バージョン管理を行う分散ストレージにおける偏り監視範囲分割の影響

The Effect of Scope Division for Handling Skews in Distributed Storage with Version Management

中野 真那♡ 小林 大♢♣ 渡邊 明嗣♢
横田 治夫♠

Mana NAKANO Dai KOBAYASHI
Akitsugu WATANABE Haruo YOKOTA

我々は，分散ストレージでバージョン管理を行う場合に，バージョン間のアクセス頻度の差を利用して各ストレージのアクセス負荷とデータ格納量を同時に均衡化させる COBALT を提案している．COBALT は，高アクセス頻度データに対する高速なアクセスと，データ格納量均衡化のための柔軟なデータ配置を可能とするアクセス構造を持つ．しかし，COBALT ではバージョン間のアクセスパスにストレージ装置をまたがる連結リストを用いているため，低アクセス頻度の旧バージョンの応答性能が悪化することが考えられる．また，偏り均衡化のための計算コストが，システム規模が拡大することによって増大するという問題点もある．本稿では，偏りを監視する範囲を分割することによって計算コスト増加の問題点を改良する．実装システムを用いて，応答性能やシステム拡張時の性能の変化を調べ，バージョン数を指定した応答性能が単なる並列 B⁺-tree を用いた場合に比べ悪くならないことと，偏り監視範囲の分割によって計算コストを低く抑えながら均衡化が可能であることを示す．

We have proposed COBALT as a method for balancing both access load and data amount on each storage node of a distributed storage system utilizing the difference of access frequency between versions. The COBALT has an access path structure enabling fast accesses for frequently accessed data and flexible data placement for balancing data amount. However, it may increase access latency for rarely accessed older versions by linked lists beyond storage nodes, and also requires high costs for handling skews in a larger scale system. In this paper, we improve the scalability of COBALT by dividing the scope of handling skews in the system, and evaluate the performance of an experimental system. The experiments demonstrate that the latency for specific version data is still better than a method using ordinary parallel B⁺-tree, and the scope division is effective for reducing costs of handling skews in a large system.

1. はじめに

繰り返し更新が行われるファイルに対し，途中の状態を複数保存しておくバージョン管理は有用である．差分を用いたファイル版管理手法は，保存しなければならないデータ量が少ないことから，広く利用されている [5, 2, 3, 9]．バージョン管理下にあるデータの量は，ファイルの更新や，管理対象ファイルの数やサイズの増加に伴って増大する [1]．そのような大量のデータを安定して格納し，効率良くアクセスするために，並列分散構成の大容量で高性能かつ高信頼なストレージシステムが用いられる．並列分散システムの性能を維持するためには，個々のストレージ装置間でアクセス負荷やデータ格納量が偏らないようにデータ配置を管理することが重要である [4, 6]．しかし，システムの拡張性を重視した，並列 B⁺-tree を用いた値域分割によるデータ管理法は，一次元的なデータ配置を行うため負荷とデータ格納量の両方を均衡化させることは難しい．また，バージョンによる利用頻度の差を考慮して，高アクセス頻度のデータを高速に提供可能な管理手法が望ましい．

これらの要求を満足させるために，我々は，並列 B⁺-tree と連結リストを用いて分散ストレージ上のバージョン管理されたファイルに対するアクセス負荷とデータ格納量を同時に均衡化させる COBALT (Combination Of Btree-node And Linked list Transfer) を提案している [8]．COBALT は，バージョン間にアクセス頻度の差があることを利用し，アクセス頻度の高い最新バージョンのファイルを直接並列 B⁺-tree で管理してアクセス負荷を均衡化させるとともに，アクセス頻度の低い旧バージョンのファイルを最新バージョンからの連結リストで管理してストレージ間で柔軟に移動させることでデータ量の均衡化を行い，二つの均衡化を両立させる手法である．これまでに，シミュレーションとプロトタイプシステムを用いてその効果を示してきた [7]．

しかし，COBALT は連結リストによって差分情報をシステムの任意の位置に配置可能にしているため，システム規模拡大によってデータ配置変更先候補が増加することにより，偏り除去の際の配置決定コストが増大する．このため，本稿では，COBALT のスケーラビリティ拡張を目的に，データ配置の管理範囲を分割することで偏り除去処理コストの削減を行う方法を提案する．また，COBALT は過去のバージョンへアクセスする際に，連結リストの逐次探索によって差分情報を読み出すため，アクセスコストがリンクをたどる数とその際発生するストレージ装置間の通信コストによって増加する可能性がある．これらを評価するため，COBALT を適用した分散ストレージシステムの応答性能および，偏り監視範囲の分割による処理コストの変化を評価する．

2. COBALT の概要

2.1 管理モデル

COBALT は Reverse-delta[5] によるバージョン管理方法を想定している．本稿では，バージョン管理下に置かれるファイルをバージョンファイルと呼ぶ．バージョンファイルの最新バージョンのファイル内容が書き込まれている最新版オブジェクト (LO) と，一つ前の版に戻るための差分情報が書き込まれている差分オブジェクト (DO) が配置される領域をレポジトリと呼ぶ．COBALT はオブジェクトをデータ配置管理の単位として扱い，ファイル更新や読み出しの際は，必要なオブジェクトの内容へアクセスを行う．このアクセスにより，最新版オブジェクトへのアクセス量が最も多く，差分オブジェクトへのアクセス量は古くなるにつれて単調減少すると考えられる．また，アクセス傾向には他の特性 [8] もあるが，本稿では単調減少を前提に配置決定を行う．

♡ 正会員 日本アイビーエムインダストリアルソリューション (株)
mana@de.cs.titech.ac.jp
♢ 学生会員 東京工業大学 大学院 情報理工学研究科 計算工学専攻
{daik,aki}@de.cs.titech.ac.jp
♣ 日本学術振興会特別研究員 DC
♠ 正会員 東京工業大学 学術国際情報センター
yokota@cs.titech.ac.jp

各オブジェクトは，複数のストレージ装置をネットワークで接続した分散ストレージシステム上に配置される．値域分割によって各ストレージ装置に分配され，それぞれのストレージ装置が並列 B$^+$-tree を用いた分散ディレクトリを持つことにより，それらの位置を管理する．

負荷管理のためには，システム内で動的に定められるコーディネータの役割をするストレージ装置がデータ移動戦略を決定し，該当ストレージ装置にデータ転送指示を送信する．コーディネータはシステム中に一台以上存在し，指定された監視領域内のストレージ装置について負荷管理を行う．コーディネータの故障時は，新たなコーディネータをシステムからランダムに選ぶ．

2.2 アクセス構造

我々が比較対象とする従来手法では，オブジェクトを管理するために並列 B$^+$-tree のみを用いる．オブジェクトを値域分割で各ストレージ装置に配分し，ストレージ装置毎にオブジェクトを B$^+$-tree の部分木によって管理する．オブジェクトは種類によらず，すべて B$^+$-tree の葉ノードに格納される．以下では，この従来手法を全 B$^+$-tree 管理手法と呼ぶ．

これに対して，COBALT では上記のオブジェクトのアクセス傾向を考慮し，並列 B$^+$-tree と連結リストを組み合わせたアクセス構造を用いる．最新版オブジェクトは全 B$^+$-tree 管理手法と同様に管理する一方，差分オブジェクトはバージョンファイル毎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システム中の任意のストレージ装置に配置可能である．

この構造は，B$^+$-tree が最新版オブジェクトのみを保持するため，全 B$^+$-tree 管理手法に比べて木のサイズが小さい．よって，高速メモリ上に Btree 部分を置くことでアクセス量の多い新しいオブジェクトに高速アクセスが可能になる．

2.3 偏り除去方法

データ移動によって，ストレージ装置間のアクセス負荷やデータ格納量の偏りを除去する．アクセス負荷の均衡化のためにはアクセス量の多い，より新しいオブジェクトを論理的に隣接したストレージ装置に移す．また，同時にデータ格納量を均衡化させるために，アクセス負荷分散に影響を与えないアクセス量の少ない，より古いオブジェクトを移す．移動先にはデータ格納量の少ないストレージ装置を選択する．差分オブジェクトの配置は値域分割の影響を受けないためにこのような配置戦略が可能となる．移動させるオブジェクトの決定には，移動境界 [8] を用い，配置決定アルゴリズム [8] に従って，データを移動する．

3. 分割管理による拡張性の改良

COBALT はアクセス構造に連結リストを用いているため，過去のバージョンを読み出すときにはリストの逐次探索が必要になる．このため古いバージョンを読み出す際のコストが大きい．偏り除去処理によって読み出し対象のオブジェクトが複数のストレージに分散している時には，台数分のディスク間通信コストが発生する．また，偏り除去方針の決定のために，システム内のすべてのストレージ装置の負荷状況を集約してから，最適なデータ配置変更方針を作成するため，システム規模が大きくなると，他のストレージ装置との通信コストや，配置方針決定までの計算コストが大きくなるという問題がある．

これを解決し，COBALT の拡張性を維持するために，偏り除去を行うときに考慮するストレージ装置の数を限定し，そのストレージ装置群の中で，アクセス負荷やデータ格納量が均衡化される状況を目指す分割管理を行う．このストレージ装置群をサブクラスタと呼ぶ．分割管理では，サブクラスタ間に重複するストレージ装置が存在するように，サブクラスタを複数定義することで，システム全体の負荷管理を行いながら，ストレージ装置間の通信量を減らすことができる．図 1 にサブクラスタの例を示す．この例では，それぞれのストレージ装置に対して左右の論理的に連続した 4 台のストレージ装置をサブクラスタとして監視する．各ストレージ装置がコーディネータとなった時には，そのサブクラスタの管理を行う．

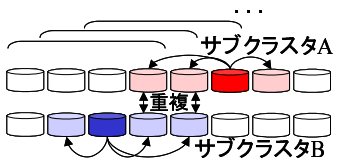

図 1: サブクラスタの例
Fig.1 An example of division with sub-cluster

4. 実験と考察

4.1 実験方法

我々はこれまでに Java を用いて COBALT のアクセス構造と負荷管理機能を PC クラスタ上に実装した分散ストレージシステムのプロトタイプ [7] を用いて，COBALT の性能を調べてきた．クラスタ内の個々の PC が上記の機能を持つストレージ装置として動作する．

このシステムにオブジェクトを格納して初期木を構成したのち，新規ファイルの挿入，既存ファイルの更新，バージョンを指定したファイルの読み出しという三種の初期クエリを送信する．これらは値域の中央部にピークがあるような偏り分布に従う．このシステムにおいて，一定時間毎に偏り除去操作を実行する．偏り除去操作を簡略化し，データ格納量の均衡化のためのデータ移動先は，システム内で一番格納量が少ないストレージ装置とする．測定するのは，クエリ送信元がクエリを送信してから，その結果を受け取るまでの時間である．ディスク間通信によるネットワーク遅延の影響を除外するために，応答時間を測定するのは，送信クエリが送信元のディスク内でのみ処理されたときとする．ただし，COBALT の測定実験では，連結リスト探索の際に発生する通信コストは測定に含めている．また，リストをたどるコストを比較するために，葉ノードとリンクノードのファンアウトを通常よりも小さくした．実験パラメタとシステム構成を表 1 に示す．

4.1.1 実験 1:バージョンを指定したファイル読み出し要求の応答時間の比較

初期クエリのうち，バージョン読み出しクエリ以外を停止した状態で，指定されたバージョンのバージョンファイルを取り出すのに必要な応答時間を測定する．また，初期クエリ送信時から応答測定前の間にデータ移動による偏り除去処理を行った場合の応答時間も測定する．データ移動間隔を 3sec とし，応答測定開始は実験開始から 1000sec 経過後とした．

全 B$^+$-tree 管理手法と COBALT の B$^+$-tree が必要とするメモリ量の違いを比較するため，キャッシュされるページ

表 1: 実験環境
Table 1 parameter for experiments

	実験 1	実験 2
クエリ送信間隔	1000msec	
ページサイズ	6KB	
ファイルサイズ	100KB（LO），10KB(DO)	
クエリ送信スレッド	8	
送信間隔	100msec	
データ測定スレッド	1	
ストレージ装置台数	32	32
測定期間	500sec	1500sec
取り出すバージョン数	0，3，9，12，17	
バージョンファイル数	64000/台	32000/台
キャッシュサイズ	32MB	32MB
Btree ノードの fanout	8	8
初期クエリ継続時間	1000sec	1000sec
偏り除去処理発動時間		200sec 後
移動境界	80	80
サブクラスタのストレージ装置数	8, 16, 32	
Sun Microsystems Sun Fire B100x		
CPU	AMD Athlon XP-M 1800+ (1.53GHz)	
MEM	PC2100 DDR SDRAM ECC 1GB	
Network	1000BASE-T	
HDD	TOSHIBA MK3019GAX (30GB, 5400rpm, 2.5inch)	
OS	Linux 2.4.20	
Local File System	ReiserFS	
Java VM	Sun J2SE SDK 1.5.0_05 Server VM	

表 2: 実験 1：偏り制御の有無による応答性能の比較 (msec)
Table 2 Response time comparison with/without skew handling

	0 個	3 個	9 個	12 個	17 個
a	245.62	678.99	1508.85	1631.28	2149.36
b	216.74	529.94	1287.22	1421.01	2072.84
c	145.93	337.45	677.49	847.20	1389.76
d	136.43	302.34	639.82	766.66	1011.56

数に対して初期木は十分大きい状態で実験を行う．

4.1.2 実験 2：分割管理による偏り除去方針決定までの処理時間の比較

データ格納量の偏りが発生している状況で偏り除去処理を行い，各ストレージ装置の状態調査からデータの移動量と移動先を決定するまでの処理時間を測定する．サブクラスタに含まれるストレージ装置数を変化させ，処理時間と偏り除去効果を観察する．実験では，図 1 のように，左右の論理的に連続した N 台のストレージ装置がサブクラスタに含まれる構成を用いた．

4.2 結果と考察

実験 1 の結果を表 2 に示す．a:全 B$^+$-tree 管理手法，b:全 B$^+$-tree 管理手法 (偏り除去後)，c：COBALT，d：COBALT(偏り除去後) を示し，読み出す差分の数と，そのときの応答時間を表している．全 B$^+$-tree 管理手法，COBALT ともに読み出す差分オブジェクトの数が増加することで平均応答時間は長くなり，全 B$^+$-tree 管理手法に比べて COBALT の読み出し時間が短くなることが分かる．さらに，測定前にデータ配置の偏りを除去しておくことで，応答時間が短縮されることも分かる．

COBALT と全 B$^+$-tree 管理手法の応答時間の差は，アクセス構造の違いによって生じる．実験では，応答時間の差を強調するために，ファンアウトを通常よりも減らし，COBALT のアクセス構造の Btree の内部ノードが入りきる程度の量の LRU キャッシュを設定している．Btree に対してランダムにオブジェクト挿入が行われると，差分オブジェクトを読み出すときに経由する内部ノードと，最新版オブジェクトを読み出すときに経由する内部ノードが同様の扱いを受ける．このため，LRU に従うキャッシュアルゴリズムでは，ファイル更新により追加された内部ノードによって，アクセス頻度の高いオブジェクトへのパス上にあるページがディスクに書き出されてしまい，アクセスに長時間必要とすることが考えられる．差分オブジェクトはバージョン番号に基づいて順番に追加されるため，全 B$^+$-tree 管理手法ではノードのスプリット後に充填率が低いまま残るノードがあることも，内部ノード数の増加に影響を与える．

一方，COBALT では Btree の内部ノードが最新版オブジェクトへのアクセスのためだけに使われているため，アクセスが必要な内部ノードが常にキャッシュにある可能性が高く，オブジェクトのアクセスが短時間で行える．また，差分オブジェクトが追加されるときには，リンクノードがいっぱいになるまでひとつのノードにオブジェクトを追加し，その後新しいリンクノードを作成する．このため，過去のバージョンにアクセスするときに読まなければいけないリンクノードの数が全 B$^+$-tree 管理手法よりも少なくなり，短時間のアクセスが可能となる．

読み出す差分の数が 0 のときの応答時間の差は，Btree の根から葉までの探索を行うための時間の差を示す．上で述べたように，内部ノードのアクセスのためにディスクアクセスを伴う可能性が相対的に高い全 B$^+$-tree 管理手法の方が応答時間が長い．また，COBALT による応答時間の改善量は，読み出すバージョンの数が増加したときと比較すると小さいが，偏り改善率も考慮すれば効果は高いと言える．読み出す差分の数が増加することで，全 B$^+$-tree 管理手法と COBALT の応答時間の差が増加するのは，葉ノードやリンクノードを読み出すコストの影響だと考えられる．全 B$^+$-tree 管理手法の方がひとつの差分情報を読み出す時間が長くなるために，読み出す差分の数が増えることで応答時間の差が広がる．ただし，読み出す差分の数がさらに増加すると，COBALT のリンクノードがキャッシュ上に見つからない率も増加するため，全 B$^+$-tree 管理手法との応答時間の差が縮まっていく．応答時間の差は内部ノードに使用されるメモリ量の差とオブジェクトへのキャッシュヒット率が影響していると考えられるため，さらにキャッシュ管理を含めた詳細な調査が必要である．

また，偏り除去操作によってアクセス集中による応答性能劣化が減少するために，全 B$^+$-tree 管理手法，COBALT ともに応答時間が短縮される．COBALT では，データマイグレーションによって差分オブジェクトがばらばらのストレージ装置に移動された場合，読み出しの際にディスク間通信が発生するために応答速度の劣化が予測された．しかし，今回の実験ではデータ移動後も応答時間は劣化していない．理由としては，偏り除去処理を行なうことによりストレージ装置の処理性能が改善されるため，通信コストを含めても応答時間が抑えられることがあげられる．

実験 2 の結果を表 3，表 4，表 5 に示す．サブクラスタに含まれるストレージ装置台数を減少させることで，システム全体の偏り除去効果が低下することが予想されるが，表 4 では，台数を変更したことによる偏り除去効果の差は僅かである．これは，送信クエリの偏りパターンのピークがひとつであったため，データ移動方針決定時のデータの移動元と移動先の決定結果に影響が少なかったからと考えられる．

表 5 では，サブクラスタに含まれるストレージ装置台数が増えることで，処理の平均時間が増加している．また，32 台のときには，処理が行われるまでに特に長い時間がかかる場合があることが分かる．処理コストに影響するのは，ネッ

表 3: 実験 2:データ格納量の偏りの推移 (MB/sec)
Table 3 Skew with scope division

	偏り除去なし	32 台	16 台	8 台
500sec	97.81	87.41	86.13	82.02
1000sec	201.53	156.18	154.46	158.02
1500sec	282.88	218.27	213.17	223.51

表 4: 実験 2:データ格納量の平均に対する標準偏差の割合 (%)
Table 4 Skew handling with scope division

	偏り除去なし	32 台	16 台	8 台
500sec	2.52	2.23	2.22	2.12
1000sec	4.03	3.01	2.9	3.03
1500sec	4.73	3.28	3.2	3.32

トワーク状況や各ストレージ装置が処理しているクエリの状況といった，負荷量の応答を得るまでの時間や，コーディネータが得られた回答を処理してデータの移動先と移動量を決定するまでの計算コストなどである．台数が多いとこれらの要因による遅延が大きくなるということが，実験結果の処理コストの平均とばらつきの増大によって分かる．さらに，サブクラスタに含まれるストレージ装置台数を減らすことで，このコストを削減できることが分かる．これにより，COBALT の偏り除去処理のパフォーマンスをシステム拡張時も維持するためには，サブクラスタを設定し，偏り監視対象のストレージ装置台数を限定することが有効であると分かる．データ移動のための処理コストの削減は，システムの通常動作に影響を与えずにデータ移動を行うために重要である．さらに，より効果的な偏り除去処理のために，アクセスパターンも考慮に入れたサブクラスタ台数の調整といったチューニングを考えることも必要である．

5. 結論と今後の課題

本稿では，ファイルバージョン管理下にあるデータを分散ストレージシステム上に格納する際に，システム性能劣化を防ぐデータ配置を行うために我々が提案している COBALT について，システム規模拡張の際に発生する問題点の解決と評価を行った．

COBALT のアクセス構造によって影響を受けると考えられる，バージョンを指定したファイル読み出しクエリの応答性能を測定する実験により，COBALT は読み出すバージョン数が多くなったときにも全 B^+-tree 管理手法より応答時間が抑えられることが示された．偏り除去操作を加えることで，さらに応答時間の劣化を抑えることもできた．

また，負荷状況を監視する対象のストレージ装置台数を変化させた実験を行った．監視対象のストレージ装置台数を減らしても偏り除去効果への影響の違いがあまり見られなかったが，台数を削減することは偏り除去方針決定までの処理コストを小さく抑えることができることがわかった．

今後の課題としては，現実に近い様々な種類やサイズの

表 5: 実験 2:サブクラスタの台数と偏り除去処理コストの比較
Table 5 Scope size and the cost for skew handling

	8	16	32
平均 (msec)	7.75	9.26	60.55
標準誤差 (msec)	4.96	3.56	47.25

ファイルが混在する環境での評価や，さらに高速な応答を可能にするためのメモリチューニング方法，アクセスパターンに応じたサブグループの規模変更などが挙げられる．また，Reverse-delta によるバージョン管理手法に限らず，段階的にアクセス頻度が低下していくようなデータに対して本手法を適用する際の配置戦略の変更や，それを一般的な計算手法で表現することが必要である．

[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，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) および東京工業大学 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」の助成により行なわれた．

[文献]

[1] Sourceforge. http://www.sourceforge.net.
[2] B. Berliner. Cvs ii: Parallelizing software development. *In Proceedings of the Winter 1990 USENIX Conference*, 1990.
[3] J. MacDonald. File system support for delta compression, 2000.
[4] H. Simitci. *Storage Network Performance Analysis*. Wiley Technology Publishing, 2003.
[5] W. F. Tichy. RCS — a system for version control. *Software — Practice and Experience*, 15(7):637–654, 1985.
[6] G. Weikum, P. Sabback, and P. Scheuermann. Dynamic File Allocation in Disk Arrays. *Proceedings of ACM SIGMOD*, pages 405–415, May 1991.
[7] 中野真那, 小林大, 渡邊明嗣, 上原年博, 田口亮, and 横田治夫. アクセス頻度と容量分散を考慮した版管理用データ配置法の実装と評価. In **信学技報** *Vol.105,No.337*, volume 105 of *DE2005-130*, pages 31–36, 東京, 10 月 2005.
[8] 中野真那, 小林大, 渡邊明嗣, 上原年博, 田口亮, and 横田治夫. 分散環境でのファイル版管理のためのアクセス頻度を考慮したデータ配置法. In **第 *16* 回電子情報通信学会データ工学ワークショップ論文集，*DEWS2005*，*5B-i5***, Feb. 2005.
[9] 天海良治, 一二三尚, 小西隆介, 佐藤孝治, 木原誠司, and 盛合敏. Linux 用ログ構造化ファイルシステム nilfs の設計と実装. In **情報処理学会**, volume 2005 of *2005-OS-099*, 5 月 2005.

中野 真那　Mana NAKANO
平 18 東工大大学院・情報理工・計算工・修士課程了．日本アイビーエムインダストリアルソリューション株式会社勤務．日本データベース学会正会員．

小林 大　Dai KOBAYASHI
平 17 東工大大学院・情報理工・計算工・修士課程了．同大大学院・情報理工・計算工・博士後期課程在学中．日本学術振興会特別研究員 DC．日本データベース学会学生会員．

渡邊 明嗣　Akitsugu WATANABE
平 14 東工大大学院・情報理工・計算工・博士前期課程了．同大大学院・情報理工・計算工・博士後期課程在学中．日本データベース学会学生会員．

横田 治夫　Haruo YOKOTA
昭 57 東工大大学院・情報・修士課程了．同年富士通 (株)．同年 6 月 (財) 新世代コンピュータ技術開発機構研究所．昭 61(株) 富士通研究所．平 4 北陸先端大・情報・助教授．平 10 東工大・情報理工・助教授．平 13 東工大・学術国際情報センター・教授．工博．日本データベース学会, 電子情報通信学会，情報処理学会，人工知能学会，IEEE，ACM 各会員．