信頼性を考慮したグリッド向け自律分散ストレージシステム

井口 寧[†,††] 渡辺浩二[†††,☆] 松澤照男[†]

本論文では，企業や大学の組織内にある一般のユーザが利用する計算機の余剰ディスク領域を統合し，仮想的な大規模ストレージを提供する手法を提案する．一般のユーザが使用する計算機を記憶ノードとして用いる場合，ユーザの不用意なシャットダウンやネットワーク断線などによる，稼働率の低さや不均一さが問題となる．本研究では，可用率の低さに対しリードソロモン符号を用いた冗長性の高い符号化を用い，データとしての信頼性を確保する．また，可用性の不均一さに対し，動的なデータの再配置を行う手法によって，保存されたデータの信頼性を確保する．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し，この数値をもと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用い，従来のレプリカ方式よりもディスクの使用効率向上を図った．これらの手法によって，信頼性が不均一な環境においても，十分なレスポンスタイムを保ちながら，システムの信頼性の確保とディスク領域の利用効率化が達成できた．

Autonomous Distributed Storage System for Grid that Considers Reliability

YASUSHI INOGUCHI,[†,††] KOJI WATANABE[†††,☆] and TERUO MATSUZAWA[†]

We propose an autonomous distributed storage system that considers reliability and utilization of disks. Nodes of a grid system are used by end users and most of them have unused disk area. The proposed system offers a virtual large storage area combining such unused disk area. However, nodes used by end users are suddenly shut downed or detached from networks. Proposed system uses Reed-Solomon code to keep high redundancy, and re-distributes data dynamically to keep highly availability. Availability of each data node are investigated and redundancy level and data node to be used are decided. Reed-Solomon code offers more efficient storage area usage than replica method that used by other researches. Using these method, high system availability and high storage area utilization can be achieved with enough response time in a GRID environment.

1. はじめに

近年，高エネルギー物理やヒトゲノム解析などの大規模データ解析を必要とする分野ではグリッド技術[3),4)]がキーテクロノジーとなっている．また，一般的な使用にはオーバスペックともいえるPCの性能向上を受け，地球規模でのグリッドだけでなく，企業や学校単位でのグリッド（組織内グリッド）にも注目が集まっている．

組織内グリッドのノードとなる個々のワークステーションなどの多くは，近年のディスクの急速な大容量化を受けて，数百GB以上のディスク容量を有しているが，実際に必要な容量はたいていの場合それほど多くなく，これらの容量の半分以上が遊休領域となっているシステムが少なくない．これらの遊休領域を組織全体で統合することができれば，仮想的に数TB以上もの巨大なディスク領域を構成することが可能となる．

大容量ストレージシステムとしては，テープライブラリからオンデマンドでデータを読み出すPetaSite[1)]やAMSS[2)]などが実用化されているが，これらのテープベースのシステムは，専用のシステムを必要とし，導入後もテープ寿命の管理などを行う必要がある．テープベースの手法に対し，グリッド内の分散ストレージであれば，(1) 専用システムを導入せずに遊休資源を利用できること，および (2) テープ管理などの運用コ

† 北陸先端科学技術大学院大学情報科学センター
Center for Information Science, Japan Advanced Institute of Science and Technology
†† 科学技術振興機構さきがけ研究21（機能と構成）
PRESTO, Japan Science and Technology Agency
††† 北陸先端科学技術大学院大学情報科学研究科
Graduate School of Information Science, Japan Advanced Institute of Science and Technology
☆ 現在，株式会社日立製作所 RAID システム事業部
Presently with Disk Array Systems Division, Hitachi, Ltd.

ストの軽減できること，の2つの利点がある．

グリッド上の分散ファイルシステムに関する先行研究では，大量のデータが扱われるために，広域に分散したデータを高速に転送が可能で，数千キロ離れた場所へ少ない遅延で転送するシステムなどが提案されている[5)~8)]．しかし，研究の主な対象がデータの保存，転送能力の向上であり，データの保存性や信頼性についての議論はほとんどされていない．これは，先行研究でのグリッド構成に使用しているコンピュータが，もともと信頼性が高く，常時稼働を前提に運用されているために，詳しく信頼性を知る必要がないためと考えられる．また，それぞれの要素システムの信頼性が十分高いことを仮定しているため，データの信頼性保全のための手段はもっぱらレプリカであり，オーバヘッド，冗長性の確保，負荷分散には有利になるが，物理的な総記憶容量に対する利用効率は高くない．

それらに対し，本研究のターゲットとなるキャンパスなどの組織内グリッドでは，一般ユーザの使用しているコンピュータやワークグループサーバ，およびネットワークをグリッドの要素計算機として構築する．しかし，このようなノードは，管理者や運用方針が異なり，個々のディスク領域に保全されたデータは必ずしも安全ではなく，またノードが不意にシャットダウンされたりネットワークから切り離されたりすることがあるため，統計的なデータの保全に関する対策が必要である．

そこで本研究では，個々の計算機要素の信頼性を統計的に管理し，それぞれの計算機要素の信頼性に応じたデータの冗長性を持たせたグリッド向けストレージシステムを提案する．グリッド上にデータを分散配置する際に，あらかじめ算出しておいた稼働率に応じて，データの配布先とデータの冗長度を動的に変化させる分散アーカイブシステムの構築，性能評価を行った．データの信頼性は，リードソロモン符号[9)]（以下RS符号）で生成した冗長データを利用し，単なるレプリカを用いた場合よりも，データの信頼性やディスク領域の利用効率を高めることができる．

本論文の構成は，次のとおりである．2章でシステムの構成と実装について述べ，3章では実験的に構築したシステムの性能を評価する．4章で他の研究事例について紹介し，最後にまとめを示す．

2. 提案システムの構成と実装

2.1 想定する利用環境

本システムは，大学や企業内にある，一般ユーザの利用する端末ややワークグループサーバの遊休ディスク領域を1つの統合化されたストレージ領域として利用することを目的としている．これらのシステムは，たとえばLinux-PCなどを想定し，基本的に24時間運転を仮定するが，システムのバージョンアップやセキュリティパッチ当てなどの作業により，不定期なシステムダウンを許容する必要がある．これらのデータの断片を格納するノードは，一般のユーザが利用しているワークステーションを想定している．計算センターなどのワークステーションでは，ある程度以上の稼働率が期待できるが，エンドユーザ用のワークステーションの場合，システムの機種がまちまちであり，システムそのものの信頼性が一定でないばかりか，個人の都合によるシステムの停止やリブートが頻繁に発生する可能性を考慮する必要がある．

本研究では，複数の研究室や部署に配置されているワークステーションにデータ断片を格納する．研究室や部署が異なると，アカウント体系や管理ポリシが異なる場合が多いので，できるだけ広範囲のシステムをノードに加えることを狙い，GlobusToolkit（以下GTK）を用いてシステム構築を行った．

格納するデータサイズは，数百MB～数GBの比較的大きなデータを想定している．このような仮想ファイルシステムは，ローカルディスクに比べてデータの入出力に時間がかかるため，アーカイブ的な使われ方が多いと考えられるからである．

データ入出力は，データの書き込みおよび読み出しのための専用コマンドによって行う．元データのファイル名や取り出したいファイル名を引数として呼び出す．たとえば，システムへのファイル書き込みは，次のコマンドで行い，指定されたファイルを断片化しRS符号による冗長を行ったうえでグリッドノードへの書き込みを完了する．

“store.pl 〈*file_name*〉”

また，読み込みの場合は，次のread.plコマンドによって，グリッドノードからの〈*stored_file_name*〉で指定されたファイル断片の収集と，欠落断片があればファイルを修復し，コマンド発行ノード上のファイル〈*write_file_name*〉としてユーザに返す．

“read.pl 〈*stored_file_name*〉〈*write_file_name*〉”

2.2 システムの構成

提案するシステムでは，最終的に統合されたデータの信頼性を保証するため，ノード計算機ごとに算出されたシステム信頼度に応じてリードソロモン符号（RS符号）[9)]に基づいて，多重パリティの冗長度を変化させる．RS符号は，誤り訂正符号の1つで，データに冗長度を持たせ，ある程度のデータの欠落ならば，生

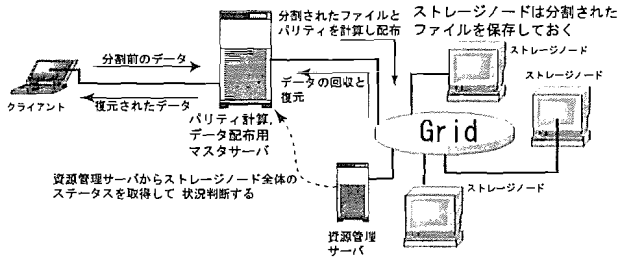

図1 システムの概念図
Fig. 1 Outline of the system.

存データからオリジナルのデータを復元することが可能である．ディクアレイなどで多用されるRAID4やRAID5[10]は，冗長度が1の誤り訂正符号だといえる．本システムでは，誤り訂正能力をRS符号によって拡張し，信頼性が低いノードにデータを格納する場合は，冗長性を高くしてデータを保全し，信頼性が高いノードの場合は冗長性を低くしてディスクの利用効率を高める．

図1に，各サービスが独立した状態の概念図を示す．アーカイブシステムは，ファイルの分割やRS符号の計算を行うマスタサーバ，データを格納するストレージノード，各ノードのディスク空き容量や空きメモリ状況を把握する資源管理サーバから構成される．

マスタサーバ

マスタサーバは，入力ファイルの分割とRS符号の生成，配布先ストレージノードの決定と格納したファイルに関するメタデータの管理を行う．マスタサーバは，主に次のような役割をする．

- 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．まずpingでの生存確認を行い，応答がなければストレージノードの候補の配列から削除する．次にGTKで必要となるGRAM（Globus Resource Allocation Manager，GRAMはGlobus上で資源管理を行うサービス）とGridFTPのtcpのポートが開いているかを確認する．
- 資源情報問合せ
  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する．
- RS符号のエンコードとデコード
  RS冗長符号を含む書き込みデータを生成する．また，読み出し時には，集めたファイル断片から元のファイルを復元する．
- ファイル断片の配布および回収
  書き込み時には，分割されたファイル断片および冗長化されたファイルをGridFTPを用いてストレージノードに配布する．読み込み時には，ストレージノードに保存されているファイル断片および冗長データを回収する．
- メタデータの管理
  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）の対応付けを行う．

資源管理サーバ

資源管理サーバでは，信頼性計算と資源情報情報の登録を行う．GTKのMDS（Globus Metacomputing Directory Service，グリッドにおけるマシン情報を提供するサービス）を使用して得られた情報をもとに，RS符号化の計算を行うノードや，ストレージホストのディスク空き容量を監視する．

ストレージノード

ストレージノードでは，GridFTPのサーバが待機しており，マスタサーバからの転送要求を待つ．

ストレージノードは，一般のユーザが利用しているワークステーションを想定している．これらのノードでは，不意のリブートや夜間の電源断などが発生しうる．ディスク故障ではないが，結果的にストレージノードに格納したデータが利用できない．そこで，障害以外の場合も含めて，格納データにアクセスできない状態を非稼働状態とする．

さらに，ストレージノードはグリッド上の計算ノードでもあるため，RS符号の計算ホストになる場合もある．

2.3 システムの動作

本システムでは，大別して，書き込み，読み出し，および信頼性の管理が行われる．

データの書き込みは，次の手順で行う．

(1) データを一定のブロックごとに分割する．分割したブロックを N 個のグループにする．
(2) RS符号に従って，冗長ブロックを生成する．この冗長ブロックのグループ数を M とおく．
(3) GridFTPを使用して，$(N+M)$ 個のストレージノードにデータを送り，書き込む．処理時間短縮のため，ストレージノードへの転送は並行して行われる．
(4) 論理ファイル名および物理ファイル名の情報を，マスタサーバのメタデータとして保存する．

処理の流れを図2に示す．また，信頼度の計算（M，N の決定）は，2.5節に示す手順によって決定される．

データの読み出し時には，ストレージノードに分散しているファイルを収集し，元のファイルに結合する．読み出しは，以下の手順で行われる．

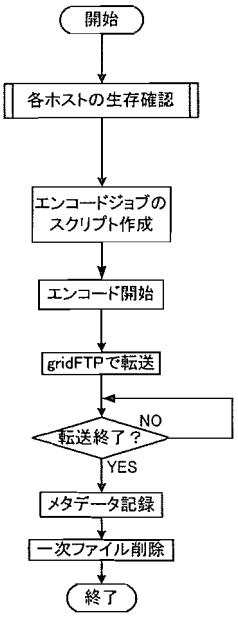

図 2 データ書き込み時の動作
Fig. 2 A flowchart to write data.

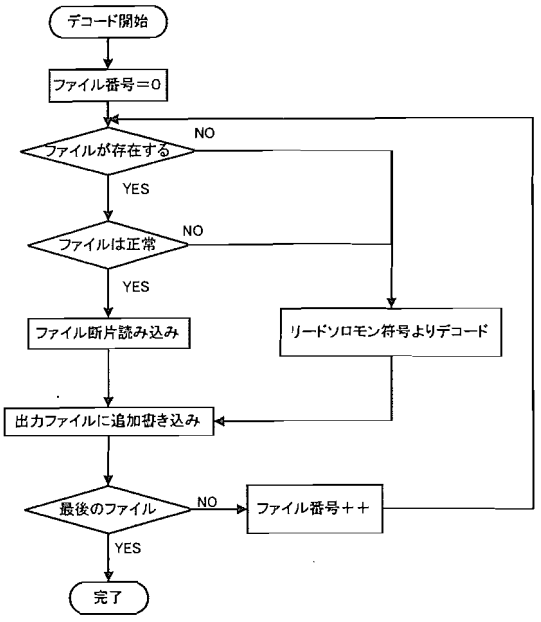

図 3 データ読み出し時の動作
Fig. 3 A flowchart to read data.

(1) マスタサーバのメタデータから，物理ファイル名および論理ファイル名を検索する．この結果，ファイルを保持しているホストおよびファイル断片が分かる．

(2) GridFTP によって，ファイルを保持しているホストからファイル断片を収集する．ファイルの収集は，複数のストレージノードから並行して行われる．

(3) 収集後，デコード処理を開始する．

デコード処理では，正常でないファイルや消失したファイルがあれば，冗長データを用いて元のファイルに復元をする．**図 3** に，デコード部分の動作を示す．

信頼性管理は，ストレージノードの可用率の変動に対しても，データの信頼性を確保するために行われる．本システムでは，RS 冗長符号を用いてデータの保護を行っているが，冗長度以上のストレージノードの障害には対応できない．そこで，定期的に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する．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，メタデータファイルを参照し，論理ファイルよりファイル断片が保存されているストレージノードを求め，保存されているファイル断片が以下のいずれかに該当すれば，配布したファイル断片が読み取り不可能と判断し，ファイルの復旧動作が開始される．

- ストレージノードが ping に応答しない．
- GTK のサービスが開始されていない．
- 保存されたファイル断片がストレージノード内で読み出せない．

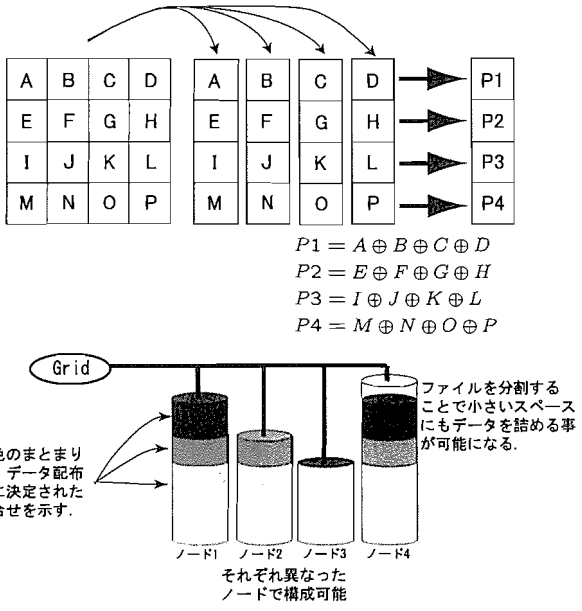

図 4 冗長データの生成と格納
Fig. 4 Redundant data synthesization and store.

ファイルの復旧を行う場合は，一度問題の論理ファイルを読み込み，再度書き込みを行う．読み込み時にはファイル断片の一部が消失しているので，RS 符号による復元が行われ，再書き込み時には新しい RS 符号の生成が行われる．

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，RS 符号に基づいて，分割したファイルの冗長性を確保する．**図 4** に冗長データの生成とデータ格納の概念図を示す．

RAID4/5 では，データを N 個のブロックに分割

し，このブロックのパリティを計算することによって，1つの冗長データを生成する．この方式では，冗長度が1つなので，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る（図4上段）．

本システムでは，RS符号を使用し，複数の冗長ブロックを生成することによって，複数台のディスク障害にも対応する．RS符号は巡回符号の1つで，バースト誤りに強いことで知られている．RS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[9]．

組織内グリッドを構成するストレージノードは，資源管理サーバによって利用可能ディスク容量を管理されるが，個々のノードの利用可能ディスク領域はバラバラである．そこで，各ディスクを等容量のチャンクに分割し，等容量のチャンクを集めてデータ格納領域とする．たとえば，図4下段の例では，色別にそれぞれ4台，3台，2台，1台のディスクからなるデータ格納領域となる．p 台のディスクからなるデータ格納領域は，分割数 N および冗長度 M の合計が p 以下である場合に利用できる．つまり，$N+M \leq p$ を満たすチャンク群を利用する．利用可能容量は，ファイルの格納結果や端末を利用しているユーザの状態によってつねに変化するので，チャンクの生成はデータの保存のたびに行われる．

2.5 冗長度の決定

本システムでは，システムの信頼性に応じてRS符号の冗長度を変化させる．ストレージノードの信頼性が高ければ，冗長度を低くし，冗長データを減らし冗長データ生成および保存のオーバヘッドを少なくする．ストレージノードの信頼性が低い場合には，冗長度を上げてデータの保護を優先する．このシステムの信頼性算出のもととなるのが，使用するストレージノードの稼働率である．個々のストレージの稼働率は，資源管理サーバが測定し算出する．資源管理サーバが，ストレージノードの生存確認を行うプログラムを一定時間ごとに実行し，順番にチェックする．この際に，各ノードのuptimeとdowntimeの積算状況を更新する．この情報より各ストレージノードのMTBF（Mean Time Between Failure）とMTTR（Mean Time To Repair）を算出し，システムの信頼性 R_{system} を以下の式によって求める．

$$R_{system} = \frac{MTBF}{MTBF + MTTR} \quad (1)$$

また，システムの信頼性モデルは簡単に表現できるようにm-out-of-nの並列システムとして計算した．

ここで問題となるのが，それぞれ異なる稼働率のサブシステムから構成される並列システムの場合，故障の起きるパターンをすべて計算する必要があるために，構成するサブシステムが多数になった場合（数百台の規模であっても）に，計算量が非常に大きくなってしまうことである．この問題の解決方法として，ある程度の稼働率でクラス分けをし，その代表値に統一してシステム信頼性を計算をする．各サブシステムの稼働率を a とするとき，n 台のサブシステムのうち i 個が故障する確率は，${}_nC_{n-i} \cdot a^{n-i} \cdot (1-a)^i$ である．したがって，生存台数を m とすると，システム信頼性 R_{system} は，

$$R_{system} = \sum_{i=0}^{n-m} ({}_nC_{n-i} \cdot a^{n-i} \cdot (1-a)^i) \quad (2)$$

と簡素な式で計算することが可能となる．

本システムでは，クラス分けには稼働率に応じて3つのクラスと，使用不可のクラスに分ける．クラス分けはノードの稼働率 R_{node} をもとに行う．

- $R_{node} > 99.99\%$（Highest priorityとして使用）
- $99.99\% > R_{node} \geq 99.9\%$（Higher priorityとして使用）
- $99.9\% > R_{node} \geq 99\%$（Normal priorityとして使用）
- $R_{node} < 99\%$（容量が不足するまで使用しない）

このクラス分けのフローチャートを図5に示す．99.99%の稼働率は，年間1時間弱のシステムダウンであり，たとえば情報センターなどのシステムをノードとして利用することを想定している．99.9%の稼働率は，数カ月に数時間のシステムダウンであり，たとえば研究室内のサーバなどがこの稼働率であることを想定した．99%の稼働率は，月間7時間（1週間2時間）程度のシステムダウンであり，一般の個人用ワークステーションを想定している．この稼働率については，筆者らの所属する北陸先端科学技術大学院大学のワークステーションの稼働率から仮定した．343台のシステムの稼働状況を2006年のある4日間について調べたところ，180,415回のpingに対して1,010回の不応答が観測された．つまり，この間平均して約0.56%の利用不可能状態が出現したので，稼働率としては99%以上が期待できる．

データの書き込み時に，マスタサーバは，クラス分けされたストレージノードを，信頼性の高いクラスから順番に使用する．使用ストレージノードの偏りを防ぐために，プログラムの開始時にクラス内の順番をシャッフルして利用する．そして，使用するノードの分だけ配列から1つずつチェックしながら取り出す．

このとき，データの分割サイズは400 MBずつに分割し，冗長度を2と設定して信頼性計算を行う．もし，要求する信頼性を満たせない場合には，冗長度を1上げてシステムの信頼性計算を再実行する．使用する予定であったクラスで，ストレージノードが不足した場合には，次の信頼性クラスを使用して，再度ストレージノードの選択およびシステム信頼性計算を実行する．システム全体のデータ信頼性は99.999%とした．**表1**には，式(2)を用いて，$R_{syste} \geq 99.999\%$ を達成するために必要な冗長度を示した．

2.6 システムの実装

実装に使用した言語はCとPerlを使用した．RS符号や組合せ計算の処理部は，C言語で記述されている．RS符号を生成するライブラリ[11]をもとに，コードの一部（ファイル入出力部分など）を改変して使用した．ジョブ生成用のスクリプト書き出し，ファイルの分散配置の準備，システム信頼性計算などの制御は，Perlによって記述された．

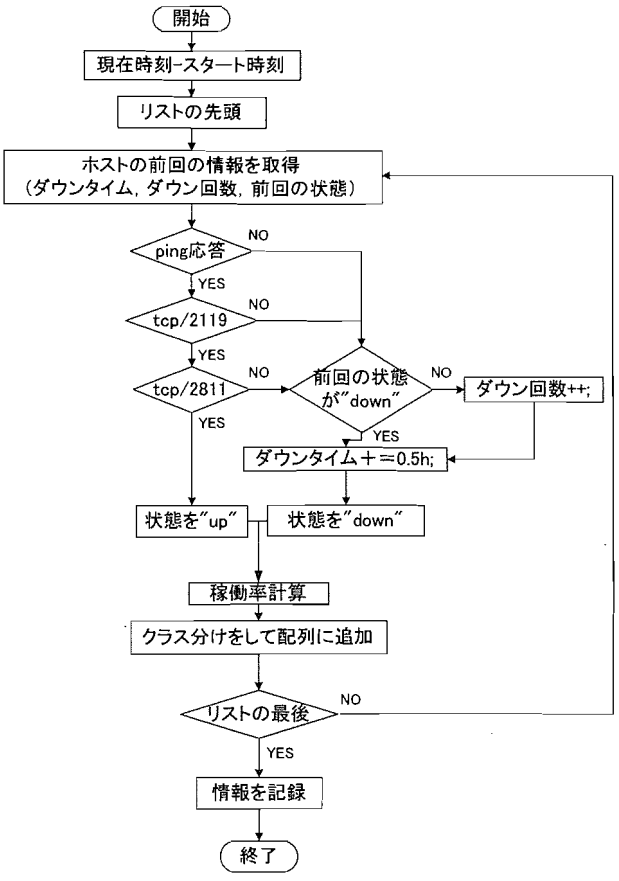

図5 稼働率チェックのフローチャート（30分ごとに監視する場合）
Fig. 5 A flowchart to check availability (Check every 30 minutes).

表1 99.999%の R_{system} を達成するのに必要な n と m の例
Table 1 A sample value of n and m to satisfy $R_{system} > 99.999\%$.

$R_{node} = 99.99\%$	$n = 10, m = 9, R_{system} = 99.99996\%$
$R_{node} = 99.9\%$	$n = 10, m = 8, R_{system} = 99.99999\%$
$R_{node} = 99.\%$	$n = 10, m = 7, R_{system} = 99.99980\%$

マスタサーバが1台だけでは，故障した場合や負荷の集中が起きることが予想される．このため，本システムでは2つの動作モードを使い分けることにした．マスタサーバが自分自身でパリティ生成などをすべて行う『自己処理モード』と，マスタサーバがグリッド内の他のマシンにパリティ生成などを依頼し，そこからGridFTPの第三者転送で分散配置を行う，『外部処理モード』である．外部処理モードは，マスタサーバが扱う元ファイルサイズと空き物理メモリ容量を比較し，マスタサーバのメモリが不足する場合には，グリッドの中を検索し，メモリに余裕のあるホストに処理を依頼する．この手法をとることで，冗長度計算のオーバヘッドを背負うホストが分散される．しかし，どのホストからもアクセスできるディレクトリに元ファイルを置いておく必要があり，現状ではNFSでマウントされたディレクトリにファイルを格納し，それを処理担当になったホストが読み出す．

3. システムの性能評価

3.1 実験システムの構成

システムの性能を評価するために，**表2**に示す機器から構成される実験環境を構築した．各端末は学内LANを通じてGigabitEthernetで接続されている．また，これらのコンピュータ，ネットワークは通常のユーザ環境で行われ，各端末のユーザは通常どおりにネットワークやコンピュータを使用している状況で実験を行った．

提案システムの目標要件としては，(1) RS符号による信頼性が確保でき，符号生成や障害時のデータ復旧の際に実用上十分な速度で処理できること，(2) テープベースの大容量ストレージシステムに劣らないアクセス性能を達成すること，の2つである．

3.2 システムの基本性能

3.2.1 ストライピングによる性能

まず予備実験として，データのストライピング転送

表2 実験環境に用いた機器の仕様
Table 2 Specification of machines for experimentations.

機種	Sun Blade 1500	自作
CPU	Ultra SPARC IIIi	Opteron
	1 GHz × 1CPU	2.2 GHz × 2CPU
メモリ	512 MB	8 GB
OS	Solaris 8	SuSE Linux 9.0 Pro
NIC	GbEthernet	GbEthernet
台数	8台	1台

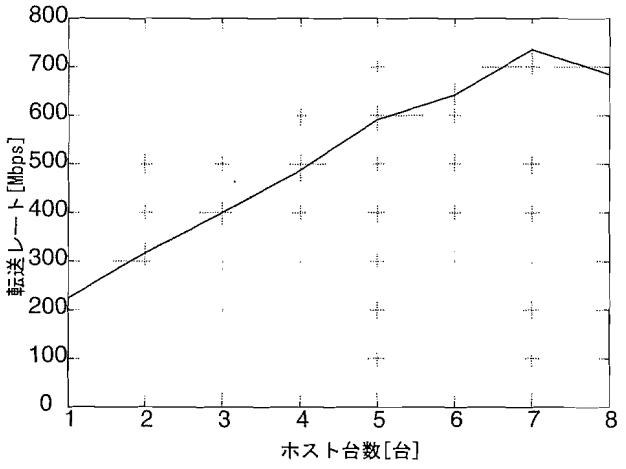

図 6 ホスト数と転送速度の関係
Fig. 6 Number of hosts and transfer speed.

における IO 性能向上効果について調査した．表 2 に示す自作 Opteron マシンから複数台の Blade 1500 に，rcp コマンドを用いて並行に書き込み動作を行ったときの転送速度を図 **6** に示す．図より，GigabitEthernet の LAN 内での転送速度は，740 Mbps 程度まではスケーラビリティが確保されることが分かった．ネットワーク性能測定ツール netperf[12] での帯域測定においても，750〜800 Mbps 程度が上限だったので，妥当な値であると考えられる．8 並列の場合の性能低下は，ディスクのランダムアクセス性能の限界と，転送チャネルどうしの帯域の取り合いなどが要因で起きたものと考えられる．

3.2.2 RS 符号のエンコード，デコードの性能

次に RS 符号のエンコード，デコードに要する時間を測定した．図 **7** に，2,000 MB 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示す．図より，RS 符号生成の処理時間は冗長度の設定に比例して増加すること，およびおよそ 20〜100 秒程度であることが分かる．

図 **8** に，ファイルサイズに対するエンコード時間を示す．冗長度の構成はデータブロック 5 に対して冗長度 2 である．エンコード時間は，元のファイルサイズに比例し，RS 符号の計算時間は，およそ 10 秒〜40 秒である．

3.3 自己処理モードでの性能評価

グリッドを構成し，自作 PC（Opteron × 2CPU）における自己処理モードでの性能評価を行った．自己処理モードでは，ローカルファイル領域に保存してあるファイルを自己でエンコード処理担当し，グリッドへと分散配置する．図 1 におけるマスタサーバとして，表 2 中の自作 Opteron マシン，ストレージノードとして Blade 1500 を用いた．

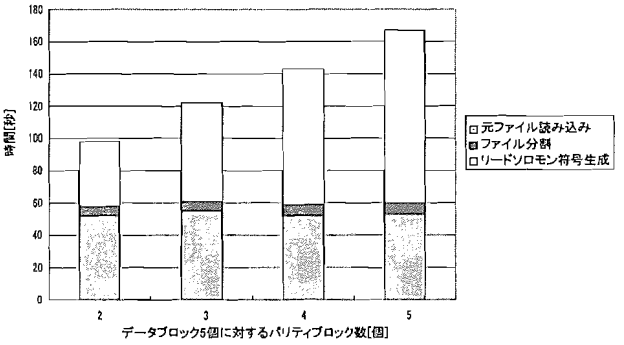

図 7 冗長度とエンコードにかかる時間の関係
Fig. 7 Degree of redundancy and time for encoding.

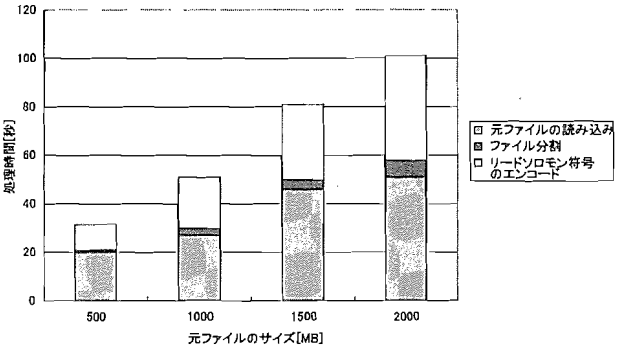

図 8 ファイルサイズに対するエンコード時間
Fig. 8 File size and time for encoding.

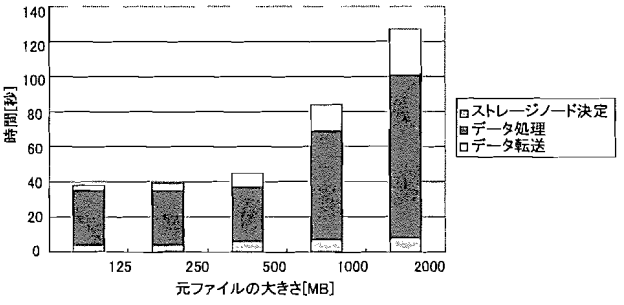

図 9 自己処理モードにおける書き込み時間
Fig. 9 Write time of self processing mode.

3.3.1 データの書き込み

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 RS 符号の生成，および GridFTP によるデータの転送を行う．実験結果を図 **9** に示す．凡例の『ストレージノード決定』は信頼性計算と生存確認を行い，配布先ストレージノードを決定する時間，『データ処理』は元データの読み込みから冗長データの生成と書き出す時間，『データ転送』は grid への転送の時間である．これらは以下の実験でも同様のものとする．

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分かる．データの処理には 125 MB から 500 MB のいずれも 30 秒程度の時間を要している．これは，データ処理のジョブ

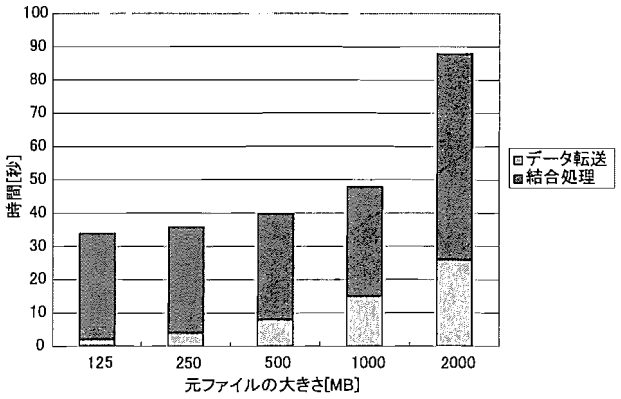

図 10 自己処理モードにおける読み出し時間
Fig. 10 Read time of self processing mode.

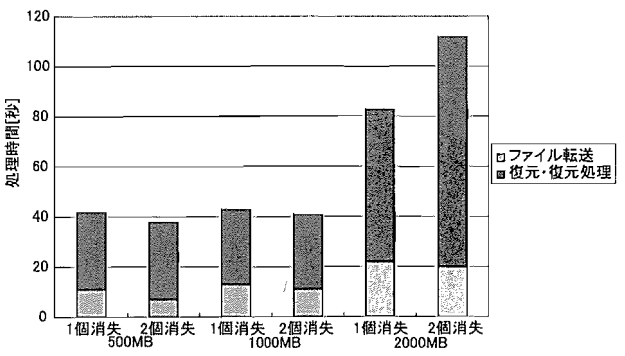

図 11 自己処理モードにおける縮退運転時の読み出し時間
Fig. 11 Read time for self processing mode at disk failure.

終了検出を globus-job-status によって行っているが，ポーリングによって検出しているため，ジョブ終了を検出する時間に誤差があるためだと考えられる．ジョブ検出時間自体の誤差と，コマンド実行タイミングによって 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 の転送速度が上がりきる前に転送が終了するからだと考えられる．

3.3.2 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，ストレージノードに異常がある場合について実験を行った．

3.3.2.1 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 125 MB，250 MB，500 MB，1,000 MB，2,000 MB の 5 つである．これらのファイルのデータブロック，冗長ブロックをストレージノードから回収し，元のファイルを復元した．故障がない場合には，すべてのファイル断片を並列に転送するものの，転送後の処理としては RS 符号による復元は必要ないため，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 **10** に示す．凡例の『結合処理』は，ファイル断片の結合および書き出し時間である．

データ転送時間に比べ，復元・結合時間が長時間要しているが，これはデータ転送の場合は複数のストレージノードから並列に転送が行われるのに対し，結合後のデータは単一の書き込み動作になるためだと考えられる．データが正しく復元されていることは，md5sum コマンドでハッシュ値を計算し確認した．

3.3.2.2 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．ファイルの大きさは同様に 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 **11** に示す．ファイル断片には，データ本体の一部が含まれる断片と，RS 符号による冗長データを含む断片がある．ここではデータ本体の一部が含まれる断片を消去した．ファイルの読み出し時にデータ本体の一部が読み出せないと，RS 符号によるデータの修復が開始される．一方，データ本体が含まれる断片が揃っていて冗長データの断片のみが読み出せない場合は，RS 符号によるデータの修復が行われず，読み出し時間としては故障が存在しないときと同じになる（別途，RS 符号の再生成は必要である）．

ファイルの転送順序には，(a) 一度に冗長データも含めて全ファイル断片を転送する方法と，(b) 最初にデータ本体を含むファイル断片のみを転送し，ファイル断片の欠落や異常があったときに初めて冗長ファイルを転送する方法，の 2 つが考えられる．このうち，本論文では，(a) の方法を採用した．(b) の方法では，ファイル断片の欠落や異常時に，GridFTP によって並行に行われる正常ファイル断片の転送の後に冗長ファイルの転送が開始され，ペナルティが大きくなる問題が予測されるからである．

本実装では，消失ファイル断片があると，総量として転送ファイル容量が少なくなるため，消失ファイル数が多ければ転送時間が短くなる．一方で RS 符号による復元が必要になるが，500 MB および 1,000 MB では，復元処理時間はファイル消失の数にはほとんど影響されないことが分かる．2,000 MB では消失ファイルが 2 個のとき，復元処理の時間が増加しているが，これは後述するメモリ容量の不足によってスワップが

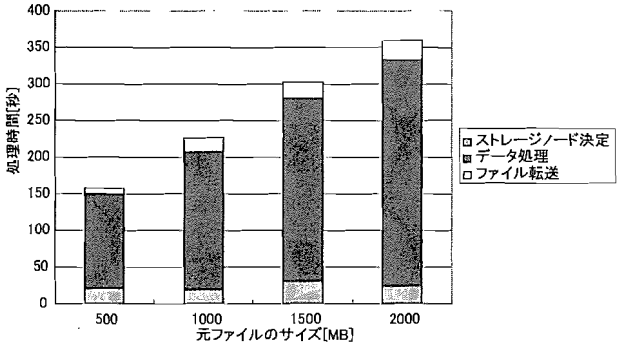

図 12 外部処理モードにおける書き込み時間
Fig. 12 Write time of global processing mode.

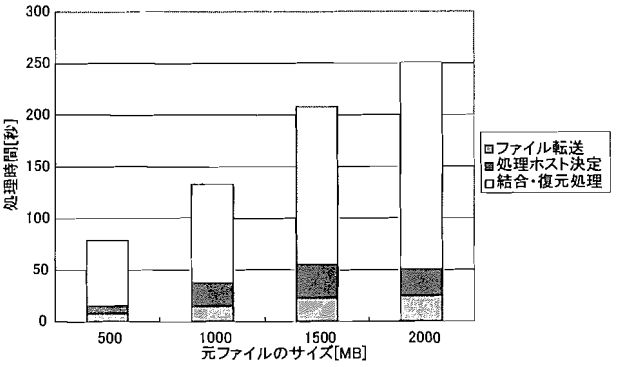

図 13 外部処理モードにおける読み出し時間
Fig. 13 Read time of global processing mode.

多発しているためと考えられる．転送時間と復元時間の合計を考えると，消失ファイルが多い方が全体の処理速度は向上する結果となった．

1つのファイルをRS符号を使用して復元する時間は，5秒から10秒程度と短い．しかしながら，GTKのジョブ終了判定の制約によって，実行時間が30秒以下の場合には，約30秒と検出されてしまっている．ストレージノードに分散配置しておいたファイルが消失している場合，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．しかし，消失ファイルが多ければRS符号によって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する．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．

3.4 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．自己処理モードでは，マスタノードでファイル分割およびRS符号生成を行ったのに対し，外部処理モードでは，ファイル分割およびRS符号生成を行うジョブをGlobusを通じて投入し，Globusによって選択されたノードでファイル分割およびRS符号生成が行われる．自己処理モードに比べて，応答性は低いが，より高い性能のホストが選択される可能性が高く，容量の大きなファイルの処理に適する．

3.4.1 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グリッド内のプログラム実行可能マシンの空きメモリを比較し，どのホストでRS符号を生成するかを決める．結果を図 **12** に示す．

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分かる．これは，元ファイルをRS符号処理を行うノードに転送するための時間が大幅に増えているからだと考えられる．

3.4.2 データ読み出し

次に，ストレージノード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．

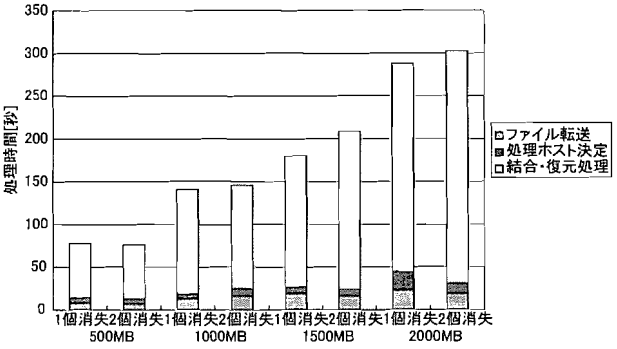

図 14 外部処理モードにおける縮退運転時の読み出し時間
Fig. 14 Read time for global processing mode at disk failure.

3.4.2.1 故障が存在しないとき

分散配布したファイルが完全に揃っている状況でテストを行った．結果を図 **13** に示す．

グラフより，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分かる．書き込み時と同様に，復元されたファイルの転送・書き込みが1つのノードに集中するため，この時間が大きくなっている．

3.4.2.2 縮退運転時

自己処理モードと同様に，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，ファイル復元にかかる時間について測定を行った．結果を図 **14** に示す．

自己処理モードと同様に，縮退運転では転送ファイルの総容量が少なくなるため，消失ファイル数が多ければ転送時間が短くなること，一方で復元時間が大きくなることが確認できた．ここでも，ファイルサイズが小さい場合には，RS符号による復元時間は5秒～10秒程度と短い．この程度の処理時間の差は，GTKがジョブ終了を検出する際のタイムラグによって吸収されてしまう．したがって500 MB，1,000 MB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．

3.5 考　　察

冗長データ生成にかかるオーバヘッドは，図8と図9を比較することによって推測できる．GTKの使用の有無など測定条件が異なるので厳密には正確ではないが，図9の“データ処理”部分が図8に相当するといえる．図9におけるデータサイズが小さい場合は，グリッドを使ったことによる誤差が大きいことは前述のとおりである．図9から(1)ストレージノードの決定はわずか，図8から(2)RS符号生成時間は全体のおよそ4割程度，であることが分かる．また，ストレージノードの決定は，どのファイルサイズでも数秒以内である．

ファイル復元の場合，図10は，断片ファイルの単純な結合に相当する．RS符号によるデータ復元のオーバヘッドは，図10と図11を比較することによって推測できる．特に2,000 MBの場合を比べると，1個消失の場合はRS符号による復元の時間よりも，転送ファイル量が減ったことによる効果が大きい．2個消失の場合でも，復元のために要する時間の増加は1割程度である．

グラフには現れていないが，RS符号の生成/復元に必要なメモリ量は，扱うファイルサイズに比例する．今回の実装では，元データをN分割した後，M冗長のRS符号を生成しているので，RSエンコーダのワークセットがファイルサイズに比例するためである．本実験では，グリッド内の最大のメモリサイズが8 GBだったので，2 GBまでのデータしか評価することができなかった（それ以上のファイルサイズだと，スワップが極端に多くなり，評価の正当性が失なわれる）．評価結果より，本手法は明らかにファイルサイズが大きい領域で有効であるが，そのためには，グリッド内に大容量メモリを持つノードを設けるか，あらかじめ元ファイルを一定の大きさのサブファイルに分割した後，RS符号を生成する必要がある．

外部処理モードでは，メモリの問題があるため，結果として8 GBのメモリを持つ自作Opteronのノードで RS符号生成/復元処理が行われている．しかし，8 GBのメモリを持つホストでダミーのメモリ消費プログラムを実行するなどの有効メモリ量を減らす措置を行うと，スワップにより処理効率が大幅に低下するものの，RS符号化処理が他のノードに移ることを確認した．

提案システムでは，最短実行時間が約30秒となっており，決して短かくはないが，テープベースの大容量ストレージシステムでキャッシュなどの助けを借りない場合に必要な，テープからのアクセス時間（2〜3分）よりは高速なアクセスが可能となっている．提案システムを用いてファイルシステムなどを構築する場合，PetaSite[1]で用いられたようなディスクキャッシュと組み合わせることも可能である．

今回の実装では外部処理モードで復元ファイルを書き込むときにNFSを用いているが，これをGridFTPにすることによって，さらなる高速化が期待できる．予備的な実験ではおよそ4割前後の転送時間の削減が確認されている．

表3 ディスク利用効率の比較
Table 3 Comparison of disk utilization.

	レプリカ	冗長度1	冗長度2
4〜15台使用時に利用可能な%	50(%)	75〜93(%)	50〜87(%)
冗長度	二重化	N+1	N+2

3.6 データの保守，再生成

いくつかファイルをストレージノードへ分散配置し，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，再度分散配置し直す．

データの確認に要する時間は，元ファイル1つにつき15秒から20秒程度の時間である．ファイルに異常がある場合には，これに引き続いて，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．

3.7 記憶領域利用効率

本システムでは，ストライピングとRS符号による冗長データを使用することにより，耐障害性確保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3に示す．

この表には計算例の一部だけを掲載したが，レプリカ方式に比べてRS符号による冗長方式を採用したほうが，ディスク使用効率が良いことが分かる．使用するディスク台数が多ければ，ディスク利用効率をさらに向上させることが可能である．

4. 関連研究

Grid Datafarm[5]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，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．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[13] に特徴があり，全体としてみたときにファイルアクセスバンド幅を大きくとることができる．Grid Datafarm は，負荷分散とデータ保全性確保のために，レプリカで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．

Europian Data Grid Project（以下 EDG）[6] は，CERN（欧州合同素粒子原子核研究機構）によって研究が開始された研究で，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．特に，データマネージメント技術に重点を置いた設計となっており，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．ファイルの保全性の確保のためには，複数のサイト間でのファイルのレプリケーションで対応している．

Grid File System[7] は，グリッド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 Globus-Toolkit に含まれる GSI（Grid Security Infrastructure），SFS（self-certifying file System）である．通信のセキュリティを確保するため，SFS は NFS の転送を暗号化し，認証データを付加して転送するが，通信を暗号化するためにオーバヘッドが大きく，通信路に 1 Gbps 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．

Data Reservoir[8] は東京大学と（株）富士通によって開発されている，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分ける方式である．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あげられる．

これらの先行研究では，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が，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，構成要素となる端末の使われ方の差が大きいと考えられる．研究機関などで使用される大型システムの場合，動作が非常に安定しているうえに常時電源を投入する使い方が一般的であり，いくつかのレプリカを設けることによって，データの信頼性が十分確保できる．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グリッドの構成要素として利用する場合には，ユーザ自身が自由に電源を切ることもでき，搭載ディスクの物理的な信頼性も均一ではない．また，レプリカによる構成では，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低い．

複数のディスクによる信頼性確保に関する研究として，市川らによって提案された連鎖ネットワーク RAID[14] がある．この RAID 機構は，ディスク間で連鎖的に XOR をとることによって，データの信頼性を保持することを狙っている．この研究では，冗長度とシステム全体の信頼性について詳しい議論がなされており，冗長度を自由に設定できる利点がある．手法は異なるものの本研究の目的に利用可能であるが，ストレージノードの可用性の自動管理にまで踏み込んだ議論はなされていない．

本研究では，(1) ストレージノードの可用性を自動管理し，一定の信頼性のあるボリュームグループを作ること，(2) ボリュームグループの信頼性に応じた冗長度でデータを格納し，システム全体としての保全性を確保する点，の 2 点において，先行研究と異なる試みを行った．

5. おわりに

本論文では，一般ユーザ環境で組織内グリッドを構築する際の信頼性に着目し，構成ノードの稼働率に応じて動的にパリティの多重度とデータ分散先を変化させる分散ストレージシステムについて提案した．本システムでは，いくつかのデータを消去した後，データの再生が可能なことを確かめた．また，実際の運用では偶発的な障害も起きたが，データの再生成プログラムによって再生されたデータが，異なるノードへと再配置され無事であった．動作速度に関する評価を行ったところ，RS 符号によるオーバヘッドは，書き込み時で約 4 割の増加，読み出し時は消失ファイル数にもよるが，おおよそ 1 割程度であることが分かった．

今後の課題としては，一時ファイルを使用せずに，処理の高速化を行うことや，データ自体の保全性（Mean Time to Data Loss）を算出し，より詳細なモデルを構築してシステム全体の信頼性を上げるための改良に取り組みたい..

参考文献

1) http://www.sony.jp/products/Professional/DataArchive/products/sait_lib/
2) http://www.adic.com/amss/
3) Foster, I. and Kesselman, C.: *The GIRD Blueprint for a New Computing Infrastructure*,

Morgan Kaufmann Publishers, Inc. (1999).
4) Foster, I. and kesselman, C.: *The GRID2 GIRD Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, Inc. (2004).
5) 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 (Jan. 2002).
6) Ghiselli, A.: DataGrid Prototype 1, *TERENA Networking Conference*, pp.3–6 (June 2002).
7) Takeda, S., Date, S. and Shimojo, S.: 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, *Proc. SC2002* (CDROM) (Nov. 2002).
9) 江藤良純，金子敏信：誤り訂正符号とその応用，テレビジョン学会（編），オーム社 (1996).
10) 宇野俊夫：ディスクアレイテクノロジ RAID，エーアイ出版 (2000).
11) GFLIB — C Procedures for Galois Field Arithmetic and Reed-Solomon Coding. http://www.cs.utk.edu/~plank/plank/gflib/
12) http://www.netperf.org/
13) 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 (May 2003).
14) 市川俊一，高橋克巳：高信頼，高可用な分散ストレージを実現する連鎖ネットワーク RAID，情報処理学会/電子情報通信学会，pp.99–106 (2005).
15) Stockinger, H., Dono, F., Laure, E., Muzzafar, S., et al.: Grid Data Management in action, *Computing in High Energy Physics (CHEP 2003)* (2003).
16) Bonnie. http://www.textuality.com/bonnie/

(平成 17 年 10 月 5 日受付)
(平成 18 年 2 月 8 日採録)

井口 寧（正会員）

1991 年東北大学工学部機械工学科卒業．1994～1997 年日本学術振興会特別研究員．1997 年北陸先端科学技術大学院大学情報科学研究科博士後期課程修了．現在，同大学情報科学センター助教授．また，2002 年から科学技術振興事業団さきがけ研究 21（機能と構成）に参加し研究に従事．この間並列システム，ウェーハスタック集積システムに関する研究を行う．IEEE，電子情報通信学会各会員．

渡辺 浩二

2003 年東京理科大学基礎工学部電子応用工学科卒業．2005 年北陸先端科学技術大学院大学情報科学研究科博士前期課程修了．現在，株式会社日立製作所 RAID システム事業部．この間，ストレージ関連に関する研究を行う．

松澤 照男（正会員）

1948 年生．1973 年信州大学大学院工学研究科修士課程修了．同年信州大学医学部助手．1986 年沼津工業高等専門学校助教授．1991 年北陸先端科学技術大学院大学助教授．1995 年同教授．数値流体力学における並列計算の研究に従事．医学博士．日本機械学会，日本数値流体力学会，日本流体力学会等各会員．