自律ディスクの仮想化における 障害回復の高速化とアベイラビリティの向上

山口　宗慶[†]　渡邊　明嗣[††]　花井　知広[††]　田口　亮[†††]　林　直人[†††]

上原　年博[†††]　横田　治夫[††††]

† 東京工業大学 工学部 情報工学科; 152-8552 東京都目黒区大岡山 2-12-1

†† 東京工業大学 大学院 情報理工学研究科 計算工学専攻; 152-8552 東京都目黒区大岡山 2-12-1

††† NHK 放送技術研究所; 157-8510 東京都世田谷区砧 1-10-11

†††† 東京工業大学 学術国際情報センター; 152-8550 東京都目黒区大岡山 2-12-1

E-mail: †muu@de.cs.titech.ac.jp, ††{aki,hanai}@de.cs.titech.ac.jp, †††{taguchi.r-cs,hayashi.n-gm,uehara.t-jy}@nhk.or.jp, ††††yokota@cs.titech.ac.jp

あらまし　近年のディスク容量の増加により，故障によるサービスの停止時間への影響が大きくなってきている．復旧速度を上昇させる事は，高いアベイラビリティを保つことにおいて非常に重要である．我々の提案している自律ディスクでは，データ配置や負荷分散，故障対策や障害回復などのデータ管理を自律的に行う．本稿では，自律ディスクの復旧時間を短縮するために，スタガード配置の改善を提案する．ディスクの仮想化を導入することによって復旧の並列化を行う．また，自律ディスク内部の仮想ディスクの数の影響を考察する．

キーワード　自律ディスク, 負荷分散, 耐故障, データ配置, リカバリ

Improvement in Recovering Speed and Availability by Virtualizing Autonomous Disks

Munenori YMAGUCHI[†], Akitsugu WATANABE[††], Tomohiro HANAI[††], Ryo TAGUCHI[†††], Naoto HAYASHI[†††], Toshihiro UEHARA[†††], and Haruo YOKOTA[††††]

† Department of Computer Science, Faculty of Engineering, Tokyo Institute of Technology 2-12-1 Oookayama Meguro Tokyo, 152-8552 Japan

†† Department of Computer Science, Graduate School of Information Science and Engineering, Tokyo Institute of Technology

††† NHK Science & Technical Research Laboratories 1-10-11 Kinuta Setagaya Tokyo, 157-8510 Japan

†††† Global Scientific Information and Computing Center, Tokyo Institute of Technology

E-mail: †muu@de.cs.titech.ac.jp, ††{aki,hanai}@de.cs.titech.ac.jp, †††{taguchi.r-cs,hayashi.n-gm,uehara.t-jy}@nhk.or.jp, ††††yokota@cs.titech.ac.jp

Abstract　Since the recent increase of disk capacity enlarges the influence of disk failures with lengthening the suspension of service, the speedup of recovery process becomes more important to derive the high availability. We have proposed the Autonomous Disks capable of managing data, such as allocating data with balancing load, handling faults, and recovering failures. In this paper, we propose to improve staggered data allocation used in the Autonomous Disks to shorten the recovery period. We parallelize the recovery process by introducing virtual disks, and consider the effect of the number of virtual disks within an Autonomous Disk.

Key words　Autonomous Disks,Load Balancing,Fault Tolerance,Data Allocation,Recovery

1. 序　　論

1.1 はじめに

今日，コンピュータで利用する情報量は増加し続けており，それに伴い，ディスクの容量も急激に増大してきている．しかし，ディスクは故障する可能性がある．ディスクの故障によるデータの損失はディスクの増大に伴い大きくなってきている．このため，ディスク故障によって，長時間のサービス停止が引

き起こされるようになってきた．この問題はネットワーク環境の普及によってさらに深刻化しており，経済的な損失を回避するために，情報ストレージシステムのアベイラビリティの向上が必要とされている．そのためには，データの多重化と回復処理が必須である [1].

データの多重化や回復処理の管理コストを削減するための手段として我々の提案している自律ディスク [2] がある．自律ディスクではディスク上のプロセッサを利用して，ディスク制御や負荷分散，故障対策，障害回復などのデータ管理を行う．また，自律ディスクではディスク制御をディスク側で行うことによって，ホストがディスククラスタ内の格納データの位置などを意識することなくアクセスが可能である．現在の自律ディスクにも故障対策はされているが，アベイラビリティの向上にはさらなる回復処理の高速化，高信頼化が必要である．

自律ディスクのデータ配置には Fat-Btree [3] を用いた並列インデックス構造を用いている．Fat-Btree では値域分割を用いており，値域が隣同士にある 2 つのインデックスの統合が容易である [4]．ディスク故障時のデータ保証をするためのデータ多重化において，スタガード配置 [5] が有効であり，特に隣に 1 つだけシフトさせたスタガード配置は，プライマリデータとバックアップデータの値域が隣同士となっている．そのため，Fat-Btree を用いた自律ディスクでは，このスタガード配置を用いることにより復旧時間を短縮できる．

本稿では，自律ディスクにおけるデータ分割とその配置方法，復旧戦略を考えることにより，その高信頼化・高速化をめざす．データの分割は自律ディスクの仮想化によって実現する．また，特にマルチメディアデータなどの大容量データを扱うものとする．そのため，ログを使ったバックアップ方式を用いた場合，ログの書き込みによるスループットの低下が著しい．そこでバックアップ方式には，バックアップをする際のコストが小さい同期バックアップを用いるものとする [6].

2. 自律ディスクの概要

2.1 自律ディスクの特徴

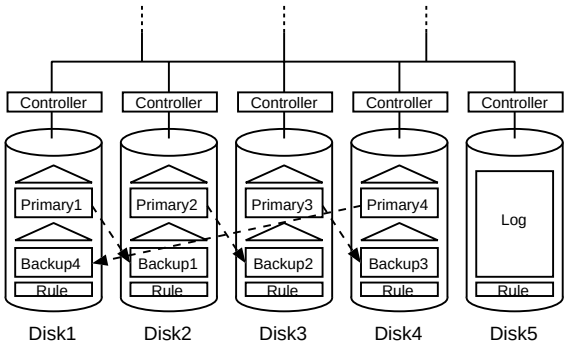

図 1　自律ディスク

ここではまず自律ディスクについて説明する．図 1 に標準的な自律ディスクのクラスタの例を示す．自律ディスクはネットワーク環境でディスククラスタを構成することを前提としている．ホストはデータにアクセスするためにクラスタ内の任意のディスクに要求を発する．クラスタ内のディスクはディスク間の局所的な通信を行うことで，協力してホストからの要求に対処する．このような前提のもとで，自律ディスクは以下のような性質を持つ [2].

データ分散　データは分割され，クラスタ内の複数のディスクに分散される．

ホストからの均質的なアクセス　クラスタ内のいかなるディスクに格納されているデータに対するアクセスの要求も，クラスタ内の全てのディスクで受け付けられる．このために各ディスクはデータ配置に関する情報を含んだ分散ディレクトリ構造を持っている．

同時実行制御　データが複数のホストから同時にアクセスされた際の同時実行制御は対象データが格納されている場所で行われる．

偏り制御　クラスタ内のディスク間でデータ分散の偏りや負荷の偏りが生じた場合は，その偏りは検出され，偏り制御プロセスが起動される．

耐故障性　クラスタ内のディスクの故障やディスクコントローラのソフトウエアバグが発生した場合，それらの障害は自動的にマスクされる．そのために，クラスタ内にプライマリデータとバックアップデータを持ち，障害時に自動的に復旧される．

異種性　上記の性質の実現方法や自律ディスク自体の情報を外部から隠蔽するように，自律ディスク–ホスト間のインターフェース及び自律ディスク–自律ディスク間のアクセスはストリームインターフェースを持つ．

これらの性質の主な長所は，ホストからの透過性とシステムのスケーラビリティである．データ分散の仕方，偏り制御の方法，耐故障性，異種性等に関してホストは関与しないため，分散ディスク制御に対するホストのオーバーヘッドをかなり減少させることができる．

2.2 自律ディスクの障害回復

自律ディスクは Fat-Btree を用いた並列インデックスによって，各ディスクのデータを管理している．インデックスは各データの位置情報をもっており，データをアクセスする際には，これを探索することによってデータの位置を特定する．

本稿では，このプライマリのインデックスとプライマリデータの組をプライマリセットと呼ぶ．バックアップについても同様に，バックアップセットと呼ぶ．同じデータに対するプライマリセットとバックアップセットは，インデックスが同じ内容を指し示しているのであって全く同じ物ではないが，内容的にはほぼ同一のものである．そのため，どちらのデータも一方と置換しても動作に全く支障が無いため，全置換によるデータ復旧が可能となり，復旧時間の短縮となる．

ここで自律ディスクにおいての定常状態を定義する．定常状態とは，全てのデータにアクセスすることが可能で，任意のディスクが故障しても復旧可能な状態とする．

自律ディスクのディスク故障検出から定常状態へ戻るまでに，大きく次の 3 つのフェーズに分けられる．

（1） プライマリデータの復旧
全てのデータがアクセス可能な状態にすること．しかし，プライマリデータの復旧が達成された状態では，バックアップが存在しないプライマリデータが存在するため，さらなる故障に耐えることができない．そのため，さらなる故障によるデータの損失が起こる可能性がある．

（2） バックアップデータの復旧
各ディスクのプライマリデータに対して，そのバックアップがプライマリデータとは異なる物理ディスクに存在する状態にすること．バックアップデータの復旧が達成された状態では，さらなる故障による自律的な定常状態の復旧が行われない．しかし，データ分散は終了しているため，故障によるデータ損失はない．

（3） 定常状態の復旧
全てのデータが各ディスクに格納された後，定常状態へ移行すること．

これら 3 つのフェーズを完了し，定常状態へ戻った時点で自律ディスクが障害から回復したとする．

3. 仮想ディスクの導入

1 つのディスク内のプライマリデータを 1 つのディスクに全てバックアップをとると，故障した際に故障したディスクのプライマリデータのバックアップを持つディスクだけがプライマリデータの復旧を行うこととなる．復旧の高速化のためには並列化が有効であり，復旧を複数台のディスクで並列化するには，1 つのディスクのバックアップを複数台のディスクに分割せねばならない．データを分割して格納するということを考えた場合，1 物理ディスク内に仮想的な自律ディスクを複数存在していると考えることができる．1 つのディスクで複数の自律ディスクソフトウェアを実行することによって，複数の仮想自律ディスクを 1 つの物理ディスク内に作ることができる．ここではまず仮想ディスクを増加させることによる影響を考察する．

3.1 仮想ディスクによる影響

データを n 分割した場合，対称性を考慮しなければ，$n!$ 通りのプライマリの配置方法と，それに対するバックアップの配置が存在する．配置方法にも依存するが，考えられる影響を述べる．

3.1.1 復旧時間

既存の方法と異なり，複数のディスクに 1 つのディスクのデータを分散することが可能である．復旧時には，複数台のディスクを用いて並列に復旧処理を行うことにより時間の短縮が可能である．

3.1.2 負荷分散

自律ディスクは機能の一つとして，負荷が大きいディスクの負荷を軽減するようにデータを移動させる．複数のディスクにおいて並列に復旧処理が行われた場合，プライマリデータが分散して復旧される．そのため，1 ディスクで復旧を行った場合より，負荷分散がすでに行われた状態で復旧され，その後に行われる負荷分散の分コストが小さくなる．

3.1.3 データ配置

自律ディスクではそれぞれのディスクが自分の値域にあったデータを持つこととなる．単純なデータ配置では，復旧してからのマッピングの再変更にかかるコストが小さい．しかし，物理ディスクをさらに細分化した場合には複雑なデータマッピングのため，バックアップ復旧後の仮想ディスク配置が故障前の規則と異なるものになる可能性がある．そのため，データマッピングの再構築とデータの移動が必要となり，定常状態の復旧コストが高くなる．

3.2 バックアップ配置の条件

データの分割数を増加し，そのバックアップを配置するにあたり，いくつかの条件を満たすと効率のよい配置とすることができる．ここではその条件について考察を行う．

まず，分割されたデータが同じ物理ディスク内にあると，結局は 1 台で複数個のデータ，すなわち分割サイズより大きいデータを復旧することとなる．すなわち，分割による復旧を高速かつ並列に行うため，分割したデータは各ディスクに 2 つ以上あってはならない．

また，前述の通り自律ディスクは Fat-Btree の並列インデックス構造を用いているので，復旧の高速化にはディスク内のプライマリセットとバックアップセットの値域が隣り合っている事が重要である．

以上をふまえると，以下のような条件が導かれる．ただし，n をディスクの総数，p を物理ディスク番号，v を仮想ディスク番号とする．f_p を v から p へのプライマリデータのマッピング関数，同様に f_b を v から p へのバックアップデータのマッピング関数とする．

（1） $if \quad \exists p, \forall v, f_p(v) = p \quad then \quad f_b(v) \neq p$
（2） $if \quad \exists p, \forall v_i, \forall v_j, f_p(v_i) = p, f_p(v_j) = p, v_i \neq v_j$
$then \quad f_b(v_i) \neq f_b(v_j)$
（3） $if \quad \exists p, \forall v, f_p(v) = p \quad then \quad f_b(v \pm 1 \bmod n) = p$

3.3 単純なパターン

条件を満たすようなバックアップの配置を考えるにあたり，マッピングが単純なパターンを考える．ここでの単純なパターンとはマッピングの再計算コストが小さく，前述の定常状態への移行コストが小さいパターンである．特に，バックアップデータの復旧が行われた時点で定常状態への移行が終わっているものが望ましい．このような単純なパターンには，連続した番号付けをしたものと，循環した番号付けをしたものを考えることができる．しかし，1 ディスクに仮想ディスクを 3 つ以上用いて構成する場合のこれらの順番付けは，それぞれにおいて条件を満たすことができない．以下に簡単な証明を示す．

3.3.1 連続した番号付け

物理ディスク内に存在する全ての仮想ディスクの値域が隣合っているものである．

式では $f_p(v) = \lceil \frac{v}{n} \rceil, v_i = v_j - 1, v_k = v_j + 1, f_p(v_i) = p, f_p(v_j) = p, f_p(v_k) = p,$ の様に表される配置を考える．

条件 (3) を考えると $f_p(v_j) = p$ であるから $f_b(v_j \pm 1 \bmod n) = p$ すなわち $f_b(v_i) = p$ または $f_b(v_k) = p$ である．ここで，条件 (1)

より $f_b(v_i) \neq p$ かつ $f_b(v_k) \neq p$ であるため矛盾する．よって，連続した番号付けにおける条件を満たすバックアップ配置は存在しない．

3.3.2 循環した番号付け

物理ディスクに順番をつけ，その順序に従い各物理ディスク内の仮想ディスク 1 つに順次値域を割り振っていくパターンである．全ての物理ディスクに 1 つずつ地域を割り振った後，同じ順序でそれぞれの仮想ディスクに再び値域を割り振る．

式では $f_p(v) = v \bmod n, v_i = v_j - n, v_k = v_j + n, f_p(v_i) = p, f_p(v_j) = p, f_p(v_k) = p$ の様に表される配置を考える．

条件 (3) を考えると $f_p(v_j) = p$ であるから $f_b(v_i \pm 1 \bmod n) = f_b(v_j \pm 1 \bmod n) = p$ である．同様に考えると $f_b(v_i \pm 1 \bmod n) = f_b(v_j \pm 1 \bmod n) = f_b(v_k \pm 1 \bmod n) = p$ である．ここで，± を考慮しても，条件 (2) を満たすことはできないことから矛盾する．つまり，循環した番号付けにおける条件を満たすバックアップ配置は存在しない．

3.4 配置シミュレータ

上記では単純なパターンについて 3 分割以上では条件を満たすことができないことを示した．そこで 2 分割の場合と，その他の 3 分割以上の配置で効率がよいものを探すために，プライマリセットとバックアップセットの配置をシミュレートするプログラムを作成した．配置をシミュレートによって，ディスク故障前から故障後への適切なバックアップ配置の存在するパターンとマッピングの再計算コストについて考察を行う．このプログラムは n 台のディスクを d 分割した場合の上記の条件を満たすプライマリセットとバックアップセットの配置をシミュレートし，条件を満たす配置を出力する．また，上記の条件を満たさないものを省く機構と，単純なパターンを除く枝刈り機構を実装した．

3.4.1 結　　果

2 分割 3 台では 48 通り，2 分割 4 台では 1092 通りの条件を満たすパターンが発見された．全部のパターンを載せることはできないので，それぞれの場合について発見されたいくつかのパターンを図 2 と図 3 に示す．図中の実線の円筒は物理ディスクを表し，破線の円筒は仮想ディスクを表す．また，三角はインデックスを表し，四角はデータを表している．データ内部の数字は割り当てられた値域を表しており，連続する数字は値域が隣合っている．例外として，最大の番号をもつデータと最小の番号をもつデータも値域が隣合っているとする．ディスク内の上部はプライマリセットを下部はバックアップセットを表している．

3 分割 4 台以上のパターンはシミュレートに非現実的な時間がかかり，結果が得られなかった．

3.4.2 評価と考察

シミュレートは故障前の配置から故障後の配置を探すという性質上，n 台 d 分割と $n+1$ 台 d 分割の結果を比較することにより，適切な配置を見つけねばならない．シミュレータはその性質上，全てのパターンについて考える．しかし，仮想ディスク数とディスク台数が増えるに従い，指数関数的に計算コストが増える．3 分割以上の最も小さなパターンである，3 分割 4

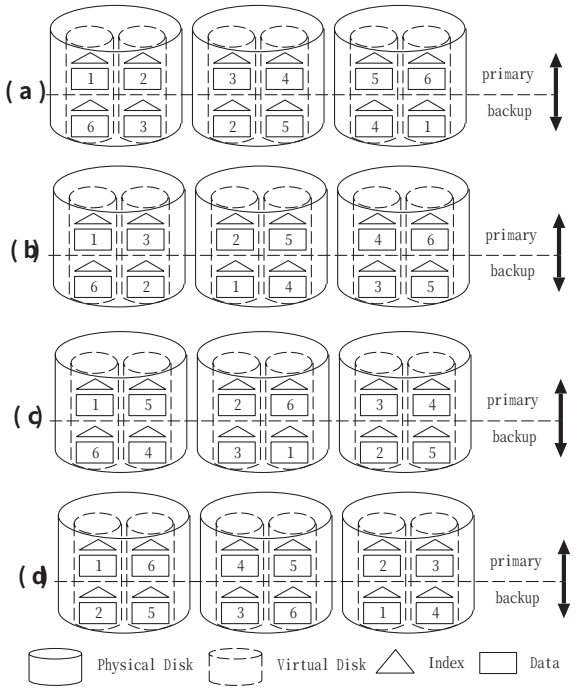

図 2　2 分割 3 台で見つかったパターンの例

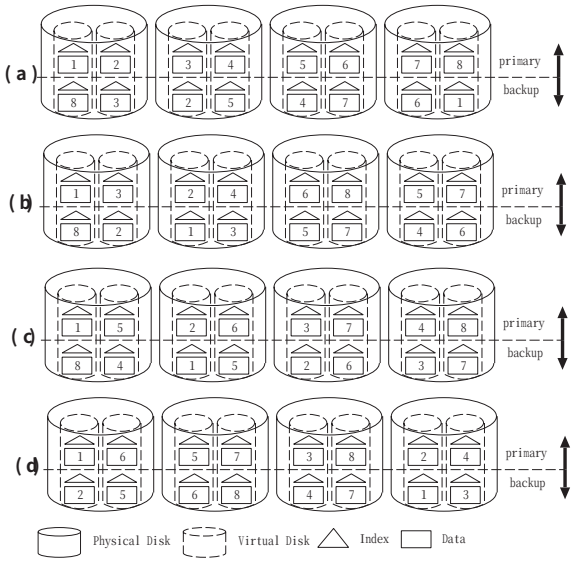

図 3　2 分割 4 台で見つかったパターンの例

台で現実的ではない時間がかかってしまった．

すなわち，この程度の枝刈りのアルゴリズムではマッピングの再計算コストは大きく，実際の復旧過程においてマッピングの再計算を行うことは好ましくない．特に，仮想ディスクのマッピングが複雑になるに従い，定常状態の復旧フェーズにおいて全てのデータのプライマリデータとバックアップデータがそろっているにも関わらず，データの多大な移動が行われてしまう可能性がある．効率のよい，データの移動量の少ない配置を見つけるには，マッピングの計算にさらに時間がかかってしまう．これは，復旧時間を短縮するという点において，大きなマイナス要素である．

発見されたパターンの多くはうまくいかないものが多かったが，2 分割をシミュレートした結果の中に復旧後もマッピングが

変更されないパターンが発見された．図 2 の (a)，図 3 の (a) のパターンである．この 2 つのパターンは同じマッピング方法で配置可能である．次の章ではそのパターンについて考察を行う．

4. バックアップのスタガード配置と復旧方法

上記で述べたとおり，マッピング関数の再計算コストなどを考えると，一般化した n 分割すなわち n 台の仮想ディスクを実現するにはいろいろと問題が多い．しかし，発見された図 2 の (a)，図 3 の (a) ような 1 ディスク 2 仮想ディスクのパターンは，1 ディスク 1 自律ディスクと同様バックアップの復旧が完了した時点で定常状態への復旧も完了する．そのため，マッピングの再計算コストはどちらのパターンでも必要無い．1 ディスク 1 自律ディスクと比較した場合，純粋にメリットだけが大きいパターンであった．ここではバックアップのスタガード配置から，既存の自律ディスクの方式をワンサイドシフト，1 ディスク 2 仮想ディスクをツーサイドシフトと命名する．それぞれの方式のバックアップのスタガード配置とその復旧方法を説明する．

4.1 ワンサイドシフト

現在，自律ディスクに実装されている方法である．図 4 のように，あるディスクのプライマリセットのバックアップを，論理的に隣のディスクへ格納する方式方法である．

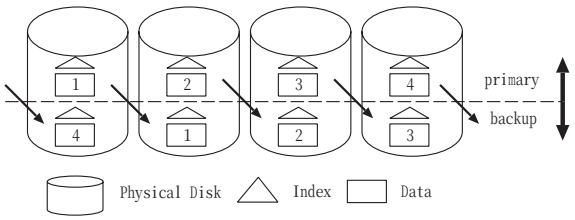

図 4　ワンサイドシフトのデータ配置

4.1.1 配置方法

ディスクの数を m 台とする．i 台目 ($1 \leqq i \leqq m$) のディスクには i 番目のプライマリセットと $i-1$ 番目のバックアップセットが入っているとする．ただし，$i = 1$ の場合，バックアップセットは m 番目のプライマリセットが入っているとする．

4.1.2 復旧手順

ディスク i が故障したとする．図 5 は，$i = 2$，$m = 4$ の場合を表したものである．

（1） ディスク i のバックアップはディスク $i+1$ の内部に存在する．そのバックアップをプライマリへ昇格させ，プライマリのインデックスを統合する．
（2） ディスク $i-1$ のバックアップが消滅したので，ディスク $i-1$ のプライマリセットをディスク $i+1$ のバックアップへコピーする．
（3） 同様にディスク $i+1$ 上のプライマリへ昇格させた，i 番目のデータのバックアップが存在しない為，そのデータをディスク $i+2$ のバックアップへコピーし，そのインデックスを統合する．

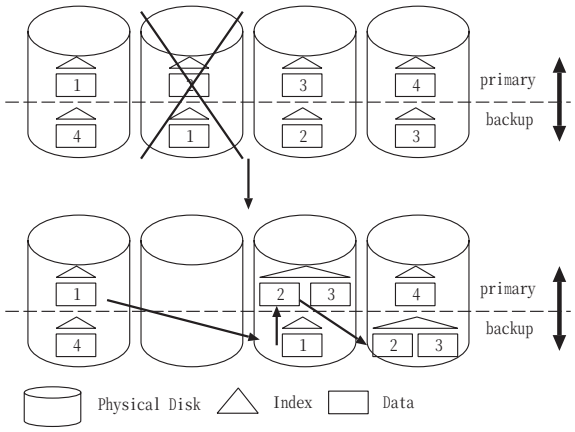

図 5　ワンサイドシフトの復旧

4.2 ツーサイドシフト

図 6 のように，あるディスクのプライマリデータを値域によって分割し，2 つのディスクに格納する方式である．1 つの物理ディスクの中に，2 つの仮想ディスクをもつことによって実現する．それぞれの仮想ディスクがプライマリセットとバックアップセットの組を持つ．このうちバックアップデータは自分の論理的隣の仮想ディスクのプライマリデータのバックアップである．また，前述の条件より 1 つの仮想ディスクにおけるプライマリデータとそのバックアップの元のプライマリデータは異なる物理ディスクに存在している．

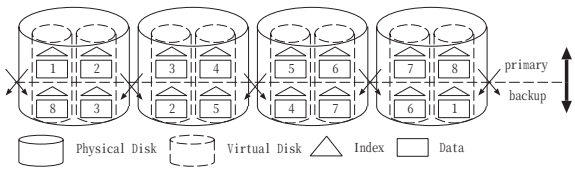

図 6　ツーサイドシフトのデータ配置

4.2.1 配置方法

ディスクの数を m 台とする．i 台目のディスクには $2i-1$ 番目のプライマリセットと $2(i-1)$ 番目のバックアップセットの組と $2i$ 番目のプライマリセットと $2(i+1)-1$ 番目のバックアップセットの組が入っているとする．ただし，1 番目のプライマリセットと組になるバックアップセットは $2m$ 番目のプライマリセットであり，$2m$ 番目のプライマリセットと組になるバックアップセットは 1 番目のプライマリセットである．

4.2.2 復旧手順

ディスク i が故障したとする．図 7 は，$i = 2$，$m = 4$ の場合を表したものである．

（1） ディスク i 内部のプライマリデータ $2i-1$ と $2i$ が消滅している．$2i-1$ 番目のデータのバックアップが存在するディスク $i-1$ と，$2i$ 番目のデータのバックアップが存在するディスク $i+1$ において，それらをプライマリへ昇格させ，そのインデックスを統合する．
（2） ディスク $i-1$ の $2(i-1)$ 番目と $2i-1$ を統合したセットのバックアップが存在しないため，ディスク $i+1$ のバックアップへコピーする．

（3） 同様にディスク $i+1$ の $2(i+1)-1$ 番目と $2i$ を統合したセットのバックアップが存在しないため，ディスク $i+1$ のバックアップへコピーする．

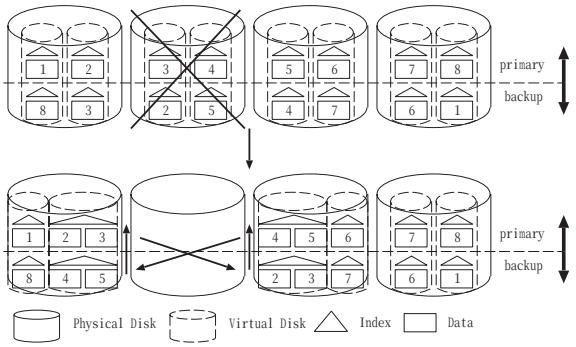

図 7　ツーサイドシフトの復旧

4.3 考　　察

ワンサイドシフトとツーサイドシフトの復旧方法は故障時のデータ移動量においては全く同じである．どちらもディスク 1 つ分のプライマリデータをバックアップからプライマリへと昇格させ，ディスク 2 つ分のデータを複数のディスク間においてコピーを行う．

ここで，現在実装されているワンサイドシフトにおいてボトルネックになるのは，故障したディスク i のバックアップデータが存在しているディスク $i+1$ である．このディスクでは図 5 からもわかる通り，プライマリデータ復旧ではバックアップ生成の為のファイル読み込みとバックアップのデータ生成が行われている．それに対し，ツーサイドシフトはファイルの読み込みとバックアップデータの生成は同じであるが，プライマリデータの復旧は 2 つのディスクにおいて並行して行うことができる．

現在のインデックスの統合アルゴリズムでは，統合する 2 つ双方のデータが小さくなれば時間は短くなる．ワンサイドシフトとツーサイドシフトを比較した場合，ツーサイドシフトの場合は純粋にワンサイドシフトの半分の容量のデータを 2 つ統合することとなるので，プライマリ復旧時間は短くなる．

また，ツーサイドシフトの方がすぐれている点として，3.1.2 の負荷分散がある．[7] にある通り，定常状態で全てのディスクに負荷が均等になるようにしようとする機構が実装されている．ワンサイドシフトでは，ディスク $i+1$ においてプライマリが 2 倍に膨れるのに対し，ツーサイドシフトではディスク $i-1$ とディスク $i+1$ のプライマリデータが 1.5 倍となり，ある程度分散された形で復旧される．

ワンサイドシフトではプライマリデータ復旧にあたりネットワーク帯域を使用しない．このメリットはツーサイドシフトにおいてもいえることである．すなわち，ツーサイドシフトはワンサイドシフトのメリットをそのまま引継ぎ，さらに速度を改善した形である．

5. ツーサイドシフトに関する性能評価

この章ではツーサイドシフトの性能を，ワンサイドシフトと比較して考察する．既存のシステム [6] にはワンサイドシフトが実装されており，このシステムを改造しツーサイドシフトを実装し比較実験を行った．

5.1 実 験 環 境

表 1 に実験システムの性能を示す．

表 1　実験システム

CPU	Intel Pentium3 933MHz
Chipset	Intel i815 (ATA100)
Memory	PC133 (CL=3) 256MB
HDD	Seagate ST320011A(7200rpm, 20GB, Barracuda ATA IV)
Network	1000BASE-SX
OS	Linux 2.2.17，glibc–2.1.3
Java	SUN JRE1.4.1
	SERVER VM
Hub	1000BASE-SX Switching Hub

5.2 復旧速度の測定

53MB のマルチメディアデータを用いて，格納するデータの総量を変化させて時間を計測した．プライマリデータの復旧時間を図 8，バックアップデータの復旧時間を図 9 に示す．

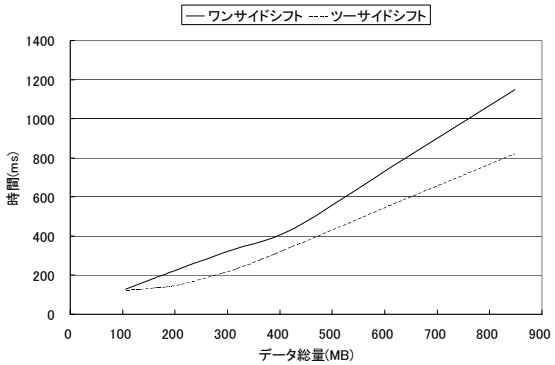

図 8　プライマリデータの復旧時間

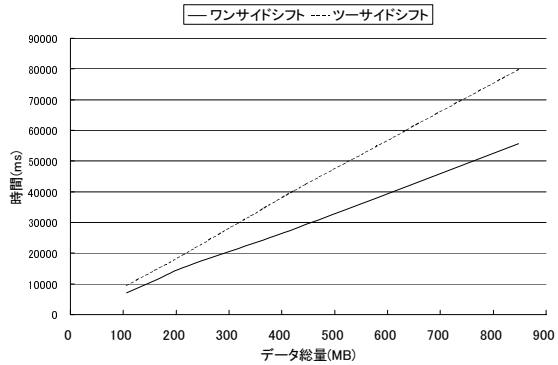

図 9　バックアップデータの復旧時間

5.3 評価と考察

図 8 に関して，復旧時間はデータ量の増大と共に線形に増大する結果となり，プライマリの復旧過程においてデータ量によるスループットの急激な低下はみられなかった．さらに，ツーサイドシフトでは復旧を並列に行うため，予想通りプライマリ

の復旧時間がワンサイドシフトに比べて短くなるという結果が得られた．これはシステムのサービス停止時間が短くになったということで，アベイラビリティの向上になったといえる．ツーサイドシフトは 2 台のディスクで復旧を行うため，その復旧過程のバックアップを生成する際に，バックアップからプライマリへの昇格が両方終わっていなければならない．そのため，早く復旧が終了したディスクがもう一方の復旧を待たねばならなず，復旧時間が待ち時間分増える．また，ディスクが故障を検出する時刻にずれがあるため，それらの差もツーサイドシフトの復旧時間に加わる．結果，バックアップからプライマリへの昇格の時間に加え，故障の検出の差と一方の待ち時間が加わるため，純粋にワンサイドシフトの復旧時間の半分より大きくなった，

図 9 に関して，図 8 同様データ量によるスループットの低下はみられなかった．結果は，ワンサイドシフトにツーサイドシフトが大きく引き離されてしまう結果となった．すなわち，次の故障によってデータが失われなくなるまでにかかる時間がツーサイドシフトの方が長い．これは現在の実装のワンサイドシフトの最も仕事量が多いディスクにおいて，ファイルの読み出しが終了してから書き込みが行われるのに対し，ツーサイドシフトでは復旧を行うディスクが 2 つとも読み出しと書き込みを同時に行っている．そのため，シーケンシャルアクセスからランダムアクセスへと変わり，その差がでていると考えられる．しかし，復旧する際のデータの移動量に違いはないため，ツーサイドシフトに読み出しと書き込みの競合の回避を行うことにより，この差は小さくすることができると考えられる．

6. バックアップ復旧優先ツーサイドシフト

前述のツーサイドシフトの復旧方法では，次の故障によるデータ損失が無い状態へ移行する時間が長くなってしまった．そこで，2 つめのフェーズであるバックアップデータの復旧に注目し，そこにかかる時間を短くする復旧手順を提案する．

これはツーサイドシフトと同じデータ配置で復旧戦略を変更したものである．なお，以下の復旧手順をバックアップ復旧優先と呼び，4.2 の復旧手順をプライマリ復旧優先と呼ぶ．

6.1 復旧手順

ディスク i が故障したとする．図 10 は，$i = 2$，$m = 4$ の場合を表したものである．

（1） ディスク $i-1$ の $2(i-1)$ 番目のプライマリデータとディスク $i+1$ の $2(i+1)-1$ 番目のプライマリデータを同一ディスクのプライマリデータと結合する．

（2） ディスク i 内部のプライマリデータ $2i-1$ と $2i$ が消滅している．$2i-1$ 番目のデータのバックアップが存在するディスク $i-1$ と，$2i$ 番目のデータのバックアップが存在するディスク $i+1$ において，それらをプライマリへ昇格させる．

（3） ディスク $i-1$ の $2(i-1)$ 番目のバックアップが存在しないため，ディスク $i-2$ のバックアップへコピーする．同様に，ディスク $i+1$ の $2(i+1)-1$ 番目をディスク $i+2$ のバックアップへコピーする．

（4） ディスク $i-1$ の $2i-1$ 番目のセットのバックアップが存在しないため，ディスク $i+1$ のバックアップへコピーする．同様にディスク $i+1$ の $2i$ 番目のセットのバックアップが存在しないため，ディスク $i-1$ のバックアップへコピーする．

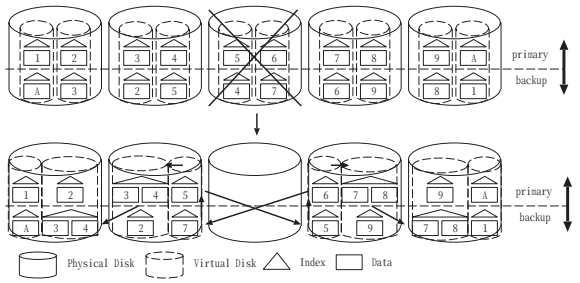

図 10　バックアップ優先ツーサイドシフトの復旧

6.2 考　　察

前述のプライマリ復旧優先のツーサイドシフトに比べて，バックアップ復旧を優先したツーサイドシフトではプライマリ復旧する際にプライマリインデックス同士の統合という操作が入る．このため，インデックス統合する量を考えればプライマリデータの復旧速度はワンサイドシフトと差がなくなることが考えられ，アベイラビリティはプライマリ復旧優先のツーサイドシフトより下がる．

一方で，バックアップ復旧のバックアップ生成過程において，異なる物理ディスクで並列に復旧が可能である．シーケンシャルアクセスのみによるバックアップ復旧の並列化によって時間の短縮になる．また，ツーサイドシフト同様プライマリデータの負荷分散が行われた状態で復旧完了となり，さらにバックアップもツーサイドシフトよりさらに負荷分散が進んだ形で復旧が完了する．

7. おわりに

本稿では，自律ディスクのアベイラビリティを向上させることを目的とし，仮想ディスクの導入によるデータ分割を行った．また，配置シミュレータを使い適切なデータ配置を捜索し，その結果ツーサイドシフトを発見した．そのツーサイドシフトにおいて，プライマリ復旧を優先した復旧方法を提案した．さらに，3 分割以上のマッピングが単純なパターンでは適切な配置が無く，現状では複雑なマッピングを持つパターンの計算コストが高いことを示した．プライマリ復旧を優先した復旧方法を用いたツーサイドシフトを実装し，ワンサイドシフトとの性能測定による比較を行った．その結果，ツーサイドシフトにおいて，プライマリ復旧時間は短くなり，アベイラビリティを向上させることができた．それに対し，バックアップの復旧にかかる時間は増大する結果となった．そのため，バックアップの復旧にかかる時間を減らすバックアップ優先復旧ツーサイドシフトを提案した．

今後の課題として，プライマリ復旧を優先したツーサイドシフトにおいての読み出しと書き込みの競合を回避を組み込んだ

実装があげられる．また，今回実装が間に合わなかったバックアップ復旧を優先したツーサイドシフトの実装と評価も今後の課題である．

復旧の並列化は並列に行える台数が多いほどよいため，アルゴリズム改良による，3 分割以上の配置と復旧方法を検討する必要がある．また，それに伴い，ツーサイドシフトのように配置によっては複数の復旧方法が考えられるものがある．そのような，復旧手順の検討も必要である．

8. 謝　　辞

本研究の一部は，文部科学省科学研究費補助金基盤研究 (14019035) および情報ストレージ研究推進機構 (SRC) の助成により行なわれた．

文　　献

[1] G. Copeland, W. Alexander, E. Boughter and T. Keller: “Data Placement in Bubba”, Proc. of ACM SIGMOD Conf. ’88, pp. 99–108 (1988).

[2] H. Yokota: “Autonomous Disks for Advanced Database Applications”, Proc. of International Symposium on Database Applications in Non-Traditional Environments (DANTE’99), pp. 441–448 (1999).

[3] H. Yokota, Y. Kanemasa and J. Miyazaki: “Fat-Btree: An Update-Conscious Parallel Directory Structure”, Proc. of the 15th Int’l Conf. on Data Engineering, pp. 448–457 (1999).

[4] 宮崎 純 , 横田：“並列ディレクトリ構造 Fat-Btree のリカバリについて”, 信学技法 ,DE2001-107 電子情報通信学会 (2002).

[5] D. J. Dewitt , S. Ghandeharizadeh , D. A. Scheneider , A. Bricker , H. Hsiao , R. Rasmussen: “The Gamma Database Machine Project”, IEEE Transactions on Knowledge and Data Engineering, VOL.2,NO.1 (1990).

[6] 渡邊 明嗣 , 花井 知広 , 山口宗慶 , 横田：“自律ディスクを用いたマルチメディアコンテンツサーバ”, 情処学会研究会報告, データベースシステム DBS-128-1 電子情報通信学会論文誌 vol.j85-D-I NO.9 (2002).

[7] 伊藤大輔 , :“自律ディスク上の分散ディレクトリの負荷均衡機構を用いたクラスタ再構成”, 第 13 回データ工学ワークショップ論文集, DEWS2002 C3–3 電子情報通信学会データ工学研究専門委員会 (2002).