Vol.2018-OS-143 No.3
2018/5/21

キャッシュファイルシステムによる下位キャッシュのアクセスの局所性の改善による VM I/O 性能の向上

吉田光太朗[†1]　光来健一[†2]　山口実靖[†1]

概要：クラウド環境の普及に伴い，仮想化環境の重要性が高まっている．ハイパーバイザ型やホスト型の仮想化環境では，ストレージアクセスはゲスト OS とホスト OS の二重のストレージ用キャッシュを介して行われる．既存研究において，二重キャッシュ環境下のホスト OS キャッシュでは LRU 置換アルゴリズムは有効に機能しないことと，ホットスポットをキャッシュ対象領域としてゲスト OS キャッシュデータを固定化することにより仮想マシンの I/O 性能が向上することが示されている．本稿では，既存研究で静的に定めていたキャッシュ対象領域を動的に制御する新しいファイルシステムを提案する．本ファイルシステムでは，データへのアクセスを監視し，データへのアクセス頻度を計算することによりホットスポットを検知する．これにより，ホットスポットを動的に検知しキャッシュ対象領域とし，I/O 性能の向上を行うことが可能となる．そして，性能評価に提案手法の有効性を示す．

Improvement of the Locality of Reference of in the Lower Cache with Cache Filesystem

KOTARO YOSHIDA[†1]　KENICHI KOURAI[†2]　SANEYASU YAMAGUCHI[†1]

1. はじめに

仮想化環境を用いたサーバ統合やクラウドコンピューティング環境などが普及し，仮想化環境は非常に重要なプラットフォームの一つになっている．コンテナ型でないハイパーバイザ型やホスト型の仮想化環境においては，ホスト OS に加えてゲスト OS が稼働する．そして，それら両 OS にストレージアクセス用のキャッシュ(Linux OS の例においてはページキャッシュ)があり，ゲスト OS 上のアプリケーションはこれらの両キャッシュを介して物理ストレージにアクセスをすることとなる．よって，仮想化環境における I/O 性能の向上には，両キャッシュのヒット率が重要となる．また，仮想化環境などにおいてデータベース処理やビッグデータ解析などが行われることも多く，I/O 性能が重要となる状況は多く存在すると予想される．両 OS のキャッシュサイズは，仮想マシンと物理マシンに割り当てられたメモリ量に依って決まる．これらメモリ割当量をバルーニング[1] などにより動的に変更可能である状況においては，この割当量の調整に依る性能向上の取り組み[2][3]も重要になるが，これらの割当量を変更できない状況においては，所与のメモリ割当量における I/O 性能の向上が重要となる．本稿では，後者の変更できない環境に着目し，考察を行う．後者の具体的な事例としては，パブリッククラウドコンピューティング環境などが考えられる．

ゲスト OS とホスト OS などの二重のキャッシュが動作する環境においては，両 OS に同一のデータが格納されてしまいキャッシュ領域が有効に活用されない問題[4][5][6]や，下位キャッシュにおけるデータアクセスに負の参照の局所性が存在し下位キャッシュのヒット率が著しく低下する課題[7][8][9][10]などが指摘されている．これらの課題に対して我々は，多くの I/O アクセスが発生するホットスポットが既知であるとの仮定を置き，下位キャッシュのアクセスの局所性を高めて下位キャッシュのヒット率と I/O 性能を向上させる手法[11][12]を提案した．本稿では，ホットスポットが既知であるとの前提を置かず，システム稼働中に動的にホットスポット領域の検出を行い，これにより下位キャッシュであるホスト OS キャッシュのヒット率と I/O 性能を向上させる手法を提案する．

2. 関連研究

2.1 参照の時間的局所性

多くのアプリケーションからのデータへの参照(アクセス要求)には，時間的および空間的な局所性があると言われている．参照の時間的な局所性[13][14]は「最近参照されたデータは近い将来に再度参照される可能性が高い」という性質であり，次節で述べる代表的なキャッシュ置換アルゴリズムである LRU(Least Recently Used)をはじめ，多くのキャッシュ置換アルゴリズムがこの性質を仮定し，活用している．

2.2 キャッシュ置換アルゴリズム

本節にて，キャッシュ置換アルゴリズムについて述べる．これらは文献[10]にて詳解されており，本稿では同文献

†1 工学院大学
Kogakuin University
†2 九州工業大学
Kyushu Institute of Technology

の説明に基づいて説明を行う．

LRU[13]は，参照の時間的局所性を期待したキャッシュ置換アルゴリズムであり，OS や CPU などで最もよく使われているアルゴリズムの一つである．最後にアクセスされてからの時間が最も短いデータが近い将来に再度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度アクセスされる確率が最も低いと仮定しており，最後のアクセスからの時間が最も長いデータを破棄する．多くの場合，アプリケーションからのアクセスには参照の時間的局所性が存在し，その様なアプリケーションに対して高い性能を示すことが多い．

LFU (Least Frequently Used)は，アクセス頻度に基づきキャッシュ置換を行うアルゴリズムである．同手法では有限のアクセス履歴を保持し，履歴内におけるアクセス頻度が最も少ないデータをキャッシュから破棄する．

MRU (Most Recently Used)は，最後のアクセスからの時間が最も短いデータをキャッシュから破棄する手法である[15][16]. 新規にキャッシュに格納されたデータがその次のアクセスで破棄され，これ以外のデータはキャッシュに格納され続けることとなり，実質的に固定的なデータを保持している状況となる．後述の負の参照の時間的局所性が生じる二重キャッシュ環境などにおいては LRU よりも高い性能を示すことがある[10][17].

2.3 二重キャッシュ環境におけるキャッシュ管理

本節にて，二重キャッシュ環境におけるキャッシュ管理について述べる．

2Q は，Johnson らによって提案されたキャッシュ置換アルゴリズムである[18]．このアルゴリズムは A1in と Am の 2 つの格納データ用リストと，A1out の破棄データ用リストを使用する．A1in および Am はそれぞれ FIFO(First In, First Out)および LRU の原理に基づいて管理される．最初にアクセスされたデータは A1in に格納され，アクセス頻度の高いデータは Am に格納される．A1in のサイズが閾値より小さい場合，Am 内のデータは置換時に削除される．サイズが閾値より大きい場合，A1in の最も古いデータが破棄され，そのデータは A1out に格納される．2Q では多重キャッシュ環境は考慮されていないが，MQ などの多重キャッシュ環境用のアルゴリズムに応用されている．

MQ は，ネットワークストレージキャッシュなどの多重キャッシュ環境のためのアルゴリズムである[19]．このアルゴリズムは，一度アクセスされたデータは近い将来に再びアクセスされることはないと想定しており，長い時間が経過したあとに再びアクセスされると想定している．よって，MQ はデータを長期間格納することによりキャッシュヒット率を向上させることを目指している．MQ は 2Q に基づいており，複数の LRU リストを管理しアクセス頻度に基づいて格納するリストを決定する．置換時には最後の LRU リストのデータが破棄される．破棄されたデータのデータ ID とアクセス頻度は Qout リストに保存される．破棄されたデータが再びアクセスされた場合，Qout からアクセスされたデータのアクセス頻度に基づいて格納する LRU リストが決定される．LRU リストには有効期限があり，この有効期限を過ぎてもアクセスされなかったデータは，次にアクセス頻度の低い LRU リストに移動される．リスト内のデータにアクセスされた場合，そのデータのアクセス頻度が更新され，格納される LRU リストも更新される．

Exclusive cache は上位キャッシュと下位キャッシュのデータが重複しないように格納するアルゴリズムである[20]. このアルゴリズムは，上位キャッシュと下位キャッシュの両キャッシュを制御することでデータの重複が発生しないようにする．

Software-defined キャッシュ[21][22][23]に関する研究では，システム上に分散して存在する複数のキャッシュに対してシステム全体に渡るグローバルな可視性と制御の実現による性能向上が取り組まれている．分散されたキャッシュを有効活用することが可能になると期待できるが，キャッシュ間の連携などの制御が必要となる．

2.4 負の参照の時間的局所性

負の参照の時間的局所性は，「最近参照されたデータは近い将来に再度参照される可能性が低い」という，通常の参照の時間的局所性と逆向きの性質である[7]. ネットワークストレージ環境[8][9]や仮想化環境[17]などの二重キャッシュ環境に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に，下位キャッシュへの参照などに現れる性質である．多くのキャッシュで使用されている LRU は，通常の参照の時間的局所性を期待しており，負の参照の時間的局所性をもつアクセスに対してはキャッシュヒット率が著しく低下することとなる[17].

負の参照の時間的局所性は，以下の様に生じる．図 1 のように上位キャッシュの置換アルゴリズムとして LRU が動作している状況では，上位キャッシュは最近参照されたデータを保持する．よって，アプリケーションから発行された I/O 要求のなかで最近参照されたデータへの要求は，上位キャッシュにおいてキャッシュヒットとなり上位キャッシュにより処理され，下位キャッシュには転送されない．一方，最近参照されていないデータへの要求は，上位キャッシュにおいてキャッシュミスとなり，下位キャッシュに転送される．よって，下位キャッシュには最近参照されたデータへのアクセス要求が届かず，最近参照されていないデータへのアクセス要求が届くこととなる．

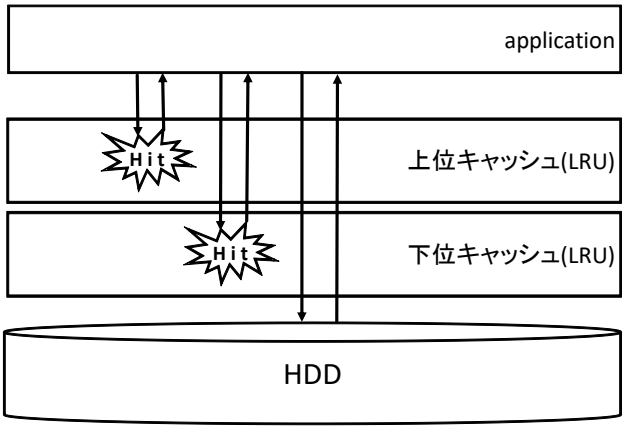

図 1　二重キャッシュ環境

2.5 Cache FS

文献[11][12]にて，二重のキャッシュが動作する仮想化環境に着目して上位キャッシュの置換の制御による下位キャッシュの参照の局所性を高め仮想化環境における I/O 性能を向上させる手法およびそれを実現するファイルシステムが提案されている．

文献[11]においては，優先的に上位キャッシュに格納するデータを定め，それ以外のデータを積極的に上位キャッシュから破棄することにより下位キャッシュにおける参照の局所性を強め，下位キャッシュのヒット率を向上させて I/O 性能を向上させる手法が提案されている．具体的には，fadvise 機能を用いて優先格納以外のデータを積極的にゲスト OS のページキャッシュから破棄し優先データをゲスト OS ページキャッシュ内に格納し続けることにより，優先データへの参照要求がホスト OS ページキャッシュに届きづらくし，ホスト OS ページキャッシュにおける参照の局所性を強めている．ホットスポットは既知であると仮定し，優先データはユーザが指定する仕組みとなっている．fadvise 機能を用いてゲスト OS ページキャッシュを制御するプロセスは，ユーザ空間で動作している．当該手法は，キャッシュが I/O 性能に大きな影響を与えるキャッシュヒット率が高い状況においては大幅な I/O 性能の向上を達成できることが確認されている．

文献[12]においては，ホットスポットが既知である前提のもとゲスト OS のキャッシュに格納するデータを固定してホスト OS ページキャッシュにおける参照の局所性を強める手法が提案されている．当該手法は，キャッシュ機能を持つファイルシステム Cache FS を実装することにより達成されており，ファイルシステムの他にユーザ空間プロセスなどの起動は要さない．当該ファイルシステムは起動時に指定されたホットスポットデータをキャッシュに格納し，キャッシュの格納データの置換は行わずに固定的なデータを保持し続ける．本手法も，大幅な I/O 性能の向上を実現できることが示されている．

3. LFU Cache FS

3.1 手法

本章にて，Cache FS を LFU を用いて拡張する手法を提案する．Cache FS は，ホットスポットが既知であると仮定し，格納データの置換を行わない．本稿では同ファイルシステムを LFU を用いて格納データを管理, 置換する様に拡張する．これにより，ホットスポットが既知であることを前提とせず，キャッシュ格納対象を動的に検出できる様にする．本稿では，拡張したファイルシステムを LFU Cache FS と呼び，従来の Cache FS を便宜上 Const. Cache FS と呼ぶ．

LFU Cache FS は，アクセス要求の履歴を保持し，LFU に基づきキャッシュ内に格納するデータと格納しないデータを決定する．すなわち，各データブロックの履歴内におけるアクセス頻度を計算し，アクセス回数が上位のデータブロックをキャッシュ格納の対象とし，下位のデータブロックを対象外とする．

具体的には，キャッシュ内に格納されているデータブロックの中で最も参照回数が少ないデータブロックを検出し，同様に格納されていないデータブロックの中で最も参照回数が多いデータブロックを検出する．検出された 2 つのデータブロックの参照回数を比較し，キャッシュに格納されているデータブロックより格納されていないデータブロックの参照回数の方が多い場合は，キャッシュに格納されているデータブロックを破棄し，格納されていないデータブロックをキャッシュに格納する．これにより，ホットスポットを動的に検出し，キャッシュにホットスポットであるデータブロックを格納する．ブロックサイズ，履歴確認(およびそれに伴うデータブロック置換)の契機，履歴長はチューニングパラメタとする．

3.2 実装

本節にて，LFU Cache FS の実装について述べる．

LFU Cache FS および Const. Cache FS は，図 2 の様な構造をしている．すなわち，本ファイルシステムはあるサイズのファイル(図内の下位ファイル)を与えられ，同サイズのファイル(図内の上位ファイル)を提供する．同ファイルシステムは下位ファイルへのアクセスをラップし，アプリケーションは下位ファイルへのアクセスは本ファイルシステムが提供する上位ファイルと本ファイルシステムを介して行う．本ファイルシステムは起動時にメモリを与えられ，対象ファイルの一部のデータはメモリ内に保持してキャッシュとする．同ファイルシステムは下位ファイルへのアクセスのすべてを仲介し，アクセス対象のデータがキャッシュ内にある場合はキャッシュヒットとなり，キャッシュ内のデータを用いて同ファイルシステムが要求に返答する．キャッシュ内にない場合はキャッシュミスとなり，単純にアクセス要求を下位ファイルに転送し，下位ファイルから

得られた応答を上位ファイルへのアクセス要求に転送する．本研究では，LFU Cache FS および Const. Cache FS を FUSE[24]を用いて実装した．

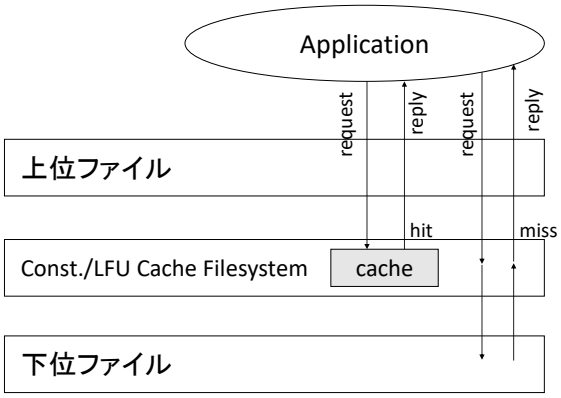

図 2　Cache FS 構造

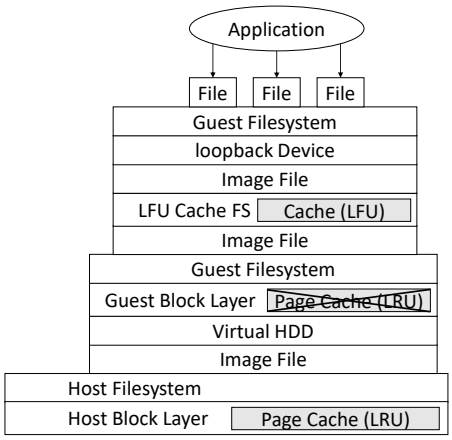

図 3　システム構成

LFU Cache FS を含む仮想化環境は，図 3 に示す構造をしている．LFU Cache FS はゲスト OS 上で動作し，下位ファイルとしてゲスト OS よりディスクイメージファイルを与えられ，上位ファイルとしてディスクイメージを提供する．そして，その上位イメージファイルをループバックデバイスとしてマウントし，その上に通常のファイルシステム(Ext2 など)を構築してアプリケーションが使用する．

LFU Cache FS を使用する場合は，ストレージアクセス用のキャッシュのメモリは，ゲスト OS ブロック層のページキャッシュでなく, LFU Cache FS が管理することとなる．よって，空きメモリはすべて LFU Cache FS に与えることとなり，図内のゲスト OS ブロック層のページキャッシュは実質的に機能しない(サイズがほぼゼロで, ほぼ全てのアクセスがキャッシュミスとなる)状態となる．このメモリはページキャッシュで管理することが好ましいと考えられ，Const./LFU Cache FS は提案するキャッシュ管理手法(すなわち上位キャッシュのデータ置換の制御による下位キャッシュの参照の時間的局所性の向上)の試作的実装の位置づけとなる．ページキャッシュ実装が管理しないことの影響などについては，5 章で考察する．

4. 評価実験

4.1 測定環境

本章では LFU Cache FS の性能評価を行う．ゲスト OS 上でファイルシステムベンチマークである FFSB(Flexible File System Benchmark)[25]を実行し，I/O 性能とホストキャッシュヒット率を評価した．

図 4 に測定環境を示す．測定環境は，ゲスト OS とホスト OS を備えた仮想マシンと物理マシンにより構成されている．どちらの OS もブロックレイヤーにページキャッシュを持っている．アクセス要求はゲスト OS 上で動作するアプリケーション(FFSB)から発行される．アクセス対象のベンチファイルのサイズは 4GB であり，FFSB はランダムに選択されたファイルにランダムに選択された位置の 1MB の読み取り要求を繰り返し送信する．ランダム選択の分布は一様分布である．

HDD へのアクセスは，以下のように処理される．アプリケーションから I/O アクセスが発行され，ゲスト OS キャッシュ(ページキャッシュまたは LFU Cache FS)に転送される．アクセス要求がゲスト OS キャッシュでヒットした場合，要求されたデータはゲスト OS キャッシュにより提供される．アクセス要求がゲスト OS キャッシュでミスした場合，その要求はホスト OS に転送される．転送された要求がホスト OS キャッシュでヒットした場合，要求されたデータはホスト OS キャッシュにより提供される．転送された要求がホスト OS キャッシュでミスした場合，その要求は物理 HDD へ転送され HDD により提供される．図内の(A)はホスト OS キャッシュに送信されるすべての要求を示す．(B)はホスト OS キャッシュでキャッシュミスし，HDD へ転送される要求を示す．ホスト OS キャッシュのミス率は，(B)のデータサイズを(A)で割ることにより近似的に求めることができる．我々は，(A)の量を記録できる様に Linux カーネルのメモリ管理実装(mm)を，(B)の量を記録できる様に同カーネルの SCSI サブシステムを改変し，カーネルモニタリング手法[26][27][28]を用いて(A)と(B)の量を観察し，近似のホスト OS キャッシュのヒット率とミス率を測定した．

物理マシンおよび仮想マシンの仕様は表 1 と表 2 の通りである．LFU Cache FS には 2.625GB のメモリを割り当てた．履歴確認およびそれに伴うデータブロック置換の契機は，キャッシュミスが 10 回生じるごととした．また，キャッシュ置換の単位のサイズ(データブロックサイズ)は 128MB とした．履歴サイズはアクセス 100 回分である．

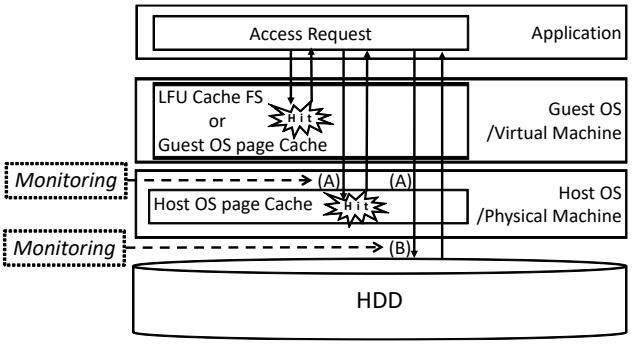

図 4　測定環境

表 1　物理マシン

OS	CentOS 6.5
Kernel	Linux 2.6.32.27
HDD	1TB
CPU	Intel Core i3
Memory	6GB
Virtualization system	KVM
Filesystem	Ext2

表 2　仮想マシン

OS	CentOS 6.7
HDD	50GB
CPU	QEMU Virtual CPU
Memory	3GB
Filesystem	Ext2

4.2 測定結果

測定結果を図 5，図 6 に示す．図 5 はホスト OS キャッシュヒット率，図 6 はスループット(1 秒あたりに読み込まれたデータサイズ)である．図 5 より，通常手法(0%)に対して LFU Cache FS のホスト OS キャッシュヒット率(92%)が大きく向上していることが分かる．これは，ゲスト OS 上で LFU Cache FS を使用することによりホスト OS キャッシュへ送られるアクセス要求の参照の時間的局所性が強まったからである．図 6 より，通常手法に対して LFU Cache FS のスループットが向上していることが分かる．これは，ホスト OS キャッシュヒット率が向上したからである．

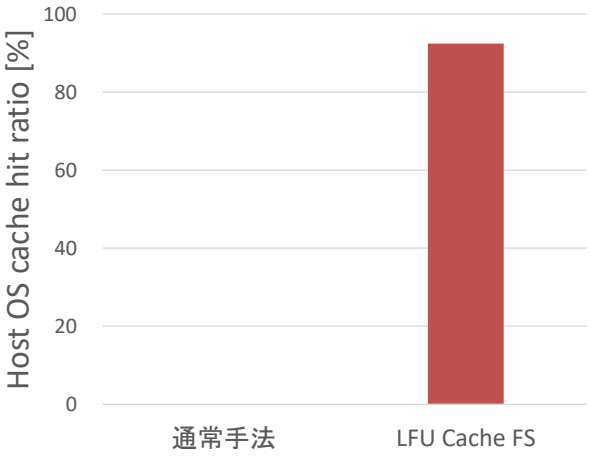

図 5　ホスト OS キャッシュヒット率

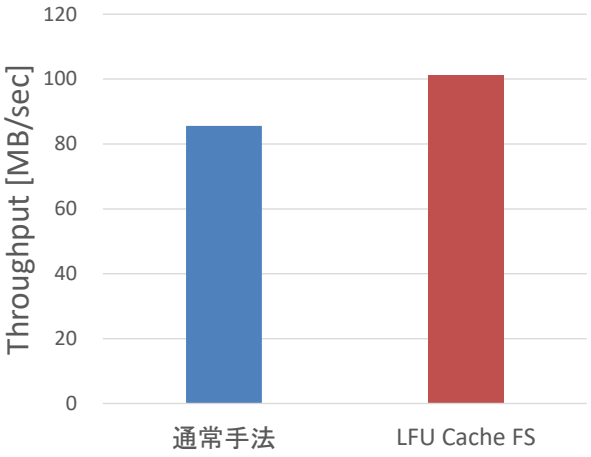

図 6　スループット

5. 考察

まず，上位キャッシュの置換アルゴリズムと下位キャッシュにおける参照の時間的局所性の関係について考察する．本稿では，二重キャッシュ環境の上位キャッシュのデータ置換を制御し，下位キャッシュにおける参照の時間的局所性を向上させる手法を提案している．2 章で述べた様に，上位キャッシュで LRU が動作することが原因で下位キャッシュにおいて負の参照の時間的局所性が生じる．本稿の様に上位キャッシュで動作するアルゴリズムを LFU とすると，上位キャッシュの格納対象の変化が LRU 使用時と比較して緩やかとなり，下位キャッシュにて参照の対象となるデータの種類の変化も緩やかになり，結果として参照の時間的局所性が強まると予想できる．

次に，アプリケーションからのデータアクセス要求に時間的局所性があった場合についての考察を行う．本稿ではファイルシステムベンチマーク FFSB を用いて性能を評価しており，全てのファイルのアクセス確率が同等である，

すなわち，アプリケーションが発行する参照要求には時間的局所性がなく，上位キャッシュにより生じる負の時間的局所性の影響についてのみ考察した．もし，アプリケーションからのデータアクセス要求に参照の局所性があれば，(ランダム選択の FFSB 実行時と比較して)より参照確率の高いデータを LFU アルゴリズムが保持することができると予想できる．これにより，アクセス頻度が高いデータに対するアクセスが下位キャッシュに届く確率がより低くなる．よって，下位キャッシュにおける参照の局所性がより増加し，提案手法はより効果的に機能すると予想できる．

最後に，上位キャッシュ置換の制御の実装方法についての考察を行う．本稿では，ゲスト OS におけるストレージアクセス用キャッシュ(通常はページキャッシュ)の置換アルゴリズムの修正を，ゲスト OS 用のファイルシステムを実装することにより実現した．上位キャッシュの置換の制御をページキャッシュ内に実装することよりさらなる性能の向上が実現できると期待できる．これをページキャッシュ内に実装すると，図 3 におけるゲスト OS の 2 個のイメージファイルの処理のオーバーヘッド，ループバックデバイスの処理のオーバーヘッド，FUSE 処理のオーバーヘッドなどを削減可能となる．

また，通常のページキャッシュはアプリケーションが必要なメモリを確保したあとの余ったメモリを活用しており，アプリケーションによるメモリ使用の要求が生じるとページキャッシュに使用されているメモリからそれに対して割り当てることが可能となっている．それに対して本稿の実装では，ファイルシステム起動時にファイルシステムプロセスにメモリを割り当ててしまい，この割り当てられたメモリを他のアプリケーションなどが使用することができない仕組みとなっている．この課題も，上位キャッシュの置換の制御の機能をゲスト OS のページキャッシュ内に実装することにより解決できると考えられる．

6. おわりに

本稿では，二重キャッシュ環境である仮想化環境に着目し，下位キャッシュであるホスト OS ページキャッシュにおける参照の時間的局所性を向上させるファイルシステムを提案した．そして，性能評価によりその有効性を示した．

今後は，様々な置換アルゴリズムによる性能評価，様々なデータサイズにおける性能の評価，Const. Cache FS との性能の比較，チューニングパラメータの最適化に関する考察などを行っていく予定である．

謝辞

本研究は JSPS 科研費 15H02696, 17K00109, 18K11277 の助成を受けたものである．

本研究は、JST、CREST JPMJCR1503 の支援を受けたものである．

参考文献

[1] Weiming Zhao and Zhenlin Wang. 2009. Dynamic memory balancing for virtual machines. In Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments (VEE '09). ACM, New York, NY, USA, 21-30.

[2] M. Sakamoto and S. Yamaguchi, "Dynamic Memory Allocation in Virtual Machines Based on Cache Hit Ratio*," 2015 Third International Symposium on Computing and Networking (CANDAR)*, Sapporo, 2015, pp. 613-615. doi: 10.1109/CANDAR.2015.34

[3] Saneyasu Yamaguchi and Eita Fujishima. 2016. Optimized VM memory allocation based on monitored cache hit ratio. In Proceedings of the 4th Workshop on Distributed Cloud Computing (DCC '16). ACM, New York, NY, USA, Article 8, 6 pages. DOI: https://doi.org/10.1145/2955193.2955200

[4] Pin Lu and Kai Shen, “Virtual machine memory access tracing with hypervisor exclusive cache,” In 2007 USENIX Annual Technical Conference on Proceedings of the USENIX Annual Technical Conference (ATC'07), Jeff Chase and Srinivasan Seshan (Eds.). USENIX Association, Berkeley, CA, USA, Article 3, 15 pages, 2007.

[5] Prateek Sharma and Purushottam Kulkarni, “Singleton: system-wide page deduplication in virtual environments,” In Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing (HPDC '12). ACM, New York, NY, USA, 15-26, 2012, DOI=http://dx.doi.org/10.1145/2287076.2287081

[6] Willick, D.L., Eager, D. L., and Bunt, R.B., “Disk Cache Replacement Policies for Network Fileservers,” 1993 Proceedings of IEEE International Conference on Distributed Computing Systems (ICDCS '93) (May. 1993).1

[7] Y. Nagasako and S. Yamaguchi, "A Server Cache Size Aware Cache Replacement Algorithm for Block Level Network Storage," 2011 Tenth International Symposium on Autonomous Decentralized Systems, Tokyo & Hiroshima, 2011s, pp. 573-576. doi: 10.1109/ISADS.2011.80

[8] 宮野 晋平，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,” IPSJ SIG Notes 2009(20(2009-DPS-138)), 7-12, 2009-02-26, Information Processing Society of Japan (IPSJ), 2009.

[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), 2012.

[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", 情報処理学会論文誌コンシューマ・デバイス＆システム（CDS）,5(4),42-51 (2015-10-03) , 2186-5728 , 2015.

[11] Yoshida Kotaro and Saneyasu Yamaguchi. 2017. Advising cache for lower cache. In Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication (IMCOM '17). ACM, New York, NY, USA, Article 68, 6 pages, 2017. DOI: https://doi.org/10.1145/3022227.3022294

[12] Yoshida Kotaro, Saneyasu Yamaguchi , "A Caching Filesystem for Increasing Locality in the Second Cache in a Virtualized Environment", 12th International Conference on Ubiquitous Information Management and Communication (ACM IMCOM 2018), 2018.

[13] P. J. Denning, "The Locality Principle," Communications of the ACM, Volume 48, Issue 7, pp. 19-24, (Jul. 2005).

[14] Peter J. Denning (2006) The Locality Principle. Communication Networks and Computer Systems: pp. 43-67. https://doi.org/10.1142/9781860948947_0004

[15] Denning, P.J. “The Working Set Model for Program Behavior,” 1968 Communications of the ACM (May. 1968).
[16] Coffman, E.G. Jr., and Denning, P.J. “Operating Systems Theory, Prentice Hall Professional Technical Reference.” (Oct. 1973).
[17] Hiroki Sugimoto, Kenichi Kourai, and Saneyasu Yamaguchi. 2015. Host OS page cache hit ratio improvement based on guest OS page drop. In Proceedings of the 17th International Conference on Information Integration and Web-based Applications & Services (iiWAS '15). ACM, New York, NY, USA, , Article 77 , 4 pages. DOI: http://dx.doi.org/10.1145/2837185.2837267
[18] Theodore, J., and Dennis, S., “2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm,” 1994 Proceedings of the 20th International Conference on Very Large Data Bases, (Sep. 1994).
[19] Zhou, Yuanyuan, Zhifeng Chen, and Kai Li. "Second-level buffer cache management." IEEE Transactions on parallel and distributed systems 15.6 (2004): 505-519.
[20] Willick, D.L., Eager, D. L., and Bunt, R.B., “Disk Cache Replacement Policies for Network Fileservers,” 1993 Proceedings of IEEE International Conference on Distributed Computing Systems (ICDCS '93) (May. 1993).
[21] Ioan Stefanovici, Eno Thereska, Greg O'Shea, Bianca Schroeder, Hitesh Ballani, Thomas Karagiannis, Antony Rowstron, and Tom Talpey. 2015. Software-defined caching: managing caches in multi-tenant data centers. In Proceedings of the Sixth ACM Symposium on Cloud Computing (SoCC '15). ACM, New York, NY, USA, 174-181. DOI: http://dx.doi.org/10.1145/2806777.2806933
[22] Stephen T. Jones, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2006. Geiger: monitoring the buffer cache in a virtual machine environment. SIGOPS Oper. Syst. Rev. 40, 5 (October 2006), 14-24. DOI: https://doi.org/10.1145/1168917.1168861
[23] N. Beckmann and D. Sanchez, "Jigsaw: Scalable software-defined caches," Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques, Edinburgh, 2013, pp. 213-224. doi: 10.1109/PACT.2013.6618818
[24] Szeredi, M, "FUSE: Filesystem in Userspace," http://fuse.sourceforge.net/
[25] Flexible File System Benchmark https://sourceforge.net/projects/ffsb/
[26] Saneyasu Yamaguchi, Masato Oguchi, Masaru Kitsuregawa, "iSCSI analysis system and performance improvement of sequential access in a long-latency environment," Electronics and Communications in Japan (Part III: Fundamental Electronic Science), Volume 89, Issue 4, Pages 55-69, Wiley Subscription Services, Inc., A Wiley Company, April 2006. DOI: https://doi.org/10.1002/ecjc.20238
[27] Yuta Nakamura, Kyosuke Nagata, Shun Nomura, and Saneyasu Yamaguchi. 2014. I/O scheduling in Android devices with flash storage. In Proceedings of the 8th International Conference on Ubiquitous Information Management and Communication (ICUIMC '14). ACM, New York, NY, USA, , Article 83 , 7 pages. DOI: https://doi.org/10.1145/2557977.2558025
[28] K. Miki, S. Yamaguchi, M. Oguchi, "Kernel Monitor of Transport Layer Developed for Android Working on Mobile Phone Terminals", Proceedings of The Tenth International Conference on Networks (ICN), pp. 297-302, 2011.