Vol.2012-DBS-156 No.4
2012/12/12

仮想化環境下における下位キャッシュの参照の時間的局所性の解析

竹内 洸祐[†1]　日名川幸矢[†2]　山口実靖[†3]

クラウドコンピューティングの普及により仮想化環境の重要性が高まっている．仮想化環境では，ゲスト OS とホスト OS の二種類の OS が動作し，それぞれがキャッシュ機能を提供している．このような二重のキャッシュ環境では，下位キャッシュ(ホスト OS キャッシュ)に対する参照の時間的局所性が通常の局所性と異なる可能性が考えられる．本稿では，仮想化環境における下位キャッシュへの参照の局所性の解析結果を示し，通常の参照の時間的局所性と異なることを示す．そして，シミュレーションにより既存のキャッシュ置換アルゴリズムの性能を示し，LRU などの既存のキャッシュ置換アルゴリズムが効果的に機能しないことを示す．最後に，実際の仮想化環境におけるゲスト OS キャッシュとホスト OS キャッシュのヒット率を示し，下位キャッシュメモリが効果的に機能していないことを示す．

An Analysis of Locality of Reference in Virtualized Environment

KOUSUKE TAKEUCHI[†1]　KOUYA HINAGAWA[†2]
SANEYASU YAMAGUCHI[†3]

As cloud computing becomes common, virtualized environment is of increasing its importance. In virtualized environment, two operating systems, which are guest operating system and host operating system, run concurrently. Both of them have caching function, thus two caches works concurrently. In most cases, LRU cache replacement algorithm is applied in guest operating system cache. Applying LRU in the upper cache, which is the guest operating system cache, leads negative temporal locality of reference in the lower cache, which is the host operating system cache. Thus LRU may not work effectively in the lower cache. In this paper, we explore locality of reference in a host operating system cache. In addition, we evaluate performance of existing cache replacement algorithms in host operating system cache with our simulation and that of an implementation of practical operating system.

1. はじめに

近年,クラウドコンピューティングの普及に伴い仮想化環境の重要性が高まっている．仮想化環境下において，物理 HDD へのアクセスは，ゲスト OS キャッシュ(上位キャッシュ)とホスト OS キャッシュ(下位キャッシュ)の二重のキャッシュを介して行わる．このような環境の場合，両キャッシュには同一のデータを重複して保存する可能性がある．たとえば，ゲスト OS キャッシュと，ホスト OS キャッシュの両方でキャッシュミスとなった場合は，そのデータは最新のデータとして，ゲスト OS キャッシュとホスト OS キャッシュの両方に重複して格納される．

両キャッシュの保持データの多くが重複している場合，ゲスト0Sキャッシュでキャッシュミスしたアクセス要求は高確率でホスト0Sのキャッシュでもキャッシュミスし，ホスト0Sのキャッシュは効果的に機能しない．[4][9]同様にホスト0Sキャッシュに格納されているデータへのアクセスは,ゲスト0Sキャッシュにおいてキャッシュヒットとなり，ホスト0Sへ転送されず，ホスト0Sのキャッシュが効果的に機能しない．これをホスト0Sキャッシュの視点で見ると，一度アクセスされたデータは近い将来に再度アクセスされる可能性が低いということになる．すなわち，通常とは逆向きの負の参照の時間的局所性が存在することとなる．これに関連する既存の研究として，長廻らのシュミュレーションによる解析[1] と，Zhou らによるDBMS やDISK I/O 用いた単一サーバ環境における解析[2] がある．しかし，これらの研究は仮想化環境下を想定しておらず，実際に仮想計算機上で実アプリケーションを動作させての解析が必要であると考えられる．

本論文では，ゲスト OS 上でアプリケーションを動作させ，ホスト OS キャッシュへのアクセスパターンの解析を行い,負の参照の時間的局所性の存在の調査を行う.また，シミュレーションによるホスト OS キャッシュにおける各キャッシュ置換アルゴリズムのヒット率の調査を行う．そして、実際のホスト OS キャッシュ上でヒット率の調査を行なう．

2. 二重キャッシュ構造

図 1 は仮想化環境における両キャッシュの動作とアクセス要求の転送の様子を示したものである．アプリケーションから HDD 上のデータへのアクセス要求が発行されると，まず該当データがゲスト OS キャッシュ内に存在するか否かが確認される．存在する場合(図 1 の(1))は，ゲスト OS キャッシュヒットとなり,ゲスト OS キャッシュによりアプリケーションにデータが返される．該当データが存在しない場合，データアクセス要求はホスト OS に転送され，ホスト OS キャッシュに該当データが存在するか否かが確認される．該当データが存在する場合(図 1 の(2))は，ホスト

†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.
†2 工学院大学工学部情報通信工学科
Department of information and Communications Engineering, Kogakuin University

OS キャッシュでデータが返され，HDD アクセスは行われない．ホスト OS キャッシュにも該当データがない場合(図 1 の(3)) にはホスト OS キャッシュもキャッシュミスとなり，実計算機の物理 HDD にアクセスを行い，該当データをアプリケーションに返す．物理 HDD へのアクセスはキャッシュ（半導体メモリ)へのアクセスに比べアクセス時間が長い．また、ゲスト OS からホスト OS へのデータの転送にも時間がかかる．よって，アクセス時間は短い順に，(1)ゲスト OS キャッシュヒット，(2) ホスト OS キャッシュヒット，(3)物理 HDD デバイスアクセスとなる．

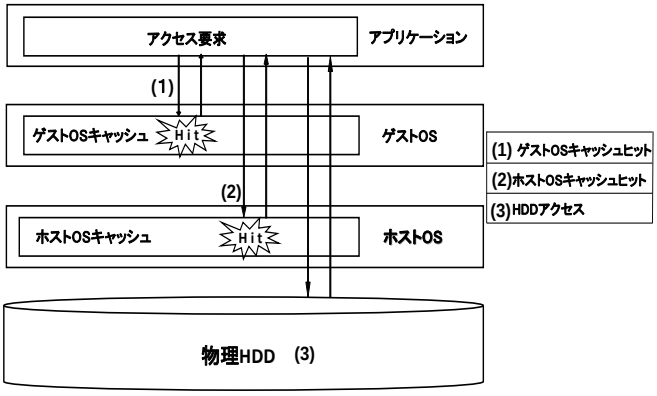

図 1　二重キャッシュ環境

3. 負の参照の時間的局所性

3.1 参照の局所性

多くのキャッシュシステムではキャッシュの記憶容量は下層の低速記憶装置の記憶容量よりも小さいが，多くの場合は十分な性能を発揮できることが経験的に知られている．これはアプリケーションが発行するデータアクセス要求には多くの場合偏りがあるためであり，この参照の偏りは“参照の局所性(Locality of Reference)”と呼ばれている．現在のメモリ管理技法の多くはこの参照の局所性の概念に基づいている．参照の局所性には，“時間的局所性（Temporal locality）”と，“空間的局所性（Spatial locality）”がある．時間的局所性は，ある時点で参照されたデータが近い将来に再び参照される可能性が高いという性質である．空間的局所性は，あるデータが参照されたとき，その近傍のデータが近い将来に参照される可能性が高いという性質である[3]．

3.2 LRU

前節の参照の局所性を期待したキャッシュ置換手法に LRU (Least Recently Used)[3] がある．LRU は，最近アクセスされたデータ(最後のアクセスからの時間が最も短いデータ) が再アクセスされる確率が最も高く，最後のアクセスからの時間が最も長いデータが再アクセス確率が最も低いと仮定し，最後のアクセスからの時間が最長であるものを破棄するキャッシュ置換手法である[3]．多くのコンピュータシステムのアプリケーションにおいて参照の時間的局所性が存在することが確認されており，現在のOS やコンピュータシステムのほとんどにおいてキャッシュの置換アルゴリズムとしてLRU が採用されている．

3.3 負の参照の時間的局所性

一般的に，上位のゲストOSのキャッシュ置換アルゴリズムにはLRU が使われている．この場合ゲストOSのキャッシュには，最近アクセスされたデータが格納されている．よって，アプリケーションからの「最近アクセスされたデータ」へのアクセス要求は，ゲストOSキャッシュにより処理されホストOSキャッシュには転送されない．このため，ホストOSキャッシュには「最近アクセスされたデータ」へのアクセス要求が到達せず，「最近アクセスされたデータが近い将来に再度アクセスされる可能性は低い」という，通常とは逆向きの負の参照の時間的局所性が存在することになる．よって，通常の参照の時間的局所性を期待しているLRU をゲストOSキャッシュにおいて用いても，効果的に機能しないと予想される．

本稿では，「最近アクセスされたデータが近い将来に再度アクセスされる可能性は低い」という通常とは逆向きの参照の局所性を“負の参照の時間的局所性”と呼び，特にこれがLRU のヒット率を低下させる性質に着目して議論を行う．

図2に，シミュレーションにより得たゲストOSキャッシュとホストOSキャッシュにおける再アクセス間隔と再アクセス回数の関係を示す．シミュレーションにおけるストレージサイズ(総データ量) は10,000[Block]，ゲストOSキャッシュサイズ（上位キャッシュ）は1,000[Block]，ホストOSキャッシュ（下位キャッシュ）サイズは1,000[Block] である．すなわち，全ブロックのうちの10%のブロックが各キャッシュに格納されている．図2 のシミュレーションでは，一様分布乱数を発生させストレージの全ブロック(10,000[Block]) に対して均一の確率でアクセス要求を発行した．ファイルシステムなどの動作は考慮されておらず，アクセス要求は全て読み込み要求を想定している．図はストレージの各ブロックへの2 回目以降のアクセスのみに着目し，各アクセスの再アクセス間隔(前回のアクセスと今回のアクセスの間隔．単位はブロック) と，その再アクセス間隔の発生回数を表している．図2 のホストOSキャッシュでは，横軸の値が2,000にて，縦軸の値が8869となっているが，これは再アクセス間隔が2,000[block] 以上で2,100[block] 未満のアクセスが8869[回] 発生したことを表している．再アクセス間隔が2,000[block] とは，あるブロックに対してアクセスが行われた後，当該ブロック以外へのアクセスが2,000[block] 分発生し，その後に当該ブロックへのアクセスが発生したということである．

図2より，再アクセス間隔が0[block]～約1000[block] のときはホストOSにアクセス要求が来ず，ホストOSキャッシュにおける再アクセス回数が0[回] になっていることがわか

る．これより，シミュレーション環境においては明確に負の参照の時間的局所性が存在することがわかる．ただし，本シミュレーションは実環境で用いられるファイルシステムなどの振る舞いが考慮されておらず，実際のOS を現実の仮想計算機上で稼働させ，その上で実アプリケーションを動作させた実環境における考察が必要であると考えられる．

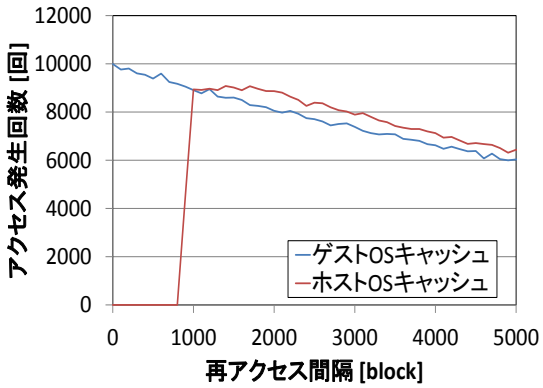

図 2　再アクセス回数の関係

4. 仮想化環境の下位キャッシュにおける参照の時間的局所の解析

4.1 解析方法

本章において，ホストOSのキャッシュに対する参照の局所性の調査を行う．図3の様な実験環境を構築し，図3内の“Monitoring”の箇所を流れるアクセス要求を調査した．図3 内の“Monitoring”の箇所を流れるアクセス要求はゲストOS のキャッシュを経由した(キャッシュミスした）後のものであり，ホストOSキャッシュへの参照の局所性を調査することができる．

図3 の実験環境にてベンチマークソフトであるFFSB (Flexible File System Benchmark）を実行した．実験に使用した計算機の仕様は表1，表2の通り，FFSBの設定は表3の通りである.なおモニタリングはゲストOS の仮想ブロックデバイス層(xvd層)にて行った．

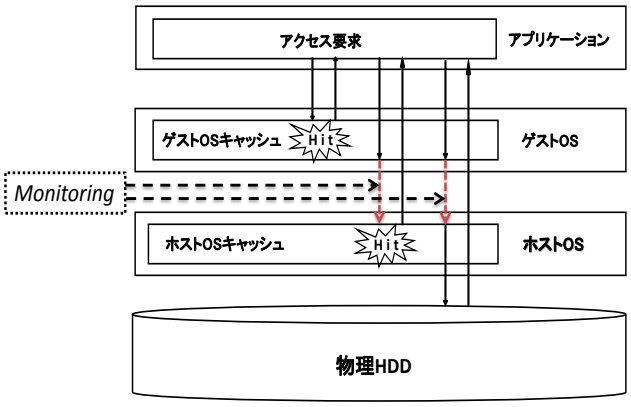

図 3　実験環境

表 1　実計算機の仕様

OS	CentOS5.5(64bit)
Kernel	Linux 2.6.18.8
CPU	AMD Athlon™
HDD(sda)	140GB
HDD(sdb)	80GB
メモリ	4GB
Cache	3.46GB
仮想化システム	Xen 3.4.3
使用ファイルシステム	ext2

表 2　仮想計算機の仕様

OS	CentOS5.5(64bit)
Kernel	Linux 2.6.18.8
CPU	AMD Athlon™
仮想HDD	100GB
メモリ	1GB,2GB,3GB
Cache	0.9GB,1.8GB.2.8GB
使用ファイルシステム	ext2.ext3

表 3　FFSB の設定

Data Size	16GB	16GB	16GB	16GB
File Size	16MB	16MB	16MB	16MB
Num File	1024	1024	1024	1024
OP Size	16KB	64KB	256KB	1MB

4.2 測定結果

アクセス間隔とアクセス発生確率(確率密度関数)の関係を図 4～図 11 に示す．OPsize は，アプリケーション(FFSB) がファイルシステムに発行するシステムコール `read()`のサイズである．ext2,および ext3 は，ゲスト OS で使用したファイルシステムである．図内のグラフ系列の“Guest1GB:Host3GB”とは，ゲスト OS(仮想計算機)に割り当てられたメモリサイズが 3GB であり，ホスト OS に割り当てられたメモリサイズが 1GB であることを示している．

参照の時間的局所性に着目すると，図 4 を除くすべての実験結果(図 5 から図 11)において負の参照の時間的局所性が存在していることがわかる．図 5 の例では，ゲスト OS のメモリが 1GB の場合は再アクセス間隔 0.7GB 以下の再アクセスがほとんど発生していない．同様に，図 5 のゲスト OS のメモリサイズが 2GB の場合は再アクセス間隔が 1.7GB 以下の再アクセスが，ゲスト OS のメモリサイズが 3GB の場合は再アクセス間隔が 2.6GB 以下の再アクセスがほとんど発生していない. 図 6 から図 11 の測定結果でもほぼ同様の結果が得られており，再アクセス間隔が 0.7GB, 1.7GB，2.6GB 以下の再アクセスがほとんど発生していない．

以上より，ファイルシステムに ext3 を用いた場合，ファイルシステムに ext2 を用い小さくない(64KB 以上)オペレーションサイズを使用した場合に負の参照の局所性が発生することが確認された．また，上位キャッシュとなるゲスト OS のメモリサイズを増やすとより大きな(再アクセスが発生しない間隔がより長い)負の参照の局所性が発生することが分かった．

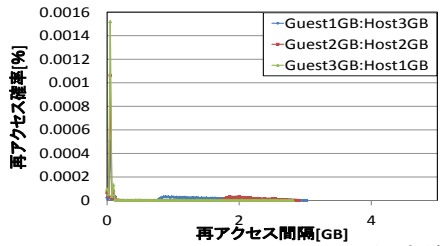

図 4　FFSB(OPsize16KB,ext3)測定結果

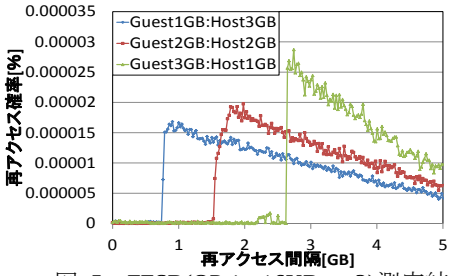

図 5　FFSB(OPsize16KB,ext3)測定結果

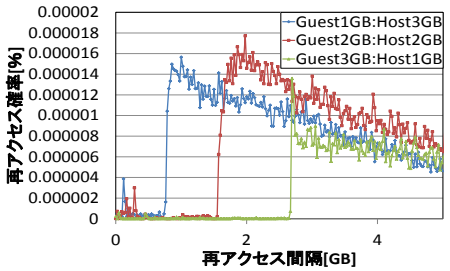

図 6　FFSB(OPsize64KB,ext2)測定結果

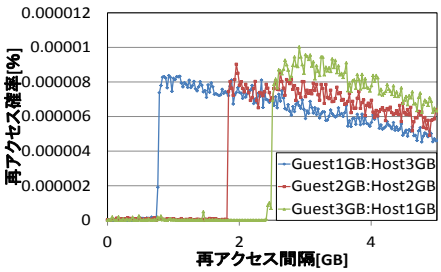

図 7　FFSB(OPsize64KB,ext3)測定結果

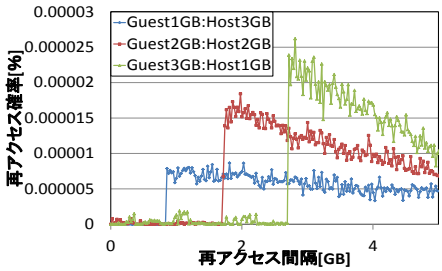

図 8　FFSB(OPsize256KB,ext2)測定結果

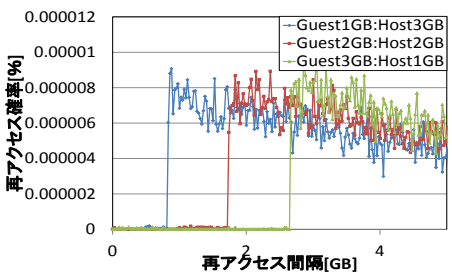

図 9　FFSB(OPsize256KB,ext3)測定結果

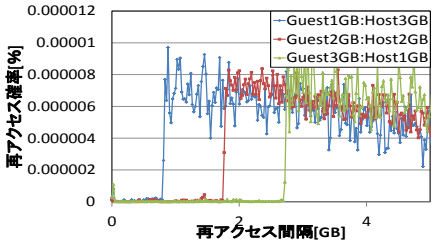

図 10　FFSB(OPsize1MB,ext2)測定結果

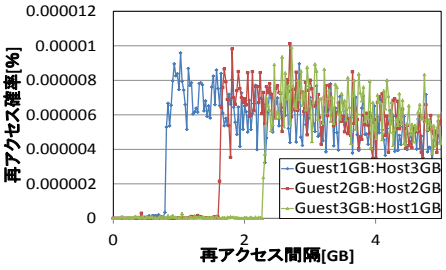

図 11　FFSB(OPsize1MB,ext3)測定結果

5. ホスト OS キャッシュにおけるキャッシュ置換アルゴリズムのヒット率評価

本章にて，ホスト OS キャッシュにおける既存キャッシュ置換アルゴリズムの性能評価を行う．アクセスログとして前章の実験で得られた実アクセスログを用い，キャッシュヒット率をシミュレーションにより求めた．置換手法としては LRU，RANDOM，FIX，MQ を用いた．LRU は 3 章にて説明した手法である．FIX はキャッシュに格納されたデータを置換せず保持し続ける手法である．RANDOM は，キャッシュから無作為に選択したデータを置換対象とする置換手法である．MQ は 6 章の関連研究にて説明する．

5.1 シミュレーション環境

シミュレーション環境を表 4～6 に示す．表 4 は下位キャッシュのメモリサイズが 1[GB]，表 5 は下位キャッシュのメモリサイズが 2[GB]，表 6 は下位キャッシュのメモリサイズが 3[GB]の時のシミュレーション環境を示している．それぞれの環境において，ホスト OS のキャッシュサイズをゲスト OS キャッシュサイズの 0.5 倍から 2.0 倍に変更して測定を行なっている．アクセスログとしては前章の FFSB の測定で得られたログを使用している．よって，下位のホスト OS キャッシュの上位にはゲスト OS キャッシュが存在しており，ゲスト OS キャッシュの置換は Linux OS 搭載の LRU 実装により行われている．

評価対象はホスト OS キャッシュであり，これのヒット率の調査を行った．当該ログはゲスト OS の仮想ブロックデバイス xvd 内で発行されたブロックアクセスコマンドの履歴であり，表 4～6 内の[block]は 4KB のブロックである．たとえば，総データ量 4,194,304 [block] は，16[GB]を意味している．

表 4　シミュレーション環境(メモリサイズ 1GB)

総データ量	4,194,304 [block]
ホストOSキャッシュサイズ(0.5倍)	117,965 [block]
ホストOSキャッシュサイズ(1.0倍)	235,930 [block]
ホストOSキャッシュサイズ(1.5倍)	353,895 [block]
ホストOSキャッシュサイズ(2.0倍)	471,860 [block]

表 5　シミュレーション環境(メモリサイズ 2GB)

総データ量	4,194,304 [block]
ホストOSキャッシュサイズ(0.5倍)	235,930 [block]
ホストOSキャッシュサイズ(1.0倍)	471,860 [block]
ホストOSキャッシュサイズ(1.5倍)	707,790 [block]
ホストOSキャッシュサイズ(2.0倍)	943,720 [block]

表 6　シミュレーション環境(メモリサイズ 3GB)

総データ量	4,194,304 [block]
ホストOSキャッシュサイズ(0.5倍)	367,001 [block]
ホストOSキャッシュサイズ(1.0倍)	734,003 [block]
ホストOSキャッシュサイズ(1.5倍)	1,101,005 [block]
ホストOSキャッシュサイズ(2.0倍)	1,468,006 [block]

5.2 シミュレーション結果

下位キャッシュ(ホスト OS キャッシュ)のヒット率を図 12 に示す．横軸は，ゲスト OS キャッシュサイズとホスト OS キャッシュの比を表しており，1.0 倍は，ゲスト OS キャッシュとホスト OS キャッシュのサイズが等しく，0.5 倍,1.5 倍,2.0 倍は，ホスト OS キャッシュのサイズがゲスト OS キャッシュの 0.5 倍,1.5 倍,2.0 倍を示している．また図 12 の Guest1GB というのはゲスト OS が 1[GB]のメモリを，Guest2GB はゲスト OS が 2[GB] のメモリを，Guest3GB はゲスト OS が 3[GB] のメモリを使用しているときのアクセスログを使用したことを示している．

図 12 の全ての場合において LRU および MQ のヒット率が極めて低く，ホスト OS キャシュにおいてこれらの手法が効果的に機能しないことがわかる．RANDOM は LRU や MQ よりは性能が高いが FIX よりも低く，これも効果的な手法とはなっていない．

ゲスト OS キャッシュのサイズと，ホスト OS キャッシュのサイズの比率について注目すると，ホスト OS キャッシュのサイズが小さいほど LRU と MQ の性能が悪くなっていることがわかる．これは，ホスト OS キャッシュのサイズがゲスト OS キャッシュのサイズと比較して相対的に小さくなるほど，ホスト OS キャッシュのデータがゲスト OS キャッシュに包含される確率が増え，負の参照の時間的局所性が強まっているからだと言える．

次にファイルシステムの違いに着目すると，前章にて最も明確に負の参照の時間的局所性が現れた ext3 において LRU と MQ のヒット率が低くなっていることがわかる．ただし負の参照の時間的局所性が比較的に弱かった ext2 においても LRU と MQ の性能劣化が一番低くなっている．オペレーションサイズの違いに着目すると，オペレーションサイズに依らす LRU と MQ のキャッシュヒット率が低く，オペレーションサイズの影響は小さいことが分かる．MQ は，下位キャッシュ用に提案されたキャッシュ置換アルゴリズムであるが，今回の評価では LRU とほぼ同等の性能となり，非常に低い性能となった．MQ は参照回数によって，データをどの LRU キューに保存するか決定する手法であり，標準的決定方法として「$\log_2$(参照回数) によりキューを選択する方法」と，標準的なキュー数として 8 本が示されている[2]．本評価もこの標準に従って行ったが，この場合は参照回数が 128 回以上になると，全てのデータが最上位の LRU キューに格納されてしまい，実質的に MQ の動作が LRU の動作とほぼ同一になってしまうことになる．これが MQ の性能が低くなった原因と考えられる．このことから，MQ を効果的に使用するにはパラメータを環境毎に適切に変更する必要があり，適切な最適化を行えないと下位キャッシュ環境における性能が極めて悪くなってしまうことがわかる．

6. 実ホスト OS におけるキャッシュヒット率の評価

6.1 評価環境

本章において，シミュレーションを用いずに，実際の仮想化環境における実ホスト OS のキャッシュのヒット率の評価を行う．上位キャッシュ(ゲスト OS キャッシュ)，下位キャッシュ(ホスト OS キャッシュ)の各キャッシュヒット率を求めるため，図 13 の様な実験環境を構築し，図 13 内の“Monitoring”の箇所を流れるアクセス要求を調査した．

図 13 内の“Monitoring”(A)から(D)の箇所を流れるアクセス要求は順に，アプリケーションから発行され上位キャッシュに入ってくるアクセス要求，上位キャッシュでキャッシュミスしホスト OS に送られるアクセス要求，上位キャッシュでキャッシュミスし下位キャッシュに送られてくるアクセス要求，下位キャッシュでキャッシュミスし物理 HDD に送られるアクセス要求である．また，各キャッシュミス率は，各キャッシュに入る前のアクセス要求量(図 23 の A や C)をキャッシュを経由した後のアクセス要求量(図 23 の B や D)で割ることにより近似的に求めることができる．なお上位キャッシュのモニタリングはブロック層，下位キャッシュのモニタリングは SCSI 層にて行なった．

図13の実験環境でFFSBを実行した．計算機の仕様は表7，表8の通り，FFSBの設定は表9の通りである．

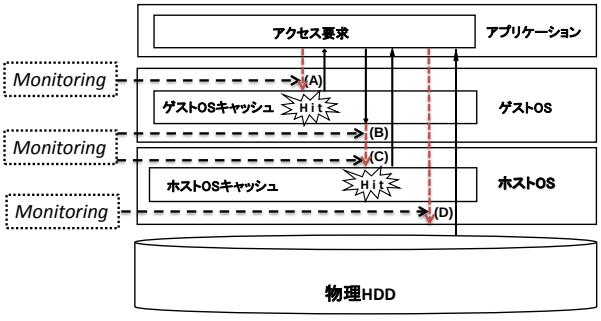

図 13 実験環境

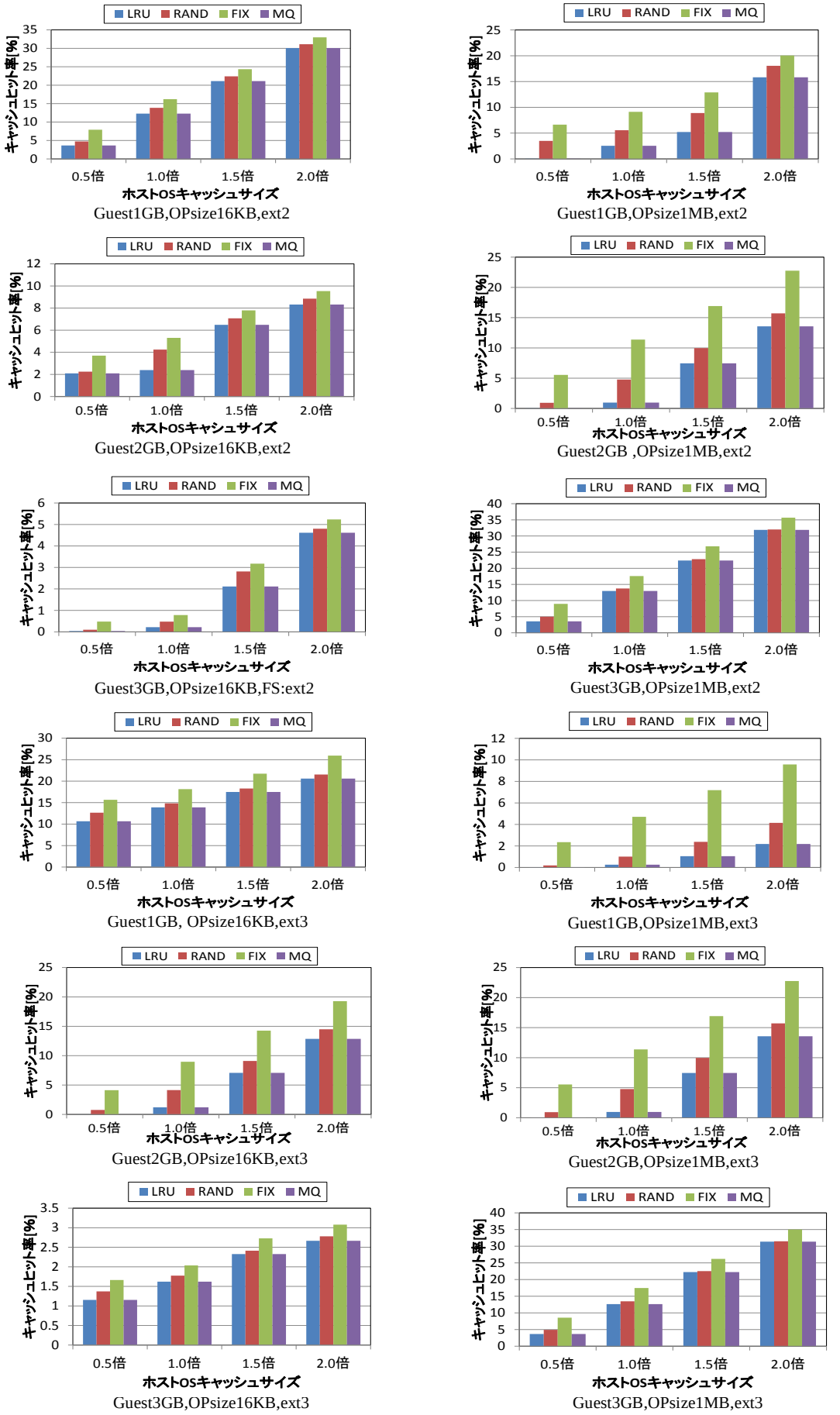

図 12　仮想化環境ホスト OS キャッシュにおけるキャッシュ置換アルゴリズムのヒット率

表 7 実計算機の仕様

OS	CentOS 6.2(64bit)
Kernel	Linux 2.6.32.57
CPU	Intel Celeron G1101
HDD	2TB
メモリ	2GB
仮想化システム	Xen 4.1.2
使用ファイルシステム	ext4

表 8 仮想計算機の仕様

OS	CentOS 6.2(64bit)
Kernel	Linux 2.6.32.57
CPU	Intel Celeron G1101
メモリ	500MB,1GB,1.5GB
Cache	237MB,744MB,1.3GB
使用ファイルシステム	ext4

表 9 FFSB の設定

Data Size[MB]	File Size[MB]	Num File	OPsize
50～1800	1	50～1800	4KB

6.2 評価結果

上位キャッシュ(ゲスト OS キャッシュ), 下位キャッシュ(ホスト OS キャッシュ)のキャッシュヒット率を図 14~16 に示す．左軸はキャッシュヒット率を示し，横軸は FFSB の総データサイズを示している．また右軸は FFSB によって得られたスループットを示している．なお，図 14 はゲスト OS にメモリを 512[MB],図 15 は 1[GB],図 16 は 1.5[GB]割り当ててキャッシュヒット率を求めた結果である．

すべての図において総データサイズが小さい状況におけるホスト OS のヒット率が存在していないが，これは総データサイズが小さい場合はゲスト OS におけるキャッシュヒット率が 100%となり，アクセス要求がホスト OS に到達しないためである．

両キャッシュヒット率とは，上位キャッシュあるいは下位キャッシュのいずれかでヒットした確率を近似的に求めたものである．これは，上位キャッシュに入ってくるアクセス要求量で，物理 HDD に対して発行されるアクセス量を割ったものである．

まず，ホスト OS のキャッシュヒット率について考察する．ゲスト OS キャッシュが 1.5GB で，ホスト OS キャッシュが 0.5GB の実験に着目すると，ホスト OS キャッシュのヒット率がほぼゼロとなっており，ホスト OS に与えたメモリが有効に活用されていないことが分かる．ゲスト OS キャッシュが 1GB でホスト OS キャッシュが 1GB の実験に着目しても，ホスト OS キャッシュのヒット率はゼロでは無いものの，データサイズの増加に対してヒット率が急激に減少しており，やはりホスト OS メモリが有効に活用されていないことが分かる．仮に参照に局所性が無く各ブロックに均等の確率でアクセスが行われた場合は，ホスト OS キャッシュのヒット率は「メモリサイズをデータサイズで割ったもの」となるはずである．しかし，図 15 におけるホスト OS キャッシュのヒット率はそれを超える速度で低下しており，負の参照の局所性によりホスト OS である Linux に実装されている LRU が効果的に機能できていないことが分かる．ホスト OS キャッシュサイズ方がゲスト OS キャッシュサイズより大きく負の参照の局所性の影響が最も小さいと期待される図 16 においても，ホスト OS キャッシュのヒット率は「メモリサイズをデータサイズで割ったもの」を大きく下回り，負の参照の局所性によりホスト OS キャッシュのヒット率が劣化してしまっていることが分かる．

次に，アプリケーションが得るスループットについて考察する．実験結果より，ゲスト OS に 1.5GB を割り当てた場合が最も性能が高いことが分かる．理由は主に 2 個考えられ，1 つはホスト OS に多くのメモリを割り当てても負の参照の局所性により効果的に利用できないからである．もう 1 つは，下位キャッシュにおいてキャッシュヒットが発生した場合より，上位キャッシュにおいてキャッシュヒットが発生した場合の方が CPU 処理量と応答時間が少ないことがあげられる．実験結果においても，両キャッシュヒット率の低下が無くてもゲスト OS キャッシュのヒット率が低下すればスループットの大幅な低下がみられ，ホスト OS キャッシュのヒットは性能向上の側面においてゲスト OS キャッシュヒットと同等でないことが確認できる．

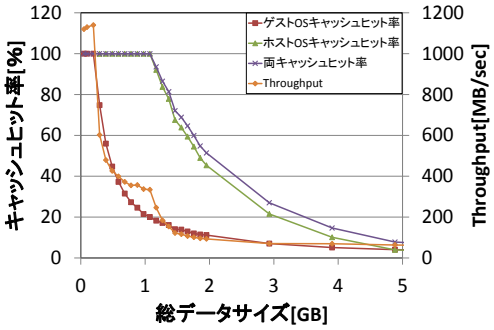

図 14 Guest 512MB キャッシュヒット率

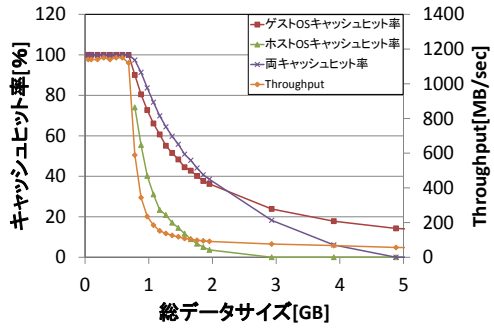

図 15 Guest 1GB キャッシュヒット率

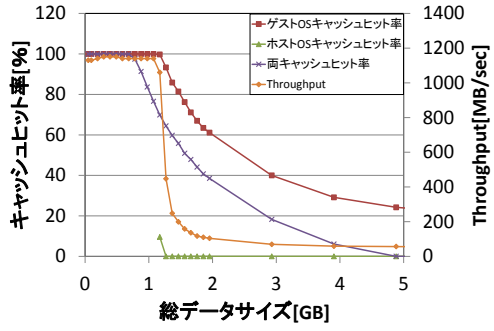

図 16 Guest 1.5GB キャッシュヒット率

7. 関連研究

7.1 MQ (Multi-Queue)

多段キャッシュ環境のための置換手法として MQ(Multi Queue) が提案されている[2]．下位キャッシュには負の参照の時間的局所性が存在しており，近い将来再度参照される確率は低い．これを考慮し，MQ は長時間データを保持することにより，キャッシュヒット率を高めること目指している. MQ は複数の LRU キューにより構成されており，各データは参照回数によって格納されるキューが選ばれる. 格納されたデータには expire time が設定されており，長期間アクセスが無かったデータは，下位のキャッシュに移される．置換対象となるデータは，最下位のキューの中で最後のアクセスからの時間が最も長いデータであり，置換対象となったデータの IDおよびその参照回数は Qout と呼ばれるキューに保管される．該当データが再度参照されキャッシュに格納される際に，該当データの参照回数として Qout に保管してあった値が加算され, 保管キューを選択する．

7.2 二重キャッシュ環境に関する研究

文献[4] にて，ネットワークストレージを用いる二重キャッシュ環境においてキャッシュデータの重複が生じ，これにより負の参照の時間的局所性, LRU 置換アルゴリズムの性能の低下が発生することが示されている．また，LRU を用いず，キャッシュ内容を固定化することで性能が向上することが示されている．ネットワークストレージ環境における既存のキャッシュ置換手法の評価として，Wilick らによる性能評価がある[5]．彼らはシミュレーションにより性能評価を行い, LRU がネットワークストレージ環境において高い性能を示すことができず, LFU の方が高い性能を示すことを確認している．

総合的にサーバ計算機とストレージ機器が協調して両者のキャッシュを管理する手法として Global L2 buer cache man-agement 手法がある[6]．本手法では両機器が通信を行いキャッシュを協調的に管理し，両キャッシュの重複を回避する．また Luo らは，ネットワークストレージのフロントエンドとしてプロキシを配置してキャッシュヒット率ではなくデータアクセスレイテンシを削減させる proxy cache scheme を提案している[7].

7.3 負の参照の時間的局所性に関する研究

負の参照の時間的局所性を考慮したキャッシュ置換手法として WC (Weighted Count) がある[8]. WC は LFU 同様にアクセス履歴を保管し，履歴にもとづき最も価値の低いと考えられるデータを破棄する．本手法は，一度参照されたデータは上位キャッシュに格納され，これが上位キャッシュから破棄されるまでの間は当該データへのアクセスが下位キャッシュに届くことはないことを前提とし，当該データが上位キャッシュから破棄されるのに最小でも上位キャッシュサイズ分のアクセスが必要であるとしている．よって，最後にアクセスされてからの発生アクセス回数が上位キャッシュサイズよりも小さい期間は当該データの価値は低いとして各データの価値を計算している．

ただし，本研究ではシミュレーションによる評価のみを行なっており，実記を用いた評価は行なっていない．

8. おわりに

本稿では，仮想化環境における参照の負の時間的局所性の調査を行った．その結果，ファイルシステムに ext3 を用いた場合は負の参照の時間的局所性が存在し，ext2 を用いた場合も負の参照の時間的局所性が存在することがあることを確認した．また，ホスト OS キャッシュにおける既存アルゴリズムの有効性を確認するために，当該調査で得られたアクセスログを使用してキャッシュヒット率の調査を行った．その結果， LRU と MQ の性能は低く，仮想化環境において，LRU や MQ が効果的に機能しないことが分かった．最後に，実際の仮想化環境におけるゲスト OS キャッシュとホスト OS キャッシュのヒット率を調査し，ホスト OS キャッシュのヒット率が低くなっていることを確認した．

今後は，負の参照の局所性に適した置換アルゴリズムに関する考察と，代表的なホスト OS である Linux への実装を行っていく予定である．

謝辞 本研究は JSPS 科研費 22700039, 24300034 の助成を受けたものである

参考文献

1) 長廻雄介，山口実靖，“ 多段キャッシュ型ネットワークストレージへのアクセスの時間的局所性を考慮したキャッシュ置換法” 情報処理学会 2010 年全国大会 5ZA-5.
2) Yuanyuan Zhou and James F .Philbin, KaiLi,“The Multi-Queue Replacement Algorithm for Second Level Buer Caches,” In Procedings of the 2001 USENIX Annual Technical Confer-ence, 2001.
3) P.J. Denning,“The Locality Principle, ”Com-munications of the ACM, Volume 48, Issue 7,pp. 1924, (2005)
4) 宮野新平，山口実靖，淺谷耕一，“多段キャッシュ型ネットワークストレージへのアクセスの時間的局所性を考慮したメモリキャッシュ制御”，社団法人情報処理学会，pp.7-12，20090226
5) D. L. Willick, D. L. Eager, and R. B. Bunt,“Disk Cache Replacement Policies for Network Fileservers, ”In ICDCS, May 1993
6) Yuanyuan Zhou and James F. Philbin, KaiLi“ Second-Level Buer Cache Management, ”IEEE Transactions on parallel and distributed systems, vol. 15, no. 7, JULY 2004
7) Yihui Luo and Changsheng Xie, Xinwei Zheng, Chengfeng Zhang, Zhen Zhao“A Proxy Cache Scheme of Network Storage ”IEEE Em-bedded Software and Systems, 2005
8) Yusuke Nagasako and Saneyasu Yamaguch,“ A Server Cache Size Aware Cache Re-placement Algorithm for Block Level Network Storage, ” The Fourth International Work-shop on Ad Hoc, Sensor and P2P Networks(AHSP2011), Jun. 2011
9) 竹内洸祐，山口実靖，“複数サーバ接続ネットワークストレージ環境での参照の局所性の解析“第24回 コンピュータシステム・シンポジウム (ComSys 2012)