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

土谷　彰義[†]　　松原　崇裕[††]　　山内　利宏[†]　　谷口　秀夫[†]

Proposal and Evaluation of Method to Set High Priority Directories for a Directory Oriented Buffer Cache Mechanism

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

あらまし　利用者が優先して実行したい処理（優先処理）の実行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．この方式は，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とし，優先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．しかし，優先処理の動作内容に関する知識をもたない場合，優先ディレクトリを適切に設定することは難しい．そこで，本論文では，効果的な優先ディレクトリを設定する手法を提案する．本設定法は，優先処理のファイルアクセスに関する情報から，ディレクトリの重要度を算出し，この重要度が高いディレクトリを優先ディレクトリに設定する．また，評価により，本設定法が，ファイルアクセス頻度の高いディレクトリを優先ディレクトリとして設定でき，優先処理の実行時間を短縮できることを示す．

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

1. まえがき

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．優先処理の実行時間を短縮する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，設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（以降，優先ファイルと略す）を優先的にキャッシュする．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設定することにより，優先処理の実行時間を短縮できる．

しかし，優先処理の動作内容に関する知識をもたない場合，優先ディレクトリを適切に設定することは難しい．このため，効果のない優先ディレクトリを指定すると，実行時間が増加する可能性がある．

そこで，本論文では，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を設定する手法を提案する．本設定法は，ディレクトリ重要度の概念を導入し，優先処理がアクセスするファイルのブロックアクセス回数やブロック数から，ディレクトリ重要度を算出する．次に，ディレクトリ重要度に基づき，優先ディレクトリを設定する．これにより，優先処理の動作内容に関する知識を必要とすることなく，効果的な優先ディレクトリを設定し，優先処理の実行時間を短縮できることを示す．

2. ディレクトリ優先方式

2.1 基本方式

文献[1]のディレクトリ優先方式について，基本方式を図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，通常プールには優先ファイル以外のファイル（以降，非優先ファイルと略す）のブロックを保持するバッファ

[†] 岡山大学大学院自然科学研究科，岡山市
Graduate School of Natural Science and Technology, Okayama University, Okayama-shi, 700–8530 Japan
[††] 岡山大学工学部，岡山市
Faculty of Engineering, Okayama University, Okayama-shi, 700–8530 Japan

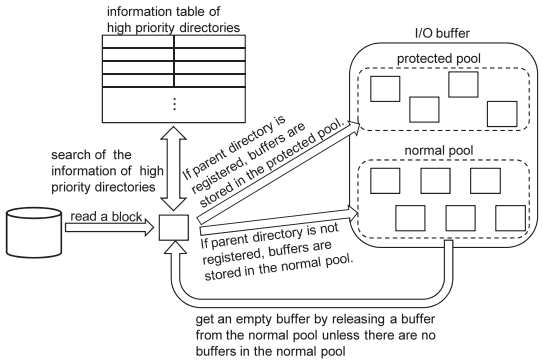


図 1　ディレクトリ優先方式の基本方式
Fig. 1　Directory oriented buffer cache mechanism.

を格納する．

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が構成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 課　　題

優先処理の動作内容に関する知識をもたない場合，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設定することは難しい．このため，誤って効果のない優先ディレクトリを指定すると，設定前より，優先処理の実行時間が増加する可能性がある．

3. 優先ディレクトリの設定法

3.1 目　　的

利用者が，優先処理の動作内容に関する知識をもたなくても，優先処理の実行時間を短縮できる優先ディレクトリを設定する方法を実現する．これにより，利用者が，優先処理の動作を解析し，試行錯誤しながら効果的な優先ディレクトリを探すことなく，LRU 方式よりも性能が向上する優先ディレクトリを設定することを可能にする．

3.2 基 本 手 順

優先ディレクトリの設定指針として，ディレクトリ重要度を導入する．ディレクトリ重要度は，次の性質をもつ値である．

（性質）　優先処理による 1 ブロック当りの平均ブロックアクセス回数が多いファイルを多く有するディレクトリほど，ディレクトリ重要度が高い．

本設定法は，優先処理がアクセスしたファイルの情報（以降，アクセスファイル情報と略す）を収集し，収集した情報から各ディレクトリのディレクトリ重要度を算出し，優先ディレクトリを決定する．また，優先処理の実行時には，あらかじめ優先ディレクトリを設定する．これにより，優先処理による 1 ブロック当りの平均ブロックアクセス回数が多いファイルを優先ファイルとすることができ，優先処理の実行時間を短縮できる．

本設定法の手順を以下に示す．

（手順 1）　優先処理のみを実行し，アクセスファイル情報を収集し，ファイルに保存する．アクセスファイル情報とは，i ノード番号，ブロック数（サイズに相当），及び総ブロックアクセス回数である．総ブロックアクセス回数は，当該ファイルのブロックにアクセスされた総回数である．

（手順 2）　アクセスファイル情報として収集されたファイル群の親ディレクトリを特定する．

（手順 3）　上記のアクセスファイル情報，親ディレクトリ，及び親ディレクトリが直下に有する全ファイルの情報から，ディレクトリ重要度を算出し，ファイルに保存する（算出方法については，後述する）．ディレクトリ重要度は処理ごとに異なる．このため，ディレクトリ重要度を保存したファイルを処理ごとに作成する．

（手順 4）　ディレクトリ重要度を用い，優先ディレクトリに設定するディレクトリを以下の方法で選択する．ただし，ディレクトリ直下の全ファイルの総サイズが，入出力バッファサイズを超えるディレクトリは，以下の方法の対象からあらかじめ除外しておく．これは，優先ファイルの総サイズが入出力バッファサイズを超えると，優先処理の性能が向上しないためである [1]．

$$\sum_{i=1}(\text{ディレクトリ } i \text{ 直下の全ファイルの総サイズ}) < \text{入出力バッファサイズの } N\% \quad (1)$$

ディレクトリ重要度が高いディレクトリから順に，式 (1) が成立する間，優先ディレクトリに選択する．ディレクトリ i は，ディレクトリ重要度が最も高いディレクトリからディレクトリ 1，ディレクトリ 2，ディレクトリ 3，… とする．なお，**4.** の評価では，$N = 100\%$ とした．

（手順 5）　優先処理実行前に，システムコールにより，（手順 4）で選択したディレクトリを優先ディレクトリ

に指定する．

上記の手順で一度効果的な優先ディレクトリが分かれば，以降は優先処理の実行前後で優先ディレクトリの設定と設定解除を行えばよい．なお，選択された優先ディレクトリのパス名は，テキスト形式でファイルに保存できるため，優先処理を起動するスクリプトを作成し，優先処理起動前と後の処理として，優先ディレクトリの設定と設定解除の処理を記述することで，実行時の設定を自動化できる．

本手法のオーバヘッドは，アクセスファイル情報を収集する処理，重要度を計算する処理，及び優先ディレクトリを選択する処理である．アクセスファイル情報を収集する処理は，**4. 2. 1** のカーネル make 処理で評価したところ，カーネル make の実行時間は，情報の収集なしで 257.11 秒，情報の収集ありで 258.21 秒であった．この結果から，オーバヘッドは 1.10 秒 (0.43%) と小さい．他の二つについては，表計算ソフト等を使えば，処理を自動化でき，簡略化できる．また，重要度の算出と優先ディレクトリの選択は，最初の 1 回行うだけでよい．

3.3 ディレクトリ重要度の算出法

3.3.1 観　　点

ディレクトリ重要度の算出法には，以下の二つの観点がある．

(観点 1)　1 ブロック当りの平均ブロックアクセス回数を求める単位

1 ブロック当りの平均ブロックアクセス回数を求める単位として，ディレクトリとファイルがある．前者は，各ディレクトリについて，直下のファイル群を一つのファイルとみなし，1 ブロック当りの平均ブロックアクセス回数を求め，これをディレクトリ重要度とする．一方，後者は，各ディレクトリについて，直下に有する各ファイルの 1 ブロック当りの平均ブロックアクセス回数を求め，求めた平均ブロックアクセス回数を基に，後述する方法でディレクトリ重要度を算出する．

(観点 2)　ディレクトリ重要度の算出対象ファイル

ディレクトリ重要度の算出対象ファイルとして，ディレクトリ下のアクセスされたファイルのみとする方法と，ディレクトリ下の全ファイルから算出する方法がある．後者は，全ファイルから算出するため，優先処理がアクセスしないファイルが多いディレクトリについて，1 ブロック当りの平均ブロックアクセス回数と，優先処理が頻繁にアクセスするファイルの割合が低くなる．

表 1　各算出法の観点
Table 1　Standpoints of each calculation method.

算出法の名称	(観点 1)	(観点 2)
(dir-part) 算出法	ディレクトリ	アクセスされたファイル
(dir-all) 算出法	ディレクトリ	全ファイル
(file-part) 算出法	ファイル	アクセスされたファイル
(file-all) 算出法	ファイル	全ファイル

ディレクトリ重要度の算出法は，(観点 1) と (観点 2) の組合せから，四つある．各算出法の名称と観点の関係を表 1 に示す．以降では，各算出法を (dir-part) 算出法，(dir-all) 算出法，(file-part) 算出法，及び (file-all) 算出法と略す．

3.3.2 (dir-part) 算出法

(dir-part) 算出法は，式 (2) により，各ディレクトリごとに，アクセスされたファイルから 1 ブロック当りの平均ブロックアクセス回数を求める．

$$\text{ディレクトリ重要度} = \frac{\text{全ファイルの総ブロックアクセス回数}}{\text{アクセスされたファイルの総ブロック数}} \quad (2)$$

3.3.3 (dir-all) 算出法

(dir-all) 算出法は，式 (3) により，各ディレクトリごとに，各ディレクトリ直下の全ファイルから 1 ブロック当りの平均ブロックアクセス回数を求める．

$$\text{ディレクトリ重要度} = \frac{\text{全ファイルの総ブロックアクセス回数}}{\text{全ファイルの総ブロック数}} \quad (3)$$

3.3.4 (file-part) 算出法

(観点 1) の単位をファイルとするため，(file-part) 算出法は，式 (4) より，ファイルごとに 1 ブロック当りの平均ブロックアクセス回数を求める．これをファイル重要度と呼ぶ．

$$\text{ファイル重要度} = \frac{\text{ファイルのブロックアクセス回数}}{\text{ファイルのブロック数}} \quad (4)$$

ファイル重要度が高いファイルは，優先処理が頻繁にアクセスするファイルであると判断できる．また，ファイルのブロックアクセス回数が同じであれば，ファイルサイズが小さいファイルほど，ファイル重要度が高くなる．

次に，式 (5) より，高重要度ファイル含有率を求め

表 2　各算出法の比較
Table 2　Advantages and disadvantages of each calculation method.

算出法	長所	短所
(dir-part)	算出に要する時間が，ファイルを単位とする方法に比べ，短い．	ファイルごとのブロックアクセス回数を考慮しないため，ブロックアクセス回数の少ないファイルを多く有するディレクトリを優先ディレクトリにする可能性がある．
(dir-all)	（1） 算出に要する時間が，ファイルを単位とする方法に比べ，短い． （2） 優先処理がアクセスしないファイルを直下に多く有するディレクトリのディレクトリ重要度を小さくできる．	ファイルごとのブロックアクセス回数を考慮しないため，ブロックアクセス回数の少ないファイルを多く有するディレクトリを優先ディレクトリにする可能性がある．
(file-part)	ファイル重要度を算出するため，ブロックアクセス回数の多いファイルの割合が高いディレクトリを優先ディレクトリにできる．	算出に要する時間が，ディレクトリを単位とする方法に比べ，長い．
(file-all)	（1） ファイル重要度を算出するため，ブロックアクセス回数の多いファイルの割合が高いディレクトリを優先ディレクトリにできる． （2） 優先処理がアクセスしないファイルを直下に多く有するディレクトリのディレクトリ重要度を小さくできる．	算出に要する時間が，ディレクトリを単位とする方法に比べ，長い．

る．高重要度ファイル含有率とは，各ディレクトリが直下に有するアクセスされたファイルのブロックの内，ファイル重要度が t 以上，つまり 1 ブロック当り t 回以上アクセスされるファイルの割合である．

$$\text{高重要度ファイル含有率} = \frac{\text{高重要度ファイルの総ブロック数}}{\text{優先処理がアクセスしたファイルの総ブロック数}} \quad (5)$$

最後に，式 (6) より，ファイル重要度と高重要度ファイル含有率から，ディレクトリ重要度を算出する．

$$\text{ディレクトリ重要度} = \text{ファイル重要度の和} \times \text{高重要度ファイル含有率} \quad (6)$$

3.3.5　(file-all) 算出法

(file-all) 算出法は，式 (4) より，各ディレクトリが直下に有する全ファイルのファイル重要度を求める．次に，式 (7) より，高重要度ファイル含有率を求める．

$$\text{高重要度ファイル含有率} = \frac{\text{高重要度ファイルの総ブロック数}}{\text{全ファイルの総ブロック数}} \quad (7)$$

最後に，式 (6) より，ファイル重要度と高重要度ファイル含有率から，ディレクトリ重要度を算出する．

3.3.6　各算出法の比較

各算出法の比較を表 2 に示す．

(dir-part) 算出法と (dir-all) 算出法は，算出法が単純で計算量が少ないため，ディレクトリ重要度の算出に要する時間が (file-part) 算出法と (file-all) 算出法に比べて短い．

(dir-part) 算出法と (dir-all) 算出法は，ファイル重要度を算出しないため，ファイルごとのブロックアクセス回数を考慮できず，ブロックアクセス回数の少ないファイルを多く有するディレクトリを優先ディレクトリにする可能性がある．一方，(file-part) 算出法と (file-all) 算出法は，これを考慮でき，ブロックアクセス回数の多いファイルの割合が高いディレクトリを優先ディレクトリにできる．

また，(dir-all) 算出法と (file-all) 算出法は，優先処理がアクセスしないファイルを直下に多く有するディレクトリのディレクトリ重要度を小さくできる．この特徴は，優先処理がアクセスしない優先ファイルに非優先処理がアクセスする場合に，このようなディレクトリの重要度を下げて優先ディレクトリとすることを防ぎ，優先処理のキャッシュヒット率の低下を抑制したいときに有効であると推察できる．

いずれの方式も有用であるが，以下の観点から実行する処理に応じて方式を選択することが有効であると推察する．

（1） (file-part) 算出法と (file-all) 算出法は，ファイル単位で平均ブロックアクセス回数を求めるため，ディレクトリ内でブロックアクセス頻度のばらつきが大きいサービスにおいて，高頻度アクセスファイルを有するディレクトリを優先ディレクトリに選択することが期待できる．一方，(dir-part) 算出法と (dir-all) 算出法は，ディレクトリ内のブロックアクセス頻度の

ばらつきが少なく平均的に各ファイルがアクセスされるディレクトリを優先ディレクトリに選択することが期待できる．また，より短時間でディレクトリの重要度を計算できる利点がある．

（2） (file-all) 算出法と (dir-all) 算出法は，優先処理がアクセスしないファイルの合計サイズが大きいディレクトリが優先ディレクトリに選ばれにくく，ディレクトリ直下のファイルのうち，アクセスされるファイルの割合が多いものが優先ディレクトリに選択されやすい．このため，同時に走行する非優先処理が優先ディレクトリにアクセスし，優先処理が利用しないファイルのブロックが保護プールに格納されることによるキャッシュヒット率低下を抑えることが期待できる．一方，(file-part) 算出法と (dir-part) 算出法は，優先処理だけの平均ブロックアクセス回数を利用するため，高頻度にアクセスされるブロックを多く含むディレクトリを優先ディレクトリに選択でき，優先処理だけを走行させる場合に向いている．

3.4 適 用 事 例

本方式は，以下の二つの性質をもつ処理に有効である．

（性質 1） 同じ処理を繰り返し実行

1 回目の実行で情報を収集してディレクトリ重要度を求め，優先ディレクトリを選択する．2 回目以降では，選択した優先ディレクトリを設定し実行する．これにより，2 回目以降の処理の実行時間を短縮できる．

（性質 2） アクセスするファイルが複数のディレクトリに分散

このような処理では，頻繁にアクセスするファイルを多く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．これにより，頻繁にアクセスするファイルを他のファイルと比べて優先的にキャッシュでき，実行時間を短縮できる．また，同一サービス内で複数のディレクトリにアクセスする場合に，キャッシュが効果的なディレクトリを優先ディレクトリとして設定でき，サービス全体の性能を向上できる．

アクセスが少数のディレクトリ集中していれば，人が経験により設定することは可能である．しかし，多数のディレクトリに複雑にアクセスが分散していると，人が適切な優先ディレクトリを設定するのは困難であり，提案方式が有効である．

二つの性質を併せもつ処理の例として，make，Web サーバ，ファイルサーバがある．ソフトウェア開発環境では，開発中に何度もコンパイルされる．また，Web サーバやファイルサーバは，長期間サービスを提供する．このため，一定期間情報を収集することで，それ以降の応答時間を短縮することができる．

4. 評　　価

4.1 評 価 内 容

優先処理の動作内容に関する知識に基づき，優先ディレクトリを設定した場合（以降，手動設定と略す）と比べ，本設定法がどの程度近い性能であるか評価する．本評価で利用する手動設定の設定ディレクトリは，何度も試行錯誤を繰り返して求めたものであるため，本設定法より，手動設定に近い性能を実現できれば十分である．

ディレクトリ直下のファイルへのブロックアクセス回数にばらつきがある処理としてカーネル make と Web サーバを用い，評価した．また，カーネル make を用いた評価では，他の処理による影響を明らかにするため，共存処理がない場合とカーネルのソースファイルのバックアップ処理を共存実行する場合について評価した．

4.2 カーネル make を用いた評価

4.2.1 評価環境と評価方法

計算機（OS : FreeBSD 4.3-RELEASE（以降，FreeBSD 4.3-R と略す），CPU : Celeron D 2.8 GHz，(メモリ，入出力バッファ) = (32 MByte，3.0 MByte)，(32 MByte，6.3 MByte)，または (64 MByte，9.0 MByte)，1 ブロックのサイズ：8.0 KByte，VMIO: オフ）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，計算機が認識できるメモリ量を 32 MByte と 64 MByte に制限し，入出力バッファサイズを 3.0 MByte，6.3 MByte，及び 9.0 MByte に制限した．これは次の理由による．入出力バッファ制御法方式の効果は，制御対象の計算機における入出力バッファサイズとアクセスするファイル群の大きさの相対的な関係によって，効果の有無が推測できる．つまり，入出力バッファサイズがアクセスするファイル群のサイズより大きければ，キャッシュヒット率はどの方式を利用しても向上する．反対に，入出力バッファサイズに対して，アクセス対象のファイル群のサイズが大きいほど，キャッシュヒット率が低下するため，入出力バッファ制御法の性能差が分かり

表 3　各方式の優先ディレクトリの情報
Table 3　Information of high priority directories.

		手動設定	dir-part 算出法	dir-all 算出法	file-part 算出法					file-all 算出法				
しきい値 t の値		-	-	-	1	2	3	5	10	1	2	3	5	10
/usr/src/sys/sys/の順位		なし	6 位	2 位	1 位	1 位	1 位	1 位	1 位	1 位	1 位	1 位	1 位	1 位
/usr/src/sys/i386/include/の順位		なし	7 位	3 位	2 位	2 位	2 位	3 位	3 位	2 位	2 位	2 位	2 位	2 位
優先ディレクトリ数（個）	3.0MByte	2	5	2	1	1	1	1	2	1	1	1	1	1
	6.3 MByte	2	11	9	6	7	8	8	9	7	7	8	8	8
	9.0 MByte	2	23	27	10	15	17	15	16	11	13	13	13	13
優先ファイルの合計サイズ (KByte)	3.0 MByte	2,704	1,448	1,800	1,696	1,696	1,696	1,696	1,800	1,696	1,696	1,696	1,696	1,696
	6.3 MByte	2,704	5,752	5,080	5,264	5,536	5,648	4,448	5,088	5,304	5,440	5,392	4,640	5,616
	9.0 MByte	2,704	8,432	8,440	7,776	8,456	5,664	7,544	7,992	8,040	8,256	7,240	8,392	8,408
カーネル make がアクセスした優先ファイルの総サイズ (KByte)	3.0 MByte	2,032	120	1,464	1,456	1,456	1,456	1,456	1,464	1,456	1,456	1,456	1,456	1,456
	6.3 MByte	2,032	3,088	3,048	3,208	3,464	3,480	2,408	3,048	4,088	4,144	4,176	3,736	4,408
	9.0 MByte	2,032	5,448	6,088	5,544	5,960	5,664	5,192	5,240	6,584	6,896	6,016	7,040	7,032

やすくなる．制御対象の計算機の搭載メモリやアクセスするファイルは様々であるので，それぞれの相対的な関係を把握しやすくするため，搭載メモリを制限し，相対的な関係を把握しやすくし，基本的な測定を行った．ただし，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，実際に使用できる領域はそれぞれ約 2.3 MByte，約 5.7 MByte，及び約 8.3 MByte である．VMIO 機能とは，FreeBSD がもつ次のような機能である．入出力バッファでキャッシュミスした場合，アクセスするブロックを保持するページがページキャッシュ内に存在するか探索する．探索の結果，当該ページが存在すれば，このページがもつデータを利用し，I/O を削減する．

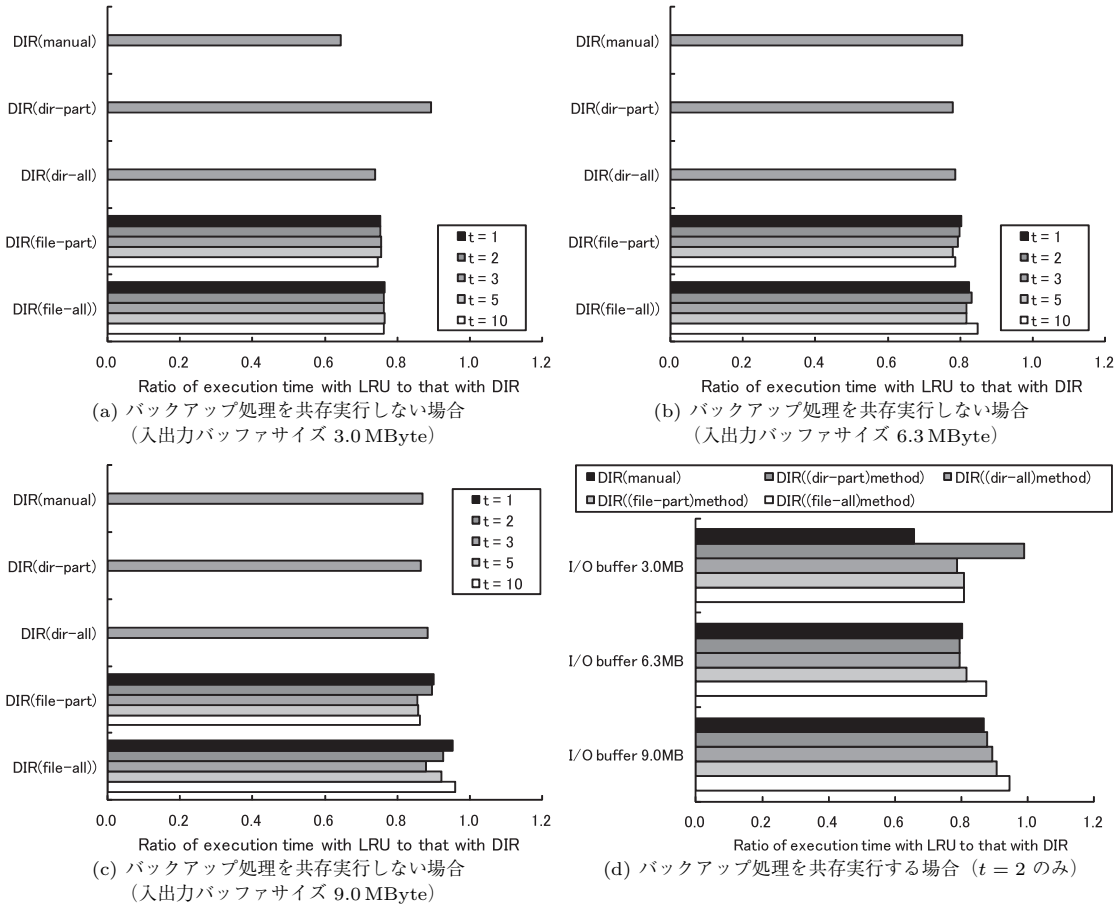
FreeBSD 4.3-R に実装されている LRU 方式，手動設定でのディレクトリ優先方式，及び本設定法を用いたディレクトリ優先方式を評価した．また，共存処理がある場合の評価として，カーネルのソースファイルのバックアップ処理を共存実行した．バックアップ処理を共存実行する場合，カーネル make とバックアップ処理はほぼ同時に開始した．測定開始直前に優先ディレクトリを設定した．なお，(file-part) 算出法と (file-all) 算出法は，バックアップ処理を共存実行しない場合，$t = 1$，2，3，5，10 の五つの場合について測定し，t による影響を評価した．この測定結果により，t の違いによる影響は小さい．これは，1 回目のアクセスでキャッシュされたブロックがキャッシュヒットするには，2 回目以降のアクセスが必須であり，平均して 2 回のアクセスがあれば，ファイルのブロックが再アクセスでキャッシュヒットすると期待できるからである．そこで，バックアップ処理を共存実行する場合は，$t = 2$ とし，算出法による違いを比較した．

設定した優先ディレクトリの情報を表 3 に示す．表 3 に示す優先ファイルの合計サイズは，各優先ファイルのサイズを 8.0 KByte の定数倍にまるめて合計した値である．これは，評価環境のファイルシステムの論理ブロックサイズが 8.0 KByte であり，ファイルを入出力バッファにキャッシュする際，ファイルのサイズを 8.0 KByte の定数倍に丸めたサイズの領域を消費するからである．この入出力バッファの消費量をディレクトリ重要度の計算に反映させるため，ファイルのサイズを 8.0 KByte の定数倍にまるめて合計した．また，手動設定では，/usr/src/sys/sys/ と/usr/src/sys/i386/include/を優先ディレクトリに設定した．カーネル make はヘッダファイルに繰り返しアクセスし，この二つのディレクトリ直下のヘッダファイルは頻繁にアクセスされる．

また，方式の違いによるカーネル make の実行時間の違いを明確にするため，ファイルの連続読込み処理を共存実行し，ディスク I/O 負荷をかけた．この処理は，入出力バッファサイズ以上のファイルを読み込むことで負荷をかけるため，カーネル make やバックアップ処理と関係ない 8 個若しくは 9 個の 1.0 MByte のファイルを順番に繰り返し読み込む．また，この処理は，1.0 KByte 単位で read システムコールを繰り返し発行する．なお，カーネル make の実行時間より，共存するファイル連続読込み処理の実行時間の方が長い．

4.2.2　評価結果

LRU 方式での実行時間を 1 としたときのディレクトリ優先方式での実行時間の比率を図 2 に示す．バックアップ処理を共存実行しない場合（図 2 (a)〜(c)），LRU 方式を用いた場合のカーネル make の実



(a) バックアップ処理を共存実行しない場合（入出力バッファサイズ 3.0 MByte）

(b) バックアップ処理を共存実行しない場合（入出力バッファサイズ 6.3 MByte）

(c) バックアップ処理を共存実行しない場合（入出力バッファサイズ 9.0 MByte）

(d) バックアップ処理を共存実行する場合（$t = 2$ のみ）

図 2　カーネル make の実行時間の比率（LRU 方式を 1 とした場合）
Fig. 2　Ratio of execution time of kernel make with LRU to that with DIR.

行時間は，入出力バッファサイズ 3.0 MByte で 934 秒，6.3 MByte で 544 秒，9.0 MByte で 434 秒であった．バックアップ処理を共存実行した場合（図 2 (d)），LRU 方式を用いた場合のカーネル make の実行時間は，入出力バッファサイズ 3.0 MByte で 1,134 秒，6.3 MByte で 587 秒，9.0 MByte で 464 秒であった．なお，以降の図では，ディレクトリ優先方式を DIR，LRU 方式と LRU と表記する．

（1） 図 2 (a) より，バックアップ処理を共存実行しない場合，入出力バッファサイズ 3.0 MByte では，提案方式は算出法にかかわらず，手動設定と比べて実行時間が長い．これは，以下の二つの理由による．

（理由 1）提案方式は手動設定と比べて優先ファイルの総サイズが小さいことが理由の一つである．このため，高速化させるのに十分な優先ファイルをキャッシュできなかったと推察できる．

（理由 2）手動設定でアクセスされた優先ファイルの総サイズは 2,032 KByte であり，アクセスされた優先ファイルを入出力バッファにすべてキャッシュでき，かつ通常プールにもブロックをキャッシュすることができたため，実行時間が特に短かったと推察できる．

（2） 図 2 (b)～(c) より，バックアップ処理を共存実行しない場合，入出力バッファ6.3 MByte と 9.0 MByte では，提案方式ディレクトリ重要度の算出法にかかわらず，手動設定に近い性能を出せている．これは，本設定法により，効果的な優先ディレクトリに設定できたためである．

（3） 図 2 (d) より，バックアップ処理を共存実行した場合，入出力バッファサイズ 3.0 MByte での (dir-all) 算出法が最も実行時間を短縮できており，LRU 方

式に比べ，241 秒 (21.2%) 短い．これは，(dir-all) 算出法は，優先処理がアクセスしないファイルも考慮するため，バックアップ処理のファイルアクセスにより，カーネル make がアクセスしないファイルのブロックが保護プールにキャッシュされることを抑制できるためである．

（4） (file-all) 算出法は，バックアップ処理の有無にかかわらず，手動設定に比べ，実行時間が長い．これは，(file-all) 算出法では，表 3 のアクセスした優先ファイルの合計サイズが，他の算出法より，比較的多いためである．(file-all) 算出法では，高重要度ファイルの含有率が高いディレクトリの重要度が高くなる．このため，優先ファイルの合計サイズが他の方式と同じでも，実際に優先処理にアクセスされるファイル総サイズは大きくなる．この結果，保護プールに多くのブロックがキャッシュされ，通常プールが小さくなり，非優先ファイルのキャッシュヒット率が低下し，非優先ファイルにもアクセスするカーネル make の実行時間が増加したと考えられる．

（5） 図 2 (a)～(d) より，入出力バッファサイズが小さいほど，ディレクトリ優先方式の効果が大きいことが分かる．LRU 方式では，入出力バッファサイズが小さいほど，頻繁にアクセスされるファイルであっても，入出力バッファから破棄されやすくなる．ディレクトリ優先方式で頻繁にアクセスさえれるファイルを優先的にキャッシュすることで，実行時間を短縮できたと推察できる．

（6） 表 3 より，(file-part) 算出法と (file-all) 算出法は，カーネル make の実行時間を短縮する効果の大きいディレクトリである/usr/src/sys/sys/と/usr/src/sys/i386/include/のディレクトリ重要度が上位になるように算出できている．**3. 3. 6** で述べたように，(dir-part) 算出法と (dir-all) 算出法は，ブロックアクセス回数が少ないファイルを直下に多く有するディレクトリであっても，ディレクトリ重要度が高くなる可能性がある．このため，(dir-part) 算出法と (dir-all) 算出法は，上記二つのディレクトリの順位が低くなっている．(file-part) 算出法で $t = 5, 10$ とすると，/usr/src/sys/i386/include/が 3 位になっている．これは，優先処理が頻繁にアクセスするファイルを有しているものの，構成するブロックに平均 5 回，または 10 回以上ブロックアクセスするファイルの割合が低いためであると考えられる．

（7） 入出力バッファサイズが 6.3 MByte で $t =$ 2 の場合，/usr/src/sys/sys/と/usr/src/sys/i386/include/のディレクトリ重要度を上位にできている (file-part) 算出法と (file-all) 算出法と比べ，(dir-part) 算出法や (dir-all) 算出法の方が実行時間が短い．これは以下の二つの理由による．

（理由 1）優先ディレクトリの個数以内の順位に入っていれば，優先ディレクトリに選択され，優先的にキャッシュされるため，優先ディレクトリ間の順位差はキャッシュのされやすさに影響を与えない．

（理由 2）(dir-part) 算出法や (dir-all) 算出法は，残る二つの算出法と比べて優先処理がアクセスした優先ファイルの総サイズが小さく，非優先ファイルもキャッシュできた．

（8） 各算出法で，優先ディレクトリ数が異なる．これは，各算出法でディレクトリ重要度の順位が異なるためである．**3. 2** で述べたとおり，ディレクトリ重要度が上位のディレクトリから優先ディレクトリに選択する．この際，各算出法でディレクトリ重要度の順位が異なるため，優先ディレクトリに選択するディレクトリが異なり，式 (1) を満たすディレクトリ数に違いが出る．

以上の結果から，本設定法は，優先処理の動作内容に関する知識をもたなくても，手動設定に近い性能を発揮できるといえる．

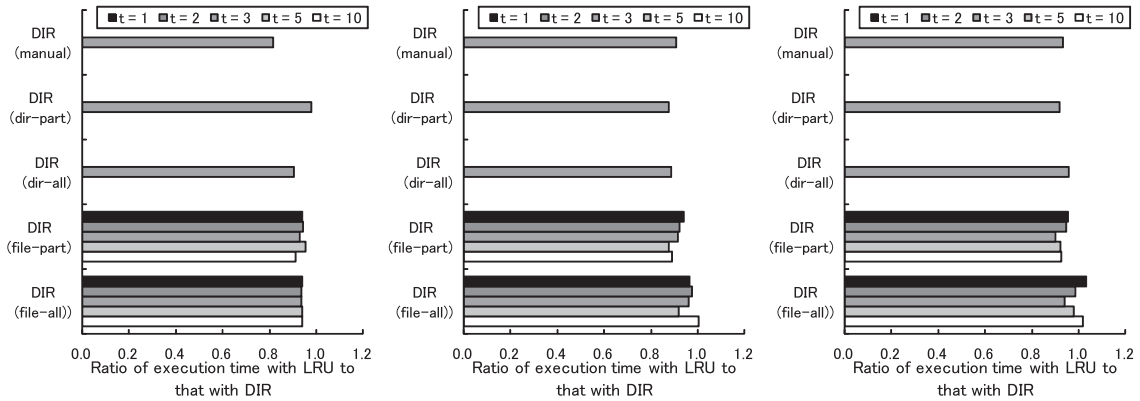
4. 2. 3 高性能な計算機での評価

プロセッサ性能による影響を明らかにするため，より高性能な CPU を搭載した計算機で，カーネル make（バックアップ処理の共存実行なし）の実行時間を測定した．評価環境として，Core i7 3.4 GHz を搭載した計算機を用いた．メモリと入出力バッファの大きさと，評価手順は，**4.2.1** と同じである．

LRU 方式での実行時間を 1 としたときのディレクトリ優先方式での実行時間の比率を図 3 に示す．LRU 方式を用いた場合のカーネル make の実行時間は，入出力バッファサイズ 3.0 MByte で 531 秒，6.3 MByte で 310 秒，9.0 MByte で 258 秒であった．図 2(a)～(c) と図 3 から，以下のことが分かる．

（1） 高性能な CPU を用いた場合（図 3），図 2 (a)～(c) と同様に，提案方式は，入出力バッファ 3.0 MByte では手動設定と比べて実行時間が長く，入出力バッファ6.3 MByte と 9.0 MByte では手動設定と近い性能を出せている．理由は，**4.2.2** と同じである．

（2） 高性能な CPU を用いた場合（図 3），図 2 (a)～(c) と比べ，ディレクトリ優先方式の効果が



(a) バックアップ処理を共存実行しない場合（入出力バッファサイズ 3.0 MByte）
(b) バックアップ処理を共存実行しない場合（入出力バッファサイズ 6.3 MByte）
(c) バックアップ処理を共存実行しない場合（入出力バッファサイズ 9.0 MByte）

図 3　カーネル make の実行時間の比率（LRU を 1 とした場合，高性能な CPU を利用）
Fig. 3　Ratio of execution time of kernel make with LRU to that with DIR (with high performance CPU).

小さい．これは，ディスクインタフェースの違いによる影響と推察できる．高性能な CPU を用いた場合，ディスクインタフェースが SATA 3 Gbit/s であり，図 2 (a)〜(c) は Ultra ATA 100 である．このため，高性能な CPU を用いた場合，ディスク I/O 時のデータ転送に要する時間が短く，キャッシュミスペナルティが小さい．

以上の結果から，CPU の性能にかかわらず，提案方式は有効であるといえる．

4.3　Web サーバを用いた評価

4.3.1　評価環境と評価方法

本設定法は，頻繁にアクセスされ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設定できる．このため，本設定法で優先ディレクトリを設定することにより，Web サーバの平均応答時間を短縮できると考えられる．

そこで，共存処理なしで Web サーバを運用した場合について，評価する．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 月における岡山大学の 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．評価で用いたファイルの情報を表 4 に示す．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．

Web サーバとして Apache 2.0.55 を用い，Web クライアントプログラムとして ApacheBench 2.40-dev 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高速に転送する 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を評価するため，ディレクトリ優先方式ではこの機能を無効にした．なお，sendfile 機能は，Apache 2.0.55 の初期設定では有効になっているため，LRU 方式では sendfile 機能を有効とした．

評価に用いた計算機を表 5 に示す．**4.2.1** で述べた理由により，本評価では，計算機が認識できるメモリ量と入出力バッファサイズを制限し，主記憶のメモリ量よりも，アクセスされる Web コンテンツの容量が十

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.

ディレクトリ数（個）	1,365
ファイル数（個）	8,849
合計要求回数（回）	100,000
合計ファイルサイズ (MByte)	600.0

表 5　Web サーバによる評価に用いた計算機
Table 5　Computers used in Web server evaluation.

	Web サーバ機	Web クライアント機
OS	FreeBSD 4.3-R	FreeBSD 4.3-R
CPU	Celeron D 2.8 GHz	Celeron D 2.8 GHz
メモリ	256 MByte	256 MByte
入出力バッファ	16 MByte 32 MByte 64 MByte	32 MByte
1 ブロックのサイズ	8.0 KByte	8.0 KByte
VMIO	オン	オン

表 6　Web サーバの平均応答時間の比（LRU 方式 = 1）
Table 6　Ratio of average response time of the Web server. (LRU= 1)

方式	入出力バッファ16 MByte		入出力バッファ32 MByte		入出力バッファ64 MByte	
	応答時間の比	応答時間の差 (%)	応答時間の比	応答時間の差 (%)	応答時間の比	応答時間の差 (%)
DIR (manual)	0.975	2.5	0.974	2.6	0.972	2.8
DIR (dir-part)	0.982	1.8	0.972	2.8	0.931	6.9
DIR (dir-all)	0.975	2.5	0.954	4.6	0.902	9.8
DIR (file-part)	0.992	0.8	0.961	3.9	0.883	11.7
DIR (file-all)	0.989	1.1	0.957	4.3	0.864	13.6

表 7　優先ファイルと非優先ファイルへのアクセスの平均応答時間 (ms)
Table 7　Average response time (ms) of high priority files and low priority files.

方式	入出力バッファ16 MByte		入出力バッファ32 MByte		入出力バッファ64 MByte	
	優先ファイル	非優先ファイル	優先ファイル	非優先ファイル	優先ファイル	非優先ファイル
LRU (manual)	1.70	14.07	1.69	13.97	1.69	13.63
LRU (dir-part)	1.63	19.59	1.82	21.66	2.44	21.16
LRU (dir-all)	1.62	20.01	2.08	21.50	2.85	23.68
LRU (file-part)	1.22	14.90	1.86	19.20	2.53	26.75
LRU (file-all)	1.19	14.41	1.90	20.28	2.88	27.78
DIR (manual)	1.20	14.36	1.15	14.29	1.17	13.91
DIR (dir-part)	1.34	19.88	1.39	22.19	1.71	21.63
DIR (dir-all)	1.34	20.11	1.52	21.95	1.74	25.01
DIR (file-part)	1.05	15.01	1.26	19.79	1.36	27.90
DIR (file-all)	1.12	14.32	1.27	20.91	1.38	30.20

表 8　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
Table 8　The number of accesses to high priority files or low priority files.

方式	入出力バッファ16 MByte		入出力バッファ32 MByte		入出力バッファ64 MByte	
	優先ファイル	非優先ファイル	優先ファイル	非優先ファイル	優先ファイル	非優先ファイル
DIR (manual)	58304	41653	58304	41660	58304	41669
DIR (dir-part)	70858	29109	74850	25126	77411	22559
DIR (dir-all)	71502	28453	75617	24350	81636	18334
DIR (file-part)	58752	41219	71463	28501	82914	17050
DIR (file-all)	57104	42873	73297	26676	84784	15190

分に大きい場合を想定して測定した．Web サーバ機と Web クライアント機は，100BASE-TX のスイッチングハブを介して接続されている．また，入出力バッファの設定値は，Web サーバ機で 16 MByte，32 MByte，64 MByte，Web クライアント機で 32 MByte であるものの，システム維持のために常時確保される領域が入出力バッファに存在するため，実際にデータのキャッシュのために使用できる入出力バッファの領域は 0.7 MByte ほど小さい．

なお，**4.2.1** と同様の理由により，本評価では $t = 2$ とした．手動設定は，文献 [1] と同じように，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設定した．これは，トップページは頻繁にアクセスされるためである．

4.3.2　評 価 結 果

LRU 方式を用いると，Web サーバの平均応答時間は入出力バッファサイズ 16 MByte で 6.86 ms，32 MByte で 6.81 ms，64 MByte で 6.67 ms であった．LRU 方式での平均応答時間を 1 としたときのディレクトリ優先方式での平均応答時間の比を表 6 に示す．表 7 に優先ファイルへのアクセスの平均応答時間と非優先ファイルへのアクセスの平均応答時間を示す．LRU 方式では優先ファイルがないため，手動設定と各算出法それぞれでの優先ファイルと非優先ファイルに基づき，集計した．表 7 の LRU の括弧内にどの方式での優先ファイルであるかを示している．表 8 に手動設定と各算出法での優先ファイルと非優先ファイルへの各アクセス回数を示す．表 6〜表 8 から，以下のことが分かる．

（1） 表 6 より，本設定法は，全入出力バッファサイズで，どの算出法を用いたとしても，提案方式は手動設定と応答時間が同等，若しくはより短縮できていることが分かる．これは，提案方式により，Web サーバの応答時間を短縮するのに効果的な優先ディレクト

リを設定できたためである．

（2） 表 6 より，特に，入出力バッファサイズ 64 MByte で (file-all) 算出法を用いた場合に，最も Web サーバの応答時間を短縮できており，手動設定と比べて 0.72 ms (11.1%) 短縮できている．これは，表 8 より，この場合，Web サーバへのリクエストの約 85%が優先ファイルへのアクセスとなっており，他の場合と比べ，大きく短縮できたためである．

（3） 入出力バッファサイズが 16 MByte と 32 MByte の場合，提案方式の効果は，LRU 方式と比べて 1〜4%程度の応答時間の短縮と小さい．これは，次の理由による．提案方式により，アクセス回数が多いファイルをもつディレクトリが優先ディレクトリに選ばれる．これにより，アクセス回数が多いファイルのキャッシュヒット率が向上し，これらのファイルの応答時間が短縮できる．一方で，優先ディレクトリ直下にないファイルの応答時間は増加する．例えば，表 7 より，入出力バッファサイズ 32 MByte で (file-all) 算出法を用いた場合，LRU 方式と比べて，優先ファイルへのアクセスの平均応答時間を 0.63 ms (33.06%) 短縮できているものの，非優先ファイルへのアクセスの平均応答時間を 0.63 ms (3.11%) 増加している．このため，応答時間を全ファイルで平均すると，頻繁にアクセスされるファイルの応答時間の向上分が打ち消され，1〜4%の向上となってる．

（4） 表 7 より，提案方式は LRU 方式と比べ，優先ファイルへのアクセスの平均応答時間を大幅に短縮できている．入出力バッファサイズ 64 MByte で (file-all) 算出法を用いた場合，優先ファイルへのアクセスの平均応答時間を最も短縮できており，1.50 ms (52.04%) 短い．これは，優先ファイルを優先的にキャッシュできたためである．

（5） 表 7 より，提案方式は LRU 方式と比べ，非優先ファイルの平均応答時間が増加する．しかし，増加幅は小さく，最も平均応答時間が長くなった入出力バッファサイズ 64 MByte での (file-all) 算出法でも 2.42 ms (8.71%) である．これは，非優先ファイルへのアクセスが頻繁におこるものではなく，LRU 方式でもキャッシュヒットしにくく，キャッシュする効果が小さいためであると推察できる．

なお，Web サーバの場合は，Web サーバのアクセスログを用いて，各ファイルへのアクセス情報を取得できるため，アクセスログを用いて，ディレクトリ重要度を算出することもできる．

5. 関連研究

入出力バッファの制御方式 [2]〜[12] が提案されている．ブロックアクセスの頻度や間隔に基づく制御方式として，2Q [2]，ARC [3]，CAR [4]，及び LIRS [5] がある．2Q，ARC，及び CAR は，一度しかアクセスされていないブロックと複数回アクセスされたブロックを別々の領域にキャッシュする．また，LIRS は，同じブロックに対する最後のアクセスとその一つ前のアクセスの間に起こった他のブロックへのブロックアクセス回数を指標とし，この指標が小さいブロックを優先的にキャッシュする．

利用者などが与えるヒントに基づく制御方式として，ACFC [6] と Karma [7] がある．ACFC は，利用者が提供した応用プログラムの動作内容に関するヒントに基づき，どのバッファを解放するか決定する．Karma 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．

アクセスパターンに基づく制御方式として，UBM [8] と PCC [9] がある．UBM と PCC は，シーケンシャル，ループ，及びその他の三つのアクセスパターンごとに入出力バッファの分割領域を割り当て，アクセスパターンごとに異なる制御方式で各領域を制御する．Karma，UBM，及び PCC は，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，分割領域のサイズを決定する．

文献 [11] で提案された制御方式は，実行する処理ごとに入出力バッファの分割領域を割り当てる．この制御方式は，システム全体のキャッシュヒット率を向上させるため，分割領域を大きくすることにより，キャッシュヒット率がより向上する処理の分割領域をより大きくする．また，ファイル操作に基づく制御方式として，FFU [10] がある．FFU は，open システムコールの発行頻度からファイルに重要度を設定し，この重要度が高いファイルのブロックを優先的にキャッシュする．更に，ディスク上で連続したブロック群は先読み可能であるため，キャッシュミスコストが小さいことに着目し，DULO [12] が提案された．DULO は，ディスク上でランダムな領域に配置されたブロックを優先的にキャッシュする．

関連研究の方式は，システム全体の処理性能向上を狙ったものである．一方，文献 [1] のディレクトリ優先方式は，特定の処理（優先処理）の性能を向上させる

ことを目的とした方式である．文献 [1] では，優先ディレクトリを選択する明確な方法が示されていなかったため，本論文で提案した．また，関連研究の方式は，実行する処理の組合せの変化など，アクセスパターンの変化に追従するため，常に情報を収集し，フィードバックをかけており，情報収集のためのオーバヘッドは小さくない．一方，提案方式は，1 回目と 2 回目以降でアクセスパターンが同じであることを前提としているものの，1 回情報を収集しておけば，2 回目以降は情報収集のオーバヘッドなしに実行できる．

6. むすび

ディレクトリ優先方式において，優先処理の動作内容に関する知識を必要とすることなく，効果的な優先ディレクトリを設定する手法を提案した．本設定法は，優先ディレクトリの明確な設定指針として，ディレクトリ重要度の概念を導入し，この重要度が高いディレクトリから順に，優先ディレクトリに設定する．この重要度は，優先処理がアクセスするファイルのブロックアクセス回数やブロック数から算出される．この算出法を四つ示し，各算出法を比較した．

カーネル make を用いた評価において，本設定法は，カーネル make の実行時間を LRU 方式に比べ，119.74 秒 (22.0%) 短縮できた．また，本設定法は，カーネル make の動作内容に関する知識に基づき，カーネル make の実行時間を短縮する効果の大きいディレクトリを優先ディレクトリに設定した場合 [1] と同等の結果が得られた．Web サーバを用いた評価において，本設定法は，Web サーバの平均応答時間を LRU 方式に比べ，0.31 ミリ秒 (4.6%) 短縮できた．また，Web サーバの動作内容に関する知識に基づき，頻繁にアクセスされると考えられ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設定した場合 [1] よりも Web サーバの平均応答時間を短縮できた．

残された課題として，ディレクトリ重要度を用いた優先ディレクトリの選択における入出力バッファサイズに対する優先ファイルの総サイズの比率 (N) の検討がある．

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,” 信学論（D），vol.J91-D，no.2, pp.435–448, Feb. 2008.

[2] T. Johnson and D. Shasha, “2Q: A low overhead high performance buffer management replacement algorithm,” Proc. 20th International Conference on Very Large Databases, pp.439–450, 1994.

[3] N. Megiddo and D.S. Modha, “ARC: A self-tuning, lowoverhead replacement cache,” Proc. 2nd USENIX Conference on File and Storage Technologies (FAST ’03), pp.115–130, 2003.

[4] S. Bansal and D.S. Modha, “CAR: Clock with adaptive replacement,” Proc. 3rd USENIX Conference on File and Storage Technologies (FAST ’04), pp.187–200, 2004.

[5] S. Jiang and X. Zhang, “LIRS: An efficient low interreference 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.

[6] P. Cao, E.W. Felten, and K. Li, “Application-controlled file caching policies,” Proc. USENIX Summer 1994 Technical Conference, pp.171–182, 1994.

[7] G. Yadgar, M. Factor, and A. Schuster, “Karma: Know-it-all replacement for a multilevel cache,” Proc. 5th USENIX Conference on File and Storage Technologies (FAST ’07), pp.169–184, 2007.

[8] J.M. Kim, J. Choi, J. Kim, S.H. Noh, S.L. Min, Y. Cho, and C.S. Kim, “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.

[9] C. Gniady, A.R. Butt, and Y.C. Hu, “Program-counter-based pattern classification in buffer caching,” Proc. 6th Symposium on Opearting Systems Design and Implementation (OSDI 2004), pp.395–408, 2004.

[10] 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,” 情処学論 (ACS)，vol.3, no.1, pp.50–60, March 2010.

[11] X. Meng, C. Si, W. Na, H.-U.-R. Khan, and L. Xu, “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, IEEE Computer Society, Washington, DC, USA, 2009. http://dl.acm.org/citation.cfm?id=1581383.1582152

[12] X. Ding, S. Jiang, and F. Chen, “A buffer cache management scheme exploiting both temporal and spatial localities,” ACM Trans. Storage, vol.3, Issue 2, Article no.5, 2007.

（平成 24 年 5 月 15 日受付，10 月 3 日再受付）

土谷　彰義

平 22 岡山大・工・情報卒．平 24 同大大学院自然科学研究科博士前期課程了．オペレーティングシステムに興味をもつ．情報処理学会会員．

松原　崇裕

平 24 岡山大・工・情報卒．オペレーティングシステムに興味をもつ．

山内　利宏　（正員）

平 10 九大・工・情報卒．平 12 同大大学院システム情報科学研究科修士課程了．平 14 同大大学院システム情報科学府博士後期課程了．平 13 日本学術振興会特別研究員 (DC2)．平 14 九州大学大学院システム情報科学研究院助手．平 17 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もつ．情報処理学会，ACM，USENIX 各会員．

谷口　秀夫　（正員）

昭 53 九大・工・電子卒．昭 55 同大大学院修士課程了．同年日本電信電話公社電気通信研究所入所．昭 62 同所主任研究員．昭 63NTT データ通信（株）開発本部移籍．平 4 同本部主幹技師．平 5 九州大学工学部助教授．平 15 岡山大学工学部教授．博士（工学）．オペレーティングシステム，実時間処理，分散処理に興味をもつ．著書「並列分散処理」（コロナ社）等．情報処理学会，ACM 各会員．