修 士 論 文

ヘテロジニアスな環境における
自律分散ファイルシステムに関する研究

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

渡辺 浩二

2005 年 3 月

修 士 論 文

ヘテロジニアスな環境における
自律分散ファイルシステムに関する研究

指導教官　井口寧 助教授

審査委員主査　井口寧 助教授
審査委員　松澤照男 教授
審査委員　田中清史 助教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

310125 渡辺 浩二

提出年月: 2005 年 2 月

概 要

従来のグリッドの分散ストレージシステムは，転送速度などの性能に特化しており，システムの信頼性や，ディスク利用効率についての議論はされていない．本研究では，一般のユーザー環境を使用して分散ストレージを構築する際に，個々のコンピュータの稼働率を測定する．この数値を元に冗長データの多重度と，データの分散先を動的に変化させる．冗長データにはリードソロモン符号を使用した多重パリティを使い，レプリカ方式よりも総データ量の削減を図る．この２つの手法によって，信頼性がばらばらな環境においてもシステムの信頼性の確保と，ディスクスペースの有効利用が可能な分散ファイルシステムについて提案を行う．

目 次

図 目 次

表 目 次

第1章　はじめに

1.1　研究の背景·目的

近年，高エネルギー物理やヒトゲノム解析等の大規模データ解析を必要とする分野では grid 技術がキーテクロノジーとなっている．grid コンピューティングでは多数のノードが接続されるため障害対策や効率的な資源の割り当てが必要不可欠である．また，grid 環境は，さまざまなコンピュータから構成されているために，それぞれのコンピュータの信頼度は均一でないために，システム全体の信頼度を詳細に計算する必要がある．

grid 上では大量のデータが扱われるために，広域分散ファイルシステムを構築する手法が注目されている．データストレージ分野での先行研究 [3][5] では，データの保存，転送能力に主眼がおかれ，データの信頼性確保のための技術はレプリカ管理のみとなっている．レプリカ管理は冗長性の確保，負荷分散には有利になるが，ディスク容量の利用効率の面では不利になるといえる．また，Data reservoir[8] は新しい研究として注目を浴びているが，システムの信頼についての議論が十分にされていない．

本研究では grid 上にデータの分散配置をするだけでなく，障害対策を盛り込んだ広域分散ファイルシステムについて研究を行う．分散配置の手法は，常に参加ノードの状態を監視しておき分散配置に使用するノードを決定しファイルを分割，分散配置する．ここでは，各参加ノードの MTBF を測定し，その結果より使用するノードを選択する．あるノードが故障，容量不足等に陥った際には, 使用ノードの再決定を行うことで動的に配布先を決定可能にする．システムに冗長性を持たせるために，分散したデータ間のパリティを保存する．しかし単純な排他的論理和を使用したパリティでは，システム中の 1 台の故障しか対応できない．そこでシステム信頼度の監視結果より，要求される信頼度を満たすために必要な冗長データの多重度を算出し，故障に十分対応できる多重パリティを生成し異なるノードに配布する．この多重パリティはリードソロモン符号を使用して生成されるデータである．上記の手法を組み合わせることで様々な状況に対応できる分散ファイルシステムを構築することを目的とする．また，動的な分散配置を行い，データの配布先を変化させることで容量に余裕のあるディスクだけにメンバを変更できる．よって容量の異なるディスクでも効率良く分散配置させることができる．

以上の手法を用いて利用効率と障害対策を施した分散ファイルシステムについての構築，評価を行う．

1.2 構成

本論文は全6章からなる．第2章では，gridコンピューティングに関する話題と，先行研究について述べ，本研究との相違点について考察を行う．第3章では，本システムの構成と実装，grid環境へ展開した際のシステムの振る舞い，そして2つの動作モードについて述べる．第4章では，システムの信頼性について各ホストのMTBFの算出方法や，システム全体に要求される信頼性をどのようにして確保するかを議論する．第5章では，システムの2つの動作モードでの性能評価，障害発生時の自動再構成についての性能を評価する．最後に第6章では，本論文の結論を述べる．

第2章　Grid

2.1　はじめに

本章では grid コンピューティングの概要について述べ，データストレージに関する既存研究についての紹介と問題点を指摘し，本研究との相違点を挙げる．

2.2　Grid とは

Grid の語源は英語の”Power Grid(電力網)”が由来である．我々の日常生活で電気を使用する際に，コンセントにプラグを刺すだけで使用できる．このとき，使用される電力は，どこの発電所で発電されたか，どの経路を通って給電されたものかを意識することが無いのと同様に，Grid を使用するときに，コンピュータのロケーションやネットワーク構成を気にすることなしに利用できるということを表している．

Grid 技術が生まれた当初はスーパーコンピュータの利用技術から発展してきたこともあり，複数のコンピュータの CPU を並列にしようして計算能力の向上を図る技術として注目されていた．しかし，近年の Grid では計算能力のみならず，データ，実験装置，センサー，人間などの資源を仮想化·統合して，仮想的な計算機や仮想組織を動的に形成するようになっている．

Grid において狭義の Grid は以下の 3 つに分類される．

- ビジネス grid … 高信頼ウェブサービスのための Grid
- データ grid … 超大規模データ処理のための grid
- コンピューティング grid … 高速計算サービスのための Grid

また，構成単位から考えた場合，以下の 3 つに分類される．

- クラスタ grid
  1 つの LAN に所属するマシンで構成するシンプルな Grid．部門ごと，プロジェクトごとなどの小規模な構成になることが多い．リソース管理やユーザ認証などの仕組みが比較的単純ですむため，ネットワークの品質も確保しやすい．そのため，高スループット，ハイコストパフォーマンスの要求に対応できる．

- キャンパス grid
  複数のクラスタ grid を束ねて稼動させ，リソース共有を実現する構成．企業や大学といった組織ごとに構成されることが多い. 単一のクラスタ grid に対し，柔軟性および拡張性に優れる．

- グローバル grid
  キャンパス grid の集合体．各キャンパス grid はインターネットなどの外部ネットワークを介して相互接続される．キャンパス grid 間で合意したリソース管理やプロトコルに従って，世界中に分散された今日リソースを利用可能にする構成である．

現在の Grid 利用形態として主なものは，計算 grid とデータ grid がある．前者は複数の計算サーバをネットワークに接続し，仮想的に巨大な計算機を実現するものである．後者は大規模なデータや広域に分散しているデータを grid 技術を用いてアクセスするものである．

2.3 Grid におけるデータストレージ関連研究

2.3.1 Grid Datafarm

Grid Datafarm[3] は産業技術総合研究所が中心となって研究開発を進めているシステムである．Grid Datafarm は世界的に分散した資源への高速，安全，効率的で信頼性のあるアクセスを提供することを目的としている．大容量実験ファイルが write-once, read-many だという特性を考慮して，ファイルの複製生成が負荷分散，バンド幅，耐故障性に確保することに成功している．

最適なファイル複製，計算ノードの選択などのスケジューリングは，効率的なジョブ実行に必要不可欠である．Grid Datafarm はデータの局所性を考慮したスケーラブルな IO バンド幅とファイルの複製生成手法 [4] に特徴がある．従来の計算ノードと独立したネットワークを経由した IO アクセスでは非効率な実行形態となることが知られている．そこで計算ノードとデータが密接に関連付けられた owner-computes, move-the-computation-to-data 手法を適用し，データインテンシブアプリケーションのデータアクセスバンド幅の向上を提供する．プロセススケジューリングの仕組みは，クラスタノードに分割·格納されたデータに対して owner-computes ルールになるようにプロセスをスケジューリングして並列実行させる．こうすることで巨大なファイルのネットワーク越しアクセスを減らし，実行プログラムをファイルを格納しているノードへスケジューリングするだけになり，全体としてみたときにファイルアクセスバンド幅が大きく勘定できることになる．

GridDatafarm では，負荷分散，冗長性のためにデータの複製を行っている．Grid Datafram では，拡張ストライピングクラスタファイルシステムを提供しデータ grid アプリケーションに必要なデータを断片化してメタデータにより管理する．Grid Datafarm では，付加分散のために複製したデータで冗長性を確保しており，レスポンスとスループットに優れたファイルシステムとなっている．

2.3.2 Europian Data Grid Project

Europian Data Grid Project(以下 EDG)[5] は EU が 1000 万ユーロの助成を与え，2001 年に開始されたプロジェクトで，CERN(欧州合同素粒子原子核研究機構) によって研究が進められている．2004 年 4 月をもって EDG は The Enabling Grids for e-science in Europe(EGEE) へとプロジェクトが引き継がれた．

EDG は主に以下のような機能を有しており，CERN の高エネルギー物理実験で生成される大容量データを処理すべく設計されている．

- 証明書に基づくシングルサインオン
- LDAP をベースとしたユーザー名マッピング
- レプリカカタログ
- リソース情報サービス
- スケジューリング管理
- データマネージメント

データマネージメントでは，ファイルの複製生成サービス，データアクセスの最適化，キャッシュ技術，ファイルの移動といった機能に重点が置かれている．特に要求される詳細機能は

- ファイルの単一名前空間の管理
- セキュアで効率の良いデータ転送
- リモートコピーの同期
- 広域データアクセス，キャッシュ機構
- メタデータの管理
- マスストレージシステムへのインターフェイス

これらの要求される機能を実現するために EDG では

1. GDMP(Grid Data Mirroring Package)
   プロジェクトの 1 年目に開発された．複数のサイト間でのファイルのレプリケーションができる．
2. edg-replica-manager
   プロジェクトの 2 年目に開発された．GDMP にいくつかのレプリケーション機能が追加された．

を実装した．どちらも EDG のソフトウェアのリリース版に含まれており，複数のテストベッドに配置されている．

2.3.3 GSI-SFS: A Grid File System

grid ファイルシステムは grid 上に展開した分散ファイルシステムで，ユーザの利便性とセキュリティを両立させたものである．基盤となる技術は GlobusToolkit に含まれる GSI(Grid Securiity Infrastructure), SFS(self-certifying file System) である．GSI はホスト間の相互認証と通信のセキュリティを提供する．SFS は NFS の転送を暗号化し，認証データを負荷して転送するシステムである．SFS のベースは NFS であるために，ユーザーはファイルの物理的な位置を気にすることなしにファイルにアクセスが可能である．

性能は，通信を暗号化するためにオーバーヘッドが大きく，通信路に 1000Mbps の LAN を用いた場合には暗号化なしの場合との差が大きいことがわかった．しかし通信帯域を 100Mbps の LAN にしたところ，ネットワーク性能がボトルネックとなり，暗号化をする，しないにかかわらずほぼ同じ性能が得られることがわかった．

2.3.4 Data reservoir

Data Reservoir[8] は東京大学と (株) 富士通が開発を行っている，巨大データの共有システムである．このシステムの基本アーキテクチャは近距離と長距離の通信をわける方式をとっている．システムはファイルサーバと複数のディスクサーバで構成される．特徴として，データアクセス時に iSCSI のプロトコルを使用し，複数ストリームによる並列転送をする点が挙げられる．転送時には 2 段階階層的データストライピングを行う．この方式は，並列ストリームを確立する段階でストライピングを行い，受け取り側のファイルサーバがディスクサーバーに書き込む際に均等分散処理を行い，記録している．SC2003 のバンド幅チャレンジにおいて，日米太平洋間を 8.2Gbps で通信することができた．

2.3.5 Grid におけるデータストレージの課題

一般的に Grid におけるデータストレージの要求案件には次のようなものがある．

- 大容量のファイルの保存
- 単一名前空間
- スケーラブルな並列 IO バンド幅
- 安全な認証
- 耐故障性
- 動的再配置，データ復元，再計算

このような要求に対して，先の関連研究では非常に高い性能とスケーラビリティを確保しているといえる．しかしながら，システムの信頼性に関しての細かい議論がされていない．これは，システム構成に使われるコンピュータの性質によるものであると考えられる．研究機関などで使用される大型計算機の場合，動作が非常に安定している上に常時電源を投入しておくという使い方が普通である．これらの計算機で grid を構築した場合には，安定した信頼性を確保できるはずであり，いくつかのデータのレプリカを持っていればそれで十分であると考えられる．これに対し，より小規模でユーザ自身が管理権限を持つコンピュータを grid の構成ノードとして利用する場合には，ユーザが自由に電源を切ることもでき，また，搭載ディスクが冗長構成になっていない等の理由で各ホストの信頼度が不均一であるといえる．また，レプリカによる構成では，最適なレプリカを選択することで負荷分散には役立つものの，レプリカ 1 つにつき元ファイルと同じだけのファイルサイズが必要となるために，ディスク容量の利用効率が悪いといえる．

これらの理由から，grid 構成ノードの各々の信頼性についての議論と，データの冗長度に対する総データ量についての議論が必要である．

2.4 まとめ

本章では，Grid の概要について述べ，既存の grid におけるストレージ関連研究について調査をした．関連研究では，データの安全性については考慮してあるものの，実際にシステム全体の信頼度がどの程度なのかについては考えていないことがわかった．また，Grid で使用されることの多いレプリカの生成については，コピーするだけで冗長データの作成はできるが，ディスクの利用効率の面では不利になってしまっていて，この点についても議論する必要がある．

第3章　システムの構成

3.1　はじめに

本章では，提案するシステム構成について述べる．提案するシステムでは，算出されたシステム信頼度に応じて冗長データとなる多重パリティの冗長度を変化させる．この多重パリティはリードソロモン符号によって生成されるものを使用する．この多重パリティを生成するマスタサーバがあり，マスタサーバは，実際にデータを保存するストレージノードへと GridFTP を使用してデータを転送する．このとき資源管理サーバからの情報と，各ストレージノードの信頼度に基づき配布先ノードを決定する．これらの設計と実装の詳細を以下の節で述べる．

3.2　システムの構成と設計

システムの構成図を図 3.1 に示す.

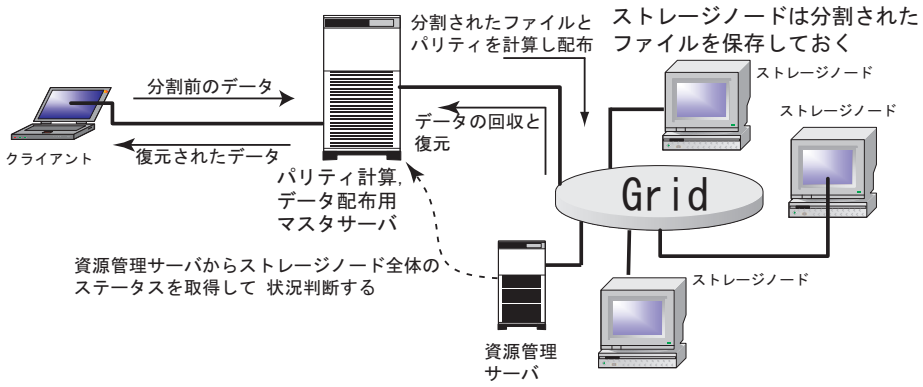

図 3.1: システム構成図 (各サービスが独立している状態)

Grid 構築のミドルウェアとして GlobusToolkit(以下 globus) を使用している．globus のサービスは

- 資源管理
- 情報サービス

- データ転送

という 3 つの柱で構成される．またこれらのサービスを支えるセキュリティ機構として GSI(Grid Security Infrastructure) がある．

GSI

GSI はオープンなネットワーク上でセキュアな通信を行うためのサービスである．セキュリティを確保するために，公開鍵暗号方式，X.509 に基づく証明書，SSL などの技術を利用している．

GRAM

GRAM(Globus Resource Allocation Manager) は資源管理をおこなうサービスである．GRAM は各マシンへの標準的なインターフェイスを提供し，GRAM の API を使い，どのようなマシンへも 1 つのインターフェイスでリクエストを送り実行することができる．GRAM では図 3.2 のようにしてジョブを実行する．まずリクエストがジョブを実行され

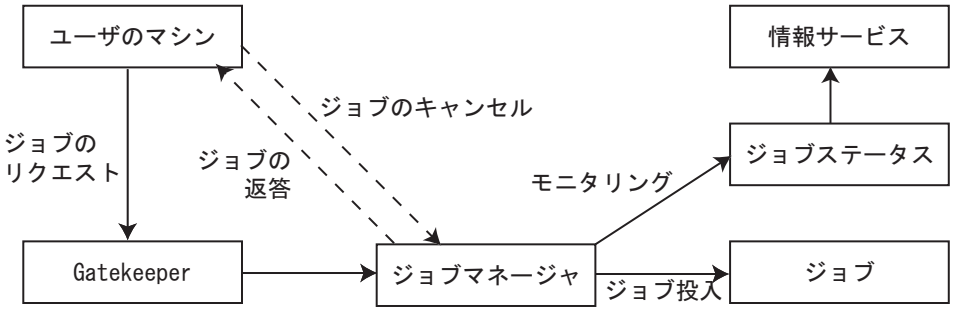

図 3.2: ジョブの実行

るホストの Gatekeeper に渡り，そのホストでのローカルユーザ名にマッピングされ，そのローカルユーザの権限でジョブマネージャを起動する．その後ジョブマネージャがジョブを生成し，実行する．

MDS

MDS(Globus Metacomputing Directory Service) は，grid におけるマシン情報を提供する．MDS は LDAP ベースのサービスである．MDS は以下の 2 つのコンポーネントを持っている．

- GRIS (Grid Resource Information Service)
  各マシン上で現在のマシン状況を監視し，情報を提供する．

- GIIS (Grid Index Information Service)
  GRIS からの情報をまとめることが可能.

また，GRIS や GIIS は階層的に配置することが可能で，一括して GIIS に問い合わせをすることも可能である.

Grid FTP

GridFTP は広域，広帯域なネットワーク上でも高性能で，安全なデータ転送をするプロトコルである．GridFTP は，GSI によるセキュリティを確保，データの一部だけの転送，データの並列転送，サードパーティ転送などが可能になっている．また既存の TCP は 1970 年代に設計されており，近年の高速ネットワークに対応しきれていないことから GridFTP では TCP のスロースタートフェーズにおける転送レートの立ち上がりを早くし，複数の TCP コネクションを並列に確立できるなど改良されている.

3.2.1 マスタサーバ

マスタサーバでは，入力ファイルの分割とリードソロモン符号の生成，配布先ストレージノードの決定と Grid FTP を使用したファイルの転送を司るホストである.

リードソロモン符号を利用した多重パリティでは，要求される信頼度を満たすように冗長度を変化させる．このリードソロモン符号の生成については 3.2.5 で述べる.

マスタサーバの機能と実装は以下の通りである.

1. ストレージノードの生存確認
   各ノードが健全な状態であるかどうかの確認を行う必要がある．まず ping での生存確認を行う．ここで応答がなければ候補の配列から削除する．次に tcp のポートが開いているかを確認する．globus で必要な GRAM と GridFTP のポートはそれぞれ 2119 と 2811 である．これらの 3 つのテストすべてをパスしたホストが利用可能ということになる． 実装は,ping では Perl モジュールの Net::Ping::External を使用して Ping の応答を監視させている．また，TCP のポートチェックでは，IO::Socket モジュールを使用し，それぞれのホストについてポートの監視を行っている.

2. 資源情報問い合わせ
   マスタサーバは処理を始める前に各ノードのメモリの空き状況と，ファイルシステムの状況を確認しておく．この情報に基づき，生存確認の取れたノードの更なる絞込みを行う．問い合わせには Perl スクリプトの中で globus の grid-info-search を制御し，必要な情報を取り出している.

3. リードソロモン符号のエンコードとデコード
   リードソロモン符号のエンコードには gflib[9] という C 言語のライブラリを使用し，このプログラムを Perl から制御して実行する.

4. GridFTP によるファイル転送
   生存確認の取れたホストにエンコードしたファイル断片を転送する．また，逆のオペレーションも行う．

5. メタデータの記録，読み出し
   保存するファイル名（論理ファイル名）と実際に各ストレージノードに配布される名前（物理ファイル名）のマッピングを行う．物理ファイル名は 11 桁の 0 から 9 までの数字とアルファベットの大文字小文字を合わせたランダムな組み合わせで決定された文字列にファイル番号を付与したものとなる．
   メタデータのフォーマットは”論理ファイル名, 物理ファイル名, 配布先ストレージノードのリスト”となり，1 論理ファイルにつき 1 行で記述される．論理ファイル削除時には，メタデータのエントリも同時に削除される．

6. ジョブの終了確認
   リードソロモン符号の処理や GridFTP の転送ジョブが終了したかどうかの確認をする．確認が取れるまで次の処理に進まないようにしておく．

3.2.2 ストレージノード

ストレージノードの機能と実装は以下の通りである．

1. GridFTP ホスト
   マスタサーバからのファイル転送要求に応答し，指定したディレクトリに物理ファイルを格納する．GridFTP は Globus インストール時にサービスとして/etc/services に登録してある．

2. MDS の応答
   マスタサーバからの MDS 要求に応答し，マシンの各種情報を返す．MDS は ldap ベースで動いており，slapd が常時待機している．

3.2.3 資源管理サーバ

資源管理サーバは MDS の情報を集め，利用可能なファイルシステムの容量を報告する機能と，リードソロモン符号計算時に空き物理メモリ量を管理する．実装では，各ストレージノードにおいて，資源管理サーバのサービスを実行している．スケーラビリティの点では，GIIS に各ストレージノードの情報を登録し，一括問い合わせを行ったほうが有利になると考えられる．しかし，本システムでは，一元管理する GIIS を動かすのに十分な信頼度を有するノードが存在するか，しないか判断がつかないことから，各ホストで GRIS と GIIS を同時に動かすことにした．

3.2.4 データのストライプ配置

本システムでは，データを分割しストレージノードへ配布する．(ストライピング) このとき，分割したファイル間のパリティ(多重パリティの場合もある) を計算し，冗長性を確保する．ストライピングとパリティ生成の概念図を示す．

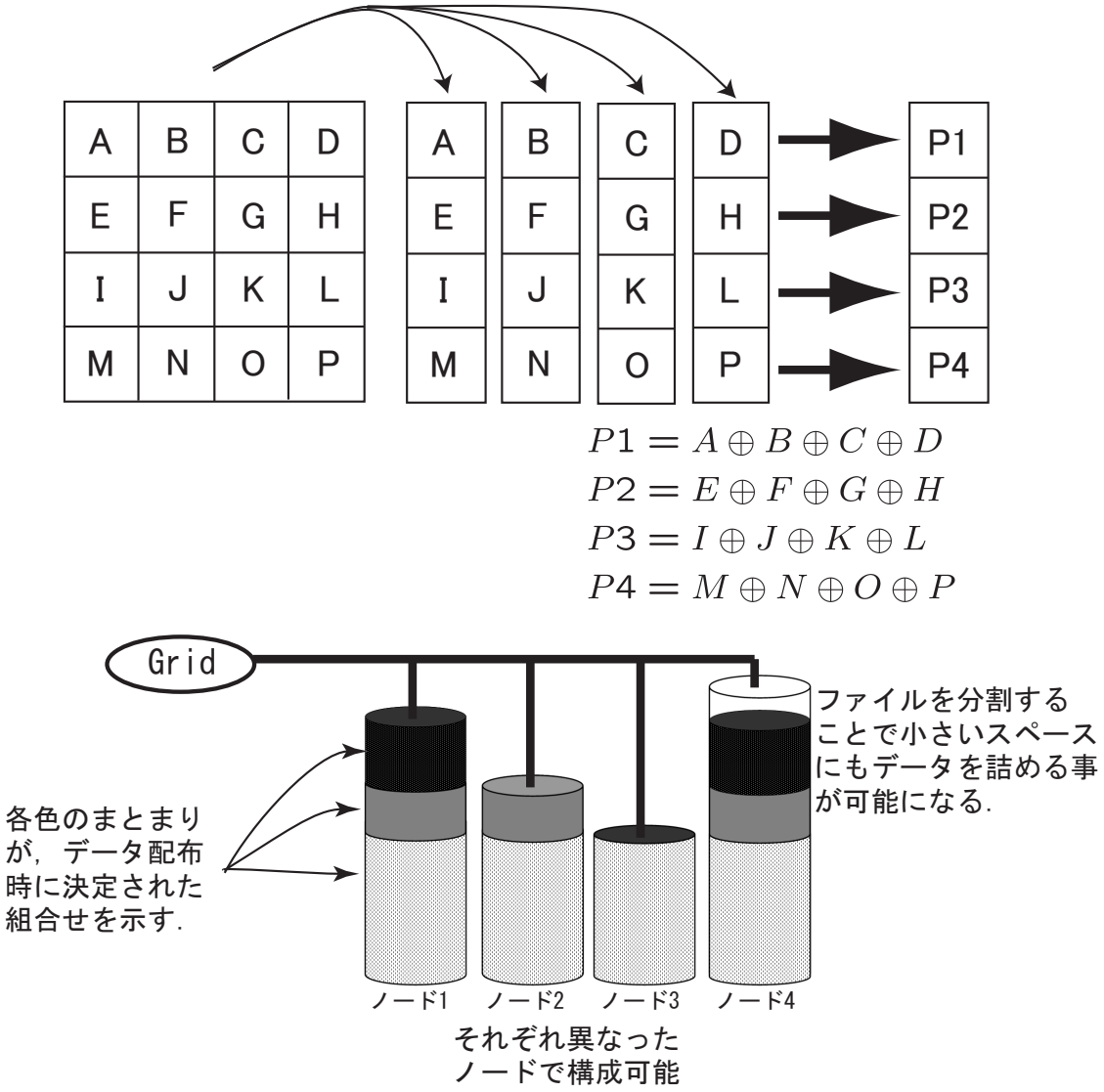

図 3.3: ストライピングと分散配置の概念図

3.2.5 既存の RAID と多重パリティ

RAID(Redundunt Arrays of Independent Disks) は複数のディスクを 1 つに結合した二次記憶システムである．多数のディスクの使用で，並列アクセスによる高性能と冗長デー

タによる信頼性を確保することができる．RAID にはレベル 0 から 5 までの 6 種類が存在し，近年，ハイエンドのディスクアレイサブシステムには冗長度を高めたレベル 6 を実装した製品もある．

RAID0

RAID0 はストライピングとも呼ばれ，データを単純に分割し，並列転送を行うことで，高速に処理を行うことが可能である．データの保護は一切行わないため，パリティの生成などにかかるオーバーヘッドが生じない．

RAID1

RAID1 は 2 台 1 組のディスクを使って常に同じ内容を保存する．データが複製されているため，どちらか片方が故障してもデータが失われることは無い．しかし，利用可能なディスクスペースは，総容量の半分となるため，コストが高い．

RAID2

ビットインターリーブによりデータを各ディスクに分散し，ハミング符号を用いた冗長情報を付け足す．HDD へのアクセスはブロック単位でアクセスを行うことが前提で設計されているため，ビット単位でストライピングするには余計な手間がかかることになる．当初 RAID2 は 1 ビット以上のエラーを見逃すことのできない高信頼システムへの適用が想定されていたが，近年は伝送経路でのエラーや，HDD 内でも高度な符号化と誤り検出，訂正が行われているため，あまり意味の無いものとなってしまった．単純なパリティと違い，コストが高くなるため，今まで製品化されたものはほとんど存在しない．

RAID3

ビットまたはバイト単位でのインターリーブでストライピングを行う．すべてのディスクの回転同期が必要なため，実装が難しいという難点がある．1 回のアクセスで全ディスクが使用されるため，シーケンシャルアクセス時には高速に処理できる反面，短くて小さなデータへのランダムアクセスが苦手となってしまっている．

RAID4

ブロック単位でのインターリーブを行う．データをストライピングして格納するディスクと，パリティを保存するディスクが 1 台必要となる．データの書き込み，更新時には必ずパリティの更新が必要となるために，パリティを保存しているディスクへのアクセスが集中するために，ディスクアレイのボトルネックとなってしまう．読み出しには高速な反

面，上記の理由により，書き込みはあまり性能が出ない．故障はディスク 1 台まで対応可能である．

RAID5

ブロック単位でのインターリーブを行う．RAID4 と同様に x-or での単純なパリティを使用した冗長データを作成する．RAID4 との違いはパリティを特定の専用ドライブに保存するのではなく，データを格納する複数のディスクに分散して格納する．こうすることで RAID4 でおきていたデータ書き込み時のボトルネックを解消することができる．

RAID6

RAID4 や 5 ではディスク故障の際，データを復旧できるのは 1 台までの故障に限られていた．しかし RAID6 では，2 台までの故障を復旧できるように設計されている．RAID6 では 2 台までの故障に対応するための方法は厳密には定義されておらず，製品によって主に 2 つの方式がある．

1. 2 次元パリティ方式
   2 組のストライプセットを使用して，2 次元的にパリティを作成し，2 台までの故障に対応.
2. リードソロモン方式
   リードソロモン符号を使用し，2 台までの故障に対応．単純な x-or のパリいティに比べて符号生成時にオーバーヘッドが大きい.

多重パリティ

本システムで使用する多重パリティとは，リードソロモン符号を使用し複数台のディスク障害にも対応する符号である．リードソロモン符号は巡回符号のひとつで，バースト誤りに強いことで知られている．リードソロモン符号はガロア体の元を基準にした多項式の加減乗除でエンコード，デコードされる．

3.3 grid 環境への適用を考慮したシステムの動作

3.3.1 動的分散配置方法

grid への動的分散配置では，globus で接続されたネットワークへとデータの分散配置を行う．まず分散配置をする際に，データの格納に使用するストレージノードの決定を行う必要がある．後述するシステムの信頼性計算に基づき配布先を絞り込み，ストレージノー

ドのディスク空き容量と生存を確認したうえで，最終的な使用ストレージノードを決定する．

globus を使用して構築された grid ならば，広域に分散することも可能なはずである．現段階では，ファイアウォールとの併用が難しいが，globus との併用が可能なプロキシが研究されているため，広域に分散する際にはその技術を流用するなどの対策が必要と考えられる．

3.3.2 データの書き込み

データの書き込み時のフローチャートを図 3.4 に示す．この書き込み時には，あらかじめ grid-proxy-init で証明書を作成しておくなどの手続きが終わっているものとする．

アルゴリズムを図 3.5 に示す．

grid 環境で使用する場合，データを格納するストレージノードの状態がわからないために，まず生存確認を行う必要がある．ストレージノードの候補リストを渡された”ノードの生存確認”プロシージャが，リスト内の各ノードに対して ping の応答，TCP のポート確認を行う．そこで，いずれかの試験に不合格だった場合には候補リストから除外する．

次に候補リストを”信頼性算出”プロシージャに渡し，要求した信頼性を確保できるよう，ホストの組み合わせを決定する．

信頼性算出時に，データデバイスの台数と，チェックサムデバイスの台数が決定されるためコマンドパスと，それぞれの台数を指定したシェルスクリプトを書き出し，実行する．元ファイルを分割したファイルブロックとエンコードされたパリティブロックは分割ジョブを実行したホストの一時作業ディレクトリに保存される．

このとき，ファイル名の生成は，

’11 桁のランダムな文字の組み合わせ’-4 桁のファイル番号.rs

という形式で作成される．これは実際にストレージノードへと保存されるときの名前 (物理ファイル名) である．たとえばランダムな文字列が’jXUoCXw0qbU’ であり，分割したファイル数が 10 個だった場合には一時作業ディレクトリに

jXUoCXw0qbU-0000.rs
jXUoCXw0qbU-0001.rs
⋮
jXUoCXw0qbU-0008.rs
jXUoCXw0qbU-0009.rs

という名前のファイルが作られる．

ファイルの準備の完了を待って，GridFTP によるファイルの転送を行う．1 ファイルにつき 1 つの転送ストリームを確立する．このとき fork() によってプロセスを分岐し，転送先ホスト分だけ並列に GridFTP のプロセスを起動する．fork されたすべての転送プロセ

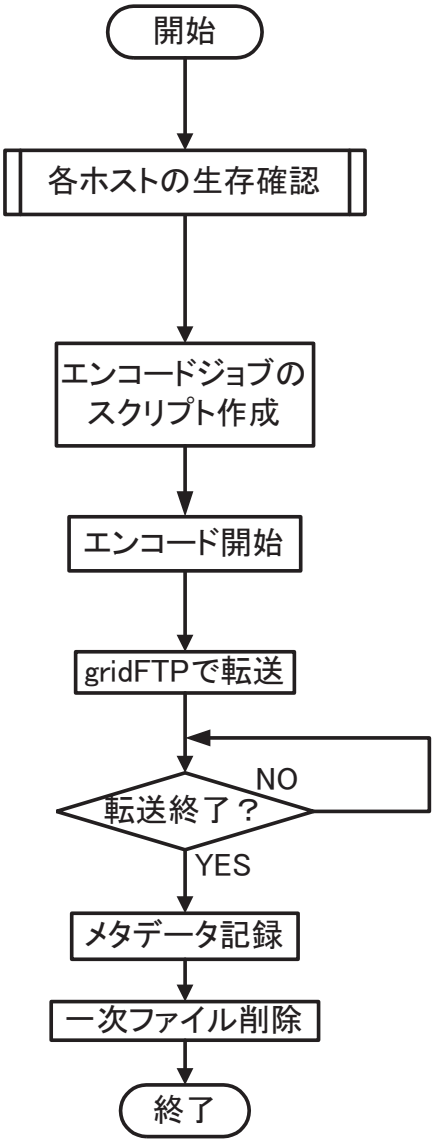

図 3.4: データ書き込み時の動作

```
1   代入：candidate[0]…candidate[total_node ];
2   minFS=100000000000; #まず最小空き容量をあり得ないほど大きい値に設定
3   for (counter = 0; counter < = total_node; counter ++){
4           host へ candidate から pop して代入 ;
5               if (host == NULL){
6                   ex it; #途中で足りなくなった。
7           }
8               if (host が ping に応答しない){
9                           ループの先頭に戻る;
10          }
11              if (host の 2119/tcp が開いていない){
12                          ループの先頭に戻る;
13          }
14              if (host の 2811/tcp が開いていない){
15                          ループの先頭に戻る;
16          }
17              if ((host の空きディスク容量) < minFS){
18                  m inFS = (host の空きディスク容量);
19          }
20          ok リストの配列へ push;
21  }
22  totalFSfree = minFS * (ok リストの配列の大きさ);
23  calc_reliability( );#  後述する信頼性計算へ
```

図 3.5: ホストの生存確認のアルゴリズム

スの終了を待って，つぎのメタデータの記録へ進む．メタデータでは，保存したい元の名前 (論理ファイル名) と，物理ファイル名のマッチング，そしてどのホストへ保存しているかを記述したファイルで，ユーザのホームディレクトリに保存する．

メタデータファイルの書式は

論理ファイル名，物理ファイル名のランダム文字列部分，信頼性クラス，ホストリスト

としてあり，各項目はカンマ区切りとしてある．3 列目の信頼性クラスについては後述する．

最後に一時作業ディレクトリに生成したファイルを削除し，データの書き込み処理を終了する．

3.3.3 データの読み出し

データの書き込み時のフローチャートを図 3.6 に示す．

データの読み出しは，プログラムの引数に論理ファイル名を与えて実行する．論理ファイル名を受け取ったプログラムはメタデータファイルを参照し，物理ファイル名と，格納しているホストを確認をする．

必要な物理ファイルを保存しているストレージノードに対し，GridFTP のファイルダウンロード要求を出す．ファイルの保存時と同様に fork し，並列にファイルを転送する．

このとき，ファイルの正当性や，ファイルが一部欠落していることも考えられる．この様な場合の対処は，まず GridFTP ですべてのファイルに対して要求を送る．ファイルが見つからなかったり，ホストに接続できない場合には，GridFTP がエラーをだし，一部分の断片が存在しないまま次の処理へ進む．ここで，リードソロモン符号のデコーダーに対し，デコードのコマンドを送る．デコーダは，指定されたファイル断片をチェックし，ファイルサイズが不正な場合などはデコーダが処理をせずに，ほかのファイルからリードソロモン符号を使って，再生成する．この動作のフローチャートを図 3.7 に示す．

3.3.4 データの再構築

本システムでは，冗長符号を用いてデータの保護を行っているが，リードソロモン符号のエンコード時に設定した以上のデータの消失を復元できるわけではない．そこで，ファイル断片を配布したストレージノードの状況を監視しする仕組みが必要である．

アルゴリズムを図 3.9 に示す．

本システムに保存されているデータの状態を確認するには，まずメタデータファイルを参照し，論理ファイルを 1 つずつ処理する．論理ファイル名より，物理ファイル名と配布先ストレージノードを確認し，プログラム内で記憶する．物理ファイルが各ストレージノードに存在しているかを判断する際，以下のいずれかの場合に該当すれば，配布したファイルの一部が読み取り不可能と判断し，ファイルの復旧に当たる．

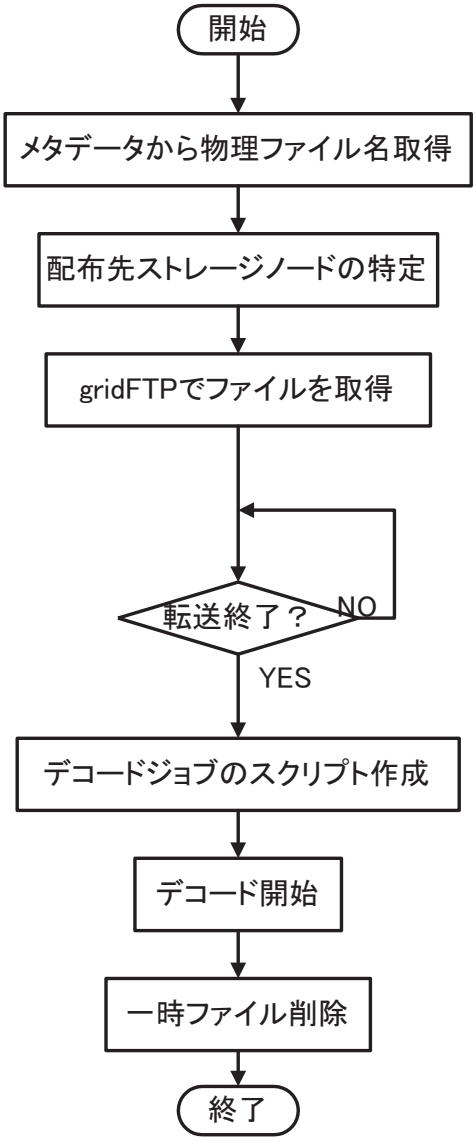

図 3.6: データ読み出し時の動作

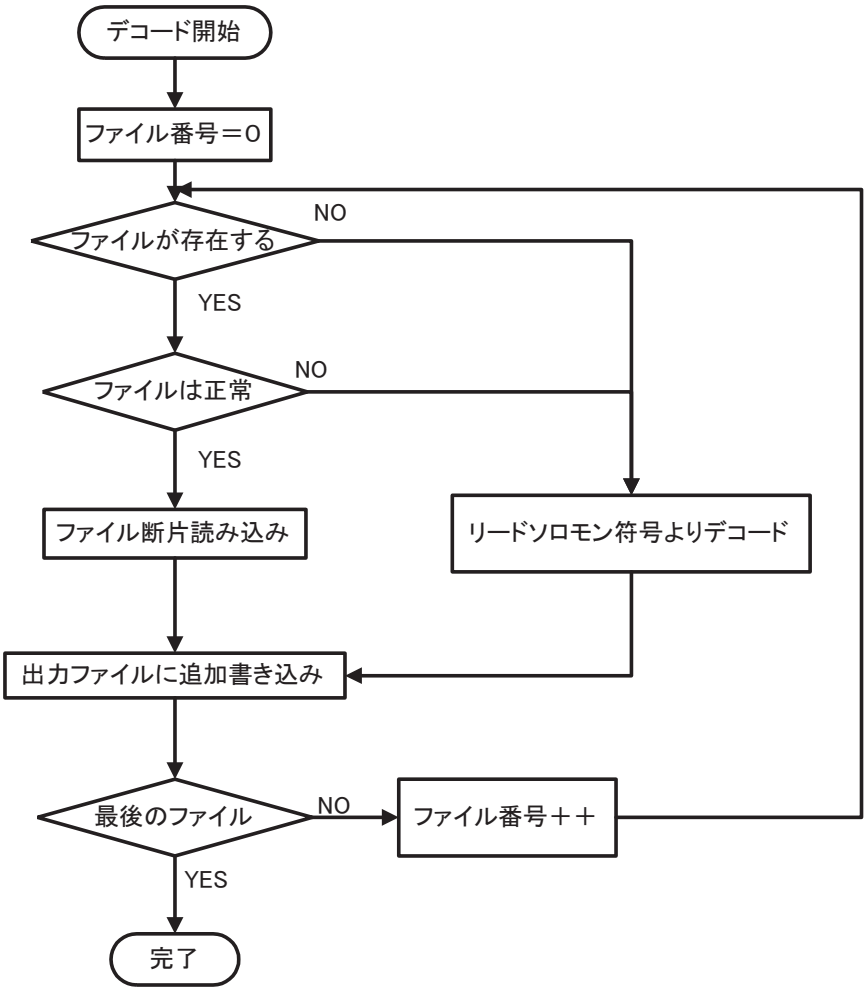

図 3.7: リードソロモン符号を使用した復号時の動作

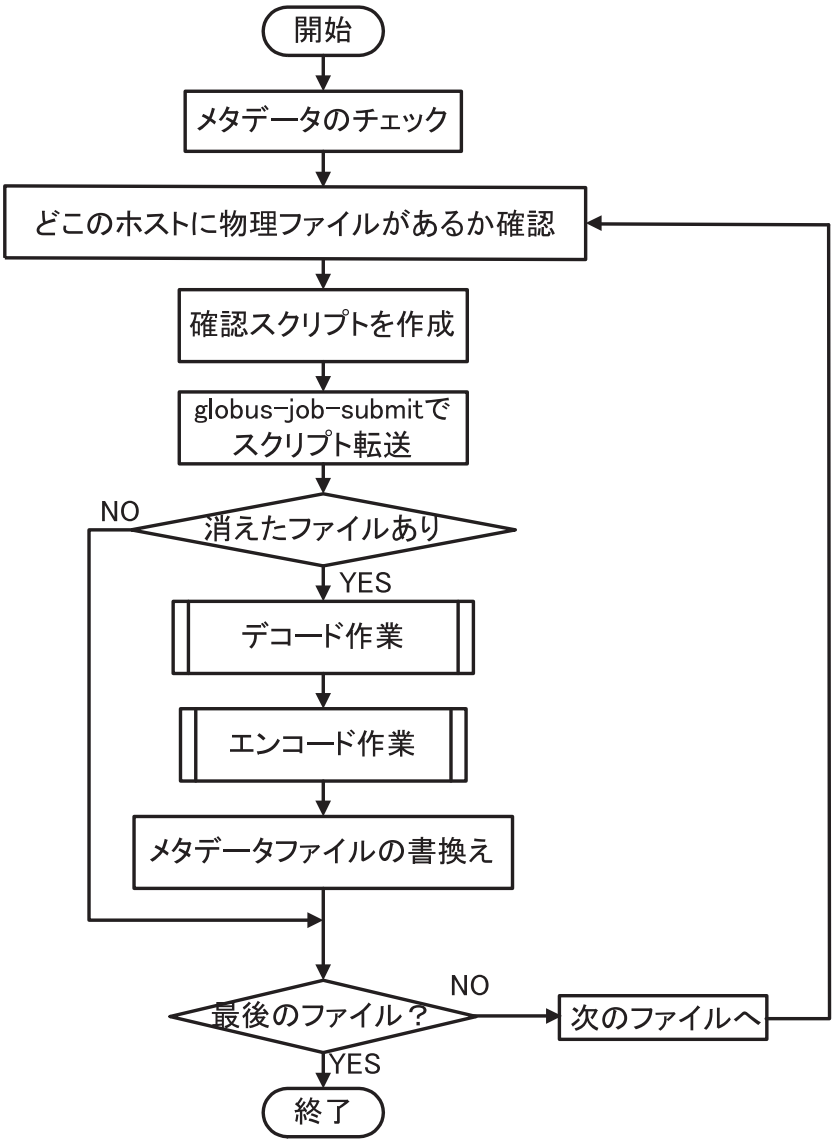

図 3.8: 保存されているファイルの健全性検証の動作

```
1   代入：file[0]…file[n]; #ファイルリストから論理ファイル名をすべて格納
2
3   foreach (file){
4           error=0;
5           phisy_name = (物理ファイル名);
6           class = (信頼性クラス);
7           hosts = (ホストリスト);
8           foreach (hosts) {
9                   if (  ホストの信頼性クラスが違う  = ture){
10                          error    =    -1;
11                  }
12                  if ( (ping に応答する) && (2119/ tcp  が開いている) && (2811/ tcp  が開いている) = false ) {
13                          error    =    -1;
14                  }
15                  if (物理ファイルが存在する  = false){
16                           error    =    -1;
17                  }
18          }
19          if (error != 0){
20                  decode ( );
21                  encode ( );
22          }
23  }
```

図 3.9: ファイルチェック時のアルゴリズム

- ストレージノードが ping に応答しない
- globus のサービスが開始されていない
- シェルで ‘-e ファイル名 ‘を実行してファイルが存在しないと返ってきたとき

ファイルの復旧を行うときは，一度デコード作業を行い，もう一度ファイルの分割処理やリードソロモン符号のエンコード処理などすべて行う．このとき，消失したファイルだけを復活させることは可能であるが，システム全体の信頼度を再度測定を行い信頼向上を図るように実装をした．

3.4 負荷分散ための動作モード

初期の実装では，保存するファイルをマスタサーバに送り，分散配置などの処理をすべてマスタサーバに任せるようにしていた．しかし，この方式では負荷集中が予想されるため，各クライアントがマスタサーバの機能を実行することにした．

図 3.1 における，マスタサーバは高性能なコンピュータを想定した．しかし，クライアントのマシンは搭載メモリも少なく非力なことが多いために，2 つの動作モードを自動的に選択することにした．入力するファイルサイズが空き物理メモリより大きい場合には，メモリのスワッピングがおきるために処理速度が極端に低下する．そこでファイルサイズに応じて処理モードを切り替えている．この動作モード選択を使用したフローチャートを図 3.10 に示す．今回は実装上の都合により，処理ホストの空きメモリ量が処理時間に与える影響が最も大きいために空きメモリ量を基準にしてモードの選択を行うようにした．しかし，使用メモリを抑えた実装に変更したときには，処理モード選択判断アルゴリズムを CPU 利用率などを基準にしたものに変更する方がよいといえる．

3.4.1 自己処理モード

自己処理モードでは，空きメモリサイズが十分にあり，スワップが起きないことが予想される場合に使用する．自己処理モードでは基本的に，クライアントマシンのローカル HDD に保存してあるデータを分割し，多重パリティを生成し，転送を行う．

3.4.2 外部処理モード

外部処理モードでは，ファイルサイズに対して空きメモリが小さい場合に，grid 内を検索し，空きメモリに余裕のあるホストに処理を任せる．このときに使用するアルゴリズムを図 3.11 に示す．

保存するファイルの分割，多重パリティの生成の終了を待って，GridFTP のサードパーティ転送で分散配置を行う．外部処理モードの概念図を図 3.12 に示す．

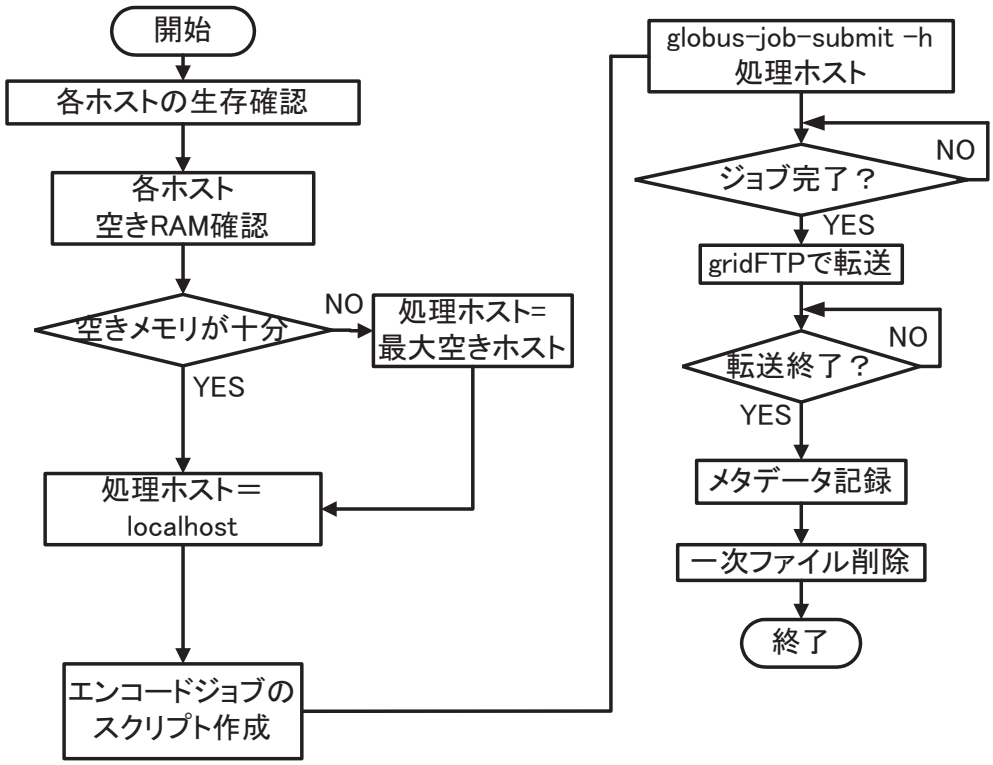

図 3.10: 外部処理モードの選択を可能にした場合のフローチャート

```
1   代入：candidate[0]…candidate[n];
2   localRAMfree = lcalhost の空きメモリ;
3   maxRAMhost = 0;
4   rs_host = localhost;
5   if(localRAMfree < filesize){
6           for(i=0;i<=n;n++){
7                   remoteRAMfree = candidate[i]の空きメモリ;
8                   if(remoteRAMfree > maxRAMfree){
9                           maxRAMfree = remoteRAMfree;
10                          rs_host = candidate[i];
11                  }
12          }
13  }
14  #ここで rs_host に入っているホストが処理担当ノード
```

図 3.11: 処理ホストを決定する時のアルゴリズム

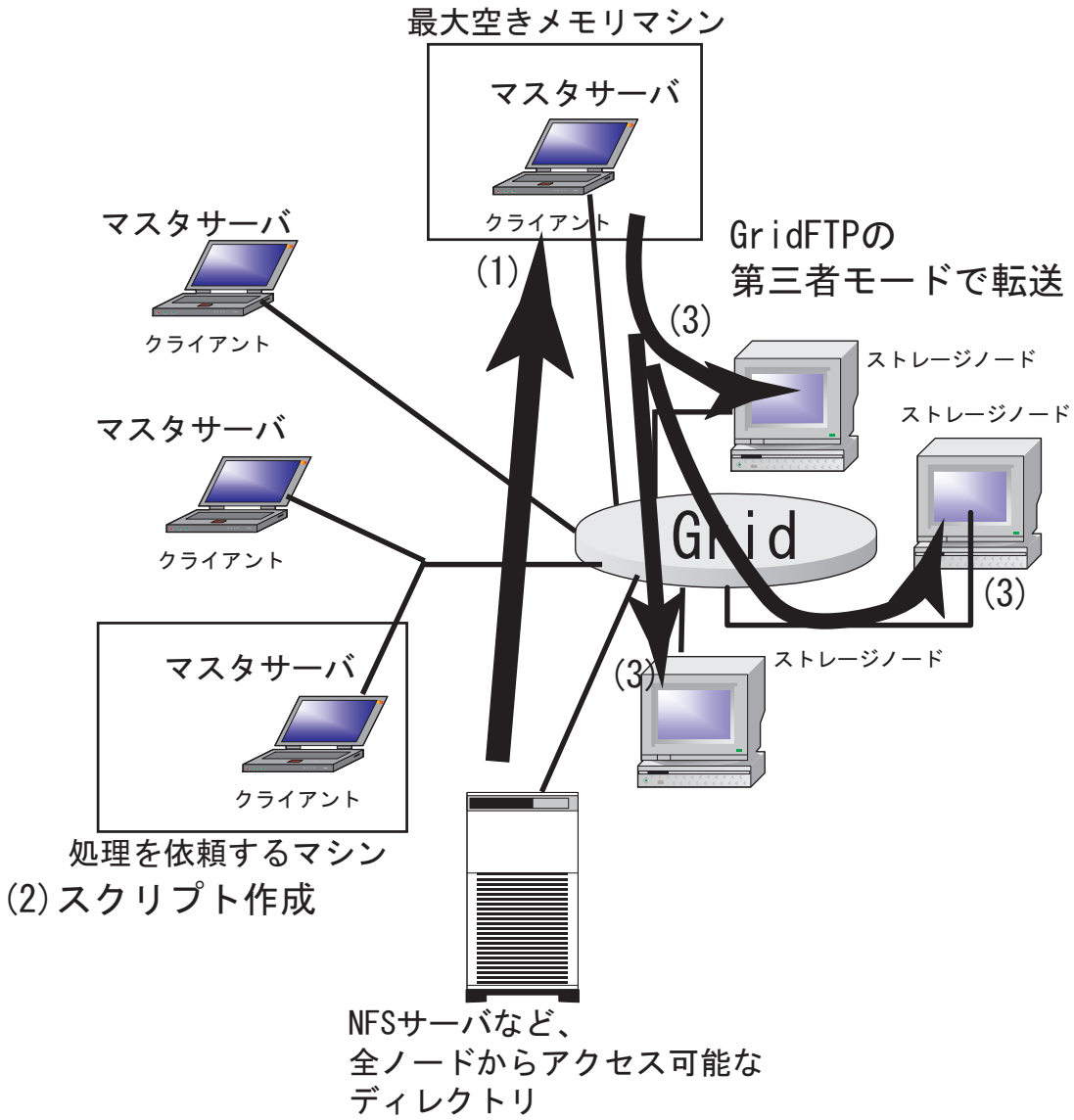

図 3.12: 外部処理モードの概念図

処理を依頼するホストがメモリ不足のために外部処理モードに入ったとする．すると，

(1) ネットワーク上のサーバから，最大空きメモリマシンへと入力ファイルが読み込まれる．このとき，処理ホストの候補に入っている全マシンから透過的に見えるパスにファイルを置く必要がある．これより，ファイルサーバは nfs でマウントしておくことにする．

(2) そのホスト上でファイルの分割と多重パリティの生成ジョブが実行される．

(3) ジョブの完了を待ち,GridFTP でストレージノードへとデータを転送する．

このとき，サードパーティ転送を利用するために，処理を依頼したマシンにはデータが流れずに，処理をしたマシンから直接ストレージノードへとデータが流れるようになっている．

3.4.3 GridFTP を使用する外部処理モード

先の外部処理モードの実装では，nfs を経由して元ファイルにアクセスした後に多重パリティを生成していた．nfs を使うことで，どのコンピュータからも簡単に透過的なアクセスが可能になるが，nfs の性能がボトルネックとなりファイル転送に時間がかかるようになってしまう．そこで，新たに元ファイルの読み込みに GridFTP を使用する外部処理モードを用意する．パリティの処理ホストが決定したら，元ファイルを保存しているホストから，処理ホストの一時ディレクトリへと GridFTP を使用して転送する．処理ホストは転送が終わり次第，一時ディレクトリに保存してある元ファイルのコピーを使用して，ファイルの分割とリードソロモン符号による多重パリティの生成を行う．このようにして出来上がったファイルのブロックをストレージノードへと転送を行う．nfs を使用せずに GridFTP で転送を行うことで性能低下を防ぐことができるはずである．更に，グリッド環境における標準的なファイル転送方式である GridFTP に統一することでシステムの汎用性も高まるといえる．２つの外部処理モードの性能差については，後に性能測定を行って検証する．

3.5 まとめ

本章では，システムの構成について記述をした．まずシステム構築に使用したミドルウェアの Globus について述べた．Globus には，資源管理，情報サービス，データ転送という 3 つの柱から成り，これらのサービスをセキュアに行う基盤として GSI というサービスが存在している．

次に，本システムで使用される各種ノードについての実装の詳細を述べた．ここでは，マスタサーバ，ストレージノード，資源管理サーバについて解説をした．基本的な実装を

元に，実際のデータ読み書きに関する動作についての説明を行い，負荷分散を考慮した動作方法についての詳細を述べた．

第4章　提案システムの信頼性

4.1　はじめに

本章では，システムの信頼性について評価し，要求する信頼性の確保の方法について述べる．

grid 構成ノードの稼働率を算出し，ある一定の値でクラス分けを行い，使用ノードと設定冗長度を変化させてシステムの信頼性を確保する方法と，データの保守についての議論，評価について述べる．

4.2　信頼性モデル

信頼性モデルとして，m-out-of-n 方式の並列システムを使用して信頼度を算出する．m-out-n 方式とは，n 台のシステムのうち m 台が正常に機能していればシステム全体の機能が継続されるものである．

m-out-of-n システムの信頼性を評価する際に問題になるのが，組み合わせの数である．例えば，1000 台のサブシステム中 995 台の機器が正常に動作する確率を求める場合，

- 1000 台すべて動作している確率は ${}_{1000}C_{1000} = 1$ 通り
- 999 台動作している確率は ${}_{1000}C_{999} = 1000$ 通り
- 998 台動作している確率は ${}_{1000}C_{998} = 499500$ 通り
- 997 台動作している確率は ${}_{1000}C_{997} = 166167000$ 通り
- 996 台動作している確率は ${}_{1000}C_{996} = 41417124750$ 通り
- 995 台動作している確率は ${}_{1000}C_{995} =$ 約 8 兆 通り

となり，膨大な組み合わせを計算する必要があり，いわゆる計算の爆発が起きる．しかし，構成システムのすべての機器が同じ信頼度 a である場合には，先の問題は

$$R = a^{1000} +_{1000} C_{999}a^{999}(1-a) +_{1000} C_{998}a^{998}(1-a)^2 +_{1000} C_{997}a^{997}(1-a)^3 \\ +_{1000}C_{996}a^{996}(1-a)^4 +_{1000} C_{995}a^{995}(1-a)^5$$

と表すことができる．この式を一般的に表すと，

$$R = \sum_{i=0}^{n-m} ({}_nC_{n-i} \cdot a^{n-i} \cdot (1-a)^i) \tag{4.1}$$

となる．

4.3 配布先ストレージノードの決定戦略

4.3.1 信頼性計算に使用する指標

システムの信頼性を議論する際の指標に MTBF と MTTR がある．

MTBF(Mean Time Between failure) は平均故障間隔のことで，あるシステムや機械が故障するまでの時間の平均値である．MTBF は

$$MTBF(\text{時間/件}) = \frac{\text{総稼動時間}}{\text{総故障件数}} \tag{4.2}$$

と表すことができる．例を図 4.1 に示す．この例での MTBF は

図 4.1: 稼働時間の例

$$MTBF = \frac{(a+b+c) - (D+E)}{2}$$

となる．

MTTR(Mean Time To Repair) は平均修理時間のことで，修理にかかった時間を平均したものであり，

$$MTTR(\text{時間/件}) = \frac{\text{総修復時間}}{\text{総故障件数}} \tag{4.3}$$

と表される．図 4.1 の例で計算を行うと，

$$MTTR = \frac{(D+E)}{2}$$

という結果になる．

システムが利用可能な状態にある割合がアベイラビリティ (稼働率) である．稼働率と MTBF，MTTR の関係は 4.4 となる．

$$\text{稼働率} = \frac{MTBF}{MTBF + MTTR} \tag{4.4}$$

4.3.2 各ストレージノードのアベイラビリティ(稼働率)算出方法

前節の指標を使って，システム内のストレージノードの稼働率を測定する．ストレージノードの稼働率測定は，マスタサーバが定期的に行い，各ノードの状態を記録する．この測定結果よりMTBFとMTTRを算出し，稼働率を求める．

まず，grid全体の総稼動時間を測定する．これは，gridを稼動させ始めた時刻と現在時刻の差を計算して求める．次に各ホストへ

- ping
- GridFTPのポートが開いているか
- GRAMのポートが開いているか

のテストを行い，すべてのテストに合格した場合は，正常動作中とみなす．どれか1つでも不合格だった場合には，ストレージノードの状態を"down"とフラグをたて，不稼動時間を足す．この不稼働時間は測定間隔を足していくことにより算出する．あわせて，故障を検出した回数も記録しておく．これで，総稼働時間，総故障時間，総故障回数が測定できたことになる．以上の数値よりMTBF，MTTRを計算し，ストレージノードの稼働率を算出する．

次に各ノードの稼働率に合わせて，クラス分けを行う．これは前節でも述べた計算の爆発を防ぐためである．ここでのクラス分けは[11]に記述されている可用性の分類を使用する．分類は稼働率をRとすると次のように書ける．

1. $99.99\% \leq R$ かなり高いレベルの稼働率．1年間に1時間程度のダウンタイム．
2. $99.9\% \leq R < 99.99\%$ 一般的に高信頼システムといわれる部類．1年間に8.5時間程度のダウンタイム．
3. $99\% \leq R < 99.9\%$ 一般的な稼働率レベル．1年間に3,4日程度のダウンタイム
4. $R < 99\%$

稼働率が99%に満たないノードは，データの格納には使用しない．しかし，稼働率の監視は続行し，稼働率が条件を満たせば使用リストに復帰させる．信頼性測定のアルゴリズムのフローチャートを図4.2に示す．

また，各ホストの稼働率を測定するときのアルゴリズムを図4.3に示す．

4.3.3 システム全体に要求される信頼性の確保

この節では，前節で求めた各ストレージノードの稼働率を基に，システム全体の信頼性を確保する方法について述べる．

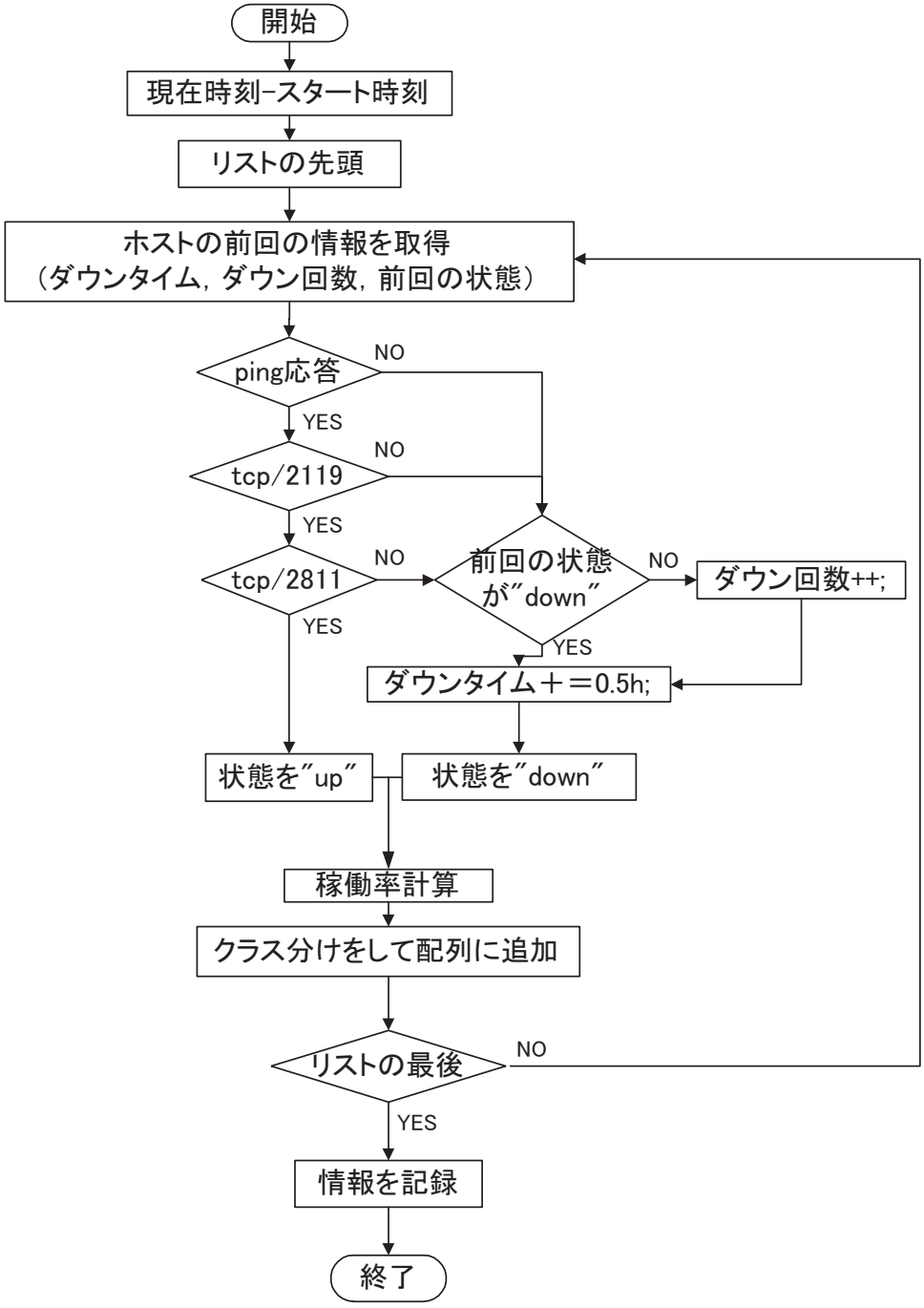

図 4.2: 稼働率チェックのフローチャート (30 分ごとに監視する場合)

```
1   total-uptime ＝ date – (グリッドの稼動開始時刻);
2   代入：host[0]…host[n];
3   for (i = 0; i <= n; i++){
4           downtime = host[i]の総ダウンタイム;
5           down-count = host[i]のダウンした回数;
6           pre-status = host[i]の前回チェックしたときの状態;
7           error = 0 ;
8           if ( (ping に応答する) && (2119/ tcp が開いている) && (2811/ tcp が開いている)  =  false ) {
9               error  =  -1 ;
10          }
11          if (pre-status == “up”){
12              down-count ++ ;
13              downtime += 0.5 ;
14          }else {
15              downtime += 0.5 ;
16          }
17          mtbf = (total-uptime – downtime) / down-count ;
18          mttr = downtime / down-count ;
19          availability = mtbf / (mtbf + mttr ) ;
20
21          if( 0.9999 <= availability){
22              host[i] を一番信頼性の高いクラスへ加える ；
23          }elseif (0.999 <= availability){
24              host[i] を二番目に信頼性の高いクラスへ加える ；
25          }elseif (0.99 <= availability){
26              host[i] を三番目の信頼性のクラスへ加える ；
27          }else {
28              host[i] を利用停止クラスへ加える ；
29          }
30  }
31  print  #クラス分け配列を書き出す ；
```

図 4.3: 各ストレージノード単体の稼働率を算出するアルゴリズム

システムは並列システムとみなして稼働率の算出を行っている．まず前章でのクラス分けされたストレージノードを，稼働率の高いクラスから使用する．このとき，使用ノードの偏りを減らすためにリストをシャッフルしてから使用する．まず開始時には予め設定したデータデバイスとパリティデバイス数で計算をする．ここでシステムの信頼性が足りない場合，同一クラスのストレージノードをもう 1 台パリティデバイスをして追加し，再度全体の信頼性の計算を行う．以後，要求する信頼性が確保できるまで同様の動作を繰り返す．同一クラスでストレージノードが足りなくなった場合には，次に稼働率の高いクラスに移動し，信頼性計算を再度実行する．1 回の信頼性計算は，リストからストレージノードを 1 つずつ取り出し，生存確認，ディスク空き容量をチェックし，問題が無ければ構成ノードのリストに追加する．計算のフローチャートを図 4.4 に示す．システム全体の稼働率の設定を 99.999% とした．

使用するストレージノードの稼働率からシステム全体の信頼性を求めるアルゴリズムを図 4.5 に示す．

4.4 まとめ

本章では，システムの信頼性確保について述べた．各ストレージノードの稼働率の算出方法は，マスタサーバが監視をして計算を行う．各ノードの稼働率が統一されていない場合のシステムの信頼性計算が複雑になりすぎるため，ノード稼働率によって単純な計算を行う手法について述べた．このクラス分けされたノードの稼働率を基にシステムの信頼性を算出し，多重パリティの冗長度を変化させ 99.999% を満たすように実装を行った．

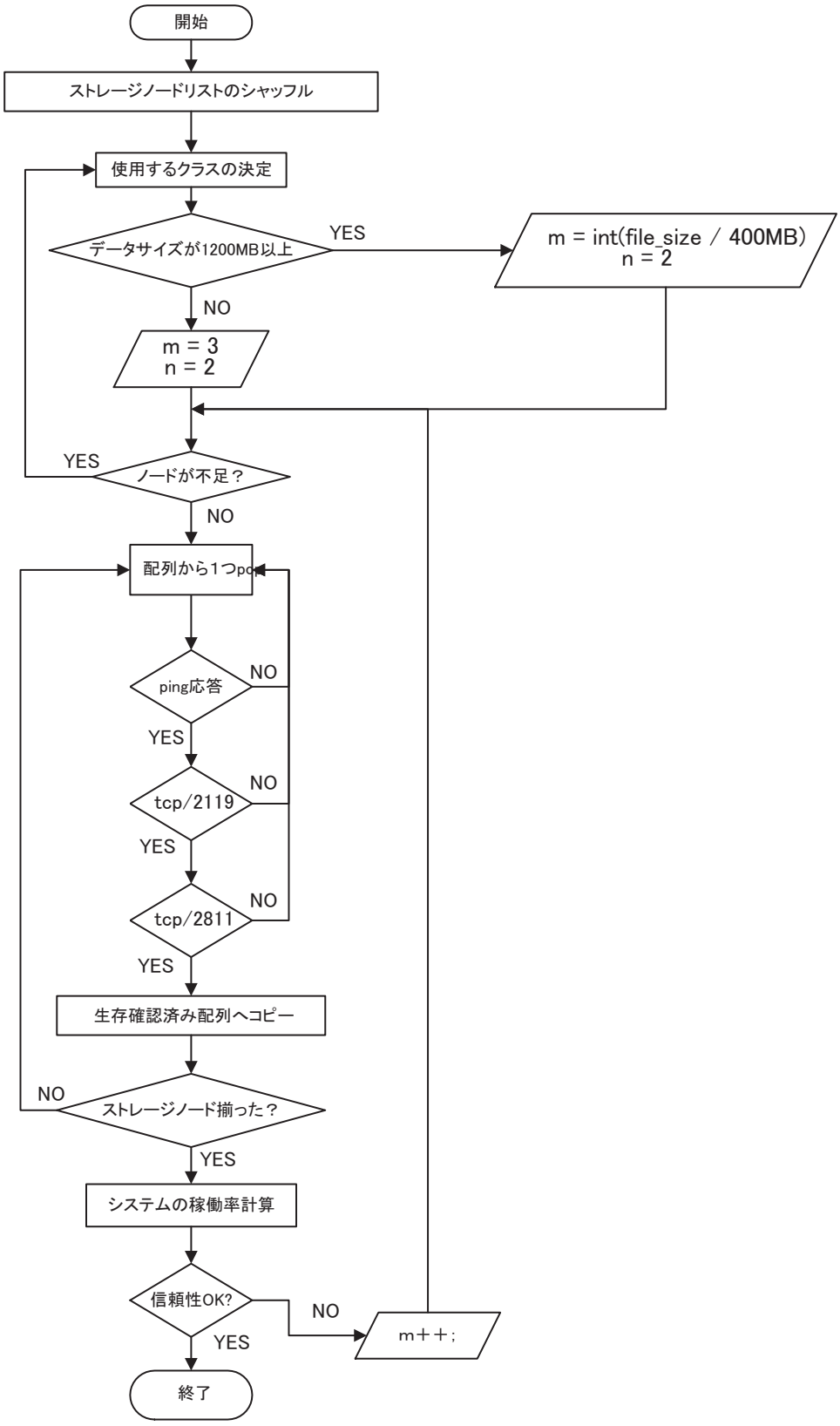

図 4.4: 信頼性決定のフローチャート

```
代入：highest_priority に一番信頼性の高いクラスを読み込む ；
代入：higher_priority に二番目に信頼性の高いクラスを読み込む ；
代入：normal_priority に一番低いクラスを読み込む ；
処理：それぞれの配列をシャッフル ；
class_name = highest_priority;
filesize = 書き込むファイルサイズ;
if(filesize > 1200MB){
        m = int(filesize / 400MB);
}else{
    m=3;
}
n = 2;
total_node = n + m;
if(total_node > (使用するクラスの配列の大きさ)){
        次のクラスへと変更して、再度実行;
}
for( i = 0; i < total_node; i++){
        host = 使用クラスからホストを一つ pop;
        if( (ping に応答する) && (2119/ tcp が開いている) && (2811/ tcp が開いている) = false ) {
                新たに pop して再実行;
        }
        living_host に host を push;
}
a = (使用クラスの代表稼働率);
for( i=0;i<=(n-m);i++){
    c = (total_node から (total_node – 1)を選ぶ組み合わせ数);
   total_reliability  +=  c* a^(total_node - i)*(1 - a)^ i;
}

if(total_reliability < required_reliability){
    n++;
        信頼性計算を再実行;
}
```

図 4.5: システム全体の信頼性を計算するアルゴリズム

第5章　提案システムの性能

5.1　はじめに

本章では，実装を行ったシステムの性能評価を行う．

まず，ローカル HDD のファイルを処理する自己処理モードでの評価を行う．読み書きの試験を行った．次に外部処理モードを使用し，ネットワーク上の NFS サーバから読み出したファイルを処理し，性能評価を行った．

5.2　実験システム

この節では，実験に用いたシステムのハードウェアおよびソフトウェアについて述べる．

5.2.1　ハードウェア構成

実験システムは，ワークステーション 11 台を使用し，ネットワークは学内 LAN をそのまま使用した．

- SunMicrosystems Blade1500 *10 台
  OS:Solaris8 CPU:UltraSPARCIIIe 1GHz RAM:512MB HDD:80GB LAN:GigabitEthernet
- AMD64 WS *1 台
  OS:SuSE Linux 9.0 Professional CPU:Opteron248*2 RAM:8GB HDD:160GB*2(RAID1)
  LAN:GigabitEthernet
- 学内 LAN (すべて GigabitEthernet)

各種ワークステーションの IO 性能の測定を行った．測定は Bonnie[10] というベンチマークソフトで行った．Bonnie は getc() や putc() などの基本的な命令のみで構成されていて，指定したサイズ (初期設定は 100MB) のファイルを利用し，キャラクタデータとブロックデータの読み書き速度を比較するものである．各機種に搭載されているディスクの性能を表 5.1 に示す．

表 5.1: 使用した HDD の性能

	Blade1500 (RAID なし)	自作 (RAID1)
read	34	33
write	26.9	30

5.2.2 ソフトウェア構成

プログラムは，C 言語と Perl(5.004) で記述した．Grid 構築に使用したミドルウェアには Globus Toolkit2.2 を使用した．

5.3 予備実験

基本的な構成の性能を調べるために予備実験を行った．項目は

1. 単純に 2000KB のファイルを分割し，複数のホストへ分散配置したときのスケーラビリティの測定
2. リードソロモン符号のエンコード，デコードにかかる時間測定
3. NFS が IO 性能に及ぼす影響

である．

ストライピングによる IO 性能の向上

まず，ストライピングによって得られる IO 性能向上について実験を行った．2000KB のファイルを分割し，GridFTP で並列コピーしたスループットを測定した．ホスト 1 台の結果はファイルの分割せずに GridFTP で転送したときの結果である．結果のグラフを図 5.1 に示す．

結果を見ると，分散配置に使用するホスト数に比例してスループットが向上していることがわかる．7 台の時間がピークで 735Mbps を記録している．これは，ネットワーク帯域の上限にほぼ達しているので，これ以上性能向上は見込めないと考えれる．また，送信元の HDD ドライブは特に高速な HDD を採用しているわけではなく，ランダムアクセスによって読み出し性能の低下が生じて，8 台へストライピングした時の性能は若干低下したと考えられる．

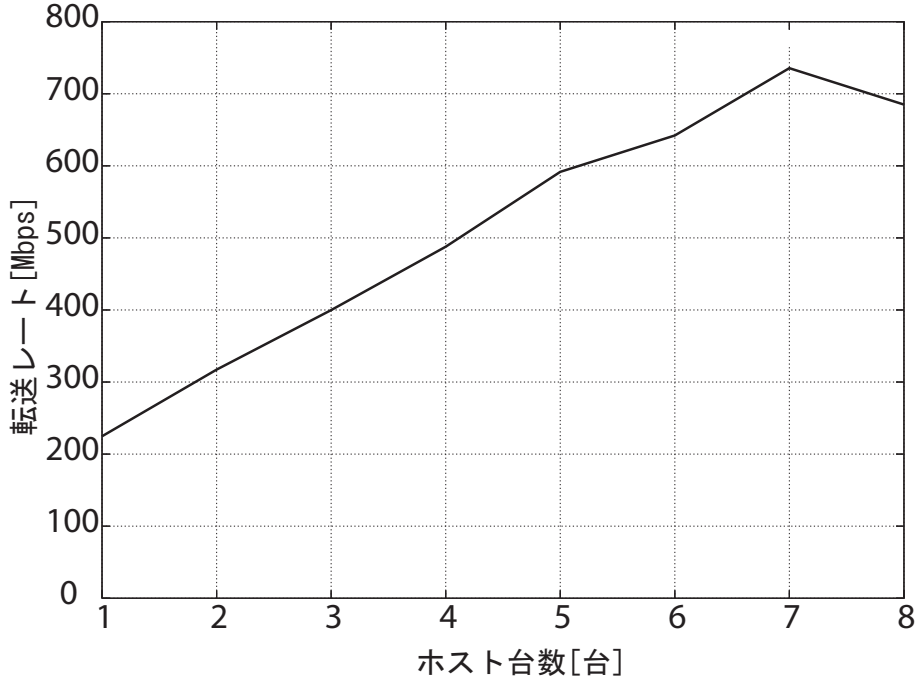

図 5.1: 単純なストライピングの結果

リードソロモン符号のエンコーダ，デコーダ単体の性能

次にリードソロモン符号のエンコード，デコードにかかる時間を測定した．リードソロモン符号のエンコードには，元ファイルのメモリへの読み込み，ファイルの分割，符号の生成という 3 つの処理に分けることができる．実験では，それぞれの処理にかかる時間と，冗長度による処理時間の違いを測定した．リードソロモン符号の生成時間はブロックの切り方や冗長度の設定で複雑に変わるため，計算量を定量的に表現することが困難である．実験では，2000MB のデータを 5 個に分割し，多重パリティを生成した場合の冗長度とエンコード時間の関係を調べた．結果を図 5.2 に示す．

この結果より，データサイズが一定でファイルの分割数が変わらない場合には，リードソロモン符号の冗長度の設定に正比例することがわかった．次に，元ファイルのサイズとエンコードにかかる時間の関係についてのグラフを図 5.3 に示す．この実験では，数種類のファイルを，データブロック 5，パリティブロックを 2 に設定して処理を行った．

結果より，ファイルの分割数と，パリティブロック数が同じ場合には，エンコードにかかる時間は元のファイルサイズに正比例することがわかった．

データブロックの欠落が無い時には，単純にファイルの結合を行って元のファイルを復元できる．途中で足りないファイルが見つかれば，パリティブロックから元のデータを再生成する．実験では，2000MB のファイルをデータブロック 15，パリティブロック 3 に分割したファイルの復元を行った．合計 18 個のファイルの中で 3 つまでファイルを消し，復元にかかる時間の測定を行った．その結果を 5.4 に示す．

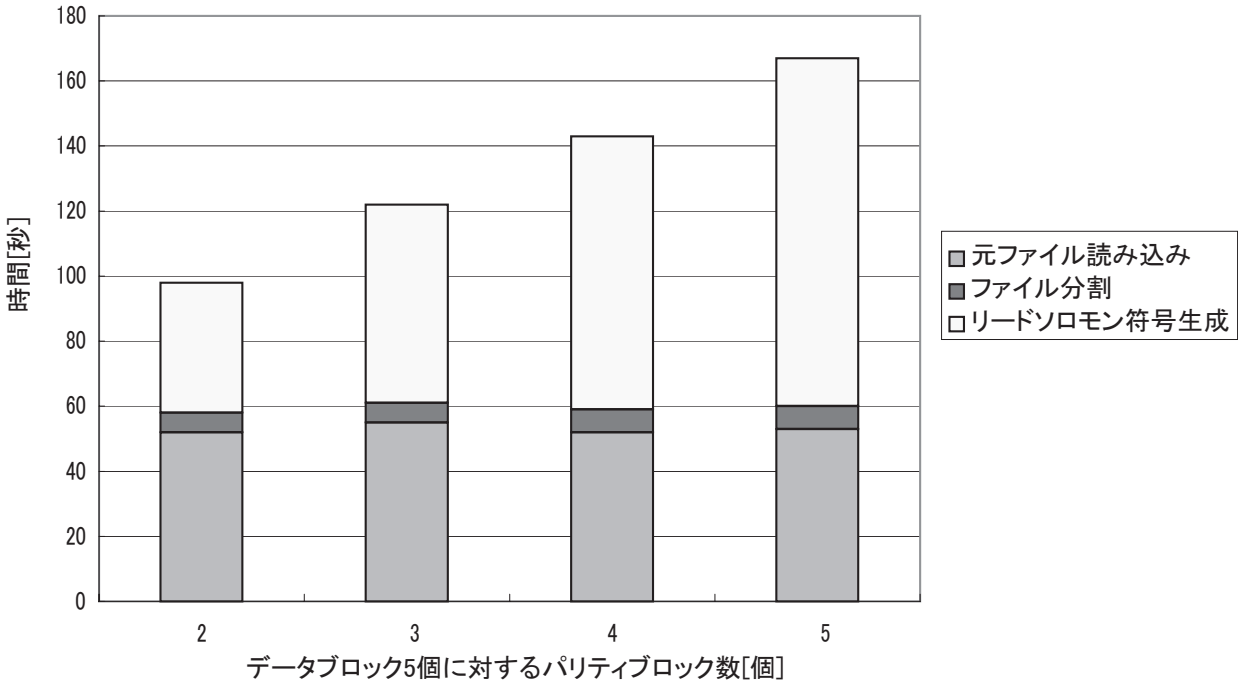

図 5.2: 冗長度とエンコードにかかる時間の関係

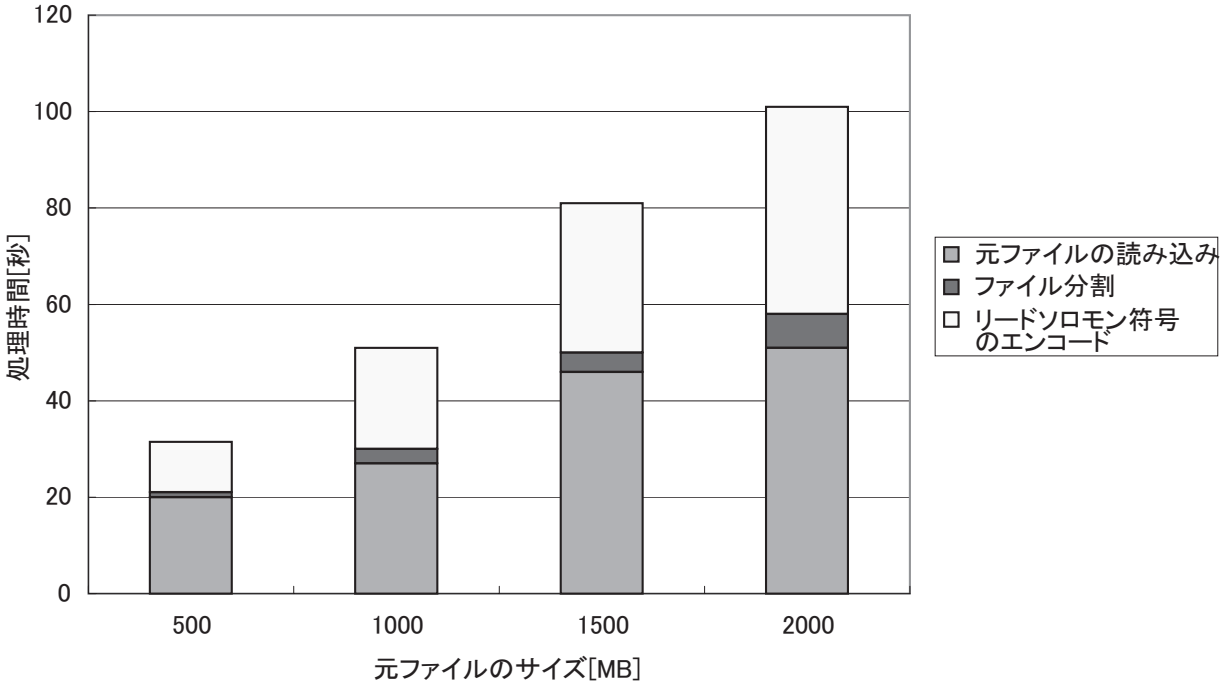

図 5.3: 元ファイルのサイズと，エンコードにかかる時間の関係 (データブロック 5, パリティブロック 2)

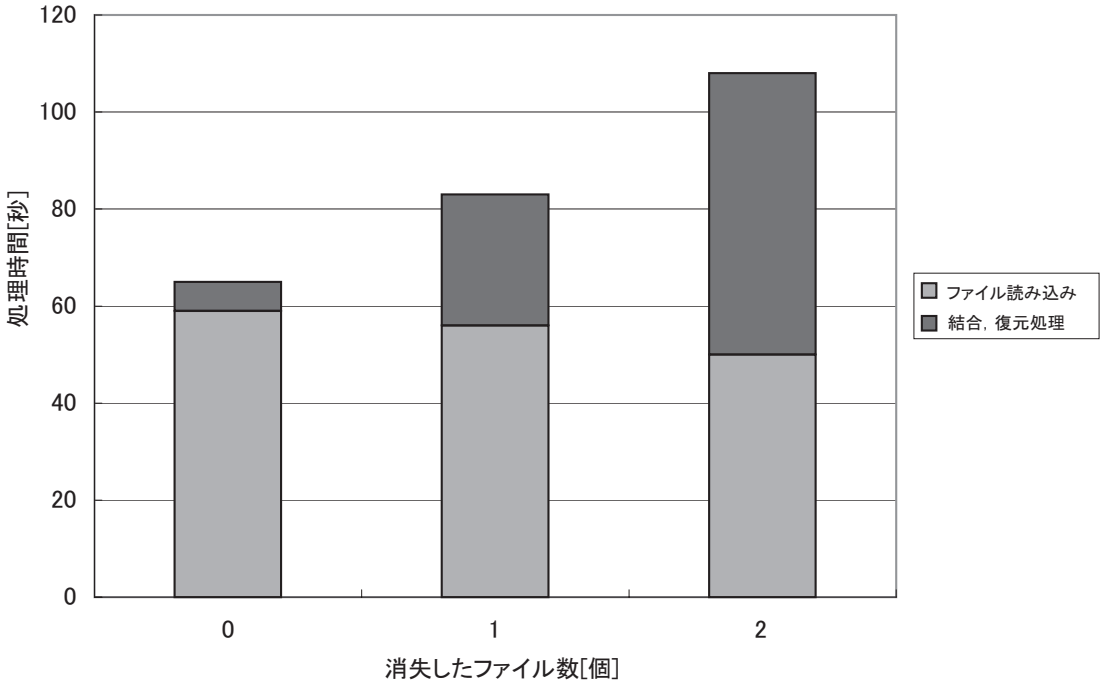

図 5.4: 元ファイルの復元にかかる時間と消失したファイル数の関係

この結果より，ファイルの再生成の時間はファイル欠落数が１つのときは短時間で処理が完了するが，複数のファイルが欠落している場合には処理が長くなることがわかる．これは，複数のファイルが足りない場合，リードソロモン符号からファイルを再生成する際に，足りていない部分同士を復号しながら再生成を進めていくために全体の計算が複雑になってしまうからと考えられる．

デコーダーについてもともとのファイルサイズと欠落したファイルの再生成にかかる時間の関係を調べた．実験には各種大きさのファイルをデータブロック 15，パリティブロック 3 にエンコードしたファイルを使用した．この合計 18 個のファイル中，3 個のファイルを消去し，もとのファイルへの復元を試みたというシナリオにした．実験結果を図 5.5 に示す．実験結果より，リードソロモン符号の計算部分は元ファイルの大きさにほぼ正比例することがわかった．

IO 性能

最後にファイルへのアクセスにおける nfs の影響について測定した．測定項目は次の 3 項目について行った．使用したファイルのサイズは 2000KB である．

- nfs でマウントしたディレクトリからローカルディスクへとコピー
- rcp でファイルをコピー．送受信ともローカルな HDD へアクセス

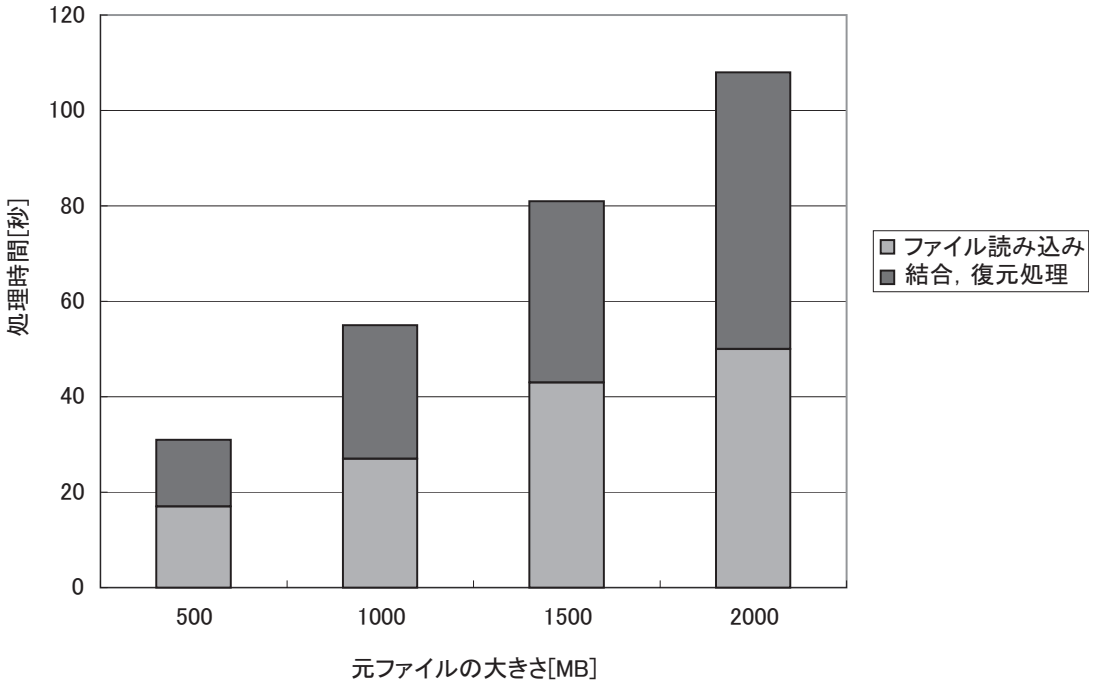

図 5.5: 元ファイルの大きさと処理時間の関係

- rcp と同様にコピー．プロトコルに GridFTP を使用した．

これらの試験結果を図 5.6 に示す．

nfs が一番遅く，rcp の場合の 6 割ほどの性能しか得られなかった．対して GridFTP では 7% 程度の向上が見られた．この結果より nfs を経由したファイルのやり取りを行う場合，nfs の性能がボトルネックとなることが予想される．

5.4 自己処理モードでの性能

自己処理モードでの性能評価を行った．マスタサーバに使用したコンピュータは，5.2.1 に記載した AMD64 を使用した．プログラム中での，処理ホストの選択部分の関数を削除し，処理ホストを localhost と固定して実験を行った．

5.4.1 データ書き込み

データの書き込み時の処理は，配布先ストレージノードの決定，ファイルの分割とリードソロモン符号の生成，GridFTP によるデータの転送を行う．実験結果を図 5.7 に示す．

実験結果では，ストレージノードの決定は，どの場合でもほぼ一定の時間で終わることがわかる．データの処理には 125MB から 500MB のいずれも 30 秒程度の時間がかかって

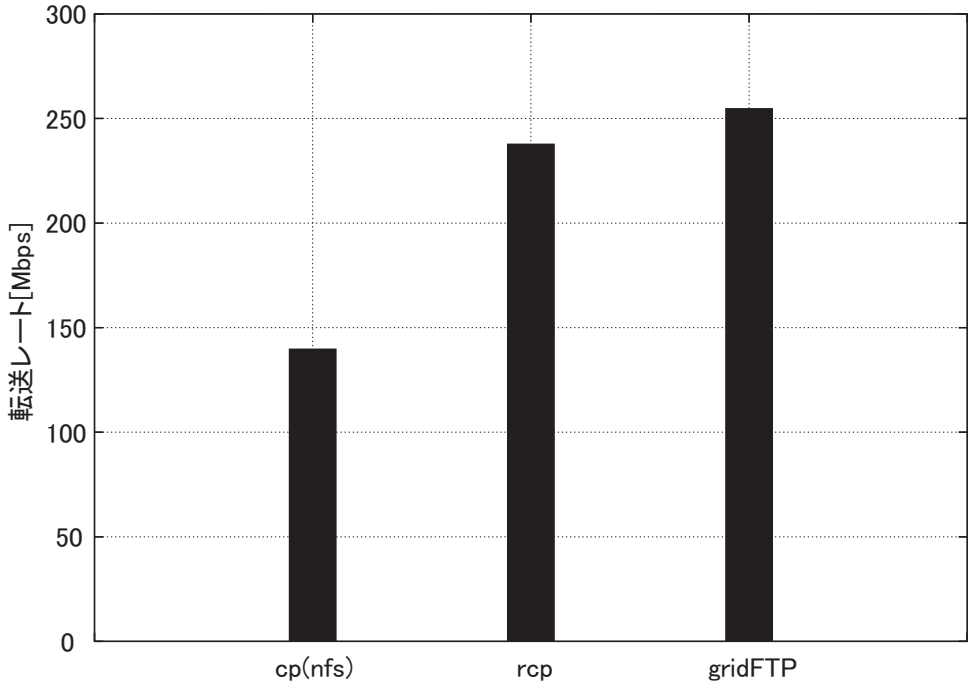

図 5.6: IO 性能の比較

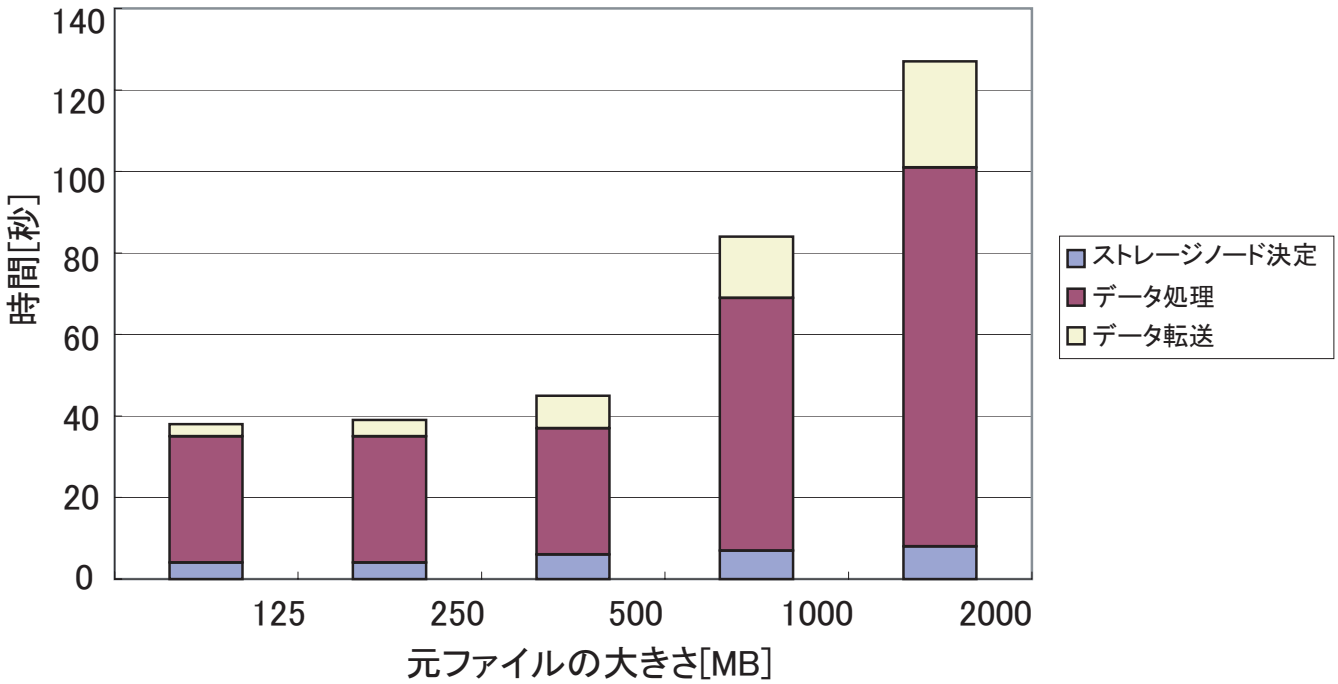

図 5.7: 自己モードでの書き込み性能 (元ファイルの大きさと処理時間の関係)

いる．これはデータ処理のジョブ終了検出を globus-job-status がジョブ完了を検出する時間に誤差があるためだと考えられる．もともと globus-job-status がジョブ完了を検出するのは，実際のジョブが完了してから数秒かかる．これに加えて globus-job-status を 10 秒ごとに実行し，ジョブ状態の監視をしているために 10 秒から２０秒程度の遅れが発生する場合がある．

データ転送は，データ量に対して比例して長くなる．ファイルサイズに対して転送時間の増加の割合が鈍いのは，転送ファイルサイズが小さいと GridFTP の転送速度があがりきる前に転送が終了するからだと考えられる．

5.4.2 データ読み出し

先の実験で grid 中のストレージノードへと分散配置したファイルを GridFTP でコピーし，元のファイルを復元する実験を行った．読み出し時には，分散配置したファイルが揃っている場合と，grid に異常がある場合について実験を行った．

故障が存在しないとき

まず，実験では故障がなく，システムが健全な場合の読み出しについて実験を行った．ファイルサイズは 125MB，250MB,500MB, 1000MB, 2000MB の５つである．これらのファイルのデータブロック，パリティブロックをストレージノードから回収し，もとのファイルを復元した．故障がない場合には，リードソロモン符号の復号は必要ないため，単純に回収したデータブロックを結合するだけでよい．実験結果を図 5.8 に示す．

この結果では，予備実験でおこなったデコーダの性能とほぼ一致する結果が得られた．データが正しく復元されているかは，md5sum コマンドでハッシュ値を計算し，元ファイルのハッシュと比較し，同じことを確認した．

縮退運転時

次に，ストレージノードに異常が発生した場合について実験を行った．

ファイルの種類は同様に５種類のファイルを使用した．まずこれらのファイルを使用して，復元可能な故障数までファイルの断片を消去し，読み出しのコマンドを実行した．実験結果を図 5.9 に示す．

先の実験と同様に実行時間が 30 秒以下の場合には，正しくジョブ終了が検出されずに，約 30 秒と検出されてしまっている．grid へと分散配置しておいたファイルが消失してしまっている場合，存在しているファイルだけ転送されるために，転送時間は短くなることが実証された．しかし，消失ファイルが多ければ再生成すべき演算量が増えるため，全体の処理時間は増加してしまうことがわかった．ここでも md5sum コマンドによるハッシュ値の確認を行い正しいデータが再生できていることを確認した．

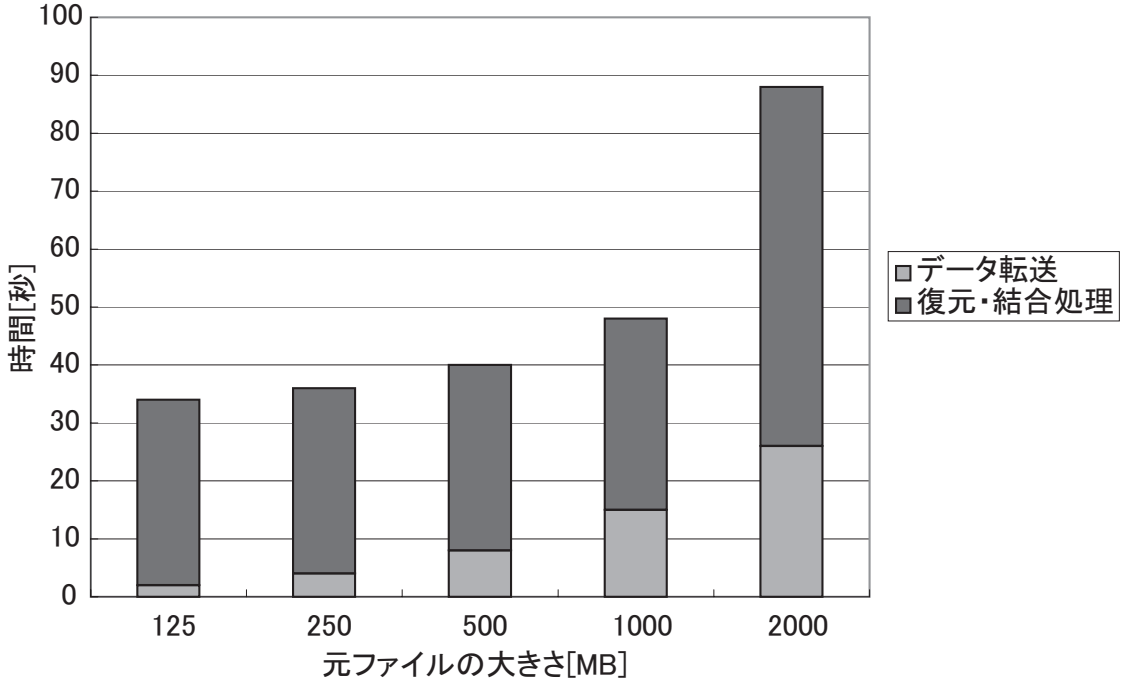

図 5.8: ファイルの読み出し，復元

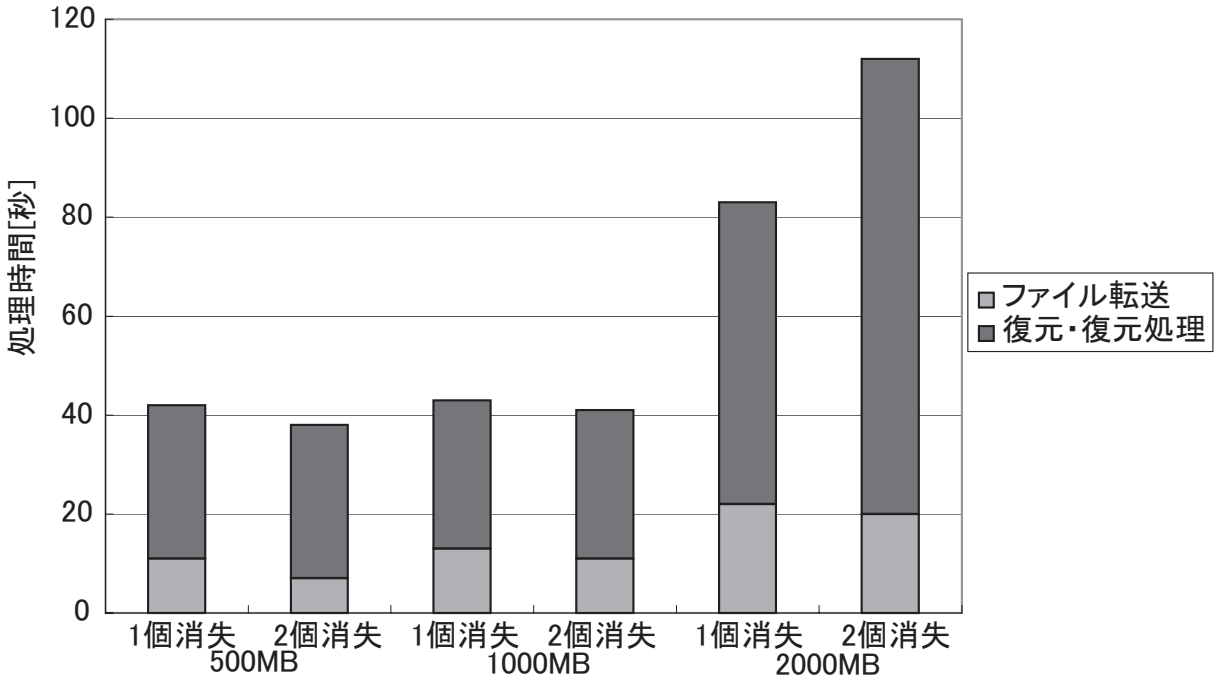

図 5.9: 縮退運転時の性能

5.5　外部処理モードでの性能

この節では外部処理モードでの性能を計測した．外部処理モード用の関数を作成し，プログラムに組み込んだ．

5.5.1　データ書き込み

データの書き込み時には，書き込むファイルサイズとプログラム実行マシンの空きメモりを比較しどのホストでリードソロモン符号を生成するかを決める．結果のグラフの,『ホスト決定』の項は，システムの信頼性算出と，リードソロモン符号生成に使用するノードの決定時間を含んでいる.『分割処理』はファイルのメモリへの読み込みと，分割，リードソロモン符号のエンコード時間と書き出し時間を含む.『転送時間』は先の処理で作成したデータを GridFTP で各ストレージノードへと転送する時間である．結果を 5.10 に示す.

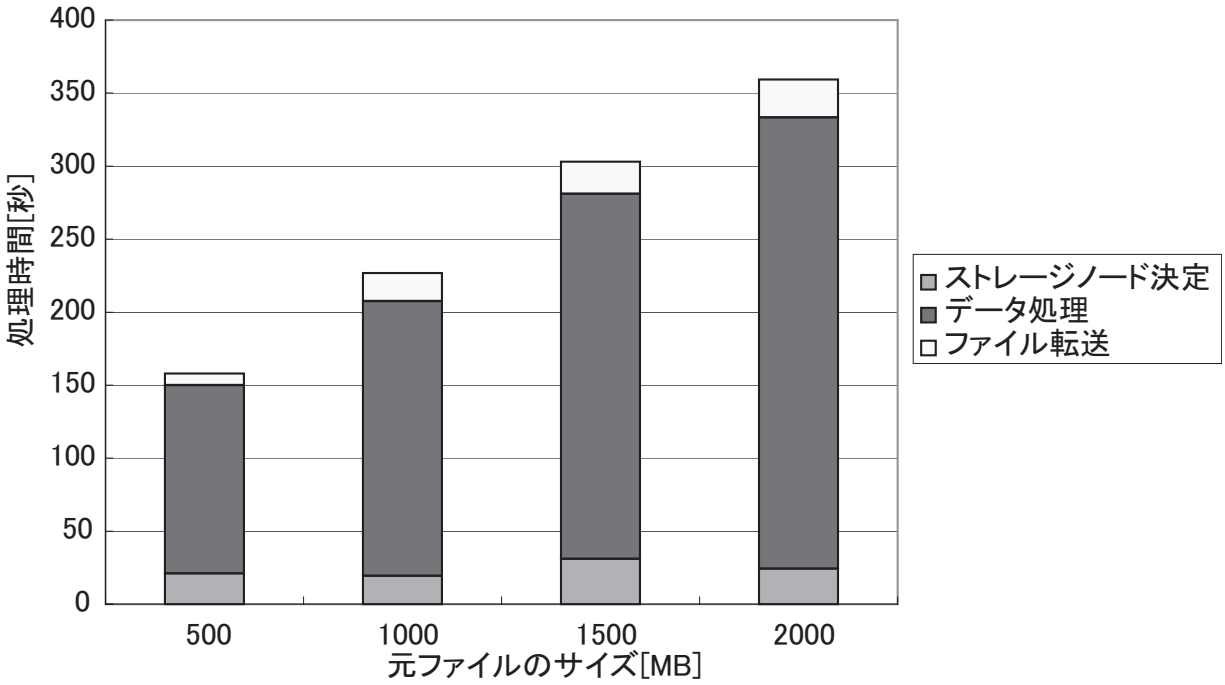

図 5.10: 外部処理モードにおけるファイルサイズと書き込み時間の関係

結果のグラフをみると，分割処理が自己処理モードに比べて大幅に増えていることがわかる．これは，ファイルの読み込み時間が大幅に増えているからだと考えられる．外部処理モードでは，リードソロモン符号の処理ホストとなる可能性のあるホスト全てからアクセス出来る必要があるために，NFS でマウントされたディレクトリに元ファイルをおいている．このために NFS の性能がボトルネックとり，分割処理が長く時間がかかった．

5.5.2 データ読み出し

次に，grid へ分散配置されているデータを収集し，元のデータを復元する処理を外部処理モードで行った．

故障が存在しないとき

この節では，分散配布したファイルたちが完全に揃っている状況でテストを行った．復元したファイルは nfs 経由でマウントされているサーバーに書き出した．結果のグラフを図 5.11 に示す．

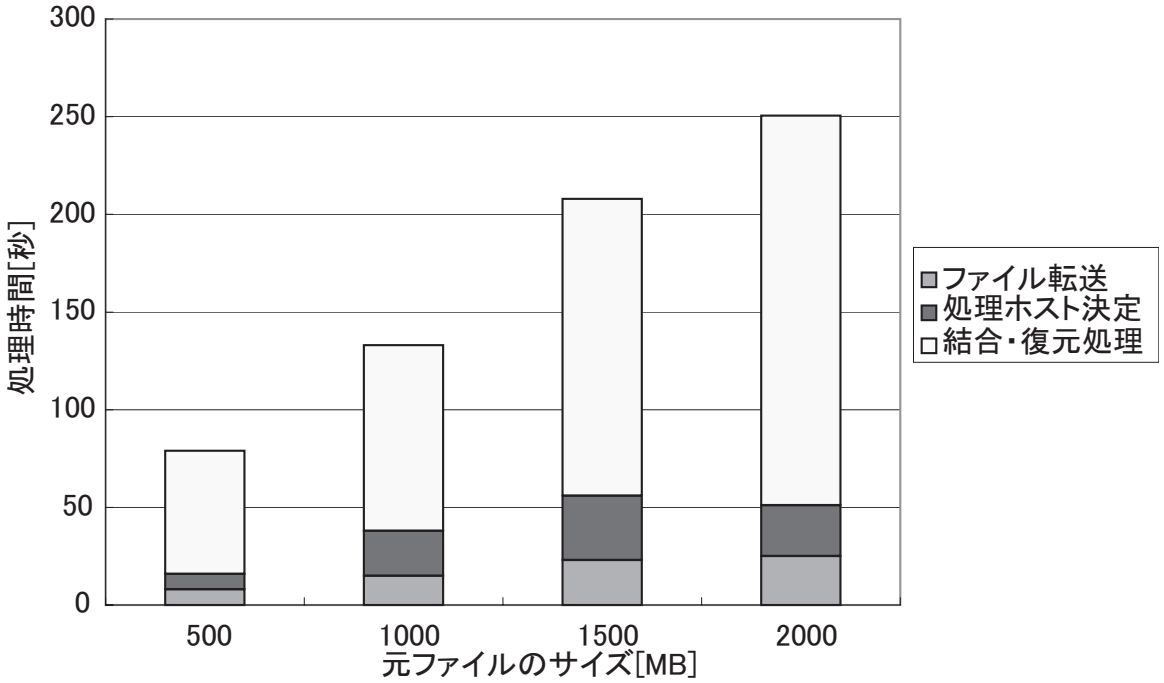

図 5.11: 外部処理モードにおけるファイルサイズと読み出し時間の関係

結果を見ると，GridFTP によるデータ収集の時間と，ファイルの結合復元処理はファイルサイズに比例して長くなることがわかる．nfs の性能がボトルネックとなり全体の処理時間は長くなっている．ファイルサイズの違いによる処理時間の差は予備実験とほぼ同様の結果となり，予測される値に近い結果を得ることができた．

縮退運転時

縮退運転では，分散配置をしたファイルを意図的に消去して，正しくファイルが復元出来ているか，その処理にかかる時間について測定を行った．結果を図 5.12 に示す．

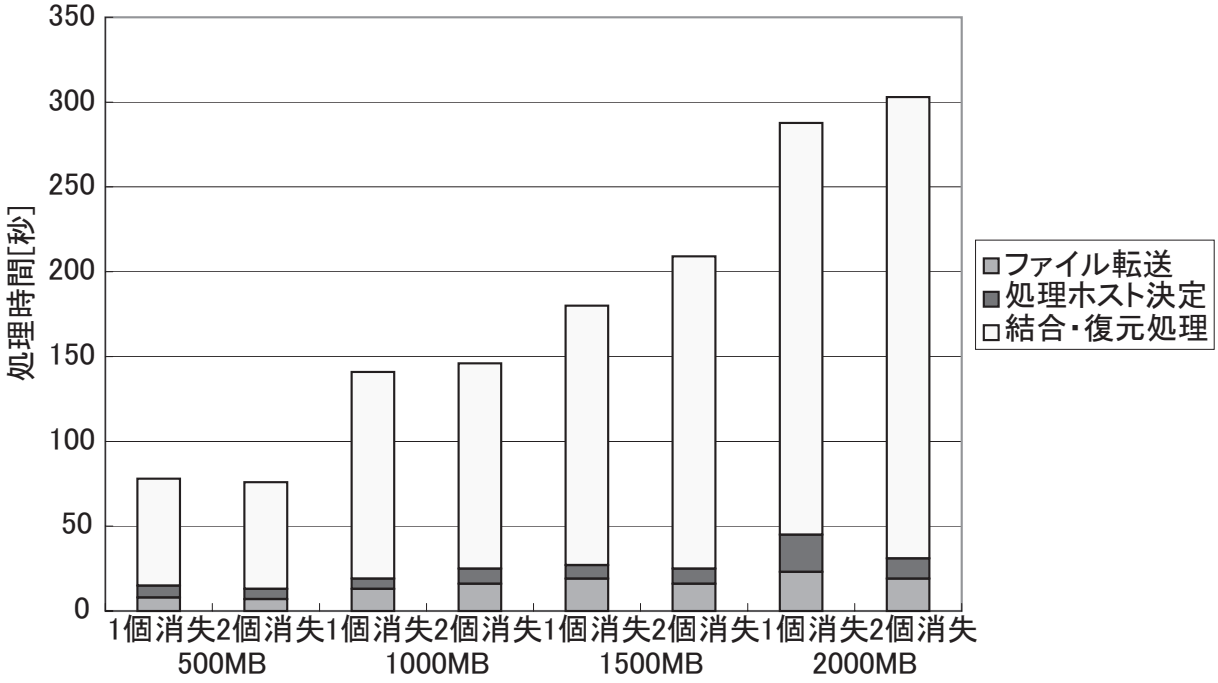

図 5.12: 外部処理モードにおける縮退運転時の性能

縮退運転では，分散配置してあるファイルがいくつか消失しているために，GridFTP によるファイル転送の際の総転送容量は減っていることになるはずである．結果のグラフでは，消失ファイル数が多ければ転送時間が短くなっていることが確認出来た．ここでも，ファイルサイズが小さい場合には，１つのファイルをリードソロモン符号を使用して復元する時間が５秒から１０秒程度と短い．この程度の処理時間の差は，globus がジョブ終了を検出する際の大きなタイムラグによって吸収されてしまう．よって 500MB，1000MB のファイルを復元する際に同じような処理時間の結果となっている箇所があるが，正常な結果だと考えられる．

5.6 GridFTP を使用する外部処理モード

外部処理モードにおいて nfs を経由しないでファイルの転送をすべて GridFTP で行った場合の性能評価を行った．

5.6.1 データの書き込み

先と同様の試験を行った．元ファイルの読み込みから各ストレージノードへ分散配置するまでの時間を測定した．結果を 5.13 に示す．

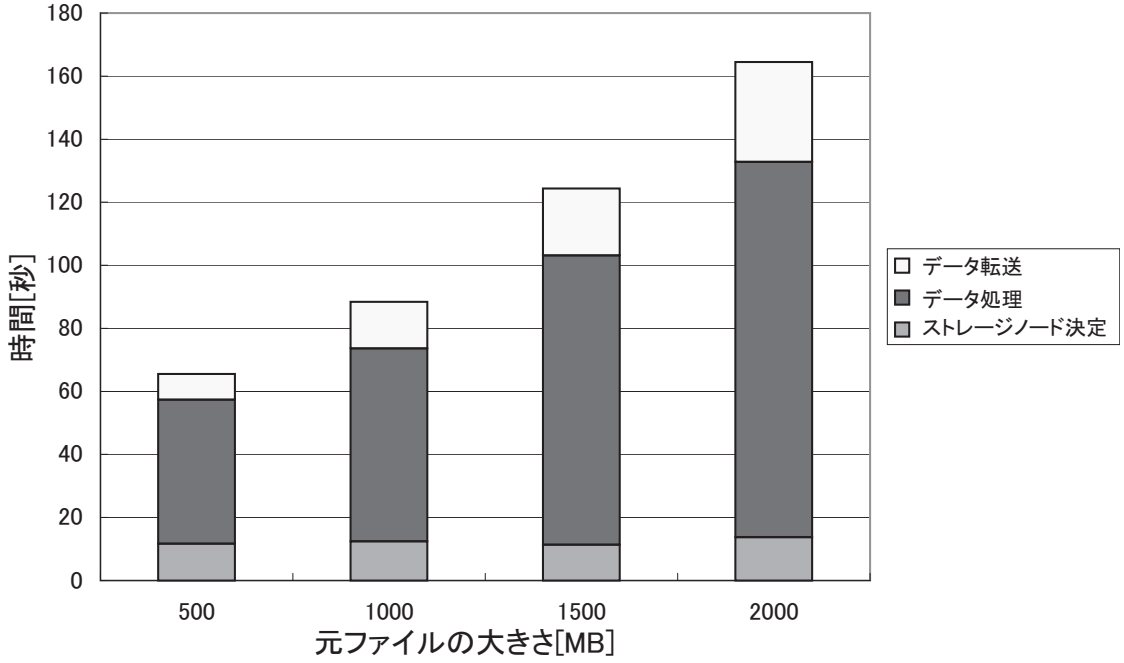

図 5.13: GridFTP のみ使用した外部処理モードにおける書き込み時の性能

結果より，nfs を使用した場合と比較して 4 割程度の時間ですべての処理が完了することがわかった．これは GridFTP と nfs のファイル転送能力の差だと考えられる．

5.6.2 データ読み出し

２つの実験パターンでファイルの読み出しを試験した．

故障が存在しないとき

まずは分散配置したファイルに異常がない場合について試験を行った．評価手順，方法は前の試験と同様とする．結果のグラフを図 5.14 に示す．

結果より nfs 使用時よりも 4 割から 5 割の時間で処理が完了することが分かった．

縮退運転時

先の試験と同様に分散配置したファイルブロックのうちいくつかを消去して，元のファイルを復元するのにかかる時間を測定した．結果のグラフを 5.15 に示す．

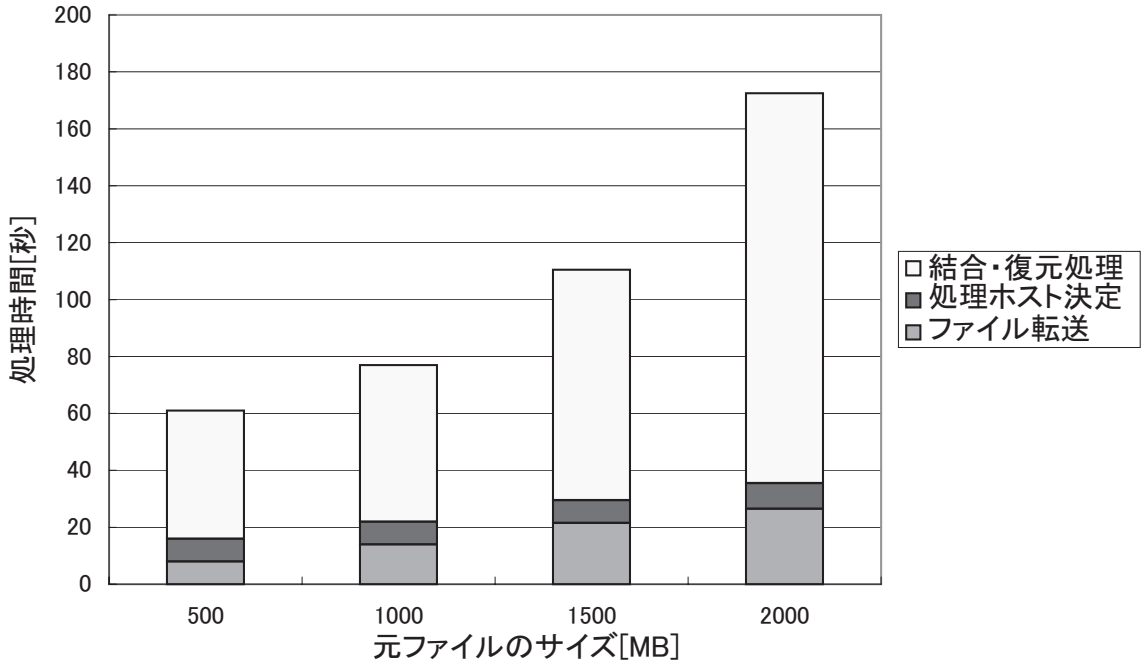

図 5.14: GridFTP のみ使用した外部処理モードにおける読み出し時の性能

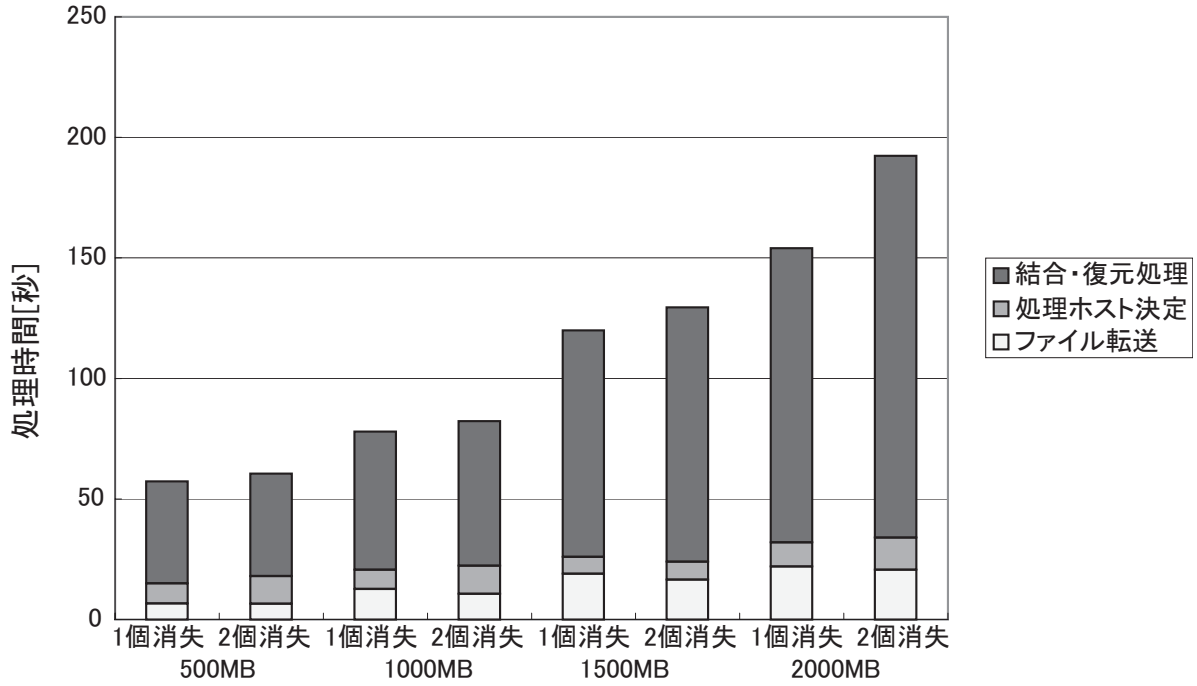

図 5.15: GridFTP のみ使用した外部処理モードにおいて故障がある時の読み出し時の性能

この試験においても先と同様の試験結果が得られた．これらの結果より，ファイル転送方法を gridFTP に統一することで汎用性向上，転送時間短縮を図ることができることがわかった．

5.7 データの再生成

いくつかのファイルを grid へ分散配置し，そのファイルたちの健全性を検証するプログラムを作成した．ファイル断片がなくなっていたり，データを保存しているストレージノードが保存時と異なる信頼度クラスに属している場合などにデータを復元し，再度分散配置しなおすようにしてある．

データの確認にかかる時間は，元ファイル１つにつき 15 秒から 20 秒程度の時間を要する．ファイルに異常がある場合には，続いてデータのデコードから再エンコードして分散配置するための時間が必要になる．

5.8 既存システムとの性能比較

この節では，既存のシステムのと違いについて述べる．既存システムでは，レプリカデータを使用して耐障害性の確保と負荷分散を行っている．本研究のシステムでは，ストライピングと多重パリティを使用することで，耐障害性と負荷分散を行っている．耐障害性とディスク利用効率を考えた場合，１つのレプリカデータを作成すると，元データと同じだけのディスク容量が必要となる．使用可能なディスク容量を基準にして考えた場合の比較を表 5.2 に示す．この表には計算例の一部だけを掲載したが，レプリカ方式にくらべ

表 5.2: ディスク利用効率の比較

	レプリカ	パリティ 1 個	パリティ 2 個
4 台から 15 台まで 使うときに利用可能な%	50(%)	75〜93(%)	50〜87(%)
冗長度	二重化	N+1	N+2

て多重パリティ方式を採用したほうが，ディスク使用効率がよいことがわかる．使用するディスク台数が多ければ，ディスク利用効率の更なる向上をすることが可能である．

本研究で使用するコンピューターは一般ユーザーでも電源を切ったり，OS の入れ替えなどを行うことが可能である．このような動作条件が不安定なコンピュータを使用するための対策として，ストレージノードの信頼度を測定し，信頼度算出に基づいて配布先とパリティ多重度を決定することで，要求する信頼度を確保することができる．

5.9 まとめ

本章では，システムの性能についての性能評価を行った．予備実験で基本性能の実測と，grid を構築しての性能の実測を行った．まず，分散配置することで IO バンド幅が向上することを検証した．

予備実験では，ストライピングによる IO 性能の向上と，リードソロモン符号のエンコーダとデコーダについての基本性能，および nfs の性能について評価をした．ストライピングをしたときの GridFTP の転送レートは最大 750Mbps 程度に達した．エンコーダ，デコーダともに，使用する元データの大きさに比例して処理時間が長くなることがわかった．元データの分割数が同じであれば，パリティの多重度があがるにつれて処理時間長くなることがわかった．元ファイルの復元時に一部のファイルが欠落していても正しく復元出来るが，このときファイルが１つだけ異常がある場合と２つ以上以上がある場合では，差が大きくなることがわかった．これは複数異常のある場合には同時に複数の行列演算を行う必要が出てくるために時間が多くかかるのだと考えられる．nfs の転送スピードは GridFTP などよりもかなり遅いことがわかった．nfs を経由したファイルのやり取りがボトルネックとなる可能性が示された．

grid へと展開したシステムの性能評価では，基本的な性能はリードソロモン符号単体の性能とほぼ一致する性能が得られた．しかし，globus のジョブ終了検出に時間がかかることや，globus-job-status の実行間隔のタイミングなどで，３０秒以下の処理はすべて３０秒まで検出されないことがわかった．grid 環境における外部処理モードでの処理は，どのコンピュータからもアクセス可能な場所に元ファイルをおく必要があるため，nfs 上のディレクトリに保存した．このときは，nfs の転送速度がボトルネックとなり，処理時間が大幅にのびることになった．

データの再生成では，保存したファイルの健全性を検証し，異常 (ホストダウン，信頼度クラスの相違，ファイル消失) があれば冗長符号からの復元と，再エンコードと分散配置をすることで正常な状態を保つことが示された．

本システムと既存のシステムの違いついて述べた．多重パリティを用いた方式ならば，レプリカ方式と同等以上の冗長度を持ちながら，ディスクの利用可能な割合が多いことを示した．

第6章　結論

本研究では，自律分散ファイルシステムについて，性能と信頼度の面から検証を行い，特に以下の点について問題点の考察と解決案の提案を行った．

- ストレージノードの信頼度とシステム全体の信頼度
- 動作方式と処理時間への影響
- データの健全性チェック

信頼度の計算では，個々のストレージノードの稼働状況を一定間隔で調査し，MTBF と MTTR を算出してそのノードの稼働率を算出した．この稼働率を基にして３クラスに分類をした．このクラスごとに代表する信頼度を設定し，m-out-of-n システムとして信頼度計算を行い，要求するシステム信頼性を確保できるように，データとパリティの数を調整するようにした．信頼度の低いクラスを使用する場合はパリティの多重度を上げて，信頼度の高いクラスを使用できるときはパリティの多重度を下げる．これで，書き込み時にはレプリカ方式と同等以上の信頼性を確保することができるようになった．

動作方式の違いでは，両方式ともリードソロモン符号の処理に時間がかかることがわかった．エンコーダ，デコーダはメモリ不足でスワッピングが起きると極端な性能低下を示す．また，マスタサーバが一台だけの場合，負荷が集中することが予測されたため，外部処理モードを作成した．メモリが足りなくなってしまうようなファイルを余裕のあるコンピュータで処理をすることで，負荷分散とメモリ不足を回避した．

性能評価では，自己処理モードと外部処理モードの場合について評価した．それぞれ書き込み性能，縮退運転時の読み出し性能，故障がないときの読み出し性能について試験を行った．先に算出した信頼度に基づき動的にデータの分散配置先を選択し，ただしくデータの保存，読み出しができることを確認した．GridFTP による並列転送では，GigabitEthernet の限界値付近までスループットが向上することが確認出来た．外部処理モードにおいては，nfs の性能がボトルネックとなることがわかった．今後 nfs を経由せずに処理を行うようにする必要性がある．縮退運転時には，障害発生の数が１か２以上かで復元時間に大きく差がでることがわかった．このため，ファイルの復旧は，１つでも障害があればすぐに行うことにするのが適していると考えられる．

以上の結果より，本研究のシステムではリードソロモン符号の扱いによるオーバーヘッドがあるものの，レプリカ方式と同等以上の信頼性を確保しながら，ディスク利用効率のよい分散ストレージを構築出来ることが示された．これにより余っているディスクスペー

スを有効に使用して，アーカイブシステムなどを構築する応用ソフトウェアの開発が期待出来る．

しかし，本研究の今後の課題として，次のようなものが考えられる．

- リードソロモン符号を扱うオーバーヘッド
  現在のシステムでは保存するデータをデータブロックとパリティブロックに分割し一時ファイルに書き出してから GridFTP で転送を行っている．ここの実装を一時ファイルを使用せずにメモリ上のデータを直接転送するようにするだけでかなりの高速化が図れるはずである．

- より詳細な信頼性モデルの構築
  ストレージシステムの信頼性を議論する際に指標にアベイラビリティと Mean Time To Data Loss (MTTDL) がある．本システムは m-out-of-n システムとして信頼性を算出している．これはアベイラビリティに相当するもので，システムにアクセスしてデータを読み書きしようとしたときにシステムが動いている確率だといえる．本システムのようなモデルを詳細に表そうとした場合，マルコフモデルを用いて MTTDL を求める方法もある．MTTDL では，データが完全に失われるまでの時間を表すことができる．より詳細な信頼性計算を行うためにマルコフを用いた信頼性モデルを求める必要がある．

- GridFTP の帯域制御
  多数のストレージノードへ分散配置する際に複数の GridFTP が帯域を全開で転送を試みるため全体のネットワーク帯域と，HDD のランダムシークが追いつかなくなり，性能の低下を引き起こす原因となる．

- MDS への登録
  メタデータや稼働率によってクラス分けしたストレージノードリストなどを MDS へ登録し，メタデータの信頼性向上を狙う．

- 信頼性の低いストレージノードの活用
  今回のシステムでは稼働率が 99% を満たさないノードは使用しないことになっているが，これらのストレージを活用する方法について検討を行う．具体的には，信頼性の低いストレージノード同士で，ファイルの一部のレプリカを作成し，部分的に信頼性を上げて，優先度の高いクラスに見せかける方法が挙げられる．

これらの課題について研究を進めれば，大幅な性能向上とより詳細な信頼性計算が実現されて，より優れた分散ファイルシステムとして実現されるはずである．

謝辞

本研究をおこなうにあたり，熱心なご指導を賜り，また対外発表の機会を与えて頂きました井口寧助教授に心から感謝いたします．また，適切なご指導を賜りました松澤照男教授，田中清史助教授に深く感謝致します．

日頃から貴重な御意見，御討論を頂きました井口研究室と堀口研究室の皆様をはじめとする多くの方々の御援助に対し，厚く御礼申し上げます．

参考文献

[1] Ian Foster and Carl Kesselman,"The GIRD Blueprint for a New Computing Infrastructure",morgan Kaufmann Publishers,Inc.,1999

[2] Ian Foster and Carl kesselman,"The GRID2 GIRD Blueprint for a New Computing Infrastructure",morgan Kaufmann Publishers,Inc.,2004

[3] 建部 修見, 森田 洋平, 松岡 聡, 関口 智嗣, 曽田 哲之, 「ペタスケール広域分散データ解析のための Grid Datafarm アーキテクチャ」, ハイパフォーマンスコンピューティングと計算科学シンポジウム HPCS2002 論文集, pp.89-96, 2002 年 1 月

[4] 竹房 あつ子, 建部 修見, 松岡 聡, 森田 洋平, 「Grid Datafarm におけるスケジューリング・複製手法の性能評価」 先進的計算基盤システムシンポジウム SACSIS2003 論文集, 情報処理学会, 電子情報通信学会, pp.121-128, 2003 年 5 月

[5] A.Ghiselli, "DataGrid Prototype 1", TERENA Networking Conference, 3-6 June 2002

[6] H.Stockinger, F.Dono,E.Laure, S.Muzzafar et al,"Grid Data Management in action", computing in high energy physics (CHEP 2003)

[7] Shingo Takeda, Susumu Date, and Shinji Shimojo, "GSI-SFS: A Grid File System", IPSJ SIG Technical Reports (2003-OS-93) in Okinawa, Japan, pp. 97-104, May 2003.

[8] Hiraki, K. Inaba, M., Tamatsukuri, J., Kurusu R., Ikuta, Y., Hisashi, K. and Jinzaki, A., " Data Reservoir: Utilization of Multi-Gigabit Backbone Network for Data-Intensive Research, " to appear Proc. SC2002(CDROM), 2002.

[9] GFLIB - C Procedures for Galois Field Arithmetic and Reed-Solomon Coding, http://www.cs.utk.edu/ plank/plank/gflib/

[10] Bonnie, http://www.textuality.com/bonnie/

[11] 宇野俊夫,「ディスクアレイテクノロジ RAID」, エーアイ出版, 2000.

本研究に関する発表論文

[1] 渡辺 浩二，松澤 照男，井口 寧,「ヘテロジニアスな環境における grid 向け自律分散ファイルシステム」，電気関係学会北陸支部連合大会，Sep 2004

[2] 渡辺 浩二，松澤 照男，井口 寧,「信頼性を考慮したキャンパス grid 向け自律分散ファイルシステム」，情報処理学会研究報告, ARC-162 HPC-101, Mar 2005