NILFSの拡張属性の設計と実装

鈴木　章浩†　　小西　隆介‡　　盛合　敏‡　　追川　修一†
†筑波大学システム情報工学研究科　　‡NTTサイバースペース研究所

1　はじめに

Linux 2.6.30にマージされた連続スナップショットファイルシステムNILFS[1]はログ構造化ファイルシステムである．NILFSは全ての変更をログとしてディスクに追記し，それらをスナップショットとして管理する．そのため利用者は任意のスナップショットを選択することで自由に過去の状態に戻ることができる．それによりNILFSはシステムのクラッシュや利用者のオペレーションミスから高速に復旧することが可能である．

現在，NILFSは拡張属性機能を実装していない．拡張属性とはファイルやディレクトリに対して任意の名前と値を持つ属性のことであり，アプリケーション独自の管理用メタデータを格納するために利用される．そのためアプリケーションは特別にデータベースを用意する必要がなく，同時にデータベースとカーネル間の同期オーバヘッドも削減される．よってアプリケーションは管理用メタデータを効率良く扱うことが可能となり，パフォーマンスを向上させることができる．

Linux以外のOSにおいても拡張属性に類似した機能が実装されている．例えばWindows NT系のNTFSは代替ストリーム，Mac OS XのHFS+はフォークが拡張属性に相当し，これらは画像のプレビューやGUIのリソース管理，ユーザによる任意の拡張データ管理に利用されている．

Linuxでは上記のような利用に加えて，OSのセキュリティを向上させる強制アクセス機構であるSELinuxやPOSIX ACLなどの重要なシステム機能の基盤としても利用されている．POSIX ACLとはPOSIXで定義されているアクセス制御リストのことで，ユーザ・グループ毎にアクセス権を自由に設定することができる．これによりアクセス制御を，従来のユーザ・グループ・その他という広範囲なものよりも，より柔軟に設定することが可能となる．この機能のサポートは，Linuxのエンタープライズ分野への導入を加速するきっかけの一つとなった．

以上のように現代のファイルシステムにおいて拡張属性のサポートは不可欠である．そこで本論文ではNILFSに対して拡張属性機能を設計・実装し，その性能を評価する．ただし実装される拡張属性は，過去の状態に戻ることができるというNILFSの特徴を合わせ持つ必要がある．

Design and Implementation of Extended Attribute for NILFS
†Akihiro Suzuki　‡Ryusuke Konishi
‡Satoshi Moriai　†Shuichi Oikawa
†Graduate School of Systens and Information Engineering, University of Tsukuba
‡NTT Cyber Space Laboratories

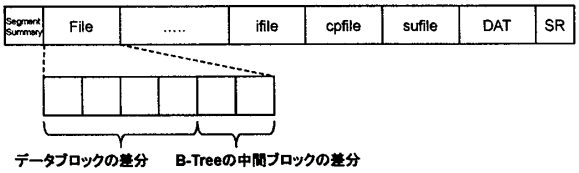

図1: 追記されるログ

2　設計

拡張属性を実現する手法の候補を以下に挙げ，それらについて順に述べる．

第1の手法は拡張属性用のメタデータファイルを追加するというものである．詳細を述べる前に，この手法に関係する内容としてNILFSがディスクに追記するログについて述べる．NILFSは図1のように前回のログ以降に編集されたデータの差分をログとし，ディスクに追記する．ここで書き込まれるログにはデータブロックだけでなく，iノードやセグメント，チェックポイントなどのメタデータの差分も含まれるが，NILFSではこれらのメタデータをファイルとして管理するという独自の手法を採っている．そのファイルをメタデータファイルという．第1の手法ではこのメタデータファイルを拡張属性用に追加する．メタデータファイルを利用する機構は既にNILFSに存在しているため，この手法を用いることによりログへの拡張属性の書き込みやガーベッジコレクションへの対応などの実装コストを低くすることができる．

第2の手法はiノードを拡張属性管理用に利用するというものである．これはUBIFS*の拡張属性管理手法であり，拡張属性が必要になった際にiノードを拡張属性管理用に変更する．この手法はiノードを使用するため，過去のフォーマットとの互換性を保つことができるという利点がある．しかし，拡張属性の増加による総iノード数の減少や拡張属性のサイズがiノードのサイズに限定されてしまうという欠点がある．拡張属性のブロックを拡張属性管理用に変更したiノードから間接的に指すことでサイズの制限を撤廃することも可能であるが，参照・変更時のディスクI/O回数が増加し，効率面での拡張属性の利点を損なう恐れがある．

第3の手法はiノードから直接拡張属性の格納されているブロックをポイントするというものである．これはext3の拡張属性管理手法であり，拡張属性をあるディスクブロックに格納し，拡張属性を設定されたinodeがそのブロックをポイントする．この手法は拡張属性に高速にアクセスすることが可能であるが，拡張属性の数やサイズがディスクブロックサイズに制限されてし

* Linuxで利用可能なフラッシュメモリ向けのファイルシステム．

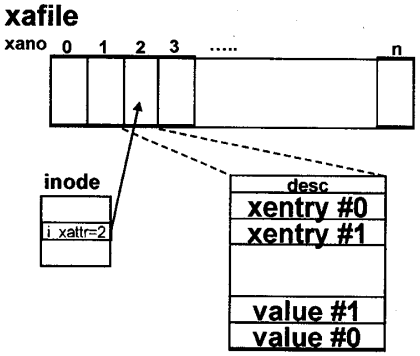

図 2: 拡張属性機能実装の概略

まう．また，拡張属性用に新しいディスクブロックを導入すると，過去のフォーマットとの互換性を保つことができなくなる．

これら 3 つの手法の中から，本論文ではフォーマットの互換性維持や実装コストを考慮して拡張属性用のメタデータファイルを追加する第 1 の手法を選択する．

3 実装

拡張属性の実装の概略を図 2 に示す．本論文では拡張属性用のメタデータファイルを xafile と呼ぶ．xafile はメタデータファイル内のディスクブロック番号を示す xano を指定することで任意のブロックにアクセスすることができる．i ノードはこの xano を保持することで i ノードから xafile 内の拡張属性が格納されているブロックに高速にアクセスすることが可能である．また，同じ拡張属性の組を持つ i ノードが複数存在する場合，それらが同じ xano を保持することでディスク容量を節約することができる．ブロック内には拡張属性の数などの管理情報を格納するディスクリプタと，拡張属性を配置する．拡張属性は xentry 構造体と value に分けられる．xentry には拡張属性の名前に加えて value のポインタやサイズを格納し，value には拡張属性の値を格納する．xentry と value は ext3 の拡張属性管理方法にならい，ブロックの上端と下端から配置する．

拡張属性を操作するシステムコールは以下の 4 つが存在する．

- setxattr　　拡張属性の追加と変更
- getxattr　　拡張属性の値を取得
- listxattr　　設定済拡張属性の名前を取得
- removexattr　　拡張属性を削除

setxattr では指定された拡張属性の名前が存在しない場合は xentry と value をブロック内に配置し，既に存在する場合は value を変更する．getxattr では xentry を順に確認し，名前が一致した xentry が指す value を返す．listxattr では xentry を順に確認し，名前のリストを形成して返す．removexattr では指定された拡張属性を削除する．なお，setxattr と removexattr でブロック内に隙間ができた場合はそれを詰める．

表 1: システムコール処理時間 (単位: μ sec)

	set	get	list	remove
ext3	5.42 7.13	2.90 3,065.12	2.44 3,256.05	5.00 5.30
XFS	961.53 6,060.81	4.19 229.32	4.03 232.78	871.54 6,030.42
NILFS	2.81 6,985.16	2.80 306.99	2.48 429.36	2.73 7,002.15

4 評価

実装量は NILFS の既存のコードへの xafile 追加分が 26 行，拡張属性機能追加分が 757 行となった．

評価として拡張属性操作システムコールの処理時間を ext3，XFS と共に計測した．実験環境は以下の通りである．計測結果を表 1 に示す．

- OS　　Linux 2.6.31-rc5
- CPU　　Intel Xeon E5335 (2.00GHz) x8
- メモリ　　4GB
- HDD　　SATA

各ファイルシステムの上段の行がディスク I/O なしの処理時間，下段の行がディスク I/O を伴う処理時間をマイクロ秒単位で示している．なお，ディスク I/O は set，remove に関してはシステムコール実行後に fsync を実行してディスクへ書き込ませ，get，list に関してはシステムコール実行前にディスクをマウントすることでディスクから読み込みを行うようにした．

表 1 の結果を見ると，NILFS の性能はディスク I/O を伴わない場合は非常に速い処理時間を示し，また，ディスク I/O を伴う場合も XFS とほぼ同等の性能を示していることが分かる．

5 おわりに

本論文では連続スナップショットファイルシステム NILFS に拡張属性機能を実装した．NILFS 特有のメタデータファイルに着目し，拡張属性用のメタデータファイルを追加することで実装のコスト削減とディスクフォーマットの互換性の維持を実現することができた．

現在の実装では拡張属性の数やサイズがディスクブロックサイズに制限されているため，これを修正する必要がある．例えば拡張属性を格納するディスクブロックが飽和状態になった場合はブロック内のディスクリプタが新しい xano を指すことで追加のブロックを利用可能にしたり，1 つのブロックに収まり切らない大容量の value が追加された場合は value を複数のブロック内に分割して配置するといった方法が考えられる．

参考文献

[1] 佐藤考治, 小西隆介, 木原誠司, 天海良治, 盛合敏: ログ構造化ファイルシステム NILFS の設計と実装, 情報処理学会論文誌 コンピューティングシステム, Vol.2, No.1, pp.110-122 (Mar. 2009)