情報処理学会論文誌　コンシューマ・デバイス & システム　Vol.5　No.4　42–51 (Oct. 2015)

コンシューマ・システム論文

# 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

杉本 洋輝[1,a)]　山口 実靖[1]

受付日 2014年12月21日, 採録日 2015年5月21日

**概要**：近年，クラウドコンピューティングの普及にともな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，ホスト OS ページキャッシュへのアクセスはゲスト OS ページキャッシュを介して行われる．このような二重キャッシュ環境下においてゲスト OS ページキャッシュの置換アルゴリズムに LRU が使用されている場合，ホストページキャッシュにおいては一度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が低くなり，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が実 OS で確認されている．また，二重キャッシュ環境ではゲスト OS ページキャッシュとホスト OS ページキャッシュに同一のデータを重複して格納している可能性が高い．このように，ホスト OS ページキャッシュは負の参照の時間的局所性やゲスト OS ページキャッシュとのデータの重複により有効に機能しない．本研究では，仮想化環境に適したページキャッシュ置換手法を提案し，評価によりその有効性を示す．

**キーワード**：仮想化，KVM，キャッシュ

# Cache Management Considering Negative Temporal Locality of Reference in Two-level Cache

Hiroki Sugimoto[1,a)]　Saneyasu Yamaguchi[1]

Received: December 21, 2014, Accepted: May 21, 2015

***Abstract:*** In virtualized environment, such as cloud computing environment, guest and host operating systems run simultaneously. Both of the operating systems have page caches for disk accesses. In such environment, the second level cache does not work effectively because of negative temporal locality of access and duplicated storing in both the caches. In this paper, we propose a method for increasing cache hit ratio of host operating system page cache. We present evaluation of cache hit ratio of the host operating system cache, and then demonstrate that our method can improve cache hit ratio.

***Keywords:*** virtualization, KVM, cache

## 1. はじめに

近年，クラウドコンピューティングの普及にともな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，ホスト OS のページキャッシュ（下位キャッシュ）へのアクセスはゲスト OS のページキャッシュ（上位キャッシュ）を介して行われる．以後，本稿における “キャッシュ” は OS カーネルのファイルアクセスにおけるページキャッシュを示す．このような二重キャッシュ環境下において上位キャッシュの置換アルゴリズムに LRU（Least Recently Used）[1] が使用されている場合，下位キャッシュにおいては一度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が低くなり，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が仮想化システム Xen [2] を用いた実環境で確認されている [3]．また，二重キャッシュ環境ではゲスト OS のキャッシュとホスト OS のキャッシュに同一のデータを重複して格納している可能性が高い．

このように，ホスト OS のキャッシュは負の参照の時間

[1] 工学院大学大学院電気・電子工学専攻
Graduate School of Electrical Engineering and Electronics, Kogakuin University, Shinjuku, Tokyo 163–8677, Japan
[a)] 5364gmms@gmail.com

的局所性やゲスト OS のキャッシュとのデータの重複により効果的に機能しないことが多い．よって，ホスト OS のキャッシュには負の参照の時間的局所性とデータの重複を考慮した置換手法が必要である．

また，近年普及が進んでいるクラウドコンピューティング環境では，仮想マシンの CPU やメモリの資源量を契約に含める形態がある．このような契約の場合，資源提供者側が自由に仮想マシンの資源を増減させることができず，仮想マシンの数が少なく資源に余裕のある物理マシンではメモリ資源をホスト OS が管理した状態のまま性能向上を図ることが重要となる．

本稿では，二重のキャッシュで構成される仮想化環境に適したキャッシュ置換手法を提案し，評価によりその有効性を示す．仮想化環境においては，単一の物理計算機上に複数の仮想計算機が稼働することがあるが，本稿では二重キャッシュ環境のヒット率向上を実現する手法の提案と単一仮想計算機環境を対象とした場合の評価結果を示す．

## 2. 関連研究

### 2.1 ページキャッシュアクセスにおける負の参照の時間的局所性

多くの場合，アプリケーションが発行するデータアクセス要求には「最近アクセスされたデータは近い将来再びアクセスされる可能性が高い」という参照の時間的局所性が存在し，LRU などのキャッシュ管理手法の多くはこの参照の局所性の概念に基づいている．そのため，参照の時間的局所性を考慮したキャッシュ置換手法を用いることにより，下層の低速記憶装置の記憶容量よりも小さいキャッシュでも多くの場合十分な性能を発揮することができる．

しかし，仮想化環境のような二重キャッシュ環境では通常とは逆向きの負の参照の時間的局所性が存在し，LRU などの参照の時間的局所性を期待している手法は効果的に機能しないと予想される．

ホスト OS のキャッシュへのアクセスは，図 1 のようにゲスト OS のキャッシュを介して行われる．アプリケーションから発行されたアクセス要求がゲスト OS のキャッシュ，ホスト OS のキャッシュの両キャッシュでキャッシュミスとなった場合，両キャッシュには同一のデータが格納される．しかし，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 のキャッシュ上で処理され，ホスト OS のキャッシュに届くことはない．したがってホスト OS のキャッシュでは “最近アクセスされたデータは近い将来再度アクセスされる可能性が低い” 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．

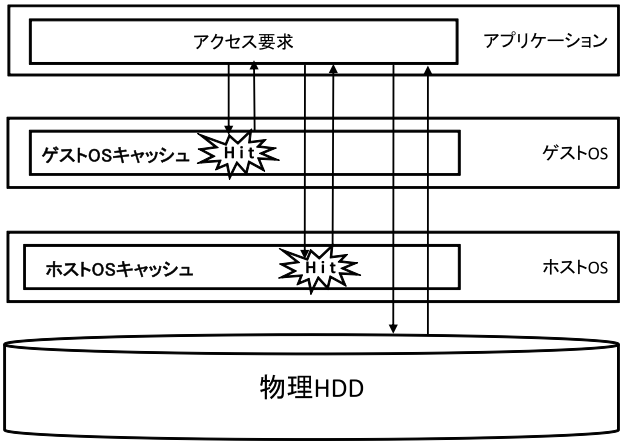


図 1　二重キャッシュ構造
**Fig. 1**　Two-level cache.

また，負の参照の時間的局所性はアプリケーションが使用するデータサイズが大きく，上位キャッシュのサイズが大きいほど影響が強くなることが確認されている [3]．

### 2.2 キャッシュ置換アルゴリズム

本節にて，本研究と関連する既存のキャッシュ置換手法を紹介する．

#### 2.2.1 LRU

参照の局所性を期待したキャッシュ置換手法に LRU 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 [4]．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．

仮に，本研究で対象とする上位キャッシュアルゴリズムに LRU が用いられる二重キャッシュ環境の下位キャッシュに，この LRU を適用すると，両キャッシュの LRU の動作は次のようになる．アクセス要求が上位キャッシュと下位キャッシュの両キャッシュでミスした場合，今回アクセスされたデータが両キャッシュに新規格納される．これにともない，両キャッシュで最後のアクセスからの時間が最長のデータが破棄される．今回アクセスされたデータは両キャッシュに重複して格納されることになる．アクセス要求が下位キャッシュでヒットした場合，下位キャッシュでは破棄と新規格納は起きないが，破棄優先度の変更が生じる．すなわち，今回アクセスされたデータが最も破棄されにくい状態に変わる．上位キャッシュでは今回アクセスされたデータ（ホスト OS から転送されたデータ）が新規格納され，上位キャッシュで最後のアクセスからの時間が最長のデータが破棄される．今回アクセスされたデータは両キャッシュに重複して格納されることになる．アクセス要求が上位キャッシュでヒットした場合，下位キャッシュはデータの破棄と新規格納および破棄優先順位の変更は

起きない．上位キャッシュでは破棄と新規格納は起きないが，破棄優先度の変更が生じる．

#### 2.2.2　MRU

MRU は「最後にアクセスされてからの時間が最も短い」データを破棄対象とする置換アルゴリズムである [5], [6]．Fetch-and-discard と呼ばれ，直前にアクセスしたデータをすぐに破棄する手法である．シーケンシャルアクセスなど，最近アクセスしたデータが近い将来に再度アクセスされることがない（あるいは少ない）状況などに適すると考えられる．

仮に，本研究で対象とする上位キャッシュアルゴリズムに LRU が用いられる二重キャッシュ環境の下位キャッシュに，この MRU を適用すると，両キャッシュの動作は次のようになる．両キャッシュミスの場合は，今回アクセスのデータが両キャッシュに新規格納され，上下のキャッシュそれぞれにおいて LRU と MRU に基づき最後にアクセスされてからの時間が最も長いデータと最も短いデータが破棄される．今回アクセスのデータは両キャッシュに重複格納される．下位キャッシュヒットの場合，下位キャッシュにて破棄と新規格納は生じないが，MRU に基づく破棄優先度の変更が生じる．

#### 2.2.3　2Q（two Queue）

2Q は Johnson らによって提案された 2 つのリストを用いるキャッシュ置換手法である [7]．FIFO 列 A1in と LRU 列 Am を用いてデータを保管し，初めてアクセスしたデータは A1in に格納し，2 回目以降は Am に格納する．置換が必要になった際，A1in のサイズが拡張可能閾値よりも小さく拡張可能である場合は Am の中で最も長い時間使われていないデータを置換対象に選択する．A1in が拡張可能でないときは A1in の中で最も古いデータを破棄し，その識別子を A1out に保管する．本手法は多重キャッシュ構造を考慮した手法ではないが，後述の MQ などの多重キャッシュ用置換アルゴリズムの研究にて参照されている．

二重キャッシュ環境における動作は，前述の LRU，MRU とほぼ同等である．すなわち，新規格納の対象は LRU，MRU と同一であり，破棄対象の決定および破棄優先度の変更が 2Q に基づいて行われる．

#### 2.2.4　MQ（Multi Queue）

MQ はネットワークストレージのキャッシュのような下位キャッシュのために提案されたアルゴリズムである [8]．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ことはなく，ある程度長い時間がたってからのみ再度アクセスされることを考慮し，データを履歴に長期間保管することでキャッシュヒット率の向上を目指したアルゴリズムである．MQ は 2Q を参考にしており，複数の LRU 列を維持してデータを管理する．各データをどの列に格納するかはアクセス頻度によって決定され，最下位の列で最も長い時間参照されていないデータを破棄対象とする．破棄したデータの ID とアクセス頻度が Qout 列に保管される．データが再びアクセスされときは，Qout に保管してあった情報に基づき格納する LRU 列を決定する．

LRU 列のデータには expire Time が設定されており，参照がないままこの時間に達すると，1 つ下位の LRU 列に移動される．LRU 列内のデータが参照されると，参照回数が更新され，新しい参照回数に基づき格納 LRU 列が再計算される．

二重キャッシュ環境における動作は，上記三手法とほぼ同等である．破棄対象の決定および破棄優先度の変更のみが MQ に基づいて行われる．

### 2.3　ネットワークストレージの下位キャッシュの性能に関する研究

文献 [9] 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 [10]．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（Least Frequently Used）[11] の方が高い性能を示すことを確認している．

これらの研究では，既存のキャッシュ置換手法の評価が行われているが，下位キャッシュに適した手法の提案はなされていない．

## 3.　二重キャッシュ環境における参照の時間的局所性の解析

### 3.1　二重キャッシュ環境における参照の局所性の解析

文献 [3], [9] では，ネットワークストレージや Xen を用いた仮想化環境における下位キャッシュに対する参照の局所性の解析を行っている．本稿では同一の調査を KVM（Kernel-based Virtual Machine）を用いて行い，本性質が Xen 以外の実装においても成り立つことを示す．**図 2** のような実験環境を仮想化システム KVM を用いて構築し，図内の “*Monitoring*” の箇所を流れるアクセス要求を調査した．図内の “*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 の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 2 の実験環境にてベンチマークソフトである FFSB（Flexible File System Benchmark）[12] をゲスト OS 上で実行した．仮想化環境において，実験に使用した計算機の仕様は**表 1**，**表 2** のとおり，FFSB の設定は**表 3** のとおりである．

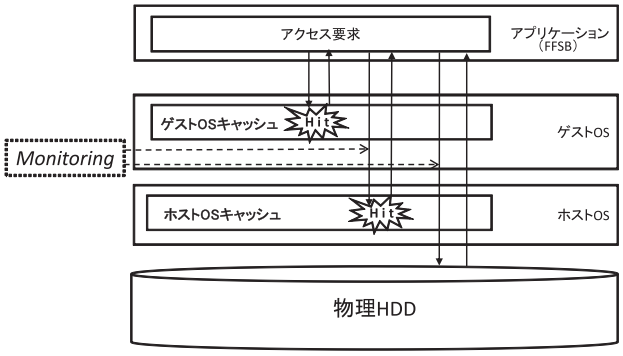


図 2　参照の局所性解析環境
**Fig. 2**　Experimental environment.

表 1　実計算機の仕様
**Table 1**　Specification of the physical machine.

| | |
|---|---|
| OS | CentOS 6.3 (64bit) |
| Kernel | Linux 2.6.32.57 |
| CPU | Intel Celeron |
| HDD | 3TB |
| Memory | 16GB |
| Cache | 15.45GB |
| 仮想化システム | KVM |
| ファイルシステム | ext2 |

表 2　仮想計算機の仕様
**Table 2**　Specification of the virtual machine.

| | |
|---|---|
| OS | CentOS 6.3 (64bit) |
| Kernel | Linux 2.6.32.57 |
| CPU | Intel Celeron |
| HDD | 50GB |
| Memory | 1GB,2GB |
| ファイルシステム | ext2 |

表 3　FFSB の設定
**Table 3**　FFSB setup.

| | |
|---|---|
| Data Size | 8GB |
| File Size | 1MB |
| File 数 | 8192 |
| Operation Size | 1MB |

### 3.2　キャッシュヒット率調査方法

キャッシュヒット率は以下の実装を用いて調査した．

アクセス要求量のモニタリングは，カーネル内の I/O 処理関数を I/O 処理時刻や I/O 要求のサイズの履歴を保存できるように改変することにより行った．

ゲスト OS のキャッシュ前のアクセス要求の量（キャッシュにアクセスするすべての要求の量）をモニタリングするために mm/filemap.c 内の do_generic_file_read 関数で I/O 処理を記録し，ゲスト OS のキャッシュ後のアクセス要求量（キャッシュミスしたアクセス要求の量）をモニタリングするために drivers/block/xen-blkfront.c 内の do_blkif_request 関数で I/O 処理を記録した．

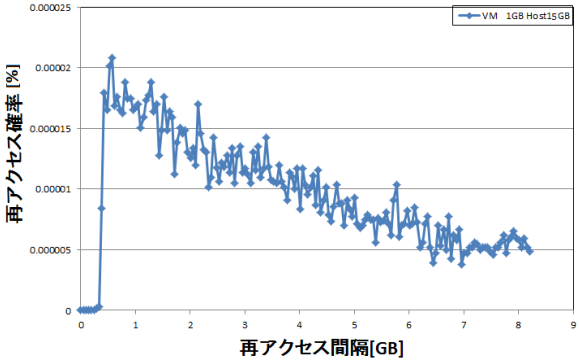


図 3　上位キャッシュ 1 GB 時の参照の局所性
**Fig. 3**　Locality of reference (first cache 1 GB).

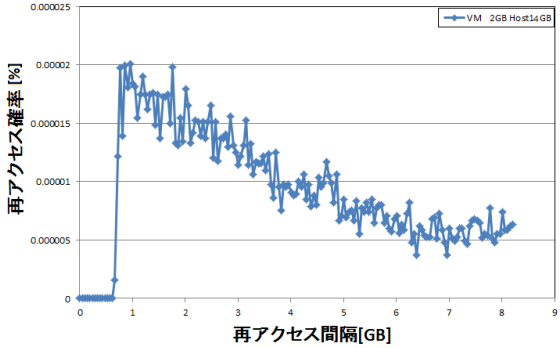


図 4　上位キャッシュ 2 GB 時の参照の局所性
**Fig. 4**　Locality of reference (first cache 2 GB).

また，ホスト OS のキャッシュ前のアクセス要求量をモニタリングするために drivers/xen/blkback/blkback.c 内の dispatch_rw_block_io 関数で I/O 処理を記録し，ホスト OS のキャッシュ後のアクセス要求量をモニタリングするために drivers/scsi/sd.c 内の sd_prep_fn 関数で I/O 処理を記録した．

### 3.3　解析結果

再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 3，図 4 に示す．再アクセス間隔とは，一度データへのアクセスがされてから，もう一度同じデータへのアクセスが来るまでに他のデータへ何 GB アクセスしたかを表している．

上位キャッシュサイズを変えた 2 つの測定結果の両方で負の参照の時間的局所性が確認できる．図 3 では再アクセス間隔 0.4 GB 以下，図 4 では再アクセス間隔 0.6 GB 以下の再アクセスがほとんど発生していない．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ことで再アクセスが発生しない間隔が長くなり，負の参照の局所性が強くなることが分かる．文献 [3], [9] と同様の結果が得られ，負の参照の時間的局所性は Xen 以外の実装においても存在することが確認された．

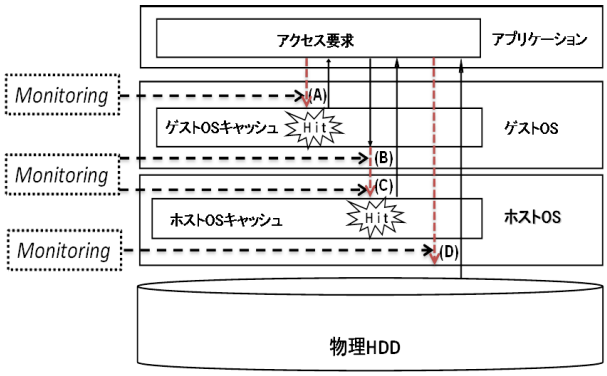


図 5　キャッシュヒット率測定環境

**Fig. 5**　Experimental environment of cahe hit ratio.

### 3.4　二重キャッシュ環境におけるキャッシュヒット率の測定

#### 3.4.1　測定方法

本章において，KVM を用いる仮想化環境のホスト OS のページキャッシュ（下位キャッシュ）のヒット率を求め，KVM 環境においても LRU を用いる下位キャッシュが効果的に動作しないことを示す．ゲスト OS のキャッシュ（上位キャッシュ），ホスト OS のキャッシュ（下位キャッシュ）のヒット率を求めるため，**図 5** のような実験環境を構築した．キャッシュヒット率は，図 5 の “*Monitoring*” の箇所を流れるアクセス要求の量を計測し，算出した．

図内の “*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たアクセス要求，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．ゲスト OS のキャッシュのミス率は，キャッシュ前のアクセス要求量（キャッシュにアクセスするすべての要求の量），図 5 の A をキャッシュを経由した後のアクセス要求量（キャッシュミスした量），図 5 の B で割ることにより近似的に求めることができる．ゲスト OS のキャッシュヒット率は，1 からゲスト OS のキャッシュミス率を引くことにより求められる．結果的に，ゲスト OS のキャッシュヒット率は，

1 – ((A) の量/(B) の量)

となる．同様に，ホスト OS のキャッシュヒット率は，

1 – ((C) の量/(D) の量)

となる．

図 5 の実験環境で FFSB を実行し，キャッシュヒット率を求めた．実計算機および仮想計算機の仕様は**表 4**，**表 5** のとおりである．

#### 3.4.2　測定結果

上位（ゲスト OS）キャッシュと下位（ホスト OS）キャッシュのヒット率，両キャッシュヒット率，ホスト OS のキャッシュサイズ/データサイズ比，FFSB のスループットを**図 6**，**図 7**，**図 8** に示す．ゲスト OS メモリサイズとは，VM に与えたメモリのサイズであり，ゲスト OS が使用できるメモリのサイズである．ゲスト OS は，このうちの一

表 4　実計算機の仕様

**Table 4**　Specification of the physical machine.

| | |
|---|---|
| OS | CentOS 6.3 (64bit) |
| Kernel | Linu2.6.32.57 |
| CPU | Intel Celeron |
| HDD | 3TB |
| Memory | 16GB |
| 仮想化システム | KVM |
| ファイルシステム | ext2 |

表 5　仮想計算機の仕様

**Table 5**　Specification of the virtual machine.

| | |
|---|---|
| OS | CentOS 6.3 (64bit) |
| Kernel | Linux 2.6.32.57 |
| CPU | Intel Celeron |
| 仮想 HDD | 50GB |
| Memory | 2GB～10GB |
| ファイルシステム | ext2 |

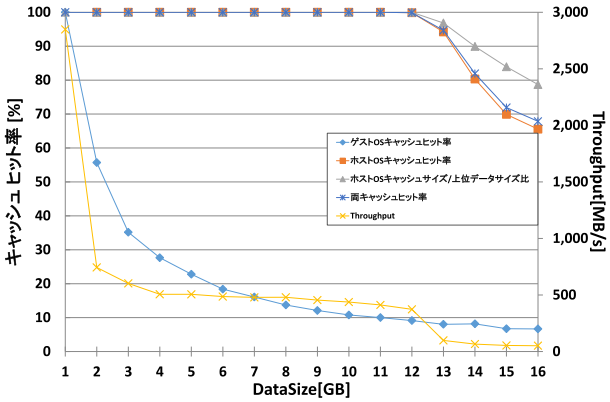


図 6　ゲスト OS メモリ 2 GB キャッシュヒット率

**Fig. 6**　Cache hit ratio (guest OS memory size 2 GB).

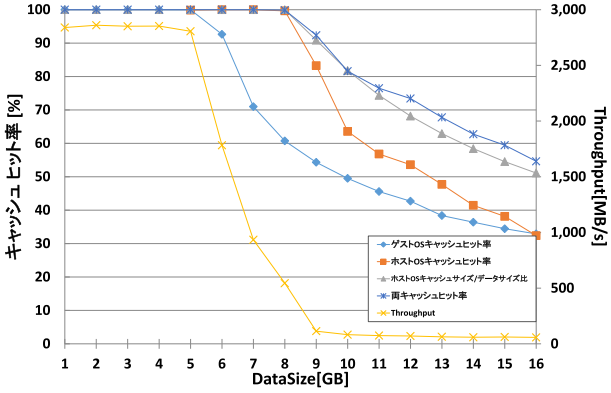


図 7　ゲスト OS メモリ 6 GB キャッシュヒット率

**Fig. 7**　Cache hit ratio (guest OS memory size 6 GB).

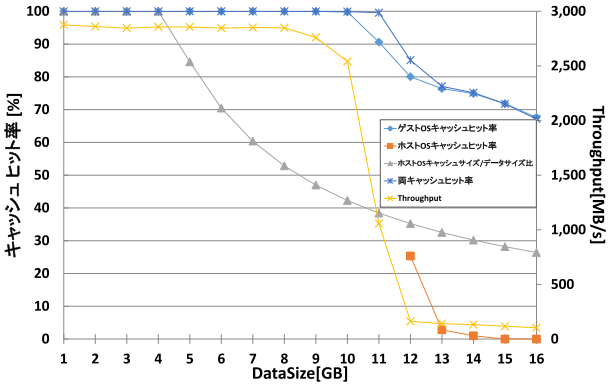


図 8　ゲスト OS メモリ 10 GB キャッシュヒット率
**Fig. 8**　Cache hit ratio (guest OS memory size 10 GB).

部をカーネルやユーザプロセスに割り当て，残りのすべてをキャッシュ（ゲスト OS のキャッシュ）に割り当てる．図の横軸は FFSB のデータサイズを示し，左縦軸はキャッシュヒット率を示し，右縦軸は FFSB のスループットを示している．両キャッシュヒット率とはゲスト OS キャッシュまたはホスト OS のキャッシュ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（両キャッシュミス率）を 1 から引いたものである．

すべての図において，ゲスト OS のキャッシュヒット率が 100%のときにホスト OS のキャッシュヒット率が存在しないのは，すべてのアクセス要求がゲスト OS のキャッシュで処理されるためホスト OS のキャッシュに要求がこないためである．

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 に 10 GB 割り当てた実験に着目すると，ホスト OS のキャッシュヒット率はほぼ 0%となっており，ホスト OS に割り当てたメモリが効果的に機能していないことが分かる．また，ゲスト OS のキャッシュサイズが大きいとき（図 8 など）は，ホスト OS のキャッシュヒット率が特に低いことが分かる．仮にホスト OS のキャッシュへの参照に局所性がなく，すべてのブロックに対して均等の確率でアクセス要求が来るとすると，ホスト OS のキャッシュヒット率は「ホスト OS のキャッシュサイズをデータサイズで割ったもの」と等しくなるはずである．多くの場合，参照には局所性がありキャッシュヒット率はこの比より高くなる．しかし，本実験結果ではホスト OS のキャッシュヒット率はこの比より低くなっており，ホスト OS に割り当てたメモリが負の参照の時間的局所性により効果的に機能していないことが分かる．これより，KVM を用いる環境においても文献 [3], [9] と同様に LRU を用いる下位キャッシュが効果的に機能していないことが分かる．

次にスループットに着目して考察する．スループットはゲスト OS キャッシュヒット率が 100%であるときに最も良く，次にホスト OS のキャッシュヒット率が 100%であるときが良い．ホスト OS のキャッシュヒット率が 100%のときが，ゲスト OS キャッシュヒット率が 100%のときに比べてスループットが低い理由として，ゲスト OS のキャッシュでヒットした場合よりホスト OS のキャッシュでヒットした場合の方が，アクセス要求に対する処理時間が長いことがあげられる．二重キャッシュ環境では，まずアクセス要求のあったデータが上位キャッシュに存在するかを確認し，存在しない場合は下位キャッシュに該当データが存在するかを確認する．よって，ホスト OS のキャッシュヒット時はゲスト OS キャッシュヒット時と比べ，アクセス要求のホスト OS への転送やホスト OS のキャッシュの確認などの追加の処理が必要となり，アクセス要求の処理時間が長くなる．これにより，スループットが相対的に低くなっていると考えられる．

## 4. 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の提案

### 4.1 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

二重キャッシュ環境では，上位キャッシュと下位キャッシュの両方でキャッシュミスした場合，両キャッシュには同一のデータ（HDD から読み込まれたデータ）が格納される．しかし，最近参照されたデータへのアクセス要求は上位キャッシュで処理され下位キャッシュには届かない．これを考慮し，ホスト OS のキャッシュの管理を以下のように行う手法を提案する．ホスト OS のキャッシュでは最後に参照されてからの時間が最も短いデータを破棄対象とする（MRU）手法を使用する．また，上位キャッシュから破棄されるデータを監視し，破棄データを下位キャッシュに最後に参照されてからの時間が最長（最も破棄されにくい状態）として格納する．上位キャッシュから破棄されたデータを下位キャッシュに格納する理由は，上位キャッシュから破棄されたデータは上位キャッシュに存在しないため，両キャッシュでのデータの重複が起きないためである．下位キャッシュに MRU を用いる既存手法と，提案手法を比較すると，破棄対象の決定は同一となっており，新規格納データの決定において異なっている．すなわち，両キャッシュミス時は，既存手法は今回アクセスのデータを新規格納し，提案手法は上位キャッシュ破棄データを新規格納する．下位キャッシュヒット時は，既存手法では新規格納は生じず，提案手法では上位キャッシュ破棄データを新規格納する．

各キャッシュのヒット時，ミス時の動作の詳細は以下のとおりである．

アクセス要求がゲスト OS のキャッシュでヒットした場合の動作を説明する（**図 9** 参照）．アプリケーションからページ D へのアクセス要求が来たとする．ページ D はゲ

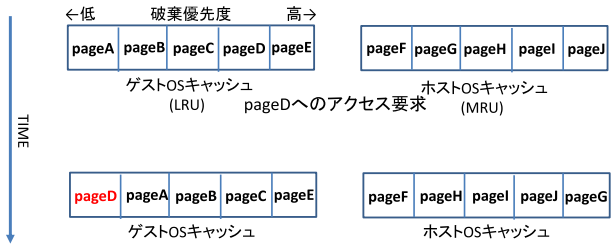


図 9　ゲスト OS キャッシュヒット時の動作
**Fig. 9**　Proposed method (guest OS cache hit).

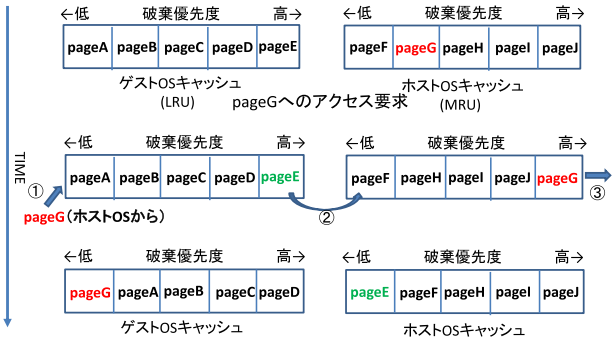


図 10　ホスト OS キャッシュヒット時の動作
**Fig. 10**　Proposed method (host OS cache hit).

スト OS キャッシュに存在するため，ゲスト OS は下位層にアクセス要求を転送せずアプリケーションにデータを返す．ゲスト OS キャッシュは置換アルゴリズムに LRU を用いているため，ページ D を最も破棄されにくい状態（LRU の先頭）とする．

次に，アクセス要求がホスト OS のキャッシュでヒットした場合の動作を説明する（**図 10** 参照）．アプリケーションからページ G へのアクセス要求が来たとする．提案手法では，ホスト OS のキャッシュは置換アルゴリズムを MRU としているため，ヒットしたページ（ページ G）をホスト OS のキャッシュにおいて最も破棄されやすい状態とする．次に，ゲスト OS はゲスト OS のキャッシュにページ G のコピーを格納する．ゲスト OS のキャッシュの最後尾データ（ページ E）はゲスト OS から破棄され，ホスト OS のキャッシュに最も破棄されにくい状態（最後に参照されてからの時間が最長）で格納される．これにともない，ホスト OS のキャッシュでは，最後に参照されてからの時間が最短のページ G が破棄される．

次に，アクセス要求が両キャッシュでミスした場合の動作を説明する（**図 11** 参照）．アプリケーションからページ K へのアクセス要求が来たとする．ページ K はゲスト OS キャッシュ，ホスト OS のキャッシュの両キャッシュに存在しないため，物理 HDD からデータの読み込みが行われる．そのとき，提案手法ではゲスト OS のキャッシュでのみページ K を格納し，ホスト OS のキャッシュにはページ K を格納しない．このとき，ゲスト OS キャッシュで破棄対象のページ E をホスト OS のキャッシュに最も破棄され

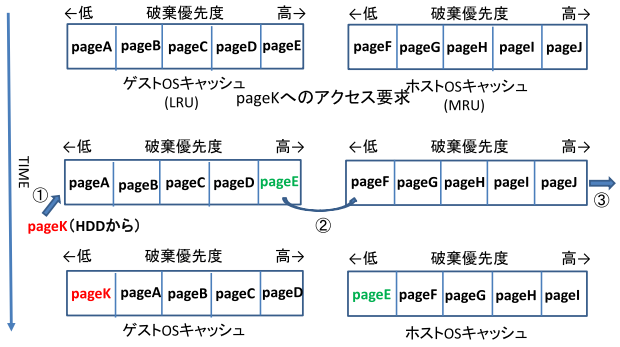


図 11　両キャッシュミス時の動作
**Fig. 11**　Proposed method (both caches miss).

にくい状態で格納する．また，ホスト OS のキャッシュで破棄対象のページ J を破棄する．提案手法は以上の動作により両キャッシュのデータの重複を抑えることができる．

## 5.　性能評価

シミュレーションにより既存手法，提案手法のキャッシュヒット率を評価した．シミュレーションではキャッシュおよびディスクは 4 KB ブロックで管理されているものとした．上位キャッシュの置換アルゴリズムには LRU を用い，下位キャッシュの置換アルゴリズムには LRU，MQ，FIFO，RADN，MRU，FIX，提案手法を用いそれぞれのキャッシュヒット率を求めた．キャッシュヒット率の計算は 3.4.1 項の計算方法と同一である．FIFO（First-In First-Out）は，最初にキャッシュに入った（キャッシュに入ってからの時間が最長の）データを破棄対象とするアルゴリズムである．RAND は，破棄対象をランダム（一様分布）に選択するアルゴリズムであり，LRU が効果的に機能しない状況では LRU よりも高い性能を示すと期待できる．LFU は，過去のアクセス履歴を保持し，これまでにアクセスされた頻度が最も低いデータを破棄対象とするアルゴリズムである．FIX はキャッシュ置換を行わないアルゴリズムである．RAND 同様に，LRU が効果的に機能しない状況では LRU よりも高い性能を示すと期待できる [8]．シミュレーションでは，大きなデータに対して細かいランダムアクセス要求を多数発行するアプリケーションをシミュレートした．データサイズは 10 GB とし，アクセス分布は一様分布とした．シミュレーションプログラムは Java 言語にて実装し，10 GB のデータ中から一様分布乱数を用いて 1 つのブロック（4 KB）を選択してアクセス要求を出す機能をプログラム内で実行させた．**図 12**，**図 13**，**図 14** にシミュレーション結果を示す．上位キャッシュの容量が 2 G，3 G，4 G の場合について，それぞれ下位キャッシュのサイズを 2 G，3 G，4 G，5 G と変化させて，キャッシュのヒット率を示したものが図 12 から図 14 である．

まず，各手法のヒット率を比較すると，すべての例において提案手法のヒット率が最も高く，提案手法が有効であ

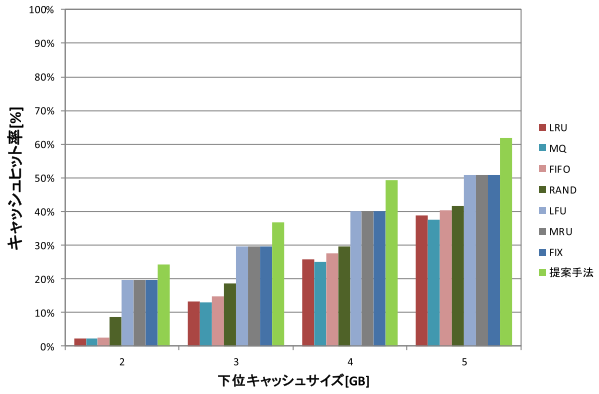


図 12　シミュレーション結果，上位キャッシュ 2 GB
**Fig. 12**　Simulation results (the first cache size 2 GB).

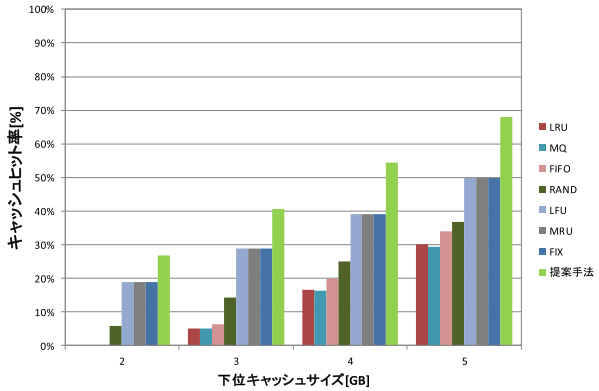


図 13　シミュレーション結果，上位キャッシュ 3 GB
**Fig. 13**　Simulation results (the first cache size 3 GB).

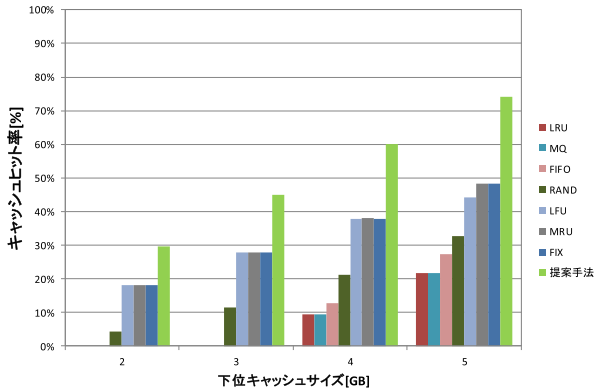


図 14　シミュレーション結果，上位キャッシュ 4 GB
**Fig. 14**　Simulation results (the first cache size 4 GB).

ることが分かる．

次に，LRU のヒット率について述べる．シミュレーションの結果より，LRU は今回使用したアルゴリズムの中で最もヒット率が低いことが分かる．また，下位キャッシュのサイズが同じ場合，上位キャッシュのサイズを増やすと下位キャッシュヒット率が低下することが分かる．この理由は 2 つ考えられる．1 つは上位キャッシュのサイズを大きくしたことにより負の参照の時間的局所性の影響が強くなり，参照の時間的局所性を期待している LRU が効果的に機能しなかったためである．もう 1 つの理由は，上位キャッシュのサイズが大きくなったことにより，両キャッシュのデータの重複が増えたため，上位キャッシュでミスしたデータは下位キャッシュでもミスする可能性が高くなり，下位キャッシュのヒット率が低下したと考えられる．

次に，下位キャッシュの置換アルゴリズムに提案手法を選択した場合のヒット率に着目して考察する．LRU のヒット率が上位キャッシュサイズの増加とともに低下するのに対し，提案手法のヒット率は上位キャッシュサイズの増加ともに向上していることが分かる．これは，提案手法ではデータの重複が発生しないため，上位キャッシュがより多くのデータを保持すると，下位キャッシュにアクセス要求が来るデータの種類が減少するからであると考えられる．たとえば，アプリケーションがアクセスするデータファイルのサイズが 10 GB であり，両キャッシュに重複が生じない場合は，次のようになる．上位キャッシュが 2 GB のデータを保持しているとき，下位キャッシュに来るアクセス要求は残りの 8 GB のデータの中のブロックに対するものとなる．これに対して上位キャッシュが 4 GB のデータを保持しているときは，下位キャッシュに来るアクセスは残りの 6 GB のデータの中のブロックに対するものとなる．

次に，下位キャッシュの置換アルゴリズムに MQ を選択した場合のヒット率に着目して考察する．下位キャッシュの置換アルゴリズムに MQ を選択した場合は LRU と変わらない性能となり，MQ が効果的に機能していないことが分かる．MQ は参照回数に基づき格納する LRU 列を決定するが，今回の測定ではすべてのデータが 1 つの LRU 列に入ってしまい実質的に LRU と同じ動作となり，同等の性能となった．今回の測定では，MQ の文献 [8] で推奨されている $log_2(f)$（f は参照回数）に基づき格納 LRU 列を決定したが，これでは効果的に動作しないことが分かり，MQ ではこの決定方法を調整しないと高い性能を示せないことが分かった．

## 6. 考察

### 6.1 提案手法の実現性について

本稿の提案手法では，ゲスト OS のキャッシュで破棄されたデータをホスト OS に伝える必要がある．その実現方法について考察する．

まず，提案手法を実装した API（ゲスト OS がホスト OS に伝達するための API）を用意し，本 API に対応したゲスト OS がこの API を使用して伝達する方法が考えられる．たとえば IaaS（Infrastructure as a Service）型のクラウドコンピューティングサービスでは，ゲスト OS の実装を契約者が自由に選ぶことができ，希望する契約者が本 API に対応したゲスト OS 実装を使用することにより提案手法を使用することが可能になると考えられる．

また，契約者が同意した場合はホスト OS や仮想化シス

テムが仮想計算機プロセスを監視し，ゲスト OS のキャッシュの破棄メモリの内容を把握して提案手法を実現する手法も考えられる．仮想計算機プロセスは仮想化システムの管理下で動作するためこの実現手法も考えられ，この場合はゲスト OS への修正は不要となる．

### 6.2　マルチゲスト OS 使用時の性能について

本稿では，研究の第一歩としてシングルゲスト OS（単一仮想計算機）環境にて評価を行った．マルチゲスト OS 環境に本提案手法を用いると，複数のゲスト OS のキャッシュと単一のホスト OS のキャッシュが動作することになる．複数ゲスト OS のキャッシュで単一のホスト OS のキャッシュを共有している状況となり，各ゲスト OS のキャッシュが破棄したデータがホスト OS のキャッシュに格納されている状況である．これは，各ゲスト OS のキャッシュが保持していないデータをホスト OS のキャッシュが格納している状況であり，ホスト OS のキャッシュがゲスト OS のキャッシュを補っている状態であるといえる．特に，各ゲスト OS のキャッシュが最近破棄したデータがホスト OS のキャッシュに格納されているため，ホスト OS のキャッシュが効率的にゲスト OS のキャッシュを補っている状況であるといえる．

次に，マルチゲスト OS 環境における上下両キャッシュのデータ重複と，負の参照の局所性の影響について述べる．本提案手法は，ゲスト OS のキャッシュの破棄データをホスト OS のキャッシュに格納するようになっており，両キャッシュミス時もそのときのアクセスデータは I/O が発行されたゲスト OS のキャッシュにのみ格納される．よって，マルチゲスト OS 環境であっても本提案手法の重複排除の性質は保たれる．また，負の参照の時間的局所性によりアクセスされないデータ（すなわちゲスト OS のキャッシュに格納されているデータ）はホスト OS のキャッシュには格納されないため，マルチゲスト OS 環境であっても負の参照の時間的局所性によるホスト OS のキャッシュヒット率の低減は回避できる．また，仮想計算機のイメージファイルは複数のゲスト OS で共有することがないため，単一のデータを複数のゲスト OS のキャッシュで重複して格納することもない．以上より，本手法はマルチゲスト OS 環境適用時も高い性能を発揮できると期待できる．

## 7.　おわりに

本稿では，二重キャッシュ環境における負の参照の時間的局所性の紹介し，下位キャッシュにて LRU が効果的に機能しないと予想されていることを述べた．そして，KVM を用いた仮想化環境を構築し，同環境の下位キャッシュ（ホスト OS のキャッシュ）へのアクセスの解析結果を示し，同環境においても負の参照の時間的局所性が存在することを示した．これにより，同局所性が多くの実装による二重キャッシュ環境に存在することを確認した．そして，KVM 環境の下位キャッシュヒット率を観測するシステムを構築し，同環境においても LRU を用いる下位キャッシュのヒット率が低いことを示した．

これに対して我々は，下位キャッシュの破棄対象を最後にアクセスされてからの時間の短さで決定し，上位キャッシュの破棄データを下位キャッシュに新規格納させる手法を提案した．そして，二重キャッシュ環境の下位キャッシュにおける既存手法と提案手法のヒット率をシミュレーションにより評価した．評価の結果，すべてのシミュレーションにおいて提案手法のヒット率がすべての既存手法（多くの OS で採用されている LRU や二重キャッシュ環境を考慮した 2Q や MQ）のヒット率を上回ることが確認され，本提案手法が二重キャッシュ環境の下位キャッシュにおいて有効であることが確認された．特に，上位キャッシュサイズが大きい状況において提案手法は既存手法を大きく上回っており，負の参照の局所性が強い環境で有効であることが分かった．

今後は，複数仮想計算機環境における考察，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 24300034，25280022，26730040 の助成を受けたものである

### 参考文献

[1] Tanenbaum, A.S. and Woodhull, A.S.: *Operating Systems Design and Implementation*, 3rd Edition, pp.401–403, Pearson (2006).
[2] Xen Project: The Xen Project, the powerful open source industry standard for virtualization, available from ⟨http://www.xenproject.org/⟩ (accessed 2015-03-08).
[3] 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）(Sep. 2012).
[4] Denning, P.J.: The Locality Principle, *Comm. ACM*, Vol.48, Issue 7, pp.19–24 (Jul. 2005).
[5] Denning, P.J.: The Working set Model for Program Behavior, *Comm. ACM* (May 1968).
[6] Coffman, E.G. Jr. and Denning, P.J.: *Operating Systems Theory*, Prentice Hall Professional Technical Reference (Oct. 1973).
[7] Johnson, T. and Shasha, D.: 2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm, *Proc. 20th International Conference on Very Large Data Bases* (Sep. 1994).
[8] Zhou, Y., Philbin, J.F. and Li, K.: Second-Level Buffer Cache Management, *IEEE Trans. parallel and distributed systems*, Vol.15, No.7 (Jun. 2004).
[9] 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，情報処理学会研究報告，マルチメディア通信と分散処理研究会報告 2009, No.20, (2009-DPS-138)，pp.7–12 (Feb. 2009).
[10] Willick, D.L., Eager, D.L. and Bunt, R.B.: Disk Cache Replacement Policies for Network Fileservers, *Proc. IEEE International Conference on Distributed Com-*

*puting Systems* (*ICDCS '93*) (May 1993).
[11] Lee, D., Choi, J., Kim, J.H., Noh, S.H., Min, S.L., Cho, Y. and Kim, C.S.: LRFU: A Spectrum of Policies that Subsumes the Least Recently Used and Least Frequently Used Policies, *IEEE Trans. Computers*, Vol.50, No.12, pp.1352–1361 (Dec. 2001).
[12] Flexible File System Benchmark, (online) available from ⟨http://sourceforge.net/projects/ffsb/⟩ (accessed 2015-03-08).

**杉本 洋輝**　（正会員）

2014 年工学院大学工学部情報通信工学科卒業．同年同大学大学院工学研究科電気・電子工学専攻入学．仮想化の IO に関する研究に従事．

**山口 実靖**　（正会員）

2002 年東京大学大学院工学系研究科電子情報工学専攻博士課程修了．博士（工学）．同年より東京大学生産技術研究所学術研究支援員，産学官連携研究員，日本学術振興会特別研究員．2006 年工学院大学工学部講師．2007 年同大学同学部准教授．オペレーティングシステム，I/O 高速化の研究に従事．電子情報通信学会，日本データベース学会各会員．