2007－OS－104 (14)
2007／1／31

ファイルの特徴に基づくディスクキャッシュのヒット率の見積もりを用いた置換アルゴリズム

林 佳寛† 鈴来 和久† 横田 裕介†† 大久保 英嗣††
†立命館大学大学院理工学研究科 ††立命館大学情報理工学部

本稿では, ファイルの特徴に基づくディスクキャッシュの置換アルゴリズムを提案する. 現在, ディスクキャッシュの置換アルゴリズムは, 時間的局所性に基づくものが主流である. また, ディスクキャッシュの参照履歴を用いたアルゴリズムも開発されている. これらのアルゴリズムでは, ディスクキャッシュの参照履歴から, ディスクキャッシュ上に複製しているファイルの特徴を再構成してディスクキャッシュの置換えに用いている. 本稿で提案するアルゴリズムでは, ディスクキャッシュの参照履歴からファイルの特徴を再構成するのではなく, オペレーティングシステムが管理しているファイルに関する情報を用いることにより, 各々のファイルが持つ特徴に応じたディスクキャッシュの置換えを実現している.

A replacement algorithm using the hit rates estimation of the disk cache based on the properties of files

Yoshihiro Hayashi† Kazuhisa Suzuki† Yusuke Yokota†† Eiji Okubo††
†Graduate School of Science and Engineering, Ritsumeikan University
††College of Information Science and Engineering, Ritsumeikan University

In this paper, we propose a replacement algorithm for the disk cache that is based on the properties of files. Most replacement algorithms that are widely used today for the disk cache are based on the temporal locality of references. Some replacement algorithms utilize not only the temporal locality of references but also the history of references to the disk cache. These algorithms extract the properties of files from history of references to the disk cache. Proposed algorithm in this paper uses not only temporal locality of references but also the properties of files. Proposed algorithm does not extract property of the file from history of references to the disk cache, but uses information in the file system as the properties of files. Namely, in the proposed algorithm, the properties of each file are extracted and the pages on the disk cache are replaced based on those extracted properties.

1 はじめに

近年, 計算機は, 低価格化が進み, 家庭, 企業, 学校などでさまざまな用途に用いられている. また, 計算機で扱うデータの量と種類も増加し, 外部記憶装置を備えた計算機が一般的になっている.

外部記憶装置は, 計算機で用いる CPU や主記憶に比べて低速であるため, CPU は, 処理に必要なデータの外部記憶装置から主記憶への読出しを待つ必要がある. この問題を解決するために, オペレーティングシステムは, ディスクキャッシュと呼ぶ仕組みを備えている.

ディスクキャッシュは, 読み出す頻度の高い外部記憶装置上のデータを主記憶上に複製する. 主記憶装置上に複製が存在するデータへの読出しが再び行われた場合, 外部記憶装置上からデータを読み出すのではなく, 主記憶上の複製を読み出す. このような処理によって外部記憶装置から読出しを行う回数を減らすことができる. ディスクキャッシュは, 主

記憶装置の一部を用いて実現するため容量が有限である．このため，主記憶装置に空きがなくなった場合，不要なデータを選び出さなければならない．このためのアルゴリズムは，置換アルゴリズムと呼ばれる．

現在，広く用いられている置換アルゴリズムの多くは，時間的局所性のみを基準として置換えを行う．しかし，ディスクキャッシュの置換えにおいては，置き換えるページはファイルの複製である．OS は，ファイルを管理するための情報を持っている．そこで，本研究ではこの情報を使用し，ディスクキャッシュの置換えを時間的局所性だけでなく ファイルの特徴を考慮した置換えを行う．

以下，本稿では，2 章で既存の置換アルゴリズムとその特徴についてまとめ，3 章でファイルの特徴に基づくディスクキャッシュのヒット率の見積もりを用いた置換アルゴリズムについて述べる．4 章でシミュレータとそれを用いたシミュレーションの結果について述べ，5 章で提案手法を Linux カーネルで用いるための考察を行う．

2 関連研究

既存の置換えアルゴリズムは，時間的局所性のみを基準に置換えを行うため，効率良く使用頻度の高いデータを選ぶことが不可能な場合がある．良く知られた問題として，既存の置換えアルゴリズムで多く用いられている LRU は，大きいファイルをシーケンシャルに繰り返し読み出すと，キャッシュのヒット率が低下するという問題がある．この問題を解決するために，問題の発生を検出し，一時的にアルゴリズムを変更するという手法を用いるアルゴリズムが開発されてきた．このような手法を用いるアルゴリズムとして SEQ [1] と EELRU (Early Eviction LRU) [2] がある．SEQ は，通常，LRU と同様の動作をする．しかし，外部記憶装置から読み出すデータの領域が長く連続している場合，アクセスがこのままシーケンシャルに続くと判断する．この場合，読み出してキャッシュしたデータは，連続の領域が終るまで読み出されないと考えられる．そこで，SEQ は，動作を LRU から MRU へ変更する．MRU は，最近使われたデータを選ぶアルゴリズムである．したがって，長い間使われないと予測されるシーケンシャルに読み出したデータを置換え対象とすることで，他の部分を残すことが可能である．しかし，シーケンシャルに読み出されているファイル以外のファイルへの読出しが同時に起こると，シーケンシャルに読み出されているファイル以外のデータに対しても MRU が適応されてしまう．

EELRU は，通常は LRU と同様の動作をする．しかし，外部記憶装置から読み出してキャッシュしたばかりのデータが LRU において置換えの対象になっているとき，最近使われていないデータではなく，一定の確率で最近使ったデータを置換え対象にする．シーケンシャルに読み出されているファイル以外のファイルへの読出しが同時に起こると，最近使ったデータを置換え対象とするが，データがどのファイルのものなのかは意識しない．

すなわち，ディスクキャッシュが扱っているメモリ上のデータ全体を 1 つの置換アルゴリズムで処理することにより，特徴をもったアクセスが行われているファイル以外のデータもアルゴリズムの変更の影響を受けてしまう．このような状況を避けるため，ファイルごとに置換えのアルゴリズムを分ける必要があると考えられる．

このような問題を解決するアルゴリズムとして，UBM (Unified Buffer Management) [3] が挙げられる．UBM は，ページの参照パターンの履歴から，参照パターンを Looping references, Sequential references, Other references の 3 つに分類する．優先的に Sequential references に属するページを置換え対象とする．Sequential references に属するページが存在しない場合は，Looping references と Other references のどちらに属するページを置換え対象とする方が，ディスクキャッシュのヒット率の減少が小さいかを求め，小さい方から置き換えるページを選び出す．Looping references から置き換えるページを選ぶ際には，ループの周期が長いものを置換え対象とする．Other references に属するページは LRU で管理されている．

これらのアルゴリズムは，ページの参照パターンから，ファイルが持っていた特徴を再構成する手法を採用している．SEQ と EELRU は，ディスクキャッシュのサイズと比較して大きなファイルに対して読出しが起きていることをディスクキャッシュのヒット率の低下から検知し，対策を行うアルゴリズムである．また，UBM も，参照履歴から参照のパターンをループ参照，シーケンシャル参照，その他の 3 つに分け，各々に異なるアルゴリズムの適用を行っている．このようなことから，以上に挙げた置換えアルゴリズムは，ページ参照の履歴から，ファイルの

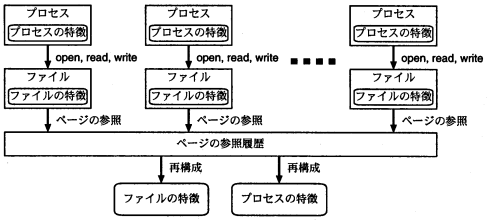

図 1　ファイルとプロセスの特徴の再構成

読出しがシーケンシャルであるかを判断し, それを置換の判断に利用しているといえる.

3　適応的ディスクキャッシュ管理手法

ページの参照履歴は, プロセスがファイルを扱った結果である. 先に挙げた手法では, ページの参照履歴からファイルの特徴とファイルを扱うプロセスの特徴を再構成している. この様子を図 1 に示す. 複数のプロセスが複数のファイルを扱った結果として, 1 つのページの参照履歴が得られる. したがって, ページの参照履歴には, その参照がどのプロセスがどのファイルを扱った結果であるのかという情報は欠落している. そのため, どのプロセスがどのファイルを読み出すことがディスクキャッシュのヒット率を下げる原因となっているかは, 分からない. 例えば MRU の場合, ページの参照履歴から現在ディスクキャッシュのヒット率が低下しているという情報を得るが, そのヒット率の低下がどのプロセスによってどのファイルを読み出すことによって発生しているのかは分からない.

しかし, ページの参照のパターンといったファイルおよびファイルを扱うプロセスの特徴は, 元々ファイルを単位として存在しており, OS は, これらの情報を管理している. そこで本手法では, ページの参照履歴からファイルの特徴に関する情報を再構成するのではなく, OS が管理しているファイルの特徴を用いて置換えを行う. 例えば, MRU が回避するディスクキャッシュのヒット率が低下する状況は, プロセスがシーケンシャルに大きいファイルを読み出したときに発生する. この状況は, OS が管理しているファイルのサイズと, プロセスがシーケンシャルな読み出しを行っているという情報によって知ることが可能である. このように, OS が管理している情報をそのまま用いることによって, どのプロセスによるどのファイルへの読出しがディスクキャッシュのヒット率を低下させる原因となっているのかを明確にすることが可能である. どのファイルへの読出しが問題なのかが明確になれば, そのファイルの複製を持つディスクキャッシュを処理し, 問題を解決することが可能となる.

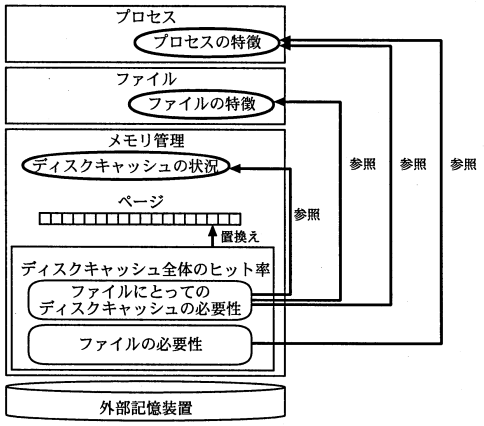

図 2　本手法の概要

このような利点から, 本手法では, ディスクキャッシュ上に複製されたページがどのファイルの複製であるかを明確に対応付け, OS が管理しているファイルの特徴とファイルを読み出すプロセスの特徴を用いた置換えを行う.

3.1　概要

本手法では, OS が管理しているファイルの特徴, ファイルを読み出すプロセスの特徴, 現在のキャッシュの状態を利用した置換えを行う. これらの情報を管理することによって, そのファイルにディスクキャッシュが必要であるか, 不要であるか判断する. しかし, ファイルが持つ特徴からディスクキャッシュが必要だと判断しても, もしそのファイルに対する読出しがなければ, そのディスクキャッシュが読み出されることはない. そこで, 本手法では, ファイルにとってディスクキャッシュが必要であるか否かというディスクキャッシュの必要性と, そのファイルに対して読出しが起こるか否かというファイルの必要性という 2 つの基準を用いて置き換えるべきページを決定する. 本手法の概要を図 2 に示す. ファイルごとのディスクキャッシュの必要性は, 各ファイルのディスクキャッシュの量とそのファイルを読み出す際のディスクキャッシュのヒット率の関

係およびそのファイルのディスクキャッシュの量が減った際のヒット率の低下量という指標で表す．各ファイルのディスクキャッシュの量とそのファイルを読み出す際のディスクキャッシュのヒット率の関係は，ファイルの特徴，ファイルを読み出すプロセスの特徴，現在のキャッシュの状態から求める．これらの要素は，ファイルごとに存在している．また，ファイルの必要性は，OS に対するファイルの読出し要求が，あるファイルに対する要求である可能性で表す．ファイルの読出しには，時間的局所性がある．そのため，過去の一定期間のファイルを単位とした読出し履歴から，そのファイルを参照する可能性を推測することで求める．どのファイルを参照するかという情報もプロセスの特徴となる．

OS が管理しているすべてのファイルに対して，ファイルを読み出す際のディスクキャッシュのヒット率とそのファイルを参照する可能性が分かると，OS に対するファイル読出し要求全体のディスクキャッシュのヒット率が求まる．また，あるファイルの複製をもつディスクキャッシュを置換えたとした場合のファイル読出し要求全体のディスクキャッシュのヒット率も求めることができる．置換アルゴリズムの目的は，使われないディスクキャッシュを置き換えることによって，ディスクキャッシュ全体のヒット率を下げないことである．したがって，どのファイルの複製をもつディスクキャッシュを置き換えると最もディスクキャッシュ全体のヒット率が低下せずに済むかを調べ，最もヒット率が低下しないディスクキャッシュを置換えの対象として選ぶ．

3.2 ファイルごとのディスクキャッシュのヒット率

ファイルにとってディスクキャッシュが必要であるか否かは，ファイル自身が持つ特徴やファイルを読み出すプロセスの特徴によって決まる．例えば，シーケンシャルな読出しが起こるファイルは，ディスクキャッシュを必要としない．ファイルを読み出した後に残ったディスクキャッシュへの参照が起きないことが予想されるためである．しかし，シーケンシャルな読出しが繰り返し起こるファイルには，ディスクキャッシュが必要である．1 度目の読出しによって複製されたディスクキャッシュ上への読出しが繰り返し起こると考えられるためである．このように，ファイルの特徴や読み出すプロセスの特徴によって，ディスクキャッシュの必要性は決まる．

ここで，主記憶装置の空き容量が不足したため，ディスクキャッシュを解放しなければならなくなった場合のことを考える．シーケンシャルな読出しが行われるファイルのディスクキャッシュは，解放してもヒット率の低下は起こらない．一方，シーケンシャルな読出しが繰り返し起こるファイルのディスクキャッシュを解放すると，再び読み出す際にディスクキャッシュが存在しなくなるためヒット率が低下する．すなわち，ディスクキャッシュを必要とするファイルのディスクキャッシュを解放すると，ディスクキャッシュのヒット率が低下するといえる．したがって，ファイルごとのディスクキャッシュの必要性は，ディスクキャッシュのヒット率の低下量という数値で表すことが可能である．

ディスクキャッシュを解放した際のディスクキャッシュのヒット率が低下する量を求めるためには，そのファイルの複製を持つディスクキャッシュの量と，そのファイルを参照した際のヒット率の関係を求めなければならない．この関係を求めるために，本手法では，*Seq*, *FileSize*, *CacheSize* の 3 つのパラメータを用いる．

Seq は，そのファイルへの読出しがシーケンシャルであるかランダムであるかを表す．OS は，シーケンシャルなファイルの先読みのために，このようなシーケンシャルであるか否かという情報を管理している．*FileSize* は，ファイルのサイズであり，ファイルシステムが管理している．*CacheSize* は，そのファイルの中でキャッシュされている量である．この 3 つのパラメータから，ファイルの特徴とファイルを読み出すプロセスの特徴を推定し，それに応じてディスクキャッシュのヒット率を求める．

Seq によって，ファイルの読出しがランダムに行われていると分かっている場合，このファイルへの読出しが起こったときのディスクキャッシュのヒット率 *FileHitRate* は，$FileHitRate = CacheSize/FileSize$ である．これは，ある読出しによって読み出すファイル内の位置がそれ以前の読出しに左右されず完全にランダムであるとした場合の数値である．

Seq によって，ファイルの読出しがランダムに行われていると分かっている場合，このファイルへの読出しが起こったときのディスクキャッシュのヒット率 *FileHitRate* は，$CacheSize/FileSize$ の値によって変化する．$CacheSize/FileSize = 1$ の場合，既にこのファイルへ 1 回以上の読出しがあり，ディ

スクキャッシュ上へすべてのページが複製されてい状況であるといえる．この場合，$FileHitRate = 1$ である．

$CacheSize/FileSize < 1$ の場合，このファイルは以下のうちのいずれかの状況にあると考える．

- 読出しは繰り返されず 1 回のみである．
- 複数回繰り返される読出しの 1 回目である．
- 2 回目以降の読出しであるが，以前の読出しによって複製されたディスクキャッシュが，主記憶装置の空き容量不足によって既に置き換えられた．

いずれの場合も，読み出すページはディスクキャッシュ上に複製されていないため $FileHitRate_i = 0$ である．

3.3 ファイルを読み出す可能性

ファイルの必要性は，今後，そのファイルに対する読出しがあるか否かで決まる．プロセスは，ファイルのある領域を読んだ後，再びそのファイルを参照する可能性が高いと考えられる．たとえば，シーケンシャルな読出しを行うファイルは，同じファイルへ連続したアクセスが発生する．このことから，ファイルへの読出し要求には時間的局所性があると考えられる．したがって，ファイルを読み出す可能性は，過去の履歴から推測する．

通常の LRU であれば，ページごとに参照の履歴を取得するが，本手法ではあるファイルを参照する可能性を計算するために，ファイルごとに読出しの履歴を取得する．このため，他の置換アルゴリズムと異なり，ファイルごとの参照履歴を取得する必要がある．ファイルの読出しには，どのファイルを，どこから，どれだけの量を読み出すのかという情報が必要となる．この情報のうち，どのファイルを読み出すのかという情報の履歴を取り，過去の一定期間におけるファイル読出しの履歴から，各々のファイルを参照する可能性を求める．

3.4 ディスクキャッシュ全体のヒット率

ファイルごとのディスクキャッシュのヒット率と，ファイルを参照する確率から，ファイルシステム全体における，ディスクキャッシュのヒット率を見積もることが可能である．OS が管理するファイルが n 個存在しているとする．そのファイルに対して，順番に 1 から n までの番号を割り当てる．OS に対する読出し要求は，どのファイルを読み出すかという指定とともに与えられる．それが i 個目のファイルへの読出しである可能性を $FileReferRate_i$(以降，FRR_i と表記) とする．また，あるファイル i を読み出した際にキャッシュがヒットする確率を $FileHitRate_i$ (以降，FHR_i と表記) とする．このとき，プロセスが OS に対してファイルの読出し要求を行った際に，ディスクキャッシュがヒットする確率 $GlobalHitRate$ (以降，GHR と表記) は，次のようになる．

$$GHR = \sum_{i=1}^{n} FRR_i \cdot FHR_i$$

3.5 置き換えるべきページの決定

既にメモリ上に存在するディスクキャッシュから j 個目のファイルのディスクキャッシュを置換え対象にしたと仮定した場合の，全体のディスクキャッシュのヒット率を $AssumptiveGlobalHitRate_j$ (以降，$AGHR_j$ と表記) とする．このとき，j 個目のファイルをキャッシュしているページを置換え対象とした場合のディスクキャッシュf のヒット率の変化量 $Diff_j$ は，次のように表すことができる．

$$Diff_j = GHR - AGHR_j$$

$Diff_j$ が小さい程，j 個目のファイルのディスクキャッシュを置換え対象としたときのヒット率の低下が大きいといえる．　次に，j 個目のファイルを置換え対象とし，j 個目のファイルのディスクキャッシュの量が減った場合の見積もりヒット率を $AssumptiveFileHitRate_j$ (以降，$AssumptiveFileHitRate_j$ と表記) とすると，$AGHR_j$ は，$AGHR_j = \sum_{i=1}^{n} FRR_i \cdot AFHR_i$ となる．したがって，$Diff_j$ は，次のように求まる．

$$\begin{aligned} Diff_j &= \sum_{i=1}^{n} FRR_i \cdot FHR_i \\ &\quad -(\sum_{k=1}^{j-1} FRR_k \cdot AFHR_k + FRR_j \cdot FHR_j + \\ &\quad \sum_{k=j+1}^{n} FRR_k \cdot FHR_k) \\ &= FRR_k \cdot FHR_k - FRR_k \cdot AFHR_k \\ &= FRR_j(FHR_j - AFHR_j) \end{aligned}$$

置換えの際には，$Diff_j$ が小さいものを選択すればよいので，すべてのファイルに対して $Diff_j$ を調べ，最も小さいファイルのディスクキャッシュを置換えの対象とする．

4 シミュレーション

他の置換アルゴリズムとの比較のために, ディスクキャッシュのシミュレータを作成した. 以下本章では, シミュレータの概要とシミュレーションの結果について述べ, 結果についての考察を行う.

4.1 概要

他の置換アルゴリズムとの比較のために, ディスクキャッシュのシミュレータを作成した. シミュレータは, ページを単位としたファイルの読出しとメモリの管理のシミュレーションを行う.

シミュレータには, シミュレータ内で用いるファイルの一覧, ファイルに対する読出し要求の一覧, ディスクキャッシュとして用いることの可能なメモリのサイズを与える. ファイルの一覧は, ファイルの ID とファイルのサイズの組が複数書かれたテキストファイルである. ファイルに対する読出し要求の一覧は, 読出すファイル, 読出しを開始するファイルのオフセット, 読出すサイズの組が複数書かれたテキストファイルである. シミュレータは, これらの与えられた入力を置換アルゴリズムで処理し, ディスクキャッシュがヒットした回数とミスした回数を返す. シミュレータ上で動作する置換アルゴリズムとして, LRU, SEQ, 提案手法の 3 つを用意した.

4.2 シミュレーション結果

各アルゴリズムの特徴を比較し, 初期的な評価を行うために, 単純なテストケースを 5 種類用意し, LRU, SEQ, 提案手法によるシミュレーションを行った. 全てのケースにおいて, サイズが 100 ページのファイルを 2 個用意し, それぞれをファイル 1 , ファイル 2 と呼ぶ. また, ディスクキャッシュとして用いることの可能なメモリのサイズは, 100 ページとする. 用意したケースの内容は以下のようになっている.

ケース 1 提案手法にとって理想的な環境を想定した. ファイル 1 に対してはシーケンシャルな読出しを繰り返し行い, ファイル 2 に対してはファイル全体をランダムに読み出した.

ケース 2 双方のファイルに対してファイル全体をランダムに読み出すが, ファイル 2 は, ファイル 1 と比較して 2 倍の頻度で読み出し, 読み出し頻度に差がある場合を想定した.

ケース 3 1 つのファイルに対し, 異なる特徴をもつ読出しが同時に発生することを想定した. ファイル 1 に対してはシーケンシャルな読出しとファイル全体に対するランダムな読出しの両方を行い, ファイル 2 に対してはファイル全体に対してランダムな読出しを行った.

ケース 4 双方のファイルに対してシーケンシャルな読出しを行った.

ケース 5 双方のファイルに対してファイル全体をランダムに読み出した.

シミュレーションの結果得られたディスクキャッシュのヒット率を表 1 に示す.

表 1 シミュレーション結果

ケース	LRU	SEQ	提案手法
1	21.35%	42.75%	44.45%
2	52.76%	52.63%	64.30%
3	42.60%	44.55%	46.10%
4	00.00%	44.55%	00.00%
5	49.25%	49.80%	49.20%

4.3 考察

ケース 1 とケース 2 においては, 提案手法が最も高いヒット率であった. ケース 1 において, 提案手法では, シーケンシャルにアクセスされているファイルは, FHR が低くなるため, 優先的に置換えの対象となり, これにより高いヒット率をとなった. 一方, SEQ は, ディスクキャッシュのヒット率の低下から, シーケンシャルな読出しが発生していると判断し動作を MRU に変えた結果, LRU よりも高いヒット率となった. ケース 2 においては, 提案手法が LRU と SEQ に比べて高いヒット率であった. これは, 本手法で用いている, ファイルが読出される頻度を示す FRR が作用した結果だと考えられる.

ケース 3 においては, どのアルゴリズムも大きな差がないという結果になった. これは, 1 つのファイルに対して, 複数の読出しが同時に行われたため, 提案手法において, ファイルの特徴を正確に得ることができなかったためであると考えられる.

ケース 4 とケース 5 において, 提案手法は, 良い結果が得られなかった. これらのケースは, 両方とも読み出すファイルの特徴が一様である. この結果から, ファイルの特徴に応じて扱いを変える本手法は, このような苦手とするケースであるということ

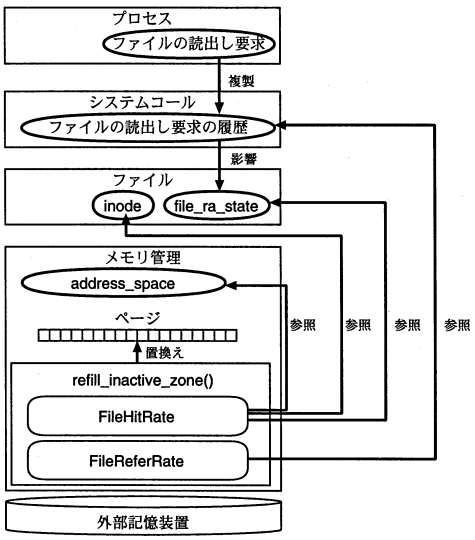

図 3　本手法の概要

が分かった.

以上の結果から, 複数のファイルがそれぞれ異なる特徴を持っている場合において, 提案手法は有効であると考えられる. また, 複数のファイルが同じ特徴を持つ場合や, 異なる特徴を持っていてもそれを検出できない場合においては, 従来の手法と比較し, 大きな差がないことが分かった.

今回のシミュレーションでは, 単純化されたテストケースを用いることによってファイルのアクセスパターンの違いによるアルゴリズムの挙動を確認したが, 実際の場面における有効性を確認するためには, より現実的なテストケースを用意する必要がある. そのため, 今後は実際のファイルアクセス状況をログとして記録し, これをテストケースとしたシミュレーションを行うことによる提案アルゴリズムの評価を行う必要があると考えている.

5　実装

本手法を実装するためには, 手法中で使用するパラメータを OS 内で取得する必要がある. ここでは, Linux Kernel 2.6 におけるパラメータの取得方法を検討した. Linux における実装の概要を図 3 に示す. Linux Kernel における実装では, ページを置換える必要がある際に呼び出される refill_inactive_list() の中で, FHR_i と FRR_i 算出を行う. また, FHR_i と FRR_i から $Diff_i$ を算出し, 置き換えるべきページを決定する. 以下, 本章では, FRR_i から $Diff_i$ に必要なパラメータの取得方法と, $Diff_i$ を算出する際に行う処理について述べる.

5.1　FHR_i の算出に必要なパラメータ

FHR_i は, Seq, $FileSize$, $CacheSize$ より求める. Seq は, ファイルごとに存在する file_ra_state という構造体から取得する. file_ra_state 構造体は, ファイルの先読みを行うために, ファイルがシーケンシャルであるか否かを示す数値をメンバとして持っている. file_ra_state 構造体は, ファイルシステムが管理をしているが, ファイルの読出しを行う際に読出しがシーケンシャルであるか否かを確認し, その結果を入れる構造体である. したがって, ファイルシステムにおいて管理されているが, その内容はプロセスの特徴である. $FileSize_i$ は, ファイルごとに存在する inode 構造体が持つファイルのサイズから取得する. $CacheSize_i$ は, address_space 構造体 のメンバである nrpages から取得する. address_space 構造体は, ファイルごとのディスクキャッシュの管理を行うための情報を記録する構造体である. address_space 構造体 のメンバである nrpages は, そのファイルの中でキャッシュされているページの数を持っている.

5.2　FRR_i の算出に必要なパラメータ

FRR_i を求めるためには, ファイルの読出しの履歴を取り, ファイルの読出しが行われている回数をファイルごとに数えなければならない. Linux において, ファイルの読出しは, open システムコールと read システムコールの組合せで行うものと, open と mmap の組合せで行うものの 2 種類に分けることができる.

open システムコールと read システムコールの組合せでファイルの読出しを行う場合, ファイルの内容を読み出すたびに, read システムコールを用いる. そのため, read システムコールをフックし, 引数として与えられるファイルディスクリプタと読出すサイズを得ることができる. ファイルディスクリプタは, そのファイルディスクリプタを経由して読み出すファイルの属する inode のアドレスをメンバとして持っている. inode は, inode が管理するファイルがどのデバイスに属するかという情報であるデバイスの id とそのデバイス内で一意に付けられる inode の id を持っている. デバイスの id と inode の id によって, ファイルを一意に識別できる.

open システムコールと mmap システムコール

の組合せでファイルの読出しを行う場合, ファイルの読出しすべてをフックすることはできない. ファイルのある部分を読み出す際, ファイルの内容がまだディスクキャッシュ上に存在していない場合にはページフォルトが発生しファイルの中の該当部分をディスクキャッシュに複製する. このディスクキャッシュを複製する際に, inode の情報を用いてファイルの内容を読出すため, この処理を行う際の情報を用いることで, ファイルの読出しの履歴を取ることが可能である. しかし, 既にファイルの内容がディスクキャッシュにファイルの内容が複製されている状況での読出しは, マッピングされたメモリ空間へのアクセスとなるため, 読出しをフックすることができない. したがって, ページフォルトが発生する初回の読出しにおいてはどのファイルへの読出しであるかを知ることが可能であるが, 2 回目以降の読出しにおいては不可能であるため, 読出しの回数を数え挙げることができない.

ファイルの読出し履歴をいずれの方法で取得する場合においても, デバイス の id と inode の id によって, ファイルを一意に識別する. そして, ファイルごとにキューを用意し, 要素の数を数えながら履歴を保存する. また, 古くなった履歴は削除する. そのため, ファイルごとにキューを用意し, 読出しが起こった時刻をキューに保存する.

一定期間内にあるファイルへの読出しの回数を $HistoryCount_i$ とすると, FRR_i は, $FRR_i = HistoryCount_i / \sum_{j=1}^{n} HistoryCount_j$ となる.

5.3 GHR および $Diff_i$ の算出

Linux Kernel 2.6 において, 空きページの確保は, read などにより空きページを利用しようとしたが既に空きページが存在しない場合に行われる. また, 定期的に空きページの量を確認し, その時点で空きページが少なかった場合にも行われる. いずれの場合も, refill_inactive_zone() が呼ばれ, この関数内で置換えのための処理を行っている. この関数内で各パラメータ参照し, GHR および $AGHR_i$ の見積もりを行う. その結果を用いて, $Diff_i$ を算出し, どのページを置き換えるか判断するように変更を加える. また, 古くなったファイルごとの read の履歴を削除する.

6 おわりに

本稿では, ファイルとプロセスの特徴に基づくディスクキャッシュ全体のヒット率の見積もりを用いた置換アルゴリズムについて述べた. 提案アルゴリズムにおいては, ディスクキャッシュのヒット率は, ファイルごとに OS が管理しているプロセスの特徴, ファイルの特徴, ディスクキャッシュの状況から見積もる. また, あるファイルのディスクキャッシュを置換えたとした場合のディスクキャッシュ全体のヒット率を見積もることも可能であるので, どのファイルの複製をもつディスクキャッシュを置き換えると最もディスクキャッシュ全体のヒット率が低下せずに済むかを調べ, 最もヒット率が低下しないディスクキャッシュを置換えの対象として選ぶ. また, シミュレータで本手法と他の置換アルゴリズムの比較を行った. 本手法は, その特徴から, 複数のことなる特徴を持つファイルを扱う際に, 良い結果を得ることができることが分かった. また, 本手法を Linux カーネルに実装する場合, 手法の実現のために必要となるパラメータを取得する方法について述べた.

今後は, 実際の利用場面を想定したシミュレーションを行い, 本手法の有効性の確認を行うとともに, Linux カーネル上に提案手法によるキャッシング機構を実装し, 評価を行う予定である.

参考文献

[1] Glass, G. and Cao, P.: Adaptive page replacement based on memory reference behavior, in *SIGMETRICS '97: Proceedings of the 1997 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pp. 115–126, New York, NY, USA (1997), ACM Press.

[2] Smaragdakis, Y., Kaplan, S. and Wilson, P.: EELRU: simple and effective adaptive page replacement, in *SIGMETRICS '99: Proceedings of the 1999 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pp. 122–133, New York, NY, USA (1999), ACM Press.

[3] Kim, J. M., Choi, J., Kim, J., Noh, S. H., Min, S. L., Cho, Y. and Kim, C.-S.: A Low-Overhead, High-Performance Unified Buffer Management Scheme That Exploits Sequential and Looping References., in *OSDI*, pp. 119–134 (2000).