ジャーナリングファイルシステムの構造を利用した非同期リモートミラーリングの高速化

藤田智成[†] 矢田浩二[††]

本稿は，ネットワークで接続されたストレージシステム間で実行する，非同期のリモートミラーリングの高速化手法，TARM を提案する．本手法は，ローカルサイトのストレージシステムが，ファイルシステムのディスク上のデータ構造に関する知識を利用することで，計算機やリモートストレージシステムの特別な機能を利用せずに，ミラーリング先データがファイルシステムとして利用可能な状態であることを保証する．ジャーナリングファイルシステムでは，ファイルシステムの更新情報を記録するジャーナルのディスク上のデータ構造に関する知識を利用する．Linux 用の 2 種類のファイルシステム，ext3，reiserfs に対して動作するストレージシステムを実装し，商用環境の負荷を模擬するベンチマークを使って性能を評価したところ，すべてのデータの更新順序を指定するミラーリング手法と比較して，2.03〜5.13 倍のスループット性能が得られた．

Asynchronous Remote Mirroring with Journaling File Systems

Tomonori Fujita[†] and Kouji Yata[††]

In this paper we describe TARM, a new asynchronous remote mirroring technique that asynchronously replicates data across multiple storage sites at the storage system level. This technique can be implemented inside a local (primary) storage system and enables remote (secondary) storage systems to keep replicated data recoverable at all times, regardless of catastrophic site failures, but without using specialized features between the storage systems or sacrificing performance. This is achieved by using the file system information such as its on-disk data structures. In particular, in journaling file systems, we require enough knowledge of file systems log format to identify commit records. We describe an algorithm that works with two popular Linux journaling file systems, ext3 and reiserfs. Our experiments show that TARM outperformed the straightforward mirroring method that writes all data in a precise order on a remote storage system by 2.03–5.13 times under realistic workloads.

1. はじめに

多くの企業が，地震や火災，洪水等の自然災害，テロ等の人的災害等，拠点が失われるような大規模な災害が発生した場合でも，データの損失を防ぎ，迅速に業務を再開するためのソリューション（ディザスタリカバリ）の導入を進めている．

ディザスタリカバリソリューションで，地理的に離れた複数の拠点のデータを，ネットワークを利用して同期するために用いられる技術が，リモートミラーリングである．

リモートミラーリングは，ファイルシステム，計算機のデバイスドライバ，ストレージシステム等で実現することができるが，本稿は，最も広く用いられている，ストレージシステム間で実行するリモートミラーリングについて議論する．

古典的なリモートミラーリングは，すべてのストレージシステムのデータをつねに同期する．主ストレージシステムと遠隔地の副ストレージシステムの両方が，データを保存した後に，計算機は書き込み完了通知を受け取る．このような同期型リモートミラーリングは，書き込み性能の悪化を避けるために，専用線等，高品質なネットワークが必要になるという欠点を持つ．企業の規模にかかわらず，計算機のデータの重要性が高まっているため，安価に実現できるリモートミラーリング技術が必要である．

非同期型のリモートミラーリングでは，主ストレージシステムは，計算機が更新したデータを保存すると，副ストレージシステムにデータを転送する前に，計算機に書き込み完了を通知する．その後，計算機の動

† NTT サイバーソリューション研究所
NTT Cyber Solutions Laboratories
†† NTT サービスインテグレーション基盤研究所
NTT Service Integration Laboratories

作とは独立に，主ストレージシステムは，副ストレージシステムにデータを転送する．非同期型リモートミラーリングは，災害で主ストレージシステムが失われた場合に，副ストレージシステムにまだ転送されていなかった最新のデータを失う危険性と引き替えに，高い書き込み性能を実現する．

非同期リモートミラーリングでは，計算機のファイルシステムが主ストレージシステムのデータを更新した順序と，副ストレージシステムのデータ更新順序が一致する必要がある．主ストレージシステムのデータ更新と異なる順序で，副ストレージシステムのデータを更新し，ミラーリング中に主ストレージのデータが失われた場合，副ストレージシステムのデータは，ファイルシステムが予期していない順序で更新されている状態となり，利用できなくなる可能性がある．

副ストレージシステムのすべてのデータの更新順序を指定する非同期リモートミラーリング手法（以降，基本ミラーリング手法と呼ぶ）の欠点は，副ストレージシステムがすべての更新をシリアライズするため，その性能が低下することである．この性能低下を避けるために，商用のストレージシステムが利用する手法は，副ストレージシステムのアトミック（不可分）なデータ更新機能，計算機と各ストレージシステムに実装したミラーリング制御機能を用いる．しかし，この手法では，計算機，各ストレージシステムにベンダ独自の機能が必要であるため，ストレージシステムの選択に制限が生じる．したがって，オープンソースソフトウェアを使って実装したストレージシステム[1]や，ストレージサービスプロバイダが提供するストレージシステムを，副ストレージシステムとして利用することが困難になり，システム構築費用が高価になる．加えて，既存手法は，計算機のオペレーティングシステムへ変更を要する点が，リモートミラーリングシステムを現在稼働中の計算機環境に導入するうえでの障害となる．

そこで我々は，主ストレージシステムに実装することで，計算機や副ストレージシステムの特別な機能を利用することなく，ミラーリング先のデータをファイルシステムが利用できる状態に保つことのできる，高速な非同期リモートミラーリング手法，TARM（Transparent Asynchronous Remote Mirroring）を提案する．計算機が発行するブロックレベルの読み書き命令を単純に実行する従来のストレージシステムと異なり，TARM は計算機が利用しているファイルシステムのデータ構造に関する知識とブロックの内容を利用して，副ストレージシステムのデータをファイルシステムが利用できる状態に保つために必要な更新順序を判断し，更新順序に制限がないデータの副ストレージシステムでの更新を並列化することで，高速なミラーリングを実現する．

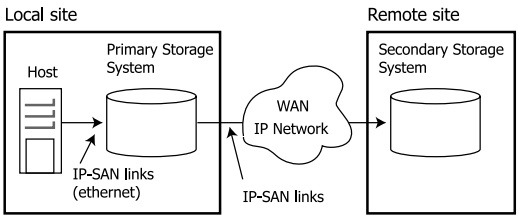


図 1 リモートミラーリングシステム構成例
Fig. 1 Typical remote mirroring deployment.

更新順序の判断方法はファイルシステムによって異なる．本稿では，Linux オペレーティングシステムの ext3 と reiserfs，2 種類のファイルシステムについて，更新順序の判断方法を述べる．

本稿の構成は以下のとおりである．まず，2 章で非同期リモートミラーリングについて述べる．次に，3 章で提案手法を説明する．4 章では，性能評価結果を示す．5 章で関連研究について言及し，6 章でまとめる．

2. 非同期リモートミラーリング

2.1 システム構成

説明を単純化するために，本稿では，主ストレージシステムと副ストレージシステムが 1 対 1 の関係である設定に集中する．また，提案手法は，すべてのブロックレベルのストレージプロトコルに適用可能であるが，広く用いられている，iSCSI プロトコル[2]を使って，説明を進める．

図 1 は，iSCSI プロトコルを使った，典型的なリモートミラーリングシステムの構成を示している．iSCSI プロトコルは，計算機とストレージシステムを Ethernet で接続し，SCSI コマンドを TCP/IP パケットに包んで転送する．iSCSI プロトコルを使うことで，計算機とストレージシステムを接続している SCSI ケーブルを Ethernet ケーブルに置き換えることができる．

図中のローカルサイトに配置された，計算機と主ストレージシステムは，Storage Area Network（SAN）と呼ばれるストレージアーキテクチャを用いている．Ethernet で接続された計算機と主ストレージシステムは，iSCSI プロトコルを使って通信する．ローカルサイトと IP ネットワークで接続されたリモートサイトに配置される副ストレージシステムも iSCSI プロトコルを使って，主ストレージシステムと通信する．

計算機で動作するオペレーティングシステムは，

iSCSI ストレージシステムを，PCI バス等のシステムバスと SCSI ケーブルで接続される従来のディスク装置と同様に認識する．したがって，Direct Attached Storage（DAS）と呼ばれる，従来のディスク装置を利用することを想定している，すべてのローカルファイルシステムやデータベースは，DAS の代わりに，iSCSI ストレージシステムを利用することができる．ここでのローカルファイルシステムとは，NFS や CIFS と異なり，ネットワークを想定していないファイルシステムを指す．TARM は，計算機がファイルシステムを利用して，iSCSI ストレージシステムにアクセスするシステム構成を対象としている．

iSCSI プロトコルでは，SCSI プロトコルと同様に，SCSI コマンドを発行し，サービスを要求する存在をイニシエータと呼ぶ．一方，イニシエータから SCSI コマンドを受け取り，サービスを提供する存在を，ターゲットと呼ぶ．したがって，図 1 では，計算機と主ストレージシステムがイニシエータとターゲットの関係である．さらに，主ストレージシステムと副ストレージシステムも，イニシエータとターゲットの関係である．

2.2 ファイルシステムの破壊

ファイルシステムは，停電等による予期せぬシステムのクラッシュ後に，ディスクに保存されたデータがファイルシステムとして利用不可能な状態になることを避けるために，正確な順序で，ディスクのデータを更新する必要がある．

ファイルにデータを追記する際に，新たに間接ブロック☆が使われた状況を考える．この場合，初期化した間接ブロックを更新してから，i ノードを更新する必要がある．逆の順番で更新し，2 つの更新が完了する前に，システムがクラッシュすると，i ノードが指している初期化されてない間接ブロックは，誤った物理ブロック番号を含んでいる可能性がある．そのブロックが，すでに他のファイルのデータブロック☆☆として使われていた場合，そのブロックは 2 つのファイルから使われている状態になる．ファイルシステムは，どちらのファイルがそのブロックの正当な所有者であるかを知る方法がないため，この状態を修復することは不可能である．

上記のように，ファイルシステムが利用できなくなる状況は，メタデータの操作時に発生する．メタデータの整合性が保たれており，ファイルシステムとして利用できる状態にあることは，メタデータの一貫性がある，と呼ばれる．

非同期リモートミラーリングで，主ストレージシステムでのデータ更新順序と，副ストレージシステムのデータ更新順序が一致しない場合，副ストレージシステムに保存されたデータが，ファイルシステムとしてメタデータの一貫性がない状態，つまり，利用できない状態になる可能性がある．

計算機で動作するファイルシステムが，主ストレージシステムのブロック 1 およびブロック 2 をこの順序で更新する必要がある状況を考える．この場合，計算機は，主ストレージシステムにブロック 1 の更新を要求し，その完了通知を受け取ってから，ブロック 2 の更新を要求する．

その後，主ストレージシステムがそれらのデータを副ストレージシステムに転送する．副ストレージシステムは，ブロック 1 のデータの更新を開始し，すぐに，ブロック 2 のデータの更新も開始することができる．これらのデータの更新順序は指定されていないため，2 つのブロックがディスクドライブに書き込まれる順序は不定である．

上記のデータ更新中に，災害によって，主ストレージシステムのデータが完全に失われ，加えて，副ストレージが予期せぬ障害によって停止した状況を考える．もし，副ストレージシステムで，ブロック 1 の更新が未完了で，ブロック 2 の更新だけが完了していると，副ストレージシステムのデータは，ファイルシステムとして利用できない状態となる．ブロック 1 の更新データとブロック 2 の更新前のデータはどこにも保存されていないため，副ストレージシステムのデータをファイルシステムとして利用できる状態に回復することは不可能である．

リモートミラーリングで，副ストレージシステムに保存されたデータを利用できる状態に保持する基本的な手法では，主ストレージシステムと副ストレージシステムで，すべてのデータ更新順序を一致させる．ファイルシステムは，メタデータの整合性を保つことのできる順序で主ストレージシステムを更新するため，副ストレージシステムのデータも，つねに，ファイルシステムとして利用可能な状態であることが保証される．しかし，この手法は，副ストレージシステムがデータを並列に更新できないため，ミラーリング性能が低い．

商用システムが用いる，性能の低下をともなわずに，データ破壊を避ける方法では，副ストレージシステムが NVRAM 等の特殊なハードウェアを用いて，ア

☆ 本稿では，特に断らない限り，ブロックとはファイルシステムブロックを意味する．

☆☆ ファイルシステムブロックは，ユーザの保存したデータ（ファイルデータ）を保存するデータブロックとファイルシステム構造に関するデータを保存するメタデータブロックに分類される．

トミックにデータを更新する．上記の例では，副ストレージシステムでは，ブロック 1 とブロック 2 の両方が更新されているか，そうでなければどちらも更新されないことが保証されるため，データがファイルシステムとして利用不可能な状態になることはない．しかし，アトミックな更新機能をサポートしていない，安価なストレージシステムも多く存在する．また，特殊なハードウェアを用いることは，コスト面で不利である．

3. 非同期リモートミラーリングの高速化手法

ファイルシステムが，メタデータの一貫性を保持するために，正確な順序で更新する必要があるのは，一部のデータのみである．しかし，主ストレージシステムは，更新を要求されたデータが更新順序に制限を持っているかどうかを判断することができない．そのため，副ストレージシステムの，すべてのデータ更新に順序制限が生じる．

TARM では，主ストレージシステムが，ファイルシステムの構造に関する知識を持ち，ファイルシステムが保存しているブロックの内容から，更新順序情報を判断し，更新順序に制限があるデータのみ，副ストレージシステムで正確な順序で更新する．副ストレージシステムは，主ストレージシステムが更新順序を指定しないデータを並列に更新するため，高速にミラーリングを実行可能である．

主ストレージシステムは，iSCSI プロトコルの標準機能を使って，副ストレージシステムでのデータ更新順序を指定する．したがって，副ストレージシステムにリモートミラーリングのための特別な機能は必要なく，すべてのストレージシステムを副ストレージシステムとして利用することが可能である．

TARM は，主ストレージシステムを利用している計算機に対して透過的であり，計算機のハードウェアや，ファイルシステム，ブロックレベルのプロトコルに変更を必要としない．これは，そのようなコンポーネントに変更を加える非透過的なアプローチ[3] と比較して，初期導入コスト，長期的な管理コスト面等で，大きなアドバンテージである（特に主ストレージシステムに接続される計算機の数が多い場合）．

透過的なアプローチを選択したことにともなう欠点として，TARM はファイルシステムの構造に強く依存する．しかし，ファイルシステムは，後方互換性保持のため，その構造を変えることはほとんどない．したがって，あるファイルシステムに一度対応すれば，その後の変更が必要となることは稀であり，この欠点は大きな問題とならない[4]．

リモートミラーリングは，論理ボリューム単位で行われる．1 つのストレージシステムは，内部に，複数の論理ボリュームを持つことができる．1 つの論理ボリュームは単一のディスクドライブ，または，複数のディスクドライブから構成される．

TARM は，I/O 命令送信時の順序保証と損失検出を提供するブロックレベルのストレージプロトコルを対象としている．iSCSI プロトコルは，ネットワークの特性に関係なく，独自の順序制御と TCP プロトコルの特性を利用することで，これら 2 種類の機能を実現している．

計算機のデータ更新速度よりも，ストレージシステム間のスループットが大きいという条件を除いて，TARM はネットワークの性質（帯域，遅延，ジッタ，パケットロス等）に関係なく動作する．ただし，ネットワーク特性は，リモートミラーリングシステムの管理者による，障害の有無の判断に影響する可能性がある（3.3.4 項で説明する）．

3.1 リモートミラーリングのデータ一貫性モデル

非同期のリモートミラーリングが提供する，副ストレージシステムに保存されたデータの一貫性モデル☆は，無保証，停電一貫性，アプリケーション一貫性の 3 種類に分類することができる[5]．

無保証は，最も弱いデータ一貫性モデルである．主ストレージシステムのデータを一定の間，変更しなければ，副ストレージシステムのデータは主ストレージシステムのデータと一致するが，同期するまでの間の副ストレージシステムのデータの状態に関しては何も保証されない．したがって，ファイルシステムのようにデータの更新順序に制限を持つ場合，無保証のデータ一貫性モデルは，副ストレージシステムのデータがファイルシステムとして利用できなくなる状態をまねく可能性があり，商用環境での利用には適さない．

停電一貫性は，多くのリモートミラーリングシステムが提供するデータ一貫性モデルである．副ストレージシステムのデータは，予期せぬ計算機のシステムクラッシュ後のストレージシステムのデータと同様の状態であることが保証される．したがって，副ストレージシステムのデータはファイルシステムとして利用可能な状態であることが保証される．主ストレージシステムが失われ，副ストレージシステムのデータをファ

☆ リモートミラーリングのデータ一貫性モデルは，ストレージシステムが実現する，副ストレージシステムのデータの状態に関するものであり，ファイルシステムが実装する，メタデータ一貫性やファイルデータ一貫性の保証機能とは異なるものである．

イルシステムとして利用する場合，ファイルシステム自身によるデータ復旧作業が必要である（fsck 等のツールが用いられる）．

多くのリモートミラーリングシステムと同様，TARM は停電一貫性を保証する．リモートミラーリング実行中のどの時点で，主ストレージシステム，ストレージ間のネットワーク，副ストレージシステムに障害が発生しても，副ストレージシステムのデータは，ファイルシステムとして利用できる状態である．

アプリケーション一貫性は，最も強いデータ一貫性モデルである．副ストレージシステムのデータは，ファイルシステムの正常なシャットダウン（アンマウント）後の主ストレージシステムのデータと同様の状態であることが保証される．主ストレージシステムが失われた後，ファイルシステムが復旧作業をすることなく，副ストレージシステムのデータをファイルシステムとして即時に利用することができる．

3.2 ファイルシステムの選択

本手法が対象とするファイルシステムは，主ストレージシステムが，計算機の仲介なしに，ブロックの内容だけを使って，更新順序情報を判断できなければならない．さらに，副ストレージシステムに保存された停電一貫性状態のデータを通常状態へ素早く復旧できる機能を備えていることが望ましい．

ファイルシステムが，高速復旧機能のために用いる技術としては，ジャーナリング[6]，Soft Updates[7]，no-overwrite[8),9)] があるが，我々は，ジャーナリングファイルシステムを TARM の対象として選択した．その理由は，(1) 保存されたデータから更新順序情報が容易に判断可能なこと，(2) 商用環境で使われているオペレーティングシステムの標準ファイルシステムの大半がジャーナリングファイルシステムであること，(3) 更新順序の制限が少ないこと（高速なミラーリングを実現できる），である．

本稿では，Linux オペレーティングシステムの 2 種類のジャーナリングファイルシステム，ext3，reiserfs を対象としたアルゴリズムについて説明する．

3.2.1 ジャーナリングファイルシステム

現在，一般的に使われているファイルシステムのすべてが，非同期にデータを更新する．ファイルシステムがそのブロックを更新する際，更新したブロックの内容は，ディスクに書かれる前，一定時間，メモリにバッファされる（遅延書き込み）．

2 章で説明したように，ファイルシステムが使用できない状態になることを防ぐためには，メタデータへの変更を正確な順番でディスクに書き込まれなければならない．

古典的なファイルシステムは，更新順序に制限があるブロックを同期的に更新し，ファイルシステムの復旧時には，fsck と呼ばれるツールを使い，ファイルシステム全体の内容を検査していた．この手法には，メタデータ更新の性能が悪く，復旧作業時間が長い，という欠点がある．

ジャーナリングファイルシステムは，メタデータ操作をジャーナルと呼ばれるディスク領域に記録し，その書き込みが終了してから，ファイルシステム本体の実際のメタデータを更新する．ジャーナルへの記録が終了しないうちに障害が発生した場合，ファイルシステム復旧時に，そのメタデータ操作は単に無視される（ファイルシステム本体の実際のメタデータはまだ変更されていない）．また，ファイルシステム本体のメタデータの更新中に障害が発生した場合は，計算機の再起動後に，ファイルシステムは，ジャーナルに保存されたメタデータを使って，実際のメタデータの更新を完了させる．つまり，複数のメタデータをアトミックに更新することで，同期更新と復旧のためのファイルシステム全体の検査を避ける．

ジャーナリングファイルシステムには，ジャーナルへメタデータ操作を記録してから実際のメタデータを更新するという，更新順序の制限がある．TARM は，ジャーナルのディスク上のデータ構造に関する知識を利用することで，ジャーナルへのメタデータの記録が完了した時点，つまり，更新順序制限に関する情報を知ることができる．

一般に，ジャーナリングファイルシステムは，メタデータの一貫性は保持するが，システム障害が発生した場合のファイルデータの欠損については関知しない☆．更新したファイルデータの一部だけが更新された状態や，ファイルにデータを追記した際に，i ノードかデータのどちらかのみが更新される状態，等が発生する可能性がある．

後述するように，ファイルデータへの変更もジャーナルに記録することで，ファイルデータもアトミックに更新されることを保証するジャーナリングファイルシステムもある．ファイルデータもアトミックに更新されている状態のファイルシステムを，ファイルデータの一貫性が保持されていると呼ぶ．3.3.4 項で説明するように，TARM では，ファイルシステムがファイルデータの一貫性も保持するように動作している場

☆ メタデータと異なり，ファイルデータが欠損しても，ファイルシステムが利用不可能ならないため．

合，ミラーリング先のデータも，ファイルシステムとして，ファイルデータの一貫性が保持された状態であることが保証される．

3.2.2 ext3

ext3 は，Linux で最も広く使われている，標準的なファイルシステムである．

ext3 のジャーナリング機能は，Journaling Block Device（JBD）と呼ばれる．JBD は，ファイルシステム作成時に，ジャーナル領域を確保する．本稿では，ジャーナリング領域として割り当てられているブロックをジャーナルブロックと呼ぶ．

JBD の用いるジャーナリング手法は，物理ジャーナリングと呼ばれ，メタデータの操作内容をジャーナルに記録するのではなく，更新されたメタデータを保存しているブロック全体を記録する．JBD は，システムコールによって，ext3 のメタデータがどのように変更されるかについて何も知識を持っていない．

ext3 は，ある一定時間内になされた，メタデータブロックへの変更を，1 つのトランザクションにまとめる．1 つのトランザクション中に，あるメタデータブロックが複数回変更された場合は，過去の変更は上書きされ，最後の状態がジャーナルに記録される．本稿では，この動作をトランザクションマージと呼ぶ．

図 2 は，JBD のジャーナルの一例である．1 つの正方形が，1 つのジャーナルブロックを示している．ジャーナルブロックには，以下の 4 種類がある．

metadata システムコールによって変更されたメタデータブロック．システム障害後の復旧時に，ファイルシステムに書き戻される．

revoke 削除されたメタデータに関する情報を記録する．

descriptor メタデータブロックの前に現れるブロックで，復旧時にメタデータブロックを書き戻すディスク位置を記録する．

commit 1 つのトランザクションの終了を示す．

図 2 の例は，1 つの完了したトランザクションと，1 つの未完のトランザクションを含んでいる．第 5 トランザクションは，1 つの revoke ブロック，1 つの descriptor ブロック，4 つの metadata ブロック，1 つの commit ブロックを含んでいる．障害後の復旧時には，revoke ブロックおよび，descriptor ブロックの内容に従って，metadata ブロックをファイルシステムに書き戻す．システム障害のため，第 6 トランザクションは，commit ブロックを含んでおらず，完了していない．したがって，このトランザクションに含まれる変更されたメタデータブロックは，ファイルシステムに書き戻されず，破棄される．

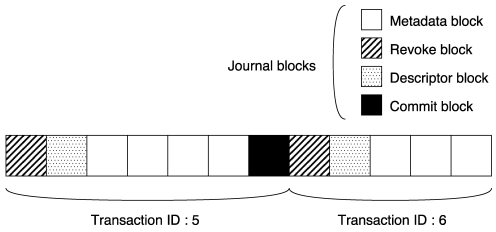


図 2 JDB ジャーナルの一例
Fig. 2 JDB journal example.

ext3 は，3 種類の異なるジャーナリング動作をサポートしている．writeback モードでは，メタデータの一貫性だけが保持され，システム障害後のファイルデータに関して何も保証されない．

journal モードでは，ファイルデータの変更もメタデータ同様にジャーナルに記録され，メタデータとファイルデータの一貫性が保たれる．

デフォルトで選択される，ordered モードは，writeback モード同様に，メタデータ一貫性だけを保証する．ただし，トランザクションが完了する前に，そのトランザクション中に変更されたファイルデータがディスクに書き込まれることを保証するため，新規ファイルの作成等，いくつかの操作に関しては，ファイルデータの一貫性も保たれる．

3.2.3 reiserfs

reiserfs は，ext3 と同様，Linux カーネルがサポートする主要なファイルシステムの 1 つである．

reiserfs のジャーナリングの仕組みは ext3 と類似しており，物理ジャーナリング，トランザクションマージを利用する．また，ジャーナルのデータ構造に関しても，revoke ブロックを使わないことを除けば，ext3 とほぼ同様である．

現在の reiserfs の実装は，ext3 と同様の 3 種類の動作モードをサポートしており，デフォルトでは，ordered モードで動作する．

3.3 アルゴリズム

説明を簡易にするために，我々は TARM ストレージシステム（主ストレージシステム）と副ストレージシステムのデータが同期している状態から説明を始める．TARM ストレージシステムは，2 種類の論理ボリュームを利用する．

主論理ボリューム 計算機に提供する論理ボリューム．計算機で動作するファイルシステムが読み書きする．

ログ論理ボリューム 計算機が更新したデータを，副ストレージシステムに転送するまで一時的に蓄え

る TARM ログとして用いる論理ボリューム．

3.3.1 初 期 化

最初に，TARM ストレージシステムは，主論理ボリュームのうち，ファイルシステムがジャーナルブロックとして利用しているブロックの認識を試みる．この動作は，計算機とストレージシステムがイニシエータとターゲットの関係（I_T nexus）を確立した場合と，ファイルシステムのスーパブロックが更新された場合に実行される．

I_T nexus の確立時に，主論理ボリュームに TARM がサポートするファイルシステムが，すでに作成されていた場合は，スーパブロックとして使用される特定の位置のセクタを検査することで，ファイルシステムの検出が可能である．TARM は，ファイルシステムの検出後，スーパブロックの内容から，ファイルシステムのブロックサイズを判断する．

I_T nexus の確立後に，計算機が主論理ボリュームにファイルシステムを作成する場合に対応するため，TARM はスーパブロックとして使われる可能性のあるブロックへの変更を監視する．スーパブロックの一部のデータは，ファイルシステム作成時のみに変更されるため，TARM は，スーパブロックの通常の更新と新規のファイルシステム作成を区別することが可能である．

ext3 は，通常のファイルと同様に，i ノードを用いて，ジャーナルブロックを管理する．i ノード番号，i ノード情報を保存しているブロックの番号は，スーパブロックに保存されている．したがって，TARM は，ファイルシステムを検出後，ジャーナルブロックとして用いられているブロックを認識できる．

reiserfs は，スーパーブロックの位置の直後に，ジャーナルブロックを連続したブロックとして割り当てるため，TARM は，容易にジャーナルブロックを識別できる．

3.3.2 ディスクアクセス

TARM ストレージシステムが，計算機から書き込み命令と更新するデータを受け取ると，以下の操作を実行する．

(1) TARM ログヘッダと更新データから構成される，TARM ログエントリを作成する．TARM ログヘッダは更新データを書き込むブロック番号（位置）とデータ長から構成される．
(2) TARM ログの最後尾に作成したエントリを書き込む．
(3) 更新データを主論理ボリュームの指定された位置に書き込む．
(4) 両方の書き込みの完了後に計算機に書き込み完了通知を送る．

読み込みの要求の場合は，主論理ボリュームからデータを読み込み，計算機に送信する．TARM ログ領域へのアクセスは必要ない．

3.3.3 リモートミラーリング

リモートミラーリングの手順は以下のとおりである．

(1) TARM ログの最後尾から最も古いエントリを読み込む．
(2) エントリに含まれている更新データの更新順序制限の有無を判断する．
(3) もし，更新順序に制限がある場合は，現在，副ストレージシステムに転送済みで，まだ更新完了通知を受け取っていないすべてのデータを待つ．
(4) 更新データを副ストレージシステムに転送する．
(5) 副ストレージシステムから完了通知を受け取り，エントリを TARM ログから削除する．

3.2.2 項で説明したように，ext3 はアトミックに更新する必要があるブロック☆をジャーナルにあらかじめ記録する．そして，commit ブロックを書き終えたところで，それらの更新が含まれるトランザクションが完了したと見なされる．したがって，ext3 の更新順序制限は，commit ブロックを書き込む前に，それ以前に変更されたすべてのブロックを書き込むことである．

あるトランザクションの commit ブロックが副ストレージシステムに転送される前に，障害により主ストレージシステムのデータが利用できなくなった場合を考える．ext3 の観点からは，副ストレージシステムのデータは，完了していないトランザクションがジャーナルの最後に記録されているファイルシステムである．このデータを利用する際，ext3 による復旧作業が実行され，このトランザクションの変更は破棄される．破棄されるブロックを副ストレージシステムがどのような順番で書いたかは問題にならない．また，ext3 がジャーナルに記録しないブロックはどのような順序で書き込まれても問題にならない．

上記の更新順序制限は，副ストレージシステムのデータの停電一貫性を実現し，そのデータがファイルシステムとして利用可能であること，つまり，ext3 自身がメタデータを一貫性のある状態に回復できることを保証する（journal モードの場合，ファイルデータも一貫性のある状態に回復できることが保証される）．

☆ journal モードではファイルデータとメタデータブロック，ordered と writeback モードでは，メタデータブロック．

この更新順序制限は，3 種類の ext3 の動作モード，すべてに適用することができる．

JBD は，制御用のジャーナルブロック（revoke，descriptor，commit ブロック）の最初の 4 バイトに特別な数値（マジックナンバ）を保存する．さらに，JBD は，次の 4 バイトに，制御用ブロックの種類を示す値を保存する．TARM ストレージシステムは，初期化時に，ジャーナルブロックとして利用されているブロックを識別しているため，上記の特性を利用することで，commit ブロックとして使われているブロックを簡単に識別できる．

reiserfs の更新順序制限も，ext3 と同様に，commit ブロックを書き込む前に，それ以前に変更されたすべてのブロックを書き込むことである．reiserfs は，commit ブロックにマジックナンバを保存しないため，commit ブロックを直接識別することはできない．しかし，descriptor ブロックはマジックナンバ，さらに，対になる commit ブロックの位置を含んでいるため，descriptor ブロックを識別することで，その内容を利用して，commit ブロックを識別することができる．

ストレージシステム間で用いられているブロックレベルのプロトコルが，命令実行順序の制御機能をサポートしている場合，上記の手順の 3 番目の操作を高速化することができる．iSCSI プロトコルの場合，*task attribute* 機能（ATTR）をこの目的のために利用することができる．iSCSI ストレージシステムは，ordered ATTR が有効になっている書き込み命令を開始する前に，現在実行中のすべての書き込み命令の完了を待つ．加えて，ordered ATTR の書き込み命令が完了するまで，その後の書き込み命令は開始されない．

現在の TARM ストレージシステムの実装では，TARM ログエントリが含んでいる更新データが，commit ブロックとして使われていることを検出した場合，そのデータを副ストレージシステムに転送する際に，ordered ATTR を指定した書き込み命令を利用する．

ミラーリングを開始するタイミングは，最後にデータを同期してからの経過時間や，TARM ログに記録されているデータ量等の値をパラメータとして制御することが考えられる．現在の実装では，データが TARM ログに記録されるとすぐに，ミラーリングを開始する．

3.3.4 障　害

主ストレージシステム内部の障害，主サイトの災害，ストレージシステム間のネットワーク障害により，副ストレージシステムが，主ストレージシステムのデータへアクセスできなくなった場合，副ストレージシステムに計算機を接続し，ファイルシステムとしてそのデータを利用することになる．

上記の障害の判断は，ミラーリングシステムの管理者が行う必要がある．たとえば，ネットワークに障害が発生してから，副ストレージシステムのデータを利用すると判断するまでの時間は，リンク層のネットワークの性質（切断が発生する確率，回復まで必要な平均時間）等によって影響を受ける．

障害後の TARM による副ストレージシステムのデータの復旧操作は必要なく，停電一貫性が保持されているデータに対して，ファイルシステム自身の復旧作業が実行される．

非同期リモートミラーリングの性質のため，副ストレージシステムの復旧後のデータは，主ストレージシステムのデータよりも古いものになる．たとえば，障害前に主ストレージシステムで削除されていたファイルが，副ストレージシステムでは削除されていない状態や，主ストレージシステムで更新されたファイルが副ストレージシステムでは更新されていない状態，等が発生する．

ファイルデータの一貫性を保証しない writeback，ordered モードでファイルシステムを運用していた場合，ファイルデータはジャーナルに記録されず（更新順序制限が発生しない），副ストレージシステムは，主ストレージシステムと異なる順番で，ファイルデータを更新する可能性が高い．その結果，副ストレージシステムのあるファイルの内容が，障害前の主ストレージシステムの対応するファイルと異なる状態が発生する．

4. 性能測定

提案手法を評価するために，Linux カーネル内部で動作し，iSCSI ストレージ機能を実現するオープンソースソフトウェアの iSCSI enterprise target software（IET）[1] に変更を加え，TARM 機能を持つストレージシステムを実装した．

表 1 に TARM のソース行の概数を示す．ジャーナリングファイルシステムは，ファイルシステムの write 性能の低下を避け，障害後の復旧時間を短くするため，単純なジャーナルのデータ構造を使う．そのため，ファイルシステムに依存する，ジャーナルのデータ構造解析部分のコードは非常に小さく，TARM を他のジャー

表 1 TARM の実装規模
Table 1 Lines of code in TARM.

TARM 共通機能部分		1,020 行
TARM ファイルシステム依存部分	ext3	240 行
	reiserfs	170 行
iSCSI enterprise target software		12,600 行

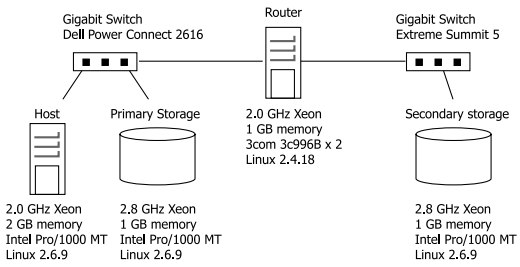


図 3 実験環境
Fig. 3 Test environment.

ナリングファイルシステムへ対応させることは容易である．

TARM 機能を実装した主ストレージシステムが，iSCSI プロトコルを利用して，副ストレージシステムにアクセスするために，Cisco 社のオープンソースの iSCSI イニシエータソフトウェア[10] を利用した．

商用環境で発生する負荷を模擬する 2 種類のベンチマークを利用して，TARM と基本ミラーリング手法の性能を比較した．Postmark[11] は，メールサーバ，ニュースサーバ，HTTP ベースのオンラインサイト用サーバでの負荷を模擬するベンチマークであり，大量の小さなサイズの短命なファイルを操作する負荷を生成する．我々は，サイズが 512 B から 16 KB のファイル，30,000 個を使い，50,000 回のトランザクションを実行した．dbench[12] は，ファイルサーバの負荷を模擬する．クライアント数を 32，実行時間を 300 秒に設定した．

基本ミラーリング手法の動作を実現するために，実装した TARM ストレージシステムに，つねに ordered ATTR を有効にした書き込み命令を利用して，副ストレージシステムのすべてのデータ更新順序を指定する動作モードを追加した．

4.1 実験環境

図 3 に，実験で用いた機器の構成を示す．

副ストレージシステムは，Linux が動作する AT 互換機と IET ターゲットソフトウェアを利用した．両ストレージシステムは，Adaptec 社の Ultra320 SCSI ホストバスアダプタ（39320A-R）につながれた，Maxtor 社の Atlas 10K SCSI ドライブ（146 GB, 10,000 RPM）を搭載する．

ホストコンピュータは，Cisco 社の iSCSI イニシエータソフトウェアを利用して，TARM ストレージシステムに接続している．

ストレージシステム間の遅延を模擬するための遅延装置として，Nist Net ソフトウェア[13] を使ったルータを配置した．遅延は，片方向につき，0，2，4 ms に設定した．4 ms の遅延は，800 km のファイバ長に相当する．

表 2 iSCSI イニシエータパラメータ設定
Table 2 iSCSI initiator parameters.

項目	値
InitialR2T	On
ImmediateData	On
MaxRecvDataSegmentLength	128 KB
MaxBurstLength	256 KB
FirstBurstLength	64 KB
MaxConnections	1
MaxOutstandingR2T	1

今回の実験では，MAN 環境を想定したネットワーク設定としている．

ホストコンピュータは，TARM ストレージシステムが提供する主論理ボリュームに，基本パーティションを 1 つ作成し，ファイルシステムを構築した．ext3, reiserfs，ともに，ブロックサイズが 4 KB，ordered モードで動作させた．

ホストコンピュータと主ストレージシステムが用いた iSCSI イニシエータの設定値を，**表 2** に示す．様々な iSCSI イニシエータ・ターゲット実装に適用できるよう，MaxRecvDataSegmentLength を除き，iSCSI プロトコルのデフォルトの値を採用した．MaxRecvDataSegmentLength の値を iSCSI プロトコルのデフォルト値（8 KB）よりも大きく設定した理由は，性能を向上させるためである．128 KB という MaxRecvDataSegmentLength の値は，今回利用したオープンソースの実装を含め，複数の iSCSI 実装でサポートされている．

4.2 スループット

TARM ストレージシステムのミラーリング動作を停止し，ホストコンピュータでベンチマークを動作させた．ベンチマーク終了後，ミラーリングを再開し，完了するまでに要した時間を測定した．我々は，TARM ログの状態を保存，復元できる仕組みを実装し，同じ TARM ログを使って，それぞれのミラーリング手法について，5 回の測定を行った．以下のミラーリング速度の測定結果は，5 回の平均値である．

4.2.1 postmark

ホストコンピュータが発行した書き込み命令の回数は，ext3 の場合が 56,558 回，reiserfs の場合が 112,140 回であった．また，ベンチマーク終了後，TARM ログに保存されたデータ量は，ext3 が 3.24 GB，reiserfs が 5.39 GB だった．

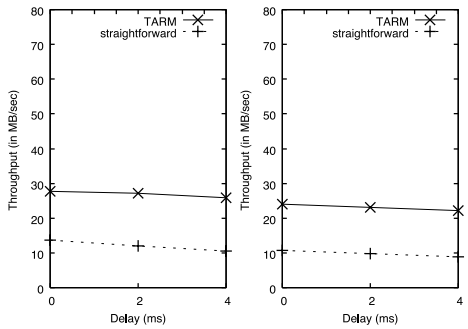
表 3 は，ホストコンピュータが発行した書き込み命令の I/O サイズごとの回数を示している．表中の「4〜

表 3 Write サイズの分布（postmark）
Table 3 Write size distribution (postmark).

	I/O size (KB)	4〜8	8〜16	16〜32	32〜64	64〜128	128〜
ext3	Order constraint (times)	40	0	0	0	0	0
	(%)	0.071	0	0	0	0	0
	No order constraint (times)	34,043	13,371	5,041	181	165	3,726
	(%)	60.2	23.6	8.91	0.321	0.291	6.59
reiserfs	Order constraint (times)	92	0	0	0	0	0
	(%)	0.082	0	0	0	0	0
	No order constraint (times)	73,788	22,513	7,844	2,120	1,626	4,157
	(%)	65.8	20.1	7.00	1.89	1.45	3.71

表 4 Write サイズの分布（dbench）
Table 4 Write size distribution (dbench).

	I/O size (KB)	4〜8	8〜16	16〜32	32〜64	64〜128	128〜
ext3	Order constraint (times)	54	0	0	0	0	0
	(%)	0.085	0	0	0	0	0
	No order constraint (times)	19,839	7,275	4,470	8,790	8,028	15,073
	(%)	31.2	11.5	7.04	13.4	12.6	23.7
reiserfs	Order constraint (times)	43	0	0	0	0	0
	(%)	0.091	0	0	0	0	0
	No order constraint (times)	11,658	3,734	3,406	1,560	7,215	19,862
	(%)	24.6	7.86	7.17	3.29	15.2	41.8



(a) ext3 (b) reiserfs

図 4 ミラーリング性能（postmark）
Fig. 4 Mirroring throughput (postmark).

8」の表記は，「4 KB 以上 8 KB 未満」を意味する．なお，ファイルシステムのブロックサイズが 4 KB であるため，4 KB 未満の I/O サイズは存在しない．ホストコンピュータで動作する Linux カーネルの I/O サブシステムの機能が，隣接するセクタへの書き込みをマージするため，ファイルシステムのブロックサイズよりも大きなサイズの書き込み命令も存在する．トランザクションマージを利用する両ジャーナリングファイルシステムでは，更新順序制限があるデータは非常に少ないことが分かる．

図 4 は，TARM が高いミラーリング性能を実現していることを示している．ext3 の場合，TARM は，基本ミラーリング手法よりも，各遅延条件でそれぞれ，2.03，2.24，2.46 倍高速であった．reiserfs の場合は，2.25，2.36，2.49 倍高速であった．

reiserfs よりも，ext3 を利用した場合のミラーリングが高速なのは，ext3 の方が大きな I/O サイズの書き込み命令回数の割合が高いためだと思われる．

予測されたように，遅延はミラーリング性能を低下させている．TARM よりも，基本ミラーリング手法が遅延の影響を大きく受けているのは，すべてのデータ更新をシリアライズしているためである．

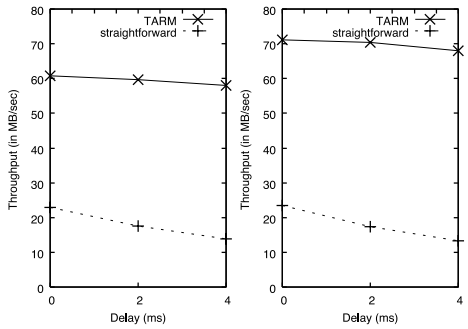
4.2.2 dbench

ホストコンピュータが発行した書き込み命令の回数は，ext3 の場合が 63,529 回，reiserfs の場合が 47,478 回であった．ベンチマーク終了後，TARM ログに保存されたデータ量は，ext3 が 12.37 GB，reiserfs が 12.86 GB だった．

表 4 は，ホストコンピュータが発行した書き込み命令のデータ長ごとの回数を示している．postmark の結果と比較して，I/O サイズの大きな書き込み命令の割合が大きい．

測定結果を**図 5** に示す．TARM が基本ミラーリング手法よりも，各遅延条件でそれぞれ，ext3 の場合が，2.65，3.41，4.18 倍，reiserfs では，3.04，4.07，5.13 倍高速であった．

postmark の結果とは逆に，ext3 よりも reiserfs の大きな I/O サイズの書き込み命令回数の割合が高いため，reiserfs の方がミラーリングが高速であった．



(a) ext3 (b) reiserfs

図 5 ミラーリング性能（dbench）
Fig. 5 Mirroring throughput (dbench).

表 5 TARM ログに保存されてた最大データ量（postmark）
Table 5 Maximum amount of the TARM log (postmark).

遅延（ms）	最大データ量（MB）			
	TARM		基本ミラーリング	
	ext3	reiserfs	ext3	reiserfs
0	2.13	2.63	109.11	81.94
2	2.75	2.49	158.71	126.14
4	3.02	3.02	196.84	236.16

表 6 TARM ログに保存されていた最大データ量（dbench）
Table 6 Maximum amount of the TARM log (dbench).

遅延（ms）	最大データ量（MB）			
	TARM		基本ミラーリング	
	ext3	reiserfs	ext3	reiserfs
0	2.30	2.37	393.66	623.25
2	2.47	3.18	976.91	1301.08
4	2.65	4.06	1352.34	1810.09

postmark と比較して，dbench は大きな I/O サイズの書き込み命令回数の割合が高いため，(1) TARM による高速化の効果が大きい，(2) ネットワーク遅延による性能への影響が大きい[14]．

4.3 ストレージ間のデータ解離量

非同期ミラーリングでは，主ストレージシステムに災害が発生した場合，副ストレージシステムに転送されていないデータは失われるため，その損失量を小さくすることが重要である．ミラーリングの高速化がデータ損失量に与える影響を調べるため，ホストコンピュータでベンチマークを実行し，0.5 秒ごとに，TARM ログに保存されているデータ量を記録した．

表 5 に postmark，**表 6** に dbench の測定結果を示す．TARM のデータ損失量は，基本ミラーリング方式の 1/31〜1/510 となっており，ミラーリングの高速化によって，データ損失量が大幅に減少していることが分かる．

5. 関 連 研 究

5.1 非同期リモートミラーリング

複数の SAN 用ストレージシステムが，非同期リモートミラーリングをサポートしているが，計算機や遠隔の副ストレージシステムに実装されたリモートミラーリング専用の機能を使わずに，停電一貫性を保証する，非同期リモートミラーリング高速化技術は提案されていない．

NetApp 社の SnapMirror[15] は，Network Attached Storage（NAS）ストレージアーキテクチャで用いられる，ファイルシステムで実装された，非同期リモートミラーリング技術である．主サーバは定期的にスナップショットを作成し，副サーバに転送する．NetApp 社のストレージシステムは，NVRAM を使ったアトミックなデータ更新機能により，停電一貫性を保証する．両サーバで動作する，Write Anywhere File Layout（WAFL）ファイルシステム[9] は，no-overwrite ファイルシステムであり，ファイルデータとメタデータの両方をアトミックに更新できる，障害後の復旧作業を必要しない，という特徴を持つ．

文献 16) で，Ji らは，ファイルシステムやデータベースへの適用を想定した，停電一貫性を保証する非同期リモートミラーリング方式，Seneca を提案している．Seneca は，副ストレージシステムは，NVRAM 等を使ったアトミックなデータ更新機能により，停電一貫性を保証する．

StarFish[17] は，TARM と同様に，計算機の特別な機能を使わずに，透過的にリモートミラーリングを実行する．しかし，非同期にリモートミラーリングを実行した場合は，リモートミラーリングのデータ一貫性モデルは無保証となり，副ストレージシステムに保存されたデータは破壊された状態になる危険性がある．

5.2 ストレージシステムのインテリジェント化

計算機とストレージシステム間のインタフェースは変更せずに，ストレージシステムが，保存されているデータから情報を得て，利用する手法がいくつか提案されている．

C-Miner ストレージシステム[18] は，一緒に使われる可能性の高いブロックを予想し，プリフェッチの精度を向上させている．

文献 4) で，Sivathanu らは，TARM 同様に，ファイルシステム構造の知識を利用し，ブロックの使用状況を判断する手法を提案し，ファイルシステムが削除したデータを，自動的に復旧不可能にするストレージシステムを実装している．

TARM は，更新順序制限に関する情報をジャーナルに記録するという，ジャーナリングファイルシステムの特徴を利用しており，様々なオペレーティングシステムのジャーナリングファイルシステムに対応することができる．広く使われているオペレーティングの多くが，ジャーナリングファイルシステムを標準のファイルシステムとして用いている．本稿で扱った ext3，reiserfs 以外のジャーナリングファイルシステムとしては，SGI 社 XFS（Irix，Linux），IBM 社の JFS（AIX，OS/2，Linux），Sun 社の UFS（Solaris），Microsoft 社の NTFS（Windows），Apple 社の HFS+（MacOS）等がある．

6. まとめ

非同期のリモートミラーリングでは，副ストレージシステムのデータがファイルシステムとして利用できる状態であることを保証する，データの停電一貫性を実現するため，すべてのストレージシステムのデータ更新順序が同一でなければいけない．副ストレージシステムのすべてのデータ更新順序を指定する基本ミラーリング手法は高い性能を実現できないため，実環境で使用される既存のミラーリング手法は，計算機やストレージシステムに実装されたミラーリングのための特別なハードウェアや機能を利用する．

本稿では，非同期リモートミラーリングの高速化手法，TARM を提案した．TARM は主ストレージシステム内に実装され，ファイルシステムの構造に関する知識を利用し，ファイルシステムが保存しているブロックの内容から，データの停電一貫性を実現するために必要な更新順序を検出する．副ストレージシステムで，更新順序に制限のないデータを並列に更新することで，ミラーリングを高速化する．

TARM は，ジャーナリングファイルシステムが大半のデータを自由な順序で更新できる点，加えて，データの更新順序制限はジャーナルの内容から容易に判断可能である点を利用している．

TARM の主要な長所は，あらゆるストレージシステムを副ストレージシステムとして利用可能な点，計算機とストレージシステム間のインタフェースに変更を必要としない点である．

Linux の ext3 と reiserfs ファイルシステムの更新順序制限を検出できるストレージシステムを実装し，商用環境の負荷を模擬するベンチマークを用いて，その性能を評価したところ，基本ミラーリング手法と比較し，最大 5.13 倍のミラーリング性能が得られた．

参考文献

1) 藤田智成，小河原成哲：iSCSI ターゲットソフトウェアの解析，情報処理学会論文誌：コンピューティングシステム，Vol.46, No.SIG 3 (ACS 8), pp.38–50 (2005).
2) Satran, J., Meth, K., Sapuntzakis, C., Chadalapaka, M. and Zeidner, E.: Internet Small Computer Systems Interface (iSCSI), RFC 3720 (2004).
3) Gibson, G.A., Zelenka, J., Nagle, D.F., Amiri, K., Butler, J., Chang, F.W., Gobioff, H., Hardin, C. and Rochberg, E.R.D.: A cost-effective, high-bandwidth storage architecture, *ACM SIGOPS Operating Systems Review*, Vol.32, No.5, pp.92–103 (1998).
4) Sivathanu, M., Bairavasundaram, L., Arpaci-Dusseau, A.C. and Arpaci-Dusseau, R.H.: Life or Death at Block-Level, *6th Symposium on Operating Systems Design and Implementation*, San Francisco, CA, pp.379–394 (2004).
5) Azagury, A.C., Factor, M.E. and Micka, W.F.: Advanced functions for storage subsystems: Supporting continuous availability, *IBM Systems Journal*, Vol.42, No.2, pp.268–279 (2003).
6) Chutani, S., Anderson, O.T., Kazar, M.L., Leverett, B.W., Mason, W.A. and Sidebotham, R.N.: The Episode File System, *The USENIX Winter Conference*, San Francisco, CA, pp.43–60 (1992).
7) Ganger, G.R., Mckusick, M.K., Soules, G.A.N. and Patt, Y.N.: Soft Updates: A Solution to the Metadata Update Problem in File Systems, *ACM Trans. Comput. Syst.*, Vol.18, No.2, pp.127–153 (2000).
8) Rosenblum, M. and Ousterhout, J.K.: The Design and Implementation of a Log-Structured File System, *The Symposium on Operating Systems Principles*, Monterey, CA (1991).
9) Hitz, D., Lau, J. and Malcolm, M.: File System Design for an NFS File Server Appliance, *The USENIX Winter 1994 Technical Conference*, San Francisco, CA, pp.235–245 (1994).
10) Cisco: Linux iSCSI initiator software. http://linux-iscsi.sf.net/
11) Katcher, J.: PostMark: A New File System Benchmark, Technical Report TR3022, Network Appliance (1997).
12) Tridgell, A.: dbench benchmark (2001). http://samba.org/ftp/tridge/dbench/
13) The National Institute of Standards and Technology: NIST Net network emulator.

http://snad.ncsl.nist.gov/nistnet/
14) Ng, W.T., Hillyer, B., Shriver, E., Gabber, E. and Özden, B.: Obtaining High Performance for Storage Outsourcing, *The USENIX Conference on File and Storage Technologies*, Monterey, CA, pp.145–158 (2002).
15) Patterson, H., Manley, S., Federwisch, M., Hitz, D., Kleiman, S. and Owara, S.: SnapMirror: File System Based Asynchronous Mirroring for Disaster Recovery, *The Conference on File and Storage Technologies* (*FAST 2002*), Monterey, CA, USENIX, pp.117–129 (2002).
16) Ji, M., Veitch, A. and Wikes, J.: Seneca: Remote mirroring done write, *The USENIX Annual Technical Conference*, San Antonio, TX, pp.253–267 (2003).
17) Gabber, E., Fellin, J., Flaster, M., Gu, F., Hillyer, B., Ng, W.T., Özden, B. and Shriver, E.: StarFish: Highly-Available Block Storage, *The USENIX Annual Technical Conference*, San Antonio, TX, pp.151–164 (2003).
18) Li, Z., Chen, Z., Srinivasan, S.M. and Zhou, Y.: C-Miner: Mining Block Correlations in Storage, *The USENIX Conference on File and Storage Technologies*, San Francisco, CA, pp.173–186 (2004).

(平成 17 年 4 月 28 日受付)
(平成 17 年 8 月 5 日採録)

藤田 智成（正会員）
2000 年早稲田大学大学院理工学研究科修士課程修了．同年日本電信電話株式会社入社．オペレーティングシステムに関する研究に従事．ACM，USENIX 各会員．

矢田 浩二（正会員）
1985 年大阪大学基礎工学部情報工学科卒業．1987 年同大学院修士課程修了．同年 NTT 研究所入所．以後，キャリアネットワークの管理システム構築技術，および次世代 IP 通信サービスの研究に従事．現在，NTT サービスインテグレーション基盤研究所勤務．