情報処理学会論文誌 Vol. 52 No. 6 1939–1950 (June 2011)

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

中村隆喜[†1,†2] 亀井仁志[†1]
山本彰[†3] 薦田憲久[†2]

ファイルシステムのアクセス制御を高機能化した場合の，ファイルアクセススループットの低下と長時間のサービス停止を改善する，POSIX ACL–高機能 ACL 併用方式を提案する．本論文では特に 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・拡張属性の参照更新時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，POSIX 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 POSIX ACL 情報から高機能 ACL に変換する場合を使い分ける．複数の条件を比較し評価した結果，ACL エントリ数が 6 以上の場合を書き込むだけの工夫でも大幅にファイルアクセススループットを改善できることが分かった．

A Concurrent Use Method of Heterogeneous Access Control Information to Improve File Access Throughput

Takaki Nakamura,[†1,†2] Hitoshi Kamei,[†1]
Akira Yamamoto[†3] and Norihisa Komoda[†2]

We propose a concurrent use method to improve file access throughput and long time service disruption in the case that highly-functional access control is applied to existing operated filesystems. In this paper, we particularly focus on a case in which Windows ACL is applied to operated filesystem with POSIX ACL. The proposed method is based on the difference of access latencies to basic/extended attributes on filesystem and use one of following two cases depending on size of access control information: a case to store converted Windows ACL information and the other case not to store it and reconvert it from POSIX ACL information every time on access control processing. We evaluate several conditions and conclude that proposed method with the condition to store only if the number of entries is 6 and more is good enough to improve file access throughput.

1. はじめに

近年，ファイルを用いた情報共有の活性化にともない，ファイルサーバとストレージ装置を一体化した Network Attached Storage（NAS）の市場が拡大している．NAS には，Linux をはじめとする UNIX 系 OS をベースとして提供される UNIX 系 NAS と，Windows OS をベースとして提供される Windows 系 NAS がある．UNIX 系 NAS は UNIX 系 OS のクライアントと親和性が高く，Windows 系 NAS は Windows OS のクライアントと親和性が高い．

UNIX 系 NAS のアクセス制御は，一般に POSIX ACL[1] に基づき行われることが多い．Windows 系 NAS のアクセス制御は，Windows ACL[2] に基づき行われる．両 ACL を比較すると，Windows ACL の方がより情報量が多い．したがって，Windows OS クライアントが UNIX 系 NAS にアクセスした場合，アクセス制御の一部機能が使えないことがある．

上記問題に対応するため，UNIX 系 OS においても Windows ACL や，Windows ACL に準拠した NFSv4 ACL[3]（以降まとめて高機能 ACL と呼ぶ）をサポートする動きが広がっている．そのような例としては，NEC SC-LX[4]，Sun Solaris 10[5] がある．

高機能 ACL 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 POSIX 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，POSIX ACL 対応ファイルシステムの共有から高機能 ACL 対応新規ファイルシステムの共有に，Windows クライアントを用いて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サービス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，NAS 使用者，NAS 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，クライアントに透過的に POSIX ACL 情報を

†1 株式会社日立製作所横浜研究所
Yokohama Research Laboratory, Hitachi Ltd.
†2 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technologies, Osaka University
†3 株式会社日立製作所研究開発本部
Research & Development Group, Hitachi Ltd.

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[6]．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アクセス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 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アクセス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，事前一括移行の間サービス運用を停止する必要がある．

本論文ではこのファイルアクセススループットの低下と長時間のサービス停止を改善する POSIX 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・拡張属性の参照更新時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，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 POSIX 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アクセス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，POSIX 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．

本論文の以降の構成は次のとおりである．2 章では，文献 6) 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アクセススループットの低下について説明する．3 章では，提案方式である POSIX ACL–高機能 ACL 併用方式を説明する．4 章では，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．5 章で関連研究について述べ，最後に 6 章で，本研究のまとめを述べる．

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

本章では，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アクセススループットの低下と長時間のサービス停止について述べる．

2.1 高機能 ACL 対応ファイルシステムへの移行方式

本節では，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) サービス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) サービス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．

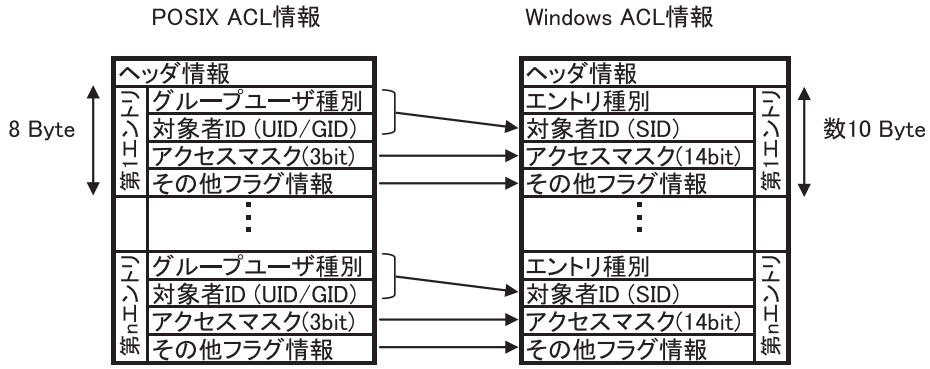

図 1　ACL 情報変換の例
Fig. 1　An example of ACL information conversion.

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，旧 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 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[6]．

ACL 情報の変換の例を**図 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 3 bit 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 8 Byte 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，14 bit 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十 Byte 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．NAS 自身が高機能 ACL に対応したコピー機能を提供していれば，NAS 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

クライアントへのファイルサービス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．サービス停止時間は，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アクセススループットを低下させてしまう．

移行方式 (2a) は，サービス停止中に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，サービス停止時間はファイル数や ACL エントリ長によるが，数十分から数十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アクセス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，サービス停止中に NAS は移行対象の旧 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．サービスを再開後，クライアント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．本移行方式ではサービス再開前の一括移行処理を行わないため，サービス停止時間をさらに短縮でき，数秒から数十秒程度にできる．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アクセススループットをさらに低下させてしまうことがある．

次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アクセススループット低下の詳細について述べる．

2.2　オンアクセス ACL 変換機能による移行でのファイルアクセススループットの低下

本節では，前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アクセス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，クライアント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 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．

高機能 ACL 情報を保持していないファイルシステムオブジェクトであった場合，POSIX ACL 情報（以降，変換前 ACL 情報と呼ぶ）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．

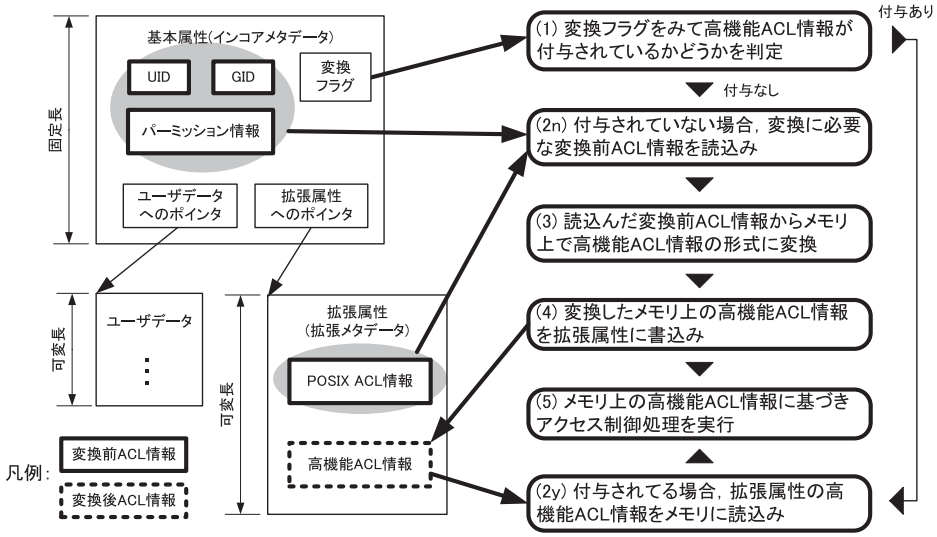

図 2　オンアクセス ACL 変換機能付きアクセス制御処理の概要
Fig. 2　Overview of access control process with on-access ACL conversion feature.

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，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，変換前 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，全体のファイルアクセススループットが低下することがある．

3.　POSIX ACL–高機能 ACL 併用方式

本章では，まず提案方式である POSIX ACL–高機能 ACL 併用方式の概要を述べる．

次に POSIX 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．

3.1　提案方式の概要

前章で述べたファイルアクセス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，変換前 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アクセス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アクセス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アクセス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変換前 ACL 情報から毎回変換する．

ここで，変換前 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，変換前 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，変換前 ACL 情報から毎回変換処理を行ったうえで，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，2.1 節の移行方式 (2a) のようにサービス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．

3.2　高機能 ACL 情報書き込み可否の選択に関する検討

本節では Linux の XFS[19] をベースに開発したオンアクセス変換機能を用いて，前節の 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 1** に示す参照更新時の処理時間の違いがある[20]．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．

表 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
Table 1　Types and processing time on read/write of XFS basic and extended attributes.

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，100 μs 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為，参照はディスクアクセスが発生し数 100 μs〜1ms 程度，更新はジャーナルログに書くだけなので 100 μs 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

そこで，変換前 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** に示す．測定は mount 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，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 の変換前 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．

変換前 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，変換前 ACL から毎回変換したほうがよい．

変換前 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 0.5 ms 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方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方がよい．

変換前 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

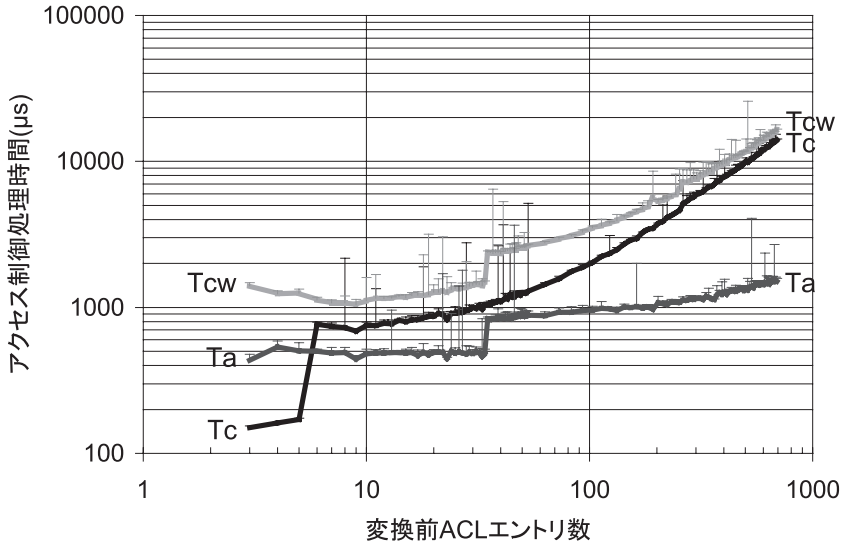

図 3　オンアクセス ACL 変換機能付きアクセス制御処理時間
Fig. 3　Elapsed time of access control process with on-access ACL conversion feature.

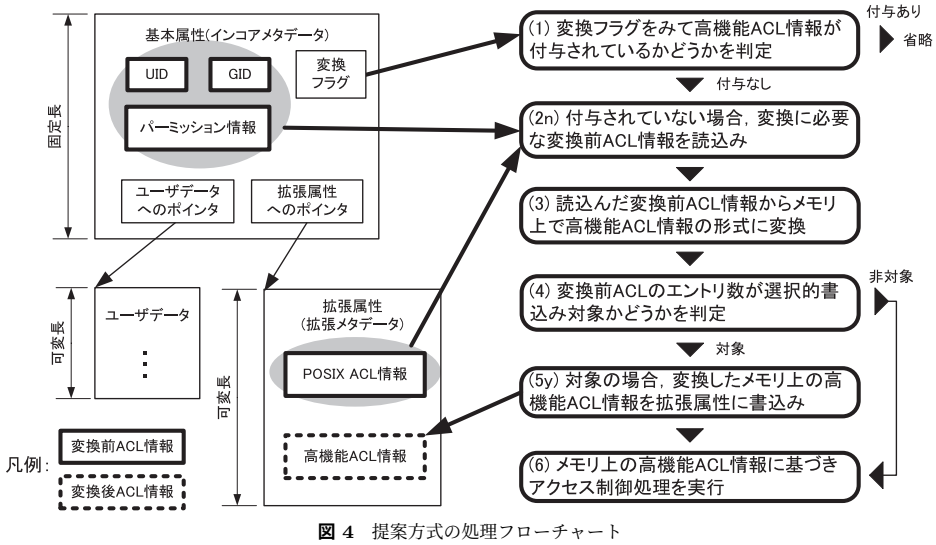

図 4　提案方式の処理フローチャート
Fig. 4　Flowchart of proposed method.

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} は変換前 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，変換前 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 $O(N \log N)$ 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方がよい．

ただし，たとえば NEC により実装・公開[4] されている ACL 専用キャッシュなどを，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方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 4.2 節で説明する．

本節では図 3 の測定値に基づき設計を示したが，機器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 5〜6，34〜35 に発生するアクセス制御処理時間の不連続点は，表 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，本節で示した設計は基本的に異なる機器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 と同様の測定を行った後，再設計が必要となる．

3.3　POSIX ACL–高機能 ACL 併用方式の詳細

前節の検討より，変換前 ACL のエントリ数に基づき，ディスク上の POSIX 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 情報を使用する場合とを，使い分ける方式が考えられる．

図 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 2 章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，変換に必要な変換前 ACL 情報を読み込む．読み込

んだ変換前 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，提案方式では変換前 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，変換前 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 情報を更新するオペレーションを受けた場合には，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アクセススループット低下の改善が期待できる．

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 POSIX ACL–高機能 ACL 併用方式は，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，クライアントからは，サービス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，3.2 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アクセス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とり，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，ファイルアクセス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．

3.5　提案方式の適用範囲

本論文に記載の提案方式は特にアクセス制御の高機能化時の複数種類の ACL の併用に対して検討を行っているが，ファイルシステムの高機能化によりある属性情報を拡張したり追加したりする場合に広く適用可能である．

また本論文では，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．他 UNIX 系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．

4.　提案方式の評価

本章では提案方式がファイルアクセス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．

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 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．(c) については SNS の関係性を分析した論文[7] 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，ファイル共有サービスの代表的ベンチマークである SPECsfs2008[8] を利用した．SPECsfs2008 は，単位時間（1 秒）あたりのファイルサービス処理オペレーション数（OPS），つまりファイルアクセス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 3** のとおりとなっている．同プログラムは作成した

表 2　仮定する ACL エントリ数分布
Table 2　Supposed distribution of the number of ACL entry.

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

表 3　SPECsfs2008 ファイルオペレーション割合
Table 3　Distribution of file operation on SPECsfs2008.

#	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

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．

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アクセス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アクセススループットでの比較が望ましい．

4.2　測定環境と測定条件

測定環境は**表 4** 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 POSIX ACL–高機能 ACL 併用方式を実装した．本論文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 POSIX 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．

表 4　測定環境
Table 4　Measurement environment.

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 5　測定条件
Table 5　Measurement condition.

#	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	全て	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	

測定条件は**表 5** 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 により実装され，Web サイト[4] 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させる．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主記憶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

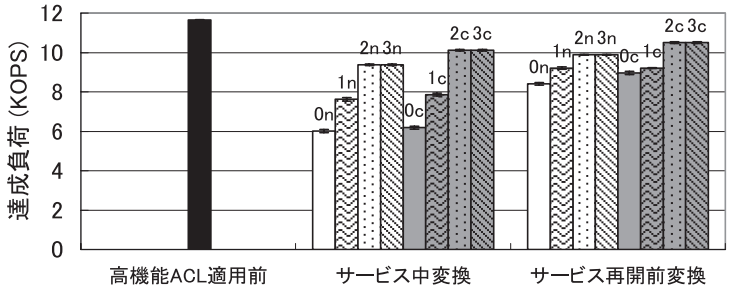

図 5　(a) 小規模ファイル共有の測定結果
Fig. 5　Measurement results of (a) small class file share.

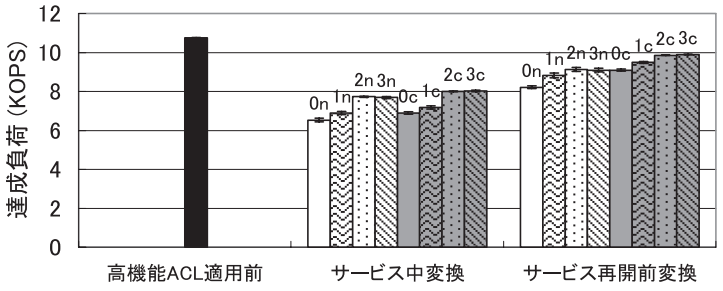

図 6　(b) 大規模ファイル共有の測定結果
Fig. 6　Measurement results of (b) large class file share.

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，本論文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アクセス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）のファイルアクセス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 適用前のファイルアクセス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，サービス再開前にあらかじめ書き込んでおいた場合のファイルアクセス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アクセス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．

4.3　測定結果と考察

本節では実際に 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従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で測定しているため，相対的な違いは比較可能である．

図 5 に (a) 小規模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（KOPS）つまりファイルアクセス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 適用後サービス中変換，高機能 ACL 適用後サービス再開前変換となる．サービス中変換とサービス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，ファイルアクセス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）でもキャッシュなしで約 1.6 KOPS（27%），ありで 1.7 KOPS（27%）性能を向上させることができた．また，測定条件#2，#3 ではキャッシュなしで約 3.4 KOPS（56%），ありで 3.9 KOPS（64%）性能を向上させることができた．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．サービス再開前変換はサービス中変換に対し，全体的に性能が改善される．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 6** に，(b) 大規模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ごく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 1.2 KOPS（18%），キャッシュありで最大で約 1.1 KOPS（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%に改善された．サービス再開前変換の傾向は事例 (a) と同様である．

最後に**図 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

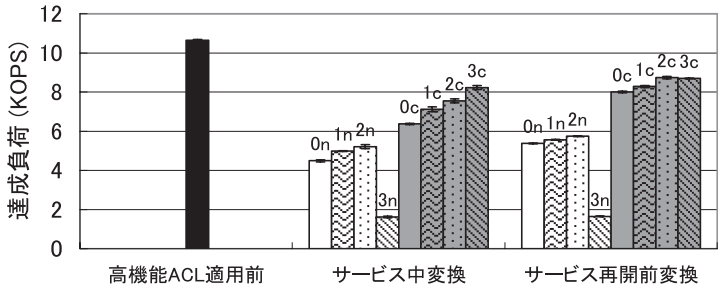

図 7　(c)SNS 連携ファイル共有の測定結果
Fig. 7　Measurement results of (c) file share integrated with SNS.

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，0.7 KOPS（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 1.9 KOPS（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 3 KOPS 程度あるため，キャッシュ＋提案方式の組合せでは約 3.7 KOPS（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 3.2 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%に改善された．サービス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，分析ツールなどを用いて分析した．提案方式によって性能が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．性能向上効果は数十%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 短縮されていた．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

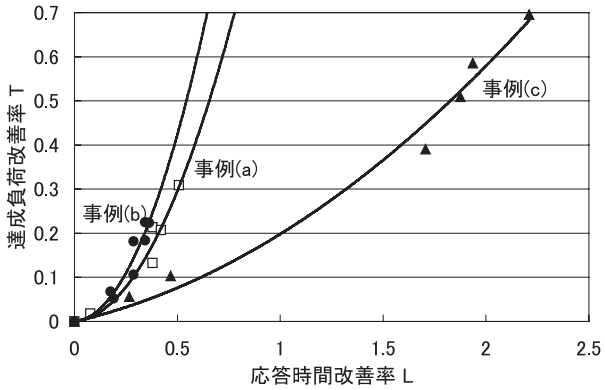

図 8　平均応答時間と達成負荷の改善率の相関
Fig. 8　Correlation of improvement rate between response time and achieved IOPS.

図 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．

提案方式のサービス停止時間は，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 1,000 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，サービス再開前変換方式のサービス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 100〜1,000 秒程度であった．仮に 200 TB のデータセットを想定した場合，そのサービス停止時間は 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，サービス再開前変換は有用である．これに対し，数十 TB 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．

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 3.4 KOPS（56%）程度，キャッシュありの場合キャッシュの効果と合わせて最大で 3.7 KOPS（83%）程度の性能向上が可能なことを明らかにした．

5. 関 連 研 究

ファイルの変換処理において，変換処理を高速化する工夫に関する学術論文は，たとえばファイル圧縮処理やファイル暗号化処理に関するものがある．

ファイル圧縮に関する論文[9),10)]では，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アクセス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[11)]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ファイルシステムに対してファイルアクセススループット改善を実現している．

これらのアイデアの一部は，旧バージョンから新バージョンのフォーマット変換の高速化にも適用が可能である．事実，本提案方式における変換処理もカーネル実装である．しかしこれらの関連研究では，本論文で述べた変換結果を選択的に書き込むという高速化手法については触れられていない．

Windows ACL や NFSv4 ACL などの高機能 ACL に対応した製品の実装として，NetApp FAS シリーズ security style[12)]，Microsoft Windows Service For Unix（SFU）[13)]，Sun Solaris 10[5),14),15)]，NEC iStorage NV シリーズ SC-LX[4)] などがある．ただしこれらのいずれも FAT[16)]，POSIX ACL などの従来アクセス制御情報のファイルシステムからの透過的変換手段は提供していない．

したがって，これらの製品において変換を行う場合は，従来アクセス制御情報ベースの移行元ファイル共有から，高機能 ACL ベースの移行先ファイル共有へ，クライアントを経由してコピーするというソリューションを採用することが最も一般的である．

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．FAT16 から FAT32 への変換のためのドライブコンバータ[17)]，FAT から NTFS への変換のための convert コマンド[18)] がある．これらは上記のクライアントコピーによるソリューションに比べて新たな変換先のディスクが不要というメリットがある．しかしいずれの方式も，ファイルサービス運用を長時間停止する必要がある．これに対し，本研究では変換処理をアクセス時に実行するので，運用停止時間を短時間に抑えることができる．

6. お わ り に

本論文では，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アクセススループット低下と長時間のサービス停止を改善する，POSIX 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 割性能を向上させることができた．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 割性能を向上させることができた．またサービス停止時間も 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．

今後の課題として，次の 2 点があげられる．第 1 の課題は，本論文の評価で性能改善への効果が合わせて確認できた 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，本論文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，本論文の考え方を，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．

参 考 文 献

1) Gruenbacher, A.: POSIX Access Control Lists on Linux, *Proc. USENIX Annual Technical Conference* (*FREENIX Track*), pp.259–272 (2003).
2) How the System Uses ACLs (online), available from ⟨http://www.ntfs.com/ntfs-permissions-acl-use.htm⟩ (accessed 2010-09-24).
3) Network File System (NFS) Version 4 Minor Version 1 Protocol (online), available from ⟨http://tools.ietf.org/html/rfc5661⟩ (accessed 2010-03-12).
4) NEC iStorage NV シリーズ（オンライン），入手先 ⟨http://www.nec.co.jp/products/istorage/product/nas/index.shtml⟩ (参照 2009-08-06).
5) Week, L. and Falkner, S.: Solaris NFSv4 ACL Implementation Experience, Connectathon Talks 2005, available from ⟨http://www.connectathon.org/talks05/falkner.pdf⟩ (accessed 2009-08-06).
6) 亀井仁志，中村隆喜，揚妻匡邦，浦野明裕：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，情報処理学会第 70 回全国大会講演論文集，pp.1-33–1-34 (2008).
7) 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).
8) SPECsfs2008 User's Guide (online), available from ⟨http://www.spec.org/sfs2008/docs/usersguide.html⟩ (accessed 2009-08-06).
9) Makatos, T., Klonatos, Y., Marazakis, M., Flouris, M.D. and Bilas, A.: Using Transparent Compression to Improve SSD-based I/O Caches, *Proc. 5th European Conference on Computer Systems*, pp.1–14 (2010).
10) Makatos, T., Klonatos, Y., Marazakis, M., Flouris, M.D. and Bilas, A.: ZBD: Using Transparent Compression at the Block, *Proc. 6th IEEE International Workshop on Storage Network Architecture and Parallel I/Os*, pp.61–70 (2010).
11) Wright, C.P., Martino, M.C. and Zadok, E.: NCryptfs: A Secure and Convenient Cryptographic File System, *Proc. USENIX Annual Technical Conference* (*General Track*), pp.197–210 (2003).
12) Berriman, E.: NetApp Storage System Multiprotocol Use Guide, NetApp Technical Report TR-3490 (2006).
13) Windows Services for UNIX (online), available from ⟨http://technet.microsoft.com/ja-jp/interopmigration/bb380242.aspx⟩ (accessed 2009-08-06).
14) Week, L. and Falkner, S.: NFSv4 ACLs: Present and Future, NAS Industry Conference 2005, available from ⟨http://nasconf.com/pres05/falkner.pdf⟩ (accessed 2009-08-06).
15) Falkner, S. and Week, L.: NFSv4 ACLs: Where are we going?, Connectathon Talks 2006, available from ⟨http://www.connectathon.org/talks06/falkner-week.pdf⟩ (accessed 2009-08-06).
16) Solomon, D.A. and Russinovich, M.E.（著），豊田　孝（訳）：インサイド Microsoft Windows 第 4 版〈下〉，日経 BP ソフトプレス (2005).
17) 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（オンライン），入手先 ⟨http://support.microsoft.com/kb/882971/ja⟩ (参照 2010-09-24).
18) FAT→NTFS にファイル・システムを変換する（オンライン），入手先 ⟨http://www.atmarkit.co.jp/fwin2k/win2ktips/350fat2ntfsconv/fat2ntfsconv.html⟩ (参照 2010-09-24).
19) XFS.org (online), available from ⟨http://xfs.org/index.php/Main_Page⟩ (accessed 2011-01-05).
20) XFS Filesystem Structure 2nd Edition Revision 1 (online), available from ⟨http://oss.sgi.com/projects/xfs/papers/xfs_filesystem_structure.pdf⟩ (accessed 2011-01-05).

(平成 22 年 10 月 14 日受付)
(平成 23 年 3 月 7 日採録)

中村　隆喜（正会員）

1973 年生．1996 年 3 月大阪大学工学部精密工学科卒業．1998 年 3 月同大学大学院工学研究科精密科学専攻博士前期課程修了．同年 4 月（株）日立製作所入社．中央研究所勤務．同社システム開発研究所勤務を経て，2011 年 4 月より同社横浜研究所勤務．2010 年 4 月より，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在籍．ファイルストレージ，オペレーティングシステムの研究開発に従事．

亀井　仁志（正会員）

1980 年生．2002 年 3 月香川大学工学部信頼性情報システム工学科卒業．2004 年 3 月奈良先端科学技術大学院大学情報科学研究科情報システム学専攻博士前期課程修了．同年 4 月（株）日立製作所入社．システム開発研究所勤務．2011 年 4 月より同社横浜研究所勤務．ネットワークストレージ，ファイルシステムの研究開発に従事．

山本　　彰（正会員）

1977 年京都大学工学部情報工学科卒業．1979 年同大学大学院修士課程修了．同年日立製作所入社．同年システム開発研究所に入所．性能評価手法，ストレージシステムの研究に従事．2007 年より同社研究開発本部所属．工学博士．

薦田　憲久

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