索

すでに存在するデータファイルを memory mapping するのでどちらも memory 上へのアクセスになる．したがって検索のスループットは SSD 上のファイルの場合も PMEM 上のファイルの場合もほぼ同様のスループットになる．PMEM の場合 20 スレッドあたりでピークになった．SSD も同様の傾向ではあるが 4 万件の検索の場合 32 スレッドがピークになった．

FBR のスケーラビリティの問題は挿入時に R-Tree のルートノードに対する mutex lock で木全体をロックしてしまうことにあると考えられる．さらにスケーラビリティを向上させるためにはなんらかの方法でボトルネックを解消する必要がある．

3.3　実験のまとめ

FBR について追試した．また，FBR の実装について mutex-lock のかわりに Test and Set (TAS) を利用した簡易的なロックフリーメカニズムを導入し評価したところ，スケーラビリティの向上がみられた．

PMEM のかわりに SSD に R-Tree をおいて評価したところ，挿入のスループットが 2.6 %〜10.9 %程度になった．一方で検索のスループットはデータファイルを PMEM においた場合と SSD においた場合の差異はほとんど生じなかった．これはデータファイルをメインメモリ上にメモリマッピングするため，一度マッピングされるとスループットはほぼ等しくなる．

4　関 連 研 究

Intel Opetane DCPMM そのものの性能評価は [22], [44] がある．多くの実装研究は実機がなかったためシミュレーションによって評価をしていた [1]．近年，Intel Optane DCPMM の出荷に伴い，徐々にその性能特性について評価が発表されてきている．汎用的なベンチマークのフレームワークは [4] にみられる．第 2 世代 Intel Optane(特に ADR(Asynchronous DRAM Refresh) および eADR (Extended ADR)) の評価を行っている．ADR および eADR の概要は [20] にある．

先行研究の提案を実機で検証しシミュレーション結果との差分を報告している [50]．Intel Optane DCPMM のレイテンシとスループットだけではなく，電力消費量の評価しているのは [37] である．Intel の NVM 向けライブラリ（NVML）についての初期の評価は [42] にある．当該ライブラリは現在，pmdk (persistent memory development kit) [19] になった．

既存のデータ構造を NVM に拡張し評価したものとして [27] などがある．B^+Tree 及びその NVM 拡張系の提案として，Persistent B^+-Trees [7]，FPTree [36]，BzTree [1]，また索引データ構造の比較検討 [21], [29] は従来提案されていたものを DCPMM で検証し，その有効性などを定量的に確認している．NVM 向けのファイルシステム NOVA [49]，Hikv(KVS) [48]，DBMS の実装 [2]，Write-behind logging [3] などがある．データベースへの適用について [25] は異なる database engine(PostgreSQL, MySQL，SQLServer，DuckDB，VoltDB，RockDB) について様々なワークロード (OLAP，OLTP，YCSB)、構成 (SSD, AppDir，Memory mode) で評価している．

High Performance Computing 分野での評価は [46] があり大規模アプリケーションでの性能を報告している．

不揮発メモリ向けの範囲索引について [29], [16], [17] などが様々な提案の比較を行った．ハッシュ索引 [17]，ファイルシステム [38]，基本的な動作特性 [46], [37]，などの報告がある．不揮発メモリ向けの範囲索引について [29] は BzTree [1]，FPTree [36], NV-Tree [51], [52]，wBTree [6]，について評価している. [16] は, [29] で取り上げられなかった以下の範囲索引 LB+-Tree [31], uTree [8]，DPTree [55]，ROART [33]，PACTree [24] をそれぞれ評価した．主に Intel Optane DC Persistent Memory Module 出荷後に提案されたものを実機で評価している．

同様に [17] はハッシュ索引を比較し，Level hashing [56]，Clevel hashing [9]，CCEH [35]，Dash [32]，PCLHT [27], SOFT [57] を評価した. [10] は様々なハッシュ索引を定量的に評価した．ファイルシステム [38]，データベースエンジン [54]，不揮発メモリ向けのデータ構造 [13]，それぞれのカテゴリで比較検討している．Intel Optane DCPMM の性能評価は [37], [46] にある. [30] は不揮発メモリを大規模メインメモリとして利用する様々なアプリケーションについて調査している．

NUMA への PMEM の適用は [23] にある．

不揮発メモリ向け R-Tree の研究は近年徐々に増えてきている. [40] は FBR のスケーラビリティの欠如の問題について検討し，many core マシンを対象に，MPR-Tree を提案した．NUMA をサポートし更新のスケーラビリティを向上させた．

SSD(Flash storage) 対応の R-tree の研究として [47] と [12] がある．前者は SSD 向けの最適化について，後者は不揮発メモリと SSD のハイブリッドな構成での最適化について検討している．空間索引構造についてはまだ十分に検討がされていない．そこで本研究では，不揮発メモリを対象とする空間検索構造の実装方式を検討し実験をおこなった．

PMwCAS [45] は PMEM 向けの multi-word CAS (compare-and-swap) である。B+Tree の兄弟ノードにリンクを張った B-link 木は [28] で提案されている。latch-free index (OLFIT) は [5] で提案されている。

5 まとめと今後の課題

不揮発メモリを対象とする空間索引構造の実装方式の検討を行った．まず空間索引構造を PMEM に置く場合と SSD に置く場合の性能の比較を行った．

FBR の実装を確認し，マルチスレッド環境での挿入および検索のスケーラビリティを確認した．その結果，挿入は R-Tree へのコンテンションが発生し，スループットはスケールしないことが確認された．簡易的な Test and Set (TAS) を利用したロックフリーの実装を FBR に適応したところ，マルチスレッド環境で 8 スレッド程度までスループットが向上した．

次に SSD の挿入のスループットを確認し，PMEM と比較して 3.2 %程度のスループットだった．検索に関してはメモリマッピングをすることによって PMEM 上にデータファイルを置く場合も，SSD 上にデータファイルを置く場合もほぼ同様のスループットが得られた．

今後は PMR の未実装の部分 (mutex lock ではなく PMw-CAS によるロックフリー同期 [45]) についても追試をし不揮発メモリを対象とする空間索引構造の拡張の提案および実装を行いたい. [26]

文 献

[1] Joy Arulraj, Justin Levandoski, Umar Farooq Minhas, and Per-Ake Larson. Bztree: A high-performance latch-free range index for non-volatile memory. *Proceedings of the VLDB Endowment*, Vol. 11, No. 5, pp. 553–565, 2018.

[2] Joy Arulraj and Andrew Pavlo. How to build a non-volatile memory database management system. In *Proc' of the 2017 ACM SIGMOD*, p. 1753–1758, 2017.

[3] Joy Arulraj, Matthew Perron, and Andrew Pavlo. Write-behind logging. *Proc' of the VLDB Endowment*, Vol. 10, No. 4, pp. 337–348, 2016.

[4] Lawrence Benson, Leon Papke, and Tilmann Rabl. PerMA-Bench: Benchmarking persistent memory access. *Proc. VLDB Endow.*, Vol. 15, No. 11, p. 2463–2476, sep 2022.

[5] Sang Kyun Cha, Sangyong Hwang, Kihong Kim, and Keunjoo Kwon. Cache-conscious concurrency control of main-memory indexes on shared-memory multiprocessor systems. In *VLDB*, Vol. 1, pp. 181–190, 2001.

[6] Shimin Chen and Qin Jin. Persistent b+-trees in non-volatile main memory. *Proceedings of the VLDB Endowment*, Vol. 8, No. 7, pp. 786–797, 2015.

[7] Shimin Chen and Qin Jin. Persistent b^+-trees in non-volatile main memory. *Proc. VLDB Endow.*, Vol. 8, No. 7, p. 786–797, February 2015.

[8] Youmin Chen, Youyou Lu, Kedong Fang, Qing Wang, and Jiwu Shu. utree: a persistent b+-tree with low tail latency. *Proceedings of the VLDB Endowment*, Vol. 13, No. 12, pp. 2634–2648, 2020.

[9] Zhangyu Chen, Yu Hua, Bo Ding, and Pengfei Zuo. Lock-free concurrent level hashing for persistent memory. In *Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Conference*, pp. 799–812, 2020.

[10] Zhiwen Chen, Daokun Hu, Wenkui Che, Jianhua Sun, and Hao Chen. A quantitative evaluation of persistent memory hash indexes. *The VLDB Journal*, pp. 1–23, 2023.

[11] Soojeong Cho, Wonbae Kim, Sehyeon Oh, Changdae Kim, Kwangwon Koh, and Beomseok Nam. Failure-atomic byte-addressable r-tree for persistent memory. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 32, No. 3, pp. 601–614, 2020.

[12] Athanasios Fevgas, Leonidas Akritidis, Miltiadis Alamaniotis, Panagiota Tsompanopoulou, and Panayiotis Bozanis. A study of r-tree performance in hybrid flash/3dxpoint storage. In *2019 10th International Conference on Information, Intelligence, Systems and Applications (IISA)*, pp. 1–6. IEEE, 2019.

[13] Philipp Götze, Arun Kumar Tharanatha, and Kai-Uwe Sattler. Data structure primitives on persistent memory: an evaluation. In *Proceedings of the 16th International Workshop on Data Management on New Hardware*, pp. 1–3, 2020.

[14] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pp. 47–57, 1984.

[15] Timothy L Harris, Keir Fraser, and Ian A Pratt. A practical multi-word compare-and-swap operation. In *International Symposium on Distributed Computing*, pp. 265–279. Springer, 2002.

[16] Yuliang He, Duo Lu, Kaisong Huang, and Tianzheng Wang. Evaluating persistent memory range indexes: Part two. *arXiv preprint arXiv:2201.13047*, 2022.

[17] Daokun Hu, Zhiwen Chen, Jianbing Wu, Jianhua Sun, and Hao Chen. Persistent memory hash indexes: An experimental evaluation. *Proc. VLDB Endow.*, Vol. 14, No. 5, p. 785–798, mar 2021.

[18] Kaisong Huang, Yuliang He, and Tianzheng Wang. The past, present and future of indexing on persistent memory. *Proc. VLDB Endow.*, Vol. 15, No. 12, p. 3774–3777, sep 2022.

[19] Intel. Persistent Memory Development Kit. https://pmem.io/pmdk/, 2019.

[20] Intel. eADR: New Opportunities for Persistent Memory Applications. https://www.intel.com/content/www/us/en/developer/articles/technical/eadr-new-opportunities-for-persistent-memory-applications.html, 2021.

[21] Abdullah Al Raqibul Islam, Anirudh Narayanan, Christopher York, and Dong Dai. A performance study of optane persistent memory: From indexing data structures' perspective. In *36th (MSST 2020)*, pp. 2–2, 2020.

[22] Joseph Izraelevitz, Jian Yang, Lu Zhang, Juno Kim, Xiao Liu, Amirsaman Memaripour, Yun Joon Soh, Zixuan Wang, Yi Xu, et al. Basic performance measurements of the intel optane DC persistent memory module. *arXiv:1903.05714*, pp. 1–61, 2019.

[23] Safdar Jamil, Abdul Salam, Awais Khan, Bernd Burgstaller, Sung-Soon Park, and Youngjae Kim. Scalable numa-aware persistent b+-tree for non-volatile memory devices. *Cluster Computing*, Vol. 26, No. 5, pp. 2865–2881, 2023.

[24] Wook-Hee Kim, R Madhava Krishnan, Xinwei Fu, Sanidhya Kashyap, and Changwoo Min. Pactree: A high performance persistent range index using pac guidelines. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pp. 424–439, 2021.

[25] Dimitrios Koutsoukos, Raghav Bhartia, Michal Friedman, Ana Klimovic, and Gustavo Alonso. Nvm: Is it not very meaningful for databases? *Proceedings of the VLDB Endowment*, Vol. 16, No. 10, pp. 2444–2457, 2023.

[26] Brandon Lavinsky and Xuechen Zhang. Pm-rtree: A highly-efficient crash-consistent r-tree for persistent memory. In *Proceedings of the 34th International Conference on Scientific and Statistical Database Management*, pp. 1–11, 2022.

[27] Se Kwon Lee, Jayashree Mohan, Sanidhya Kashyap, Taesoo

Kim, and Vijay Chidambaram. Recipe: converting concurrent DRAM indexes to persistent-memory indexes. In *Proc' of 27th ACM SOSP*, pp. 462–477, 2019.

[28] Philip L. Lehman and s. Bing Yao. Efficient locking for concurrent operations on b-trees. *ACM Trans. Database Syst.*, Vol. 6, No. 4, p. 650–670, dec 1981.

[29] Lucas Lersch, Xiangpeng Hao, Ismail Oukid, Tianzheng Wang, and Thomas Willhalm. Evaluating persistent memory range indexes. *Proc' of the VLDB Endowment*, Vol. 13, No. 4, pp. 574–587, 2019.

[30] Hai-Kun Liu, Di Chen, Hai Jin, Xiao-Fei Liao, Binsheng He, Kan Hu, and Yu Zhang. A survey of non-volatile main memory technologies: State-of-the-arts, practices, and future directions. *Journal of Computer Science and Technology*, Vol. 36, No. 1, pp. 4–32, 2021.

[31] Jihang Liu, Shimin Chen, and Lujun Wang. Lb+ trees: optimizing persistent index performance on 3dxpoint memory. *Proceedings of the VLDB Endowment*, Vol. 13, No. 7, pp. 1078–1090, 2020.

[32] Baotong Lu, Xiangpeng Hao, Tianzheng Wang, and Eric Lo. Dash: Scalable hashing on persistent memory. *arXiv preprint arXiv:2003.07302*, 2020.

[33] Shaonan Ma, Kang Chen, Shimin Chen, Mengxing Liu, Jianglang Zhu, Hongbo Kang, and Yongwei Wu. Roart: Range-query optimized persistent art. In *FAST*, pp. 1–16, 2021.

[34] Yannis Manolopoulos, Apostolos N Papadopoulos, Apostolos N Papadopoulos, and Yannis Theodoridis. *R-Trees: Theory and Applications: Theory and Applications*. Springer Science & Business Media, 2006.

[35] Moohyeon Nam, Hokeun Cha, Young-ri Choi, Sam H Noh, and Beomseok Nam. Write-optimized dynamic hashing for persistent memory. In *FAST*, Vol. 19, pp. 31–44, 2019.

[36] Ismail Oukid, Johan Lasperas, Anisoara Nica, Thomas Willhalm, and Wolfgang Lehner. FPTree: A hybrid scm-dram persistent and concurrent b-tree for storage class memory. In *Proc' of the 2016 SIGMOD*, pp. 371–386, 2016.

[37] Ivy B. Peng, Maya B. Gokhale, and Eric W. Green. System evaluation of the intel optane byte-addressable nvm. In *Proceedings of the International Symposium on Memory Systems*, MEMSYS '19, p. 304–315, New York, NY, USA, 2019. Association for Computing Machinery.

[38] Gianlucca O. Puglia, Avelino Francisco Zorzo, César A. F. De Rose, Taciano Perez, and Dejan Milojicic. Non-volatile memory file systems: A survey. *IEEE Access*, Vol. 7, pp. 25836–25871, 2019.

[39] Andy Rudoff. Persistent memory programming. *Login: The Usenix Magazine*, Vol. 42, No. 2, pp. 34–40, 2017.

[40] Abdul Salam, Safdar Jamil, Sungwon Jung, Sung-Soon Park, and Youngjae Kim. Future-based persistent spatial data structure for nvm-based manycore machines. *IEEE Access*, Vol. 10, pp. 114711–114724, 2022.

[41] Steve Scargall. *Programming Persistent Memory A Comprehensive Guide for Developers*. Apress, Berkeley, CA, 2020.
https://doi.org/10.1007/978-1-4842-4932-1.

[42] H. Shu, H. Chen, H. Liu, Y. Lu, Q. Hu, and J. Shu. Empirical study of transactional management for persistent memory. In *2018 IEEE 7th Non-Volatile Memory Systems and Applications Symposium (NVMSA)*, pp. 61–66, 2018.

[43] SNIA. SNIA Storage Networking Industry Association.
https://www.snia.org, 2017.

[44] Alexander van Renen, Lukas Vogel, Viktor Leis, Thomas Neumann, and Alfons Kemper. Persistent memory I/O primitives. In *Proc' of DaMoN'19*, pp. 1–7. ACM, 2019.

[45] Tianzheng Wang, Justin Levandoski, and Per-Ake Larson. Easy lock-free indexing in non-volatile memory. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pp. 461–472. IEEE, 2018.

[46] Michèle Weiland, Holger Brunst, Tiago Quintino, Nick Johnson, Olivier Iffrig, Simon Smart, Christian Herold, Antonino Bonanni, Adrian Jackson, and Mark Parsons. An early evaluation of Intel's optane DC persistent memory module and its impact on high-performance scientific applications. In *Proc' of SC '19*, pp. 1–19, 2019.

[47] Chin-Hsien Wu, Li-Pin Chang, and Tei-Wei Kuo. An efficient r-tree implementation over flash-memory storage systems. In *Proceedings of the 11th ACM international symposium on Advances in geographic information systems*, pp. 17–24, 2003.

[48] Fei Xia, Dejun Jiang, Jin Xiong, and Ninghui Sun. Hikv: A hybrid index key-value store for dram-nvm memory systems. In *USENIX/ATC 17*, pp. 349–362, 2017.

[49] Jian Xu and Steven Swanson. NOVA: A log-structured file system for hybrid volatile/non-volatile main memories. In *14th USENIX/FAST*, pp. 323–338, 2016.

[50] Jian Yang, Juno Kim, Morteza Hoseinzadeh, Joseph Izraelevitz, and Steve Swanson. An empirical guide to the behavior and use of scalable persistent memory. In *18th USENIX/FAST 20*, pp. 169–182, 2020.

[51] Jun Yang, Qingsong Wei, Cheng Chen, Chundong Wang, Khai Leong Yong, and Bingsheng He. NV-Tree: Reducing consistency cost for NVM-based single level systems. In *13th USENIX Conference on File and Storage Technologies (FAST 15)*, pp. 167–181, Santa Clara, CA, February 2015. USENIX Association.

[52] Jun Yang, Qingsong Wei, Chundong Wang, Cheng Chen, Khai Leong Yong, and Bingsheng He. Nv-tree: A consistent and workload-adaptive tree structure for non-volatile memory. *IEEE Transactions on Computers*, Vol. 65, No. 7, pp. 2169–2183, 2016.

[53] Hirotaka Yoshioka, Yuto Hayamizu, Kazuo Goda, and Masaru Kitsuregawa. pmmeter: A microbenchmark for understanding synchronization cost on persistent memory. In *2023 IEEE International Conference on Big Data and Smart Computing (BigComp2023)*, pp. 326–327. IEEE, 2023.

[54] Hao Zhang, Gang Chen, Beng Chin Ooi, Kian-Lee Tan, and Meihui Zhang. In-memory big data management and processing: A survey. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 27, No. 7, pp. 1920–1948, 2015.

[55] Xinjing Zhou, Lidan Shou, Ke Chen, Wei Hu, and Gang Chen. Dptree: differential indexing for persistent memory. *Proceedings of the VLDB Endowment*, Vol. 13, No. 4, pp. 421–434, 2019.

[56] Pengfei Zuo, Yu Hua, and Jie Wu. Write-optimized and high-performance hashing index scheme for persistent memory. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pp. 461–476, 2018.

[57] Yoav Zuriel, Michal Friedman, Gali Sheffi, Nachshon Cohen, and Erez Petrank. Efficient lock-free durable sets. *Proceedings of the ACM on Programming Languages*, Vol. 3, No. OOPSLA, pp. 1–26, 2019.