ハイブリッド型メインメモリを対象とする KVS サーバの試作

小沢公基 [1]　広渕崇宏 [2]　高野了成 [2]　菅谷みどり [1]

概要：スマートデバイス，センサ等の IoT の発展により地理的に離れたクラウドサーバだけでなくユーザ近傍のエッジでデータの処理を行うエッジコンピューティングに注目が集まっている．そこで我々はエッジで大量のデータを保持しかつ高い即応性で処理するために Key-Value-Store (KVS)に着目した．低レイテンシで大量のデータを処理するためには KVS サーバに大容量のメモリを搭載する必要があるが，現在使用されている DRAM は容量が小さく今後の大容量化の望みも薄い．そこで DRAM と同様にアクセスが可能，大容量かつ不揮発性という特徴も持つ不揮発性メインメモリ(NVMM)を使用して，DRAM と NVMM という 2 種類のメモリデバイスから構成されるハイブリッド型メインメモリを対象とする KVS サーバを提案する．NVMM は DRAM よりも読み書き性能が劣る．大容量化の恩恵を受けつつも NVMM による性能悪化を防ぐ必要がある．本稿では代表的な KVS サーバである Memcached に 2 種類のメモリデバイスを用い，アクセス頻度に応じてキャッシュデータを 2 種類のメモリデバイス間で移動する機構を提案，実装した．NVMM を搭載する実機を用いて評価を行った結果，提案実装の基本的な動作を確認し，キャッシュデータの配置の最適化により性能悪化を回避出来る事を確認した．

キーワード：エッジコンピューティング，インメモリ KVS，　NVDIMM，NVMM，DCPMM

1. はじめに

IoT (Internet of Things) 技術の進展により，スマートデバイスやセンサなどから大量のデータがサーバに転送され処理されている．今後 IoT 技術が我々の生活の隅々まで普及した際には，地理的に離れたクラウドデータセンタでデータを処理するだけではなく，ユーザ近傍のエッジ（エンドユーザとクラウドデータセンタとの中間層）においてもデータの処理が必要となる．このような考えはエッジコンピューティング技術[1]（Fog，cloudlet，MEC 等）として注目が集まっている．エッジコンピューティングにおいては，遠方のクラウドデータセンタ側にデータを保持するだけではなく，より高い即応性を実現するためにエッジ側においてもデータを保持する必要がある．我々は大量のデータを一時的にキャッシュすることを可能とする Key-Value-Store（KVS）サーバに注目している．大量のデータに対する低レイテンシでのアクセスを実現するためには，KVS サーバが大容量のメモリを搭載する必要がある．しかし現在メインメモリとして使われている DRAM の容量は小さく，今後大幅に増加する可能性は低い．また DRAM は揮発性であり，外部からの給電が途絶えると記憶内容が失われてしまう．そこで DRAM より大容量かつ不揮発性という特徴を備えうる不揮発性メインメモリ（NVMM）の使用を考える．例えば 2019 年 4 月に発売された Intel Optane DC Persistent Memory (Optane DCPMM)は，DRAM モジュールよりも一桁大きい容量を提供する．NVMM は DRAM と同様にメモリバス上に接続され，CPU からは DRAM と同様にアクセスできる．KVS サーバにおいて NVMM を用いることで，キャッシュサーバの容量を大幅に拡張することが可能になる．大容量化によって，クライアントのリクエストをクラウドで処理するのではなくエッジで処理できるようになるためリクエストのレイテンシを改善する事が出来る．さらに，本稿では議論しないものの，NVMM が備える不揮発性という特徴を活用できれば KVS サーバの堅牢性を高められる可能性がある．

しかし，NVMM は DRAM よりも大きな容量を提供する反面，その読み書き性能は DRAM よりも劣る．例えば，Optane DCPM の読み込みレイテンシはランダムアクセスの場合 DRAM よりも約 3 倍遅く，連続的なアクセスの場合は約 2 倍遅いとされる．読み込み帯域幅は最大でも 39.4 GB/s，書き込み帯域幅は 13.9 GB/s であり，DRAM よりも大幅に低下する[2]．KVS サーバにおいては，NVMM を用いてメインメモリの容量を拡張してキャッシュのヒット率を上げつつ, DRAM よりも低速な DCPMM による性能の悪化を避ける必要がある．

そこで本稿では，DCPMM を使用した大容量キャッシュサーバを提案する．DRAM と DRAM よりは大容量であるものの性能は劣る NVMM の両方をメインメモリとして搭載する計算機（ハイブリッド型メインメモリを搭載する計算機）において動作する KVS サーバである．アクセス頻度に応じて両者のメモリを使い分けることで，高い性能を発揮することを目指す. 本稿では代表的なインメモリ KVS の Memcached を対象に提案機構を開発した．アクセス頻度が高いキャッシュデータを DCPMM から DRAM に自動的に移動する．アクセス頻度が低いキャッシュデータは DRAM から DCPMM に動的に移動する．大容量化の恩恵を享受しつつ，高速な DRAM へのアクセスを増やす事でハイブリッド型メインメモリに対応した KVS サーバの性能を最大化する. この配置の最適化は Memcached が備えるセグメント LRU アルゴリズムを拡張して実装した．なお，

1 芝浦工業大学
Shibaura Institute of Technology
2 国立研究開発法人　産業技術総合研究所
National Institute of Advanced Industrial Science and Technology(AIST)

ⓒ 2019 Information Processing Society of Japan

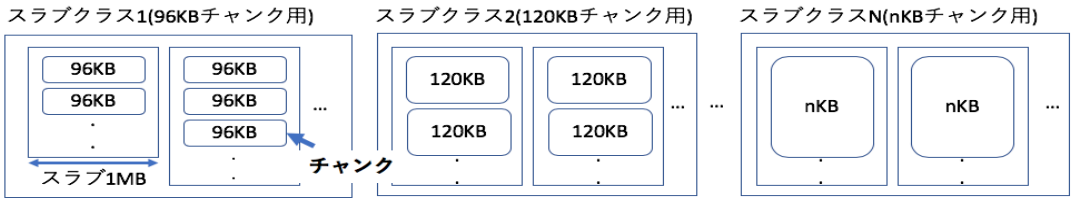

図 1　Memcached のメモリ管理データ構造

Memcached にはストレージを用いてサーバの容量を拡張する ext_store という機能があるが，本稿では用いていない．我々が想定する DCPMM は DRAM よりも読み書き性能が劣るものの，DRAM 同様に計算機のメインメモリとしてアクセスできる．そのような NVMM をストレージとして取り扱うことのオーバーヘッドを回避し，そのようなメモリに即した最適な管理手法を研究することを念頭に ext_store を用いずに開発を行った．Optane DCPMM を搭載した計算機を用いて評価実験を行い，提案機構の基本的な動作を確認した．キャッシュデータの最適化により性能低下を回避できることを確認した．

2 節で Memcached の構造を説明し，3 節で課題を述べる．4 節で提案機構を説明する．5 節で NVMM へのアクセス方法を述べ，6 節で評価実験を行う．7 節で本稿をまとめる．

2. Memcached

2.1 Memcached の仕組み

Memcached は，インメモリ KVS サーバとして代表的なプログラムの一つであり，今日データセンタにおいて広く用いられている．他の KVS サーバと比較して，全体の構造が比較的単純であることから，新たなメモリ管理手法を実装する対象として適している．クライアントからのリクエストに対応する複数のワーカスレッドやメモリ管理を担当するスレッドなどからなるマルチスレッドプログラムである．Memcached サーバは単独で動作し，サーバ間で通信を行わないので，複数の Memcached サーバを使用する際にはクライアント側で使い分ける必要がある．本稿では，Memcached の version 1.5.16 を対象とする．

まず初めに Memcached 内部のメモリ管理手法を説明する．Memcached においては，key とその key に対する value，その key の長さ等の情報を取りまとめたメタデータを合わせてアイテムとよぶ．メインメモリから確保したメモリの利用効率を高めるため，サイズが近いアイテムをグループ化して管理する．

- Memcached はスラブと呼ばれる単位でメインメモリからメモリを確保する．一つのスラブのサイズは 1 MB である．各スラブはいずれかのスラブクラスと呼ばれるグループに属する．スラブクラスごとにデータ管理サイズが決まっている．各スラブは，属するスラブクラスのデータ管理サイズごとにチャンクと呼ばれる単位に分割される．
- Memcached がアイテムをメモリ上に保存するときには，そのアイテムのサイズが収まる最小のデータ管理サイズを有するスラブクラスを見つけて，そのチャンクにデータを保存する．

図 1 に概要図を示す．例えばアイテムのサイズが 100 KB の場合には，データ管理サイズ が 120 KB であるスラブクラス 2 のチャンクに格納される．スラブクラス中の各チャンクは 2 重リンクで管理されており，新しく追加されたアイテムはリンクリストの先頭に追加される．

クライアントが Memcached サーバにデータを格納する際には，SET コマンドによって格納したい key と value を Memcached サーバに伝える．またクライアントが Memcached サーバに格納されているデータを参照する際には，GET コマンドによって key を伝える．対応する value がサーバ上に存在する場合には，サーバはそのデータをクライアントに返す．

SET コマンドに対する Memcached の処理を以下に示す．

1. key と value のデータおよびその付加情報の合計サイズに応じて，保存するスラブクラスを決め，そのアイテム用のチャンクを確保する．もし空きチャンクを確保できない場合には，LRU アルゴリズムを用いて，古いアイテムをチャンクから削除し，そのチャンクにアイテムを保存する．
2. key からハッシュ値を参照しハッシュテーブルに追加する．ハッシュテーブルはアイテムへのポインタの管理のために用いる．
3. スラブのリンクリストにアイテムを追加する．

GET コマンドに対する Memcached の処理を以下に示す．

1. key のハッシュ値を元にハッシュテーブルを参照し，アイテムを見つける．
2. アクセスがあったアイテムをリンクリストの先頭に移動する．
3. value の値をクライアントに返す．

また value を更新する UPDATE コマンドに対しては GET コマンドに対する処理と同様に対応するアイテムを見つけた上でその value を更新する．SET コマンドおよび GET コマンド等のクライアントからの要求はワーカスレッドが対応する．ワーカスレッド数はデフォルトで 4 スレッドである．

ⓒ 2019 Information Processing Society of Japan

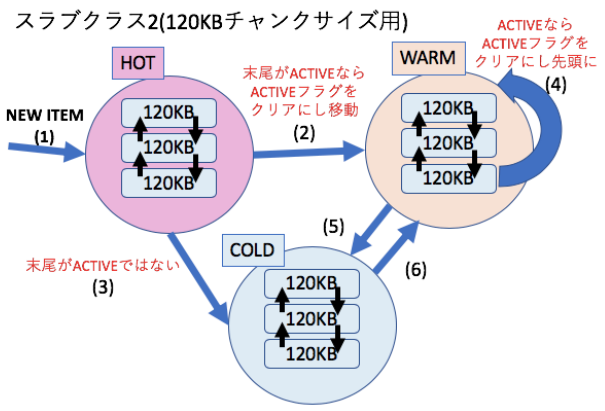

図 2　Memcached のセグメント LRU アルゴリズム

クライアントが SET コマンドを Memcached サーバに要求する際には，付加情報として TimeToLive（有効期間）を送信できる．Memcached サーバは，後述する LRU メンテナンススレッドにおいて，保持するアイテムの有効期間を随時監視し，有効期間が過ぎていればそのアイテムを削除する．また，クライアントが DELETE コマンドなどにより明示的にアイテムの削除を要求した場合にも，そのアイテムを削除する．

Memcached サーバは，新たにアイテムを追加する際にメモリが不足する場合には，セグメント LRU アルゴリズムによって古いアイテムをその有効期間に関わらず削除し，新たなアイテムを追加する．セグメント LRU アルゴリズムの概要図を図 2 に示す．各スラブクラスにはアイテムの参照頻度を管理する LRU が存在し，その LRU は HOT，WARM，COLD の 3 つのセグメントに分かれている．各アイテムはアクセス回数によって変動するフラグを持ち，そのフラグの状態によって 3 つのセグメント間を移動する．フラグは，アイテムが SET された直後には初期状態である．その後 1 度目に参照されると FETCHED 状態になり，2 度目に参照されると ACTIVE 状態となる．なおアイテムがセグメントを移動する際に ACTIVE 状態は初期状態に戻る．クライアントから新たに追加されたアイテムはまず初めに HOT セグメントの LRU キュー（以下 HOT キュー）に追加される．時間的局所性からこの HOT キューのアイテムが近い将来も多くアクセスされることが期待される．WARM キューには，追加されてから一定時間が経過したものの，今後もアクセスが期待されるアイテムが保持される．COLD キューには，アクセスが少ないアイテムが保存される．メモリが不足し古いアイテムを削除する際には，COLD キューの末尾のアイテムが対象となる．ワーカスレッドとは別に，セグメント LRU キューの維持管理を行うスレッド(LRU メンテナンススレッド)が存在する．LRU メンテナンススレッドは周期的に LRU キューをスキャンする．

（1）　ワーカスレッドは，スラブに新しく追加するアイテムを HOT キューに追加する．

（2）　LRU メンテナンススレッドは, HOT キューの末尾のアイテムが ACTIVE なら（つまり参照回数が 2 回以上なら）そのアイテムを WARM キューの先頭に付け替える．アイテムのフラグを初期状態に戻す．

（3）　LRU メンテナンススレッドは, HOT キューの末尾のアイテムが ACTIVE 状態ではない場合（つまり参照回数が 1 回以下）ならそのアイテムを COLD キューの先頭に付け替える．アイテムのフラグを初期状態に戻す．

（4）　LRU メンテナンススレッドは，WARM キューの末尾のアイテムが ACTIVE 状態ならそのアイテムを WARM キューの先頭に付け替える．アイテムのフラグを初期状態に戻す．

（5）　LRU メンテナンススレッドは，WARM キューの末尾のアイテムが ACTIVE 状態でなければ COLD キューの先頭に付け替える．アイテムのフラグを初期状態に戻す．

（6）　LRU メンテナンススレッドは，COLD キューの末尾のアイテムが ACTIVE 状態であれば WARM キューの先頭に付け替える．アイテムのフラグを初期状態に戻す．

サーバのメモリに空きがない場合には，COLD キューの末尾のアイテムが削除対象となる．

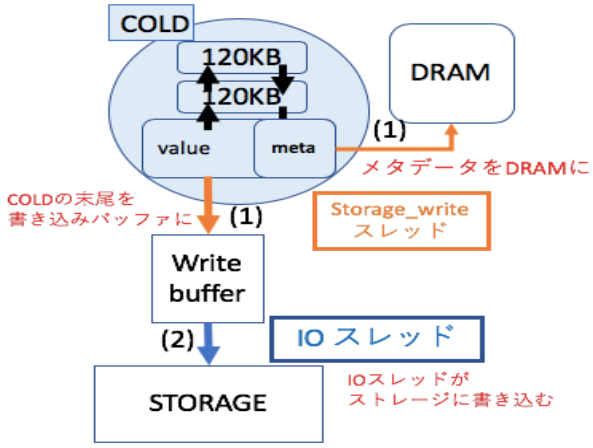

図 3　Memcached のストレージ利用機能（ext_store）の動作概要

2.2　Memcached のストレージ利用機能

Memcached ver1.5.4 以上には，メインメモリに加えて，ストレージにデータを保持する機能 (ext_store) が存在する．主にフラッシュストレージの使用が想定されている．その動作概要は以下の通りである．

（1）　ストレージ書き込みスレッド（storage_write）は，COLD キューの末尾のアイテムをチェックし，一定時間参照されていないなど，ストレージへの移

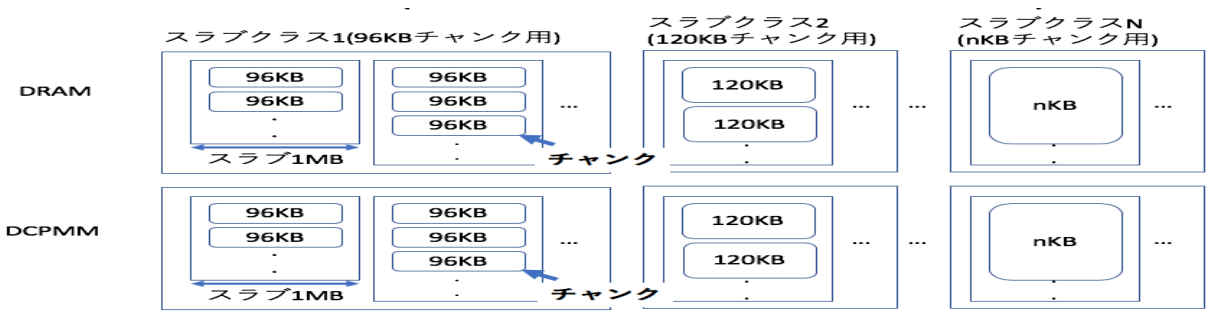

図 4　提案機構におけるメモリ管理データ構造

動条件が満たされるか確認する．アイテムのサイズやメモリの空き状況も考慮する．ストレージへの移動条件が満たされ，書き込み待ちのバッファ(write buffer)に空きがある場合には，アイテムを書き込み待ちバッファに移動する．

（2）　ストレージへの実際の書き込みを行う．IO スレッドが書き込み待ちバッファ (write buffer)からアイテムを取得し，実際にストレージに書き込む．

なお，アイテムのメタデータは引き続きメモリ上で保持される．DRAM においては 1 MB 単位でのメモリを確保していたのに対して，ストレージにおいてはデフォルトで 64 M 単位で確保する．書き込み待ちバッファの容量はデフォルトで 4 MB である．

3. 課題

DRAM と NVMM の両方から構成されるハイブリッド型メインメモリにおいては，両者のメモリデバイスをどのように使い分けるのかが課題となる．NVMM は DRAM よりも大きな容量を提供する反面，その読み書き性能は DRAM よりも劣るとされる．KVS サーバにおいては，NVMM を用いてメインメモリの容量を拡張して，キャッシュするデータに対するヒット率を上げつつ，DRAM よりも低速な NVMM による性能の悪化を避ける必要がある．

Optane DCPMM をサポートする Intel 社製 CPU においては，メモリコントローラにおいて両者のメモリデバイスを使い分ける機能（DCPMM Memory Mode）が搭載されている．DRAM を DCPMM に対するキャッシュとして使うことで，DRAM よりも性能が劣る DCPMM へのメモリアクセスを減らして，アプリケーションの性能低下を抑制する．ハードウェアレベルで実装されているため，KVS サーバを拡張せずとも DCPMM によるメインメモリの容量拡大の恩恵を受けられるという利点がある．しかし，今後他社からも販売されるであろう NVMM においても同様の仕組みが存在するかは不透明である．また，メモリコントローラによる使い分けの仕組みはアプリケーションに特化しないため，ハイブリッド型メインメモリに対して KVS サーバの性能を最大限引き出すことが困難である．

また，Memcached には DRAM に加えてストレージも利用してデータを保持する機能が既に存在している．しかし，書き込み待ちバッファでのバッファリングなど，DRAM よりも大幅に速度が劣るフラッシュストレージ等を想定した設計となっている．DRAM 同様にメインメモリとしてアクセスが可能であり，DRAM よりは大きいもののストレージよりは小さいレイテンシを有する NVMM に対して，Memcached のストレージ利用機能は十分な性能を引き出すことはできないと予想される．NVMM に即した新たな仕組みが必要である．

4. 提案

DRAM と NVMM の両方から構成されるハイブリッド型メインメモリを対象とする KVS サーバを提案する．Memcached を対象として，そのメモリ管理機構であるセグメント LRU を拡張し，アイテムのアクセス頻度によって 2 種類のメモリにおいてアイテムを動的に移動する仕組みを設けた．アクセスが多いアイテムを高速な DRAM に保持し，アクセスが少ないアイテムを低速な NVMM に保持することで，大容量のデータ保持を可能としつつ，低速な NVMM を用いることによる性能低下を緩和する．

4.1 ハイブリッド型メインメモリ向けのメモリ管理構造

図 4 に提案機構のメモリ管理構造の概要を示す．図 1 で示した Memcached が本来備えるスラブクラス群は DRAM 用として維持しつつ，新たに NVMM 用のスラブクラス群を導入した．NVMM 用に作成したスラブ構造においても各スラブクラスのチャンクサイズは DRAM と同様である．メモリデバイスごとにスラブクラス群を用意した理由は，既存の Memcached の仕組みを大きく変更せずに，NVMM を用いてメモリ容量を拡張する事を試みたのに加えて，クライアントからのキャッシュ要求量が DRAM のメモリサイズ以下の場合には既存の Memcached と同じ動作をする事を目指したからである．

4.2 セグメント LRU アルゴリズムの拡張

既存の DRAM 用セグメント LRU に加えて，新たに

ⓒ 2019 Information Processing Society of Japan

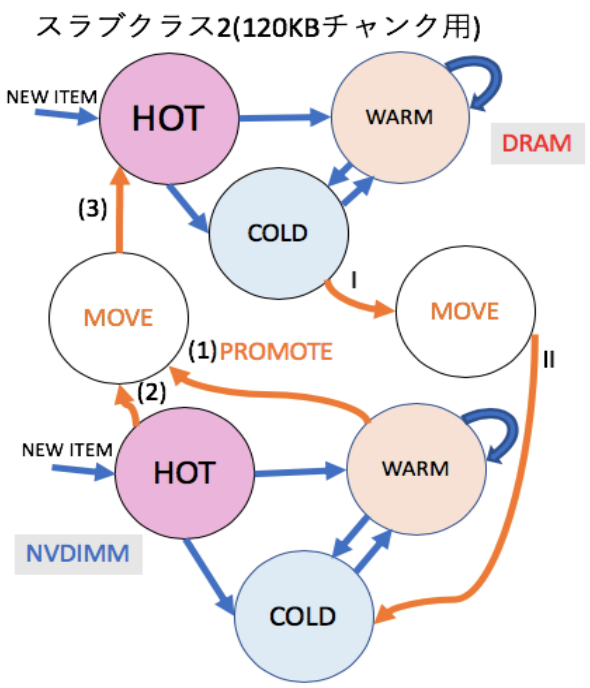

図 5　提案セグメント LRU アルゴリズム

NVMM 用のセグメント LRU を設けた．さらに MOVE キューとよばれるメモリ間移動の対象となったアイテムを追加するキューを作成した．両セグメント LRU は 2 つの MOVE キューによって連結されている．DRAM から NVMM へアイテムが移動する際に経由する MOVE キューと，NVMM から DRAM へアイテムが移動する際に経由する MOVE キューが存在する．図 5 に概要を示す．各セグメント LRU の末尾をチェックするため，LRU メンテナンススレッドも DRAM と NVMM 用にそれぞれ設けた．それぞれ設けることで，大容量の NVMM を追加した際に DRAM 用のセグメント LRU のスキャン頻度が大きく低下することを避ける．

既存のセグメント LRU アルゴリズムでは，アイテムへの参照が 2 度あるとそのアイテムは ACTIVE 状態となり，キューの末尾における処理が変わる．本提案では，後述するように NVMM から DRAM への移動対象となるアイテムを区別するため，アイテムに対して PROMOTE 状態を追加した．現状では，ACTIVE 状態のアイテムがさらに参照された際に PROMOTE 状態に遷移する．

アクセスの多いアイテムを NVMM から DRAM に移動する処理について述べる．NVMM 用の LRU メンテナンススレッドが NVMM の HOT キューもしくは WARM キューに存在するアイテムで PROMOTE 状態にあるアイテムを見つけ出し，DRAM への移動用の MOVE キューに追加する．NVMM 用の LRU メンテナンススレッドは DRAM への移動用の MOVE キューを随時参照して，アイテムが存在すれば DRAM の HOT キューにそのアイテムを移動する．アイテムの移動は，移動先のスラブからアイテムを確保，移動先のアイテムにデータ等をコピー，移動前のアイテムを削除するという処理となる．この時に移動先のスラブからアイテムを確保ができない際には COLD キューの末尾のアイテムを削除する．それでも確保に失敗した時には，メモリ間のアイテムの移動を諦める．図 5 にその概要を示す．

（1）　NVDIMM の HOT キュー又は WARM キューのアイテムに PROMOTE フラグが立つ．
（2）　そのアイテムを MOVE キューに移動する．
（3）　MOVE キュー内のアイテムを DRAM に移動する．

次にアクセスの少ないアイテムを DRAM から NVMM に移動する処理について述べる．アクセスが少ないアイテムが DRAM に長く存在しメモリを浪費しないように，DRAM に存在できる時間を制限する機能を新たに設けた．DRAM の COLD キューの末尾のアイテムが，設定した時間を超過している場合には，NVMM にアイテムを退避する．一旦 MOVE キューに移動した後に，NVMM への移動が行われる．図 5 にその概要を示す．

I.　DRAM の COLD キューに存在するアイテムについて，DRAM のメモリにほぼ空きがない場合あるいはそのアイテムにアクセスがないまま一定時間以上 DRAM に存在する場合に，MOVE セグメントに移動する．
II.　MOVE キュー内のアイテムを NVMM に移動する．

MOVE キューを作成した理由を説明する．LRU メンテナンススレッドはキューの走査中にキュー毎のスレッド間排他ロックを行う．LRU メンテナンススレッド内において COLD キューの走査中にアイテムのメモリ間移動を行うと，DRAM の COLD キューがロックされる時間が長くなる．Memcached では SET コマンド時にアイテムが確保できない場合は，COLD キューから削除対象を見つける仕組みになっており，LRU メンテナンススレッドが COLD キューを頻繁にロックすると SET コマンド時に著しくスループットが落ちる．このことから，一旦 MOVE キューにアイテムを移動させる方法を採用する事により COLD キューのロック頻度を上げないようにした．

5.　実装

以上に述べた提案機構を Memcached 1.5.16 に対して試作した．拡張機能部分は，C 言語で約 400 行である．Optane DCPMM へのアクセスには，Linux の NVDIMM ドライバが提供する Device Dax モードを用いた．NVMM がマップされている物理メモリアドレス領域から，指定のサイズだけメモリページを切り出せる．ユーザ空間から Device Dax 用デバイスファイル（/dev/dax 等）を通じて安全に操作できる．提案機構は，NVMM に対してはそのデバイスファイルを mmap()して仮想アドレス空間にマッピングし，通常の

ⓒ 2019 Information Processing Society of Japan

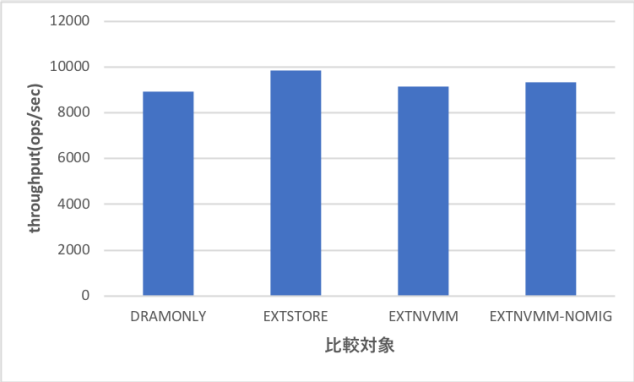

図 6　スループット

表 1　キャッシュミス率

比較対象	キャッシュミス率
DRAMONLY	0 %
EXTSTORE	79.51 %
EXTNVMM	1.68 %
EXTNVMM-NOMIG	6.58 %

DRAM と同様に読み書きを行う．

6. 評価

提案機構のプロトタイプの基本的な動作を確認し，その基本的な性能を明らかにするため，複数の評価実験を行った．既存の Memcached との性能比較を行った．Memcached サーバに対するベンチマークツールとして，オープンソースの NoSQL データベース向けベンチマークプログラムである YCSB [7]を用いた．

6.1 評価内容

6.1.1 実験環境

Memcached を実行するサーバ計算機と，Memcached に対するベンチマークツールを実行するクライアント計算機の 2 台を用意した．サーバとクライアント間のデータ転送は 10 GbE を用いた．クライアント計算機は Intel Xenon CPU E5-2630 2.20 GHz を 2 個および 120 GBytes のメモリを搭載している．サーバ計算機は Intel Xeon Gold 6230 2.10 GHz を 2 個および 180 GBytes のメモリを搭載している．またサーバ計算機は約 1500 GB の Optane DCPMM を搭載している．比較対象のため，以下 4 通りの設定で実験を行った．

- DRAM 1280 MB を用いる既存 Memcached サーバ（以下 DRAMONLY）．
- DRAM 128 MB を用いる既存 Memcached サーバ．既存 Memcached サーバのストレージ利用機能（ext_store）が SSD 1280 MB を用いて有効（以下 EXTSTORE）．
- DRAM 128 MB および DCPMM 1280 MB を用いる提案機構による Memcached サーバ．動的なアイテム移動は有効．アクセス頻度により DCPMM と DRAM 間を移動させる提案サーバ（以下 EXTNVMM）
- DRAM 128 MB および DCPMM 1280 MB を用いる提案機構による Memcached サーバ．提案機構を用いるものの，比較のため DCPMM から DRAM への動的なアイテム移動が無効．（以下 EXTNVMM-NOMIG）

6.1.2 YCSB の設定

YCSB ではロードセクションと実行セクションが存在している．今回はロードセクションにてサイズが 1 KB のデータセットを 50 万個，Memcached サーバにロードした．実行セクションでは Memcached サーバからデータを読み込む READ コマンドと既存のデータを上書きする UPDATE コマンドを 9:1 の割合で 500 万回実行した（以下 ReadWrite ワークロード）．加えて実行セクションにおいて 500 万回の READ コマンドを実行するワークロードを用意した（以下 ReadOnly ワークロード）．YCSB の実行スレッドは 1 つとした．なお READ コマンドにおいてアクセスするアイテムの出現頻度はジップの法則[8]に従うように設定した．

6.2 評価

6.2.1 スループット/キャッシュミス率

図 6 に ReadWrite ワークロードを用いた際の 4 つの比較対象毎のスループット，表 1 にキャシュミス率を示す．DRAMONLY，EXTNVMM，EXTNVMM-NOMIG の 3 種類のサーバはほぼ同様のスループットであるのに対し，EXTSTORE は 1 割程度高いスループットとなった．一方，ミス率は DRAMONLY が 0 %，EXTNVMM が約 1 %，EXTNVMM-NOMIG が約 7 %であるのに対し，EXTSTORE は 80 %近い結果となった．

EXTSTORE のスループットが良いのは，ミス率と大きく関係がある．YCSB のロードセクションにおいて，EXTSTORE の場合は DRAM から溢れたアイテムはストレージに移動するが，ストレージへの書き込みが間に合わず，書き込みバッファに入りきらなかった DRAM のアイテムは削除される．これによってサーバに保存されるアイテム数が他の 3 種類のサーバより少ない．これが理由により，クライアントからの READ コマンドおよび UPDATE コマンドに対して，ハッシュテーブルへのアクセス時点でキャッシュミスが判明しその応答を返すことになり，データへのアクセスがない分，スループットが他の 3 種類に比べて高くなった．

6.2.2 レイテンシ

READ コマンドおよび UPDATE コマンドの 95th-Percentile レイテンシを図 7 に示す．DRAMONLY，EXTNVMM，EXTNVMM-NOMIG の 3 種類のサーバは，READ コマンドおよび UPDATE コマンド 共に 135-145 us である．これに対して，EXTSTORE は，READ コマンドおよび UPDATE コマンドとも 130 us 前後となった．理由は 6.2.1 に述べた内容と同様に，サーバ保存されるアイテム数が他の 3 種類のサーバより少なく， コマンドに対す

ⓒ 2019 Information Processing Society of Japan

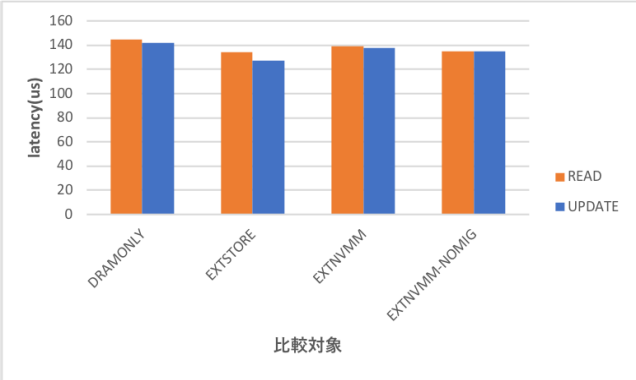

図 7　レイテンシ

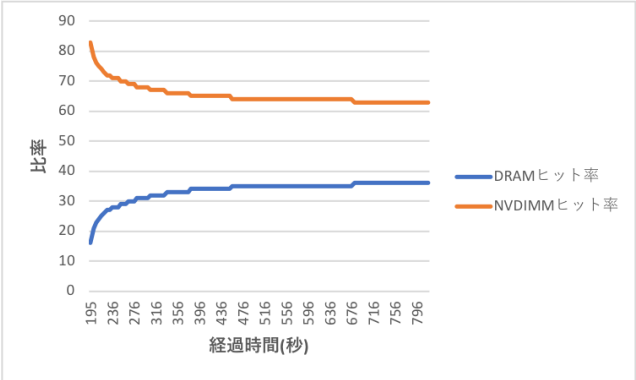

図 8　メモリデバイス毎のヒット率

る応答性が高い結果となる．

DRAMONLY，EXTNVMM，EXTNVMM-NOMIG のレイテンシにほとんど差がない事から LRU アルゴリズムを拡張しメモリ間のアイテム移動を自動的に行う実装の仕組自体が性能を下げるような影響を及ぼしていないと考える．

6.2.3 メモリデバイスごとのヒット率

図 8 に ReadOnly ワークロードを使用した際の EXTNVMM における全体のヒット数に対する DRAM へのヒットと NVMM へのヒットの割合を時系列で表した結果を示す．実行直後は容量差から予想できるように NVMM の方が多い．しかし時間が経つにつれて DRAM に保存されているアイテムへのヒットが増加し，最終的には DRAM へのヒット率は最初の約 2.4 倍の割合となった．

6.2.4 DRAM への移動数

図 9 は ReadOnly ワークロードにおける NVMM から DRAM へ移動したアイテムの数の時系列グラフである．移動数は，開始直後はリニアに増加したが，開始から 1 分を境に時間に沿って徐々に減少する結果となった．

6.3　考察

6.2.1 節, 6.2.2 節 の評価では，ロードするデータサイズはサーバのメモリサイズより小さくなるように設定し，実験を行った．DRAMONLY と同様に EXTNVMM，EXTNVMM-NOMIG もミス率が 0 %になると予想したが EXTNVMM で約 2 %，EXTNVMM-NOMIG で約 5 %となった．これは Memcached においてバックグラウンドで動作している Slab_automove という機能の影響であると考えられる．Memcached は一度確保したスラブは基本的には解放しない．スラブクラス X で確保したスラブを使わなくなり余っている状態の時に，多くの保存の要求がスラブクラス Y に来てもスラブクラス X がスラブを浪費しているため，スラブクラス Y にスラブをアサインする事が出来ない．これを解消するのが，Slab_automove である．Slab_automove はスラブクラス X からスラブを解放し，スラブクラス Y に再割り当てを行う．再割り当ての際，スラブクラス X 内のスラブにアイテムが残っている場合は極力同スラブクラス内の別スラブに避難させるが，出来ない場合アイテムを削除する．50 万回の UPDATE コマンドとメモリ間の移動によりスラブの再割り当てが強く必要となり，Slab_automove の機能が働き，一部アイテムがサーバから削除される結果となった．そのため，提案手法のミス率が若干上がったと考えられる．

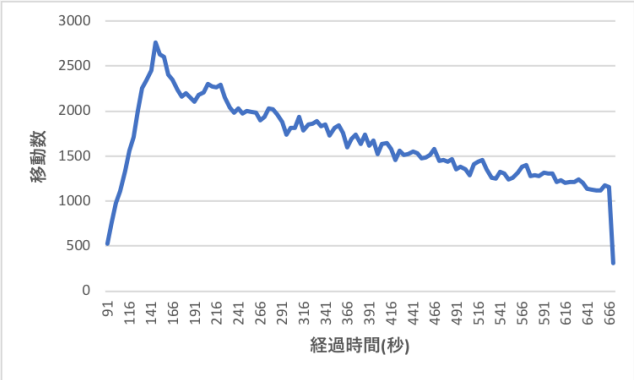

図 9　NVMM から DRAM への移動数

6.2.3 節については，これはアクセス頻度によるメモリ間のアイテム移動の仕組みにより，DRAM へのヒットが増加したと考えられる．グラフが示すメモリヒット率の推移から提案した仕組みの動作が想定通りであることを確認できる．

6.2.4 節の DRAM への移動数の推移の結果については，最初リニアに増加した後に，減少する傾向にある．これはよくアクセスされるアイテムが DRAM に集まる事によって, NVMM のアイテムへのアクセスが減少したという事が移動数の推移から分かる．

以上の結果から，提案機構は正しく動作していることが確認できた．提案機構による大きな性能低下は今回実験した範囲では見られなかった．また既存の Memcached の機能である Slab_automove と提案手法の相性が良くないことが分かった．

7.　まとめと今後の課題

従来インメモリ KVS サーバで用いられてきた DRAM は，

その容量の成長に限界が近づきつつある．DRAM のみを用いていては大容量のデータをキャッシュすることが難しい．Intel Optane DCPMM に代表される新たなメインメモリデバイスも併用することで，大容量の KVS サーバを実現できるはずである．本稿では，DRAM と，DRAM よりも大容量であるものの性能に劣るメモリデバイスの両者からなるハイブリッド型メインメモリに対する KVS サーバを提案した．アクセス頻度によってメインメモリ上におけるデータの配置場所を最適化することで，大容量メインメモリを享受しつつ，性能低下を抑制する．代表的な KVS サーバである Memcached に提案機構を実装した．DRAM と Optane DCPMM を搭載した計算機を用いた評価実験によって，プロトタイプが基本的には想定通り正しく動作することを確認した．DRAM よりも性能が劣る Optane DCPMM を併用しても，大幅な性能低下を回避できることを確認した．

今後の課題として，既存の Memcached のスラブ再割り当て機能との両立を図ることを検討する．またエッジコンピューティング環境に適した KVS サーバとして開発を行う．また複数クライアントによる性能評価を予定する．

謝辞　本研究の一部は JST/CREST JPMJCR19K1 の支援を受けた．本研究の一部は科研費 19H01108 の助成を受けた．

参考文献

[1] Kasif Bilal, Osman Khalid, Aiman Erbad, Samee U. Khan. Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers. Computer Networks. 2018, vol.130, p.94-120.

[2] インテル® Optane™ DC パーシステント・メモリー .https://www.intel.co.jp/content/www/jp/ja/architecture-and-technology/optane-dc-persistent-memory.html, (参照 2019-11-7).

[3] Joseph Izraelevitz, Jian Yang, Lu Zhang, Juno Kim, Xiao Liu, Amirsaman Memaripour, Yun Joon Soh, Zixuan Wang, Yi Xu, Subramanya R. Dulloor, Jishen Zhao, Steven Swanson. Basic Performance Measurements of the Intel Optane DC Persistent Memory Module.2019.

[4] Assaf Eisenman,Darryl Gardner,Islam AbdelRahman,JensAxboe,Siying Dong,Kim Hazelwood,Chris Petersen,Asaf Cidon,Sachin Katti . Reducing DRAM footprint with NVM in Facebook. EuroSys. 2018, Article Np.42.

[5] dormando. Memcached wiki. https://github.com/memcached/memcached/wiki, 2019.

[6] “Extstore In The Cloud” . https://memcached.org, (参照 2019-11-7).

[7] “Yahoo Cloud Serving Benchmark” . https://research.yahoo.com/news/yahoo-cloud-serving-benchmark/, (参照 2019-11-7).

[8] G. K. Zipf. Human Behavior and the Principle of Least Effort. 1949, Addison-Wesley.

ⓒ 2019 Information Processing Society of Japan