高信頼PCクラスタ・ストレージ実現のための連鎖ネットワークRAIDの設計と実装

市川　俊一† 　豊田真智子† 　高橋　克巳†

† 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　〒 180–8585 東京都武蔵野市緑町 3–9–11
E-mail: †{ichikawa.toshikazu,toyoda.machiko,takahashi.katsumi}@lab.ntt.co.jp

あらまし　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安価に構築することを目的として，高信頼なブロックレベルの 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．提案手法は 2 台以上のノードの同時故障に耐えうる高い信頼性を実現できることと，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．また，プロトタイプ実装を用いた評価により，提案手法が冗長化を行わない場合の iSCSI アクセスと比べて，読み出し時に 1/4 以上，書き込み時に 1/5 以上のパフォーマンスを達成することと，ノード数に比例したスループットを実現する高いスケーラビリティを持つことを示す.

キーワード　RAID，クラスタ・ストレージ，高信頼性，SAN，オンラインストレージ

Design and Implementation of Chain Network RAID for high-reliability PC clustered storage

Toshikazu ICHIKAWA†, Machiko TOYODA†, and Katsumi TAKAHASHI†

† NTT Information Sharing Platform Laboratories, NTT Corporation　Midoricho 3–9–11, Musashino-shi, Tokyo, 180–8585 Japan
E-mail: †{ichikawa.toshikazu,toyoda.machiko,takahashi.katsumi}@lab.ntt.co.jp

Abstract　To compose the high reliability block-level storage using PC cluster for building the cost effective 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. Our proposal features the reliability to be able to sustain the concurrent failures of more than two nodes, and the flexibility to adapt the modification of composition such as the addition of nodes. The evaluation with the prototype implementation shows that our proposal achieves the performance of quarter in read and one fifth in write related to the direct iSCSI access without any redundancy, and the high scalability of the throughput in proportion to the number of nodes.

Key words　RAID, clustered storage, high reliability, SAN, online storage

1. はじめに

計算機によって扱われるデータ量は年々増加し，ストレージデバイスの設置・運用コストが増大している．そのため，計算機とストレージデバイスがシステムバスで接続される従来のストレージアーキテクチャDirect Attached Storage (DAS) から，計算機とストレージデバイス間を高速なネットワークで接続する Storage Area Network (SAN) や Network Attached Storage (NAS) とよばれるストレージアーキテクチャへの移行が進んでいる．

また，PC のプロセッサ性能の向上と低価格化を背景に，高性能計算機の分野を中心に PC を Ethernet や InfiniBand などの高速なネットワークで結合した PC クラスタに注目が集まっている．そうした中，PC クラスタで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実用化されている [1], [2]．これらは廉価なコモディティ・ハードウェアを用い，個々のノードの信頼性の低さを補うための冗長化の仕組みを提供している．また，個々のノードの性能を集めることで，数多くのホストコンピュータに対する高いパフォーマンスを実現している．

これらのクラスタ・ストレージはファイルレベルのインタフェースを提供する NAS として機能する．NAS にはファイル共有の機能を直接提供できるという利点がある．一方，ブロックレベルのインタフェースを提供する SAN 環境では OS，デー

タベースや任意のファイルシステムを格納できるという利点がある．例えば，バージョン管理の機能をブロックレベルやファイルシステムレベルで効率良く実現するコンポーネント [3], [4] などを組合せることできる．

そこで，本研究は SAN 環境を前提としたブロックレベルのクラスタ・ストレージについて検討する．また特に，マスユーザや中小企業を対象としたオンラインストレージサービスのストレージを安価に構築することを目的とする．そのための要求条件として次の三つが上げられる．一つ目は，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．二つ目は，ノードの追加や交換など保守・運用にかかる費用を低く抑えるために，柔軟なシステム構成を実現することである．三つ目は，WAN を越えるアクセスを前提としているため，個々のリクエストに対して極めて短い応答時間を実現する必要はないが，システム全体で高いスループットを実現することである．こうした要件を踏まえ，ブロックレベルの PC クラスタ・ストレージにおける個々のノードの信頼性の低さを補うための冗長化方式について検討する．

2. 節では，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. 節では，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. 節では，プロトタイプ実装を用いた実験により連鎖ネットワーク RAID のスループットを評価し，そのパフォーマンスについて述べる．最後に 5. 節では，まとめと今後の課題について述べる．

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高信頼化する手法として RAID [5] 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起きたり，不良セクタが発生する確率が無視できなくなり，データの信頼性を維持する方式が問題となる．

FARM [6] と clustered RAID [7] 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る．FARM はドライブの故障が発生した時，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時，再構築によってかかる負荷を分散させる．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共に，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る．

RAID6 [8] と Row-Diagonal Parity [9] は，耐えうる故障の台数を一つ増やすことで，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，コモディティ・ハードウェアである廉価な PC を用いてクラスタ・ストレージを構成する場合は，ノードの故障を想定すると共に，OS のアップデートなどシステム保守のための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．また，ネットワーク・スイッチの故障により，複数のノードが同時に切り離される障害も想定しなければならない．そのため PC クラスタ・ストレージでは，より高い信頼性が求められる．

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 [10]. RAID6 は Reed-Solomon 符号の有限体多項式を二つに絞った場合に相当するが，これを任意の個数に拡張することも考えられる．しかし，ストレージへのアクセスがすべてコントローラに集中することで，コントローラがボトルネックとなり，システム全体のスループットが低下してしまうことが問題となる．また，書き込み要求を処理する時に，パリティを保持するノードにアクセスが集中することも同様に問題となる．

Intermemory [11] 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ため，読み書き要求を処理するためのオーバーヘッドが問題となる．その結果，提供できるストレージは write-once 型に限られ，ISO-9660 CD-ROM イメージとしてマウントしなければならない．

3. 提案手法

我々は，ブロックレベルの PC クラスタ・ストレージにおけるデータの冗長化方式として，連鎖ネットワーク RAID を提案する．連鎖ネットワーク RAID は，2 台以上のノードの故障に耐えうる高い信頼性を実現しながらも，読み書き可能なブロックレベルのストレージを提供する．また，ノードの追加や RAID 構成の変更を柔軟に行うことができ，システムのノードの台数に比例した高いパフォーマンスを提供する．以下，本節では連鎖ネットワーク RAID の設計について述べる．

3.1 ボリューム管理

連鎖ネットワーク RAID は，ノードが保持する物理ボリュームからホストへ仮想的な論理ボリュームを提供する．これまでの RAID は，少数の物理ボリュームから RAID グループが構成され，それを論理ボリュームとして提供する．RAID グループは，入れ子の関係になることはあるが，互いに独立であり，論理ボリュームと物理ボリュームの関係は 1 対多であった．連鎖ネットワーク RAID は，論理ボリュームと物理ボリュームに多

対多の依存関係を持たせることを特徴としており，多くの物理ボリュームから 1 つの RAID グループを構成し，複数の論理ボリュームを提供する．

連鎖ネットワーク RAID 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 (Add/Remove physical)
- 論理ボリュームの追加と削除 (Add/Remove logical as physical)
- 物理ボリュームと論理ボリュームの結びつけ (Bind/Unbind physical with logicals ...)

まず，物理ボリュームの追加によって，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．ここで連鎖ネットワーク RAID は，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．論理ボリュームの追加時に，他の論理ボリュームに結び付けられていない物理ボリュームが 1 つ指定され，そのデータが論理ボリュームの初期データとなる．最後に，論理ボリュームと物理ボリュームの結びつけによって，論理ボリュームのデータは他の複数の物理ボリュームにも格納され，RAID グループが構成される．ここで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．また，論理ボリュームと物理ボリュームはすべて同じサイズであるとする．

連鎖ネットワーク RAID の特長は，任意の論理ボリュームと物理ボリュームを多対多の関係になるように結びつけることができ，そこに XOR 演算を用いたパリティが生成される点にある．類似の機能を提供する方式として Logical Volume Manager(LVM) があるが，LVM では物理ボリュームが複数の論理ボリュームに結び付けられることはなく，論理ボリュームと物理ボリュームの関係は一対多である．また，連鎖ネットワーク RAID は高信頼化の機能を提供するが，LVM のような仮想化の機能を完全に提供するものではなく，LVM と組み合わせて用いることを想定している．

3.2 RAID の構成例

連鎖ネットワーク RAID による RAID の構成例を示す．まず，冗長化のない最も基本的な構成が図 1 の (a) である．物理ボリューム P0 は論理ボリューム L0 として参照される．次に，RAID4 に相当する冗長化を行った構成が図 1 の (b) である．物理ボリュームのいずれか 1 個が故障しても，論理ボリュームのデータは失なわれず，論理ボリュームへの要求を継続して処理できる．そして，さらなる冗長化を行った構成が図 1 の (c) である．これは連鎖ネットワーク RAID の特長が出ている構成であり，論理ボリューム L1 は，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すると共に，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，同様にデータは失われず，いずれか 3 個が故障しても，80%の確率で同様にデータは失われない．例えば，物理ボリューム P1,P4,P5

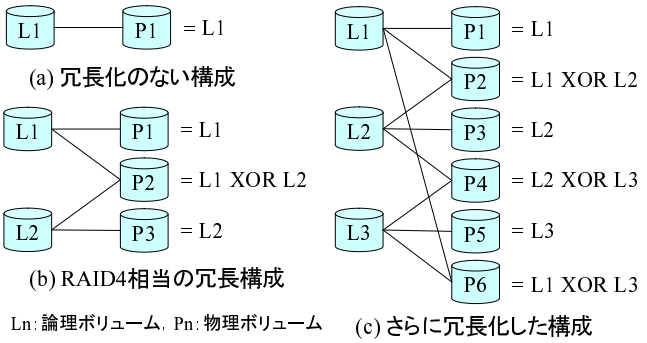

図 1　連鎖ネットワーク RAID の構成例

が残った場合，論理ボリューム L1 は P1 により，L2 は P1 XOR P4 により，L3 は P1 XOR P4 XOR P5 により得ることができる．

また，より多くの物理ボリュームが故障しても，確実にデータが失われない構成が可能である（付録参照）．更に，より大規模になった場合でも，ヒューリスティックに高信頼な RAID が記憶効率よく構成できることが検証できている [12].

こうした構成は一旦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，ノードの追加やノードの交換など構成の変更を柔軟に行うことができる．また，Reed-Solomon 符号を用いた方式に比べて記憶効率は劣るが，構成を選ぶことでパリティを保持する特定のノードにアクセスが集中しないようにすることができる．

3.3 物理ボリュームの状態

物理ボリュームが障害・停止状態から復旧した時や 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．

- 同期がとれており，読み書きできる状態 (SYNC)
- 再構築中であり，書き込みのみ行うべき状態 (REBUILDING)
- 同期がとれておらず，利用できない状態 (TAINT)

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．この変換処理で対象となる物理ボリュームは，その状態が SYNC のボリュームのみである．ある物理ボリュームに注目すると，その物理ボリュームに結びついた論理ボリュームのうち，1 つの論理ボリュームを除いてデータが取得可能であるとき，その論理ボリュームはその物理ボリュームを用いることでデータが取得可能になる．この規則を利用して，すべての物理ボリュームを繰り返し評価することで，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．連鎖ネットワーク RAID が物理ボリュームのリストを得るためのアルゴリズムを図 2 に ruby 風の擬似コードで示す．これらの物理ボリュームの XOR 演算結果が論理ボリュームのデータとなる．

```
# 探索フェーズ
logicals.each { |logical|
  logical.requisite_physical = nil
}
found_flag = 1
while (found_flag != 0) {
  found_flag = 0
  physicals_status_is_sync.each { |physical|
    expectant_logicals.clear
    physical.binded_logicals.each { |logical|
      if (logical.requisite_physical == nil)
        expectant_logicals.push (logical)
    }
    if (expectant_logicals.size == 1) {
      the_logical = expectant_logicals.pop
      the_logical.requisite_physical = physical
      found_flag = 1
    }
  }
}
# 変換フェーズ
untranslated_logicals = [ requested_logical ]
translated_physicals.clear
while (untranslated_logicals.size != 0) {
  the_logical = untranslated_logicals.pop
  physical = the_logical.requisite_physical
  if (physical == nil)
    return nil  # 変換に失敗
  translated_physicals.push (physical)
  physical.binded_logicals.each { |logical|
    if (logical != the_logical)
      untranslated_logicals.push (logical)
  }
}
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

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，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．読み出し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．

この変換処理で書き込みの対象となる物理ボリュームは，要求の対象である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 SYNC と REBUILDING のボリュームである．その物理ボリュームの状態が REBUILDING である場合は，前節で示した読み出し要求の変換を用いることで，結びつけられたすべての論理ボリュームのデータが得られる．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書き込み要求で新たに書き込むデータとの XOR 演算結果よりその物理ボリュームに書き込むデータが得られる．また，その物理ボリュームの状態が SYNC である場合は，前節で示した読み出し要求の変換を用いることで，書き込み対象の論理ボリュームのデータが得られる．このデータと論理ボリュームに書き込むデータとその物理ボリュームのデータの XOR 演算結果よりその物理ボリュームに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は，前者とほぼ同様の手順で処理される．前節で示した読み出し要求の変換を用いることで，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が得られる．これらのデータの XOR 演算結果が，書き込むデータとなる．

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．この時，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．例えば，図 1 の (b) の構成の場合，論理ボリューム L0 と L1 への書き込みは，どちらも物理ボリューム P1 への読み出しと書き込みを伴う．これらの処理が，L0 のための読み出し，L1 のための読み出し，L0 のための書き込み，L1 のための書き込みの順で行われてしまうと，L0 への書き込みによる P1 のデータの更新が失われてしまい，物理ボリューム P1 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，RAID 全体での要求処理の並列性を保ったまま，データの整合性を維持することができる．

3.7　システム設計

これまでに述べた処理を各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ノードで行うと，その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 をやり取りして，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

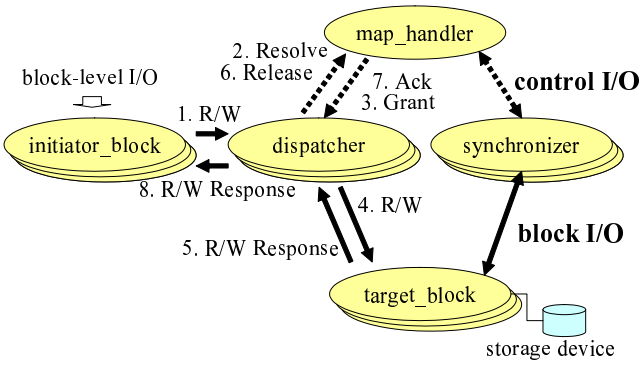

図 3　連鎖ネットワーク RAID の構成部品

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

物理ボリュームが故障している場合など，dispatcher からの Resolve に対して，map_handler がすぐに Ack を返すことや，dispatcher からの Release に対して，map_handler が再度異なる Grant を返すこともある．

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 により処理される．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．

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 を Linux SCSI target framework [13] の backed_io_template として実装することで，Linux SCSI target framework が提供する iSCSI [14] 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れるようにした．

現時点の実装上の制約として，map_handler が排他制御のために管理する物理ボリュームの読み書きロックは，ボリューム単位にまとめて一つだけしか設けていない．また，dispatcher が同時に処理できる読み書き要求は，一つに制限されている．

表 1　実験機材の仕様

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

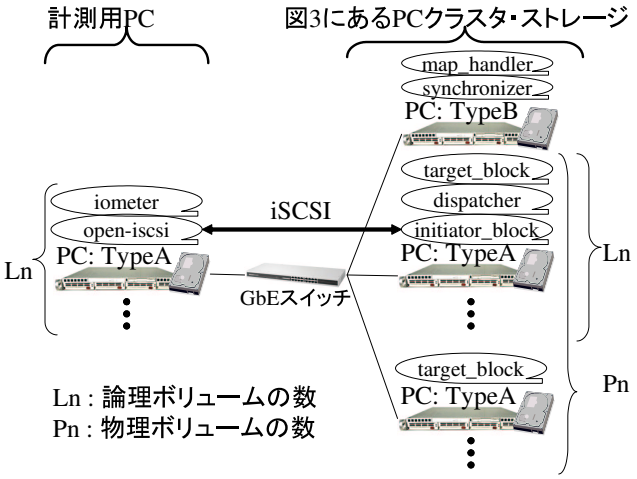

図 4　実験環境のシステム構成

4.　評　　価

4.1　実 験 環 境

プロトタイプ実装を用いて連鎖ネットワーク RAID のスループットを評価する．まず，実験に用いた PC の仕様を表 1 に示す．PC 間は 1000Base-T の GbE スイッチで接続した．そして，プロセスを図 4 のように配置した．論理ボリュームへの要求を受ける dispatcher とその論理ボリュームが占有する物理ボリュームを同じ PC に配置し，物理ボリューム同士はすべて別の PC になるように配置した．initiator_block の Linux SCSI target framework の実装には，revision 616 [15] を用いた．

また，論理ボリュームと同じ数だけの TypeA の PC を計測用として用いた．計測用の PC は，open-iscsi [16] を用いて iSCSI セッションを確立し，論理ボリュームをディスクドライブとして認識する．

4.2　スループット

計測用 PC 上で，iometer (version 2006-07-27) [17] を用いてブロックレベルのシーケンシャルな読み書き要求を発生させ，1K, 4K, 16K, 64K, 256K バイトのリクエストサイズに対するスループットを計測した．各計測用 PC には iometer の Worker スレッドを一つだけ動作させ，同時に発行する I/O の数を 1 に設定した．

読み出しのスループットを図 5 に，書き込みのスループットを図 6 に示す．まず比較のために，TypeA の PC で Linux SCSI target framework だけを動作させた場合のスループットを計測し，図中の iSCSI Direct に示した．そして，図 1 の (b) の構成

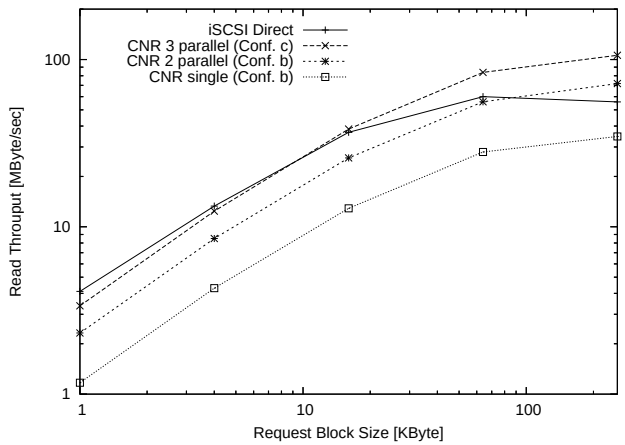

図 5　読み出しのスループット

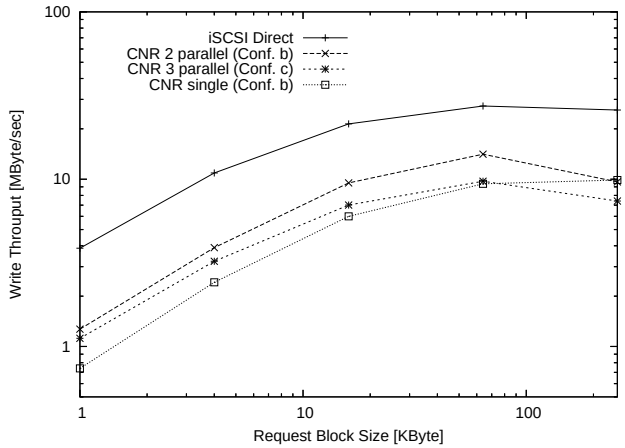

図 6　書き込みのスループット

で 1 個の論理ボリュームに読み書き要求を行った場合のスループットを計測し，図中の CNR single に示した．また，同じ構成で 2 個の論理ボリュームに同時並行に読み書き要求を行った場合の合計のスループットを計測し，図中の CNR 2 parallel に示した．最後に，図 1 の (c) の構成で 3 個の論理ボリュームに同時並行に読み書き要求を行った場合の合計のスループットを計測し，図中の CNR 3 parallel に示した．

図中の iSCSI Direct と CNR single を比較すると，連鎖ネットワーク RAID のオーバーヘッドが分かる．読み出しでは，リクエストサイズが小さい場合に 1/4 程度に，リクエストサイズが大きい場合に 1/2 程度に，スループットが落ちていることが分かる．

スループット低下の原因は図 3 に示したように，コンポーネント間のソケット通信が行われる回数が増えるためである．リクエストが処理されるまでの応答時間が長くなるため，それに反比例してスループットが低下している．リクエストサイズが小さいほどその影響が顕著に現れ，スループットが低下していることが分かる．

また，書き込みでは，リクエストサイズが小さい場合に 1/5 程度に，リクエストサイズが大きい場合に 1/3 程度に，スループットが落ちていることが分かる．読み出しに比べて書き込みのオーバーヘッドが大きい理由は，3.5 節で示したように，論理ボリュームへの書き込み要求が物理ボリュームへの読み込みと書き込みに変換されて，read-modify-write の順に実行されるためである．

図中の CNR single，CNR 2 parallel と CNR 3 parallel により，連鎖ネットワーク RAID のシステム全体のスループットが分かる．読み出しでは，論理ボリュームの数に比例して，スループットは 2 倍，3 倍となっている．制御 I/O とブロック I/O を分離したことで，特定のノードに負荷が集中せずに，システム内で読み出し要求が並列して処理されていることが分かる．このように連鎖ネットワーク RAID は，個々のリクエストの応答時間は長くなるものの，システム全体で高いスループットを実現できる．

また，書き込みでは，論理ボリュームの数に比例したスループットの増加は見られない．これは，異なる論理ボリュームへの書き込み要求が，同じ物理ボリュームへの書き込みを必要とするため，並列して処理されないように排他制御されているからである．3.8 節で述べたとおり，現状は物理ボリュームの読み書きロックは，ボリューム単位にまとめて一つだけしか設けていないという実装上の制約がある．そのため，図 1 の (b) と (c) の構成では，いずれの論理ボリュームへの書き込み要求も互いに競合するため，1 個の論理ボリュームへ書き込み要求を行った場合と同等のスループットしか得られていない．しかし，本来は同じ物理ボリュームへの要求であっても同じブロックアドレスへの要求でなければ，並行して処理することができる．要求の対象となる領域はボリューム全体のごく一部に過ぎないため，連鎖ネットワーク RAID において本質的に書き込みの並列性を保つことは可能である．具体的には，map_handler の読み書きロックの機能をボリューム単位からより粒度の細かい単位に改善することで，書き込みについても読み込みと同様なシステム全体のスループット向上が期待できる．

5.　まとめと今後の課題

我々は，ブロックレベルの PC クラスタ・ストレージにおけるデータの冗長化方式として，連鎖ネットワーク RAID を提案した．本手法が論理ボリュームと物理ボリュームを多対多の関係で結びつけることで，2 台以上のノードの故障に耐えうる高い信頼性を実現できることを示した．そして，プロトタイプ実装を用いた評価により，本手法が冗長化を行わない場合の iSCSI アクセスと比べて，読み出し時に 1/4 以上，書き込み時に 1/5 以上のパフォーマンスを達成することを確認し，読み書き可能なブロックレベルのストレージを提供できることを確認した．また，読み出し時にはシステムのノードの台数に比例したパフォーマンスを提供できることを確認した．

今後の課題として，次の事があげられる．まず，4.2 節で述べたように，要求間の排他制御をブロックアドレスなどの細かい粒度で行うことで，書き込み時にもシステムのノードの台数に比例したパフォーマンスを提供できるようにする．次に，ボリュームサイズが同じであるとする仮定を取り除き，結びつけ処理でボリュームサイズやオフセットを扱えるようにする．最後に，map_handler が処理できる I/O のスループットを評価し，

システムのスケーラビリティを評価する．

付　　録

Lx は論理ボリュームの識別子，Px は論理ボリュームの識別子，+は Exclusive OR 演算を表す．

9 個の物理ボリュームから 4 個の論理ボリュームを提供し，いずれか 3 個の物理ボリュームが故障しても，データが失われない構成は次の通り．P1 = L1．P2 = L2．P3 = L3．P4 = L4．P5 = L1 + L2．P6 = L2 + L3．P7 = L3 + L4．P8 = L4 + L1．P9 = L1 + L2 + L3 + L4．

13 個の物理ボリュームから 5 個の論理ボリュームを提供し，いずれか 4 個の物理ボリュームが故障しても，データが失われない構成は次の通り．P1 = L1．P2 = L2．P3 = L3．P4 = L4．P5 = L5．P6 = L1 + L2．P7 = L2 + L3．P8 = L3 + L4．P9 = L4 + L5．P10 = L5 + L1．P11 = L1 + L2 + L3．P12 = L3 + L4 + L5．P13 = L1 + L2 + L4 + L5．

18 個の物理ボリュームから 6 個の論理ボリュームを提供し，いずれか 5 個の物理ボリュームが故障しても，データが失われない構成は次の通り．P1 = L1．P2 = L2．P3 = L3．P4 = L4．P5 = L5．P6 = L6．P7 = L1 + L2．P8 = L2 + L3．P9 = L3 + L4．P10 = L4 + L5．P11 = L5 + L6．P12 = L6 + L7．P13 = L1 + L2 + L3．P14 = L2 + L3 + L4．P15 = L3 + L4 + L5．P16 = L4 + L5 + L6．P17 = L5 + L6 + L1．P18 = L6 + L1 + L2．

文　　献

[1] S. Ghemawat, H. Gobioff, and S.-T. Leung, “The google file system,” ACM SOSP, October 2003.

[2] Isilon IQ Clustered Storage, http://www.isilon.com/.

[3] M. D. Flouris and A. Bilas, “Clotho: Transparent Data Versioning at the Block I/O Level,” MSST2004, April 2004.

[4] NILFS: log-structured file system developed for the Linux, http://www.nilfs.org/.

[5] D. Patterson, G. Gibson, and R. Katz, “A Case for Redundant Array of Inexpensive Disks (RAID),” Proc. of ACM SIGMOD’88, pp.109–116, June 1988.

[6] Q. Xin, E. L. Miller, and T. J. E. Schwarz, “Evaluation of Distributed Recovery in Large-Scale Storage Systems,” HPDC-13 ’04, pp.172–181, 2004.

[7] A. Merchant, and P. S. Yu, “Analytic Modeling of Clustered RAID with Mapping Based on Nearly Random Permutation,” IEEE Transactions on Computers, Vol.45(3), pp.367–373, March 1996.

[8] H. P. Anvin, “The mathematics of RAID-6,” http://kernel.org/pub/linux/kernel/people/hpa/raid6.pdf.

[9] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leoung, and S. Sankar, “Row-Diagonal Parity for Double Disk Failure Correction,” Proc. of the 3 rd USENIX Conference on File and Storage Technologies, San Francisco, CA, March 2004.

[10] H. Weatherspoon, and J. D. Kubiatowicz, “Erasure Coding vs. Replication: A Quantitative Comparison,” Proc. of IPTPS’02, March 2002.

[11] Y. Chen, J. Edler, A. Goldberg, A. Gottlieb, S. Sobti, and P. Yianilos, “A prototype implementation of archival intermemory,” Proc. of the 4th ACM Conference on Digital Libraries, pp.28–37, 1999.

[12] 市川, 高橋, “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.

[13] 藤田, “Linux におけるストレージシステムフレームワークの実現,” 情報処理学会誌:コンピューティングシステム, Vol.47, No.SIG 12(ACS15), pp.411–419, May 2006.

[14] J. Satran, K. Meth, C. Sapuntzakis, M. Chadalapaka, and E. Zeidner, “RFC3720: Internet Small Computer Systems Interface (iSCSI),” April 2004.

[15] Linux SCSI target framework (tgt) project, http://stgt.berlios.de/.

[16] Open-iSCSI, http://www.open-iscsi.org/.

[17] Iometer, http://www.iometer.org/.