情報処理学会論文誌 Vol. 49 No. 6 2106–2117 (June 2008)

PCクラスタを用いた高信頼・高可用ブロックレベル・ストレージの設計と実装

市川俊一[†1] 豊田真智子[†1] 高橋克巳[†1]

オンラインストレージサービスのストレージサブシステムを高機能かつ安価に構築することを目的に，高信頼なブロックレベル・ストレージをPCクラスタを用いて構成するためのデータ冗長化方式である連鎖ネットワークRAIDを提案し，その設計と実装について述べる．提案手法は複数のPCノードの同時故障に耐えうる高い信頼性を持つことと，PCノードの追加などの構成の変更に柔軟に対応できることを特長とする．シミュレーションを用いた評価により，本手法が100台中10台が停止か故障した状態でも5 nines以上の高い可用性を実現することを示す．また，プロトタイプ実装を用いた評価実験により，本システムがPCノードの台数に比例したスループットを実現することと，70 K[IO/sec]の処理性能を持つことを示す．

Design and Implementation of Highly Reliable and Highly Available Block-level Storage Using PC Cluster

Toshikazu Ichikawa,[†1] Machiko Toyoda[†1] and Katsumi Takahashi[†1]

To compose the high reliability block-level storage using PC cluster for building the cost effective and sophisticated storage subsystem of the online storage service, we propose the method to build the redundant array of PC nodes, named as Chain Network RAID, and describe its design and implementation. Our proposal features the reliability to be able to sustain the concurrent failures of multiple PC nodes, and the flexibility to adapt the modification of composition such as the addition of PC nodes. The results of our simulation shows that our proposal achieves the availability of 5 nines and more when 10 out of 100 PC nodes are stopped or failed. The evaluation with the prototype implementation shows that our system achieves the throughput in proportion to the number of PC nodes, and provides up to 70 K IOPS capability.

†1 日本電信電話株式会社 NTT 情報流通プラットフォーム研究所
NTT Information Sharing Platform Laboratories, NTT Corporation

1. はじめに

計算機によって扱われるデータ量が年々増加するなか，ユーザは用途に応じて複数のPCを使い分けるようになっており，それらに格納されているデータを管理する手間がユーザの負担となっている．また，広帯域なネットワークが普及するなかで，データを管理する手段としてオンラインストレージサービスの有効性が高まっている．

オンラインストレージサービスを提供するうえで，高い信頼性を持つシステムを安価に構築することが重要となる．高い信頼性を持つとは，ストレージが利用できる期間の割合が高く，ユーザのデータが失われるまでの期間が長いことを意味する．こうしたシステムを構築する手段として，コモディティなハードウェアを活用する方法が注目されている．PCをEthernetやInfiniBandなどの高速なネットワークで結合したPCクラスタを用いてストレージサブシステムを構成する分散ファイルシステムがすでにいくつか実用化されている[1)–3)]．

コモディティなハードウェアでシステムを構成する場合，問題となるのは個々のデバイスの信頼性の低さである．それを補うためにシステムで冗長化の仕組み提供し，ストレージシステムとして高い信頼性を実現することが課題となる．これまで最下位レイヤにあるデバイスが担っていた信頼性という特性を上位レイヤにあるソフトウェアが担う場合，そのソフトウェアがどのようなインタフェースを提供するかによって，システム全体の階層構成に与える影響が異なる．そのインタフェースが下位レイヤのものであるほど，上位レイヤの資産をそのまま継承することが可能となる．すでに実用化されている高信頼化機能を持つクラスタ・ストレージはファイルレベルのインタフェースか用途を限定した特殊なインタフェースしか持たない．これまでデバイスが提供したものと同じブロックレベルのインタフェースがあれば，ボリュームの仮想化機能[4),5)]，ファイルシステム[6)]や他のクラスタ・ストレージ[7)]といった既存の資産を組み合わせて活用することが可能となる．たとえば，任意の時点でのスナップショットをとる機能[5),6)]と組み合わせることで，ユーザの操作ミスによって生じるデータの損失を防ぐ機能を効率的に提供することが可能となる．

また，オンラインストレージサービスは需要予測が困難であり，過剰な設備投資は利益率を圧迫する．スモールスタートでサービスを開始した後に，需要の伸びに応じてシステムを増強していけることが重要となる．そして，運用にかかるコストを圧縮するために，故障したPCノードの交換や増強のためのPCノードの追加などのシステム構成の変更を柔軟に行えることが重要となる．さらに，PCノードの数に応じた高いパフォーマンスが発揮され

る，インクリメンタルなスケーラビリティを実現することが求められる．

すなわち，本研究の要求条件として次の 3 つがあげられる．1 つ目はユーザのデータを失わないために高いデータの信頼性を実現することである．2 つ目はシステム構成の変化に柔軟に対応できることである．3 つ目はインクリメンタルなスケーラビリティがあり，システム全体で高いスループットを実現できることである．こうした要件をふまえ，ブロックレベルのインタフェースを持つ PC クラスタ・ストレージの冗長化方式について検討する．

2 章では，ストレージ高信頼化技術として広く用いられている RAID などの既存手法について述べる．3 章では，クラスタ・ストレージのための冗長化方式として，連鎖ネットワーク RAID を提案し，その設計について述べる．4 章では，シミュレーションを用いて提案手法の信頼性を評価し，また，プロトタイプ実装を用いた実験によりスループットとスケーラビリティについて評価する．5 章では，関連する研究について述べ，6 章では，まとめを述べる．

2. 既存手法

複数のディスクドライブを束ね，ストレージサブシステムを高信頼化する手法として RAID[8] が広く用いられている．しかし，ストレージサブシステムの規模が大きくなりディスクドライブの数が多くなると，RAID グループの再構築中に他のディスクドライブの故障が発生する確率が無視できなくなり，データの信頼性について問題が生じる．

FARM[9] と Clustered RAID[10] は障害発生時のアレイの再構築にかかる時間を短くすることで，データの信頼性を高めている．FARM はドライブの故障が発生したとき，故障により一時的に失われたデータを再構築するために必要なデータを，たくさんのドライブの余分な記憶領域にコピーして分散させている．また，Clustered RAID は，RAID グループを構成するメンバ数より多くのノードを準備し，データをそれらのノードに分散して配置しておくことで，ドライブの故障が発生したとき，再構築によってかかる負荷を分散させている．これらの工夫により FARM と Clustered RAID は，再構築時のアレイのパフォーマンス低下を防ぐとともに，再構築にかかる時間を短縮させて，データの信頼性を高めている．しかし，複数のドライブが同時に故障すると，データが失われてしまう．

RAID6[11] と Row-Diagonal Parity[12] は，耐えうる故障の台数を 1 つ増やすことで，データの信頼性を高める．RAID6 はガロア体の多項式演算でパリティを生成することで，RAID グループ内に 2 種類のパリティを持つ．Row-Diagonal Parity は，RAID4 と RAID5 と同じようにストライプ上のブロックから Exclusive OR 演算でパリティを生成するのに加え，対角線上のブロックから Exclusive OR 演算でパリティを生成することで，RAID グループ内に 2 種類のパリティを持つ．RAID6 と Row-Diagonal Parity は 2 種類のパリティを持っているため，2 台までのドライブの同時故障に耐えることができ，データの信頼性が高まる．しかし，PC クラスタを用いてストレージを構成するには，PC ノードの故障のほかに，OS のアップデートなどシステム保守のために計画的な停止が発生することも想定しなければならない．そのため，もっと多くのドライブの故障と停止に対応する必要がある．

消失訂正符号の Reed-Solomon 符号を用いることで上記の方式に比べて高い信頼性が実現できることが知られている[13]．RAID6 の多項式演算を一般化したものであり，任意個のパリティを生成することができ，高いデータの信頼性を実現できる．Reed-Solomon 符号を用いて多くのドライブを扱う場合は，RAID グループのサイズと数が重要なパラメータとなる．しかし，RAID グループの数は柔軟に変更することができないため，システムの拡張性と柔軟性に乏しいという問題がある．この点の詳細については 4.1 節で述べる．

3. 提案手法

我々はブロックレベルの PC クラスタ・ストレージを実現するためのデータ冗長化方式として，連鎖ネットワーク RAID を提案する．連鎖ネットワーク RAID は低密度パリティ検査符号[14]（以下，LDPC 符号と呼ぶ）のビットに PC ノードを対応づけた形の冗長化方式であり，PC クラスタで 1 つの符号語を形成するボリューム構成法である．本手法は複数の PC ノードの同時故障に耐えうる高い信頼性を持つこと，読み書き可能なブロックレベルのストレージ・インタフェースを提供すること，また，柔軟に PC ノードの追加や構成の変更を行うことができることを特徴とする．以下，本章では連鎖ネットワーク RAID の設計と実装について述べる．

3.1 ボリューム管理

連鎖ネットワーク RAID は，PC ノードが保持する物理ボリュームから高信頼化された論理ボリュームを提供する．本手法は次の 3 種類のボリューム操作で RAID を構成する．

- 物理ボリュームの追加と削除
- 論理ボリュームの追加と削除
- 物理ボリュームと論理ボリュームの結びつけ

まず，物理ボリュームの追加によって，PC ノードが保持する物理ボリュームが RAID の構成要素として認識される．次に，論理ボリュームの追加によって，ホストに提供される論理ボリュームが RAID に定義される．ここで論理ボリュームは最低でも 1 つの物理ボリュー

ムを占有するという制約を設ける．占有された物理ボリュームをデータ用物理ボリュームと呼ぶ．データ用物理ボリュームにはその論理ボリュームのデータが格納される．論理ボリュームの追加時には，データ用物理ボリュームを指定し，そのデータが論理ボリュームの初期データとなる．そして，論理ボリュームと物理ボリュームの結びつけによって，論理ボリュームと物理ボリュームの間に多対多の関係が定義される．このとき複数の論理ボリュームが結びつけられた物理ボリュームは，それら論理ボリュームのデータの Exclusive OR（以下，XOR）演算結果を保持する．XOR 演算結果を保持する物理ボリュームをパリティ用物理ボリュームと呼ぶ．論理ボリュームの追加時に指定された物理ボリュームも，その論理ボリュームに結びつけられたものとして管理される．すなわち，論理ボリュームは 1 つ以上の物理ボリュームと結びつけられ，そのうち最低でも 1 つはデータ用物理ボリュームである制約が課される．論理ボリュームは複数のデータ用物理ボリュームと複数のパリティ用物理ボリュームとの結びつけを持つことができる．

連鎖ネットワーク RAID は，論理ボリュームが複数の物理ボリュームとの結びつけを持つと同時に，そのパリティ用物理ボリュームが異なる論理ボリュームのセットから構成される点で従来手法と異なる．また，論理ボリュームと物理ボリュームのサイズはシステムで一意に定められる．通常，PC ノードが持つディスクの容量と同じオーダである数十ギガバイト〜数百ギカバイトという値が指定される．物理ボリュームは PC ノードが持つディスクの一部に格納される．

類似の機能を提供する方式として Logical Volume Manager（LVM）[4] があるが，LVM において提供されるボリュームのマッピング，結合分割と RAID 構成機能は互いに独立である．また，連鎖ネットワーク RAID は高信頼化のための RAID 構成機能を提供するものであり，ボリュームの結合分割などの機能を提供しない．こうした機能は LVM と組み合わせて用いることで実現することができる．

3.2　RAID の構成

連鎖ネットワーク RAID の構成は論理ボリュームの数，物理ボリュームの数および論理ボリュームと物理ボリューム間の結びつけによって決まることを前節で述べた．まず，その基本的な例として 3 個の論理ボリュームと 6 個の物理ボリュームからなる構成を**図 1** の (a) に示し，特徴を説明する．論理ボリューム L1 は，物理ボリューム P1 を占有してそのデータを保持し，物理ボリューム P2 において論理ボリューム L2 とのパリティを保持し，物理ボリューム P6 において論理ボリューム L3 とのパリティを保持する．この構成では，物理ボリュームのいずれか 2 個が故障しても，論理ボリュームのデータは失われず，論理ボリュームへの要求を継続して処理することができる．さらに，いずれか 3 個が故障しても，80%の確率で同様にデータはまったく失われない．たとえば，物理ボリューム P1，P2，P4 だけが稼動している場合，論理ボリューム L1 は P1 により，L2 は P1 XOR P2 により，L3 は P1 XOR P2 XOR P4 により得ることができる．P2 と P4 といった異なる論理ボリュームのセットを持つパリティが同時に複数用いられることが，本手法の特徴である．またこの例では，物理ボリューム P1，P2，P6 が故障すると，論理ボリューム L1 を得ることはできなくなる．

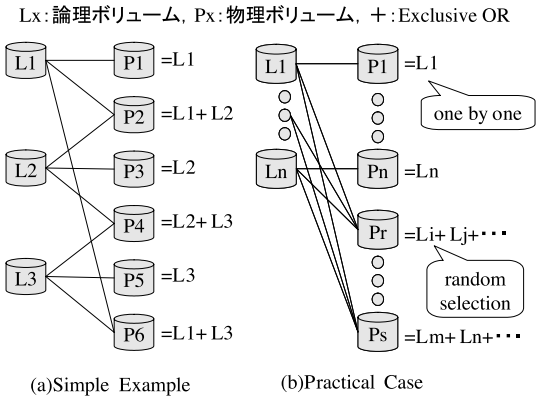

図 1　連鎖ネットワーク RAID の構成
Fig. 1　Configuration of Chain Network RAID.

実用的な場面においてはもっと多くの物理ボリュームから構成されため，連鎖ネットワーク RAID は，図 1 の (b) に示すようなランダム LDPC 符号である Gallager 符号[14] と類似の構造をとる．まず論理ボリュームがそれぞれ 1 つのデータ用物理ボリュームと結びつけられ，それ以外の物理ボリュームはパリティ用物理ボリュームとして用いられる．そして論理ボリュームとパリティ用物理ボリュームの結びつけは，無作為に選択される．また論理ボリュームが持つ結びつけの数は固定値で与えられ，これを冗長度と呼ぶ．パリティ用物理ボリュームが持つ結びつけの数は分布を持つことになるが，それが物理ボリュームの負荷の偏りの原因となってしまうため，パリティ用物理ボリュームを選択する場合に，いくつかを無作為に抽出して，その中で最も結びつけの数が少ないものを選択することで，偏りの平滑化を図る．

Gallager 符号を本手法の構成にあてはめると，論理ボリュームとパリティ用物理ボリュー

ムの双方の結びつけの数に固定値を与えた正則構造を持つものととらえることができるが，本手法は論理ボリュームのみに固定値を与える点で異なり，結びつけに対する制約は緩い．結びつけの対象は無作為に選択されるため，特定の物理ボリュームの結びつけを変更しても，その物理ボリュームが構成変更のために一時的に利用不可になるということ以外の影響はシステムの信頼性に及ばない．そのため構成はいったんシステムが稼動し始めた後も自由に変更することができる．また，結びつけの制約が緩いため，結びつけの変更を 1 つ 1 つの物理ボリュームに対して順番に実施することで，論理ボリュームと物理ボリュームの構成変更を，すなわちシステム全体の構成変更を柔軟に行うことが可能となる．

システムの構成においては，ボリュームの数，記憶効率と要求される信頼性が設計上の制約となるパラメータとして与えられ，冗長度が調節可能なパラメータとなる．冗長度は論理ボリュームのデータ更新を反映すべき物理ボリュームの数となるため，データ処理の負荷を減らしてスループットを向上させるために，小さい値であることが望ましい．最適な冗長度は条件によって異なるが，たとえば，物理ボリューム数が 200，論理ボリューム数が 100，記憶効率が 50%，要求される信頼性が PC ノード 100 台中 10 台が停止する状態で 5 nines 以上とした場合に 5 となる．提案手法における設計上の制約となるパラメータと最適な冗長度の関係について，4.1 節で評価し検証する．また，文献 15) はさらに規模の大きい場合について検証しており，最適な冗長度はボリュームの数が少ない場合に小さく，多い場合に大きくなることが分かっている．

3.3　物理ボリュームの状態

物理ボリュームが障害か停止の状態から復旧したときや RAID 構成の変更で保持すべきデータが変わった場合に，物理ボリュームを再構築する必要がある．再構築には時間がかかるため，再構築処理は RAID を停止することなく，読み書き要求と並行して実行されなければならない．これを実現するため，物理ボリュームの状態を次の 3 つに分けて管理する．

- 同期がとれている状態（UPDATE）
- 再構築中の状態（REBUILDING）
- 利用できない状態（TAINT）

UPDATE は読み書きできる状態であり，REBUILDING は読み出すことはできないが書き込みが必要な状態であり，TAINT は読み書きに使えない状態である．

3.4　読み出し要求の変換

論理ボリュームへの読み出し要求は，物理ボリュームからの読み出し要求に変換され，処理される．変換アルゴリズムを**図 2** に ruby 風の擬似コードで示す．変換アルゴリズム

```
 1:# 探索フェーズ
 2:logicals.each { |logical|
 3:  logical.requisite_physical = nil
 4:}
 5:found_flag = 1
 6:while (found_flag != 0) {
 7:  found_flag = 0
 8:  physicals_status_is_update.each { |physical|
 9:    expectant_logicals.clear
10:    physical.binded_logicals.each { |logical|
11:      if (logical.requisite_physical == nil)
12:        expectant_logicals.push (logical)
13:    }
14:    if (expectant_logicals.size == 1) {
15:      the_logical = expectant_logicals.pop
16:      the_logical.requisite_physical = physical
17:      found_flag = 1
18:    }
19:  }
20:}
21:# 変換フェーズ
22:untranslated_logicals = [ requested_logical ]
23:translated_physicals.clear
24:while (untranslated_logicals.size != 0) {
25:  the_logical = untranslated_logicals.pop
26:  physical = the_logical.requisite_physical
27:  if (physical == nil)
28:    return nil  # 変換に失敗
29:  translated_physicals.push (physical)
30:  physical.binded_logicals.each { |logical|
31:    if (logical != the_logical)
32:      untranslated_logicals.push (logical)
33:  }
34:}
35:return translated_physicals  # 変換に成功
```

図 2　読み出し要求の変換アルゴリズム
Fig. 2　Translation algorithm of read request.

は論理ボリューム（図中 logical(s)）のデータを得るために，どの物理ボリューム（図中 physical(s)）を用いればよいかを求める．アルゴリズムは論理ボリュームのデータを得るために，すでにデータが得られている論理ボリュームに加えて必要な物理ボリューム（図中 requisite_physical）を求める探索フェーズ（図中 1〜20 行目）と，その結果を用いて対象

の論理ボリューム（図中 requested_logical）のデータを得るために必要な物理ボリュームのセット（図中 translated_physicals）を求める変換フェーズ（図中 21〜35 行目）によって構成される．

探索フェーズにおいて対象となる物理ボリュームは，その状態が UPDATE のボリュームのみである（図中 8 行目）．物理ボリュームに結びつけられた論理ボリュームのうち（図中 10 行目），まだデータが得られない論理ボリュームが 1 つだけあった場合（図中 14 行目），その論理ボリュームのデータは，その物理ボリュームを用いることで取得可能となる（図中 16 行目）．たとえば，その物理ボリュームがデータ用であれば，物理ボリュームのデータそのものとして得られ，パリティ用であれば，結びつけられた他の論理ボリュームとその物理ボリュームのデータの XOR 演算結果として得られる．この条件を利用してすべての物理ボリュームを繰り返し評価することで（図中 6 行目），論理ボリュームのデータを得るためにどの物理ボリュームを用いればよいかが明らかになる．変換フェーズにおいては，変換すべき論理ボリューム（図中 untranslated_logicals）を探索フェーズによって得られた結果を参照し変換する（図中 26 行目）．パリティ用物理ボリュームを用いる場合は，それに結びつけられた他の論理ボリュームについても再帰的に処理を行い（図中 30〜32 行目），必要な物理ボリュームのセットを求める（図中 35 行目）．得られた物理ボリュームのセットの XOR 演算結果が対象の論理ボリュームのデータとなる．

3.5　書き込み要求の変換

論理ボリュームへの書き込み要求は，書き込むデータを得るために必要に応じて行われる物理ボリュームからの読み出し要求と物理ボリュームへの書き込み要求に変換され，順に処理される．

書き込みの対象となる物理ボリュームは，書き込み対象の論理ボリュームに結びつけられたすべての物理ボリュームの中でその状態が UPDATE と REBUILDING のボリュームとなる．物理ボリュームの状態が REBUILDING である場合は，結びつけられたすべての論理ボリュームのデータを前節で示した読み出し要求の変換を用いて取得し，書き込み対象の論理ボリュームを除くこれらのデータと新たに書き込むデータの XOR 演算結果より書き込むデータが得られる．また，物理ボリュームの状態が UPDATE である場合は，書き込み対象の論理ボリュームのデータを前節で示した読み出し要求の変換を用いて取得し，このデータと新たに書き込むデータとその物理ボリュームのデータの XOR 演算結果より書き込むデータが得られる．後者は，論理ボリュームの更新差分を求め，それを利用して物理ボリュームに書き込むデータを得る方法である．

物理ボリュームの再構築要求では，物理ボリュームに結びつけられたすべての論理ボリュームのデータを前節で示した読み出し要求の変換を用いて取得し，これらのデータの XOR 演算結果より書き込むデータが得られる．

3.6　データの整合性

1 つの物理ボリュームに複数の論理ボリュームが結びつけられているため，異なる論理ボリュームへの要求が同じ物理ボリュームへの要求に変換される．この場合に，これらの処理を同時並行に行ってしまうと，データの整合性が失われてしまう恐れがある．たとえば，図 1 の (a) の構成の場合，論理ボリューム L1 と L2 への書き込みは，どちらも物理ボリューム P2 への読み出しと書き込みをともなう．これらの処理が，L1 のための読み出し，L2 のための読み出し，L1 のための書き込み，L2 のための書き込みの順で行われてしまうと，L1 への書き込みによる P2 のデータの更新が失われてしまい，物理ボリューム P2 は不適切なデータを保持することになる．

そこで，データの整合性を維持するために，連鎖ネットワーク RAID は競合する論理ボリュームへの要求が同時に処理されないように排他制御を行う．物理ボリュームごとに読み書きロックが管理され，論理ボリュームへの要求は，物理ボリュームへの要求に変換された後，それらの物理ボリュームの読み書きロックを取得する．必要なロックをすべて取得できた要求だけを処理することで，データの整合性を維持することができる．また，ロック情報はアクセスする番地とサイズを含み，同じ物理ボリュームでも異なる番地への要求は競合しないと判断される．システムの全記憶領域に対して要求が出されている領域はごく一部であり競合が発生する確率はきわめて低い．そのため，RAID 全体での要求処理の並列性は保たれる．

3.7　システム設計

これまでに述べた処理を各 PC ノード上で自律分散的に行うことを考えると，ネットワークの遅延による性能の劣化と異常系処理の複雑化が問題になる．しかし，単純にすべての処理を単一の PC ノードで行うと，その PC ノードの処理性能がボトルネックとなり，スケーラビリティが低くなる．そこで，制御 I/O とブロック I/O を分離して，スケーラビリティの向上を図る．本システムは，**図 3** に示す次の 5 個のコンポーネントで構成される．

- initiator_block：論理ボリュームへ要求を出す．
- target_block：物理ボリュームを提供する．
- map_handler：ボリューム情報の管理，要求の変換処理と排他制御を行う．
- dispatcher：initiator_block から要求を受け，map_handler と制御 I/O をやりとりし

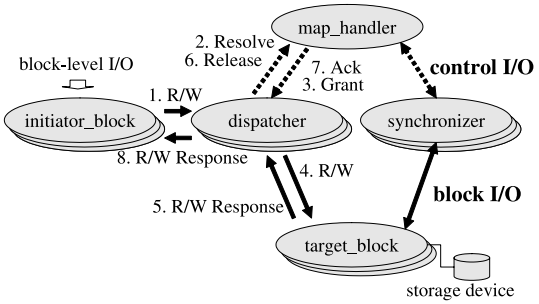

図 3　連鎖ネットワーク RAID の構成部品
Fig. 3　Components of Chain Network RAID.

て，ブロック I/O の処理を行う．

- synchronizer：map_handler から物理ボリュームの再構築処理のための制御 I/O を受け，ブロック I/O の処理を行う．

ブロック I/O は，initiator_block と dispatcher，target_block と dispatcher，target_block と synchronizer の間で交換される．このブロック I/O は，番地を指定した読み書き要求とそのデータから構成される．制御 I/O は，dispatcher と map_handler，synchronizer と map_handler の間で交換される．この制御 I/O は，読み書きすべき論理ボリュームと物理ボリュームの情報から構成され，読み書きデータは含まれない．

制御 I/O は，次の 4 種類の信号から構成され，これらが map_handler と dispatcher との間で順に交換され，読み書き要求が処理される．

- Resolve：dispatcher が論理ボリュームへの要求を伝える．
- Grant：map_handler が変換と排他制御を行い，物理ボリュームへの要求を伝える．
- Release：dispatcher が物理ボリュームへの要求の処理結果を伝える．
- Ack：map_handler が論理ボリュームへの要求の処理結果を伝える．

ただし，論理ボリュームへの要求を処理するために必要な物理ボリュームが利用不可能であり，要求の変換ができない場合には，dispatcher からの Resolve に対して，map_handler がすぐに Ack を返す．また，dispatcher からの Release が失敗応答であるときに，別の物理ボリュームのセットを用いて論理ボリュームへの要求の処理を継続することができる場合は，map_handler が再度異なる Grant を返すこともある．前者は冗長化の許容度を超えた異常状態に，後者は PC ノードの故障すなわち物理ボリュームの状態変更を map_handler が検知するまでの過渡的な状態に発生する．

物理ボリュームの再構築処理は，管理者が map_handler に対して要求を行い，map_handler と synchronizer との間で Grant と Release の制御 I/O が交換され，物理ボリュームへの書き込みが行われる．

システムの整合性を維持する機能は map_handler に集中しており，dispatcher や synchronizer などの他のコンポーネントは，システム内で複数個，並列して動作することができる．制御 I/O が map_handler に集中しているため，デッドロックが起きないように要求間の依存関係を検証しながら処理することが容易となる．また，データ I/O を扱う dispatcher や synchronizer はその数を増やすことができるため，データ I/O の処理について特定の PC ノードがボトルネックとなることはない．

一方，map_handler はシステムに 1 つだけ動作するため，単一障害点（Single Point of Failure）になりうる．これに対しては，アクティブ・スタンバイ構成で map_handler のバックアップノードを準備して，冗長化を行う．アクティブな map_handler が持つ情報のうち，論理ボリュームと物理ボリュームの情報と状態，およびその結びつけ情報は更新が起きるたびにつねにバックアップノードへ複製する．複製される情報に排他制御を行うための処理中の要求情報は含まれない．そのため，PC ノードの異常検出やシステム構成の変更といった少ない機会にのみ複製は行われ，その量もたかだか数 MB 程度と少なく，map_handler の処理性能に影響を及ぼさない．map_handler のスタンバイへの切替えが発生した場合，処理中の要求情報が引き継がれないため，新しくマスタになった map_handler は dispatcher と synchronizer からの再接続とそれらがすでに保持している要求の処理が終わるのを待つため，一定時間新たな要求を保留する．また，切替えの発生と同時に dispatcher や target_block に障害が発生した場合，処理中の書き込み要求が正しく処理されずに終了してしまい，データの不整合が生じる可能性がある．これは既存の RAID で Write Hole と呼ばれる問題と同じである．

3.8　プロトタイプ実装

前節で示した 5 個のコンポーネントをユーザ空間で動作するプロセスとして実装した．プロセス間の通信機能は，制御 I/O とブロック I/O を独自のプロトコルで規定し，ソケットを用いて実装した．また，initiator_block は Linux SCSI target framework [16] の backed_io_template として実装されており，Linux SCSI target framework が提供する iSCSI [17] のインタフェースから論理ボリュームへの読み書き要求を受け取る iSCSI ターゲットとして動作する．map_handler は，dispatcher や synchronizer とのコネクションごとに

スレッドを起動するマルチスレッドプログラムであり，Fast Userspace Mutexes [18] という軽量な IPC を用いてスレッド間の排他制御を実装した．

典型的には，クラスタ側の各データ保持用 PC ノードに dispatcher，synchronizer，initiator_block と target_block が 1 つずつ配置される．また，map_handler はクラスタ側の制御用 PC ノードに単独で配置される．本来，initiator_block はクライアント側で動作すべきコンポーネントだが，本プロトタイプ実装においては実装上の汎用性のためにさらに iSCSI ターゲット機能を持たせたため，クラスタ側に配置する．この典型的なコンポーネントのノード配置例が後述する実験環境（図 5）である．

4. 評　　価

本章ではまず，提案手法の冗長化方式としての特性を明らかにするために，PC ノードの構成と稼働台数を定めた場合のそれぞれの論理ボリュームの読み出し可能な確率を可用性と定義し，シミュレーションを用いて評価する．また，提案手法の読み書きスループットとスケーラビリティをプロトタイプ実装を用いて評価する．

4.1 可用性と信頼性

4.1.1 評 価 条 件

2 個の物理ボリュームを持つ PC ノード 100 台から構成されるストレージシステムがどのような可用性を実現するかを評価する．各 PC ノード 2 つの物理ボリュームのうち，1 つをデータ用物理ボリュームとして，もう 1 つをパリティ用物理ボリュームとして用いる．すなわち，200 個の物理ボリュームより 100 個の論理ボリュームが作られる，記憶効率 50%の構成とする．この構成において，PC ノードの稼働台数を変化させ可用性を評価した．可用性の値は，結びつけの選択を無作為に 100 通り行い，それぞれの結びつけにおいて稼動 PC ノードの選択を無作為に 1,000 通り行い，それぞれの稼動状態において論理ボリュームが読み書き可能な確率の平均値として求めた．さらに，冗長度を 3 から 7 まで変化させ，それぞれについて評価した．また，比較のため Reed-Solomon 符号を用いた場合についても，文献 13) にある計算式より同様の値を求めた．

4.1.2 評 価 結 果

評価結果を小数点以下に 9 がいくつ続くかを示す nines という単位で**図 4** にまとめる．また，図中右側に小数点での表記も示し，提案手法については「CNR：冗長度」と Reed-Solomon 符号については「k out of n」と示した．提案手法において冗長度を 5 にした場合，100 台中 10 台が停止する状態にあっても 5 nines という高い可用性が実現できることが分かる．さらに冗長度を 7 に上げることで，20 台が停止する状態にあっても同等の可用性が実現できる．また，Reed-Solomon 符号を用いた場合は，5 out of 10 符号や 10 out of 20 符号を用いることで同様の可用性が実現できることが分かる．

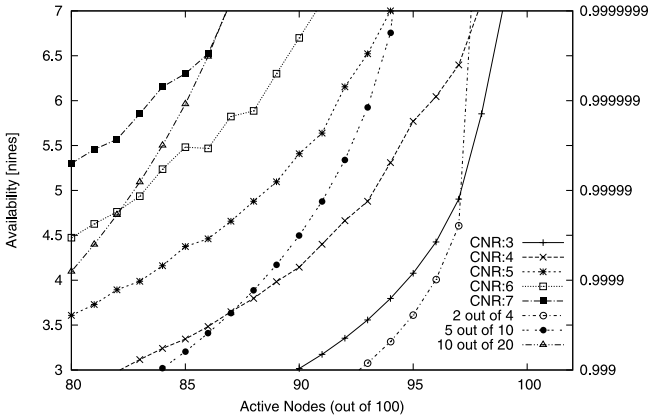

図 4　様々な冗長度における可用性
Fig. 4　Availability with various redundancy degrees.

本節では冗長化方式の特性を比較するため稼動台数を固定した場合の可用性を評価したが，ストレージの信頼性は物理ボリュームの故障率，物理ボリュームの再構成にかかる時間と管理者が故障した物理ボリュームを交換するまでにかかる時間に基づくマルコフ連鎖の状態遷移確率および図 4 の結果より求まる．物理ボリュームを再構成するために必要なデータ用物理ボリュームとパリティ用物理ボリュームの数をグループサイズと呼ぶが，提案手法のグループサイズの期待値は，システム稼動時の大部分を占める稼働台数が比較的多い場合においてほぼ冗長度と等しく，図 4 において同等な可用性を持つ Reed-Solomon 符号に比べて同程度である．すなわち，物理ボリュームの再構成にかかる時間も同程度であり，提案手法が Reed-Solomon 符号と同様の高い信頼性を実現できることが分かる．

両者とも高い可用性を達成することができるが，Reed-Solomon 符号を用いてシステムを構成する場合は RAID グループの数を自由に変更することはできない．たとえば，20 台の物理ボリュームを Reed-Solomon 符号で構成する場合，5 out of 10 符号の 2 つの RAID グループから構成される状態と 10 out of 20 符号の 1 つの RAID グループから構成される状態を可用性を保ったまま行き来することはできない．なぜなら，移行させる際に物理ボ

表 1　実験に用いた PC の仕様
Table 1　Specification of PC nodes.

PC	TypeA	TypeB
CPU	Pentium4 3 GHz	Xeon 3.4 GHz
Memory	2 GByte	2 GByte
HDD	SATA2 160 GByte	Ultra320 SCSI 73 GByte
Network	1000Base-T	1000Base-T
OS	Linux 2.6.19	Linux 2.6.19

リュームを一時的に RAID から外す必要が出てくるため，可用性が大幅に低下してしまうからである．

システムを運用する過程で設計上の制約となるパラメータの見直しが行われる．その際に要求される可用性がより高いものに変更になったとすれば，図 4 は Reed-Solomon 符号ではその時点で RAID グループのサイズを増やすことで対応できることを示している．しかし，RAID グループの数を減らす必要がでてくるため，そのように構成を変更することができない．逆にこのような変更に対応しようと，事前に RAID グループのサイズを大きくしておくと，データ更新を反映すべき物理ボリュームの数が増えるためにスループットが犠牲となる．一方，提案手法は冗長度を変えることでこれに相当する構成の変更を行うことができ，その時点での設計上の制約を満たす，最適なスループットを実現する構成に変更することができる．多くのボリュームを扱う際に Reed-Solomon 符号を用いるといくつかの RAID グループに分けることになるが，RAID グループの数が構成の変更を柔軟にできない要因となる．提案手法は全体を 1 つの RAID グループとして管理できるため，このような問題が発生しない．

4.2　スループット

4.2.1　実 験 環 境

7 台の PC からクラスタ・ストレージを構成し，スループットを評価する．実験に用いた PC の仕様を**表 1** に，実験のシステム構成を**図 5** に示す．GbE スイッチで接続される 6 台の PC ノードを物理ボリュームを格納するためのストレージノードとして用いた．これらの PC ノードに格納されるデータは図 1 の (a) と同じ構成として，3 つの論理ボリュームを作成した．また，これらを制御するためのコントロールノードを別に 1 台用意した．

クラスタに対して 3 台の計測用 PC から読み書きの負荷をかけ，そのスループットを計測した．計測用 PC はそれぞれ別の 1 つの論理ボリュームに対して読み書きの要求を出す．計測用 PC は open-iscsi [19] を用いて initiator_block プロセスとの間に iSCSI セッションを

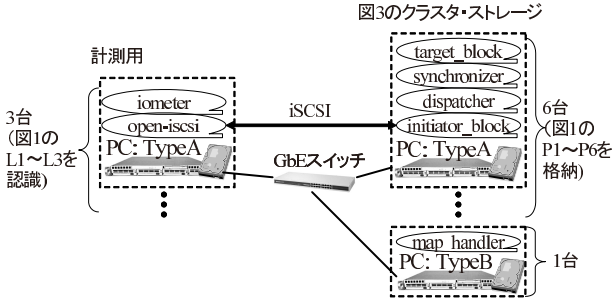

図 5　実験環境のシステム構成
Fig. 5　System configuration of experiment.

確立し，論理ボリュームをディスクドライブとして認識する．iSCSI セッションはその論理ボリュームが占有するデータ用物理ボリュームを格納する PC ノードとの間に確立した．

計測用 PC 上で iometer [20] を動作させ，リクエストサイズを変えてシーケンシャルな読み出しスループットと書き込みスループットを計測する．また，クラスタ・ストレージにアクセスをする各アプリケーションのスレッドが得られるスループットを評価するために，iometer の Worker スレッドは 1 つだけ動作させ，同時に発行する I/O の数を 1 に設定した．

4.2.2　実 験 結 果

計測用 PC を 1 台だけ動作させたときの読み出しの結果を**図 6** の “single” に，書き込みの結果を**図 7** の “single” に示す．リクエストサイズが十分に大きい場合は，読み出し時に 30 [MB/sec]，書き込み時に 7 [MB/sec] 程度のスループットが得られている．読み出し処理が 1 つの物理ボリュームからデータを読み出すのに対し，書き込み処理は 3 つの物理ボリュームに read-modify-write の操作を行うため，書き込みのスループットは読み出しと比較して小さくなる．

次に 6 台のストレージノードのうち 1 台を停止させたときの読み出し結果を図 6 の “single under 1 failed node” に，書き込みの結果を図 7 の “single under 1 failed node” に示す．また，3 台を停止させたときの結果についても同様に示す．読み出し処理では，異なる PC ノードから多くのデータを取得する必要がでるため，そのスループットは小さくなる．一方，書き込み処理では書き込むべき物理ボリュームが減るため，そのスループットはわずかであるが大きくなる．

最後にすべてのストレージノードが動作した状態で計測用 PC を 2 台並列して動作させ

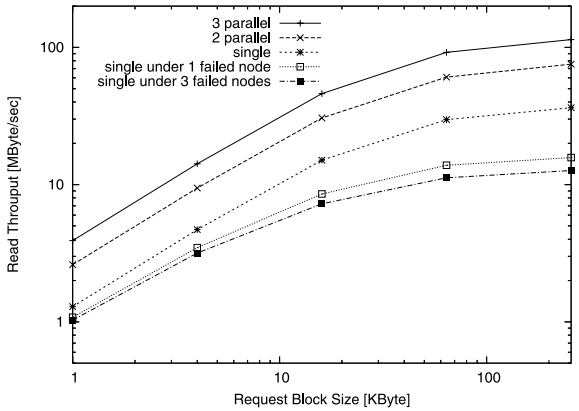

図 6　読み出しのスループット
Fig. 6　Read throughput.

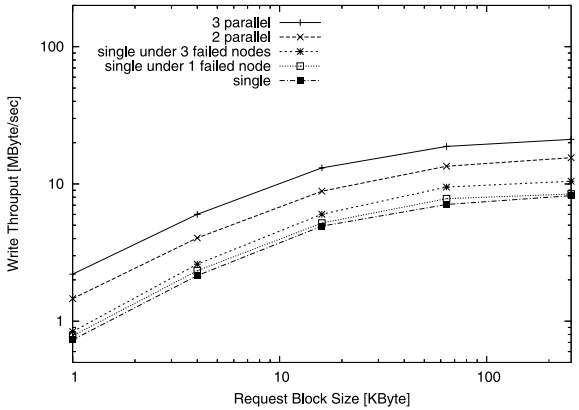

図 7　書き込みのスループット
Fig. 7　Write throughput.

たときの読み出しの結果を図 6 の “2 parallel” に，書き込みの結果を図 7 の “2 parallel” に示す．また，3 台を動作させたときの結果についても同様に示す．結果はすべての計測用 PC のスループットの合計値である．読み出し処理と書き込み処理ではともに，2 倍，3 倍と計測用 PC の台数に比例したスループットが得られている．提案手法では制御 I/O とブロック I/O が分離されており，特定の PC ノードに負荷が集中せずにブロック I/O が並列して処理されることが確認できた．

OceanStore [21] はオブジェクトレベルのインタフェースを持つ Reed-Solomon 符号を用いたストレージシステムであるが，文献 22) 中の Figure 5 において LAN 環境でのプロトタイプの性能評価が行われている．クラスタ全体で得られる更新のスループットは 2.6 [MB/sec] であり，符号化の処理コストが原因となり低く抑えられている．冗長度や PC ノード数など条件が異なるため本評価との単純な比較はできないが，提案手法では 20 [MB/sec] を超える書き込みスループットが得られており，提案手法のスループットが十分に高いことが分かる．

4.3　スケーラビリティ

4.3.1　実 験 環 境

PC ノードの台数が増加すると，システム全体の整合性を保つために制御 I/O が集まる map_handler の処理性能がボトルネックになる．そこで，map_handler が論理ボリュームへの読み書き要求をどれだけのレートで処理できるか評価する．

map_handler を表 1 の TypeB の PC で動作させ，4.1 節で示したものと同じボリューム構成とした．9 台の計測用 PC を用いて，計測用 PC にブロック I/O の実処理を行わずに論理ボリュームへの要求を生成し続けるよう改造した dispatcher からすべての論理ボリュームへ並列に要求を送った．

4.3.2　実 験 結 果

冗長度を変えて，map_handler の処理レートを計測した．dispatcher から読み出しの要求を送ったときの結果を**図 8** の “Read” に，書き込みの要求を送ったときの結果を図 8 の “Write” に示す．

読み出し処理では，処理レートは冗長度によらずに 90 K [IO/sec] 程度であった．書き込み処理では，冗長度が増えると書き込むべき物理ボリュームの数とともに取得すべきロックも増えるため，冗長度が増えるに従って処理レートは小さくなる．仮に読み出しと書き込みの比率が 1 : 1 であるとすると，map_handler は 70 K [IO/sec] の処理レートで要求を処理できることが分かった．

文献 23)，24) によると，一般的なトランザクションのワークロードにおける SATA，SAS ドライブの IOPS は 100 から 200 程度である．すなわち，提案システムに 350 台のディスクドライブを収容しても map_handler がボトルネックになることはない．また，すべてのドライブがつねに負荷のかかった状態でなければ，その分収容できるドライブの数は多く

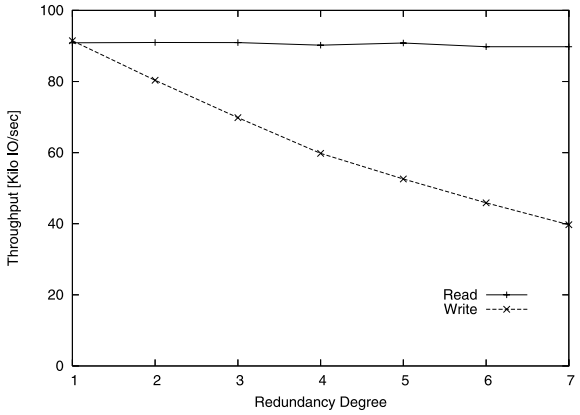

図 8　コントロールノードの処理性能
Fig. 8　Performance of the control node.

なる．map_handler 以外のコンポーネントはすべて並列に動作させて負荷分散することができるため，提案システムは 350 台のディスクドライブを収容するに十分なスケーラビリティを有しているといえる．4.2 節の評価結果とあわせ，提案手法では PC ノードの台数を増やすことでシステム全体のスループットを向上させることができることが確認できた．

5. 関 連 研 究

Intermemory [25] は消失訂正符号を使ってブロックレベルのストレージを提供する分散型のシステムである．IM-0 というプロトタイプ実装は，データを 16 out of 32 符号で分配することで高い信頼性を実現する．また，各ワークステーションでデーモンが動作し，自律的に動作することで中央の制御を不要に，ボトルネックを解消している．しかし，データを多くの断片に分割しノードに分散させる処理のオーバヘッドが大きいため，その応用は write-once 型のアーカイブ用途に限られている．

DR-net [26] は，ディスクノードを 2 次元トーラスのトポロジで接続し，各ノードが異なる 2 種類の近隣のノード群との間でパリティを構成することで，高い信頼性を実現する手法である．我々の手法では，制御 I/O を集中させて処理しているため，パリティを構成するディスクノードをネットワークのトポロジとは独立に選択することができる．そのため，冗長度の調節やノードの追加などの柔軟なシステム構成変更が可能である．また，DR-net はパリティを構成するディスクノード間で制御 I/O を自律的に処理するが，文献 26) では要求間の排他制御方式とそのデッドロックの回避策について十分な議論がなされていない．

Ohde ら [27] は，分散ストレージの高信頼化に用いる LDPC 符号の構成法を提案した．Steiner triple system と additive-3 code を用いた triple-erasure correcting code を提案し，Reed-Solomon 符号に比べて 3 桁高い MTTDL が実現できることを示した．この文献で提案される Sequential direct-decoding アルゴリズムは，連鎖ネットワーク RAID の読み出しアルゴリズムと同じロジックを持つが，連鎖ネットワーク RAID はシーケンシャルに処理を実行するのではなく，必要な物理ボリュームのセットとしてすべて展開し，パラレルな処理を可能とする点で異なる．また，この文献においては DR-net と同様に，要求間の排他制御方式について十分な議論がなされていない．

Kaneko ら [28] は，Ohde ら [27] の手法を対象に，スペア用ディスクを準備せずに故障したディスクの数に応じてパリティ用ディスクを再構成する手法を提案し，対象として手法に比べさらに 3 桁高い MTTDL が実現できることを示した．データ用物理ボリューム（連鎖ネットワーク RAID における論理ボリュームに相当）とパリティ用物理ボリュームの結びつけを動的に変更する点は，連鎖ネットワーク RAID と同じある．この手法は物理ボリュームの数が初期に設定された数より故障によって減っていく場合を対象としているが，連鎖ネットワーク RAID はシステムの拡張が必要になり物理ボリュームの数が初期に設定された数よりも増えていくことを想定している点で両者は異なる．

GPFS [29] はディスク共有型のクラスタファイルシステムであり，集中型と分散型の両方のロック管理機構を持つ．バイトレンジを指定したロック管理を行っており，複数のノードが同じファイルへアクセスする際の排他制御に用いられている．我々の手法は，物理ボリュームに対して同様に番地を指定したロック管理を行っているが，同じ要求に対しても物理ボリュームの状態に応じてロックの対象となる物理ボリュームが変化する点が異なる．

6. ま と め

我々はブロックレベルの PC クラスタ・ストレージを実現するためのデータ冗長化方式として連鎖ネットワーク RAID を提案し，その設計と実装について述べた．連鎖ネットワーク RAID は，複数の PC ノードの同時故障に耐えうる高い信頼性と柔軟に PC ノードの追加や構成の変更ができるという特長を持つ．

シミュレーションを用いた評価により，本手法が 100 台中 10 台が停止か故障した状態でも 5 nines 以上の高い可用性を実現することを示した．また，プロトタイプ実装を用いた評

価実験により，本システムが PC ノードの台数に比例したスループットを実現することと，70 K [IO/sec] の処理性能を持つことを示した．

参 考 文 献

1) Ghemawat, S., Gobioff, H. and Leung, S.-T.: The google file system, *ACM SOSP* (Oct. 2003).
2) MogileFS. http://www.danga.com/mogilefs/
3) Isilon IQ Clustered Storage. http://www.isilon.com/
4) Teigland, D. and Mauelshagen, H.: Volume Managers in Linux, *USENIX Annual Technical Conference*, Boston, MA, pp.185–198 (June 2001).
5) Flouris, M.D. and Bilas, A.: Clotho: Transparent Data Versioning at the Block I/O Level, *MSST2004* (Apr. 2004).
6) NILFS: log-structured file system developed for the Linux. http://www.nilfs.org/
7) Lustre. http://www.lustre.org/
8) Patterson, D., Gibson, G. and Katz, R.: A Case for Redundant Array of Inexpensive Disks (RAID), *Proc. ACM SIGMOD'88*, pp.109–116 (June 1988).
9) Xin, Q., Miller, E.L. and Schwarz, T.J.E.: Evaluation of Distributed Recovery in Large-Scale Storage Systems, *HPDC-13 '04*, pp.172–181 (2004).
10) Merchant, A. and Yu, P.S.: Analytic Modeling of Clustered RAID with Mapping Based on Nearly Random Permutation, *IEEE Trans. Comput.*, Vol.45, No.3, pp.367–373 (1996).
11) Anvin, H.P.: The mathematics of RAID-6. http://kernel.org/pub/linux/kernel/people/hpa/raid6.pdf
12) Corbett, P., English, B., Goel, A., Grcanac, T., Kleiman, S., Leoung, J. and Sankar, S.: Row-Diagonal Parity for Double Disk Failure Correction, *Proc. 3rd USENIX Conference on File and Storage Technologies*, San Francisco, CA (Mar. 2004).
13) Weatherspoon, H. and Kubiatowicz, J.D.: Erasure Coding vs. Replication: A Quantitative Comparison, *Proc. IPTPS'02* (Mar. 2002).
14) Gallanger, R.G.: *Low Density Parity-Check Codes*, MIT Press, Cambridge, MA (1963).
15) 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，SACSIS2005, IPSJ Symposium Series Vol.2005, No.5, pp.99–106 (2005).
16) 藤田智成：Linux におけるストレージシステムフレームワークの実現，情報処理学会論文誌：コンピューティングシステム，Vol.47, No.SIG12(ACS15), pp.411–419 (2006).
17) Satran, J., Meth, K., Sapuntzakis, C., Chadalapaka, M. and Zeidner, E.: RFC3720: Internet Small Computer Systems Interface (iSCSI) (Apr. 2004).
18) Franke, H., Russell, R. and Kirkwood, M.: Fuss, futexes and furwocks: Fast User-level Locking in Linux, *Ottawa Linux Symposium 2002* (2002).
19) Open-iSCSI. http://www.open-iscsi.org/
20) Iometer. http://www.iometer.org/
21) Kubiatowicz, J., et al.: OceanStore: An Architecture for Global-Scale Persistent Storage, *ASPLOS* (Dec. 2000).
22) Rhea, S., Eaton, P., Geels, D., Weatherspoon, H., Zhao, B. and Kubiatowicz, J.: Pond: The oceanstore prototype, *Proc. USENIX File and Storage Technologies FAST* (2003).
23) Barclay, T., Chong, W. and Gray, J.: A Quick Look at Serial ATA (SATA) Disk Performance, Technical Report MSR-TR-2003-70, Microsoft Corporation (Oct. 2003).
24) Optimizing Storage With SAS: Beyond the 10K Compromise, Technology Paper TP-543, Seagate (Sep. 2005).
25) Chen, Y., Edler, J., Goldberg, A., Gottlieb, A., Sobti, S. and Yianilos, P.: A prototype implementation of archival intermemory, *Proc. 4th ACM Conference on Digital Libraries*, pp.28–37 (1999).
26) Yokota, H. and Mimatsu, Y.: A Scalable Disk System with Data Reconstruction, *Input/Output in Parallel and Distributed Computer Systems*, Chapter 16, Kluwer Academic Publishers (June 1996).
27) Ohde, H., Kaneko, H. and Fujiwara, E.: Low-Density Triple-Erasure Correcting Codes for Dependable Distributed Storage Systems, *Proc. 2006 IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems*, pp.175–183 (Oct. 2006).
28) Kaneko, H. and Fujiwara, E.: Reconstruction of Erasure Correcting Codes for Dependable Distributed Storage System without Spare Disks, *Proc. 2007 IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp.349–357 (Sep. 2007).
29) Schmuck, F. and Haskin, R.: GPFS: A Shared-Disk File System for Large Computing Clusters, *Proc. FAST'02*, pp.231–244 (Jan. 2002).

(平成 19 年 10 月 4 日受付)
(平成 20 年 3 月 4 日採録)

市川　俊一（正会員）

2000 年早稲田大学理工学部電子・情報通信学科卒業．2002 年同大学院理工学研究科修士課程修了．同年日本電信電話株式会社入社．ネットワークストレージシステムの研究開発に従事．電子情報通信学会，日本データベース学会各会員．

豊田真智子（正会員）

2004 年お茶の水女子大学理学部情報科学科卒業．2006 年同大学院人間文化研究科数理・情報科学専攻修士課程修了．同年日本電信電話株式会社入社．本会 2005 年度大会奨励賞受賞．データストリーム処理の研究開発に従事．電子情報通信学会，日本データベース学会各会員．

高橋　克巳（正会員）

1988 年東京工業大学理学部数学科卒業．2006 年東京大学大学院情報理工学系研究科電子情報学専攻博士後期課程修了．1988 年日本電信電話株式会社入社．現在，NTT 情報流通プラットフォーム研究所セキュリティ社会科学グループリーダ．情報セキュリティ，プライバシ保護，情報検索，データマイニング，地理情報処理等の研究に従事．本会 1999 年度論文賞受賞．博士（情報理工学）．