分散ストレージシステム上での複製を含むデータの分割配置とアクセススケジューリング

片居木　誠[†]　　小林　　大[††]　　吉原　朋宏[††]　　小林　隆志[†††]　　田口亮[††††]

横田　治夫[†††,††]

† 東京工業大学 工学部 情報工学科
†† 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, †††† NHK 放送技術研究所
E-mail: {{mkataigi,daik,yoshihara}@de,tkobaya@,yokota@}cs.titech.ac.jp, taguchi.r-cs@nhk.or.jp

あらまし　我々はこれまで分散ストレージシステムにおける負荷分散，容量分散，耐故障，規模変更等を行う様々な自律的な管理機能について研究を行ってきた．その際，これらの機能を実現する処理や戦略を単純化するため，アクセス単位となる対象の 1 つのオブジェクトが 1 つのストレージ装置に格納されることを前提にしていた．一方，近年，高精度のマルチメディアストリームオブジェクトなどの増加により，ストレージシステムに対する安定した高速アクセスの要求が高まっている．この要求を満たすためには，複数の分散ストレージ装置にまたがるストライピングが必要となる．しかし，従来の RAID で用いられているストライピングはマルチメディアストリームオブジェクトには適さない．本稿では，安定した高いスループットの分散ストレージシステムを実現するため，これまでの研究を拡張して，負荷を考慮した動的な配置を行うストライピング戦略に関して検討を行う．また，スループット改善のために，ストライピングされた複製に対するスケジューリング戦略に関しても検討を行う．さらに，我々が提案している自律ディスク上に実装し，実験によりそれらの効果を評価する．

キーワード　複製，並列・分散 DB，ストレージシステム，ストライピング

Access Scheduling for Dynamic Allocatable Striping Data and Replicas on a Distributed Storage System

Makoto KATAIGI[†], Dai KOBAYASHI[††], Tomohiro YOSHIHARA[††],

Takashi KOBAYASHI[†††], Ryo TAGUCHI[††††], and Haruo YOKOTA[†††,††]

† Dept. of Computer Science, Faculity of Engineering, Tokyo Institute of Technology
†† Dept. of Computer Science, Grad. School of Info. Sci. and Eng, Tokyo Institute of Technology
††† Global Scientific Information & Computing Center, Tokyo Institute of Technology
†† NHK Science & Technical Research Laboratories
E-mail: {{mkataigi,daik,yoshihara}@de,tkobaya@,yokota@}cs.titech.ac.jp, taguchi.r-cs@nhk.or.jp

Abstract　We have studied many functions to realize the distributed autonomous storage management, such as workload balancing, data amount balancing, failure recovery and online reconstruction. To make processes and strategies in these functions simple, we assumed that a target object was stored into a storage device. Recently, the demands for stable and high-speed accesses to storage systems have risen according to the increase of high-definition multimedia stream objects. To satisfy the requirement, the object should be striped across multiple storage devices. However, the ordinary striping technique used in RAIDs are not suitable for the multimedia stream objects. In this paper, we consider a workload-aware dynamic allocatable striping strategy to provide stable and high-speed access to a distributed storage system by enhancing our previous research. We also consider a scheduling strategy of using striped replicas to improve system throughput. We then evaluate these strategies using an experimental autonomous disk system we proposed.

Key words　Replica, Parallel Distributed DB, Storage System, Striping

1. はじめに

近年，コンピュータシステムで使用されるデータ量が増大し続けている．このため，HDD に代表されるストレージ装置を多数結合することで大容量でかつ信頼性の高いシステムを安価に構成する方法が取られることが多い．しかし，ストレージ装置数や記憶容量が増えることで，システムが複雑化し，データ管理が煩雑化するため，アクセス QoS の保証，容量割当て，データ冗長性の管理等の管理コストの増大が問題となっている．

この問題に対し，各ストレージ装置毎に計算処理能力を持たせるストレージシステムにおいて，その計算処理能力をデータ管理に利用する試みが行われている [1–4]．我々も，ストレージ装置毎に持つ計算処理能力を用いて負荷均衡化，容量均衡化，耐故障，および柔軟な規模変更の機能を実現する研究を行ってきた [5–9]．

これまで我々は，アクセス単位となるある 1 つのオブジェクトがいずれか 1 つのストレージ装置に格納されることを前提に，オブジェクトをストレージ装置間で移動させることで，負荷均衡化，容量均衡化，プライマリ・バックアップによる耐故障化，規模変更機能の実現を試みてきた．1 つのオブジェクトの格納を 1 つのストレージ装置内に限ることで，負荷，容量の配分の計算や戦略を単純化することができ，耐故障処理や規模変更にも柔軟性を持たせることができた．

一方，近年の高精度のマルチメディアオブジェクトなどの増加で，ストレージ装置中のオブジェクトごとに高いスループットが求められるようになってきている．特に，オブジェクトの種類も多様になっているため，オブジェクト毎に異なるスループットが求められるようになってきている．しかし，オブジェクトを 1 つのストレージ装置内だけで格納していては，そのストレージ装置単体のスループット以上の性能を実現することはできない．

ストレージ装置単体以上の高いスループットを実現するためには，RAID 0，3〜6 などのようなストライピングを行う必要がある [10–12]．しかし，これまでの RAID に代表される分割されたデータの配置が固定されるストライピングでは，大規模構成において，負荷均衡化，容量均衡化や柔軟な規模変更，さらにオブジェクト毎に異なるスループットに対応させるような柔軟な管理を行うことが容易ではい．また，RAID 3〜6 で用いられるようなパリティ計算に基づく耐故障化では，データ回復のために複数ストレージ装置にまたがるパリティ計算を行わなくてはならないため，故障発生時においても安定した転送レートを保ちながらストリーミングを行うという QoS を保証することが困難である．

そこで，本研究では一部のストレージ装置が故障した場合も含めオブジェクト毎に柔軟に高いスループットで安定したデータ供給を行うことを目的として，これまで行ってきたストレージ装置単位のオブジェクトに対する負荷均衡化，容量均衡化，プライマリ・バックアップによる耐故障化，規模変更機能をストレージ装置間にまたがるストライピングに拡張する方法を考える．これは，分割されたデータの配置を動的に変更可能なストライピングに対する，負荷均衡化，容量均衡化，耐故障化，規模変更機能の実現と位置づけられる．

しかし，ストレージ装置間にまたがるストライピングを行う場合には，ストレージ装置単位で負荷や容量を計算するだけでは不十分であり，プライマリ・バックアップによる耐故障化，規模変更もこれまでの方法をそのまま適用することはできない．複数のストレージ装置の負荷や容量が密接に関連して変化することを想定した場合の，負荷均衡化，容量均衡化，耐故障化，規模変更機能を一時に実現することは非常に難しい．

本稿ではまず，アクセス傾向が大きく変化しないという前提を置いた上で，ストライピングのために分割したデータを配置するプリミティブな配置手法の実機実験による性能評価を行う．また，バックアップを耐故障化に用いるだけでなく，ストライピング時のスループット向上に利用する可能性を調べるために，プライマリデータだけでなくバックアップデータからも読み出すよう，コネクション数に注目したプライマリ・バックアップ間の読み出し比率調整を行う手法を提案し，実機実験によって評価する．

2. 関連研究

近年多くのストライピング法が研究されている．例えば，RAID0+1 や RAID1+0，擬似ランダムハッシュ関数を使ったストライピング [13]，スタガードストライピング [14] などが提案されている．

ストライピングおよびミラーリングを使う方法である RAID0+1 や RAID1+0 は高速な読み出しとミラーリングによる冗長化を両立した手法である．しかし，ともに RAID の問題点である分割されたデータの配置が固定であるという問題点がある．そのため，オブジェクト毎に柔軟なスループットに対応させることができず，システム拡張時にも問題が生じる．

擬似ランダムハッシュ関数を用いたストライピングは，擬似ランダムハッシュ関数を再計算することにより分割されたデータの配置を自由に変更することが可能な手法である．しかし，大量の分割されたデータの配置を同時に変更する際の擬似ランダムハッシュ関数の再計算コストが非常に高く，オブジェクト毎の柔軟な管理ができないという問題点がある．

スタガードストライピングは安定した高速読み出しを行うことができ，オブジェクト毎に違うスループットが適用できるストライピング法である．しかし，分割されたデータの自由な配置ができないため，耐故障処理や規模変更時に，全データの構成を変更しなければならないという問題がある．

3. 前提となるシステム

3.1 用語定義

本稿において用いる用語を以下に定義する．

分散ストレージシステム　複数のストレージ装置で構成され，システム全体がネットワークでつながれており，各ストレージが別々に動作することが可能であるストレージシステム

アクセス　クライアントがストレージシステムに対して単一の読み込みあるいは書き込みリクエストを送り，それに対してシ

ステムがファイルの保存場所を返し，その保存場所に対してクライアントがファイルの送信または受信を行う一連の動作をいう．アクセスが行われているとき，その対象となるファイルが置いてあるストレージのアクセス数が 1 であると定義する．

フラグメント　固定長に分割されたデータ，このデータがストライピングの最小単位になる．

サブオブジェクト　同じストレージ上に配置された元が同じオブジェクトであるフラグメントの集合

3.2　自律ディスク

我々は負荷均衡化，容量均衡化，耐故障，柔軟な規模変更などの機能を持つ分散ストレージシステムとして自律ディスク [5, 15] の研究を行ってきた．自律ディスクとは高機能な分散ストレージシステムの 1 つである．自律ディスクにおいては，各オブジェクトは値域分割により配置ストレージが決定される．また，それぞれのストレージがシステム内のいかなるデータへのアクセス要求にも対応するために，各ストレージが分散索引構造 [16, 17] を持っている．このようなシステムで，負荷均衡化，容量均衡化，耐故障，柔軟な規模変更を以下の手順で自律的に行う．

負荷均衡化　システム内の値域が隣のストレージとの負荷を比較し，負荷の多いストレージから少ないストレージへデータを移動させることにより，システム内の各ストレージの負荷を均衡化させる．

容量均衡化　システム内の値域が隣のストレージとの容量を比較し，容量の多いストレージから少ないストレージへデータを移動させることにより，システム内の各ストレージの容量を均衡化させる．

耐故障　スタガード割り当て [18] によるバックアップを持ち，故障発生時に以下の手順で障害復旧を行う．

（1） 故障したストレージ上のプライマリデータに対するバックアップデータをプライマリデータにする．

（2） (1) のプライマリデータと故障したストレージ上のバックアップデータに対するプライマリデータのバックアップを作成する．

（3） 負荷分散および容量分散を行う．

柔軟な規模変更　新しいストレージが追加されたとき，システム中の全ストレージに対し，新しいストレージが追加されたという情報を送信し，負荷分散および容量分散を行う．

自律ディスクは以上の機能をクライアントに対して透過的に行うシステムである．

4.　サブオブジェクト配置法

フラグメント同士の位置関係が固定であるストライピング法では，一部のフラグメントの配置を変更する際に，他のフラグメントの配置も変更する必要がある．そのため，障害復旧や規模拡張を行うときに問題が生じる．フラグメント同士の位置関係が固定でないストライピング法でも，フラグメント単位で配置を決める配置法では，負荷均衡化や容量均衡化処理を行う時にフラグメントの移動先をフラグメントごとに計算しなければならない．そのため，計算コストが高くなるという問題が生じる．また，ストライピングにおいてスループットはオブジェクトの分割数によって決定される．そのため，要求スループットの違うオブジェクトを柔軟に管理するためにはオブジェクトごとに分割数を柔軟に変更できる必要がある．

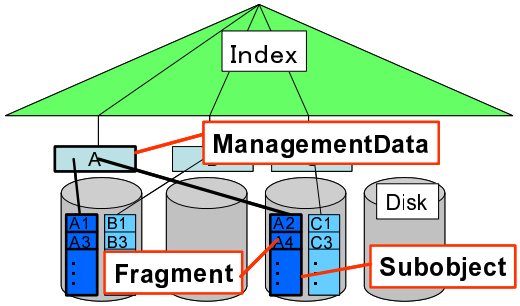

図 1　任意配置の構成

そのため，本稿では自律ディスクに対して負荷均衡化，容量均衡化，耐故障，柔軟な規模変更を実現可能で，要求スループットの異なるオブジェクトを柔軟に管理がすることが可能であるストライピング法を適用する．

4.1　任意配置の構成法

まず最初に要求スループットに応じて分割数を決定する．その後，分割数に応じてフラグメントを配置するストレージを選択する．選択したストレージ上にラウンドロビン形式でフラグメントを配置する (図 1)．ここで同じオブジェクトの同じストレージ上のフラグメントをサブオブジェクトとする．そして，そのサブオブジェクトを配置するストレージの情報を管理するために索引構造中に新しいデータを作成する (この索引構造中のデータを分割情報データと呼ぶ)．

このように配置することにより，サブオブジェクトを配置する場所が限定されていないため障害復旧や規模拡張を問題なく行うことができる．また，オブジェクトの移動を行う際も，移動場所の計算はサブオブジェクトごとに行えばよいため計算コストはフラグメント単位で管理する場合に比べ少ない．さらに，サブオブジェクト数を調整することによりオブジェクトごとに違うスループットを提供することも可能である．

4.2　サブオブジェクトの配置

4.1 のような配置の場合，読み出し速度は最も遅いストレージによって決定されるため，少数のストレージにアクセスが集中する場合，全体として読み出し速度が低下するという問題が生じる [19]．よって，サブオブジェクトを無計画に配置させた場合，アクセスが多いオブジェクトのサブオブジェクトが同じストレージに配置され，一部のストレージの読み出し速度が遅くなる．そのため，少数のストレージにアクセスを集中させないためのサブオブジェクトの配置法が不可欠である．またサブオブジェクトの配置は一度決定してしまうと，移動するためには非常にコストがかかる．そのため，配置をする瞬間だけでなくそれ以降のアクセスも考慮に入れたサブオブジェクト配置法が必要である．

これらの理由により，我々は長期的スパンにおけるアクセス分散を目指し，過去のストレージ上のデータへのアクセス状況

によって配置場所を決定するアルゴリズムを用いる．なお，長期的なスパンでのデータへのアクセス傾向の変化はないと仮定している．長期的に見てデータへのアクセスが多いストレージはある瞬間においてもそのストレージ上のデータへのアクセスが行われている可能性が高い．また，将来的にもストレージへのアクセスが多い可能性が高い．よって，過去にデータへのアクセスの多いストレージにデータを配置すると，そのストレージへのアクセス数が増える．そのため，過去のデータへの読み書き量の多いストレージにサブオブジェクトを配置することは避けるべきである．

以上のことにより，次のような並列ストレージシステムにおいて少数のストレージにアクセスが集中しないようなサブオブジェクト配置場所決定アルゴリズムを用いる．分割情報データを配置したストレージがアルゴリズム中で用いるサブオブジェクト配置可能ストレージのアクセス数情報 (アクセス数リスト) を取得し，サブオブジェクト配置決定法によってサブオブジェクトの配置場所と各サブオブジェクトに含まれるフラグメントの決定を行う．

サブオブジェクト配置決定法

input: サブオブジェクト数：n
フラグメント数：k
サブオブジェクト配置可能ストレージ台数：$M(>n)$
各ストレージのアクセス数リスト：$AD[M]$

output: フラグメント配置ストレージ ID リスト：$fragments[k]$

Step1: $AD[M]$ 中のアクセス数が少ないほうから n 台選び，そのストレージのリスト $Subobjects[n]$ を作成．

Step2: $i \in [0,k]$ に対して $j = i \bmod n$，$fragments[i] = Subobjects[j]$ とする．

Step3: $fragments[k]$ を $output$ して終了

上記のように，過去のアクセスが多いストレージへのサブオブジェクトの配置を避け，アクセスの少ないストレージへ配置させる．このアルゴリズムは，すでにアクセスが多いストレージのアクセスを増やさず，アクセスの少ないストレージのアクセスを増やすため，システム全体でディスク間のアクセス数を均等化し，長期的なスパンでのストレージ間でのアクセスの分散を行うことが可能である．

5. 複製アクセススケジューリング

また，4. のサブオブジェクト配置アルゴリズムに加え，バックアップを読み出しに利用することによって読み出し速度を向上させることが可能である．しかし，ストレージ装置の読み出し速度はそれぞれのストレージへのアクセス数により変わるため，プライマリデータの存在するストレージとバックアップデータの存在するストレージの読み出し速度は同じではない．そのため，読み出しを行う瞬間に速度が違う場合に，読み出し時間を最小化するためにプライマリとバックアップの読み出し量をコネクション数に応じて変更する方法を提案する．

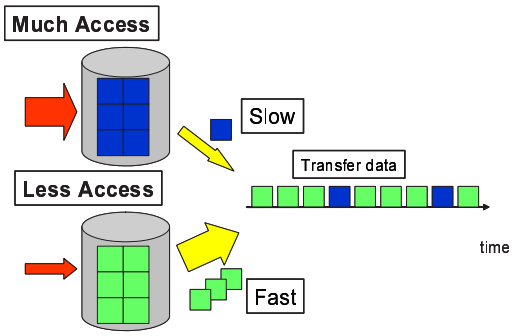

図 2　アクセス数に応じた読み出し量

なお以下ではバックアップは常に最新であることを仮定する．

5.1 読み出し時間の最小化

バックアップは必ずプライマリと別のストレージ上に存在する．そのため，プライマリだけでなくバックアップも読み出しに利用することによって，読み出し時間を減少させることが可能である．しかし，プライマリオブジェクトの置いてあるストレージとバックアップオブジェクトの置いてあるストレージのアクセス数が異なるとき，プライマリオブジェクトとバックアップオブジェクトの読み出し速度は異なる．このとき，読み出し時間を最小化するためには図 2 のように，プライマリ・バックアップ間で読み出し割合を変える必要がある．

まず読み出し時間を最小化する読み出し割合を求める．

- プライマリストレージの読み出し速度 v_p
- バックアップストレージの読み出し速度 v_b
- プライマリオブジェクト読み出し割合 α
- オブジェクトのサイズ S
- プライマリ読み出し時間 T_P
- バックアップ読み出し時間 T_b

このときプライマリ・バックアップのそれぞれの読み出し量と読み出し時間の関係は，

$$\text{プライマリ}：\alpha S = T_p \cdot v_p \quad (1)$$

$$\text{バックアップ}：(1-\alpha)S = T_b \cdot v_b \quad (2)$$

となる．このとき読み出し時間 $T = max\{T_p, T_b\}$ が最小になるのは，プライマリとバックアップの読み出し時間が等しいとき，つまり

$$\alpha = \frac{v_p}{v_p + v_b}$$

のときである．よって，v_p と v_b を取得することができれば読み出し時間を最小化できる．

5.2 コネクション数の比による読み出し量決定

読み出し速度に大きな影響を与える要因として，同時読み出し数が考えられる．そのため，読み出し速度と同時読み出し数の関係を調べるため補助実験を行った．その結果が図 3 である．図 3 より分かるとおり，同時読み出し数と読み出し速度には一定の関係がある．実際図 3 の結果は

$$v = \frac{k}{access\_number} \quad (3)$$

であり，同時読み出し数と読み出し速度が反比例することを示

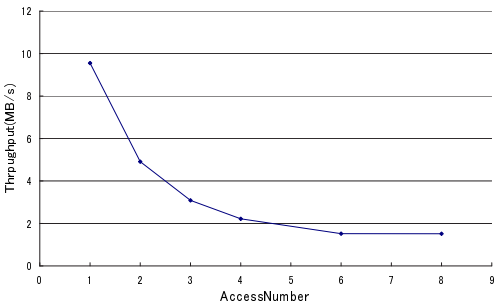

図 3　補助実験：同時読み出し数とアクセス速度

している．なお k は係数である．また，$access\_number = 1$ としたとき $v = k$ となるため，k はそのストレージの読み出しの最大速度である．さらに同時読み出し数はそのときのディスクへのコネクション数に等しい．よって，この関係式を用いて最適な読み出し量を計算することができる．

- プライマリストレージのコネクション数 L_p
- バックアップストレージのコネクション数 L_b

とする．このときの読み出し速度はそれぞれ，

$$\text{プライマリ読み出し速度 } v_p = k/L_p \tag{4}$$

$$\text{バックアップ読み出し速度 } v_b = k/L_b \tag{5}$$

となる．このとき，読み出し時間を最小化する読み出し量は，式 (5.1) に式 (4) と式 (5) を適用させることにより係数 k，つまり最高速度によらずによらず，

$$\text{プライマリ読み出し率 } R_p = \frac{L_b}{L_p + L_b} \tag{6}$$

$$\text{バックアップ読み出し率 } R_b = \frac{L_p}{L_p + L_b} \tag{7}$$

となる．よって，速度がコネクション数に反比例する場合，最高速度によらず式 (6) と式 (7) から読み出し時間を最小にする読み出し割合が求められる．

5.3　過去の読み出し実績による読み出し量決定

式 (3) は特定環境に依存したものであり，実際の式は環境によって変わってくる．そのため，より一般的な速度予測法が必要である．

通常，同じストレージで同じコネクション数であればストレージ故障以外で速度が変化することはない．そのため，以前に読み出しが行われた際のコネクション数とそのときの読み出し速度を履歴として取得しておくことによって将来の読み出し速度予測にも使用できる．

よって過去の履歴からコネクション数と読み出し速度のテーブルを作成し，新しいアクセスがあった瞬間のコネクション数をそのテーブルに照らし合わせることによって読み出し速度の予測ができる．

読み出し開始時　コネクション数・速度テーブルからそのときのコネクション数に対応する速度を取得する

読み出し終了時　そのときのコネクションと読み出しの平均速度をコネクション数・速度テーブルに書き込む

5.4　読み出し量決定法

5.2 または 5.3 を用いて読み出し時間を最小にするプライマリ・バックアップ読み出し量決定法を提案する．なお，以下のアルゴリズム中ではサブオブジェクト配置ストレージの情報が必要なため，分割情報データが配置されているストレージが以下のアルゴリズムを行うとする．

プライマリ・バックアップ読み出し量決定法

input　プライマリストレージのコネクション数 ： L_p
バックアップストレージのコネクション数 ： L_b
サブオブジェクト中のフラグメント数 ： k
プライマリストレージ ID ： I_p
バックアップストレージ ID ： I_b
読み出し量計算関数 ： $F(L_p, L_b)$

output　フラグメント読み出しストレージ ID リスト： $List[k]$

Step1:　プライマリサブオブジェクトの読み出しフラグメント数 $S_p = F(L_p, L_b)$
バックアップサブオブジェクトの読み出しフラグメント数 $S_b = F(L_p, L_b)$ を計算する．

Step2:　pos = 0

Step3:　$i \in [pos, pos + S_p]$ について $List[i] = I_p$ とする．

Step4:　$pos = pos + S_p$ ここで $pos > k$ となったら Step7 へ

Step5:　$i \in [pos, pos + S_b]$ について $List[i] = I_b$ とする．

Step6:　$pos = pos + S_b$ ここで $pos > k$ となったら Step7 へ，$pos \leqq k$ ならば Step3 へ

Step7:　$List$ を $output$ して終了

プライマリ・バックアップ読み出し量決定法により，その瞬間のコネクション数が多い読み出しの遅いストレージからの読み出しを少なくし，コネクション数が少なく読み出しの速いストレージからの読み出しを多くすることによって，全体での読み出し時間を最小化することができる．

6.　実　　験

6.1　実験環境

PC クラスタ上に実装して実験を行った．実験システムの構成は表 1 に示す PC をクライアントとなるストレージとして 4 台，分散ストレージとなるストレージとして 8 台用い，それぞれがお互いにネットワークで接続されている．

実験方法は，以下の通りである．まず各クライアントがそれぞれファイル読み出しまたは書き込みを行った．このとき，最初に初期構成作成のために計 100 回ファイル送信を行い，アクセス数データ取得のために計 200 回ファイル受信を行った後，ファイル送信確率 0.02 で計 2000 回ランダムにファイル送受信を行った．

ファイル送信に用いられる格納ファイルはクライアント上に保存されているファイルからランダムに選ばれる．ファイル受信時の受信ファイル決定は，アクセス量に偏りを持たせるため，

表 1　ストレージノード性能緒元

Number of Client nodes	4 nodes
Number of Disk nodes	8 nodes
CPU	AMD Athlon XP-M 1800+ (1.53GHz)
MEM	PC2100 DDR SDRAM 1GB
Network	1000BASE-T
HDD	TOSHIBA MK3019GAX (30GB，5400rpm，2.5inch)
OS	Linux 2.4.20
Local File System	reiser FS
Java VM	Sun J2SE 1.5.0_03

表 2　実験パラメタ

初期書き込み	100 回
初期読み出し	200 回
合計送受信回数	2000 回
送信率 α	0.02
ピーク数 k	1
Zipf 母数 θ	0.9
Fragment サイズ	8kB
Subobject 数	3

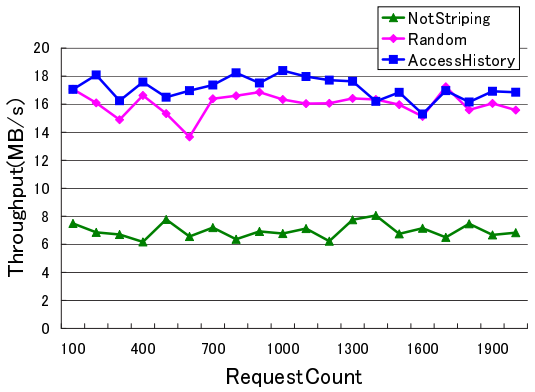

図 4　配置によるスループットの変化

$2k$ 個の Zipf 分布を合成し生成した，ピーク数が k 個のアクセスパターンを用いた．Zipf 分布の偏りはパラメタ θ によって決まる．今回行った実験のパラメタは表 2 の通りである．

6.2　配置法の効果

まずサブオブジェクト配置決定法の効果を検証する．実験内容は以下の通りである．

（1） ストライピングを用いない
（2） ランダムに配置する
（3） アクセスの履歴による配置決定法を用いて配置する

上記の 3 通りの配置法を用いてランダムにリクエストを複数クライアントから送信する．その際の各クライアントによるオブジェクト読み出しの平均スループットの時間変化を測定した．その結果は表 3 および図 4 の通りである．なお図 4 は全クライアント合わせて 100 回ごとの平均のスループットの推移を表したグラフである．

6.2.1　考　　察

まず，表 3 より，ストライピングを用いなかった場合とストライピングを用いた 2 手法を比較すると，ストライピングを用いることによってスループットが 2 倍以上に向上している．また，図 4 から分かるとおり，常に 2 倍以上のスループットを達成している．これは従来のストライピングを行わない方法では 4 台のクライアントの読み出しに対して同時に最大 4 台のストレージしか用いられなかったが，ストライピングを行うことにより同時に最大 8 台のストレージが用いられるようになったためである．

表 3　サブオブジェクト配置決定法の効果

	Average Throughput (MB/s)	Standard Deviation
ストライピングを用いない	6.96	4.10
ランダムに配置	16.01	5.38
アクセスの履歴による配置決定法	17.12	6.35

また，表 3 および図 4 から分かるとおり，サブオブジェクトをランダムに配置した手法とアクセスの履歴を用いて配置を決定した手法を比較すると，アクセスの履歴を用いた手法のほうがスループットが上昇している．この理由としては，ランダムに配置した場合はある一部のストレージへのアクセスが多くなり，そのストレージの読み出し性能が減少し，さらにそのストレージにアクセスされることが多いため，結果として平均の読み出し性能が遅くなる．それに対して，アクセス履歴を用いた手法ではストレージ間のアクセス数が均等化されるため，ストレージの読み出し速度の減少量が平均化され，またシステム全体のストレージから均等に読み出しが行われるため，平均の読み出し性能が遅くならないためである．

しかし，表 3 の標準偏差から分かるとおり読み出し速度のばらつきは増大している．これは，アクセス履歴を用いた手法ではオブジェクト毎のスループットが，高スループットと低スループットに二極化されてしまうためである．そのため，スループットの安定化と更なる高速化のためには，スループットが二極化されてしまった原因を特定し，低スループットとなる読み出しを減らす必要がある．

6.3　プライマリ・バックアップスケジューリングの効果

プライマリ・バックアップスケジューリングの効果を確かめるための実験を行う．実験には，以下の 2 種類のプライマリ-バックアップ読み出し法を使用し，フラグメントの配置法はサブオブジェクト配置決定法を用いる．

（1） 全てプライマリから読み出す
（2） プライマリ・バックアップの読み出し割合 1:1
（3） コネクション数の比による読み出し量決定法
（4） 過去の処理実績による読み出し量決定法

なお，この実験も配置法の実験と同様にリクエストの送信を行い，各クライアントによるオブジェクト読み出しの平均スループットの時間変化を測定した．その結果は表 4 および図 5 の通りである．なお図 5 も配置法の実験と同様に全クライアント合わせて 100 回ごとの平均のスループットの推移を表したグラフである．

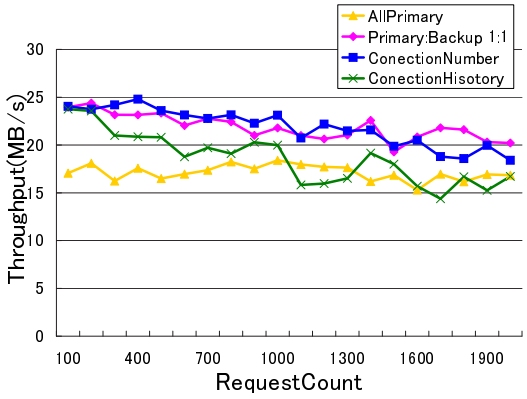

図 5　スケジューリングによるスループットの変化

表 4　フラグメント読み出し量決定アルゴリズムの効果

	Average Throughput (MB/s)	Standard Deviation
全てプライマリ	17.12	6.35
読み出し割合 1:1	21.87	8.02
コネクション数の比	21.86	7.57
過去の処理実績	18.61	7.43

6.3.1　考　　察

表 4 のように，バックアップからの読み出しを行うことによってスループットが向上した．これはバックアップへもアクセスすることにより，システム全体として各クライアントへのアクセス数が均等化されたためである．

また，図 5 および表 4 から分かるとおり，読み出し割合をプライマリ・バックアップ間で常に 1:1 にした方法と，コネクション数の比を用いて決定した方法を比較すると，平均スループットは同じである．これはコネクション数の比を用いて予測する方法を用いたとき，読み出し途中でのコネクション数の変化が生じたり，システムプロセスなどが動作し読み出し性能へ影響を与えたりするが，これらの読み出し性能への影響が考慮できていないためである．しかし，偏りはコネクション数の比を用いたほうが小さくなって読み出し速度が安定している．これはコネクション数の比を用いたことにより，高スループットの読み出しも減ったが，低スループットの読み出しも減ったため，平均値付近のスループットが多くなったためであり，速度が安定したことが分かる．

しかし，過去の処理実績を用いた方法では，他のバックアップを読み出しに用いた 2 手法に比べてスループットが向上していない．これは今回の実験では履歴として過去全てのアクセスを用いたため，時間が経つにつれ過去の履歴が多くなり，現在の性能が反映しにくくなるためである．しかし，図 5 から分かるとおり，リクエスト回数 200 回程度から減少が始まっている．そのため，履歴の保持期間の設定をすることによって，古い履歴を破棄し現在の性能を反映しやすくすることによってスループット減少を抑えられると考えられる．

7.　議　　論

プライマリとバックアップの存在するストレージの片方だけ

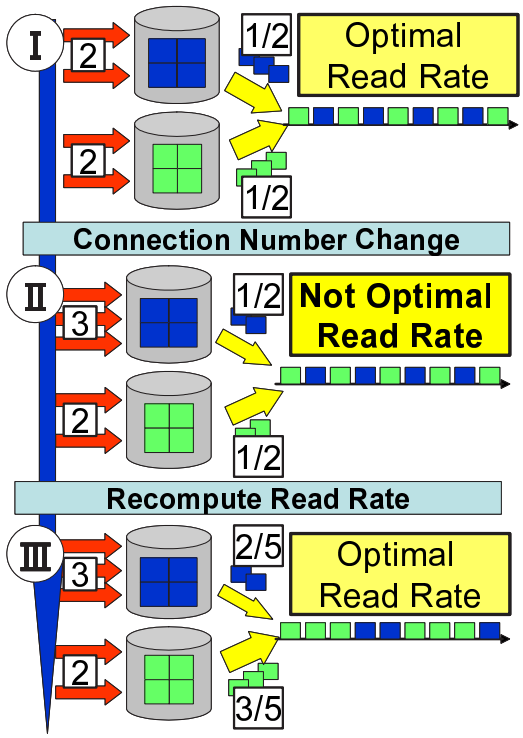

図 6　読み出し量再計算

コネクション数が変化した場合，コネクション数の変化が起こったストレージの読み出し速度が変化するため，式 (5.1) による最適な読み出し割合が変わってしまう．そのため，コネクション数が変化した場合にも読み出し時間を最小化するためには，コネクション数が変わった瞬間にプライマリ・バックアップのフラグメントの読み出し割合を再計算する必要がある．

例えば図 6 において，初めのプライマリストレージとバックアップストレージへのコネクションの数がそれぞれ 2 である場合 (I)，それぞれのストレージの読み出し割合は 1/2 (1 : 1) となる．ここで，バックアップストレージのコネクションの数が増え 3 になったとする (II)．この時，式 (5.1) による最適な読み出し割合はプライマリ 3/5，バックアップ 2/5 (3 : 2) となる．そのため，1/2 (1 : 1) のままでは最適ではないため，読み出し割合を最適な値に直すことで読み出し時間を短くすることができる (III)．

以上のこと次のように読み出し率の再計算を行うことによって読み出し時間を最小にすることができる．

（1） 一定時間ごとにコネクション数の変化がないかシステムに問い合わせを行う．

（2） コネクション数の変化があったときだけ，プライマリ・バックアップ読み出し量を再計算する．

以上のように，定期的に読み出し率の再計算を行うことによって，読み出しを開始した後にコネクション数の変化があった場合でも，読み出し率を最適化でき，読み出し時間を短くすることができる．

8.　まとめと今後の課題

本研究では，自律ディスク上でストライピングを行うために

分割したデータを配置するプリミティブな配置手法を提案し，実機実験による性能評価を行った．また，バックアップを耐故障化に用いるだけでなく，ストライピング時のスループット向上に利用する可能性を調べるために，プライマリのアクセス負荷が高い場合はバックアップデータから読み出すよう，アクセス数に注目したプライマリ・バックアップ間の読み出し比率調整を行う手法を提案し，実機実験によって評価した．

今後の課題についてはまず，異なるパラメタでの環境に対する各アルゴリズムの挙動について実験評価を行う必要があることである．また今回はデータへの読み書きリクエストの傾向が変化する場合を考慮していない．データへの読み書きリクエストの傾向が変化する場合に，ストレージ間のアクセスを均等化するにはデータマイグレーションが適する．よって本稿で提案した手法とデータマイグレーションを両方用いた場合のコストと性能の評価なども必要であると考える．

また，現実的なシステムではストレージノード内部で並行動作しているシステムプロセス等があるため，これらのディスクI/O 性能への影響調査と，それを考慮したスケジューリング手法の改善を行うことが必要である．

また，近年のストレージノード上には大容量の半導体記憶によるキャッシュ機構が期待できるため，高速化のためにそれらのディスクキャッシュも考慮したスケジューリングを行うことを検討している．

これらをふまえ，動的配置が可能なストライピングにおける負荷均衡化，容量均衡化，耐故障化，規模変更機能の実現を目指す．

謝　　辞

本研究の一部は，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] Gregory R. Ganger, John D. Strunk, and Andrew J. Klosterman. Self-* storage: Brick-based storage with automated administration. Technical Report CMU-CS-03-178, Carnegie Mellon University, Aug 2003.

[2] Svend Frølund, Arif Merchant, Yasushi Saito, Susan Spence, and Alistair Veitch. FAB: enterprise storage systems on a shoestring. In *HOTOS 2003*, Kauai, HI, May 2003.

[3] John MacCormick, Nick Murphy, Marc Najork, Chandramohan A. Thekkath, and Lidong Zhou. Boxwood: Abstractions as the foundation for storage infrastructure. In *OSDI*, pp. 105–120, 2004.

[4] 武理一郎, 野口康生, 土屋芳浩, 荻原一隆, 田村雅寿, 丸山哲太郎, オーガニックストレージシステム, 信学技報 CPSY2004-57 pp.55-60, 電子情報通信学会, 2004.

[5] Haruo Yokota. Autonomous disks for advanced database applications. In *Proc. of International Symposium on Database Applications in Non-Traditional Environments (DANTE'99)*, pp. 441–448, Nov. 1999.

[6] D. Ito and H. Yokota. Automatic reconfiguration of an autonomous disk cluster. In *Proc. of 2001 Pacific Rim International Symposium on Dependable Computing (PRDC 2001)*, pp. 169–172. IEEE, Dec. 2001.

[7] 渡邊明嗣 , 横田治夫. 分散ディレクトリ探索コストを考慮した並列データアクセス偏り制御. 電子情報通信学会論文誌, Vol. J85-D-I, No. 9, pp. 831–840, Sep. 2002.

[8] Akitsugu Watanabe and Haruo Yokota. Adaptive lapped declustering: A highly available data-placement method balancing access load and space utilization. In *ICDE*, pp. 828–839. IEEE Computer Society, 2005.

[9] Mana Nakano, Dai Kobayashi, Akitsugu Watanabe, Toshihiro Uehara, Ryo Taguchi, and Haruo Yokota. The versioning system balancing data amount and access frequency on distributed storage system. In *ICDE Workshops*, p. 1264, 2005.

[10] David A. Patterson, Garth A. Gibson, and Randy H. Katz. A case for redundant arrays of inexpensive disks (raid). In Haran Boral and Per-Åke Larson, editors, *SIGMOD Conference*, pp. 109–116. ACM Press, 1988.

[11] Peter M. Chen, Edward L. Lee, Garth A. Gibson, Randy H. Katz, and David A. Patterson. Raid: High-performance, reliable secondary storage. *ACM Comput. Surv.*, Vol. 26, No. 2, pp. 145–185, 1994.

[12] Kenneth Salem and Hector Garcia-Molina. Disk striping. In *Proceedings of the Second International Conference on Data Engineering*, pp. 336–342, Washington, DC, USA, 1986. IEEE Computer Society.

[13] Andre Brinkmann, Kay Salzwedel, and Christian Scheideler. Efficient, distributed data placement strategies for storage area networks. In *ACM Symposium on Parallel Algorithms and Architectures*, pp. 119–128, 2000.

[14] Steven Berson, Shahram Ghandeharizadeh, Richard Muntz, and Xiangyu Ju. Staggered striping in multimedia information systems. In *SIGMOD '94: Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, pp. 79–90, New York, NY, USA, 1994. ACM Press.

[15] 伊藤大輔 , 風戸広史 , 横田治夫. 負荷分散機構と組み合わせた自律ディスクのクラスタ再構築. 信学技報, FTS2001-20, pp. 119–126. 電子情報通信学会, Jul. 2001.

[16] Haruo 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*, pp. 448–457, 1999.

[17] Hiroshi Kazato and Haruo Yokota. Implementation of a fat-btree in an autonomous-disk cluster. In *IPSJ Technical Report, DBS-125-71*, pp. 45–52. Information Processing Society of Japan, Jul. 2001.

[18] J. Gray and A. Reuter. *Transaction Proceeding: Concepts and Techniques*. Morgan Kaufmann, San Francisco, 1992.

[19] H. Simitci. *Storage Network Performance Analysis*. Wiley Technology Publishing, 2003.