優先処理の実行時間を短縮する入出力バッファ分割法

土谷　彰義†　　田端　利宏‡　　谷口　秀夫‡

†岡山大学工学部　　‡岡山大学大学院自然科学研究科

1 はじめに

入出力バッファを複数の領域に分割し，キャッシュヒット率を向上させる研究が行われている[1][2]．しかし，ファイルへのアクセスパターンは応用プログラム（以降，AP）毎に異なり，かつ利用者が優先して実行したい処理（以降，優先処理）とそうでない処理（以降，非優先処理）が共存実行されるため，様々なAPに対応できる入出力バッファの分割は容易ではない．そこで，本稿では，優先処理の実行時間を短縮できる2レベルの入出力バッファ分割法を提案する．

2 入出力バッファ分割法

2.1 ディレクトリ優先方式の問題点

2レベルの入出力バッファとしてディレクトリ優先方式[1]がある．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．また，保護プールに優先してキャッシュするファイルをディレクトリ単位で指定する．以降では，指定するディレクトリを優先ディレクトリ，優先ディレクトリ直下のファイルを優先ファイル，これ以外のファイルを非優先ファイルと呼ぶ．保護プールには，優先ファイルのバッファを格納し，通常プールには，非優先ファイルのバッファを格納する．バッファの解放時には，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する．これにより，優先処理が頻繁にアクセスするファイルを多く格納したディレクトリを優先ディレクトリに指定することで，優先処理の実行時間の短縮が期待できる．

しかし，この入出力バッファの分割方法は，優先処理の性能を低下させる場合がある．これは，優先処理がアクセスするファイルを格納しているディレクトリの全てを優先ディレクトリに指定することはできないことに起因する．優先処理がアクセスするファイルを格納したディレクトリ内には，優先処理が頻繁にアクセスしないファイルが含まれている．このため，これらのディレクトリの全てを優先ディレクトリに指定すると，多くのアクセス頻度が低いファイルのバッファを保護プール内に格納することとなり，優先ファイルのキャッシュヒット率が低下することがある．そこで，優先処理がアクセスするファイルの割合が多いディレクトリのみを優先ディレクトリに指定する．この場合は，優先処理のアクセスするファイルの一部は非優先ファイルとなる．このため，保護プールが入出力バッファの大部分を占めることにより，非優先ファイルのキャッシュヒット率が低下し，優先処理が非優先ファイルにもアクセスする場合，優先処理の実行時間が長くなる．したがって，優先処理が利用するファイルを考慮し，保護プールと通常プールの大きさを決定する分割法が必要である．

2.2 提案方式

優先処理の実行時間を短縮するには，通常プールに非優先ファイルのバッファも保持する必要がある．通常プールの領域を確保するため，保護プールサイズの上限を制限する．また，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，S_{max}）と現在の保護プールサイズ（以降，S_{cur}）の関係に基づき，バッファを解放するプールを選択する．選択規則を以下に示す．

(1) $S_{cur} < S_{max}$ の場合，S_{cur} 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．

(2) $S_{cur} = S_{max}$ の場合，S_{cur} を大きくできない．このため，読み込むブロックが優先ファイルのブロックか否かを判断する．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．

(3) $S_{cur} > S_{max}$ の場合，S_{cur} 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．

上記の規則に従い選択したプール内に解放できるバッファが存在すれば，選択したプールから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バッファを解放する．

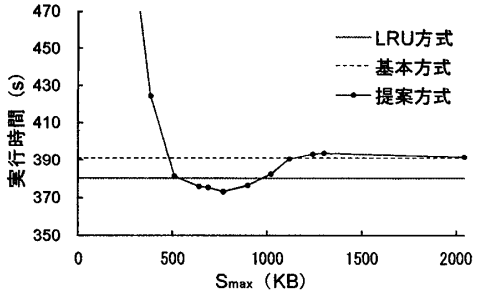
3 評価

3.1 測定環境と測定内容

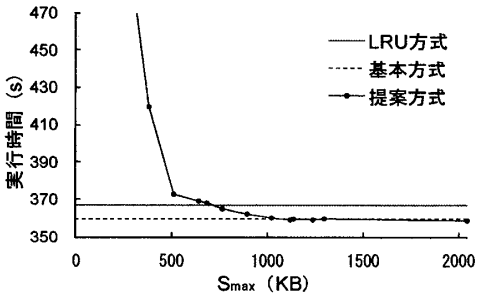
提案方式をFreeBSD 4.3-RELEASEに実装し，カーネルmakeの実行時間を測定した．入出力バッファの性能を評価するためにVMIO機能を停止させた．バックアップ先の計算機は，100Mbpsのイーサネットで接続し，オリジナルのFreeBSD 4.3-RELEASEをOSに用いた．測定は，FreeBSD 4.3-RELEASEに元から実装されているLRU方式，文献[1]で提案されたディレクトリ優先方式の基本方式（以降，基本方式），および提案方式について行った．

測定では，測定前にmake dependを実行した．提

I/O Buffer Partition Method to Improve Execution Time of Priority Processing
Akiyoshi Tsuchiya†, Toshihiro Tabata‡, and Hideo Taniguchi‡
†Faculty of Engineering, Okayama University
‡Graduate School of Natural Science and Technology, Okayama University



(A) 入出力バッファサイズ3.5MB



(B) 入出力バッファサイズ6.3MB

図1 カーネルmakeの実行時間

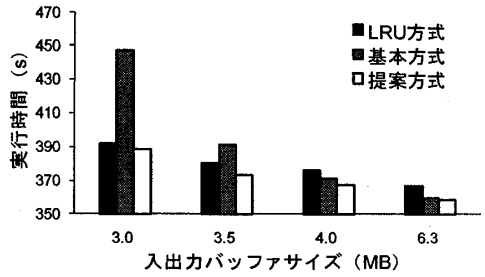


図2 カーネルmakeの実行時間（最良S_{max}の場合）

案方式と基本方式では，make depend 実行直後に，/usr/src/sys/sys/と/usr/src/sys/i386/include/を優先ディレクトリに指定した．バックアップ処理は，rsyncにより，カーネル make を行う計算機から他計算機に 1.0MB のファイルを 6,000 個転送するものである．

3.2 カーネルmakeの実行時間

図1にS_{max}の変化に伴うカーネル makeの実行時間の変化を示す．入出力バッファサイズが 3.5MBの場合，基本方式は，LRU 方式よりも実行時間が長い．これは，入出力バッファの多くを優先ファイルのバッファが占め，非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．提案方式で保護プールサイズの上限を適切な大きさに制限することにより，実行時間が短縮され，LRU 方式よりも実行時間が短い．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が小さく，保護プールサイズの拡大により，非優先ファイルのキャッシュヒット率は低下しない．このため，S_{max}が大きいほど提案方式の実行時間が短い．

次に，各入出力バッファサイズで，最良S_{max}に

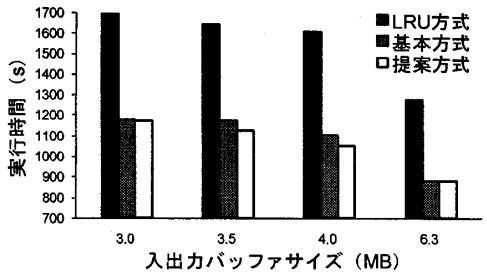


図3 カーネルmakeの実行時間（バックアップ処理が共存）

おける実行時間を**図 2** に示す．入出力バッファサイズが 3.0MB，3.5MB，4.0MB，および 6.3MB の場合では，それぞれ最良のS_{max}は，512KB，768KB，896KB，2048KB であった．入出力バッファサイズが 4.0MB 以上の場合，基本方式のほうが LRU 方式よりも実行時間が短い．また，全ての入出力バッファサイズで，提案方式が最も実行時間が短い．

3.3 共存処理の影響

カーネル make とバックアップ処理が共存動作した場合の実行時間を**図 3** に示す．提案方式のS_{max}は，3.2 節の測定と同じにした．全ての場合で，共存するバックアップ処理の影響により，カーネル make のみを実行した場合と比べて実行時間が長くなっている．しかし，基本方式と提案方式では，LRU 方式と比べてカーネル make の実行時間の増加を抑えることができている．これは，カーネル make が頻繁にアクセスするファイルのバッファを保護プールに保持し，優先的にキャッシュできるためである．

全ての入出力バッファサイズで，提案方式が最も実行時間が短い．また，全ての入出力バッファサイズで，提案方式は，LRU 方式と比べて実行時間を 30%以上短縮できている．提案方式の実行時間は，基本方式と比べて短いか同等である．この結果から，バックアップ処理が共存する場合でも適切にS_{max}を設定することが有効であることがわかる．

4 おわりに

優先処理の実行時間を短縮する入出力バッファ分割法を提案した．カーネル make による評価では，共存処理の有無に関係なく，提案方式が最も実行時間が短いことを示した．

残された課題として，様々な AP を用いた提案方式の詳細な評価とS_{max}の設定自動化がある．

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] T. Johnson and D. Shasha, “2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm,” Proc. the 20th VLDB, pp.439-450, 1994.