論文 / 著書情報
Article / Book Information

論題(和文)	アベイラビリティの高い分散ストレージ管理ソフトウェアの更新手法
Title(English)	A high-availability software update method for distributed storage systems
著者(和文)	小林 大, 渡邊 明嗣, 上原年博, 横田 治夫
Authors(English)	Dai KOBAYASHI, Akitsugu WATANABE, Toshihiro UEHARA, Haruo YOKOTA
出典(和文)	電子情報通信学会論文誌（D-I), Vol. J-88-D-I, No. 3, pp. 684-697
Citation(English)	IEICE Transactions on Information and Systems, Vol. J-88-D-I, No. 3, pp. 684-697
発行日 / Pub. date	2005, 3
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) 2005 Institute of Electronics, Information and Communication Engineers.

アベイラビリティの高い分散ストレージ管理ソフトウェアの更新手法

小林　大[†]　渡邊　明嗣[†]　上原　年博[††]　横田　治夫[†††,†]

A High-Availability Software Update Method for Distributed Storage Systems

Dai KOBAYASHI[†], Akitsugu WATANABE[†], Toshihiro UEHARA[††], and Haruo YOKOTA[†††,†]

あらまし　分散ストレージシステム内でストレージ管理機能を提供するソフトウェアは複雑化している．一方でストレージシステムは今やIT基盤の中核であり，たとえバグフィックスや機能向上のためであっても，そのためにサービスを中断することは許容されがたい．本論文では分散ストレージの管理ソフトウェアに対してアベイラビリティを減少せず更新する手法を提案する．提案手法では分散ストレージのもつストレージ仮想化技術と透過的データ移動技術の応用により，物理ノード内に構成された複数の論理ノード間で格納データに対する管理権限の委譲を行うことで，サービス無停止かつわずかな性能低下で管理ソフトウェアを停止し更新を行うことが可能である．また本手法について我々の提案する自律ディスクの模擬実装上で実験を行った結果を用いてその有効性を示す．

キーワード　ネットワークストレージ，ハイアベイラビリティ，無停止アップデート，ストレージ管理

1. まえがき

近年，大容量化に伴い，データ格納・提供を行うストレージシステムで利用されるストレージ管理ソフトウェアが複雑化している．一方で，ストレージシステムには高いアベイラビリティが求められている．

大規模ストレージシステムは，コストと性能の面から，複数のHDDを並列に配置することで構成される．そして，その上で動作するストレージ管理ソフトウェアにより，ユーザからは単一の大きなストレージとして見せる仮想化機能や，ストレージノードの故障からシステムを守る耐故障性の管理等の機能を実現している．しかし，管理機能へのバグの混入や，新機能追加は避けられず，実運用後もソフトウェアの更新を行う必要がある．一方で，ストレージシステムは今やIT基盤の中核であるから，たとえバグフィックスや機能向上のためであっても，システムを停止することはサービス機会の損失や生産性の低下などその高いコストを考えると許容されがたい．バグフィックスや新機能追加を起因とするストレージ管理ソフトウェアの更新を，サービスを停止せず行える機能を備えることは，高いアベイラビリティを求める分散ストレージシステムには必須である．

分散ストレージシステムにおいて，ストレージ管理機能とデータ格納機能の関係は2種類ある．データ格納ノードとデータ管理ノードが分離している分離管理モデルと，データ格納ノードが同時にデータ管理機能を有している結合管理モデルである．詳細については**2.**で述べるが，後者の結合管理モデルでは，ストレージ管理ソフトウェアの更新時に，同ノード内のデータへのアクセスが停止しないよう考慮する必要がある．

既存の，高いアベイラビリティを必要とするクラスタシステムにおける，計算機ノードの保守及びソフトウェア更新手法として，同機能の代替機にサービスを移譲し，ノードをオフライン化してソフトウェアの更新を行う手法が使われている[2]．これはノードの冗長化による耐故障性の向上も伴っている．しかし，結合管理モデルの分散ストレージシステムではこの手法は性能やコストの面で許容されがたい．単一ノードの代替機を逐一挿入する場合，サービスの移譲はデータの

† 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
†† NHK 放送技術研究所，東京都
NHK Science & Technical Research Laboratories, 1-10-1 Kinuta, Setagaya-ku, Tokyo, 157-8510 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

転送を伴うので，ディスクアクセスやネットワーク資源を消費しシステム性能を低下させてしまう．また，すべてのノードに対して代替機を用意する場合は，システム規模を肥大化させ運用コストを増加させるとともに，常に代替機内とデータの一貫性を保持するためシステム性能を低下させてしまう．

ストレージノードの故障と人為的なノードの取り外しを区別せずに扱い，システムのもつ障害回復機能を利用しノードをオフライン化する手法も利用されている [3], [4]．この方式では新バージョンのソフトウェアが旧バージョンとの上位互換性を保つ条件のもとでは，どんな更新も可能である．しかし，ノードを物理的にオフライン化することで一時的にデータの冗長度を低下させるため，その回復のためのディスクアクセスとネットワーク転送により，システム性能を低下させる．またすべてのストレージ管理ノードのメンテナンスが必要な場合，格納されている全データ量に比例した時間を必要とする．

そこで本論文では，結合管理モデルの分散ストレージシステムを管理・構成するソフトウェアに対し，サービスの停止や性能の低下を伴わず，かつ高速に更新する手法を提案する．提案するストレージ管理ソフトウェア更新手法では，ストレージシステムを構成する各物理ノード内に，複数の論理的なノードを構成する．そして，分散ストレージシステムのもつストレージ仮想化技術や透過的データ移動機能を利用し，共生する論理ノード間でディスクアクセスやネットワーク転送を伴わないデータ移動を行うことにより，旧バージョンのソフトウェアをデータ管理から開放し，高速なソフトウェア更新を実現する．また本手法についてその有効性を示すために，我々の提案する分散ストレージ技術である自律ディスク [1] を，PC 上に Java を用いて模擬実装した環境に適用し実験を行った．その結果，提案手法を用いた場合の性能低下は通常運用時の5%程度であり，単純に障害回復機能を利用した手法を用いた場合の 49～55%といった大きな性能低下量に比べ，極めてわずかであることを実証した．

以下に本論文の構成を述べる．続く **2.** では分散ストレージシステムの概要と提案手法に関連する技術について述べる．**3.** では，アベイラビリティと性能に優れたソフトウェア更新手法として，物理ノード内に複数の論理ノードを構成する機構及び，論理ノードを用いた手法を提案し，その手順を述べる．**4.** では我々の提案する自律ディスクの模擬実装を用いた実験により提案手法と基本的な手法の比較を行う．**5.** で，本提案手法の適用可能性と，本論文では扱わない分離管理モデルにおけるソフトウェア更新について論ずる．**6.** において，動的ソフトウェア更新手法に関する関連研究を述べる．最後に **7.** でまとめと今後の課題を述べる．

2. 分散ストレージと管理ソフトウェア

本章では，分散ストレージシステムとその機能について概略を述べ，そのストレージ管理ソフトウェアとソフトウェア更新の問題について触れる．

2.1 分散ストレージシステム

近年の大規模データ格納需要に従い，ストレージシステムは多数のストレージ装置をネットワーク結合することにより構成される．システムは，利用者に対して格納されるデータへの読出し・書込みアクセスといったサービスを提供する．また，複数のクライアントに対して，多数のストレージ装置を単一のストレージとして共有可能とする仮想化機能，システム利用中にノード間のデータ配置を変更できる透過的データ移動機能，部分ノードへのアクセス集中による性能低下を回避するための負荷分散機能，ノード故障からサービスを保護する障害回復機能等の高度なデータ管理機能を有し，管理コストの軽減やユーザビリティの向上を図っている．

2.1.1 2種類の管理モデル

分散ストレージシステムのストレージ管理形態は大きく分けて二つ考えられる．データ格納ノードとデータ管理ノードが分離している分離管理モデル（図 1）と，データ格納ノードが同時にデータ管理機能を有している結合管理モデル（図 2）である．

結合管理モデルの例としては LAN 上で複数の NAS

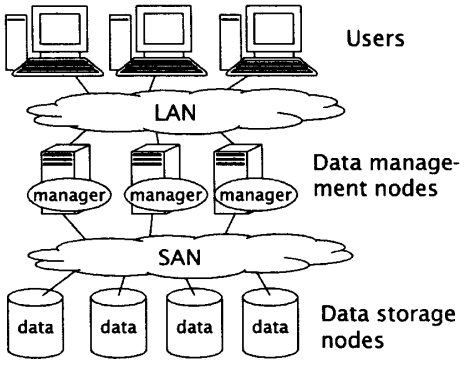

図 1 データ管理ノードとデータ格納ノードが分離しているモデル（分離管理モデル）

Fig. 1 A storage construction model dividing data management and storage.

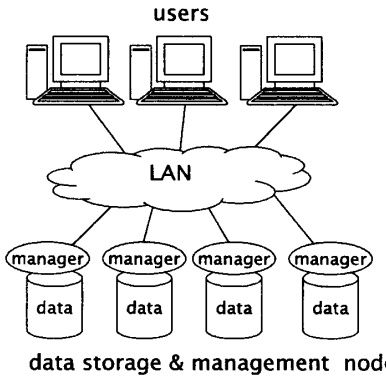

図 2 データ管理とデータ格納を同じノードで行うモデル（結合管理モデル）
Fig. 2 A storage construction model binding data management with storage.

(Network Attached Storage) により構成されたシステムが挙げられる．並列 NAS では，データを格納するストレージ装置自身がソフトウェアによりデータの管理も行っている．X-NAS [5] や自律ディスク [1] は，データ格納ノード上にデータ管理機能を追加し，それらのノードが協調しデータ管理機能を提供している．また，データ格納 PC のクラスタ結合によるシステムも同様に扱える．GFS (Google File System) [4] における chunk server や Oracle の RAC がこのような構成をとる．

本論文では結合管理モデルを対象とし，分離管理モデルについては **5.** にて議論する．

2.1.2 ストレージ仮想化

分散ストレージシステムはストレージの仮想化機能を用いて，データ配置や格納方式を利用者に対して隠蔽し，仮想的に単一の資源として見せる手法が取り入れられている．仮想化により，利用者側は，データが実際にどのストレージノードにあるのかを意識することなく，単一のストレージとしてアクセスし続けることが可能となる．

結合管理モデルにおける仮想化として，並列 NAS 環境では，単一仮想化コントローラを置く手法 [6] や，分散ディレクトリを用いることにより個々のノードが協調して実現する手法 [1], [5] が提案されている．

2.1.3 透過的データ移動

ストレージ仮想化を拡張することで，任意のノード内のデータを他のノード上に移動させることを利用者に対して透過的に行うことが可能となる．実際のシステムでは，異なるデバイスを階層的に構成した HSM (Hierarchical Storage Management) におけるデータ配置管理 [7] や，システム内負荷均衡化に利用 [1] されている．また，透過的データ移動を利用することで，あるノード内のデータをすべて他のノードへ移動し空のノードをシステムから切り離す作業を，ユーザから隠蔽することも可能である．

2.1.4 複製を用いた障害対策

透過的データ移動機能を備えたストレージ仮想化と，複製データ配置を組み合わせ，ノード障害検知時に動的にデータ配置情報を変更することで，サービス停止期間をほとんど発生させることなく，障害を回復させることが可能となり，ストレージシステムを容易に高信頼化することができる．ストレージシステムの高信頼化手法としては RAID3〜5 が用いられることが多いが，規模の大きなシステムではスケーラビリティやアベイラビリティを考慮し，一つの RAID グループを分散ストレージの要素として位置づけ，その上で更に複製配置による障害対策を行うことが多い．

2.2 管理ソフトウェア更新の問題点

ストレージ管理機能等の追加により，ストレージ管理ソフトウェアは複雑化し，バグフィックスやバージョンアップは必須となる．そして，これらの作業はシステム運用中にサービスを停止せずに行うことが求められる．

分散ストレージシステムにおいて，サービスとはデータの格納と格納されたデータに対するアクセスの提供である．よってアベイラビリティを減少させないソフトウェア更新とは，システムが格納するすべてのデータに対して提供不可能な時間を作ることなく，また新たなデータの格納要求を拒むことなくストレージを管理するソフトウェアを更新できることと定義できる．ここで，提供不可能でないとは，システムごとに定められた最長レスポンスタイムを超えない範囲で要求を処理可能であることとする．

ここで問題の単純化のために，ソフトウェア更新中のノードはデータ提供不可能であるという強い制約を仮定する．この仮定のもとで，すべてのデータがいかなるときでも提供可能であるためには，通常運用時にあるノードに格納されているデータを，当該ノードがソフトウェア更新を行う時点では一時的に（あるいは恒久的に）他のサービス可能なノードへと移動すればよい．よって，以降ではいかにシステムの性能を低下させることなくデータの提供サービスを他のサービス可能なノードへと移譲するかについて議論する．

結合管理モデル（図 2）では，システム構成ソフト

ウェアの保守を障害回復機能を利用してノードを除去した場合，当該ノードの保持するデータが失われてしまう．その結果，システム内に残った複製データによる冗長性回復のために多数のノード上でディスクアクセスやネットワーク転送が起こり，システムの性能を低下させてしまう．また，すべてのストレージ管理ノードのメンテナンスが必要な場合，それぞれのノードを除去する際に失われたデータの復旧処理が起こるため，結果として格納されている全データ量に比例した時間にわたり，性能低下が発生する．

そこで本論文では，このようなデータ管理とデータ格納機能を同一ノードで行っている状況下においても，システム性能を低下させない，高いアベイラビリティをもつソフトウェア更新手法を目指す．

3. 提案手法

本章では，論理ノードを用いたストレージ管理ソフトウェアの更新手法を提案する．まず，物理ノード内に複数の論理的なノードを構成する手法を提案し，続いて論理ノードを用いて管理ソフトウェアを低コストで更新する手法を提案する．

3.1 論理ノード

論理ノードは分散ストレージシステムにおけるデータ管理・格納機能をもつノードと同等の機能を果たす．そして，一つの物理ノードの中に複数の論理ノードを構成できる．ネットワークを介しては，各論理ノードはこれまでの物理ノードと同様にアクセス・管理でき，同じ物理ノード内に共生する論理ノードからのみそれが論理ノードであると認識される．各ノードは内部に保持するデータを固定長分割あるいは意味的分割を行い複数のデータ断片として管理する．どのノードがデータ断片を管理しているかについては，ストレージ仮想化機能により利用者に対して隠蔽される．

図 3 に，一つの物理ノード内に二つの論理ノードを構成した例を示す．論理ノードはそれぞれ外部とのネットワークの受け口となる一意のアドレス，ノードの内外のデータ配置を管理するデータ配置情報とメタデータ，並行制御ロックやアクセス状況等を管理するための管理テーブル，及びそれらの制御を行うプロセスから構成される．そして，共用のデータ格納領域内の各データ断片に対し排他的に管理権限をもつ．各データ断片の管理権限のやり取りは分散ストレージシステムにおける透過的データ移動機能に従い，管理用の情報やメタデータの送受信を論理ノード間で行うことで実現する．ただし，実際のデータは移動せず，後述する論理的な移動処理によって代替する．

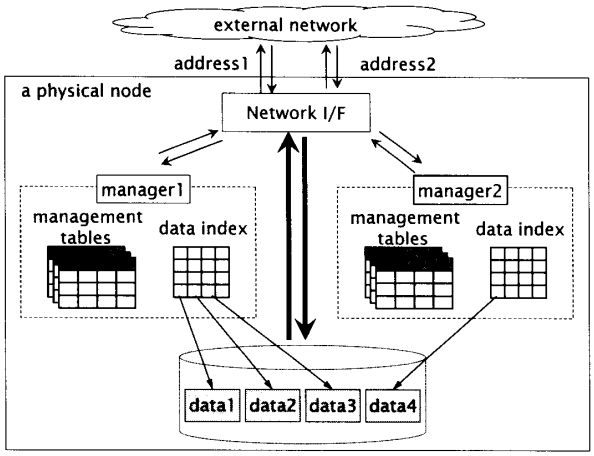

図 3 論理ノードの概念
Fig. 3 The concept of logical nodes.

3.1.1 論理ノード構成の制限

システムによっては，複製配置やデータ移動に制限がある場合があり，論理ノードを物理ノード内に共生させるには注意が必要である．システムの構成方法によっては，ノード障害（主にディスク物理障害）からの復旧を考慮し，ノード内に他ノードのデータの複製を保持することも考えられる．同じ物理ノード内の複数の論理ノードが複製を保持し合う場合，障害復旧の意味を成さないためこのような複製配置は避けなければならない．例として，同じノードグループ内で複製を持ち合う仕様の複製配置の場合は，共生する論理ノードは異なるノードグループに属さなければいけない．

また，インデックス構造の簡単化やアプリケーションに特有のデータ間関係保持のために，データの配置及び移動先に制限のある場合がある．本手法は後述する論理移動を行うことで性能低下を抑制するため，同じ物理ノード内の論理ノード間ではデータの移動が可能でなくてはいけない．例として，関係データベースにおけるテーブルを水平分割（値域分割）して各ノードに分散させる場合，論理ノードはそれぞれ格納したデータの key 値が隣接していなくてはいけない．

これらを満たさない場合，いずれかの制限を緩和する戦略変更が必要となる．**3.2.5** において，そのような例を紹介する．

3.1.2 論理移動

同じ物理ノード内の論理ノード間でのデータ断片の移動は，管理権限の移譲と分散ストレージのもつ透過的データ移動機能の利用によって行う．つまり論理移

動とは，UNIX ファイルシステムにおける同パーティション内での mv コマンドと同様に，物理ディスク上の格納アドレスの変更を行わず，管理プロセス間でデータ断片に対するディスク内位置情報（アドレス）を譲渡し合うことで同ディスク内でデータの管理権限を変更する操作を表す．そして，それらの論理ノード間で管理権限が移動したことはストレージ仮想化機能と透過的データ移動機能により利用者から隠蔽される．実際のデータの移動は伴わないため，論理移動にかかるコストはデータのサイズによらず，データ配置情報の更新コスト及びメタデータ情報のやり取りのコストが大半となる．また，他の物理ノードとのデータ転送が発生しないため，通信コストはデータ配置情報変更に伴う外部ノードとの更新情報の授受にかかるわずかなコストに集約される．それに加え，データの論理移動中のいかなる時点においても，同じデータがディスク上に複数存在することはなく，この手法を用いることで必要となる余分なディスク領域は，複数の論理ノードのそれぞれのための実行プログラム格納領域のみである．

論理移動中のデータに対するアクセス要求は，そのデータに対するアクセスの提供が停止しないように適切に処理されなければならないが，それは透過的データ移動機能による並行性制御で達成する．以下にその様子を示す．まずデータへのアクセス要求は，ストレージ仮想化を担う機能部分に到達する．ストレージ仮想化のもつデータ配置情報のうち当該データ及びデータ配置情報内のそのデータに関する情報を含む部分へのアクセスは，透過的データ移動機能のもつ並行制御機能により禁止されているため，リクエストはストレージ仮想化機能中で保持される．その後，論理移動が終了しデータとデータ配置情報へのアクセスが許可されると，保持されていたリクエストの処理が再開される．ストレージ仮想化機能は論理移動により変更された後のデータ配置情報を走査し，リクエストをデータの保持されているノード，つまり当該データの論理移動先の論理ノードへと転送する．分散ストレージシステムの仮想化機能を用いて物理ノード間データ移動を隠蔽する場合と同様に，以降の当該データへのリクエストは，すべて論理移動先ノードへのみ到達する．当該データへのアクセス禁止時間は発生するが，物理ノード内のデータ配置情報の受渡しにかかるわずかな時間のみであるため，サービス停止には至らない．

同様に，現在アクセス処理中のデータに対する論理移動要求も，ストレージ仮想化機能と透過的データ移動機能の並行性制御により，適切なタイミングまで保留されるため，データへのアクセスを中断してしまうことはない．

3.1.3 その他のノード間通信

論理ノード間，あるいは論理ノードと他物理ノード間で，データの移動以外にもシステム構成情報や，複製配置情報等の相互通信が行われる．各論理ノードは，実データ部分が共有されている以外は，これまでの物理ノードと同等の機能を保持することにより，従来のストレージシステムがもつ，ストレージ仮想化機能や透過的データ配置，障害回復機能等々のノード間通信が必要な機能をそのまま使用することが可能である．これにより従来のシステムに対し，我々の提案手法を適用する際の追加コストを低く抑えられる．

3.2 論理ノードを用いたソフトウェア更新手法

本節では，データ管理とデータ格納を同じノードで扱う結合管理モデルにおいて，論理ノードを用いた管理ソフトウェア更新手法を提案する．まずその手順を述べ，その特徴について述べる．またその適用例を示す．

3.2.1 基本戦略

2.2 で述べたように，当該ノードのデータを他のサービス可能なノードへと移譲することで，システム全体としてはサービスを停止せずに，当該ノードのソフトウェアを更新することができる．しかし，データを物理ノードの外に出力するとシステム性能が低下する．

そこで，当該物理ノード内で，ソフトウェア更新中でもサービス可能な論理ノードを起動し，それらの間でデータの論理移動（管理権限の移譲）を行う．

3.2.2 手　　順

新しいバージョンのソフトウェアで動作するノードは，古いバージョンのソフトウェアで動作するノードと，管理情報及びデータのやり取りができるだけの上位互換性をもつと仮定する．次に述べる手順により，物理ノード内の論理ノードのソフトウェアをサービスを停止せずに更新することができる．

（1） まず，前章で述べた機構により，物理ノード内に新しいバージョンのソフトウェアで構成された論理ノードを構成する（図 4）．更新中は，バージョンの違う二つの論理ノードが共生することになる．

（2） 新しいバージョンの論理ノードを，分散ストレージシステムのもつ，ストレージノード追加機能を

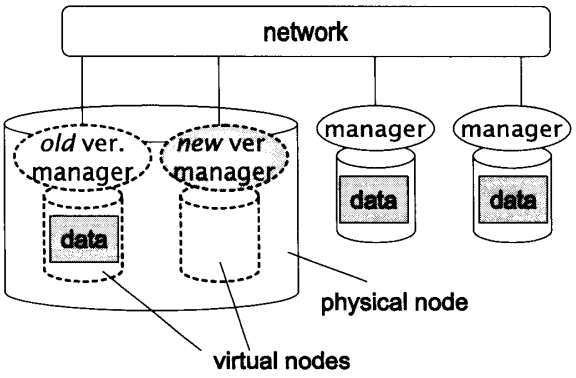

図 4　論理ノードと論理移動を用いた旧 Ver. ノード除去
Fig. 4　Detaching a old version node with logical move between virtual nodes.

用いて動的にシステムへ参加させる．

（3）　そして論理的移動を用いて，旧バージョンノードの管理下にあるデータ断片をすべて，新バージョンノードの管理下へと順次権限変更する．これにより，旧バージョンのノードはデータの管理から開放される．

（4）　その後，旧バージョンのノードを故障ノード除去機能を用いてシステムから除去する．

以上の操作により，この物理ノードはサービスを停止することなくストレージ管理ソフトウェアの更新を完了することが可能となる．

システムを構成するすべてのノード上の管理ソフトウェアを更新するためには，上記の操作をすべての物理ノード内で行わなくてはならない．提案手法は，処理中にシステムの冗長度が低下することがないので，すべての物理ノードで同時に処理することも可能であるし，あるいは各物理ノードごとに更新処理を行うことも可能である．

3.2.3　特　　徴

提案手法は，物理ノード外にデータ自体を転送する必要がなく，長いディスクアクセスやネットワーク資源の消費も伴わないためシステム性能を低下させることなくソフトウェアの更新を行うことができる．

またノード間のデータ移動は，分散ストレージシステムのもつ透過的データ移動機能のもとで行われるため，利用者からデータ管理ノードが変更されたことを隠蔽でき，データへのアクセス提供も停止することはない．同様に旧バージョンの論理ノードをシステムから除去する場合も分散ストレージシステムのもつ障害回復機能を用いるが，当該ノードはデータの管理からは開放されているため，複製データの再生成が発生することはなく，そのための性能低下も発生しない．

更に，これまでの物理的にノードを切り離す手法では，常に一定数のノードがシステム内に残存しサービスを提供しなくてはならず，部分ノードごとに更新せざるを得なかったが，本提案手法では，すべての物理ノード内で論理ノードを構成しソフトウェアを更新することも可能であり，更新時間の短縮に有効である．

ただし，大きな制約として同時に複数を起動できないソフトウェアの更新には適用できない点が挙げられる．つまり，OS やデバイスを直に操作するようなコードを含む個所には適用できない．しかし，新機能追加やバグフィックスの場合を考慮すれば，ノードの再起動を伴わない範囲での更新にのみ適用可能でより高速な手法の存在が有効である．

3.2.4　コスト比較

本手法により削減される更新コストを見積もる．比較対照（既存手法）として物理ノード除去による更新を用いた場合，本提案手法は常に既存手法よりもコストは低くなる．データ断片数が少なくなるにつれ，より有効に作用する．

物理ノード除去のコスト C_{phy} は，システム構成変更コスト C_{mdfy} とデータ移動コスト C_{move} の和である．

$$C_{phy} = C_{mdfy} + C_{move}$$

更に，データ移動コスト C_{move} はデータ断片 1 個当りの，データ断片配置・管理情報（メタデータ）転送コスト Cm_{meta}，実データ転送コスト Cm_{data}，データ配置情報更新コスト Cm_{map}，に分割される．

$$C_{move} = n \times (Cm_{meta} + Cm_{data} + Cm_{map})$$

データ断片数 n が非常に大きい場合，C_{mdfy} が無視できる．また一般にメタデータは実データと比較して少量となるよう設計されるため，$Cm_{meta} \ll Cm_{data}$ となる．よって物理ノード除去による性能低下は，データ配置情報更新による Cm_{map} と，実データ移動による Cm_{data} が主である．

一方，提案手法を 1 物理ノード内に 2 論理ノードの構成で用いた場合の更新コスト C_{lg} は，ストレージ仮想化機能と透過的データ移動機能について物理ノード除去と同じ技術を使用すると仮定すると，論理ノード間でのデータ配置情報更新と，メタデータ移動が支配的となり，以下のようになる．

$$C_{lg} = C_{mdfy} + n \times (Cm_{map} + Cm'_{meta})$$

ここで，Cm_{meta} はメタデータの read と write が異なるノードで起こり，Cm'_{meta} は read と write が同じ物理ノード内で起こる．このため性能低下時間については後者の方が長いが，単位当り性能低下量は前者の方が多く，性能低下コストとしては $Cm_{meta} \simeq Cm'_{meta}$ である．

これより，同量の格納サイズのストレージノードについて以下のことがいえる．

- （データ断片サイズ → 大）⇒（n → 小，Cm_{data} → 大）の場合，Cm_{meta}, Cm_{map} の回数が減り，我々の提案手法がより効果的に働く
- （データ断片サイズ → 小）⇒（n → 大，Cm_{data} → 小）の場合，提案手法と既存手法の性能差である Cm_{data} が $n \times (Cm_{meta} + Cm_{map})$ に対し，小さくなり，我々の提案手法の性能低下は物理ノード除去による更新コストに近づく．
- $C_{phy} > C_{lg} \cdot n$ によらず，既存手法も提案手法も常にデータのノード間移動は発生する．既存手法ではそれに加え，どんなに小さくとも実データ移動が加わる．

ここで，Cm_{map} による性能低下は主にデータ配置情報の並行性制御に伴うリクエスト待ち時間に起因するスループット/レスポンスタイム悪化であり，Cm_{data} による性能低下は，データアクセスのための資源（ネットワーク帯域及びディスク I/O）消費によるスループット/レスポンスタイム悪化である．また，Cm_{data} による性能低下時間はデータ総量に比例し，$Cm_{map} + Cm_{meta}$ による性能低下時間は $n \times$ メタデータサイズ に比例する．

3.2.5 複製配置に制約のある場合の戦略

提案手法では，同物理ノード内の論理ノード間で論理的データ移動を行っているが，システムによっては複製の配置やデータの移動方法に制約がある場合がある．この場合，障害回復のための複製が同物理ノード内に配置されてしまい，耐故障性が失われてしまうことも考えられる．複製配置に Chained Declustering [8] を用い，データ配置戦略に値域分割を利用したシステム（図 5）はそのようなシステムの例である．このような場合でも，複製配置を工夫することでその信頼性やアクセスの透過性を損なうことなく提案手法を適用可能であることを以下で述べる．

Chained Declustering はストレージ装置列に対しリング状に複製を配置（図 5）する，信頼性が高い複製配置戦略である．また値域分割は，データ断片に対する識別子の一意な順序付けに従って並べ，その連続部分範囲を各ストレージに格納する，並列 DB において用いられる手法である．値域戦略を採用した場合データの移動は隣接ノード間でのみしか行えない．また，Chained Declustering により複製は隣接ノードに配置する．このような場合，論理ノードを挿入する場所に複製が配置されてしまい，その物理ノードが故障した場合データが失われてしまう．

このような場合に提案手法を適用可能とするために，複製配置を変更した例を図 6 に示す．隣接ノードに配置という制約を，2 ノード隣へ配置という制約に変更する．また，論理ノードはソフトウェア更新対象物理ノードだけでなく，隣接の，当該ノードの複製を保持するノード内にも挿入する．これによりデータ移動に関する制約による論理ノード挿入個所と，複製配置個所を別のノードにでき，システムのもつ耐故障性を失うことなく提案手法を適用できる．また，データ移動に伴う複製の配置変更もそれぞれの物理ノード内で行うことができ，システム性能をほとんど低下させることはない．

このように，提案手法では分散ストレージシステムのもつ耐故障性やアクセスの透過性を損なうことなく，わずかな性能低下で各ノードの構成ソフトウェアを更新可能である．

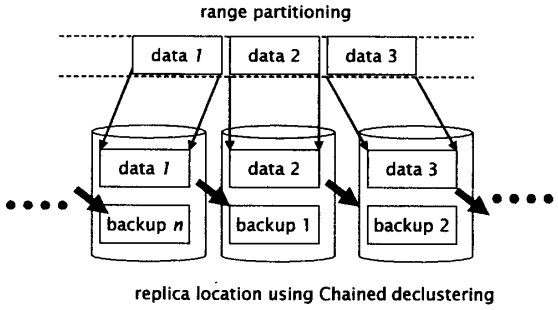

図 5 Chained Declustering と値域分割を利用した分散ストレージの例
Fig. 5 The example of distributed storage systems: using range partitioning strategy and Chained Declustering replication.

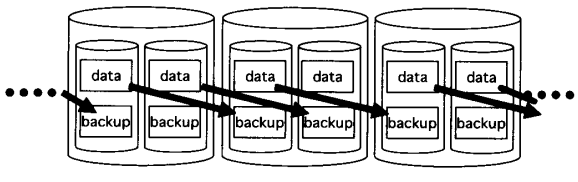

図 6 提案手法適用のために複製配置変更
Fig. 6 A small modification of replication storategy in order to apply our methods.

4. 実　　験

提案手法の有効性を確認するために，我々が提案している自律ディスク [1] システムの模擬実装上に提案手法を実装した．その上で，通常運用時からの性能低下を測定し，単純な障害回復処理のみによるノードオフライン化と比較した．

以下では，まず我々の提案する自律ディスクの概要について述べ，その後実験設定の詳細に触れる．そして，実験結果とその考察を行う．

4.1 自律ディスク

自律ディスクは我々が提案しているアベイラビリティやスケーラビリティに優れた，データ管理とデータ格納を同じノードで行うモデルの分散ストレージ技術である．自律ディスクではディスク装置内の制御用プロセッサとキャッシュ用のメモリを利用し，負荷分散や障害回復等のシステム管理を行う．自律ディスクではシステムはネットワークに接続されたディスクノードのクラスタにより構成される．システムを運用する上で特別な集中管理サーバは必要としない．

動的データ配置管理などのデータ管理をクライアントから透過的に行うための分散ディレクトリ構造として，分散 Btree 構造を用いることを想定し，また，障害復旧のための複製配置戦略として Chained Declustering [8] を採用している．

そのほかに ECA ルールやトランザクション処理等の機能をもち，これらを組み合わせることによりデータ分散配置，偏り制御，耐故障性及び障害からの回復といった高度なシステム管理を自律的に行うことができる．

4.2 実験構成

自律ディスクは，Linux クラスタ上に Java を用いて模擬実装されている．そこで，提案手法を次のように実現した．まず，複数の論理ノードは，それぞれ異なる UNIX プロセスとした．また，ネットワークインタフェースは単一の NIC に，IP Alias 機能を用いて複数の IP アドレスを割り当てることで実現した．そして，複数の論理ノードから共有されるデータ格納領域は，ext3 ファイルシステムによるパーティションを用意した．その上で，データ管理単位はファイルとし，データ格納位置はファイルシステム上のパス名を用いた．

性能測定のために，表 1 の PC と十分なバックボーン性能をもったスイッチを用いて 6 台構成の自律ディスククラスタを構成した．自律ディスク上のストレージ仮想化機能としての分散ディレクトリとして，aB$^+$-Tree [9] を用いた．これは，配列構造のグローバルインデックスによりデータ格納ノードを決定し，通常の B$^+$-Tree 構造のローカルインデックスにより内部位置を決定するインデックス構造である．グローバルインデックスの複製をすべてのノードで保持し，データ配置に変更があった場合全ノード間で同期更新を行う．性能への影響については考察において触れる．B$^+$-Tree の葉ノード（管理対象）は各データのメタデータとし，メタデータサイズは一つ当り 4 kByte 固定長とした．

表 1　ストレージノード・クライアントノード性能緒元
Table 1　Specifications of PCs used as storage nodes and clients.

#nodes	6 台 (Storage) + 6 台 (Clients)
CPU	AMD Athlon XP-M 1800+ (1.53 GHz)
MEM	PC2100 DDR SDRAM 1 GByte
Network	1000BASE-T + TCP/IP
HDD	TOSHIBA MK3019GAX (30 GByte, 5400 rpm, 2.5 inch)
OS	Linux 2.4.20
Local File System	ext3 FS
Java VM	Sun J2SE SDK 1.4.2_04 Server VM

格納するデータサイズは簡単化のためにすべてのデータで同じとし，格納するサイズは総量 24 GByte とした．1 物理ノード当りデータ 4 GByte と他ノードのバックアップ 4 GByte の計 8 GByte が格納される．データ断片サイズ及びデータ数は 16 MByte × 1536，8 MByte × 3072 の 2 種類用意し，それぞれ各ノードに均等に分散格納した．

アクセス負荷を発生させるクライアントには自律ディスクノードと同じ構成の PC を 6 台用いた．各 PC 上でスレッドを六つ用意し各々のスレッドから各ストレージノードに格納されているデータに対して均等に読出しによる負荷をかけた．これは負荷均衡化機能が有効に作用している状況を想定している．

このような構成のもとで，次の四つの状況下における読出しアクセスのスループット及びレスポンスタイム，及び実行時間を測定した．

- 通常運用時（16 MByte データ断片時のみ）
- 障害復旧機能を用いたノード除去により特定のノード内を空にし除去する間
- 提案手法を用い，特定の物理ノード内で旧バージョン論理ノード内のすべてのデータを新バージョン

の論理ノードへ管理移譲し旧バージョンノードを除去する間.

- 提案手法を用い，六つすべての物理ノード内で旧バージョン論理ノード内のすべてのデータを新バージョンの論理ノードへ管理委譲する間（16 MByte データ断片時のみ）

4.3 結　　果

4.3.1 論理ノードによるオーバヘッド

まず，物理ノード内に複数の論理ノードを構成した場合の性能低下について測定した．図 7 に 5 回測定した結果の 95%信頼区間を示す．通常の運用時に平均 51.7 MByte/s であったシステムにおいて，複数の論理ノードを物理ノード内に包含した状態での性能は平均 51.4 MByte/s であった．これにより，論理ノードを複数構成することによるオーバヘッドはわずかであることを確認した．

4.3.2 実行時スループットと処理時間

負荷をかけた状態でノード除去を行いスループットの減少量を観測した．実験はそれぞれ 5 回ずつ行い平均をとった．結果として，図 8 に，実行時スループット平均の 95%信頼区間を示す．

通常運用時（usual）に対し，物理ノードの除去を用いた場合，物理ノード間のデータ移動により平均スループットはデータサイズ 16 MByte，8 MByte に対してそれぞれ 49%，55%，低下した．一方，論理ノードを用いた場合，論理ノード間の論理データ移動中の平均スループットは通常運用時に比べデータサイズ 16 MByte，8 MByte に対していずれも 5%低下した．ここで，データサイズが小さくなるにつれスループットが低下しているのは，データ量当りの処理回数が増えオーバヘッドが増加しているためである．

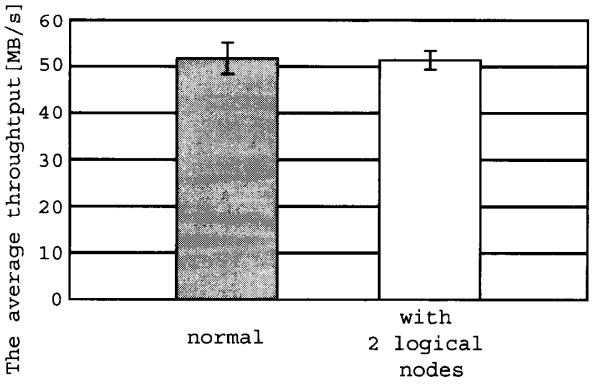

図 7　論理ノード動作のオーバヘッド
Fig. 7　The average throughput when two logical nodes run in a physical node.

また，一つのノード内に格納されている全データ（4 GByte × 2）を移動させるのに必要とした時間平均の 95%信頼区間を図 9 に示す．物理ノードの除去を用いた場合は 16 MByte × 1536 個で平均 1160 秒，8 MByte × 3072 個で平均 1075 秒を要した．一方，論理ノードを用いた場合は，それぞれ平均 61 秒，91 秒を要した．

4.3.3 レスポンスタイム

アクセス要求に対するレスポンスタイムは，読出し命令を発行する直前から一つのデータ断片がクライアントに到着するまでの時間を計測した．

図 10 に，データ断片サイズ 16 MByte の場合の，レスポンスタイム相対累積度数分布グラフを示す．データ断片サイズ 8 MByte の場合も同様の傾向を示した．このグラフでは，ある横軸上の値 x[s] 以下の時間で処理できたリクエストが全体のリクエスト数の何%であったかを表す．

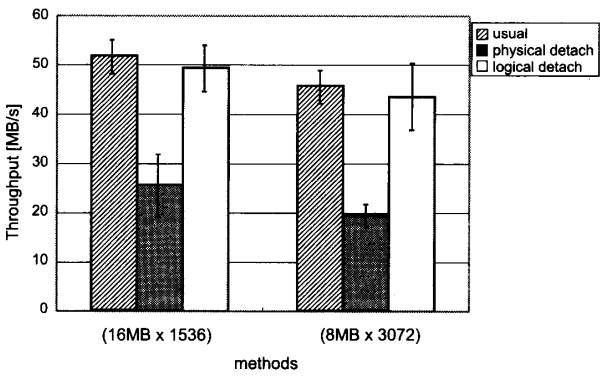

図 8　ノード除去処理の実行時平均スループット
Fig. 8　The average througput when the process that detach a node physically/logically.

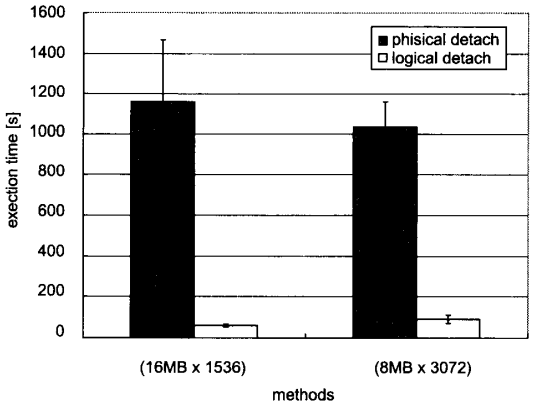

図 9　4 GByte × 2 のデータをもつノード除去処理の平均実行時間
Fig. 9　The mean execution time of detaching a node which has 4 GByte data.

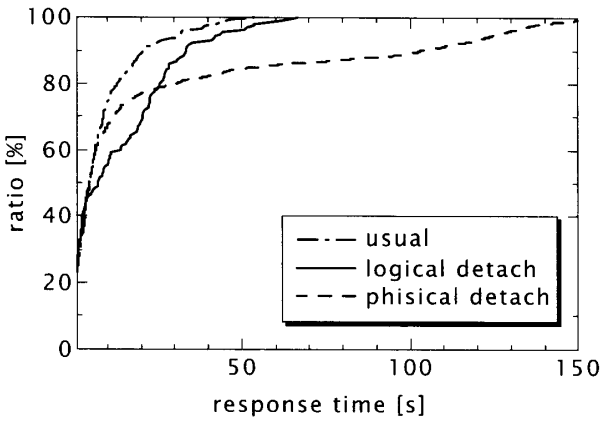

図 10　実行時レスポンスタイムの相対累積度数分布
Fig. 10　The relative cumulative frequency of Response time.

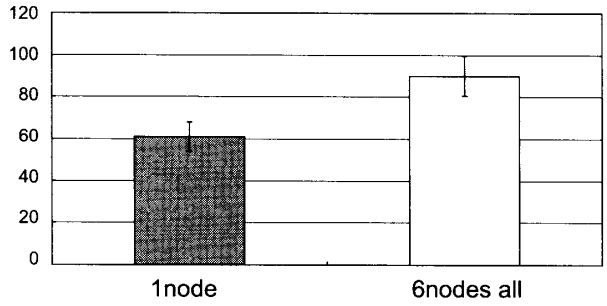

図 11　6 ノード並列データ移動時実行時間
Fig. 11　The execution time of logical move at 6 nodes concurrency.

物理ノード間のデータ移動を伴う手法の場合，通常運用時に比べ 30 秒以上を要するアクセスが倍に増えていることが読み取れる．一方で，論理ノードを用いた場合，10 秒～30 秒程度レスポンスタイムを必要とするリクエストが増加しているが，最大レスポンスタイムは通常運用時とほぼ一致しており，最大レスポンスタイムがシステム利用者の要請により固定されている場合でも，アベイラビリティを損なわない．

物理ノード除去による性能低下は，物理ノード間データ転送に伴うディスクアクセスやネットワーク負荷に起因する．クライアントからの読出しリクエストはディスクアクセスを伴い，物理ノード間データ移動操作と並行に処理できないため，リクエストのレスポンスタイムは長くなったと考えられる．一方，論理ノードを用いた手法では，性能低下は分散ディレクトリの更新に起因する．分散インデックスの更新による競合はデータに対するアクセス要求全体に対してわずかなため，レスポンスタイムへの影響が少なかったと考えられる．

4.3.4　複数同時実行時の挙動

6 ノードすべてのノード内で論理ノード間のデータ移動を行ったところ，すべてのノード内でデータ移動が完了するまでに 90.1 ± 9.7 [s] を要した．**4.3.2** で計測した 1 ノードごとのデータ移動に必要な 61 ± 7.1 [s] に比べわずかに 50%の上昇にとどまった．この差を図 11 に示す．これは，1 ノードごとに更新を行う場合は 1 秒当り 4 回のグローバルインデックス更新要求が各ノードに到達しており，6 台同時では 1 秒当り 17 回の更新要求が到達していることになる．

また，この際のスループットは 49.3 ± 2.3 [MByte/s] であり，1 ノード時の 49.4 ± 4.6 [MByte/s] とほぼ同等の性能低下であった．

今回の実装では，データ配置に値域分割を利用している．そのため，物理ノード内の論理ノード間データ移動が同時発生しても，データ配置情報変更において競合が発生することはほとんどない．実験に用いた実装では，グローバルインデックスのディスク書込み時のみ排他制御を行っている．よってスケーラビリティに優れ，同時実行における性能低下もわずかであった．他のデータ構造でストレージ仮想化を実現しており，データ配置情報変更において競合が多く発生する場合，スループット性能低下時間が本実験よりも長くなることが考えられ，またデータ配置情報の並行性制御の待ち時間のためにスループットが低下することが考えられる．

4.4　考　　察

4.4.1　既存手法との性能差

性能低下要因について考察する．ここで，性能低下を表す指標を（スループット低下量）×（スループット低下時間）であると考える場合，提案手法は単純な手法に比べわずか 1%未満の性能低下でソフトウェアのオフライン化が可能であると考えられる．**3.2.4** で述べたように，各手法におけるデータ移動時の性能低下は主に，データ配置情報更新コスト Cm_{map}，メタデータ移動コスト Cm_{meta}，実データ移動コスト Cm_{data} による．データを異なるノードへ管理委譲するアプローチのもとでは，この二つのコストは必須である．

2 手法間の性能低下の差は，主に Cm_{data} であり，今回の実験のおいては主にディスクアクセス性能に起因するものと考えられる．今回実験において測定されたスループット約 50 MByte/s（300 Mbit/s）は GbE の帯域の飽和量以下であり，また実行中に観測した

CPU 利用率も 100%ではなかったためである．

物理ノードを用いた既存手法における性能低下時間について，16 MByte × 1536 個の場合は 8 MByte × 3072 個とほぼ同時間である．これより，既存手法では実データ量が性能低下において支配的であると考えられる．一方提案手法による性能低下時間は 16 MByte × 1536 個の場合は 8 MByte × 3072 個の 2/3 であり，データ数に依存していることが分かる．よって，提案手法をより効果的に用いるためには，例えばデータ断片の位置情報を複数まとめて変更する等が考えられる．

論理ノードを用いた場合の性能低下の要因として考えられるのは透過的データ配置機能のデータ配置情報を更新するためのコスト Cm_{map} + メタデータ移動コスト Cm_{meta} である．ここで，性能低下時間は Cm_{map} に，論理ノードを用いた場合の性能低下量は Cm_{meta} と Cm_{map} によるものであると考えられる．今回の実験では模擬実装としてデータ配置情報の管理に aB$^+$-tree を用いており，グローバルインデックスを全ノードで共有している．この場合，一つのデータを論理/物理ノード間で移動するたびに，グローバルインデックスの更新のためすべてのノードの保持する根ノードを更新する必要があり，性能低下時間を長くする要因となっている．性能低下時間について，システム内ノード数が増加した場合，この同期コストはより顕著に現れると考えられる．より効率の良い分散インデックスとして Fat-Btree [10] 等が提案されている．Fat-Btree を用いれば，ノード間境界情報の更新の伝搬は隣接する数ノードのみに限定されるため，提案手法による性能低下をより小さなものに抑えることができると考えられる．

性能低下量については，メタデータ一つ当り 4 kByte とはいえ，HDD 読出しのためのヘッド移動や回転待ち等のレイテンシを含む，HDD 占有時間が発生するため，これによるもの，及び読出しサービスとデータ移動機能の競合のための待ち時間である．特に今回の実験については競合待ち時間が大きいと考えられる．これは，6 ノード同時実行時にはメタデータアクセス量は 6 倍になるにもかかわらず，性能低下量は 1 ノード時から変化しなかったことから分かる．競合待ちの割合は，単位時間当りのサービスとデータ移動の発生確率の比により変化する．

以上のように，提案手法はスループット及び実行時間のいずれに対しても単純な手法よりも優れた結果を出しており，更新中の性能低下を抑えるという我々の主張に対し期待どおりの結果を示したといえる．また，本手法をより有効に使用するためには，データ断片（データ管理単位）のサイズを大きく，データ位置更新回数を減らすのがよい．今回はデータサイズとして 16 MByte，8 MByte を使用した．これは GFS [4] 等に用いられているものよりも小さいが，ローカルシステム上のデータ管理単位と比べると大きい．データ断片サイズの決定に自由度が少ないシステムの場合，複数のデータ断片を一括して移動することで，データ断片サイズが大きい場合と同等の有効性が得られると考えられる．

4.4.2 更新処理実行時間

本実験では，合計量がおよそ 20 GByte 程度の量での結果として，どちらの手法も数分〜20 分程度の結果となった．しかし，現実のシステムにおいては本実験の 20 倍程度の量，1 ノード当り 100 GByte のデータを格納することも十分考えら，システム構成ノード数も増加することが考えられる．この場合，全データ量，データ断片数ともに本実験のものよりもはるかに増加し，更新処理実行時間もデータ量の規模に比例して増加するものと考えられる．

仮に 1 ノード当り 100 GByte であるとし，本実験で得られた結果から推定すると，6 ノード構成のデータをすべて移動しソフトウェアを更新するのに，既存手法で 20 分 × 20 倍 × 6 = 40 時間 必要とする．一方，提案手法では 16 MByte 断片のとき 1 分 × 20 倍 × 6 = 2 時間 あればよい．更に，提案手法により 6 ノード同時更新する場合，わずか 1.5 分 × 20 倍 = 30 分 で終了可能である．

5. 議　　論

5.1 適用可能な更新の範囲

本提案手法の適用可能な更新の範囲を明確にする．

3.2.2 で記述した仮定のように，本提案手法では一時的に旧バージョンノードと新バージョンノードがシステムの中に混在する．そのため，(i) 新バージョンが旧バージョンと互換性をもつ範囲の更新，であることが必要とされる．これには (i-a) 更新によりノード間通信が変更されない場合と，(i-b) 新バージョンが旧バージョンと互換性を保つように設計されている場合が考えられる．これは，本手法の対比として用いている，物理ノード除去によるソフトウェア更新にも同様に要請される条件であり，現実的な制限であると考える．

また，**3.2.3** で述べたように，(ii) 同時に複数の主体が共生できる必要がある．このため，低レベルで単一資源の排他制御を行うようなソフトウェアについては適用は難しい．より低レベルなソフトウェアについては後述する関連研究 [12] のような複雑な手法を用いる必要がある．

以上の (i)，(ii) を満たす範囲においては，ソフトウェアの更新が可能である．

5.2 分離管理モデルにおけるソフトウェア更新

本論文では，**2.** において述べた結合管理モデルの分散ストレージシステムでのストレージ管理ソフトウェア更新について扱った．本節では他方のモデル，分離管理モデル（図 1）での更新について考察する．

分離管理モデルの代表的な例として，SAN (Storage Area Network) にストレージ装置とサーバを結合したシステムが挙げられる．SAN を用いたシステムでは，データ格納ノード（HDD や RAID 装置）と，データ位置やアクセスの管理をするノード（サーバ）は分離されている．

分離管理モデルにおけるストレージ管理ソフトウェアの更新には，システムのもつ複製を用いた障害回復機能を利用し管理ノードをシステムから取り外す手法が有効に作用する．故障と判断された管理ノードの管轄であったデータの管理権限は，他の正常な管理ノードへと動的に移譲される．管理ノードは実データを保持しないため，管理ノードの除去に伴うデータアクセスやネットワーク転送は，実データ転送に比較してわずかな管理データ転送のみである．よって，本論文で述べたような性能低下は，分離管理モデルにおいては基本的に起こらないといえる．

ただし，性能向上のために管理ノードが積極的にデータの複製を保持する場合，ノード除去後の再挿入時に性能が低下することは考えられる．

6. 関連研究

高いアベイラビリティが求められるシステムに対する無停止ソフトウェア更新手法は，ストレージシステムに限らず，様々なアプローチが提案されている．

Ajmani は既存の代替機を用いる手法に，高度な更新スケジューリングや異なるバージョンのノードの同時運用等の手法 [11] を提案している．この手法は，本論文における提案手法と同様，異なるバージョンのソフトウェアをもつノードの混在を許している．しかし，各ノードごとの更新は代替機を用いた既存手法であるため，分散ストレージシステムに適用した場合，ディスクアクセスが発生しネットワーク帯域を消費する点は変わらない．

また，動的にソフトウェアを更新する手法としてメモリ上の実行プロセスに対して動的にパッチを当てる KernInst [12] が提案されている．この手法は原理的にはいかなるソフトウェア層の更新に対しても適用可能であり，また無用な性能低下もないため，高いアベイラビリティの求められる OS 等で既に用いられている．しかし，動的に適用可能なパッチを作成するのが困難であるという問題を抱え，その適用可能範囲は広くない．本論文における我々の提案手法では，論理ノード内での送受信によりソフトウェア更新時にディスクアクセスやネットワーク送受信が発生しないため，性能低下がわずかである．またデータ提供サービスに特化し分散ストレージのもつ仮想化機能と透過的データ移動機能を利用するため，複雑なパッチを構成する必要はない．ただし，提案手法は OS のようなソフトウェアに更新要求が発生した場合には適用できないため，そのような場合には KernInst のような技術や故障ノード除去機能を利用した手法を併用する必要がある．

7. むすび

本論文では論理ノードを用いて，分散ストレージシステムのアベイラビリティを低下させずにシステム内の管理ソフトウェアを更新する手法を提案した．提案手法では，分散ストレージシステムのもつストレージ仮想化機能と透過的データ移動機能を利用し，物理ノードの中に複数の論理ノードを構成する手法，及びその論理ノード間で，データに対する管理権限移譲を行う手法を提案した．そして，その論理ノード間のデータ管理権限移譲を利用して，管理ソフトウェアをデータ管理から開放する手法を提案した．これにより性能低下を抑えたまま高速にストレージ管理ソフトウェアの更新を行うことが可能となった．また，処理中においても，分散ストレージシステムのもつ耐故障性やアクセス透過性等の性質を損なわないことを述べた．更に，複製配置に制約のある Chained Declustering を用いたシステムにおいて，その信頼性を損なうことなく提案手法を適用する手法について述べた．

実験システム上に実装した提案手法のスループットを，基本的な手法である障害回復機能のみを利用した手法と比較することで，提案手法の有効性を確認した．

本手法における性能低下は，単純な手法に比べわずかであったことを確認した．また，本提案手法を有効に使用するためには，移動するデータサイズを大きく，少ない回数で移動させることが効果的であることが得られた．本手法を複数のノード内で並行に利用することで性能低下時間を減少し，短時間での保守を行えることを確認した．提案手法を用いることで，素早く性能低下の少ない無停止システム更新を行うことが可能である．

今後の課題としては，本提案手法の適用可能範囲外，特にオペレーティングシステムやファームウェアレベルのソフトウェアにおけるアベイラビリティに優れたソフトウェア更新手法の探求が肝要であると考える．あるいは，本論文ではストレージシステムのサービスは格納されたデータへのアクセスであると考えたが，近年ストレージシステムの余剰計算資源によるDBMSの一部機能委譲等のより高度なサービスをストレージシステムで行う研究[13]も行われている．このような場合の本提案手法の適用可能性に関する考察も今後の課題である．

謝辞 本研究の一部は，科学技術振興事業団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）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] 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.

[2] 森　良哉，小林　茂，金子哲夫，原　修一，"PCサーバ・クラスタ—アベイラビリティ（可用性）向上を狙って，" 情報処理，vol.39, no.1, pp.49–54, Jan. 1998.

[3] 伊藤大輔，横田治夫，"自律ディスクにおけるディスク故障時/追加時のクラスタ再構築法，" 第12回電子情報通信学会データ工学ワークショップ（DEWS2001）論文集，pp.17–24, March 2001.

[4] S. Ghemawat, H. Gobioff, and S.T. Leung, "The google file system," 19th ACM Symposium on Operating Systems Principles, pp.29–43, 2003.

[5] Y. Yasuda, S. Kawamoto, A. Ebata, J. Okitsu, T. Higuchi, and N. Hamanaka, "Scalability of X-NAS: A clusterd NAS system," Trans. IPSJ Advanced Computing Systems, vol.44, no.SIG11, pp.68–78, 2003.

[6] W. Katsurashima, S. Yamakawa, T. Torii, J. Ishikawa, Y. Kikuchi, K. Yamaguti, K. Fujii, and T. Nakashima, "NAS Switch: A novel CIFS server virtualization," IEEE Symposium on Mass Storage Systems, pp.82–86, 2003.

[7] H. Hulen, O. Graf, K. Fitzgerald, and R.W. Watson, "Storage area networks and high performance storage system," 10th NASA Goddard Conference on Mass Storage Systems and Technologies, UCRL-JC-146951, 2002.

[8] H.I. Hsiao and D.J. DeWitt, "Chained Declustering: A new availability strategy for multiprocssor database machines," Proc. Sixth International Conference on Data Engineering, pp.456–465, 1990.

[9] M.L. Lee, M. Kitsuregawa, B.C. Ooi, K.L. Tan, and A. Modal, "Towards self-tuning data placement in parallel database systems," Proc. ACM SIGMOD, pp.225–236, 2000.

[10] H. Yokota, Y. Kanemasa, and J. Miyazaki, "Fat-Btree: An update-conscious parallel directory structure," Proc. 15th International Conf. on Data Engineering, pp.448–457, 1999.

[11] S. Ajmani, Automatic software upgrades for distributed systems, Ph.D. thesis proposal, Massachusetts Institute of Technology, April 2003.

[12] A. Tamches and B.P. Miller, "Fine-grained dynamic instrumentation of commodity operating system kernels," 3rd Symposium on Operating Systems Design and Implementation (OSDI), pp.117–130, Feb. 1999.

[13] E. Riedel, G.A. Gibson, and C. Faloutsos, "Active storage for large-scale data mining and multimedia applications," Proc. 1998 Very Large Data Bases conference (VLDB), pp.62–73, 1998.

（平成15年6月14日受付，10月8日再受付）

小林　大

平14東工大・工・情報工卒．現在，同大大学院情報理工学研究科修士課程在学中．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．

渡邊　明嗣

平11東工大・工・情報工卒．平13同大大学院・情報・修士課程了．現在，同大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．

上原　年博　（正員）

昭 54 慶大・工・電気卒．昭 56 同大大学院修士課程了．同年 NHK 入局．昭 59 より NHK 放送技術研究所．デジタル VTR，サーバー型放送，自律ストレージシステムの研究・開発に従事．現在，同所主任研究員．平 4 市村学術賞功績賞，平 7 発明協会東京支部長賞など受賞．

横田　治夫　（正員）

昭 55 東工大・工・電物卒．昭 57 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．昭 61（株）富士通研究所勤務．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．主として分散インデキシング，データ工学向けアーキテクチャ，高機能ストレージシステム，ディペンダブルシステム等に関する研究に従事．日本データベース学会理事，ACM SIGMOD 日本支部評議委員，本会データ工学研究専門委員会委員長，情報処理学会，人工知能学会，IEEE，ACM 各会員．