Vol.2012-OS-120 No.9
2012/2/29

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の設定法

松原崇裕[†1]　土谷彰義[†2]
山内利宏[†2]　谷口秀夫[†2]

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持たない利用者が，優先処理の実行時間を短縮するのに効果的な優先ディレクトリを設定することは難しい．そこで，本稿では，優先ディレクトリを自動的に設定する手法について述べ，評価により有効性を示す．

Method to Set Effective Directories for a Directory Oriented Buffer Cache Mechanism

Takahiro Matsubara, Akiyoshi Tsuchiya,
Toshihiro Yamauchi and Hideo Taniguchi

For improving performance of high priority processing, we proposed a directory oriented buffer cache mechanism. This mechanism gives high priority to directories, which are associated with high priority processing. Files in high priority directories are cached with the high priority. However, it is difficult to set effective directories without knowledge about behavior of high priority processing. This paper describes method which automatically set effective directories. Additionally, this paper describes the effectivity of this method by evaluation.

†1 岡山大学工学部
Faculty of Engineering, Okayama University
†2 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．優先処理の実行時間を短縮す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，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とにより，優先処理の実行時間を短縮する．しかし，優先処理の動作内容に関する知識を持たない利用者が，優先処理の実行時間を短縮するのに効果的な優先ディレクトリを設定することは難しい．また，現在，優先ディレクトリの設定指針がないため，利用者が優先処理の動作内容を理解したとしても，優先ディレクトリを設定することは難しい．

そこで，本稿では，優先ディレクトリを自動的に設定する手法を提案する．本設定法は，優先ディレクトリの明確な設定指針を示し，優先処理がアクセスするファイルを構成するブロックのアクセス回数やブロック数から，ディレクトリに重要度を設定する．これにより，優先処理の動作内容に関する知識や経験を必要とすることなく，利用者は，優先処理の実行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る．

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を構成するブロックをキャッシュし，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）を構成するブロックをキャッシュする．

ブロックアクセス時，通常プール内にブロックが存在する限り通常プールからブロックを解放し，空き領域を確保する．空き領域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キャッシュする．

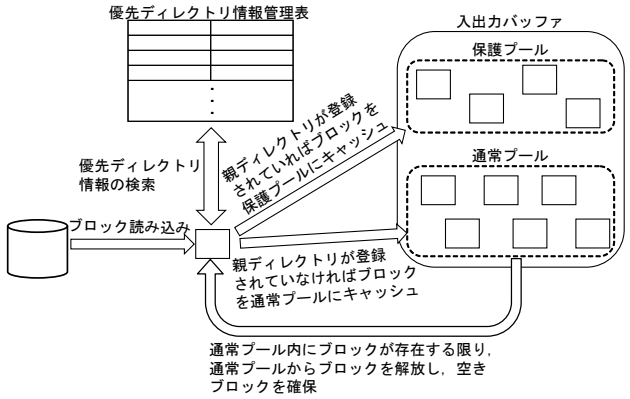

図 1 ディレクトリ優先方式の基本方式

2.2 優先ディレクトリ設定の課題

ディレクトリ優先方式は，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持たない利用者が，優先処理の実行時間を短縮するのに効果的な優先ディレクトリを設定することは難しい．また，現在，優先ディレクトリの設定指針がないため，利用者が優先処理の動作内容を理解したとしても，優先ディレクトリを設定することは難しい．

3. 効果的な優先ディレクトリの設定法

3.1 方　　針

優先処理の実行時間を短縮するのに効果的な優先ディレクトリを判別する指針として，ディレクトリに重要度（以降，ディレクトリ重要度と略す）を導入する．本設定法では，ディレクトリ重要度に基づき，優先ディレクトリを選択する．

本設定法を用いたディレクトリ優先方式の処理の流れを以下に示す．

（1） 利用者は，優先処理を共存処理なしで実行する．このとき，カーネルは，優先処理がアクセスしたファイルの情報を管理表（以降，アクセスファイル情報管理表と略す）に登録する．アクセスファイル情報管理表は，ファイル毎に，i ノード番号，ブロック数，および総ブロックアクセス回数を情報として持つ．カーネルは，ブロックアクセスが起こる度に，アクセスされたブロックに対応するファイルの i ノード番号を取得する．ファイルの i ノード番号が，アクセスファイル情報管理表に登録されていなければ，ファイルの i ノード番号，ブロック数，および総ブロックアクセス回数を新たに登録し，登録されていれば，当該ファイルの総ブロックアクセス回数を 1 増やす．

（2） 優先処理完了後，アクセスファイル情報管理表に登録された情報から，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．（優先ディレクトリの選択方法は，3.4 節で述べる）

（3） 利用者は，選択したディレクトリを優先ディレクトリに設定して，優先処理を実行する．

3.2 ディレクトリ重要度

ディレクトリ優先方式では，1 ブロック当たりのブロックアクセス回数が多いファイルを構成するブロックを優先的にキャッシュすると，キャッシュヒット率を向上させ，優先処理の実行時間を短縮できる．このため，ディレクトリ重要度は，次の特性を持つ値にする．

（特性） 1 ブロック当たりのブロックアクセス回数が多いファイルを有するディレクトリほど，ディレクトリ重要度が高い．

また，ディレクトリ重要度の算出法を定義するに当たり，以下の 2 つの観点がある．

（観点 1） 1 ブロック当たりのブロックアクセス回数を求める単位

（観点 2） ディレクトリ重要度の算出に利用する対象ファイル

（観点 1）から，1 ブロック当たりのブロックアクセス回数を求める単位をディレクトリとする方法とファイルとする方法がある．前者は，ディレクトリ毎に求めた 1 ブロック当たりのブロックアクセス回数をディレクトリ重要度とする．後者は，ファイル毎に 1 ブロック当たりのブロックアクセス回数を求め，この情報から親ディレクトリのディレクトリ重要度を求める．これにより，後者は，ディレクトリ直下の頻繁にアクセスされるファイルとそうでないファイルの割合がわかる．この割合は，優先処理の実行時間を短縮するのに効果的な優先ディレクトリの判別に利用できる．

（観点 2）から，ディレクトリ重要度を優先処理がアクセスしたファイルのみから求める方法と，全ファイルから求める方法がある．全ファイルの場合，ディレクトリ直下のファイルについて，優先処理がアクセスしないファイルの割合も考慮してディレクトリ重要度を求める．これは，同時に走行する非優先処理が，優先処理がアクセスしない優先ファイルにアクセスすると，優先処理がアクセスしない優先ファイルを構成するブロックが保護プールにキャッシュされ，優先処理のキャッシュヒット率が低下するためである．

表 1 各算出法の名称と観点の関係

算出法の名称	（観点 1）の単位	（観点 2）の対象ファイル
（ディ・個）算出法	ディレクトリ	優先処理がアクセスしたファイル
（ディ・全）算出法	ディレクトリ	全ファイル
（ファ・個）算出法	ファイル	優先処理がアクセスしたファイル
（ファ・全）算出法	ファイル	全ファイル

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を求める方法として，まず,（観点 1）から，1 ブロック当たりのブロックアクセス回数を求める単位をディレクトリとする方法とファイルとする方法の 2 通りがある．さらに,（観点 2）から，ディレクトリ重要度を優先処理がアクセスしたファイルのみから求める方法と全ファイルから求める方法の 2 通りがある．これらの方法の組み合わせから，ディレクトリ重要度の算出法は 4 つ存在する．以降では，1 ブロック当たりのブロックアクセス回数を求める単位をディレクトリとする方法とディレクトリ重要度を優先処理がアクセスしたファイルのみから求める方法の組み合わせを（ディ・個）算出法，同様に，各組み合わせを（ディ・全）算出法,（ファ・個）算出法，および（ファ・全）算出法と略す．各算出法の名称と観点の関係を**表 1** に示し，以下に各算出法を説明する．

3.3.2 （ディ・個）算出法

$$\text{ディレクトリ重要度} = \frac{\text{アクセスしたファイルの総ブロックアクセス回数}}{\text{アクセスしたファイルを構成する総ブロック数}} \tag{1}$$

式 (1) は，優先処理がアクセスしたファイルをディレクトリ毎に 1 つにまとめて，1 ブロック当たりのブロックアクセス回数を求めている．

3.3.3 （ディ・全）算出法

（観点 2）の対象ファイルを全ファイルとして求める算出法である．

$$\text{ディレクトリ重要度} = \frac{\text{アクセスしたファイルの総ブロックアクセス回数}}{\text{全ファイルを構成する総ブロック数}} \tag{2}$$

式 (2) は，ディレクトリ直下の全ファイルを構成する総ブロック数で計算しているため，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した場合の影響を考慮できる．このため，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，特にサイズの大きい優先処理がアクセスしないファイルを直下に有するディレクトリのディレクトリ重要度を（ディ・個）算出法と比べ，低くできる．

3.3.4 （ファ・個）算出法

（観点 1）の単位をファイルとするため，ファイルに重要度（以降，ファイル重要度と略す）を設ける．これにより，ディレクトリ直下のファイル毎に頻繁にアクセスされるか判断できる．

$$\text{ファイル重要度} = \frac{\text{ファイルのブロックアクセス回数}}{\text{ファイルを構成するブロック数}} \tag{3}$$

式 (3) は，ファイルの 1 ブロック当たりのブロックアクセス回数を求めている．また，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ようにした．この理由として，以下の 2 つがある．

（理由 1） ファイルサイズが小さければ，アクセスが多いファイルにブロックアクセスの少ないブロックが含まれる可能性が小さくなる．

（理由 2） 先読み機能により，ブロック数が大きいファイルは，ディスク I/O 回数を削減できる場合が多い．

なお，ファイル重要度が t 以上，つまり t 回以上アクセスされるファイルをファイル重要度が高いファイル（以降，高重要度ファイルと略す）とする．本稿では $t = 2$ とした．ファイル重要度を用いて，ディレクトリ直下の頻繁にアクセスされるファイルの割合がわかる．

$$\text{高重要度ファイル含有率} = \frac{\text{高重要度ファイルを構成する総ブロック数}}{\text{アクセスしたファイルを構成する総ブロック数}} \tag{4}$$

式 (4) では，ファイルが入出力バッファをどの程度使用するか考慮するため，ファイルを構成するブロック数を用いた．高重要度ファイル含有率を用いて，ディレクトリ重要度を算出する．

$$\text{ディレクトリ重要度} = \text{ファイル重要度の和} \times \text{高重要度ファイル含有率} \tag{5}$$

式 (5) は，ファイル重要度が高いファイルを多く持てば，ファイル重要度の和と高重要度ファイル含有率が共に大きくなり，ディレクトリ重要度は高くなる．一方，ファイル重要度が低いファイルを多く持てば，高重要度ファイル含有率が小さくなり，ディレクトリ重要度は低くなる．このため，ディレクトリ重要度の（特性）を満たしている．

3.3.5 （ファ・全）算出法

（観点 1）の単位をファイルとし，(観点 2）の対象ファイルを全ファイルとして，非優先処理を考慮した高重要度ファイル含有率を求めることができる．

$$\text{非優先処理を考慮した高重要度ファイル含有率} = \frac{\text{高重要度ファイルを構成する総ブロック数}}{\text{全ファイルを構成する総ブロック数}} \quad (6)$$

非優先処理を考慮した高重要度ファイル含有率を用いて，ディレクトリ重要度を算出する．

$$\text{ディレクトリ重要度} = \text{ファイル重要度の和} \times \text{非優先処理を考慮した高重要度ファイル含有率} \quad (7)$$

式 (6) は，ディレクトリ直下の全ファイルを構成する総ブロック数で計算しているため，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した場合の影響を考慮できる．このため，式 (7) 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，特にサイズの大きい優先処理がアクセスしないファイルを直下に有するディレクトリのディレクトリ重要度を（ファ・個）算出法に比べ，低くできる．

3.4 優先ディレクトリの選択方法

3.3 節の算出法で求めたディレクトリ重要度を用いて，優先ディレクトリに設定するディレクトリを以下に従い選択する．

$$\sum_{i=1}^{D}(\text{ディレクトリ } i \text{ が直下に有する全ファイルの合計サイズ}) < \text{入出力バッファサイズの } N\% \quad (8)$$

$$\text{全ファイルの合計サイズ} > \text{入出力バッファサイズ} \quad (9)$$

ディレクトリ重要度が高いディレクトリから順に，式 (8) が成立する間，優先ディレクトリに設定する．式 (8) を満たす最大の D を優先ディレクトリの上限個数とする．また，ディレクトリ i は，ディレクトリ重要度の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 としていく．なお，本稿では，保護プールとして利用可能な入出力バッファを制限する N を 100%とする．

ただし，式 (9) を満たすディレクトリは除外する．これは，優先ファイルの合計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)]．

表 2 各算出法の比較

算出法	長所	短所
（ディ・個）算出法	算出処理時間が比較的短い．	ファイル毎の重要度がわからないため，ディレクトリ直下のファイルの 1 ブロック当たりのブロックアクセス回数にばらつきがある処理を考慮できない．
（ディ・全）算出法	(1) 算出処理時間が比較的短い． (2) 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した場合を考慮できる．	ファイル毎の重要度がわからないため，ディレクトリ直下のファイルの 1 ブロック当たりのブロックアクセス回数にばらつきがある処理を考慮できない．
（ファ・個）算出法	ファイル毎の重要度がわかるため，ディレクトリ直下のファイルの 1 ブロック当たりのブロックアクセス回数にばらつきがある処理を考慮できる．	ファイル重要度が高いか否かの閾値の設定が難しい．
（ファ・全）算出法	(1) ファイル毎の重要度がわかるため，ディレクトリ直下のファイルの 1 ブロック当たりのブロックアクセス回数にばらつきがある処理を考慮できる． (2) 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した場合を考慮できる．	ファイル重要度が高いか否かの閾値の設定が難しい．

3.5 各算出法の比較

各算出法の比較を**表 2** に示す．

（ディ・個）算出法は，算出法が単純で計算量が少ないため，優先ディレクトリの設定に要する実行時間は抑えることができる．しかし，ファイル毎のアクセス頻度を考慮しないため，「ディレクトリ直下のファイルの 1 ブロック当たりのブロックアクセス回数にばらつきがある処理」を考慮できない．

（ディ・全）算出法は，(ディ・個）算出法の長所と短所に加えて，優先処理と非優先処理が共存し，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した場合を考慮できる．

（ファ・個）算出法は，ディレクトリ直下のファイル毎に重要度を算出するため，「ディレクトリ直下のファイルの 1 ブロック当たりのブロックアクセス回数にばらつきがある処理」を考慮できる．このような処理として，共存処理無しでカーネル make を実行した場合や共存処理無しで Web サーバを運用した場合がある．ただし，ファイル重要度が高いか否かの閾値を適切に設定することが難しい．

（ファ・全）算出法は,（ファ・個）算出法の長所と短所に加えて，優先処理と非優先処理が共存し，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した場合を考慮できる．このような処理として，カーネル make とカーネルのソースファイルのバックアップ処理が共存動作した場合や Web サーバを運用中に Web コンテンツのファイルをバックアップする場合がある．

4. 評　　価

4.1 評 価 内 容

評価では，本設定法を用いることにより，優先処理の動作内容に関する知識を持つ利用者が優先処理の実行時間を短縮するのに効果的な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，どの程度近い性能であるか評価する．

「ディレクトリ直下のファイルの 1 ブロック当たりのブロックアクセス回数にばらつきがある処理」の評価として，カーネル make と非優先処理を共存実行した場合と共存処理無しで Web サーバを実行した場合について評価する．

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 2.8GHz，メモリ : 32MB，入出力バッファ : 6.3MB，1 ブロックのサイズ : 8.0KB，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 32MB に制限した．また，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，実際に使用できる領域は約 5.7MB である．

測定前に make depend を実行し，make depend 実行完了直後に優先ディレクトリを設定した．設定した優先ディレクトリの情報を**表 3** に示す．表 3 の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 8.0KB の定数倍にまるめて合計した値である．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．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，他のヘッダファイルと比べ，より頻繁にアクセスされる．FreeBSD 4.3-R に元から実装されている LRU 方式，手動設定でのディレクトリ優先方式，および本設定法を用いたディレクトリ優先方式について，カーネル make の実行時間を比較した．

表 3 各方式の優先ディレクトリの情報

		手動設定	（ディ・個）算出法	（ディ・全）算出法	（ファ・個）算出法	（ファ・全）算出法
ディレクトリ	/usr/src/sys/sys/	なし	6 位	2 位	1 位	1 位
重要度の順位	/usr/src/sys/i386/include/	なし	7 位	3 位	2 位	2 位
優先ディレクトリ数		2 個	11 個	9 個	7 個	7 個
優先ファイルの合計サイズ（KB）		2,704	5,752	5,080	5,536	5,440
アクセスした優先ファイルの合計サイズ（KB）		2,032	3,088	3,048	3,464	4,144

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，カーネル make に関係ないファイルの連続読み込み処理を共存させてディスク I/O 負荷をかけた．この共存処理は，8 つのカーネル make に関係ない 1.0MB のファイルを順番に繰り返し読み込む．ファイルの読み込みは，1.0KB 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み込み処理の実行時間の方が長い．測定は 3 回行い，その平均値を評価に用いた．

4.2.2 評 価 結 果

カーネル make の実行時間の測定結果を**図 2** に示す．なお，以降の図では，ディレクトリ優先方式を DIR 方式と表記する．図 2 から，以下のことがわかる．

（1） 図 2 より，すべての算出法は，カーネル make の実行時間を LRU 方式より短縮できた．特に（ディ・個）算出法は，LRU 方式と比べ，119.74 秒（22.0%）短縮できた．これは，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に設定できたためである．

（2） 図 2 より,（ディ・個）算出法,（ディ・全）算出法，および（ファ・個）算出法は，手動設定よりカーネル make の実行時間を短縮できた．特に（ディ・個）算出法は，手動設定と比べ，13.33 秒（3.0%）短縮できた．これは，表 3 のアクセスした優先ファイルの合計サイズより，これらの算出法が，手動設定と比べ，多くのブロックを保護プールにキャッシュできたためである．

（3） 図 2 より,（ファ・全）算出法は，手動設定と比べ，カーネル make の実行時間が 14.29 秒（3.3%）長くなった．これは，保護プールに多くのブロックをキャッシュしすぎたことにより，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たためである．表 3 のアクセスした優先ファイルの合計サイズより,（ファ・全）算出法は，他の算出法の約 1.5 倍，手動設定の約 2 倍ファイルを保護プールにキャッシュしていることがわ

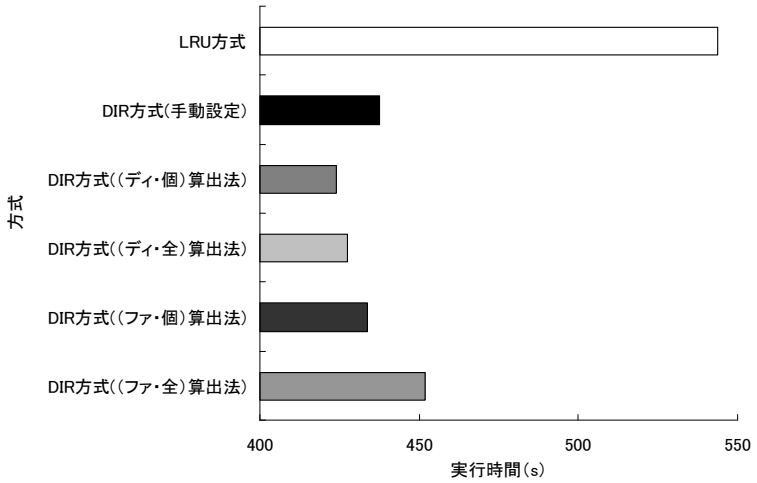

図 2 カーネル make の実行時間

かる．ただし，3.4 節の N の値により，保護プールにキャッシュできる優先ファイルの合計サイズは変更できる．

上記 (1) より，本設定法により，優先ディレクトリの設定を自動化しても，LRU 方式より高速であることがわかる．また，(2) と (3) より，本設定法は，実行時間が手動設定より長くなったとしても，手動設定と比べて 3.3%の増加に収まっている．この結果から，優先処理の動作内容に関する知識を持たなくても，手動設定に近い性能を発揮できることがわかる．

（4） 表 3 より，(ファ・個）算出法と（ファ・全）算出法は，カーネル make の高速化に効果的な優先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をディレクトリ重要度の順位の最上位に算出できている．保護プールとして利用可能な入出力バッファサイズを制限する N の値を小さくすることで，指定できる優先ディレクトリの個数は制限される．このため，優先処理の実行時間を短縮するのに効果的な優先ディレクトリを上位に導き出せることは重要であるといえる．

表 4 Web サーバによる評価に用いた計算機

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

4.3 Web サーバを用いた評価

4.3.1 評価環境と評価方法

共存処理無しで Web サーバを運用した場合の評価に用いた計算機を**表 4** に示す．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約 31MB である．

4.2.1 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 256MB 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，ディレクトリ優先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．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，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．

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．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．

評価で用いたファイルの情報を**表 5** に示す．手動設定は，**表 6** に示す 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．また，各算出法で算出したディレクトリ重要度と優先ディレクトリの選択方法に従い，優先ディレクトリを設定した．

本設定法を用いたディレクトリ優先方式を Web サーバに適用すると，頻繁にアクセスさ

表 5　評価で用いたファイルの情報 (100,000 回要求)

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（MB）	600.0

表 6　各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のアクセスに関する情報

ディレクトリ	構成ファイルの合計サイズ（KB）	ファイル数（個）	各ファイルの要求数の合計（回）
岡山大学	1,378	276	53,458
文学部	1,170	96	485
教育学部	231	9	154
医学部保健学科	183	92	1,857
農学部	8,303	361	2,461
大学院社会文化科学研究科	121	43	1,052

れるページを含むディレクトリを優先ディレクトリに指定する．このため，Web サーバの全体の応答時間を短縮できる．したがって，全ファイルの平均応答時間で評価した．

4.3.2 評 価 結 果

全ファイルの平均応答時間を**図 3** に示す．図 3 から，以下のことがわかる．

（1） 図 3 より，どの算出法を用いたとしても，本設定法を用いたディレクトリ優先方式は，全ファイルの平均応答時間を LRU 方式より短縮できていることがわかる．全ファイルの平均応答時間について，手動設定は，LRU 方式に比べ，0.54 ミリ秒（2.3%）短縮できた．これに対し，最も短縮できた（ファ・全）算出法は，LRU 方式に比べ，0.49 ミリ秒（2.1%）短縮でき，最も短縮できなかった（ディ・個）算出法は，LRU 方式に比べ，0.23 ミリ秒（1.0%）短縮できた．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する．このため，頻繁にアクセスされるページの表示に要する時間を特に短縮でき，利用者のストレスが軽減されると考えられる．

（2） 図 3 より，(ファ・個）算出法と（ファ・全）算出法は，手動設定に近い性能であることがわかる．これは，3.5 節のとおり，本評価で用いた処理が「ディレクトリ直下のファイルの 1 ブロック当たりのブロックアクセス回数にばらつきがある処理」であるためである．例えば，1 つのディレクトリに様々なページから参照される画像ファイルをまとめている場合もあるため，アクセス回数にばらつきが生じる．また，これにより，(ファ・個）算出法と（ファ・全）算出法が，共存処理無しで Web サーバを実行した場合に適していることがいえる．

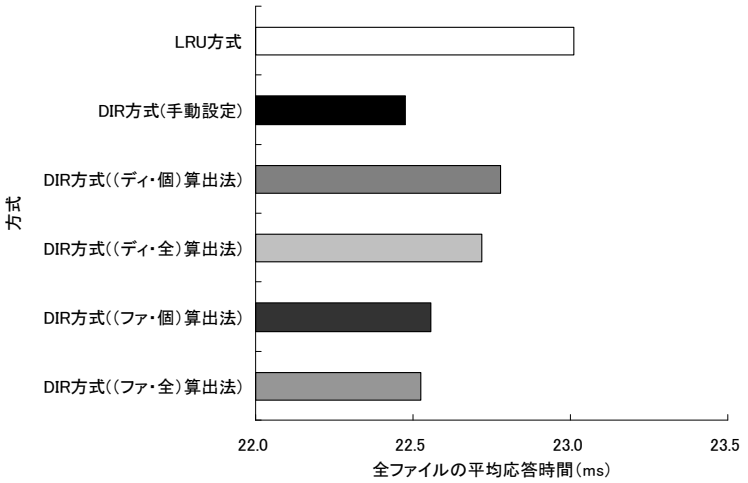

図 3　全ファイルの平均応答時間

5. 関 連 研 究

入出力バッファの制御方式[2)–8)] が提案されている．ブロックのアクセス頻度を基にした制御方式として，2Q[2)]，ARC[3)]，および CAR[4)] がある．2Q，ARC，および 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．利用者が与えるヒントを基にした制御方式として，ACFC[5)] がある．ACFC は，利用者が提供した応用プログラムに関するヒントに基づき，どのブロックを解放するか決定する．アクセスパターンを基にした制御方式として，UBM[6)] と PCC[7)] がある．UBM と PCC 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毎に領域を割り当て，アクセスパターン毎に異なる方式で各領域を管理する．ファイル操作を基にした制御方式として，FFU[8)] がある．FFU は，システムコールの発行頻度からファイル毎に重要度を設定し，重要度の高いファイルを構成するブロックを優先的にキャッシュする．

これらの方式とのディレクトリ優先方式の違いは，優先処理と非優先処理を区別して利用できることである．また，本設定法を用いると，優先処理の実行時間を短縮するのに効果

的な優先ディレクトリを一度求めてしまえば，以降，優先処理実行中に動的に変更するパラメータを最適に決定する処理は必要ない．このため，優先処理のアクセス情報を取得した後は，パラメータを最適に決定する処理のオーバヘッドを考慮しなくてよい．

6. おわりに

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や経験を必要とすることなく，優先ディレクトリを自動的に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，優先処理がアクセスするファイルを構成するブロックのアクセス回数やブロック数から，ディレクトリに重要度を設定する．このディレクトリ重要度を求める算出法を 4 つ示し，各算出法を比較した．また，求めたディレクトリ重要度が高いディレクトリから順に優先ディレクトリを選択する手法についても述べた．

カーネル make と非優先処理を共存実行した場合の評価において，本設定法を用いて優先ディレクトリを設定した場合，カーネル make の実行時間を LRU 方式より 119.74 秒（22.0%）短縮し，優先処理の動作内容に関する知識を持つ利用者が，優先処理の実行時間を短縮するのに効果的な優先ディレクトリを設定した場合[1)] と同等の結果が得られた．また，共存処理無しで Web サーバを実行した場合の評価において，本設定法を用いて優先ディレクトリを設定した場合，全ファイルの平均応答時間を LRU 方式より 0.49 ミリ秒（2.1%）短縮し，優先処理の動作内容に関する知識を持つ利用者が，優先処理の実行時間を短縮するのに効果的な優先ディレクトリを設定した場合[1)] と同等の結果が得られた．

残された課題として，優先ディレクトリの選択方法において，保護プールとして利用可能な入出力バッファを適切に制限する手法の確立がある．また，優先ディレクトリを設定するまでのオーバヘッドの評価，および各算出法に適している処理を用いた評価がある．

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) 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).
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) Cao, P., Felten, E.W. and Li, K.: Application-Controlled File Caching Policies, *Proc. the USENIX Summer 1994 Technical Conference*, pp.171–182 (1994).
6) 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).
7) 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).
8) 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).