Vol.2016-OS-136 No.4
2016/2/29

KSM とバルーニングを用いた仮想化環境における I/O 性能の向上

徳田大輝[†1]　御代川翔平[†1]　山口実靖[†1]

概要：クラウド環境の普及により，多くのサーバは消費電力や設置スペースの肥大化が問題となっている．この解決策の 1 つとして仮想化技術を用いて複数のサーバを 1 台の物理マシンに集約する手法がある．しかし，1 台の物理マシン上に複数の仮想マシン(VM)を稼働させた場合，物理マシン上の CPU やメモリは共有して使用されることから，各 VM で同一の OS やアプリケーションが起動していると，物理メモリ上で重複が発生してメモリを圧迫する要因となる．この重複をなくす Linux カーネルのシステムとして KSM(Kernel Same-page Merging)がある．本研究では，KVM 仮想化環境で KSM を適用し，KVM のメモリバルーニング機能を用いたメモリ最適化手法を提案する．具体的には，KSM によって重複が解消されたメモリ量をメモリバルーニング機能によって VM メモリに動的再配置する．本稿では，NoSQL の 1 つである Cassandra に着目し，仮想化環境上での Cassandra の性能評価を行い，提案手法による有効性を示す．

キーワード：仮想化，KVM，KSM,，KVS，ballooning，Cassandra

1. はじめに

クラウドコンピューティングの普及により，大量の計算機資源を容易に利用できるようになり，データセンタ等において多数のサーバが稼働するようになった．これに伴い，サーバの消費電力の増加，設置スペースの肥大化が問題となっている．この解決策の一つとして仮想化技術を用いて複数のサーバを 1 台の物理サーバに集約する手法がある．仮想化環境では，1 台の物理マシン上に複数の仮想マシン(VM)が稼働したとき，同じ OS とアプリケーションのセットが複数起動していると，物理メモリ上で同一内容のメモリページが発生する．Linux にはこのメモリページを 1 つのページに集約し残りのページを破棄する KSM(Kernel Same-page Merging)[1]という機能があり，本機能を用いることにより実質的に使用可能であるメモリ量を増やすことができ，これらのメモリをホスト OS やゲスト OS のページキャッシュに使用することによりゲスト OS 内のアプリケーションの I/O 性能を増加させることが可能であると期待できる．

また，クラウド環境ではユーザ数やサービスへの負荷の増減に応じてサーバ数の増減が容易であることから，スケーラビリティ(規模拡張性)の高い DBMS(Database Management System)が注目されている．その 1 つに KVS(Key-Value Store)がある．KVS は NoSQL の一種で，データ構造の簡素化とデータの一貫性保証を下げることでスケーラビリティの向上を図っており，サーバ増設によってリニアなスケールアウトと耐障害性向上を実現することができる．そのため，KVS はサーバ増設が容易な仮想化環境を用いたクラウド環境上で実行されることが多く，実際に Netflix 社では AWS(Amazon Web Services)上に著名な KVS の一つである Cassandra を用いたサービスを展開している[2]．これらのことから，仮想化環境における KVS の性能が重要であると考えられる．また，1 つのサーバ上では複数の VM が稼働し，それぞれが別々のデータベースであるマルチテナント環境として動作することも想定される．本研究では， 仮想化環境の一つである KVM，メモリ共有機能の KSM と KVS の一つである Cassandra に着目し，複数 VM 環境における I/O 性能の向上手法について考察する．

2. KVM と KSM

KVM(Kernel-based Virtual Machine)は代表的な仮想化システムの一つであり，本研究では KVM を用いて調査を行う．KVM は Linux2.6.20 以降のカーネル内に標準実装されており，OS をハイパバイザとして稼働する．

KVM の仮想 HDD の構築方法にはイメージファイルを使用するモードとパーティションを用いるモードがある．本研究では，仮想 HDD はイメージファイルモードを使用し，同モードでは図 1 の様に，ゲスト OS 上のアプリケーションはゲスト OS ファイルシステム，ゲスト OS のブロック層，仮想マシン，ホスト OS ファイルシステム，ホスト OS のブロック層を介して HDD へのアクセスが行われる．これらのうち，両 OS のブロック層にはキャッシュ(ページキャッシュ)機能が搭載されており，これら OS が保持するメモリ量が多いほど I/O 性能が高くなる．ゲスト OS のページキャッシュにおけるヒットと，ホスト OS のページキャッシュにおけるヒットを比較すると，物理 HDD へのアクセスを回避できるとの意味では同等であるが，上位であるゲスト OS のページキャッシュにてヒットした方が高い性能を得られる．上位であるゲスト OS のページキャッシュにてヒットすると，それ以下の処理が行われない．これに対して上位であるゲスト OS ページキャッシュでミスをして下位であるホスト OS ページキャッシュでヒットした場

†1 工学院大学 大学院 工学研究科 電気電子工学専攻
Electrical Engineering and Electronics, Kogakuin University Graduate School

合は，ゲスト OS ページキャッシュ以下の仮想マシン，ホスト OS ファイルシステム，ホスト OS ページキャッシュなどが動作し，生じる処理負荷の量が大きくなる[3].

また，KVM にはメモリバルーニング機能が搭載されており，ホスト OS がゲスト OS の割り当てるメモリ量を動的に変更することができる．メモリバルーニングでは VM に初めに割り当てたメモリ量よりも大きいメモリを割り当てることはできないため，実際にはホスト OS で使えるメモリ量を増やすために利用されることが多い.

KSM は各 VM が利用しているページの内，同一内容のメモリページ群を 1 つの物理ページにまとめる機能であり，VM のメモリ量を減らすことなく利用可能である物理メモリ量を多くすることができる. KSM はカーネル内でデーモンとして動作し，主に匿名ページを周期的に観測し，重複するページ群を特定して CoW 方式でマージする．この重複ページを特定する操作はスキャンと呼ばれ，スキャンは設定ファイルを動的に読み込んで行われる.

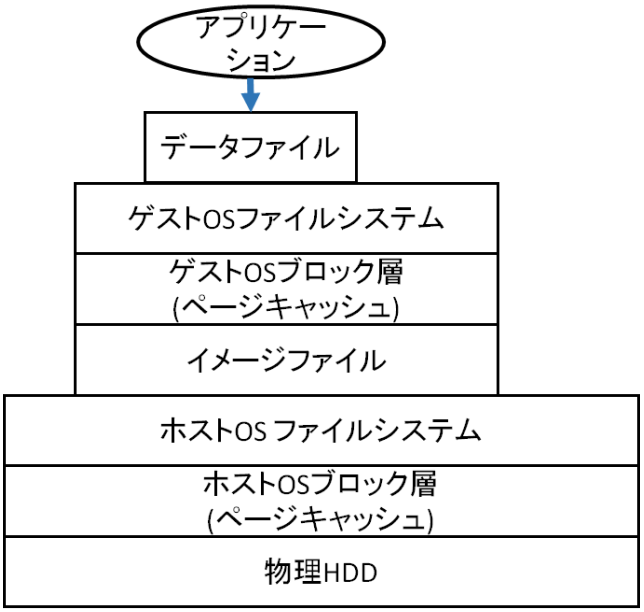

図 1　KVM 環境の構成

スキャンの設定は動的に変更可能であり，これを利用したデーモンとして ksmtuned がある．ksmtuned は，スキャンを積極的に行うか，消極的に行うかなどの状況に応じて調整することができ，これにより KSM の負荷を調整することができる. また, KSM によるマージ状況は監視可能であり，/sys/kernel/mm/ksm/以下の 4 つのファイルに結果が書き込まれる．すなわち，page_shared は共有されている物理ページ数が書かれており，page_sharing はマージされ節約された仮想ページ数が書かれており，page_unshared はスキャン対象である物理ページ数が書かれており，page_volatile は頻繁に変更される物理ページ数が書かれている.

3. KVS と Cassandra

KVS は，Key と Value の組を書き込み，Key を指定することで Value を得ることができるデータ管理手法の 1 つである．RDBMS(Relational DataBase Management System)より構造が単純になっていることから，高いスケーラビリティを得ることができる．KVS の代表的な実装に Cassandra [4]がある.

Cassandra は Facebook 社が開発した KVS であり，現在は Apache Software Foundation のトップレベルプロジェクトである. Amazon 社の Dynamo[5]の分散ハッシュテーブルと Google 社の BigTable[6]のデータモデルを併せ持ち，結果整合性の一貫性を持つ. 耐障害性の高さ，ノードの非集中性，高可用性，動的に伸縮可能なスケーラビリティなどの特徴を持つ.

Cassandra では複数のノードで分散してデータを保持し，そのノードの集合はクラスタと呼ばれる．図 2 の様に，Cassandra クラスタを構成する各ノードはトークンと呼ばれるハッシュ値が割り当てられ，リング上のハッシュ空間にトークンを元に配置される．リング上の各ノードは，ハッシュ値が自身のトークン値以下で，かつ直前のノードのトークン値より大きい範囲を担当し，その範囲の Key と Value の組を保持する．このとき，トークン値の担当範囲をノード別に指定することで，各ノードに公平にデータ量を分散することができる．KVS の読み込み処理または書き込み処理を行う際，Key をハッシュ関数にかけ，そのハッシュ値を担当するノードが読み込み処理や書き込み処理を実行するノードとなる．ただし，レプリカが存在する場合，それを持つノードも実行するノードの対象となる.

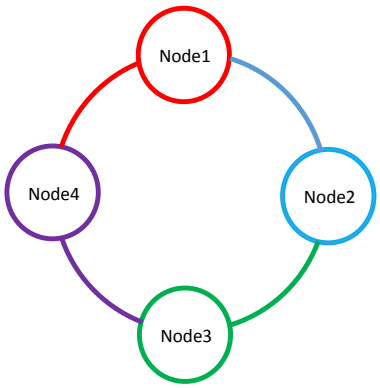

図 2　Cassandra のノード配置

Cassandra ではデータベースの複製（レプリカ）の数を指定することが可能である．レプリカ数は初期設定では 1 であるが，2 以上の値にすることによって耐障害性を向上させることができる．レプリカは上記の担当ノードの後続ノードに配置される.

4. 基本性能調査

本章にて，仮想化環境における基本 I/O 性能調査として，I/O 性能と，VM メモリ量と，KSM およびバルーニングの動作の関係の調査を行う．基礎 I/O 性能調査としてはファイルシステムベンチマーク FFSB(Flexible File System Benchmark)[7]を用い，応用 I/O 性能調査としては KVS の一つである Cassandra と YCSB(Yahoo! Cloud Serving Benchmark)[8]を用いる．

4.1 基本 I/O 性能の測定環境

基本性能調査として VM 上で FFSB を実行したときに得られる性能を評価した．

2 台の物理マシンを使用し，1 台の物理マシン上で 3 台の VM を起動し，合計 6 台の VM 上で FFSB を実行した．使用したマシンの仕様は表 1 の通り，FFSB の設定内容は表 2 の通りである．また，VM メモリを 2[GB]，2.25[GB]，2.5[GB]と増加させ，オーバーコミット状態における FFSB 実行時の性能評価を行った．

表 1　使用したマシンの仕様

	物理マシン	VM	Client
OS	CentOS6.5		
Kernel	Linux 2.6.32.27		
Memory	8[GB]	適宜変更	8[GB]
CPU	AMD Turion II Neo N54L Dual-Core Processor	QEMU Virtual CPU version (cpu64-rhel6)	Intel Core i7-2600

表 2 FFSB の設定

Data Size	3[GB]，16[GB]
File Size	1[MB]
Num File	16384
Operation	Read : 100[%]

4.2 仮想化環境における基本 I/O 性能

評価結果を図 3，4 に示す．

図 3 より，データサイズが大きい環境では，各 VM メモリを増加させるとスループットが減少することが確認された．また，キャッシュヒット率を見ると各 VM メモリサイズが増加するほどキャッシュヒット率が増加することが確認された．VM メモリサイズを増加させ，キャッシュヒット率を向上させてもスループットが減少している理由は，ホスト OS のメモリサイズが 8[GB]に対して，ゲスト OS が占有する合計メモリサイズが 6[GB]，6.75[GB]，7.5[GB]と増加するため，ホスト OS 上でスワップが発生してディスクアクセスが起きることでスループットが減少するためだと思われる．

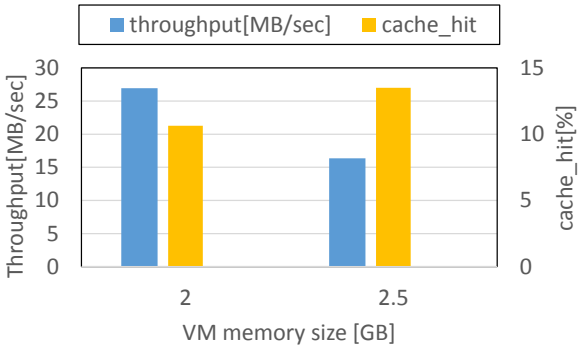

図 3　仮想化環境における DB サイズ 16[GB]での FFSB のスループットとキャッシュヒット率

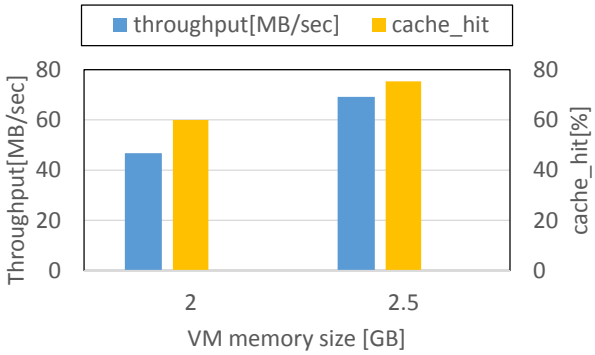

図 4　仮想化環境における DB サイズ 3[GB]での FFSB のスループットとキャッシュヒット率

4.3 KSM 適用時における FFSB での性能調査

次に KSM を適用時における FFSB のスループットと VM 上のキャッシュヒット率の関係について述べる．測定環境は前節と同様である．結果を図 5 に示す．

図 5 において KSM 非適用時と適用時を比較すると，性能は同等であるか KSM 適用時の方が大幅に性能が高いことがわかる．このことから，KSM によって VM の重複ページ分のメモリ量がホスト OS に還元され，ホスト OS のページキャッシュが増えたことでスループットが上昇したと考えられる．また，KSM 非適用時と適用時ともキャッシュヒット率は各 VM メモリサイズが増加するごと増加することが確認された．

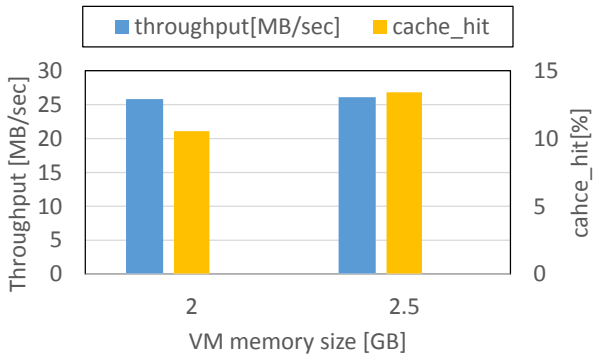

図 5　KSM 適用時おける FFSB のスループットとキャッシュヒット率

4.4 応用 I/O 性能の測定環境

VM 上で Cassandra を稼働させ，クライアント PC 上から YCSB で得られる性能を評価した．

2 台の物理マシンを使用し，1 台の物理マシン上で 3 台の VM を起動し，合計 6 台の VM 上で Cassandra クラスタを構築した． Cassandra のデータベースサイズは 17[GB]，レプリカ数 1 とし，各ノードが担当するデータ範囲は均等とした(各ノード約 2.9[GB]のデータを保持)．以下この環境をシングルテナント環境と呼ぶ．また，データベースの個数を 1→3 に増やし，仮想マシン 2 台ずつでデータベースサイズ 17[GB]の Cassandra クラスタを構築し，各ノードは約データサイズ 8.5[GB]のデータを保持する環境を構築した．この環境を以下マルチテナント環境と呼ぶ．

YCSB は読込負荷 100%，スレッド数 18，操作数 50000 回，一貫性レベル ONE とし，10 回測定した平均を結果とした．測定環境を図 6，7 に示し，使用した物理マシンと仮想マシンの使用は表 1 の通りである．

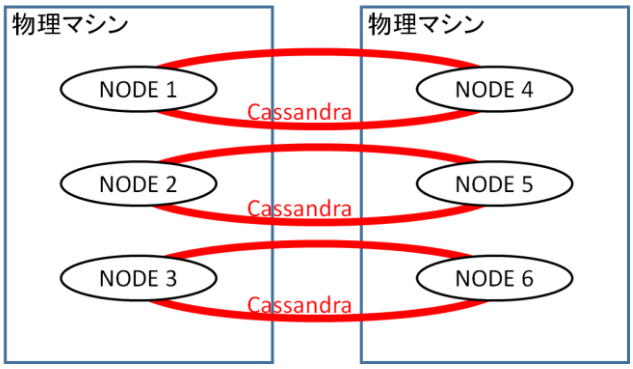

図 6　シングルテナント環境

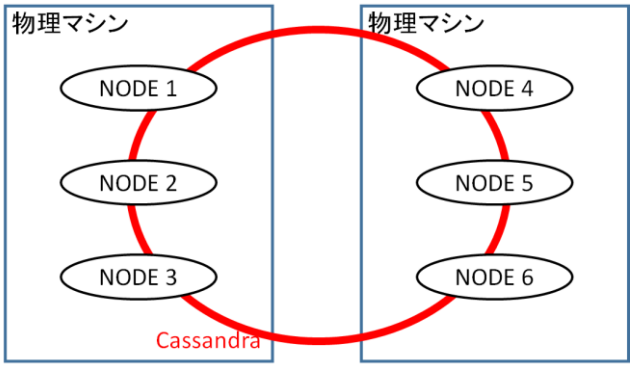

図 7　マルチテナント環境

4.5 仮想化環境における応用 I/O 性能

シングルテナント環境とマルチテナント環境における Cassandra 性能(非 KSM 適用時) を図 8 の“single_tenant”と“multi_tenant”に，ベンチマーク時の物理マシンの平均 CPU・I/O 使用率を図 9，10 に示す．

図 8 より，各 VM が担当するデータベースサイズはシングルテナント環境よりもマルチテナント環境の方が大きいため，マルチテナント環境のスループットが低く，負荷が高いことがわかる．

この時，シングルテナント環境では KSM による性能向上率は 15%に対してマルチテナント環境では 22%となったことから，データベースが複数台ある環境では KSM による性能向上率が高くなることが確認された．

また，図 9，10 よりベンチマーク中では disk I/O 使用率が高くなることが確認された．このことから，本ベンチマーク中のボトルネックは物理マシンの disk I/O であると考えられる．

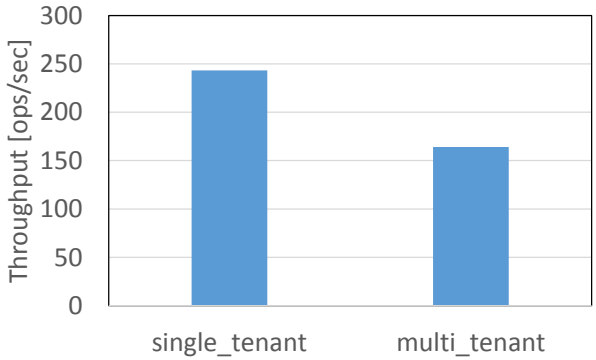

図 8　シングルテナントとマルチテナントにおける Cassandra 性能

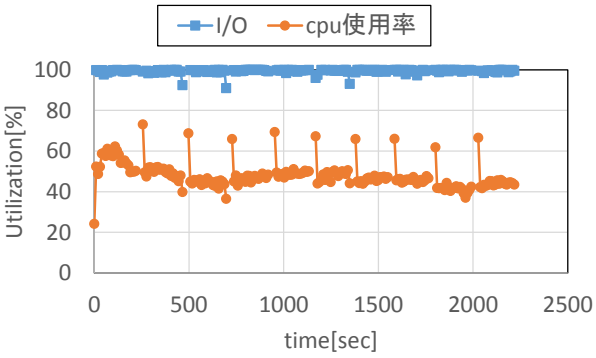

図 9　シングルテナント環境におけるベンチマーク時の平均 CPU，I/O 使用率

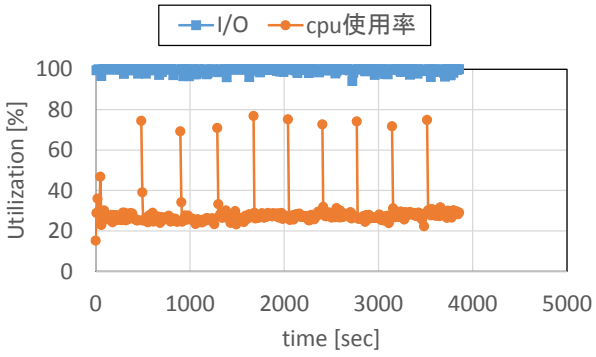

図 10　マルチテナント環境におけるベンチマーク時の平均 CPU，I/O 使用率

4.6 KSM 適用時おける応用 I/O 性能

前節の環境に対して KSM を適用した時におけるシングルテナント環境とマルチテナント環境で性能評価を行った．

測定結果を図 11 に，ベンチマーク時の物理マシンの I/O 使用率の比較を図 12 に示す．

図 11 より，KSM を適用することによりスループットが向上することが確認された．これは，すべての仮想マシン上で同じアプリケーション，同じカーネルが起動しているため，KSM によるページマージングが効果的に働き，物理ページの重複削除により物理マシンのメモリキャッシュが増え，disk I/O 処理が速くなったことが原因だと考えられる．図 12 より，シングルテナント環境では I/O 使用率に変化はなかったものの，マルチテナント環境では 5%ほど I/O 負荷が低減されたことを確認した．

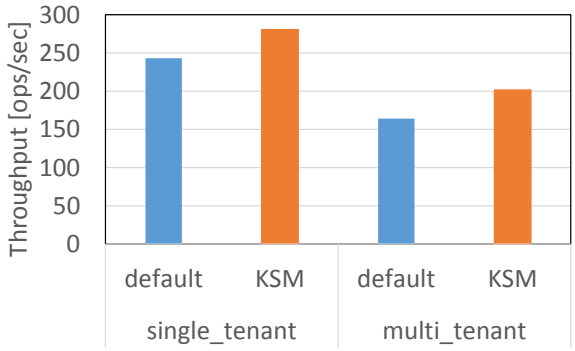

図 11　KSM 適用時におけるシングルテナントとマルチテナントでの Cassandra 性能

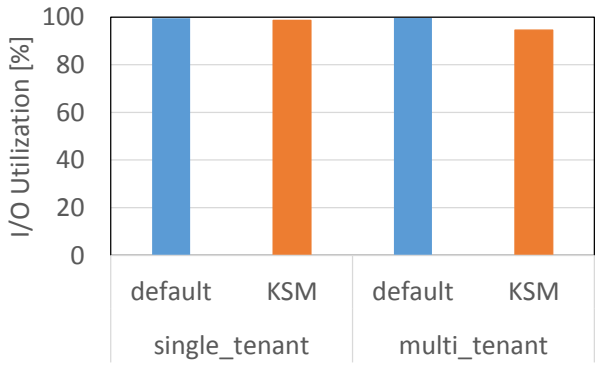

図 12　KSM 適用時おけるベンチマーク中の平均 I/O 使用率

4.7　VM メモリ増加と KVS 性能

4.5 節，4.6 節の環境に対して VM メモリサイズを拡大した状況におけるシングルテナント環境の性能を図 13 に示す．

図 13 より，KSM 非適用時はメモリサイズが大きいほどスループットが向上することが確認された．一方，KSM 適用時ではメモリサイズを増やしても性能に変化は見られなかった．このことから，VM メモリ量を単純に静的に増加させても必ずしも I/O 性能が増加しないことがわかる．

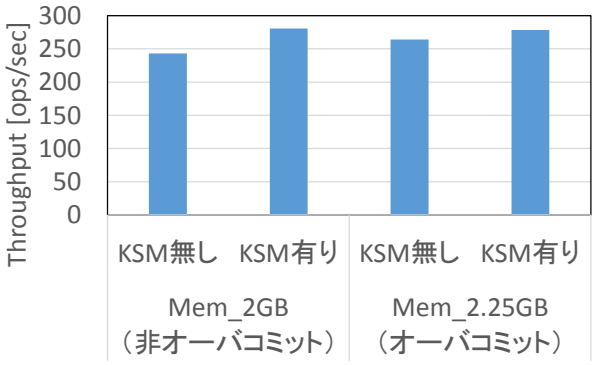

図 13　メモリ量と KSM 有無と KVS 性能

5.　提案手法

5.1　KSM 共有ページ数による動的 VM メモリ制御

前章の調査より，VM メモリ量増加による性能向上が実現可能であることが確認されたが，単調に静的に増加させても性能は必ずしも向上しないことがわかった．

本章では，KSM とメモリバルーニング機能を用いて VM のメモリ量を適切に増加させ，動的メモリ最適化による I/O 性能向上手法を提案する．具体的には，KSM によって VM の重複ページをスキャンする．そして，共有ページ数である“page_shared”を常時観測し，KSM で得られたメモリ量をバルーニング機能を用いて全 VM に均等追加配分する．これにより，KSM によって節約されたメモリは VM に還元されるため，VM の I/O 処理速度向上に繋がると考える．

追加配分量は page_shared の常時観察により動的に調整し，page_shared が減少した場合は追加配分量も減少し，VM 割り当てメモリ量も減少することとなる．

5.2　実験環境への適用

メモリバルーニングでは初めに VM に割り当てたメモリ量を上回るメモリを与えることができないため，十分なメモリを与えた状態で VM を起動させる．本稿の実験環境では，提案手法適用時は VM 起動時にメモリを 2.25GB に設定し，バルーニングを行う前にメモリを 2GB に再設定してベンチマークを行う．

6.　性能評価

6.1　FFSB

FFSB を用いて提案手法の性能評価を行った．測定環境は，4 章と同様である．評価結果を図 14 に示す．図より，FFSB においては性能はほぼ同等となることがわかった．

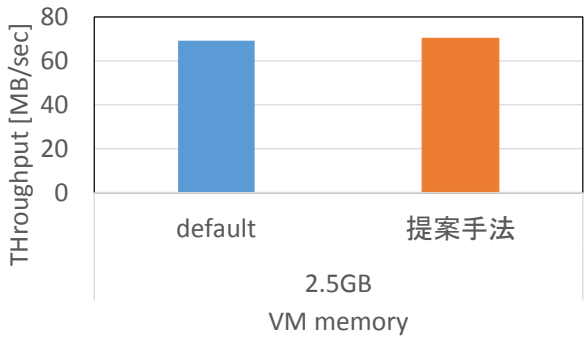

図 14　提案手法性能(FFSB)

6.2 KVS 性能

Cassandra を用いて，提案手法の性能評価を行った．測定環境は前章と同じである．測定結果を図 15 に示す．図 15 より．提案手法により KVS 性能が大幅に向上することが確認され，提案手法の有効性が確認された．

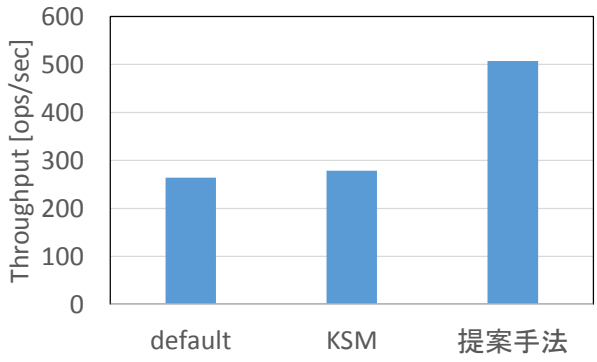

図 15　提案手法性能(Cassandra)

7. まとめ

本稿では，仮想化環境において KSM を活用し利用可能メモリ量を増加させ I/O 性能を向上させる手法に着目し，時間的に変化する KSM 共有量に動的に対応して VM メモリ量を増加させる手法を提案した．また，提案手法の性能を基礎 I/O ベンチマークおよび応用 I/O ベンチマークにより評価し，同等の性能あるいは大幅な性能向上を確認した．今後はさらなる性能向上を目指していく予定である．

謝辞　本研究は JSPS 科研費 25280022, 26730040, 15H02696 の助成を受けたものである．

参考文献

[1] Andrea Arcangeli, Izik Eidus, Chris Wright,"Increasing memory density by using KSM",Proceedings of the Linux Symposium,2009

[2] "Revisiting 1 Million Writes per second ". http://techblog.netflix.com/2014/07/revisiting-1-million-writes-per-second.html

[3] 杉本洋輝，山口実靖，“二重キャッシュ環境における負の参照の時間的局所性を考慮したキャッシュ管理手法”，情報処理学会論文誌コンシューマ・デバイス＆システム（CDS） Vol 5 No.4 pp 42 - 51，(2015)

[4] Avinash Lakshman and Prashant Malik, “Cassandra- A Decentralized Structured Storage System”, LADIS 09, 2009

[5] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels, “Dynamo: Amazon’s Highly Available Key-value Store”, SOSP ’07, 2007

[6] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes and Robert E. Gruber,“Bigtable: A Distributed Storage System for Structured Data”, IOSDI ’06 pages 205--218, 2006

[7] Flexible File System Benchmark: http://sourceforge.net/projects/ffsb/

[8] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, Russell Sears “Benchmarking Cloud Serving Systems with YCSB”