情報処理学会論文誌 Vol.57 No.6 1539–1553 (June 2016)

キャッシュヒット率に着目した入出力バッファの分割法の実現と評価

土谷 彰義[1,a)] 山内 利宏[1] 谷口 秀夫[1]

受付日 2015年6月7日, 採録日 2016年3月4日

概要：利用者が優先したい処理のファイルアクセスを高速化するには，その際の入出力バッファのキャッシュヒット率を向上させることが有効である．そこで，入出力バッファを 2 つの入出力バッファ領域に分割し，指定されたディレクトリ直下のファイルを優先的にキャッシュするディレクトリ優先方式が提案されている．しかし，この方式は，優先的にキャッシュするファイルの入出力バッファ領域サイズを単調増加させる．このため，他方の入出力バッファ領域サイズが単調減少し，そのキャッシュヒット率が大きく低下し，計算機全体の性能低下を招いてしまう問題がある．そこで，本論文では，各入出力バッファ領域のキャッシュヒット率に着目した入出力バッファ分割法を提案する．具体的には，優先的にキャッシュするファイルのキャッシュヒット率を高く維持できる範囲で，そのほかのファイルのキャッシュヒット率の低下を抑制するように入出力バッファを分割する．提案方式を実現し，評価により，その有効性を示す．

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

Implementation and Evaluation of Partitioning Method of I/O Buffer Based on Cache Hit Ratio

Akiyoshi Tsuchiya[1,a)] Toshihiro Yamauchi[1] Hideo Taniguchi[1]

Received: June 7, 2015, Accepted: March 4, 2016

Abstract: In order to improve file access performance of the processing that users would like to execute at high priority, it is effective to improve cache hit ratio of I/O buffer. Thus, a directory oriented buffer cache mechanism was proposed. This mechanism divides I/O buffer into two areas, and gives higher caching priority to files in specified directories. However, this mechanism monotonically expands the area used for caching files given higher priority. Therefore, this mechanism declines the performance of the whole computer due to deterioration of cache hit ratio of files not given higher priority. Thus, this paper proposes the dynamic partitioning method based on cache hit ratio. The proposed method divides I/O buffer to maintain high cache hit ratio of files given higher priority and prevent cache hit ratio of the other files from degradation. Additionally, this paper describes the evaluation of effectivity of the proposed method.

Keywords: I/O buffer, buffer cache, operating system, directory, file

1. はじめに

計算機で実行される処理には，利用者が優先したい処理（以降，優先処理と略す）とそうでない処理（以降，非優先処理と略す）がある．優先処理のファイルアクセスを高速化するには，その際の入出力バッファのキャッシュヒット率を向上させることが有効である．

そこで，ディレクトリ優先方式 [1] を提案した．この方式は，入出力バッファを 2 つの入出力バッファ領域に分割し，指定したディレクトリ（以降，優先ディレクトリと略す）直下のファイル（以降，優先ファイルと略す）と他のファイル（以降，非優先ファイルと略す）を別々の入出力バッファ領域にキャッシュし，前者を優先的に残す．これにより，優先処理が頻繁にアクセスするファイルを直下に有するディレクトリを優先ディレクトリに指定することに

[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
[a)] gen422415@s.okayama.u.ac.jp

より，優先処理のファイルアクセスを高速化でき，実行処理時間を短縮できる．

しかし，この方式は，優先ファイルのキャッシュヒット率を向上させるために，優先ファイルをキャッシュする入出力バッファ領域のサイズを単調増加させる．このため，非優先ファイルをキャッシュする入出力バッファ領域のサイズは単調減少し，そのキャッシュヒット率は低下する．このために，次の 2 つの問題がある．

1 つ目は，非優先ファイルにアクセスする優先処理の実行処理時間が増加する問題である．この問題は，優先処理がアクセスするファイルが多くのディレクトリに分散している場合に発生する．ディレクトリ優先方式では，制御オーバヘッドの抑制のため，優先ディレクトリ数が少ないことが好ましい．また，優先処理と非優先処理がアクセスするファイルが混在するディレクトリは，優先ディレクトリとしないことが好ましい [2]．このため，優先処理がアクセスするファイルを直下に有するディレクトリのすべてを優先ディレクトリに設定することはできない．したがって，優先処理が非優先ファイルにもアクセスすることになり，非優先ファイルのキャッシュヒット率が低いことにより，ファイルのアクセス時間が増加し，優先処理の実行処理時間が増加する．2 つ目は，非優先ファイルにアクセスする非優先処理の実行処理時間が増加する問題である．この問題は，優先処理と非優先処理が共存する場合に発生する．これら 2 つの問題を解決するためには，優先ファイルのキャッシュヒット率を高く維持しつつ，非優先ファイルのキャッシュヒット率の低下を抑制する必要がある．

そこで，本論文では，ディレクトリ優先方式において，キャッシュヒット率に着目した入出力バッファ分割法を提案する．提案方式は，分割した各入出力バッファ領域のキャッシュヒット率に着目し，優先ファイルのキャッシュヒット率を高く維持できる範囲で，非優先ファイルのキャッシュヒット率の低下を抑制するように，入出力バッファを分割する．また，提案方式をカーネル make，Web サーバ，および gnuplot において評価した結果を報告する．

2. ディレクトリ優先方式

2.1 基本方式

ディレクトリ優先方式は，以下の 2 つの着目点に基づく方式である．

（着目点 1） 様々な処理が同時に実行される場合，扱うデータに応じてその処理優先度が異なり，特定の処理を優先したい場合が存在

たとえば，Web サーバにおいて特定の Web ページの応答時間を短縮したい，バックアップ処理などのバックグラウンドで実行される処理による性能影響を抑えたいという要求である．

（着目点 2） ユーザが今から実行する重要な処理に必要な特定のファイル群を優先的にキャッシュしたいという要求が存在

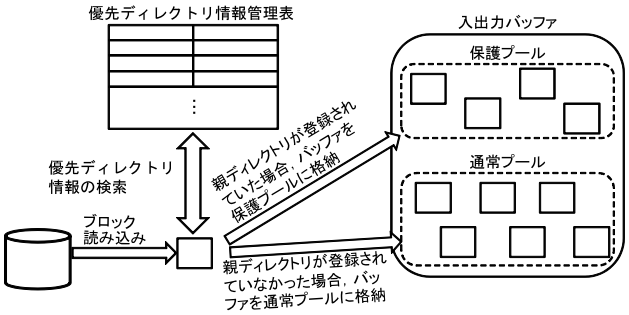

図 1 ディレクトリ優先方式の基本方式
Fig. 1 Directory oriented buffer cache mechanism.

LRU 方式では，他の処理が使用したファイル群によって，今から実行する重要な処理が使用するファイル群のブロックが入出力バッファから解放されることがある．この場合，重要な仕事が以前に使用したファイルを再度使用する際に，再度ディスク I/O を行わなければならない．もし，重要な仕事に必要なファイル群のブロックを入出力バッファに残すことができていれば，重要な仕事のディスク I/O の削減が期待できる．また，ファイルアクセスが多い処理に対しては，アクセスするファイル群のうち，アクセス頻度が高いファイル群を残すことにより，ディスク I/O をより削減できると考えられる．

文献 [1] のディレクトリ優先方式の基本方式（以降，基本方式と略す）を図 **1** に示す．ディレクトリ優先方式は，入出力バッファを保護プールと通常プールに分割し，各プール内のバッファを LRU 方式で管理する．保護プールには優先ファイルのブロックを保持するバッファを格納し，通常プールには非優先ファイルのブロックを保持するバッファを格納する．ここで，バッファとは，1 ブロックをキャッシュするためのメモリ領域である．入出力バッファは，このバッファを複数持ち，リスト構造で管理している．また，ディレクトリ優先方式は，保護プールが拡大すると拡大分だけ通常プールが小さくなり，同様に通常プールが拡大すると拡大分だけ保護プールが小さくなる．

ブロックアクセス時，通常プール内にバッファが存在する限り通常プールからバッファを解放し，空きバッファを確保する．空きバッファにブロックを読み込んだ後，読み込んだブロックに対応するファイルが優先ファイルであれば保護プールに，非優先ファイルであれば通常プールにバッファを格納する．

2.2 問題点

基本方式は，以下の（要求 1）を満足するように設計されている．

（要求 1） 優先ファイルへのアクセス時間を短縮

このため，基本方式は，優先ファイルのブロックをできる

だけ多くキャッシュするように設計されており，保護プールのサイズが単調増加する．これにともない，通常プールのサイズが単調減少し，非優先ファイルのキャッシュヒット率が低下する．したがって，以下の 2 つの問題がある．

1 つ目は，非優先ファイルにアクセスする優先処理の実行処理時間が増加する問題である．この問題は，優先処理がアクセスするファイルが多くのディレクトリに分散している場合に発生する．ディレクトリ優先方式では，制御オーバヘッドの抑制のため，優先ディレクトリ数が少ないことが好ましい．また，優先処理と非優先処理がアクセスするファイルが混在するディレクトリは，優先ディレクトリとしないことが好ましい [2]．このため，優先処理がアクセスするファイルを直下に有するディレクトリのすべてを優先ディレクトリに設定することはできない．したがって，優先処理は非優先ファイルにもアクセスすることになる．このとき，優先ファイルの総サイズが入出力バッファサイズ以上となる設定である場合，優先処理がアクセスする非優先ファイルのキャッシュヒット率が低くなる．この結果，非優先ファイルへのアクセス時間の増加により，優先処理の実行処理時間が増加する．

2 つ目は，非優先処理の実行処理時間が増加する問題である．この問題は，優先処理と非優先処理が共存する場合に発生する．非優先処理は，非優先ファイルに多くアクセスする．このため，非優先ファイルへのアクセス時間の増加により，非優先処理の実行処理時間が増加する．

3. キャッシュヒット率に着目した入出力バッファ分割法

3.1 目的

2.2 節で述べた問題を解決するため，（要求 1）に加え，次の要求も満足するバッファキャッシュ制御法を提案する．

（要求 2） 優先ファイルと非優先ファイルにアクセスする優先処理の実行処理時間の短縮，もしくは増加の抑制

（要求 2）を満足することで，優先ファイルの総サイズが入出力バッファサイズ以上となる優先ディレクトリ設定であっても，基本方式と比べて実行処理時間の増加を抑える．

さらに，（要求 1）と（要求 2）を満足したうえで，可能な範囲で次の要望にも対応する．

（要望 1） 非優先ファイルにアクセスする非優先処理の実行処理時間の増加を抑制

（要求 1）と（要求 2）を満足するためには，保護プールが優先ファイルのブロックを十分にキャッシュできているか把握し，優先ファイルのキャッシュヒット率を高く維持する必要がある．また，（要求 2）の満足，および（要望 1）への対応のためには，通常プールについても同様に把握し，優先ファイルのキャッシュヒット率を高く維持できる範囲で，通常プールの領域を確保し，非優先ファイルのキャッシュヒット率の低下を抑制する必要がある．

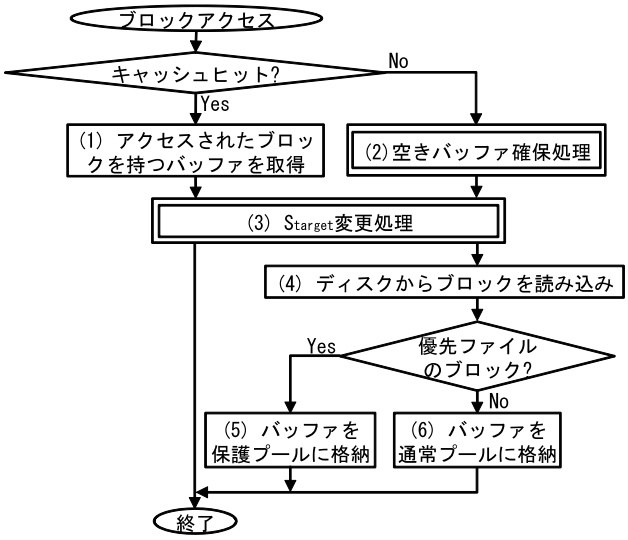

図 2 入出力バッファ分割法に基づく処理の流れ

Fig. 2 Processing flow of the proposed partitioning method of I/O buffer.

3.2 基本的な処理の流れ

入出力バッファ分割法として，各プールのキャッシュヒット率に着目し，保護プールのサイズ（S_{target}）を決定する方式を提案する．保護プールのキャッシュヒット率が低い場合，S_{target} を大きくする．また，保護プールのキャッシュヒット率が高く，かつ通常プールのキャッシュヒット率が低い場合，S_{target} を小さくし，通常プールを大きくする．これにより，保護プールのキャッシュヒット率を高く維持できる範囲で，通常プールの領域を確保し，非優先ファイルのキャッシュヒット率の低下を抑制する．なお，S_{target} の単位は，バッファ数である．

S_{target} に基づく制御の方針として，以下の 2 つがある．

（方針 1） 動的な S_{target} の変更

保護プールと通常プールの両プールのキャッシュヒット率を監視し，キャッシュヒット率が低くなった際に，S_{target} を変更する．S_{target} は，利用者により設定されたパラメータに基づき決定する．このとき，保護プールのキャッシュヒット率の向上を優先する．

（方針 2） キャッシュしているブロックの有効利用

S_{target} を変更すると，実際の保護プールのサイズ（S_{cur}）と S_{target} が一致しなくなる．ただちに S_{cur} と S_{target} を一致させるために，その差分のバッファを解放すると，解放後に，解放したバッファにアクセスがあった場合にキャッシュヒット率が低下する．このため，バッファをただちに解放せず，空きバッファ確保処理において徐々に差を縮める．

入出力バッファ分割法を適用したディレクトリ優先方式の処理の流れを**図 2** に示し，以下で述べる．なお，図 2 の内，2 重の四角で示す処理が，提案方式の実現のため，ディレクトリ優先方式の基本方式に対して追加，もしくは変更

する処理である．

(1) キャッシュヒットした場合，アクセスされたブロックを持つバッファを取得する．

(2) キャッシュミスした場合，空きバッファ確保処理を実行する．基本方式のこの処理は，通常プールにバッファが存在する限り，通常プールからバッファを解放し，空きバッファを確保する．一方，提案方式は，$S_{cur} = S_{target}$ を維持，または S_{cur} が S_{target} に近づくようにバッファを解放し，空きバッファを確保する．

(3) S_{target} 変更処理を実行する．この処理により，保護プールのキャッシュヒット率が低いならば，S_{target} を増加させる．そうでなく，保護プールのキャッシュヒット率が高く，かつ通常プールのキャッシュヒット率が低いならば，S_{target} を減少させる．

(4) キャッシュミスしていた場合，ディスクから (2) で確保した空きバッファにブロックを読み込む．

(5) 優先ファイルのブロックへのアクセスであれば保護プールにバッファを格納する．

(6) 非優先ファイルのブロックへのアクセスであれば通常プールにバッファを格納する．

3.3 課題と対処

3.3.1 課題

課題として以下の 3 つがある．

(1) (方針 1) の課題

(課題 1) 分割サイズ決定の契機

キャッシュヒット率の判定契機が課題となる．

(課題 2) 分割サイズ決定の方法

S_{target} の増減量の決定方法が課題となる．

(2) (方針 2) の課題

(課題 3) バッファの解放規則

$S_{cur} \neq S_{target}$ の場合，S_{cur} を S_{target} に近づけるように制御する方法が課題となる．

3.3.2 分割サイズ決定の契機

入出力バッファは，ブロックアクセスごとに参照される．このため，直近のブロックアクセスの傾向を反映させるため，一定回数（ω 回）のブロックアクセスごとに，式 (1) の処理を行う．

$$
\begin{aligned}
&\text{if}(H_{protected} < T_{protected})\{\\
&\quad S_{target}\text{を増加}\\
&\}\text{else if}(H_{normal} < T_{normal})\{\\
&\quad S_{target}\text{を減少}\\
&\}
\end{aligned}
\tag{1}
$$

式 (1) 中の $H_{protected}$ は保護プールのキャッシュヒット率，H_{normal} は通常プールのキャッシュヒット率，$T_{protected}$ は保護プールのキャッシュヒット率の閾値，T_{normal} は通常プールのキャッシュヒット率の閾値を示す．保護プールのキャッシュヒット率が $T_{protected}$ 未満である場合，S_{target} を増加させる．また，保護プールのキャッシュヒット率が $T_{protected}$ 以上，かつ通常プールのキャッシュヒット率が T_{normal} 未満である場合，S_{target} を減少させる．なお，ω 回のブロックアクセスのうちに，優先ファイル，もしくは非優先ファイルへのアクセスがなかった場合，それぞれ $H_{protected}$ や H_{normal} を 100%とする．

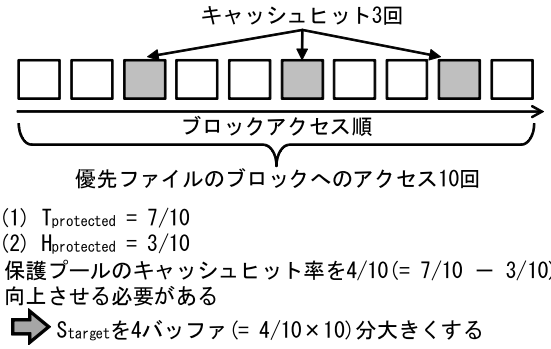

図 3 S_{target} の増加量決定の例

Fig. 3 Example of decision of increment of S_{target}.

3.3.3 分割サイズ決定の方法

各プールのキャッシュヒット率を閾値以上に維持することを目指す．このため，分割サイズ決定の方法は，2 つのプールのキャッシュヒット率が閾値以上となるのに不足したキャッシュヒット数分，S_{target} を増減させる．S_{target} の増加式を式 (2) に示す．

$$
\begin{aligned}
&S_{target} \ += (T_{protected} - H_{protected}) \times A_{protected}\\
&\text{if}((B - S_{target}) < B_{nmin})\{\\
&\quad S_{target} = B - B_{nmin}\\
&\}
\end{aligned}
\tag{2}
$$

また，S_{target} の減少式を式 (3) に示す．

$$
\begin{aligned}
&S_{target} \ -= (T_{normal} - H_{normal}) \times A_{normal}\\
&\text{if}(S_{target} < B_{pmin})\{\\
&\quad S_{target} = B_{pmin}\\
&\}
\end{aligned}
\tag{3}
$$

式 (2)，(3) 中の $A_{protected}$ は優先ファイルのブロックへのアクセス数，A_{normal} は非優先ファイルのブロックへのアクセス数，B は入出力バッファに保持できるバッファ数である．また，両プールで最低限必要なバッファを確保するため，保護プールのサイズの下限 B_{pmin} と通常プールのサイズの下限 B_{nmin} を導入する．B_{pmin} と B_{nmin} の単位は，S_{target} と同様に，バッファ数である．これにより，(保護プールのサイズ $\geq B_{pmin}$) と (通常プールのサイズ $\geq B_{nmin}$) を維持する．

S_{target} の増加量決定の例を**図 3** に示す．図 3 では，

表 1 パラメータの設定値
Table 1 Parameters setting for examination.

処理	パラメータ	設定値
カーネル make	ω	(1) 入出力バッファサイズ 3.0 MB の場合（バッファ数：296 個） 148，296，592，1,184，2,368 (2) 入出力バッファサイズ 6.3 MB の場合（バッファ数：720 個） 360，720，1,440，2,880，5,760，8,640，11,520
	$T_{protected}$	90%，95%，100%
	T_{normal}	80%，85%，90%，95%，100%
	B_{pmin}	B の 5%，10%，20%，30%，40%，50%
	B_{nmin}	B の 5%，10%，20%，30%，40%，50%
Web サーバとバックアップ	ω	734（入出力バッファに持てるバッファ数）
	$T_{protected}$	100%
	T_{normal}	50%，55%，60%，70%，80%，85%
	B_{pmin}	B の 50%，60%，70%
	B_{nmin}	32
gnuplot とカーネル make	ω	720（入出力バッファに持てるバッファ数）
	$T_{protected}$	100%
	T_{normal}	50%，55%，60%，65%，70%，75%，80%
	B_{pmin}	B の 40%，50%，60%，70%，75%，80%
	B_{nmin}	32

$T_{protected} = 7/10$，$H_{protected} = 3/10$ であるため，さらにキャッシュヒット率を 4/10 向上させる必要がある．このためには，さらに最低 4 個のブロックを多くキャッシュする必要があった．このため，S_{target} をバッファ 4 個分大きくする．S_{target} を減少させる場合は，H_{normal} と T_{normal} に基づき，S_{target} の減少量を決定する．

3.3.4 バッファの解放規則

以下の規則により，解放するバッファを決定し，空きバッファを確保するとともに，$S_{cur} = S_{target}$ を維持，もしくは $S_{cur} = S_{target}$ に近づける．これにより，S_{cur} と S_{target} の差を徐々に縮めることができ，（方針 2）を満たす．

(1) 以下の規則に従い，バッファを解放するプールを選択する．
  (a) $S_{cur} < S_{target}$ の場合
    S_{cur} を大きくし，S_{target} に近づけなければならないため，通常プールを選択する．
  (b) $S_{cur} = S_{target}$ の場合
    S_{cur} を変化させないため，読み込むブロックが優先ファイルのブロックか否かで，バッファを解放するプールを選択する．優先ファイルのブロックであれば保護プールを，非優先ファイルのブロックであれば通常プールを選択する．
  (c) $S_{cur} > S_{target}$ の場合
    S_{cur} を小さくし，S_{target} に近づけなければらないため，保護プールを選択する．
(2) (1) で選択したプール内に解放できるバッファが存在するか判定する．バッファが存在すれば選択したプールから，バッファが存在しなければ選択しなかったプールから LRU 方式で解放するバッファを決定する．

3.4 パラメータの設定指針

3.4.1 設定指針に影響を与える条件

提案方式を利用するためには，5 つのパラメータ（ω，$T_{protected}$，T_{normal}，B_{pmin}，B_{nmin}）を設定する必要がある．本節では各パラメータの設定指針を示す．

パラメータの特性を把握するため，5 つのパラメータを変化させ，測定した．この測定には，4 章で述べる 3 つの評価と同様の応用プログラム（以降，AP と略す）と測定環境を用いた．AP の選定理由は，4.2 節で述べる．各 AP において測定した各パラメータの設定値を**表 1** に示す．表 1 のとおり，パラメータは範囲を絞って測定した．各 AP におけるこの絞り込みの考え方を以下で述べる．

(1) カーネル make

カーネル make を優先処理として実行した．S_{target} を変更してその効果が現れるまでの時間には入出力バッファサイズの影響があると考え，ω は，入出力バッファサイズを基準に入出力バッファサイズの 1/2 倍から 8 倍もしくは 16 倍までを測定した．優先ファイルのキャッシュヒット率はできるだけ高いほうがよいため，$T_{protected}$ は，90%以上の値を複数測定した．通常プールのキャッシュヒット率は保護プールのキャッシュヒット率より低くてもよい可能性がある．また，カーネル make は，優先ファイルと非優先ファイルの両者にアクセスするため，非優先ファイルのキャッシュヒット率の低下を抑える必要がある．したがって，T_{normal} は，$T_{protected}$ より小さい 80%以上の値を複数測定した．B_{pmin} と B_{nmin} は，入出力バッファの半数以

下の値を複数測定した．

(2) Web サーバとバックアップ処理を同時実行

Web サーバが持つトップページを構成するファイルの要求に対する Web サーバの処理を優先処理，Web サーバの他の処理と Web コンテンツのファイルのバックアップ処理を非優先処理として実行した．ω と B_{nmin} は，カーネル make による測定により，それぞれ 3.4.2 項と 3.4.6 項で述べる特性が明らかであったため，単一の設定のみ測定した．また，$T_{protected}$ は，100%のみを測定した．これは，本測定では，優先処理が優先ファイルのみにアクセスするため，保護プールのキャッシュヒット率をできるだけ向上させることで，優先処理の実行処理時間を短縮できるためである．T_{normal} は，50%以上，かつ優先すべき保護プールのキャッシュヒット率の閾値 $T_{protected}$ 未満の値を複数測定した．また，B_{pmin} は，優先処理が優先ファイルのみにアクセスするため，保護プールを優先すべきと考え，50%以上の値を複数測定した．

(3) gnuplot とカーネル make を同時実行

gnuplot を優先処理，カーネル make を非優先処理として実行した．ω，$T_{protected}$，および B_{nmin} は，Web サーバとバックアップ処理を同時実行した場合の測定と同様の理由により，1 通りのみ測定した．また，Web サーバとバックアップ処理を同時実行した場合の測定と同様に，gnuplot は優先ファイルのみにアクセスするため，T_{normal}，B_{pmin} は，(2) と同様の考え方により，複数測定した．

測定の結果，いくつかのパラメータは，以下の 2 つの条件により，設定すべき値が異なるという特性を持つことが分かった．

(条件 1) 優先ファイルの総サイズが入出力バッファサイズ以上となる設定

(条件 2) 優先処理が非優先ファイルにもアクセス

3.4.2 項から 3.4.6 項では，2 つの条件で場合分けしつつ，各パラメータの特性と代表値を示す．

3.4.2 ω：キャッシュヒット率を判定するブロックアクセス回数

S_{target} 変更は，前回変更時から十分にバッファが利用され，S_{target} 変更による作用を観測できる間隔をあけて行う必要がある．このためには，入出力バッファ内の多くのバッファが，1 度はアクセスされていることが望ましい．いい換えると，入出力バッファ内のバッファが，キャッシュヒットにより再利用された，もしくはキャッシュミスしてディスクから読み込まれたバッファとなっている状態である．これを狙い，たとえば $\omega = B$ とするのがよい．

3.4.3 $T_{protected}$：保護プールのキャッシュヒット率の閾値

（条件 1）と（条件 2）が両方成立する場合，保護プールが大きくなりすぎることにより，非優先ファイルのキャッシュヒット率が低下し，（要求 2）を満足できない可能性がある．これを防ぐため，$T_{protected} < 100\%$ とする必要がある．ただし，優先処理は優先ファイルに頻繁にアクセスするため，$T_{protected} > T_{normal}$ とする．たとえば，$T_{protected} = 95\%$ が性能に良い値である結果を実験で得ている．

それ以外の場合，$T_{protected} = 100\%$ とする．（条件 1）が成立しない場合，優先ファイルのブロックをすべて保護プールにキャッシュしても，通常プールの領域を確保できる．また，（条件 2）が成立しない場合，優先処理は非優先ファイルにアクセスしない．これらの場合，$T_{protected} = 100\%$ とし，優先ファイルをできるだけ多くキャッシュした方がよい．

3.4.4 T_{normal}：通常プールのキャッシュヒット率の閾値

（条件 2）が成立する場合，優先処理の実行処理時間を短縮するためには，非優先ファイルのキャッシュヒット率も高くする必要がある．ただし，優先処理は優先ファイルに頻繁にアクセスするため，$T_{normal} < T_{protected}$ とする．たとえば，$T_{normal} = 85\%$ が性能に良い値である結果を実験で得ている．

（条件 2）が成立しない場合，優先処理は非優先ファイルにアクセスしないため，優先ファイルをできるだけ多くキャッシュすることにより，優先処理の実行処理時間を短縮できる．したがって，T_{normal} を小さくする．ただし，T_{normal} を小さくしすぎると，非優先ファイルのキャッシュヒット率が低下し，（要望 1）に対応できない．これらを両立するように，通常プールのキャッシュヒット率を少なくとも 50%以上とすることを目指す．たとえば，T_{normal} を 60%から 70%の値とする．

3.4.5 B_{pmin}：保護プールのサイズの下限

（条件 2）が成立する場合，優先処理の実行処理時間を短縮するためには，通常プールもある程度大きくする必要がある．このために，B_{pmin} を小さくし，通常プールの拡大を可能とする．しかし，B_{pmin} を小さくしすぎると，保護プールが小さくなり，優先ファイルのキャッシュヒット率が低下し，優先処理の実行処理時間を短縮できない．これらを両立するように，入出力バッファの半分未満の設定とする．たとえば，$B_{pmin} = (B$ の 30%) とする．

（条件 2）が成立しない場合，優先処理は非優先ファイルにアクセスしないため，B_{pmin} を大きくし，保護プールを常に大きくする．ただし，$B_{pmin} < (B$ の 100%) とする．これは，（要望 1）に対応するためには，通常プールを確保する必要があるためである．両者を両立するように，保護プールとして入出力バッファの大部分を利用しつつ，通常プールとして利用可能な領域を残す．たとえば，$B_{pmin} = (B$ の 70%から 80%) とする．

3.4.6 B_{nmin}：通常プールのサイズの下限

測定の結果，$B_{nmin} =$ (先読みブロック数の最大数) とし，通常プールに最低限必要なバッファを確保することがよいと判明した．

3.5 期待される効果

提案方式は（要求 1）に加え，（要求 2）を満足し，（要望 1）にも対応しているため，以下の効果が期待できる．

(1) 多くのディレクトリ直下のファイルにアクセスする優先処理の実行処理時間を短縮，もしくは増加を抑制

2.2 節で述べたとおり，頻繁にアクセスされるファイルを多く含むディレクトリのみを優先ディレクトリに指定することが有効である．この場合，優先処理は非優先ファイルにもアクセスする．提案方式は（要求 2）を満足するため，非優先ファイルにもアクセスする優先処理の実行処理時間を短縮，もしくは基本方式と比べて実行処理時間の増加を抑制できる．

(2) 非優先処理の実行処理時間の増加を抑制

提案方式は，（要望 1）に対応しているため，非優先処理の実行処理時間の増加を抑制できる．

4. 実装と評価

4.1 実装上の課題と対策

4.1.1 実装上の課題

提案方式は，3 章で述べた方式を実装するうえで，保護プールと通常プールのキャッシュヒット率の計測が課題となる．

既存方式には，キャッシュヒット率を基に，分割サイズを決定する方式はない．この理由を次のように推察する．既存方式は，ブロックのアクセス頻度やアクセスパターンを基に格納先の入出力バッファ領域を決定する．このため，意味のある単位でキャッシュヒット率を測定しようとすると，ブロックやファイル単位でキャッシュヒット率を測定する必要があり，オーバヘッドが大きくなる．一方，提案方式では，保護プールと通常プールごとにキャッシュヒット率を測定すればよいため，キャッシュヒット率の測定数が少ない．ただし，キャッシュヒット率の測定が性能に悪影響を及ぼさないように，低オーバヘッドの実現方式を検討する必要がある．

4.1.2 実現方式

入出力バッファの保護プールと通常プールへの分割については，文献 [1] と同様である．これに対し，図 2 の 2 重四角で示した処理を実装し，保護プールサイズを可変とした．実装量は約 250 ステップである．

低オーバヘッドでのキャッシュヒット率の測定のため，提案方式は，バッファに優先ファイルのブロックを保持するか否かを示すフラグを持たせる．これにより，アクセスの度に，アクセスしたバッファが優先ファイルのブロックを保持するか否か確認するために，優先ディレクトリ情報管理表を検索することを不要とした．このフラグを用い，提案方式は，以下の手順で各プールのアクセス数とヒット数を測定し，ヒット数をアクセス数で割った商としてキャッシュヒット率を求める．なお，提案方式は，ω ごとに測定値を 0 に初期化する．また，キャッシュミスした場合，このキャッシュヒット率の算出は，ディスクからブロックをバッファに読み込んだ後に行う．

(1) フラグが ON であるバッファへのアクセスの場合
  (a) 保護プールのアクセス数をインクリメント
  (b) キャッシュヒットであれば，保護プールのヒット数をインクリメント

(2) フラグが OFF であるバッファへのアクセスの場合
  (a) 通常プールのアクセス数をインクリメント
  (b) キャッシュヒットであれば，通常プールのヒット数をインクリメント

4.2 評価内容

基本方式で満足できなかった（要求 2）を満足できるか否かを明らかにするため，以下の評価を行った．また，（要望 1）は必ずしも満足する必要はないが，（要求 1）（要求 2）を満足したうえで，どの程度対応できているか評価した．

(1)（要求 2）の評価

2.2 節で述べた 1 つ目の問題点の状況を作るため，アクセスするファイルが複数のディレクトリに分散している処理を優先処理として実行する．このとき，アクセス頻度が高いファイルを持つディレクトリのみを優先ディレクトリに指定する．このような処理として，カーネル make を選定した．カーネル make がアクセスするソースコードやヘッダファイルは，多数のディレクトリに分散している．カーネル make の実行処理時間を測定し，（要求 2）を満足するか否かを明らかにする．また，カーネル make 実行中のキャッシュヒット率の変化，および S_{target} と S_{cur} の増減を測定した．これにより，3.2 節で述べた（方針 1）と（方針 2）に従い，提案方式が入出力バッファを制御できているか否かを評価する．

(2)（要望 1）の評価

この評価では，優先処理と非優先処理を共存実行する．また，優先処理は，ディレクトリ優先方式の（着目点 1）（着目点 2）に対応する処理を用いて評価した．

(a)（着目点 1）に対応する優先処理

（着目点 1）に対応する処理として，アクセスするデータに応じて，処理優先度が異なる処理を選定する必要がある．そこで，文献 [1] と同様に Web サーバを選定し，Web サーバが管理する特定のページに対する要求を高い優先度で実行することを想定する．また，Web サーバと同時にバックアップ処理を実行することで，アクセスされる非優先ファイルを増加させ，提案方式の効果を明確化する．Web サーバが持つトップページを構成するファイルの要求に対する Web サーバの処理を優先処理，Web サーバの他の処理と Web コンテンツのファイルのバックアップ処理を非優先処理として評価し，（要求 1）を満足するか否か，および（要望 1）への対応の程度を明らかにする．

(b)（着目点 2）に対応する優先処理

（着目点 2）に対応する評価として，優先処理を繰り返し実行し，評価する．また，次の点を明らかにするため，ファイルへシーケンシャルにアクセスし，ディスク I/O 負荷の高くない処理を選定する．このような処理を優先処理として単独で実行した場合，LRU 方式でもキャッシュヒット率が高くなるため，ディレクトリ優先方式の優位性は乏しい．しかし，このような処理とともに，ファイルアクセスの多い非優先処理を共存実行した場合，LRU 方式は，非優先処理がアクセスしたファイルのブロックを入出力バッファに格納するために，優先処理が繰り返しアクセスするファイルのブロックを入出力バッファから破棄することがある．この結果，優先処理のキャッシュヒット率が低下し，優先処理の実行処理時間が増加する．ディレクトリ優先方式は，優先処理がアクセスするファイルのブロックを保護プールにキャッシュすることにより，これを防止可能である．提案方式も同様に優先処理の実行処理時間を短縮して（要求 1）を満足し，そのうえで（要望 1）へ対応可能か否かを明らかにする必要がある．そこで，gnuplot を連続して実行することとした．gnuplot は，データファイルをシーケンシャルに読み込み，画像ファイルを出力する．評価では，gnuplot を繰り返し実行し，同一のデータファイルから，複数のグラフを生成する．このとき，データファイルは繰り返しアクセスされる．また，共存する処理として，ファイルアクセスの多いカーネル make を選定し，非優先処理として実行した．

上記の評価では，要求と要望を満足したと判断する基準を以下のようにおいた．

(1)（要求 1）：優先処理の実行処理時間を LRU 方式と比べて短縮

(2)（要求 2）：優先処理の実行処理時間を LRU 方式と比べて短縮，もしくは基本方式と比べて増加を抑制

(3)（要望 1）：非優先処理の実行処理時間を基本方式と比べて短縮

LRU 方式は多くの場合で比較的良い性能を出せるということが知られており，一般的に多くの OS で利用されているため，LRU 方式を比較対象とした．

4.3 カーネル make を優先処理とした評価

4.3.1 評価方法

FreeBSD 4.3-RELEASE（以降，FreeBSD 4.3-R と略す）のカーネル make の実行処理時間を測定した．測定前に make depend を実行した．本評価では，make depend 実行完了直後，/usr/src/sys/sys/と/usr/src/sys/i386/include/を優先ディレクトリに指定した．2 つのディレクトリは，直下に総サイズ約 1.6 MB（338 ブロック）のヘッダファイルを有す．カーネル make は，この内約 1.2 MB 強（254 ブロック）の優先ファイルにアクセスする．カーネル make はヘッダファイルに繰り返しアクセスし，この 2 つのディレクトリ直下のヘッダファイルは，ほかのヘッダファイルと比べ，より頻繁にアクセスされる．また，カーネル make がアクセスする非優先ファイルの総サイズは，約 44 MB である．なお，提案方式では，make depend 実行前に S_{target} を 0 に初期化した．

4.3.2 評価環境

計算機（CPU：Celeron 2.0 GHz，メモリ：768 MB，OS：FreeBSD 4.3-R，VMIO：オフ，1 バッファのサイズ：8.0 KB）を用いて評価した．入出力バッファサイズは，文献 [1] での評価と同様の 6.3 MB，およびこの半分である 3.0 MB について測定した．ただし，システム維持のために常時確保される領域があるため，実際に利用できる領域は入出力バッファサイズより 0.7 MB ほど小さく，前者は 720 個，後者は 296 個のバッファを保持できる．

VMIO は，入出力バッファでキャッシュミスした際に，ページキャッシュにアクセスされたブロックが存在しないか探索し，存在すればブロックを再利用する機能である．ページキャッシュは LRU 方式で制御されている．本評価では，提案方式の効果を示すため，VMIO をオフとし，LRU 方式で制御されるページキャッシュの効果が混在することを避ける．

評価に使用した make と gcc は，FreeBSD 4.3-R に付属のバージョンを使用している．これらの AP は，最新バージョンも，ソースコードとそれがインクルードするヘッダファイルを読み込み，オブジェクトファイルを作成するという処理に違いはない．したがって，最新バージョンの AP を使用しても，本評価と同様の効果があると推察する．

3.4 節の指針に従い，パラメータを以下のとおり設定した．

(1) 入出力バッファサイズ 3.0 MB（3.4.1 項（条件 1），（条件 2）成立）

$$(\omega, T_{protected}, T_{normal}, B_{pmin}, B_{nmin}) = (296, 95\%, 85\%, 89, 32)$$

(2) 入出力バッファサイズ 6.3 MB（3.4.1 項（条件 2）成立）

$$(\omega, T_{protected}, T_{normal}, B_{pmin}, B_{nmin}) = (720, 100\%, 85\%, 216, 32)$$

4.3.3 実行処理時間の評価結果

測定結果を**図 4** に示す．図 4 から，以下のことが分かる．

入出力バッファサイズが 3.0 MB の場合，提案方式は，LRU 方式と比べて約 6.0 秒（約 1.5%）増加している．一方，基本方式は，LRU 方式と比べて約 55.4 秒（約 14.2%）増加している．これは，非優先ファイルのキャッシュヒット率の低下によるものである．提案方式は，基本方式と比べて，LRU 方式に対する実行処理時間の増加を抑制できている．

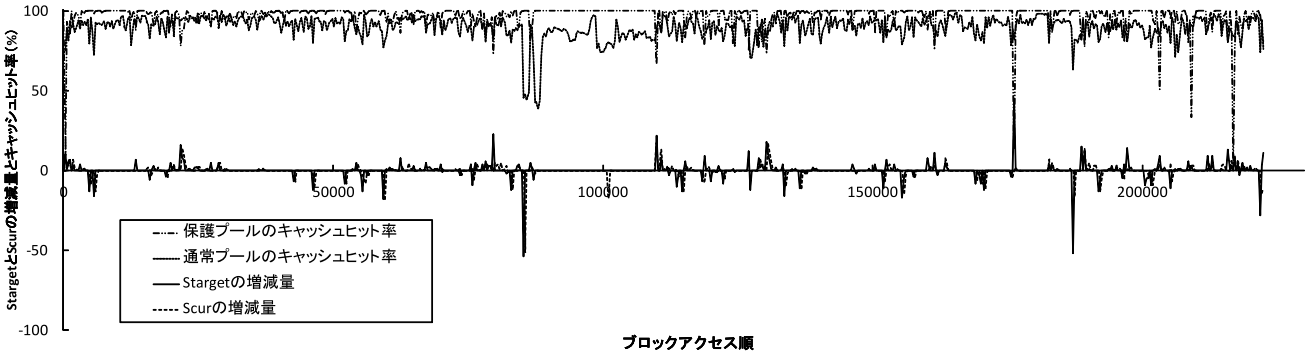

図 5 キャッシュヒット率の変化と S_{target} と S_{cur} の増減量
Fig. 5 Change of cache hit ratio, and amount of S_{target} and S_{cur} change.

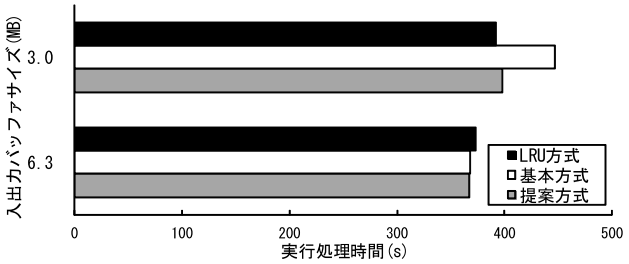

図 4 カーネル make の実行処理時間
Fig. 4 Execution time of kernel make.

また，入出力バッファサイズが 6.3 MB の場合，提案方式は，LRU 方式と比べて約 6.5 秒（約 1.7%）短縮できた．

上記の結果より，提案方式は，非優先ファイルにアクセスする優先処理の実行処理時間を LRU 方式と比べて短縮，もしくは基本方式と比べて増加を抑制できており，（要求 2）を満足したといえる．

4.3.4 キャッシュヒット率の変化とこれにともなう S_{target} と S_{cur} の増減の評価結果

カーネル make 実行中のキャッシュヒット率の変化，および S_{target} と S_{cur} の増減量を**図 5** に示す（入出力バッファサイズ 3.0 MB, $\omega = 296$, $T_{protected} = 95\%$, $T_{normal} = 85\%$, $B_{pmin} = 89$, $B_{nmin} = 30$）．図 5 から以下のことが分かる．

(1) キャッシュヒット率の変化に応じて，S_{target} が増減している．保護プールのキャッシュヒット率が $T_{protected}$ 未満に低下すると，S_{target} が増加し，通常プールのキャッシュヒット率が T_{normal} 未満に低下すると，S_{target} が減少している．

(2) S_{target} の増減と比べ，S_{cur} の増減の契機が遅れている．この結果から，（方針 2）を満たし，バッファを有効利用できているといえる．

(3) S_{target} の増減にともない，S_{cur} も増減している．この結果から，3.3.4 項で述べたバッファの解放規則により，保護プールのサイズを S_{target} に近づけるように制御できているといえる．

(4) 保護プールや通常プールのキャッシュヒット率が大きく低下することがある．たとえば，横軸 175,000 アクセスや 220,000 アクセス付近では，保護プールのキャッシュヒット率が低下している．前者は，この直前まで，非優先ファイルへのアクセス数も多く，通常プールのキャッシュヒット率が T_{normal} を下回ることが多いため，S_{target} が B_{pmin} まで小さくなってることが原因である．このとき，優先ファイルへのアクセス数が増加したため，その変化へ追従して保護プールを大きくするまでの間，保護プールのキャッシュヒット率が低下したと考えられる．また，この現象とは反対に，85,000 アクセス付近では，非優先ファイルへのアクセス数の増加に追従して通常プールを大きくするまでの間，通常プールのキャッシュヒット率が大きく低下している．一方，後者の 220,000 アクセス付近では，優先ファイルのブロックへのアクセス数が非常に少ないため，1 回のキャッシュミスがキャッシュヒット率に与える影響が大きく，キャッシュヒット率が低下している．たとえば，キャッシュヒット率が 0%に低下している箇所では，ω 回のブロックアクセスのうち，1 回しか優先ファイルにアクセスしていない．この 1 回がキャッシュミスしたため，保護プールのキャッシュヒット率が 0%になっている．ただし，この原因によってキャッシュヒット率が低下したとしても，提案方式は，S_{target} を過剰に増加・減少させることはない．この理由は，式 (2)，(3) のとおり，優先ファイルや非優先ファイルのアクセス数を基準に，S_{target} の増加量や減少量を決定しているためである．

4.4 Web サーバ運用中に Web コンテンツのファイルをバックアップする場合の評価

4.4.1 評価方法

岡山大学の Web サーバ（www.okayama-u.ac.jp）のディレクトリ構造を再現し，Web サーバを介してファイルにアクセスした際の応答時間を測定した．また，このディレクトリ構造のバックアップ処理を同時実行し，この実行処理時間も測定した．2006 年 7 月の岡山大学の Web サーバへの要求から 100,000 回を抽出し，Web サーバにアクセスし

表 2 評価で用いたファイルの情報（100,000 回要求）
Table 2 Information of target file of 100,000 access.

	全体	優先ファイル	6 部局のトップページを構成するファイル
ディレクトリ数（個）	1,365	14	14
ファイル数（個）	8,849	877	118
合計要求回数（回）	100,000	59,467	21,391
合計ファイルサイズ（MB）	600.0	11.1	0.7

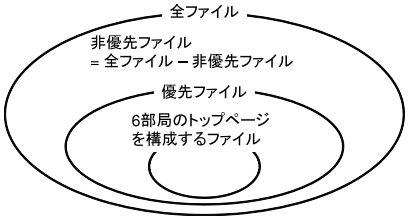

図 6 評価で用いたファイルの包含関係
Fig. 6 Inclusion relation of files for Web server evaluation.

表 3 バックアップしたファイルの情報
Table 3 Information of files backup accessed.

	全体	優先ファイル
合計ファイルサイズ（MB）	2,818.5	13.9

た．なお，岡山大学の Web サーバが持つ 6 部局のトップページを構成するファイルの要求に対する Web サーバの処理を優先処理，Web サーバの他の処理とバックアップ処理を非優先処理とした．

本評価では，文献 [1] と同様に，6 部局のトップページを構成するファイルを直下に有するディレクトリを優先ディレクトリに指定した．評価で用いたファイルの情報を**表 2** に示し，これらのファイルの包含関係を**図 6** に示す．表 2 と図 6 より，6 部局のトップページを構成するファイル（0.7 MB）は，優先ファイル（11.1 MB）の一部である．したがって，優先ファイルには，6 部局のトップページを構成するファイルとそうでないファイルがある．さらに，バックアップしたファイルの情報を**表 3** に示す．表 3 に示す 2,818.5 MB のファイルは，表 2 のファイルを含む．

測定前に 100,000 回 Web サーバにアクセスすることにより，入出力バッファに Web コンテンツのファイルがキャッシュされている状態にした．この 100,000 回のアクセス終了後，バックアップ処理と次の 100,000 回のアクセス（アクセス間隔 100 ms）をほぼ同時に開始した．また，提案方式では，最初の 100,000 回のアクセス前に，S_{target} を 0 に初期化した．

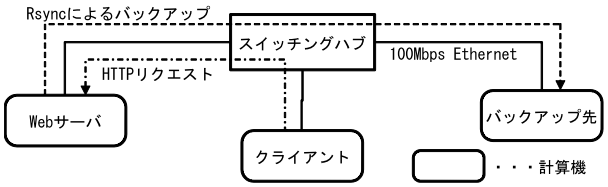

図 7 Web サーバによる評価環境
Fig. 7 Environment of Web server evaluation.

表 4 Web サーバによる評価に用いた計算機
Table 4 Specification of computers for Web server evaluation.

	Web サーバ	クライアント	バックアップ先
CPU	Celeron D 2.8 GHz	Celeron D 2.8 GHz	Celeron 2.0 GHz
メモリ	32 MB	256 MB	768 MB
入出力バッファ	6.4 MB	32 MB	87 MB
OS	FreeBSD 4.3-R	FreeBSD 4.3-R	FreeBSD 4.3-R
VMIO	オフ	オン	オン
1 バッファのサイズ	8.0 KB	16.0 KB	16.0 KB

4.4.2 評価環境

評価に用いた環境を**図 7** に示す．また，図 7 の各計算機の性能を**表 4** に示す．Web サーバを実行する計算機は，734 個のバッファを保持できる．

入出力バッファのサイズは，2.2 節で述べた問題が顕著に現れるように，文献 [1] での評価と比べて小さい 32 MB に制限した．これは，メモリを数 GB 搭載した場合であっても，Web サーバがアクセスするファイルの総サイズが増加すれば，提案方式は本評価と同様の効果を示すと考えられるためである．

Web サーバとして Apache 2.0.55，クライアントプログラムとして ApacheBench 2.40-dev を用いた．また，バックアップ用プログラムとして rsync 2.4.6 を用いた．Apache 2.0.55 には，入出力バッファを介さずにファイルを転送する sendfile システムコールを利用する機能がある．入出力バッファの制御方式の評価を行うため，本評価ではこの機能を無効にした．

また，Apache は，file cache モジュールや cache モジュールを持ち，最新の Apache 2.4 では cache モジュールの改善がなされている．このような AP レベルのキャッシュ方式は，AP 固有のアクセスパターンをキャッシュ管理に反映しやすい利点がある．しかし，本評価の目的は，OS のキャッシュ方式の評価であるため，これらのモジュールを使用しなかった．また，これらのモジュールには，次の欠点もあり，必ずしも使用されるとは限らない．まず，OS の入出力バッファやページキャッシュとは別のポリシでメモリを管理するため，必ずしもシステム全体のメモリ使

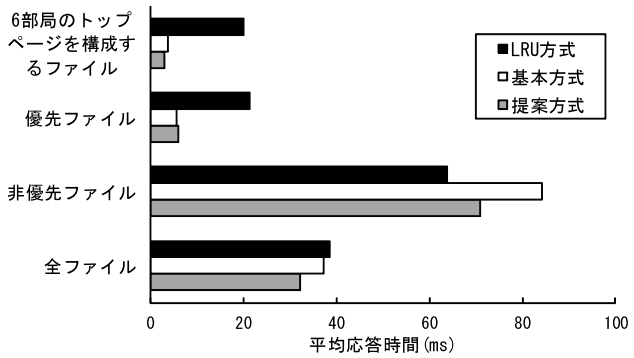

図 8 Web サーバの平均応答時間（バックアップ処理動作中）
Fig. 8 Response time of web server during a backup.

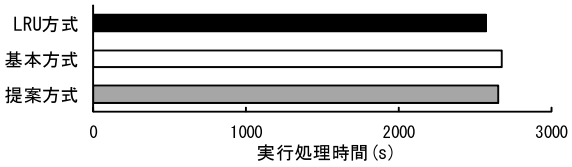

図 9 バックアップ処理の実行処理時間
Fig. 9 Execution time of backup.

用効率が良くなるとは限らない．たとえば，AP レベルでキャッシュしたファイルデータは当該 AP でしか使用できない．また，Apache の file cache モジュールは，キャッシュしたファイルの一部でも更新されると，Apache を再起動する必要があり，運用上問題となる可能性がある．したがって，これらのモジュールを使用しない環境は多くあり，この環境では提案方式の効果を得ることができる．

本測定は，3.4.1 項の（条件 1）が成立する場合である．このため，パラメータは，3.4 節の指針に従い，$(\omega, T_{protected}, T_{normal}, B_{pmin}, B_{nmin}) = (734, 100\%, 60\%, 514, 32)$ と設定した．

4.4.3 評価結果

Web サーバの平均応答時間を**図 8** に，バックアップ処理の実行処理時間を**図 9** に示す．図 8 と図 9 から，以下のことが分かる．

提案方式は，6 部局のトップページを構成するファイルの平均応答時間を LRU 方式と比べて約 17.2 ミリ秒（約 85.4%）短縮できた．また，提案方式は，非優先ファイルの平均応答時間を基本方式と比べて約 13.2 ミリ秒（約 15.7%）短縮できた．また，提案方式は，バックアップ処理を基本方式と比べて約 24.7 秒（約 0.9%）短縮できた．

この結果より，提案方式は，LRU 方式と基本方式と比べて優先処理の実行処理時間を短縮できており，（要求 1）を満足したといえる．これに加え，提案方式は，（要求 1）を満足したうえで，基本方式と比べて非優先処理の実行処理時間の増加を抑制できており，（要望 1）に対応できたといえる．

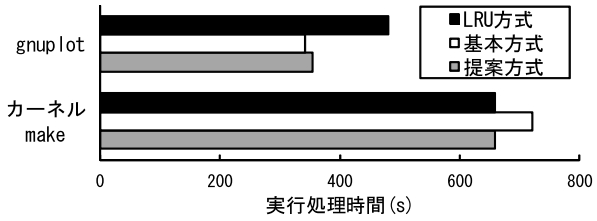

図 10 gnuplot とカーネル make の実行処理時間
Fig. 10 Execution time of gnuplot and kernel make.

4.5 gnuplot とカーネル make を同時実行した場合の評価

4.5.1 評価方法

gnuplot とカーネル make を同時実行し，両者の実行処理時間を測定した．なお，gnuplot を優先処理，カーネル make を非優先処理とした．

gnuplot は，5 個のデータファイル（1 個約 3.7 MB）から 4 個ずつ，計 20 個のグラフの画像ファイル（1 個約 0.26 MB）を作成する．この際，gnuplot は，グラフの設定が記述されたファイル（約 0.02 MB）を読み込む．これらすべてのファイルは同一のディレクトリに存在する．

本評価では，gnuplot がアクセスするファイルを直下に有するディレクトリを優先ディレクトリに指定した．優先ディレクトリを指定後，gnuplot とカーネル make をほぼ同時に実行する．また，提案方式では，測定前に，S_{target} を入出力バッファに保持できるバッファ数に初期化した．

4.5.2 評価環境

計算機（CPU：Celeron 2.0 GHz，メモリ：768 MB，入出力バッファサイズ：6.3 MB, OS：FreeBSD 4.3-R, VMIO：オフ，1 バッファのサイズ：8.0 KB）を用いて評価した．本評価は，4.3 節と同様，6.3 MB で測定した．

評価に使用した gnuplot，make，gcc は，FreeBSD 4.3-R に付属のバージョンを使用している．gnuplot は，最新バージョンも，データファイルと設定ファイルを読み込み，グラフの画像ファイルを出力する処理に違いはない．また，make と gcc は，4.3.2 項に記載のとおりである．したがって，最新バージョンの AP を使用しても，本評価と同様の効果があると推察する．

本評価は，3.4.1 項の（条件 1）が成立する場合である．このため，パラメータは，3.4 節の指針に従い，$(\omega, T_{protected}, T_{normal}, B_{pmin}, B_{nmin}) = (720, 100\%, 70\%, 576, 32)$ と設定した．

4.5.3 評価結果

測定結果を**図 10** に示す．図 10 から，以下のことが分かる．

提案方式は，gnuplot（優先処理）の実行処理時間を LRU 方式と比べて 127 秒（26.3%）短縮できた．これは，カーネル make がアクセスした非優先ファイルのブロックを入出力バッファに保持するために，繰り返し実行する gnuplot

がアクセスする優先ファイルのブロックが入出力バッファから破棄されることを抑えることができたためである．また，提案方式は，カーネル make（非優先処理）の実行処理時間を基本方式と比べて 62 秒（8.6%）短縮した．

上記の結果より，提案方式は，優先処理の実行処理時間を LRU 方式と比べて短縮できており，（要求 1）を満足したといえる．これに加え，提案方式は，（要求 1）を満足したうえで，非優先処理の実行処理時間の増加を抑制できており，（要望 1）に対応できたといえる．

4.6 考察

4.6.1 パラメータ設定の妥当性

提案方式のパラメータ設定が妥当であるといえる条件は，提案方式が（要求 1）と（要求 2）を満足できること，および（要望 1）に対応できることである．各評価において，提案方式は，（要求 1）と（要求 2）を満足でき，（要望 1）に対応できた．したがって，各評価におけるパラメータ設定は妥当であったといえる．

さらに，各評価におけるパラメータ設定は，3.4 節の設定指針に従った値である．各評価においてパラメータ設定が妥当であったことから，各評価のパラメータ設定の根拠であるこの設定指針も妥当であると推察できる．

4.6.2 VMIO を ON にした場合の考察

各評価では，FreeBSD の VMIO を OFF にして測定した．VMIO を ON にすると，ページキャッシュがキャッシュしているファイルデータを使用可能になる．ページキャッシュは，優先ファイルと非優先ファイルを区別せずキャッシュする．このため，VMIO を ON にすると，ページキャッシュの効果により，各評価における各方式の優先ファイルと非優先ファイルのキャッシュヒット率が向上し，キャッシュヒット率の改善の余地が小さくなるため，方式間の優先処理と非優先処理の実行処理時間の差は小さくなる．

しかし，VMIO を ON にしても，提案方式と基本方式の入出力バッファは，優先ファイルを優先的にキャッシュする．このため，優先ファイルのキャッシュヒット率が LRU 方式と比べて高くなり，優先処理の実行処理時間を短縮できると推察される．また，提案方式は，キャッシュヒット率に応じて，入出力バッファの通常プールへメモリを割り当てるため，提案方式は，基本方式と比べて，非優先処理の実行処理時間を短縮できると推察される．

4.6.3 他 OS への実現方式と効果

最新の FreeBSD 10.2-RELEASE も，FreeBSD 4.3-R と同様，入出力バッファを備えている．したがって，提案方式の効果を得ることができる．

Linux などの他 OS では，FreeBSD 4.3-R と異なり，入出力バッファがページキャッシュへ統合されていることがある．このような OS は，ファイルデータをページキャッシュへキャッシュする．入出力バッファは，あらかじめ予約されたメモリ量を使用して，ファイルのブロックをキャッシュする．一方，ページキャッシュは，各プロセスや OS が使用していない余っているメモリ領域をすべてキャッシュ用途として使用できる．

ページキャッシュにおいて，このキャッシュ領域を保護プールと通常プールに分割することで，ディレクトリ優先方式を実現できる．さらに，保護プールの大きさを提案方式により決定することにより，提案方式を他 OS に実現することができる．これにより，他 OS においても，本評価と同様，提案方式の効果を得ることができる．

また，ページキャッシュにおける提案方式は，以下の効果がある．計算機の搭載メモリ量に対し，実行中のプロセスが使用しているメモリ量が多い場合，ページキャッシュとして使用できるメモリが少なくなる．この場合，ページキャッシュは，キャッシュヒット率が低下する．ページキャッシュに対して提案方式を適用すると，この場合，保護プールに優先的にメモリを割り当てることができる．これにより，メモリの枯渇時も，優先処理の実行処理時間の増加を抑制できると期待できる．また，提案方式は，キャッシュヒット率に応じて，通常プールにもメモリを割り当てるため，非優先ファイルのキャッシュヒット率低下を抑制でき，非優先処理の実行処理時間の増加を抑制できる．

4.6.4 より高性能な計算機での効果

現行の計算機は，評価に用いた計算機と比べ，より高速な CPU やより大容量のメモリを搭載している．このような計算機であっても，HDD や SSD へのディスク I/O は，メモリへのアクセスと比べ，依然として遅い．したがって，現行の計算機であっても，ディスク I/O 回数を削減し，ディスク I/O 待ち時間を削減することにより，実行する処理の実行処理時間を短縮することができる．

優先処理がアクセスするファイルの総サイズが入出力バッファより大きい場合，優先処理のアクセス頻度が高いファイルを持つディレクトリを優先ディレクトリに指定することにより，基本方式や提案方式は，保護プールにアクセス頻度の高いファイルのブロックをキャッシュできる．これにより，優先処理のディスク I/O 回数を削減し，実行処理時間を短縮できる．また，提案方式は，キャッシュヒット率に応じて通常プールにもメモリを割り当てるため，非優先ファイルのキャッシュヒット率の低下を抑制できる．これにより，優先処理がアクセスする非優先ファイルのキャッシュヒット率が大きく低下することを抑制でき，非優先ファイルのキャッシュヒット率の低下による優先処理の実行処理時間の増加を抑制できる．

また，共存実行する非優先処理がアクセスするファイルの総サイズが入出力バッファより大きい場合，優先処理がアクセスするファイルを持つディレクトリを優先ディレクトリに指定することにより，基本方式や提案方式は，保護プールに優先処理がアクセスするファイルのブロックを

キャッシュできる．これにより，優先処理のディスクI/O回数を削減し，実行処理時間を短縮できる．また，提案方式は，キャッシュヒット率に応じて通常プールにもメモリを割り当てるため，非優先ファイルのキャッシュヒット率の低下を抑制でき，非優先処理の実行処理時間の増加を抑制できる．たとえば，バックアップ処理やバッチ処理が，入出力バッファサイズを超えるサイズのファイル群へアクセスすることがある．

4.6.5 提案方式が適さない用途

提案方式は，以下の用途には適さない．

(1) ディレクトリごとにファイルのアクセス頻度の偏りがない処理を優先処理として単独で実行

このような処理では，アクセスするファイルを有するディレクトリを優先ディレクトリとして指定し，一部のファイルへのアクセスのキャッシュヒット率を向上させても，それ以外のファイルへのアクセスのキャッシュヒット率が低下する．したがって，このような処理を優先処理として単独で実行した場合，処理全体でのキャッシュヒット率の向上が期待できない．これには，単一のファイルのみへアクセスする処理も含む．このような処理として，たとえば，ファイル群の検索や解析がある．ただし，このような処理と非優先処理を共存実行する場合，提案方式は有効である．これは，非優先処理がアクセスしたファイルのブロックを入出力バッファに格納するために，優先処理がアクセスするファイルのブロックを入出力バッファから破棄することを抑制できるためである．

(2) 非優先ファイルに続けて多くアクセスした後に優先ファイルにアクセスすることが多い処理を優先処理として実行

4.3.4 項で述べたとおり，提案方式は，多くの非優先ファイルへのアクセスが続き，非優先ファイルのキャッシュヒット率が低下すると，通常プールが大きくなり，保護プールが小さくなる．この後に，優先ファイルへアクセスすると，ただちに保護プールは大きくならないため，優先ファイルのキャッシュヒット率が低下する可能性がある．したがって，このようなアクセスの変化が多い処理は，優先ファイルのキャッシュヒット率が何度も低下し，実行処理時間が増加する可能性がある．

5. 関連研究

入出力バッファを分割して管理する方式 [3], [4], [5], [6], [7], [8], [9], [10], [11], [12] が提案されている．この内，2Q [3]，LIRS [4]，および DULO [5] は入出力バッファを静的に分割し，文献 [6] の方式は重要と判断したファイルをできるだけ多くキャッシュする．一方，ARC [7]，CAR [8]，UBM [9]，PCC [10]，Karma [11]，および文献 [12] で提案された方式は，分割した入出力バッファ領域のサイズを動的に決定する．

ARC [7] と CAR [8] は，入出力バッファを 2 つの入出力バッファ領域に分割し，各領域から破棄されたブロックの情報を一定量保持しておき，各領域のサイズの決定に利用する．

UBM [9] と PCC [10] は，ブロックをシーケンシャル，ループ，およびその他の 3 つのアクセスパターンに分類し，アクセスパターンごとに入出力バッファ領域を割り当てる．Karma [11] は，ヒントとして与えられたアクセス頻度とアクセスパターンにより，ブロック群を互いに素な集合に分割し，各集合に入出力バッファ領域を割り当てる．UBM，PCC，および Karma は，ブロックアクセス時に，入出力バッファ全体のキャッシュヒット率が最も高くなるように，各領域のサイズを変更する．

文献 [12] で提案された方式は，実行する処理ごとに，入出力バッファ領域を割り当てる．この方式は，システム全体のキャッシュヒット率を向上させることを目的に，各領域のサイズを決定する．

このように，文献 [3], [4], [5], [6], [7], [8], [9], [10], [11], [12] の方式は，システム全体のキャッシュヒット率の向上を目的とした方式であり，優先処理の実行処理時間を短縮することはできない．一方，提案方式は，優先処理が頻繁にアクセスするファイルのブロックを優先的にキャッシュし，優先処理の実行処理時間を短縮できる．また，提案方式は，分割した入出力バッファ領域のサイズの決定に，各領域のキャッシュヒット率とその目標値を利用している．したがって，一方の領域のキャッシュヒット率を目標値以上に高く維持できる範囲で，他方のキャッシュヒット率の低下を抑制するように入出力バッファ領域のサイズを決定できる新しい方式である．

優先処理のキャッシュヒット率を向上させるために，設定パラメータを減らして，各入出力バッファ領域の大きさを決定する方式が提案されている [13]．この方式は，提案方式よりも設定パラメータが少なく，設定が簡単である利点がある．一方，提案方式は，各入出力バッファ領域のキャッシュヒット率を直接把握するため，パラメータの設定により，キャッシュヒット率の変化に合わせて適切に各入出力バッファ領域の大きさを決定できる利点がある．

近年では，大容量のメモリを活用し，tmpfs [14] を用いた処理の高速化が可能である．tmpfs を用いた処理高速化方法として，AP が頻繁にアクセスするファイルを tmpfs へコピーし，AP は tmpfs 上のファイルを使用する方法が考えられる．tmpfs に格納したファイルは，スワップアウトされない限り，メモリに格納されるため，アクセス時間が短い．一方，AP がアクセスしないファイルを tmpfs へコピーすると，このようなファイルもメモリに保持することとなり，メモリの無駄が大きくなる．また，tmpfs 上のファイルを更新した場合，データの永続化のため，ディスクへ書き出す必要がある．さらに，この永続化処理の実行

前や実行中に電源断などの障害が発生すると，tmpfs のみに保存していたデータを喪失する．そこで，以下の場合，tmpfs と比べ，提案方式が有用であるといえる．

(1) アクセスするファイルの総サイズに対し，キャッシュとして使用できるメモリ容量が少ない
(2) ファイルの更新が多い
(3) tmpfs 上のファイルを利用するように，AP を改変することができない

6. おわりに

分割した各入出力バッファ領域のキャッシュヒット率に着目し，各領域のサイズを動的に変更する方式を提案した．提案方式は，分割した各領域のキャッシュヒット率が閾値より低い場合，当該領域のサイズを大きくする．なお，両領域ともキャッシュヒット率が閾値より低い場合は，優先ファイルをキャッシュする領域のサイズを大きくする．また，キャッシュしているブロックをできるだけ有効利用するバッファ解放規則を述べた．さらに，提案方式を有効に利用するために，5 つのパラメータの設定指針を示した．

提案方式が優先処理と非優先処理の実行処理時間の増加を防止，もしくは抑制する効果を有することを明らかにするため，カーネル make，Web サーバ，および gnuplot を優先処理とした 3 つの評価を述べた．

カーネル make を用いた評価では，入出力バッファサイズ 3.0 MB の場合，文献 [1] のディレクトリ優先方式は，実行処理時間が LRU 方式と比べて約 55.4 秒（約 14.2%）増加した．この場合，提案方式は，LRU 方式と比べて約 6.0 秒（約 1.5%）の増加に抑えた．これに加え，入出力バッファサイズ 6.3 MB の場合，提案方式は，実行処理時間を LRU 方式と比べて約 6.5 秒（約 1.7%）短縮した．これらの結果から，提案方式は，非優先ファイルにもアクセスする優先処理であっても，実行処理時間を短縮，もしくは実行処理時間の増加を抑制できることを示した．

また，Web サーバを用いた評価では，トップページを構成するファイルの要求に対する Web サーバの処理を優先処理として評価した．この結果，提案方式は，トップページを構成するファイルの平均応答時間を LRU 方式と比べて約 17.2 ミリ秒（約 85.4%）短縮しつつ，非優先ファイルの平均応答時間を文献 [1] のディレクトリ優先方式と比べて約 13.2 ミリ秒（約 15.7%）短縮できた．これに加え，gnuplot を用いた評価では，提案方式は，優先処理である gnuplot の実行処理時間を LRU 方式と比べて 127 秒（26.3%）短縮しつつ，非優先処理であるカーネル make の実行処理時間を文献 [1] のディレクトリ優先方式と比べて 62 秒（8.6%）短縮した．これらの結果から，優先処理と非優先処理を同時実行した場合，優先処理の実行処理時間を短縮したうえで，非優先処理の実行処理時間の増加を抑制できることを示した．

参考文献

[1] 田端利宏，小峠みゆき，乃村能成，谷口秀夫：ファイルの格納ディレクトリを考慮したバッファキャッシュ制御法の実現と評価，電子情報通信学会論文誌 D，Vol.J91-D, No.2, pp.435–448 (2008).

[2] 土谷彰義，松原崇裕，山内利宏，谷口秀夫：ディレクトリ優先方式における効果的な優先ディレクトリ設定法の提案と評価，電子情報通信学会論文誌 D，Vol.J96-D, No.3, pp.506–518 (2013).

[3] Johnson, T. and Shasha, D.: 2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm, *Proc. 20th International Conference on Very Large Data Bases*, pp.439–450 (1994).

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

[5] Ding, X., Jiang, S. and Chen, F.: A buffer cache management scheme exploiting both temporal and spatial localities, *ACM Trans. Storage*, Vol.3, No.2, Article No.5 (2007).

[6] 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).

[7] 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).

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

[9] 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, *Proc. 4th Symposium on Operating System Design and Implementation* (*OSDI 2000*), pp.119–134 (2000).

[10] Gniady, C., Butt, A.R. and Hu, Y.C.: Program-Counter-Based Pattern Classification in Buffer Caching, *Proc. 6th Symposium on Operating Systems Design and Implementation* (*OSDI '04*), pp.395–408 (2004).

[11] Yadgar, G., Factor, M. and Schuster, A.: Karma: Know-it-All Replacement for a Multilevel cAche, *Proc. 5th USENIX Conference on File and Storage Technologies* (*FAST '07*), pp.169–184 (2007).

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

[13] 山本光一，土谷彰義，山内利宏，谷口秀夫：未参照バッファ数に着目した入出力バッファ分割法，情報処理学会研究報告，Vol.2014-OS-130, No.5, pp.1–8 (2014).

[14] Snyder, P.: tmpfs: A virtual memory file system, *Proc. Autumn 1990 European UNIX Users' Group Conference*, pp.241–248 (1990).

土谷 彰義 （正会員）

2010 年岡山大学工学部情報工学科卒業．2012 年同大学大学院自然科学研究科博士前期課程修了．同年株式会社日立製作所横浜研究所入所．オペレーティングシステムに興味を持つ．

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．

谷口 秀夫 （正会員）

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．本会フェロー．