

メモリストレージ容量拡張手法 VEMS の アプリケーションによる評価

追川 修一¹

概要: メモリストレージは、メモリインタフェースを提供するストレージである。次世代不揮発性メモリ (NV メモリ) や NV-DIMM 等の開発が進み、メモリストレージの実用化が近いと考えられる。メモリストレージは、現在広く用いられているブロックストレージよりも高性能である一方、容量に制限がある。メモリストレージ容量拡張手法 VEMS (Virtually Extended Memory Storage) 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、メモリストレージの容量を仮想的に拡張する手法である。VEMS の評価結果としては、単純なベンチマークプログラムを用いた実験結果のみがあった。そこで、実際に VEMS が有効となるアプリケーションを明確にしていく必要がある。本論文は、VEMS をアプリケーション環境を用いて評価するための、実験を行った結果を示す。

1. はじめに

メモリストレージは、メモリインタフェースを提供するストレージである。現在、次世代不揮発性メモリ (NV メモリ) や NV-DIMM 等の開発が進んでおり、メモリストレージの実用化が近いと考えられる。NV メモリとしては、MRAM, PCM (Phase Change Memory), ReRAM 等の開発が行われている。PCM, ReRAM は書き込み時の性能や耐久性に制限があるが、MRAM はそのような制限がなく DRAM に相当する性能を持つ点で優れている。また、NV-DIMM には、DRAM の DIMM にキャパシタ等を搭載し、電源断時にキャパシタ等からの電源供給によりデータを保持するものや、フラッシュメモリへデータを退避するものがある。

MRAM や NV-DIMM といったメモリストレージは、DRAM 相当の性能を提供するため、現在広く用いられているブロックストレージよりも遙かに高性能である一方、容量に制限がある。従って、これらのメモリストレージのみが構成するストレージは非常に高価となるため、その用途は限られたものになると考えられる。しかしながら、メモリストレージは、そのメモリインタフェースによる直接アクセスが可能であり、またその性能から同期アクセスが可能であることから、処理コストの大きいブロックデバイスドライバが不要または大幅に簡素化でき、アクセスコストを低減できるという長所がある。

メモリストレージ容量拡張手法 VEMS (Virtually Extended Memory Storage) [1] 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、メモリストレージの容量を仮想的に拡張し、ブロックストレージの容量を提供する手法である。VEMS は、メモリストレージとしてのアクセスを提供する。そのため、メモリストレージの特徴である、直接同期アクセスによるアクセスコストの低減が可能である。一方、ブロックストレージと組み合わせることで、ブロックストレージと同じ容量を提供する。透過的に大容量を提供可能にすることで、メモリストレージをストレージの主体デバイスの一部として利用可能にする。VEMS は Linux カーネルのデバイスドライバとして実装されている。ファイルシステムからのアクセスを効率化するため、VEMS は、XIP^{*1} インタフェースを拡張したファイルシステムインタフェースを必要とする。VEMS ファイルシステムインタフェースは、Ext2 に実装されている。

VEMS の評価結果としては、単純なベンチマークプログラムを用いた実験結果のみがあった。そこで、実際に VEMS が有効となるアプリケーションを明確にしていく必要がある。本論文は、VEMS をアプリケーション環境を用いて評価するための、実験を行った結果を示す。実験には、大規模データ分散処理環境である Hadoop、およびインターネットサーバのファイルアクセスを模擬する Postmark [2] を用いた。実験結果から、実際にアプリケーション環境においても VEMS が有効であると考えられることがわかった。

¹ 筑波大学 システム情報系情報工学科
University of Tsukuba, Tsukuba, Ibaraki 305-8573, Japan

^{*1} XIP: eXecution-In-Place

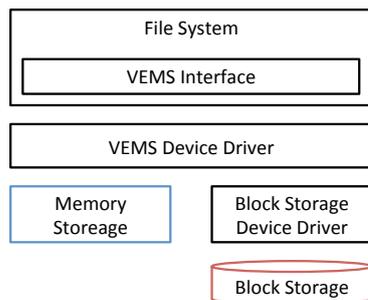


図 1 VEMS の実装形態

以下、2章で、背景となる技術としてVEMSについて述べる。3章は、アプリケーション環境を用いた実験結果を示し、4章は今後の課題を述べる。5章で関連研究について述べ、6章で本論文をまとめる。

2. 背景

背景となる技術として、メモリストレージ容量拡張手法VEMS (Virtually Extended Memory Storage) [1]について述べる。VEMS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て、メモリストレージの容量を仮想的に拡張する手法である。VEMSは、メモリストレージとしてのアクセスを提供する一方、ブロックストレージと組み合わせることで、ブロックストレージと同じ容量を提供する。そのため、メモリストレージの特徴である直接同期アクセスによりアクセスコストを低減可能であり、また透過的にブロックストレージの大容量が利用可能である。

VEMSはLinuxカーネルのデバイスドライバとして実装されている。ファイルシステムからのアクセスを効率化するため、VEMSは、XIPインタフェースを拡張したファイルシステムインタフェースを必要とする。現状では、VEMSファイルシステムインタフェースは、Ext2に実装されている。ブロックストレージへのアクセスには、そのためのブロックデバイスドライバが必要になる。ブロックデバイスドライバは、VEMSデバイスドライバが必要に応じて呼び出す。従って、アクセス要求の処理がメモリストレージへのアクセスで済む場合は、ブロックデバイスドライバは呼び出されない。図1に、VEMSの実装形態を示す。

VEMSの特徴を以下にまとめる。

- 同期・非同期アクセスの適応的切替
- ページキャッシュが不要
- メモリストレージへの書き込みによるデータ永続化

以下、それぞれについて述べる。

同期・非同期アクセスの適応的切替は、メモリストレージは同期アクセス、ブロックストレージは非同期アクセスであることから、アクセス先に応じて、同期・非同期を切り替えることで実現する。読み出しの場合、データがメモリストレージ上にあれば、同期アクセスとなる。書き込み

の場合、メモリストレージ上に書き込み可能領域があれば、同期アクセスとなる。それ以外の場合は、ブロックストレージへのアクセスが必要となるため、非同期アクセスとなる。このように必要に応じて同期・非同期を切り替えることで、高性能を実現する。

VEMSはXIPインタフェースを拡張したファイルシステムインタフェースを提供し、メモリストレージへの直接アクセスを行っている。直接アクセスできないブロックストレージは、ページキャッシュを経由することでアクセス処理を効率化するが、直接アクセス可能なメモリストレージはページキャッシュの機能をはたすため、メモリストレージとは別にページキャッシュを用いる必要はない。そのため、VEMSへのアクセス時には、ページキャッシュは用いられない。従って、メモリストレージから読み出す場合は、ページキャッシュを経ずに、データはユーザプロセスへ直接コピーされる。また、書き込みの場合は、ユーザプロセスからメモリストレージへ、直接コピーされる。

メモリストレージは、メモリアクセスインタフェースを提供するが、そのメモリは不揮発性である。従って、メモリストレージへ書き込んだデータは、その時点で永続化が保証される。ページキャッシュは、揮発性のメインメモリから割り当てられるため、ページキャッシュへ書き込んだ時点では、データは永続化されていない。データの永続化は、ページキャッシュからブロックストレージへの書き込み終了を待つ必要がある。VEMSは、メモリストレージへ書き込み時点で、データの永続化を保証する。さらに、ページキャッシュを用いないため、アプリケーションが書き込みを終了した時点で、データが永続化されることになる。

3. 実験結果

本章は、VEMSを用い、アプリケーション環境を用いて評価する実験を行った結果を示す。まず、大規模データ分散処理環境であるHadoopを用いて実験を行った。Hadoopを用い、ファイル入出力のスループットおよびソートの性能を計測した。次に、大量の小さいファイルを処理するインターネットサーバのファイルアクセスを模擬するプログラムの性能を計測した。

3.1 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、MRAMの代わりに通常のDRAMをメモリストレージとして用い、実験を行った。実験は、Intel Core i7-4930K 3.4GHz CPU、64GBメモリを搭載するPC互換機上にCentOS 7 Linuxをインストールし、KVMにより仮想化環境を構築したうえで、仮想マシン上でベンチマークプログラムを実行した。仮想マシンのメインメモリには1GBを割り当てた。各仮想マシンには、CentOS 6 Linuxをインストールし、VEMSを実装したLinuxカーネ

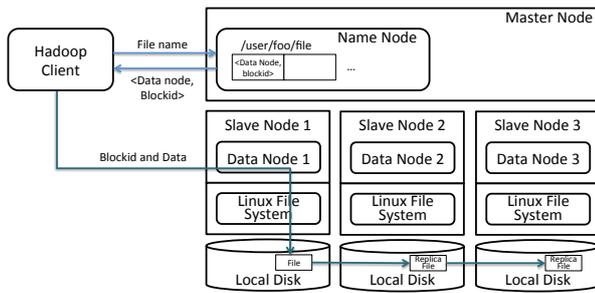


図 2 HDFS のアーキテクチャ概要

ル 3.14.12 を実行する。Linux 環境は、ディスクイメージ上に構築した Ext4 ファイルシステムにインストールし、通常の virtio ブロックデバイスを介してマウントした。

VEMS を構成するメモリストレージには 1GB を割り当て、ブロックストレージには、SATA 3.0Gbps 接続の CFD S6TNHG6Q SATA SSD を用いた。ブロックストレージを接続する仮想マシンごとに 1 台の SSD 全体を割り当て、ディスクイメージではなく、仮想マシンが SSD に直接アクセスする設定とした。これらの SSD に対する KVM のキャッシュモードは none とし、KVM はライトバックキャッシュを提供するが、ホスト OS のページキャッシュは用いない設定とした。VEMS 上に構築するファイルシステムとしては Ext2 を用いた。VEMS は、それぞれの実験におけるファイル入出力の対象となる箇所のみ用いた。

比較対象は SSD 単体として、実験を行った。この SSD は、上記のブロックストレージに用いたものと同一である。

3.2 Hadoop による計測

大規模データ分散処理環境である Hadoop を用いて実験を行った。まず Hadoop の実験環境を示し、次に TestDFSIO を用いて計測したファイル入出力のスループット性能の結果、Terasort を用いて計測したソートの性能の結果を示す。

3.2.1 Hadoop の実験環境

Hadoop は分散処理環境であるため、基本的にマスタノードと複数のスレーブノードがその環境を構成する。Hadoop は、大規模データを独自の方法で管理するため、ファイルサービスとして HDFS (Hadoop Distributed File System) を用いる [3]。図 2 に HDFS のアーキテクチャ概要を示す。HDFS は、マスタノード上でネームノードを実行し、スレーブノード上でデータノードを実行する。ネームノードはディレクトリサービスを提供し、ファイルはデータノードが管理する。Hadoop は、できるだけ処理対象のデータがあるノードに処理を割り当てることで、データ取得のオーバーヘッドを抑える設計となっている。

Hadoop を用いた実験を行うため、複数の仮想マシンを構成し、Hadoop 実行環境を構築した。1 つの仮想マシンをマスタノード、3 つの仮想マシンをスレーブノードとし

た。HDFS へのファイル入出力はデータノードでのみ行われるため、データノードを実行するスレーブノードの仮想マシンにのみ、ブロックストレージは接続した。仮想マシン間の通信は、仮想ネットワークにより接続した。HDFS は、複数データノードにファイルを複製することで、データに冗長性を持たせる。複製数は、スレーブノード数と同じ 3 に設定した。これにより、HDFS への書き込みにあたり、3 つのデータノード全てに書き込みが行われることになる。

VEMS は、dfs.datanode.data.dir が指定する、データノードが HDFS のファイルのデータ格納するディレクトリにのみ用いている。

3.2.2 Hadoop TestDFSIO 単一ファイル

Hadoop は、ファイル入出力のスループットを計測するプログラムとして TestDFSIO を実装している。TestDFSIO は、引数としてファイルサイズとファイル数をとる。ここでは、ファイル数は 1 とし、ファイルサイズを変え、計測を行った。計測のため、まず、TestDFSIO により指定サイズのファイルを HDFS に書き込み、全データノードにおいてページキャッシュを無効化する処理をばさみ、TestDFSIO により HDFS から読み出した。計測結果としては、TestDFSIO が書き込み、読み出し時に出力する Throughput mb/sec の値を用いた。図 3, 4 に、それぞれ読み出しおよび書き込みの計測結果を示す。

計測結果から、VEMS は SSD よりも高い性能を発揮していることがわかる。読み出しでは 79.30% から 147.14%、平均 120.97%、書き込みでは 5.74% から 32.38%、平均 19.53%、VEMS が高いスループットとなっている。どちらも、ファイルサイズが 100MB の時に差が最も小さく、600MB の時に差が最も大きい結果となった。

HDFS のファイルは、データノードが管理する。そのため、通常、HDFS のファイルを読み出すクライアントプログラムは、データノードとの間で RPC による通信を行い、ファイルデータを取得する。しかしながら、データノードはユーザプロセスとして実現されており、HDFS のファイルは、Linux のローカルファイルシステムのファイルとして保存されている。そのため、クライアントプログラムがローカルファイルシステムから直接読み出すことも可能である。これを実現したのが、Short Circuit Read (SCR) と呼ばれる機能である。クライアントプログラムは、データノードから読み出すファイルのファイルディスクリプタを受け取り、読み出す。データノードを介さずにファイルアクセスを行い、データノードとの通信が不要になるため、読み出しのオーバーヘッドは小さくなる。

SCR を有効にして計測を行った結果を、図 5 に示す。SCR は、VEMS には特に有効であり、351.78% から 464.25%、平均 432.28%、SSD よりも高いスループットとなった。SCR を有効とした時、ファイルサイズが 100MB から 600MB ま

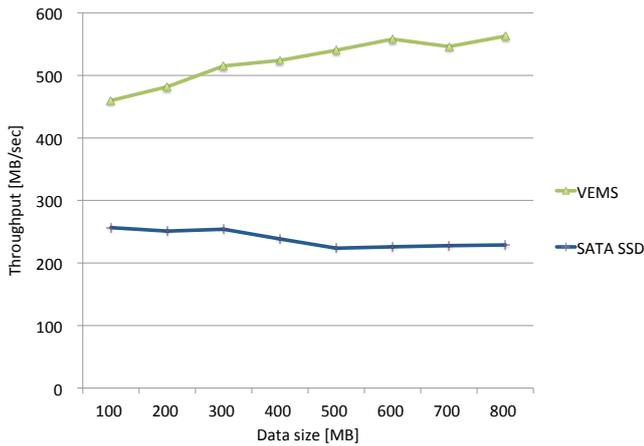


図 3 Hadoop TestDFSIO 単一ファイル読み出し性能の比較

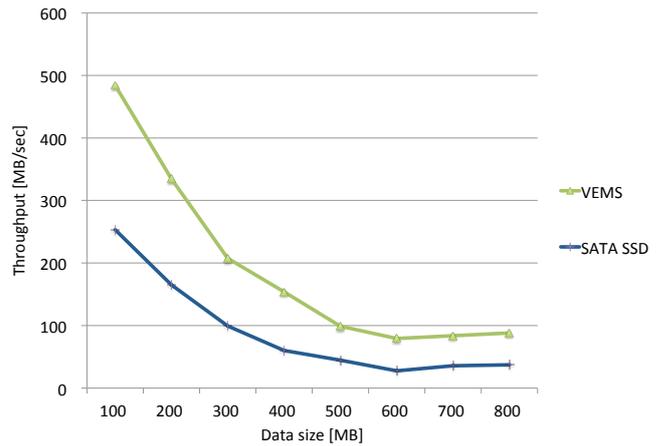


図 6 Hadoop TestDFSIO 複数ファイル読み出し性能の比較

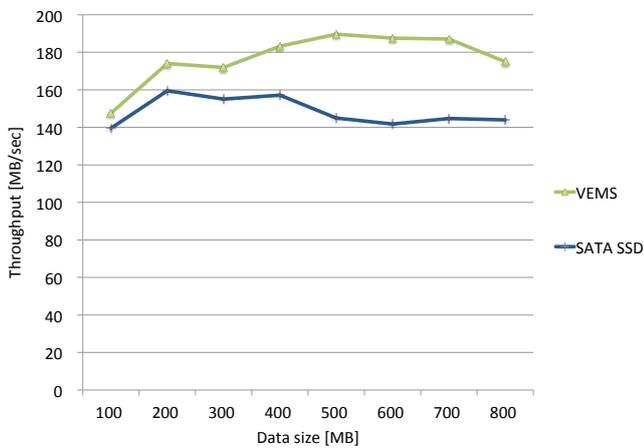


図 4 Hadoop TestDFSIO 単一ファイル書き込み性能の比較

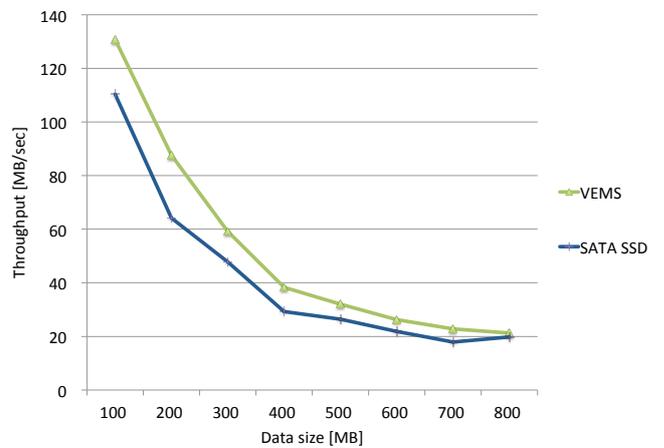


図 7 Hadoop TestDFSIO 複数ファイル書き込み性能の比較

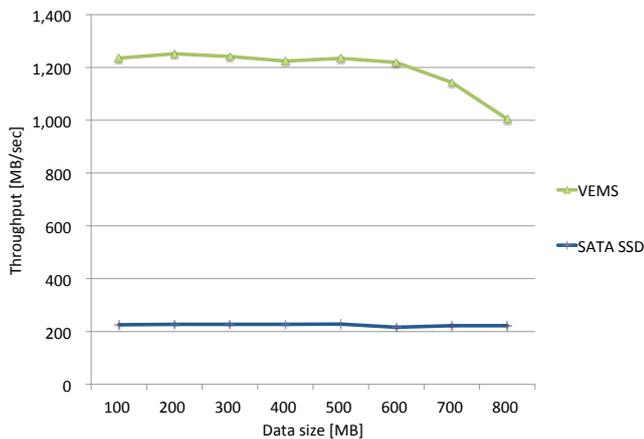


図 5 Hadoop TestDFSIO 単一ファイル読み出し性能の比較 (Short Circuit Read)

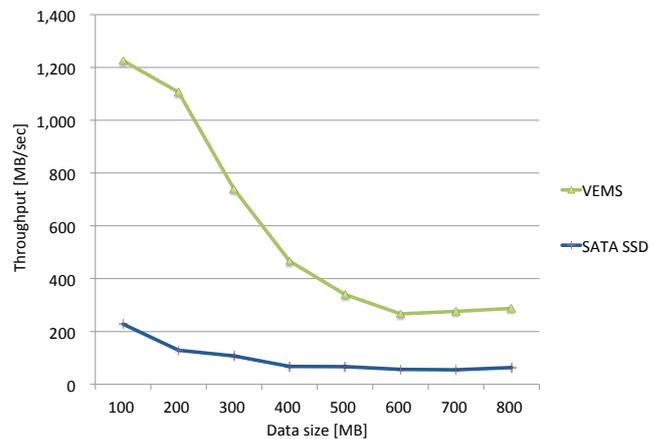


図 8 Hadoop TestDFSIO 複数ファイル読み出し性能の比較 (Short Circuit Read)

では、VEMS の性能に大きな変化はないが、それよりファイルサイズが大きくなるにつれ、性能が低下している。そのため、ファイルサイズが 800MB の時に最も差が小さくなっている。

3.2.3 Hadoop TestDFSIO 複数ファイル

次に、同じ TestDFSIO を使い、ファイルサイズは等しく 100MB とし、ファイル数を変えることで合計データサ

イズを変更し、計測を行った。計測は前項と同様に、まず、TestDFSIO により指定サイズのファイルを HDFS に書き込み、全データノードにおいてページキャッシュを無効化する処理をささみ、TestDFSIO により HDFS から読み出した。計測結果としては、TestDFSIO が書き込み、読み出し時に出力する Throughput mb/sec の値を用いた。図 6, 7, 8 に、それぞれ読み出しおよび書き込み、SCR を有効

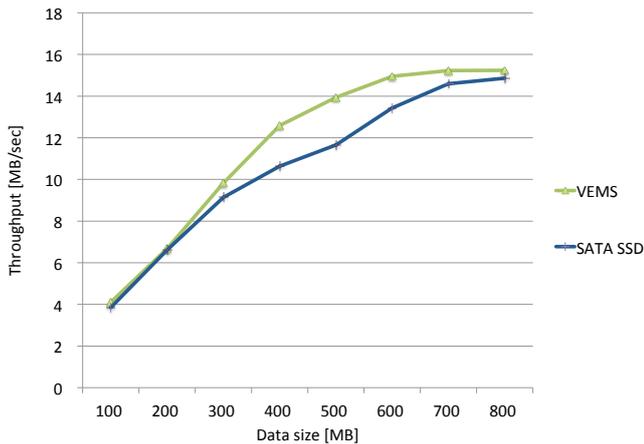


図 9 Hadoop Terasort 性能の比較

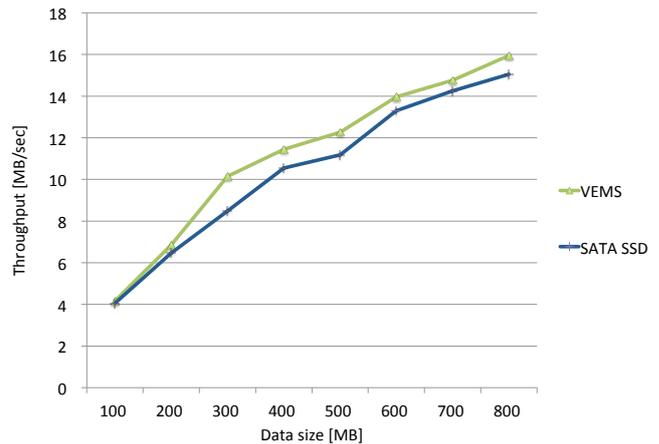


図 10 Hadoop Terasort 性能の比較 (Short Circuit Read)

にした読み出しの計測結果を示す。

計測結果から、VEMSはSSDよりも高い性能を発揮していることがわかる。読み出しでは91.09%から188.55%、平均130.01%、書き込みでは7.20%から36.49%、平均23.18%、SCRを有効にした読み出しでは350.87%から759.92%、平均487.75%、VEMSが高いスループットとなっている。これらの複数ファイル時の性能向上は、いずれも単一ファイル時の性能向上比を上回る結果となっている。

複数ファイルのファイル入出力性能と合計ファイルサイズとの相関は、単一ファイルの場合と大きく異なっている。単一ファイルの場合、ファイルサイズの変化に伴うスループットの増減は多少あるものの、単純な相関関係ではない。しかしながら、複数ファイルの場合、ファイル数とスループットは反比例の関係となっている。

3.2.4 Hadoop Terasort

HadoopにおけるMapReduce [4]を伴う処理の例として、大規模データを対象にソートを行うTerasortプログラムがある。Terasortを用いて実験を行う場合、まずTeragenプログラムが指定されたファイルサイズの処理対象データを生成し、そのデータをTerasortがソートする。Terasortは、対象データをHDFSから読み出し、ソート結果をまたHDFSに書き込む。計測時には、Teragenが生成したデータがある状態で、ページキャッシュを無効化した後に、Terasortを実行した。TestDFSIOと異なりTerasortは特に性能指標を出力しないため、timeコマンドにより計測した実行時間(Elapsed Time)を計測し、スループットを算出した。図9、10に、それぞれSCRが無効の場合と有効の場合の計測結果を示す。

計測結果から、Terasortについても、VEMSはSSDよりも高い性能を発揮していることがわかる。SCR無効の場合、1.01%から19.62%、平均8.93%、SCR有効の場合、2.74%から19.71%、平均7.64%、VEMSが高いスループットとなっている。Terasortは、入出力以外の処理も含むため、その差はTestDFSIOの場合よりも小さい。また、

TestDFSIOではSCRを有効とすることでVEMSの性能は大きく向上したが、Terasortの場合は、SCRの性能への寄与は小さいものとなっている。他にTestDFSIOの結果とは異なる点としては、データサイズとスループットとの相関関係がある。TestDFSIOでは、単一ファイルの場合は明確な相関関係はなく、複数ファイルの場合は反比例の関係となった。Terasortでは、データサイズとスループットは比例している。TerasortのスループットはTestDFSIOを大幅に下回るため、MapReduceによるソート処理の影響が大きいと考えられる。

3.3 Postmarkによる計測

Postmark [2]は、メールサーバやwebサーバといったインターネットサーバにおけるファイルアクセス性能を評価するため作成されたベンチマークプログラムである。そのようなサーバのファイル処理を模擬するため、短命で小さいサイズのファイルを大量に処理する性能を計測する。Poastmarkは、指定したディレクトリに処理対象のファイルを作成するため、そのディレクトリにのみVEMSを用い、計測を行った。計測には、表1に示す、3つの異なる設定を用いた。Small, Medium, Largeは、それぞれファイルサイズを表す。即ち、Smallは小さいファイルサイズの設定、Largeは大きめのファイルサイズの設定である。これらの設定には、既存の性能測定結果として[5]を参考にした。図11に計測結果を示す。

計測結果から、Poastmarkについても、VEMSはSSDよりも高い性能を発揮していることがわかる。Small, Medium, Largeの設定で、それぞれ6.76%、129.97%、302.63%、VEMSが高いスループットとなっている。SSDの性能は、Smallの設定で最も高く、Largeの設定で最も低い。VEMSは、Largeの設定で最も低いのは同じであるが、Medium設定で最も高くなっている。性能差は、Largeの設定の場合が最も大きい。絶対的な性能としてはVEMSのMedium設定で最も高い。

表 1 Postmark の設定

Parameters	Small	Medium	Large
size	5000 20000	30000 100000	500000 1000000
number	30000	10000	1000
transactions	300000	500000	200000

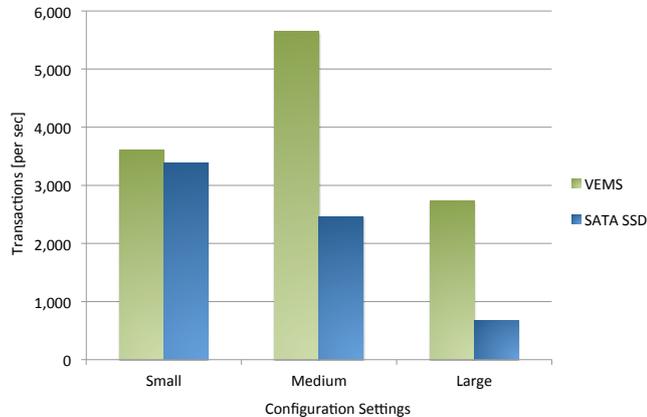


図 11 Postmark 性能の比較

4. 今後の課題

本論文は、VEMS をアプリケーション環境を用いて評価するための、実験を行った結果を示した。実際にアプリケーション環境においても VEMS が有効であると考えられる実験結果を得たが、その詳細な解釈と、VEMS における解析、そして VEMS の性能向上方法への反映が今後の課題である。

本論文で示した実験結果において、直感的に理解し難い点がいくつかある。まず、TestDFSIO の単一ファイルの場合、SSD ではファイルサイズが増加するにつれ、読み書きとみに性能がやや低下する傾向にある一方で、VEMS では向上する傾向にある点である。しかし、TestDFSIO の単一ファイルで SCR を有効とした場合、VEMS でも性能が向上する傾向はない。複数ファイルの場合は、SSD、VEMS ともに明確に性能が低下している。また、TestDFSIO では、SCR 無効の場合より有効の場合の方が、単一ファイルの場合、78.50%から 168.60%、平均 129.79%、複数ファイルの場合、158.13%から 256.00%、平均 222.58%、と高いスループットとなっている。一方で、Terasort では、ファイルサイズが 400MB から 700MB の間は、SCR 有効の場合の性能の方が低い結果となっており、TestDFSIO の結果とは異なっている。これらの点を含め、VEMS の性能向上に向け、詳細な解析を行う必要がある。

5. 関連研究

Hadoop のような大規模データ処理環境において、既存の HDD に加えて異種のストレージを導入した場合の性能への影響を調べた論文には [6], [7], [8] がある。[6] は、HDFS 上

に構築された HBase の Facebook Messages のワークロードにおける解析を行い、アクセスの少ないデータは SSD に保存するには大きすぎ、また頻りにアクセスされるデータもメモリ上に置くには大きすぎるが、小さい SSD を追加は費用対効果が高いことを明らかにした。[7] は、Hadoop における Terasort と HDFS のベンチマーク結果から、データノードに中間ファイルのストレージとして SSD またはより大容量のメインメモリを導入することで、MapReduce のシャッフルを高速化できることを示している。そして、どのような組み合わせが費用対効果が高いかの検討を行っている。[8] は、Hadoop においてより多くのアプリケーションを実行し、また HDD と SSD で同等のバンド幅を提供する設定の場合、HDFS が HDD と SSD の両方を用いバンド幅を向上させて場合での比較を行い、たとえ同等のバンド幅を提供する場合でも、ランダムアクセス性能が高い SSD を用いることで高速化されるアプリケーションがあることを明確にしている。

これらは、いずれも HDD と SSD の組み合わせについての研究であり、メモリストレージに関する研究は行われていない。

6. まとめ

メモリストレージ容量拡張手法 VEMS 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、メモリストレージの容量を仮想的に拡張する手法である。VEMS の評価結果としては、単純なベンチマークプログラムを用いた実験結果のみがあった。そこで、実際に VEMS が有効となるアプリケーションを明確にしていく必要がある。本論文は、VEMS をアプリケーション環境を用いて評価するための、実験を行った結果を示した。そして、実際にアプリケーション環境においても VEMS が有効であると考えられる実験結果を得た。その詳細な解釈と、VEMS における解析、そして VEMS の性能向上方法への反映は今後の課題である。

参考文献

- [1] 追川修一: 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, 情報処理学会コンピュータシステム・シンポジウム論文集, pp. 63-70 (2014).
- [2] Katcher, J.: PostMark a New Filesystem Benchmark, Network Appliance Technical Report TR3022 (1997).
- [3] Shvachko, K., Kuang, H., Radia, S. and Chansler, R.: The Hadoop Distributed File System, *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, MSST '10, Washington, DC, USA, IEEE Computer Society, pp. 1-10 (online), DOI: 10.1109/MSST.2010.5496972 (2010).
- [4] Dean, J. and Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters, *Proceedings of the 6th Conference on Symposium on Operating Systems Design Implementation, OSDI'04*, Berkeley, CA, USA, USENIX Association, pp. 137-149 (2004).

- [5] Microsystems, S.: File System Performance: The Solaris OS, UFS, Linux ext3, and ReiserFS, Technical White Paper (2004).
- [6] Harter, T., Borthakur, D., Dong, S., Aiyer, A., Tang, L., Arpaci-Dusseau, A. C. and Arpaci-Dusseau, R. H.: Analysis of HDFS Under HBase: A Facebook Messages Case Study, *Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST 14)*, Santa Clara, CA, USENIX, pp. 199–212 (online), available from (<https://www.usenix.org/conference/fast14/technical-sessions/presentation/harter>) (2014).
- [7] Moon, S., Lee, J. and Kee, Y. S.: Introducing SSDs to the Hadoop MapReduce Framework, *Proceedings of the 2014 IEEE International Conference on Cloud Computing, CLOUD '14*, Washington, DC, USA, IEEE Computer Society, pp. 272–279 (online), DOI: 10.1109/CLOUD.2014.45 (2014).
- [8] Kambatla, K. and Chen, Y.: The Truth About MapReduce Performance on SSDs, *28th Large Installation System Administration Conference (LISA14)*, Seattle, WA, USENIX Association, pp. 118–126 (online), available from (<https://www.usenix.org/conference/lisa14/conference-program/presentation/kambatla>) (2014).