自律ディスククラスタの階層化構成における性能改善と評価

花井　知広[†]　渡邊　明嗣[†]　小林　大[†]　山口　宗慶[†]　田口　亮[††]
林　直人[††]　上原　年博[††]　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
†† NHK 放送技術研究所
††† 東京工業大学 学術国際情報センター
E-mail: †{hanai,aki,daik,muu}@de.cs.titech.ac.jp, ††{taguchi.r-cs,hayashi.n-gm,uehara.t-jy}@nhk.or.jp,
†††yokota@cs.titech.ac.jp

あらまし　我々はストレージシステムにおいて負荷分散，故障対策，障害回復などの高度な機能を実現するためのアプローチとして，自律ディスクを提案している．しかし，近年のコストや性能に対する様々な要求に応えるためには，多種のデバイスを組み合わせることが有効である．これらの要求に応えるために我々は，半導体メモリと磁気ディスクの各デバイスから構成されるクラスタを組み合わせた，自律ディスククラスタの階層的構成手法を提案している．本稿では自律ディスククラスタの階層的構成手法において，ディレクトリ探索と通信回数の削減による高速化手法を提案するとともに，試作システムにより様々な条件での性能評価を行い，我々の提案する構成手法の有効性を示す．
キーワード　自律ディスク, 階層化ストレージ, 半導体ディスク, 性能評価

Performance Improvement and Evaluation of A Hierarchical Autonomous Storage System

Tomohiro HANAI[†], Akitsugu WATANABE[†], Dai KOBAYASHI[†], Munenori YAMAGUCHI [†],
Ryo TAGUCHI[††], Naoto HAYASHI[††], Toshihiro UEHARA[††], and Haruo YOKOTA[†,†††]

† Department of Computer Science, Graduate School of Information Science and Engineering,
Tokyo Institute of Technology
†† NHK Science & Technical Research Laboratories
††† Global Scientific Information & Computing Center, Tokyo Institute of Technology
E-mail: †{hanai,aki,daik,muu}@de.cs.titech.ac.jp, ††{taguchi.r-cs,hayashi.n-gm,uehara.t-jy}@nhk.or.jp,
†††yokota@cs.titech.ac.jp

Abstract　We have proposed autonomous disks to import advanced functionality into storage systems, which balances loads, tolerates faults, and recovers data within a cluster. However, since the higher performance is strongly required for the storage systems, a combination of various devices is significant. To solve these problems, we have proposed hierarchical architecture of autonomous storage constructed from a magnetic autonomous disk cluster and a solid state autonomous disk cluster. In this paper, we propose a performance improvement method that eliminates directory traversing and message transferring, and evaluate it. We also evaluate our experimental system with various conditions to demonstrate the effectiveness of our approach.
Key words　autonomous disks, hierarchical storage, solid state disks, performance evaluation

1. はじめに

大規模データ処理システムにおけるストレージ管理コストの増加に伴い，ストレージをシステムの中心に据えるストレージセントリックシステムが注目されている．我々はストレージシステムにおける負荷分散，故障対策，障害回復などの高度な機能を実現するためのアプローチとして，ディスク装置内の制御用プロセッサとキャッシュ用メモリを利用する自律ディスク [1]

を提案してきた．しかし，ストレージシステムで扱われるデータ量は未だ増加の一途をたどり，その性能に対する要求もより高いものが求められている．例えば典型的なトランザクションシステムの処理性能向上のために，より高速なレスポンスがストレージに対して求められている．しかし，現在のストレージシステムの基礎となる記憶媒体は磁気ディスクであり，その構造から飛躍的な速度向上は望めない状況である．

これらの問題に対して我々は，性能の異なるデバイスを組み合わせて階層的な自律ストレージを構成する手法を提案し，性能評価を行ってきた [2], [3].

本稿ではこの自律ディスククラスタの階層的構成手法において，ディレクトリ探索とメッセージ通信の回数削減による高速化手法を提案する．さらに模擬実装を用いた実験により性能評価を行い，提案手法の有効性を示す．

以下に本稿の構成を述べる．まず 2. 章で自律ディスクとその階層化構成の概要について述べる．次に 3. 章ではメッセージ通信回数の削減による高速化手法について述べる．4. 章では提案手法の実験による評価について述べる．最後に 5. 章でまとめと今後の課題を述べる．

2. 自律ディスクとその階層化構成の概要

2.1 自律ディスク

はじめに自律ディスクの概要について説明する．自律ディスクはネットワーク環境でディスククラスタを構成することを前提としている．クライアントはデータにアクセスするためにクラスタ内の任意のノードに要求を発行し，クラスタ内のノードはノード間の局所的な通信を行うことで，協力してクライアントからの要求に対処する．標準的な構成ではクラスタ内の各ノードは，プライマリディレクトリと他ノードのバックアップを行うバックアップディレクトリを持つ．データに対するアクセスは分散ディレクトリをトラバースし，リクエストを適切なノードに転送することにより行われる．このような前提のもとで，自律ディスクはデータ分散，ホストからの均質的なアクセス，同時実行制御，偏り制御，耐故障性，異種性といった性質を持つ．自律ディスクの詳細については文献 [1] を参照されたい．

2.2 自律ディスクの階層化構成

つぎに我々の提案する自律ディスクの階層化構成手法の概要を示す．我々の提案する手法では，磁気ディスクのみを用いて構成された自律ディスククラスタ (D クラスタ) と，半導体ディスクのみを用いて構成された自律ディスククラスタ (M クラスタ) をそれぞれ構成し，M クラスタが D クラスタのキャッシュとして動作する．このアーキテクチャの概要を図 1 に示す．

我々はこのアーキテクチャにおいて Write-Through-Sync(WTS) プロトコル，Write-Through-Async(WTA) プロトコル，Delayed-Write(DW) プロトコルという 3 種類の挿入・更新プロトコルと読み出しプロトコルを提案している．3 種類の挿入・更新プロトコルはいずれも耐故障性を持つように設計されており，主な違いはクライアントのリクエストに対して同期的に書き込みが行われる箇所である．以下に各プロトコルの概要を述べる．詳細は [2], [3] を参照されたい．

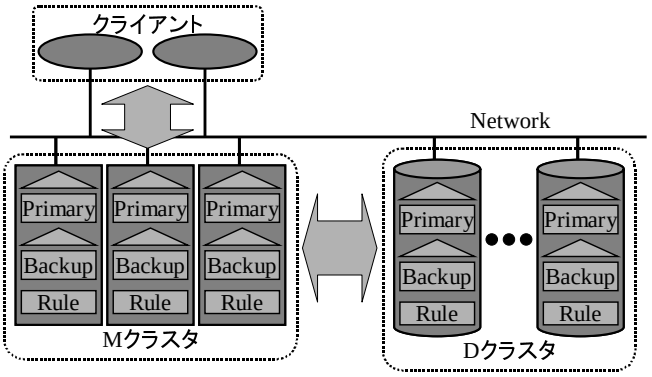

図 1　階層化アーキテクチャ

WTS プロトコル　M クラスタのプライマリと D クラスタのプライマリとバックアップへ同期的に書き込みが行われる．

WTA プロトコル　D クラスタ内のバックアップ生成を非同期的に行うことにより WTS を高速化したもの．

DW プロトコル　M クラスタ内でのみ書き込みを行い，クライアントのリクエストとは非同期に D クラスタへ書き込みを行うため高速な書き込みが行える．

読み出しプロトコル　M クラスタ内でリクエストされたデータを探索し，存在すればそれをクライアントに返す．存在しない場合は D クラスタからリクエストされたデータを読み出す．

3. ディレクトリ探索とメッセージ転送回数の削減

自律ディスクではクラスタ中のいかなるノードに格納されているデータに対するリクエストも，任意のノードが受け付けることができるとしている．もしあるノードが他のノードに格納されているデータに対するリクエストを受け付けた場合は，そのリクエストは分散ディレクトリを探索することによってデータを格納しているノードに自動的に転送される．リクエストを任意のノードが受け付けることができるため，これまで研究においてはクライアントがリクエストを送信する際にクラスタ中の各ノードをランダムに選ぶ条件の元で行われてきた．自律ディスククラスタの階層的構成においては，M クラスタは D クラスタのクライアントとして動作する．この構成において M クラスタが D クラスタへリクエストを送信する際に，D クラスタ内のデータを格納しているノードに直接にリクエストを送信することができれば，ランダムにノードを送信する場合に比べて D クラスタ内でのディレクトリ探索処理やリクエストの転送処理を削減することができる．関連研究として [4] で提案されている $RP*_C$ や $RP*_S$ があり，これらはクライアントが各ノードの値域分割割り当てを記録してリクエスト送信時の送信先決定に利用するが，値域分割したデータの配置方法については述べていない．本稿で提案する手法はこれらの手法に比べ，D クラスタの変更が必要なく，推測情報の管理が単純であるという特徴を持つ．

この章ではまず階層化手法についてディレクトリ探索回数とメッセージ転送回数の見積もりを行い，高速化を適用する箇所を示す．次に，本稿で提案する高速化手法について説明する．

表 2　メッセージ通信回数の見積もり

	WTS	WTA	DW	読み出し
クライアント→ M クラスタ	1	1	1	1
M クラスタ内	$(M-1)/M$	$(M-1)/M$	$(M-1)/M+2$	$(M-1)/M$
M クラスタ→ D クラスタ	1	1	—	$1-r$
D クラスタ内	$(N-1)/N+2$	$(N-1)/N$	—	$(1-r)((N-1)/N)$
D クラスタ→ M クラスタ	1	1	—	$1-r$
M クラスタ→クライアント	1	1	1	1
計	$(M-1)/M+(N-1)/N+6$	$(M-1)/M+(N-1)/N+4$	$(M-1)/M+2$	$(M-1)/M+(1-r)(N-1)/N+(4-2r)$

表 1　ディレクトリ探索回数の見積もり

	WTS	WTA	DW	読み出し
M クラスタ	$\frac{M-1}{M}+1$	$\frac{M-1}{M}+1$	$\frac{M-1}{M}+2$	$\frac{M-1}{M}+1$
D クラスタ	$\frac{N-1}{N}+2$	$\frac{N-1}{N}+1$	—	$(1-r)(\frac{N-1}{N}+1)$

3.1　ディレクトリ探索とメッセージ転送回数の見積もり

はじめに，各プロトコルにおいてリクエスト処理に必要な分散ディレクトリ探索回数とメッセージ転送回数を見積もったものをそれぞれ表 1 に示す．M クラスタのノード数を M，D クラスタのノード数を N，読み出しの場合は M クラスタでのキャッシュヒット率を r としている．また，リクエスト処理とは非同期に行われる処理は含まれていない．表の各欄中に下線部で示した部分は，リクエストを受信したノードがリクエストを処理できない場合に，リクエストを実際に処理するノードへ転送するための処理によるものである．

3.2　提 案 手 法

この節では，M クラスタから D クラスタへのリクエスト送信時に送信先を予測することにより，前節で示した表 1,2 中の下線部分の処理を削減し，D クラスタの負荷を低減することによる高速化手法を提案する．

自律ディスクではデータの配置管理を行う分散ディレクトリとして，Fat-Btree [5] などの分散 Btree の利用が想定されている．この場合，データ空間は値域分割され自律ディスクの各ノードに値域が割り当てられる．また，自律ディスクアーキテクチャでは挿入・更新や読み出しが行われたノードがクライアントにレスポンスを返すため，クライアントはどのノードで実際の処理が行われたかを知ることができる．そこで，M クラスタが D クラスタからレスポンスを受信した場合に，どのキー (ストリーム ID) に対して D クラスタのどのノードで処理が行われたかを記録しておく．そして，M クラスタが D クラスタへリクエストを送信する場合には，その記録を元にリクエストが処理されるノードを推測することにより，D クラスタ内でのメッセージ転送回数を削減することができる．また，D クラスタ内でのディレクトリ探索処理が効率的に行われるため，D クラスタの負荷を低減することができる．

実際にはリクエストの処理結果を全て記録する必要はなく，D クラスタの各ノードに対して割り当てられている値域を記録できればよい．この手法の概略を図 2 に示す．過去に処理結果が得られていないため推測が不可能な場合（図 2 で推測情報が白い部分）は，その左右のノードどちらかを等確率で選ぶものとする．また，D クラスタ内での偏り制御により D クラスタの値域分割の割り当ては変化するが，偏り制御による値域分割割り当ての変化は時間的に緩やかであり，記録された推測情報との差は推測情報の更新により修正されてゆくので，大量の予測ミスを引き起こすことはない．予測ミスの際にも自律ディスクの通常動作として D クラスタ内で適切にリクエストが転送されるので，提案手法を用いない場合に比べ処理の増大を引き起こすことはない．

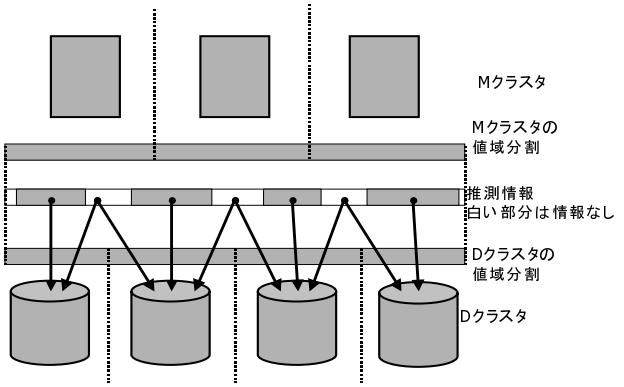

図 2　リクエスト送信先の予測

この手法により，十分な処理結果が記録された後には，非常に高い精度で M クラスタから処理を行う D クラスタのノードへリクエスト送ることができ，D クラスタ内でのディレクトリ探索回数やそれに伴うリクエスト転送の回数を抑えることができる．これにより D クラスタ全体の負荷を低減させることができ，クライアントのリクエストの処理時間を短縮することが可能である．

このようにリクエストの処理結果から推測情報を構築することにより，D クラスタが能動的に値域分割の情報を配布する必要がなくなる．D クラスタ内の値域分割は偏り制御のために頻繁に変更されるので，変更の度に M クラスタへ値域分割の情報を配布すると通信量や処理負荷が大きくなる可能性がある．また能動的に値域分割の情報を配布することにより各クラスタの独立性が減少することになり，階層化構成における柔軟性が減少する．さらに D クラスタのノード数が多い場合には値域分割の情報が大きくなる可能性があるが，提案手法では M クラスタの各ノードが独立した推測情報を持ち，M クラスタの各ノードに割り当てられた値域に関する推測情報のみを保持するため，推測情報の量を D クラスタ全体の値域分割情報の量に比べて非常に少なく抑えることができる．

3.3 推測情報管理アルゴリズム

推測情報は以下のアルゴリズムで管理する．まず記法を以下のように定義する．

- M クラスタのノード数を M，D クラスタのノード数を N とする
- M クラスタの各ノードには 0 から $M-1$ まで，D クラスタの各ノードには 0 から $N-1$ までのノード ID が割り当てられているものとする．
- D クラスタでの値域分割において，ノード i の値域 $R_{Di} = [K_i, K_{i+1})$ とする．ここで K_i は値域分割における境界のキーである．

M クラスタのノード m が持つ推測情報は範囲のリスト $G_m = (r_i, r_j, r_k, \ldots)$ である．$r_i = [k_{iL}, k_{iR}]$ であり，範囲（値域）をあらわす．$k_{iL} \leqq k_{iR}$ とする．G_m 中の各 r_i は範囲が重ならず，小さい順にソートされているものとする．i, j, k は D クラスタのノード ID である．

D クラスタへキー k に関するリクエストを送信するために送信先を推測する場合は，G_m から k を含む r_i を探索し，存在した場合は D クラスタのノード i に対してリクエストを送信する．存在しなかった場合は k の左側の範囲 r_i と右側の範囲 r_j を探索し，ノード i または j を等確率で選んでリクエストを送信する．送信したリクエストが D クラスタで処理され，レスポンスがノード d から返ってきた場合は，以下のように推測情報を更新する．

（1） 初期状態では $G_m = ()$ である．

（2） G_m から r_d を探索する．r_d が存在する場合は（3）へ．存在しない場合は（4）へ．

（3） $k < k_{dL}$ ならば $r_d = [k, k_{dR}]$ として（5）へ．$k > k_{dR}$ ならば $r_d = [k_{dR}, k]$ として（6）へ．その他の場合は終了．

（4） $r_d = [k, k]$ として G_m に追加して終了．

（5） r_d の右の r を r_R とする．$k_{RR} < k$ ならば r_R を削除して終了．$k_{RL} < k$ ならば $r_R = (k, k_{RR}]$ として終了．その他の場合は何もせず終了．

（6） r_d の左の r を r_L とする．$k_{LL} > k$ ならば r_L を削除して終了．$k_{LR} > k$ ならば $r_L = [k_{LL}, k)$ として終了．その他の場合は何もせず終了．

4. 性能評価

次に前章で述べた高速化手法を実験により性能測定を行い評価する．まず本稿で提案する高速化手法を，我々が現在 PC 上に Java を用いて実装を行っている試作システム上に実装した．またこれまでの評価実験は数 KB 程度の比較的小さなサイズのデータを用いて行ってきたが，ここでは 1MB と比較的大きなサイズのデータを用いての評価実験も行う．これまでの評価実験に用いた実装は Java のオブジェクトシリアライズ機構を利用したものであるため，大きなサイズのデータの扱いには適さない．そこで大きなサイズのデータを用いた性能測定では，自律ディスクの HTTP インターフェースを用いる．自律ディスクの HTTP インターフェースについては [6], [7] を参照されたい．

表 4　M, D クラスタの性能測定（データサイズ 1KB）

	Insert		Retrieve	
	M クラスタ	D クラスタ	M クラスタ	D クラスタ
合計所要時間	133785 [ms]	1602118 [ms]	64411 [ms]	329942 [ms]
1 リクエスト処理時間	4.46 [ms]	53.4 [ms]	2.15 [ms]	11.0 [ms]
処理時間比	1 : 11.97		1 : 5.12	

4.1 小さいサイズのデータを用いた測定

まずこれまでの評価実験と同様に小さいサイズのデータを用いて性能測定を行う．

4.1.1 実験方法

性能測定に用いた実験環境を表 3 に示す．M クラスタの記憶媒体としては，Linux の ramfs ファイルシステムにより M クラスタノード上の主記憶を用いた．M クラスタにおける最大キャッシュエントリ数は 1 ノードあたり 600 とした．

また，D クラスタについては OS のバッファキャッシュの影響を小さくし，実際にディスクアクセスが発生するように，主記憶の容量を 32MB に制限するとともに，模擬実装上でのページサイズを M クラスタの 4KB に対して 16KB と大きく設定した．実装の簡便化のために自律ディスクの分散ディレクトリとして aB$^+$-tree [8] を用いている．

この環境に対して，以下の手法で性能測定を行う．挿入・更新操作，読み出し操作をそれぞれ一定回数発行し，処理完了までの所要時間を計測する．リクエストは 6000 個のキー集合を用い，連続して 30000 回行う．格納されるデータのキーは 16 バイトのランダムな文字列であり，データサイズは 1K バイトとする．値域分割は値域の範囲が均等になるように行い，性能測定中に偏り制御は行わないものとする．本稿で提案する推測手法を使用した場合，推測手法を使用せずに D クラスタのノードをランダムに選択した場合，階層化を行わずに D クラスタのみで構成した場合に対して測定を行う．推測手法を用いた場合の推測情報は，空の状態から測定を始める．

4.1.2 予備実験

各操作の性能測定を行う前に，M クラスタと D クラスタのそれぞれの性能を測定する．それぞれのクラスタで独立に従来の自律ディスククラスタを構成し，偏りのない状態で Insert, Retrieve 操作の性能を計測した結果が表 4 である．この結果より，この実験システムにおいて M クラスタは D クラスタに比べ，Insert で約 12 倍, Retrieve で 5.1 倍程度高速であることがわかる．Insert と Retrieve で性能比が異なる理由としては，Insert 処理では実際に書き込みを行うために記録媒体の性能差が大きく現れるのに対し，Retrieve 処理の場合は D クラスタにおいてもディスクキャッシュが利用されることにより記録媒体の性能差が現れにくいためであると考えられる．また Insert 処理ではプライマリデータ領域とバックアップデータ領域の両方にアクセスするのに対して，Retrieve 処理ではプライマリデータ領域にのみアクセスすればよいので D クラスタにおけるディスクキャッシュの利用率が向上するという理由もある．

表 3　実験システムの構成

	D クラスタ	M クラスタ	クライアント
ノード数	6 台	3 台	1 台
CPU	Intel Pentium 3 933MHz	Intel Xeon 2.4GHz, HT	Intel Pentium4 2.0GHz
メモリ	PC133 SDRAM 小サイズ時: 32MB 大サイズ時: 64MB	PC2100 DDR SDRAM 1024MB	PC2100 DDR SDRAM 1024MB
ディスク	Seagate Barracuda IV 20.4GB 7200rpm	小サイズ時: Linux ramfs 大サイズ時: BiTMICRO E-Disk 4GB （FC-HBA: QLogic QLA2310）	Seagate Barracuda IV 20.4GB 7200rpm
OS	Linux 2.2.17, glibc 2.1.3	Linux 2.4.20 SMP, glibc 2.2.5	Linux 2.4.18, glibc 2.2.5
Apache	Apache 1.3.29		—
Java 環境	Sun JDK 1.4.1		
Network	1000BASE-SX, Switching Hub		

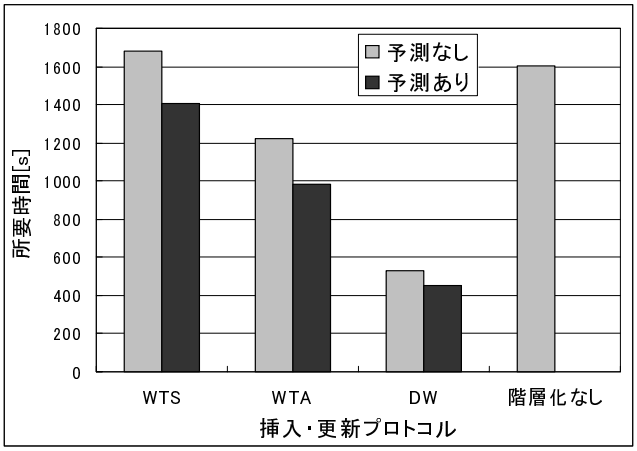

図 3　挿入・更新操作の実行時間（データサイズ 1KB）

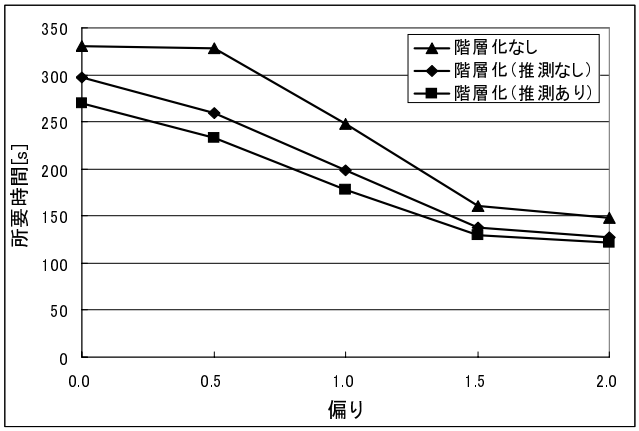

図 4　読み出し操作の実行時間（データサイズ 1KB）

表 5　偏りに対するキャッシュヒット率

偏りの度合い θ	0.0	0.5	1.0	1.5	2.0
キャッシュヒット率 r	24.7%	34.6%	63.5%	77.4%	74.7%

4.1.3　挿入・更新性能

前述の手法によって挿入・更新操作の性能を測定した結果を図 3 に示す.

各プロトコルともに提案手法により処理時間が短縮されており，本稿で提案する推測手法が有効であることがわかる．提案手法により，WTS プロトコルでは 16%，WTA プロトコルでは 19%，DW プロトコルでは 15%の時間短縮になっている．特に WTS プロトコルでは提案手法の適用により階層化なしの場合に比べて高速になっている．また DW プロトコルでは D クラスタへの挿入・更新処理がリクエストと非同期に行われるため提案手法の効果は小さいはずであるが，提案手法により高速化されている．これは，今回の実験環境では M クラスタと D クラスタの性能差が大きいため，M クラスタであふれたキャッシュの追い出し処理による D クラスタへの挿入・更新処理がリクエストの処理時間に影響を与えているためである.

また，階層化構成の各挿入・更新プロトコルと階層化しない場合を比較した場合，WTS プロトコルでは 12%，WTA プロトコルでは 38%，DW プロトコルでは 72%の時間短縮になっている．これにより階層化構成の全ての挿入・更新プロトコルで階層化しない場合に比べて高速になり，階層化構成の有効性が確認された.

4.1.4　読み出し性能

読み出し操作の性能測定では，リクエストされるキーの偏りによる性能の変化を評価するために，各キーの利用頻度を Zipf 分布とし，その偏りの度合いを表すパラメータ θ を 0.0〜2.0 に変化させて測定を行った．このパラメータは $\theta = 0$ の場合には偏りがなく，θ が大きくなるに従って偏りも大きくなるものである.

読み出し操作の性能を測定した結果を図 4 に示す．また，この測定において M クラスタでのキャッシュヒット率 r を測定した結果を表 5 に示す.

読み出し処理についても提案手法により処理時間が短縮されていることがわかる．偏りの度合い $\theta = 1.0$ の場合で提案手法により 10%の時間短縮が見られた．しかし，偏りの度合いが大きくなるにつれて M クラスタでのキャッシュヒット率が向上するため，D クラスタへのリクエストの割合が少なくなり，提案手法の効果は小さくなっている.

階層化手法と階層化しない場合を比べた場合，提案手法を利用した読み出しプロトコルは階層化しない場合に比べ $\theta = 0.5$ の場合で 29%，$\theta = 1.0$ の場合で 28.5%の時間短縮になってい

表 6　M, D クラスタの性能測定（データサイズ 1MB）

	Insert		Retrieve	
	M クラスタ	D クラスタ	M クラスタ	D クラスタ
合計所要時間	179.4 [s]	287.1 [s]	386.2 [s]	1457 [s]
1 リクエスト処理時間	59.8 [ms]	95.7 [ms]	25.7 [ms]	97.1 [ms]
処理時間比	1 : 1.60		1 : 3.77	

る．これにより，読み出しの場合についても階層化構成の有効性が示された．

4.2　大きなサイズのデータを用いた測定

4.2.1　実 験 方 法

この実験環境は表 3 とほぼ同様であるが，以下の点で異なる．M クラスタのディスクにフラッシュメモリによる半導体ディスクを用いる．フラッシュメモリによる半導体ディスクは FibreChannel によって接続されている．データ本体についてはこのフラッシュメモリによる半導体ディスク上に格納し，分散ディレクトリの情報は Linux の ramfs ファイルシステム内に格納するようにした．M クラスタにおける最大キャッシュエントリ数は 1 ノードあたり 300 とした．また，D クラスタについては OS のバッファキャッシュの影響を小さくし，実際にディスクアクセスが発生するように，主記憶の容量を 64MB に制限している．大きなサイズのデータを用いた性能測定に際しては，間接ディレクトリ方式を用いた自律ディスクの HTTP インターフェース [7] を用いる．

この環境に対して，以下の手法で性能測定を行う．挿入・更新操作，読み出し操作をそれぞれ一定回数発行し，処理完了までの所要時間を計測する．リクエストは 3000 個のキー集合を用いる．挿入・更新操作については 3000 個のデータを各 1 回ずつ挿入する．読み出し操作については 3000 個のデータに対して，15000 回の HTTP リクエストを行う．格納されるデータのキーは 16 バイトのランダムな文字列であり，データのサイズは 1M バイトとする．

4.2.2　予 備 実 験

前節の場合と同様に，各操作の性能測定を行う前に M クラスタと D クラスタのそれぞれの性能を測定する．それぞれのクラスタで独立に従来の自律ディスククラスタを構成し，偏りのない状態で Insert，Retrieve 操作の性能を計測した結果が表 6 である．この結果より HTTP インターフェースを用いた大きなサイズのデータに対しては，この実験システムにおいて M クラスタは D クラスタに比べ，Insert で約 1.6 倍, Retrieve で 3.8 倍程度高速であることがわかる．

4.2.3　挿入・更新性能

前述の手法によって挿入・更新操作の性能を測定した結果を図 5 に示す．

大きなサイズのデータに対しては，推測手法により WTS プロトコルでは 8.0%，WTA プロトコルでは 3.0%，DW プロトコルでは 2.4%の時間短縮になっている．小さなサイズのデータの場合ほどではないものの，各プロトコルともに推測手法により

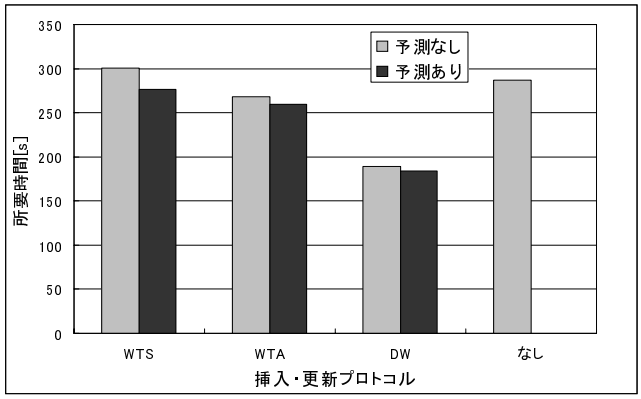

図 5　挿入・更新操作の実行時間（データサイズ 1MB）

処理時間が短縮されおり，推測手法が有効であることがわかる．大きなサイズのデータに対する場合でも，全体的な傾向としては図 3 の小さなサイズのデータの場合とほぼ同様である．大きなサイズのデータを用いた場合の環境では，挿入・更新処理のクラスタ間の性能差があまり大きくないため，小さなサイズのデータの場合では処理時間に大きな影響を与えていた M クラスタでの追い出し処理が大きな負荷にならずに，Delayed-Write プロトコルが予備実験における M クラスタ単体の性能に近い性能が出ていることがわかる．このことから，Delayed-Write プロトコルにおいて連続的な挿入・更新操作が行われるような状況では，それらの挿入・更新リクエストを M クラスタ内に保持できるよう十分な記憶領域を確保する必要がある．

また，階層化構成の各挿入・更新プロトコルと階層化しない場合を比較した場合，WTS プロトコルでは 3.6%，WTA プロトコルでは 9.5%，DW プロトコルでは 35.8%の時間短縮になっている．

大きなサイズを用いた測定で小さなサイズの場合に比べて推測手法の効果が小さかった理由は間接ディレクトリ方式を用いた自律ディスクの HTTP インターフェースの実装にある．間接ディレクトリ方式での Insert リクエスト処理は，以下の流れで行われる．

（1） HTTP リクエストを受けた Apache 自律ディスクモジュールが分散ディレクトリに対してデータの格納位置を問い合わせる．

（2） 分散ディレクトリはプライマリデータの格納位置とバックアップデータの格納位置を含むリストを Apache 自律ディスクモジュールへ返す．

（3） Apache 自律ディスクモジュールはリストの各格納位置に対してデータを書き込む．格納位置が自ノードである場合は通常のファイル操作により書き込む．格納位置が他ノードである場合はそのノードへ HTTP により書き込む．

この場合，クライアントが HTTP リクエストを送信した先のノードが，リクエストデータに対するプライマリデータを格納すべきノードである場合とバックアップデータを格納すべきノードである場合で，リクエストの処理コストは同じにな

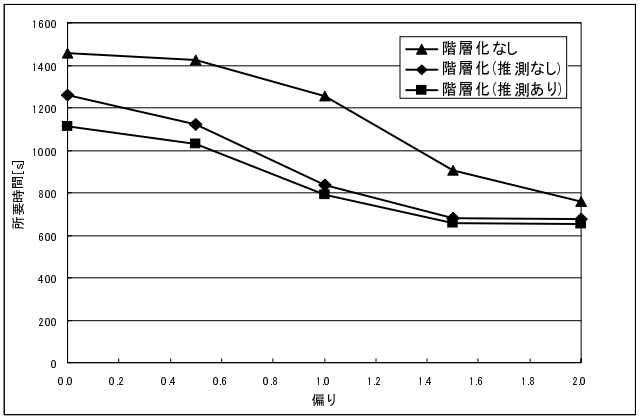

図 6　読み出し操作の実行時間（データサイズ 1MB）

表 7　偏りに対するキャッシュヒット率（データサイズ 1MB）

偏りの度合い θ	0.0	0.5	1.0	1.5	2.0
キャッシュヒット率 r	22.3%	34.1%	63.6%	79.8%	80.8%

る(注1)．そのため推測を行わずにランダムにノードを選択した場合でも，適切なノードへリクエストを送信できる確率が 2 倍になる．今回の実験環境では D クラスタのノード数が 6 台であり，ランダムに選択した場合に適切なノードを選択する確率は，Java インターフェースを用いた場合では $\frac{1}{6}$ であるのに対し，HTTP インターフェースを用いた場合は $\frac{2}{6} = \frac{1}{3}$ となる．このため HTTP インターフェースを用いた大きなデータサイズによる測定では，推測を行わずにランダムに送信先ノードを選択した場合と推測を行った場合の速度差が小さくなっている．

4.2.4　読み出し性能

読み出し操作の性能測定では小さなサイズのデータの場合と同様に，リクエストされるキーの偏りの大きさによる処理時間の変化を測定した．

HTTP インターフェースにおける読み出し操作の性能を測定した結果を図 6 に示す．この測定において，M クラスタでのキャッシュヒット率 r を測定した結果を表 5 に示す．

大きなサイズのデータにおける読み出し処理についても，小さなサイズのデータの場合と同様の傾向であり，階層化手法により高速化が達成されていることがわかる．階層化手法と階層化しない場合を比べた場合，予測手法を利用した読み出しプロトコルは階層化しない場合に比べ $\theta = 0.5$ の場合で 27.7%, $\theta = 1.0$ の場合で 36.9%の時間短縮になっている．

また，大きなサイズのデータを用いた読み出し処理においても推測手法により処理時間が短縮されている．推測手法により偏りの度合い $\theta = 0.0$ の場合で 11.4%，$\theta = 0.5$ の場合で 8.2%の時間短縮が見られた．偏りの度合いが大きくなるにつれて M クラスタでのキャッシュヒット率が向上するため，D クラスタへのリクエストの割合が少なくなり，予測手法の効果は小さくなる傾向も，小さなサイズのデータの場合と同様である．

(注1)：Java インターフェースを用いる場合はメッセージが一旦プライマリデータを格納すべきノードへ転送された後，バックアップデータを格納すべきノードに対してメッセージが転送される．

5.　まとめと今後の課題

本稿では，自律ディスククラスタの階層的構成手法において，D クラスタ内のディレクトリ探索とメッセージ転送の回数を低減することによる高速化手法を提案した．これは，高速なクラスタである M クラスタから低速なクラスタである D クラスタへリクエストを送信する際に，過去のリクエスト処理の記録から送信先ノードを推測する手法である．さらに実験による性能測定を行い，本稿で提案する推測手法の有効性を確認した．

また自律ディスククラスタの階層化構成における大きなサイズのデータに対する性能を自律ディスクの HTTP インターフェースを用いて測定し，大きなサイズのデータの処理についても自律ディスククラスタの階層化構成が有効であることを確認した．

本稿の評価実験では単一のクライアントを想定して測定を行ったが，複数のクライアントが存在する場合についても階層化構成の有効性を検証する必要がある．またクライアントのアクセスパターンが時間的に変化するような場合の性能についても検討する必要がある．

さらに，これまでは 2 種の異なるデバイスを用いて 2 階層のストレージシステムを構築することを対象としてきたが，我々が提案する自律ディスククラスタの階層的構成手法は 3 階層以上のストレージシステムに対しても適用が可能であると考えられる．3 階層以上のストレージシステムに対して本研究の提案手法の適用のためには，さらなる検討が必要である．

謝　　辞

本研究の一部は，文部科学省科学研究費補助金特定領域研究 (15017233)，情報ストレージ研究推進機構 (SRC)，独立行政法人科学技術振興機構 CREST，および 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] 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.

[2] 花井知広, 渡邊明嗣, 山口宗慶, 田口亮, 林直人, 上原年博, 横田治夫. 半導体ディスクを用いた自律ディスクの階層化. 情処学会研究会報告, データベースシステム DBS-131-19. 情報処理学会, 2003.

[3] 花井知広, 渡邊明嗣, 山口宗慶, 田口亮, 林直人, 上原年博, 横田治夫. 半導体ディスクによる自律ディスククラスタの階層化構成. 日本データベース学会 Letters, Vol. 2, No. 3, pp. 41–44, Dec. 2003.

[4] Witold Litwin, Marie-Anne Neimat, and Donovan A. Schneider. RP*: A Family of Order Preserving Scalable Distributed Data Structures. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, Santiago de Chile, Chile*, pp. 342–353. Morgan Kaufmann, September 1994.

[5] 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.

[6] 花井知広, 横田治夫. 自律ディスクを用いた Web サーバーの構成. 第 13 回データ工学ワークショップ論文集, DEWS2002 C3–4. 電子情報通信学会データ工学研究専門委員会, March 2002.

[7] 花井知広, 横田治夫. 自律ディスクを用いた Web サーバにおけ

る負荷偏りの影響. 情処学会研究会報告, データベースシステム DBS-128-29. 情報処理学会, July 2002.

[8] Mong Li Lee, Masaru Kitsuregawa, Beng-Chin Ooi, Kian-Lee Tan, and Anirban Mondal. Towards Self-Tuning Data Plaement in Parallel Database Systems. *SIGMOD Record*, Vol. 29, No. 2, pp. 225–236, Sep. 2000.