論文 / 著書情報
Article / Book Information

論題(和文)	COBALT:バージョン管理を行う並列分散ストレージシステムにおけるアクセス負荷と記憶空間利用率の同時均衡化手法
Title(English)	COBALT: A Method for Balancing Both Access Load and Utilization Ratio in a Parallel Distributed Storage System with Version Management
著者(和文)	中野真那, 小林大, 渡邊明嗣, 横田 治夫
Authors(English)	Mana Nakano, Dai KOBAYASHI, Akitsugu WATANABE, Haruo Yokota
出典(和文)	電子情報通信学会和文論文誌（D）, Vol. J90-D, No. 2, pp. 349-358
Citation(English)	IEICE Transaction on Information and System(D), Vol. J90-D, No. 2, pp. 349-358
発行日 / Pub. date	2007, 2
URL	http://search.ieice.org/
権利情報 / Copyright	本著作物の著作権は電子情報通信学会に帰属します。 Copyright (c) 2007 Institute of Electronics, Information and Communication Engineers.

COBALT：バージョン管理を行う並列分散ストレージシステムにおけるアクセス負荷と記憶空間利用率の同時均衡化手法

中野　真那[†]　小林　大[†,††]　渡邊　明嗣[†]　横田　治夫[†,†††]

COBALT: A Method for Balancing Both Access Load and Utilization Ratio in a Parallel Distributed Storage System with Version Management

Mana NAKANO[†], Dai KOBAYASHI[†,††], Akitsugu WATANABE[†], and Haruo YOKOTA[†,†††]

あらまし　並列分散ストレージシステムを効率的に利用するためには，各ストレージ装置のアクセス負荷と記憶空間利用率が均衡化されることが望ましい．本論文では，並列分散ストレージ上でバージョン管理を行う場合に，新旧バージョンのアクセス頻度の差を利用することで，上記の均衡化を両立させるCOBALTを提案し，その評価結果を報告する．COBALTでは，アクセス頻度の高い最新バージョンのファイル移動でアクセス負荷の均衡化を行い，相対的にアクセス頻度の低い旧バージョンのファイル移動で記憶空間利用率の均衡化を行う．その際，最新バージョンの管理のためにB^+-treeを用い，古い差分情報の管理に連結リストを用いることで，アクセス頻度の高いデータに対する高速アクセスと，記憶空間利用率均衡化のための柔軟な配置を可能とする．PCクラスタ上の試作システムを用いて，アクセス負荷と記憶空間利用率の偏り除去効果に関する評価を行う．

キーワード　負荷分散，インデクス，ストレージ技術，並列分散DB，バージョン管理

1. まえがき

時間を追って作成・更新されるファイルについて，要求に従ったバージョンの状態を提供可能にするためのデータ・メタデータ管理をバージョン管理と呼ぶ．バージョン管理の機能をストレージシステム内部で実現することは，データバックアップの容量効率の向上や，システムソフトウェアや地図情報といった頻繁に更新されるデータの差分配布の観点で有用である．

これまでに，長大するバックアップ時間と一貫したバックアップ作成を両立するシャドーイング[1]を実装したストレージシステムが実現されている．シャドーイングでは半日や1日といった単位で差分情報が保持される．より細粒度の，単一更新ごとの差分を保持するものとして，SNIAによりCDP (Continuous Data Protection)の概念が提案されている[2]．CDPは，安全性を高めるために更新のたびにバックアップしてデータを保護する機能である．頻発するデータ更新をすべて記録するため，格納対象データは差分データを含み，爆発的に増加せざるを得なくなる．

増加し続ける大量の差分データを含む大量のデータを，安定して格納し高性能で供給するシステムには，複数のストレージ装置をネットワーク結合し構成された並列・分散ストレージシステムを用いるのが現実的である．並列ストレージシステムは，システム中のデータ配置特定のためのグローバル索引構造を有する．索引操作のスケーラビリティのため索引構造は複数の装置に分散可能なことが求められる．B^+-treeをもとにした並列索引構造は，値域分割による並列分散や，高性能な並行性制御に関する研究が進んでおり[3]～[5]，並列B^+-treeを用いたストレージシステムや分散データベースが様々提案されている[6],[7]．そこで本研究では，並列B^+-treeを用いたグローバル索引構造を拡張し，並列ストレージシステム上で差分データに

† 東京工業大学大学院情報理工学研究科計算工学専攻，東京都
Department of Computer Science, Graduate School of Information Science and Engineering, Tokyo Institute of Technology, Tokyo, 152–8552 Japan
†† 日本学術振興会特別研究員 DC，東京都
Research fellow, Japan Society for the Promotion of Science, Tokyo, 102–8472 Japan
††† 東京工業大学学術国際情報センター，東京都
Global Scientific Information & Computing Center, Tokyo Institute of Technology, Tokyo, 152–8552 Japan

よるバージョン管理を実現することを考える．

並列・分散ストレージにおいては，一部のストレージ装置にリクエストが集中するとレスポンスタイムが極端に遅くなる [6]．データマイグレーションによるデータ再配置によりアクセス負荷の多いデータ群をストレージ装置間で均等に配置することで，アクセス負荷均衡化を達成可能である．しかし，バージョン管理に必要なすべての差分オブジェクトを単一の並列 B^+-tree 構造により配置管理する場合，値域分割が前提となり，各データは一意に定められて key 値に従い格納ストレージ装置が決定される．バージョン管理下にあるデータは最新データと差分データがあるが，差分データは通常時にはほとんどアクセスされず，また相対的に多くアクセスされる最新データは一意の key 値に従って B^+-tree 値域内に分散格納されるため，アクセス負荷均衡化時に最新データを均等に配置するため値域において最新データの間に含まれる大量の差分データが付随して配置され，各ストレージノード間で格納データ量を制御できない弊害がある．

一方で，ストレージノード間の格納データ量を制御することは様々なメリットが存在する．例えば空き領域を集中することで空きストレージ装置をつくり電源を切ることでシステム消費電力の削減が実現できる．また，特に値域分割を前提としたデータ配置では格納対象データは値域によってのみ格納されるストレージ装置が決定されるため，すべてのストレージ装置がデータ挿入に備え空き領域を有することが望ましい．並列 B^+-tree へバージョン管理を導入するに際し，アクセス負荷と記憶空間利用率の柔軟な制御の両立を実現する必要がある．本論文では特に記憶空間利用率の柔軟な制御の結果として利用率の均衡化を目標とする．

バージョン管理を取り入れた B^+-tree 構造に関する研究として，MultiVersioning (MV) B^+-tree が行われてきた [8]～[10]．MV B^+-tree は主にデータ追加・削除により更新される B^+-tree 構造に対するバージョン管理を対象としており，実際のデータ構造は各節点にバージョン管理情報を加えた一次元 B^+-tree である．このような MV B^+-tree を並列 B^+-tree 化することは可能であるが，その場合従来用いられている値域分割を適用することになり，偏り制御の両立に関して上記の単一 B^+-tree と同様の問題がある．

本論文では，バージョン管理機能をもつ大規模並列ストレージシステムの実現を目的とし，バージョン管理下にあるデータ群のアクセス傾向の特性を利用した，アクセス負荷の均衡化とデータ記憶空間利用率の均衡化を両立するためのデータ配置管理手法 COBALT (Combination Of B^+-tree-node And Linked List Transfer) を提案する．我々の先行研究 [11] におけるシミュレーションによる知見から，COBALT はバージョン管理と柔軟なデータ配置をアクセス負荷とデータ格納量の両均衡化を両立しつつ実現するための索引構造であり，並列 B^+-tree と連結リストを組み合わせ柔軟なデータ配置を可能とし，バージョン管理下データのアクセス傾向性質を利用したデータ配置を行うことで，各偏り制御の両立を実現する．COBALT では最新バージョンのファイルを並列 B^+-tree で，差分データをストレージ装置をまたがる連結リストにより管理する．また，COBALT で用いるアルゴリズムでは，アクセス頻度の高い最新バージョンのデータのデータ配置制御によりアクセス負荷の均衡化を行い，アクセス頻度の低い旧バージョンの差分データファイルを記憶空間利用率の均衡化に利用することで目的を達成する．

COBALT を利用したデータ配置管理の評価を行うために，PC クラスタ上に COBALT を実装した並列ストレージシステムを試作した．試作システムを用いた実験により，提案手法によるアクセス負荷と記憶空間利用率の両立が，実現できたことが示された．

以下に本論文の構成を示す．次章で前提とするバージョン管理とその性質について述べ，**3.** で前提とする並列ストレージシステムと負荷均衡化処理に関して紹介する．**4.** で提案手法である COBALT について説明する．次に **5.** で，評価のための試作システムについて述べ，**6.** でその試作システムを用いた実験の結果とその考察を行う．最後に **7.** でまとめと今後の課題について述べる．

2. 前提とするバージョン管理とその性質

バージョン管理とは，時間を追って作成されるファイルについて要求に従ったバージョンのファイルを提供可能にするためのデータ及びメタデータの管理を指す．

2.1 バージョン管理方法

本論文では，Reverse-delta [12] 方式によるバージョン管理方法を想定する．Reverse-delta 方式では，最新のバージョンのファイルを基準として保持し，ファイルを更新するたびに更新された最新のバージョンと 1 世代前のバージョンとの差分を残す．最新バージョンに対し過去の差分を順次適用することで任意の古

いバージョンのファイルを取得できる．一般に，最新バージョンへのアクセスの方が，古いバージョンへのアクセスに比べて多いため，最新バージョンを最も効率良く取得可能な Reverse-delta は多くのシステムで利用される [12]～[17]．

本論文の議論で用いる用語を以下に定義する．バージョン管理の対象となるファイルをバージョンファイル（あるいはファイル）と呼び，バージョンファイルのある時点の状態のことをバージョンと呼ぶ．あるバージョンファイルの最新バージョンの内容が書き込まれたデータを最新オブジェクト，連続する二つのバージョン間の差分情報データを差分オブジェクトと呼ぶ．最新オブジェクトと差分オブジェクトを記憶する領域をレポジトリと呼ぶ．更に，あるバージョンファイルの差分オブジェクトのうち，あるバージョン v よりも新しいバージョンすべてに対応する差分オブジェクトの集合を Newer(v) で表し，それより古いバージョンすべてに対応する差分オブジェクトの集合を Older(v) で表す．

2.2 バージョン管理動作

想定するバージョン管理では，レポジトリに保持されたバージョンファイルに対する更新機能とバージョンを指定したファイル読出し機能をもつ．

ファイル更新ではファイルの一部分若しくは全体が書き換えられるものとする．システムは最新オブジェクトの内容を更新すると同時に変更差分を記述した差分オブジェクトを生成し，それぞれをレポジトリに追加する．

あるバージョンファイルのバージョン v の読出しでは，ファイルの最新オブジェクトと Newer(v) のすべてが読み出され，Newer(v) をもとに最新オブジェクトの内容を該当バージョンの内容に書き換えた後送信する．

ソフトウェアの安定バージョンのように，最新ではないが読出し要求が多いバージョンが存在する場合，当該バージョンを得るための差分適用が頻繁に発生し性能劣化につながる．本論文ではこのような場合には特定バージョンに対し，差分情報を介さずに直接アクセスできるオブジェクトであるスナップショット(注1)を作成することで頻繁な差分適用を回避する．スナップショットオブジェクトに対して更に差分を適用することによって更に古いバージョンを取り出すことは可能とする．

2.3 バージョン管理に用いるオブジェクト間のアクセス傾向の性質

前提とする Reverse-delta 方式によるバージョン管理におけるオブジェクト間のアクセス傾向を考察する．

2.2 で述べたように，前提とする Reverse-delta では，あるバージョン v のファイルを取得するため，最新オブジェクトと v までの差分オブジェクトすべてにアクセスする必要がある．このことから，バージョンファイルの第 v バージョン，すなわちファイルが生成されてから v 個目に作成された差分オブジェクトへの読出し要求回数量は，バージョン 1（ファイル生成時）からバージョン v までの各バージョンへの読出し要求回数量の積分値となる．よって，差分オブジェクトのアクセス頻度は古くなるほど最新オブジェクトに比較して極端に小さくなる（図 1）．

また，**2.2** で述べたように，本論文では，最新でないある特定のバージョン v に対する読出し要求が頻発する場合には，差分適用を行わずに直接バージョン v を取得可能なオブジェクト（スナップショット）を作成し，差分適用回数を削減するものとしている．アクセス傾向の観点では，最新オブジェクトと差分オブジェクトの間と同様の関係が，スナップショットオブジェクトと当該バージョン以前の差分オブジェクトとの間にも成立することが分かる（図 2）．

（性質 1） バージョンファイルのアクセス時には，その最新オブジェクトは毎回アクセスされる．

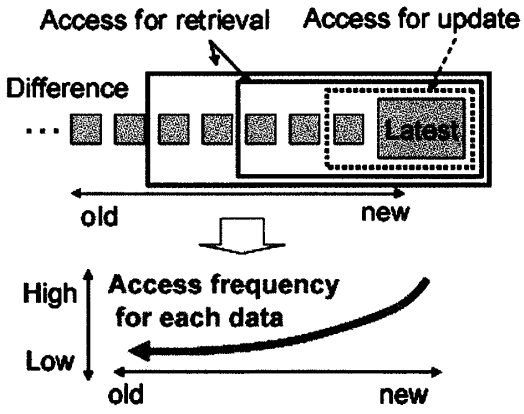

図 1 最新オブジェクトと差分オブジェクトのアクセス傾向

Fig. 1 Access tendency of the latest and delta objects.

(注1)：ここでは，バックアップ生成等のために用いる「システム全体のある時点での一貫したファイル状態を取得可能なデータ構造（シャドーイング [6]）」とは異なる意味で用いていることに注意されたい．

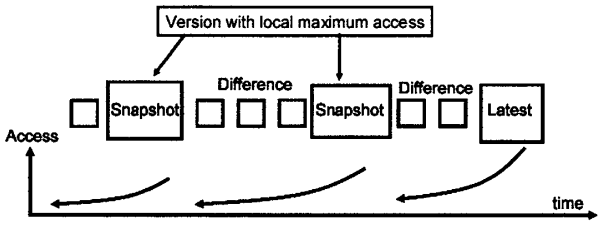

図 2　スナップショットを前提としたアクセス傾向
Fig. 2　Access tendency using snapshot objects.

(性質 2)　差分オブジェクトのアクセス負荷は最新オブジェクトやスナップショットオブジェクトのアクセス負荷と比較してバージョンが古くなるほど急激に減少する.
(性質 3)　差分オブジェクトのアクセス負荷は時間とともに低下する.

3. 並列ストレージシステムと負荷均衡化

本論文では，複数の高機能なストレージ装置（ノード）をネットワークで接続した並列ストレージクラスタ上にレポジトリを配置することとする．例えば，我々が提案する高機能ストレージである自律ディスク [18] や並列 NAS (Network Attached Storage) 等を用い，ストレージノード上に存在する計算資源を用い索引構造，各種偏り制御及びバージョン管理を実現することを想定する.

ストレージノードは自身の現在あるいは将来のアクセス負荷を定量的に表すアクセス負荷値の定義を保持しており，格納データに対するアクセス負荷値を計算可能である．アクセス負荷値の定義は様々なものが存在し，用いる負荷分散機構や対象となるアクセスパターンを考慮し事前に決定をする.

ストレージノード同士は互いに通信することで，それぞれのアクセス負荷値や記憶空間利用率情報をやり取りできる．また，データ提供サービスを継続したままストレージ装置間でデータ移動可能な機能を有するとする．データ移動によって動的に偏り除去を行うことが可能である.

なお本論文の以降の議論では，システムを構成する各ストレージノードのデータ格納可能量はすべて等しいことを仮定する．異なる容量をもつストレージノードが混在するシステムへの適用に関しては，今後の課題とする.

3.1　負荷均衡化と記憶空間利用率均衡化

並列分散ストレージシステムにおいて拡張性を維持しながらアクセス負荷を均衡化させる手法として，並列 B^+-tree を用いたグローバル索引構造と値域分割によるデータ配置が提案されている [19], [20]．各オブジェクトのアクセス負荷値をもとに，すべてのオブジェクトを種類によらず，各ストレージノードごとのアクセス負荷値が均等化するよう，ストレージ装置内で識別に用いる一意な値をもとに値域分割で各ストレージノードに配分する.

単純にすべてのオブジェクトを一律に B^+-tree の各葉ノードに格納し，上記の値域分割によりアクセス負荷均衡化を行うことが可能である．以下，この手法を全 B^+-tree 管理手法と呼ぶ.

バージョン管理に用いるオブジェクトは **2.3** で述べたアクセス傾向性質をもつため，一次元的なデータ配置が前提となる全 B^+-tree 管理手法では，アクセス負荷を均衡化することで記憶空間利用率の均衡化が崩れてしまう．次章の提案ではこれらを同時に均衡化させることを考える.

4. COBALT を用いたアクセス負荷均衡化とデータ量均衡化の両立

以上の前提とするバージョン管理の性質に基づき，格納ファイルに対するバージョン管理を実現する並列ストレージシステム上で，アクセス負荷と記憶空間利用率の均衡化を両立させる手法を提案する.

4.1　COBALT のアクセス構造

我々は，最新オブジェクトと差分オブジェクト間のアクセス頻度の差に着目し，並列 B^+-tree と連結リストを組み合わせた COBALT (Combination Of Btree-node And linked List Transfer) を提案する.

COBALT では，最新オブジェクトは全 B^+-tree 管理手法と同様に並列 B^+-tree を用いて配置管理し，差分オブジェクトはバージョンファイルごとに新しい順にリンクノードで管理し，B^+-tree の葉ノードからリンクノードをたどることでその位置を管理する．連結リストを用いるため，差分オブジェクトは値域によらずシステム中の任意のストレージ装置に配置可能である．図 3 に COBALT のアクセス構造の概念を示す.

COBALT のアクセス構造は，B^+-tree が最新オブジェクトのみを保持するため，差分オブジェクトも同様に管理する全 B^+-tree 管理手法に比べて木のサイズが小さくなる．このため，木を構成する節点数が少なく，各ストレージノードが備えるキャッシュメモリのキャッシュヒット率がより高く，結果として高速アク

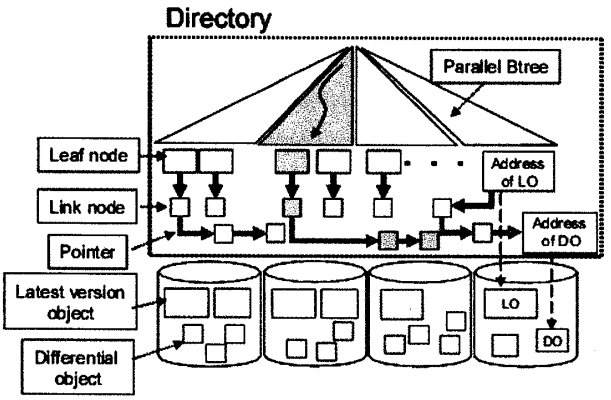

図 3　COBALT のアクセス構造の概念
Fig. 3　Conceptual access structure of COBALT.

セスすることが可能となる長所をもつ．

4.2　COBALT によるデータ配置管理下における偏り制御

本節では，差分オブジェクトの柔軟な配置と，**2.3** で示したバージョン管理に用いるオブジェクトのアクセス傾向を利用し，アクセス負荷均衡化と記憶空間利用率均衡化の両立を実現するアルゴリズムを提案する．

4.2.1　配置方針

アルゴリズムを構成する方針として (i) 高アクセス負荷オブジェクトの移動によりアクセス負荷均衡化を，(ii) 低アクセス負荷オブジェクトの移動により記憶空間利用率均衡化を達成する．また移動オブジェクト選択の際，(iii) 頻繁に同時アクセスされるオブジェクト群は同一ストレージ内に格納することを考慮する．

アクセス負荷を均衡化させるために，アクセス負荷に影響を与える最新版オブジェクト及びいくつかの新しい差分オブジェクトを移動させる．最新版オブジェクトは値域分割により管理されているため，これらのオブジェクトの移動先は論理的に隣接したストレージ装置となる．

一方，記憶空間利用率を均衡化させるために，性質 2 により，いくつかの古い版の差分オブジェクトを動かすことでアクセス負荷の状態に大きな影響を与えずに記憶空間利用率の偏り減少を試みる．オブジェクトの移動先には記憶空間利用率の少ないストレージ装置を選択する．前述した COBALT のアクセス構造では，差分オブジェクトが連結リストで管理され，柔軟な配置が可能であるためにこのような配置戦略が可能となる．

同時に読出し要求をされる差分オブジェクトはストレージノード間差分オブジェクト送信によるコストを最小に抑えるため，できるだけ同じストレージ装置に配置することが望ましい．アクセス負荷均衡化に用いるオブジェクト数，記憶空間利用率均衡化に用いるオブジェクト数を，上記の要求を満たしつつ決定するため，次項に述べる移動境界の概念を導入する．

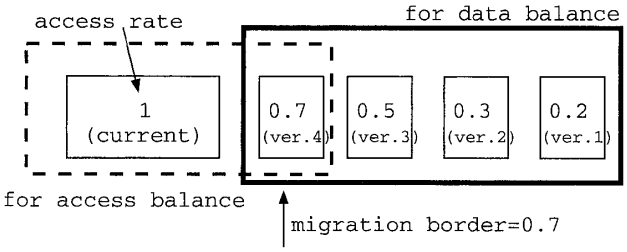

図 4　移動境界率 β を 0.7 とした場合の差分オブジェクト
Fig. 4　Migration border: Threshould β is set to 0.7.

4.2.2　移動境界と移動境界率

もとのストレージ装置に残す差分オブジェクトと，移動する差分オブジェクトの境界を移動境界と定義する．移動境界はバージョンファイルのいずれかのオブジェクトであり，移動境界にあたるバージョン v_b に対して Newer(v_b) と Older(v_b) を，それぞれアクセス負荷の均衡化と記憶空間利用率の均衡化のためのオブジェクトの移動の対象とする．

移動境界は，最新オブジェクトのアクセス負荷値に対する各オブジェクトのアクセス負荷値の割合の移動境界率 β で指定する．つまり，最新オブジェクトのアクセス負荷値を A としたとき，$A \times \beta$ に最も近いアクセス負荷値をもつ差分オブジェクトを移動境界とする．

図 4 は移動境界率 β を 0.7 とした場合の移動境界の概念を示している．図の例では ver.4 で示される差分オブジェクトのアクセス負荷値（access rate）が 0.7 となっており移動境界上に位置する．この例では，current，ver.4 の二つのオブジェクトが負荷均衡化のための移動対象オブジェクト，ver.4〜1 の四つのオブジェクトが記憶空間利用率均衡化のための移動対象オブジェクトとなる．ver.4 についてはどちらの戦略にも利用される．

4.2.3　オブジェクト配置アルゴリズム

以下に，アクセス負荷値の差を均衡化するオブジェクトの配置アルゴリズム（アクセス負荷均衡化アルゴリズム）と記憶空間利用率の差を均衡化するオブジェクトの配置アルゴリズム（記憶空間利用率均衡化アルゴリズム）を示す．並列ストレージシステム中で双方のアルゴリズムをある一定時間間隔 τ で起動するものとする．

なお，アクセス負荷値及び記憶空間利用率の定義は事前に与えられるものとする．後述する試作システムにおける両値の定義は **5.3** において与える．

[アクセス負荷均衡化アルゴリズム]

Step 1（状態変数の取得）：各ストレージ装置のその時点のアクセス負荷と記憶空間利用率を得る．

Step 2（起動条件のチェック）：自分のアクセス負荷値と，クラスタ内の全ストレージ装置のアクセス負荷値の平均の差が一定割合 ϵ を超えていたら Step 3 以降を実行する．

Step 3（移動先と移動量の決定）：左右のストレージ装置のうちアクセス負荷の少ないストレージ装置を移動先とし，移動先と自分のストレージ装置のアクセス負荷値の差を D，移動先と自分のストレージ装置のうちアクセス負荷値の小さい方を C，移動比率を θ としたとき，$M = C \times D \times \theta$ なる M を移動量とする．

Step 4（移動対象の決定）：自分の値域の移動先側の境界から F 個のファイルに関し，最新オブジェクトからあらかじめ設定された移動境界率 β で示される移動境界までの負荷値の合計 T が $T \geq M$ となる最小の F を求める．

Step 5（オブジェクトの移動）：自分の値域の移動先側の境界から F 個のファイルの最新オブジェクトと移動境界までの差分オブジェクトを移動先側のストレージ装置に移動する．

[記憶空間利用率均衡化アルゴリズム]

Step 1（状態変数の取得）：各ストレージ装置のその時点のアクセス負荷と記憶空間利用率を得る．

Step 2（起動条件のチェック）：自分の記憶空間利用率が，クラスタ内のストレージ装置の記憶空間利用率の平均の一定割合 δ を超えていたら Step 3 以降を実行する．

Step 3（移動先と移動量の決定）：ストレージ装置を記憶空間利用率の少ない方から η 台選択し，その中からランダムに移動先とし，移動先と自分のストレージ装置の記憶空間利用率の差を D，移動先と自分のストレージ装置の記憶空間利用率の小さい方を C，移動比率を θ，としたとき，$M = C \times D \times \theta$ なる M を移動量とする．

Step 4（移動オブジェクトの決定）：自分がもつファイルの中でアクセス頻度の低い F 個のファイルに関し，最も古い差分オブジェクトからあらかじめ設定された移動境界率 β で示された移動境界までのサイズの合計 T が $T \geq M$ となる最小の F を求める．

Step 5（オブジェクトの移動）：アクセス頻度の低い F 個のファイルの最も古い差分オブジェクトから移動境界までの差分オブジェクトを移動先側のストレージ装置に移動する．

上記の両アルゴリズムに関連する移動境界率 β，均衡化処理起動間隔 τ，移動比率 θ，アクセス負荷値差比 ϵ，記憶空間利用率差比 δ，移動先ストレージ装置数 η は，調整用のパラメータである．なお，移動比率 θ は，スピードファクタ [21] と呼ばれるもので，1 回の移動処理の起動で移動可能なデータ量に制限を設け，過剰な負荷が発生しないように制御するためのものである．

5. 試作システム

前述した機能要件に基づいて，COBALT の試作システムを Java により PC クラスタ上に実装した．以下にその概要を述べる．

5.1 並列 B$^+$-tree 構造

全 B$^+$-tree 管理手法に用いる並列 B$^+$-tree 構造には，吉原らによる Fat-Btree の実装 [22] に対して葉ノードにサイドリンクを追加し，B$^+$-tree の形にしたものを用いた．全 B$^+$-tree 管理手法で，あるバージョンファイルのバージョンを取得するときには，B$^+$-tree の根からそのバージョンファイルの最新オブジェクトを手繰り，そこからサイドリンクで目的の差分オブジェクトまでアクセスする．

提案する COBALT のアクセス構造は，この Fat-Btree 実装に差分オブジェクトを管理する連結リスト構造を付加したものを用いた．連結リストを構成するリンクノードは，B$^+$-tree の分岐ノードと同様にファンアウト分の差分オブジェクトを管理する．生成された差分オブジェクトは，最新版オブジェクトから手繰った一番新しいリンクノードにその位置を管理される．ファイルのバージョンが増えて差分オブジェクトがファンアウトを超えたときや，移動境界で連結リストを分ける場合には，新しいリンクノードが生成され，そのポインタがそれより一つ古いリンクノードのアドレスを保持する．古いバージョンの読出し時には，このポインタを手繰り古いリンクノードに順次アクセスすることで必要な差分オブジェクトの位置を得る．

試作システムでは，各オブジェクトはクラスタを構成する PC 内のローカルファイルシステム上の 1 ファイルとし，オブジェクトへのポインタはそのファイル

を表すファイル名とした．

5.2 アクセス要求変換モジュール

クライアントから並列分散ストレージシステムに送信されたアクセス要求は，その要求を受け取ったストレージ装置のアクセス要求変換モジュールによって内部アクセス要求に変換され，クラスタ内の適切なストレージ装置に渡される．

このため，アクセス要求変換モジュールは各ストレージ装置に配置される．モジュール内部では，クライアントから送られたバージョンファイルの更新と読出し要求に対して，更新要求の場合には最新オブジェクトの更新と差分オブジェクト生成，読出し要求の場合には最新オブジェクトから要求されたバージョンまでの差分オブジェクトの読出しと要求バージョンの再構成といった処理要求に変換し，実際にそれらの処理を行うストレージ装置に送信する．このモジュールのインタフェースによって，クライアントは並列分散ストレージシステム内部のバージョン管理手法を気にする必要がない．

5.3 監視モジュール

監視モジュールも各ストレージ装置に配置され，そのストレージ装置のアクセス負荷と記憶空間利用率の情報を保持する．

アクセス負荷値としては，単位時間に発生したオブジェクトへのアクセス回数と，オブジェクトのサイズの積をそのオブジェクトのアクセス負荷値とする熱[7]の概念を利用した．そのストレージ装置に格納されたオブジェクトのアクセス負荷値の総和をストレージ装置のアクセス負荷値として扱う．

記憶空間利用率としては，本論文では **3.** においてシステムを構成するすべてのストレージノードが等しい容量であると仮定していることから，(格納データ量)/(記憶可能データ量) とする．システムではまず情報を収集し移動戦略を決定するストレージ装置（以下，コーディネータと呼ぶ）を1台決定する．コーディネータは毎回異なるストレージ装置に割り当てられるのが負荷分散の観点から望ましい．コーディネータとなったストレージ装置は，監視対象のストレージ装置群に対してアクセス負荷値と記憶空間利用率の報告要求をブロードキャストで送信する．各ストレージ装置は報告要求を受信した時点での各値を収集し，コーディネータに返信する．コーディネータは得られた情報をもとにデータ配置変更の方針決定とその実行指示を行う．

6. 試作システムを用いた実験

上記の試作システムを表1に示す仕様のPCクラスタ上に実装して実験を行った．偏り制御のための情報収集に用いるコーディネータは1台とし，すべてのPCから必要な情報を集めることとした．

6.1 実験方法

表2に実験設定及び実験に用いたパラメータを示す．

初期設定としてバージョン数1のバージョンファイル100個を各ストレージノードに分配して格納する．その後，クライアントノードからバージョンファイルに対して，8個のスレッドから更新，過去バージョンの読出し要求を送信する．各スレッドごとにアクセス要求の送信間隔は，平均各100 ms間隔のポワソン分布に従うものとしその傾向は実験中は一定とした．アクセス先オブジェクトの選択にはzipfに従う偏りを発生させた．この偏りに関しても実験中は一定とした．このため，差分オブジェクト数は実験時間経過とともに増加し，各バージョンファイルに所属する差分オブジェクト数の間には同様のzipf分布に従う偏りが存在

表1 実験に用いたPCクラスタの仕様
Table 1 The specification of PCs in the experiments.

CPU	AMD Athlon XP-M1800+ (1.53 GHz)
メモリ	PC2100 DDR SDRAM 1 GByte
ディスク	TOSHIBA MK3019GAX (30 GByte, 5400 rpm, 2.5 inch)
OS	Linux 2.4.20
Java VM	Sun J2SE SDK 1.4.2_04 Server VM
Network	TCP/IP on 1 Gbit/s Ethernet

表2 実験のパラメタ
Table 2 Experimentation parameters.

parameter	value
移動境界率 β	0.7
移動比率 θ	0.5
アクセス負荷値差比 ϵ	0.2
記憶空間利用率差比 δ	0.2
移動先ストレージ装置数 η	1
均衡化処理起動間隔 τ	3 s, 5 s
過去のアクセス負荷値の記憶時間	5 s
アクセス要求送信間隔	100 ms
アクセス要求送信スレッド	8
クライアントのノード数	1
ストレージ装置のノード数	10
バージョンファイル数	100
最新オブジェクトサイズ	500 kByte
差分オブジェクトサイズ	50 kByte
ページサイズ	500 kByte
B^+-tree ノードのファンアウト	30
B^+-tree ノードのキャッシュサイズ	32 MByte

する．

初期設定の終了後 COBALT による偏り制御アルゴリズムを稼動させ，一定時間ごとに偏り除去操作を行うこととした．

実験では，記憶空間利用率を均衡化するためにはクラスタ内の記憶空間利用率が最大のストレージ装置から最小のストレージ装置に向けてオブジェクト移動が行われるとする（つまり，移動先ストレージ装置数 $\eta = 1$）．アクセス負荷分散を記憶空間利用率の分散より優先度を高くし，負荷値の偏りが基準値を満たし，かつ記憶空間利用率が基準値から外れた場合にのみ記憶空間利用率均衡化アルゴリズムが起動するようにした．

以上の条件で，アクセス負荷値と記憶空間利用率の偏りの推移の測定を行う．

実験 1 として，両均衡化アルゴリズムを稼動させた状況で，一定時間ごとに各ストレージ装置の記憶空間利用率とアクセス負荷値を観測し，均衡化の効果を測定する．同時に，均衡化処理起動間隔 τ を変化させた場合の偏り除去効果についても測定する．

ストレージ装置のアクセス負荷値は，1 秒当りの平均値で表す．過去の負荷は一定時間（5 秒間）だけ記憶されるものとする．つまり，5 秒以上前のアクセス傾向は考慮されない．このため，負荷分散処理の発生間隔を 3 秒と 5 秒の 2 種類とした．比較の対象として，COBALT の索引構造のみを用い均衡化処理を行わない場合（COBALT(index only) と表記）と，全 B^+-tree 管理手法と値域分割を用いた均衡化処理（single B^+-tree と表記）を用意した．記憶空間利用率とアクセス負荷値の偏り除去程度は，初期化後に均衡化アルゴリズムを起動してからの経過時間に対する標準偏差を用いて表す．

6.2 実験結果と考察

実験の結果を図 5 に示す．実験期間の 150 秒中に，差分オブジェクトは合計でおよそ 12000 個（600 MByte）増加したことになる．

アクセス負荷値の偏りが従来手法の全 B^+-tree 管理手法と同程度かやや低くなっているのと同時に，記憶空間利用率の偏りが従来手法に比べて小さいまま推移していることが分かる．このことから，従来手法に比べて COBALT による偏り制御アルゴリズムはアクセス負荷と記憶空間利用率の同時均衡化の効果が高いといえる．

均衡化処理の起動間隔を変えた場合は，アクセス負荷の偏りに関しては，COBALT も従来手法も，長時間経過後に発動間隔が長い方がアクセス偏りがやや小さくなるという結果になった．一方，記憶空間利用率の偏りは，従来手法は発動間隔が長い方が偏りが小さくなるのに対して，COBALT では発動間隔によらずほぼ同等の性能を示した．

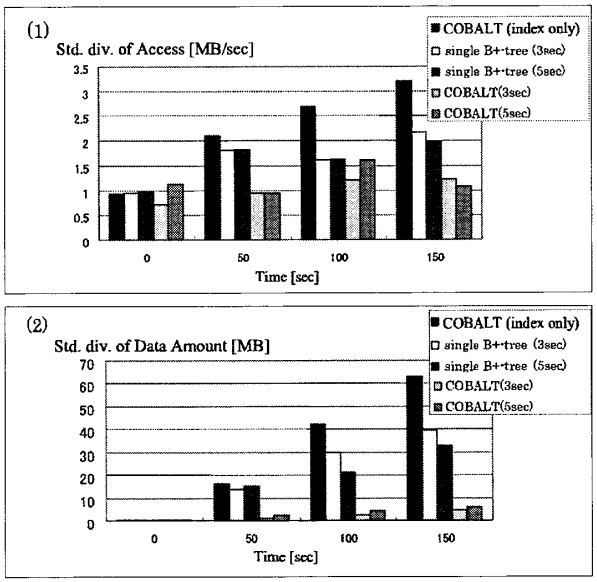

図 5 経過時間に伴う偏り推移（標準偏差）(1) アクセス負荷偏り (2) 記憶空間利用率偏り

Fig. 5 Experiment 1: Transition of (1) access load and (2) space utilization ratio skews (standard deviation) for execution time.

実験では長時間経過した後で均衡化処理起動間隔を小さくとった方が偏りが大きくなる場面がある．この原因としては，バージョンファイルの数が少ないということが考えられる．また，ファイル間のアクセス偏りは zipf 分布で発生させているために，実際にアクセスの多いバージョンファイル数はごく少数である．このため，アクセス負荷の高い一部のオブジェクトが移動を繰り返すことになり，偏りがうまく除去されないことがあるためではないかと予想される．

これを確認するために，オブジェクトのサイズを小さくし，ファイル数を増やし負荷分散の発動間隔を変えて実験を行った．変更したパラメータを表 3 に，このときのアクセス負荷の偏りの変化を図 6 に示す．この実験では，アクセス偏りが大きい状態から負荷分散処理を始め，偏りが除去される様子を観察している．横軸 $Time$ は実験開始後（データ挿入前）からの経過時間を示しており，図 5 と異なることに注意されたい．

結果は，均衡化処理起動間隔が短い方がより偏りを小さくできることを示しており，上で述べた予想と一

表 3 ファイル数変更による確認実験時に変更したパラメータ
Table 3 Modified parameters in the further exeperiment of Exp.1.

均衡化処理起動間隔 τ	1 s, 5 s
最新オブジェクトサイズ	3.5 kByte
差分オブジェクトサイズ	350 Byte
ページサイズ	4 kByte
バージョンファイル数	12800
アクセス要求送信スレッド	64

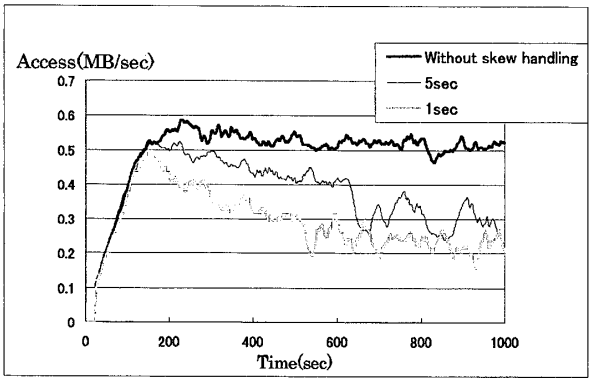

図 6 ファイル数が多いときのアクセス偏り推移
Fig. 6 Access load skews with a large number of files.

致することが分かる．

7. むすび

本論文では，並列分散ストレージシステム上でファイルのバージョン管理を行うため，バージョン管理を行う並列分散ストレージシステムにおけるアクセス負荷と記憶空間利用率の同時均衡化手法である COBALT を提案し，その評価を行った．

本論文では，提案する COBALT の効果を評価するために，PC クラスタ上に並列ストレージの試作システムを構築し，偏り除去効果とシステムの応答性能を，すべての差分オブジェクトを単一の並列 B^+-tree のみで管理する手法と比較した．実験の結果，COBALT ではアクセス負荷の均衡化とデータ量の分散の両立を実現できることを確認した．

今後の課題としては，様々なサイズのファイルが混在する状況での評価や，多種多様なアクセス傾向や更新頻度などパターンを使用した評価による COBALT の適用可能性の考察が考えられる．また，異なるストレージユニットが混在するシステムにおけるアクセス負荷とデータ均衡に関する研究が重要であると考えている．

格納データのアクセス傾向を決定する別の要因として，ソフトウェア開発等でバージョン管理を行う場合，プロジェクト単位，ワークタスク単位といった意味的なファイル間利用傾向が存在する．Li らのデータマイニング手法を利用したストレージ管理手法 [23] のように，このような意味的なファイル間利用傾向の差の抽出及び利用も将来の課題と考えられる．

謝辞 本研究の一部は，独立行政法人科学技術振興機構戦略的創造研究推進事業 CREST，NHK 放送技術研究所，情報ストレージ研究推進機構（SRC），文部科学省科学研究費補助金特定領域研究（16016232）及び東京工業大学 21 世紀 COE プログラム「大規模知識資源の体系化と活用基盤構築」の助成により行われた．

文　献

[1] Netapp snapshot. http://www.sourceforge.net
[2] SNIA. Data management forum. http://www.snia-dmf.org/
[3] J. Miyazaki and H. Yokota, "Concurrency control and performance evaluation of parallel B-tree structures," IEICE Trans. Inf. & Syst., vol.E85-D, no.8, pp.1269–1283, Aug. 2002.
[4] D. Amarmend, M. Aritsugi, and Y. Kanamori, "POP: A concurrency control protocol for parallel B-trees," IPSJ Trans. Databases, vol.45, no.SIG14, pp.30–38, Dec. 2004.
[5] 吉原朋宏，小林　大，田口　亮，横田治夫，"Fat-Btree における B-link を用いた並行性制御手法，" 情処学研報，DBS-140(II)(90)，July 2006.
[6] H. Simitci, Storage Network Performance Analysis, Wiley Technology Publishing, 2003.
[7] G. Weikum, P. Sabback, and P. Scheuermann, "Dynamic file allocation in disk arrays," Proc. ACM SIGMOD, pp.405–415, May 1991.
[8] V. Gaede and O. Günther, "Multidimensional access methods," ACM Comput. Surv., vol.30, no.2, pp.170–231, 1998.
[9] A. Henrich, H.W. Six, and P. Widmayer, "The lsd tree: Spatial access to multidimensional and non-point objects," Proc. Fifteenth International Conference on Very Large Data Bases, pp.45–53, Morgan Kaufmann Publishers, 1989.
[10] B. Becker, S. Gschwind, T. Ohler, B. Seeger, and P. Widmayer, "An asymptotically optimal multiversion b-tree," VLDB J., vol.5, no.4, pp.264–275, 1996.
[11] 中野真那，小林　大，田口　亮，上原年博，横田治夫，"バージョン管理用差分情報のアクセス頻度に着目した分散データ配置，" 日本データベース学会 Letters, vol.4, no.1, pp.121–124, June 2005.
[12] W.F. Tichy, "RCS—A system for version control," Software—Practice and Experience, vol.15, no.7, pp.637–654, 1985.

[13] M.J. Rochkind, "The source code control system," IEEE Trans Softw. Eng., SE-1, pp.364–370, 1975.
[14] B. Berliner, "Cvs ii: Parallelizing software development," Proc. Winter 1990 USENIX Conference, pp.341–352, 1990.
[15] J. MacDonald, File System Support for Delta Compression, Masterthsis, Department of Electrical Engineering and Computer Science, University of California at Berkley, 2000.
[16] S.-Y. Chien, V.J. Tsotras, and C. Zaniolo, "Efficient management of multiversion documents by object referencing," The VLDB J., pp.291–300, 2001.
[17] J.J. Hunt, K.-P. Vo, and W.F. Tichy, "Delta algorithms: An empirical analysis," ACM Trans. Softw. Eng. Methodol., vol.7, no.2, pp.192–214, 1998.
[18] H. Yokota, "Autonomous disks for advanced database applications," Proc. International Symposium on Database Applications in Non-Traditional Environments (DANTE'99), pp.441–448, Nov. 1999.
[19] B. Seeger and P.-Åke Larson, "Multi-disk b-trees," SIGMOD Conference, J. Clifford and R. King, ed., pp.436–445, ACM Press, 1991.
[20] H. Yokota, Y. Kanemasa, and J. Miyazaki, "Fat-btree: An update-conscious parallel directory structure," ICDE, pp.448–457, IEEE Computer Society, 1999.
[21] H. Feelifl and M. Kitsuregawa, "Ring: A strategy for minimizing the cost of online data placement reorganization for btree indexed database over shared-nothing machines," DASFAA, pp.190–199, IEEE Computer Society, 2001.
[22] 吉原朋宏，渡邊明嗣，小林 大，田口 亮，上原年博，横田治夫，"並列 btree 構造における負荷分散処理の並行性制御への影響，" 情処学研報，2005-DBS-137, 2005.
[23] Z. Li, Z. Chen, S.M. Srinivasan, and Y. Zhou, "C-miner: Mining block correlations in storage systems," 3rd USENIX Conference on File and Storage Technologies, pp.173–186, 2004.

（平成 18 年 5 月 16 日受付，8 月 25 日再受付）

中野 真那

2004 東工大・工・情報工卒．2006 同大大学院・情報・修士課程了．

小林 大

2003 東工大・工・情報工卒．2005 同大大学院・情報・修士課程了．現在，同大学院博士後期課程在学中．2006 日本学術振興会特別研究員 DC．並列ストレージシステムの自律管理に関する研究に従事．日本データベース学会学生会員．

渡邊 明嗣

2000 東工大・工・情報工卒．2002 同大大学院・情報・修士課程了．現在，同大学院博士後期課程在学中．並列データベースの研究に従事．日本データベース学会学生会員．

横田 治夫 （正員）

1980 東工大・工・電物卒．1982 同大大学院・情報・修士課程了．同年富士通（株）入社．同年 6 月（財）新世代コンピュータ技術開発機構研究所．1986（株）富士通研究所勤務．1992 北陸先端大・情報・助教授．1998 東工大・情報理工・助教授．2001 東工大・学術国際情報センター・教授．工博．主としてデータベース，データ工学向けの並列アーキテクチャなどに関する研究に従事．情報処理学会，人工知能学会，日本データベース学会，IEEE，ACM 各会員．