情報処理学会論文誌 Vol.56 No.6 1451–1462 (June 2015)

推薦論文

ファイル操作のシステムコール発行頻度に基づくバッファキャッシュ制御法における重要度更新契機の設定法

山内 利宏[1,a)] 横山 和俊[2] 乃村 能成[1] 谷口 秀夫[1] 芦塚 正雄[1]

受付日 2014年9月6日, 採録日 2015年3月4日

概要：外部記憶装置との入出力の回数を削減するために，バッファキャッシュが利用されている．バッファキャッシュのブロック置き換え規則としては，ブロックへのアクセスパターンに着目したものが多い．これに対し，著者らはファイルの情報に基づくバッファキャッシュ制御方式（FFU 方式）を提案した．この方式は，ファイルの OPEN 回数によりファイルの重要度を決定し，この重要度に基づき 2 レベルに分割したバッファキャッシュを制御する．この方式は，キャッシュヒット率を向上できるものの，ファイルの重要度を更新する契機については，その設定法が明らかでない．そこで，本論文では，FFU 方式における重要度更新契機の設定法について述べる．具体的には，各ファイルの OPEN システムコール情報からファイルが集中的に OPEN されているか否かという状態を抽出し，この状態変化をとらえて重要度を更新する契機を設定する．また，OPEN システムコール間隔の分布が異なる場合，および実データの場合について評価し，本手法が有効であることを示す．

キーワード：バッファキャッシュ制御法，オペレーティングシステム，ディレクトリ，ファイル

Setting Method of Opportunity of Updating File Importance on FFU

Toshihiro Yamauchi[1,a)] Kazutoshi Yokoyama[2] Yoshinari Nomura[1]
Hideo Taniguchi[1] Masao Ashizuka[1]

Received: September 6, 2014, Accepted: March 4, 2015

Abstract: Buffer cache is implemented to improve I/O performance with data in disks. As buffer cache management, there are many mechanisms based on access pattern of block. On the other hand, we proposed I/O buffer cache mechanism based on the frequency of file usage (FFU). Our previous proposed mechanism calculates file importance from the information of system-call of the file operation. Then, it controls two level buffer cache based on the file importance. In this paper, we propose a setting method of opportunity of updating file importance on FFU. The proposed method focuses on whether the file state is access intensive or not. This paper also describes a setting method of parameters of the proposed method based on access information of a target system. Finally, this paper reports the evaluation results of the proposed method by using typical access pattern data and real access patterns.

Keywords: buffer cache mechanism, operating system, directory, file

1. はじめに

多くのオペレーティングシステム（以降，OS と略す）で

[1] 岡山大学大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University, Okayama 700–8530, Japan
[2] 高知工科大学情報学群
School of Information, Kochi University of Technology, Kochi 782–8502, Japan
[a)] yamauchi@cs.okayama-u.ac.jp

本論文の内容は 2013 年 10 月の支部連合大会にて報告され，支部長により情報処理学会論文誌ジャーナルへの掲載が推薦された論文である．

は，外部記憶装置との入出力の回数を削減するために，バッファキャッシュが利用されている．バッファキャッシュでは，ブロックと呼ばれる固定長のデータ単位でデータを管理し，ブロック単位でバッファキャッシュのデータを置き換える．このため，キャッシュヒット率を向上させるためには，バッファキャッシュのブロック置き換え規則が重要である．

ブロック置き換え規則には，ブロックへのアクセスに関する情報を基に制御する方式とファイルに関する情報を基に制御する方式がある．ブロックへのアクセスに関する情報を基に制御する代表的な方式として LRU 方式がある．しかし，LRU 方式では，1 つの大きなファイルへの入出力により，他のファイルがバッファキャッシュから解放されてしまい，キャッシュヒット率が低下するという問題がある．また，特定ファイル群に対して全データに繰り返しアクセスする場合，そのファイル群のデータ総量がバッファキャッシュの大きさを上回るとキャッシュヒット率が低下する．そのほかの方式として，アクセス頻度 [1], [2]，アクセス間隔 [3], [4]，ブロックアクセスの周期性 [5], [6] の情報を利用するものがある．しかし，特定の応用プログラム（以降，AP（Application Program）と略す）を指向した方式では，他の AP の入出力の性能を低下させるという問題がある．

これに対し，ファイルの情報に基づくバッファキャッシュ制御方式として，著者らは，ファイル操作のシステムコール発行頻度に基づくバッファキャッシュ制御法 Frequency of File Usage 方式（以降，FFU 方式と略す）を提案した [7]．FFU 方式は，従来の LRU 方式に比べ，キャッシュヒット率を向上できる．しかし，ファイルの重要度を更新する契機については，その設定法が明らかでない．

そこで，本論文では，FFU 方式における重要度更新契機の設定法について述べる．具体的には，過去の各ファイルの OPEN システムコール情報からファイルが集中的に OPEN されているか否かという状態を抽出し，この状態変化をとらえて重要度を更新する契機を設定する．また，OPEN システムコール間隔の分布が異なる場合，および実データの場合について，評価結果を報告する．

2. ファイル操作のシステムコール発行頻度に基づくバッファキャッシュ制御法

2.1 基本方式

文献 [7] で提案したファイル操作のシステムコール発行頻度に基づくバッファキャッシュ制御法 FFU 方式について説明する．FFU 方式は，ファイルを操作するシステムコールの発行を契機として，ファイルを操作した情報（以降，ファイル操作情報と略す）を取得する．また，一定の周期でファイル操作情報を集計し，ファイルの重要度を算出し，この重要度に基づき，2 レベルに分割したバッファキャッシュを制御する．

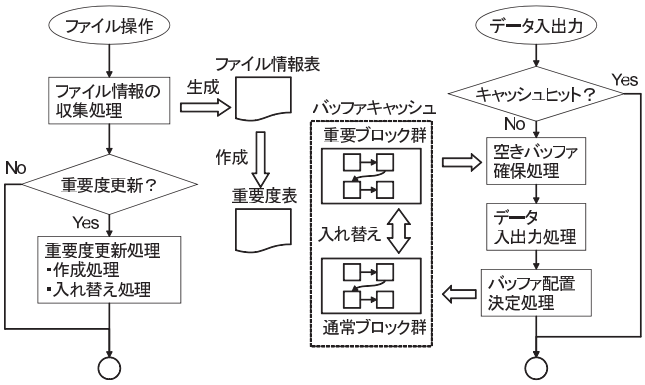

図 1 基本方式
Fig. 1 Basic mechanism.

表 1 ファイル情報表
Table 1 File information table.

項目	内容
inode 番号	ファイルの識別情報
オープン回数	更新前のオープン回数に重みを乗じた数と最後の更新契機からのオープン回数の和
ファイルサイズ	ファイルの大きさ
最新オープン時刻	最後にオープンされた時刻

表 2 重要度表
Table 2 Importance table.

項目	内容
inode 番号	ファイルの識別情報
重要度	バッファキャッシュに格納される優先度

提案手法の基本方式を**図 1** に示し，説明する．FFU 方式では，バッファキャッシュのために確保された領域を，重要と判断されたファイルを構成するブロック（重要ブロック群）が格納されたキューと，重要と判断されなかったファイルを構成するブロック（通常ブロック群）が格納されたキューで管理する．

ファイル操作時（OPEN または CLOSE システムコール発行時）の処理の流れを説明する．

(1) OPEN と CLOSE のシステムコールからファイル操作情報を取得し，**表 1** に示すファイル情報表に格納する．
(2) 重要度更新処理を行うか否か判断する．重要度を更新する契機であれば重要度更新処理を行い，契機でなければ処理を終了する．なお，重要度の初期値は 0 である．
(3) 重要度更新処理を行う場合，ファイル操作情報から重要度を算出し，**表 2** に示す重要度表を作成する．次に，重要度更新処理後に，新たに重要と判断したファイルに属するブロックをバッファキャッシュの通常ブロック群から重要ブロック群に移動する．また，重要

でなくなったファイルに属するブロックを重要ブロック群から通常ブロック群に移動する（入れ替え処理）．

データ入出力時の処理の流れを以下に述べる．

(1) read または write システムコールを契機として，要求されたデータがキャッシュヒットしたか否かを調べる．
(2) キャッシュミスした場合，バッファキャッシュから空きバッファを確保する．
(3) データ入出力処理を行う．
(4) データ入出力処理でアクセスしたブロックが構成するファイルが重要であれば重要ブロック群に，重要でなければ通常ブロック群に格納する．なお，重要かどうかの判断には，保護ファイル数という閾値を用い，重要度の順番で上位から保護ファイル数分だけのファイルを重要と判断する．

2.2 設計

2.2.1 収集するファイル操作情報

FFU 方式では，再利用される可能性が高いファイルの重要度を高くするため，オープン回数とファイルサイズをファイル情報表に格納する．また，これらに加えて，ファイルを特定するために i ノード番号も格納する．さらに，2 つのファイルの重要度が等しい場合に順位付けをするために，最近 OPEN システムコールが発行された時刻も格納する．

2.2.2 重要度の算出方法

重要度の算出方法を下記の擬似コード (1) に示す．擬似コード (1) 中において，O_{cnt} は計算対象ファイルのオープン回数，F_{size} はそのファイルサイズ，O_{min} と $F_{sizemax}$ は閾値，$Importance$ は重要度を示す．重要度は，オープン回数を基にして算出する．また，オープン回数が多くても，一定のサイズより大きいものを通常ブロック群で管理する．

$$
\begin{aligned}
&\text{if } ((O_{cnt} > O_{min})\&\&(F_{size} < F_{sizemax}))\{\\
&\quad Importance = O_{cnt};\\
&\}\ \text{else}\{\\
&\quad Importance = FALSE(=0);\\
&\}
\end{aligned}
\tag{1}
$$

2.2.3 重要度を更新する契機

OPEN，CLOSE システムコールの発行状況は，つねに変動するため，重要度表の更新契機は，前回の重要度表の更新から一定回数のオープンまたはクローズが行われたときとする．また，重要度表の更新時に，更新直前のファイル操作情報のオープン回数に重み w をかけることで，古い情報の影響を周期的に小さくする．これにより，最近，頻繁にオープンされたファイルの重要度を高くする．

2.2.4 バッファキャッシュの入れ替え処理

重要度を算出したときに，新しく算出した重要度に合わせて，バッファキャッシュの格納領域を入れ替える．これによって，重要度の高いファイルに属するブロックが通常ブロック群に格納されること，および重要度の低いファイルに属するブロックが重要ブロック群に格納されることを防ぐ．

2.2.5 空きバッファ確保処理

空きバッファの確保は，通常ブロック群から LRU 方式でブロックを選んで解放することで行う．もし通常ブロック群に解放可能なブロックがないとき，重要ブロック群内で最も重要度の低いファイルを構成するブロックすべてを重要ブロック群から通常ブロック群に移動し，通常ブロック群から LRU 方式でブロックを選んで解放する．

3. 重要度更新契機の設定法

3.1 要求

重要度は，ファイルの OPEN 回数に大きく影響される．このため，重要度の更新契機は，OPEN システムコールの発行状況に連動することが望ましい．たとえば，特定のファイル群へ OPEN システムコールが頻繁に発行されている場合，それらのファイル群は重要度の高いファイルとして，それらが使用しているバッファキャッシュを保護する．逆に，あまり OPEN システムコールが発行されないファイル群については重要度を低くし，使用しているバッファキャッシュを置き換え対象とする制御を行う．つまり，ファイルの重要度の更新は，頻繁に使用されるファイル群が変化したときに行う必要がある．以下に，ファイルの重要度を更新する契機への要求を示す．

(要求条件 1) 頻繁に使われるようになり重要度が高いと判断できるファイルは，迅速に保護することが求められる．つまり，OPEN されるファイル群やそれらへの OPEN 回数が変化した場合，速やかに重要度を再算出し，保護対象とするファイルを見直すこと．

(要求条件 2) 重要度を算出するオーバヘッドが少ないことが求められる．つまり，重要度を算出する回数ができるだけ少ないこと．

3.2 基本方式

ファイルが集中的に OPEN されているか否かという状態変化に着目した重要度更新契機の設定法について，以下に説明する．

(1) OPEN されるファイルの状態を，閾値 P を用いて集中状態と非集中状態に分類する．このため，OPEN 番号と OPEN 間隔を導入する．OPEN 番号とは，OPEN を発行された順に並べ，番号をつけたものである．また，OPEN 間隔とは，着目したファイルについて OPEN から次の OPEN までに発生した OPEN の総数

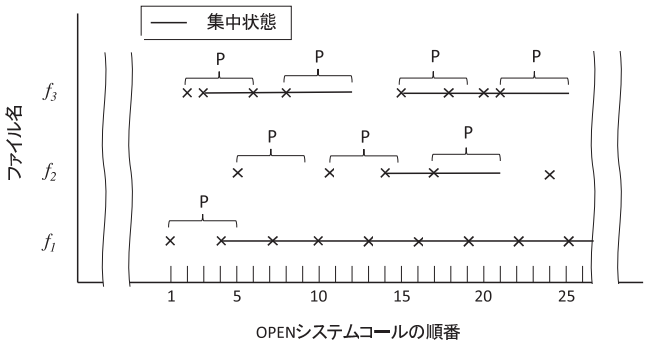

図 2 ファイルの状態変化
Fig. 2 State change of file.

である．つまり，着目したファイルについてOPEN番号がaであり，次のOPEN番号がbのとき，OPEN間隔は$b-a$となる．集中状態のファイルとは，OPEN間隔が閾値P以下のファイルであり，非集中状態のファイルとは，OPEN間隔が閾値Pより大きいファイルと定義する．

(2) ファイルをOPENするごとに，管理対象のファイルすべてについて，ファイルのOPEN状態の更新処理を行い，ファイルの状態を更新する．

(3) 状態が変化したファイルの数が閾値Rを超えたとき，ファイルの重要度を更新する契機とする．以降では，状態が変化したファイル数を状態変化数と呼ぶ．また，ある状態変化が発生し，次の状態変化までに発生するOPENの総数を，状態変化間隔と呼ぶ．つまり，OPEN番号がcで状態変化が発生し，次の状態変化がOPEN番号dで発生した場合，状態変化間隔は$d-c$となる．

図 2 を用いて，ファイルの状態変化について説明する．図の横軸は，発行された順にOPENシステムコールを並べている．また，縦軸は操作対象のファイルを示し，この例ではf_1からf_3までのファイルがある．また，閾値Pは4と仮定している．ファイルf_1は，番号1でOPENされ，次は番号4でOPENされる．このときのOPEN間隔は3であり，閾値Pより小さい．つまり，ファイルf_1は，番号4のOPENで集中状態に移行する．ファイルf_2は，番号5の後，番号11でOPENされる．このときのOPEN間隔は6のため，非集中状態のままである．ファイルf_2は，次に番号14でOPENされる．このときのOPEN間隔は3のため，番号14で集中状態に移行する．さらに，番号21では，直前のOPENとの間隔が閾値の4を超えるため，非集中状態へ移行する．

次に，同じ例を用いて，状態変化間隔と状態変化数について説明する．図2の例では，番号3のOPENでファイルf_3の状態が変化する．次の状態変化は，番号4でファイルf_1が集中状態に移行する．このときの状態変化間隔は1である．また，図2の例での状態変化は，番号3，4，12，14，15，21，25の7回である．提案する設定では，状態変化数が閾値Rを超えた場合に重要度を更新する．頻繁に使用されるファイル群が急に変化する場合は，状態変化数が急速に増加するため，短期間で閾値Rを超え重要度が更新される．つまり，(要求条件1)を満たす動作を実現している．逆に，使用されるファイル群に変化がない場合は，状態変化数が閾値Rを超えるまで長い期間が必要になる．つまり，過度な重要度の更新は発生せず，(要求条件2)を満たす特徴を持っている．

3.3 閾値の設定法

提案する設定法で使用している閾値Pと閾値Rは，重要なパラメータである．提案手法では，対象とする処理が過去に実行されたときのOPEN情報を解析し，閾値Pと閾値Rを決定する．その決定した閾値を設定しサービスを再開することで，重要度更新契機の閾値を変更する．以下に閾値Pと閾値Rの決定法を示す．

(1) 閾値Pの設定
閾値Pは，ファイルを集中状態と非集中状態に分けるパラメータである．閾値Pの候補として，OPEN間隔の平均値と中央値の2つがある．ここでは，中央値を用いる．これは，次の理由による．平均値は，あまり使用されないファイルのOPEN間隔が極端に長いとき，その値が大きく影響されてしまう．つまり，重要でないファイルのOPEN間隔が支配的になり，重要でないファイルも集中状態に移行してしまう可能性が生じる．このことを防ぎ，集中状態に移行するファイルが頻繁にアクセスされる重要度が高いファイルに限るために，中央値を用いる．

(2) 閾値Rの設定
閾値Rは，ファイルの重要度を更新する契機に直接影響を与えるパラメータである．提案手法では，以下の式で示される値を閾値Rとして用いる．

$$R = P \times Q$$

ここで，Pは，OPEN間隔の中央値，Qは，状態変化間隔の中央値である．

OPEN間隔の中央値Pは，「多くのファイルは，P回ごとにOPENされる」ことを意味している．また，状態変化間隔の中央値Qは，「多くの場合，Q回のOPENが発生すると，1回状態変化が起こる」ことを意味しており，いい換えれば「OPENが発生した場合，それが状態変化をともなう確率は$1/Q$である」ことを意味している．つまり，多くのファイルは，「P回ごとにOPENされ，それがQ回繰り返された場合」に状態が変化する．そこで，この間隔を閾値Rとする．

4. 評価

4.1 考え方

3 章で述べた提案手法により，重要度更新契機を設定できる．ここでは，この設定が有効か否かを明らかにするために，提案手法が利用している OPEN システムコール情報のパターンとして，複数のパターンについて評価する．

重要度更新契機の設定においては，「OPEN が頻繁に行われるファイルを重要ファイルとして認識できる」ことが重要である．したがって，設定法の評価には，OPEN 時に対象ファイルが重要ファイルとして保護されている割合を示す保護率を尺度として用いる．重要度更新契機を設定できても，保護率が低下してはならない．このため，保護率は，文献 [7] と同等もしくは高いか否かを評価する．

提案手法は，各ファイルの OPEN システムコールの発行間隔を利用している．そこで，OPEN システムコール間隔の分布として，1 点集中，2 点集中，および指数分布的を基本パターンとした．また，文献 [7] と比較するために，OS の make 処理（正規分布的）と Web サーバ処理（指数分布的）も行った．具体的な内容を以下に示す．

(1) OPEN システムコールの発行間隔の違いに着目した基本パターン
  (a) 1 つのファイルについて OPEN システムコールの発行間隔が同じ（一定）であり，かつすべてのファイルで同じ場合，つまり，各ファイルへの OPEN システムコールが順次繰り返し発行される場合（1 点集中）
  (b) OPEN システムコールの発行間隔がランダムに発生する場合（指数分布的）
  (c) 上記 (a) の場合が連続して起こり，最初のファイル群と 2 番目のファイル群で OPEN システムコールの発行間隔が異なる場合（2 点集中）
(2) 実データとして，OS の make 処理，および Web サーバ処理

4.2 比較対象と評価尺度

文献 [7] より，一定数の OPEN システムコールが発行されたことを契機に重要度を更新する手法（以降，従来手法）は，LRU 方式よりキャッシュヒット率が高いため，ここでは，提案手法と従来手法を比較する．評価尺度として，保護率を用いる．ここで，ある OPEN システムコールが発行されたとき，操作対象のファイルが重要なファイルとして保護されているとき，その OPEN システムコールは保護ファイルにヒットしたと呼び，保護率とは，すべての OPEN システムコールに対し，ヒットか否かを求め，その割合を算出したものである．なお，重要度の算出で用いる重みは，文献 [7] と同様に 0.5 としている．

表 3 繰返しパターンの OPEN システムコール発行状況
Table 3 Information of OPEN system-call in repeat pattern.

項目	値
OPEN システムコールの発行総数	50,000 回
操作したファイル総数	100 個
OPEN 間隔の中央値	100

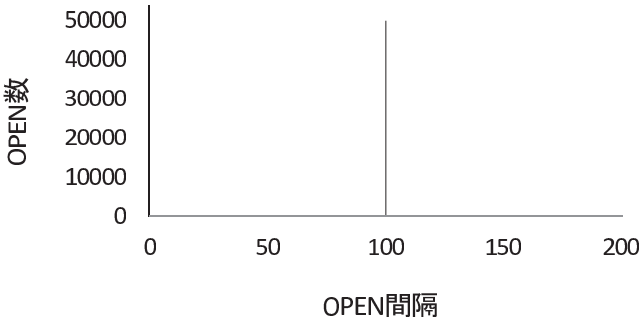

図 3 繰返しパターンの OPEN 間隔の分布
Fig. 3 Distribution of OPEN system-call interval in repeat pattern.

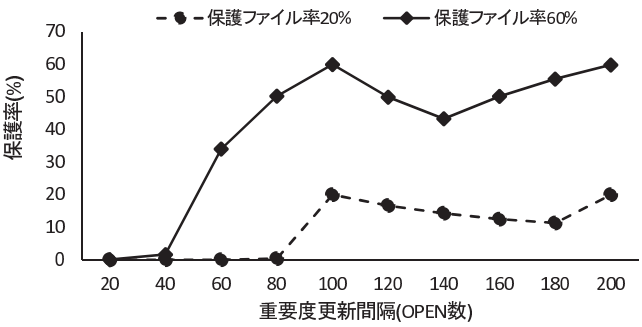

図 4 従来手法の保護率（繰返しパターン）
Fig. 4 Protected rate of previous method (repeat pattern).

4.3 基本パターンの評価

4.3.1 繰返しパターンの評価

各ファイルの OPEN が順次繰り返される場合について，評価に用いた OPEN システムコールの発行状況を**表 3** に示し，OPEN 間隔の分布を**図 3** に示す．繰返しパターンでは，100 個のファイル（識別子を 1〜100 とする）について，1 から順番に 100 のファイルへの OPEN を繰り返す．つまり，図 3 に示すように，すべての OPEN 間隔は 100 であり，その中央値も 100 である．

まず，従来手法について，重要度更新間隔を変化させた場合の保護率を**図 4** に示す．図中の保護ファイルの割合は，操作したファイルの総数に対し，保護ファイルを設定した割合（保護ファイル率）を示している．たとえば，総ファイル数が 100 で保護ファイル率 20%の場合は，総ファイル数の 20%である 20 個に設定している．基本評価では，保護ファイル数が多い場合と少ない場合での傾向を見るために，保護ファイル率が 20%と 60%の場合で評価した．図 4 より，従来手法では，重要度を更新する間隔である OPEN システムコールの発行数により，保護率が変動する

表 4 状態変化 OPEN の発行状況（繰返しパターン）
Table 4 Information of state change OPEN system-call in repeat pattern.

項目	値
状態変化数	100 回
状態変化間隔の中央値	1

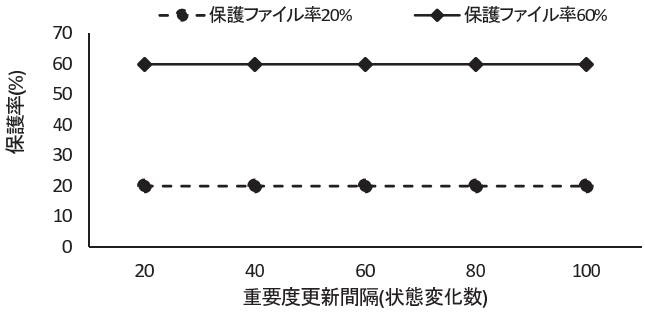

図 5 提案手法の保護率（繰返しパターン）
Fig. 5 Protected rate of proposed method (repeat pattern).

ことが分かる．これは，ファイルが OPEN される周期と，重要度更新の周期が一致するか否かで影響を受けるということである．たとえば，保護率が最も良いのは，OPEN システムコール数が 100 の場合であり，これは，ファイルが OPEN される周期と一致している．つまり，重要度の算出に用いた OPEN システムコールの系列とまったく同じ系列が次の周期でも出現するため，保護率が最も良くなる．

次に，提案手法の評価を示す．**表 4** は，状態変化の状況を示している．表 4 より，この場合の閾値 R は，$100 \times 1 = 100$ である．重要度更新契機を変化させた場合の保護率を**図 5** に示す．図 5 より，状態変化数が変化しても，保護率の変動がほとんどないことが分かる．この理由は，繰返しアクセスの場合は，ファイルの状態変化が最初の周期で固定されるためである．評価で用いた繰返しアクセスでは，100 個のファイルへの OPEN が規則的に繰り返される．これは，最初に OPEN される周期において，集中状態のファイルと非集中状態のファイルに分けられ，以降の周期では，同じ順番で OPEN システムコールが繰り返し発行されるため，ファイルの状態が変化することはなく，最初の周期での状態変化での重要度が最後まで用いられるからである．このため，保護率は，R によらず一定でかつ従来手法のピークと同等である．この点は，従来手法の保護率が重要度更新間隔（OPEN 数）の選択によって大きく悪化することに比べて優位な点といえる．

4.3.2 ランダムパターンの評価

ランダムパターンにおける OPEN システムコールの発行状況を**表 5** に示す．ランダムパターンでは，50,000 回の OPEN システムコールが，100 個のファイルをランダムに OPEN する．ランダムパターンでの OPEN 間隔の分布を**図 6** に示す．この場合の OPEN 間隔の中央値は 70 である．

従来手法について，保護率を**図 7** に示す．図 7 より，重要度を更新する間隔である OPEN 数を変化させても，保護率はほとんど変化しないことが分かる．これは，ランダムパターンのため，各ファイルが均等に OPEN されているため，どの周期で計算してもほぼ同じ重要度になるからである．

次に，提案手法の評価を示す．また，状態変化の状況を**表 6** に示し，保護率を**図 8** に示す．評価で用いたランダムパターンでは，ファイルの状態変化が頻繁に発生しているため，状態変化間隔の中央値が 1 である．つまり，この場合の閾値 R は，$70 \times 1 = 70$ となる．提案手法でも，重要度の更新間隔である状態変化数が変化しても，保護率はほとんど変わらない．これは，頻繁に状態変化が発生して

表 5 ランダムパターンの OPEN システムコール発行状況
Table 5 Information of OPEN system-call in random pattern.

項目	値
OPEN システムコールの発行総数	50,000 回
操作したファイル総数	100 個
OPEN 間隔の中央値	70

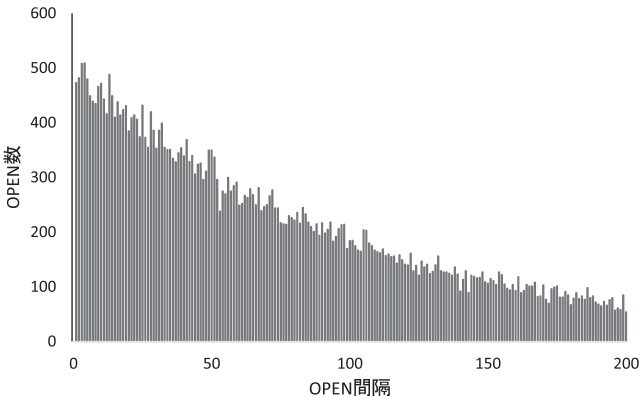

図 6 ランダムパターンでの OPEN 間隔の分布
Fig. 6 Distribution of OPEN system-call interval in random pattern.

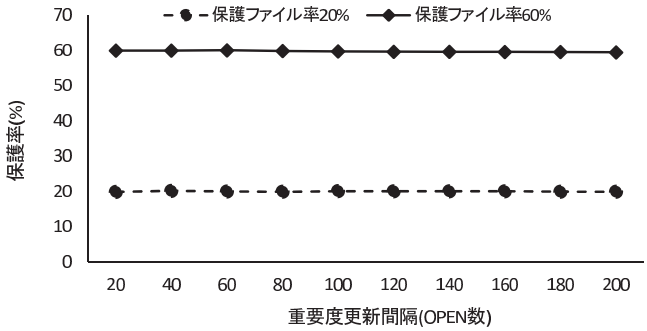

図 7 従来手法の保護率（ランダムパターン）
Fig. 7 Protected rate of previous method (random pattern).

表 6 状態変化の状況（ランダムパターン）
Table 6 Information of state change (random pattern).

項目	値
状態変化数	24,894 回
状態変化間隔の中央値	1

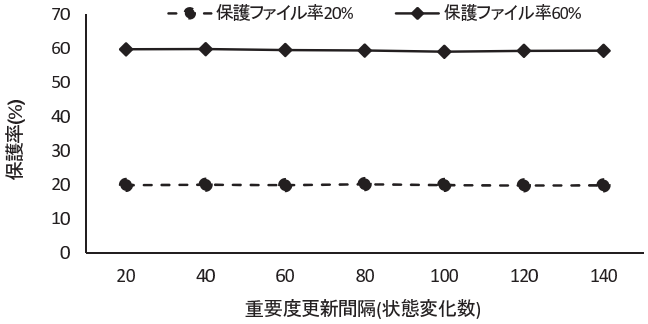

図 8 提案手法の保護率（ランダムパターン）
Fig. 8 Protected rate of proposed method (random pattern).

表 7 連続繰返しパターンの OPEN システムコールの発行状況
Table 7 Information of OPEN system-call in series of repeat pattern.

項目	値
OPEN システムコールの発行総数	40,000 回
操作したファイル総数	500 個
OPEN 間隔の中央値	300

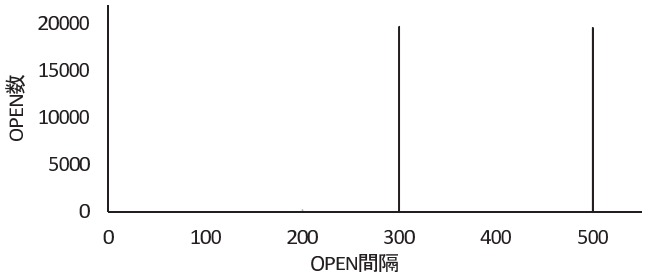

図 9 連続繰返しアクセスの OPEN 間隔の分布
Fig. 9 Distribution of OPEN system-call interval in series of repeat pattern.

いるため，どの区間においてもほぼ同じ数の状態変化が発生しているからである．つまり，完全なランダムパターンにおいては，提案手法は，従来手法と差が生まれない（劣らない）ということが分かる．

4.3.3 連続繰返しパターンの評価

連続繰返しパターンについて，評価で用いた OPEN システムコールの発行状況を**表 7** に示す．最初の 20,000 回の OPEN は，300 個のファイルを繰り返し OPEN する．その後の 20,000 回は，500 個のファイルを繰り返し OPEN する．このアクセスでの OPEN 間隔の分布を**図 9** に示す．OPEN 間隔が 300 と 500 のみに分布し，OPEN 間隔の中央値は 300 である．

従来手法と提案手法の保護率を，**図 10** と**図 11** に示す．また，状態変化の状況を**表 8** に示す．従来手法と提案手法を比較すると，提案手法の方が高い保護率を達成していることが分かる．これは，前半と後半の繰返し周期が異なるため，従来手法では，その周期に一致する重要度更新の間隔を設定できないためである．つまり，前半の周期に合わせると後半の保護率が低下し，逆に後半の周期に合わせると前半の保護率が悪化することが起こる．一方，提案手法は，繰返し周期に依存しないため，異なる周期の繰返しがある場合でも良好な保護率を達成することができる．

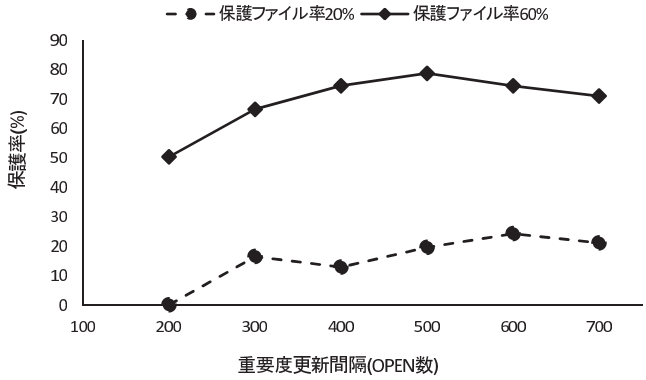

図 10 従来手法の保護率（連続繰返しパターン）
Fig. 10 Protected rate of previous method (series of repeat pattern).

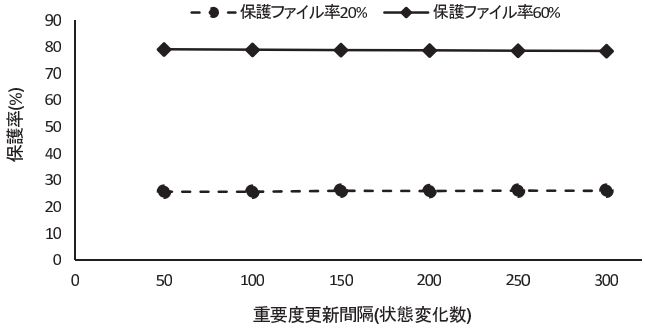

図 11 提案手法の保護率（連続繰返しパターン）
Fig. 11 Protected rate of proposed method (series of repeat pattern).

表 8 状態変化の状況（連続繰返しパターン）
Table 8 Information of state change (series of repeat pattern).

項目	値
状態変化数	600 回
状態変化間隔の中央値	1

従来手法では，重要度の更新間隔を変化させると，保護率が変化することが分かる．つまり，ファイルへの操作が異なる周期を持つ場合，重要度更新契機を適切に設定することが難しい．一方，提案手法では，繰返し周期に依存していないため，状態変化数を変化させても保護率の変動は小さい．つまり，重要度更新の契機を簡単に設定できる．表 8 より，状態変化間隔の中央値は 1 のため，閾値 R は $300 \times 1 = 300$ となる．300 の場合の保護率に着目すると，他の値での保護率とほぼ同じである．繰返しパターンと同様，従来手法の保護率が重要度更新間隔（OPEN 数）の選択によって大きく悪化することに比べて，提案手法は，安定性において優位であるといえる．つまり，本手法は，OPEN 数の間隔による従来手法と比べて，パラメータ調整が容易であるといえる．

表 9 OPEN システムコールの発行状況（make 処理）
Table 9 Information of OPEN system-call (kernel make).

項目	値
OPEN システムコールの発行総数	58,274 回
操作したファイル数	2,737 個
OPEN 間隔の中央値	83

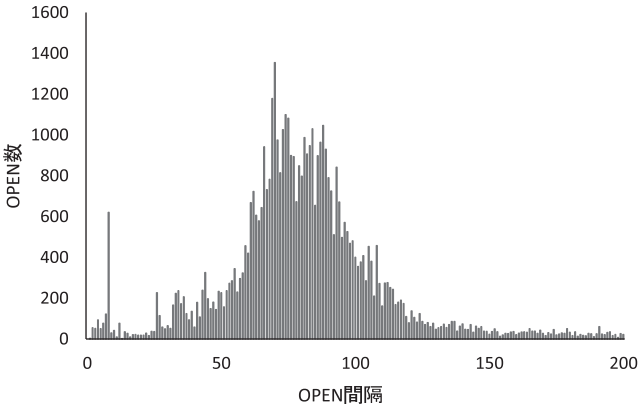

図 12 OPEN 間隔の分布（make 処理）
Fig. 12 Distribution of OPEN system-call interval (kernel make).

4.4 実データでの評価

実データを用いた評価として，文献 [7] のデータと同様に OS の make 処理のデータと，Web サーバ処理のデータでの評価を示す．

4.4.1 OS の make 処理

OS の make 処理での OPEN システムコールの状況を**表 9** に示す．make 処理中に 58,274 回の OPEN システムコールが発行され，操作したファイルの総数は 2,737 である．また，OPEN 間隔の分布を**図 12** に示す．OPEN の間隔の中央値は 83 であり，中央値 83 近辺の間隔で同じファイルが OPEN されている．つまり，make 処理は，繰返しパターンの特徴を持った処理といえる．

make 処理について，従来手法と提案手法を適用した際の保護率を，**図 13** と**図 14** にそれぞれ示す．また，make 処理での状態変化の状況を**表 10** に示す．なお，文献 [7] での保護ファイル数の割合を参考にし，保護ファイル率は 20%の場合で評価した．表 10 より，make 処理では，頻繁に状態変化が発生していることが分かる．つまり，あるファイル群を短い間 83 近辺の周期で繰り返し OPEN し，その後別のファイル群を 83 近辺の周期で OPEN している．従来手法の結果に着目すると，中央値である 83 近辺での保護率が高い．また，重要度更新の間隔を大きくすると保護率が下がる．これは，make 処理は，周期 83 近辺の繰り返しアクセスであるが，間隔を長くすると，別のファイル群への OPEN が混在するため，重要度の精度が落ちることが原因である．提案手法についても，状態変化数 40 までが最も良い保護率であり，状態変化数を大きくすると，保護率が下がる．閾値 R は，表 10 より，$83 \times 1 = 83$ である．最も良い保護率からは多少下がるが，比較的高い保護率であり，適切な値が設定できているといえる．従来手法と提案手法の比較を**図 15** に示す．両者を同じ尺度で比較するために，横軸を，重要度更新間隔から重要度更新回数に変更している．重要度更新間隔は，次のように計算できる．

(1) 従来手法

OPEN 数÷重要度更新の間隔（OPEN 数）

(2) 提案手法

状態変化数÷重要度更新の間隔（状態変化数）

図 15 より，両者に大きな違いは見られない．つまり，make 処理のように，OPEN の繰り返し中央値近辺に集中

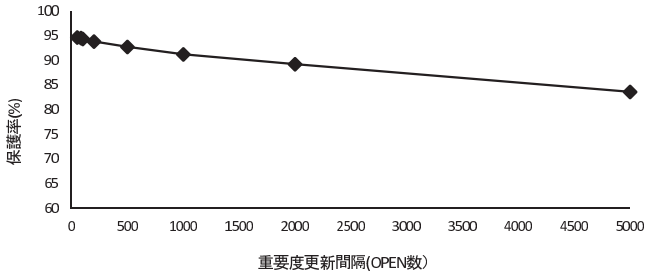

図 13 従来手法の保護率（make 処理）
Fig. 13 Protected rate of previous method (kernel make).

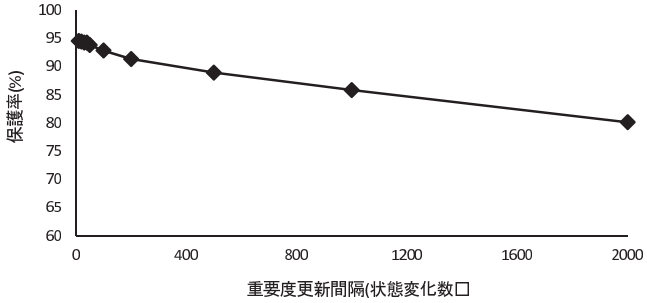

図 14 提案手法の保護率（make 処理）
Fig. 14 Protected rate of proposed method (kernel make).

表 10 状態変化の状況（make 処理）
Table 10 Information of state change (kernel make).

項目	値
状態変化数	17,514 回
状態変化間隔の中央値	1

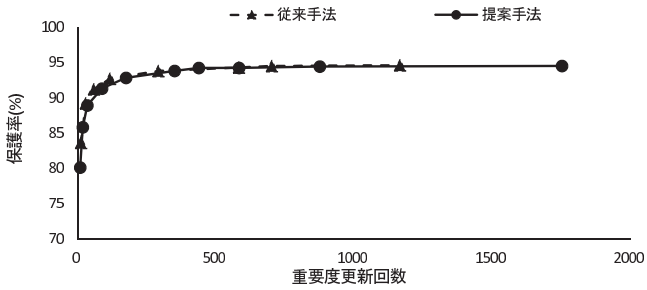

図 15 保護率の比較（make 処理）
Fig. 15 Comparison of protected rate (kernel make).

表 11 OPEN システムコールの発行状況（Web サーバ処理）
Table 11 Information of OPEN system-call (web server).

項目	値
OPEN システムコールの発行総数	99,849 回
操作したファイル数	8,781 個
OPEN 間隔の中央値	260

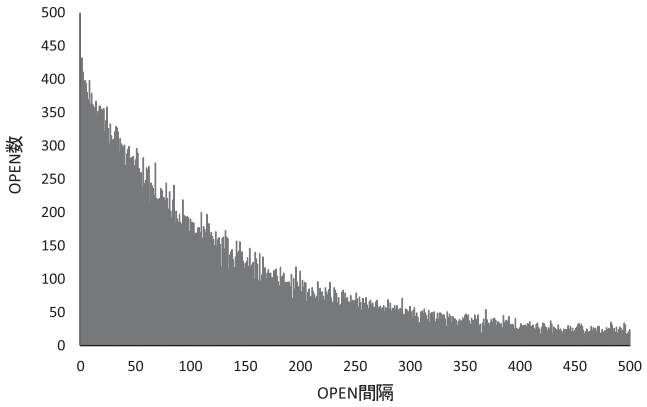

図 16 OPEN 間隔の分布（Web サーバ処理）
Fig. 16 Distribution of OPEN system-call interval (web server).

している処理では，OPEN システムコールの周期と状態変化の周期が一致し，差がほとんどなくなる．

4.4.2 Web サーバ処理

Web サーバ処理についての OPEN システムコールの状況を**表 11** に示す．全体で 99,849 回の OPEN が発行され，操作しているファイル数は 8,781 である．また，OPEN 間隔の分布を**図 16** に示す．OPEN 間隔の中央値は 260 であり，この OPEN 間隔の分布は，4.3.2 項で示したランダムパターンの特徴を持っている．

Web サーバ処理について，状態変化の状況を**表 12** に示す．make 処理に比べ，状態変化は少なく，状態変化間隔の中央値は 4 である．従来手法と提案手法を適用した際の保護率を，**図 17** と**図 18** にそれぞれ示す．なお，評価では，保護ファイル率 20%の場合である．両者とも，重要度更新の間隔が長くなると保護率が低くなるが，大幅に変動することはない．これは，Web サーバ処理では均等にファイルが OPEN されており，状態変化も均等に発生しているからである．そのため，どの間隔で計算しても重要度が大きく変わることはない．評価で用いた状態変化間隔の中央値が 4 である．つまり，この場合の閾値 R は，$260 \times 4 = 1{,}040$ である．閾値 $R(= 1{,}040)$ での保護率に着目すると，最良な値ではないが，他の値を設定したときとほとんど差がない．従来手法と提案手法の比較を**図 19** に示す．make 処理と同様に，両者を同じ尺度で比較するために，横軸を，重要度更新間隔から重要度更新回数に変更している．Web サーバ処理についても，両者の間に大きな差はない．これは，Web サーバ処理が，ランダムパターンの特徴を持っているためである．

表 12 状態変化の状況（Web サーバ処理）
Table 12 Information of state change (web server).

項目	値
状態変化数	18,382 回
状態変化間隔の中央値	4

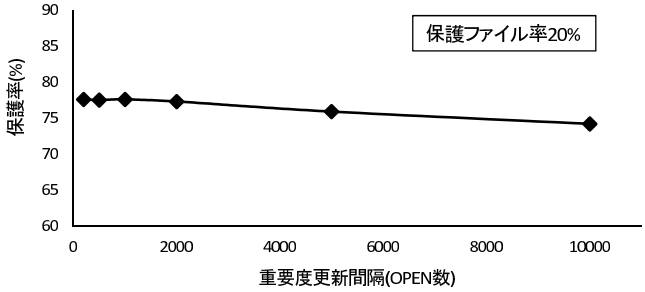

図 17 従来手法の保護率（Web サーバ処理）
Fig. 17 Protected rate of previous method (web server).

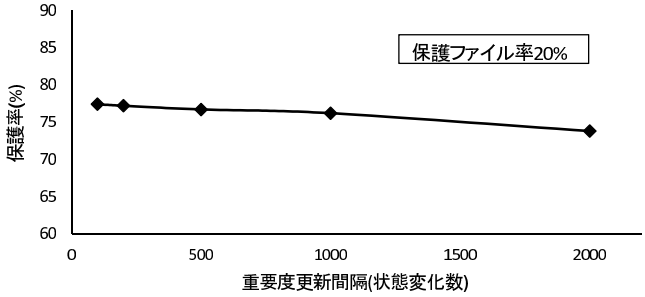

図 18 提案手法の保護率（Web サーバ処理）
Fig. 18 Protected rate of proposed method (web server).

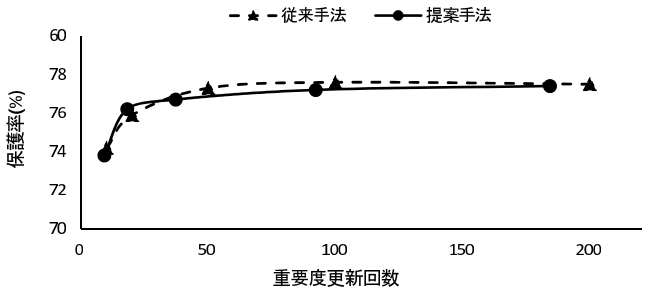

図 19 保護率の比較（Web サーバ処理）
Fig. 19 Comparison of protected rate (web server).

4.5 実装による評価

4.5.1 Web サーバ処理における平均応答時間

提案方式を実装し，4.4.2 項と同様に，文献 [7] の Web サーバ処理の平均応答時間の測定と同じ処理内容で測定し，評価した．測定には，クライアント計算機 1 台とサーバ計算機（CPU：Celeron 430（2.4 GHz），メモリ：64 MB，OS：FreeBSD 6.3-RELEASE，バッファキャッシュサイズ：12 MB，VMIO：無効）を用いた．測定結果を**図 20** に示す．

図 20 から，従来方式と比べて，同等の平均応答時間であることが分かる．また，提案する設定法の閾値 $R = 1{,}040$ では，平均応答時間は最良ではないものの，比較的良い値

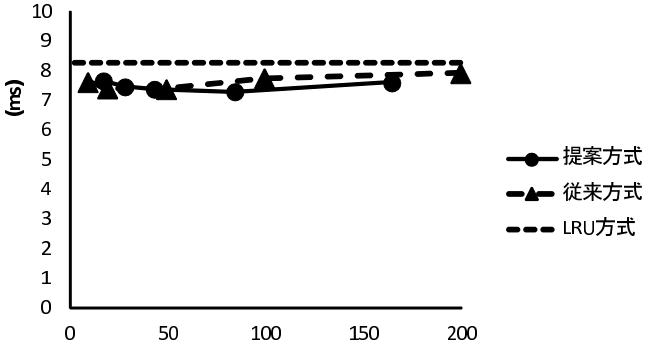

図 20 Web サーバ処理の平均応答時間
Fig. 20 Average response time of web server processing.

を設定できることが分かる．以上のことから，提案手法は，本評価において，従来手法と同等の平均応答時間となる閾値を設定できることが分かる．

4.5.2 オーバヘッド

OS の make 処理（保護ファイル数 548，$P = 83$，$Q = 1$，$R = 83$）と Web サーバ処理（保護ファイル数 1,757，$P = 260$，$Q = 4$，$R = 1{,}040$）における提案方式のオーバヘッドを測定した．測定には，平均応答時間の測定に用いた計算機を用いた．

提案方式でオーバヘッドが生じる処理は，以下の 2 つの契機がある．

(1) 毎回のファイル OPEN 時
状態変化が起こったファイル数を更新し，ファイルの状態を更新する．

(2) 重要度更新契機
重要度を再計算し，重要度表を作成し，バッファの入れ替えを行う．

毎回のファイル OPEN 時のオーバヘッドは，LRU 方式と，LRU 方式にファイルの状態を管理し，ファイルの状態変化数をカウントする処理を実装したものを比較した．重要度更新契機のオーバヘッドは，毎回の重要度更新処理の時間を測定し，その合計処理時間を求めた．測定結果から，以下のことが分かる．

(1) については，make 処理と Web サーバ処理の処理時間全体におけるオーバヘッドはそれぞれ 2.90 秒と 1.84 秒であった．これは，make 処理時間全体 963.58 秒の 0.30%，Web サーバ処理時間全体 825.41 秒の 0.22%と小さい．

(2) については，make 処理では，重要度の更新処理の合計処理時間が，1.54 秒，Web サーバ処理では，0.57 秒であった．また，重要度更新回数は，それぞれ 211 回と 43 回であった．1 回あたりの重要度更新処理時間は，それぞれ 7.31 ミリ秒と 13.24 ミリ秒で小さいとはいえないものの，その全体の処理時間に占める割合は，0.16%と 0.07%と小さく，処理全体への影響は小さい．

以上のことから，必要以上に重要度を更新することはオーバヘッドの面で好ましくないため，更新処理を少なくすることが重要である．

4.6 考察

4.3 節と 4.4 節の結果より，重要度更新契機の設定法について以下のことが分かる．

(1) 単純な繰返しパターンやランダムパターンの処理の場合，従来手法と提案手法で保護率に大きな差はない．これらの処理は，OPEN システムコールが繰り返される周期と，ファイルの状態変化の周期がほぼ一致するためである．

(2) 繰返しパターンを持つ処理について，従来手法では，重要度の更新間隔を変化させると保護率が変動する．つまり，重要度更新契機を適切に設定することが難しい．一方，提案手法では，繰返し周期に依存していないため，状態変化数を変化させても保護率の変動は小さい．このため，提案手法は，従来手法に比べて重要度更新契機を容易に設定できる．

(3) 異なる OPEN システムコールの発行間隔を持つ処理が続く場合は，提案手法は，高い保護率を得られる．これは，従来手法は，複数の周期に一致する重要度更新の間隔を設定できない．一方，提案手法は，繰返し周期に依存しないため，異なる周期の繰返しがある場合でも適切な重要度更新の間隔を設定できるためである．これにより，提案手法は (要求条件 1) を満足したといえる．

以上に示したように，提案手法において閾値 P を「OPEN 間隔の中央値」に設定し，閾値 R を「OPEN 間隔の中央値×状態変化間隔の中央値」に設定することで，その有効性を明らかにした．

なお，OS の make 処理と Web サーバ処理の実データの評価では，従来手法と提案手法で大きな差は見られなかった．これは，両者とも 1 種類のサービス処理について評価した結果である．実システムでは，複数の処理が同時あるいは逐次に走行することが多い．また，実システムでは日付や時間帯によりアクセスパターンが違うことがある．このような場合についても，提案手法は，(3) で示したように，異なる周期を持つ繰返し処理について極端に悪化することがないため，従来手法より有効と考えられる．

5. 関連研究

異なるストレージデバイスが共存する環境で，入出力コストを考慮したバッファ置き換えアルゴリズム ARC が提案されている [8]．また，文献 [9] では，フラッシュディスクを考慮したバッファキャッシュ置き換えアルゴリズムが提案されている．これらの研究は，ブロックアクセスのパターンを意識しているか否か，入出力性能の異なるディスクに着目している点で提案方式と異なる．

文献 [10] では，電力消費を抑えるために，キャッシュ

ヒット率を落とさずに電力を抑制し，チップの温度を削減するバッファキャッシュ置き換えアルゴリズムが提案されている．提案方式とは，LRU 方式よりもキャッシュヒット率を向上させることを目的とするか否か，評価尺度に電力を考慮しているか否かが異なる．

文献 [11] では，Prefetch を行うことにより，既存のバッファキャッシュ置き換えアルゴリズムの性能を向上できることを示している．この方式は，既存のバッファキャッシュ置き換えアルゴリズムと共存可能であり，提案方式と共存させることで，さらなるキャッシュヒット率の向上を実現できると推察できる．

ファイルの OPEN に着目した提案方式と異なり，ブロックアクセスの頻度や間隔に基づく制御方式として，2Q [5]，ARC [6]，および CAR [12] がある．2Q，ARC，および CAR は，一度しかアクセスされていないブロックと複数回アクセスされたブロックを別々の領域にキャッシュする．また，文献 [13] と [14] では，ブロックアクセスのパターンに着目した方式が提案されている．文献 [13] の研究では，2 階層のバッファキャッシュ方式を提案しており，AP ごとに適応的にキャッシュを割り当てることができ，かつアクセスパターンに応じてバッファ置き換えアルゴリズムを選択できる．文献 [14] は，ファイルごとにブロックアクセスパターンを分析し，制御する方式である．この 2 つの方式は，入出力バッファを分割し，制御する．文献 [15] は，入出力実行契機となったプログラムカウンタごとにアクセスパターンを分類するものである．提案方式の利点として，ファイル単位で OPEN 情報を取得するため，ブロック単位で情報を取得する方式に比べ，取得し管理する情報が少ないことが期待できる．

6. おわりに

ファイル操作のシステムコール発行頻度に基づくバッファキャッシュ制御法における重要度更新契機の設定法を提案した．提案手法は，ファイルが集中的に OPEN されているか否かという状態に着目し，この状態変化をとらえて重要度を更新する．このため，使用されているファイル群が変化することに合わせて重要度を更新することができる．特に，処理の実行経過にともなって使用されるファイル群が変化し，かつ，それらのファイル群へ異なる間隔で OPEN する場合に有効である．また，提案手法の重要なパラメータの設定法について，対象とするシステムの OPEN 情報を一定期間監視し，その OPEN 情報を基に閾値を決定する方法を示した．

評価により，提案手法での設定値は，良好な保護率を達成していることを示した．また，提案方式を実装し，Web サーバ処理の平均応答時間を評価した結果，従来方式と同等の平均応答時間となる閾値を設定できることを示した．さらに，オーバヘッドを評価し，全体の処理時間に占める割合が小さいことを示した．

残された課題として，複数のサービス処理が混在して走行する実システムでの評価がある．

謝辞 実装と評価にご協力いただいた岡山大学工学部情報系学科の寺岡明彦氏に感謝します．

参考文献

[1] Robinson, J.T. and Devarakonda, M.V.: Data Cache Management Using Frequency-Based Replacement, *Proc. 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pp.134–142 (1990).

[2] Lee, D., Choi, J., Kim, J.-H., Noh, S.H., Min, S.L., Cho, Y. and Kim, C.S.: LRFU (Least Recently/Frequently Used) Replacement Policy: A Spectrum of Block Replacement Policies, *IEEE Trans. Computers*, Vol.50, No.12, pp.1352–1360 (1996).

[3] Phalke, V. and Gopinath, B.: An Inter-Reference Gap Model for Temporal Locality in Program Behavior, *Proc. USENIX Summer 1994 Technical Conference*, pp.291–300 (1995).

[4] Jiang, S. and Zhang, X.: LIRS: An Efficient Low Interreference Recency Set Replacement Policy to Improve Buffer Cache Performancer, *Proc. 2002 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pp.31–42 (2002).

[5] Johnson, T. and Shasha, D.: 2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm, *Proc. 20th International Conference on Very Large Databases*, pp.297–306 (1993).

[6] Megiddo, N. and Modha, D.S.: ARC: A Self-Tuning, Low Overhead Replacement Cache, *Proc. 2nd USENIX Conference on File and Storage Technologies* (*FAST '03*), pp.115–130 (2003).

[7] 片上達也，田端利宏，谷口秀夫：ファイル操作のシステムコール発行頻度に基づくバッファキャッシュ制御法の提案，情報処理学会論文誌コンピューティングシステム（ACS），Vol.3, No.1, pp.50–60 (2010).

[8] Kim, Y.-J. and Anggorosesar, A.: Device-Aware Cache Management based on Adaptive Replacement, *Proc. 9th WSEAS International Conference on Software Engineering, Parallel and Distributed Systems*, pp.85–89 (2010).

[9] Ou, Y., Harder, T. and Jin, P.: CFDC: A Flash-Aware Buffer Management Algorithm for Database Systems, *Proc. 14th East European Conference on Advances in Databases and Information Systems*, pp.435–449 (2010).

[10] Yue, J., Zhu, Y., Cai, Z. and Lin, L.: Energy and Thermal Aware Buffer Cache Replacement Algorithm, *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies* (*MSST*), pp.1–10 (2010).

[11] Butt, A., Gniady, C. and Hu, Y.: The Performance Impact of Kernel Prefetching on Buffer Cache Replacement Algorithms, *IEEE Trans. Computers*, Vol.56, No.7, pp.889–908 (2007).

[12] Bansal, S. and Modha, D.: Car: Clock with adaptive replacement, *Proc. 3rd USENIX Conference on File and Storage Technologies* (*FAST'04*), pp.187–200 (2004).

[13] Meng, X., Si, C., Na, W., Khan, H.-U.-R. and Xu, L.: A Flexible Two-Layer Buffer Caching Scheme for Shared Storage Cache, *11th IEEE International Conference on High Performance Computing and Communications*, pp.424–431 (2009).

[14] Kim, J., Choi, J., Kim, J., Noh, S.H., Min, S., Cho, Y. and Kim, C.: A Low-Overhead High-Performance Unified Buffer Management Scheme that Exploits Sequential and Looping References, *Proc. 4th Symposium on Operating System Design and Implementation* (*OSDI 2000*), pp.119–134 (2000).
[15] Gniady, C., Butt, A. and Hu, Y.: Program-Counter-Based Pattern Classification in Buffer Caching, *Proc. 6th Symposium on Operating Systems Design and Implementation* (*OSDI 2004*), pp.395–408 (2004).

推薦文

情報処理学会中国支部表彰規定にのっとり，平成 25 年度（第 64 回）電気・情報関連学会中国支部連合大会で発表された中から特に優秀であることが認められた優秀論文発表賞を授賞した論文である．

（中国支部支部長　菅原一孔）

山内 利宏　（正会員）

1998 年九州大学工学部情報工学科卒業．2000 年同大学大学院システム情報科学研究科修士課程修了．2002 年同大学院システム情報科学府博士後期課程修了．2001 年日本学術振興会特別研究員（DC2）．2002 年九州大学大学院システム情報科学研究院助手．2005 年岡山大学大学院自然科学研究科助教授．現在，同准教授．博士（工学）．オペレーティングシステム，コンピュータセキュリティに興味を持つ．2010 年度 JIP Outstanding Paper Award，2012 年度情報処理学会論文賞各受賞．電子情報通信学会，ACM，USENIX，IEEE 各会員．

横山 和俊　（正会員）

1988 年広島大学工学部第二類（電気系）卒業．1990 年同大学大学院工学研究科博士課程前期修了．2006 年岡山大学大学院自然科学研究科博士後期課程修了．1990 年 NTT データ通信株式会社（現，株式会社 NTT データ）入社後，オペレーティングシステム，分散処理の研究開発に従事．2012 年高知工科大情報学群教授．博士 (工学)．電子情報通信学会会員．

乃村 能成　（正会員）

1995 年九州大学工学部電子工学科卒業．1997 年同大学大学院情報工学専攻修士課程修了．同年九州大学工学部助手を経て，現在，岡山大学大学院自然科学研究科准教授．博士（情報科学）．

谷口 秀夫　（フェロー）

1978 年九州大学工学部電子工学科卒業．1980 年同大学大学院修士課程修了．同年日本電信電話公社電気通信研究所入所．1987 年同所主任研究員．1988 年 NTT データ通信株式会社開発本部移籍．1992 年同本部主幹技師．1993 年九州大学工学部助教授．2003 年岡山大学工学部教授．2010 年岡山大学工学部長．2014 年岡山大学理事・副学長．博士（工学）．オペレーティングシステム，実時間処理，分散処理に興味を持つ．著書『並列分散処理』（コロナ社）等．電子情報通信学会，ACM 各会員．

芦塚 正雄

2012 年岡山大学工学部情報工学科卒業．2014 年同大学大学院自然科学研究科博士前期課程修了．オペレーティングシステムに興味を持つ．