データベースシステム 125－16
（２００１．７．１７）

自律ディスククラスタの活性管理

伊藤大輔† 横田治夫‡
† 東京工業大学 大学院情報理工学研究科 計算工学専攻
‡ 東京工業大学 学術国際情報センター
daisuke@de.cs.titech.ac.jp, yokota@cs.titech.ac.jp

あらまし 我々はネットワーク接続ディスクの負荷分散, 信頼性向上のために自律ディスクを提案している. 自律ディスクはクラスタを構成し, アクセスをクラスタ単位で処理するため, クラスタ再構築をオンラインで行う事が望まれる. 本稿では, クラスタ再構築プロトコルの改良と実験システム上への実装を行う.

キーワード 自律ディスク, ネットワーク接続ディスク, NAS, SAN, クラスタ, 再構築

Online Cluster Reconfiguration of Autonomous Disks

Daisuke Ito† and Haruo Yokota‡
† Department of Computer Science, Tokyo Institute of Technology
‡ Global Scientific Information and Computing Center, Tokyo Institute of Technology
daisuke@de.cs.titech.ac.jp, yokota@cs.titech.ac.jp

Abstract We have proposed an Autonomous Disks for the advanced data management. The Autonomous Disks provide data distribution and high reliability. The Autonomous Disks configure a cluster and handle access from host in the cluster. In this paper, we propose dynamic reconfiguration algorithms allowing the usual operations dulling the reconfiguration, and implement them on an experimental system using Java on PC cluster.

key words autonomous disks, network disks, NAS, SAN, cluster, reconfiguration

1 はじめに

大規模データ処理の分野において, ストレージをシステムの中心に据える構成が注目されている. このような考え方はストレージセントリックと呼ばれている. ストレージセントリックなシステムのスケーラビリティを高めるために, NAS や SAN に注目が集まっている. NAS とは IP ネットワークに直結され, 複数のホストに共有されるストレージデバイスを指す. その結果, 一般に NAS は既存の LAN に接続される. これに対して, SAN は LAN とは異なる独立したネットワークを指す. 一般に SAN はストレージ専用のシリアルコネクションを用いて構築される. 最もよく用いられる物にファイバーチャネルがある.

これらの構成の下では, ディスクはパッシブデバイスとして扱われる. つまり, ネットワーク越しにホストからの命令を受けないとディスクは何もできない. さらに, 命令の分だけネットワークの帯域を無駄に使う事になる. また, システムを効率よく稼動させるためには, レプリカまで考慮に入れたデータ配置を行うことが必要になるが, データ配置は, 通常それ専用の中央サーバで行われる. その結果, データ配置用サーバはデータアクセスの制御まで行う必要に迫られ, システムが大きくなるに連れてボトルネックとなることは避けられない.

ストレージセントリックシステムをスケーラブルにするにはボトルネックを取り除く必要があり, リライアブルにするには複雑な中央管理を止める必要がある. つまり, ストレージノードの自律分散管理こそが効果的である.

ここで技術的な見地から今日のディスクを考えると, 一昔前のコンピュータの CPU よりも演算能力の高いディスクコントローラと, 大容量のキャッシュメモリを内蔵している. この能力の一部をディスク自体の管理に用いる事で, ストレージノードの自律分散管理が安価に実現可能となる. 我々はこのような背景のもとで自律ディスクを提案している [1]. 自律ディスクはネットワークを介した通信コストとホストの負荷を減少させ, フレキシビリティ, アベイラビリティ, スケーラビリティを増大させる.

本稿では, 自律ディスクのクラスタの再構築をオンラインで行う方法を提案する. まず第 2 章で自律ディスクについて簡単に説明する. 次に第 3 章でクラスタ再構築及びそれに伴うデータの移動及びクラスタの構成情報の扱いについて述べる. 第 4 章では PC クラスタ上の模擬自律ディスククラスタ上での実験結果を示す. 第 5 章で関連研究との比較を行い, 最終章で本稿のまとめと今後の課題を述べる.

2 自律ディスク

自律ディスクはネットワーク上でクラスタをなす. データはクラスタ内で適当に分散配置され, 均一なアクセス性を保つ. クラスタはネットワーク越しに複数のホストからの同時アクセスを受ける事ができる. ディスク上のコントローラがデータの分散配置と負荷の偏りを制御し, クラスタとしてのデータ処理能力を高める. また, 自律ディスクは物理ディスク及びコントローラにおける障害への耐性をも高める. 故障ディスクの取り外しやデータ配置変更の際にはクラスタの再構築を行える. これらデータ分散配置, 偏り制御, 対障害制御といった特性はディスクコントローラ上で実現され, ホストからは隠蔽される. そのため中央サーバはもちろん存在せず, システムに高いスケーラビリティをもたらす.

我々は上記の特性を実現するために, Event–Condition–Action (ECA) ルールを用いたコマンド階層とストリームインターフェースを利用することを提案している [1]. ECA ルールはアクティブデータベースで利用されている [2, 3] イベントドリブン型のルールである. また, ストリームインターフェースは NSIC によって提案された OBSD[4] のように, データオブジェクトを論理的に取り扱うことを可能にする. これら二つを組み合わせることで高いフレキシビリティを確保しうる. さらにホストからのアクセス透過性を確保するために, 分散ディレクトリの管理に Fat-Btree[5] を用いる. Fat-Btree とルールを組み合わせることで全てのディスクがクラスタ内の全データの入出力要請を受け付けることが可能となり, またクラスタ内での負荷分散も可能となる.

図 1 に `Insert` リクエスト時のデータフローを例示する. この例は, Host_i が Disk_0 にストリームの `Insert` リクエストを発したが, 並列ディレクトリ上の格納場所は Disk_1 上にあった, というシチュエーションを想定している. Disk_0 に入った `Insert` リクエストは `Insert` ルールを発火し, そのルールによって Disk_0 内で分散ディレクトリをサーチして, 格納場所が Disk_1 であることを知る. その結果, Disk_1 に `Insert` リクエストが転送される. Disk_1 でも同様に `Insert` ルールが発火され, 分散ディレクトリをサーチして適当な位置に格納する. 格納の際には, Write-Ahead-Log (WAL) プロトコルに従いログディスクにログが送られる. 格納後, Host_i に `Insert` 完了が伝えられる.

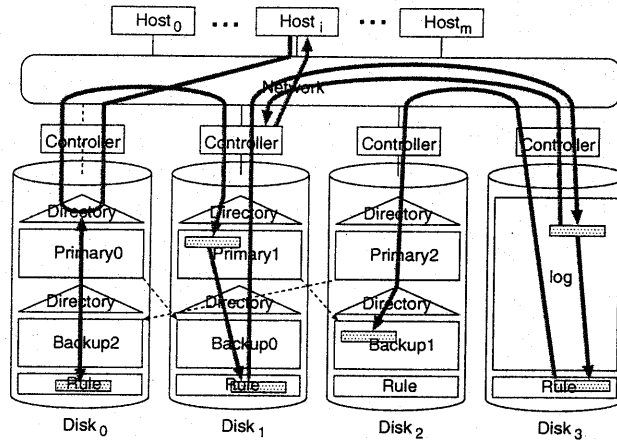

図 1: Insert リクエスト時のデータフロー

ログディスクに送られたログは, ログディスク内のログの量を監視するルールを発火し, ログが一定量を超えるとCatch-up リクエストが発火される. Catch-up リクエストによってログディスク内の全ログが, バックアップ内の適切な位置に格納される.

自律ディスクがリクエストを受けた際の動作はユーザがルールを修正する事で自由に変更可能である. よって, ログディスクの台数やバックアップの数の変更, キャッチアップのトリガの変更が簡単に行える. 自律ディスクのより詳細な説明は論文 [1] にある.

3 自律ディスクのクラスタ再構築

本稿では自律ディスクのクラスタ再構築について述べる. データの整合性を保ちつつクラスタ再構築を行うには, 全自律ディスク間で同期を取る必要がある. 我々はクラスタの再構築を活性で行うプロトコルを提案している [6]. このプロトコルは分散トランザクションで用いられる 2 フェーズコミット (2PC)[7] に良く似ている.

3.1 クラスタ内のデータ配置

システムの信頼性を高めるためには, 冗長性を持たせることが肝心である. データを冗長化させる手法として, パリティを用いる手法がある. これは RAID5[8, 9] で用いられた手法であり, 少ないデータ量で冗長化を行える優れた手法である. しかし, 我々は単純にデータそのものの全コピーを保持する手法を用いる. それはパリティの計算にはディスク間の通信を要するからだ. これは RAID のように中央コントローラの配下で全ディスクが密に結合したシステムでは問題にならないが, 自律ディスクのようにネットワークを介して疎結合したクラスタにではトラフィック増大という問題を引き起こす. また, 近年はディスクのコストが下がり, 少ないデータ量で冗長化を行う必然性が薄れてきたことも要因の一つである.

冗長化とデータ分散を両立させるために, ディスクを複数の領域に分割して用いる. バックアップ用の領域はプライマリ用の領域とは論理的に独立したものとなる. ここで, 一台のディスクの中にプライマリとそれに対するバックアップが同時に含まれることはあってはならないとする. また, 冗長性を高め, 複数故障に耐えうるシステムを作るためには, さらにディスクを分割してバックアップを複数個もたせることが有効である.

プライマリデータが変更された際には, バックアップデータも更新しないといけない. しかしプライマリとバックアップを同時にアップデートすると, ホスト側から見た更新のレスポンスが悪化する. そのため, ログを用いた非同期更新を行う. なお, 後述する我々の実験システムにおいてはログは独立したログディスクに配置される. この配置法はデータ分散という点において, 一見矛盾したものに思える. しかし独立したログディスクには外的要因を受けずにシーケンシャルにアクセスすることが可能であり, ディスクの最大性能を引き出せる. さらに同一のディスクにログとプライマリデータを配置した

場合にはログへのアクセスがプライマリデータへのアクセスを妨げる要因になる．これらの点から，我々はログを独立配置する手法を選んだ．クラスタの規模が大きくなりログがボトルネックとなった場合は，ログディスクを複数台用意する．

バックアップには大きく分けて物理バックアップ方式と論理バックアップ方式の二つの方式がある．物理バックアップ方式とはプライマリとバックアップがページレベルで同一のデータを保持する方式で，対して論理バックアップ方式とは論理的に同一とみなせるストリームを保持する方式である．例えばUNIXのddコマンドやdumpコマンドは物理バックアップを作成し，cpコマンドやtarコマンドは論理バックアップを作成する．我々は論理バックアップ方式を採用する．その理由は自由度の高さにある．論理バックアップ方式では，プライマリとバックアップに別々のディレクトリを割り当てられる．

図1の例では，Disk_i はプライマリの一部 ($\mathrm{Primary}_i$) とバックアップの一部 (Backup_j) を保持している．Backup_j は Disk_j の保持するプライマリに対するバックアップであり，この例では $j = mod(i-1,n)$ としてある．ただし n はデータディスクの台数である．この種の構成はスタッガードアロケーションと呼ばれている [7]．本手法はバックアップの多重化に対応可能であり，例えばバックアップを二重化するには Disk_i に $\mathrm{Primary}_i$ と Backup_j 及び $\mathrm{Backup}_k(k = mod(i-2,n))$ を配置すればよい．

バックアップの配置には他の方法もある．ここではクラスタ内に異なる特性のディスクが存在する場合を考える．ここでディスクの特性とは容量やアクセス速度のことである．各ディスクの保持するプライマリデータ量は特性に応じて変化することが望ましい．例えば速いディスクや大容量ディスクには多くのデータが割り当てられてしかるべきである．$\mathrm{Primary}_i$ と Backup_i のサイズは等しくなるため，この場合には，同じ特性のディスク同士で互いのバックアップを保持するほうがいい．この種の構成はグループアロケーションと呼ばれている．

また，停電や自然災害に備えて地理的にはなれた場所にバックアップを配置したい場合もある．この種の構成はロケーションディペンデントアロケーションと呼ばれている．

ユーザの要求に対してクラスタを柔軟に構成できるようにすることは重要である．さらに，クラスタ再構築後にもクラスタの構成をそのポリシーに従って可能な限り保つことも重要である．今後，これらの異なる構成を単一のECAルールを用いて記述できるようなコマンドを準備する予定である．

3.2 クラスタ再構築の要因

クラスタ再構築の要因は，大きく分けてディスクの故障と運用ポリシーの変更の二つである．

ディスクの故障に関して，我々の想定する自律ディスククラスタにおいてはデータディスク及びログディスクの故障の二つの場合がある．故障はディスク間の通信時に発見される．

運用ポリシーの変更とクラスタの状態変化は以下のように関連付けられる．

- データディスクの追加・・・容量増加，データ分散によるパフォーマンス強化
- ログディスクの追加・・・信頼性向上，ログがボトルネックになった際のパフォーマンス強化
- バックアップの増加・・・信頼性向上，ホットスポットの複製化によるパフォーマンス強化

これらは反対の理由で取り外される事もある．本稿ではこれらを戦略変更に伴うクラスタ再構築の指標とする．図2～4にこれらの操作を示す．

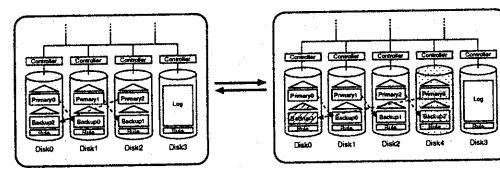

図2: データディスクの追加/取り外し

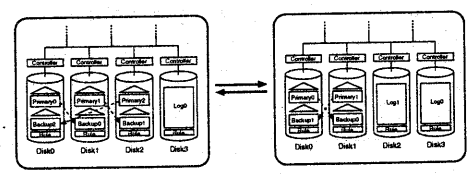

図3: ログディスクの追加/削除

3.3 クラスタのコンフィグレーション情報

前述の通り，自律ディスクのクラスタにはネットワークモニタやマネージャといった特別なノードはない．その代わりに各ディスクがクラスタの構成に関する情報をローカルに保持する．これをコンフィグレーションと呼ぶ．各自律ディスクは共通な

表 1: コンフィグレーション

データディスクの台数:	DataDiskNum
データディスクの ID の配列:	DataDiskID[DataDiskNum]
ログディスクの台数:	LogDiskNum
データディスクとログディスクの対応マップ:	LogMap[ID][LogDiskNum]
バックアップの数:	BackDiskNum
データディスクとバックアップの対応マップ:	BackMap[ID][BackDiskNum]

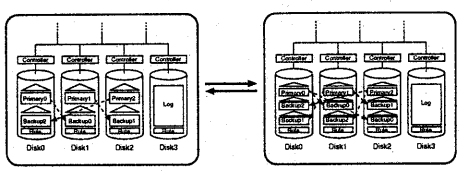

図 4: バックアップの追加/削除

コンフィグレーションを保持する. 表 1 にコンフィグレーション中の主な項目を示す.

コンフィグレーションをローカルに保持する事で, ログの転送やキャッチアップといった通常動作において, 通信を行う事なくエンドポイントを特定できる. しかし, その代わりにクラスタ再構築に伴いコンフィグレーション中のいくつかの項目が更新されると, コンフィグレーションそのものを同期させるための通信を行う必要が生じる. もっとも, クラスタ再構築は通常動作に対して圧倒的に頻度が低いため, この事は問題にはならない.

3.4 クラスタ再構築プロトコル

前述の通り, 自律ディスクのクラスタ内に特別なサーバを置くことは好ましくない. なおかつ, 全ディスクで共通のコンフィグレーションをローカルに保持する必要がある. 我々は上記の用件を満たすようなクラスタ再構築アルゴリズムを提案する. 本アルゴリズムではクラスタ内に動的にコンフィグレーションの同期を保証するコーディネータをたてる. このコーディネータは, 2PC に似た同期フェーズを制御する事で同期を保証する.

また, 通常の分散システムにおいては, クラスタ再構築を行う際にその後の負荷分散まで考慮に入れる必要がある. 自律ディスクにおいてはデータ分散がこれに対応し, 再構築後にはデータの移動を行う必要がある. しかし我々はそのための特別な手段を用いない. その代わりに自律ディスクのファイルシステムとして Fat-Btree[5] を採用する. Fat-Btree は負荷の偏りの検出/除去機能を有する. つまりクラスタ再構築直後の状態を負荷偏りの極端な場合とみなし, 自動的にデータを分散させる. その結果, クラスタ再構築とデータ分散を切り離して考えられる.

ここでは文献 [6] で提案したクラスタ再構築プロトコルの中から, 一台のデータディスクの追加プロトコルを紹介する.

一台のデータディスクの追加 以下にクラスタに一台のデータディスク D_a を追加するプロトコルのアウトラインを示す.

Step 1: D_a はクラスタ内の任意のディスク D_c をコーディネータに任命する.

Step 2: D_c は D_a に許可/不許可を返す. 許可の場合は次のステップへ進む. 不許可の場合は D_a は適当な時間待機してから Step 1 からやり直す.

Step 3: D_c はクラスタ内の全ディスクに対し, コンフィグレーション変更準備メッセージと D_a の ID を送る.

Step 4: クラスタ内の全ディスクは D_c に準備完了/拒否を返す. 全ディスクが準備完了の場合は次のステップへ進む. 一台でも拒否の場合は D_c はクラスタ内の全ディスクに処理の取り消しを発し, D_a に不許可を返し, D_a は適当な時間待機してから Step 1 からやり直す.

Step 5: D_c はクラスタ内の全ディスクに対し, コンフィグレーション変更メッセージを送る.

Step 6: クラスタ内の全ディスクは D_c に成功/失敗を返す. 全ディスクが成功の場合は次のステップへ進む. 一台でも失敗の場合は D_c はクラスタ内の全ディスクに処理の取り消しを発し,

D_a に不許可を返し, D_a は適当な時間待機してから Step 1 からやり直す.

Step 7: D_c は D_a に対し, 新しいコンフィグレーションを送り, クラスタへの参加を許可する.

この場合は, Step 3 が 2PC のプリペアコミットに対応し, Step 5 が 2PC のコミットに対応する. なお, Step 2, 4 で不許可が返ってくる理由としては, 既に他の再構築業務のコーディネータを行っていた等が考えられる.

これ以外に、文献 [6] において複数台の追加、一台/複数台の追加/取り外し、バックアップの追加/削除、ログの追加/削除のプロトコルを提案した。詳細は文献 [6] を提案されたい。

3.5 オンライン再構築

クラスタ再構築をそのままオンラインで行うと排他制御の点で問題が生じる。例えばディスク取り外しプロトコルを実行しながらそのディスクへの`Insert` 命令を実行すると、ディレクトリ中の適切な位置に格納される保証はない。しかし再構築中に常時排他ロックをかけてしまうと、クラスタとしての通常業務が行えなくなる。排他ロックをかける時間は短いほどよい。

ここで再び一台のデータディスクの追加プロトコルを取り上げる。プロトコル中の Step 1, 2 はコーディネータを決定するための通信でありデータの移動を伴わないため、排他制御とは無関係である。また、Step 3, 4 はプリコミットであり、これもデータの移動を伴わない。しかし最後の Step 5 以降の段階ではロックが必要となる。このように、排他ロックが必要なのはデータ移動の伴うフェーズだけであり、他のフェーズではロックをかける必要はない。また、データの移動を伴うフェーズでも、読み出しのみのコマンドは受理できる場合もある。

クラスタ再構築にオンライン性を付加するにはこのような排他制御を行う必要がある。これはオートマトンのようなもので状態遷移を正確に把握することで可能となる。

4 実験用模擬自律ディスク

我々は PC 上に模擬自律ディスクの実装を行っている. 我々は未だプログラム可能なプロセッサの搭載されたディスクを自由に使う事はできない, そのため PC 上の Java で, さらに通常の LAN を用いた模擬環境で実装及び実験を進めている.

図 5 に模擬自律ディスクのブロック図を示す. Communication, Configuration, Data, Directory, Log, Lock, Rule マネージャという 6 つのコンポーネントが現在 Java で記述された部分である. ホスト及び他のディスクからのリクエストは Communication マネージャが受信し, Receive キューを通して Rule マネージャで処理される. また, ログディスクからのリクエスト用に Log キューを設けてある. これはデッドロックを避けるための処置である。ルールとコンフィグレーションをディスク内に保持する. クラスタ再構築完了時には, このコンフィグレーションが書き換えられる. コンフィグレーションは 6 つのコンポーネントで共有する.

我々の実験システムのスペックを表 2 に示す. 我々はこれまで本システムを用いて`Insert` のレスポンスタイム、スループット、バックアップコスト等について実験を行ってきた [10]. 例えば 1`Insert` 命令に対するレスポンスは平均 6 ~ 8ms である。

表 2: 実験システムのスペック

CPU	Intel Celeron708MHz
Chipset	Intel 440BX (ATA33)
Memory	PC100 (CL=2) 256MB
HDD	Seagete ST315323A (5400rpm)
Network	VIA VT6102 (100Mbps)
OS	Linux 2.2.16, glibc-2.1.3
Java	IBM JRE1.3.0 (IBM build cx130-20000623 (JIT enabled: jitc))
Hub	100BaseT Switching

4.1 自律ディスクの追加

以下に n 台からなるクラスタに自律ディスクを追加する場合にかかる時間を示す. 表 3 は無負荷状態での時間での時間である.

表 3: 自律ディスクの追加:無負荷状態での計測時間

台数	2	3	4	5	6	7
時間 (ms)	20.0	25.4	28.3	29.0	27.1	31.3

4.2 オンラインクラスタ再構築の実装

現在の模擬自律ディスクの実装は, Rule マネージャのスレッド数を 2 つに限定している. これは Lock マネージャを簡素化する為の措置で, 自律ディスクのように貧弱なハードウエアを想定したシステムには適している. しかし, 現在の Rule マネー

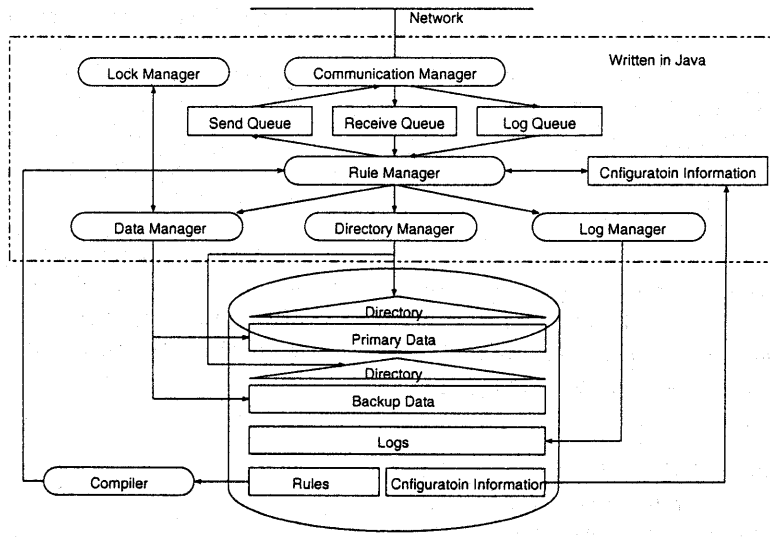

図 5: 模擬自律ディスクの構成

ジャの上でクラスタ再構築を行うとスレッドを使い切ってしまうため, クラスタ全体で通常業務が行えなくなる. よって, 実装を改善し, スレッドを開放する.

提案プロトコルは同期フェーズに時間を要するため, この過程を擬似的にマルチタスク化することでクラスタの通常業務をある程度は実現できる. よって, 協調的マルチタスクの手法に習い, プロトコルを同期フェーズ毎に別々の EIS コマンドとして実装する. 各コマンドが終了したら, 次のコマンドを自らのキューにエンキューする. それに伴ない同期通信の許可/不許可のカウント機能を Rule マネージャ側に移す. さらに Rule マネージャ側で第 3.5 節で述べた排他制御を行う。我々は現在、この方針に基づいて実装を進めている。

5 関連研究との比較

クラスタにおけるノードの追加/削除を自動化するプロトコルは多数提案されている. 例えば DHCP はその中でも特に有名な物としてあげられる. しかし, その殆どは何らかの中央サーバを介したプロトコルであり, 我々は自律ディスクのクラスタに応用するべきではないと判断した. また, 中央サーバを用いる事なく, 耐故障性に重点を置いたもの一つとして, Rodeheffer らの手法 [11] が上げられる. この手法の特徴は自律分散ネットワーク上の各ノードがモニタリング, トポロジー取得, ルーティングを行うことである. 各ノードがアプリケーションを分散環境で実行する類のネットワークにおいては故障したノードの処理を迅速に他のノードに割り振らねばならないため, このようにモニタリングを行う事は有効な手法である. しかし, 自律ディスクにおいてはユーザの戦略に応じたバックアップを保持しているため, 故障ノードの発見を急ぐ必要はない. 逆にモニタリングをソフトウエア的に行うには余分な通信が発生し, ハードウエア的に行ってこれを回避しても余分なコストがかかる. よって, このような手法も自律ディスクには不適切と判断した.

6 おわりに

本稿では先に提案した自律ディスクのクラスタ再構築プロトコルを改善した. このプロトコルはダイナミックコーディネータを介してマルチフェーズを実現するもので, 分散トランザクションにおける 2PC によく似ている. 今回の改善によって, 自律ディスククラスタの活性管理を行えるようになった.

また, PC を用いた実験システム上に再構築プロトコルを一部実装し, 評価実験を行った. 過去に我々が行った実験と比較して, 同期にかかるコストは許容範囲内と評価できる. ここで用いた実験システムはメッセージ通信にキューを用いている. 我々は本稿で提案したプロトコルはキューを用いたシステム全般に応用可能であると考えている.

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 (12680333) 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．

参考文献

[1] Haruo Yokota. Autonomous Disks for Advanced Database Applications. In *Proc. of International Symposium on Database Applications in Non-Traditional Environments (DANTE'99)*, pages 441–448, Nov. 1999.

[2] Dennis R. McCarthy and Umeshwar Dayal. The Architecture of an Active Data Base Management System. In *Proc. of SIGMOD Conf. '89*, pages 215–224, 1989.

[3] J. Widom and S. Ceri (ed.). *Active Database Systems: Triggers and Rules for Advanced Database Processing*. Morgan Kaufmann Pub, 1996.

[4] National Storage Industry Consortium (NSIC). Object based storage devices: A command set proposal. http://www.nsic.org/nasd/1999-nov/final.pdf, Nov 1999.

[5] Hauo Yokota, Yasuhiko Kanemasa, and Jun Miyazaki. Fat-Btree: An Update-Conscious Parallel Directory Structure. In *Proc. of the 15th Int'l Conf. on Data Engineering*, pages 448–457, 1999.

[6] 伊藤 大輔 and 横田 治夫. 自律ディスクにおけるディスク故障時/追加時のクラスタ再構築法. In 第 *12* 回データ工学ワークショップ論文集, *DEWS2001 2B–4*. 電子情報通信学会データ工学研究専門委員会, 2001.

[7] Jim Gray and Andreas Reuter. *Transaction Processing:Concepts and Techniques*. Morgan Kauf-Mann, 1993.

[8] David A Patterson, Garth Gibson, and Randy H. Katz. A case for redundant arrays of inexpensive disks (raid). In *ACM SIGMOD Conference*, pages 109–116, Jun. 1988.

[9] Peter M. Chen, Edward K. Lee, Garth A. Gibson, Randy H. Katz, and David A. Patterson. Raid : High–performance, reliable secondary storage. In *ACM Conmuting Surveys*, volume 26, pages 145–185, Jun. 1994.

[10] 阿部 亮太. ルール処理機能を有する高機能化ディスクの構成に関する研究. Master's thesis, 東京工業大学, 2001.

[11] Thomas Rodeheffer and Michael D. Schroeder. Automatic reconfiguration in autonet. In *13th ACM Symposium on Operating System Principles*, pages 183–187. ACM, 1991.