ネットワーク RAID ファイルシステム
Network RAID File System

松本　尚 [1)]
Takashi MATSUMOTO

[1)] 国立情報学研究所 情報基盤研究系 計算機アーキテクチャ研究部門
(〒 101-8430 東京都千代田区一ツ橋 2–1–2　E-mail:**tmatsu@nii.ac.jp**)

ABSTRACT. The NRFS is a brand-new kernel-level subsystem for a low-cost distributed file system with fault-tolerant abilities. We call it "Network RAID File System (NRFS)" which does not require any dedicated-hardware for conventional RAID devices. Whereas the conventional RAID consists of multiple inexpensive disk drives, the NRFS consists of multiple inexpensive PCs/WSs that are members of a distributed environment and include some disk drive(s) inside. The NRFS was developed based on the NFS. The NRFS adopts a new error detection method for RAIDs "majority decision" which is an extension of the mirroring method. The method enable the NRFS to support not only disk crashes but also memory errors and/or controller (i.e. PC) errors. Since the NRFS has machine-level redundancies, it can replace a wrong disk or repair a server PC without stopping the NRFS services.

1 背　　景

パーソナルコンピュータおよびワークステーションの価格性能比の改善と低価格化は著しく、オフィスや工場や研究所はもちろん家庭や SOHO にも複数台の計算機が設置されるようになってきている。現在の計算機はアプリケーションや操作環境整備のためプログラムおよびデータの格納場所としてハードディスク装置を使用している。複数台の計算機を使用する場合に、分散共有ファイルシステムによってハードディスク装置を共有できると、一台ごとにディスク内容を整備する必要がないためにマシン管理コストが大幅に低減できる。また、計算機間のデータの移動が共有ファイルによって自然に行われるため、テープやフロッピーディスクを介したデータ移動よりも大幅に手間が少ない。また、明示的に通信プロトコルやファイル転送プログラムによってデータ転送するよりも簡単である。分散共有ファイルシステムでは十分に高速なプロトコルを使用して多くのシステムではデータ転送に必要な時間コストも十分に小さく抑えられている。これらの利点により、複数の計算機を導入している環境では、分散共有ファイルシステムが導入されている。

分散共有ファイルシステムは他のマシン上のハードディスクを仮想的に使用可能にする機構であるため、プログラムやデータは最終的にハードディスク装置内に保存される。ハードディスク装置は物理的に高速稼働する部分が多いために発生する装置自身がクラッシュする確率や、非常に高密度の磁気記憶を行っているため誤り訂正不能なデータ化けが発生する確率が無視できない大きさで存在する。重要なデータや時間をかけて整備した操作環境をハードディスク装置の故障から安全に守ることはシステムとして非常に重要な事項である。また、安価なハードディスク装置では、装置内に内蔵されている半導体メモリによるキャッシュ機構にエラー訂正機能がなく、このレベルでデータ化けを起こすといったトラブルも存在する。

2 目　　的

ネットワーク RAID ファイルシステム：NRFS（エヌアールエフエス）は低コストで高信頼の分散共有ファイルシステムの実現を目指して開発されたソフトウェアである。NFS をベースに開発され、書き込み時には複数の NFS サーバに同一のファイルを格納しておき、読み出し時には複数のサーバから読み出した内容に差異がないことを確認しながら、信頼性を確保する。複数（3 台以上）の PC を持っていれば、高価な専用高信頼ディスク装置を買うこと無しに、高信頼分散共有ファイルシステムである NRFS を使用することができるようになる。訂正可能な障害はファイルシステムが自動的に修正し、ディスククラッシュのようなハードウェア交換が必要な障害に対してもファイルシステムを停止させることなく交換作業を行うことが可能である。マシンレベルの冗長性を持っているため、故障ノードの電源を切断して修理作業を行ってもファイルシステムを運用できる（システムレベルのホットスワップ）。そして、故障ノードを再起動後、ソフトウェアによって自動的に交換されたディスクのファイルシステムを無故障状態に復帰させる。つまり、データのビット化け等を自動修正し、ディスククラッシュ時もファイルシステムが停止することなく運用でき、ディスククラッシュからの復旧に莫大な作業時間を取られる必要がなくなる。これらの利点のうち、ディスククラッシュに関する NRFS のメリットは既存の高信頼ディスク装置（RAID 装置等）でも得る

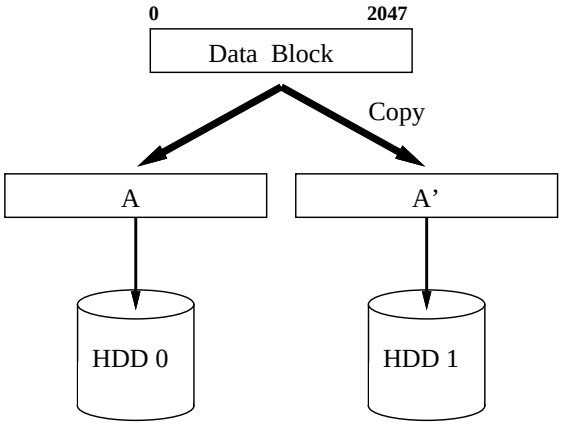

図 1　ミラーリング方式

ことができるメリットであるが、ホットスワップ可能な RAID 装置は高価である。データ化けに関しては既存の RAID 装置ではディスクドライブに起因するデータ化けは修正可能であるが、コントローラに関するデータ化けは検知できない。

以上に述べたように低投資コストで高信頼分散共有ファイルシステムが構築できるため、今まで以上に多くのプログラマおよびエンジニアがディスクトラブルに対する復旧作業から開放され、本来の業務に専念することができる。ファイルサーバ用のマシンを新しいマシンに交換する場合も、一台ずつ交換することにより、自動的にファイルシステムの中味を引き継ぐことができる。この様に低コストかつ便利でメンテナンスが容易な NRFS が普及すれば、マシンを買い替える度にローカルディスク上のアプリケーションをすべてインストールしなおす必要のある欠陥オペレーティングシステムによって引き起こされる無駄な作業からも人々を開放することができる。

3　従来方式について

NRFS について述べる前に従来の高信頼ディスク装置の方式について述べる。ハードディスク装置上にあるデータの安全性向上のために近年普及しつつあるのが、RAID (Redundant Array of Inexpensive Disks) 装置である [1]。RAID 装置では安価なハードディスクドライブを複数使用して、プログラムやデータを冗長な形で格納しておき、1 台（もしくは少数台）のディスクドライブが故障しても正しい情報が復元可能なディスク装置である。また、複数のディスクドライブを同時にストライピングアクセスすることにより、ディスク装置としての性能も単体ディスクよりも向上させることが可能である。

RAID における冗長なデータ形式への変換方式として、ミラーリング方式とパリティ方式が主に使用されている。ミラーリング方式はデータ処理に負荷をかけないことを重視した方式であり、ディスクドライブに格納すべきデータの完全なコピーを他のディスクドライブにも格納する（図 1）。パリティ方式はディスク容量の有

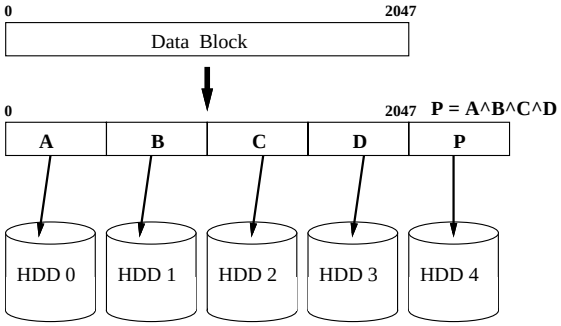

図 2　パリティ方式

効活用を指向した方式であり、例えば 2048byte のデータブロックを 512byte ずつ 4 つのサブブロックに分け、さらにサブブロック間の排他論理和による 512byte のパリティを計算し、5 台のハードディスクドライブに分散して格納する方式である（図 2）。パリティを格納するディスクをデータブロック毎に変えて分散させる方式もある。元データもしくはパリティを格納したディスクドライブが故障しても、故障が 1 台であれば、他のディスク上のデータから故障したディスクのデータが復元できる。

ミラーリング方式の RAID はディスクに格納するデータに関して変換をかけていないため、メイン CPU のソフトウェアによる実現が低コストで可能であり、ソフトウェアで実現されたシステムも多く存在する。これに対して、パリティ方式の RAID はパリティ生成を行うオーバヘッドが大きいため、ソフトウェアレベルの実装では性能が低下してしまう。ミラーリング方式によるソフトウェア RAID を採用したとしても、通常の SCSI や IDE で接続されたディスクに対してシステムを停止させずにディスク交換を行うこと（ホットスワップ）は不可能である。これらの理由から、高可用性と信頼性が求められる環境では、 RAID 装置は専用ハードウェアによって構成され、結果として高価なものとなっている。

本開発では高価な専用 RAID 装置を購入することなしに、ソフトウェアのみで低コストかつ効率よく高信頼分散共有ファイルシステム:NRFS を提供することを目的にしている。

4　NRFS の方式

4.1　計算機アレイによる RAID

分散共有ファイルシステムは複数の計算機によって共有されるため、ディスク故障等によってシステムがダウンすることの影響が非常に大きい。分散共有ファイルシステムには高い信頼性と高い可用性が不可欠である。このため、分散共有ファイルシステムのディスク装置として専用ハードウェアを搭載した RAID 装置を導入して信頼性を向上させる方式が一般的になりつつある。しかし、この場合は RAID 装置が接続される計算機（ファイルサーバ）に信頼性と性能が要求され、システムのコストが高くなってしまう。本開発では安価になった普及型のパーソナルコンピュータを活用してファイル

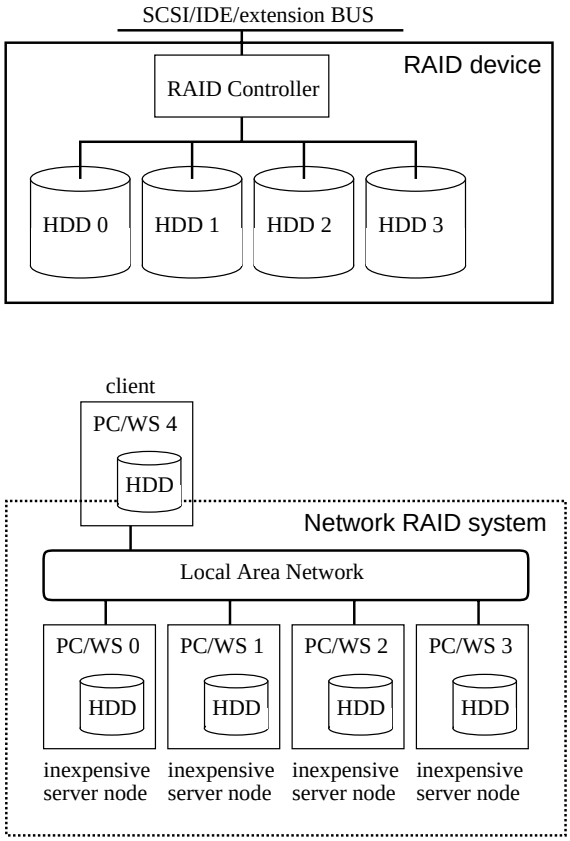

図 3　RAID vs. Network RAID

システムの信頼性を向上させる方式 [2]を採用する。つまり、 RAID は安価なディスク装置を複数台使って冗長性を増して信頼性を上げていたのに対して、ネットワーク RAID では安価な計算機を複数台使って冗長性を増して信頼性を向上させる（図 3）。ファイルシステムの信頼性向上目的のみを目指して新たに計算機を増設したとすれば実現コストが大きくなるが、分散共有ファイルシステムを必要とする環境には元々複数台の計算機が存在する。これらの資源を活用してファイルシステムの信頼性を向上させれば資源の有効活用であり、全体としてのコストは増大しない。計算機レベルで冗長性を持たせることにより、 RAID 装置の高コストの原因である、電源の多重化、ファンの多重化が自然に達成され、 1 台の計算機の電源を落しても他の計算機には影響がでないため活線挿抜のための専用回路も不要である。

4.2　障害検出訂正方式の検討

NRFS や従来方式の RAID 装置に限らず冗長なディスクドライブを持つ RAID 一般についてミラーリング方式とパリティ方式を比較する。パリティ方式はストライピングアクセスが方式自体に組み込まれているため、単純なミラーリング方式よりも性能面において有利である。しかし、ミラーリング方式もストライピングを併用することで性能向上を図ることが可能である（図 4）。実装上の問題を除けば、ミラーリング方式とパリティ方式の本質的な差はディスク容量の利用率のみということ

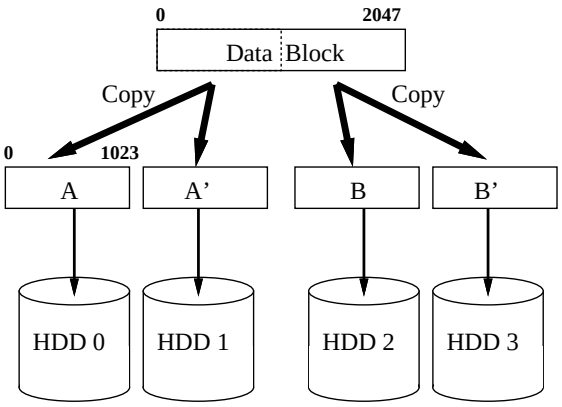

図 4　ストライピングを併用したミラーリング方式

になる。複数の計算機からなる分散計算機環境に分散共有ファイルシステムを導入して、プログラムやデータの共有を推し進めると、各計算機のディスク容量は余り気味になることが多い。これらの余剰ディスクを計算機資源と一緒に有効活用するファイルシステムが NRFS であるため、ディスク容量の利用率は大きな問題とはならない。このため、実装が低コストかつ簡単なミラーリング方式もしくはミラーリングを拡張した方式が NRFS に適している。

ミラーリングをネットワークレベルに拡張した方式として、複数台（2 台）の計算機上のディスクに同一のデータを格納しておき、ディスク障害が発見された場合のみ予備のデータを他の計算機から獲得するという方式が考えられる。しかし、安価な計算機を冗長に使用する場合は、ディスク装置の故障のみではなく計算機自体のデータ化けやクラッシュも無視できない。計算機自体のエラーまで考慮に入れて信頼性を考えると、前述の RAID のミラーリングを単純にネットワークを介して実現した方式では不十分である。計算機レベルでエラーが発生している可能性がある場合は、クライアント側（ファイルをアクセスする側）でデータ読み出し時にデータの正しさを確認する必要がある。つまり、データ読み出し時にミラーされた計算機（サーバ）から複数のコピーを読み出して、本当にデータが一致しているか確認を行う必要がある。データ使用時にデータ不一致が発見されても、ディスク故障以外の場合、どのサーバから送られたデータが間違えているか確定できない可能性があるため、ミラーの数は最低 3 セット必要であり、「多数決原理」によって正しいデータを決める必要がある。ディスクレベルでは CRC チェック等によりエラーが発生したディスクのセクタが特定できるが、書き込み前に ECC なしメモリやキャッシュ上でデータ化けが発生した場合等は、冗長度が 2 ではエラー発生側を特定できない。多数決原理で正しいデータを確定するため、本方式を今後「多数決方式」と呼ぶことにする。

4.3　NRFS の実装方式

前節において、ネットワーク RAID ではクライアント側において多数決による格納データの認証を行う必要が

あることを述べた。クライアント側が複数のミラーサーバからデータを取り寄せて、データを逐一比較していたのではクライアント側のメイン CPU に掛ける負荷が大きくなってしまう。また、ネットワーク（LAN）のバンド幅が小さい場合には、複数のミラーサーバからデータを転送することが分散共有ファイルシステムとしてのボトルネックになりかねない。さらに、複数のミラーサーバと通信を行うこと自体が通信オーバヘッドを引き起こし、システム全体の性能を低下させる可能性がある。これらの問題に対する解決策／改善策となる実装方式について本小節で述べる。

データ転送量を削減する方式

データのディスクへの書き込みはすべてのミラーサーバに反映される必要があるため、ネットワーク上のデータ転送量を削減することは原理的に不可能である。しかし、読み出しに関してはデータ転送量削減の可能性がある。データの誤りが高い確率で検出できることがクライアント側の認証の目的である。そこで、サーバが読み出したデータに対応する認証データ（チェックサムや MD5 [3]や SHA-1 [4]等のハッシュ関数）を一定サイズ（通信パケットサイズ）毎にサーバ側で計算して、1 台のサーバはデータと認証データをクライアントに送信し、他のサーバは認証データのみを送信して、クライアントは認証データが一致していることを確認する。そして、認証データが一致していれば、データも一致していると信用してデータを使用する。認証データの bit 幅を大きくすればデータ誤りの検出率を高くすることができる。この方式はクライアント側の処理の一部をサーバ側に分散する方式でもあり、サーバに MD5 や SHA-1 計算用のハードウェアアクセラレータがある場合に特に適している。これらのアクセラレータはセキュリティ対策のために高価なサーバマシンには徐々に普及しつつあるが、ネットワーク RAID がターゲットとするような安価なパーソナルコンピュータには現状では実装されておらず、将来的にそこまで普及するかどうかは定かでない。ただし、ネットワークバンド幅不足がネットワーク RAID 方式のボトルネックとなっている場合には、この実装方式は非常に効果が高い。

NIC を利用して認証負荷を削減する方式

ネットワーク RAID は分散共有ファイルシステムを対象としているため、本質的に計算機間の通信が発生する。事実上の標準通信プロトコルである TCP/IP および UDP/IP にはデータのチェックサム機能が定義されているため、最近の多くのネットワークインタフェースカード（NIC） [5]にはチェックサム計算支援用ハードウェアが搭載されている。各サーバは生のデータをネットワーク経由でクライアントに転送すると、各サーバの NIC はチェックサムを計算して通信パケットに付与する。このチェックサム値（16bit）を認証データとして流用して、クライアント側ではチェックサムが一致していることによりデータが一致していると見做す。ただし、TCP/IP や UDP/IP で通信を行った場合はチェックサムの範囲がヘッダの一部を含むため、若干補正計算を行う必要がある。なお、チェックサム計算支援ハードウェアを持たない場合でも、分散共有ファイルシステムの通信プロトコルとして TCP/IP（または UDP/IP）を用いる場合は通信レイヤで計算されるチェックサムを流用することが可能である。

チェックサムが一致してエラー発生が確認されない通常ケースでは、複数のサーバノードからデータ本体を送信することはネットワークバンド幅の浪費である。既存の NIC を流用する場合にはこのオーバヘッドはやむを得ないが、ネットワーク RAID 対応 NIC を新たに開発する場合は、チェックサムだけ転送できるような通信オプションを用意することで、この無駄を無くすことができる。ただし、Gigabit クラスの高速ネットワークでは、通信オーバヘッドによってバンド幅を使い切ることが難しいため、この程度のバンド幅の浪費は問題とならない。

また、多数決方式といっても不整合が発見されるまでは、2 ノードのデータの一致のみで運用を続けられるので、クライアントにデータ（もしくは認証データ）を転送するのは 2 台のサーバで十分である。2 ノードしかデータ（もしくは認証データ）を転送しない場合はアクセスが一部のミラーサーバに偏っていると、データ誤りを長期間見落として放置してしまう可能性が高くなる。このことは回復不能なエラーの発生率を高めるので、アクセスするミラーサーバの選択制御に十分な注意が必要である。

5 Linux 版 NRFS の実装について

5.1 Linux 版 NRFS の概要

前述の NRFS 方式に従って、Linux の NFS (Network File System) [6]を改造して作成した NRFS 機能を VFS（Virtual File System）から使用可能にしたものが Linux 版 NRFS である。Linux 版 NRFS の実装と密接な関係がある NFS や VFS がカーネル内の実装であるため、Linux 版 NRFS はカーネル依存のコードとならざるを得なかった。今回開発した Linux 版 NRFS は 情報処理振興事業協会の平成 12 年度未踏ソフトウェア創造事業に採択されて、開発が始まった。このため、平成 12 年 10 月当時の最新安定版カーネルであるカーネル 2.2.16 を対象として開発が行われた。よって、公開／配布されるソースやバイナリは **Red Hat Linux カーネル 2.2.16 専用**のものである。逆に言えば、今回の NRFS を含むバイナリを Linux カーネルとして採用していただいた場合はカーネルバージョンは 2.2.16-3 となる。十分ご注意されたい。なお、NRFS 対応カーネルの入手先の URL は

`http://www.ssscore.org/nrfs/`

である。

上記で公開されるプログラムは、開発チームが保有するいくつかの環境での動作は確認されていますが、すべての環境で動作することは保証できません。高信頼ファイルシステムを目指して開発を行っていますが、我々が気がついていないバグや我々が想定していないユーザの使用方法によって、ユーザの重要なファイルが破壊されても責任をとることはできません。ユーザ自らのリスク

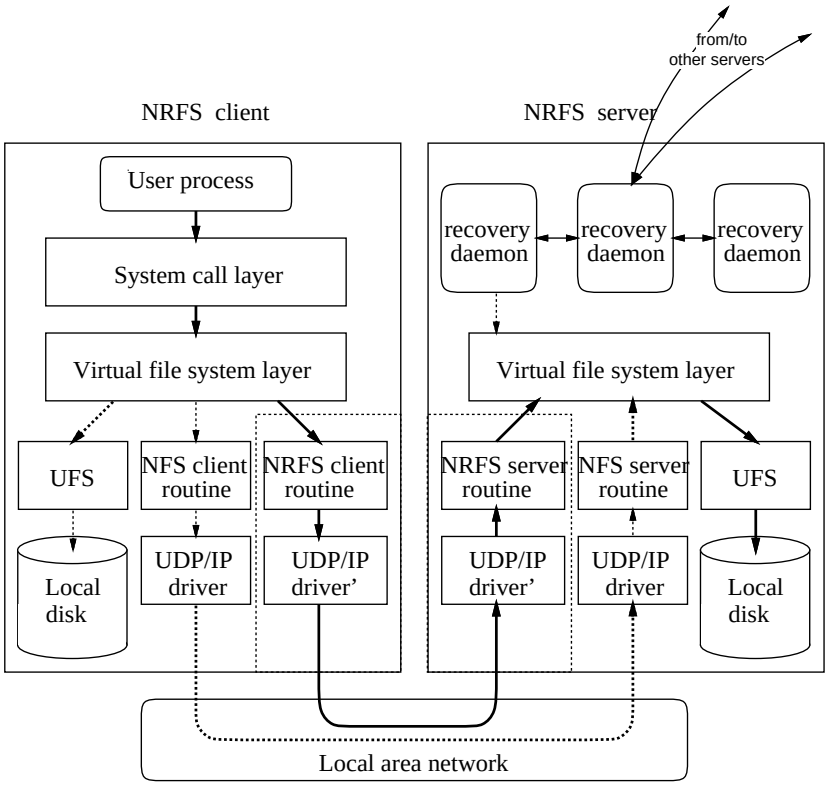

図 5　ファイルシステムの構成

で使用して下さい。

5.2　Linux 版 NRFS の構成

前述のように NRFS は NFS の拡張として実装されるため、 VFS がサポートするファイルシステムの一種類としてオペレーティングシステムに組み込まれる。図 5に NRFS を含むファイルシステムの構成を示す。図では便宜上 NRFS クライアントと NRFS サーバが別のノードとして記述されているが、通常は各ノードはファイルシステムのある部分に関してはサーバとして振舞い、他の部分ではクライアントとして振舞う。この辺りの事情は NFS と同じである。

NRFS システムはクライアント側が NRFS client routine と UDP/IP driver (RPC) で構成されており、サーバ側が NRFS server routine と UDP/IP driver (RPC) で構成されている。また、サーバ側にはユーザプロセス（root 権限）として recovery daemon（リカバリデーモン）が常駐しており、障害復旧の程度に応じて fork によって子プロセスを作りながら、障害復旧処理を行う。障害を発見したクライアントは障害のないサーバに対して、復旧要求を NRFS の RPC の一貫として発行する。これを受け取った NRFS server は自分のノードのリカバリデーモンに対して復旧要求を通知して、復旧処理を開始させる。リカバリデーモンはクライアントからの情報をヒントとして利用して、多重化されたサーバのリカバリデーモン間で連絡を取り合って、クライアントの認証と障害の確認を行い、障害を確認できた場合のみ障害復旧処理を行う。

5.3　開発のキーポイント

NRFS の方式自体は前節に示したとおりであり、 NFS を流用することにより開発開始当初は簡単に実装可能な予定であった。しかし、いざ実装を開始すると、動作原理を考案した時には思い至らなかった問題が数多く発生した。その中で、技術的に面白い話題についていくつか紹介する。

ステートフル or ステートレス

NFS はサーバがダウンしてもリブートすればコネクションが回復することから、ステートレスの通信によるファイルシステムのように一見思われる。しかし、内部を解析すると、ファイルハンドルと呼ばれるサーバ側のローカルファイルシステムの i-node 情報をベースにしたサーバ依存の状態や各種キャッシュを使ってファイルアクセスの高速化を図っている。一見ステートレスに見えるのは、それらのキャッシュや状態を注意深くフラッシュする機構が付いているからであり、毎回フラッシュするような本当のステートレスな実装では大幅に性能がダウンしてしまう（将来マシンパワーに物を言わせてこちらの実装にする計画もあるようだが）。このため、NFS をベースに行った NRFS 開発では、 NFS クライアント側のサーバ依存の関数や構造体の詳細を完全に把握して、サーバを多重化する方法を開発する必要があった。基本的には、サーバ依存の構造体を多重化して、クライアント側の通信層の直前でサーバアクセスを多重

化して、複数のサーバにアクセス要求を行っている。また、NFS サーバをアクセスする際の基本情報として使用されるファイルハンドルはシステム依存の構造体であり、今回の Linux 版 NRFS の開発では、Linux 固有の多重化方式によってファイルハンドルの一部のみを多重化している。ファイルハンドルの構造がシステム依存であることに気がついていなかったため、この実装になっているが、本来はファイルハンドル全体を多重化するべきである。

VFS におけるキャッシュの取り扱い

VFS 層にはファイルキャッシュやディレクトリキャッシュが実装されている。NRFS として複数のサーバが mount されていても、VFS としてはあくまでも一つの実体として見えないと、VFS までも書き換える必要がでてくる。このため、NRFS では VFS からは NFS とまったく同じ形に見えるように、VFS-NFS 間のインタフェースをそのまま流用した。しかし、VFS は各サーバの i-node 番号（ファイル固有の ID）をキャッシュの ID（ファイル ID）として活用しており、このファイル／ディレクトリに付随する固有番号がないとキャッシュにおけるエントリの一致が判定できない。そこで、VFS のファイル ID に関する処理のみは、NRFS のために拡張することにした。i-node 番号の上位にサーバ ID（多重化しているサーバ間の ID）を付加したものを NRFS ではファイル ID とすることにした。当然、多重化されたサーバ数のファイル ID が存在するが VFS にはその内の一つのみが引き渡される。ただし、VFS から NRFS に関するファイル ID の問い合わせがあった場合は、保持しているすべてのサーバに対応した i-node 番号を検索して、問い合わされたファイル ID と対応するものがあるかどうかチェックする。この変更により、サーバノードをまったく対等に扱うことが可能となり、かつクライアント側のキャッシュを無駄にフラッシュする必要もなくなる。

RPC における非同期タスクの活用

Linux の NFS は性能の向上を図るため、ファイル読み出しやファイル書き込みは先読みやバッファリングを活用して、大きな単位で行っている。このため、RPC レベルでは複数の RPC 要求に分割されるが、これらを RPC の非同期タスクという機構を使って同時処理している。非同期に対して同期タスクというのも存在するが、このタイプの RPC が行われると、これに続く RPC はこの同期タスクの RPC の返答が戻るまで遅延される。これに対して、非同期タスクの RPC では、後続の RPC を可能であれば、発行することによって、処理性能を向上させる。NRFS では、通常同期タスクとして処理される lookup や readdir であっても、複数のサーバに対して実行され、サーバ間には依存関係はない。このため、同一メッセージに対する複数サーバ向けの RPC メッセージは非同期タスクとして扱うこととして、性能向上を図った。

ディレクトリチェックサムキャッシュ機構の新設

NFS では 1 ディレクトリにメモリ 1page 分では納まらない容量のエントリがある場合には、複数回の readdir を発行してディレクトリのエントリ名に関する情報を収集する。こういった大きなディレクトリの 2 回目以降の readdir では前回までに次に読み出すエントリの位置に関するサーバ依存の情報を提供する必要がある。この情報は直前の readdir の実行結果といっしょにクライアントに戻されている。ディレクトリ内のエントリ名の格納位置はサーバに依存しており、同じ名前のエントリであってもマシンが異なれば同じ位置にあることは期待できない。このため、NRFS ではディレクトリの誤り検出をディレクトリ全体の内容のチェックサムを用いて行うこととし、一回目の readdir においてチェックサムデータをすべてのサーバから取り寄せることにした。チェックサムの値が一致していれば内容は正しいものとし、大きなディレクトリに対する 2 回目以降の readdir は正しいと信用されるサーバ 1 台のみを対象として実行する。当然、全部のチェックサムが一致しない場合には、多数決に勝ったサーバが正しいと信用される。このように NRFS の readdir はディレクトリの全エントリに対するチェックサム計算結果を要求する。readdir の度にこのチェックサム値を計算するのは、ディレクトリのサイズが大きい場合には無視できないオーバヘッドが発生すると考え、ディレクトリチェックサムキャッシュ機構をサーバ側に新設することにした。名称通りに一度計算したディレクトリのチェックサムの値をキャッシュしておく機構である。もちろん、ディレクトリのエントリの更新や削除があった場合には該当キャッシュエントリはフラッシュされる。

ディレクトリチェックサムキャッシュ機構の更新

前小節で述べたように readdir の認証用のチェックサム計算の高速化のためのキャッシュ機構を新設した。しかし、最初に作った機構は、NFS や NRFS からのファイルアクセスでは厳密にキャッシュがフラッシュされて常に正しい値がキャッシュされるように管理されるが、サーバ側でローカルファイルシステムとしてファイルやディレクトリを直接操作した場合にはキャッシュがうまくフラッシュされないという問題が発生した。NRFS の障害復旧機構は NFS の枠組みで障害を取り除くのではなく、通常の通信手段とサーバ上のデーモンによるローカルファイルシステムの操作によって障害復旧を実現している。このため、障害復旧機構が働いて障害を取り除いても、ディレクトリチェックサムキャッシュの値が古いままで、エラーが解除されないという事態が発生した。この問題は NFS のサーバにあるキャッシュをどうやってローカルファイルシステムの操作と同期させるかという一般的な問題である。Linux では、i-node に i_version というメンバを設けて、NFS とローカルファイルシステム（EXT2 や UFS）のどのパスから実ファイルシステムが操作された場合でも、この i_version をインクリメントすることによりバージョン管理していることが判明した。つまり、NFS は NFS でサーバ側に独自のディスクキャッシュを持つことがあるが、このキャッシュに i_version の値もキャッシュしておき、キャッシュ参照された時には通常のヒット判定の他に i_version が現在の値と一致していることを確認する。

もしも一致していなければ、ディスクから最新データを読み込み直す。ディレクトリチェックサムキャッシュ機構にもこの i_version によるバージョン管理を導入することで、実際のディレクトリのエントリとチェックサムキャッシュが矛盾する事態を回避できるようになった。

時計のずれに関する問題

NRFS は複数台のサーバに同じファイルシステムを保持することが高信頼性を得るための冗長性の根源である。NRFS がベースにした NFS ではファイルやディレクトリの生成時刻や更新時刻はサーバ側の時計によって管理されている（ただし、NFS Ver.2）。複数あるサーバの時刻が完全に一致していれば問題にならないわけであるが、完全な一致を期待することは不可能であり、NFS と同じ時刻の取扱いをするためには、どのサーバの時刻をユーザに提示するかという問題が発生する。そこで、NRFS ではサーバ内のローカルファイルシステムには該当サーバの時計で刻印するが、クライアントに示す時刻はあくまでもクライアントの時計に基づいて決定することにした。つまり、クライアントと各サーバの時計の時差をあらかじめ測定しておき、この時差で補正した値をファイルやディレクトリの時刻としてユーザに提供するわけである。この方式であれば、ユーザはクライアントの時計と矛盾した時刻の提示は受けず、サーバ間の時刻のずれも大きな問題とはならない。逆に、大きく時刻がずれていても使用中にずれが大きく変化しなければ、問題なく使用可能である。つまり、NRFS のサーバを世界各地に配置して同一マウントポイントに接続しても、時刻のずれによる問題は発生しない。

パス情報の復元

前述のように、NRFS は NFS をベースに実装されたため、NRFS 層においてファイルハンドルをサーバごとに管理して、複数のサーバに対するアクセスを行っている。よって、NRFS にとっての根源的な情報はファイルハンドルということになる。しかし、ファイルハンドルは個別のサーバ依存の情報であり、本来根源的な情報はユーザ／アプリケーションが指定した（ファイル）パス情報である。このパス情報が VFS 層から NFS 層・RPC 層に渡る間に分解され、原型が判らなくなってしまってファイルハンドルベースでサーバ・クライアント間の通信がなされている。障害復旧機構にクライアントから障害をレポートする場合に、可能であればパス情報を教える方が見通しがよい。なぜなら、ファイルハンドルを用いて指示を出す場合は、サーバ毎のファイルハンドルをすべて示さないと意味がない。しかし、マウントポイントからのパス情報であればすべてのサーバが理解することが可能である。そこで、VFS が管理するディレクトリキャッシュ（dentry）の内容を逆にたどることにより、パス情報が復元可能なことを突き止めて、障害復旧機構への連絡はパス情報に基づいて行うこととした。

6 Linux 版 NRFS の使用方法

client1 というクライアントマシンにおいて

```
serv1:/export/serv1, serv2:/export/serv2,
serv3:/export/serv3,
```

を /usr/nrfs に NRFS マウントする場合を考える。あらかじめ、

```
serv1:/export/serv1, serv2:/export/serv2,
serv3:/export/serv3,
```

が client1 に rw で root アクセス可能で export されている（client1 から NFS マウント可能であり読み書き可能になっている）必要がある。三つのマウントされるディレクトリの内容は一致していることを仮定する。初回は三つのサーバディレクトリが空の状態でマウントされることを推奨する。また、NRFS のサーバディレクトリはエラー訂正機能のテスト以外では、必ずすべてのサーバディレクトリを NRFS マウントした状態でクライアント経由でアクセスする必要がある。サーバマシンからローカルにサーバディレクトリのファイルの読み書きを行うと、サーバ間のディレクトリの内容に差異が生じる。export の準備を行った後に、以下のように client1 において root 権限で三回 mount コマンドを発行する。

```
client1# mount serv1::/export/serv1 /usr/nrfs
client1# mount serv2::/export/serv2 /usr/nrfs
client1# mount serv3::/export/serv3 /usr/nrfs
```

mount の発行順序は問題とはならない。mount コマンドで NFS マウントと NRFS マウントの差はマシン名の後ろが「:」が「::」になっている点のみである。最大８ディレクトリまで同一のマウントポイントにマウントできる。通常の使用では３ディレクトリ（３サーバ）で使用することを推奨する。この状態で df を取ると図 6に示されるような表示がなされる。

この表示から /usr/nrfs のディレクトリには三つのサーバのディレクトリが NRFS mount されていることが判る。なお、具体的な数値は環境により変わる可能性がある。

この後、/usr/nrfs の下にファイル／ディレクトリを作れば、三つのマシンに同じファイル／ディレクトリが作成される。ファイルやディレクトリを読み出せば、すべてのサーバから読み出して多数決の結果、多数側の情報を返す（厳密には、一つのサーバからデータとチェックサムを読み出し、他のサーバからはチェックサムだけ読み出して、クライアントでチェックサムの比較を行う）。故意にサーバマシンのファイルを削除や追加して、その操作を行なったディレクトリ／ファイルに対してクライアントからアクセスを行えば、リカバリデーモン (/usr/sbin/recovery) に修正要求が飛んでファイルが修正（or 追加 or 削除）される。

安全性は保証できないが、NRFS 内でファイル作成中やコンパイル中に一台のサーバのネットワークを故意に外しても、少しの間故障確認のため停止した後、残りのサーバでファイルシステム機能は継続され処理が続行される。その後、外したネットワークをつなぎ直せば、また全サーバの動作が復旧し、接続切断中に起こった不具合は、その不具合のあるディレクトリ／ファイルをクライアントがアクセスした際に、復旧される。ただし、完全復旧前に他のサーバのネットワークを外したりすると、３台サーバ構成では多数決が成立しなくなってファ

```
client1# df
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/sda8               256667     61921    181494  25% /
/dev/sda1                23302      9872     12227  45% /boot
/dev/sda6              8024868    467060   7150164   6% /home
/dev/sda5              8024868   1194400   6422824  16% /usr
/dev/sda7               256667     68587    174828  28% /var
serv1::/export/serv1    256667     84465    158950  35% /usr/nrfs
serv2::/export/serv2    256667     84460    158955  35% /usr/nrfs
serv3::/export/serv3    256667     84696    158719  35% /usr/nrfs
client1#
```

図 6　df コマンドの結果

イルシステムが見えなくなる可能性がある。実験時には、十分ご注意されたい。

7　Linux 版 NRFS の基本性能評価

デスクトップマシンとサーバマシンを使って基本性能評価とディレクトリチェックサムキャッシュの効果の評価を行った。本章ではその結果を示す。

7.1　デスクトップマシンによる基本性能評価

以下のようなスペックのデスクトップマシンを使用して、2Gbyte のファイルを local ディスクから NFS もしくは NRFS に転送する時間を測定する実験と、逆に NFS もしくは NRFS から local ディスクに転送する実験を行った（Large File 転送）。

- Pentium-III 800MHz, 128MB, 440BX
- Celeron 850MHz/128KB, 128MB (Server 4)
- NIC 3COM 3C905B-J-TX
- Red Hat Linux 6.2J Second Edition / Kernel 2.2.16-3

結果は以下の通りである。ただし、NRFS3 はサーバ 3 台（3 冗長度）の NRFS、NRFS4 はサーバ 4 台（4 冗長度）の NRFS である。

表 1　Large File 転送（2.0GB のファイルコピー）

操作	時間 (秒)	性能 (MB/ 秒)
local から NFS へ	1001	2.00
local から NRFS3 へ	1647	1.22
local から NRFS4 へ	2152	0.93
NFS から local へ	967	2.07
NRFS3 から local へ	1236	1.62
NRFS4 から local へ	1231	1.62

この結果から、NRFS へのファイル書き込みの性能がかなり NFS より劣っていることが判る。ただし、このデスクトップマシンでは NFS への書き込みが 2Mbyte/sec しか性能が出ていないため、書き込み時にはすべてのサーバにデータのコピーを送っていることを考慮に入れると、この性能劣化は通信自体のオーバヘッドに拠るものだと考えられる。このため、高性能なイーサネットドライバや Gigabit Ethernet 等の高速通信を使用すれば大幅に改善されるはずである。特に、この NIC 用のイーサネットドライバの性能が悪いことが大きく影響していることが、サーバマシンにおける別の NIC の 100BASE における性能から類推される。この推論の正否は、後述のサーバマシンを使った測定によって示される。読み出しがそれほど劣っていないのは、読み出し時には 1 台のサーバのみがデータ転送を行い、他のサーバはチェックサムのみを転送するため、ネットワークへの負荷が軽いからだと考えられる。

次に同一スペックのデスクトップマシンを使用して、たくさんのファイルが含まれる大きなディレクトリのコピーを行った（Many Files 転送）。サンプルに使ったディレクトリは Linux のカーネルソースディレクトリを複数並べて作った 2.271Gbyte のディレクトリである。

表 2　Many Files 転送（2.271GB のディレクトリコピー）

操作	時間 (秒)	性能 (MB/ 秒)
local から NFS へ	1403	1.62
local から NRFS3 へ	2337	0.97
local から NRFS4 へ	2864	0.79
NFS から local へ	1360	1.67
NRFS3 から local へ	1652	1.37
NRFS4 から local へ	1820	1.25

やはり、NRFS への書き込みが NFS に比べて大きく劣っている。この理由は、Large File 転送の場合と同様に、100BASE-TX のネットワークがドライバのオーバヘッドも含めて飽和してしまっていると考えられる。

7.2　サーバマシンによる基本性能評価

以下のスペックのサーバマシンを使用して、デスクトップマシンと同様に Large File 転送と Many Files 転送の性能評価を行った。ただし、総データ転送サイズを若干小さくして実験を行った。

このサーバ機には 100BASE と Gigabit Ethernet が搭載されている。両者による性能の違いを知るために、

それぞれのネットワークで実験を行った。ただし、表中の「100」が 100BASE を「GbE」が Gigabit Ethernet を表す。

- Pentium-III 1GHz, 512MB, ServerSet III HE-SL
- 100BASE-TX:intel82559, GbE:SysKonnect SK-9843
- Red Hat Linux 6.2J Second Edition / Kernel 2.2.16-3smp

表 3 Large File 転送（1.26GB のファイルコピー）

操作	NIC	時間 (秒)	性能 (MB/秒)
local から NFS へ	100	116.1	10.4
local から NFS へ	GbE	51.4	23.4
local から NRFS へ	100	346.2	3.47
local から NRFS へ	GbE	85.4	14.1
NFS から local へ	100	136.1	8.83
NFS から local へ	GbE	64.13	18.74
NRFS から local へ	100	158.2	7.60
NRFS から local へ	GbE	95.7	12.56

なお、この測定において NRFS はサーバ 3 台の構成である。まず、デスクトップ機よりも圧倒的に 100BASE の接続でも NFS の性能が高いことが注目される。これはマシン自体のスペックが高いことよりも、NIC のドライバの性能が大きく響いている可能性が高い。100BASE の書き込みでは NFS よりも NRFS は 3 分の 1 の性能であるが、GbE であれば 4 割程度しか性能が低下していない。

やはり、100BASE ではネットワーク負荷がかかり過ぎることが判る。また、GbE を使用した NRFS は 100BASE の NFS よりも性能が高いため、100BASE の NFS の性能に満足しているユーザであれば、GbE 環境でより高性能と高信頼性を享受することが可能である。もちろん、ディスクトラブルは少々性能がダウンしても回避したいユーザには現状の NRFS でも十分使ってもらえると考えている。

次ぎにサーバマシンにおける Many Files の転送実験の結果を示す。

表 4 Many Files 転送（1.429GB のディレクトリコピー）

操作	NIC	時間 (秒)	性能 (MB/秒)
local から NFS へ	100	298.6	4.56
local から NFS へ	GbE	189.3	7.20
local から NRFS へ	100	564.2	2.42
local から NRFS へ	GbE	284.1	4.80
NFS から local へ	100	308.0	4.43
NFS から local へ	GbE	234.5	5.81
NRFS から local へ	100	369.5	3.69
NRFS から local へ	GbE	301.0	4.53

単一ファイルの転送よりも NFS, NRFS 共に大幅に性能が悪い。しかし、GbE を使った NRFS は 100BASE の NFS よりも性能が高い。

7.3 ディレクトリチェックサムキャッシュの効果

デスクトップマシンとサーバマシンを使用して、ディレクトリチェックサムキャッシュが存在しない場合と存在する場合に関して、checksumreaddir の所要時間を測定した。測定したのは、クライアントがサーバに checksumreaddir を発行して結果が返るまでの時間である。

デスクトップマシンとして以下のスペックのマシンを使用した。

- Pentium-III 800MHz, 128MB, 440BX
- NIC 3COM 3C905B-J-TX
- Red Hat Linux 6.2J Second Edition / Kernel 2.2.16-3

表 5 checksumreaddir の所要時間（ミリ秒）

エントリ数	1,000	2,000	10,000
総文字数	20,000	40,000	200,000
キャッシュなし	2.376	2.703	5.456
キャッシュあり	2.089	2.096	2.096

エントリ数は checksumreaddir の対象となったディレクトリが何個のエントリ（ファイルまたはディレクトリ）を持っているかを示し、総文字数はそのディレクトリに含まれるエントリの名前の総文字数である。1,000 エントリ、20,000 文字は少し想定ディレクトリが大きすぎるきらいはあるが、100BASE のネットワークではレーテンシが大きいにもかかわらず、有意な時間差が観測された。

8 参加企業および機関

NRFS の研究開発には以下の 4 つの個人および企業が参加している。

- 松本尚
  平成 12 年度当時東京大学大学院理学系研究科所属,
  平成 13 年度は時間外兼業で個人として実施
- 三菱マテリアル株式会社
- 三精システム株式会社 (平成 12 年度)
- 株式会社フューチャーテクノロジー (平成 13 年度)

なお、情報処理振興事業協会（IPA）の平成１２年度未踏ソフトウェア創造事業「ネットワーク RAID ファイルシステムの開発」（長谷川正治プロジェクトマネージャ）、平成１３年度未踏ソフトウェア創造事業「Linux 版ネットワーク RAID ファイルシステムの実用化」（新部裕プロジェクトマネージャ）として松本の NRFS 開発提案を採択していただいて、これらの予算を基に上記三社には松本からの委託開発業務として NRFS の開発に参加していただいた。特に、三菱マテリアルの黄強（Huang Qiang）さんと鏑木健二さんには大きな貢献をいただいた。なお、三菱マテリアルには事務一般を行うプロジェクト管理会社の任も行っていただいた。

9 おわりに

Linux 2.2.16 カーネル用 NRFS は、動作保証や安全性を保証することはできないが、かなりの完成度で動いている。元々、NRFS は松本が開発中のスケーラブルオペレーティングシステム *SSS–CORE* [7] [8] の分散共有ファイルシステムとして考案されたものであり、Linux への実装は考えていなかった。しかし、NRFS の有用性を多くの人に享受してもらうためと、NFS をベースとした実装を行えば簡単に実現可能であろうという見通しから、Linux 版 NRFS を *SSS–CORE* 版に先行して開発することにした。平成 12 年 10 月の開発開始当初は Linux カーネルの改造を甘く考えていたため、開発に 2 年（実質 16ヶ月）もかかるとは考えていなかった。Linux の NFS を改造するために Linux カーネルに詳しい人材を探したが、日本国内ではなかなか見つけることができずに、結局 Linux のファイルシステムのソースコードを読むことから開発を始めた。最初の年度は未踏事業の期間が短かったこともあいまって、Linux の NFS と VFS の構成を理解するのに大半の時間を費やし、実用とはほど遠い玩具レベルのプロトタイプしか開発できなかった。次年度である平成 13 年度の実用化と成果のオープンソースリリースをかかげて再び採択していただき、何とか公開できるレベルの成果を完成させることができた。平成 13 年度の開発を始めた 7 月の時点では 2.2.16 カーネルはかなり古いバージョンとなっていたが、2.2.19 以降のカーネルでは NFS のコードが大幅に変更されているため、NRFS として開発を終了させることを優先させてあえて 2.2.16 を使用して開発を継続した。開発チームメンバの多くが、gcc, PC-UNIX, Linux といったオープンソースのソフトウェアの恩恵を受けているため、NRFS をソース公開することには抵抗は少なかった。ただし、開発は完全にクローズド体制で行わせていただいた。NRFS はカーネル内の一つのまとまった機能であり、これをネットワーク上で分散開発しても効率が落ちるだけでメリットはないと判断した。Linux の解析と詳細仕様作成とコーディングを並行して行うため、頻繁に開発者が一堂に集まって長時間かつ有意義な技術打ち合せを重ねながら、開発を進めた。開発を一段落させた現段階で振り返っても、オープン体制で NRFS 開発が行えたとは到底思えない。

NRFS の本格的な普及を目指すためには、以下の項目をクリアする必要があると思われる。

1. Linux 2.4.x 以降への NRFS の対応
2. lockd の NRFS への対応
3. ストライピングによる高性能化
4. NIC のハードウェアチェックサム機構の流用
5. 素人ユーザでも NRFS を設定できるツールを用意
6. NRFS マウントしているディレクトリをローカルにアクセスできなくする

今後これらの課題に挑戦する機会があれば、ぜひ挑戦してみたいと考えている。もちろん、誰か代わりにやってくれる方が現れれば、それも非常にありがたいことであると考えている。ともかく、NRFS の使用に際して私達は何の保証もできませんが、好奇心旺盛な Linux ユーザに NRFS を使ってみてもらって、ご意見やご感想がいただければ、何よりも嬉しいです。

参考文献

[1] D. A. Patterson, G. A. Gibson and R. H. Katz: A Case for Redundant Arrays of Inexpensive Disks (RAID). In *Proc. of Int. Conf. on Management of Data*, pp. 109–116 (September 1988).

[2] 松本 尚: NIC を活用したネットワーク RAID 方式の提案. 情報処理学会研究会報告 Vol.2000, No.74, pp.79–84 (August 2000).

[3] R. Rivest: The MD5 Message-Digest Algorithm. RFC 1321, (April 1992).

[4] National Institute of Standards and Technology: FIPS PUB 180-1:Secure Hash Standard. U.S. Department of Commerce, (April 1995).

[5] Sun Microsystems, Inc.: Fast Ethernet, Parallel Port, SCSI (FEPS) User's Manual Revision 1.0. Sun Microsystems, Inc., (April 1996).

[6] Sun Microsystems, Inc.: NFS: Network File System Protocol Specification. RFC 1094, (March 1989).

[7] 松本 尚, 他: 汎用並列オペレーティングシステム SSS–CORE の資源管理方式. 日本ソフトウェア科学会第 11 回大会論文集, pp.13–16 (October 1994).

[8] 松本 尚, 他: 汎用超並列オペレーティングシステムカーネル *SSS–CORE*. 第 17 回技術発表会論文集, 情報処理振興事業協会, pp.175–188 (October 1998).