ブロックストレージとの組み合わせによるメモリストレージ容量拡張手法

追川 修一[1]

概要：フラッシュメモリよりも記憶デバイスとしての性能が遙かに高い MRAM，PCM (phase change memory) , ReRAM といった次世代不揮発性メモリ（NV メモリ）の実用化が進んでおり，これらのメモリを用いた SSD の研究開発も行われている．NV メモリのなかでも MRAM は，DRAM に相当する性能と耐久性を持つという点で優れているが，集積度では劣っているため，ストレージの主体デバイスとしての用途は，現状では考えにくい．そこで，大容量のブロックストレージと組み合わせ，MRAM の容量を仮想的に拡張して用いることで，その高速性と不揮発性を活かす手法を提案する．提案手法は，Linux カーネルに実装した．実験結果から，従来手法よりも低コストなアクセスを実現できることがわかった．

1. はじめに

フラッシュメモリよりも記憶デバイスとしての性能が遙かに高い MRAM, PCM (phase change memory) , ReRAM といった次世代不揮発性メモリ（NV メモリ）の実用化が進んでいる．これらのメモリを用いた SSD の研究開発も行われており，例えば，Moneta [1] は PCM をストレージデバイスとした高性能 SSD である．また，フラッシュメモリに ReRAM をキャッシュとして組み合わせることで高性能化を実現した例 [2] もある．本論文では，不揮発性のバイトアクセス可能なメモリをメモリストレージと呼び，SSD や HDD のようなブロック単位でアクセスする従来のストレージをブロックストレージと呼ぶ．

本論文では，Linux カーネルを対象とし，メモリストレージにブロックストレージを組み合わせ，メモリストレージの容量を拡張する手法 VEMS (Virtually Extended Memory Storage) について述べる．VEMS は，バイトアクセス可能なメモリストレージへの直接アクセスを活用することで，ストレージへのアクセスコストの低減を可能にする．そのために，メモリストレージを隠蔽するのではなく，むしろ逆に露出し，ブロックストレージと合わせたストレージ全体を，メモリストレージとしてアクセス可能にする．メモリストレージは，CPU が直接アクセスできるため，オペレーティングシステム (OS) カーネルがメインメモリ上に管理するページキャッシュを介さずに，アクセス可能である．全体をメモリストレージとみなすことで，一旦ページキャッシュへデータをコピーする必要がなくなる．またこれにより，ページキャッシュとして用いられるメモリ容量を削減することができる．

VEMS に適したメモリストレージとして，MRAM を想定する．MRAM は，NV メモリの中では，DRAM に相当する性能と耐久性を持つという点で優れている．しかし，集積度では劣っているため，ストレージ全体を MRAM で構成することは，現状では考えにくい．そこで，大容量のブロックストレージと組み合わせ，MRAM の容量を仮想的に拡張して用いることで，その高速性と不揮発性を活かすことができる．

主ストレージに，より高速なストレージを組み合わせ，アクセスを高速化する手法は，これまで数多くの研究開発が行われてきた [3], [4], [5], [6]．これらはいずれも，低速大容量な HDD を主ストレージとし，高速な SSD を組み合わせるものである．どちらのストレージもブロックデバイスであり，OS カーネルはブロックストレージとしてアクセスする．一方，VEMS は，大容量の主ストレージと高速なストレージを組み合わせるという点は共通しているが，全体をメモリストレージとして提供するという点で，これらの既存手法とは異なっている．

VEMS は，Linux カーネルに実装した．VEMS は，メモリストレージのデバイスドライバと，ファイルシステムからメモリストレージへアクセスするためのレイヤから構成される．実験結果から，従来手法よりも低コストなアクセスを実現できていることがわかった．なお，VEMS は XIP (eXecution-In-Place) 機能を拡張したインタフェースを提供するため，それを通して，メモリストレージをユーザプ

[1] 筑波大学 システム情報系情報工学域
University of Tsukuba, Tsukuba, Ibaraki 305–8573, Japan

ⓒ 2014 Information Processing Society of Japan

ロセスの仮想アドレス空間にマップ可能である．そのための実装は行われているが，性能は現状では十分ではないため，評価の対象としていない．

本論文の構成は以下の通りである．2 章で背景を述べる．3 章では VEMS の設計について，4 章では実装について述べる．5 章は実験結果を示す．6 章は関連研究を述べ，7 章で本論文をまとめる．

2. 背景

本論文の背景として，NV メモリ，高速ストレージとの組み合わせによるアクセス高速化手法，NV メモリとの組み合わせにおける問題点について述べる．

2.1 NV メモリ

MRAM, PCM (phase change memory) , ReRAM といった次世代不揮発性メモリ（NV メモリ）は，不揮発性であることからストレージデバイスとして用いることできる一方，バイトアクセス可能であるためメインメモリの一部として用いることができるという特徴を持つ．これらの NV メモリは，実現するための技術が異なるため，それぞれ異なった特徴を持つ [7]．そのため，用途も異なってくると考えられる．

PCM, ReRAM は，読み出しの遅延は比較的短いが，書き込みについて遅延および耐久性の問題がある．しかしながら，高い集積度を実現できるため，主にストレージでの利用が研究されている [1], [2]．

MRAM は，DRAM に相当する性能と耐久性を持つという点で優れている．MRAM の特徴として，PCM, ReRAM と異なり，書き込みについて遅延や耐久性といった問題を持たないため，メインメモリの一部，またはプロセッサのキャッシュとしての利用が研究されている [8]．集積度では劣っているため，ストレージの主体デバイスとしての用途は，現状では考えにくい．しかしながら，他の大容量ストレージと MRAM の組み合わせで構成されるストレージは，十分に考えられる．

2.2 NV メモリの OS サポート

NV メモリをメモリストレージとすると，その上にファイルシステムを構築して使用することになる．その場合，単にメモリストレージをブロックストレージと同様に見なし，メモリストレージのデバイスドライバは，ブロック単位のアクセス要求を処理するようにすることができる．しかしながら，この方法では，メモリストレージが直接バイトアクセス可能であるという特徴を活かしておらず，メモリストレージ上のデータは一旦ページキャッシュに読み出され，必要に応じてまた書き戻されることになる．

そこで，メモリストレージへの直接アクセスを可能にするために導入されたのが，XIP (eXecution-In-Place) 機能である．XIP を用いることで，ページキャッシュの介在はなくなる．read, write システムコールを用いたアクセスの場合，ユーザプロセスのバッファとメモリストレージ間で読み書きが行われるようになる．また，mmap システムコールがファイルがマップする場合，ユーザプロセスの仮想アドレス空間が，メモリストレージの領域を参照する．XIP を用いるためには，ファイルシステムとデバイスドライバ両方のサポートが必要となる．

NV メモリの管理については，上記のようにメインメモリとは別にメモリストレージとして管理する方法の他に，不揮発性ではないメインメモリ上にも動的にメモリ割り当てを行うことでファイルシステムも構築可能であるため，メインメモリと同様に管理可能であるとの主張もあり，両者の間で議論がある [9]．NV メモリをメインメモリと同様に管理可能であるとすれば，既存のページキャッシュ機構に組み入れることで，ページキャッシュとストレージの統合の可能性がある．しかしながら，ページキャッシュ機構を含め，メインメモリの管理機構は，実行中のカーネルと密接に関係しており，カーネルの再起動やバージョン変更をまたいでデータを保持する前提とはなっていない．そのため，ページキャッシュとストレージの統合には，大きな変更が必要であり，これは将来的な課題である．

2.3 高速ストレージとの組み合わせによるアクセス高速化

主ストレージに，より高速なストレージを組み合わせ，アクセスを高速化する手法は，これまで数多くの研究開発が行われてきた [3], [4], [5], [6]．これらはいずれも，低速大容量な HDD を主ストレージとし，高速な SSD を組み合わせるものである．頻繁にアクセスされるデータを，SSD 上に置くことで，アクセスを高速化する．一方，アクセスされないデータは HDD 上に置き，全体としては大容量を提供する．

これらは，別個の HDD と SSD を，ソフトウェアまたはハードウェアにより組み合わせ，全体として単一のストレージとする．そのため，最新データの所在が分散する，組み合わせのための情報が失われると単一ストレージに戻すことができない，SSD の書き込み耐久性は限りがある，といった問題点があり，これらに対処するための研究が行われてきた．

2.4 NV メモリとの組み合わせにおける問題点

NV メモリを高速なメモリストレージとし，主ストレージとなるブロックストレージと組み合わせる場合，組み合わせたストレージのインタフェースを，メモリストレージとするか，ブロックストレージとするかが問題となる．HDD に SSD を組み合わせる場合，どちらのストレージもブロックデバイスであるため，OS カーネルは，組み合わせたストレージを，ブロックストレージとしてアクセスする．

ⓒ 2014 Information Processing Society of Japan

同様に，メモリストレージをブロックストレージとして扱い，全体としてもブロックストレージとしてアクセスすることができる．この場合，単に SSD の代替として，NV メモリを使用することになる．

メモリストレージは，1) CPU からの直接アクセスが可能であり，ページキャッシュを介す必要が無いことから，アクセスコストが低減される，2) ページキャッシュのためのメモリ容量が不要になる，といった利点を持つ．しかしながら，NV メモリをブロックストレージとして扱う方法では，これらの長所を活かすことができないという問題点がある．

3. 設計

本章では，Linux カーネルを対象とし，メモリストレージにブロックストレージを組み合わせ，メモリストレージの容量を拡張する手法 VEMS (Virtually Extended Memory Storage) について述べる．まず，対象とするシステム構成を明確にする．次に，目的と実現すべき要件を定義した後に，提案手法について述べる．

3.1 対象とするシステム構成

メモリストレージ，ブロックストレージの組み合わせ方には，コンピュータシステムへの接続方法およびストレージ間の制御方法について，それぞれ以下のいくつかの形態が考えられる．まず，コンピュータシステムへの接続方法としては，以下の 3 つの形態があり得る．

(1) メモリストレージ，ブロックストレージの両方が，I/O バスを通してコンピュータシステムに接続．
(2) メモリストレージはメモリバス，ブロックストレージは I/O バスを通し，それぞれ別個にコンピュータシステムに接続．
(3) メモリストレージ，ブロックストレージの両方が，メモリバスを通してコンピュータシステムに接続．

また，メモリストレージ，ブロックストレージ間の，データ転送を含む制御方法としては，以下の 2 つの形態があり得る．

(a) ストレージ間のデータ転送は，デバイス側で制御．
(b) ストレージ間のデータ転送は，コンピュータシステム側のソフトウェアで制御．

接続方法と制御方法の組み合わせのうち，現実的な形態としては，(1-a), (2-b) が考えられる．(1-a) は，メモリストレージとブロックストレージを組み合わせた，単一のストレージデバイスとしての提供となる．(1-b), (2-b) は，メモリストレージとブロックストレージは，別個のデバイスとしての提供となる．(1-b), (2-b) の形態を，図 1, 2 に示す．

本論文では，メモリストレージとして用いる NV メモリとして MRAM を想定し，(2-b) の組み合わせ形態をターゲットとする．その理由として，メインメモリとして適した特性を持つ MRAM はメモリバスに接続することでその性能を発揮することができ，また別途制御用のハードウェアを用意することなく，既存のブロックストレージと組み合わせて使用することができるからである．さらに，タブレットやスマートフォンなどのモバイルシステムでは，基本的にメモリやブロックストレージ等のデバイスは交換不可能であり，その場合，ソフトウェアでストレージ間の制御を行ったとしても，単一のストレージデバイスと見なすことができると考えるからである．

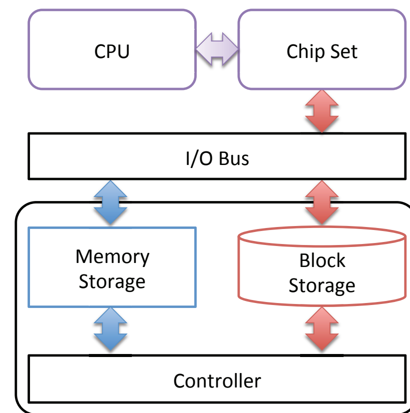

図 1 メモリストレージ，ブロックストレージの組み合わせ形態 (1-a)

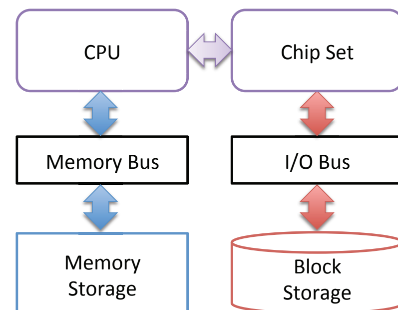

図 2 メモリストレージ，ブロックストレージの組み合わせ形態 (2-b)

3.2 目的と要件

本論文で提案する手法の目的は，メモリストレージとブロックストレージを組み合わせ，ブロックストレージの容量を持つメモリストレージを提供すること，である．メモリストレージを提供することは，即ち，メモリストレージのインタフェースを提供することである．

この目的のために実現すべき要件を以下にまとめる．

(1) ブロックストレージの大きさのアドレス空間を提供する．
(2) 提供するアドレス空間の範囲内で指定されたブロックの最新データを提供する．
(3) アクセスに際しては，基本的には同期的なインタフェースを提供する．
(4) 仮想アドレス空間へマップ可能にする．そのために，

ⓒ 2014 Information Processing Society of Japan

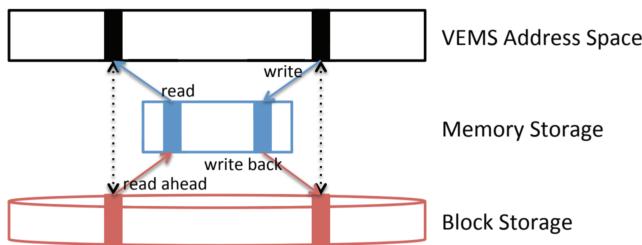

図 3 提案手法の概観

少なくともページフレームのサイズの領域に分割し管理する．

(1), (2) は，組み合わせにより提供されるストレージが，ブロックストレージと同等に機能することを意味する．(1) は，容量はブロックストレージと等しくなることを意味する．(2) は，最新データが，メモリストレージにあればメモリストレージから，ブロックストレージにあればブロックストレージから提供されることを意味し，古く無効になったデータが提供されることがないことを意味する．

(3), (4) は，そのストレージが，基本的にはメモリストレージとしてアクセス可能になることを意味している．(3) は，メモリストレージへのアクセスは同期的であること，(4) は，メモリストレージは仮想アドレス空間へマップ可能であることを意味する．

3.3 VEMS: Virtually Extended Memory Storage

本論文は，メモリストレージの容量を拡張する手法 VEMS (Virtually Extended Memory Storage) を提案する．図 3 に提案手法の概観を示す．VEM はブロックストレージと同じ大きさのアドレス空間を提供し，VEMS とブロックストレージのアドレスは 1 対 1 の対応をとる（黒破線）．メモリストレージは，ブロックストレージよりも容量が小さいため，VEMS のアドレスと 1 対 1 に対応する特定の領域は存在せず，ある領域はその時々で異なる VEMS のアドレスのデータを格納することとなる（赤実線）．ある VEMS のアドレスに対応するデータをメモリストレージ上に格納することで，メモリストレージのインタフェースを提供可能にする（青実線）．

以下に，VEMS の実現手法をまとめる．

- メモリストレージは，プロセッサのページフレームサイズに分割し管理する．分割した各領域が，ブロックストレージのどのブロックに対応するか，また各領域の状態遷移の情報を，各領域の属性として別途管理する．
- メモリストレージに書き込まれたデータは，適宜，ブロックストレージへの書き込みを行うことで，メモリストレージ上の再利用可能な領域を確保する．
- メモリストレージには，メインメモリを仮想アドレス空間へのマップを管理するために用いられる，ページフレーム管理のためのデータ構造体 `struct page` を割り当てる．`struct page` を割り当てることで，あるページが仮想アドレス空間にマップされた状態にあるかどうか，またアンマップ後にそのページへの書き込みが起こったかどうかの情報を取り出すことが容易になる．そこで，ページへの書き込みが起こったかどうかの管理は，`struct page` に統合する．
- 必要に応じて，データアクセスに際し，メモリストレージへの先読みやメモリストレージをバイパスする処理を行う．読み出しに際しては，メモリストレージ上にデータがない場合，ブロックストレージからの読み出しには大きな遅延が伴うため，それを回避するための先読み処理を行う．また，アクセスにあたり，メモリストレージを活用できない場合，できる状態になるまで待つのではなく，メモリストレージをバイパスし，ブロックストレージへアクセスする．

メモリストレージは，仮想アドレス空間へマップ可能にするため，プロセッサのページフレームと同じサイズに分割し管理する．そのため，メモリストレージとブロックストレージの間では，ページフレームサイズでデータをやりとりする．しかしながら，一般に，ブロックストレージは 512 バイトのセクタに分割され，この単位でアドレスが割り振られている．ストレージは，最小アクセス単位のサイズを OS カーネルに伝えることで，基本的にはそのサイズでのみアクセスされるが，マウント時等で例外的にセクタ単位でのアクセスが要求される場合もある．セクタサイズはページフレームサイズよりも小さい場合が多いため，ページフレームサイズにアラインされていないアクセスにも，対応が必要である．

メモリストレージに書き込まれたデータは，適宜，ブロックストレージへの書き込みを行う．メモリストレージへの書き込が起こった領域は，最新データがメモリストレージ上にあるため，ブロックストレージへの書き込みが終了するまで，ブロックストレージの異なるブロックのデータを置くために再利用できない．メモリストレージ上の，ブロックストレージへの書き込みが終了した領域，またブロックストレージから読み出されただけで書き込まれていない領域は，再利用可能である．

ブロックストレージへの書き込みは，ブロックストレージ上で連続するブロックを検索し，できるだけ大きなサイズで書き込みを行う．SSD であっても，シーケンシャルアクセスの方が高速であるため，連続ブロックを構成することで，書き込みの効率を高めることができる．

メモリストレージのある領域に書き込みが起こったかどうかは，write システムコールを通して書き込む場合は，容易に把握することが出来る．しかしながら，XIP を通して，ユーザプロセスの仮想アドレス空間にマップされた場合，書き込みがカーネルの実行を伴うとは限らない．この

ⓒ 2014 Information Processing Society of Japan

場合，ページテーブルエントリに含まれる dirty ビットの情報を使用する必要がある．この情報は，アンマップ時に struct page に反映されることから，取り出すことができる．そのため，ページへの書き込みが起こったかどうかの管理は，struct page に統合する．

上述したように，VEMS は，メモリストレージに struct page を割り当て，struct page によって提供される機能を活用する．struct page の割り当ては，メインメモリを消費する．そのため，メモリストレージがブロックストレージ相当の容量を提供する場合に，struct page を割り当てることには慎重な検討が必要である．しかし，VEMS が対象とするシステム構成では，メモリストレージとして MRAM を想定しており，その容量としてはメインメモリ程度が考えられるため，struct page を割り当てることに問題はない．

遅延を削減するため，データアクセスに際し，必要に応じて，メモリストレージへの先読みやメモリストレージをバイパスする処理を行う．読み出しに際しては，メモリストレージ上にデータがない場合，ブロックストレージからの読み出しには大きな遅延が伴う．先読み処理を行い，予めデータをメモリストレージ上に置くことで，遅延を回避することができる．また，書き込みに際して，メモリストレージ上に使用可能領域がない場合，使用可能な領域ができるまで待つと，大きな遅延が生じる．また，強制的にメモリストレージ上のデータのブロックストレージへの書き込みを起動し，使用可能領域を確保しようとしても，小さなサイズの書き込みを行うことになり，書き込みサイズに対する遅延は大きい．そこで，メモリストレージ上に使用可能領域がない場合，メモリストレージをバイパスし，ブロックストレージへのアクセスを行う．メモリストレージをバイパスする処理は，読み出し時にメモリストレージ上にデータがない場合にも行う．

4. 実装

3.3 節で述べた提案手法の，Linux における具体的な実現方法について述べる．まず，メモリストレージの管理について述べた後に，XIP をベースしたメモリストレージへのインタフェースを提供する実装について述べる．そして，従来のブロックデバイスインタフェースを提供する実装について述べる．

4.1 メモリストレージの管理

メモリストレージは，プロセッサのセット連想方式キャッシュと同様に管理する実装を行った．セット連想方式キャッシュとしたのは，メモリストレージがメインメモリ相当の容量の場合であっても，ブロックストレージのブロックアドレスに対応するメモリストレージ領域の検索を，単純なアルゴリズムにより一定コストで行えるようにするためである．メモリストレージを，プロセッサのページフレームサイズに分割したデータ領域が，キャッシュラインに相当することになる．セット数は，デバイスドライバのモジュールパラメータで指定可能である．デフォルトのセット数は 16 とした．

セット連想方式キャッシュでは，各キャッシュラインのデータのメインメモリでのアドレスや状態を，対応するタグに格納し，キャッシュラインを管理する．VEMS でのメモリストレージの管理でも，同様に，メモリストレージを分割した各データ領域にもタグを付与する．タグは，ブロックストレージのセクタアドレスを格納するために十分な大きさとし，さらに状態遷移情報を格納する．

メモリストレージのタグは，ページフレームサイズに分割したデータ領域とは別に，まとめて管理する．データ領域とタグ領域を別にすることで，各データ領域の先頭アドレスは，無駄なくページフレームサイズにアラインすることができる．

4.2 VEMS インタフェースの提供

VEMS が提供する，メモリストレージへのインタフェースの実装について述べる．メモリストレージへのインタフェースは，XIP のインタフェースを元に，拡張したものとなっている．

まず，元となった XIP のインタフェースの概要について述べる．XIP を有効にしてマウントされたファイルシステムのファイルを，read, write システムコールによりアクセスする場合，xip_file_read(), xip_file_write() 関数が呼ばれる．どちらの関数も，ファイルシステムが提供する get_xip_mem() インタフェースを呼び出すことで，アクセス先のページフレームのアドレスを取得し，データの読み出し，または書き込みを行う．ファイルシステムが直接メモリストレージを管理しない場合，メモリストレージを管理するドライバが提供する direct_access() インタフェースを呼び出し，セクタアドレスに対応するメモリストレージのアドレスを取得する．

VEMS が，XIP のインタフェースをそのまま使用すると，以下の情報および機能不足により，処理を効率的に行うことができない．

- アクセスが読み出しなのか，書き込みなのかの情報が渡されない．そのため，読み書きの如何に関わらず，書き込みとみなす必要があり，無駄なブロックストレージへの書き戻しが必要になる．
- アクセスするデータサイズの情報が渡されない．そのため，書き込み時にアクセス先アドレスのデータがメモリストレージ上にない場合，全て新しいデータで上書きされてしまう場合でも，無駄にブロックストレージからメモリストレージへの読み込みが必要になる．
- 書き込み時にアクセス先アドレスのデータがメモリス

ⓒ 2014 Information Processing Society of Japan

トレージ上にない，またはメモリストレージ上に再利用可能なデータ領域がないことを，`xip_file_read()`, `xip_file_write()` 関数のレベルで知ることができない．そのため，要求されたデータサイズに対応して，メモリストレージをバイパスする処理を行うことができない．

- XIP は，全てのデータがメモリストレージ上に置かれることを前提としている．そのため，`xip_file_read()` 関数が，ブロックストレージからメモリストレージへの先読みを指示するためのインタフェースを提供していない．

上記の情報不足を解消するため，XIP のインタフェースの引数を拡張し，VEMS のためのインタフェースを定義した．拡張したインタフェースの実装では，上記の情報を用いた処理を行い，メモリストレージとブロックストレージ間の無駄なデータの移動を省き，処理を効率化する．既存インタフェースの引数の拡張では，メモリストレージをバイパスする処理を要求するためのインタフェース，先読みを指示するインタフェースが不足するため，それらを追加した．

メモリストレージをバイパスする処理は，現状では読み出しについてのみ行っている．`xip_file_read()` に対応する VEMS の関数から，メモリストレージをバイパスし，ブロックストレージからデータを読み出す場合，同期的な処理となる．そのため，ある程度大きなサイズの連続したブロックに対して読み出しを行わないと，性能が低下する．そのために，ファイルのブロック情報を取得することのできる `xip_file_read()` に対応する VEMS の関数から，メモリストレージをバイパスするインタフェースを呼び出す必要がある．また，ページキャッシュは用いられないため，読み出し先は，ユーザプロセスのバッファ領域となる．

4.3 ブロックデバイスインタフェースの提供

VEMS は，メモリストレージのインタフェースでの性能と比較するため，従来のブロックデバイスのインタフェースも提供する．

VEMS は，複数のストレージを組み合わせる．Linux は，複数のブロックストレージを組み合わせるためのフレームワークとして，device mapper を提供している．device mapper は，例えば，複数のブロックストレージから 1 つの論理ボリュームの構成を可能にする LVM (Logical Volume Manger) の実現に用いられている．VEMS のブロックデバイスのインタフェースは，当初，メモリストレージを管理するラムディスクドライバとブロックストレージのドライバを，device mapper により組み合わせる実装を検討した．しかしながら，プロトタイプ実装の評価から，device mapper を経由することのオーバヘッドが若干あり，メモリストレージに対応するための改変コストを考慮すると，これを用いることによる優位性はないと判断し，device mapper は用いない実装とした．

ブロックデバイスのインタフェースには，基本的には 2 種類，1) ブロックストレージへのアクセスをより効率良くするため，アクセス要求のスケジューリングを行うものと，2) 単純にアクセス要求をブロックストレージへ渡すだけのもの，がある．VEMS でのブロックデバイスのインタフェースとしては，2) を実装した．2) の実装には，`blk_queue_make_request()` 関数を呼び出し，ブロック単位でのアクセス要求を受け取る関数を登録する．以下に，VEMS で登録する関数のプロトタイプ宣言を示す．

```
void vems_make_request(struct request_queue *q,
    struct bio *bio)
```

`vems_make_request()` 関数は以下の処理を行う．各アクセス要求は，第 2 引数 `bio` で渡される．読み出しの場合，要求されたデータがメモリストレージ上にあるか検索する．あれば，そのデータを `bio` が指定するバッファ領域にコピーする．なければ，ブロックストレージのドライバに `bio` を転送し，ブロックストレージから読み出しを行う．書き込みの場合，書き込み先のアドレスのデータがメモリストレージ上にあるか，もしなければ，メモリストレージ上に再利用可能なデータ領域があるか検索する．あれば，`bio` が指定するデータをメモリストレージへコピーする．なければ，ブロックストレージのドライバに `bio` を転送し，ブロックストレージへ書き込みを行う．メモリストレージへのアクセスで処理が完結した場合，`vems_make_request()` 関数での処理の最後に，`bio_endio()` 関数を呼ぶことで，アクセス要求の同期的な処理が可能になる．一方，ブロックストレージのドライバに転送した `bio` は，ブロックストレージのドライバが非同期的に処理する．

なお，`bio` は連続する領域へのアクセス要求を指定することができる．VEMS では，`bio` がページフレームサイズを超えたサイズの要求を含むと，処理が煩雑となるため，1 つの `bio` にページフレームサイズを超えたサイズの要求が入らないように設定している．

5. 実験結果

本章では，VEMS を用いて実験を行った結果を示す．まず，実験環境をまとめ，ファイルアクセス性能を示す．そして，読み出し性能について，先読みとバッファサイズの影響を示す．

5.1 実験環境

MRAM を装備したシステムは一般に入手することは困難であるため，実験には MRAM の代わりに通常の DRAM を用いた．実験には，Intel Core i7-920 2.66GHz を搭載する PC 互換機を用いた．メインメモリに 256MB，メモリストレージに対応するメモリに 256MB を割り当てた．ブ

ⓒ 2014 Information Processing Society of Japan

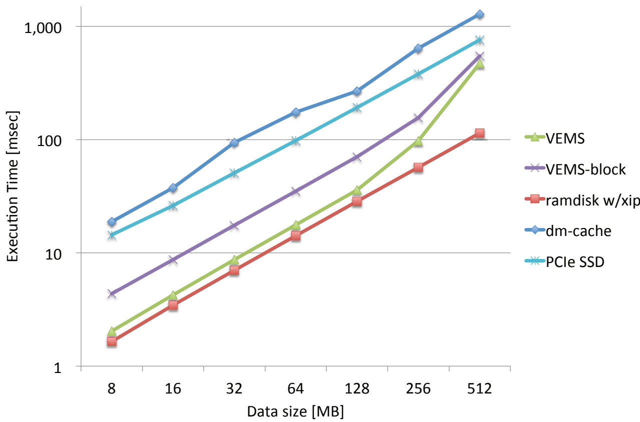

図 4 ファイル読み出し性能の比較

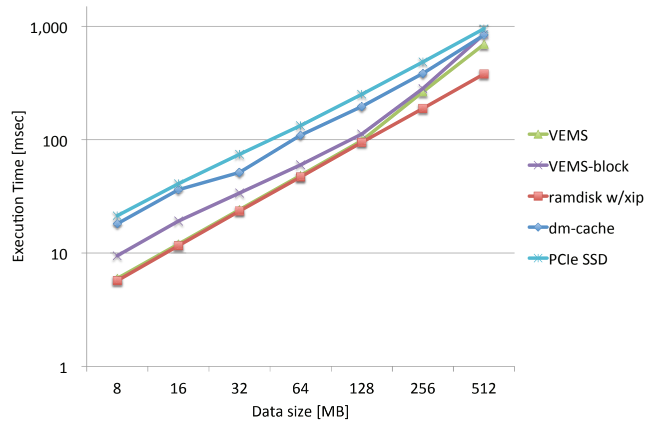

図 5 ファイル書き込み性能の比較

ロックストレージには，PCIe Gen2 x2 接続の Plextor M6e PCI Express SSD を用いた．

VEMS を実装した Linux カーネル 3.14.12 上で，ファイルの読み出し，書き込みを行うベンチマークプログラムを実行し，実行時間を計測した．実行時間の計測には，TSC (Time Stamp Counter) を用いた．

比較対象は，VEMS のメモリストレージをアクセスする部分を取り出して作成したラムディスクドライバ，Linux カーネルに含まれている device mapper を用いてラムディスクとブロックストレージの組み合わせを可能にする dm-cache，そして SSD である．ラムディスクのみの性能計測時には，4GB のメモリを割り当てた．

5.2 ファイルアクセス性能

ファイルを作成し書き込みを行い，次にそのファイルの読み出しにかかった実行時間を計測した結果を，図 4, 5 に示す．図では，実行時間を示す縦軸はログスケールとなっている．VEMS は 4.2 節で述べた VMES インタフェースを使用した場合，VEMS-block は 4.3 節で述べた従来のブロックインタフェースを使用した場合を表す．ファイルを新たに作成し，書き込みを行っているため，書き込みに関しては，ファイルへのブロック割り当てのコストを含む．また，書き込みと読み出しの間に，ページキャッシュを無効化する操作を行っているため，読み出しに関しては，ストレージからの読み出しコストとなっている．

ラムディスクを除くと，ファイルサイズが 8MB から 512MB までの読み出し，書き込みの両方で，VEMS が最も高速，その次が VEMS-block という結果となった．VEMS は，8MB から 128MB までは，ラムディスクに近い性能となっており，読み出しで 23~26%，書き込みでは 3~5%の性能低下に留まっている．VEMS-block と比較すると，読み出しで 96~115%，書き込みでは 14~59%高速である．メモリストレージのサイズと同じ 256MB，またそれを越える 512MB のファイルサイズとなると，読み出し，書き込みともに，VEMS-block の性能に近づいてはいるが，ファイルサイズ 512MB では，読み出しで 16%，書き込みで 23%，VEMS の方が高速である．

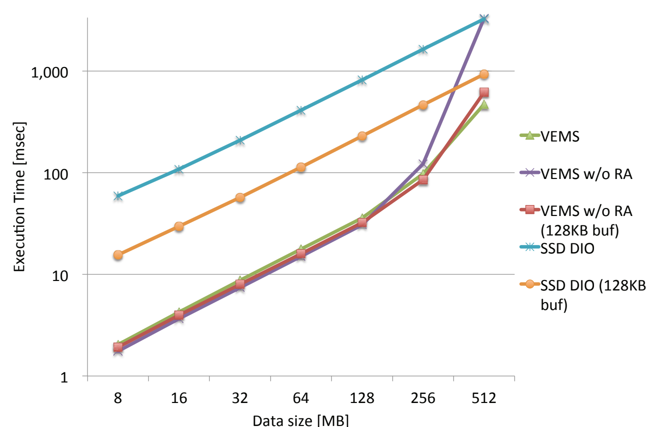

図 6 ファイル読み出し性能への先読みの影響

5.3 先読みとバッファサイズの影響

VEMS によるファイルアクセス高速化の要因を解析するため，読み出しの場合について，先読みを行わない場合，先読みを行わずユーザプロセスにおけるバッファ領域を拡大した場合について計測を行った．ユーザプロセスにおけるバッファ領域は，デフォルトでは BUFSIZ マクロに定義された 8KB を用いており，拡大した場合は 128KB とした．また，VEMS はページキャッシュを介さずに読み出しを行うため，比較のため，SSD からダイレクト I/O (DIO) による読み出し性能を計測した．なお，DIO は先読みを行わない．計測した結果を図 6 に示す．

結果からは，ファイルサイズ 512MB では，デフォルトのバッファ領域サイズにおいて，VEMS の先読みを行う場合と行わない場合の性能差は 7.0 倍あり，先読みを行わない場合の性能劣化が著しいことがわかる．先読みを行わない VEMS は，その結果が DIO とほぼ同じになっていることから，バッファ領域のサイズで同期的に読み出した結果，性能劣化を引き起こしていると考えられる．バッファ領域サイズを 128KB に拡大することで，先読みを行わない

ⓒ 2014 Information Processing Society of Japan

VEMS と DIO の性能は改善され，VEMS の先読みを行う場合と行わない場合の性能差は 31%まで縮まる．バッファ領域サイズの拡大は，一度のアクセス要求で SSD から読み出すデータ量を増加するため，全体としては読み出しの遅延が減少し，読み出し性能が向上する．それでも，先読みを行わない VEMS は VEMS-block 以下の性能に留まっている．上記の結果から，読み出し性能の向上には，先読みの影響が大きいことがわかる．

6. 関連研究

複数のブロックストレージの組み合わせにより，アクセス性能を向上させる初期の試みとしては，DCD [10] がある．DCD は，SSD 出現以前に，ブロックストレージはシーケンシャルアクセスの方が高速であることに着目し，キャッシュとするブロックストレージに，シーケンシャルアクセスを行うようにすることで，高速化を実現した．その後，高速なブロックストレージとして SSD が出現したことにより，同様な手法の研究開発が行われた [3], [4], [5], [6]．これらはいずれも，複数のブロックストレージを組み合わせるものであり，メモリストレージとブロックストレージを組み合わせ，メモリストレージのインタフェースを提供する VEMS とは異なる．

eNVy [11] は，バイトアクセス可能かつ読み出しは DRAM 相当の性能を持つ NOR 型フラッシュメモリに，SRAM を書き込みバッファとして組み合わせ，メインメモリの一部として使用可能にするシステムである．バイトアクセス可能なメモリを組み合わせている点が VEMS と類似しているが，ブロックストレージとの組み合わせでない点で異なっている．

7. まとめ

フラッシュメモリよりも記憶デバイスとしての性能が遙かに高い MRAM, PCM (phase change memory) , ReRAM といった次世代不揮発性メモリ（NV メモリ）の実用化が進んでおり，これらのメモリを用いた SSD の研究開発も行われている．そのなかでも MRAM は，DRAM に相当する性能と耐久性を持つという点で優れているが，集積度では劣っているため，ストレージの主体デバイスとしての用途は，現状では考えにくい．そこで，大容量のブロックストレージと組み合わせ，MRAM の容量を仮想的に拡張して用いることで，その高速性と不揮発性を活かす手法として VEMS (Virtually Extended Memory Storage) を提案した．VEMS は，Linux カーネルに実装し，実験結果から，従来手法よりも低コストなアクセスを実現できていることがわかった．

参考文献

[1] Akel, A., Caulfield, A. M., Mollov, T. I., Gupta, R. K. and Swanson, S.: Onyx: a protoype phase change memory storage array, *Proceedings of the 3rd USENIX conference on Hot topics in storage and file systems*, HotStorage'11, Berkeley, CA, USA, USENIX Association, pp. 2–2 (online), available from ⟨http://dl.acm.org/citation.cfm?id=2002218.2002220⟩ (2011).

[2] Tanakamaru, S., Doi, M. and Takeuchi, K.: Unified solid-state-storage architecture with NAND flash memory and ReRAM that tolerates 32x higher BER for big-data applications, *2013 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pp. 226–227 (online), DOI: 10.1109/ISSCC.2013.6487711 (2013).

[3] Kgil, T. and Mudge, T.: FlashCache: A NAND Flash Memory File Cache for Low Power Web Servers, *Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, CASES '06, New York, NY, USA, ACM, pp. 103–112 (online), DOI: 10.1145/1176760.1176774 (2006).

[4] Facebook: FlashCache, https://github.com/facebook/flashcache (2014).

[5] Saxena, M., Swift, M. M. and Zhang, Y.: FlashTier: A Lightweight, Consistent and Durable Storage Cache, *Proceedings of the 7th ACM European Conference on Computer Systems*, EuroSys '12, New York, NY, USA, ACM, pp. 267–280 (online), DOI: 10.1145/2168836.2168863 (2012).

[6] Koller, R., Marmol, L., Rangaswami, R., Sundararaman, S., Talagala, N. and Zhao, M.: Write Policies for Host-side Flash Caches, *Proceedings of the 11th USENIX Conference on File and Storage Technologies*, FAST'13, Berkeley, CA, USA, USENIX Association, pp. 45–58 (online), available from ⟨http://dl.acm.org/citation.cfm?id=2591272.2591278⟩ (2013).

[7] Song, Y.-J., Jeong, G., Baek, I.-G. and Choi, J.: What Lies Ahead for Resistance-Based Memory Technologies?, *Computer*, Vol. 46, No. 8, pp. 30–36 (online), DOI: http://doi.ieeecomputersociety.org/10.1109/MC.2013.221 (2013).

[8] 藤田忍，安部恵子，野村久美子，野口紘希：ノーマリーオフコンピューティング:3. 携帯情報端末におけるノーマリーオフコンピューティング-STT-MRAM で実現するノーマリーオフメモリ技術-，情報処理，Vol. 54, No. 7, pp. 668–676（オンライン），入手先 ⟨http://ci.nii.ac.jp/naid/110009579885/⟩ (2013).

[9] Supporting filesystems in persistent memory, http://lwn.net/Articles/610174/ (2014).

[10] Hu, Y. and Yang, Q.: DCD – Disk Caching Disk: A New Approach for Boosting I/O Performance, *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, pp. 169–178 (online), DOI: 10.1109/ISCA.1996.10021 (1996).

[11] Wu, M. and Zwaenepoel, W.: eNVy: a non-volatile, main memory storage system, *Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS VI, New York, NY, USA, ACM, pp. 86–97 (online), DOI: 10.1145/195473.195506 (1994).

© 2014 Information Processing Society of Japan