2007－OS－104 (13)
2007／1／31

汎用ミラーリングファイルシステムの設計

西村 悟[†] 佐野睦夫[††] 池田克夫[††]

Linux 用の汎用ミラーリングファイルシステム GMFS(General-purpose Mirroring FileSystem) を提案する．GMFS は他のファイルシステムのラッパとして動作することにより，既存環境を変更することなく柔軟にミラーリングを行う．GMFS は他のファイルシステムを透過的に扱うので，利用者は GMFS の存在を意識する必要はなく，既存ファイルシステムがミラーリングを行っているように見える．本稿では GMFS の設計と実装方法，動作速度の実験結果について述べる．

Design of a General-purpose Mirroring FileSystem

NISHIMURA SATOSHI,[†] SANO MUTSUO[††] and IKEDA KATSUO[††]

We propose a General-purpose Mirroring FileSystem "GMFS" for Linux operating system. The GMFS performs mirroring without changing existing environments by functioning as a wrapper of other filesystems. Users need not be aware of the GMFS, and existing filesystem does seem to perform mirroring, because the GMFS operates the other filesystems transparently. This paper presents an algorithm, implementation, and performance assessment of the GMFS.

1. はじめに

データをローカルディスク上でミラーリングするためには RAID[1] が頻繁に利用されている．また，ネットワーク上でのミラーリングはクラスタリングシステム[2] の機能として実装されていることが多い．

これら既存の手法はデバイスレイヤでミラーリングを行っている．そのため，何らかのシステムを構築するにあたって最初にミラーリング設計が必要である．システム稼働中に，機能追加によるミラーリングが必要になった場合，物理ディスクの割り当てに戻って検討しなければならないので対応することが困難である．

そこで柔軟にミラーリングを行うことを目的としたファイルシステム GMFS(General-purpose Mirroring FileSystem) を提案する．

GMFS はデバイスレイヤではなくファイルシステムレイヤでミラーリングを行う．そうすることによって，デバイスドライバでは扱えないファイルシステムのパス情報やファイル属性など細かな設定が可能となる．

ネットワークを通じたミラーリングはバックアップや負荷分散などで利用される機会が多く，様々な手法が提案されている．GMFS がネットワークの環境で適用される場合，独自の通信機能を実装する必要はなく既存の通信機能を持つ機構が利用できる．

本稿では，GMFS がネットワークファイルシステムを利用してミラーリングを行った場合の処理速度を測定し，デバイスドライバでのミラーリング速度と比較することによってこの手法の有効性を検証する．

2. ソフトウェアでのミラーリング

ソフトウェアでミラーリングを行う場合，次の三つのレイヤが考えられる．

- ブロックデバイス
- ファイルシステム
- アプリケーション

アプリケーションレイヤは他のレイヤと比べて自由度が高い．特定の用途に特化したミラーリングを行うためには適したレイヤだが，利用者はミラーリングアプリケーションを利用することを考えてシステムを設計しなければならないため汎用性が低い．

ブロックデバイスレイヤは既に述べたように既存システムにミラーリング機能だけを追加することが難しい．特定のデータだけをミラーリングしたい場合でもディスク構成を設計し直す必要がある．

ファイルシステムレイヤは既存環境に適応させやすい構造になっている．

† 大阪工業大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka Institute of Technology
†† 大阪工業大学情報科学部
Faculty of Information Science and Technology, Osaka Institute of Technology

Linux☆のファイルシステムは，VFS(Virtual File System) によりインターフェースが統一されている．同じディレクトリに別のファイルシステムを何度もマウントすることができるなど，柔軟な設計になっている．VFS のインターフェースを通じて特定のシステムコールに対するファイル操作の挙動を変更するといった細かな機能も提供できる．これによって既存のファイルシステムを利用しながらミラーリング機能だけを追加することができる．

3. 目　　標

Linux で汎用的なミラーリングを行うための目標を以下のように定めた．

- 既存のパーティション構成を変更することなく任意のタイミングでミラーリング機能を追加できる
- 既存ファイルシステムの機能をそのまま使用することができる
- ミラーリングでの複製データ数を制限しない
- スナップショットが利用できる

パーティション構成を変更することなくファイルシステムを構築するには，ミラーリング用のパーティションを確保せずにメモリ上のみで動作しなければならない．しかし，システムの再起動やアンマウント処理によってファイルシステムのメモリは破棄されるので永続的に情報を保存することはできない．そこで GMFS はミラーリングデータを統合的に管理せず，データにアクセスした時点でそのデータのみミラーリング管理を行う方式をとる．コピーしたデータは既存のファイルシステムに保存する．

既存ファイルシステムの機能を利用するには，GMFS を他のファイルシステムのラッパとして動作させ，既存ファイルシステムに対する操作を全て扱えるようにする．

データの複製を複数作成する場合は，複製されたデータに対する統合的な管理を行わない．GMFS は常にデータを二カ所に保存する．三カ所に保存する場合はミラーリングを二回行うという形で数を増やしていく方式をとる．

スナップショットはデータの整合性を維持するために重要な機能である．ファイルシステムのスナップショットについては LVM☆☆を使う方法や，ext3 にスナップショット機能を実装する研究[3] がある．いずれも物理ディスクにスナップショットデータ用の領域確保が必要なので，これらの方式をそのまま GMFS で利用することはできない．GMFS は専用のディスク領域を利用しないので，メモリ上でスナップショットデータを保持しなければならない．しかし大量にデータ書き込みが行われた場合にメモリを浪費してしまうので，全てのデータをメモリ上で管理するのは現実的でない．スナップショット機能については現在検討中である．

☆ Linux は，Linus Torvalds の米国およびその他の国における登録商標または商標．

☆☆ Logical Volume Manager

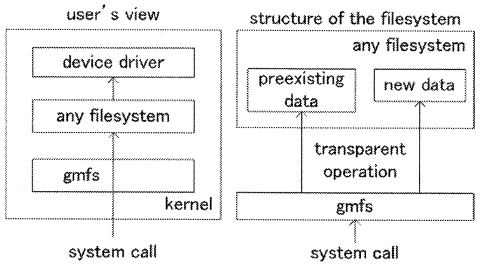

図 1　他のファイルシステムのラッパとしての動作
Fig. 1　Function as a wrapper of other filesystem.

4. 汎用ミラーリングファイルシステム GMFS

GMFS をマウント後も，ユーザの視点では GMFS マウント前のファイルシステムのように見え，GMFS を扱っていることを意識しなくてよい（**図 1**）．しかし内部的には GMFS をマウントした後の書き込みからミラーリングが行われるので，単純なバックアップツールのように任意の時点でミラーリングを開始できる．ファイルシステムのマウントは通常の mount コマンドで行い，オプションでミラーリング先ディレクトリを指定する．

デバイスドライバや既存ファイルシステムの機能を透過的に扱うことが可能なので，RAID や Coda[4] などと組み合わせることもできる．

保存したデータの実体はマウントを行う前のファイルシステムとミラーリング先ファイルシステムに保存されている．GMFS 上に書き込んだデータは，GMFS をアンマウント後もマウント前のファイルシステムを通じてアクセス可能なので，必要なときに必要なデータだけミラーリングを行うことができる．

4.1 データの読み込みと書き込み

データの読み込みでは，GMFS はキャッシュの設定を行い，GMFS マウント前のファイルシステムからデータを読み取って返却する．GMFS マウント前と GMFS マウント後のデータ読み取り操作について，ユーザの視点での違いはない．以下読み取り用に使用するファイルシステムをプライマリ，もう一方のファイルシステムをセカンダリと呼ぶ．

データの書き込みでは，プライマリとセカンダリにデータを書き込む．このときの書き込む順番やロックに関する制限は利用者が設定できるものとする．

4.2 キャッシュ

Linux の VFS には dentry キャッシュ機能が備わっている．GMFS はこれを利用してキャッシュを扱う．

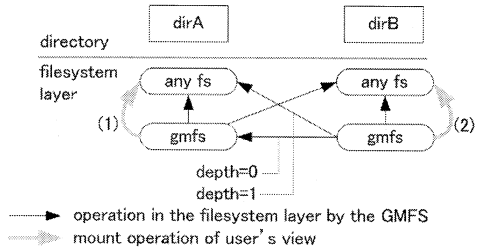

図 2 ローカル上での双方向ミラーリング
Fig. 2 Duplex mirroring on the local filesystem.

また，GMFS のデータ構造体に対するプライマリとセカンダリのデータ構造体などの固定的な情報も独自にキャッシュする．

4.3 ネットワーク越しのミラーリング

GMFS はネットワーク通信機能を持たないが，NFS[5] などを通してネットワーク越しにミラーリングすることが可能である．GMFS から NFS を利用するには，GMFS のミラーリング先ディレクトリに NFS 上のディレクトリを指定する．

書き込み順序を固定している場合，通常デッドロックは起こらない．しかしネットワークを通じてミラーリングを行う場合，ネットワークの通信速度によって書き込み順序が異なる可能性がある．その場合は，片方のファイルシステムのディスク書き込みを完了してからもう一方のファイルシステムの書き込みを開始することによってデッドロックを回避できるが，処理速度が大幅に落ちる可能性がある．

4.4 双方向のミラーリング

ここまでに述べた内容は，プライマリで指定したパスからセカンダリで指定したパスへの片方向のミラーリングである．セカンダリのパスを指定して書き込みを行った場合，プライマリに対してミラーリングは行われない．双方向のミラーリングを行うためにはプライマリとセカンダリの指定を逆にして再度マウントすることによって行う．

4.4.1 ローカル上での双方向ミラーリング

図 2 はローカル上で双方向ミラーリングを行う場合の構成を表している．(1) 初めにプライマリのディレクトリを dirA に，セカンダリのディレクトリを dirB に指定してマウントする．(2) 次にプライマリとセカンダリのディレクトリを逆にしてマウントを行う．

しかしこのように単純にマウントしただけでは，dirB への書き込みに対するコピーが二回行われてしまう．これを解決するために，マウント時にセカンダリの深さを指定できるようにする．図 2 の depth はマウントされる前のファイルシステムにアクセスする深さを表している．dirB のマウント時にセカンダリのディレクトリ dirA の深さを 1 に指定することにより，ミラーリング先ファイルシステムから再度ミラーリングが行われることを防止している．

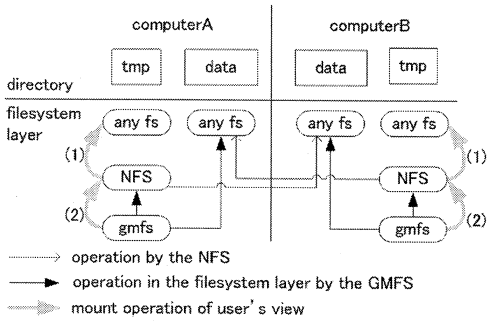

図 3 ネットワーク上での双方向ミラーリング
Fig. 3 Duplex mirroring on the network filesystem.

4.4.2 ネットワーク上での双方向ミラーリング

図 3 はネットワーク上で双方向ミラーリングを行う場合の構成を示している．セカンダリのディレクトリ tmp は NFS をマウントするために利用する空のディレクトリである．

(1) 初めに tmp ディレクトリの NFS マウントを各々のコンピュータで行う．(2) 次に tmp ディレクトリに対してプライマリのルートディレクトリを data，セカンダリのルートディレクトリを tmp と指定して GMFS のマウントを行う．これにより，両方のコンピュータで tmp ディレクトリへの書き込みがミラーリングされ，データの実体が data ディレクトリに保存される．

通常 GMFS にマウントされる側のファイルシステムをプライマリとするが，ここではマウントされる側をセカンダリに指定している．これは GMFS にマウントされる側をプライマリにするとミラーリングがループするからである．NFS はパス名からデータ書き込みを行うので，GMFS を通さず直接プライマリのデータを扱うことができない．マウントに対してプライマリとセカンダリの指定を逆にした場合，書き込みで指定したパスに対応するファイルシステムと，データの実体が存在するファイルシステムが異なることになる．そのため既存環境へ適用させることが難しい．

これを解決するためには，GMFS にマウントされるファイルシステムを GMFS のマウント後に NFS から読み取る必要がある．そこで，GMFS にミラーリングを行わない透過モードを設定する（**図 4**）．(1) まず透過モードで GMFS をマウントし，(2) そのディレクトリに対して NFS マウントを行う．(3) この後に既存環境で使用している data ディレクトリを GMFS でマウントする．そうすることによって元々利用していたディレクトリ data での双方向ミラーリングを行うことができる．このとき，data は既存ファイルシステム上の data に対するミラーリングを有効にした操作用ディレクトリ，trans は既存ファイルシステム上の data に対するミラーリングを無効にした操作用ディレクトリ，remote はもう一方のコンピュータの

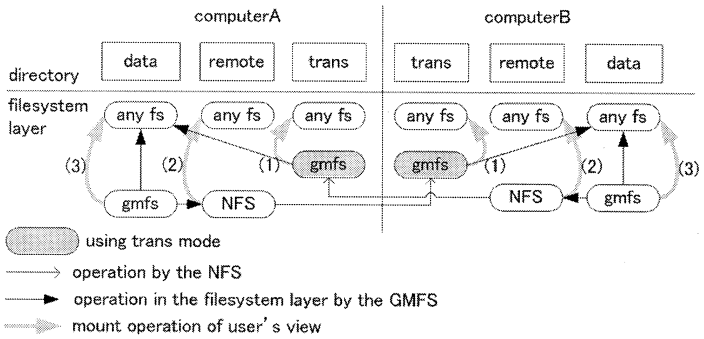

図 4　透過モードを利用したネットワーク上での双方向ミラーリング
Fig. 4　Duplex mirroring using trans mode on network filesystem.

trans 操作用ディレクトリである．データの実体は全て既存ファイルシステム any fs 上の data ディレクトリに保存される．

5. GMFS の実装

実装は Linux のカーネルモジュールとして行う．モジュールをカーネルにロードすることによって GMFS をマウントできるようになる．

ファイルシステムがマウントされると，スーパーブロックやルート inode，ルート dentry が初期化される．このときオプションで指定されたパス名に対応する dentry を取得する．

マウント前に存在したファイルはミラーリング対象にならないが，後述するパス名探索機構によってミラーリングを開始することが可能である．

5.1 パス名の探索

マウント時にプライマリ用ルートディレクトリとセカンダリ用ルートディレクトリが与えられるのでそれらの dentry を保存する．これらを基準にパス名の探索を行う．

パス名を解決するには，プライマリパス名探索機構とセカンダリパス名探索機構が必要である（**図 5**）．

パス名を探索するときにはプライマリに対して lookup を行う．プライマリの inode が存在する場合，inode データを GMFS 上の inode にコピーする．そして GMFS 上の dentry を作成しカーネルに対してキャッシュの登録を行う．これにより，次回以降の探索ではキャッシュが使われることになる．同様にセカンダリに対しても lookup 処理を行い，プライマリとセカンダリの dentry 情報を GMFS のデータ領域に格納する．

プライマリに inode データが存在してセカンダリに存在しない場合，GMFS はデータのコピーを開始する．読み込みや書き込みに先立ってセカンダリにデータをコピーすることにより，利用者がアクセスしているデータは常にミラーリングの同期が完了している状態を維持できる．

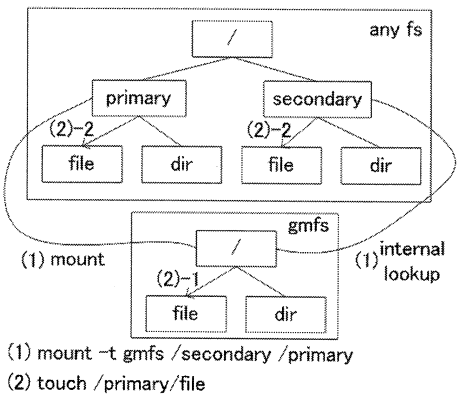

図 5　パス名の探索
Fig. 5　Searching path name.

5.2 メタデータの作成

GMFS はプライマリとセカンダリに対してメタデータ作成処理を順に依頼する．ミラーリング用の inode や dentry はパス名の探索のときに確保したものを利用する．両方のメタデータ作成に成功した場合は GMFS 上のメタデータを作成する．

セカンダリのメタデータ作成に失敗した場合，既にプライマリのメタデータ作成は完了しているのでメタデータの不一致が起こる．そのためロールバック処理としてプライマリのメタデータを削除してからエラーを返却する．

ミラーリング元のメタデータ作成に失敗した場合はそのままエラーを返却する．

5.3 メタデータの読み込み

読み込み要求に対して GMFS はセカンダリを操作しない．プライマリのみからデータ取得を行う．パス名解決の時点でセカンダリへのミラーリングが完了しているので，データを読み込む段階でセカンダリの

データを扱う必要がない．

データが存在しないなどのエラー応答もプライマリからの応答をそのままを返却する．

読み込んだメタデータはロールバック用に GMFS 上の領域に保持する．

5.4 メタデータの書き込み

データの書き込みはプライマリとセカンダリの両方に行う．通常 Linux の書き込み処理は最初にメモリ上に行われるので，書き込み完了時点でディスク上にデータが書き込まれているわけではない．ディスクへの書き込みを同期することも可能であるが，それは GMFS が扱うファイルシステムの実装に依存する．

プライマリに書き込んだ後にセカンダリでエラーが発生すると，二つのファイルシステムで不整合が生じる．整合性を維持するには書き込みをロールバックする必要があるので，GMFS 上のロールバックデータからプライマリにデータを書き戻すことによってロールバック処理を行う．

5.5 ファイルのオープン

ファイルオープンは，モードにより動作が異なる．読み込みモードで呼び出された場合，プライマリのファイルをオープンする．書き込みモードでオープンされた場合にはセカンダリのファイルも同時にオープンする．ここでオープンした file 構造体は GMFS の file 構造体に格納する．

5.6 ファイルの読み込み

file 構造体からプライマリの file 構造体を取得し，読み込み処理を依頼する．そして読み込んだ結果をそのまま返却する．

5.7 ファイルの書き込み

file 構造体からプライマリとセカンダリの file 構造体を取得し，書き込み処理を順に依頼する．メタデータの書き込みと同様にエラーの場合は不整合が生じる．また，要求したバイト数を実デバイスに書き込めるとは限らず，書き込みバイト数が異なる可能性もある．

整合性を維持するためにはロールバックが必要であるが，GMFS は書き込みデータに対応する現在のデータを把握していない．ファイルの書き込み用データはメタデータのようにデータ量が少なくないので，ロールバック用のデータを書き込み前に確保することは現実的でない．GMFS が利用するファイルシステムが書き込み前の状態を把握しているファイルシステム[6]ならロールバックが可能であるが，そのようなファイルシステムはまれであるためファイルの書き込みに対してはロールバックを行わない．

6. 実験と評価

6.1 ローカル上での実験

GMFS が正常に動作していることを検証するため，まずはローカル上でシステムコールに対する速度測定を行った．測定にはベンチマークツール mkfbmt☆を使用した．実験環境は次の通りである．

- OS: Debian GNU/Linux 3.1 (sarge)
- kernel: 2.6.8
- CPU: Intel Pentium4 2.60Hz
- Memory: 1GB

結果を**表 1** で示す．ローカルデバイス上で二回処理を行っているのでシステムコールにかかる時間も二倍になっている．

表 1 ローカル上でのミラーリング速度
Table 1 Performance of local mirroring.

system call function	local	local mirroring
open() for existing file	0.317(s)	0.663(s)
creat() for new file	3.607(s)	7.251(s)
write() for existing file	0.401(s)	0.792(s)
write() for new file	3.696(s)	7.529(s)

Average of 10,000 execution times.

表 2 NFS 上でのミラーリング速度
Table 2 performance of mirroring using NFS.

system call function	local	GMFS and NFS
open() for existing file	0.032(s)	0.477(s)
creat() for existing file	0.032(s)	0.897(s)
creat() for new file	0.361(s)	4.617(s)
write() for existing file	0.040(s)	5.237(s)
write() for new file	0.340(s)	7.693(s)

Average of 1,000 execution times.
Network: 1000Base-T

6.2 NFS を利用した場合の実験

GMFS はネットワーク通信機能を持っていないので，ネットワーク越しにミラーリングするためには NFS などの通信機能を持った機構を利用する必要がある．そこで NFS でマウントしたディレクトリに対して GMFS でマウントし，ローカルとネットワーク上コンピュータの二台でミラーリングを行う場合について実験を行った．

NFS を利用したミラーリングによって，システムコールの速度がどの程度影響されるかを**表 2** で示す．

かなりの遅延がみられるが，これは GMFS がローカル上への操作を完了し，更に NFS への操作を完了した時点でシステムコールが完了しているからである．この結果から，小さいサイズのファイルを頻繁に書き込むようなシステムで GMFS を利用すると，性能が 1/10 になる可能性があることを示している．これを改善するためには，ミラーリングを非同期で行う必要がある．

次にミラーリング速度が実用的なものかどうかを検証するために，既存のクラスタリングシステム CLUSTERPRO☆☆でのミラーリング速度と比較した．クラ

☆ http://wiki.livedoor.jp/linuxfs/d/
☆☆ CLUSTERPRO は，日本電気株式会社の登録商標．

表 3 GMFS とクラスタリングシステムのミラーリング速度
Table 3 Mirroring performance of GMFS and clastering system.

GMFS and NFS		
	putc	write
local	37.8(MB/s)	49.2(MB/s)
mirror	31.6(MB/s)	31.7(MB/s)
mirror/local	0.84	0.64
clustering system		
	putc	write
local	20.1(MB/s)	34.9(MB/s)
mirror	13.6(MB/s)	14.3(MB/s)
mirror/local	0.68	0.41

スタリングシステムの実験環境は次の通りである☆.

- OS: TurboLinux EnterpriseServer8
- kernel: 2.4.19-340
- CPU: Intel Xeon 2.40Hz x 2
- Memory: 1GB
- Network: 1000Base-T

測定にはベンチマークツール bonnie☆☆を使用した.

結果を**表 3** に示す. テスト環境が異なっているのでローカルデバイスへの書き込み速度とミラーリング環境での書き込み速度の割合で比較した. GMFS ではローカル書き込みに対するミラーリング書き込みによる速度低下がキャラクタ型 write では 16%, ブロック型 write では 36%であった. 既存クラスタリングシステムではキャラクタ型 write が 32%, ブロック型 write が 59%である. 既存のクラスタリングシステムよりも GMFS の方がミラーリングによる速度低下の割合が少ない結果になった.

GMFS を処理速度の面のみで考えれば有用なミラーリング手法だということが分かる. しかし実験段階の GMFS では厳密なエラーチェックや整合性チェックを行っていないので, 今後エラーチェックや整合性チェック機能を実装して再度検証する必要がある.

7. 関連研究

ファイルシステムによるミラーリングに関する研究では, ディザスタリカバリを対象としたファイルシステムでのミラーリングとして SnapMirror[7] ☆☆☆がある. また, ファイルシステムのジャーナリングを利用した非同期ミラーリング[8] や RAID と NFS を組み合わせた NRFS[9] がある. これらの研究は独自ファイルシステムのみでミラーリングを行う方式であり, 既存ファイルシステム上に存在するデータは扱えない.

既存ファイルシステムを変更せずに機能を追加する研究として union mounts[10] がある. これは他のファイルシステムを透過的に扱うという視点では本研究と同じであるが, ミラーリングを目的とはしておらず, パス情報を柔軟に扱うことを目的としており, 本提案とは大きく異なる.

8. まとめと今後の課題

汎用的なミラーリングファイルシステムとして GMFS を提案し, ミラーリングを行う部分の実装を行った. またそれに対する性能評価を行い, ファイルシステムのラッパとしてミラーリングを行うことの有効性を検証した. その結果, この手法が実用的な速度で動作することが実証できた.

今後の課題として, 今回実装していなかった

- 対応していないファイルシステム関数
- エラーが発生した場合の対処
- 整合性チェック

を実装し, 再度検証する必要がある. また, 機能的な課題として次の項目が挙げられる.

- スナップショット機能
- データ複製個数を増やした場合の検証
- 非同期でのミラーリング

参考文献

1) Patterson, D.A., at al: A Case for Redundant Arrays of Inexpensive Disks (RAID), *Proc. ACM SIGMOD*, pp.109–116 (1988).
2) Fox, A., at al.: Cluster-Based Scalable Network Services, *Symposium Symp. on Operating Systems Principles*, pp.78–91 (1997).
3) 前野真輝, 松尾隆利, 山幡為佐久 : ext3 ファイルシステムへのスナップショット機能の設計と実装, *Linux Conference* (2005).
4) Braam, P.J.: The Coda Distributed File System, *LINUX JOURNAL*, pp.46–50 (1998).
5) SunMicrosystems, I.: NFS: Network File System Protocol Specification, *RFC1094* (1989).
6) Rosenblum, M. and Ousterhout, J.K.: The design and implementation of a log-structured file system, *ACM Trans.*, Vol.10, pp.26–52 (1992).
7) Patterson, H., at al: SnapMirror: File System Based Asynchronous Mirroring for Disaster Recovery, *USENIX Conference* (2002).
8) 藤田智成, 矢田浩二 : 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, 情報処理, Vol. 46, No.SIG16(ACS12), pp.56–68 (2005).
9) 松本尚 : ネットワーク RAID ファイルシステム, http://www.ssspc.org/nrfs/.
10) Pendry, J.-S. and McKusick, M. K.: Union mounts in 4.4BSD-Lite, *Proc. of the USENIX Technical Conference*, pp.25–33 (1995).

☆ http://www.ace.comp.nec.co.jp/clusterpro/doc/ver30/L30_env_21.pdf
☆☆ http://www.coker.com.au/bonnie++/
☆☆☆ SnapMirror は米国およびその他の国における Network Appliance, Inc. の登録商標.