データベースシステム 125−71
（2001. 7. 19）

自律ディスクへのFat-Btreeの実装

風戸 広史
kazato@de.cs.titech.ac.jp

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

横田 治夫
g yokota@cs.titech.ac.jp

東京工業大学
学術国際情報センター
〒152–8552 東京都目黒区大岡山 2–12–1

あらまし 我々は，ディスク装置中の制御用プロセッサやキャッシュメモリの余剰能力をデータ管理に活用することにより負荷分散，故障対策，障害回復などの高度な機能を実現するアプローチとして自律ディスクを提案している．自律ディスクでは分散ディレクトリを用いることにより，ホストからクラスタ内の任意のディスクに処理要求を出したり，データをクラスタ内に均等に分散させることが可能となる．本稿ではこの自律ディスクの分散ディレクトリ構造としてFat-Btreeを実装する手法について示す．

キーワード 自律ディスク，Fat-Btree, 並列ディレクトリ, 負荷分散

Implementation of A Fat-Btree in An Autonomous-Disk Cluster

Hiroshi Kazato
kazato@de.cs.titech.ac.jp

Tokyo Institute of Technology
2–12–1 Oookayama, Meguro,
Tokyo 152–8552, Japan

Haruo Yokota
yokota@cs.titech.ac.jp

Tokyo Institute of Technology
2–12–1 Oookayama, Meguro,
Tokyo 152–8552, Japan

Abstract We have proposed *autonomous disks* in order to realize the high functionalities such as load balancing, tolerating faults and crash recoveries by utilizing a contorl processor and a large amount of memory within disk drives. Autonomous disks can dispatch requests from hosts to appropriate disk and balance data amount in a cluster by using distributed directory. In this paper, we show a method to implement a *Fat-Btree* as the distributed directory in an autonomous-disk cluster.

key words autonomous disks, Fat-Btree, parallel directory, load balancing

1 はじめに

システム構築の際にストレージシステムをシステムの中心に据えて考える，ストレージセントリックな構成が主流となってきている．このようなシステム構成においては，複数のホストによって共有されるデータを格納する二次記憶装置がシステムの中心となるため，SAN や NAS のようなネットワークに直接接続された二次記憶装置がスケーラビリティの観点から大きな注目を集めている．

また，近年の半導体技術の進歩によってハードディスクなどの二次記憶装置には高性能な制御用コントローラや大容量のキャッシュメモリがすでに搭載されており，これらを利用してディスク上でより高度な処理を行うための提案も行われてきている．たとえば，カリフォルニア大学バークレイ校の IDISK [1] やカーネギーメロン大学の Active Disk [2]，カリフォルニア大学サンタバーバラ校とメリーランド大学共同の ActiveDisk [3] などではディスク上のプロセッサやメモリを利用してアプリケーションを実行することを提案している．

これに対して，我々の提案している自律ディスク [4, 5, 6] ではディスク上のプロセッサを利用してアクセス制御や負荷分散，故障対策，障害回復などのデータ管理を行うことによって，ストレージセントリックなシステム構成における信頼性およびスケーラビリティを向上させることに主眼を置いている．このため，自律ディスクはより広い範囲のアプリケーションへと適用することが可能である．

これまで我々は，ネットワークの利用効率を中心とした自律ディスクのコストの見積もり [5]，PC 上に Java を用いて模擬自律ディスクを実装をした性能解析，さらには非同期バックアップの実装と性能解析 [6] などを行ってきた．これらの検討の中で，自律ディスクの分散ディレクトリ構造として並列 B-Tree の一種である Fat-Btree を用いることでディスク間の同期機構や負荷分散，高速アクセスなどの望ましい機能が提供できることを述べてきた．

しかし，これまでの試作システムでは各ディスク間で値域を水平レンジ分割し，それぞれは通常の B-Tree で管理するディレクトリ構造を用いていた．そこで，本稿では自律ディスクの分散ディレクトリ構造に Fat-Btree を用いるための戦略を検討し，実際に模擬自律ディスクに対して実装を行った結果について報告を行う．

2 自律ディスク

自律ディスクは NAS や SAN などのネットワークへ接続されてクラスタを構成することを前提としたディスク群であり，ホストからはクラスタを単位として一様なアクセスが行われる．クラスタの構成要素となる各ディスクではディスクコントローラ上のプロセッサやキャッシュメモリの余剰能力を利用して自律的なデータ管理を行い，ディスク間の局所的な通信によって，連携してホストからの要求を処理する．

このような自律ディスクの特徴を実現するため，我々はアクティブデータベースにおいて用いられる Event-Condition-Action(ECA) ルールに基づくコマンド階層とストリームインターフェースを用いる．ルール記述はユーザーが自らの戦略にあわせてシステムの振る舞いを柔軟にカスタマイズすることを可能とし，ストリームはホスト–ディスク間およびディスク–ディスク間でのスケーラビリティの高いインターフェースを提供する．

コマンドには外部インターフェースストリーム (EIS)，内部インターフェースストリーム (IS)，内部ディスクアクセス (IDA) の 3 つの階層がある．EIS コマンドはホストに対して**search**，**insert**，**delete** などの基本的なインターフェースを提供する．IS コマンドはルール記述に対するストリームインターフェースを提供し，上述の EIS コマンドは IS コマンドとルールの組み合わせによって記述される．IDA コマンドは実際のディスク入出力を行う SCSI コマンドなどである．

自律ディスクの動作例として，Host_i が自律ディスクにストリームの**Insert** 要求を行った時の処理の流れを図 1 に示す．

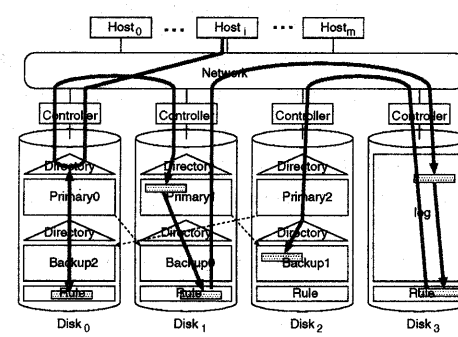

図 1: 自律ディスクに対する Insert 要求

この例では要求を受けた $Disk_0$ で分散ディレクトリを探索したところ，$Disk_1$ にこのストリームを格納すべきであることが判明した．そこで，$Disk_0$ 内のルールに従って $Disk_1$ に**Insert** 要求の転送が行われる．$Disk_1$ でも同様の処理が行われ，ローカルにこのストリームを格納すべきであると判断する．そこで，ディスクへ実際に書き込みを行う前に，Write-Ahead-Log(WAL) プロトコルに従ってログディスク対するログ要求が送信される．ストリームを格納したのちに，$Host_i$ に**Insert** の完了が伝えられる．このように，ホストと自律ディスク間の通信は最初の**Insert** 要求と最後の**Insert** 完了メッセージのみであり，クラスタ内部での処理はホストからは隠蔽される．

ログディスクでは，ルールによってログディスク内のログの量を監視し，ログが一定量を超えると**Catch-up** 要求が発火される．この要求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．

3 Fat-Btree

我々の提案している Fat-Btree [7] は無共有並列計算機上における並列 B-Tree を用いた分散ディレクトリ構造の一種であり，ディレクトリ更新時に必要となる PE 間の同期コストの低減を考慮して設計されている．Fat-Btree では B-Tree の葉ページ (データページ) を各 PE に均等に分配し，格納している葉ページから見て再帰的に上位にあたるインデックスページのみを各 PE に配置する (図 2).

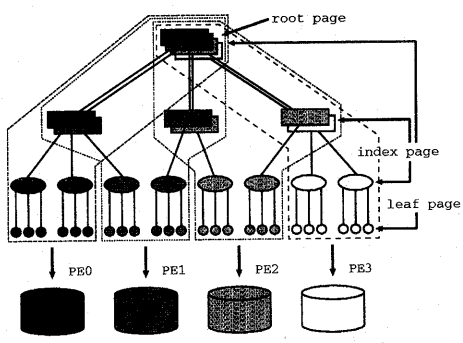

図 2: Fat-Btree の例

更新頻度が高い下位のインデックスページほどそのコピーを持つ PE の数が減少していくため，ディレクトリ更新時に同期が必要となる PE 数が少なくなり，PE 間の局所的な通信によって更新処理を行うことができる．

また，各 PE では格納している葉ページの探索に必要のないインデックスページを持たないため，各 PE でインデックスページのキャッシュを行った場合にヒット率を高く保つことができる．このため更新処理だけでなく，探索処理においても従来の並列 B-Tree 構造と比較して高速に行うことが可能となる．

4 実装における戦略

自律ディスクではホストからの透過性や高いスケーラビリティを実現するため，データ分散，ホストからの均質なアクセス，同時実行制御，偏り制御，異種性などの性質を備えていなければならない．ここでは，分散ディレクトリとして Fat-Btree を採用する場合にこれらの性質を獲得するための戦略を検討する．

4.1 データ分散

自律ディスクへ格納されるデータは分割され，クラスタ内の複数のディスクに分散される．信頼性や性能を向上させるためにデータが複数のディスクに重複して置かれるケースも考えられる．このデータ分散を特定のサーバによって集中的に管理することはスケーラビリティを低下させるため好ましくない．

Fat-Btree ではディレクトリ構造によって自動的にデータが分散されるが，葉ページのコピーを複数の PE に持たせることはできないため，信頼性を向上させるにはプライマリ-バックアップ方式を用いたディレクトリの複製を行う必要がある．

4.2 ホストからの均質なアクセス

自律ディスクはネットワーク上にクラスタを構成しており，ホストからはクラスタに対して透過的にアクセスが可能でなければならない．Fat-Btree ではすべての PE がルートページのコピーを持っており，探索や更新を任意の PE から開始することができる．また，ローカル PE 上に存在しないページに対する処理要求を自動的に該当の PE へ転送さ

せる機能も持っているため上述の性質を満足することができる．

4.3 同時実行制御

自律ディスクはネットワークへ直接接続されており，複数のホストから並列にアクセスすることが可能である．このため，自律ディスクでは何らかの同期機構が必要となる．手軽に実装できる同期機構としてはロックを用いる手法があるが，いずれかのホストが集中してロックを管理する場合は必然的にボトルネックを生じる．

したがって，各ディスクは自身に格納しているリソースに対するロックを分散して管理することが望ましい．Fat-Btree では分散ロックを前提とした同時実行制御プロトコルである INC-OPT 法 [8] が提案されており，ディレクトリ更新時においても高いスループットを得られることが示されており，このプロトコルも自律ディスクへ適用可能である．したがって，本稿では分散ディレクトリに付随して分散ロック機構も提供する方針を取る．

4.4 偏り制御

自律ディスクはディスクコントローラ上で動作させることを前提としているため，ディスクアクセスやネットワーク負荷をモニタリングする専用のハードウェアなどは利用できない．したがって，我々はソフトウェアのみで負荷分散を実現するアプローチを取る．Fat-Btree における偏り制御のアルゴリズムは [7] で示されているが，実際には負荷分散の戦略はユーザーの要求に応じて変更できなければならない．

これを可能にするためには，Fat-Btree の負荷分散機能に自律ディスクのルールを組み合わせて使用すればよい．このようなルール記述の例は 6.2 節で示す．

4.5 異種性

上記に述べられているような性質を実現し，異なるベンダーによって提供された自律ディスクを接続してもクラスタが構成可能となるためには，共通のインターフェースを定義しておく必要がある．

Fat-Btree を分散ディレクトリ構造として用いる場合，ディスク–ディスク間のインターフェースは Fat-Btree のセマンティクスを満たしていなければならず，他の分散ディレクトリ構造を採用した自律ディスクはクラスタに組み込むことはできない．

5 試作システム

本研究では，これまで述べてきた戦略に従って PC 上に Java を用いて模擬自律ディスクの実装を行っている．以下にその概要を示す．

5.1 模擬自律ディスクの構成

図 3 は模擬自律ディスクの大まかなモジュール構成を表している．点線で囲まれた部分が Java によって実装されている部分である．

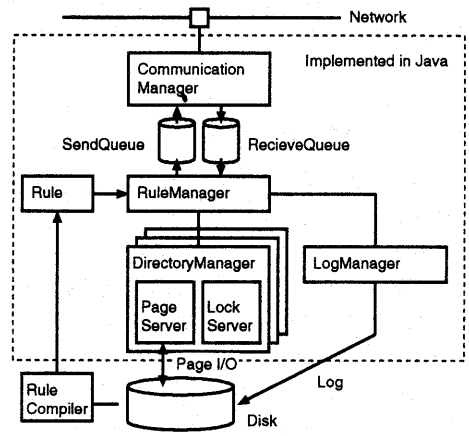

図 3: 模擬自律ディスクの構成

この試作システムでは，1〜8 台の Linux PC を 100Base-TX の LAN 上に接続して模擬自律ディスクのクラスタを構成している．これらの PC はすべて同じ性能を示すように設定されており，その諸元を表 1 に示す．

5.2 通信サーバ

通信サーバはディスク–ホスト間，およびディスク–ディスク間のメッセージに基づく非同期通信を行うモジュールである．自律ディスクの EIS コマンドやログメッセージ，Fat-Btree の同期メッセー

表 1: 実験用 PC の諸元

項目	パラメータ
アーキテクチャ	IBM PC/AT 互換機
CPU	Intel Celeron 708MHz
メモリ	256MB SDRAM (CL = 2)
ハードディスク	Seagate U10 15323A 5,400 rpm, 15.3GB
OS	Linux 2.2.16
Java 実行環境	IBM JRE 1.3.0

ジなどはすべてストリームとして表現される．送信用と受信用のメッセージキューを使用する．

通信サーバは起動されると自律ディスクのネットワークインターフェース上の特定のポートを監視し，他のディスクやホストからの接続要求を待つ．いったん接続要求を受け付けると，新しいスレッドを作成してコネクション処理を引き渡し，新しい接続要求を待つ．

通信サーバは，以下の IS コマンドを提供している．**transfer** コマンドはメッセージを他のディスクへ転送するためのものであり，メッセージ送信元のサイト識別子を変更しないため応答メッセージの送信先を適切に保つことができる．

- **send(Message, SiteID)**
  メッセージ**Message** をサイト識別子**SiteID** で表されるディスクまたはホストに送信する．このとき，メッセージには送信元のサイト識別子**Origin** を付加する．

- **transfer(Message, SiteID)**
  メッセージ**Message** をサイト識別子**SiteID** で表されるディスクまたはホストに転送する．送信元のサイト識別子は変更しない．

5.3 ルールマネージャ

ルールマネージャでは ECA ルールに基づいて自律ディスク内のイベントの監視を行う．発生したイベントに応じて条件式をチェックし，真となれば定義されているアクションをトリガする．

ルール記述は各ディスク内に格納されており，ルール変換部によって自律ディスク内に読み込まれる．このルール変換部は現在は実装されていないため，ルールを前もってコンパイルしておく必要がある．

5.4 ディレクトリマネージャ

ディレクトリマネージャでは分散ディレクトリとして用いる Fat-Btree に対するローカル操作を行う．Fat-Btree のノードへアクセスするためにページサーバを使用し，ページの取得に先立ってロックサーバからロックを取得する．ローカルでは処理できない操作が発生すると，通信サーバを経由して他の自律ディスクへ処理を依頼する．このモジュールでは，以下の IS コマンドを提供する．

- **traverse_directory(StreamID)**
  分散ディレクトリ上で探索を行い，ストリーム識別子**StreamID** によって表されるストリームが格納されるべき位置を返す．

- **insert_local(StreamID, Stream)**
  ストリーム識別子**StreamID** によって表されるストリーム**Stream** をローカルディスク上へ挿入する．

- **delete_local(StreamID)**
  ストリーム識別子**StreamID** によって表されるストリームをローカルディスク上から削除する．

5.5 ページサーバ

ページサーバは内部ディスクアクセス (IDA) コマンドを用いて，自律ディスクのデータの入出力を行う．このデータの入出力は固定長のページ単位で行なわれる．ページサイズはフォーマット時に任意の値を指定できるが，通常は 4KB に固定している．

インデックスページおよび葉ページは Java のオブジェクトとして表現されるため，ページサーバでは Java のオブジェクト直列化 (serialization) 機構によってこれらのオブジェクトをディスク上に永続化し，要求に応じて復元する．すべてのページをメモリ上に保持しておくことは不可能であるから，復元したページの一部をページサーバ内にキャッシュしておくことによって高速化を図る．キャッシュの管理アルゴリズムには LRU を用いた．

5.6 ロックサーバ

模擬自律ディスクの同時実行制御には階層ロックを用いており，ページ単位でのロックを行っている．

ロックの適合性は図 4 のとおりである．ページに対する IS・IX ロックはローカルコピーでのみ取得すればよいが，S・SIX・X ロックはすべての PE 上で取得しなければならない．

mode	IS	IX	S	SIX	X
IS	○	○	○	○	×
IX	○	○	×	×	×
S	○	×	○	×	×
SIX	○	×	×	×	×
X	×	×	×	×	×

図 4: ロックマトリクス

ロックサーバはロック要求を受けるとページに対するロックの取得を試みるが，現在のページのロックモードと要求されたロックが適合しなければ指定されたタイムアウト時間までブロックして待つ．このモジュールでは，以下の IS コマンドを提供する．

- `lock(StreamID, Mode)`
  ストリーム識別子`StreamID` をもつストリームに対してロックモード`Mode` の取得を試みる．このコマンドの呼び出しはロックの取得に成功するか，タイムアウトとなるまでブロックされる．
- `convert(StreamID, Mode)`
  ストリーム識別子`StreamID` をもつストリームに対してすでに取得しているロックをアトミックにロックモード`Mode` へ変換する．このコマンドの呼び出しはロックの変換に成功するか，タイムアウトとなるまでブロックされる．
- `unlock(StreamID)`
  ストリーム識別子`StreamID` をもつストリームに対して取得しているすべてのロックを開放する．

5.7 ログマネージャ

ログマネージャでは自律ディスクに対して発行されたコマンドをログとして記録することによって障害回復を容易にするとともに，ホストからの要求とは非同期にバックアップを行うことを可能としている．このモジュールでは，以下の IS コマンドを提供する．

- `put_log(Disk, Command)`
  ディスク識別子`Disk` をもつディスクに対して発行されたコマンド`Command` をログに記録する．
- `catch_up(Disk, Backup, Amount)`
  ディスク識別子`Disk` をもつディスクに対して発行されたコマンドをログから`Amount` の数だけ取り出し，ディスク識別子`Backup` をもつバックアップディスクに対して等価な操作を行う．

6 Fat-Btree へのルール適用

自律ディスクに Fat-Btree を組み込むことによって，Fat-Btree の動作に対してルールを適用することが可能になる．すると，Fat-Btree 単体では実現しにくい複雑な機能をルールによって柔軟に記述できるようになる．以下ではこのメリットの例として，バックアップと負荷分散をルールによって実現する方法について述べる．

6.1 バックアップ

Fat-Btree はローカルでの探索に不要なインデックスページを省くことによって更新時の性能向上を図っている．このため，ディレクトリ構造に冗長性をもたせて信頼性を向上することは望めない．そこで我々は，自律ディスク内にバックアップ用の独立したディレクトリ構造をもつことで信頼性を向上させるアプローチをとり，プライマリとバックアップの間の整合性を取るための仕組みとして自律ディスクのルールを用いる．

6.1.1 バックアップ方式とログ方式

バックアップ戦略には，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がある．物理バックアップ方式ではページレベルの変更に対してプライマリとバックアップの同一性を保証するのに対して，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対する論理操作の同一性のみを保証する．

バックアップ方式とログ方式は密接に関連しており，物理バックアップ方式ではページレベルの変更に対するログを保持する必要があるが，論理バックアップ方式では自律ディスクに対する論理操作のコマンドログを保持していればよい．

6.1.2 ルール記述例

ここでは，より自由度の高い論理バックアップ方式を採用し，この方式を用いた場合のルール記述の例を Rule_1 に示す．この例はタイマーと閾値を組み合わせて使用するもので，一定期間ごとにログの量がチェックされ，バックアップに送られていないログの数がある閾値をこえている場合はCatch-Up要求が発火される．

```
Rule_1:
When timer(Interval);
if   count(Log, D) > Threshold;
then B = mappping(D, backup),
     catch_up(D, B, Amount);
```

6.2 負荷分散

Fat-Btree における負荷分散は，負荷集計と負荷移動の二つのフェーズに分けて考えることができるので，以下ではこれらのフェーズについて順番に検討する．

6.2.1 負荷集計フェーズ

負荷集計フェーズでは，各ディスクがローカルに格納している葉ページごとの負荷点数の集計を行う．insert および retrieve コマンドによる葉ページへのアクセスを ECA ルールのトリガに用いれば，この負荷集計は Rule_2, Rule_3 のようなルールを自律ディスクへ追加することによって実現可能である．$W_{\mathrm{read}}, W_{\mathrm{write}}$ はそれぞれ葉ページの retrieve,insert コマンドに対する負荷点数であり，ユーザーがルール記述によって変更することが可能である．

```
Rule_2:
When retrieve(StreamID);
  Load.StreamID = Load.StreamID + W_read
```

```
Rule_3:
When insert(StreamID, Stream)
  Load.StreamID = Load.StreamID + W_write
```

集計した負荷情報を交換するために，クラスタ内に負荷情報を格納したメッセージを循環させる．このメッセージを受け取ったディスクは自分自身の負荷情報をメッセージに書き込んでから，論理的に隣接するディスクへ転送する．

6.2.2 負荷移動フェーズ

前項の負荷情報メッセージをクラスタ内で循環させることによって，各ディスクはクラスタ全体の負荷平均を算出し，自分自身の負荷点数を比較することが可能となる．平均負荷と自分自身の負荷点数の差がある閾値以上である場合に，Fat-Btree によって値域分割された境界にある葉ページを論理近傍となるディスクへと移動させる．

負荷移動フェーズのトリガはルールで変更可能であり，負荷情報メッセージを受信したときやタイマーで一定時間おきにイベントを発生させるなどのトリガを利用してホストからの操作とは非同期に行うことが可能である．負荷情報メッセージの受信時をトリガとする場合のルールの例を以下に示す．

```
Rule_4:
  When token(Load, Data);
  if   OwnLoad() - Load.average > Load_T;
  then N = mapping(D, neighbor);
       migrate_node(D, N);
```

7 おわりに

本稿では，自律ディスクの分散ディレクトリ構造として Fat-Btree を実装した場合に必要となる戦略について検討し，Java による試作システムへ実装を行った．自律ディスクの備えるべき性質のうち，ホストからの一様なアクセス，偏り制御，同時実行制御は Fat-Btree の基本機能として提供することができる．

逆に，自律ディスクのルール記述を Fat-Btree へ適用することによって Fat-Btree の動作に対して柔

軟性を持たせることができることも述べた．このように，自律ディスクと Fat-Btree を組み合わせた相乗効果によるメリットは大きく，これらの間の相性は非常によいと言える．

我々はすでに PC 上に Java を用いた模擬自律ディスクを試作しており，現在この試作システムのディレクトリ構造に Fat-Btree を実装しているところである．残念ながら本稿では性能解析を行うまでに至らなかったが，今後さらに実装を進めていき，この試作システムを用いて従来との性能比較などを行う予定である．

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] Kimberly Keeton, David A. Patterson, and Joseph M. Hellerstein. A Case for Intelligent Disks (IDISKs). *SIGMOD Record*, 27(3):42–52, Sep. 1998.

[2] Erik Riedel, Garth Gibson, and Christos Faloutsos. Active Storage for Large-Scale Data Mining and Multimedia Application. In *Proc. of the 24th VLDB Conf.*, pages 62–73, 1998.

[3] Anurag Acharya, Mustafa Uysal, and Joel Saltz. Active disks: Programming model, algorithms and evaluation. In *8th International Conference on Architectural Support for Programming Languages and Operating System*, pages 81–91, October 1998.

[4] 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.

[5] 阿部 亮太 and 横田 治夫. 自律ディスクと従来のディスクのコスト比較. In 第 *11* 回データ工学ワークショップ論文集, *DEWS2000 3B-1*. 電子情報通信学会データ工学研究専門委員会, 2000.

[6] 阿部 亮太 and 横田 治夫. 自律ディスクにおける非同期バックアップのルールによるタイミング調整. In 信学技法 *Technical Report of IEICE FTS2000-33*, pages 41–48. 電子情報通信学会, October 2000.

[7] 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.

[8] J. Miyazaki and H.Yokota. An Incremental Optimistic Concurrency Control for Parallel B-tree Structures. *Tokyo Institute of Technoglogy Technical Report*, (TR00-0011), Jul. 2000.