Vol.2015-DPS-163 No.19
Vol.2015-MBL-75 No.19
2015/5/28

ゲスト OS ページキャッシュの監視による ホスト OS ページキャッシュのヒット率の向上手法の試作と評価

杉本 洋輝[†1]　光来　健一[†2]　山口 実靖[†1]

仮想化環境ではゲスト 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）に同一のデータを重複して格納している可能性が高い．このため，ホスト OS キャッシュは効果的に機能しない．上位キャッシュが破棄したページは下位キャッシュと重複しないため，上位キャッシュの破棄したページを下位キャッシュが優先的に保持することでホスト OS キャッシュヒット率の向上ができると期待でき，我々は過去に上位キャッシュが破棄したページを下位キャッシュに格納する手法を提案し，シミュレーションによりその有効性を示した．本稿では，仮想化環境のゲスト OS を改変しゲスト OS ページキャッシュが破棄したページをホスト OS に転送しホスト OS ページキャッシュに格納する手法の実装を紹介する．そして，本実装を用いた性能評価を紹介し提案手法の有効性を示す．

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，ホスト OS 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 ページキャッシュ(上位キャッシュ)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上位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，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[8]が存在することが仮想化システム Xen[1]を用いた実環境での検証[2]や仮想化システム KVM を用いた検証[10]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とが確認されている[2][10].

この様に，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効果的に機能しないことが多い．よって，ホスト OS ページ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，契約に基づき物理マシン上に仮想マシンを稼働させ契約者に提供する．このような例では，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[10]に着目し，Linux を用いたその実現方法について考察する．そして，その試作実装を用いた評価を行い，本手法の実環境における性能について考察を行う．

†1 工学院大学大学院工学研究科電気・電子専攻†Electrical Engineering and Electronics, Kogakuin University Graduate School
†2 九州工業大学
Kyushu Institute of Technology

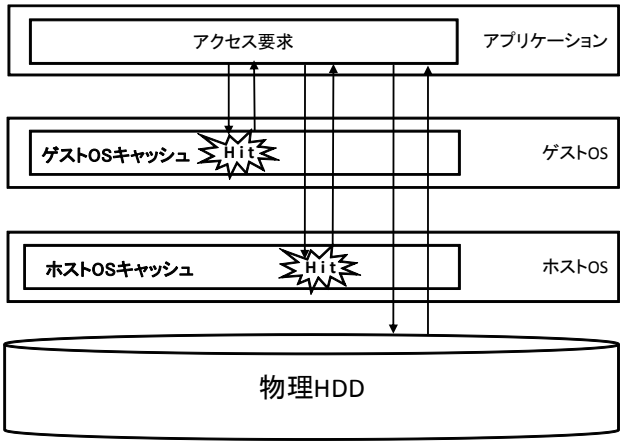

図 1 二重キャッシュ構造

2. 関連研究

2.1 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 ページキャッシュへのアクセスは，図 1 のようにゲスト OS ページ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 ページキャッシュ，ホスト OS ページキャッシュの両ページキャッシュでキャッシュミスとなった場合，両ページ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 ページキャッシュ上で処理され，ホスト OS ページキャッシュに届くことはない．従ってホスト OS ページキャッシュでは“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．

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[2].

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 (Least Recently Used) 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．

仮想化環境のような二重キャッシュ環境において，LRU の動作，新規格納と破棄データは次のようになる．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．これに伴い，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づらい状態に変わる．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[4][5]．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．

上位キャッシュに 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 MRU が用いられたときの両キャッシュの動作，新規格納と破棄データは次のようになる．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．上位キャッシュにおいては，今回アクセスのデータが新規格納され，LRU に基づくデータが破棄される．上位キャッシュヒットの場合，下位キャッシュにおいては，優先度の変更，破棄と新規格納は起きない．上位キャッシュにおいて優先度の変更が生じ，破棄と新規格納は起きない．

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[6]．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[7]．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もとづき格納する 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，新しい参照回数にもとづき格納 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[8]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[9]．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．

Vol.2015-DPS-163 No.19
Vol.2015-MBL-75 No.19
2015/5/28

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．

2.4 仮想化環境におけるキャッシュヒット率の調査

文献[10]にて，仮想化環境において負の参照の時間的局所性やキャッシュデータの重複が生じホスト OS ページキャッシュが効果的に機能しないことを示した．

また，同論文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を考慮したキャッシュ管理手法を提案している．3 章にて提案手法を紹介する．

3. 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紹介

本章にて文献[10]で提案している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シミュレーションによる評価について紹介する．

3.1 負の参照の時間的局所性考慮したキャッシュ置換手法

仮想化環境のような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 ページキャッシュの管理を以下の様に行う手法を提案する．ホスト OS ページ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通りである．

アクセス要求がゲスト OS キャッシュでヒットした場合の動作を説明する（図 2 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 を最も破棄されづらい状態（LRU の先頭）とする．

次に，アクセス要求がホスト OS ページキャッシュでヒットした場合の動作を説明する（図 3 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 ページ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 ページ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 キャッシュにページ G のコピーを格納する．ゲスト OS 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 ページキャッシュに最も破棄されづらい状態（最後に参照されてからの時間が最長）で格納される．これに伴い，ホスト OS ページキャッシュでは，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．

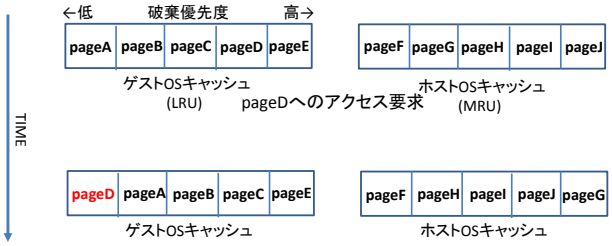

図 2 ゲスト OS キャッシュヒット時の動作

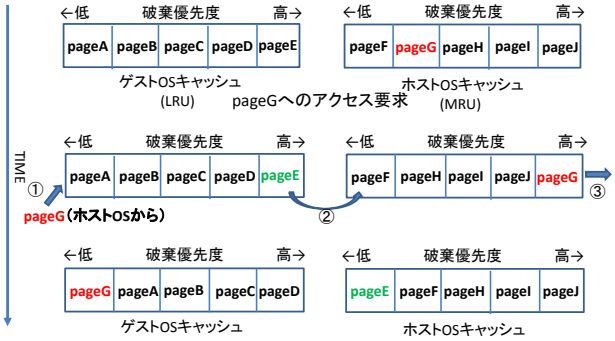

図 3 ホスト OS キャッシュヒット時の動作

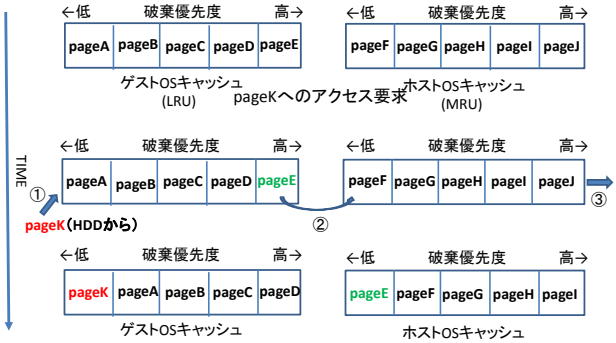

図 4 両キャッシュミス時の動作

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 4 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 ページ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．その時，提案手法ではゲスト OS キャッシュでのみページ K を格納し，ホスト OS ページキャッシュにはページ K を格納しない．この時，ゲスト OS キャッシュで破棄対象のページ E をホスト OS ページキャッシュに最も破棄されにくい状態で格納する．また，ホスト OS ページ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事ができる．

3.2 シミュレーションによる評価

Vol.2015-DPS-163 No.19
Vol.2015-MBL-75 No.19
2015/5/28

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の評価を行った[10]．シミュレーションではキャッシュおよびディスクは 4KB 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．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 (Least Frequently Used)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[7]．データサイズは 10GB とし，アクセス分布は一様分布とした．シミュレーション結果は図 5 から図 7 の通りである[10]．

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ることがわかる．

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わかる．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わかる．この理由は 2 つ考えられる．一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．もう一つの理由は，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わかる．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．例えば，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 10GB であり，両キャッシュに重複が生じない場合は，次の様になる．上位キャッシュが 2GB 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 8GB 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 4GB 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 6GB 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わかる．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[7]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わかり，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わかった．

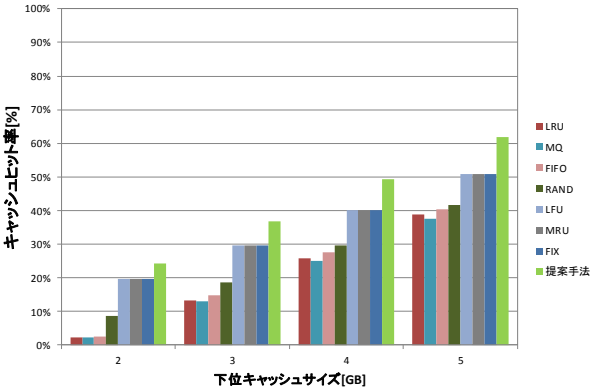

図 5 文献[10]におけるシミュレーション結果，上位キャッシュ 2GB

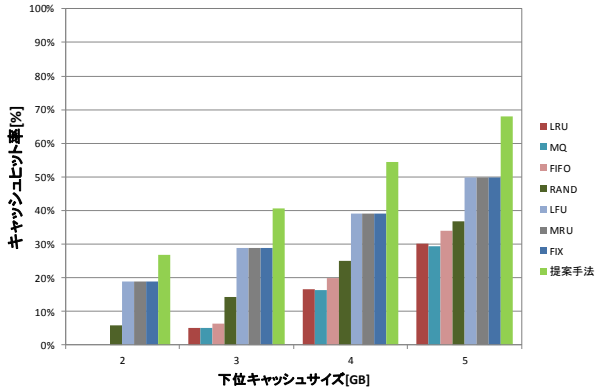

図 6 文献[10]におけるシミュレーション結果，上位キャッシュ 3GB

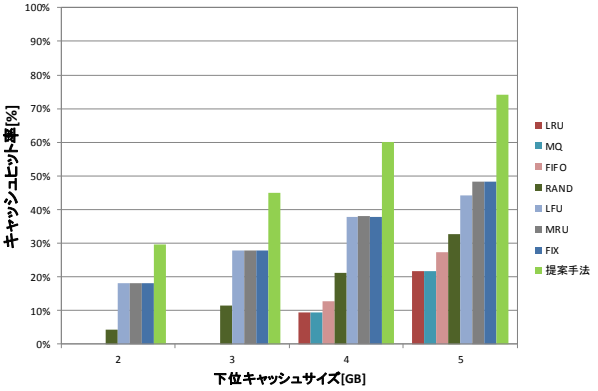

図 7 文献[10]におけるシミュレーション結果，上位キャッシュ 4GB

Vol.2015-DPS-163 No.19
Vol.2015-MBL-75 No.19
2015/5/28

4. ゲスト OS ページキャッシュの監視によるホスト OS ページキャッシュのヒット率の向上手法の試作実装

4.1 試作実装

本節にて前述した提案手法の試作実装について述べる．試作実装では，ゲスト OS ページキャッシュ（上位キャッシュ）から破棄されるページを監視し，破棄ページデータを VM イメージファイルの当該箇所に対して書き込みを行うことによりページをホスト OS ページキャッシュ（下位キャッシュ）に格納する．仮想化システムには KVM を，ゲスト OS とホスト OS には Linux を用いる．

ゲスト OS ページキャッシュから破棄されるページを監視は，カーネル内のページキャッシュ処理関数をキャッシュから破棄されるページのブロック番号(ファイル index)やページデータを保存できる様に改変することにより行った．ゲスト OS のページキャッシュから破棄されるページを監視するために，`mm/filemap.c` 内の `__remove_from_page_cache` 関数で処理を監視した．破棄されるページがあった場合，ゲスト OS からホスト OS にページデータやブロック番号を転送し，ホスト OS では転送されてきたページデータを VM イメージファイルの当該箇所に対して書き込みを行うことによりページをホスト OS ページキャッシュに格納した．

4.2 試作実装の評価実験

本節において，試作実装を用いて提案手法の評価実験を行う．ゲスト OS 上に 20GB のファイルを作成し当該ファイルの先頭から 16GB 目までに対してランダムリードを行うベンチマークプログラムを実行し，提案手法を適用した場合と提案手法を適用しない場合でホスト OS ページキャッシュのヒット率を比較した．ランダムアクセスは指数分布乱数で行った．表 2，3，4 に実験環境を，図 8 に実験結果を示す．図より，提案手法の適用によりホスト OS キャッシュヒット率の大幅な向上が見られ，提案手法が試作実装を用いた実環境においても有効であることが分かる．

表 1 実計算機の仕様

OS	CentOS 6.3 (64bit)
Kernel	Linux 2.6.32.27
CPU	Intel Celeron(R) CPU G530
HDD	250GB
Memory	16GB
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

OS	CentOS 6.3 (64bit)
Kernel	Linux 2.6.32.57
CPU	Intel Celeron(R) CPU G530
HDD	50GB
Memory	10GB
使用ファイルシステム	ext2

表 3 ベンチマークプログラムの仕様

データサイズ	16GB
総読み込み量	64GB
Read size	16MB
アクセスアドレスの偏り	指数分布

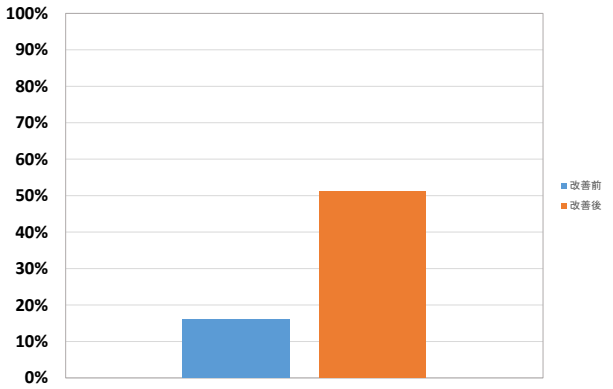

図 8 ホスト OS キャッシュキャッシュヒット率

4.3 考察

本節にて，試作実装を用いて構築した環境と，実環境の差違について考察する．

試作実装を用いる環境では，上位キャッシュで破棄されたデータをゲスト OS のカーネル空間からゲスト OS のユーザ空間に転送し，さらにゲスト OS のユーザ空間からホスト OS のユーザ空間に転送し，これを VM イメージファイルに対して書き込んでいる．これらの処理は少なくないオーバーヘッドとなり，性能劣化の原因につながると予想できる．しかし，本稿ではアプリケーション性能は評価しておらず，ホスト OS ページキャッシュのヒット率のみを評価しているため，この差違は本稿の評価には影響を与えておらず，本稿で行った評価は実環境においても有効であると予想される．

次に，試作実装と提案手法のホスト OS データ管理手法の差違について考察する．提案手法では，下位ページキャッシュは MRU にて管理することになっているが，試作実装ではホスト OS ページキャッシュは LRU にて管理されている．この違いはホスト OS ページキャッシュでヒット時に現れる．提案手法では下位キャッシュでヒットしたページは「最も破棄されやすいページ」として扱われるが，試作実装では「最も破棄されづらいページ」として扱われる．ホスト OS ページキャッシュでヒットしたページはゲスト OS ページキャッシュにも格納されるため，この動作はデータの重複及びホスト OS ページキャッシュヒット率の低下に繋がっていると予想される．これが，試作実装における性能が 3 章で紹介した性能より低い理由であると考えられる．ゲスト OS ページキャッシュで破棄されたページを「最も破棄されづらい状態」でホスト OS ページキャッシュに格納させることに関しては，イメージファイルの書き込みにより LRU の「最も破棄されづらい状態」で格納されているため，ほぼ実現できていると言える．

また，ゲスト OS ページキャッシュで破棄されたデータをホスト OS ページキャッシュに格納するのに，書き込みを行っていることについて考察する．書き込みを行うと，データがページキャッシュに格納されるが，当該データを HDD に対して書き込む必要が生じる．よってこれがディスクアクセス回数を増加させると考えられる．ただし，本稿測定では，ホスト OS ページキャッシュへの読み込み回数とページキャッシュミス回数（HDD への読み込み回数）の比を求めているため，この差異は本稿の測定に影響を与えていない．

Vol.2015-DPS-163 No.19
Vol.2015-MBL-75 No.19
2015/5/28

5. 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ことを述べた．そして，二重キャッシュに適したページキャッシュ管理手法を紹介し，その試作実装と試作実装を用いた性能評価を示した．性能評価より，提案手法は試作実装の環境において効果的に機能することが分かり，提案手法の有効性が確認された．

今後はゲスト OS ページキャッシュ破棄データのホスト OS ページキャッシュ格納機能の改善，ホスト OS ページキャッシュの MRU 化などを行っていく予定である．

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

参考文献

[1]. Xen Project: The Xen Project, the powerful open source industry standard for virtualization, available from < http://www.xenproject.org/> (accessed 2015-03-08).
[2]. 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).
[3]. P. J. Denning, “The Locality Principle,” Communications of the ACM, Volume 48, Issue 7, pp. 19-24, (Jul. 2005).
[4]. P.J Denning, “The Working set Model for Program Behavior” Communications of the ACM, (May. 1968).
[5]. E.G. Coffman, Jr., P.J Denning. “Operating Systems Theory,” Prentice Hall Professional Technical Reference, (Oct. 1973).
[6]. Theodore Johnson, Dennis Shasha, “2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm,” Proceedings of the 20th International Conference on Very Large Data Bases, (Sep. 1994).
[7]. Yuanyuan Zhou, James F. Philbin, Kai Li “Second-Level Buffer Cache Management,” IEEE Transactions on parallel and distributed systems, vol. 15, no. 7, (Jun. 2004).
[8]. 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).
[9]. D. L. Willick, D. L. Eager, R. B. Bunt, “Disk Cache Replacement Policies for Network Fileservers,” Proceedings of IEEE International Conference on Distributed Computing Systems (ICDCS '93), (May. 1993).
[10]. 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”，マルチメディア，分散，協調とモバイル (DICOMO2014)シンポジウム, pp. 867-872, (Jul 2014).