ネットワークの監視技術を用いたファイル更新履歴保存システムの実現

種村昌之[†] 新城靖[††]
板野肯三[††] 千葉滋[††,†††,☆]

この論文は，ネットワークを流れるパケットを監視することによって，ファイルの更新履歴を保存する方法を提案している．パケットの監視にはブリッジを用い，ブリッジにフロー制御を実装することで信頼性があるパケットの取得を実現している．この方法は，ファイルサーバとクライアントの間にファイル更新履歴保存用のコンピュータを追加するだけでよいという利点を持つ．ファイル更新履歴保存システムは，ファイル単位で更新履歴を保存し，本来のファイルと同じようにアクセスすることができる．この論文は，NFS version 2 を対象として，Linux での実装とその性能について述べている．

Preserving File Update History with a Network Monitoring Technique

Masayuki Tanemura,[†] Yasushi Shinjo,[††] Kozo Itano[††] and Shigeru Chiba[††,†††,☆]

This paper proposes the method of preserving file update history with a network monitoring technique. This method achieves reliable packet capturing by using a bridge which has a flow control feature. This method has an advantage of its simple implementation of preserving history by attaching a dedicated machine between a server and clients. The implemented system preserves update history on a file basis, so users can use update history as regular files. This paper shows the implementation of the actual system for NFS version 2 on Linux.

1. はじめに

組織内でコンピュータが増えるとともに，複数のコンピュータでデータを共有するようになった．現在では，ネットワーク上にファイルサーバを構築し，共有するデータを集中的に管理することが多い．

ファイルサーバの障害は大きな影響を及ぼすことになる．そのため RAID などによる媒体の信頼性向上と，データバックアップによる内容の保存が重要である．データが頻繁に更新されるような環境において，こまめにバックアップを行うことが理想である．

しかし，こまめにバックアップを行うことは現実的ではない．それは，バックアップにかかるコストは大きく，かつ，安全なバックアップを保証するためには，バックアップ時に書込みを禁止する必要があるためである．最近では，VxFS（VERITAS File System）[16] や，Solaris8 の UFS スナップショット[14] のように，書込みを禁止せずにバックアップを行うことができるシステムもあるが，これらのシステムでこまめにバックアップを行うには，やはりコストが大きい．また，ネットワーク・アプライアンス社のファイルサーバが採用するログベースのファイルシステムである WAFL（Write Anywhere File Layout）[4] は，低いコストでスナップショットを作成するという面で優れているが，導入するにはファイルサーバを変更する必要がある．LVM[6] に代表されるディスク装置の仮想化によるスナップショットも，同様のことがいえる．

こまめなバックアップとほぼ同様の効果が得られるものとして，ログベースのファイルシステムによる

† 筑波大学大学院修士課程理工学研究科
Master's Program in Science and Engineering, University of Tsukuba
†† 筑波大学電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba
††† 科学技術振興事業団さきがけ研究 21
PREST, Japan Science Technology Corp.
☆ 現在，東京工業大学情報理工学研究科数理・計算科学専攻
Presently with Department of Mathematical and Computing Sciences, Tokyo Institute of Technology

ファイル管理，および Elephant File System [10] で行われるファイルのバージョン管理がある．これらの従来の仕組みは，既存のファイルシステムの変更が必要であった．安全性が求められるファイルサーバを変更することが許されない場合も多い．

そこで我々は，ネットワーク経由で行われるファイルサーバへのアクセスに対して，ネットワークの監視技術を用いることでファイルの更新履歴を保存する方法を提案する[15]．ここでいう更新履歴とは更新前データの集合を意味する（詳しい定義は 3.1 節で述べる）．この方法の利点は，既存のファイルサーバをいっさい変更することなく，更新履歴を保存できるようになるという点にある．それに加えて，この論文で述べる更新履歴保存システムは，次のような利点を持つ．

- 更新時刻に基づいて更新履歴を管理する．
- 保存された履歴は，本来のファイルと同じようにアクセスできる．

我々は，実際にこの提案方法に基づき，NFS version 2 [13] を対象として，Linux においてファイル更新履歴保存システムを実現した．本ファイル更新履歴保存システムは，WAFL や LVM とは異なり，スナップショットではなく更新履歴を残すものであるが，それらの既存のシステムと比較して，導入するには既存の環境に追加するだけでよいという点が優れている．このシステムでは信頼性のあるネットワークパケットの取得を可能にするため，ブリッジを改良している．今回は，100Base Ethernet において実現を行った．その結果，ブリッジという方法でも，提案方式が実現できることが分かった．

本論文では，2 章で関連した研究について述べ，3 章では実現したファイル更新履歴保存システムについて述べる．4 章では実装について述べる．5 章では実現したファイル更新履歴保存システムに実験結果を示し，考察する．

2. 関 連 研 究

Elephant File System [10] はファイルのバージョンを管理し，過去の内容を残すことができるファイルシステムである．このファイルシステムの特徴は，すべての過去のデータを無条件に残すのではなく，ユーザが残す方針を柔軟に指定できることである．また，ユーザは指定したファイルを過去の内容に戻すことができる．このファイルシステムは FreeBSD のローカルのファイルシステムを変更し，過去のファイルの inode をログとして保存している．

Self-Securing Storage [12] は変更操作のログと過去のデータを保持するファイルシステムである．Self-Securing Storage のサーバ（S4）は，S4-NFS Translator と S4 drive で構成され，NFS サーバとして動作する．クライアントは，NFS version 2 で S4-NFS Translator に要求を送ると，S4-NFS Translator は独自の RPC プログラムで，S4 drive に要求を送る．S4 drive はローカルファイルシステムを使わず，独自の形式でディスクにデータを保存する．

Elephant File System も Self-Securing Storage もファイルシステムそのものを置き換えることで過去のデータを保存している．すなわち，これらはまったく独自の形式でデータを保存している．これに対してこの論文で提案するファイルの更新履歴を保存するシステムでは，ファイルシステムの置き換えは必要なく，かつ過去のデータがすべて通常のファイルの形でアクセス可能になっている．このため，導入が簡単であり，さらに過去のデータの他の媒体への移動が容易になる．

ネットワーク監視技術は NIDS [5),17] （Network Intrusion Detection System）の中核技術である．NIDS はネットワークを監視し，通信を解析することでネットワークへの攻撃や侵入を検知する．ネットワーク監視技術の特徴は，監視対象であるシステムにいっさい影響を及ぼさないことである．本論文で提案する方法の特徴はネットワーク監視技術をファイルの更新履歴の保存に用いるところにある．

3. ファイル更新履歴保存システムの機能

3.1 ネットワークの監視による更新履歴の保存

この論文において **更新履歴**とは，時系列に沿った過去のデータの順序集合を意味する．さらに，**ファイル更新履歴保存システム**とは，ファイルに対する更新履歴を保存するシステムである．

我々はファイル更新履歴保存システムを実現する方法として，ネットワークパケットの内容を監視する方法を提案する．この方法はファイルサーバ 1 台に対し，ファイル更新保存システムを 1 台割り当てる．そしてファイル更新履歴保存システムをブリッジとしてネットワークに設置する．この様子を**図 1** に示す．設置されたコンピュータ（以降，**履歴マシン**と呼ぶ）は，取得したネットワークパケットをブリッジとして中継するとともに，そのネットワークパケットをファイル更新履歴を保存するために用いる．

本論文は，NFS version 2 のファイルサーバを対象としたファイル更新履歴保存システムの実現について述べる．

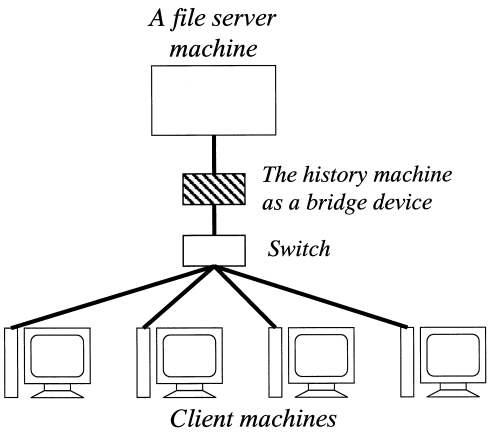


図 1 履歴マシンのファイルサーバマシンへの接続法
Fig. 1 Connection of a history machine to the file server machine.

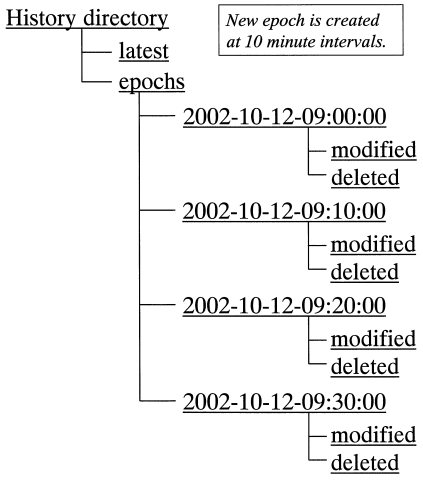


図 2 更新履歴のディレクトリ構造
Fig. 2 A directory structure of the update history.

3.2 信頼性のあるネットワークパケットの取得

本システムは，ネットワークパケットを監視することでファイルの更新を検出する．したがって，ファイルの更新の検出漏れがないように，信頼性のある方法でネットワークパケットを取得する必要がある．

図 1 では，履歴マシンはブリッジであるので，すべてのネットワークパケットを取得できる．しかし，バッファに空きがない場合，パケットを破棄する必要がある．これはクライアントとサーバでは，一時的にブリッジを経由する通信が途絶するように観測される．このような場合，上位のネットワークプロトコル（TCP または RPC）では再送が行われるため，破棄しても問題ない．これにより，履歴マシンでは信頼性のある方法でネットワークパケットを取得することができる．したがって，このネットワークのフロー制御を実現によって，ファイルサーバと同等の書き込むスループットを確保できない場合でも，本システムは更新履歴の保存が可能となる．

3.3 時刻を基にした更新履歴の管理

本システムでは，次の 3 種類のデータを保持する．

(1) ファイルサーバに存在するファイルと同じ名前と内容と属性
(2) ファイルサーバで内容が更新されたファイルの更新前の名前と内容と属性
(3) ファイルサーバで削除されたファイルの名前と内容と属性

ただし，リネームにおいては，リネーム前の名前は保持しない．また，属性の更新に対して，現在の実装では更新前の属性を保存していない．属性の更新を内容の更新と同様に扱うことで，更新前の属性を保存することもできる．特に，(2) と (3) を更新前ファイルと呼ぶ．

本システムは，時間を区切って更新前ファイルを管理する．この区切りをエポックと呼ぶ．1 つのエポックでは，一番古い更新前ファイルだけを保存する．新しくエポックを作るには，`mkepoch` というコマンドを本システムで用意した．

履歴マシンでは，上記 (1) から (3) を，各々次のディレクトリの下に格納する．

(1) `latest`
(2) `modified`
(3) `deleted`

これらのディレクトリは，各々がファイルサーバのルートディレクトリに対応する．そして，それぞれの下にファイルサーバ上のディレクトリ木を再現する．`modified` と `deleted` はエポックごとに生成され，エポックの生成された日時を名前に持つディレクトリの子ディレクトリとなる．これらのディレクトリ構造の例を**図 2** に示す．

`latest` 以下には，最新の（更新後の）ファイルとディレクトリを再現する．このために本システムを最初に稼動させる前にあらかじめファイルサーバからすべてのファイルを `latest` 以下にコピーしておく．

ファイルサーバでファイルの更新が起きると，履歴マシンでは `latest` 以下の対象ファイルを最新のエポックに対応したディレクトリ以下に退避させた後で，`latest` 以下に更新を反映する．また，ファイルサーバでファイルが削除が起きると，履歴マシンでは `latest` 以下の対象ファイルを最新のエポックに対応したディレクトリに移動する．

mkepoch コマンドによりエポックを作成すると，日時を名前に持つディレクトリと modified，deleted だけを作成する．modified と deleted 以下では，ファイルサーバ上のすべてのディレクトリ木を再構成するのではなく，更新前ファイルを保存する際に必要な親ディレクトリだけを作成する．

更新前ファイルを利用する際，ファイルサーバ上のディレクトリ木が再現されているので，検索が簡単である．また，通常のファイルとディレクトリなので，特別な操作を必要とせず，通常の less コマンドを利用したファイル内容のブラウズや，cp コマンドを利用したファイルのコピーが可能である．

本システムでは更新履歴（更新前ファイル）をとるものであり，スナップショットをとるものではない．特定の時期の更新前ファイルは，対象となるエポックに保存されているため，それを参照することで過去の状態に戻すことはできる．たとえば，あるファイルが 2002/12/31 の 10:00 に更新されたとする．元に戻したければ，更新した日時以前で，2002/12/31 の 10:00 に最も近い日時をディレクトリ名とするエポックから更新前ファイルを得て，過去の状態に戻すことができる．

このようなディレクトリ構造を使った方法は，Plan 9[8] のファイルサーバで行われる方法と同じである．Plan 9 では日時を名前とするディレクトリ以下には，名前の日時でのスナップショットが保存されるのに対し，本システムでは，名前の日時以降に行われた更新に対する更新前ファイルを保存している．

3.4 更新前ファイルの保存

本システムは，更新前のデータをファイルとして保存するため，ファイルサーバ以上の容量を備える必要がある．本システムは稼動する時間が長くなるほど必要とする容量が増えていく．したがって，履歴マシンのディスクに入りきらないデータに関して，ディスクからテープや CD-R などの別媒体にデータを移行する必要がある．

本システムでは更新前ファイルを最新のエポックに対応するディレクトリ以下に保存するため，他のエポックに対応するディレクトリ以下のファイルが変更されることはない．この特性を生かして，本システムの稼動中においても，古いエポックに対応するディレクトリをテープや CD-R などの別媒体に安全に移動させることができる．これは，実行時にシステムの排他的なアクセスが必要な dump コマンドによる差分バックアップよりも有用である．たとえば，図 2 の例では，最新のエポックに対応したディレクトリ 2002-10-12-09:30:00 だけがアクセスされ，それ以外のエポックは自由に処理することができる．また，複数のエポックをまとめて，1 つのエポックに圧縮することもできる．

3.5 更新前ファイルの保護

本システムは，Plan 9 と同様にユーザが明示的に削除したファイルを保持し続けるが，これがセキュリティの面から問題になる場合がある．セキュリティを高めるには，まず外部からの侵入を防ぐために，運用時に履歴マシンに対するリモートからのアクセスを禁止する．さらに，IP アドレスを割り当てないことも考えられる．

本システムは，Plan 9 と同様に更新前ファイルをファイルサーバと同じアクセスモードで保存する．この利点は，ファイルサーバと同じように管理できることである．高いセキュリティを要求する環境では，更新前ファイルを暗号化して保存することが考えられる．

4. 実　　装

この章では，前章で述べた方式に基づいて Linux 上に実現した，ファイル更新履歴保存システムについて述べる．ファイル更新履歴保存システムの構成を**図 3** に示す．本システムは，ネットワークパケットを取得するカーネル内ブリッジデバイスドライバと，そのブリッジを扱うためのユーザレベルのライブラリ，および更新履歴保存サーバで構成される．更新履歴は既存のファイルシステム上に保存されるので，ユーザは通常のファイルとしてそれを扱うことができる．

実装環境は Linux kernel 2.4.18 である．

クライアントとサーバの間にコンピュータを追加す

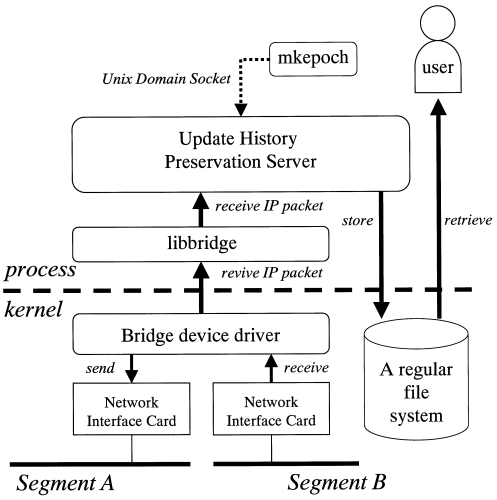


図 3 ファイル更新履歴保存システムの構成
Fig. 3 The structure of the history machine.

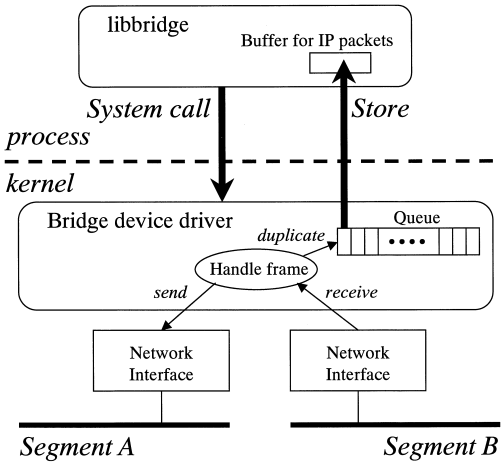


図 4 ブリッジデバイスドライバの構成
Fig. 4 The structure of the bridge device driver.

ることにより更新履歴を保存する方式の実現において，解決すべき重要な問題は次のようなものがある．

- クライアントとサーバの間のパケットを漏れなく取得する
- パケットから手続きを再構成する
- NFS 手続きを解析する
- NFS 手続きで用いられるファイル識別子にファイル名を対応づける

以下の節ではこれらの解決すべき問題について詳しく述べる．

4.1 ブリッジの実装

ブリッジ機能を実装するために，Linux kernel 2.4.18 でソフトウェアとして実装されたブリッジ，および libbridge[2] を改変した．実装したブリッジデバイスドライバの構成を**図 4** に示す．

ブリッジデバイスドライバは，その内部に有限長のキューを持ち，2 つのネットワークインタフェースの間でネットワークパケットを中継する．一方のネットワークインタフェースが受信したネットワークパケットは，ブリッジデバイスドライバの Handle frame 関数に送られる．このネットワークパケットは Linux では `sk_buff` 構造体[9] で管理される．キューは `sk_buff` 構造体のポインタを保持する☆．

Handle frame 関数は，Linux カーネル内の関数 `skb_clone()` を用いてリファレンスカウンタを操作することで，受け取った `sk_buff` 構造体を仮想的に複製し，そのポインタをキューに格納する．その後本来のブリッジの機能に従って，他のネットワークインタフェースに `sk_buff` 構造体を中継する．ただし，キューに格納できなかった場合は `sk_buff` 構造体を破棄し，中継しない．

ユーザレベルで動作する更新履歴保存サーバは，キューから IP パケットをシステムコールで取り出すことができる．このとき，ブリッジデバイスドライバはネットワークパケットから IP パケットを再構成する．ただし，IP フラグメントの再構成は行わない．

これらの動作をユーザレベルのアプリケーションから行えるよう，ブリッジデバイスドライバを制御する libbridge を改変した．ブリッジの制御と IP パケットを取り出すために用意したライブラリ関数を**表 1** に示す．

☆ 現在の実装では，約 32,000 個のネットワークパケットを格納でき，約 50 MB のデータを保存できる．

表 1 ブリッジ制御関数
Table 1 Bridge control functions.

`int br_get_spool_entry()`	IP パケットを 1 つ取り出す
`int br_enable_spool()`	キューイング開始
`int br_disable_spool()`	キューの解放
`int br_clear_spool()`	キューの初期化

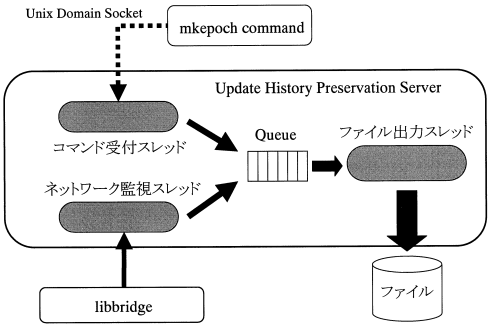


図 5 更新履歴保存サーバの構成
Fig. 5 The structure of the update history preservation server.

4.2 更新履歴保存サーバの実装

更新履歴保存サーバはユーザレベルのアプリケーションとして実装した．これは優先度が指定できるスレッドが必要である．今回は Linux の POSIX スレッド[3] を用いた．

4.2.1 構　　成

実現した更新履歴保存サーバ（以降，**履歴サーバ**と呼ぶ）は**図 5** のように次の 3 つのスレッドと 1 つのキューから構成される．

ネットワーク監視スレッド　ネットワークパケットを取得し，フィルタリングと加工を行い，その結果をキューに入れる．ネットワークパケットの取得には，4.1 節で述べた libbridge を用いる．

ファイル出力スレッド キューからデータを取り出し，更新履歴をファイルとして記録する．

コマンド受付スレッド エポックの作成や統計情報表示といった外部からの要求を受付ける．UNIX ドメインのソケットから得た要求をキューに入れる．

ブリッジのバッファに空きがない場合，ネットワークパケットの中継がとまってしまう．この中継の遮断を最小限にするために，ネットワーク監視スレッドには，他のスレッドより高い優先度を与え，速やかにネットワークパケットをユーザ空間に吸い出すようにする．スレッドの優先度の設定するために POSIX スレッドのスケジューリングポリシーとして `SCHED_FIFO` を使用している．そしてネットワーク監視スレッドには最高の優先順位を，ディスク出力スレッドとコマンド受付スレッドには最低の優先順位を指定している．

4.2.2 RPC の要求・応答の再構成

NFS Version 2 ではサーバとクライアントの通信に UDP/IP 上の Sun RPC[11]（Remote Procedure Call）が用いられる．IP パケットから NFS の要求と応答に再構成するまでには，基本的にはプロトコルスタックに従って，IP パケット，UDP データグラム，RPC メッセージの順に再構築を行えばよい．

ネットワーク監視スレッドは，libbridge からフラグメント化された IP パケットの形でネットワークパケットを得て，それを RPC メッセージに再構成する．さらに RPC メッセージの要求と応答の組をつくり，キューに入れる．SunRPC の再送終了時間が 30 秒であることから，30 秒を過ぎても要求と応答の組がそろわない場合，それを破棄する．また，履歴サーバでは，処理を減らすため，プロトコルスタックのそれぞれのレベルでフィルタリングを行い，不要なデータを破棄する．履歴サーバの特徴は，NFS の手続き番号を用いてフィルタリングを行っている点にある．これについては 4.2.3 項で述べる．

履歴サーバでは，BSD 系 UNIX のカーネルで採用された `mbuf` 構造体[7] が用いる仕組みと同様に，メモリ中のデータのコピーを行うことなく IP フラグメントの再構成を行っている．`mbuf` の方法と比較して履歴サーバの方法の特徴は，IP フラグメントの再構成を NFS の手続きのフィルタリングの後まで遅延している点にある．これにより NFS の手続き番号を用いたフィルタリングが可能になるので，遅延しない方法と比較して，コピーのオーバヘッドを減らすことが可能になっている．

4.2.3 NFS 手続きと更新履歴の保存

ファイル出力スレッドは，RPC の要求と応答の組をキューから取り出し，RPC メッセージの解析を行い，解析結果に基づいて更新履歴を保存する．この過程で，ファイル出力スレッドは，RPC メッセージの解析を行う前に，NFS の手続き番号でフィルタリングを行っている．履歴サーバが利用する NFS Version 2 と MOUNT Version 1 の手続きを**表 2** にまとめる．履歴サーバの特徴は NFSPROC_READ を無視している点にある．これにより処理量を減らしている．この節では，履歴サーバで使われる NFS の手続きとそれに対応した履歴サーバの動作について述べる．

表 2 更新履歴保存サーバが利用する NFS Version 2 と MOUNT Version 1 の手続き
Table 2 NFS Version 2 and MOUNT Version 1 procedures which are used by the update history preservation server.

Procedure	Used or not
NFSPROC_SETATTR	used
NFSPROC_GETTATTR	not used
NFSPROC_LOOKUP	used
NFSPROC_STATFS	not used
NFSPROC_CREATE	used
NFSPROC_WRITE	used
NFSPROC_READ	not used
NFSPROC_REMOVE	used
NFSPROC_RENAME	used
NFSPROC_LINK	used
NFSPROC_SYMLINK	used
NFSPROC_READLINK	not used
NFSPROC_MKDIR	used
NFSPROC_RMDIR	used
NFSPROC_READDIR	not used
MOUNTPROC_MNT	used
MOUNTPROC_DUMP	not used
MOUNTPROC_EXPORT	not used
MOUNTPROC_UMNT	not used
MOUNTPROC_UMNTALL	not used

NFSPROC_LOOKUP はファイルの名前からファイルハンドルを得る手続きである．履歴サーバでは，新しいファイルハンドルが返されるたびにファイルハンドルの対応表に追加する．ファイルハンドルの管理について詳しくは 4.2.5 項で述べる．

NFSPROC_WRITE はファイルへの書込みを行う手続きである．3.3 節で述べたようにファイル更新履歴保存システムでは更新前のデータを残す．履歴サーバは NFSPROC_WRITE を検出すると `latest` 以下のディレクトリにある対象となるファイルを最新のエポックに対応する `modified` 以下のディレクトリにコピーする．そして `latest` 以下のディレクトリの対象となるファイルに対して，要求メッセージに指定された更新を行う．ただし，`modified` 以下のディレクトリにすでにファイルがあるときはコピーを行わない．

NFSPROC_CREATE はファイルを新規に作成する手続きである．履歴サーバでは，対象ファイルを latest 以下のディレクトリに作成する．

NFSPROC_REMOVE はファイルの名前を削除する手続きである．履歴サーバは NFSPROC_REMOVE を検出すると，まず，deleted 以下に削除対象のファイルを移動する．この際，ディスク I/O を減らすため，ファイルコピーでなく，rename() システムコールを用いてファイルを移動して実現する．ただし，削除対象のファイルが別のハードリンクを持っていた場合，そのハードリンクを経由して，delete 以下のファイルの内容が変化する可能性がある．したがってそのような場合，rename() システムコールではなく，ファイルコピーを行うことで更新前ファイルを保存する．こうして保存したあと，latest 以下の削除対象ファイルを削除する．

NFSPROC_LINK と NFSPROC_SYMLINK は，それぞれハードリンクとシンボリックリンクを作成する手続きである．履歴サーバはこれらの手続きを検出すると，latest 以下の対象ファイルに対してそれぞれハードリンクとシンボリックリンクを作成する．こうすることで，ファイルサーバと latest 以下のディレクトリ構造を厳密に同一に維持することができる．ただし，更新前ファイルを保存する modified と deleted に対しては，これらの NFS の手続きでは何もしない．なお，NFS の手続き NFSPROC_WRITE および NFSPROC_REMOVE においては，更新に使われたハードリンクについてのみ，更新前の状態を保存し，他のハードリンクについては何もしない．これは，更新前ファイルを保存する際，対象となるファイルと実体が同じ他のハードリンクを探すことが難しいためである．

NFSPROC_MKDIR と NFSPROC_RMDIR は，それぞれディレクトリを作成または削除する手続きである．履歴サーバはこれらの手続きを検出すると，latest 以下においてぞれぞれディレクトリを作成または削除する操作を行う．

NFSPROC_SETATTR はファイルの属性を設定する手続きである．履歴サーバは標準では latest 以下のファイルの属性を設定するだけである．NFSPROC_WRITE と同様に，ファイル全体をコピーして属性を保存することも可能である．

NFSPROC_RENAME は，ファイルの名前を変更する手続きである．履歴サーバは標準では，latest 以下のファイルの名前を変更するだけである．

MOUNTPROC_MNT は，NFS クライアントが NFS サーバに対してマウントポイントとなるディレクトリのファイルハンドルを要求する手続きである．履歴サーバは，要求に含まれている NFS サーバ上のディレクトリ名と，応答に含まれているファイルハンドルを，ファイルハンドルの対応表に追加する．ファイルハンドルの管理について詳しくは，4.2.5 項で述べる．

4.2.4 mkepoch コマンドとエポックの作成

ファイルサーバで手続きが行われ，応答が返ってくると，履歴サーバのネットワーク監視スレッドはキューに要求・応答の組を追加する．キューにたまっている状態で mkepoch コマンドを実行しても，すぐにエポックを作成するのではない．まず，コマンド受付スレッドが mkepoch コマンドからエポックを作成する要求を受け取ると，それをキューに追加する．ディスク出力スレッドは，要求・応答の組，または，エポック作成のための命令をキューから取り出して，それを 1 つずつ逐次的に処理する．すなわち，ディスク出力スレッドがエポックを作成する時点では，mkepoch コマンドの実行以前にファイルサーバで行われた手続きはすべて処理されている．また，本システムでは前回の更新から 10 秒以上☆経過したファイルのみ，更新前のファイルを保存するため，連続した書込みを行う複数の手続きの間に mkepoch を実行した場合でも，書込み途中のファイルが更新前ファイルとして保存されることはない．

4.2.5 名前とファイルハンドルの管理

NFS ではクライアントとサーバの間で操作対象のファイルやディレクトリを指定するために，名前でなくファイルハンドルという 32 バイトの識別子が用いられる．3.3 節で述べたように，本システムは履歴マシン上で NFS サーバのディレクトリ構成を再構築する．そのためにファイルハンドルから NFS サーバ上のファイル名を調べる必要がある．これを行うために，内部的に関数 getpathbyfh() を定義している．関数 getpathbyfh() は引数にファイルハンドルを取り，NFS サーバ上の絶対パス名を返す．

この関数の実装には，次のエントリを含むファイルハンドルと名前の対応表を用いている．

- ファイルハンドル

☆ Elephant File System の実証実験では，実際のファイルサーバの置換えが許可されなかった．そこで NFS の手続きを基にローカルのファイルアクセスを再現して実証実験を行った．Elephant ではファイルのクローズという操作を捕捉する必要があるが，NFS にはクローズはない．そこで 10 秒以上経過したものをクローズされたと見なしている．本システムでも同じ 10 秒という値を用いた．

- 親ディレクトリのファイルハンドル
- ファイル名
- 手続き MOUNTPROC_MNT に現れたディレクトリかどうかのフラグ

対応表の各エントリは，ファイルハンドルをキーとして高速にアクセスできるようにしている．この関数は，与えられたファイルハンドルで対応表を検索し，そのエントリを得る．そして `getcwd()` ライブラリ関数と同様に，再帰的に親ディレクトリのファイルハンドルを得て，手続き MOUNTPROC_MNT で現れたファイルハンドルにぶつかるまでディレクトリ名に対応させていく．最後に手続き MOUNTPROC_MNT で現れたファイルハンドルを検出したとき，保存してあるディレクトリ名を結果として返す．

ファイルハンドルとファイル名の対応は次のような場合に動的に変化する．

- ファイルが作成された場合
- リネームによってファイルハンドルに対するファイル名が変わった場合
- ファイルの削除などによりファイルハンドルが無効になった場合

本システムでは NFS クライアントと NFS サーバの要求と応答に応じてファイルハンドルとファイル名の対応表を更新し，ファイルが削除された場合にファイルハンドル情報を消す．

5. 実　　験

本システムのオーバヘッドを調べるために，次の 3 種類の接続方法について実験を行った．

直接接続 NFS サーバとクライアントを直接接続する

ソフトウェアブリッジ経由 ネットワークパケットを格納しないソフトウェアのブリッジを経由して NFS サーバとクライアントを接続する

履歴マシン経由 本ファイル更新履歴保存システムを経由して NFS サーバとクライアントを接続する

実験に用いた NFS サーバ，NFS クライアント，および履歴マシンの基本構成を次に示す．

CPU Intel Pentium III 1 GHz
RAM 512 MB SDRAM
HDD 7200 rpm, ATA100 75GB
Network card 100Base-TX
OS Linux kernel 2.4.18
File system Ext2 file system

表 3 最短応答時間
Table 3 Minimum response time.

接続方法	1 byte [ms]	1,024 byte [ms]
直接接続	0.192	0.365
ソフトウェアブリッジ経由	0.272	0.536
履歴マシン経由	0.272	0.536

実験では，NFS サーバに非同期☆書込みを許し，NFS の書込みスループットを高くした状態で行った．クライアント 1 台と 2 台で比較したところ，1%未満の違いしかみられなかった．そこで，実験で用いるクライアントは 1 台とした．

はじめに，どの程度の細かさでエポックを作成することができるかを計測した．人間が利用する状況において，ファイルを保存する頻度が，1 秒より短くなることは考えにくいので．目標は 1 秒につき 1 回に設定した．

負荷をかけていない状態で，`mkepoch` コマンドを 10,000 回実行するのに 27.9 秒必要とした．これは，目標を十分に満たしている．

通常のシステムコールを用いたプログラムでは，カーネルによるバッファリングのため，応答時間を計測することができなかった．そこで，応答時間を計測するために Sun RPC を直接実行する NFS クライアントプログラムを作成した．この計測プログラムは与えられたバイト数を NFSPROC_WRITE で書き込み，応答時間を計測する．

100 回計測したときの最短の応答時間を**表 3** に示す．ソフトウェアブリッジ経由とファイル更新履歴保存システム経由の値に差はない．これは 4.1 節で述べたように，ネットワークパケットを仮想的にコピーしているためである．また，これらと直接接続との間では，1 byte の場合で 0.08 ms，1,024 byte の場合で 0.171 ms の差がある．この差の主な要因はブリッジで中継する際のネットワークパケットのコピーである．

本システムは，NFS の読込みを無視する．通常の読込みの多い状況[1] では NFS サーバよりも負荷が低い．本システムの負荷が NFS サーバよりも高くなるような，次の 3 つの実験を行った．

copy NFS クライアントにおいて，NFS サーバ上の 10 MB のファイル 100 個をコピーする．

write NFS クライアントにおいて，10 MB のファ

☆ Linux には NFS Version 2 において同期（sync）と非同期（async）という考え方がある．同期とは，ファイルに対するすべての書込みにおいて，対応するハードウェアに物理的に書き込まれるまで呼び出し元に制御が戻らないというものである．同期の場合，書込みスループットは 500 KB/s 程度になる

表 4 実行時間
Table 4 Excution time.

接続方法	copy [s]	write [s]	over write [s]
直接接続	268.4	95.6	97.0
ソフトウェアブリッジ経由	272.8	97.3	97.6
履歴マシン経由	274.1	98.1	145.9
履歴マシン経由（1 秒ごとに mkepoch）	275.6	100.9	165.3

イルを 100 個書き込む．

over write NFS クライアントにおいて，NFS サーバ上の 10 MB のファイル 100 個に，10 MB のファイル 100 個を上書きする．

この実験において，実験前に実験に用いるコンピュータすべてを再起動し，また，ファイルシステムを初期化する．そして，コピー元のファイルを新たに作成している．

実験の結果を**表 4** に示す．直接接続に対してソフトウェアブリッジ経由では，実行時間が copy で 1.7%，write で 1.8%低下した．直接接続に対して履歴マシン経由では，copy で 2.1%，write で 2.6%の低下した．ソフトウェアブリッジ経由と履歴マシン経由の差，copy の 0.4%，write の 0.8%がファイル更新履歴保存システムの純粋なオーバヘッドである．1 秒ごとに `mkepoch` コマンドを実行した場合，copy で 2.7%，write で 5.5%低下する．write のオーバヘッドが大きい理由は，エポックあたりのディスクへの書込みが多いためである．copy と write では，本システムがボトルネックになっていない．over write の場合では，実行時間が 1.5 倍になった．これは更新前ファイルの保存を行う処理のため，本システムの負荷が大きくなっているためである．このように，本システムがボトルネックになるような状況であっても，正しく更新履歴を保存することができる．

この実験から，汎用の PC とソフトウェアを用いたブリッジを用いるという方法でも，提案方式が実現できることが分かった．パケット中継のオーバヘッドを減らすには，ブリッジによりパケットを取り込む部分をハードウェア化する方法が考えられる．

6. おわりに

本論文では，ネットワークを流れるパケットを監視することによって，ファイルの更新履歴を保存する方法を提案し，ブリッジを用いた実現について述べた．この方法の利点は，ファイルサーバをいっさい変更することなく，ファイルの更新履歴を保存できるようになるということである．さらに，履歴が時刻に基づいて管理され，通常のファイルの形で整理・保存されているため，履歴のブラウズや，他の媒体への移動を安全かつ容易に行うことができるという利点も持つ．

参考文献

1) Baker, M.G., Hartman, J.H., Kupfer, M.D., Shirriff, K.W. and Ousterhout,J.K.: Measurements of a Distributed File System, *Proc. 13th Symposium on Operating Systems Principles* (1991).
2) bridge sourceforge: libbridge (2002). http://bridge.sourceforge.net/
3) Butenhof, D.R.: *Programming with POSIX Threads*, Addison-Wesley (1997).
4) Dave Hitz, N.: A Storage Networking Appliance, *NetApp Library Technical Reports TR3001* (2000).
5) Handley, M. and Paxson, V.: Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics, *Proc. 10th USENIX Security Symposium* (2001).
6) Lewis, A.: LVM HOWTO (2002). The Linux Documentation Project.
7) McKusick, M.K., Bostic, K. and Karels, M.J.: *The Design and Implementation of the 4.4BSD Operating System*, Addison-Wesley (1996).
8) Pike, R., Presotto, D.K.T. and Trickey, H.: Plan 9 from Bell Labs, *Proc. Summer 1990 UKUUG Conference*, pp.1–9 (1990).
9) Rubini, A.: *LINUX DEVICE DRIVER*, O'Reilly & Associates, Inc. (1998).
10) Santry, D.S., Feeley, M.J., Hutchinson, N.C., Veitch, A.C., Carton, R.W. and Ofir, J.: Deciding when to forget in the Elephant file system, *Proc. 17th ACM Symposium on Operating Systems Principles*, pp.110–123 (1999).
11) Srinivasan, R. and Sun Microsystems, Inc.: RPC: Remote Procedure Call Protocol Specification Version 2, *RFC1831* (1995).
12) Strunk, J.D., Goodson, G.R., Scheinholtz, M.L., Soules, C.A. and Ganger, G.R.: Self-Securing Storage: Protecting Data in Compromised Systems, *Proc. 4th Symposium on Operating System Design and Implementation* (2000).
13) Sun Microsystems, Inc.: NFS: Network

File System Protocol Specification, *RFC1094* (1989).
14) Sun Microsystems, Inc.: Solaris 8 1/01 Update Collection, Solaris 8 のシステム管理 (2001).
15) 種村昌之，新城 靖，千葉 滋，板野肯三：ネットワークの監視によるバックアップシステムの実現，情報処理学会コンピュータシステム・シンポジウム，Vol.2001, No.16, pp.57–64 (2001).
16) Veritas, Inc.: VEARITAS File System (2002).
17) Zhang, Y. and Paxson, V.: Detecting Stepping Stones, *Proc. 9th USENIX Security Symposium* (2000).

(平成 14 年 12 月 23 日受付)
(平成 15 年 4 月 13 日採録)

種村 昌之（学生会員）

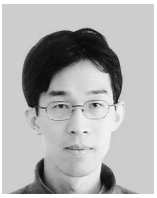
1978 年生．2001 年筑波大学第三学群情報学類卒業．2003 年同大学院修士課程理工学研究科修了．現在，同大学院博士課程システム情報工学研究科在学中．データ圧縮技術の研究に従事．電子情報通信学会学生員．

新城 靖（正会員）

1965 年生．1993 年筑波大学大学院博士課程工学研究科電子・情報工学専攻修了．同年琉球大学工学部情報工学科助手．1995 年筑波大学電子・情報工学系講師，2003 年同助教授．分散型オペレーティング・システム，並列処理，情報セキュリティの研究に従事．1995 年情報処理学会山下記念研究賞受賞．博士（工学）．日本ソフトウェア科学会，ACM，IEEE CS 各会員．

板野 肯三（正会員）

1948 年生．1977 年東京大学大学院理学系研究科物理学専門課程単位取得後退学．1993 年より筑波大学電子・情報工学系教授．計算機アーキテクチャ，分散システム，プログラミングシステム等に関する研究に従事．理学博士．日本ソフトウェア科学会，電子情報通信学会，ACM，IEEE CS 各会員．



千葉 滋（正会員）

1968 年生．1991 年東京大学理学部情報科学科卒業．1993 年同大学院理学系研究科情報科学専攻修士課程修了．1996 年同専攻博士（理学）取得．同専攻助手，筑波大学電子・情報工学系講師，東京工業大学情報理工学研究科講師を経て，現在同助教授．言語処理系およびオペレーティングシステム等システムソフトウェアの研究に従事．日本ソフトウェア科学会，ACM 各会員．