2008－EVA－27 (3)
2008／12／1

ファイルの使用頻度に基づくバッファキャッシュ制御法の評価

片上達也† 田端利宏† 谷口秀夫†

我々は，応用プログラムのファイルの使用頻度に基づき，優先して保持するブロックを決定するバッファキャッシュ制御法を提案した．ここでは，提案手法を評価し，評価結果から提案手法の有効性を示す．具体的には，LRU 方式でキャッシュヒット率を低下させる 2 つの事例について評価した．評価の結果，提案手法は，LRU 方式と比較して，性能を向上させることを明らかにした．また，実 AP の例として，カーネルの make 処理において提案手法を評価し，処理時間を最大 22.4 秒（7.1 %）短縮できることを明らかにした．

Evaluation of I/O Buffer Cache Mechanism Based on the Frequency of File Usage

Tatsuya KATAKAMI,† Toshihiro TABATA† and Hideo TANIGUCHI†

We proposed an I/O buffer cache mechanism based on the frequency of file usage. This paper describes an evaluation of the proposed mechanism and effectiveness of the proposed mechanism from evaluation results. In particular, we evaluated two examples to decrease a cache hit rate by LRU algorithm. The results show that the proposed mechanism improves the performance as compared to the LRU algorithm. And we evaluated the proposed mechanism by kernel make processing, and the proposed mechanism improves the processing time by 22.4 s (7.1 %) as compared to the LRU algorithm.

1. はじめに

既存の多くのオペレーティングシステム（以降，OS と略す）は，バッファキャッシュを制御する単位をブロックとし，置き換えアルゴリズムとして，LRU 方式を用いている．LRU 方式は，キャッシュヒット率が高く，効率が良いことが知られている．

一方，LRU 方式には，特定の読み込みパターンに対して，著しくキャッシュヒット率を低下させるという問題がある．LRU 方式においてキャッシュヒット率を低下させる読み込みパターンは，大きく 2 つある．1 つ目は，バッファキャッシュの上限を上回る数のブロックに対するシーケンシャルな読み込みである．この読み込みが発生すると，ブロックは，バッファキャッシュから解放される．これにより，キャッシュヒット率は著しく低下し，システム全体でのキャッシュヒット率も低下する．

2 つ目は，バッファキャッシュに格納できる上限ブロック数を超えるブロック群に対するループ読み込みである．ループ読み込みを行うと，ループで読み込まれたブロックは，再び読み込まれる前にバッファキャッシュから解放される．このため，ループ読み込みは，ブロックを再利用する頻度の高い処理であるにも関わらず，LRU 方式ではキャッシュヒット率を著しく低下させる可能性がある．

これらの問題に対して，様々なブロック置き換えアルゴリズムが提案されてきた[1)−3)]．しかし，提案された方式は，ブロックの参照順序，参照頻度，参照間の間隔，あらかじめ用意された参照パターン，およびブロックの読み込みの規則性を意識するものの，応用プログラム（以降，AP と略す）の入出力要求単位であるファイルを意識しない．

一方，AP は，ファイルを単位として入出力の要求を行う．また，AP 毎に利用するファイルは異なり，ファイル毎に参照頻度や入出力の傾向も異なる．これに対して，我々は，文献情報の 4) で，AP が操作したファイルの情報（以降，ファイル操作情報と略す）を収集して，ファイル毎に重要度を決定し，決定した重要度に基づきブロック単位でバッファキャッシュを制御す

† 岡山大学大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University

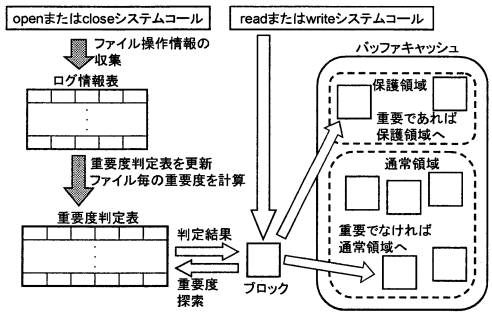

図 1 基本方式

る手法を提案した．

本論文では，提案手法について，LRU 方式で性能を低下させる 2 つの事例について基本評価を行った結果した結果を示す．また，実 AP の例として，カーネルの make 処理について評価した結果を示す．

2. ファイルの使用頻度に基づくバッファキャッシュ制御法

2.1 基本方式

提案手法では，ファイル操作情報を収集して，ファイル毎に重要度を決定し，決定した重要度に基づきブロック単位でバッファキャッシュを制御する．

提案手法の基本方式を図 1 に示し，以下に述べる．提案手法には，大きく分けて二つの処理の流れがある．

(1) ファイル操作情報として，open と close システムコールに着目し，ファイル操作情報をログ情報表に収集する．

(2) ログ情報が一定量たまると，ログ情報表を基にファイル毎に重要度を計算し，重要度判定表を更新する．

(3) read または write システムコールが発行され，ブロック読み込み要求が発生すると，重要度を基にバッファキャッシュの制御を行う．

重要度が高いファイルのブロックを格納する保護領域と，重要度が低いファイルのブロックを格納する通常領域にバッファキャッシュを分割して管理する．バッファキャッシュ制御の具体的な処理の流れを以下に述べる．

(1) read または write システムコールが発行され，新規ブロック読み込み要求が発生した場合，通常領域から LRU 方式で置き換え対象のブロックを選択する．通常領域にブロックがなければ，保護領域内で最も重要度の低いファイルを構成するブロックをすべて通常領域に移し，通常領域から LRU 方式で置き換え対象のブロックを選択する．

(2) ブロックの読み込みが終了すると，重要度を基に，読み込んだブロックのファイルが重要であるか判定し，重要であれば保護領域へ，重要でなければ通常領域へ格納する．

2.2 ファイルの重要度決定処理

2.2.1 重要度の決定規則

文献 5) で述べた重要度の目標を達成するため，各ファイルの重要度計算には，ファイル毎に参照数，オープン回数，およびファイルサイズを利用する．

参照数とは，現在ファイルをオープンしているプロセスの数である．オープン回数とは，ファイルをオープンするシステムコールの発行回数である．ファイルサイズとは，オープンまたはクローズしたファイルの大きさである．

上記の 3 つの情報のうち，参照数は現在ファイルがオープンされている状態を示すため，過去にオープンした履歴であるオープン回数と比べて，参照数の大きいファイルは今後読み書きされる可能性が高いと推察できる．ファイルサイズは，バッファキャッシュを制御する観点から重要度計算に利用しているため，今後ファイルを読み書きされる可能性に対して直接的に影響しない．

したがって，今後読み書きが行われる可能性を考慮し，重要度計算に与える影響を以下のようにする．

（参照数）＞（オープン回数）＞（ファイルサイズ）

2.2.2 収集するファイル操作情報

ファイル操作情報には，重要度決定に必要な情報である参照数，オープン回数，およびファイルサイズを格納する．これらに加えて，ファイルを特定するために，i ノード番号を用いる．また，2 つのファイルの重要度が等しい場合に順位を付けるため，時刻が必要である．提案手法では重要度決定にオープン回数を用いているため，オープン回数に関連した最新オープン時刻を用いる．

2.2.3 重要度判定表の更新方法

＜重要度判定表を更新する契機＞

判定表更新契機は，ファイルオープンまたはクローズの操作回数に連動するのが好ましい．このため，前回の重要度判定表の更新から，一定回数オープンまたはクローズが行われたときを重要度判定表の更新契機とする．

また，一定回数のオープンまたはクローズがある前に，ログ情報表があふれることがあるため，ログ情報表に一定数のファイル操作情報が蓄積されたときを契機とする．

＜重要度判定表を更新する方法＞

ファイル操作情報のうち，オープン回数は，単純に加算すると，古い情報と新しい情報が等しく扱われる．そこで，最近のオープン操作をより大きく重要度に反映させるため，古い情報ほど影響を小さくする．具体的には，重要度判定表を更新するときに，古い情報ほど影響を小さくする重み w をかけることで，古い情報の影響を小さくする．

2.3 期待される効果

(1) AP の入出力性能の向上
(2) 大きいファイルによるサービスへの影響抑制
(3) バックアップによるサービスへの影響抑制

3. 評　　価

3.1 目的と条件

3.1.1 評価の目的

本章では，1 章で述べた LRU 方式でキャッシュヒット率を低下させる 2 つの読み込みパターンについて，提案手法を評価し，提案手法の有効性を示す．また，LRU でキャッシュヒット率が低下する事例となる AP での提案手法の効果を明らかにする．

3.1.2 評価環境

提案手法を FreeBSD 4.3-RELEASE において実装し，計算機（CPU:Pentium4(1.95 GHz)，メモリ:512 MB，バッファキャッシュ:3.0MB）評価を行った．

なお，提案手法の効果を把握しやすくするため，バッファキャッシュ内に該当するブロックが見つからなかった場合に，実メモリ上に該当するページがあるかどうかを探索する機能（VMIO 機能）を無効にして測定した．

3.1.3 評価において決定する計算式と閾値

2.2.1 項で述べた重要度の決定規則に基づき，各ファイルの重要度の計算式を式 (1) とした．式 (1) において，I は重要度，N_{open} はオープン回数，F_{num} は参照数，k は定数，F_{size} はファイルサイズ，B_{size} は 1 ブロックのサイズである．

$$I = \frac{N_{open} + F_{num} \times k}{[\frac{F_{size}}{B_{size}}] + 1} \qquad (1)$$

式 (1) では，オープン回数と参照数の重要度計算への影響の違いを考慮し，定数 k を導入した．本評価では，参照数 F_{num} の重みを上げるため，$k = 10$ とした．また，1 ファイルあたりのブロック数が増加するほど，バッファキャッシュを占有する．そこで，1 ファイルあたりのブロック数の逆数をかける．

2.2.3 項で述べた重み関数 w を式 (2) とした．式 (2) において，N_{open} はオープン回数，Dec は重みである．なお，式 (2) は，参照頻度を利用する手法である LRFU[2] の重み付けと等しく，提案手法においても，この重み付けが有効に働くと考えられる．

$$w(N_{open}) = N_{open} \times Dec \qquad (2)$$

評価における各閾値を以下に示す．

(1) R_c：重要度を計算した直後から open または close システムコールが発行された回数の総和が R_c になると，重要度を再計算（重要度再計算契機）
(2) F_v：重要度に基づき重要ファイルと判定するファイルの数（重要度ファイル数）
(3) F_{max}：重要度判定の対象となるファイル数の最大値
(4) U_d：重要度判定表の更新時に，オープン回数が U_d 以下のものをまとめて重要度判定表から削除

3.2 基本評価

3.2.1 評価項目

提案手法の有効性について，以下の観点から評価した．

評価 1 キャッシュのサイズより大きいファイルの読み込みの影響
評価 2 キャッシュの上限ブロック数を上回る数のファイルのループ読み込みの影響

3.2.2 キャッシュのサイズより大きいファイルの読み込みの影響

＜評価内容＞

2 種類 5 つのファイル（A(4 KB)，B(3 MB)，C(3 MB)，D(3 MB)，E(3 MB)）を用意し，各ファイルについて，open，n 回の read，および close の処理を行う．このとき，read のサイズは 4 KB である．したがって，A は 1 回，B〜E は，769 回の read 処理が行われる．なお，ファイルのアクセス順序は，A → B → A → C → A → D → A → E であり，これを 25 回繰り返す．これにより，ファイル A については，open，close が 25 回，read が 100 回発行される．また，ファイル B，C，D，および E については，open，close が 25 回，read が 19225 回発行される．バッファキャッシュサイズが 3MB であるため，LRU 方式の場合，ファイル A の read のキャッシュヒット率は 0 となる．これに対し，提案手法の場合，open の回数を考慮するため，ファイル A の read において，キャッシュヒット率が向上すると期待できる．

＜評価結果＞

評価 1 におけるキャッシュヒット率を図 2 に，処理時間を図 3 に示す．本評価では，F_v を 1，F_{max} を 10 個，U_d を 0 とした．図 2 と図 3 より，以下のことがわかる．

(1) バッファキャッシュの上限を上回るサイズのファ

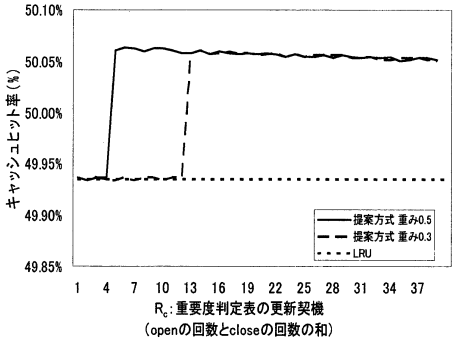

図 2 評価 1 におけるキャッシュヒット率

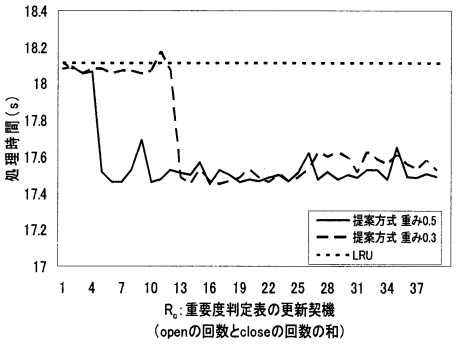

図 3 評価 1 における処理時間

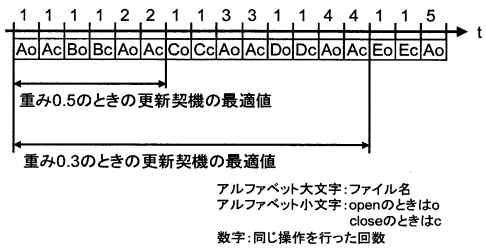

図 4 評価 1 における重要度判定表の更新契機

イルに対する読み込みに対して，提案手法は，LRU 方式と比較して，重みが 0.5，R_c が 5 のときに，キャッシュヒット率を 0.13 %向上させ，処理時間を 0.65 秒 (4.4 %) 短縮させる．これは，ファイル A を優先的にバッファキャッシュに保持することによって，ファイル A のキャッシュヒット率が向上したことによる．

(2) 提案手法では，重みが 0.5 の場合，R_c が 6 のときに最もキャッシュヒット率を向上させる．また，重みが 0.3 の場合，R_c が 14 のときに最もキャッシュヒット率を向上させる．これは次の理由による．評価 1 において，キャッシュヒット率を最大にする重要度判定表の更新契機を重み毎に分類したものを図 4 に示す．保護すべきファイルである A は，ファイルシステム全体で open と close システムコールが合計 4 回発行される度に open される．提案手法では，重みが 0.5 の場合，A の 2 度目の close システムコール発行時に，重みが 0.3 の場合，A の 4 度目の close システムコール発行時に，それぞれ最も高いキャッシュヒット率となっている．提案手法では，オープン回数を整数値として扱う．このため，重みをかけてオープン回数が 1 未満になった場合，オープン回数は，切り捨てによって 0 となる．これにより，閾値 U_d の条件を満たすため，該当するファイル操作情報は重要度判定表から削除される．また，バッファキャッシュ制御に重要度判定表の内容をできるだけ早く反映させた方がキャッシュヒット率が向上する．これより，重要度判定表の更新契機に関して，重みをかけて，重要なファイルのファイル操作情報が削除されない最小の値がキャッシュヒット率を最大にする値となる．

3.2.3 キャッシュの上限ブロック数を上回る数のファイルのループ読み込みの影響

＜評価内容＞

8 KB のファイル 500 個を用意し，各ファイルについて open，3 回の read，close の処理を行う．このとき，read のサイズは 4 KB である．read システムコール発行の終了条件となる 0 バイトの読み込みを含めると，全てのファイルに対して 3 回の read 処理が行われる．なお，500 個のファイルが 1 回ずつ読み込まれることをループの単位とし，このループを 100 回繰り返す．これにより，各ファイルについて，open/close が 100 回，read が 300 回発行される．

これに対し，提案手法の場合，保護領域の大きさをバッファキャッシュの上限に設定することで，このループで用いられるファイルのブロックをバッファキャッシュの上限分保護し，キャッシュヒット率を向上させることが期待できる．

＜評価結果＞

評価 2 におけるキャッシュヒット率を図 5 に，処理時間を図 6 に示す．F_v をバッファキャッシュの上限である 170，F_{max} を 520 個，U_d を 0，重みを 0.5 とした．図 5 と図 6 より，以下のことがわかる．

(1) ループで読み込むブロックの数がバッファキャッシュの上限個数を上回るため，LRU 方式では，各ファイルのキャッシュヒット率が 66.7%となる．ここで，キャッシュヒット率が 0 とならないのは，FreeBSD 4.3-RELEASE に先読み機能があり，各ファイルにおける 3 回の read のうち，2 回についてキャッシュヒッ

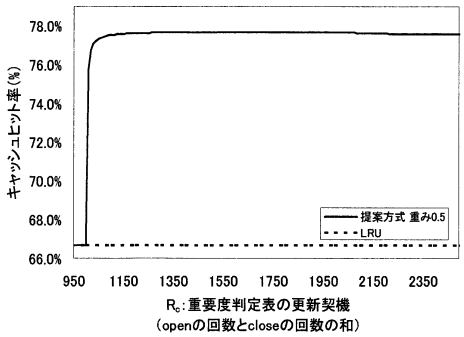

図 5 評価 2 におけるキャッシュヒット率

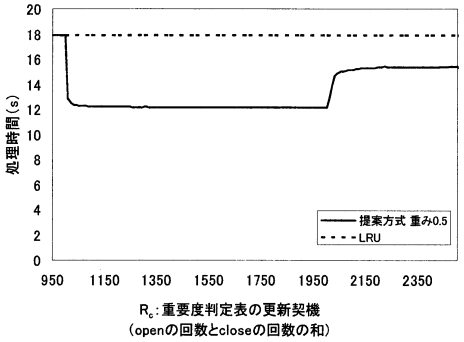

図 6 評価 2 における処理時間

トするためである．

(2) 図 5 より，提案手法は，LRU 方式と比較して，R_c が 1,340 のときに，キャッシュヒット率を約 11 %向上させ，処理時間を約 5.7 秒（31.9 %）短縮させることがわかる．これは，提案手法が，ループで読み込まれるファイルの一部を保護することにより，保護したファイルに関してキャッシュヒット率が向上したためである．

(3) R_c がループの周期よりも小さい場合，ループの全てが重要度判定表に反映される前に，重要度判定表から削除されるため，キャッシュヒット率が低い．また，R_c がループの周期よりも大きく，一定値未満の場合，最も高いキャッシュヒット率となる．さらに，R_c が一定値以上になると，キャッシュヒット率は低下し，処理時間は増加する．これは，次の理由による．保護したいファイルのブロックを保護するには，重要度判定表の更新の後に，最低一回読み込む必要がある．このとき，ループ読み込みの性質より，n 回目のループの始まりまでに重要度判定表を更新できなければ，重要度判定表の内容がキャッシュ制御に反映されるのは，n+1 回目のループ以降となり，このループ読み込みの

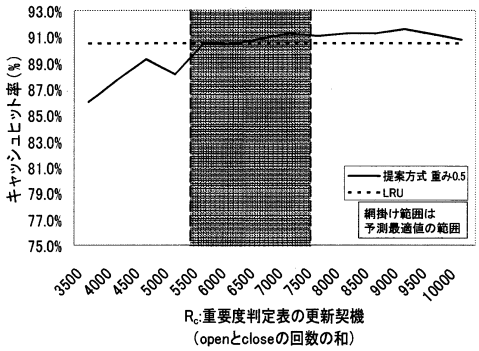

図 7 カーネルの make 処理におけるキャッシュヒット率

分だけキャッシュヒット率が低下する．

上記の 2 つの評価によって，U_d が 0 であるとき，単純なループ処理において，最もキャッシュヒット率を向上させる重要度判定表の更新契機は，重要なファイルのファイル操作情報が重みによって削除されず，かつ重要度判定表の内容をできる限り早く反映させることのできる値である必要がある．これにより，式 (3) が確からしいということがわかる．式 (3) において，R_{max} は，最もキャッシュヒット率を向上させる重要度判定表の更新契機，U_c は，ループの周期（ループ中に open と close システムコールが発行された回数），w は重み，F_p は，保護したいファイルの数を示す．

$$U_c[1/w] + 2F_p \leq R_{max} \leq U_c([1/w] + 1) \quad (3)$$

3.3 実 AP による評価

3.3.1 評価項目

LRU でキャッシュヒット率を低下させる実 AP において，提案手法の性能を評価するため，以下の評価を行った．カーネルの make 処理は，キャッシュサイズを上回るループ読み込みを行う処理であり，提案手法によってキャッシュヒット率を向上させることが期待できる．

(1) カーネルの make 処理

(2) バックアップ処理中のカーネルの make 処理

3.3.2 評価方法と結果

＜カーネルの make 処理＞

カーネルの make 処理のキャッシュヒット率と処理時間を測定し，評価する．また，カーネルの make 処理において，重要度判定表の更新契機の予測が正しいことを示す．

カーネルの make 処理におけるキャッシュヒット率を図 7 に，処理時間を図 8 に示す．カーネルの make 処理では，約 4,500 個のファイルが利用されるため，F_{max} は，5,000 とした．また，F_v は，100 とした．こ

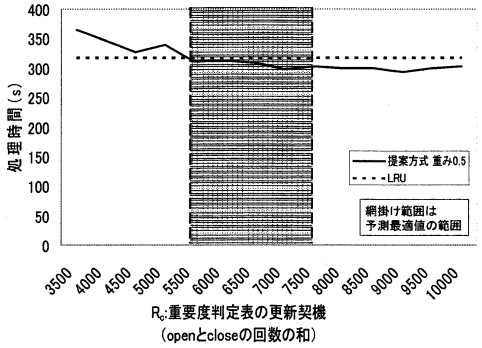

図 8 カーネルの make 処理における処理時間

れは，カーネルの make 処理全体において，重要度の高いファイルの上位からブロック数を加算したときに，バッファキャッシュの上限に達するファイル数である．U_d は 0，重みは 0.5 とした．図 7 において，網掛けの範囲は，カーネルの make 処理の読み込みパターンを解析し，式 (3) によって算出したキャッシュヒット率を最大にする重要度判定表の更新契機の予測値である．

図 7 と図 8 より，以下のことがわかる．

(1) 提案手法は，R_c が 9,000 のときに，LRU 方式と比較して，キャッシュヒット率を 1.1 %向上させ，処理時間を 22.4 秒（7.1 %）短縮させることがわかる．

(2) キャッシュヒット率と処理時間を最も短縮した R_c は，キャッシュヒット率を最大にする重要度判定表の更新契機の予測値よりも大きい．これは，基本評価とは異なり，カーネルの make 処理では，複数のループが組み合わさり，複雑な読み込みパターンを形成していることによる．よって，カーネルの make 処理において，提案手法の効果を最大限に発揮するには，予測値よりも長期間のファイル操作情報を収集して，重要度の精度を高める必要がある．

＜バックアップ処理中のカーネルの make 処理＞

実環境を想定し，バックグラウンドで，/usr/src/以下にあるデータを rsync で他計算機にバックアップしながら，カーネルの make 処理を行ったときのカーネルの make 処理の処理時間とキャッシュヒット率を測定し，LRU 方式と比較した．カーネルの make 処理の処理時間を測定する計算機は，基本評価で用いたものと同じ計算機である．また，カーネルの make 処理を行う計算機とバックアップデータを保存する計算機は，100Mbps の LAN によって接続されている．バックアップデータを保存する計算機の構成について，OS は FreeBSD 4.3-RELEASE，CPU は Celeron (2.8 GHz)，メモリは 768 MB，バッファキャッシュ制御は LRU 方式である．

表 1 バックアップ処理中のカーネルの make 処理の性能

方式	キャッシュヒット率 (%)	処理時間 (s)
LRU	87.6	427.2
提案手法	88.5	406.2

各閾値はカーネルの make 処理の評価と同じとする．また，R_c は，カーネルの make 処理の評価で最も処理時間を短縮した 9,000 とする．

評価結果を表 1 に示す．表 1 より，提案手法は，LRU と比較して，キャッシュヒット率を 0.9 %，処理時間を 21.0 秒（4.9 %）短縮させることがわかる．これは，提案手法では，カーネルの make 処理で利用されているブロックがバッファキャッシュに優先的に保持されているからである．これにより，提案手法では，各ファイルを一度ずつ連続で読み込む処理の影響を抑制できる．

4. おわりに

ファイルの使用頻度に基づくバッファキャッシュ制御法の評価について述べた．LRU 方式でキャッシュヒット率を低下させる事例として，キャッシュのサイズより大きいファイルの読み込みの影響と，キャッシュの上限ブロック数を上回る数のファイルのループ読み込みの影響を評価した．評価の結果，提案手法は，従来方式（LRU 方式）と比較して，処理時間を最大 5.7 秒 (31.9 %) 短縮できることを明らかにした．

また，LRU で性能を低下させる実 AP の例として，カーネルの make 処理について評価し，処理時間を最大 22.4 秒（7.1 %）短縮できることを明らかにした．

謝辞 本研究の一部は，科学研究費補助金 基盤研究 (B)(課題番号：18300010) による．

参考文献

1) E.J. O'Neil, P.E. O'Neil, and G. Weikum, "The LRU-k Page Replacement Algorithm for Database Disk Buffering," Proc. the 1993 ACM SIGMOD Conference, pp.297–306, 1993.
2) D. Lee, J. Choi, J.H. Kim, S.H. Noh, S.L. Min, Y. Cho, C.S. Kim, "LRFU (Least Recently/Frequently Used) Replacement Policy: A Spectrum of Block Replacement Policies," IEEE Transactions on Computers, Vol. 50, No 12, 1352–1360, 1996.
3) H. Choo, Y. J. Lee, and S. Yoo, "DIG: Degree of Inter-Reference Gap For a Dynamic Buffer Cache Management," Inf. Sci., pp.1032–1044, 2006.
4) 片上達也，田端利宏，谷口秀夫，渡辺弘志，乃村能成，"ファイルの使用頻度に基づくバッファキャッシュ制御法，" 情報処理学会研究報告 2008-OS-108，Vol.2008，No.35，pp.115–122 (2008).