キャッシュヒット率に着目した入出力バッファの自動分割法

土谷彰義[†]　山内利宏[†]　谷口秀夫[†]

利用者が優先して実行したい処理（優先処理）の実行処理時間を短縮する方式として，ディレクトリ優先方式を提案した．ディレクトリ優先方式は，入出力バッファを 2 つの領域に分割し，指定したディレクトリ直下のファイル（優先ファイル）と他のファイル（非優先ファイル）を別領域にキャッシュし，非優先ファイルのキャッシュが優先ファイルのキャッシュを無効化しないようにしている．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指定することにより，優先処理の実行処理時間を短縮できる．しかし，最適なサイズで分割することは難しい．そこで，本稿では，2 つの領域のキャッシュヒット率に着目し，2 つの領域のサイズを自動的に決定し，更新する方式について述べる．また，カーネル make 処理と Web サーバ処理において提案方式を評価し，有効性を示す．

Automatic Method of Partitioning I/O Buffer Based on Cache Hit Ratio

Akiyoshi TSUCHIYA,[†] Toshihiro YAMAUCHI[†] and Hideo TANIGUCHI[†]

Performance of high priority processing can be improved by improving the cache hit ratio in I/O buffer. Thus, we proposed a directory oriented buffer cache mechanism. This mechanism gives a high priority to important directories, which are associated with high priority processing. Files in important directories are important files. This mechanism partitions I/O buffer into 2 areas, and cache important files and unimportant files in a different area. This prevents cache of unimportant files from invalidating cache of important files. Therefore, performance of high priority processing is improved. However, it is difficult to partition I/O buffer into optimal size areas. This paper proposes I/O buffer partitioning method based on cache hit ratio. This method automatically decides and updates the sizes of the areas. Additionally, This paper also describes the effectivity of this method by the evaluation with kernel make or Web server.

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．このとき，優先処理の入出力バッファのキャッシュヒット率を向上させ，ディスク I/O 回数を削減することにより，優先処理の実行処理時間を短縮できる．そこで，優先処理の実行処理時間を短縮させるための入出力バッファの制御方式としてディレクトリ優先方式[1] を提案した．

ディレクトリ優先方式は，入出力バッファを 2 つの領域に分割し，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）を別領域にキャッシュし，非優先ファイルのキャッシュが優先ファイルのキャッシュを無効化しないようにしている．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指定することにより，優先処理の実行処理時間を短縮できる．しかし，ディレクトリ優先方式には，優先ファイルをキャッシュする領域が必要以上に拡大し，非優先ファイルのキャッシュヒット率が低下する問題点がある．これにより，優先処理と非優先処理の実行処理時間が増加する可能性がある．

文献 2) では，上記の問題点を解決する入出力バッファ分割法を提案した．この分割法は，優先ファイルをキャッシュする領域の分割サイズを静的に決定し，必要以上に拡大することを防止する．しかし，分割サイズを静的に決定する場合，応用プログラム（以降，AP と略す）毎にアクセスパターンが異なるため，適切に決定することが難しい．また，分割サイズを静的

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

に決定すると，実行する AP のアクセスパターンの変化に対応できない．

そこで，本稿では，キャッシュヒット率に着目し，分割サイズを動的に決定する入出力バッファ分割法を提案する．具体的には，優先ファイルと非優先ファイルのキャッシュヒット率を確認し，結果に基づき，分割サイズを動的に自動で決定し，更新する．さらに，カーネル make 処理と Web サーバ処理において提案方式を評価し，有効性を示す．

2. ディレクトリ優先方式

2.1 基 本 方 式

文献 1) で提案した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では，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．

ブロック読み込み時には，読み込むブロックを保持するための空きバッファを確保する．このとき，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の親ディレクトリが優先ディレクトリであれば保護プールに，親ディレクトリが優先ディレクトリでなければ通常プールにバッファを格納する．このように，保護プール内のバッファが保持するブロックは，通常プール内のバッファが保持するブロックと比べて優先的に入出力バッファ内に残される．このため，優先処理が頻繁にアクセスするファイルを直下に多く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より，優先処理のキャッシュヒット率を向上させ，高速に実行できる．保護プールの分割サイズは，入出力バッファサイズを超えない範囲で大きくなる．

2.2 問 題 点

ディレクトリ優先方式には，以下の問題点がある．

（問題点 1） 優先処理の実行処理時間の増加

優先処理が複数のディレクトリ直下のファイルにアクセスする場合がある．この場合，全てのディレクトリを優先ディレクトリに指定すると，繰り返しアクセスするファイルだけでなく，一度しかアクセスしないファイルも優先ファイルとなる．ディレクトリ優先方式では，優先ファイルのブロックを保持するバッファを LRU 方式で管理するため，一度しかアクセスされないファイルへのアクセスにより，繰り返しアクセスするファイルのブロックを保持するバッファが入出力バッファから解放され，優先処理の実行処理時間が増加する可能性がある．一方，優先処理が頻繁にアクセスするファイルを直下に多く有するディレクトリのみを優先ディレクトリに指定すると，優先処理は非優先ファイルにもアクセスすることとなり，優先処理の実行処理時間が増加する可能性がある．

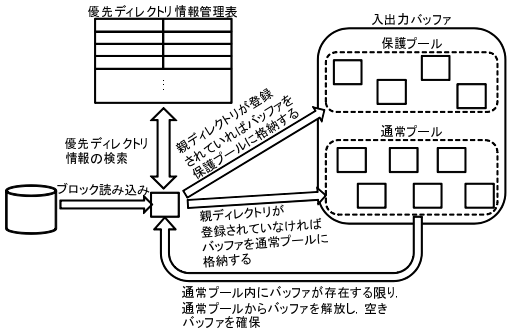


図 1 ディレクトリ優先方式の基本方式
Fig. 1 Directory oriented buffer cache mechanism.

（問題点 2） 非優先処理の実行処理時間の許容以上の増加

保護プールの分割サイズは単調増加するため，非優先ファイルのキャッシュヒット率が低下し，非優先処理の実行処理時間が許容以上に増加する可能性がある．

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 設 計 方 針

設計方針として，以下の 3 つがある．

（方針 1） 保護プールの分割サイズの上限を設定

保護プールの分割サイズの上限（以降，保護プール上限サイズ（S_{max}）と略す）を導入し，保護プールの分割サイズを制限する．S_{max} の単位は，バッファ数である．これにより，通常プールの領域を確保でき，非優先ファイルのキャッシュヒット率の低下を防ぐことができる．また，現在の保護プールの分割サイズを保護プール現サイズ（S_{cur}）と呼ぶ．本方式は，$S_{cur} = S_{max}$ となるように制御する．

（方針 2） 保護プールの分割サイズの緩やかな制限

$S_{cur} > S_{max}$ の場合，S_{cur} を S_{max} まで小さくする必要がある．このためには，$(S_{cur} - S_{max})$ 分のバッファを保護プールから解放する必要がある．しかし，S_{max} と S_{cur} を常に一致させるためには，保護プールからバッファを直ちに解放する必要があり，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．このため，ブロック読み込み時に空きバッファが必要に

なった時，保護プールからバッファを解放し，徐々に S_{cur} を S_{max} に近づける．これにより，空きバッファが必要になるときまでバッファの解放を遅らせることができる．

（方針 3） 保護プールと通常プールのキャッシュヒット率に着目して S_{max} を決定し，更新

保護プールと通常プールの両プールにおいて，キャッシュヒット率が低くなりすぎないように，S_{max} を決定する必要がある．そこで，両プールでのキャッシュヒット率に着目して S_{max} を自動的に決定し，更新する．

3.2 課　　題

課題として以下の 3 つがある．

（課題 1） バッファの解放規則

（方針 1）と（方針 2）より，ブロック読み込み時に，S_{cur} を S_{max} まで増加，または減少させるため，バッファの解放規則を変更する必要がある．

（課題 2） 分割サイズ決定の契機

（方針 3）より，保護プール，または通常プールのキャッシュヒット率が低いと判定した場合，S_{max} を更新する．そこで，キャッシュヒット率が低いと判定する方法が課題となる．

（課題 3） 分割サイズ決定法

（方針 3）より，更新する S_{max} の値の決定方法が課題となる．

3.3 対　　処

3.3.1 バッファの解放規則

S_{max} と S_{cur} を用いたバッファの解放規則を以下で述べる．なお，本解放規則は，文献 2) で提案した解放規則である．

(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)　$S_{cur} < S_{max}$ の場合

保護プールを大きくできるため，通常プールからバッファを解放する．優先ファイルへのアクセス時であれば，ブロック読み込み後，バッファを保護プールに格納する．これにより，S_{cur} が大きくなる．

(b)　$S_{cur} = S_{max}$ の場合

保護プールの分割サイズを変更できないため，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決定する．優先ファイルのブロックであれば保護プールから，非優先ファイルのブロックであれば通常プールからバッファを解放する．

(c)　$S_{cur} > S_{max}$ の場合

保護プールを小さくする必要があるため，保護プールからバッファを解放する．非優先ファイルへのアクセス時であれば，ブロック読み込み後，バッファを通常プールに格納する．これにより，S_{cur} が小さくなる．

(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 LRU 方式に従いバッファを解放する．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 方式に従いバッファを解放する．

表 1　分割サイズ決定法
Table 1　Method to decide increment and decrement of S_{max}.

S_{max} の増加量の決定方法	（方法 1）(α − 保護プールのキャッシュヒット率) × 優先ファイルのブロックへのアクセス数 （方法 2）(入出力バッファ内に保持できるバッファ数 − 現在の S_{max}) の x% （通常プール内のバッファ数 $\geq N$ を維持）
S_{max} の減少量の決定方法	（方法 1）(β − 通常プールのキャッシュヒット率) × 非優先ファイルのブロックへのアクセス数 （方法 2）現在の S_{max} の y% （保護プール内のバッファ数 $\geq M$ を維持）

3.3.2 分割サイズ決定の契機

ω 回のブロックアクセス毎に，保護プールと通常プールのキャッシュヒット率を確認する．このとき，保護プールのキャッシュヒット率が保護プールのキャッシュヒット率の閾値 α 未満である場合，S_{max} を増加させる．これにより，保護プールが拡大し，キャッシュヒット率が向上する．同様に，通常プールのキャッシュヒット率が通常プールのキャッシュヒット率の閾値 β 未満である場合，S_{max} を減少させる．これにより，通常プールが拡大し，キャッシュヒット率が向上する．保護プールと通常プールの両方において，キャッシュヒット率が閾値未満となった場合，保護プールのキャッシュヒット率が閾値 α 未満であることを優先し，S_{max} を増加させる．これは，保護プールは優先処理が頻繁にアクセスするバッファを多く保持するため，保護プールのキャッシュヒット率が低いと，優先処理の実行処理時間が増加する可能性が高いためである．

3.3.3 分割サイズ決定法

分割サイズ決定法を**表 1** に示す．

（方法 1）での S_{max} の増加量決定の例を**図 2** に示す．図 2 では，保護プールのキャッシュヒット率 $= 3/10$，$\alpha = 7/10$ であるため，さらに 4 回キャッシュヒットする必要があった．このため，S_{max} を 4 大きくする．同様に，（方法 1）では，通常プールのキャッシュヒット率と β に基づき，S_{max} を減少させる．

（方法 2）は，S_{max} を（入出力バッファ内に保持できるバッファ数 − 現在の S_{max}）の x%増加，現在の S_{max} の y%減少させる．この方法により，保護プール，または通常プールの分割サイズが急激に減少する

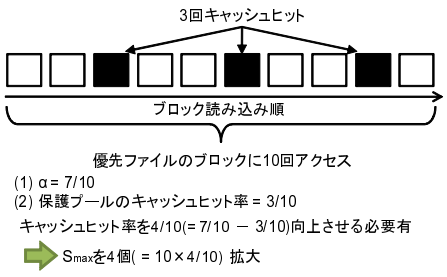


図 2 （方法 1）での S_{max} の増加量決定の例
Fig. 2 Example of decision of increment of S_{max} by (method 1).

ことを防止できる．

両プールで最低限必要なバッファを確保するため，保護プールと通常プールの分割サイズは，それぞれ M，N 以上を維持する．

4. 評　　価

4.1 優先処理の実行処理時間短縮の評価

4.1.1 評 価 方 法

優先処理としてカーネル make を実行し，実行処理時間を FreeBSD 4.3-RELEASE（以降，FreeBSD 4.3-R と略す）に元から実装されている LRU 方式（LRU），ディレクトリ優先方式の基本方式（以降，基本方式（基本）と略す），および提案方式（提案）と比較する．

測定前に make depend を実行した．提案方式では，make depend 実行前に S_{max} を 0 に初期化した．基本方式と提案方式では，make depend 実行完了直後に /usr/src/sys/sys/と/usr/src/sys/i386/include/を優先ディレクトリに指定した．この 2 つのディレクトリは，直下にヘッダファイルを有するディレクトリであり，カーネル make は直下のファイルの内，約 1.2MB 強にアクセスする．カーネル make は，ヘッダファイルを繰り返し読み込み，この 2 つのディレクトリ直下のヘッダファイルは，他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 44MB である．測定は 3 回行い，その平均値を評価に用いた．

4.1.2 評 価 環 境

計算機（CPU : Celeron 2.0GHz，メモリ : 768MB，OS : FreeBSD 4.3-R，VMIO : オフ，1 バッファのサイズ : 8.0KB）を用いて評価した．入出力バッファの制御方式の性能が問題になるのは，入出力バッファサイズがアクセスするファイルの総サイズよりも小さく，キャッシュミスが起こる場合である．このため，制御方式の違いによる性能の差を明確にするため，入出力バッファサイズを小さく制限し，3.0MB と 6.3MB の場合について測定した．入出力バッファには，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 0.7MB ほど小さい．入出力バッファサイズが 3.0MB の場合は 296 個，入出力バッファサイズが 6.3MB の場合は 720 個のバッファを保持できる．

4.1.3 パラメータ

パラメータと評価で利用した設定値を**表 2** に示す．S_{max} の増加量/減少量の決定方法に（方法 1）を用いた場合，(ω，α，β，M，N) = (入出力バッファに保持できるバッファ数, 95, 90, 32, 32) を基本とし，ω，α，β，M，N の順に，設定値を様々な値に変化させ，適切な値を探索する．つまり，α を変化させる影響の評価では，ω に探索した適切な値を設定，β を変化させる影響の評価では，ω と α に探索した適切な値を設定というように，順番に適切な設定を探索する．ただし，M の変化の影響の評価では，N に入出力バッファに保持できるバッファ数の 10%を設定した．優先ファイルは頻繁にアクセスされるため，α を 95%と高い値にした．優先ファイルと比べ，非優先ファイルはアクセス頻度が低いため，β は α より低い 90%とした．また，ブロックの先読みの最大量が 32 ブロックであるため，M と N を 32 個とした．

（方法 2）を用いた場合，上記 5 つに加えて，x，y の 2 つのパラメータが必要となる．（方法 2）を用いた場合の評価では，ω，α，β，M，および N に（方法 1）での適切な値を設定した上で，x，y の順に適切な値を探索する．x の適切な値を探索する際，y に 20%を設定した．これは，y の適切な値を探索する際に y に設定した値の中央値である．

4.1.4 キャッシュヒット率を確認する周期を変化させる影響

本項から 4.1.8 項までは，S_{max} の増加量/減少量の決定方法に（方法 1）を用いた場合の評価結果である．

図 3 に，ω を変化させた場合のカーネル make の実行処理時間の変化を示す．入出力バッファサイズが 3.0MB の場合，ω = 296（入出力バッファに保持できるバッファ数）で最も実行処理時間が短い．入出力バッファサイズが 6.3MB の場合，ω = 5760（入出力バッファに保持できるバッファ数の 8 倍）で最も実行処理時間が短い．ただし，入出力バッファサイズが 6.3MB の場合，ω = 720（入出力バッファに保持できるバッファ数）の場合と ω = 5760 の場合の差は，3.8 秒（1.0%）と小さい．また，ω = 720 と ω = 5760 の両方で，次項以降の測定を行った結果，ω = 720 の方が実行処理時間が短くなった．これは，ω が大きす

表 2 パラメータの設定値
Table 2 Parameters in evaluation with kernel make.

パラメータ	説明	設定値
ω	キャッシュヒット率を確認する周期	(1) 入出力バッファサイズ 3.0MB（296 個）の場合 148，296，592，1184，2368 (2) 入出力バッファサイズ 6.3MB（720 個）の場合 360，720，1440，2880，5760，8640，11520
α	保護プールでのキャッシュヒット率の閾値	90%，95%，100%
β	通常プールでのキャッシュヒット率の閾値	80%，85%，90%，95%，100%
M	保護プールサイズの下限	入出力バッファに保持できるバッファ数の 5%，10%，20%，30%，40%，50%
N	通常プールサイズの下限	入出力バッファに保持できるバッファ数の 5%，10%，20%，30%，40%，50%
x	S_{max} の増加量	5%，10%，20%，30%，40%
y	S_{max} の減少量	5%，10%，20%，30%，40%

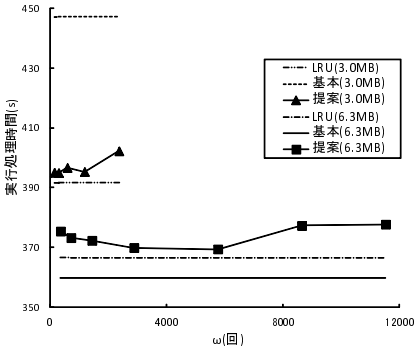


図 3 カーネル make の実行処理時間（周期 ω を変化）
Fig. 3 Execution time of kernel make (in several ω).

ぎると，S_{max} の更新回数が減少し，アクセス状況にあった分割ができなくなるためであると考えられる．

このため，以降では，入出力バッファサイズが 6.3MB の評価では，$\omega = 720$ での結果を示す．

4.1.5 保護プールでのキャッシュヒット率の閾値を変化させる影響

図 4（a）に閾値 α を変化させた場合のカーネル make の実行処理時間の変化を示す．図 4（a）より，入出力バッファサイズ 3.0MB では $\alpha = 95\%$，入出力バッファサイズ 6.3MB では $\alpha = 100\%$ で最も実行処理時間が短いことがわかる．これは，次の理由による．入出力バッファサイズが 3.0MB の場合，入出力バッファサイズが小さいため，保護プールサイズを制限し，通常プールの領域を確保する必要がある．このため，保護プールが大きくなりすぎないよう，$\alpha < 100\%$ とした方が良い．一方，入出力バッファサイズが 6.3MB の場合，読み込む優先ファイルの総サイズに対して，入出力バッファが十分大きい．このため，保護プールに必要なだけバッファを割り当てることができるよう，$\alpha = 100\%$ とした方が良い．

4.1.6 通常プールでのキャッシュヒット率の閾値を変化させる影響

図 4（b）に β を変化させた場合のカーネル make の実行処理時間の変化を示す．図 4（b）より，入出力バッファサイズが 3.0MB と 6.3MB である場合で共に，$\beta = 85\%$ の場合が最も実行処理時間が短いことがわかる．また，β を大きくすると，実行処理時間が増加している．これは，β を大きくすると，通常プールが大きくなり，優先ファイルのキャッシュヒット率が低下するためである．

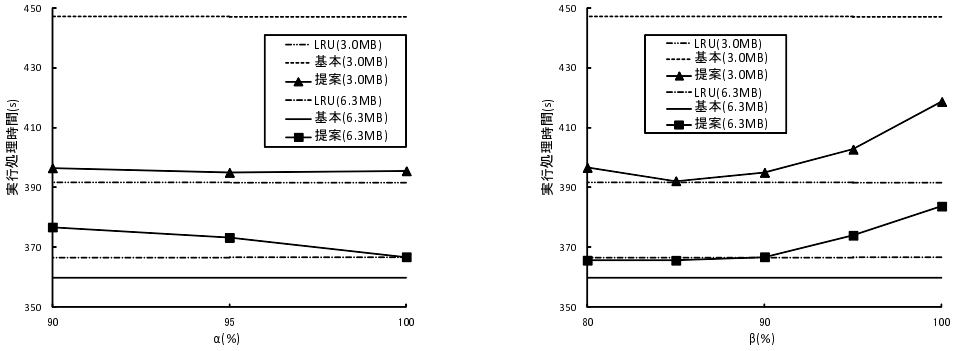
4.1.7 保護プールサイズの下限を変化させる影響

図 5（a）に M を変化させた場合のカーネル make の実行処理時間の変化を示す．図 5（a）より，入出力バッファサイズが 3.0MB の場合，$M = 89$（入出力バッファ内に保持できるバッファ数の 30%）で最も実行処理時間が短いことがわかる．また，M を大きくしすぎると，実行処理時間が増加することがわかる．これは，優先ファイルの総サイズに対して，入出力バッファが小さいため，常に保護プールが大きいと，非優先ファイルのキャッシュヒット率が向上しないためであると推察できる．

一方，入出力バッファサイズが 6.3MB の場合，$M = 216$（入出力バッファ内に保持できるバッファ数の 30%）で最も実行処理時間が短いことがわかる．また，M を小さくしすぎると，実行処理時間が増加することがわかる．これは，非優先ファイルにアクセスが集中した際に，保護プールが小さくなるためであると推察できる．

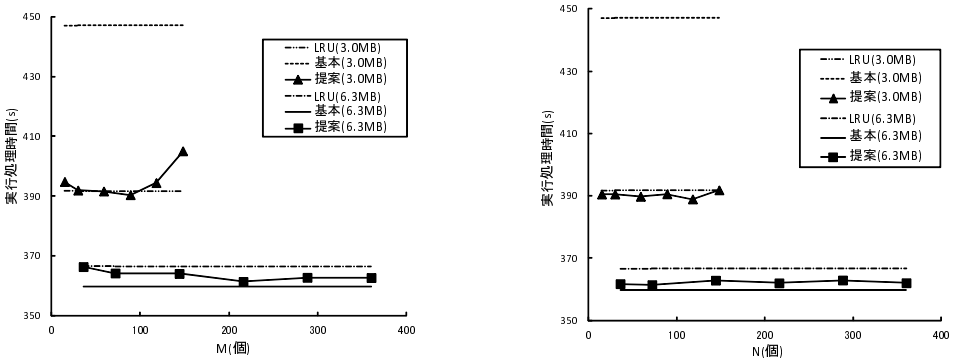
4.1.8 通常プールサイズの下限を変化させる影響

図 5（b）に N を変化させた場合のカーネル make の実行処理時間の変化を示す．図 5（b）より，N の変化に伴う実行処理時間の変化は小さいことがわかる．入出力バッファサイズが 3.0MB の場合，$N = 118$（入



(a) α を変化　　(b) β を変化

図 4　カーネル make の実行処理時間（α，または β を変化）
Fig. 4　Execution time of kernel make (in several α or β).



(a) M を変化　　(b) N を変化

図 5　カーネル make の実行処理時間（M，または N を変化）
Fig. 5　Execution time of kernel make (in several M or N).

出力バッファ内に保持できるバッファ数の 40%）で最も実行処理時間が短く，LRU 方式と比べて 2.8 秒（0.71%），基本方式と比べて 58 秒（13%）短縮している．また，入出力バッファサイズが 6.3MB の場合，$N = 72$（入出力バッファ内に保持できるバッファ数の 10%）で最も実行処理時間が短く，LRU 方式と比べて 5.2 秒（1.4%）短縮，基本方式と比べて 1.6 秒（0.44%）増加している．

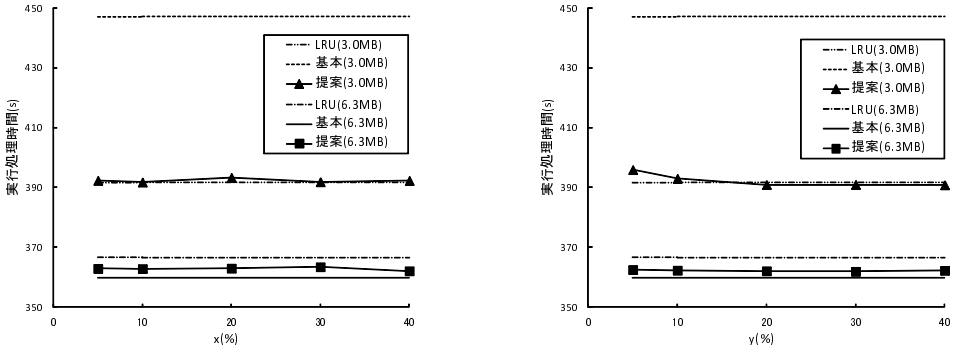
上記の結果より，S_{max} の増加量/減少量の決定方法に（方法 1）を用いることで，基本方式と比べ，カーネル make の実行処理時間を短縮，または同等の結果を得ることができたといえる．また，提案方式は，(方法 1）により，入出力バッファサイズ 3.0MB の場合，基本方式が非優先ファイルのキャッシュヒット率の低下により，LRU 方式より実行処理時間が長くなっていた（問題点 1）を解決できたといえる．

4.1.9　保護プール上限サイズの増加量を変化させる影響

本項と次項は，S_{max} の増加量/減少量の決定方法に（方法 2）を用いた場合の評価である．**図 6**（a）に x を変化させた場合のカーネル make の実行処理時間の変化を示す．図 6（a）より，x の変化に伴う実行処理時間の変化は小さいことがわかる．入出力バッファサイズが 3.0MB の場合，$x = 10\%$ で最も実行処理時間が短い．入出力バッファサイズが 6.3MB の場合，$x = 40\%$ で最も実行処理時間が短い．

4.1.10　保護プール上限サイズの減少量を変化させる影響

図 6（b）に y を変化させた場合のカーネル make の実行処理時間の変化を示す．図 6（b）より，y の変化に伴う実行処理時間の変化は小さいことがわかる．入出力バッファサイズが 3.0MB の場合，$y = 40\%$ で最も実行処理時間が短く，LRU 方式と比べて 0.86 秒（0.22%），基本方式と比べて 56 秒（13%）短縮して



(a) x を変化　　(b) y を変化

図 6　カーネル make の実行処理時間（x，または y を変化）
Fig. 6　Execution time of kernel make (in several x or y).

いる．また，入出力バッファサイズが 6.3MB の場合，$y = 30\%$ のときに最も実行処理時間が短く，LRU 方式と比べて 4.6 秒（1.3%）短縮，基本方式と比べて 2.1 秒（0.59%）増加している．

上記の結果から，S_{max} の増加量/減少量の決定方法に（方法 2）を用いた場合，(方法 1）を用いた場合と比べ，やや実行処理時間が増加しているといえる．

4.1.11　静的に S_{max} を手動設定する場合との比較

文献 2) では，静的に S_{max} を手動で設定し，評価した．そこで，文献 2) で最もカーネル make の実行処理時間を短縮できた場合と提案方式を比較する．なお，文献 2) では，利用者に使いやすいインタフェースを提供するため，S_{max} の単位をデータサイズ（Byte）としている．しかし，提案方式では，S_{max} を自動的に設定するため，この点を考慮する必要がない．このため，提案方式での S_{max} の単位を入出力バッファの制御単位であるバッファ数とした．

提案方式では，S_{max} の増加量/減少量の決定方法に（方法 1）を用いた場合，LRU 方式と基本方式と比べ，カーネル make の実行処理時間を短縮，または同等の結果を得ることができた．しかし，入出力バッファサイズが 3.0MB の場合，手動設定時と比べ，0.42 秒（0.11%）増加している．また，入出力バッファサイズが 6.3MB の場合，手動設定時と比べ，2.9 秒（0.82%）増加している．

上記の結果から，提案方式を用いた場合，適切な値を S_{max} に設定した場合と比べ，優先処理の性能がやや低下する可能性があると考えられる．しかし，自動的に S_{max} を設定することで，利用者が適切な S_{max} を設定する必要がなく，利便性が向上すると考えられる．また，S_{max} を動的に設定することで，アクセスパターンの変化に対応できると考えられる．

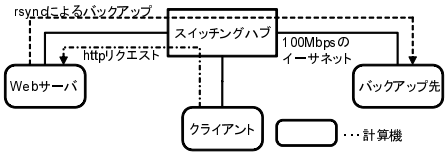


図 7　Web サーバによる評価環境
Fig. 7　Environment of Web server evaluation.

表 3　Web サーバによる評価に用いた計算機
Table 3　Computers used in Web server evaluation.

	Web サーバ	クライアント	バックアップ先
CPU	Celeron D 2.8GHz	Celeron D 2.8GHz	Celeron 2.0GHz
メモリ	32MB	256MB	768MB
入出力バッファ	6.4MB	32MB	87MB
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	8.0KB	16.0KB	16.0KB

4.2　非優先処理の実行処理時間短縮の評価

4.2.1　評価環境と評価方法

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について評価する．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 3** に示す．

前節で述べたように，入出力バッファの制御方式の性能が問題になるのは，入出力バッファサイズがアクセスするファイルの総サイズよりも小さい場合である．このため，Web サーバを実行する計算機が認識できるメモリ量を 32MB に制限した．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．

Web サーバとして Apache 2.0.55 を用い，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．

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，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た．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 100ms）をほぼ同時に開始した．また，提案方式では，最初の 100,000 回のアクセスの前に，S_{max} を 0 に初期化した．バックアップ処理は，岡山大学の Web サーバが持つファイルをバックアップする．100,000 回のアクセスよりバックアップ処理が早く終了したため，Web サーバの応答時間の測定結果をバックアップ処理動作中のものとバックアップ処理終了後のものに分割して集計した．本稿では，バックアップ処理動作中の Web サーバの応答時間の測定結果を示す．

Web サーバには，入り口となるページ（以降，ホームページと略す）が存在する．Web サービスにおいて，ホームページの表示に要する時間を短縮することは重要である．このため，文献 1) において，ディレクトリ優先方式を用い，ホーム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することにより，ホームページの表示に要する時間を短縮できることを示した．ここで，ホームページを構成するファイルとは，ホームページの HTML ファイルとこの HTML ファイルから参照される画像ファイルのことである．本評価では，岡山大学，文学部，教育学部，医学部保健学科，農学部，および社会文化科学研究科の 6 部局のホーム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 4** に示す．さらに，バックアップしたファイルの情報を**表 5** に示す．

なお，岡山大学の Web サーバが持つ 6 つの部局のホーム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．また，提案方式の S_{max} の増加量/減少量の決定方法には，カーネル make の実行処理時間をより短縮できた（方法 1）を用い，評価する．

表 4 評価で用いたファイルの情報 (100,000 回要求)
Table 4 Files information where the total access number is 100,000.

	全体	優先ファイル	ホーム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

表 5 バックアップしたファイルの情報
Table 5 Information of files backup accessed.

	全体	優先ファイル
合計ファイルサイズ (MB)	2818.5	13.9

4.2.2 パラメータ

提案方式のパラメータは以下のように設定した．

(1) ω:734（入出力バッファに保持できるバッファ数）
前節の評価において，ω = 入出力バッファに保持できるバッファ数でカーネル make の実行処理時間を短縮できたためである．

(2) α：100%
6 部局のホームページを速く表示させたいためである．

(3) β：50%，55%，60%，70%，80%，85%

(4) M：367，440，514
6 部局のホームページを速く表示させたいため，入出力バッファの半分以上を保護プールとして利用可能にした．

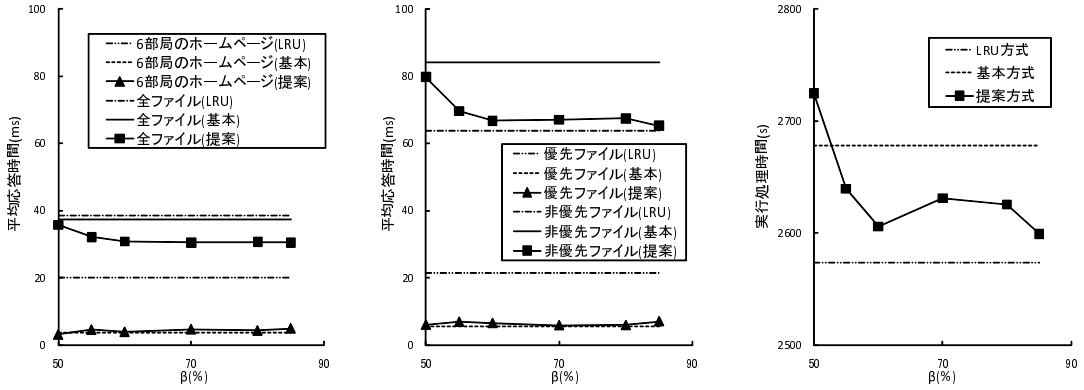
(5) N：32
先読みブロック数の最大数である 32 を設定し，最低限必要なバッファを確保した．

本評価では，β，M に様々な値を設定して測定し，適切な値を探索する．このとき，β，M の順に適切な値を探索する．

4.2.3 通常プールでのキャッシュヒット率の閾値を変化させる影響

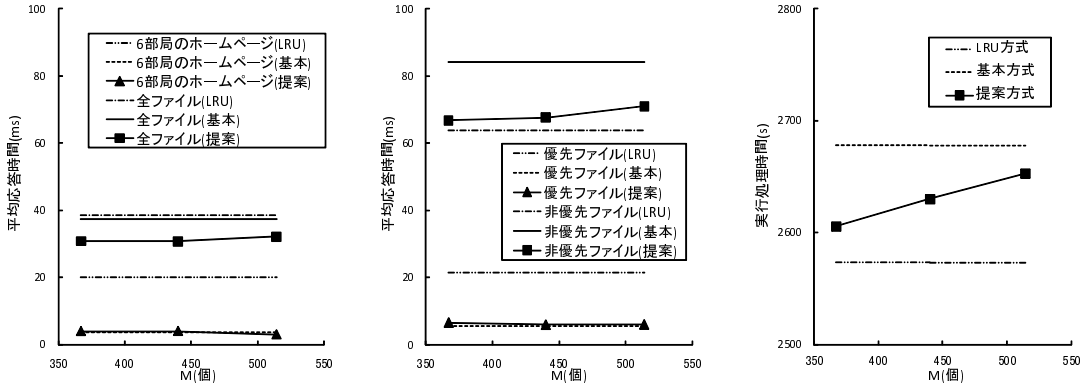
バックアップ動作中の 6 部局のホームページを構成するファイルと全ファイルの平均応答時間を**図 8**（a）に，優先ファイルと非優先ファイルの平均応答時間を図 8（b）に示す．また，バックアップ処理の実行処理時間を図 8（c）に示す．

図 8（a）より，提案方式では，$\beta = 50\%$ の場合，6 部局のホームページを構成するファイルの応答時間を最も短縮したことがわかる．これは，β を小さくする



(a) 6 部局のホームページを構成するファイルと全ファイルの平均応答時間　(b) 優先ファイルと非優先ファイルの平均応答時間　(c) バックアップ処理の実行処理時間

図 8　β を変化させた場合の測定結果（バックアップ処理動作中）
Fig. 8　The response time of web server while a backup is running, and execution time of backup (in several β).



(a) 6 部局のホームページを構成するファイルと全ファイルの平均応答時間　(b) 優先ファイルと非優先ファイルの平均応答時間　(c) バックアップ処理の実行処理時間

図 9　M を変化させた場合の測定結果（バックアップ処理動作中）
Fig. 9　The response time of web server while a backup is running, and execution time of backup (in several M).

ことで通常プールが大きくならず，保護プールサイズを大きく保つことができたためであると考えられる．しかし，$\beta = 50\%$ の場合，図 8（c）より，バックアップ処理の実行処理時間が増加し，基本方式より長い．また，図 8（b）より，非優先ファイルへの要求の平均応答時間が増加している．

したがって，50%の場合に次いで，6 部局のホームページを構成するファイルの応答時間が短い 60%を次項の M を変化させる評価で β に設定する．このとき，バックアップ処理の実行処理時間を基本方式と比べて短縮でき，非優先ファイルの平均応答時間も基本方式より短い．

4.2.4　保護プールサイズの下限を変化させる影響

バックアップ動作中の 6 部局のホームページを構成するファイルと全ファイルの平均応答時間を**図 9**（a）に，優先ファイルと非優先ファイルの平均応答時間を図 9（b）に示す．また，バックアップ処理の実行処理時間を図 9（c）に示す．

図 9（a）より，提案方式は，$M = 514$ 個の場合，最も 6 部局のホームページを構成するファイルの応答時間を短縮できている．これは，M を大きくすることにより，保護プールが小さくなりすぎることを防止できたためであると考えられる．このとき，LRU 方式と比べて 17 ミリ秒（85%），基本方式と比べて 0.79 ミリ秒（21%）短縮できている．

このとき，図 9（c）より，バックアップ処理の実行処理時間を短縮し，基本方式より 25 秒（0.9%）短い．これは，保護プールサイズを制限することにより，通

常プールの領域を確保でき，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たためである．また，図 9（b）より，非優先ファイルの平均応答時間を基本方式と比べ，13 ミリ秒（16%）短縮した．これにともない，全ファイルの平均応答時間を基本方式と比べ，5.2 ミリ秒（14%）短縮した．

上記の結果から，6 部局のホームページを構成するファイルの応答時間を LRU 方式と比べて大きく短縮できており，かつバックアップ処理の実行処理時間と非優先ファイルの応答時間を短縮できている．この結果から，非優先処理の実行処理時間の増加を抑制できており，(問題点 2）を解決できたといえる．

4.2.5 静的に S_{max} を手動設定する場合との比較

文献 2) の評価では，$S_{max} = 2.3$MB の場合，6 部局のホームページを構成するファイルの応答時間を基本方式と比べて増加させることなく，バックアップ処理の実行処理時間と非優先ファイルの応答時間を短縮できた．

前項までで述べたように，提案方式は，6 部局のホームページを構成するファイルの応答時間を基本方式と比べて増加させることなく，バックアップ処理の実行処理時間と非優先ファイルの応答時間を基本方式と比べて短縮できた．このとき，提案方式は，$S_{max} = 2.3$MB の場合と比べてバックアップ処理の実行処理時間が 26 秒（1.0%）増加した．また，このとき，6 部局のホームページを構成するファイルの応答時間では，提案方式は 0.05 ミリ秒（1.8%）短く，同等の結果が得られたといえる．さらに，提案方式は，非優先ファイルの平均応答時間が短く，2.5 ミリ秒（3.4%）短縮した．これにともない，全ファイルの平均応答時間を 0.75 ミリ秒（2.3%）短縮した．

S_{max} を手動設定する場合と比べ，バックアップ処理の実行処理時間は増加した．しかし，提案方式には，4.1.11 項で述べた通り，S_{max} を動的に自動で設定する利点があり，有用だといえる．

5. 関 連 研 究

入出力バッファを分割して管理する方式[3)~11)] が提案されている．この内，ARC[3)]，CAR[4)]，UBM[5)]，PCC[6)]，および Karma[7)] は，分割領域のサイズを自動的に決定する．

ARC[3)] と CAR[4)] は，入出力バッファを 2 つの領域に分割する．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．このため，破棄されたブロックの情報を保持する必要があり，制御に要する情報量が大きい．

UBM[5)] と PCC[6)] は，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，アクセスパターン毎に領域を割り当てる．Karma[7)] 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の分割領域を割り当てる．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 1 ブロックずつ変更する．このため，キャッシュヒット率の向上を予測するため，オーバヘッドが大きい．

提案方式は，優先ファイルと非優先ファイルのアクセス回数とキャッシュヒット回数，および制御パラメータを保持するのみで良く，ARC と CAR のように，制御のために多くの情報を保持する必要が無い．また，UBM と PCC のように，ブロックアクセス時にオーバヘッドの大きい計算を行う必要がない．

また，提案方式は，優先処理のファイルアクセスと非優先処理のファイルアクセスを区別した利用が可能である．しかし，他の方式では，これらを区別しないため，入出力バッファの分割の観点と方法が異なる．

6. お わ り に

ディレクトリ優先方式に基づく入出力バッファの制御において，キャッシュヒット率に基づき，入出力バッファを分割する方式について述べた．本方式は，優先ファイルのキャッシュヒット率が低い場合，優先ファイルをキャッシュする領域の分割サイズを増加させる．同様に，非優先ファイルのキャッシュヒット率が低い場合，非優先ファイルをキャッシュする領域の分割サイズを増加させる．この増加量の決定方法として，キャッシュヒット率とキャッシュヒット率の閾値にも基づく方法，および現在の分割サイズと入出力バッファ内に保持できるバッファ数に基づく方法について述べた．

カーネル make を用いた実行処理時間の評価では，提案方式を用いることで，ディレクトリ優先方式の基本方式と比べ，カーネル make の実行処理時間を最大で 58 秒（13%）短縮できた．また，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では，ディレクトリ優先方式の基本方式と比べ，非優先処理であるバックアップ処理の実行処理時間を 25 秒（0.9%）短縮できた．このとき，ホーム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 17 ミリ秒（85%），基本方式と比べて 0.79 ミリ秒（21%）短縮できた．

残された課題として，提案方式のパラメータを決定

する方法の確立，およびカーネル make や Web サーバ以外の AP を用いたより詳細な評価がある．また，評価では，提案方式によってアクセスパターンの変化へ対応し，適切な分割サイズに更新できているか否か評価する必要がある．

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, 山内利宏, 谷口秀夫: 優先/非優先処理の実行時間を短縮する入出力バッファ分割法, コンピュータシステム・シンポジウム論文集, Vol.2010, No. 13, pp. 39–46 (2010).
3) Megiddo, N. and Modha, D. S.: ARC: A SELF-TUNING, LOWOVERHEAD REPLACEMENT CACHE, *Proc. the 2nd USENIX Conference on File and Storage Technologies (FAST '03)*, pp. 115–130 (2003).
4) Bansal, S. and Modha, D.S.: CAR: Clock with Adaptive Replacement, *Proc. the 3rd USENIX Conference on File and Storage Technologies (FAST '04)*, pp. 187–200 (2004).
5) 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. the 4th Symposium on Operating System Design and Implementation (OSDI 2000)*, pp. 119–134 (2000).
6) Gniady, C., Butt, A. R. and Hu, Y. C.: Program-Counter-Based Pattern Classification in Buffer Caching, *Proc. the 6th Symposium on Opearting Systems Design and Implementation (OSDI 2004)*, pp. 395–408 (2004).
7) Yadgar, G., Factor, M. and Schuster, A.: Karma: Know-it-All Replacement for a Multilevel cAche, *Proc. the 5th USENIX Conference on File and Storage Technologies (FAST '07)*, pp. 169–184 (2007).
8) Johnson, T. and Shasha, D.: 2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm, *Proc. the 20th International Conference on Very Large Databases*, pp. 439–450 (1994).
9) Jiang, S. and Zhang, X.: LIRS: An Efficient Low Inter-reference Recency Set Replacement Policy to Improve Buffer Cache Performance, *Proc. the 2002 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pp. 31–42 (2002).
10) Ding, X., Jiang, S. and Chen, F.: A Buffer Cache Management Scheme Exploiting Both Temporal and Spatial Localities, *ACM Transactions on Storage*, Vol. 3, Issue. 2, Article no. 5 (2007).
11) 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).