Vol.2011-OS-118 No.18
2011/7/28

キャッシュヒット率に着目した入出力バッファ分割法

土谷彰義[†1]　山内利宏[†1]　谷口秀夫[†1]

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により，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が提案されている．ディレクトリ優先方式は，指定したディレクトリ直下のファイル（優先ファイル）を優先的にキャッシュする．したがって，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，優先ファイルとその他のファイルをキャッシュする量として，最適な値を設定することは難しい．そこで，本稿では，キャッシュヒット率に着目し，動的に優先ファイルとその他のファイルのキャッシュ量を決定し設定する方式について述べる．

I/O Buffer Partitioning Method Based on Cache Hit Ratio

Akiyoshi TSUCHIYA,[†1] Toshihiro YAMAUCHI[†1] and Hideo TANIGUCHI[†1]

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. Blocks of important files are given a high priority for caching. Therefore, performance of high priority processing is improved. However, it is difficult to set the appropriate amount of cached blocks of important files. This paper proposes I/O buffer partitioning method based on cache hit ratio. This method dynamically decides and changes the amount of cached blocks of important and unimportant files.

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

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，実行処理時間を短縮できる．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，指定した特定のディレクトリ（以降，優先ディレクトリと略す）直下のファイルを優先的にキャッシュする．以降では，優先ディレクトリ直下のファイルを優先ファイル，優先ファイル以外のファイルを非優先ファイルと呼ぶ．

ディレクトリ優先方式には，非優先ファイルのキャッシュヒット率が低下することにより，優先処理と非優先処理の実行処理時間が増加する問題点が存在した．この問題点に対処するため，文献 2) では，キャッシュする優先ファイルのブロックの量に上限を設けることにより，優先ファイルと非優先ファイルの間で，キャッシュするブロックの量のバランス化を図る入出力バッファ分割法を提案し，有効性を示した．しかし，処理毎にアクセスするファイルやアクセスパターンが異なるため，上限に最適な値を設定することは難しい．そこで，本稿では，この上限を自動的に決定し設定する方式を提案する．

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[7)]，CAR[8)]，UBM[9)]，PCC[10)]，および Karma[11)] は，各領域の大きさを自動的に決定する．

ARC[7)] と CAR[8)] 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，各領域の大きさの決定に利用する．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[9)] と PCC[10)] 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[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，各領域の大きさを 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．

そこで，本稿では，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計測し，計測結果に基づき，キャッシュする優先ファイルのブロックの量の上限を自動的に決定し設定する

方式を提案する．提案方式は，優先ファイルのキャッシュヒット率が低い場合，キャッシュする優先ファイルのブロックの量の上限を増加させる．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，キャッシュする優先ファイルのブロックの量の上限を減少させる．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．

2. ディレクトリ優先方式

2.1 基本方式

文献 1) で提案され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，それぞれの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．保護プールの大きさは，入出力バッファサイズを超えない範囲で大きくなる．

優先ディレクトリの指定の手順を以下に示す．

(1) 優先処理実行前に，優先処理が頻繁にアクセスするファイルを調査し，それらを直下に多く有するディレクトリを明らかにする．
(2) 優先処理の実行直前に，上記ディレクトリを優先ディレクトリに指定する．
(3) 優先処理の終了直後に，優先ディレクトリの指定を解除する．

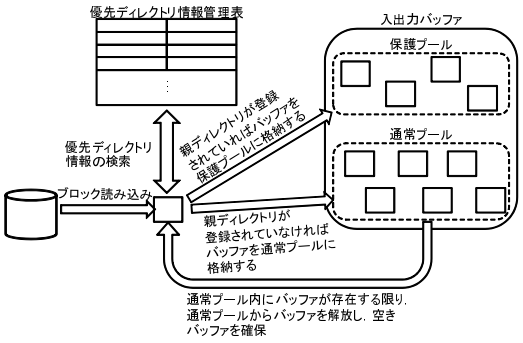


図 1 ディレクトリ優先方式の基本方式

2.2 入出力バッファ分割法

文献 2) で提案した入出力バッファ分割法について述べる．

保護プールに格納できるバッファの総サイズの上限（以降，保護プール上限サイズ（S_{max}）と略す）を導入し，保護プールの大きさを制限する．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．入出力バッファのキャッシュヒット率を向上させる設定を自由に行えるようにするため，S_{max} の値は，保護プール現サイズ（S_{cur}）の値にかかわらず，入出力バッファサイズの範囲内で利用者が任意の契機で設定する．このため，S_{max} は S_{cur} 以上とは限らず，S_{cur} より小さい値を指定できる．

S_{max} と S_{cur} を用いた入出力バッファ分割法を**図 2** に示し，以下で述べる．

(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，保護プールが拡大し，S_{cur} が大きくなる．
  (b) $S_{cur} = S_{max}$ の場合
  保護プールの大きさ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．

Vol.2011-OS-118 No.18
2011/7/28

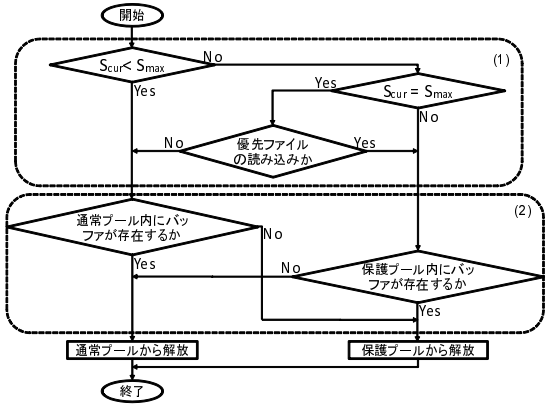


図 2 入出力バッファ分割法

(c) $S_{cur} > S_{max}$ の場合

保護プールを小さくしなければならないため，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，保護プールが縮小し， 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．

バッファ解放の規則に (2) が存在するのは，以下の 2 つの場合に (1) で選択したプール内にバッファが存在しないためである．

(1) $S_{cur} = S_{max} = 0$ の状態で，優先ファイルにアクセスした場合

(2) $S_{cur} = S_{max} =$ 入出力バッファサイズの状態で，非優先ファイルにアクセスした場合

2.3 入出力バッファ分割法の問題点

前節で述べた分割法では，高い性能を得るために，S_{max} に最適な値を設定する必要がある．しかし，処理毎にアクセスするファイルやアクセスパターンが異なるため，S_{max} に最適な値を設定することは難しい．S_{max} が大きすぎると，通常プールのキャッシュヒット率が低下し，S_{max} が小さすぎると，保護プールのキャッシュヒット率が低下する．

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 設 計 方 針

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，S_{max} を動的に再設定する．つまり，保護プールでのキャッシュヒット率が低い場合，S_{max} を増加させる．同様に，通常プールでのキャッシュヒット率が低い場合，S_{max} を減少させる．

3.2 課　　題

課題として以下の 2 つがある．

(課題 1) 分割サイズ決定の契機

保護プールと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に，S_{max} を再設定する．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 2) 分割サイズ決定法

キャッシュヒット率が低い場合，S_{max} を再設定する．そこで，再設定する S_{max} の値の決定方法が課題となる．

3.3 対　　処

3.3.1 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，保護プールの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，通常プールの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．

3.3.2 分割サイズ決定法

分割サイズ決定法を**表 1** に示す．

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) としていた．しかし，S_{max} の設定を自動化する場合，この点を考慮する

表 1 分割サイズ決定法

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 $-S_{max}$) の x% (通常プール内のバッファ数 $\geq M$ 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 $\geq N$ を維持)

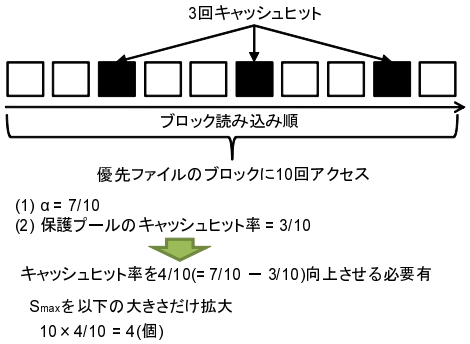


図 3 (方法 1) での S_{max} の増加量決定の例

必要がない．このため，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する．

(方法 1) での S_{max} の増加量決定の例を**図 3** に示す．図 3 では，周期 ω の間に 10 回優先ファイルのブロックにアクセスし，その内 3 回でキャッシュヒットしている．このとき，α が 7/10 であり，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 $-S_{max}$) 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サイズが急激に減少することを防止できる．

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サイズは，それぞれ M，N 以上を維持する．

4. 評　　価

4.1 評 価 方 法

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，文献 2) でのカーネル make の評価における最適な S_{max} の場合 (以降，最適な S_{max} の場合と略す)，およびキャッシュヒット率に基づき S_{max} を再設定する方式 (以降，提案方式と略す) と比較する．文献 2) での評価において，S_{max} に最適な値を設定することで，ディレクトリ優先方式の基本方式と比べてカーネル make の実行処理時間を短縮できることを示した．

測定前に make depend を実行した．make depend 実行前に，提案方式では S_{max} の初期値を 0 とした．最適な S_{max} の場合と提案方式では，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，直下のファイルの総データサイズは約 1.6MB である．しかし，実際にカーネル make で読み込まれるのは約 1.2MB 強である．カーネル make は，ヘッダファイルとソースファイルを読み込み，オブジェクトファイルと実行形式のカーネルを生成する．このうち，ヘッダファイルは繰り返し読み込まれ，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．

4.2 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，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．3.0MB と 6.3MB は，文献 2) で測定した入出力バッファサイズ (3.0MB，3.5MB，4.0MB，6.3MB) の内，最適な値を S_{max} に設定した場合，基本方式と比べてカーネル make の実行処理時間を最も短縮できたサイズ (3.0MB) と短縮できなかったサイズ (6.3MB) である．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．

表 2 パラメータの設定値

パラメータ	説明	設定値
ω	キャッシュヒット率を計測する周期	(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%

4.3 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，最適値を探索する．つまり，α を変化させる影響の評価では，ω に発見した最適値を設定，β を変化させる影響の評価では，ω と α に発見した最適値を設定というように，順番に最適な設定を探索する．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．

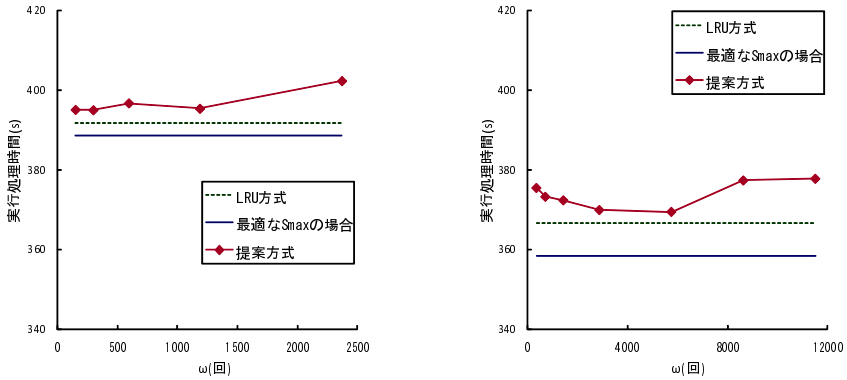
（方法 2）を用いた場合，上記 5 つに加えて，x，y の 2 つのパラメータが必要となる．（方法 2）を用いた場合の評価では，ω，α，β，M，および N に（方法 1）での最適値を設定した上で，x，y の順に最適値を探索する．x の最適値を探索する際，y に 20%を設定した．これは，y の最適値を探索する際に y に設定した値の中央値である．

4.4 評 価 結 果

4.4.1 キャッシュヒット率を計測する周期を変化させる影響

本項から 4.4.5 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 4 に，キャッシュヒット率を計測する周期 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．図 4 より，ω を大きくしすぎると実行処理時間が増加していることがわかる．これは，ω を大きくすると，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．



(a) 入出力バッファサイズ 3.0MB　(b) 入出力バッファサイズ 6.3MB

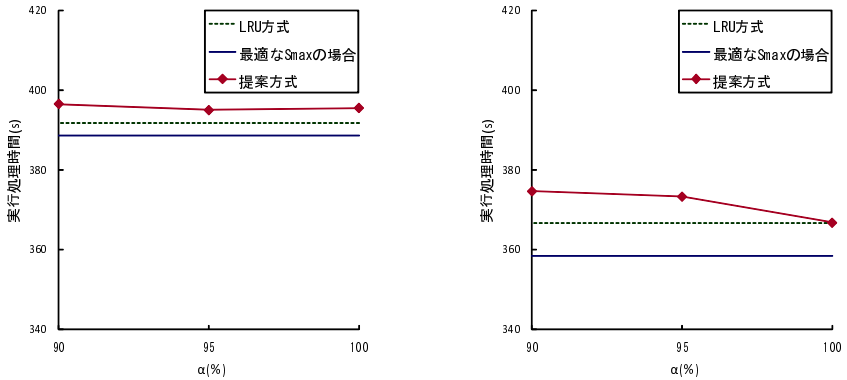
図 4 周期 ω を変化させた場合

入出力バッファサイズが 3.0MB の場合，$\omega = 296$（入出力バッファに保持できるバッファ数）のとき，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，$\omega = 5760$（入出力バッファに保持できるバッファ数の 8 倍）のとき，最も実行処理時間が短い．ただ，入出力バッファサイズが 6.3MB の場合，$\omega = 720$ とした場合と $\omega = 5760$ とした場合の差は，3.8 秒（1.0%）と小さい．また，$\omega = 720$ と $\omega = 5760$ の両方で，次項以降の測定を行った結果，$\omega = 720$ の方が実行処理時間が短くなった．このため，以降では，入出力バッファサイズが 6.3MB の場合でも，ω に 720（入出力バッファに保持できるバッファ数）を設定した結果を示す．

4.4.2 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

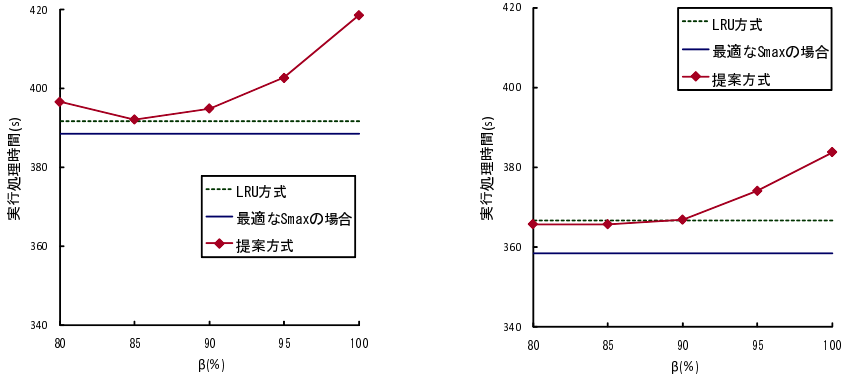
図 5 に保護プールでのキャッシュヒット率の閾値 α を変化させた場合のカーネル make の実行処理時間の変化を示す．

図 5 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 $\alpha = 100\%$ の場合に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



(a) 入出力バッファサイズ 3.0MB　(b) 入出力バッファサイズ 6.3MB

図 5　保護プールでのキャッシュヒット率の閾値 α を変化させた場合



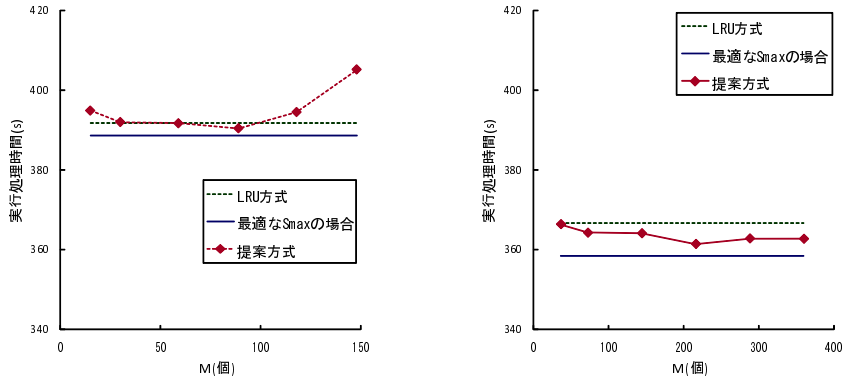
(a) 入出力バッファサイズ 3.0MB　(b) 入出力バッファサイズ 6.3MB

図 6　通常プールでのキャッシュヒット率の閾値 β を変化させた場合

くなりすぎないよう，$\alpha < 100\%$ とした方が良い．入出力バッファサイズが 6.3MB の場合，保護プールサイズを制限する必要がないため，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．

4.4.3　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

図 6 に通常プールでのキャッシュヒット率の閾値 β を変化させた場合のカーネル make の



(a) 入出力バッファサイズ 3.0MB　(b) 入出力バッファサイズ 6.3MB

図 7　保護プールサイズの下限 M を変化させた場合

実行処理時間の変化を示す．

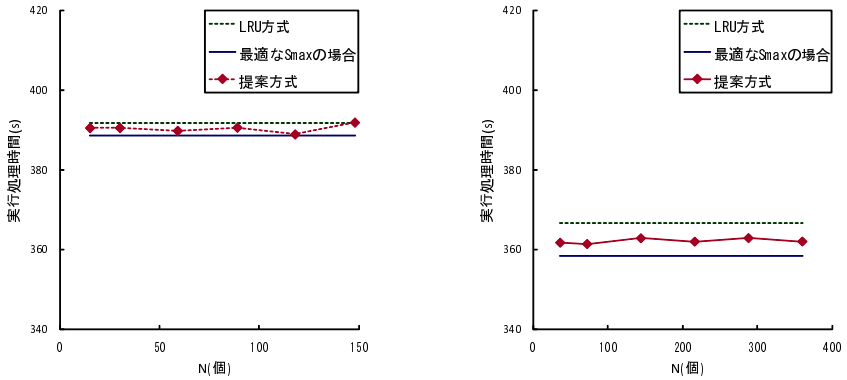
図 6 より，β を大きくすると，実行処理時間が増加していることがわかる．これは，β を大きくすると，通常プールのキャッシュヒット率を高めるために通常プールが大きくなり，優先ファイルのキャッシュヒット率が低下するためであると考えられる．入出力バッファが 3.0MB と 6.3MB である場合で共に，$\beta = 85\%$ の場合が最も実行処理時間が短い．

4.4.4　保護プールサイズの下限を変化させる影響

図 7 に保護プールサイズの下限 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．

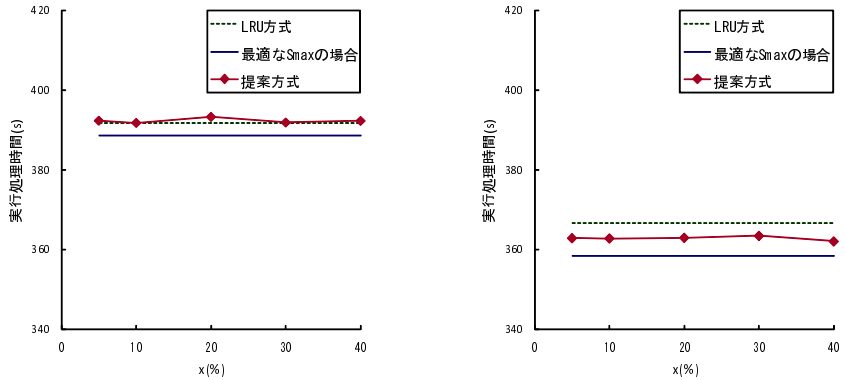
図 7（a）より，入出力バッファサイズが 3.0MB の場合，M を大きくしすぎると，実行処理時間が増加することがわかる．入出力バッファサイズが 3.0MB の場合，優先ファイルの総サイズに対して，入出力バッファが小さい．このため，常に保護プールが大きいと，非優先ファイルのキャッシュヒット率が向上しないためであると考えられる．入出力バッファサイズが 3.0MB の場合，$M = 89$（入出力バッファ内に保持できるバッファ数の 30%）のとき，最も実行処理時間が短い．

一方，図 7（b）より，入出力バッファサイズが 6.3MB の場合，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．入出力バッファサイズが 6.3MB の場合，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に大きい．このため，保護プールをなるべく大きくした方が，実行処理時間が短縮する．しかし，M を小さくした場合，非優先ファ



(a) 入出力バッファサイズ 3.0MB　(b) 入出力バッファサイズ 6.3MB

図 8 通常プールサイズの下限 N を変化させた場合



(a) 入出力バッファサイズ 3.0MB　(b) 入出力バッファサイズ 6.3MB

図 9 S_{max} の増加量 x を変化させた場合

イルにアクセスが集中した際に，保護プールが小さくなってしまうためであると考えられる．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%）のとき，最も実行処理時間が短い．

4.4.5 通常プールサイズの下限を変化させる影響

図 8 に通常プールサイズの下限 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 8 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（図 8（a））$M = 118$（入出力バッファ内に保持できるバッファ数の 40%）のときに最も実行処理時間が短い．このとき，LRU 方式と比べて 2.8 秒（0.71%）短縮，最適な S_{max} の場合と比べて 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，$M = 72$（入出力バッファ内に保持できるバッファ数の 10%）のときに最も実行処理時間を短い．このとき，LRU 方式と比べて 5.2 秒（1.4%）短縮，最適な S_{max} の場合と比べて 2.9 秒（0.82%）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 1）を用いると，入出力バッファサイズ 3.0MB では最適な S_{max} の場合と同等，入出力バッファサイズ 6.3MB では最適な S_{max} の場合よりやや長い結果が得られた．また，入出力バッファサイズ 6.3MB であっても，LRU 方式よりは実行処理時間が短い．

4.4.6 保護プール上限サイズの増加量を変化させる影響

本項以降は，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．（方法 2）を用いた場合，S_{max} を（入出力バッファ内に保持できるバッファ数 $-S_{max}$）の x%大きくする．**図 9** 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．

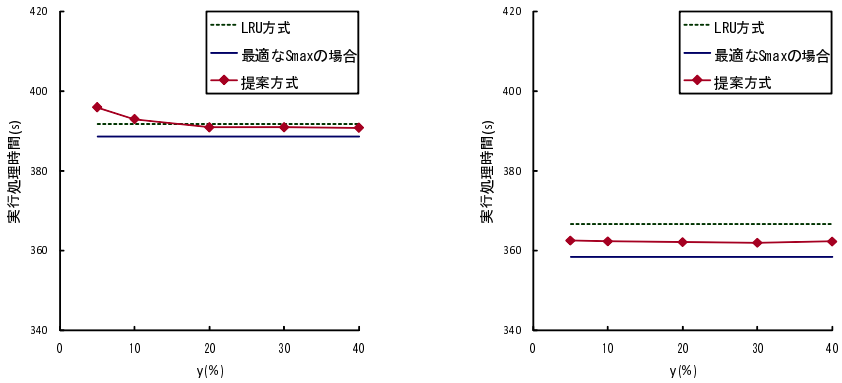
図 9 より，x の変化に伴う実行処理時間の変化は小さい．入出力バッファサイズが 3.0MB の場合，$x = 10\%$ のときに最も実行処理時間が短い．入出力バッファサイズが 6.3MB の場合，$x = 40\%$ のときに最も実行処理時間が短い．

4.4.7 保護プール上限サイズの減少量を変化させる影響

S_{max} の増加量/減少量の決定方法に（方法 2）を用いた場合，S_{max} を現在の S_{max} の y%小さくする．**図 10** 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 10 より，y の変化に伴う実行処理時間の変化は小さい．入出力バッファサイズが 3.0MB の場合，$y = 40\%$ のときに最も実行処理時間が短い．このとき，LRU 方式と比べて 0.86 秒（0.22%）短縮，最適な S_{max} の場合と比べて 2.3 秒（0.60%）増加している．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い．このとき，LRU 方式と比べて 4.6 秒（1.3%）短縮，最適な S_{max} の場合と比べて 3.5 秒（1.0%）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．



(a) 入出力バッファサイズ 3.0MB　　(b) 入出力バッファサイズ 6.3MB

図 10　S_{max} の減少量 y を変化させた場合

4.5　制御に要する情報量

ARC[7] と CAR[8] は，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，各領域の大きさの決定に利用する．この情報は，入出力バッファに保持できるバッファ数と同量保持する．このため，破棄されたブロックの情報として，整数型（4Byte）のブロック番号と i ノード番号を保持すると，入出力バッファサイズが 6.3MB（バッファ数 720 個）において，5760Byte の情報を保持する必要がある．

一方，提案方式では，優先ファイルと非優先ファイルそれぞれへのアクセス回数と，保護プールと通常プールでのキャッシュヒット回数を保持し，キャッシュヒット率を求める．また，S_{max}，S_{cur}，および優先ディレクトリの i ノード番号を保持する．さらに，S_{max} の増加量/減少量の決定方法に（方法 1）を用いた場合，整数型の ω，α，β，M，N，を保持し，S_{max} の増加量/減少量の決定方法に（方法 2）を用いた場合，これらに加えて整数型の x，y を保持する．よって，提案方式は，(方法 1）を用いた場合，(44 + 優先ディレクトリ数 ×4）Byte，(方法 2）を用いた場合，(52 + 優先ディレクトリ数 ×4）Byte，の情報を保持するのみで良く，ARC[7] と CAR[8] と比べて制御に要する情報量が少ないといえる．

5. 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，優先ファイルと非優先ファイルのキャッシュヒット率を計測し，計測結果に基づき，キャッシュする優先ファイルのブロックの量の上限を自動的に決定し設定する．優先ファイルのキャッシュヒット率が閾値より低い場合，この上限を増加させ，優先ファイルのブロックをキャッシュする量を増加させる．同様に，非優先ファイルのキャッシュヒット率が閾値より低い場合，この上限を減少させ，非優先ファイルのブロックをキャッシュする量を増加させる．このとき，一方のファイルのキャッシュヒット率が著しく低下することを防ぐため，優先ファイルのブロックと非優先ファイルのブロックを一定以上キャッシュできるようにした．上限の増加量/減少量の決定方法は，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いて決定する方法と，現在の上限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式について述べた．

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，LRU 方式と比べて，入出力バッファサイズが 3.0MB の場合 2.8 秒（0.71%），入出力バッファサイズが 6.3MB の場合 5.2 秒（1.4%）短縮できた．

残された課題として，提案方式で必要となるパラメータの決定方法の検討とカーネル make 以外の AP を用いたより詳細な評価がある．

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) 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).
4) 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).
5) Ding, X., Jiang, S. and Chen, F.: A Buffer Cache Management Scheme Exploiting Both Temporal and Spatial Localities, *ACM Transactions on Storage*, Vol.3, No.2 (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

Vol.2011-OS-118 No.18
2011/7/28

（ACS），Vol.3，No.1，pp.50–60 (2010).
7) 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).
8) 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).
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. the 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. the 6th Symposium on Opearting Systems Design and Implementation (OSDI 2004)*, pp.395–408 (2004).
11) 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).