ファイルシステムのアクセス性能改善に関する研究

提出先　　大阪大学大学院情報科学研究科

提出年月　　2011 年 7 月

中 村　隆 喜

研究業績

A. 学術論文誌論文

[1] 中村隆喜, 薦田憲久 : “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”, 情報処理学会論文誌, Vol.50, No.11, pp.2690–2698 (2009).

[2] 中村隆喜, 亀井仁志, 山本彰, 薦田憲久 : “アクセス制御統合による性能低下を改善する階層型アクセス制御方式”, 情報処理学会論文誌, Vol.51, No.11, pp.2066–2080 (2010).

[3] 中村隆喜, 亀井仁志, 山本彰, 薦田憲久 : “ファイルアクセススループットを改善する POSIX ACL-高機能 ACL 併用方式”, 情報処理学会論文誌, Vol.52, No.6, pp.1939–1950 (2011).

B. 国際会議

[1] Ono, T., Tsukamoto, S., Inagaki, K., Sugiyama, K., Hirose, K., and Nakamura, T.: “Development of Ab initio Molecular-Dynamics Simulation Program Based on Real-Space Finite-Difference Method”, in *Proceedings of the 9th International Conference on Production Engineering (Precision Science and Technology for Perfect Surfaces)*, JSPE Publication Series, No.3, pp.1037–1042 (1999).

[2] Nakamura, T. and Komoda, N.: “Pre-allocation Adjusting Methods Depending on Growing File Size”, in *Proceedings of the 5th International Workshop on Storage Network Architecture and Parallel I/Os (SNAPI),* pp.19–25 (2008).

C. 国内講演

[1] 広瀬喜久治，坂本正雄，中村隆喜，小野倫也：“GGA を用いた実空間差分電子状態計算 I”，日本物理学会講演概要集，Vol.52, No.1–3, p.716 (1997).

[2] 広瀬喜久治，坂本正雄，杉山和久，稲垣耕司，中村隆喜，小野倫也：“GGA を用いた実空間差分電子状態計算 II”，日本物理学会講演概要集，Vol.52, No.2–3, p.807 (1997).

[3] 中村隆喜，小野倫也，中東幸子，広瀬喜久治：“GGA を用いた実空間差分電子状態計算 III”，日本物理学会講演概要集，Vol.53, No.1–3, p.640 (1998).

[4] 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-”，FIT2004 情報科学技術フォーラム講演論文集，Vol.3, No.1, pp.119–120 (2004).

[5] 徳田晴介，中村隆喜，藤原真二：“RAID 内蔵型 NAS の差分スナップショット機能における差分データ格納ボリューム拡張”，電子情報通信学会技術研究報告 CPSY コンピュータシステム，Vol.104, No.537, pp.13–18 (2004).

[6] 亀井仁志，揚妻匡邦，浦野明裕，中村隆喜: “ファイルアクセスを契機としたファイルシステム管理情報変換方式”，情報処理学会全国大会講演論文集，Vol.70，No.1, pp.1.33–1.34 (2008).

[7] 根本潤，須藤敦之，中村隆喜：“高速差分抽出方式による非同期リモートバックアップの効率化”，情報処理学会全国大会講演論文集，Vol.71, No.1, pp.1.49–1.50 (2009).

D. その他

[1] Hirose, K., Sugiyama, K., Inagaki, K., Bessho, Y., Tutumi, K., and Nakamura, T.: “Development of Real-space Simulation program for First-principles Molecular-dynamics”, Activity Report 1995 Supercomputer Center Institute for Solid State Physics University of Tokyo, pp.23–24 (1996).

[2] Hirose, K., Sugiyama, K., Sakamoto, M., Inagaki, K., and Nakamura, N.: “Development of Real-space Simulation program for First-principles Molecular-dynamics”, Activity Report 1996 Supercomputer Center Institute for Solid State Physics University of Tokyo, pp.66–67 (1997).

[3] Hirose, K., Sugiyama, K., Sakamoto, M., Inagaki, K., Nakamura, T., Ono, T., and Nakahigashi, S.: “Basis-set-free Real-space Simulation for First-principles Molecular-dynamics”, Activity Report 1997 Supercomputer Center Institute for Solid State Physics University of Tokyo, pp.87–88 (1998).

[4] 横田治夫 監修：“戦略的創造研究推進事業 CREST 研究領域「情報社会を支える新しい高性能情報処理技術」研究課題「ディペンダブルで高性能な先進ストレージシステム」研究終了報告書” (2009) (3.5 節「関連技術動向調査(日立，ストレージ技術動向調査グループ)」担当).

内容梗概

本論文は，筆者が 2002 年から現在まで(株)日立製作所 中央研究所，システム開発研究所，横浜研究所において取り組み，2010 年から現在まで大阪大学大学院情報科学研究科マルチメディア工学専攻在学中に発展させたファイルシステムのアクセス性能改善に関する研究成果をまとめたものである．

近年，非構造化データの急激な増加にともない，非構造化データの格納対象の 1 つであるファイルシステムの重要性がますます高まっている．ファイルシステムの最も重要な指標の 1 つはファイルアクセス性能である．ファイルアクセス性能とは，たとえば，単位時間当たりのファイルリクエスト処理数であるファイルリクエストスループットや，単位時間当たりのデータアクセス量であるファイルデータスループットである．ファイルアクセス性能を向上させる研究は多くあるが，高機能 ACL によるアクセス制御を用いないなどの限られた条件でのみ，高いファイルアクセス性能を達成可能であることが多い．ここで高機能 ACL とは，Windows ACL，NFSv4 ACL に低機能 ACL を統合したものを指す．また，低機能 ACL はパーミッション，POSIX ACL を指す．ファイルシステムをより使いやすくするためには，上で述べたような限られた条件以外でもファイルアクセス性能が大きく低下しないことが重要となる．そこで，本論文ではファイルアクセス性能低下の改善に関する 3 つの課題を取り上げる．具体的にその 3 つの課題は，ファイルシステムのアクセス制御の高機能化による性能低下の改善，ファイルシステムに格納済みの低機能 ACL 付きファイルに対する高機能アクセス制御適用による性能低下の改善，多数のファイルが同時作成される際に発生するフラグメントによる性能低下の改善である．

このような背景のもと，本論文では上述の 3 つの課題それぞれに対応して，(1) 拡張属性アクセス処理回数を低減する階層型アクセス制御方式, (2) 複数の ACL 形式を選択的に使用する低機能 ACL-高機能 ACL 併用方式，(3) ファイル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．本論文は全 5 章から構成される．

第 1 章の序論では，ファイルシステムの概要とそのファイルアクセス性能の重要性を述べる．さらに本研究で取り上げる課題を述べ，関連研究を概観するとともに，本論文の目的と位置づけを明らかにする．

第 2 章では，拡張属性アクセス処理回数を低減する階層型アクセス制御方式について述

べる．まず，アクセス制御の高機能化時に発生するファイルリクエストスループット低下について説明する．次に，提案方式である階層型アクセス制御方式を説明する．次に，アクセス制御情報を基本属性領域に格納する方式について検討し，模擬実験に基づき最適な格納方式を評価する．最後に，提案方式により見込まれる性能改善効果を検討し，提案方式を実装したシステムにおいて性能改善効果を確認する．

第 3 章では，複数の ACL 形式を選択的に使用する低機能 ACL-高機能 ACL 併用方式について述べる．まず，ファイルシステムに格納済みの低機能 ACL 付きファイルに対して高機能アクセス制御を適用した場合に発生するファイルリクエストスループット低下について説明する．次に，提案方式である低機能 ACL-高機能 ACL 併用方式を説明する．さらに，提案方式を実装したシステムにおいて性能改善効果を確認する．

第 4 章では，ファイル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について述べる．まず，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．次に，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．さらに，提案方式を実装したシステムを用いた測定と模擬実験に基づき，その効果を確認する．

第 5 章では，結論として本研究で得られた成果を要約した後，今後に残された課題について述べる．

目次

第 1 章

序論

1.1 研究の背景

近年，非構造化データの急激な増加にともない，そのデータを格納するシステムの重要性がますます高まっている．ここで，非構造化データは，文章，画像，音声などのデータベースでの管理が困難なデータを指す．一般に非構造化データは，ファイルという管理単位でのデータアクセス手段を提供するファイルシステムに格納される．ファイルシステムには主に 2 つの利用形態がある．第 1 の利用形態は，ファイルの入出力を行うファイルシステムプログラムと，そのファイルに対して情報処理を行うアプリケーションとが同一計算機上で動作する形態である．第 2 の利用形態は，それらが異なる計算機で動作する形態である．第 2 の利用形態では，ファイルシステムプログラムが動作する計算機上でファイル共有サービスのサーバプログラムが動作し，アプリケーションが動作する計算機上でファイル共有サービスのクライアントプログラムが動作する．近年のネットワークの発展にともなって，現在はこの第 2 の利用形態が主流となりつつある．この第 2 の利用形態において，ファイル共有サービスのサーバプログラムが動作する計算機を，ファイルストレージ, ファイルサーバ, Network Attached Storage (NAS)等と呼ぶことがある[1]–[9]. 以降，本論文ではこの計算機をファイルストレージと呼ぶ．ファイルストレージは歴史的には 1980 年代後半から普及しはじめたと考えられる．1980 年代後半にはファイルストレージがサポートするネットワーク対応の主要なファイルシステムは UNIX 向けの Network File System (NFS)[10]–[14]しか存在しなかったが，1990 年には Windows 向けのファイルシステムである Common Internet File System (CIFS)[15][16]が普及するようになった.

図 1.1 にファイルシステムの構成と基本動作を示す．ファイルシステムプログラムの基盤機能は，アクセス制御機能，ブロック割り当て機能，デバイスアクセス機能の 3 つである．たとえば，ユーザアプリケーションがファイル参照リクエストを発行すると，ファイルシステムはアクセス制御機能において，そのリクエストを処理してよいかどうかを判断する．具体的には，その該当ファイルに関連付けられたアクセス制御情報に基づき判断す

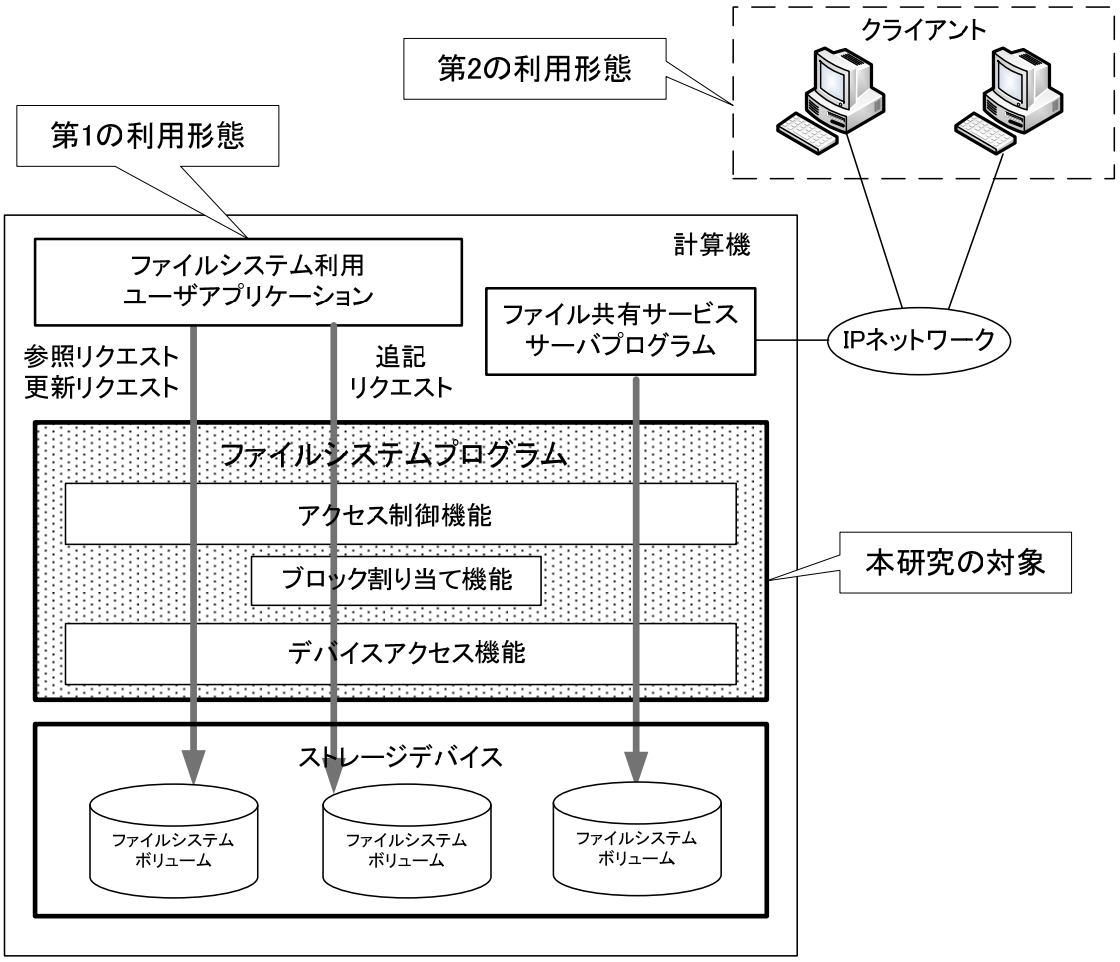

図 1.1 ファイルシステムの構成と基本動作

る．処理してよいと判断した場合，デバイスアクセス機能が，ファイルシステムボリュームより，指定されたファイルデータを取得し，ユーザアプリケーションに応答する．また，ユーザアプリケーションがファイル追記リクエストを発行すると，同様にアクセス制御機能を経由した後，ブロック割り当て機能により格納ブロック位置を決定し，デバイスアクセス機能により追記データをファイルシステムボリューム上の指定ブロック位置に格納する．

上記で述べたファイルシステムのアクセス制御情報の形式は主に 4 種類あり，情報量が少ない，つまり機能が低い順に，パーミッション，POSIX ACL[17]，Windows ACL[18]，NFSv4 ACL[19][20]となっている．ここで ACL とは Access Control List の略である．以降では，パーミッションと POSIX ACL をまとめて便宜上，低機能 ACL と呼ぶ．また，低機能 ACL に Windows ACL，NFSv4 ACL を統合したものを高機能 ACL と呼ぶ．初期のファイルシステムは低機能 ACL 形式のみに対応したアクセス制御が採用されていたが，

現在ではほとんどのファイルシステムが低機能 ACL 形式と高機能 ACL 形式に対応したアクセス制御を採用している．

ファイルシステムにおける最も重要な指標の 1 つはファイルアクセス性能である．ファイルアクセス性能とは，たとえば，単位時間当たりのファイルリクエスト処理数であるファイルリクエストスループットや，単位時間当たりのデータアクセス量であるファイルデータスループットである．ファイルリクエストスループットは，小サイズファイルが中心のデータセットやメタデータアクセスリクエストが中心のリクエスト分布の場合に適した指標である．またファイルデータスループットは，大サイズファイルが中心のデータセットやデータアクセスリクエストが中心のリクエスト分布の場合に適した指標である．これらの指標の値が大きければ大きいほど，多数のクライアント，多数のユーザからのリクエストを高速に処理でき，大量のデータを高速に処理できる．

ファイルアクセス性能を向上させる技術はこれまでに多くの取り組みがある[21]–[49]．高いファイルアクセス性能の達成は研究成果や製品としてアピールしやすいが，これらの性能は限られた条件でのみ達成可能であることが多い．たとえば，高いファイルアクセス性能を達成するための測定では，高機能 ACL 形式に対応したアクセス制御は用いないのが一般的である．ファイルシステムをより使いやすく信頼性の高いシステムとするためには，そのような限られた条件以外でもファイルアクセス性能が大きく低下しないということが重要となる．

このような重要性にもかかわらず，ファイルアクセス性能を低下させないという観点での検討の取り組み[50][51]は少ない．ファイルアクセス性能が低下するケースは大きく 3 つある．第 1 のケースは，ファイルシステムに対して何らかの高機能化[52]–[60]を行った場合である．第 2 のケースは，ファイルシステムに格納済みの低機能対応ファイルに対してその高機能を適用した場合である．第 3 のケースは，過酷な状況や環境でファイルシステムを運用している場合である．本論文ではこれら 3 つのケースに対して，それぞれ重用度の高い課題を取り上げる．

まず，前述したようにアクセス制御機能はファイルシステムの基盤機能であることと，同機能は近年 Windows 系 Operating System (OS)だけでなく UNIX 系 OS にも高機能化の動向があることから，本論文で取り上げる第 1 の課題を，アクセス制御を高機能化することによるファイルリクエストスループットの低下とする．本課題において，ファイルリクエストスループットを性能指標として用いる理由は，アクセス制御処理は小サイズファイルが中心のデータセットやメタデータアクセスリクエストが中心のリクエスト分布の場合に高頻度で実行されるためである．つまり，そのようなデータセットやリクエスト分布

では，アクセス制御の高機能化による性能低下の影響が大きい．アクセス制御を高機能化した場合，アクセス対象のファイルに付与されるアクセス制御情報の量が増える．ファイルは一般に基本属性領域，拡張属性領域，データ領域の 3 種類の領域に分割されてディスク上に格納されており，アクセス制御情報は拡張属性領域に格納されることが多い．高機能化により拡張属性領域へのディスクアクセス回数が増えることがあり，性能が低下することがある．

次に，第 1 の課題と同様の理由で，本論文で取り上げる第 2 の課題を，低機能 ACL が付与されたファイルに高機能アクセス制御を適用することによるファイルリクエストスループットの低下の改善とする．本課題において，ファイルリクエストスループットを性能指標として用いる理由についても，第 1 の課題の理由と同様である．低機能 ACL が付与されたファイルに高機能アクセス制御を適用する場合，対象の全ファイルを新規に用意した高機能 ACL 対応ファイルシステムに移行することが一般的である．この場合，移行によりファイルシステム運用が長時間停止するだけでなく，適用前に比べてアクセス制御情報の量が増えるため，第 1 の課題と同様の理由で性能が低下することがある．

最後に，近年情報の高付加価値化や高精度化にともないファイルサイズが増大し，フラグメント発生の潜在ポテンシャルが上がりつつあることから，本論文で取り上げる第 3 の課題を，フラグメントによるファイルデータスループットの低下の改善とする．本課題において，ファイルデータスループットを性能指標として用いる理由は，フラグメントは大サイズファイルが中心のデータセットやデータアクセスリクエストが中心のリクエスト分布の場合に発生しやすいためである．つまり，そのようなデータセットやリクエスト分布では，フラグメントによる性能低下の影響が大きい．ファイルに関するフラグメント防止技術は主にブロック割り当て機能に適用されているが，非同期書き込みにのみ適用可能な技術が多い．ここで非同期書き込みとは，更新・追記リクエストのデータをファイルストレージの揮発性メモリに格納した後で，ファイルシステムボリュームに格納する前に，格納完了をユーザアプリケーションやクライアントに応答するファイルシステムプログラムの動作である．一方，データをファイルシステムボリュームに格納した後に格納完了を応答する同期書き込みに対して有効なフラグメント防止技術は，ここ 10 年大きな進歩が見られない．具体的には，同期書き込みを用いて様々なサイズのファイルを同時に多数作成するケースで，副作用が少ない方法でフラグメントを防止することが難しかった．ファイルのフラグメントが発生した場合，ファイルの格納位置が断片化するため，アクセスパターンがランダム化することにより，性能が低下する．

以上をまとめると，本論文では，次の 3 点の課題に取り組む．第 1 の課題は，アクセス

制御機能を高機能化した場合のファイルリクエストスループットの低下を改善することである．第 2 の課題は，低機能 ACL が付与されたファイルに対して，高機能アクセス制御を適用した場合のファイルリクエストスループットの低下を改善することである．第 3 の課題は，同期書き込みを用いて様々なサイズの多数ファイルを同時に作成するケースで，副作用が少ない方法でフラグメントを防止し，ファイルデータスループット低下を改善することである．

1.2 関連研究

1.2.1 アクセス制御高機能化によるファイルリクエストスループット低下改善

アクセス制御の高機能化は，多数のストレージベンダーや OS ベンダーによりなされている[61]–[68]．この中で，高機能化にともなうファイルリクエストスループット低下の改善についてアプローチしている方式は，2 点挙げられる．

1 点目は，Sun Solaris 10 や Windows Storage Server で採用されている格納領域変更方式[69][70]である．格納領域変更方式は，各ファイルに付与される ACL のエントリ数もしくは ACL のサイズが一定値以下の場合に限り，ファイルの基本属性と呼ばれる領域にアクセス制御情報を格納する．一定値を超えた場合は，従来どおり拡張属性に格納する．これにより，ACL のサイズが小さい場合やエントリ数が少ない場合の，ファイルリクエストスループットを改善する．ただし，格納領域変更方式では ACL のサイズが大きい場合やエントリ数が多い場合の性能は改善できない．また，基本属性領域は通常ほとんど空き領域はないため，本方式実現のためには基本属性領域のサイズを増やしたディスクレイアウトに変更する必要がある．しかし，ディスクレイアウトの大幅変更はソフトウェア保守性や既存データの継続利用の観点から望ましくない．

2 点目は，NEC 社 SC-LX で採用されている ACL 専用キャッシュ方式[71]である．ACL 専用キャッシュ方式は，通常 OS で用意しているページキャッシュに加えて，アクセス制御情報専用のキャッシュをファイルサーバのメモリ上に持つ方式である．これにより，ページキャッシュのみのシステムよりも，アクセス制御情報がメモリ上に載っている時間，つまりライフタイムを長期化することができる．したがって，アクセス制御情報のキャッシュヒット率を向上することができる．ACL 専用キャッシュ方式は同一ファイルに繰り返しアクセスされる場合には有効である．ただし，同一ファイルに対するアクセス回数が非常に少ない場合，アクセス間隔が非常に長い場合などは，ACL 専用キャッシュでもキャッシュミスが発生することがあり，その場合は効果が得られない．

つまり，基本属性領域のサイズを増やすようなディスクレイアウト変更の必要がなく，キャッシュヒットが期待できない場合での，ファイルリクエストスループットを改善する方式はこれまで提案されていない．

1.2.2　低機能 ACL 付与ファイルに対する高機能アクセス制御適用によるファイルリクエストスループット低下改善

低機能 ACL が付与されたファイルに対して高機能アクセス制御を適用する場合の従来方式としては，ファイルコピーによるファイルシステム移行が挙げられる．具体的には，運用している低機能 ACL 形式対応ファイルシステムから，新規作成した高機能 ACL 形式対応ファイルシステムに，全データをコピーする．このコピーによる移行方式は，移行先の新規ファイルシステムの容量が必要で，ファイルシステム運用を長時間停止する必要があることから，ファイルシステム利用者，ファイルシステム管理者に大きな負担となっている．さらに移行完了後は，前節と同様の理由でファイルリクエストスループットも低下するが，これに対する取り組みは，前節で示したアプローチ以外は見られない．

また，上記の新規ファイルシステムの容量が必要という負担を軽減するため，Windows においては，ボリューム，ファイルシステムそのものを変換する手段を提供している．具体的には，File Allocation Tables (FAT)16 から FAT32 への変換のためのドライブコンバータ[72]，FAT から New Technology File System (NTFS)への変換のための convert コマンド[73]がある．しかしこれらの方式も，ファイルシステム運用を長時間停止する必要がある点は変わらず，ファイルリクエストスループットの低下も改善しない．

またアクセス制御機能そのものではないが，ファイル圧縮処理やファイル暗号化処理でファイルの形式を高速に変換するという研究が行われている．ファイル圧縮に関する論文[74][75]では，Solid State Drive (SSD)の適用，参照系リクエストの処理を更新系リクエストよりも優先する，書き込みのアクセスパターンがなるべくシーケンシャルになるようなディスクレイアウトを採用するなどのファイルリクエストスループットを改善する工夫が提案されている．ファイル暗号化に関する論文[76]では，スタッカブルファイシステムを利用し，暗号化処理をカーネルに実装することで，ユーザ空間で暗号化処理を実装した既存暗号化ファイルシステムに対してファイルリクエストスループット改善を実現している．これらの研究の一部は考え方を参考にできるが，本質的な部分はアクセス制御情報の変換への応用は難しい．

以上をまとめると，低機能 ACL が付与されたファイルに対して高機能アクセス制御を適用する場合に，ファイルシステム運用を長時間停止することなく，ファイルリクエストスループットを考慮した研究はこれまでなされていない．

1.2.3 ファイルフラグメント防止

ファイルフラグメントを防止するための研究は古くから行われている．この中でも最も有効なファイルフラグメント防止方式は，遅延アロケーション方式[77]である．遅延アロケーション方式は，ファイルシステムが非同期書き込みリクエストを受信した際に，データをメモリ上に書き込んで，ディスク上の格納位置を確定させずに，リクエストを完了する方式である．ディスク上の格納位置が確定していないメモリ上のデータは，その後の定期的なフラッシュ処理で,メモリからディスクに書き込まれる際に,格納位置が確定する．フラッシュ処理する頻度が適切であれば，複数の非同期書き込みリクエストのデータがまとめて処理できるため，たとえば同一ファイルに対する連続データをディスク上の連続位置に格納することができる．これによりファイルフラグメントが防止できる．しかしこの遅延アロケーション方式は，同期書き込みリクエストには対応できないという適用範囲の限界がある．

同期書き込みにおいても有効なファイルフラグメント防止方式としては，プリアロケーション方式[78]がある．プリアロケーション方式は，将来の書き込みに備えて，あらかじめ格納位置を予約しておく方式である．プリアロケーション方式は限られた条件においては有効な方式であるが，条件によっては管理が煩雑となることや，空き領域が有効に使えないという副作用が発生することがある．

つまり，高信頼サーバなどでデータの消失を極小化するために用いる同期書き込みにおいて，煩雑な管理が不要で副作用の少ないファイルフラグメント防止方式はこれまで提案されていない．

1.3 研究の方針

前節までに述べた課題をふまえ，それぞれの課題に対する本研究のアプローチを図 1.2 に示す．以下，各章節で本研究の方針について説明する．

(1) アクセス制御の高機能化にともなうファイルリクエストスループット低下改善

アクセス制御を高機能化する場合の，アクセス制御処理の処理時間増加に起因するファイルリクエストスループット低下を改善する階層型アクセス制御方式を提案する．本研究では特に Windows ACL を Linux[79]などの UNIX 系 OS に実装し，高機能化する場合のファイルリクエストスループットの低下を取り扱う．提案方式は，ファイルの所有者，もしくはそのファイルへ高頻度のアクセスが予想されるユーザやグループのアクセス制御情報の一部をファイルの基本属性に格納することにより，ほとんどのファイルリクエストを拡張属性にアクセスすることなく，実行可否の判定を可能とする．限られた基本属性領域

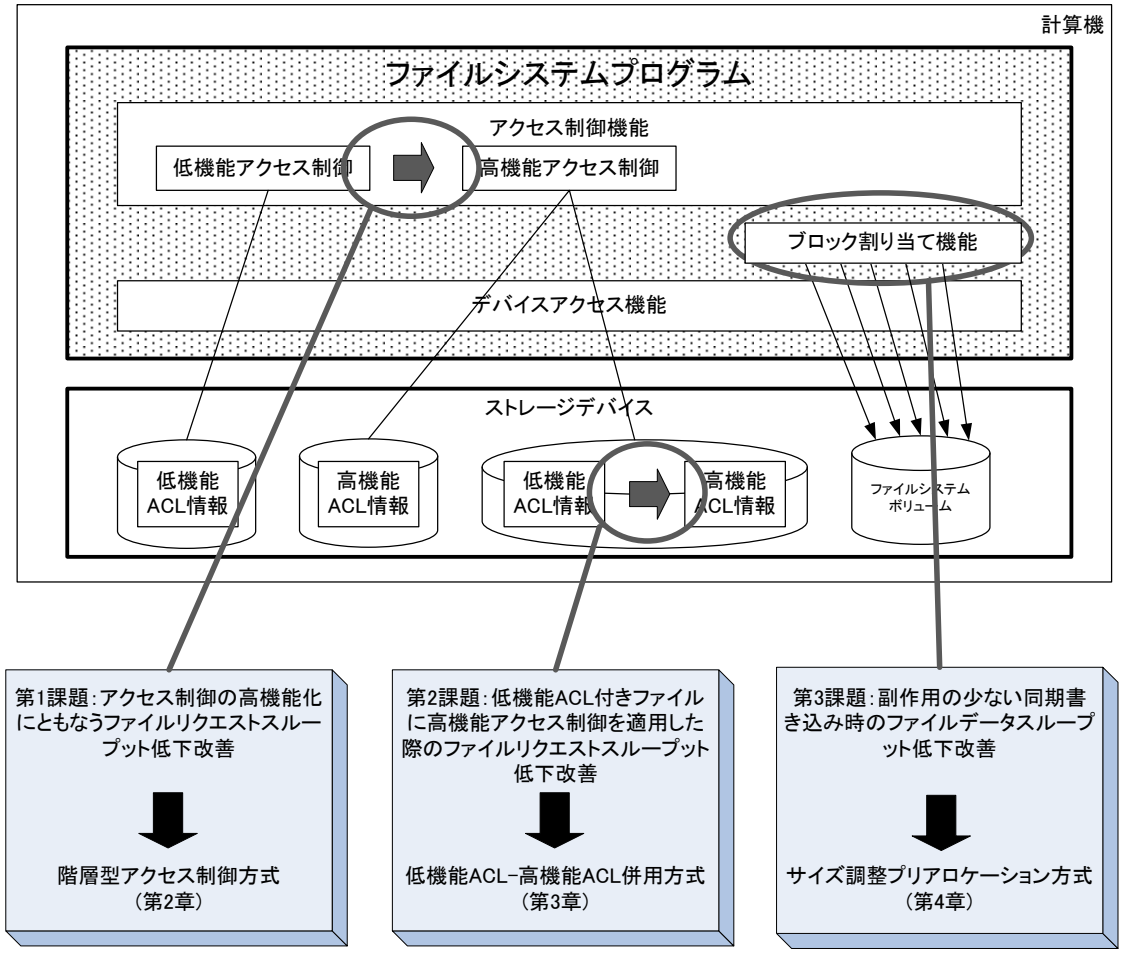

図 1.2 本研究のアプローチ

にどのような情報を格納するのがよいのかを検討し，評価する．

(2) 低機能 ACL 付きファイルに高機能アクセス制御を適用した際のファイルリクエストスループット低下改善

低機能 ACL が付与されたファイルに高機能アクセス制御を適用する場合の，長時間の運用停止とファイルリクエストスループットの低下とを改善する低機能 ACL-高機能 ACL 併用方式を提案する．本研究では特に低機能 ACL の 1 つである POSIX ACL で運用していたファイルシステムに，高機能 ACL の 1 つである Windows ACL を適用する高機能化を行った場合を取り扱う．提案方式は，ファイルシステムの基本・拡張属性に対する参照更新処理の処理時間の違いに着目し，アクセス制御情報のサイズに応じて，低機能 ACL の変換後結果である高機能 ACL を書き込む場合と，変換後結果は書き込まずアクセス制御処理時に毎回低機能 ACL 情報から高機能 ACL に変換する場合を使い分ける．どのような書き込みの使い分けがよいかを複数の条件を比較し，評価する．

(3) 副作用の少ない同期書き込み時のファイルデータスループット低下改善

複数ファイルを同時に同期書き込みで作成する場合の，ファイルフラグメントの発生を防止しファイルデータスループットの低下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．提案方式は，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．これにより従来方式の副作用であった，空き領域の利用効率低下を抑えつつ，ファイルフラグメントの発生を防止する．提案方式を，一般的な Personal Computer (PC)のファイルサイズ分布に基づくワークロードに適用し，ファイルフラグメントの発生防止と，ファイルデータスループットの改善に効果があることを確認する．

1.4 本論文の構成

本論文では，第 2 章以降を以下のように構成する．

第 2 章では，文献[80]に基づき，アクセス制御を高機能化する場合の，アクセス制御処理の処理時間増加に起因するファイルリクエストスループット低下を改善する階層型アクセス制御方式について述べる．まず，アクセス制御の高機能化時の課題である，ファイルリクエストスループット低下の改善について説明する．次に，提案方式である階層型アクセス制御方式を説明する．次に，アクセス制御情報を基本属性領域に格納する方式について検討し，模擬実験に基づき最適な格納方式を評価する．最後に，提案方式により見込まれる性能改善効果を検討し，提案方式を実装したシステムにおいて性能改善効果を確認する．

第 3 章では，文献[81][82]に基づき，低機能 ACL が付与されたファイルに高機能アクセス制御を適用する場合の，長時間の運用停止とファイルリクエストスループットの低下を改善する低機能 ACL-高機能 ACL 併用方式について述べる．まず，文献[81]で提案したオンアクセス ACL 変換を用いたファイルシステム移行方式，その課題であるファイルリクエストスループットの低下の改善について説明する．次に，提案方式である低機能 ACL-高機能 ACL 併用方式を説明する．さらに，提案方式を実装したシステムにおいて性能改善効果を確認する．

第 4 章では，文献[83]–[85]に基づき，複数ファイルを同時に同期書き込みで作成する場合の，ファイルフラグメントの発生を防止しファイルデータスループットの低下を改善するサイズ調整プリアロケーション方式について述べる．まず，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．次に，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．さらに，提案方式を実

装したシステムを用いた測定と模擬実験に基づき，その効果を確認する．

第 5 章では，結論として本研究で得られた成果を要約し，今後の課題を述べる．

第 2 章

拡張属性アクセス処理回数を低減する階層型アクセス制御方式

2.1 緒言

本章では，アクセス制御を高機能化する場合の，アクセス制御処理の処理時間増加に起因するファイルリクエストスループット低下を改善する階層型アクセス制御方式について述べる．

ファイルシステムにおける代表的なアクセス制御の形式は，パーミッション，POSIX ACL，Windows ACL，NFSv4 ACL の 4 種類である．このうち NFS 専用のファイルシステムで最もよく用いられるアクセス制御の形式はパーミッションであり，そのアクセス制御情報は固定サイズである．また CIFS 専用のファイルシステムで最もよく用いられるアクセス制御の形式は Windows ACL であり，そのアクセス制御情報は可変サイズである．Windows ACL は NTFS ACL，CIFS ACL とも呼ばれる．

パーミッションと Windows ACL を比較すると，基本的には Windows ACL のほうがより情報量が多く高機能であるが，パーミッションに対して上位互換の仕様となっているわけではない．パーミッションと Windows ACL のアクセス制御情報と制御プログラムを統合する高機能化は, Linux Samba[61]–[65], NetApp FAS[66], Microsoft Windows Storage Server[67]，Sun Solaris 10[68]などで実現されている．

セキュリティを重視して両アクセス制御形式を統合する高機能化を行った場合，パーミッションを用いる NFS 単体で運用していた場合に対して，ファイルリクエストスループット(以降の本章では，単に性能と呼ぶことがある)が低下することがあった．これはパーミッションベースの NFS のアクセス制御処理(以降便宜上，低機能アクセス制御処理と呼ぶ)から，より情報量が多い高機能アクセス制御処理に置き換わるためである．既存の高機能化方式は，この性能低下を考慮できていなかった．本章では，このファイルリクエストスループット低下を改善する方式について検討を行う．

Linux をはじめとする UNIX 系 OS のファイルは，基本属性，拡張属性，ユーザデータから構成される．基本属性の格納領域は固定サイズであり，拡張属性とユーザデータの格納領域は可変サイズである．パーミッションのアクセス制御情報は基本属性の 1 つであり，基本属性領域に格納されるのが一般的である．これに対し高機能アクセス制御情報は Windows ACL の仕様により可変サイズとなるため，拡張属性領域に格納せざるをえない．したがって高機能アクセス制御処理では，拡張属性領域への読み込みが発生し，またその読み込んだ情報を保持しておくためのメモリが必要となるため，処理性能が低下する．

そこで本章では，拡張属性領域に格納される高機能アクセス制御情報の一部情報を基本属性領域に格納し，その一部情報のみで先にアクセス制御処理を実行することにより，性能低下を改善する階層型アクセス制御方式を提案する．高機能アクセス制御情報はサイズが大きいため，そのすべてを基本属性に格納することは難しい．したがって，どのような情報を切り出して基本属性に格納するのがよいかを，各ファイルリクエストで使用するアクセス制御情報を整理することにより検討する．

本章の以降の構成は次のとおりである．2.2 節ではアクセス制御機能と形式の概要について説明する．2.3 節では，アクセス制御の高機能化時の課題である，ファイルリクエストスループット低下の改善について説明する．2.4 節では，提案方式である階層型アクセス制御方式を説明する．2.5 節では，基本属性領域に格納する情報に関する複数の格納方式について検討し，模擬実験に基づき最適な格納方式を評価する．2.6 節では，提案方式により見込まれる性能改善効果を検討し，提案方式を実装したシステムにおいて性能改善効果を確認する．最後に 2.7 節で，本章のまとめを述べる．

2.2 アクセス制御機能と形式の概要

ファイルシステムにおけるアクセス制御機能の処理概要を図 2.1 に示す．アクセス制御機能はファイルアクセスリクエストを発行したユーザが，そのリクエストで指定したファイルに対してアクセスする権利を有しているかどうかを判断する機能である．その判断のための情報は，そのファイルに関連付けられたアクセス制御情報に記載されている．アクセスする権利がある場合は，ファイルシステムでそのリクエストが処理され，その処理された結果がユーザアプリケーションに応答される．たとえばリードリクエストならば読み込んだデータが応答される．アクセスする権利がない場合は，リクエストは実行されず，アクセス権エラーとしてユーザアプリケーションにエラーが通知される．

アクセス制御の形式には前述したように主に 4 種類ある．NFS サービス用データと CIFS サービス用データを別々のファイルシステムボリュームに格納して運用しているケ

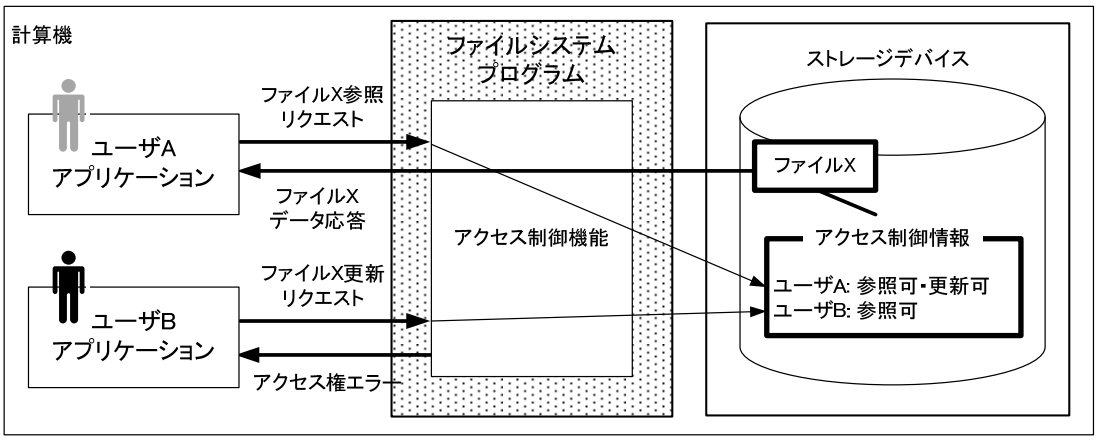

図 2.1 アクセス制御機能の処理概要

表 2.1 パーミッションと Windows ACL の違い

分類	#	項目	パーミッション (NFS)	Windows ACL (CIFS)
基本	1	各ACEマスク種類	3種類 (rwx)	14種類
	2	ACL長	3エントリ固定 (owner, group, other)	可変(最大64KByte, 典型的ACEサイズ36Byteでは最大1820エントリ)
	3	許可・拒否種別	許可のみ	許可もしくは拒否
	4	ACLの評価順序	owner, group, other	格納順
	5	ユーザID	UID, GID (4Byte)	SID (28Byte)
	6	初期設定ACL	umask	親フォルダの継承ACL
	7	ACL格納領域	一般に基本属性領域	一般に拡張属性領域
特別なACE	8	owner ACE	あり	なし
	9	group ACE	あり	なし
	10	other ACE	あり(所有者と所有グループ以外の全てのユーザ)	なし(ただし所有者，所有グループを含む全てのユーザが対象のEveryone ACEはあり)
特権	11	プロトコル特権	実行ビットオン時にデータ参照可能 属性参照可能	特になし
	12	所有者特権	データ参照変更可能 属性変更可能	ACL変更可能
オペレーションの挙動	13	所有者変更	一般ユーザ不可	一般ユーザも可
	14	所有グループ変更	所有者の非所属グループへの変更不可	任意のグループに変更可
	15	タイムスタンプ更新	rootと所有者以外は任意の時間へ変更不可	全ユーザ任意の時間に可能

ースでは，NFS 向けにはパーミッション，CIFS 向けには Windows ACL のアクセス制御が用いられるのが一般的である．両形式の違いを表 2.1 に示す．

#1 はアクセス制御をどの程度細かく行うかの違いである．パーミッションでは参照，書き込み，実行の 3 種類のみだが，Windows ACL ではこれに加えて，削除，属性参照，属性書き込みなどさらに細分化され 14 種類となっている．

#2 は ACL のエントリ数の違いである．パーミッションでは所有者，所有グループ，そ

の他の 3 エントリ固定となっているが，Windows ACL は最大 64KB までの可変である．ACL エントリのサイズは可変だが, 典型的な 36Byte の場合は最大 1820 エントリとなる．

#3 は ACL が許可・拒否エントリをサポートしているかの違いである．パーミッションでは許可エントリのみをサポートするが，Windows ACL では許可エントリ，拒否エントリの両方をサポートする．

#4 は ACL の各エントリをどのような順序で評価するかの違いである．パーミッションは所有者エントリ，所有グループエントリ，その他エントリの順で評価する．これに対し Windows ACL ではエントリ格納順で評価する．

#5 は ACL エントリで使用するユーザ識別子に何を用いているかの違いである．パーミッションでは 4Byte の User ID (UID)もしくは Group ID (GID)だが，Windows ACL では 28Byte の Security ID (SID)を用いている．

#6 はファイルやディレクトリを作成した直後に自動的に設定される ACL の違いである．パーミッションでは umask に基づき設定されるが, Windows ACL では親フォルダに付与されている継承 ACL に基づき設定される．

#7 は ACL が格納される領域の違いである．本項目は内部仕様もしくは実装に関するものであり，外部仕様である他の項目とは同列には扱えないが，本章の骨子に関係するため記載する．パーミッションは ACL 長が短い固定サイズであることから，基本属性領域であるインコアメタデータに格納されるのが一般的である．これに対し Windows ACL は ACL エントリ長が可変であることから, 拡張属性領域である拡張メタデータに格納されるのが一般的である．

#8～#10 は特別な Access Control Entry (ACE)に関する項目である．パーミッションでは前述したように，所有者エントリ，所有グループエントリ，その他エントリの 3 種類の特別なエントリを持つが，Windows ACL ではそれに対応するエントリはない．ただし Windows ACL では，その他エントリに類似する Everyone エントリを持つことがある．パーミッションのその他エントリは所有者，所有グループ以外の全ユーザのアクセス権を規定するが，Windows ACL の Everyone エントリは所有者，所有グループも含む全ユーザのアクセス権を規定する．

#11 はプロトコル特権に関する違いである．NFS は 2 つのプロトコル特権を持つ．1 つ目は参照に関する特権であり, NFS ではパーミッションの参照ビット(r)が立っていない場合でも，パーミッションの実行ビット(x)が立っていてれば参照が許可される[13]．2 つ目は属性参照に関する特権であり，NFS ではパーミッションの rwx の権限をなんら持たないユーザであっても属性の参照が可能である．これに対し CIFS ではプロトコルに関する

特権は特にない.

#12 は所有者の持つ特権に関する違いである．NFS は所有者であれば，パーミッションの rwx の権限をなんら持たなくても,データの参照変更および属性変更が可能である[13]. これに対し Windows ACL では所有者は ACL の変更のみが特権となっている.

#13 は所有者変更の挙動に関する違いである. パーミッションでは root ユーザのみが所有者変更可能であるのに対し，Windows ACL では一般ユーザであっても所有者変更権を持つ他者所有のファイルを，自身所有のファイルに変更することが可能である.

#14 は所有グループ変更の挙動に関する違いである．パーミッションでは所有者の所属グループのみに変更可能であるが，Windows ACL ではそのような制限はなく，任意のグループに変更できる.

#15 はタイムスタンプ更新の挙動に関する違いである. パーミッションでは root と所有者以外は任意の時間にタイムスタンプを更新することはできない．これに対し Windows ACL では適切な権限があればどのユーザでも任意の時間に更新可能である.

本節で述べた比較の一部は，文献[86][87]にも詳しい.

2.3　アクセス制御高機能化によるファイルリクエストスループット低下

前述したパーミッション，Windows ACL の違いのうち，ファイルリクエストスループットに関する項目は前節表 2.1 に示した#1，#2，#3，#5，#7 である.

両アクセス制御を統合する高機能化を行う場合，#1，#2，#3，#5 は情報がより豊富な形式つまりこの場合 Windows ACL の形式にあわせれば運用上の問題が少ない．Windows ACL 形式にあわせた場合のアクセス制御情報の格納位置がどのように変化するかを図 2.2 に示す.

高機能化前のパーミッションベースの NFS 単独システムにおいて，ファイルは固定長の基本属性領域と可変長のユーザデータ領域から構成される．アクセス制御処理時に参照する情報は,ファイルの所有者識別子である UID と,所有グループ識別子である GID と，所有者，所有グループ，その他ユーザのアクセス権を規定するパーミッションであり，これらはすべて基本属性に格納される.

これに対し，高機能化後のシステムにおいて，ファイルは基本属性領域，ユーザデータ領域，に加えて可変長の拡張属性領域から構成される．これは Windows ACL のアクセス制御情報が可変であるため，高機能化後のアクセス制御情報も可変となるためである．高機能化後のアクセス制御情報はファイルの所有者識別子である所有者 SID，所有グループ

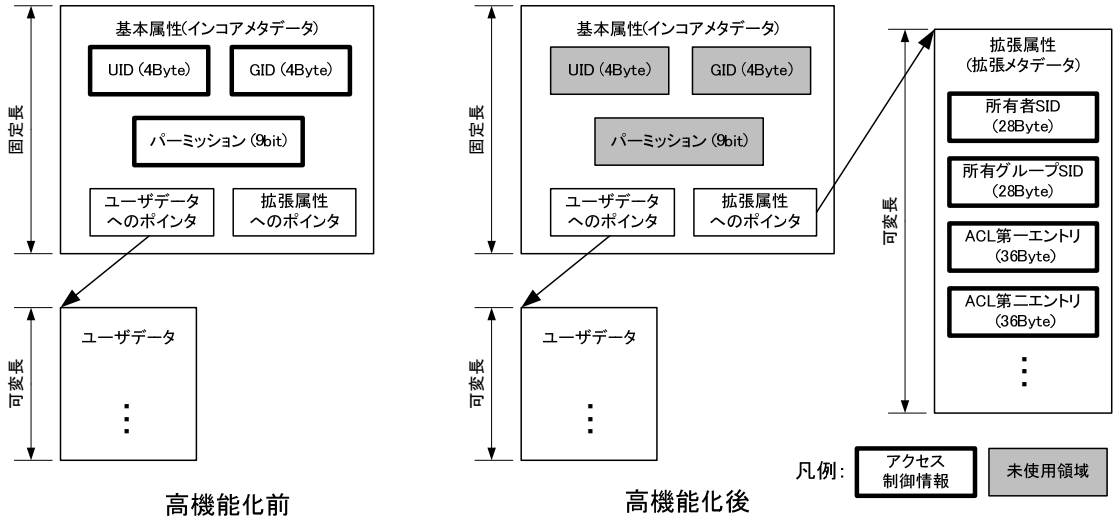

図 2.2 高機能化前後のアクセス制御情報の格納位置の変化

の識別子である所有グループ SID，各 SID に対するアクセス権を規定する ACL エントリからなる．ここで，ACL エントリ数は可変である．

高機能化前は基本属性の情報だけでアクセス制御処理が実行可能であるが，Windows ACL 形式にあわせて統合する高機能化を行った場合，たとえ NFS であっても拡張属性から高機能アクセス制御情報を読み込んだ後，アクセス制御処理をする必要がある．これにより，ファイルリクエストで実行されるアクセス制御処理の時間が増加してしまうことがある．またパーミッション情報が，拡張属性上の高機能アクセス制御情報に統合されてしまうため，パーミッション参照更新リクエストの処理時間も増加してしまう．これらの処理時間の増加にともない, 単位時間当たりのファイルリクエストスループットが低下する．

次節以降では，主に前者のアクセス制御処理の処理時間増加を改善する方式について検討する．ただし，後者のパーミッション参照更新リクエストの処理時間増加の改善も提案方式に関係するため，あわせて検討する．

2.4 階層型アクセス制御方式

ファイルへのアクセスパターンを考えた際，あるファイルに対してアクセスするユーザは偏りがあり，またそのファイルに対して発行されるリクエストも偏りがあることが予想される[88].

たとえば，アクセスユーザに関しては，個人オンラインドライブとしての使用目的であれば所有者のアクセス頻度が最も高く，共有ドライブとしての使用目的であれば共有範囲の主グループに属するユーザのアクセス頻度が高いことが予想される．ファイルリクエス

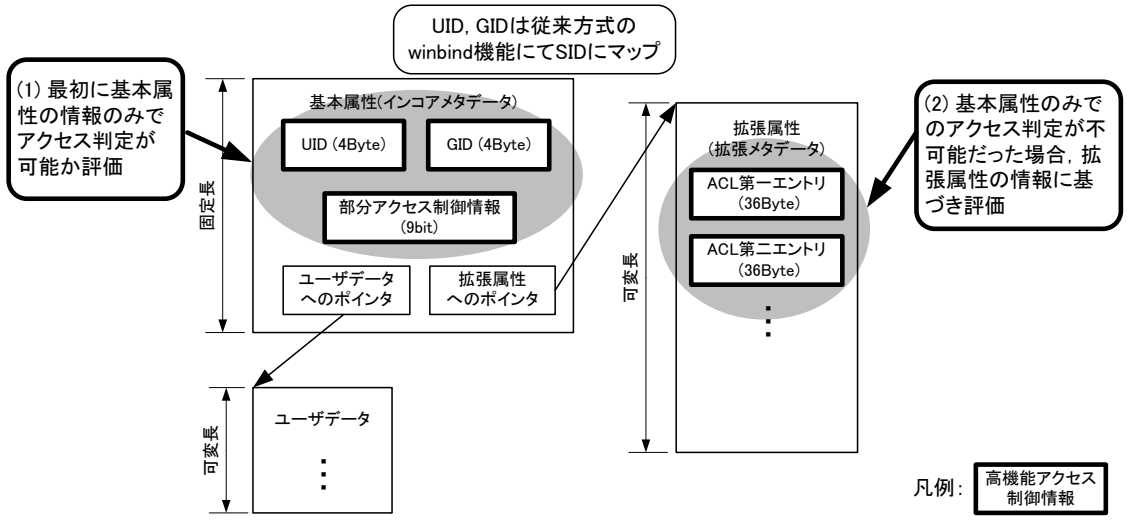

図 2.3 提案方式のアクセス制御情報の格納位置

トに関しては，ディレクトリであればルックアップ要求が高く，ファイルであればデータ参照要求が高いことが予想される．

そこで本章では，アクセスの確率の高いユーザ，また実行される確率の高いリクエストに関するアクセス制御情報を基本属性に格納し，アクセス制御処理を基本属性の情報のみで先に実行する階層型アクセス制御方式を提案する．

図 2.3 に提案方式のアクセス制御情報の格納位置と処理概要を示す．基本属性のみでアクセス制御処理を可能とするため，所有者，所有グループに関しては Windows ACL 仕様の SID は用いず，UID，GID を用いる．UID，GID と SID のマッピングは従来技術である winbind[89]を利用する．また，所有者や所有グループに関するアクセス制御情報の一部を基本属性に格納するために，高機能化前にパーミッション領域として使用していた 9bit の領域を，部分アクセス制御情報格納領域として利用する．何の情報をこの部分アクセス制御情報格納領域に格納するかによって，性能改善効果に差が出る．本方式により，一部のリクエストは基本属性の情報を参照するだけでアクセス可否が判断でき，アクセス制御高機能化による性能低下の改善が見込まれる．

またパーミッション参照更新リクエストの応答性能低下についても，本提案の部分アクセス制御情報の格納方式を工夫すれば改善が可能である．ただし，格納方式の組みあわせによっては，本性能低下改善のために同リクエストのユーザビリティを犠牲にしなければならないこともある．本性能低下の改善については，2.5.9 節で検討を行う．

本方式の適用によって生じるデメリットは次の 2 点である．第 1 に，アクセス制御情報の更新処理が多少複雑となる．ただし，更新処理の性能支配主要因はディスクアクセスで

あり，処理の複雑さ自体はほとんど性能に影響を及ぼさない．第 2 に，部分アクセス制御情報の領域には，基本的に従来格納されていた情報とは異なる情報が格納されるため，従来のファイルシステムプログラムとは互換性がなくなってしまう．ただし，次節で述べる格納方式を工夫すれば, 互換性を維持することも可能である. また互換性がない場合でも，ディスクレイアウト自体を変えるわけではないため, ファイルシステムの mount 処理が失敗するなどの深刻な問題は発生しない.

なお，提案方式は，既存の UNIX 系 OS で，パーミッションをネイティブアクセス制御方式とするファイルシステムを高機能化適用前とした場合を主な検討対象としている．また，本章ではパーミッションベースのファイルシステムを Windows ACL 対応とするための検討を中心としている．これに加えて，POSIX ACL や NFSv4 ACL などのパーミッションより豊富な情報を持つアクセス制御の形式に対応する場合にも考え方を参考にできる. また，NFSv4 ACL をすでにネイティブサポートしている ZFS[90]のような高機能アクセス制御対応済みファイルシステムにも考え方は適用できる．さらに，本章ではアクセス制御情報を階層化の対象としているが，これ以外のファイルの属性情報にも考え方を参考にできる.

提案方式が適用できるファイルシステムは，基本属性と拡張属性を別領域に格納するファイルシステムである．したがって，現在一般に利用されているすべてのファイルシステムに適用可能と考える.

2.5 部分アクセス制御情報の格納方式

本節では, 従来のパーミッション領域である 9bit にどのような部分アクセス制御情報を格納するのが効果的かを検討し, 比較評価する. 2.5.1 節で格納方式の概要を述べる. 2.5.2, 2.5.3 節で比較評価方法と比較する格納方式について述べる．2.5.4 節で各条件における比較評価結果を述べる. 2.5.5 節で格納方式がパーミッション参照更新リクエストのファイルリクエストスループット低下に対して与える影響について述べる. 2.5.6 節で比較評価結果をまとめる.

2.5.1 格納方式の概要

本小節では，部分アクセス制御情報領域の格納方式の概要を述べる．Windows ACL のアクセス権の種類は 14 種類であり，つまりアクセスマスクは 14bit であるため，仮に 1 ユーザもしくは 1 グループの情報に限定した場合にも，すべてを部分アクセス制御情報領域にはマップできない.

所有者からアクセスされる可能性が非常に高い場合は，所有者のアクセス制御情報の一

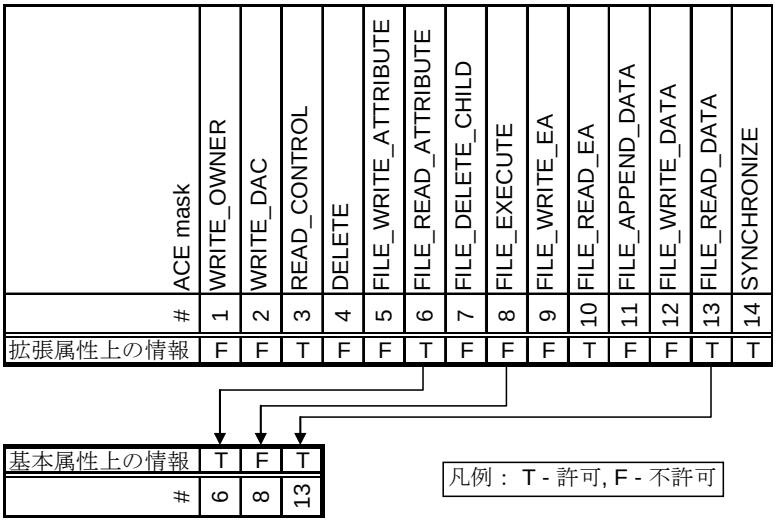

図 2.4 格納方式 U の概要

部を部分アクセス制御情報領域の 9bit すべてに割り当てる方式が考えられる．複数のユーザからアクセスされる可能性が高い場合には，9bit を 2 つの領域に分割して所有者，所有グループのアクセス制御情報を格納する方式が考えられる．さらに，所有者，所有グループ，その他ユーザに対してそれぞれ 3bit の部分アクセス制御情報領域を割り当てる方式も考えられる．

領域を分割して格納する場合，それぞれで異なる格納方式を用いてもかまわない．つまり，たとえば，所有者についてはある格納方式を用い，所有グループとその他ユーザについては別の格納方式を用いるなども可能である．複数の格納方式を併用する場合，どの領域にどの格納方式を用いるかの情報を，ファイルシステムプログラムもしくはファイルシステムメタデータに記録させる必要がある．

基本格納方式としては，以下の 3 種類が考えられる．

(1) 高頻度使用アクセスマスクビット格納方式(most frequent USED access mask bit storing style)：以降，格納方式 U

図 2.4 に格納方式 U の概要を示す．格納方式 U は 14 種類のアクセスマスクビットの中から高頻度で使用されるアクセスマスクビットの上位を基本属性の部分アクセス制御情報領域に格納する．

たとえば，ファイルの参照要求の頻度が高い場合は，Windows ACL の FILE_READ_DATA ビットを優先的に部分アクセス制御情報領域に格納する．本方式はファイルリクエストに偏りがあり，使用されるアクセスマスクビットにも偏りがある場合は有効である．

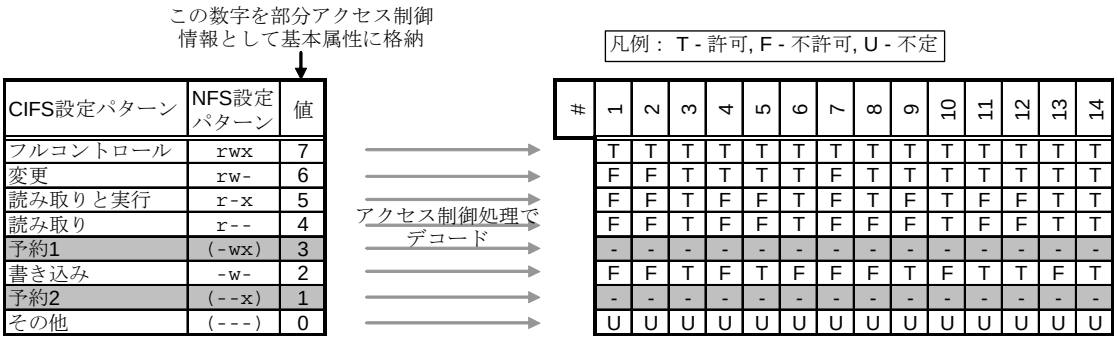

図 2.5 格納方式 S の概要

(2) 高頻度設定アクセスマスクパターン格納方式(most frequent SET access mask pattern storing style)：以降，格納方式 S

図 2.5 に格納方式 S の概要を示す．格納方式 S は高頻度で設定されるアクセスマスクパターン(代表的設定パターン)の情報を基本属性の部分アクセス制御情報領域に格納する．Windows ではアクセス許可の簡易設定パターンとして「フルコントロール」「変更」「読み取りと実行」「読み取り」「書き込み」の 5 種類がある．本格納方式では，たとえばこれら簡易設定パターンを代表的設定パターンとし，これらの設定パターンに合致している場合はその値を部分アクセス制御情報領域に格納する．上記簡易設定パターンのみを格納する場合は，5 種類 + それ以外を意味する 1 種類 = 合計 6 種類 $< 2^3$ であるため 3bit で表現可能である．アクセス制御処理時には部分アクセス制御情報の値より図右のアクセスマスクパターンにデコードしアクセス可否を判断する．本方式はアクセスマスク設定パターンに偏りがある場合は有効である．図の例では，低機能 ACL の設定パターンを格納対象の設定パターンとしている．この場合，方式適用前のファイルシステムプログラムとの互換性が維持できるという利点がある．しかし，図で示した設定パターン以外も格納対象とすることは可能である．その場合は，従来パーミッションとして使用していた情報の意味を変えることになるため，方式適用前のファイルシステムプログラムとの互換性は損なわれる．

代表的設定パターンの数よりも部分アクセス制御情報格納領域が大きい場合，余剰ビットを格納方式 U として用いる方法も考えられる．つまり，格納方式 S と格納方式 U を組みあわせて用いる方式である．

(3) アクセスマスクビット論理積格納方式(logical production [AND] of access mask bit storing style)：以降，格納方式 A

図 2.6 に格納方式 A の概要を示す．格納方式 A は複数のアクセスマスクビットの論理積

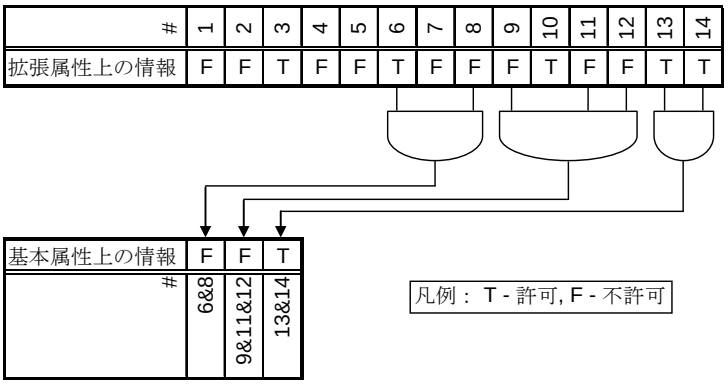

図 2.6 格納方式 A の概要

を基本属性の部分アクセス制御情報に格納する．たとえば，FILE_READ_ATTRIBUTE ビットと FILE_EXECUTE ビットの論理積を部分アクセス制御情報の 1bit に割り当てる．これにより，多くのアクセスマスクビット情報を部分アクセス制御情報領域に格納可能である．本方式は各アクセスマスクビットが真となる確率が高い場合に有効である．

本格納方式を用いる場合，以下のような工夫が重要となる．同時設定確率が高いアクセスマスクビット群を論理積対象とする．高使用確率のアクセスマスクビットは論理積対象とせずそのまま部分アクセス制御情報領域の 1 ビットを割り当てる．低使用確率のアクセスマスクビットは論理積対象から除外し，基本属性領域には格納しない．

その他の格納方式

以上述べた基本格納方式 U，S，A 以外にも，各システムコールやネットワークプロシージャの実行可否を各ビットにマップし格納する方式(格納方式 P)や，全アクセスマスクビットを格納する方式(格納方式 F)も考えられる．格納方式 P については，システムコール，ネットワークプロシージャの種類は数 10 以上あり，ユニークなものだけを抽出したとしても他の格納方式と比べて非効率であるため，以降の評価対象からは外した．また格納方式 F については，基本属性領域に十分な空き領域があれば最も優れた格納方式であることは間違いないが，すでに述べたとおり基本属性領域の空き領域は有限であり，以降では最も一般的な 9bit 以下で格納可能な方式を議論するため，今回の評価対象からは外した．

2.5.2 格納方式の前提

あるファイルアクセスパターンを仮定した場合，ファイルアクセスパターンは複数のファイルリクエストから構成される．それぞれのファイルリクエストではアクセス権チェックに用いられるアクセスマスクが規定されている．ファイルリクエストごとに規定されたアクセスマスクの許可・非許可の状態は，格納方式に応じて，部分アクセス制御情報のみ

表 2.2 SPECsfs2008 ファイルオペレーション分布

#	NFS Version 3 Operation	割合
1	LOOKUP	24%
2	READ	18%
3	WRITE	10%
4	GETATTR	26%
5	READLINK	1%
6	READDIR	1%
7	CREATE	1%
8	REMOVE	1%
9	FSSTAT	1%
10	SETATTR	4%
11	READDIRPLUS	2%
12	ACCESS	11%
13	COMMIT	N/A

で決定可能な場合と，決定不可能な場合がある．

ファイルアクセスパターンにおける各ファイルリクエストの実行割合が分かれば，部分アクセス制御情報のみで決定可能なファイルリクエストの割合が算出できる．部分アクセス制御情報のみで決定可能なファイルリクエストの種類は，格納方式ごとに異なるため，この部分アクセス制御情報のみで決定可能なファイルリクエストの割合が評価指標となる．

以降では，仮定するファイルリクエストの割合として，Standard Performance Evaluation Corporation (SPEC)が提供しているファイルストレージの代表的な性能ベンチマークソフトである SPECsfs 2008 を参考にする[91]. SPECsfs2008 は, 単位時間(1 秒)あたりのファイル共有サービス処理オペレーション数(Operations Per Second: OPS)，つまりファイルリクエストスループットを測定するベンチマークプログラムである．プログラムが発行する NFS version 3 のオペレーションの割合は表 2.2 のとおりとなっている．同プログラムは作成した全ファイルのうち約 3 割のファイルに，周期的ポアソン分布に基づく確率でファイルアクセスを行う．アクセスされるファイルの平均ファイルアクセス回数は約 4 である．

2.5.3　具体的な格納方式

2.5.1 節の基本格納方式に基づき，具体的には以下の格納方式の中から比較する．

［格納方式 Un］よく使用されるアクセスマスクビットの上位 nbit を格納

［格納方式 S3］よく設定されるアクセスマスクパターンを 3bit で表現

［格納方式 SUn］よく設定されるアクセスマスクパターンの 3bit に加え，よく使用されるアクセスマスクビットの上位 n−3bit を格納

［格納方式 A3］なるべく多くのアクセスマスクビットの論理積を 3bit にマップし格納

なお，これ以外にも $n \geq 4$ での格納方式 Sn や格納方式 An を評価することも考えられるが，本節では以下の理由で行わなかった．格納方式 Sn については，基本的に格納方式 S3 と同様の傾向となるが，S3 と Sn との定量評価結果は，想定環境でのセキュリティレベルに大きく依存する．その中でも一般的な部門ファイル共有のセキュリティレベルにおいては 3bit でも代表的なアクセスマスクパターンは十分カバーできるため，4bit 以上にしても効果は小さいと判断した．格納方式 An については，ビット数が大きくなるにつれ，格納方式 Un との違いがほとんどなくなる．

評価軸は，データセット全体に対する代表的アクセスマスク設定パターンの割合とした．この評価軸では格納方式 Un，S3，SUn は，一意に評価指標の算出が可能であるが，格納方式 A3 については仮定を設定しなければ算出不能である．そこで A3 の評価指標の算出にあたり以下の 2 つの仮定を設定した．

1 つ目の仮定は，代表的設定パターンの具体例とその割合である．これは著者らの実験環境での割合を参考にし，表 2.3 のように設定した．

2 つ目の仮定は，非代表的設定パターンの論理積が真となる確率である．これは代表的設定パターンの論理積が真となる確率の平均値の半分を期待値とした．また，論理積がすべて偽となる場合と，論理積がすべて真となる場合の最悪，最良ケースもあわせてグラフ

表 2.3 仮定した代表的アクセスマスクパターンとその割合

#	CIFS	割合	NFS	割合
1	フルコントロール	30%	rwx	40%
2	変更	30%	rw-	30%
3	読み取りと実行(フォルダの内容の一覧表示)	30%	r-x	20%
4	読み取り	5%	r--	10%
5	書き込み	5%	-w-	—

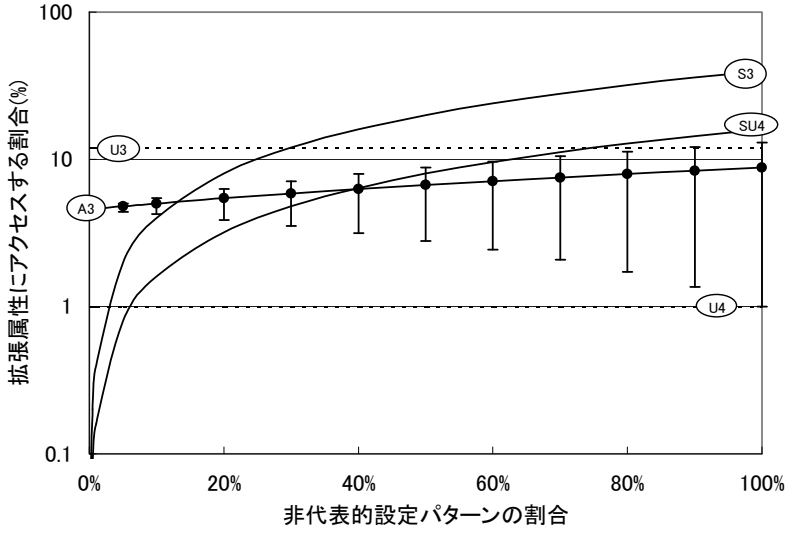

図 2.7 所有者の NFS アクセスの評価結果

にプロットした.

なお，1 つ目の仮定である代表的設定パターンの割合を変えた場合でも，以降の比較グラフでの中心線は多少変動するが，最悪，最良ケースはほとんど変動しない.

2.5.4 拡張属性のアクセス割合の評価

本小節では,拡張属性のアクセス割合を各格納方式ごとに定量化し評価する.以降では,(1) NFS 所有者，(2) NFS 非所有者，(3) CIFS，(4) NFS，CIFS 混在時所有者，(5) NFS，CIFS 混在時非所有者の 5 つのメニューをそれぞれ評価する.NFS は所有者特権の関係で,所有者と非所有者とでは，各ファイルリクエストで用いられるアクセスマスクが異なるため，この 5 つのメニューを設定した.

(1) NFS 所有者

NFS の所有者は所有者特権により 5bit あれば完全にすべてのリクエストのアクセスマスクがカバーできる．つまり格納方式 U5 ならばどんな状況でも基本属性の情報のみでアクセス可否の判定が可能となる．具体的に，その 5 種類のアクセスマスクは使用割合の高い順に，FILE_EXECUTE，FILE_READ_DATA，FILE_WRITE_DATA，FILE_APPEND_DATA，DELETE である.

図 2.7 に，格納方式 U3，U4，S3，SU4，A3 を比較した結果を示す．横軸は非代表的設定パターンの割合，縦軸はファイルリクエストのアクセス制御処理に拡張属性の情報が必要な割合であり対数スケールである(図 2.8 以下のグラフも同様)．下であればあるほど望ましい結果となる．A3 のみ期待値に加えて最悪，最良ケースをあわせてエラーバーで

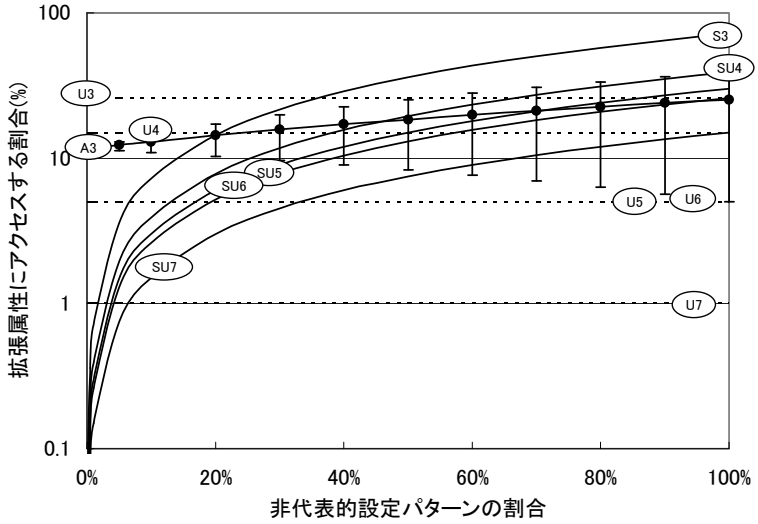

図 2.8 非所有者の NFS アクセスの評価結果

示している.

格納領域が 3bit である場合は, 一般的な部門ファイル共有のユースケースでは非代表的設定パターンは数%未満であると考えられるため方式 S3 が良い. ただし, 12%以上のファイルのセキュリティを細かく設定することが分かっている場合は方式 A3 が良い.

格納領域が 4bit 以上である場合は, 全体的に低く安定する方式 U4, U5 が良い.

(2) NFS 非所有者

NFS の非所有者は 8bit あれば完全にすべてのリクエストのアクセスマスクがカバーできる. つまり格納方式 U8 ならばどんな状況でも基本属性の情報のみでアクセス可否の判定が可能となる.

図 2.8 に, 格納方式 U3, U4, U5, U6, U7, S3, SU4, SU5, SU6, SU7, A3 を比較した結果を示す.

格納領域が 3bit である場合は, 方式 S3 が良い. ただし, 20%以上のファイルのセキュリティを細かく設定することが分かっている場合は方式 A3 が良い. 可能性は低いと考えるが, 50%以上のファイルのセキュリティを細かく設定することが分かっている場合は, 条件によっては方式 U3 が良いことがある.

格納領域が 4bit から 6bit である場合は基本的には方式 SU4, SU5, SU6 が良い.

格納領域が 7bit 以上である場合は, 全体的に低く安定する方式 U7, U8 が良い.

(3) CIFS

SPECsfs2008 では CIFS の所有者特権は特に影響しないため, 所有者, 非所有者とも同

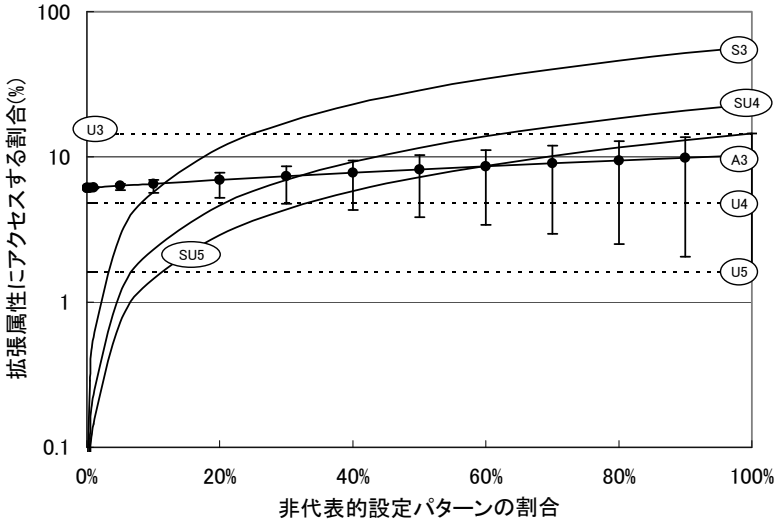

図 2.9 CIFS アクセスの評価結果

様の結果となる.

SPECsfs 2008 の発行するリクエストでは 6bit あれば完全にすべてのリクエストのアクセスマスクがカバーできる．つまり格納方式 U6 ならばどんな状況でも基本属性の情報のみでアクセス可否の判定が可能となる.

図 2.9 に，格納方式 U3，U4，U5，S3，SU4，SU5，A3 を比較した結果を示す.

格納領域が 3bit である場合は，方式 S3 が良い．ただし，12%以上のファイルのセキュリティを細かく設定することが分かっている場合は方式 A3 が良い.

格納領域が 4bit である場合は，基本的に方式 SU4 が良い.

格納領域が 5bit 以上である場合は，全体的に低く安定する方式 U5，U6 が良い.

(4) NFS，CIFS 混在時所有者

所有者が NFS と CIFS の両方を用いる場合，8bit あれば完全にすべてのリクエストのアクセスマスクがカバーできる．つまり格納方式 U8 ならばどんな状況でも基本属性の情報のみでアクセス可否の判定が可能となる.

図 2.10 に，格納方式 U3，U4，U5，U6，U7，S3，SU4，SU5，SU6，SU7，A3 を比較した結果を示す.

格納領域が 3bit である場合は，方式 S3 が良い．ただし，20%以上のファイルのセキュリティを細かく設定することが分かっている場合は方式 A3 が良い．可能性は低いと考えるが，50%以上のファイルのセキュリティを細かく設定することが分かっている場合は，条件によっては方式 U3 が良いことがある.

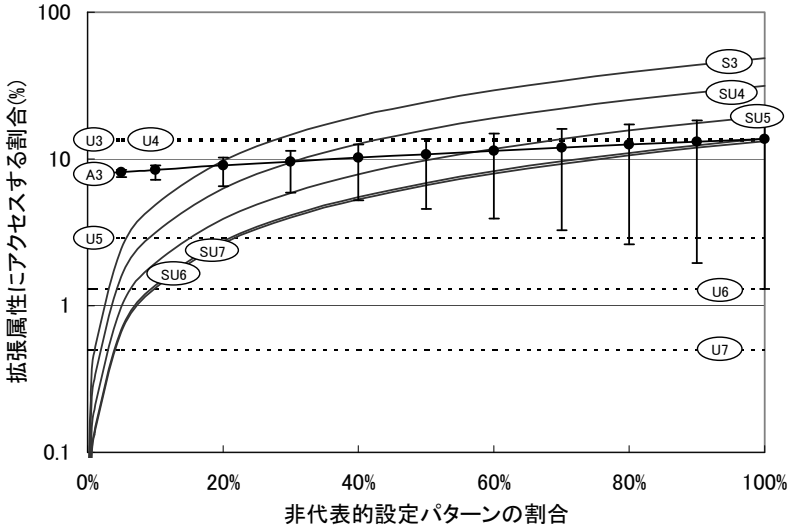

図 2.10 所有者の NFS および CIFS アクセスの評価結果

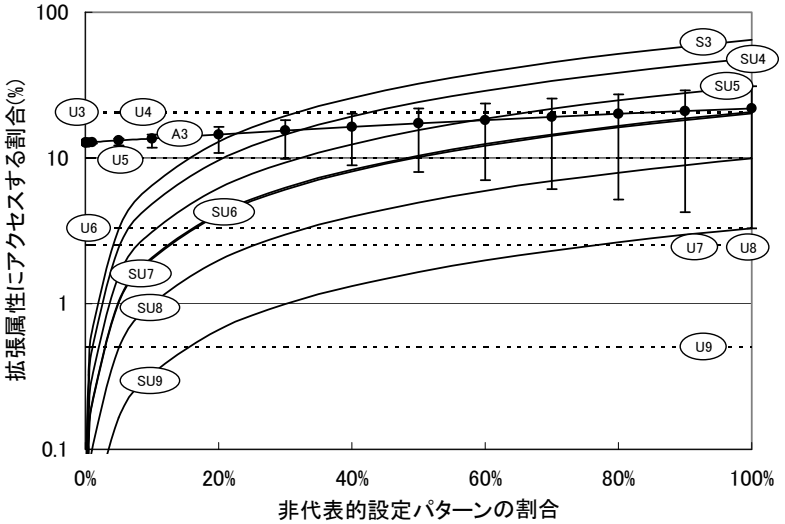

図 2.11 非所有者の NFS および CIFS アクセスの評価結果

格納領域が 4bit である場合は，基本的に方式 SU4 が良い.

格納領域が 5bit 以上である場合は，全体的に低く安定する方式 U5-8 が良い.

(5) NFS，CIFS 混在時非所有者

非所有者が NFS と CIFS の両方を用いる場合，10bit あれば完全にすべてのリクエストのアクセスマスクがカバーできる. つまり格納方式 U10 ならばどんな状況でも基本属性の情報のみでアクセス可否の判定が可能となる.

図 2.11 に，格納方式 U3，U4，U5，U6，U7，U8，U9，S3，SU4，SU5，SU6，SU7，SU8，SU9，A3 を比較した結果を示す.

格納領域が 3bit である場合は，方式 S3 が良い．ただし，22%以上のファイルのセキュリティを細かく設定することが分かっている場合は方式 A3 が良い．可能性は低いと考えるが，45%以上のファイルのセキュリティを細かく設定することが分かっている場合は，条件によっては方式 U3 が良いことがある．

格納領域が 4bit から 5bit である場合は，基本的に方式 SU4，SU5 が良い．

格納領域が 6bit 以上である場合は，全体的に低く安定する方式 U6-10 が良い．

2.5.5 格納方式がパーミッション参照更新リクエストに与える影響

2.4 節で述べたようにアクセス制御の高機能化にともない，パーミッション参照更新リクエストは，拡張属性へのアクセスが必要になりファイルリクエストスループットが低下する．パーミッション参照更新リクエストとはたとえば“`ls -l`”や“`chmod`”である．

この性能低下を改善する方針は大きく 2 つある．

第 1 の方針は，パーミッション参照更新リクエストで必要となる情報を，部分アクセス制御情報に格納することである．たとえば，格納方式 U であれば，パーミッションに対応するアクセスマスクビット(FILE_READ_DATA， FILE_WRITE_DATA，FILE_EXECUTE)を部分アクセス制御情報として格納すればよい. 格納方式 A であれば，対応するアクセスマスクビットもしくはその論理積を部分アクセス制御情報領域に格納すればよい．格納方式 S であれば，パーミッションビットが“- - -”の場合を非代表的設定パターンとして割り当て，それ以外を従来と同じ NFS の設定パターンに割り当てておけばよい．

格納方式 U と格納方式 A については，アクセス制御処理でよく使用されるビットとパーミッション参照更新リクエストで使用されるビットが必ずしも一致するわけではないので，この観点ではあまり良い格納方式とはいえない．

格納方式 S については, NFS の設定パターンと CIFS の簡易設定パターンをマップすることにより，アクセス制御処理によるファイルリクエストスループットの低下改善とパーミッション参照更新リクエストによるファイルリクエストスループットの低下改善を両立させることができる．

第 2 の方針は，ユーザビリティを犠牲としてファイルリクエストスループットを改善することである．たとえば“`ls -l`”時には部分アクセス制御情報のみの情報で応答する．つまり不完全なパーミッション情報の応答を許容する．この方針での性能改善の効果は，採用する格納方式に依存しないというメリットがある．運用への影響を考えるとあまり望ましい方針ではないが，Microsoft Windows Storage Server のように一部のユーザビリテ

表 2.4 評価結果まとめ

	3bit		4bit	5bit	6bit	7bit以上	完全カバー可能ビット数
	S3-A3 境界	A3-U3 境界	最適な方式				
NFS所有者	12%	なし	U4	U5			5
NFS非所有者	20%	50%	SU4	SU5	SU6	U7, U8	8
CIFS	12%	なし	SU4	U5	U6		6
NFS,CIFS所有者	20%	50%	SU4	U5	U6	U7, U8	8
NFS.CIFS非所有者	22%	45%	SU4	SU5	U6	U7 - U10	10

ィを犠牲にしている実例はある．たとえば，Windows Storage Server において NFS からファイル作成や“chmod”を行うと，その後は CIFS から Windows ACL が更新できない仕様となっている．また，Windows ACL の Everyone エントリの情報が，所有グループとその他ユーザのパーミッション表示に反映される仕様となっているため，所有グループのパーミッション表示が正確ではない．

本節で述べた性能低下の影響については 2.6 節であわせて議論する．

2.5.6 アクセス制御処理時間改善に適した格納方式まとめ

本小節では 2.5.4 節と 2.5.5 節の検討結果をまとめる．

表 2.4 に示すように 1 ユーザに割り当てる格納領域をパーミッションの 1 ユーザに割り当てるサイズの 3bit とする場合には格納方式 S3 が最も優れていると考える．非代表的設定パターンの割合が 12～22%以上の場合には格納方式 A3 が，45～50%以上の場合には格納方式 U3 が優れていることもある．ただしこの場合は，パーミッション参照更新リクエストのユーザビリティが犠牲となる．また，非代表的設定パターンの割合が所有者と非所有者で大きく異なり，最適な格納方式がそれぞれ異なる場合には，2.5.1 節で述べたとおり，所有者と非所有者(所有グループ，その他ユーザ)で格納方式を変えてもよい．

また，1 ユーザに割り当てる格納領域をパーミッション領域のすべてである 9bit とする場合には格納方式 U が最も優れている．NFS，CIFS 非所有者のパターンを除いて，適切なアクセスマスクを格納すれば，完全に基本属性の情報のみでアクセス可否の判定が可能である．この場合，格納対象者以外のパーミッション参照更新リクエストのユーザビリティが低下する．

2.6 提案方式の実装と評価

本節では，まず提案方式の実装について述べる．次に，部分アクセス制御情報のみでアクセス判定が可能となった場合，どの程度ファイルリクエストスループットが改善するかを事前見積もりし，その見積もりを実測により検証する．そして，複数の事例における提

案方式のファイルリクエストスループット改善効果を計算する．最後に，格納方式の選択に対する指針をまとめる．

2.6.1 提案方式の実装

提案方式の実装は Linux 2.6.12.5 + XFS[92]–[94]に対して行った．XFS は SGI 社によって開発された高性能ファイルシステムであり，本章で用いた理由は以下のとおりである．また以降の章でもほぼ同様の理由で XFS を用いる．

- 他のファイルシステムと比べると多機能であり，エンタープライズ用途に優れる．たとえば，ファイルサービス中でも実行可能なファイルシステムボリュームのサイズ拡張機能，ファイルシステム容量を効率的に使用できる動的 inode アロケーション機能，などがある．
- NEC 社による Windows ACL の優れた XFS 用実装が GNU General Public License (GPL)として公開されており，参考にできる．
- エクステント管理，B+ツリー，遅延アロケーション，プリアロケーションなど多数の性能向上機能が実装されている．

なお，上記理由にあるようにアクセス制御機能そのものについては，NEC 社 SC-LX のコードを一部参考にした．

提案方式を組み込んだアクセス制御機能はカーネルモジュールとして実装した．カーネルモジュールのインターフェースに Linux Security Module (LSM)を利用し，カーネルの Virtual File System (VFS)レイヤが LSM 経由でアクセス制御処理を実行する構造とした．ただし VFS レイヤだけで完全に Windows ACL 互換のアクセス制御機能を実現するのは困難だったため，一部 Samba サーバ，NFS サーバ，XFS も改造して，アクセス制御カーネルモジュールを呼び出す処理を追加している．また標準の LSM フック関数だけでは一部不足があったので，追加でフック関数を独自定義している．

以上の工夫により，提案方式を他のローカルファイルシステムや他のネットワークファイルシステムプロトコルへ容易に適用できる構造とした．

2.6.2 提案方式により期待されるファイルリクエストスループット改善効果の事前見積もり

ファイルリクエストスループット改善効果の見積もりを行う場合にも，ファイルアクセスパターンに何らかの仮定をおく必要がある．本小節でも 2.5 節と同様に，仮定するファイルアクセスパターンを SPECsfs2008 とする．

SPECsfs2008 ベンチマークではキャッシュの効果を考えなければ，ランダムディスクアクセス性能と CPU 性能が性能支配要因となる．

ランダムディスクアクセス性能の観点では，ユーザデータアクセス量を D_U，高機能化により追加で発生するアクセス制御情報アクセス量を D_A とすると，アクセス制御高機能化により，ベンチマーク性能であるファイルリクエストスループットは $D_U/(D_U+D_A)$となることが予測される．D_A は各ファイルのアクセス制御のリスト長に依存し，アクセス制御が実行される総リクエスト回数を R_A，各ファイルの平均アクセス制御情報サイズを L とすると，$D_A = R_A \times L$ となる．したがってベンチマーク性能の予測式は $D_U/(D_U+R_A \times L)$となる．

たとえば，要求負荷 16KOPS の場合の D_U は 575GB，R_A は 2.1M 回である．L は最小で 4KB, 最大で 64KB となるため, 予測式の値は 0.986(4KB 時)～0.815(64KB 時)となる．つまり，アクセス制御に関するファイルリクエストスループット低下率は 1.4%(4KB 時)～18.5%(64KB 時)程度になると考えられる．要求負荷にかかわらず D_U と R_A の比率はほぼ一定であるため，上記の低下率は要求負荷に依存せず一定である．この要因によるファイルリクエストスループット低下は，提案方式により改善できる．

また，2.4 節，2.5.5 節で述べたパーミッション参照更新リクエストに関するファイルリクエストスループットの低下についても同様に計算すると，低下率は 1.3%(4KB 時)～17%(64KB 時)と予測される. パーミッション参照リクエストのユーザビリティを犠牲にすれば，性能低下率は 0.13%(4KB 時)～2%(64KB 時)となり大幅に軽減される．パーミッション更新リクエストのユーザビリティも犠牲にすれば，この要因によるファイルリクエストスループット低下は改善できる．またすでに議論したように格納方式 S を採用すれば，ユーザビリティを犠牲にせずこの要因によるファイルリクエストスループット低下は改善できる．

他にも上記以外のランダムディスクアクセス性能に起因するファイルリクエストスループット低下要因として，ファイルの作成削除リクエストの処理時間増加がある．アクセス制御高機能化により，これらのリクエストには，アクセス制御情報自体を作成削除する処理が追加されるためである．同様に計算すれば本要因によるファイルリクエストスループット低下は 0.097%(4KB 時)～1.5%(64KB 時)程度になると考えられる. この要因によるファイルリクエストスループット低下の改善は，提案方式の適用対象範囲外である．

次に，CPU 性能の観点では，処理プログラムの増加によるファイルリクエストスループット低下が考えられる．実装したプログラムのコード量を調査したところ，従来処理に対して 1 割程度増加していたため，Cycles Per Instruction (CPI)が同程度とすれば，ファイルリクエストスループットも 1 割程度低下することが予想される．この要因によるファイルリクエストスループット低下の改善も，提案方式の適用対象範囲外である．

以上により，アクセス制御高機能化によりファイルリクエストスループットは最悪で約40%以上低下するが，本提案の適用により，最良で約 10%の低下まで改善することが期待できる．より一般的な条件でのファイルリクエストスループット改善効果は 2.6.4 節で検討する．

2.6.3 実機測定による評価

提案方式の適用により，部分アクセス制御情報でアクセス判定が可能になった場合に，どの程度ファイルリクエストスループットを改善するかを実機測定により評価する．測定環境は表 2.5 のとおりである．

本評価システムで実装した格納方式は S3 である．他の格納方式での実装を行わなかった最大の理由は，他の格納方式ではパーミッション参照更新リクエストによるファイルリクエストスループット低下を改善しにくいためである．ただし，前述したように他の格納方式でもユーザビリティを犠牲にすれば，この低下の影響を排除可能であり，その場合のファイルリクエストスループットは基本的に本節に示す格納方式 S3 のファイルリクエストスループットと同様の傾向を示す．

また，通常条件と最悪条件での改善効果を見積もるためアクセス制御情報サイズ L は通常サイズの ACL が格納可能な 4KB と最大サイズの 64KB とした．

測定方法は次のとおりである．NFS 版の SPECsfs2008 ベンチマークを用いて，12KOPS から 18KOPS までの要求負荷を 1KOPS 刻みに与え，その最大値を集計する．この要求負荷の範囲は，最大達成負荷との乖離が起きない範囲を参考にして設定した．測定はそれぞれ 3 回実施し，各測定で得られた達成負荷の最大値の平均値を算出する．アクセス制御高機能化を行ったシステムにおいて，全リクエストのうち拡張属性にアクセスするリクエストの割合を 0%から 44%まで 4.4%刻みで変化させて測定を行う．本測定条件では，全リク

表 2.5 測定環境

項目	台数	仕様
サーバ	1	CPU: Intel Xeon CPU 5140 2.33GHz (Dual Core) x 2 Memory: 10,960,444KB Kernel: Linux 2.6.12.5 + XFS + 提案方式アクセス制御機能 OS distribution: SUSE Linux Professional 9.3 NFS utils package version: 1.0.7
ディスク	38	108GB/LU (RAID1+0, 2D+2D) x 38
クライアント	4	CPU: Intel Xeon CPU 2.80GHz (Dual Core) Memory: 3,632,416KB Kernel: Linux 2.6.26-2-686 OS distribution: Debian 5.0.2

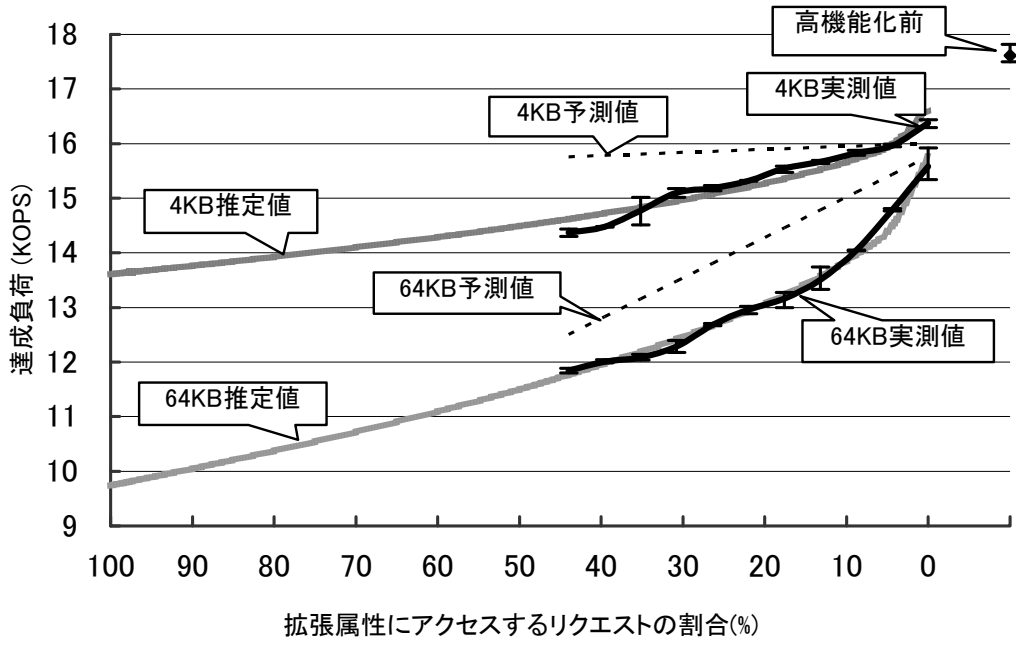

図 2.12 提案方式適用システムによる実測結果

エストの 56%はアクセス制御処理自体が実行されないため，44%までの測定としている．これはアクセス者にかかわらず 28%のリクエストはアクセス制御処理自体が不要なのに加えて, SPECsfs2008 は所有者権限によりファイルアクセスが行われるため所有者特権によりさらに 28%のリクエストのアクセス制御処理が省略されるためである．また，比較のため，高機能化前のシステムについても測定を行う．

結果を図 2.12 に示す．横軸は図 2.7 から図 2.11 の縦軸の表記とあわせるため，拡張属性にアクセスするリクエストの割合とした．縦軸の値は，3 回測定で得られた達成負荷の最大値の平均値である．3 回測定の最大値，最小値も誤差範囲として示した．拡張属性にアクセスするリクエストの割合 44%～0%の範囲の値を表現できる代表的な関数を調べた結果，双曲線関数が最もよく一致した．そこで，本節以降での議論では，100%～44%の範囲については双曲線近似による推定値を用いる．なお本測定値は，SPEC が指定する測定ルールに厳密にしたがっているわけではないため，SPEC のサイトに登録，公開されている値とは単純比較はできない. ただし, 本測定のデータは同一条件にて測定しているため，相対的な違いは比較可能である．

単にアクセス制御を高機能化しただけのシステムに，所有者がアクセスする場合，全リクエストの 70%で拡張属性アクセスが発生する．非所有者の場合は 98%となる．これがアクセス制御を高機能化したシステムでの最悪性能となる．横軸値 98%での達成負荷は，高機能化前の達成負荷に対し, L = 4KB で 23%, L = 64KB で 44%の性能低下が見込まれる．

これに対し，本提案方式の格納方式 S を適用すれば，拡張属性にアクセスするリクエストの割合を所有者の場合 44%以下に，非所有者の場合 72%以下に抑えることができる．

さらに，格納方式 S の効果が最大，つまりすべてのアクセスユーザが部分アクセス制御情報の対象ユーザで，すべてのファイルの部分アクセス制御情報の対象アクセス制御情報の設定パターンが代表的設定パターンである場合(横軸値 0%)，高機能化前の最大達成負荷に対する性能低下率は L = 4KB で 7%，L = 64KB で 11%に改善した．

2.6.2 節の予測値に対する実測値の傾向を L = 4KB，L = 64KB のそれぞれで考察する．

L = 4KB の場合，拡張属性にアクセスする割合が小さい領域(< 10%)では予測値と実測値はよく一致しているが，拡張属性にアクセスする割合が大きくなればなるほど，乖離が大きくなる．この理由としては，小さいサイズの拡張属性アクセスにより，単位 I/O あたりの平均データサイズが小さくなることから，ランダムアクセス傾向がより強くなることによるファイルリクエストスループット低下の拡大が考えられる．

L = 64KB の場合，全体の傾向はある程度一致しているが拡張属性にアクセスする割合が 10%～30%に乖離が見られる．これも同様にランダムアクセス傾向が強くなることと，キャッシュの影響が考えられる．これは，拡張属性にアクセスする割合が小さい領域ではファイルリクエストスループット値の立ち上がりが大きくなる傾向を示していることが根拠となる．予測値と実測値で乖離の見られる部分に関しては，実入出力回数の計測や実データアクセス量などの統計情報を取得分析することにより，今後検討を行う必要がある．

なお，本節で用いたハードウェア環境以外においても，多少の性能改善効果の差は出ることが考えられるが，基本的な傾向は同様である．その理由は，ほとんどのシステムはディスク I/O が性能ボトルネックとなることと，その場合ある程度ディスク I/O 回数の増加で性能改善効果を説明できるためである．

また，本節で用いたワークロード以外では，性能改善効果は異なることがある．たとえば，大サイズファイルに対してローカルアクセスを行った場合は，性能改善効果はほとんど生じない．なぜなら，そのようなワークロードでは処理するファイルのデータ量に対して，アクセス制御の頻度が少ないためである．逆に小サイズファイルに対して，メタデータアクセス系のファイルリクエストを高頻度に実行する場合は，より大きな性能改善効果が得られる．

2.6.4　提案方式の具体的事例でのファイルリクエストスループット改善効果

提案方式の性能改善効果を具体的事例で見積もるためには，以下の 4 つの情報が必要となる．

(1) 各ファイルに付与されるアクセス制御情報の平均サイズ

(2) 各ファイルの部分アクセス制御情報対象ユーザ(所有ユーザ，所有グループ，その他ユーザ)のアクセス制御情報が代表的設定パターンと一致する確率

(3) 各ファイルに対する全アクセスリクエストのうち，部分アクセス制御情報非対象ユーザからのアクセスリクエストの割合

(4) 各ファイルに対する全アクセスリクエストのうち，所有者からのアクセスリクエストの割合

このうち(3)，(4)については，実際のアクセスログを解析するのが望ましいが，そのようなデータは一般に取得困難であるため，ここでは次の仮定を行った．(3)については，あるファイルの部分アクセス制御情報対象ユーザの数(そのエントリがあるかどうかに依存するが，最も一般的には所有ユーザと所有グループのメンバ数の合計値)を α とし，部分アクセス制御情報非対象だがそのファイルに対するアクセス権を持つユーザの数を β とする．仮に α のユーザと β のユーザのファイルに対するアクセス頻度を同一とした場合，(3)の割合は $\beta/(\alpha+\beta)$ として計算される．(4)についても同様の仮定をおいた．(3)，(4)の情報と 2.6.3 節に記載した各条件での拡張属性アクセス割合から，提案方式適用前後の拡張属性にアクセスするリクエストの割合が計算可能となる．

これらの情報について，ある組織で運用している約百個のファイル共有を調査したところ，(1)，(2)はどれも同じ傾向だったが，(3)，(4)は大きく 4 つの種類の事例に分類できることが分かった．具体的には，(a) 個人共有フォルダ，(b) 公開共有フォルダ，(c) 組織共有フォルダ，(d) 非組織共有フォルダの 4 種類である．この情報を表 2.6 にまとめる．

(1)のアクセス制御情報の平均サイズは，調査した事例の範囲においてはいずれも 4KB 未満だった．

(2)の代表的設定パターンの一致確率は，調査した事例の範囲においては 100%だった．したがって最適な格納方式は，いずれの事例においても格納方式 S3 となる．ただし(a)，(b)の事例においてはアクセス対象者が限られるため，格納方式 U9 であってもよい．

表 2.6 実事例のアクセス制御に関する情報

#	事例	(1) アクセス制御情報平均サイズ	(2) 代表的設定パターン確率	最適な格納形式	(3) 非対象ユーザからのアクセス割合	(4) 所有者からのアクセス割合	拡張属性にアクセスするリクエストの割合	
							提案方式適用前	提案方式適用後
(a)	個人共有フォルダ	4KB未満	100%	S3, U9	0%	100%	70%	0%
(b)	公開共有フォルダ	4KB未満	100%	S3, U9	0%	0.2%	98%	3%
(c)	組織共有フォルダ	4KB未満	100%	S3	18%	4%	97%	13%
(d)	非組織共有フォルダ	4KB未満	100%	S3	76%	8%	96%	55%

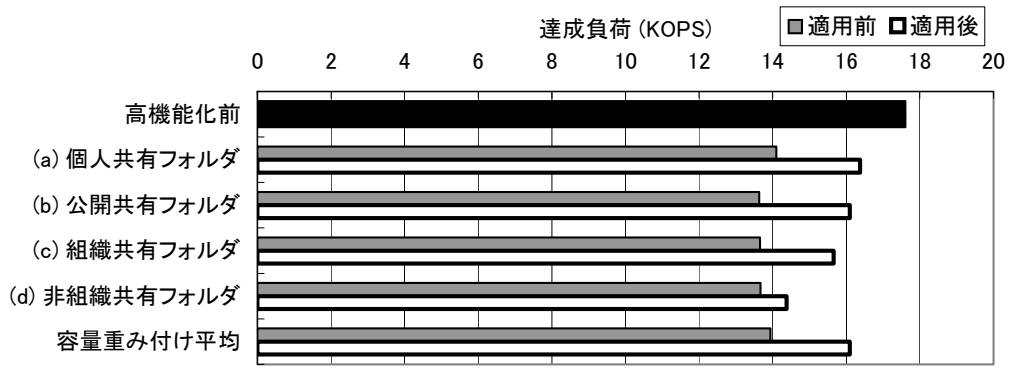

図 2.13 提案方式適用によるファイルリクエストスループットの改善効果

(3)の非対象ユーザからのアクセス割合は，事例(a)，(b)では所有者もしくはその他ユーザのエントリが存在するため 0%となる．事例(c)では適切なグループが設定されているため，18%と比較的低い確率となっている．事例(d)では組織の構造と連携しない単位の共有のため，適切なグループが設定されていないことにより，76%と高い確率となっている．

(4)の所有者からのアクセス割合は，事例(a)は 100%となるが，それ以外の事例では 10%未満となる．それぞれの事例における拡張属性にアクセスするリクエストの割合の変化は表に示すとおりとなる．

図 2.12 と表 2.6 のデータより推定した，各事例における高機能化後の提案方式適用前のファイルリクエストスループット，提案方式適用後のファイルリクエストスループットを図 2.13 に示す．さらに各事例の容量での重み付けを行った平均のファイルリクエストスループットもあわせて示す．平均では，提案方式適用前では高機能化前に対してファイルリクエストスループットが 21%低下するが，提案方式を適用すれば低下率を 9%に抑えることができる．

2.6.5　格納方式の選択に対する指針

これまでの議論をふまえ，システム設計者が格納方式をどのように選べばよいかを本小節にまとめる．まず，パーミッション参照更新リクエストのユーザビリティの低下が許容できるかどうかを選択する．

ユーザビリティの低下が許容できない場合は，所有者，所有グループ，その他ユーザ(以降まとめて，パーミッション対象ユーザ)に対してそれぞれ格納方式 S3 を適用するのが望ましい．

ユーザビリティの低下が許容できる場合において，アクセスが想定されるユーザ(もしくはグループ)が限定的である場合，想定アクセスユーザ(もしくはグループ)のアクセス制御

情報に対して格納方式 U9 を適用するのが望ましい．アクセスが想定されるユーザが限定的でない，もしくは事前にそのような情報を得ることが難しい場合は，パーミッション対象ユーザで 3bit ずつ使用するのが望ましい．この場合パーミッション対象ユーザに設定されるアクセスマスクパターンの割合が得られる場合は，その割合にしたがって S3，A3，U3 の中から選択する．割合を得るのが難しい場合は，S3 を選択するのが望ましい．

2.7 結言

本章では，CIFS のアクセス制御仕様である Windows ACL を Linux で実装し，アクセス制御を統合する高機能化を行う際の課題であるファイルリクエストスループット低下を改善する階層型アクセス制御方式を提案した．そして，提案方式で用いる基本属性の部分アクセス制御情報領域に格納する情報について，基本属性の情報のみでアクセス判定可能な確率を評価指標として，高頻度使用アクセスマスクビット，高頻度設定アクセスマスクパターン，アクセスマスクビット論理積の 3 格納方式を比較した．その結果，3bit の格納領域の場合には高頻度設定アクセスマスクパターン，9bit の格納領域の場合には高頻度使用アクセスマスクビットの格納方式が最も適していることを明らかにした．提案方式を実装したシステムを用いて評価を実施したところ，4 事例の容量重み付け平均で，アクセス制御高機能化前に対し，提案方式適用前はファイルリクエストスループットが約 21%低下していたが，提案方式適用により低下率を約 9%に改善できることを明らかにした．本章で述べた格納方式 S3 を用いた提案方式は日立ファイルストレージ[95]で実用化済みである．

今回は部分アクセス制御情報領域のサイズが最も一般的な 9bit の場合を重点的に検討したが，一部のファイルシステムではこれ以外にも基本属性中の予約領域やパディング領域など使える領域を増やせる場合がある．今後の課題としては，部分アクセス制御情報のために使える領域のサイズが 9bit よりも大きい場合に，どのような情報が格納対象として望ましいかを検討することが挙げられる．

第 3 章

複数の ACL 形式を選択的に使用する低機能 ACL-高機能 ACL 併用方式

3.1 緒言

本章では, 低機能 ACL が付与されたファイルに高機能アクセス制御を適用する場合の, 長時間の運用停止とファイルリクエストスループットの低下を改善する低機能 ACL-高機能 ACL 併用方式について述べる.

高機能アクセス制御に対応していない UNIX 系 OS を，対応済みの OS へバージョンアップする際，これまで運用していた低機能 ACL 対応ファイルシステムを高機能 ACL 対応ファイルシステムへ移行する場合がある．この場合，低機能 ACL 対応ファイルシステムから高機能 ACL 対応新規ファイルシステムに, 全データをコピーするのが一般的である．この移行方法はファイルシステム運用を長時間停止する必要があり，さらに移行先の新規ファイルシステムの容量が必要となることから，ファイルシステム使用者，管理者に大きな負担となっている．

そこで我々は初回ファイルアクセス時に，ファイルサーバやユーザアプリケーションに透過的に低機能 ACL 情報を高機能 ACL 情報に変換する ACL 変換機能(以降，オンアクセス ACL 変換と呼ぶ)を開発した[81]．オンアクセス ACL 変換機能を用いた移行は，前述したコピーによる移行に比べると大幅に運用に対する負担を軽減することができる．しかし運用中にアクセス制御情報を変換するため，変換負荷によりファイルリクエストスループットを低下させてしまう．また，運用開始前に同変換機能により ACL 情報を一括移行しておく場合でも，アクセス制御の高機能化によって ACL のサイズが増大するため，ディスク I/O 負荷が増加しファイルリクエストスループットを低下させてしまうことがある．加えて事前一括移行では，全データコピーによる移行よりは軽減されるものの，事前一括移行の間ファイルシステム運用を停止する必要がある．

本章ではこのファイルリクエストスループットの低下と長時間の運用停止を改善する低

機能 ACL-高機能 ACL 併用方式を提案する．提案方式は，ファイルシステムの基本・拡張属性に対する参照更新処理の時間の違いに着目し，アクセス制御情報のサイズに応じて，変換後の高機能 ACL 情報を書き込み，次回以降のアクセスにそれを参照する場合と，変換後の高機能 ACL 情報は書き込まず，次回以降のアクセスも毎回低機能 ACL 情報から変換する場合を使い分ける．つまり，移行によりファイルリクエストスループットの観点で不利になる場合は，高機能 ACL 情報形式へのディスク上での移行は行わず，低機能 ACL 情報形式を使い続ける．これにより性能低下を改善する．

なお，本章では低機能 ACL として POSIX ACL を，高機能 ACL として Windows ACL を取り上げる．つまり，NFS サービス用データと CIFS サービス用データを同一ファイルシステムボリューム上で運用していたユースケースを想定する．高機能 ACL がサポートされていない環境で，同ユースケースの運用を行う場合は，POSIX ACL の利用が一般的である．

本章の以降の構成は次のとおりである．3.2 節では，文献[81]で提案したファイルシステム移行方式，移行の手段であるオンアクセス ACL 変換，その課題であるファイルリクエストスループットの低下について説明する．3.3 節では，提案方式である低機能 ACL-高機能 ACL 併用方式を説明する．3.4 節では，提案方式を実装したシステムにおいて性能改善効果を確認する．最後に 3.5 節で，本研究のまとめを述べる．

3.2　高機能 ACL 対応ファイルシステムへの移行方式とその課題

本節では，まず高機能 ACL 対応ファイルシステムへの移行方式について述べる．その後，移行方式の共通の課題であるファイルリクエストスループットの低下と長時間の運用停止について述べる．

3.2.1　高機能 ACL 対応ファイルシステムへの移行方式

本小節では，運用していたファイルシステム(以降旧 FS)を高機能 ACL 対応ファイルシステム(新 FS)に移行する従来方式について述べる．従来の移行方式は大きく分けて 2 種類あり，(1)全データコピーによる移行方式，(2)高機能 ACL 情報のみを付与する移行方式がある．さらに，(2)には(2a)運用再開前に一括して全ファイルに高機能 ACL 情報を付与する移行方式と，(2b)運用再開後にユーザアクセスのあったファイルの初回アクセス時に高機能 ACL 情報を付与する移行方式がある．

著者らは移行方式(2a)(2b)を実現するために，ファイルの初回アクセス時に，低機能 ACL 情報(たとえば POSIX ACL 情報)を読み出し，高機能 ACL 形式(たとえば Windows ACL

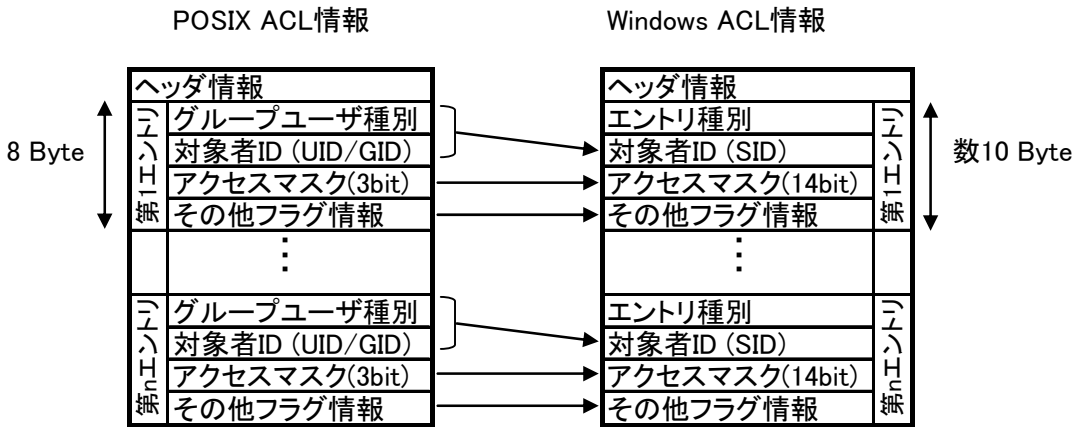

図 3.1 ACL 情報変換の例

情報)に変換し，高機能 ACL 情報を同ファイルに付与するオンアクセス ACL 変換方式を提案している[81].

ACL 情報の変換の例を図 3.1 に示す．POSIX ACL はヘッダ情報と 3 つ以上のエントリからなる．各エントリは，そのエントリの対象者がユーザを意味するかグループを意味するかの種別情報，その対象者を示す対象者 ID，その対象者がどういったアクセス権を持つかを規定する 3bit のアクセスマスク，その他フラグ情報からなる．各エントリのサイズは実装依存だが，Linux では 8Byte である．Windows ACL はヘッダ情報と 1 つ以上のエントリからなる．各エントリは，そのエントリが許可を意味するか拒否を意味するかのエントリ種別情報，エントリの対象者を示す対象者 ID，14bit のアクセスマスク情報，その他フラグ情報からなる．各エントリのサイズは実装依存だが，情報量が増えるため数 10Byte 程度になる．オンアクセス ACL 変換方式では，この変換をユーザアクセスの処理中に透過的に実行する．

移行方式(1)は，高機能 ACL に対応したファイル共有サービスのクライアント経由で旧 FS から新 FS に全ファイルをコピーする．ファイルシステムプログラムが動作する計算機の OS が高機能 ACL に対応したコピー機能を提供していれば，同計算機上のコピープログラム経由でのコピーであってもよい．本方式ではコピーの間ファイル共有サービスのクライアントやユーザアプリケーションへのファイルシステム運用を停止する必要があり，旧 FS 用ボリュームと同サイズ以上の容量を持つ新 FS 用ボリュームを新たに用意する必要がある．運用停止時間は，容量が大きい場合数日を要することがある．また ACL のサイズは移行前より増大するためファイルリクエストスループットを低下させてしまう．

移行方式(2a)は，運用停止中にファイルストレージが前述のオンアクセス ACL 変換を旧

FS 上の全ファイルに対し実行する．移行方式(1)と比較すれば，旧 FS を直接新 FS に移行できるため，新 FS 用ボリュームを用意する必要がない．また，運用停止時間はファイル数や ACL エントリ長によるが，数 10 分から数 10 時間程度に短縮できる．しかし本移行方式でも，ACL のサイズは移行前より増大するためファイルリクエストスループットを低下させてしまうことがある．

移行方式(2b)は，運用停止中にファイルストレージが移行対象の旧 FS に対して，新 FS に対応したことを示す情報のみを付与する．この時点では各ファイルにオンアクセス ACL 変換は適用しない．運用を再開後，ファイル共有サービスのクライアントやユーザアプリケーションからのファイルアクセス時にオンアクセス ACL 変換が動作し，高機能 ACL 情報が付与される．本移行方式では運用再開前の一括移行処理を行わないため，ファイルシステム運用停止時間をさらに短縮でき，数秒から数 10 秒程度にできる．しかし移行方式(2b)は，ユーザアクセスの延長で高機能 ACL 情報を付与するため，ユーザへのファイルリクエストスループットをさらに低下させてしまうことがある．

次小節ではこの移行方式(2a)と移行方式(2b)の共通の課題であるファイルリクエストスループット低下の詳細について述べる．

3.2.2 オンアクセス ACL 変換機能による移行でのファイルリクエストスループットの低下

本小節では，前小節で述べた課題であるオンアクセス ACL 変換による移行でのファイルリクエストスループット低下の詳細を説明する．

オンアクセス ACL 変換は，ファイル共有サービスのクライアントやユーザアプリケーションがファイルシステムオブジェクト(ファイルやフォルダ)にアクセスし，アクセス制御処理を行うタイミングで ACL の変換処理を行う．図 3.2 にオンアクセス ACL 変換機能付きのアクセス制御処理の概要を示す．まず，アクセスされたファイルシステムオブジェクトが変換後の高機能 ACL 情報を保持しているかどうかを確認する．具体的には，拡張属性に高機能 ACL 情報があるかどうかを確認するか，基本属性にその有無を示すフラグを設けてその値を確認する．

ファイルシステムオブジェクトが高機能 ACL 情報を保持していなかった場合，オンアクセス ACL 変換機能は，低機能 ACL 情報を参照し，高機能 ACL 情報形式に変換する．その変換結果を拡張属性に書き込み，変換結果に基づきアクセス制御処理を実施する．高機能 ACL 情報を保持しているファイルシステムオブジェクトであった場合，高機能 ACL 情報を参照し，アクセス制御処理を実施する．

以上の処理の概要から分かるとおり，オンアクセス ACL 変換では，情報形式の変換と，

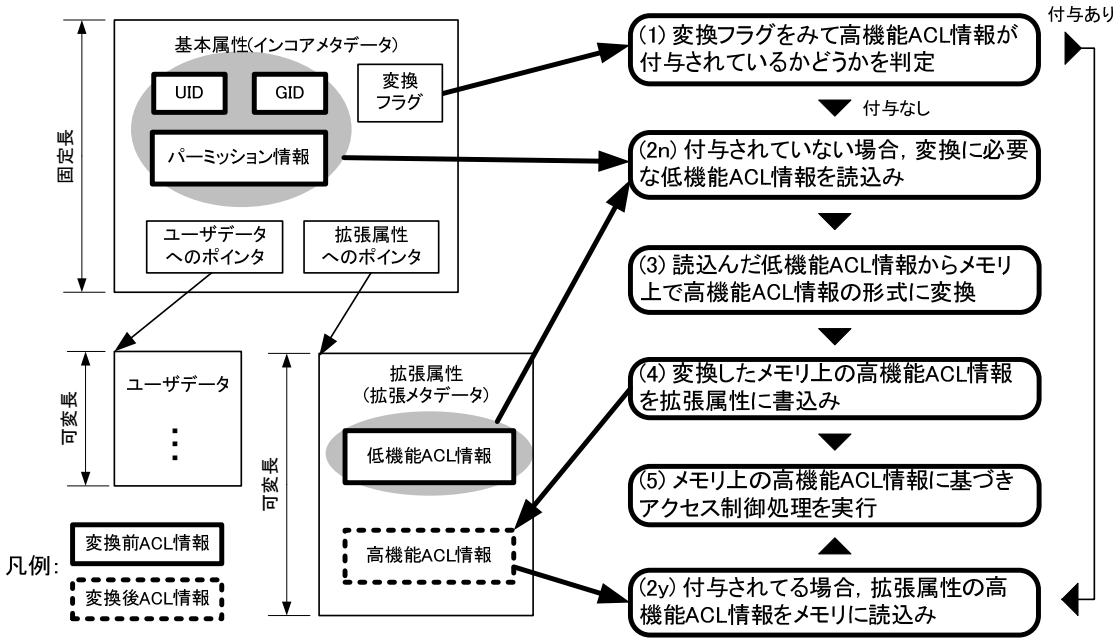

図 3.2 オンアクセス ACL 変換機能付きアクセス制御処理の概要

変換後のアクセス制御情報の書き込みのため，少なくとも最初の 1 回目のアクセス制御処理時間は増加する．さらに，2 回目以降のアクセス制御処理においても，基本・拡張属性の参照時の処理時間の違いにより，低機能 ACL 情報への参照時間より高機能 ACL 情報への参照時間が長くなることがある．

これらのアクセス制御処理時間の増加により，オンアクセス ACL 変換機能を用いた移行では，全体のファイルリクエストスループットが低下することがある．

3.3 低機能 ACL-高機能 ACL 併用方式

本節では，まず提案方式である低機能 ACL-高機能 ACL 併用方式の概要を述べる．次に低機能 ACL 形式と高機能 ACL 形式をどのような条件で使い分けるべきかを検討し，さらに提案方式の詳細を説明する．続いて，提案方式と従来の移行方式の関係を議論し，最後に提案方式の適用範囲を明確化する．

3.3.1 提案方式の概要

前節で述べたファイルリクエストスループットの低下を改善するためには，変換結果の拡張属性への書き込みを条件によっては行わず，低機能 ACL 情報を使い続ける方式が考えられる．つまり従来のオンアクセス ACL 変換機能による移行では，初回ファイルアクセス時に無条件に高機能 ACL を付与していたが，提案方式ではファイルリクエストスループットを考慮して付与の可否を判断する．具体的には提案方式は，ファイルシステムの

基本・拡張属性の参照更新時の処理時間の違いに着目し，アクセス制御時，変換結果を書き込むとファイルリクエストスループットの点で有利な場合は変換結果を拡張属性に書き込み，変換結果を書き込むとファイルリクエストスループットの点で不利な場合は変換結果を書き込まず，次回以降のアクセスも低機能 ACL 情報から毎回変換する．

ここで，低機能 ACL 情報から高機能 ACL へ変換し，変換結果は書き込まず，アクセス制御処理を実行する場合の時間を T_C，低機能 ACL 情報から高機能 ACL へ変換し，変換結果を拡張属性に書き込んだ後，アクセス制御処理を実行する場合の時間を T_{CW}，変換済み高機能 ACL を参照し，アクセス制御処理を実行する場合の時間を T_A，あるファイルに対して実行されるアクセス制御処理回数を n とする．このとき，

$$T_{CW} + (n-1) \times T_A > n \times T_C$$

の不等式が成立する場合は，変換結果の拡張属性への書き込みは行わず，低機能 ACL 情報から毎回変換処理を行った上で，アクセス制御処理を実行するのがよい．ただし，あるファイルに対して実行されるアクセス制御回数である n は一般に予測困難であるため，上記不等式を基準に書き込み可否の場合分けを行うのは難しい．しかし，$T_A > T_C$ が成立する場合は，上記不等式は絶対不等式となるため，書き込み可否の場合分けが可能となる．また，3.2.1 節の移行方式(2a)のように運用再開前に変換を行う場合は，T_{CW} を考慮する必要はなく，単に $T_A > T_C$ の条件が成立する場合に書き込みを行わず繰り返し変換するのがよい．

次小節ではどのような場合に書き込みを行うべきか，そうでないかを定量的に事前検討する．

3.3.2 高機能 ACL 情報書き込み可否の選択に関する検討

本小節では Linux の XFS[92][93]をベースに開発したオンアクセス変換機能を用いて，前小節の T_A，T_C，T_{CW} を計測し，書き込み可否の条件の検討を行う．XFS の基本属性，拡張属性は表 3.1 に示す参照更新時の処理時間の違いがある[94]．どの形式になるかは拡張属性のサイズ，つまり ACL のエントリ数に依存する．

そこで，低機能 ACL のエントリ数を変化させ，T_A，T_C，T_{CW} がどのように変化するかを計測した．結果を図 3.3 に示す．測定は mount 処理直後の状態，つまり拡張属性がキャッシュされていない状態で 9 回実施した．空き領域検索やジャーナルの書き戻しなどによる偶発的なアクセス制御処理時間悪化の影響を排除して傾向をつかむため，平均値ではなく最小値を示している．平均値はエラーバーで示した．横軸が ACL エントリ数，縦軸は

表 3.1 XFS 基本属性・拡張属性の形式分類と参照更新時の処理時間

属性種別	形式	格納可能 情報サイズ	参照更新時の処理時間
基本属性		N/A	アクセス判定時には確実にメモリ上にあるため，100μs 程度で参照更新アクセス可能
拡張属性	Short form	約 30 B 未満	基本属性と同様
	Leaf local form	3 KB 未満	メモリ上にないことが多いため，参照はディスクアクセスが発生し数 100μs～1ms 程度，更新はジャーナルログに書くだけなので 100μs 程度
	Leaf remote form	3 KB 以上	参照は 2 ページ以上のディスクアクセスが発生し 1ms 以上，更新は同期書き込みとなるため 1ms 以上

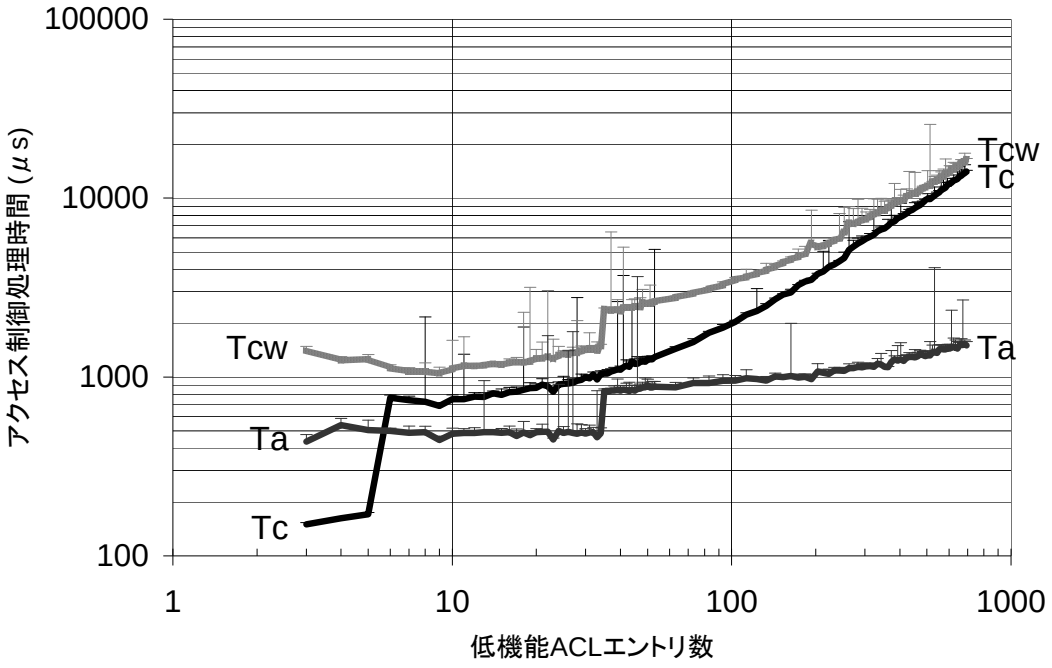

図 3.3 オンアクセス ACL 変換機能付きアクセス制御処理時間

アクセス制御に要した時間であり両対数グラフである．横軸の ACL エントリ数は 3 から開始しているが，低機能 ACL である POSIX ACL では最低エントリ数が 3 となっているためである．なお，エントリ数 3 の低機能 ACL はパーミッション情報のみとなり，拡張属性には何も格納されない．

低機能 ACL のエントリ数 3～5 の領域では，変換前のファイルの状態は基本属性のみもしくは拡張属性の Short form 形式で，変換結果書き込み後は拡張属性の Leaf local form 形式となる．この領域では，$T_A > T_C$ が成立するため，変換結果の書き込みは行わず，低

機能 ACL から毎回変換したほうがよい.

低機能 ACL のエントリ数 6～34 の領域では，変換前後ともファイルの状態は拡張属性の Leaf local form 形式である．この領域では，T_A，T_C，T_{CW} の順に時間が長くなり，それぞれの時間差が約 0.5ms となっている．したがって，各ファイルに対する平均アクセス制御回数が 2 回程度以上であれば変換結果を書き込んだほうが性能上有利であるため，初回アクセス時に変換結果を書き込んだほうがよい.

低機能 ACL のエントリ数 35～の領域では，変換前のファイルの状態は拡張属性の Leaf local form 形式で，変換結果書き込み後は Leaf remote form 形式となる．この領域のエントリ数が数 10 個までは T_{CW} の時間は T_A，T_C に対して著しく長大化している．また T_C，T_{CW} は低機能 ACL エントリ数の増加にともない時間が急激に伸びる. これは, 低機能 ACL のアクセス時に ACL エントリのソート処理が行われるためである．ACL エントリ数を N とすると，このソート処理の処理時間は O(NlogN)となる．したがって，この領域では基本的に変換結果を書き込んだほうがよい.

ただし, たとえば NEC 社により実装・公開されている ACL 専用キャッシュ[71]などを，本提案方式と併用する場合，変換結果をメモリ上に長時間保持することができるため変換結果を書き込まないほうが有利な場合もある．有利な条件は ACL エントリ数によって異なるが, 各ファイルに対する ACL 専用キャッシュミス回数の平均値が 2～8 回以下となる．なお，ACL 専用キャッシュの詳細については 3.4.2 節で説明する.

本小節では図 3.3 の測定値に基づき設計を示したが，ハードウェア環境により処理時間は多少異なることが考えられる．たとえば，低性能 CPU を用いた場合，ACL エントリ数 35 以上の T_{CW} と T_C の傾きはより顕著となり，低性能ディスクを用いた場合，ACL エントリ数 3～5 エントリの T_C 以外の時間は, 全体的に長大化すると考えられる. しかし, ACL エントリ数 5 - 6 間，34 - 35 間に発生するアクセス制御処理時間の不連続点は，表 3.1 に示した格納形式の変更にともない，アクセス時のディスク I/O 回数が変わったことにより発生したものである．したがって，本小節で示した設計は基本的に異なるハードウェア環境にも適用できる．ただし，同一ファイルに対する平均アクセス制御回数を予測して，その予測回数によって書き込み可否を設計する場合や，XFS 以外のファイルシステムに適用する場合は，図 3.3 と同様の測定を行った後，再設計が必要となる.

3.3.3 低機能 ACL-高機能 ACL 併用方式の詳細

前小節の検討より，低機能 ACL のエントリ数に基づき，ディスク上の低機能 ACL 情報を毎回高機能 ACL 情報に変換し使い続ける場合と，初回ファイルアクセス時に変換後の高機能 ACL 情報を書き込み，次回以降はそのディスク上の高機能 ACL 情報を使用する場

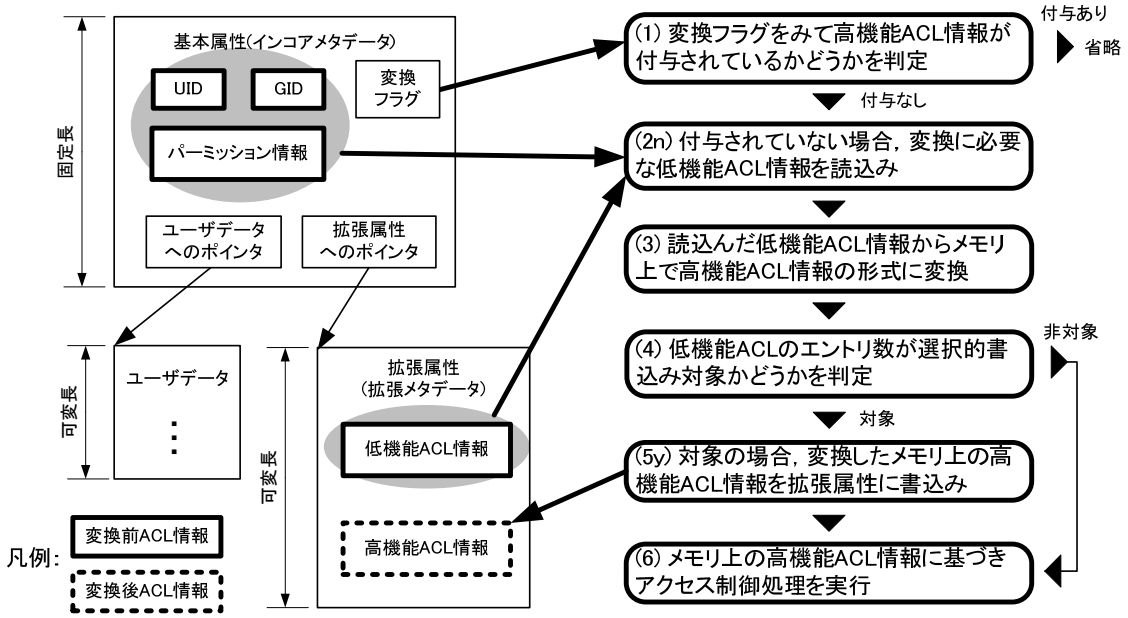

図 3.4 提案方式の処理フローチャート

合とを，使い分ける方式が考えられる．

図 3.4 に提案方式の処理フローチャートを示す．まず，変換フラグをみて高機能 ACL 情報が付与されているかどうかを判定する．付与されている場合は 3.2 節の動作と同様のため説明は省略する．付与されていない場合，変換に必要な低機能 ACL 情報を読み込む．読み込んだ低機能 ACL 情報からメモリ上で高機能 ACL 情報の形式に変換する．

従来のオンアクセス変換方式では，この後無条件にメモリ上の高機能 ACL 情報を拡張属性に書き込んでいたが，提案方式では低機能 ACL のエントリ数に基づき，高機能 ACL 情報の書き込み可否を判断する．たとえば，低機能 ACL のエントリ数が 6 以上を選択的書き込み対象とした場合，エントリ数が 3～5 の場合は書き込みを行わず，6 以上の場合は書き込みを行う．書き込みを行った場合は変換フラグを変換済みに設定する．

ただし，高機能 ACL 情報を更新するリクエストを受けた場合には，エントリ数にかかわらず，書き込みを実施する．この場合も，事前に高機能 ACL 情報に変換し，その結果を書き込んでおく方式に比べると，書き込み回数を 1 回に低減できる効果がある．

以上，本提案方式の適用により，オンアクセス ACL 変換にともなうファイルリクエストスループット低下の改善が期待できる．

なお，提案方式適用後のファイルシステムボリュームは，適用前に対してディスクレイアウト上の互換性は維持されるため，従来ファイルシステムプログラムでも認識は可能である．ただし，従来ファイルシステムプログラムで mount 処理を実行した場合，高機能

ACL 情報が付与されたファイルについては，高機能化が適用される前のアクセス制御情報に戻ってしまう．

また提案方式により，単一のファイルが複数の形式のアクセス制御情報を持つことになるため，管理が複雑となるデメリットと，使用容量が増加するデメリットがある．管理の複雑さについては，アクセス制御情報のいずれの形式も拡張属性というファイルメタデータ格納基盤を利用することで軽減できる．さらに，耐障害性の低下も防止できる．また使用容量の増加が問題になる場合は，高機能 ACL 情報付与時に，低機能 ACL 情報を削除することも可能である．

3.3.4　アクセス制御情報の移行完了に関する議論

提案方式である低機能 ACL-高機能 ACL 併用方式は，3.2.1 節(2)のオンアクセス ACL 変換による移行方式とは異なり，ある特定の ACL エントリ数のファイルは高機能 ACL 情報が付与されないままとなるため，全ファイルのアクセス制御情報の移行は永遠に完了しない．つまり提案方式は移行方式としては使えない．しかし，ファイル共有サービスのクライアントやユーザアプリケーションからは，運用再開直後からすべてのファイルのアクセス制御処理は高機能 ACL 情報により行われているように動作するため，移行されないこと自体は特に問題とならない．

仮に，一部ファイルに高機能 ACL 情報が付与されないままの状態でシステムの再起動が起こったとしても，3.3.2 節の設計はキャッシュヒットを前提としていないため，提案方式によってファイルリクエストスループットが低下することはない．

ただし，高機能 ACL 対応クライアント経由でバックアップを取り，それをリストアした場合，すべてのファイルのアクセス制御情報が高機能 ACL 形式でリストアされるため，ファイルリクエストスループットは提案方式を適用する前の状態に戻ってしまう．これを防止するためには，ファイルシステムの内部レイアウトを認識可能な専用のバックアップソフトを用いるか，ディスクブロックレベルのバックアップソフトを用いる必要がある．

3.3.5　提案方式の適用範囲

本章に記載の提案方式は特にアクセス制御の高機能化時の POSIX ACL と Windows ACL の併用に対して検討を行っているが，パーミッションや NFSv4 ACL にも適用可能である．ただし，パーミッションはエントリ数が固定値の 3 であるため，予備実験の結果をふまえると，アクセス制御情報更新オペレーションを処理するまでは低機能 ACL であるパーミッションを使い続ける方式となる．

また提案方式の考え方は，アクセス制御以外の属性情報を拡張したり追加したりするファイルシステムの高機能化にも広く適用可能である．

さらに本章では，XFS を対象ファイルシステムとして検討を行ったが，基本属性・拡張属性の参照更新時の処理時間がアクセス制御情報のサイズによって異なるファイルシステムであれば考え方を流用可能である．低機能 ACL 対応ファイルシステムにおいて，エントリ数 3 の場合とエントリ数 4 以上の場合は参照更新時の処理時間が異なることはその構造上明らかであり，ほとんどの場合適用可能であると考える．

3.4 提案方式の評価

本節では提案方式がファイルリクエストスループット低下をどの程度改善できるかを実測により評価する．まず，仮定するデータセットについて述べ，次に測定環境と測定条件を述べる．そして，測定結果を示し，その結果を考察する．

3.4.1 仮定するデータセット

仮定しなければならないデータセットとしては，ACL エントリ数分布，ファイルサイズ分布，ファイルオペレーション分布の 3 種類がある．

まず，ACL エントリ数分布として，(a) 大学研究室などの小規模組織のファイル共有サービス，(b) 企業などの大規模組織のファイル共有サービス，(c) Social Network Service (SNS)などのソーシャルサービスと連携するファイル共有サービスの 3 種類の事例を仮定した．具体的な ACL エントリ数の分布を表 3.2 に示す．(a)(b)については組織で運用されている約 100 種類の共有の実例を参考にして，分布関数とエントリ数の平均値を設定した．なお，ここでの小規模，大規模とは，あくまでも組織の大きさであって，データセットの大きさは規定していない．また，(c)については SNS の関係性を分析した論文[96]の次数分布を参考にして，分布関数とエントリ数の平均値を設定した．

次に，ファイルサイズ分布とファイルオペレーション分布としては，前章と同様ファイル共有サービスの代表的ベンチマークである SPECsfs2008[91]を利用した．本ベンチマークプログラムが提案方式の評価に適している理由は次の 2 点である．第 1 に，同プログラムではあらかじめどのファイルが何回アクセスされるかを事前に予見できず，その回数も

表 3.2 仮定する ACL エントリ数分布

想定するサービス事例	分布関数	エントリ数平均
(a) 小規模組織ファイル共有	指数分布	4
(b) 大規模組織ファイル共有	指数分布	9
(c) SNS 連携ファイル共有	べき乗分布	42

ポアソン分布に基づくばらつきを持っている．第 2 に同プログラムは，単なるディスク性能ではなく，ファイルサーバの CPU やメモリにも負荷がかかるファイルリクエストスループットを測定する．提案方式はディスクへの負荷は減る代わりに，CPU やメモリに対して負荷が増えるため，このような総合的なファイルリクエストスループットでの比較が望ましい．

3.4.2 測定環境と測定条件

測定環境は表 3.3 に示すとおりである．Linux 2.6.30 をベースにアクセス制御機能に提案方式である低機能 ACL-高機能 ACL 併用方式を実装した．本章ではユースケースとして，NFS クライアントにサービスしていた低機能 ACL 対応ファイルシステムに，高機能 ACL を適用した場合を取り上げる．つまり，Windows クライアントからも利用されるが，主に Linux などの UNIX クライアントから継続して利用されるケースを想定する．

測定条件は表 3.4 に示すとおりである．ACL エントリ数にかかわらず変換結果はすべて書き込む従来方式として条件#0，実装の容易さを重視し POSIX ACL が設定されていない場合つまり ACL エントリ数が 3 の場合のみ書き込まない方式である条件#1, 3.1 節の不等

表 3.3 測定環境

項目	台数	仕様
サーバ	1	CPU: Intel Xeon CPU 3.60GHz (Quad Core) Memory: 8,179,856KB Kernel: Linux 2.6.30.1 + XFS + 提案方式アクセス制御機能 OS distribution: Debian 5.0.4 NFS utils package version: 1.1.2
ディスク	19	108GB/LU (RAID1+0, 2D+2D) x 19
クライアント	4	CPU: Intel Xeon CPU 2.80GHz (Dual Core) Memory: 3,632,416KB Kernel: Linux 2.6.26-2-686 OS distribution: Debian 5.0.2

表 3.4 測定条件

#	書き込む ACL エントリ数	書き込まない ACL エントリ数	備考
0	すべて	なし	従来方式
1	4 以上	3	提案方式
2	6 以上	3～5	
3	6～34	3～5, 35 以上	

式が絶対不等式となる場合に書き込まない方式である条件#2, #2 に加えて書き込み時のアクセス制御処理時間が極端に悪くなる場合つまり ACL エントリ数 35 以上も書き込まない方式である条件#3 の 4 条件を測定し比較する．

上記条件に加えて ACL 専用キャッシュのある場合とない場合についても測定を行う．つまり合計 8 条件での測定を行う．ACL 専用キャッシュは，NEC 社により実装され，Web サイト[71]に公開されている．一般にファイルシステムは OS 標準のページキャッシュにより，ディスクアクセス回数を低減する．これに対し，ACL 専用キャッシュはページキャッシュよりもメモリ上のライフタイムを長くすることにより，ACL 情報のキャッシュヒット率の向上を実現している．この ACL 専用キャッシュは提案方式との親和性が良く併用可能なため，本章であわせて評価した．ACL 専用キャッシュは主に変換結果を書き込まない場合のファイルリクエストスループット改善を狙うが，あるファイルに対するアクセスの時間間隔が中程度の場合(つまりページキャッシュミスだが ACL キャッシュヒットとなる程度の時間間隔)のファイルリクエストスループット改善も期待できる．

さらに今回の測定では参考のため，高機能 ACL 適用前のファイルリクエストスループットも測定した．加えて，高機能 ACL 適用後に，各条件の書き込み対象ファイルの変換後 ACL 情報を，運用再開前にあらかじめ書き込んでおいた場合のファイルリクエストスループットも測定した．同性能は，提案方式で長期間ファイルシステム運用した場合のファイルリクエストスループットに相当する．

なお，測定は各 ACL 分布の事例，各測定条件ごとに，各々3 回実施した．

3.4.3 測定結果と考察

本小節では実際に SPECsfs2008 を実行し，測定した結果とその考察を示す．なお本測定値は，SPEC が指定する測定ルールに厳密にしたがっているわけではないため，SPEC のサイトに登録，公開されている値とは単純比較はできない．ただし，本測定のデータは同一条件にて測定しているため，相対的な違いは比較可能である．

図 3.5 に(a)小規模組織ファイル共有の結果を示す．縦軸は 3 回測定した達成負荷(Operations Per Second: OPS)つまりファイルリクエストスループットの平均であり，標準偏差を誤差範囲として示した．横軸は各測定条件であり，左からそれぞれ初期条件が，高機能 ACL 適用前，高機能 ACL 適用後運用中変換，高機能 ACL 適用後運用再開前変換となる．運用中変換と運用再開前変換の初期条件では前節で述べたとおり，8 つの測定条件がある．図中の“n”はキャッシュなし，“c”はキャッシュありを示す．たとえば“0n”は測定条件#0 キャッシュなしである．なお，本事例の ACL エントリ数分布では 35 より大きい ACL は存在しないため，測定条件#2 と#3 は同じ動作となり，ファイルリクエスト

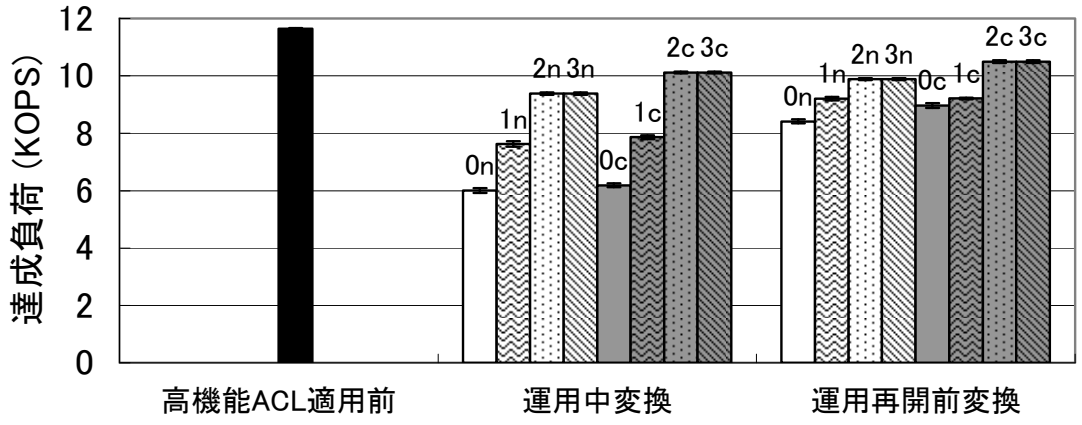

図 3.5 (a)小規模組織ファイル共有の測定結果

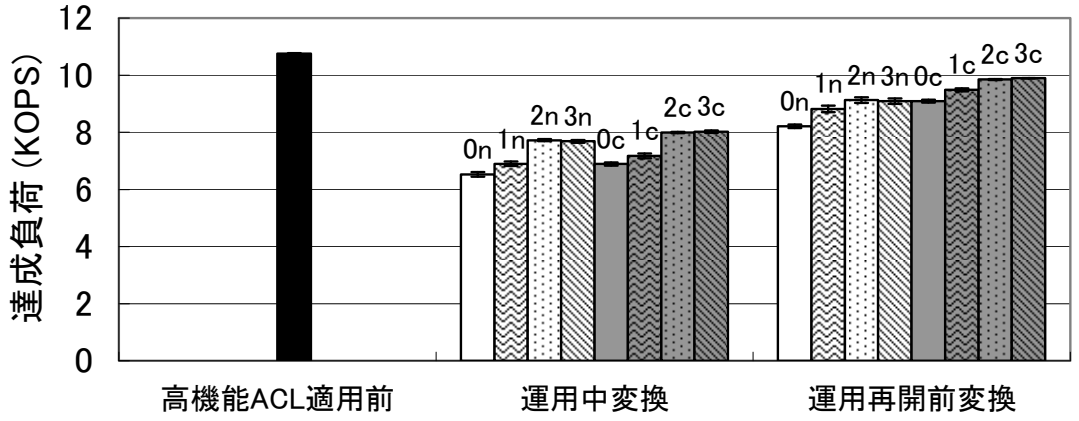

図 3.6 (b)大規模組織ファイル共有の測定結果

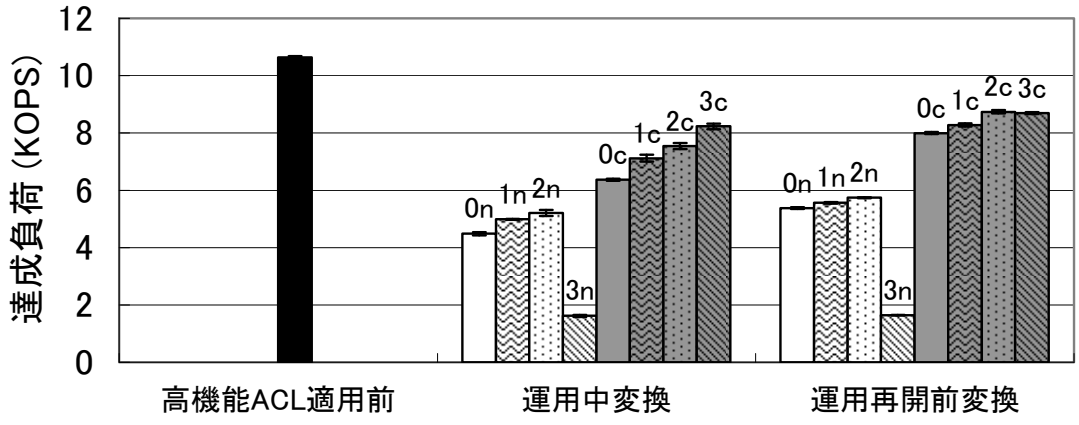

図 3.7 (c)SNS 連携ファイル共有の測定結果

スループットも同様となるため，測定条件#3 の結果は測定条件#2 の結果と同じデータを用いている．

POSIX ACL エントリが付いていない場合のみを書き込まないという単純な方式(測定条件#1)での性能向上効果は，キャッシュなしで約 1.6KOPS(27%)，キャッシュありで 1.7KOPS(27%)であった．また，測定条件#2, #3 ではキャッシュなしで約 3.4KOPS(56%)，キャッシュありで 3.9KOPS(64%)性能を向上できた．平均の差の検定を実施したところ，99.9%の信頼度で測定条件#0，#1，#2-3 の平均値の有意差が確認できた．高機能 ACL 適用前と比較すると，従来方式である測定条件#0 キャッシュなしでは 48%の性能低下だったのに対し，測定条件#2 キャッシュなしでは 19%の性能低下に抑えることができた．運用再開前変換は運用中変換に対し，全体的に性能が改善する．また性能差は小さくなるものの，提案方式の優位性は変わらない．

次に図 3.6 に，(b)大規模組織ファイル共有の結果を示す．前事例と同様に，平均の差の検定を実施したところ，95%の信頼度で測定条件#0，#1，#2 の平均値の有意差が確認できたが，測定条件#2，#3 の間では有意差は確認できなかった．これは 35 より大きい ACL エントリ数のファイルの割合が極わずかであるためと考える．本事例では提案方式の適用によりキャッシュなしで最大で約 1.2KOPS(18%)，キャッシュありで最大で約 1.1KOPS(16%)性能が向上した．高機能 ACL 適用前と比較すると，キャッシュなしで性能低下が 39%から 29%に改善した．運用再開前変換の傾向は事例(a)と同様である．

最後に図 3.7 に，(c)SNS 連携ファイル共有の結果を示す．同様に平均の差の検定を実施したところ，全測定条件で 95%の信頼度で平均値の有意差が確認できた．本事例では大きい ACL エントリ数のファイルの割合が多いため，キャッシュの効果が非常によく出ている．キャッシュなしで最も効果が大きかった測定条件#2 の性能向上効果は，0.7KOPS(16%)程度であるが，キャッシュありで最も効果が大きかった測定条件#3 の性能向上効果は，約 1.9KOPS(29%)となった．加えてキャッシュ適用自身の性能向上効果が 3KOPS 程度あるため，キャッシュ＋提案方式の組みあわせでは約 3.7KOPS(83%)の大幅な向上となる．なお，測定条件#3 キャッシュなしは大幅に性能が低下するが，これは 3.3.2 節で述べた ACL エントリのソート処理に起因するものと考える．高機能 ACL 適用前と比較すると，キャッシュなしで性能低下が 58%から 51%に改善した．運用再開前変換の傾向は事例(a)(b)と同様である．

これらの性能向上の要因を，分析ツール等を用いて分析した．提案方式によって性能向上する要素は大きく 2 つある．第 1 は ACL 情報のサイズを小さいままにできることによる，総ディスクアクセス量の低減である．第 2 はアクセス制御処理時間の短縮による，各

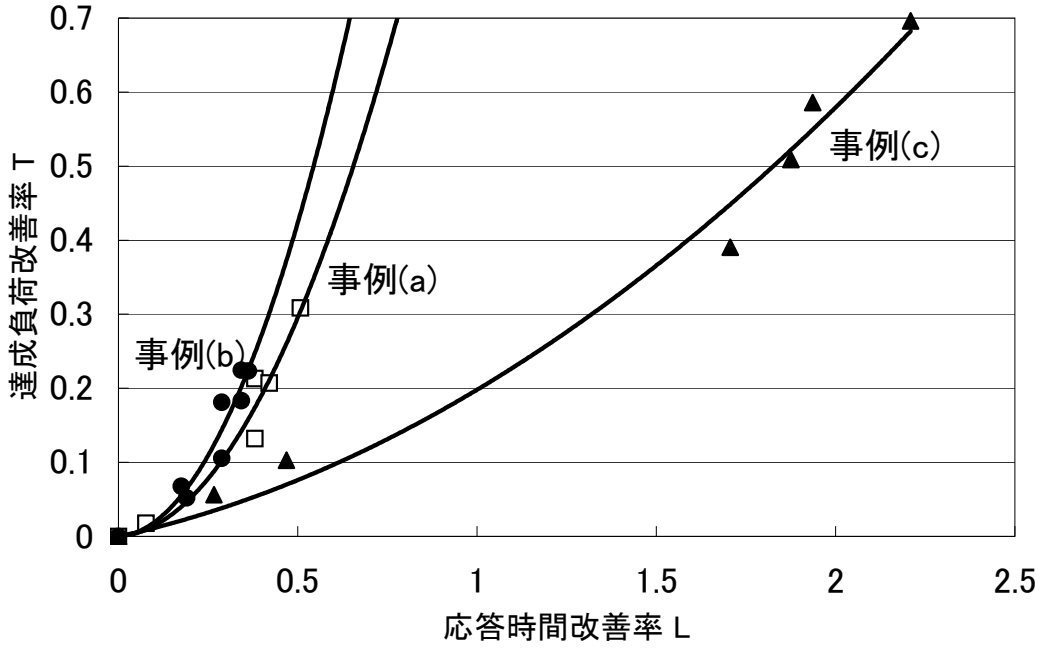

図 3.8 平均応答時間と達成負荷の改善率の相関

アクセスリクエストの応答時間の短縮である.

第 1 の要素である総ディスクアクセス量の低減の割合を分析したところ，1～2%程度であった．性能向上効果は数 10%程度あるためこの要素だけでは説明ができない.

第 2 の要素であるアクセスリクエストの平均応答時間を，同一負荷を投入して比較したところ，数 ms 短縮していた．平均応答時間の短縮率と達成負荷の向上率の相関関係を分析したところ，強い相関が見られることが分かった．ここで平均応答時間の短縮率(以降，L)とは「従来方式平均応答時間／提案方式平均応答時間－1」であり，達成負荷の向上率(以降，T)とは「提案方式達成負荷／従来方式達成負荷－1」である．L と T の相関グラフを図 3.8 に示す．L と T の関係はシステムに強く依存するため一般化が困難であるが，本評価システムにおいては，横軸を L，縦軸を T とした場合に 2 次関数で近似できた．ただしその近似関数の係数は各事例により異なる．以上の分析により，性能改善の要因はこの第 2 の要素によるものが大きいことが推定できる.

提案方式のファイルシステム運用停止時間は，基本的にファイルシステム数に依存するが，本測定環境では測定条件にかかわらず 4 秒程度であった．仮にファイルシステム数が 1000 あったとしても，4 分程度となり十分実用的である．これに対し，運用再開前変換方式のファイルシステム運用停止時間は基本的にファイル数や ACL サイズに依存する．本測定環境では測定条件によって異なるが，数 100～1000 秒程度であった．仮に 200TB の

データセットを想定した場合，その運用停止時間は 1〜2 日程度となる．したがってデータセットが小中規模であれば，運用再開前変換は有用である．これに対し，数 10TB 以上の大規模データセットに適用する場合は提案方式が有用である．

本節で示した性能改善効果は，ハードウェア環境の相違により，多少異なることがある．いずれにせよ 3.3.2 節で示した方針に基づき設計すれば，効果の差があるにせよ，ハードウェア環境によらず性能の改善は可能である．また，本節で用いたワークロード以外では，性能改善効果は異なることがある．その傾向は 2 章で示したものと同様である．

2 章で提案した階層型アクセス制御方式と，本提案方式を組みあわせた場合の，性能改善効果について考察する．両方式とも，拡張属性アクセス処理を省力化する効果により性能を改善しているため，基本的には性能改善効果が重複する．両方式を組みあわせた場合の最大の性能改善効果は，2 章での拡張属性アクセス割合が 0%となった場合に相当する．したがって，高機能化前に対して最大で 10%前後の性能低下まで改善可能である．

以上全体をまとめると，提案方式はどの事例に対しても有効であることが確認できた．各事例を比較すると，キャッシュなしのケースでは(a)(b)(c)の順で効果が高く，キャッシュ併用のケースでは(c)(a)(b)の順で効果が高かった．具体的には，キャッシュなしの場合最大で 3.4KOPS(56%)程度，キャッシュありの場合キャッシュの効果とあわせて最大で 3.7KOPS(83%)程度，の性能向上が可能なことを明らかにした．

3.5 結言

本章では，運用しているファイルシステムに高機能なアクセス制御機能を新たに適用する際の課題であるファイルリクエストスループット低下と長時間のファイルシステム運用停止を改善する，低機能 ACL-高機能 ACL 併用方式を提案した．初回アクセス時に必ず変換結果を書き込む従来方式に対して，性能がどの程度向上するかを，提案方式を実装したシステムで評価した．小規模組織ファイル共有，大規模組織ファイル共有，SNS 連携ファイル共有のいずれのケースも約 1〜5 割性能を向上した．さらにアクセス制御情報専用キャッシュと提案方式を併用した場合，SNS 連携ファイルサーバのケースで約 8 割性能を向上した．また運用停止時間も 4 秒程度に抑えられることを確認した．これらの評価により，提案方式の有効性が確認できた．なお，3.2.1 節で述べたオンアクセス変換機能と 3.3 節で述べた低機能 ACL-高機能 ACL 併用方式の一部は日立ファイルストレージで実用化済みである．

今後の課題として，次の 2 点が挙げられる．第 1 の課題は，本章の評価で性能改善への効果があわせて確認できた ACL 専用キャッシュの有効性をさらに追及することである．

たとえば，本章で示した各事例において，適切なキャッシュサイズやキャッシュアルゴリズムを検討することが考えられる．第 2 の課題は，本章の考え方を，同様のファイルシステムの機能拡張に応用することである．たとえば，暗号，圧縮，重複排除などへの応用が考えられる．

第 4 章

ファイル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

4.1 緒言

本章では，複数ファイルを同時に同期書き込みで作成する場合の，ファイルフラグメントの発生を防止しファイルデータスループットの低下を改善するサイズ調整プリアロケーション方式について述べる．

ファイルシステムに格納されるコンテンツは，テキストのような小サイズ(数 KB〜数 10KB 程度)のファイル, ビジネスドキュメントのような中サイズ(数 MB 程度)のファイル, ストリーミングデータのような大サイズ(数 10MB〜数 GB 程度)のファイルが考えられる [97].

複数のサイズのファイルが混在するファイルシステムボリュームでは，ファイルデータの格納領域やファイルシステムの空き領域が断片化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が肥大化し，ディスク容量・メモリなどのリソースを圧迫する．また，読み込み，書き込みのファイルデータスループットが低下することや，ファイルの削除時間が増加することがある [98].

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り当て単位サイズ(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[99][100].

外部ディスクフラグメントは，ファイルシステムの空き領域が大幅に不足している状況下で，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いる．本章では，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ファイルストレージに加えて，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，多人数による同時格納可能な動画共有サーバがある．これらのファイルストレージ，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．

本章ではこの同期書き込み時において効果的に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．提案方式は，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する.ここで伸長中とは，追記書き込みリクエストにより，ファイルサイズが今後増加する可能性の高いファイルの状態を意味する．

本章の以降の構成は次のとおりである．4.2 節では，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．4.3 節では，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．4.4 節では，提案方式を実装したシステム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 4.5 節で，本章のまとめを述べる．

4.2　従来のフラグメント防止技術とその課題

本節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，割り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．

4.2.1 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[77]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り当ては，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．

4.2.2 割り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り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り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り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り当てブロックを選択する方式である[101]–[103]．小サイズファイル用の数 KB 程度の小サイズ割り当てブロックと，通常サイズファイル用の数 10KB 程度の中サイズ割り当てブロックの 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り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り当てブロックの配分の決定がより難しくなってしまう．

4.2.3 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[78]．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 4.2.2 節の方式と異なり，割り当てブロック単位サイズは一定とし，事前に連

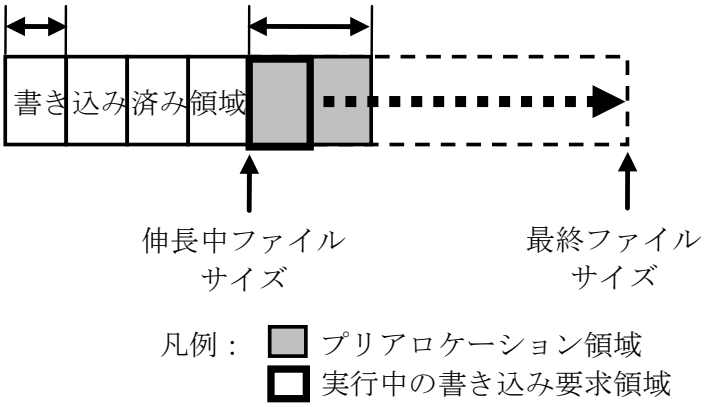

図 4.1 作成中のファイルの状態

続した領域の複数の割り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，割り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，未使用の割り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，10KB～数 10KB 程度の一定サイズの領域を割り当てる方式であり，XFS，EXT2，EXT3，EXT4 に採用されている[104]–[106]．図 4.1 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[107]．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，ファイルストレージ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．

4.3　サイズ調整プリアロケーション方式

本節では，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．

4.3.1　従来方式の課題に対する解決のアプローチ

4.2 節で述べたとおり，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，割り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り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，割り当てサイズ，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，ファイルデータスループットと削除時間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．

4.3.2　プリアロケーションサイズの調整方法に関する検討

本小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，書き込みのファイルデータスループット，1 ファイルの削除時間がそれぞ

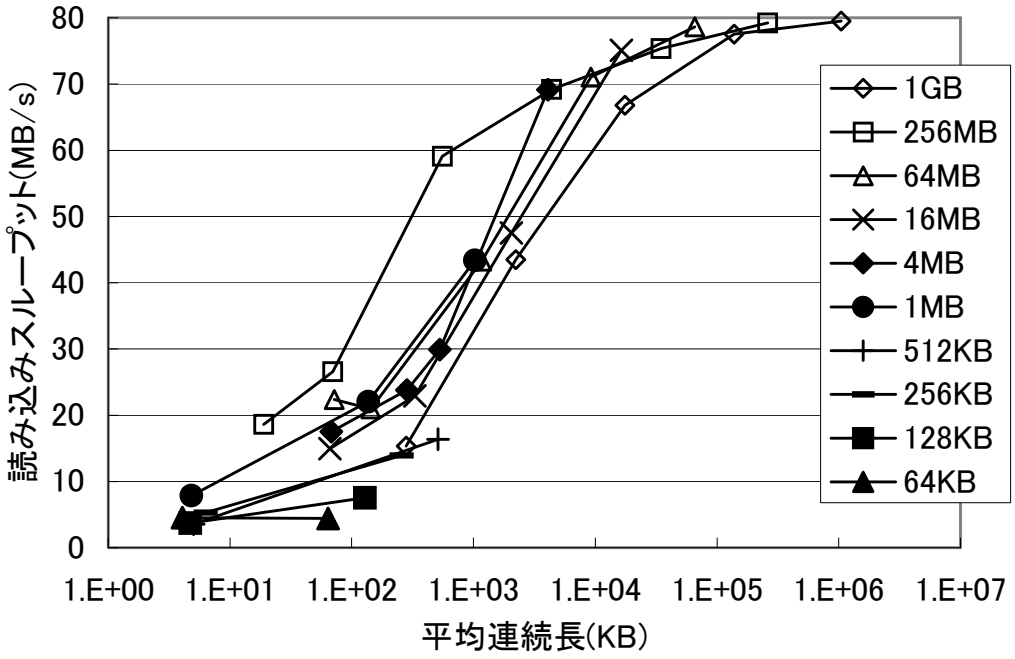

図 4.2 読み込みファイルデータスループットと平均連続長の関係

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．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 64KB)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．一般的なファイルサイズの範囲を参考にして，測定を行うファイルサイズは 64KB～1GB の 10 種類とした．ファイルサイズ 64KB～16MB の 7 種類は 512 個を同時に，64MB～1GB の 3 種類は 16 個を同時に，同期書き込みで作成した.

読み込みファイルデータスループットとファイルの平均連続長の相関関係を図 4.2 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 100KB 程度までは性能に大きな改善は見られない．平均連続長 1MB～10MB 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しない.

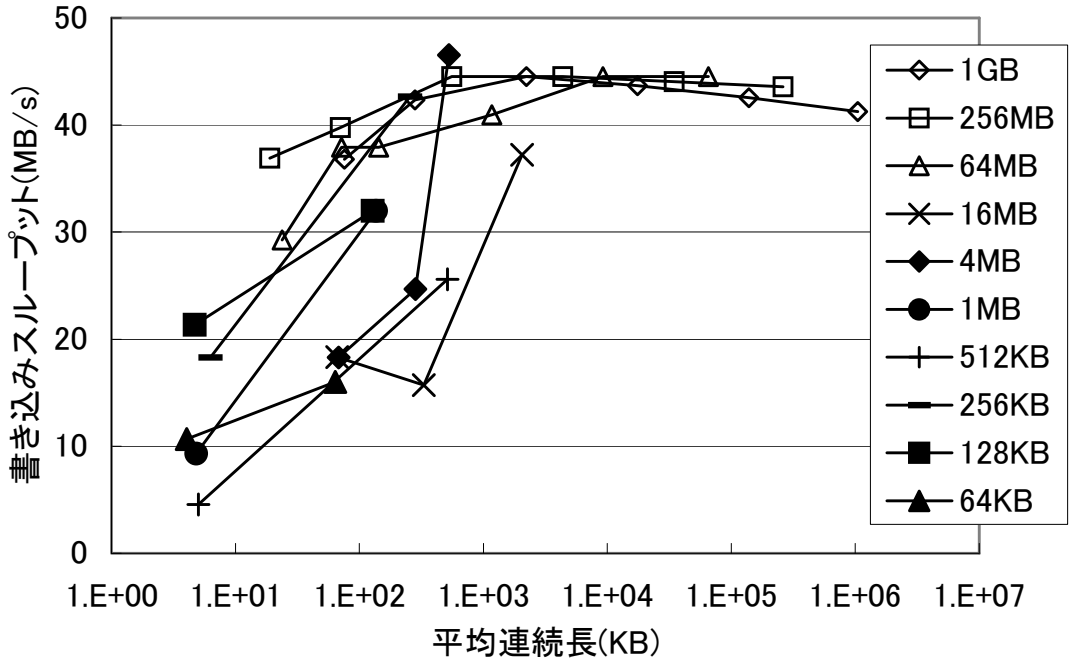

図 4.3 書き込みファイルデータスループットと平均連続長の関係

書き込みファイルデータスループットとファイルの平均連続長の相関関係を図 4.3 に示す．読み込みよりは早く立ち上がり，平均連続長 1MB 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 4.4 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，書き込みのファイルデータスループットを重視する場合は，伸長中ファイルサイズが数 100KB 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 10MB 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．

削除時間を重視する場合は，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

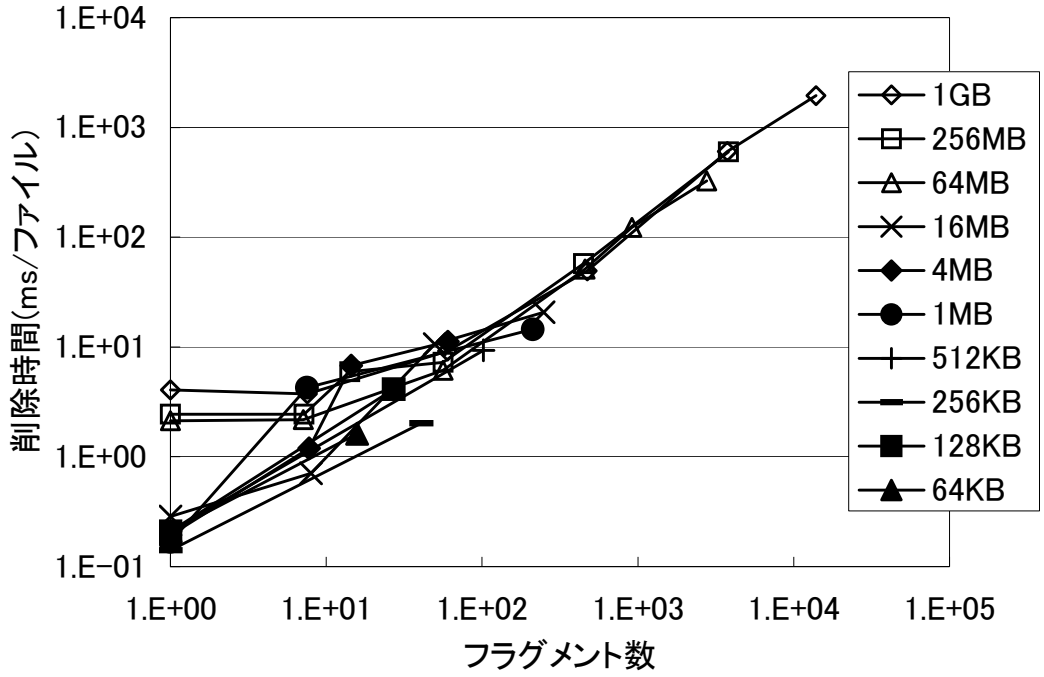

図 4.4 削除時間とフラグメント数の関係

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．

提案方式の性能改善効果は，ハードウェア構成によって多少異なる．具体的にはストレージデバイスのランダムアクセス性能とシーケンシャルアクセス性能に強く依存する．たとえば，SSD のようなランダムアクセス性能が高いストレージデバイスでは，提案方式による性能改善効果は小さくなる．したがって，本小節で示したような予備実験を行った後に，各ストレージデバイスに適したプリアロケーションサイズ関数を選択すべきである．

4.3.3 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[108]や ZFS[90]など，Copy on Write (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，EXT3，EXT4 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 4.2.1 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 10MB〜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．

4.3.4 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 を用いる理由は，2.6.1 節に記載のとおりである．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 64KB の固定長である．

図 4.5 に，提案方式の模擬コードを示す．XFS 1.2 の書き込み処理を実行する関数は `linux/fs/xfs/pagebuf/page_buf_io.c` の `pagebuf_file_write()` である．この書き込み処理実行関数のプリアロケーションサイズ設定部を模擬コードのように変更した．`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

```
{
   ...
   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 4.5 提案方式の模擬コード

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あわない場合は，ブロックサイズにあ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り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．

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，open したファイルに対して書き込みが発生してから close されるまでの間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．

提案方式は，ファイルシステムのディスクレイアウトはまったく変更しないため，従来のファイルシステムプログラムとの互換性は完全に保たれる．

4.4 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，書き込みファイルデータスループット，読み込みファイルデータスループット，削除時間，内部ディスクフラグメントの観点で，評価を行う．

4.4.1 測定環境と測定条件

測定システムとして，2.8GHz Xeon × 2，メモリ 3.2GB の Intel Architecture (IA)32 機を用いた．数百人規模以上の多数ユーザによる同時書き込みを想定モデルとし，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た．その際の，フラグメント数，書き込みファイルデータスループット，読み込みファイルデータスループット，削除時間を測定する．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[109]に基づく．残りの 30 個は将来的に平均ファイルサイズ，最大ファイルサイズが増加することを想定し，64MB，256MB，1GB 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 4.1 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 4.2 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．4.3.2 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，XFS オリジナルの方式であるシステムプリアロケーションサイズ 64KB 固定(#1),

表 4.1 作成ファイルのサイズ分布

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 4.2 使用するプリアロケーションサイズ決定関数

#	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

さらに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 16MB 固定(#2)も測定する．16MB を選択した理由は，予備実験の結果から平均連続長が 16MB 程度あれば性能が十分飽和するためである．

また，本章の評価で#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あわせ，#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，ブロック割り当てだけではなく，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 個の平均値とあわせて，本方式の効果がより顕著に現れる 64MB，256MB，1GB の大サイズファイル 30 個を除いた 500 個のファイルのみの平均値もあわせて分析，考察する．

4.4.2 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 4.6 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 XFS オリジナル(64KB 固定)の条件よりもフラグメント数が改善した. 特に 530 ファイル平均では，約 30～110 倍と大幅に改善する．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．

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．

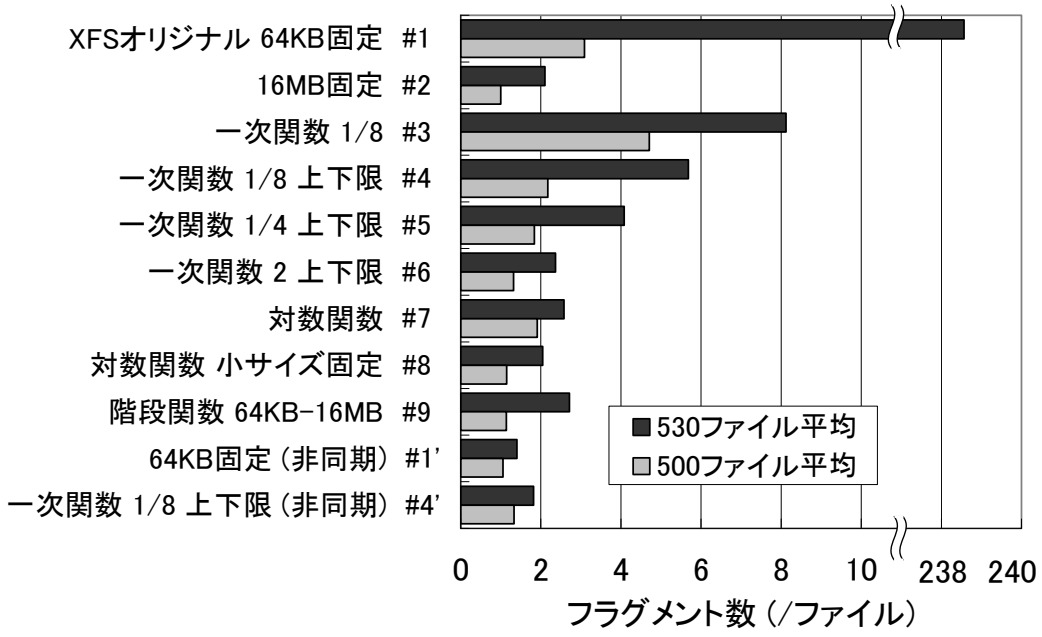

図 4.6 各測定条件におけるフラグメント数

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．

4.4.3　読み込み・書き込みスループットと削除時間の評価

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ファイルデータスループットを図 4.7 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を改善した．提案方式の条件間の大きなスループットの違いはないが，530 ファイル平均では，#7 対数関数が最もスループットを改善する結果となった．非同期書き込み後の読み込みファイルデータスループット(#1')と比較すると，従来方式では 59%低下していたのに対し，提案方式#7 の適用でその低下を 2.6%にでき，大幅に改善した．一次関数は傾きが強ければ強いほどスループットを改善する．また上下限値を設けてもスループットにはそれほど大きいインパクトはないことが分かった．提案方式の非同期書き込み後の読み込みファイルデータスループット(#4')は，提案方式(#1')に比べて若干低下する結果となった．

500 ファイル平均においてもスループットは改善した．500 ファイル平均では，#6 一次関数傾き 2 の上下限値ありが最もスループットを改善する結果となった．非同期書き込み後の読み込みファイルデータスループット(#1')と比較すると，従来方式では 39%低下していたのに対し，提案方式#6 の適用でその低下を 6.7%にでき，大幅に改善した．530 ファイルと比較した場合，スループット改善の絶対量は少ない．しかし，これは 500 ファイル

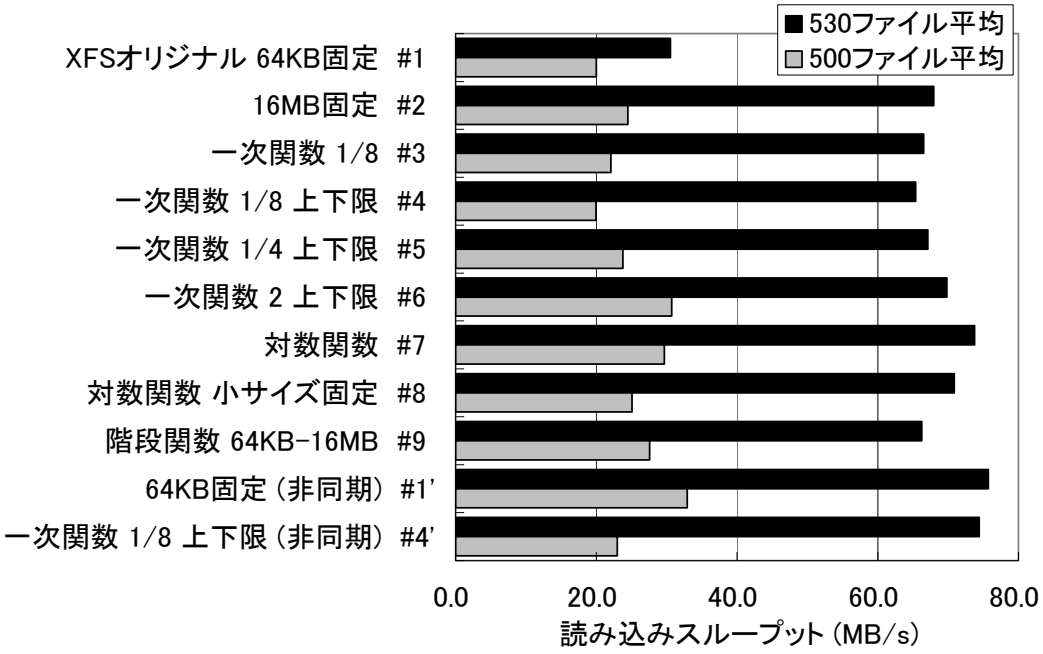

図 4.7 各測定条件における読み込みファイルデータスループット

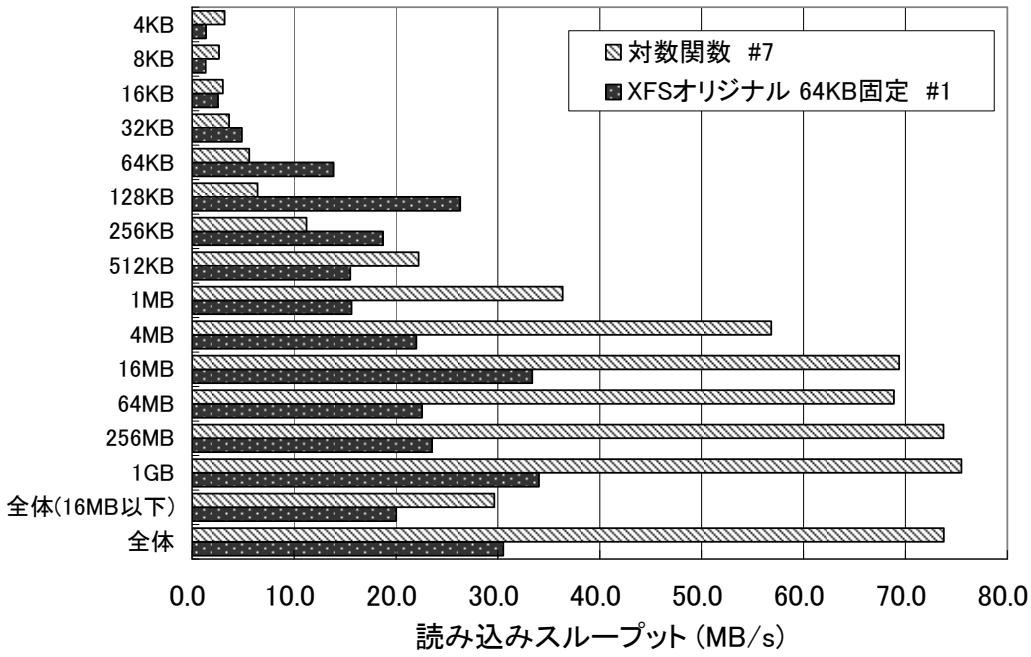

図 4.8 ファイルサイズごとの読み込みファイルデータスループット

が小サイズファイルで構成されているため，非フラグメント時(#1')でもそもそもそれほどスループットが大きくないことに起因する．

ファイルサイズごとの読み込みファイルデータスループットについて，XFS オリジナルの方式と対数関数とを比較した結果を図 4.8 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スループットが改善できている．

次に，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ファイルデータスループットを図 4.9 に示す．読み込みスループットほどではないが，書き込みスループットも提案方式により大幅に改善した．提案方式の条件間のスループットの大きな違いはない．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スループットが向上する結果となった．

最後に，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 4.10 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 を除いて削除時間を改善した．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と推定できる．これは，ファイルの断片サイズが不揃いの場

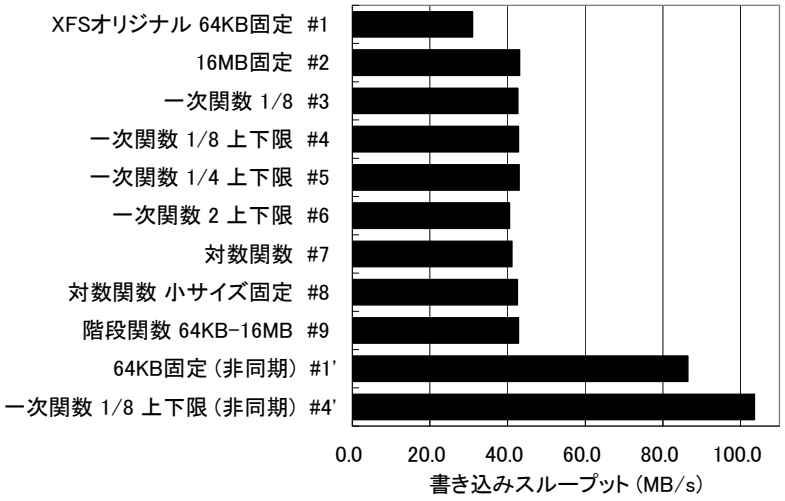

図 4.9 各測定条件における書き込みファイルデータスループット

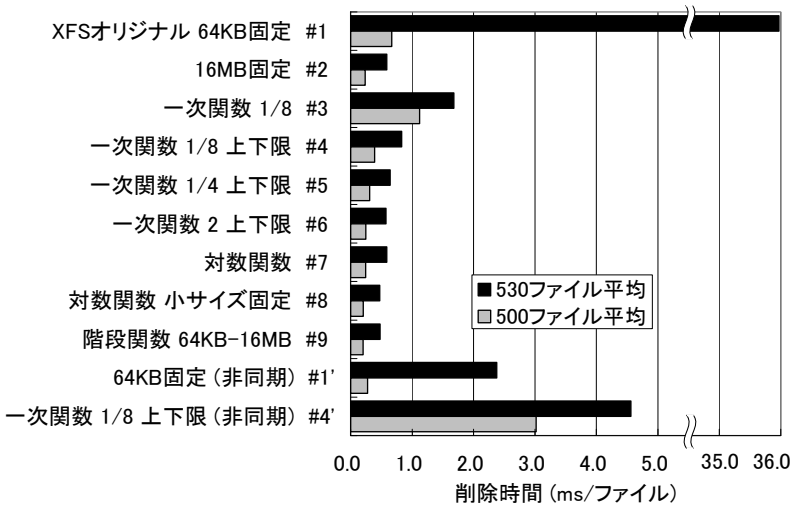

図 4.10 各測定条件における削除時間

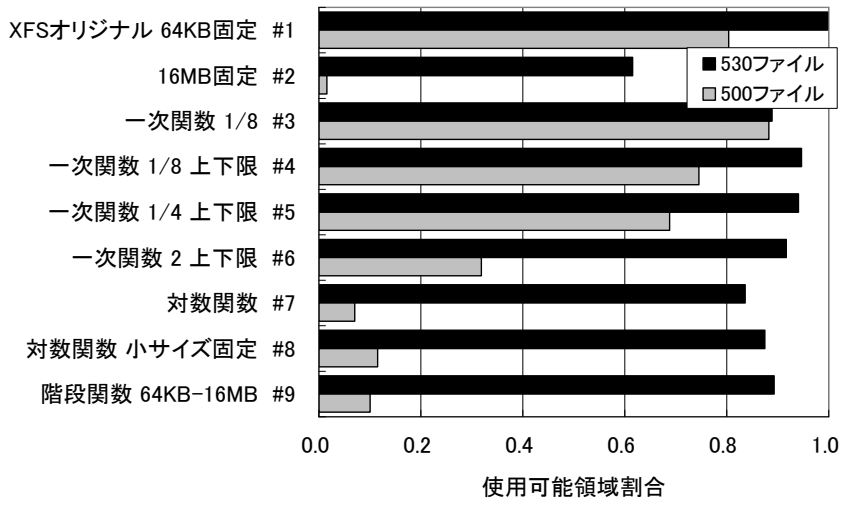

図 4.11 各測定条件における使用可能領域割合

合空き領域管理テーブルのサイズが大きくなり，テーブル操作の処理コストが大きくなるためと考えられる．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．

4.4.4 内部ディスクフラグメントの評価

本小節では，提案方式が内部ディスクフラグメントに与える影響を評価する．各条件において最悪ケースでの使用可能割合を図 4.11 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．この図の横軸の 1.0 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を意味し，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 16MB 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 16MB 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．

4.4.5 異なるファイルサイズ分布での性能予測手順

本小節では 4.4.1 節で仮定したファイルサイズ分布以外の場合の予測性能について考察する．定性的には，大サイズファイルの占める割合が大きくなればなるほど，提案方式の性能改善効果は大きくなる．逆に，小サイズファイルの占める割合が大きくなればなるほど，提案方式の性能改善効果は小さくなるが，適切な関数を選べば，性能が悪化することはない．

定量的に性能を予測するには，次の手順が考えられる．第 1 に，提案方式適用後のフラグメント数の予測最悪値を算出する．フラグメント数の予測最悪値は，設定したプリアロケーションサイズ関数に基づき行われるプリアロケーションの回数であるため，関数とファイルサイズが決まれば自動的に計算可能である．第 2 に，フラグメント数の予測最悪値

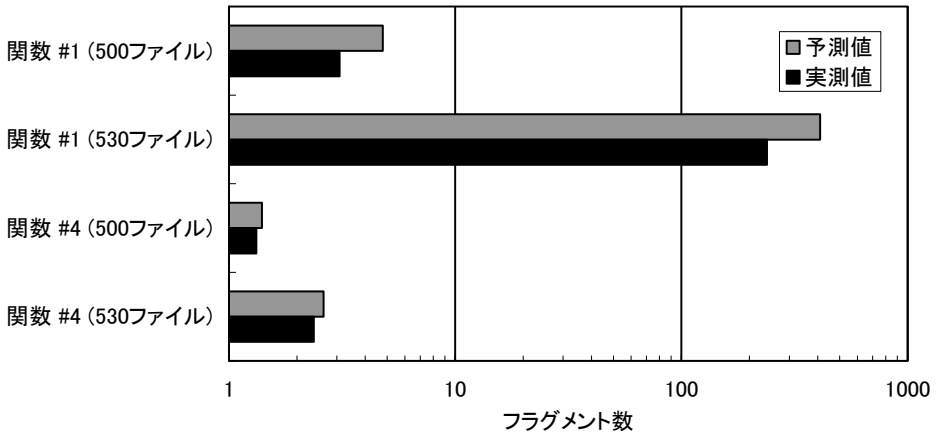

図 4.12 フラグメント数の予測値・実測値比較

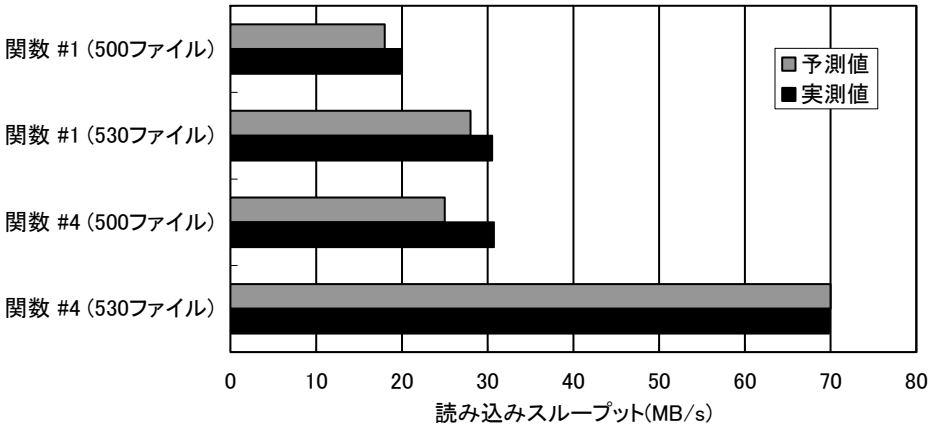

図 4.13 読み込みスループットの予測値・実測値比較

からファイルの平均連続長の予測最悪値を算出する．第 3 に，予備実験の読み込みファイルデータスループット，書き込みファイルデータスループット，削除時間の結果から，その平均連続長に対応する性能値を抽出する．

この予測手順が有効なのは，フラグメント数の予測最悪値と実測値がよく一致することと，予備実験の性能からの予測値と，性能の実測値がよく一致することの 2 点が重要である．そこで，今回の実測データにより検証を行う．

図 4.12 に，フラグメント数の予測最悪値と実測値の比較結果を示す．提案方式適用前の#1 と提案方式適用後の#4 を比較する．提案方式適用前(#1)は，フラグメント数の実測値は予測最悪値と比べるとオーダは一致しているが，少し小さい値になっている．これは，プリアロケーションサイズが 64KB と小さいため，連続領域が他のファイルから使われていない確率が上がり，連続領域にプリアロケーションが行われたためである．提案方式適用

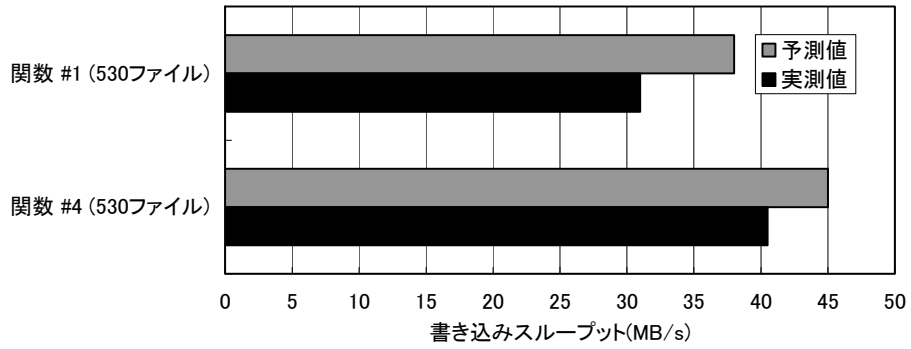

図 4.14 書き込みスループットの予測値・実測値比較

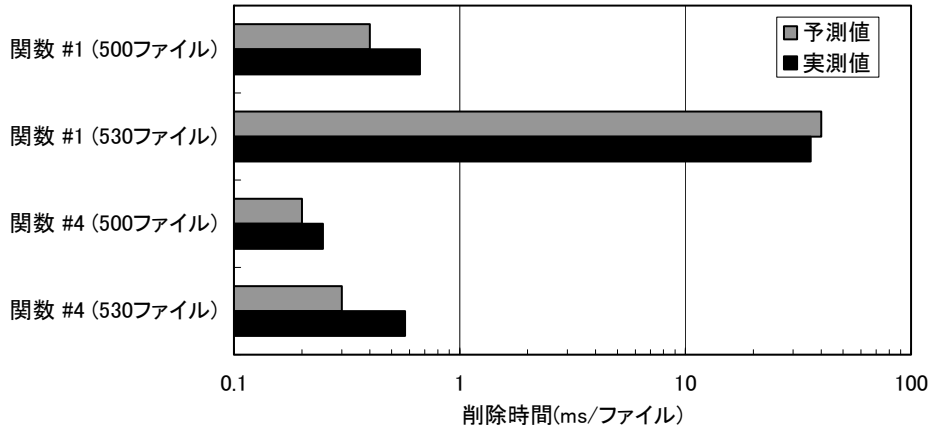

図 4.15 削除時間の予測値・実測値比較

後(#4)は，フラグメント数の実測値と予測最悪値は非常によく一致している．

次に図 4.13 に, フラグメント数の予測最悪値から算出した読み込みスループットの予測値と実測値の比較結果を示す．いずれの条件でも予測値と実測値は基本的によく一致することが確認できた．ただし，関数#4 の 500 ファイルでは，2 割程度の誤差が見られる．この理由としては，平均連続長が 1MB 以下の領域において，読み込みスループットの予備実験の結果のばらつきが大きくなっているためと考える．

次に図 4.14 に, フラグメント数の予測最悪値から算出した書き込みスループットの予測値と実測値の比較結果を示す．予測値と実測値は基本的にはよく一致しているが，予測値は若干大きめの値となっている．この理由も読み込みスループットの予測誤差と同様と考える．つまり，書き込みスループットの予備実験の結果も，平均連続長が 1MB 以下の領域においてばらつきが大きくなっているためである．

最後に図 4.15 に, フラグメント数の予測最悪値から算出した削除時間の予測値と実測値の比較結果を示す．予測値と実測値は基本的にはよく一致しているが，削除時間が短い領

域，つまり，フラグメント数が小さい場合に少し差が大きくなる．この理由も，読み込み，書き込みスループットと同様に，削除時間の予備実験の結果は，フラグメント数が 100 以下の領域において，ばらつきが大きくなっているためである．

以上の検証により，フラグメント数の予測最悪値の大小と，性能の高低には強い相関が見られることが分かる．つまりフラグメント数の予測最悪値により，性能の定量的な予測はある程度可能である．

また，フラグメント数の予想最悪値と予備実験のデータから考えると，1MB 以上のファイルサイズのファイルを複数個含んでいれば，任意のファイルサイズ分布のファイルセットにおいて提案方式は性能改善効果があることが分かる．

4.4.6 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，書き込みのファイルデータスループットと，削除時間を改善できた．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，システムプリアロケーションサイズの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．

提案方式の中で読み込み，書き込みのファイルデータスループットや削除時間を重視するならば，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，読み込みファイルデータスループットと削除時間も若干悪化するが，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．

また，2 章の階層型アクセス制御方式，3 章の低機能 ACL-高機能 ACL 併用方式と本章のサイズ調整プリアロケーション方式を組みあわせた場合の性能改善効果について考察する．2 章，3 章の方式は小サイズファイルに対して効果が高く，本章の方式は前述したように大サイズファイルに対して効果が高い．たとえばファイルサイズが 1MB 以下のファイル分布とアクセスパターン分布は SPECsfs2008 相当で，ファイルサイズが 1MB より大きいファイル分布が本章のファイルサイズ分布，アクセスパターンがシーケンシャルリードのワークロードを仮定する．このワークロードにおいて，非同期書き込みであらかじめファイル作成し，低機能 ACL を適用した場合に対して，同期書き込みであらかじめファ

イル作成し，高機能 ACL を適用した場合の性能を比較する．全提案方式の適用により，ファイルリクエストスループット換算で性能低下を 24%から 8.5%程度に改善できる．

4.5 結言

本章では，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．提案方式の一部は，日立ファイルストレージで実用化済みである．

今後の課題は，外部ディスクフラグメントも考慮に入れたフラグメント改善方式の検討である．様々なサイズのファイルの作成削除が繰り替えされるファイルシステムにおいて外部ディスクフラグメントを予防することは非常に困難である．したがって，外部ディスクフラグメントが発生した場合，デフラグツールを実行するのが一般的である．しかし，このデフラグツールの実行はファイルアクセス性能を低下させてしまう．ファイルアクセス性能の低下を最小化するデフラグツールの検討が必要である．

第 5 章

結論

5.1 本研究のまとめ

本論文では，ファイルシステムのアクセス性能の改善について検討し，拡張属性アクセス処理回数を低減する階層型アクセス制御方式，複数の ACL 形式を選択的に使用する低機能 ACL-高機能 ACL 併用方式，ファイル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した. 本論文では, これらの研究成果を以下の 4 章に分けて報告した.

第 1 章では，ファイルシステムの概要と重要性について述べ，そのファイルアクセス性能改善の課題である背景を述べ，各ケースにおける課題を整理した．そして，それらの課題に対する関連研究について検討し，研究方針を説明した.

第 2 章では, 拡張属性アクセス処理回数を低減する階層型アクセス制御方式を提案した. 提案方式は，アクセス制御情報の一部情報を基本属性の領域に格納することによって，ファイルリクエストスループットを改善する．どのような情報をその基本属性の領域に格納すべきか検討するため, 基本属性の情報のみでアクセス判定可能な確率を評価指標として, 高頻度使用アクセスマスクビット，高頻度設定アクセスマスクパターン，アクセスマスクビット論理積の 3 格納方式を比較した．その結果，3bit の格納領域の場合には高頻度設定アクセスマスクパターン，9bit の格納領域の場合には高頻度使用アクセスマスクビットの格納方式が最も適していることを明らかにした．提案方式を実装したシステムを用いて，アクセス制御高機能化前に対するアクセス制御高機能化後のファイルリクエストスループットを 4 事例の容量重み付け平均で評価したところ，提案方式適用前は約 2 割ファイルリクエストスループットが低下していたが，提案方式適用によりその低下率を約 1 割に改善できることを明らかにした.

第 3 章では，複数の ACL 形式を選択的に使用する低機能 ACL-高機能 ACL 併用方式を提案した．提案方式は，アクセス制御情報のサイズに応じて，低機能 ACL の変換後結果である高機能 ACL を書き込む場合と，変換後結果は書き込まずアクセス制御処理時に毎回低機能 ACL から高機能 ACL に変換する場合を使い分ける．初回アクセス時に必ず変換

結果を書き込む方式に対して,ファイルリクエストスループットがどの程度向上するかを,提案方式を実装したシステムで評価した．小規模組織ファイルサーバ，大規模組織ファイルサーバ，SNS 連携ファイルサーバのいずれのケースも約 1〜5 割ファイルリクエストスループットを向上した．また運用停止時間も 4 秒程度に抑えられることを確認した．これらの評価により，提案方式の有効性を確認した．

第 4 章では，ファイル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した.提案方式は,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，プリアロケーションサイズを適用する．提案方式を実装したシステムを用いて，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．またファイルフラグメントを改善することにより，読み込みのファイルデータスループットを 59%の低下から 2.6%の低下に改善できるなど，読み込み，書き込みのファイルデータスループットをそれぞれ改善できることを確認した．

本論文の研究成果を活用することにより，幅広い条件でファイルアクセス性能の低下を抑えることができ,より使いやすく信頼性の高いファイルシステムを実現できると考える．また本論文で述べた考え方は，非構造化データを格納する他のシステムにも応用が可能である．たとえばそのようなシステムとしては，Key-Value 型ストレージシステム，オブジェクトストレージシステム，Peer-to-Peer (P2P)型分散ストレージシステムなどが挙げられる．

5.2 今後の課題

最後に，本研究の今後の課題について述べる．

(1) ファイルシステムの応用機能がアクセス性能に与える影響に関する検討

本論文では 2 章および 3 章において，ファイルシステムの基盤機能の 1 つであるアクセス制御機能に関するファイルリクエストスループットの改善を取り上げた．ファイルシステムは近年多機能化が進んでおり，基盤機能以外の応用機能についても，同様の検討が必要であると考える．そのような応用機能としては，データ暗号，スナップショット，バックアップ，重複排除，符合圧縮などが挙げられる．たとえばブロックレベルの重複排除では，重複排除前はデータが連続位置に配置されていたが，重複排除後にはそのデータが非連続になることがある．この場合 4 章で述べた原因と同じくフラグメントによりアクセス性能が低下する．このような応用機能による，アクセス性能低下改善を検討する必要があ

る．

(2) フラグメント以外の過酷な状況や環境がアクセス性能に与える影響に関する検討

本論文では 4 章において，ファイルフラグメントに関するファイルデータスループットの改善を取り上げた．これ以外の過酷な状況や環境としては，たとえばファイルストレージへのクライアント同時接続数が増えることにより，セッション管理情報のためにファイル共有サービスのサーバプログラムがメモリを大量に消費する場合がある．メモリが大量に消費されると，メタデータやファイルデータのカーネル空間メモリ上のキャッシュが追い出されたり，ユーザ空間メモリがスワップアウトしてしまう．これにより，キャッシュミスや不要な I/O が発生しアクセス性能が低下する．このような，メモリを大量に消費する状況や環境でのアクセス性能改善を検討する必要がある．

謝辞

本研究の全過程を通じて，終始懇切丁寧なるご指導とご鞭撻，格別のご配慮を賜りました大阪大学大学院情報科学研究科マルチメディア工学専攻 薦田憲久教授に深く感謝申し上げます.

本研究をまとめるにあたり，貴重なお時間を割いて頂き，丁寧なるご教示を賜りました大阪大学大学院情報科学研究科マルチメディア工学専攻 藤原融教授ならびに馬場健一准教授に深く感謝申し上げます.

大学院博士後期課程において，情報工学全般に関して親切なるご指導とご助言を賜りました大阪大学大学院情報科学研究科マルチメディア工学専攻 西尾章治郎教授, 細田耕教授, 下條真司教授，ならびに秋吉政徳准教授に深く感謝申し上げます.

筆者が大阪大学大学院情報科学研究科マルチメディア工学専攻博士後期課程に在学することへのご配慮と援助を賜りました(株)日立製作所 情報制御システム社 CTO 前田章博士(システム開発研究所 元所長)，横浜研究所 所長 堀田多加志博士，研究開発本部 技師長 山本彰博士，横浜研究所 情報プラットフォーム研究センタ センタ長 松並直人氏，横浜研究所 主管研究長 岩嵜正明博士に感謝申し上げます．加えて，山本彰博士には直接の研究のご指導を賜り重ねて深く感謝申し上げます.

本研究の機会と大学院博士後期課程に進学する機会を与えて頂くとともに，研究を進めるにあたりご配慮を賜りました(株)日立製作所 RAID システム事業部 システム第 2 設計部 部長 山本康友氏(システム開発研究所 第八部 元部長)，研究開発本部 技術戦略室 ストラテジースタッフ 藤林昭氏(システム開発研究所 第八部 前部長)，横浜研究所 ストレージシステム研究部 部長 山本政行氏，ソフトウェア事業部 DB 設計部 主管技師 藤原真二氏(中央研究所 プラットフォームシステム研究部 前部長)，RAID システム事業部 主任技師 薗田浩二氏(システム開発研究所 元ユニットリーダ)，RAID システム事業部 ファイルストレージ開発設計部 主任技師 山崎康雄氏(システム開発研究所 元ユニットリーダ)，RAID システム事業部 システム第 3 設計部 主任技師 西本哲氏(システム開発研究所 元ユニットリーダ)，日立アジア社 シニアリサーチャ 志賀賢太氏(システム開発研究所 元ユニットリーダ)，日立アメリカ社 シニアリサーチャ 中野隆裕氏(システム開発研究所 前ユニットリーダ)，横浜研究所 ストレージシステム研究部 ユニットリーダ研究員 須藤敦之氏

に心より御礼申し上げます.

本研究の機会を与えて頂き，研究への多数のご助言を賜りました(株)日立製作所 RAID システム事業部 主管技師長 岩崎元昭氏, RAID システム事業部 ファイルストレージ開発設計部 部長 里山元章氏に心より感謝いたします．また本研究に関し，日々様々なご討論ご助言を頂くとともに多大なるご支援を頂きました(株)日立製作所 横浜研究所 ストレージシステム研究部 研究員 亀井仁志氏に心から御礼申し上げます．ならびに格別なるご指導とご配慮を賜りました(株)日立製作所 中央研究所, 横浜研究所(旧システム開発研究所), RAID システム事業部，大阪大学大学院情報科学研究科マルチメディア工学専攻，薦田研究室 OB・OG の各位に心から御礼申し上げます.

筆者が研究活動をはじめるにあたり，大阪大学工学部精密工学科および大阪大学大学院工学研究科精密科学専攻において懇切なるご指導とご鞭撻を賜りました大阪大学大学院工学研究科精密科学・応用物理学専攻 広瀬喜久治 特任教授(同専攻 名誉教授)に心から感謝申し上げます.

最後に，健康な心身に成長させてくれた，また学位取得に向け応援してくれた両親に感謝致します．また，本論文の執筆にあたり，応援してくれるとともに，執筆の時間確保に配慮してくれた妻 千尋と娘 絢音に心から感謝致します.

参考文献

[1] Preston, W.，豊沢聡，金崎裕己：“SAN & NAS ストレージネットワーク管理”，オライリー・ジャパン(2002).

[2] 喜連川優（編）：“ストレージネットワーキング”，オーム社(2002).

[3] Troppens, U., Erkens, R., Mueller-Friedt, W., Wolafka, R., and Haustein, N.: *“Storage Networks Explained: Basics and Application of Fibre Channel SAN, NAS, iSCSI, InfiniBand and FCoE”*, Wiley (2009).

[4] Dwivedi, H.: *“Securing Storage: A Practical Guide to SAN and NAS Security”*, Addison-Wesley Professional (2005).

[5] Chirillo, J. and Blaul, S.: *“Storage Security: Protecting SANs, NAS and DAS”*, Wiley (2002).

[6] Worden, D.J.: *“Storage Networks”*, Apress (2004).

[7] Spalding, R.: *“Storage Networks: The Complete Reference”*, McGraw-Hill Osborne Media (2003).

[8] Farley, M.: *“Storage Networking Fundamentals: An Introduction to Storage Devices, Subsystems, Applications, Management, and File Systems”*, Cisco Press (2004).

[9] Jepsen, T.C.: *“Distributed Storage Networks: Architecture, Protocols and Management”*, Wiley (2003).

[10] ブレントカラハン:“NFS バイブル(ASCII Addion Wesley Programming Series)”，アスキー(2001).

[11] ハルスターン，リカルドラビアガ，マイクアイスラー：“NFS & NIS 第 2 版”，オライリー・ジャパン(2002).

[12] イレズザドック：“NFS & AMD (Advanced Linux)”，翔泳社(2003).

[13] Callaghan, B., Pawlowski, B., and Staubach, P.: “NFS Version 3 Protocol Specification”, IETF (online), available from <http://tools.ietf.org/html/rfc1813> (accessed 2011-04-11).

[14] Sandberg, R., Goldberg, D., Kleiman, S., Walsh, D., and Lyon, B.: “Design and

Implementation of the Sun Network Filesystem", in *Proceedings of the Summer 1985 USENIX Conference*, pp.119–130 (1985).

[15] Hertel, C.: *"Implementing CIFS: The Common Internet File System (Bruce Perens' Open Source Series)"*, Prentice Hall (2003).

[16] French, S.: "A New Network File System is Born: Comparison of SMB2, CIFS and NFS", in *Proceedings of the Linux Symposium*, Vol.1, pp.131–140 (2007).

[17] Grunbacher, A. and Nuremberg, A.: "Posix Access Control Lists on Linux", in *Proceedings of the 2003 USENIX Annual Technical Conference (ATC): FREENIX track,* pp.259–272 (2003).

[18] "How the System Uses ACLs", NTFS.com (online), available from <http://www.ntfs.com/ntfs-permissions-acl-use.htm> (accessed 2011-04-11).

[19] Shepler, S., Callaghan, B., Robinson, D., Thurlow, R., Beame, C., Eisler, M., and Noveck, D.: "Network File System (NFS) version 4 Protocol", IETF (online), available from <http://www.ietf.org/rfc/rfc3530.txt> (accessed 2011-04-11).

[20] Shepler, S., Eisler, M., and Noveck, D.: "Network File System (NFS) Version 4 Minor Version 1 Protocol", IETF (online), available from <http://www.ietf.org/rfc/rfc5661.txt> (accessed 2011-04-11).

[21] Weil, S., Brandt, S., Miller, E., Long, D., and Maltzahn, C.: "Ceph: A Scalable, High-Performance Distributed File System", in *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI)*, pp.307–320 (2006).

[22] Weil, S.: "Ceph: Reliable, Scalable, and High-Performance Distributed Storage", PhD Thesis, University of California (2007).

[23] Josephson, W., Bongo, L., Li, K., and Flynn, D.: "DFS: A File System for Virtualized Flash Storage", *ACM Transactions on Storage (TOS)*, Vol.6, No.3, pp.1–25 (2010).

[24] Ungureanu, C., Atkin, B., Aranya, A., Gokhale, S., Rago, S., CaAlkowski, G., Dubnicki, C., and Bohra, A.: "HydraFS: A High-Throughput File System for the HYDRAstor Content-Addressable Storage System", in *Proceedings of the 8th USENIX Conference on File and Storage Technologies (FAST)*, p.17 (2010).

[25] Dubnicki, C., Gryz, L., Heldt, L., Kaczmarczyk, M., Kilian, W., Strzelczak, P., Szczepkowski, J., Ungureanu, C., and Welnicki, M.: "Hydrastor: A Scalable

Secondary Storage", in *Proccedings of the 7th Conference on File and Storage Technologies (FAST)*, pp.197–210 (2009).

[26] Welch, B., Unangst, M., Abbasi, Z., Gibson, G., Mueller, B., Small, J., Zelenka, J., and Zhou, B.: "Scalable Performance of the Panasas Parallel File System", in *Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST)*, p.2 (2008).

[27] Eisler, M., Corbett, P., Kazar, M., Nydick, D., andWagner, J.: "Data ONTAP GX: A Scalable Storage Cluster", in *Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST)*, pp.139–152 (2007).

[28] Rhea, S., Eaton, P., Geels, D., Weatherspoon, H., Zhao, B., and Kubiatowicz, J.: "Pond: The OceanStore Prototype", in *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST)*, pp.1–14 (2003).

[29] Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Wells, C. et al.: "Oceanstore: An Architecture for Global-Scale Persistent Storage", *ACM SIGARCH Computer Architecture News*, Vol.28, No.5, pp.190–201 (2000).

[30] Schmuck, F. and Haskin, R.: "GPFS: A Shared-Disk File System for Large Computing Clusters", in *Proceedings of the 1st USENIX Conference on File and Storage Technologies (FAST)*, pp.231–244 (2002).

[31] Thekkath, C., Mann, T., and Lee, E.: "Frangipani: A Scalable Distributed File System", in *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP)*, pp.224–237 (1997).

[32] Haskin, R. and Schmuck, F.: "The Tiger Shark File System", in *Proceedings of the Compcon*, pp.226–231 (1996).

[33] Carns, P., LigonIII, W., Ross, R., and Thakur, R.: "PVFS: A Parallel File System for Linux Clusters", in *Proceedings of the 4th Annual Linux Showcase and Conference*, pp.391–430 (2000).

[34] Rodeh, O. and Teperman, A.: "zFS-A Scalable Distributed File System Using Object Disks", in *Proceedings of the 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST)*, pp.207–218 (2003).

[35] Hasan, R., Anwar, Z., Yurcik, W., Brumbaugh, L., and Campbell, R.: "A Survey of Peer-to-Peer Storage Techniques for Distributed File Systems", in *Proceedings of*

the International Conference on Information Technology: Coding and Computing (ITCC), Vol.2, pp.205–213 (2005).

[36] Howard, J., Kazar, M., Menees, S., Nichols, D., Satyanarayanan, M., Sidebotham, R., and West, M.: "Scale and Performance in a Distributed File System", *ACM Transactions on Computer Systems (TOCS)*, Vol.6, No.1, pp.51–81 (1988).

[37] Huber, Jr., J., Chien, A., Elford, C., Blumenthal, D., and Reed, D.: "PPFS: A High Performance Portable Parallel File System", in *Proceedings of the 9th International Conference on Supercomputing*, pp.385–394 (1995).

[38] Satyanarayanan, M., Kistler, J., Kumar, P., Okasaki, M., Siegel, E., and Steere, D.: "Coda: A Highly Available File System for a Distributed Workstation Environment", *IEEE Transactions on Computers*, Vol.39, No.4, pp.447–459 (1990).

[39] Ghemawat, S., Gobioff, H., and Leung, S.: "The Google File System", in *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, pp.29–43 (2003).

[40] Borthakur, D.: "The Hadoop Distributed File System: Architecture and Design", Hadoop Project (online), available from <http://hadoop.apache.org/common/docs/r0.18.0/hdfs_design.pdf> (accessed 2011-04-11).

[41] Shvachko, K., Kuang, H., Radia, S., and Chansler, R.: "The Hadoop Distributed File System", in *Proceedings of the 26th IEEE Symposium on Mass Storage Systems and Technologies (MSST)*, pp.1–10 (2010).

[42] Hildebrand, D. and Honeyman, P.: "Exporting Storage Systems in a Scalable Manner with pNFS", in *Proceedings of the 22nd IEEE/13th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST)*, pp.18–27 (2005).

[43] Hildebrand, D., Eshel, M., Haskin, R., Andrews, P., Kovatch, P., and White, J.: "Deploying pNFS across the WAN: First Steps in HPC Grid Computing", in *Proceedings of the 9th LCI International Conference on High-Performance Clustered Computing*, pp.1–12 (2008).

[44] Schwan, P.: "Lustre: Building a File System for 1000-Node Clusters", in *Proceedings of the Linux Symposium*, pp.380–386 (2003).

[45] Shepard, L. and Eppe, E.: "SGI InfiniteStorage Shared Filesystem CXFS: A High-Performance, Multi-OS Filesystem from SGI", SGI (online), available from

<http://www.irixworld.net/irix/docs/pdfs/2691.pdf> (accessed 2011-04-11).

[46] Muthitacharoen, A., Morris, R., Gil, T., and Chen, B.: "Ivy: A Read/Write Peer-to-Peer File System", in *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*, pp.31–44 (2002).

[47] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., and Vogels, W.: "Dynamo: Amazon's Highly Available Key-Value Store", in *Proceedings of the 21st ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*, pp.205–220 (2007).

[48] Soares, L. and Stumm, M.: "FlexSC: Flexible System Call Scheduling with Exception-Less System Calls", in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI)*, pp.1–8 (2010).

[49] 横田治夫 監修："戦略的創造研究推進事業 CREST 研究領域「情報社会を支える新しい高性能情報処理技術」研究課題「ディペンダブルで高性能な先進ストレージシステム」研究終了報告書" (2009) (3.5 節「関連技術動向調査(日立，ストレージ技術動向調査グループ)」担当).

[50] 徳田晴介，中村隆喜，藤原真二："RAID 内蔵型 NAS の差分スナップショット機能における差分データ格納ボリューム拡張", 電子情報通信学会技術研究報告 CPSY コンピュータシステム，Vol.104, No.537, pp.13–18 (2004).

[51] 根本潤，須藤敦之，中村隆喜："高速差分抽出方式による非同期リモートバックアップの効率化"，情報処理学会全国大会講演論文集，Vol.71, No.1, pp.1.49–1.50 (2009).

[52] Silverberg, S.: "Opendedup: A userspace deduplication file system (SDFS)", Opendedup (online), available from <http://opendedup.org/> (accessed 2011-04-11).

[53] Edwards, J., Ellard, D., Everhart, C., Fair, R., Hamilton, E., Kahn, A., Kanevsky, A., Lentini, J., Prakash, A., Smith, K. et al.: "FlexVol: Flexible, Efficient File Volume Virtualization in WAFL", in *Proceedings of the 2008 USENIX Annual Technical Conference (ATC)*, pp.129–142 (2008).

[54] Macko, P., Seltzer, M., and Smith, K.: "Tracking Back References in a Write-Anywhere File System", in *Proceedings of the 8th USENIX Conference on File and Storage Technologies (FAST)*, pp.15–28 (2010).

[55] Zhang, Y., Rajimwale, A., Arpaci-Dusseau, A., and Arpaci-Dusseau, R.: "End-to-end Data Integrity for File Systems: A ZFS Case Study", in *Proceedings*

of the 8th USENIX Conference on File and Storage Technologies (FAST), pp.29–42 (2010).

[56] Hasan, R., Sion, R., and Winslett, M.: "The Case of the Fake Picasso: Preventing History Forgery with Secure Provenance", in *Proccedings of the 7th USENIX Conference on File and Storage Technologies (FAST)*, pp.1–14 (2009).

[57] Muniswamy-Reddy, K. and Holland, D.: "Causality-Based Versioning", *ACM Transactions on Storage (TOS)*, Vol.5, No.4, pp.13:1–13:28 (2009).

[58] Spillane, R., Gaikwad, S., Chinni, M., Zadok, E., and Wright, C.: "Enabling Transactional File Access via Lightweight Kernel Extensions", in *Proccedings of the 7th USENIX Conference on File and Storage Technologies (FAST)*, pp.29–42 (2009).

[59] Peterson, Z., Burns, R., Ateniese, G., and Bono, S.: "Design and Implementation of Verifiable Audit Trails for a Versioning File System", in *Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST)*, pp.93–106 (2007).

[60] Peterson, Z., Burns, R., Herring, J., Stubblefield, A., and Rubin, A.: "Secure Deletion for a Versioning File System", in *Proceedings of the 4th USENIX Conference on File and Storage Technologies (FAST)*, pp.143–154 (2005).

[61] 高橋基信：“改訂版 Samba のすべて”，翔泳社(2011).

[62] Carter, G., Ts, J., and Eckstein, R.: *"Using Samba (3rd Edition)"*, Oreilly & Associates Inc (2007).

[63] Terpstra, J.H. and Vernooij, J.R.: *"Official Samba-3 HOWTO and Reference Guide (2nd Edition) (Bruce Perens' Open Source Series)"*, Prentice Hall (2005).

[64] 武田保真：“Samba 逆引きリファレンス Samba3.4 対応”，秀和システム(2009).

[65] 高橋基信：“サーバ構築の実例がわかる Samba[実践] 入門(Software Design Plus)”，技術評論社(2010).

[66] Berriman, E.: "NetApp Storage System Multiprotocol User Guide", Technical Report TR-3490, NetApp (2006).

[67] "Windows Services for UNIX", Microsoft (online), available from <http://technet.microsoft.com/ja-jp/interopmigration/bb380242.aspx> (accessed 2011-04-11).

[68] Week, L. and Falkner, S.: "Solaris NFSv4 ACL Implementation Experience, Connectathon Talks 2005", Sun Microsystems (online), available from

<http://www.connectathon.org/talks05/falkner.pdf> (accessed 2011-04-11).

[69] "ZFS On-Disk Specification Draft", Sun Microsystems (online), available from <http://hub.opensolaris.org/bin/view/Community+Group+zfs/docs> (accessed 2011-04-11).

[70] "NTFS General Information > NTFS Basics > NTFS File Types", NTFS.com (online), available from <http://www.ntfs.com/ntfs-files-types.htm> (accessed 2011-04-11).

[71] "NEC iStorage NV シリーズ", NEC（オンライン），入手先<http://www.nec.co.jp/products/istorage/product/nas/index.shtml>（参照 2011-04-11）.

[72] "マイクロソフトサポートオンライン FAT16 ファイルシステムから FAT32 に変換するには", Microsoft（オンライン），入手先<http://support.microsoft.com/kb/882971/ja>（参照 2011-04-11）.

[73] "Windows TIPS FAT → NTFS にファイル・システムを変換する", @IT（オンライン），入手先<http://www.atmarkit.co.jp/fwin2k/win2ktips/350fat2ntfsconv/fat2ntfsconv.html>（参照 2011-04-11）.

[74] Makatos, T., Klonatos, Y., Marazakis, M., Flouris, M., and Bilas, A.: "Using Transparent Compression to Improve SSD-based I/O Caches", in *Proceedings of the 5th European Conference on Computer Systems (EuroSys)*, pp.1–14 (2010).

[75] Makatos, T., Klonatos, Y., Marazakis, M., Flouris, M., and Bilas, A.: "ZBD: Using Transparent Compression at the Block Level to Increase Storage Space Efficiency", in *Proceedings of the 6th IEEE International Workshop on Storage Network Architecture and Parallel I/Os (SNAPI)*, pp.61–70 (2010).

[76] Wright, C., Martino, M., and Zadok, E.: "NCryptfs: A Secure and Convenient Cryptographic File System", in *Proceedings of the 2003 USENIX Annual Technical Conference (ATC)*, pp.197–210 (2003).

[77] "Linux ファイルシステム技術解説 第 7 回 64bit ファイルシステム XFS の実装", @IT（オンライン），入手先<http://www.atmarkit.co.jp/flinux/rensai/fs07/fs07a.html>（参照 2011-04-11）.

[78] Powell, M.: "The DEMOS File System", in *Proceedings of the 6th ACM Symposium on Operating Systems Principles (SOSP)*, pp.33–42 (1977).

[79] Bovet, D.P. and Cesati, M.: "詳解 Linux カーネル第 3 版", オライリー・ジャパン (2007).

[80] 中村隆喜，亀井仁志，山本彰，薦田憲久：“アクセス制御統合による性能低下を改善する階層型アクセス制御方式”，情報処理学会論文誌，Vol.51, No.11, pp.2066–2080 (2010).

[81] 亀井仁志，揚妻匡邦，浦野明裕，中村隆喜：“ファイルアクセスを契機としたファイルシステム管理情報変換方式”，情報処理学会全国大会講演論文集，Vol.70，No.1, pp.1.33–1.34 (2008).

[82] 中村隆喜，亀井仁志，山本彰，薦田憲久：“ファイルアクセススループットを改善する POSIX ACL-高機能 ACL 併用方式”，情報処理学会論文誌，Vol.52，No.6, pp.1939–1950 (2011).

[83] 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-”, FIT2004 情報科学技術フォーラム講演論文集，Vol.3, No.1, pp.119–120 (2004).

[84] Nakamura, T. and Komoda, N.: “Pre-allocation Adjusting Methods Depending on Growing File Size”, in *Proceedings of the 5th International Workshop on Storage Network Architecture and Parallel I/Os (SNAPI)*, pp.19–25 (2008).

[85] 中村隆喜，薦田憲久：“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”，情報処理学会論文誌，Vol.50, No.11, pp.2690–2698 (2009).

[86] Allison, B., Hawley, R., Borr, A., Muhlestein, M., and Hitz, D.: “File System Security: Secure Network Data Sharing for NT and UNIX”, in *Proceedings of the Large Installation System Administration of Windows NT Conference*, p.3 (1998).

[87] Hitz, D., Allison, B., Borr, A., Hawley, R., and Muhlestein, M.: “Merging NT and UNIX Filesystem Permissions”, in *Proceedings of the 2nd USENIX Windows NT Symposium*, Vol.2, pp.10–10 (1998).

[88] Leung, A., Pasupathy, S., Goodson, G., and Miller, E.: “Measurement and Analysis of Large-Scale Network File System Workloads”, in *Proceedings of the 2008 USENIX Annual Technical Conference (ATC)*, pp.213–226 (2008).

[89] “The Official Samba 3.5.x HOWTO and Reference Guide, Chapter 24. Winbind: Use of Domain Accounts”, Samba Team (online), available from <http://www.samba.org/samba/docs/man/Samba-HOWTO-Collection/winbind.html> (accessed 2011-04-11).

[90] 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).

[91] “SPECsfs2008 User’s Guide”, Standard Performance Evaluation Corporation (SPEC) (online), available from <http://www.spec.org/sfs2008/docs /usersguide.html> (accessed 2011-04-11).

[92] Mostek, J., Earl, B., Levine, S., Lord, S., Cattelan, R., McDonell, K., Kline, T., Gaffey, B., and Ananthanarayanan, R.: “Porting the SGI XFS File System to Linux”, in *Proceedings of the 2000 USENIX Annual Technical Conference (ATC): Freenix Track*, pp.65–76 (2000).

[93] “Main Page”, XFS.org (online), available from <http://xfs.org/index.php /Main_Page> (accessed 2011-04-11).

[94] “XFS Filesystem Structure 2nd Edition Revision 1”, SGI (online), available from <http://oss.sgi.com/projects/xfs/papers/xfs_filesystem_structure.pdf> (accessed 2011-04-11).

[95] 鰭崎克巳，檜垣誠一，金井宏樹，川崎徹：“コストパフォーマンスに優れた NAS ゲートウェイ製品「Hitachi Essential NAS Platform」”，日立評論，Vol.90, No.3, pp.242−245 (2008).

[96] 松尾豊，安田雪：“SNS における関係形成原理 - mixi のデータ分析 - ”，人工知能学会論文誌， Vol.22, No.5, pp.531–541 (2007).

[97] Douceur, J. and Bolosky, W.: “A Large-Scale Study of File-System Contents”, in *Proceedings of the 1999 ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, pp.59–70 (1999).

[98] Smith, K. and Seltzer, M.: “File Layout and File System Performance”, Technical Report TR-35-94, Harvard University (1994).

[99] McKusick, M., Joy, W., Leffler, S., and Fabry, R.: “A Fast File System for UNIX”, *ACM Transactions on Computer Systems (TOCS)*, Vol.2, No.3, pp.181–197 (1984).

[100] “JFS fragment size”, IBM (online), available from <http://publib.boulder.ibm.com/infocenter/pseries/v5r3/index.jsp?topic=/com.ibm.aix.prftungd/doc/prftungd /jfs_frag_size.htm> (accessed 2011-04-11).

[101] “Sun QFS, Sun SAM-FS, Sun SAM-QFS ファイルシステム管理者マニュアル”, Sun Microsystems （オンライン），入手先 <http://download.oracle.com /docs/cd/E19314-01/816-7683-10/816-7683-10.pdf>（参照 2011-04-11）.

[102] Koch, P.: “Disk File Allocation Based on the Buddy System”, *ACM Transactions on Computer Systems (TOCS)*, Vol.5, No.4, pp.352–370 (1987).

[103] Seltzer, M. and Stonebraker, M.: “Read Optimized File System Designs: A Performance Evaluation”, in *Proceedings of the 7th IEEE International Conference on Data Engineering*, pp.602–611 (1991).

[104] Cooper, B.P.: “Linux file systems”, Technical Report TR-3274, NetApp (2003).

[105] Bar, M.: *“Linux File Systems (Application Development)”*, Mcgraw-Hill Osborne Media (2001).

[106] 小松克行：“ext2 ファイルシステム”, *Linux magazine*, No.3, pp.153–160 (1999).

[107] “VERITAS File System 4.1 管理者ガイド Linux”, VERITAS（オンライン），入手先 <http://www.symantec.com/business/support/resources/sites/BUSINESS/content/live/TECHNICAL_SOLUTION/45000/TECH45179/en_US/280553.pdf>（参照 2011-04-11）.

[108] Hitz, D., Lau, J., and Malcolm, M.: “File System Design for an NFS File Server Appliance”, Technical Report TR-3002, NetApp (2005).

[109] Agrawal, N., Bolosky, W., Douceur, J., and Lorch, J.: “A Five-Year Study of File-System Metadata”, *ACM Transactions on Storage (TOS)*, Vol.3, No.3, p.9 (2007).