情報処理学会論文誌　Vol. 50　No. 11　2690–2698 (Nov. 2009)

同期書き込みでのファイル同時作成時におけるフラグメントと性能を改善するサイズ調整プリアロケーション方式

中村隆喜[†1]　薦田憲久[†2]

様々なサイズのファイルの同期書き込みによる同時作成が発生する高信頼ファイルサーバ向けに，フォーマット時などにおける事前設定なしに，フラグメントを防止するサイズ調整プリアロケーション方式を提案する．伸長中ファイルサイズが小さいときには小さいプリアロケーションサイズを適用し，大きいときには大きいプリアロケーションサイズを適用する．提案方式は，一般的なPCのファイルサイズ分布に基づくワークロードにおいて，数十KB程度の小サイズファイルの内部ディスクフラグメントと，数十MBから数GB程度の大サイズファイルのファイルフラグメントと，アクセス性能の改善に効果があることが確認できた．

Size Adjusting Pre-allocation Methods to Improve Fragmentation and Performance on Simultaneous File Creation by Synchronous Write

Takaki Nakamura[†1] and Norihisa Komoda[†2]

We propose size adjusting pre-allocation methods to prevent file fragmentation without pre-configuration on filesystem format for high-reliable file server which serves simultaneous and various sized file creation by synchronous write. Proposed methods apply small size pre-allocation when growing file size is small and large size pre-allocation when growing file size is large. We confirmed that proposed methods have effects for improving internal fragmentation of disk for small sized file around a couple 10 KB, file fragmentation for large sized file from a couple 10 MB to a couple GB, and access performance under a workload based on file size distribution of general PC.

1. はじめに

近年，ファイルを用いた情報共有の活性化にともない，ファイルサーバとストレージ装置を一体化したNetwork Attached Storage（NAS）の市場が拡大している．NASに格納されるコンテンツとしては，テキストファイルのような小サイズ（数KB〜数十KB程度）のもの，ビジネスドキュメントファイルのような中サイズ（数MB程度）のもの，ストリーミングデータファイルのような大サイズ（数十MB〜数GB程度）のものが考えられる[1)]．

小サイズファイルのみで占めるファイルシステムボリュームや大サイズファイルのみで占めるファイルシステムボリュームを同一のファイルシステムプログラムで動作させる場合，ディスク上の非効率な箇所にファイルを格納するフラグメントと呼ばれる現象が発生しやすくなる．フラグメントにはファイルフラグメント，内部ディスクフラグメント，外部ディスクフラグメントとの3種類がある．

ファイルフラグメントは，あるファイルを追記書きによって伸長する際に，そのファイルが格納されている領域の連続領域が別ファイルによって使用されてしまっている状況で発生する．本現象は，ファイルの格納領域を不連続かつ分散的な状態にする．ファイルフラグメントが発生すると，その断片化された状態を管理するためのレイアウト情報のためにディスク容量・メモリなどのリソースを圧迫する．また，シーケンシャルアクセス，削除処理などのアクセス性能が低下する[2)]．

内部ディスクフラグメントは，ファイルサイズがファイルに割り当てたディスクの領域の割当て単位サイズ（ファイルシステムブロックサイズ）と比べて小さい状況で発生する．本現象では，ファイルに割り当てられた領域の一部のみを使用し，残りを使用できない状態にしてしまう．したがって，内部ディスクフラグメントが発生すると，ディスク容量の使用効率が低下する．内部ディスクフラグメントを防止する方法として，分割ブロックアロケーション方式が提案されている[3),4)]．

外部ディスクフラグメントは，ファイルシステムフルに近い状況下で，ファイルシステムのエージングが起こったときに発生する．本現象では，ファイルシステムの空き領域がディスク上に広く分散する．この状況は，ファイルフラグメントも起こしやすいことが知られて

†1 株式会社日立製作所システム開発研究所
Systems Development Laboratory, Hitachi Ltd.
†2 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technologies, Osaka University

いる．本論文では，ファイルシステムの空き容量を十分とった形で実験を行い，前者 2 種類のフラグメント，つまりファイルフラグメントと内部ディスクフラグメントを中心に評価する．

従来のフラグメント防止方式では，様々なサイズのファイルを同期書き込みにおいて同時に多数作成するケースで，ファイルフラグメントと内部ディスクフラグメントの両方を防止することが難しかった．また，それにともないアクセス性能も低下していた．たとえばそのようなケースの例として，多クライアント環境での NAS やファイルサーバに加えて，複数の監視対象機器からのログを同時に作成する障害監視サーバ，近年急速に普及している多人数による同時格納可能な動画共有サーバがある．これらのサーバは信頼性を高めるため，書き込みを同期で実行する運用をとることがある．

一般にファイルは，ユーザアプリケーションもしくはファイルクライアントからの書き込み要求を複数回繰り返して伸長し，作成される．書き込み要求とは，ローカルファイルシステムであればライトシステムコールであり，ネットワークファイルシステムであればライトプロシージャである．

本研究ではこの伸長中のファイルサイズに着目し，伸長中のファイルサイズが小さいときには内部ディスクフラグメントを起こしにくい設定値を，伸長中のファイルサイズが大きいときにはファイルフラグメントを起こしにくい設定値を適用することで，両フラグメントを防止し，アクセス性能を改善するサイズ調整プリアロケーション方式を提案する[5),6)]．

本論文の以降の構成は次のとおりである．2 章では，従来のフラグメント防止技術を紹介し，その課題について述べる．3 章では，提案方式であるサイズ調整プリアロケーション方式を説明し，調整方法に関する検討と実装の詳細を述べる．4 章では，試作を用いた測定と模擬実験に基づき，その効果を確認する．最後に 5 章で，本研究のまとめを述べる．

2. 従来のフラグメント防止技術とその課題

本章では従来のフラグメント防止技術である 3 つの方式，遅延アロケーション方式，割当てブロック単位サイズ選択方式，プリアロケーション方式について説明し，その課題について述べる．

2.1 遅延アロケーション方式

ファイル新規書き込み要求の際に，書き込み先となる具体的なディスク領域を割り当てるのではなく，ディスクの空き領域のサイズのみを割り当てる方式であり，最近のほとんどのファイルシステムが採用している．具体的なディスク領域の割当ては，フラッシュデーモンや sync 要求などにより実行されるデータフラッシュ処理の延長で行う．これにより同一ファイルに対して連続する新規書き込み要求のデータを，連続の領域に格納できる可能性が高まる．本方式はファイルフラグメントの低減に有効な手法であり，大サイズファイルにも効果がある．ただし，非同期書き込みにしか適用できない．

2.2 割当てブロック単位サイズ選択方式

ファイルに割り当てる領域の最小単位である割当てブロックのサイズを複数用意し，それを選択する方式である．フォーマット時に選択する方法と，フォーマット後に選択する方法がある．

(1) フォーマット時サイズ選択方式

ファイルシステムのフォーマット時に割当てブロックサイズを 1 つ選択して，作成する方式である．本方式はファイルシステム内に同一サイズのファイルのみが格納される場合には有効な方式である．多くのファイルシステムが採用している．

(2) フォーマット後サイズ選択方式

ファイルシステムのフォーマット時には複数のサイズの割当てブロックで作成して，ファイル書き込み時にいずれかのサイズの割当てブロックを選択する方式である[7)–9)]．小サイズファイル用の数 KB 程度の小サイズ割当てブロックと，通常サイズファイル用の数 10 KB 程度の中サイズ割当てブロックの 2 つのサイズのブロックを使い分けるのが一般的である．本方式は内部ディスクフラグメントの低減に有効な手法である．ただし，本方式は大サイズのファイルのファイルフラグメント防止には効果が低い．大サイズファイル用の大サイズ割当てブロックを用意すれば，効果が上がる可能性があるが，フォーマット時の各サイズの割当てブロックの配分の決定がより難しくなってしまう．

2.3 プリアロケーション方式

将来の書き込み要求に備え，あらかじめファイルの書き込み領域を事前に割り当てておく方式である[10)]．事前に割り当てておくことをプリアロケーションという．本方式はファイルフラグメントの低減に有効な手法である．

本方式は 2.2 節の方式と異なり，割当てブロック単位サイズは一定とし，事前に連続した領域の複数の割当てブロックをファイルに割り当てる．このようにすれば，割当てブロックサイズの配分などを事前に考慮・設計する必要がない．また，書き込み終了後，未使用の割当てブロックは解放することができるので，内部ディスクフラグメント低減にも一定の効果がある．

本方式にはシステムが自動的にプリアロケーションする方式と，ユーザがプリアロケー

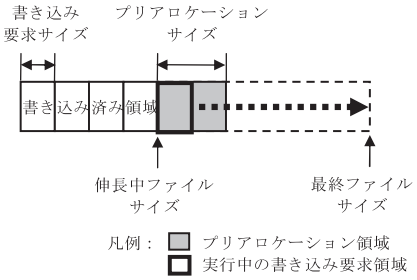

図 1　作成中のファイルの状態
Fig. 1　Status of growing file.

ションする方式がある．

(1) システムプリアロケーション方式

書き込み要求の際に，その書き込み要求のサイズに加えて，10 KB〜数 10 KB 程度の一定サイズの領域を割り当てる方式であり，XFS，EXT2，3，4 に採用されている[11)–13)]．**図 1** は，作成中のあるファイルの状態である．ファイルは書き込み要求を複数回繰り返して伸長し，作成される．図では伸長中ファイルサイズまでファイルが伸長し，そのオフセットから書き込み要求が実行されている状態である．システムプリアロケーション方式では，この書き込み要求時に，次回以降の書き込み要求のための領域を割り当てる．本方式はファイルフラグメント低減に有効な方式であるが，プリアロケーションサイズが小さいため動画ファイルのような大サイズのファイルには効果が低い．

(2) ユーザプリアロケーション方式

ユーザもしくはユーザアプリケーションがあるファイルの最終サイズを予見し，書き込み前にその予想最終サイズでプリアロケーションする方式であり，VxFS に採用されている[14)]．本方式はファイルフラグメント低減に有効な方式であり，適切に使用すれば効果は大きい．ただし，ユーザもしくはユーザアプリケーションが最終サイズを予見できないログファイルのようなファイルには適用できない．また，NAS のような形態ではネットワークファイルシステムプロトコルにプリアロケーションを指示するプロシージャがないため適用が難しい．さらに，ユーザがファイルごとに個別に指定しなければならないので管理が煩雑である．

3. サイズ調整プリアロケーション方式

本章では，まず従来方式の課題に対する解決のアプローチを述べる．次に，提案するサイズ調整プリアロケーション方式の予備実験を実施し，サイズの調整方法に関する事前検討を行う．最後に，提案方式の適用範囲と実装に関してまとめる．

3.1　従来方式の課題に対する解決のアプローチ

2 章で述べたとおり，従来技術は大部分の条件下でフラグメントを防止可能であるが，以下の 3 つの条件が同時に成立する場合，フラグメントを防止するのは困難であった．

(A)　同期的に多数ファイルの書き込みが発生

(B)　ファイルサイズやファイルサイズ分布の事前予測や事前通知が困難

(C)　特に大サイズファイルを含む任意のファイルサイズが混在

まず，条件 (A) を解決するには，遅延アロケーション方式の採用は難しい．次に条件 (B) を解決するには，ユーザプリアロケーション方式，割当てブロック単位サイズ選択方式の採用は難しい．条件 (C) を解決するには，ファイルサイズに応じて割当てサイズを変更する方式が考えられる．つまり，3 条件下でのフラグメントを防止するには，システムプリアロケーション方式をベースとして，割当てサイズ，すなわちプリアロケーションサイズをファイルサイズに応じて変更する解決方式が考えられる．

そこで，伸長中のファイルサイズが小さいときには内部ディスクフラグメントを起こしにくいプリアロケーションサイズを，伸長中のファイルサイズが大きいときにはファイルフラグメントを起こしにくいプリアロケーションサイズを適用することで，両フラグメントを防止し，アクセス性能を改善するサイズ調整プリアロケーション方式を提案する．

従来のシステムプリアロケーション方式でのプリアロケーションサイズは伸長中のファイルサイズにかかわらず一定であったが，提案方式では伸長中のファイルサイズに応じて，プリアロケーションのサイズを変化させる．つまり，ファイルが伸長するにつれて，プリアロケーションのサイズも大きくしていく．

3.2　プリアロケーションサイズの調整方法に関する検討

本節ではプリアロケーションサイズをどのように調整するのが良いかを事前検討する．検討のため，ファイルフラグメント数（以降，フラグメント数と略す）もしくは平均連続長と，読み込み，書き込み，削除の各性能がそれぞれどのような相関関係にあるかを調査した．

ここでフラグメント数とは，ファイルが何箇所の不連続領域に断片化しているかを示す値である．たとえば，まったくファイルフラグメントしていない状態のフラグメント数は 1 で

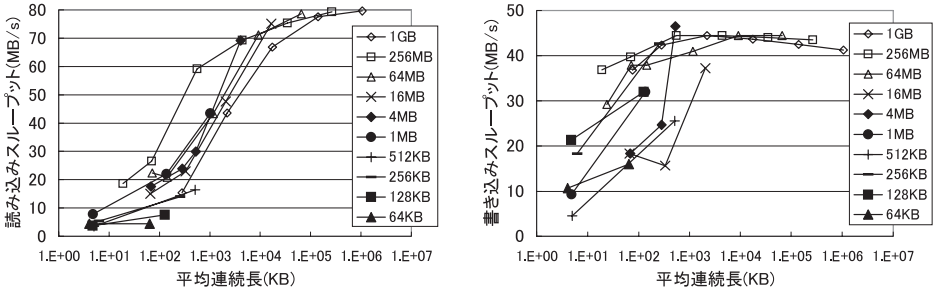

図 2　読み込み・書き込み性能と平均連続長の関係
Fig. 2　Read/Write throughput as a function of average continuous size.

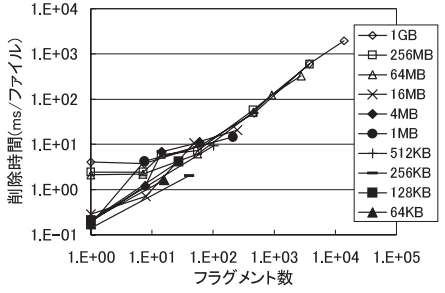

図 3　削除時間とフラグメント数の関係
Fig. 3　Removal time as a function of number of fragmentation.

ある．平均連続長は，そのファイルがどのくらいの長さ平均して連続領域に格納されているかを示す値で，ファイルサイズ/フラグメント数に相当する．

予備実験はシステムプリアロケーション方式を採用している Linux 2.4 と XFS 1.2 を改造して行った[15]．様々なフラグメント数の状態のファイルを得るため，XFS のシステムプリアロケーションサイズ（デフォルト値 64 KB）の値を変更し，ファイルを作成した．なお本予備実験ではファイル作成開始から作成完了までの間は，プリアロケーションサイズは変化させない．測定を行うファイルサイズは 64 KB〜1 GB の 10 種類とした．ファイルサイズ 64 KB〜16 MB の 7 種類は 512 個を同時に，64 MB〜1 GB の 3 種類は 16 個を同時に，同期書き込みで作成した．

読み込み性能とファイルの平均連続長の相関関係を**図 2** の左に示す．横軸はファイルの平均連続長のログスケール，縦軸は読み込みスループットのリニアスケールである．平均連続長が数 100 KB 程度までは性能に大きな改善は見られない．平均連続長 1 MB〜10 MB にかけて急激に立ち上がり，それ以上大きくなっても限界性能に達するためほとんど改善されない．

書き込み性能とファイルの平均連続長の相関関係を図 2 の右に示す．読み込みよりは早く立ち上がり，平均連続長 1 MB 程度で性能は飽和する．

次に，削除時間とフラグメント数の相関関係を**図 3** に示す．横軸はファイルのフラグメント数，縦軸は 1 ファイルあたりの削除時間，両軸ともログスケールである．図から明らかなように XFS においては削除時間とフラグメント数は基本的に比例する．

以上の結果より，最終ファイルサイズが予見できない場合は伸長中ファイルサイズの増加にともない，プリアロケーションサイズを増加させるのが良いことが分かる．

読み込み性能，書き込み性能を重視する場合は，伸長中ファイルサイズが数 100 KB までは小さくても良いが，それを超えた場合急激に立ち上がり，最終的には平均連続長が 10 MB を超えるようにプリアロケーションサイズを調整するのが良い．

削除性能を重視する場合は，フラグメント数が少しでも小さくなるようにプリアロケーションサイズを大きくすべきである．ただし闇雲にプリアロケーションサイズを大きくすると，内部ディスクフラグメントが増大するので，伸長中ファイルサイズが小さいときは小さいプリアロケーションサイズを適用すべきである．

これらの要求に応えるには，傾きの大きい一次関数，対数関数，階段関数の採用が考えられる．ただし一次関数は無制限にプリアロケーションサイズが増加してしまうので，内部ディスクフラグメント防止の観点から上限値を設けておくのが良い．対数関数や階段関数も同様に，最終的に読み込み，書き込み性能が飽和する平均連続長になるような設定とすべきである．

3.3　提案方式の適用範囲

提案方式はシステムプリアロケーション方式をベースとしているので，適用範囲は同ベース方式が採用可能なファイルシステムに限られる．たとえば，NetApp WAFL[16] や ZFS[17] など，COW ベースのログストラクチャードファイルシステムへの適用は困難である．一方，EXT2，3，4 ではシステムプリアロケーション方式を採用しているので，カーネル内のシステムプリアロケーション処理箇所の特定ができれば，提案方式の実装は比較的容易である．

また提案方式が効果的に働くのは，複数のファイルが同時に作成される，もしくは伸長するケースである．単一ファイルが作成され，伸長するケースでは，通常連続する領域が未使用でアロケート可能であるため，提案方式なしにフラグメント防止が可能である．

提案方式は，主に同期書き込みに対して効果的である．非同期書き込みでは 2.1 節に述べ

た遅延アロケーション方式によって基本的にフラグメントの防止が可能である．ただし，非同期書き込みでもダーティーデータがファイルサーバのメモリ上限を超えると，フラグメントの結合が行われる前にディスクにフラッシュされることになり，そのようなケースでは提案方式は効果がある．

提案方式は，数十 MB〜数 GB の大サイズファイルのファイルフラグメントを防止するのに特に効果的であるが，任意のファイルサイズ分布に適用可能である．

3.4　提案方式実装

本提案方式を，予備実験と同様に Linux 2.4 と XFS 1.2 をベースに実装した．XFS は SGI 社により開発された高機能，高信頼なファイルシステムである．XFS は，フラグメントを防止する方式として，遅延アロケーション方式，フォーマット時サイズ選択方式，システムプリアロケーション方式を備えている．システムプリアロケーションのサイズは 64 KB の固定長である．

書き込み処理を実行する関数である，`linux/fs/xfs/pagebuf/page_buf_io.c` の `_pagebuf_file_write()` のプリアロケーションサイズ設定部を**図 4** のように変更した．

`f(current_file_size)` がプリアロケーションサイズを決定する関数であり，計算されたプリアロケーションサイズがブロックサイズのアラインに合わない場合は，ブロックサイズに合うように切り捨てる．また，プリアロケーション時の容量不足で失敗した場合には，書き込み要求サイズのみの割当てを行う処理を追加した．これは，計算されたプリアロケーションのサイズがファイルシステムの空き容量に対して大きすぎると，書き込み要求自体が実行可能な空き領域があるにもかかわらず，書き込み要求も巻き込まれて失敗するケースを防止するための処理である．

```
{
  ...
  psize = f(current_file_size);
  prize = max(psize, BLOCKSIZE);
  psize = (psize / BLOCKSIZE ) * BLOCKSIZE;
  err = preallocate(psize);
  if (err == ENOSPC) {
    err = preallocate(request_write_size);
  }
  if (!err) err = exec_write();
  return err;
}
```

図 4　提案方式の模擬コード
Fig. 4　Pseudo-codes of proposed methods.

XFS ではファイル close 時に，プリアロケートしたが，実際には使用しなかった領域を解放する処理が実装されている．本処理により，提案方式の内部ディスクフラグメントの発生をファイル open 中に限定することができる．領域解放処理は `linux/fs/xfs/xfs_vnodeops.c` の `xfs_release()` 内に実装されている．

4.　提案方式の評価

提案方式を実装したシステムを用いて，フラグメント数，書き込み性能，読み込み性能，削除性能，内部ディスクフラグメントの観点で，評価を行う．

4.1　測定環境と測定条件

測定システムとして，2.8 GHz Xeon × 2，メモリ 3.2 GB の IA32 機を用いた．530 個の様々なサイズのファイルを同期書き込みで同時作成し，フラグメント数，書き込み性能，読み込み性能，削除性能を測定する．530 個のファイル分布のうち 500 個は PC の統計的なファイル分布[18] に基づく（**表 1**）．残りの 30 個は本提案の効果がより顕著に現れる，64 MB，256 MB，1 GB の大サイズファイルをそれぞれ 10 個ずつとした．具体的なファイル分布は表 1 のとおりである．

測定で用いるプリアロケーションサイズ決定関数を**表 2** に示す．本表のプリアロケーションサイズ決定関数を変えて 530 個のファイルを作成して測定を実施し，その結果を比較する．3.2 節の事前検討に基づき，伸長中ファイルサイズに応じてプリアロケーションサイズを変更させる方式として，一次関数，対数関数，階段関数を採用した．またこれに加えて，

表 1　作成ファイルのサイズ分布
Table 1　Size distribution of creation files.

ファイルサイズ	ファイル数	ファイルサイズ	ファイル数	ファイルサイズ	ファイル数
4KB	250	128KB	20	16MB	5
8KB	50	256KB	20	64MB	10
16KB	50	512KB	10	256MB	10
32KB	50	1MB	10	1GB	10
64KB	30	4MB	5	—	—

表 2　使用するプリアロケーションサイズ決定関数
Table 2　List of pre-allocation size function.

#	プリアロケーションサイズ決定関数
1	64KB 固定（XFS オリジナルの方式）
2	16MB 固定
3	一次関数 伸長中サイズ × 1/8
4	一次関数 伸長中サイズ × 1/8　上限 32MB　下限 64KB
5	一次関数 伸長中サイズ × 1/4　上限 32MB　下限 64KB
6	一次関数 伸長中サイズ × 2　　上限 32MB　下限 64KB
7	対数関数 2MB×log2(伸長中サイズ/4KB)
8	対数関数 2MB×log2(伸長中サイズ/4KB) 伸長中サイズ 64KB 未満は 64KB 固定
9	階段関数 伸長中サイズ 64KB 以上は 16MB 固定, 64KB 未満は 64KB 固定

XFS オリジナルの方式である 64 KB 固定（#1），単純にシステムプリアロケーションのサイズを大きくした 16 MB 固定（#2）も測定する．

本論文の評価で #3–#8 のサイズ決定関数を用いた理由は次のとおりである．一次関数において最適な傾きを評価するため，#4，#5，#6 の条件を設定した．一次関数，対数関数，階段関数の特性を比較評価するため，#4，#7，#9 の条件を設定した．プリアロケーションサイズに上下限値を設定した場合の，性能低下と内部ディスクフラグメントの改善の影響を比較評価するため，#3，#4 および #7，#8 の条件を設定した．#4，#8 の下限値は従来方式と合わせ，#4 の上限値は読み込み，書き込み性能がほぼ飽和する値とした．

なお，本測定での同期書き込みとは，ブロック割当てだけではなく，データ自体も同期で書き込まれる．また非同期書き込み時の提案方式の影響を確認するため，#1，#4 の条件のみ非同期書き込みでの測定も実施する．

また，提案方式と従来方式を公平に比較する観点で，530 個の平均値と合わせて，本発明の効果がより顕著に現れる 30 個のファイルを除いた 500 個のファイルのみの平均値もあわせて分析，考察する．

4.2　フラグメント数の評価

各プリアロケーション条件においてファイル作成した際の 1 ファイルあたりのフラグメント数を**図 5** に示す．同期書き込み時には，提案方式の適用により，500 ファイル平均の #3 を除いてオリジナル XFS の条件よりもフラグメント数が改善された．特に 530 ファイル平均では，約 30〜110 倍と大幅に改善される．非同期書き込み時には，提案方式のフラグメント数は，従来方式に比べて若干増加する．

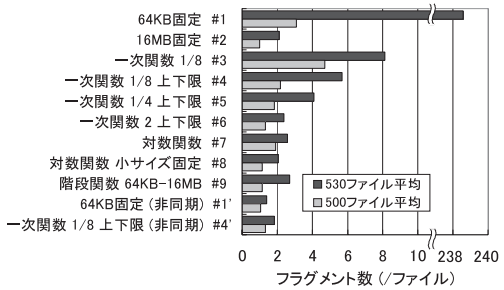

図 5　各測定条件におけるフラグメント数
Fig. 5　Comparison of a file fragmentation number.

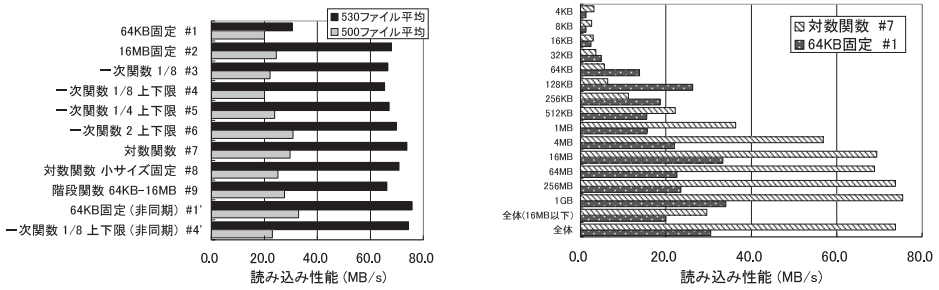

図 6　各測定条件における読み込み性能
Fig. 6　Comparison of read throughput.

一次関数に関しては，#3，#4 の比較より，下限値を設けることがフラグメント数低減にさらに効果があることが分かる．また #4，#5，#6 の比較より，傾きが大きければ大きいほどフラグメント数低減効果が高いことが分かる．

対数関数に関しても下限値を設定することでフラグメント数がさらに低減する．

階段関数は，対数関数や，傾き 2 の一次関数と同等のフラグメント低減効果がある．

4.3　読み込み性能の評価

各条件において作成したファイルのすべてをシーケンシャルで読み込んだ際の合計スループットを**図 6** の左に示す．

530 ファイル平均では提案方式のいずれの条件も，従来方式に対して大幅にスループット性能が改善された．500 ファイル平均では若干性能が改善された．提案方式の条件間の大き

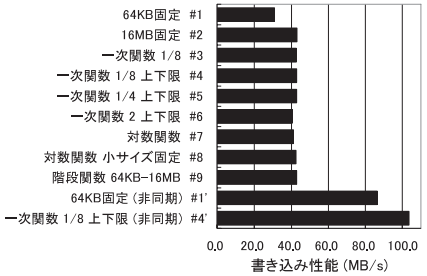

図 7　各測定条件における書き込み性能
Fig. 7　Comparison of write throughput.

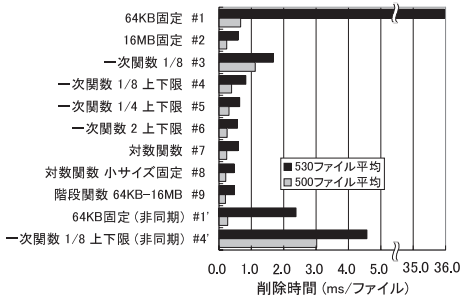

図 8　各測定条件における削除時間
Fig. 8　Comparison of removal time.

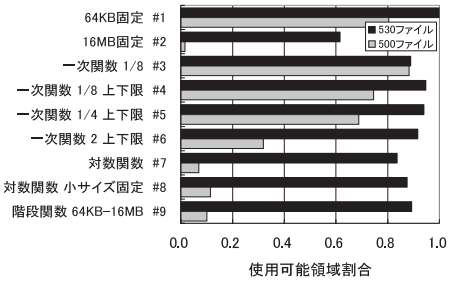

図 9　各測定条件における使用可能領域割合
Fig. 9　Comparison of useable space percentage.

な性能の違いはないが，#7 対数関数が最も性能を改善する結果となった．

一次関数は傾きが強ければ強いほど性能が改善される．また上下限値を設けても読み込み性能にはそれほど大きいインパクトはないことが分かった．

非同期書き込み時は，提案方式が従来方式に比べて若干性能低下する結果となった．

ファイルサイズごとの読み込み性能について，XFS のオリジナル方式と対数関数とを比較した結果を図 6 の右に示す．対数関数では大サイズファイルから中サイズファイルまで大きく性能が改善できている．

4.4　書き込み性能の評価

各条件において最初にファイルを作成開始した時間から，最後のファイル書き込みが完了した時間の差より算出した書き込みスループットを**図 7** に示す．

読み込み性能ほどではないが，書き込み性能に関してもスループットが大幅に改善された．提案方式の条件間の大きな性能の違いはない．また，非同期書き込み時においても，提案方式が従来方式に比べて性能向上する結果となった．

4.5　削除性能の評価

各条件において作成したファイルをすべて削除したときの，1 ファイルあたりの削除時間を**図 8** に示す．同期書き込み時において，提案方式の適用により，500 ファイル平均の #3 を除いて性能が改善された．これは削除時間がフラグメント数とほぼ比例の関係にあることを示した予備実験の結果を裏付ける．ただし，フラグメント数の結果と比較して，#3，#4 の差が大きいことや，#9 が短くなっていることをふまえると，フラグメント数が同程度の場合，XFS ではプリアロケーションサイズがそろっていたほうが，削除時間が早くなる傾向がある．これは管理テーブルのサイズが小さくなるためと推定する．

削除時間を重視する場合は，#8 もしくは #9 のプリアロケーションサイズ関数にするのが良い．

非同期書き込み時には，全体的に同期書き込み時よりも削除時間が増加する傾向がある．非同期書き込みの場合，ディスクフラッシュ間隔が長いため，フラグメント数が同程度でも，位置的な分散がより強いためと推定する．なお非同期書き込み時の条件の違いの観点では，フラグメント数の結果と同様に従来方式のほうが良い．

4.6　内部ディスクフラグメントの評価

最後に，各条件において最悪ケースでの使用可能割合を**図 9** に示す．最悪ケースとは 530 個もしくは 500 個のファイルがファイル書き込みの open 中で余分な領域が解放されていない状態である．1 は内部ディスクフラグメントがまったく発生しない状況で，この値に近け

れば近いほど良い．

530 ファイルにおいて #2 の 16 MB 固定は，最悪で約 40%の内部ディスクフラグメントが発生する．これに対し，提案方式は約 5〜15%程度の内部ディスクフラグメントに抑えることができた．特に，一次関数においてプリアロケーションの上限値や，対数関数において小サイズ向けの固定値を設けることは内部ディスクフラグメント割合の改善に効果がある．

500 ファイルの結果では #2 はさらに悪化し，最悪で約 98%の内部ディスクフラグメントが発生する．提案方式においても，急激にプリアロケーションサイズが伸長する関数である #6，#7，#8，#9 では大幅に低下する．ただし，500 ファイルのファイルサイズは最大で 16 MB であり，ファイル書き込みの open 時間はそれほど長くないため，過度に深刻に考える必要はない．

4.7　測定結果の全体的な考察

提案方式は，同期書き込み時において内部ディスクフラグメントの増加を抑えつつ，読み込み，書き込み，削除の各性能を改善できた．特に大サイズファイルを含む場合大幅に改善する．これに対し，固定値を単に大きくする方式では，性能は同様に改善できるが，内部ディスクフラグメントの割合を大幅に増加してしまう．

提案方式の中で性能を重視するならば，対数関数のような急激に立ち上がる関数を用いるのが良い．大サイズのファイルが格納される可能性が低い場合には，1/8 から 1/4 程度の傾きの一次関数で上下限を設定するのが良い．また，実装の容易な階段関数でも十分な効果が得られる．

非同期書き込み時には，若干のフラグメント数増加にともない，一部性能も若干低下するが，許容範囲内であると考える．提案方式の実装箇所は，カーネルのファイルシステム処理部であるため，書き込みが同期か非同期の情報を取得することが可能である．したがって，性能低下が許容不可能な場合は，非同期書き込み時には提案方式をオフにすることもできる．

5. おわりに

本論文では，大サイズファイルに対してはファイルフラグメント防止効果が高く，小サイズファイルに対しては内部ディスクフラグメント防止効果が高い，サイズ調整プリアロケーション方式を提案した．評価システムを用いて，提案方式がファイルフラグメント，内部ディスクフラグメントの両面で高い効果があることを確認した．また，従来方式ではフラグメントを防止するためにユーザに事前準備などの余計な作業を強いることがあったが，本提案方式ではプリアロケーションサイズの決定ルールはシステムが提供するのでユーザに余計な作業を強いることはない．

参考文献

1) Douceur, J.R. and Bolosky, W.J.: A large-scale study of file-system contents, *Proc. SIGMETRICS '99*, pp.59–69 (1999).
2) Smith, K.A. and Seltzer, M.I.: File Layout and File System Performance, Harvard University Technical Report TR-35-94 (1994).
3) McKusick, M.K., Joy, W.N., Leffler S.J. and Fabry, R.S.: A Fast File System for UNIX, *ACM Trans. Computer Systems*, Vol.2, No.3, pp.181–197 (1984).
4) JFS fragment size (online). available from http://publib.boulder.ibm.com/infocenter/systems/index.jsp?topic=/com.ibm.aix.prftungd/doc/prftungd/jfs_frag_size.htm (accessed 2009-08-07).
5) 中村隆喜：RAID システム内蔵型 NAS (2)—高信頼ファイルシステム，第 3 回情報科学技術フォーラム講演論文集 (2004).
6) Nakamura, T. and Komoda, N.: Pre-allocation Size Adjusting Methods Depending on Growing File Size, *Proc. SNAPI 2008*, pp.19–25 (2008).
7) Sun™ QFS and Sun™ SAM-QFS 3.5 File Management Software 製品ガイド（オンライン）．入手先 http://jp.sun.com/products/guide/2001/qfs_jtf.pdf（参照 2009-08-07）．
8) Koch, P.D.L.: Disk File Allocation Based on the Buddy System, *ACM Trans. Computer Systems*, Vol.5, No.4, pp.352–370 (1987).
9) Seltzer, M. and Stonebraker, M.: Read Optimized File System Designs, A Performance Evaluation, *Proc. IEEE 7th International Conference on Data Engineering*, pp.602–611 (1991).
10) Powell, M.L.: The DEMOS File System, *Proc. 6th ACM Symposium on Operating Systems Principles*, pp.33–42 (1977).
11) Linux®ファイルシステム（オンライン）．入手先 http://media.netapp.com/documents/tr-3274-ja.pdf （参照 2009-08-07）．
12) Bar, M.: *Linux File Systems*, Osborne/McGraw-Hill (2001).
13) 小松克行：ext2 ファイルシステム，*Linux magazine*, No.3, pp.153–160 (1999).
14) VERITAS File System 4.1 管理者ガイド Linux（オンライン）．入手先 http://ftp.support.veritas.com/pub/support/products/Foundation_Suite/280553.pdf（参照 2009-08-07）．
15) Bovet, D.P. and Cesati, M.: 詳解 Linux カーネル第 2 版，オライリー・ジャパン (2003).
16) Hitz, D., Lau, J. and Malcolm, M.: File System Design for an NFS File Server Appliance, NetApp TR3002 (online) (2005).
available from http://media.netapp.com/documents/wp_3002.pdf

17) 長原宏治，佐藤通敏，今井悟志，加藤久慶：ZFS 仮想化されたファイルシステムの徹底活用，アスキー・メディアワークス (2009).
18) Agrawal, N., Bolosky, W.J., Douceur, J.R. and Lorch, J.R.: A Five-Year Study of File-System Metadata, *ACM Trans. Storage*, Vol.3, No.3, Article 9 (2007).

(平成 21 年 4 月 28 日受付)
(平成 21 年 9 月 11 日採録)

中村　隆喜（正会員）

1973 年生．1996 年 3 月大阪大学工学部精密工学科卒業．1998 年 3 月同大学大学院工学研究科精密科学専攻博士前期課程修了．同年 4 月（株）日立製作所入社．中央研究所勤務．2005 年 10 月より，同社システム開発研究所勤務．ファイルストレージ，オンラインストレージ，オペレーティングシステムの研究開発に従事．

薦田　憲久

1950 年生．1974 年 3 月大阪大学大学院工学研究科電気工学専攻修士課程修了．同年（株）日立製作所入社．システム開発研究所勤務．1981〜1982 年 UCLA 留学．1991 年 4 月大阪大学工学部情報システム工学科助教授，1992 年 8 月同大学教授．2002 年 4 月より，同大学大学院情報科学研究科マルチメディア工学専攻教授．工学博士．情報システムの計画，評価，電子商取引システム等の研究に従事．電気学会 2000 年度進歩賞等を受賞．IEEE，電気学会の各会員．