柔軟なコンテンツ管理に向けた個々のメタデータ中のルール処理の効率化

太田　健介† 小林　大†† 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: †ohta@de.cs.titech.ac.jp, ††daik@de.cs.titech.ac.jp, †††tkobaya@gsic.cs.titech.ac.jp,
††††taguchi.r-cs@nhk.or.jp, †††, ††yokota@cs.titech.ac.jp

あらまし　近年，扱われる情報の量が増大し，それに伴うデータ管理コストの増加が問題になってきている．また，システム上には扱いの異なる様々なコンテンツが存在するが，個々のコンテンツの特徴が管理に反映されていない．我々はこれまでに高機能ストレージの計算資源によるストレージの自律管理に関する手法を提案している．本稿では、各コンテンツに対応したメタデータに ECA ルールで処理を記述することによって，個々のコンテンツに合った管理を高機能ストレージ上の大量コンテンツに対して実現できることを示す．コンテンツ毎にルールを記述できるため，その数は格納コンテンツ数が増えるに従い増加する．それに伴い，イベント発生後のコンディション評価時間の増大が無視できない問題となる．そこで、ECA ルール発火制御時のルール処理コストを下げるため，実行ルールの絞り込みを行う．また，高機能ストレージとして提案している自律ディスクへ実装し，その有用性を示す．

キーワード　メタデータの管理, コンテンツ処理, 並列・分散 DB, 情報ライフサイクルマネジメント

Improvement on Processing Rules Stored in Individual Metadata for Flexible Contents Management

Kensuke OTA†, Dai KOBAYASHI††, Takashi KOBAYASHI†††, Ryo TAGUCHI††††,
and Haruo YOKOTA†††,††

† Department of Computer Science, Faculty of Engineering, Tokyo Institute of Technology
†† Department of Computer Science, Graduate School of Information Science and Engineering, Tokyo Institute of Technology
††† Global Scientific Information & Computing Center, Tokyo Institute of Technology
†††† NHK Science & Technical Research Laboratories
E-mail: †ohta@de.cs.titech.ac.jp, ††daik@de.cs.titech.ac.jp, †††tkobaya@gsic.cs.titech.ac.jp,
††††taguchi.r-cs@nhk.or.jp, †††, ††yokota@cs.titech.ac.jp

Abstract　Recently, the data management cost becomes a big problem because the amount of data to be treated in a system becomes very huge. The characteristics of individual contents is not reflected in the management especially, although the characteristics are widely vary and the contents should be treated in different manners with considering the characteristics. To attack the problem, we have proposed the technique for autonomous management based on ECA rules in metadata of the contents by using system resource of a high functional storage. In this paper, we study the feasibility of treating a large number of ECA rules corresponding to the number of contents stored in a storage system. The increase contents incurs increase of the cost for evaluating conditions in the rules. To reduce the cost of processing the rules, we construct discrimination networks for each event. We implement the method in the autonomous disks, a high functional storage we proposed, and evaluate the efficiency of the method.

Key words　management of metadata, processing of contents, parallel・distributed DB, Information Lifecycle Management

1. はじめに

近年，人々によって創出され，扱われるデータ量が爆発的に増大していることが明らかになっている．それに伴い，ストレージ上でのデータ管理コストの増大が問題になってきている [1]. 特に，多種多様な大量のデーが格納されたストレージシステム上には扱いの異なる様々なコンテンツの管理コストが高い．そのため，効率のよい管理方法が求められている．

それらの多様なコンテンツを管理するために，コンテンツ毎に情報を付与することで，それぞれの特徴を反映したデータ管理を自動的に行うことが可能となる．また近年，ストレージシステムに計算資源を集約し，自律的なデータ管理機能や DBMS の一部機能等の高度なデータ処理を可能にする高機能ストレージシステムに関する様々な研究が行われている [2]〜[6]．大量のコンテンツに対し自律的な管理を行う上でそのようなシステムは用な基盤となる．

既存のコンテンツの特徴を活かしたストレージ管理として，情報ライフサイクルマネジメント（Information Lifecycle Management: ILM）[7]〜[9] や，ストレージ間の移動ポリシーを設定できるシステムがある．これらはシステム側にポリシーを記述し，管理を行っている．しかし，データ量の増大とともにデータの多様性も増加してきているので，ストレージ資源を効率よく扱うためシステム単位のポリシー規定だけではなく，コンテンツ毎に特化した処理や制御ポリシーを詳細に規定することは有用である．

本稿では，より細い粒度での制御を実現するために，個々のデータに対する処理をコンテンツ単位に指示する手段の一つとして，それぞれのコンテンツにメタデータを持たせる方法を採用する．また，そのメタデータの一部として制御ルールを記述し，それを随時システムが読み込み，状況に応じた制御を行うことで制御の細粒度化を図る．また，制御ルールはアクティブデータベース [10], [11] において用いられる ECA(Event-Condition-Aciton) ルールを用いて記述する．ECA ルールは ECA アーキテクチャの枠組みで表現され，あるイベントに対して，アクションの発火条件とそのアクションを宣言的に記述する．コンテンツ管理においては，発火条件に，コンテンツやストレージの状況に関する条件の設定をすることで，特定コンテンツやストレージの状態が変化するタイミングでルールを発火させることができる．ECA ルールで処理を記述することで格納されているコンテンツやストレージの状態を考慮したルールを記述できるため，柔軟なコンテンツ管理が実現できる．

しかし，ストレージ上のコンテンツ数増加により，各コンテンツに対応した制御ルール数も増大する．それに伴い，各ルールの発火条件評価にかかる時間の増大が無視できない問題となる．大量の ECA ルールの発火条件評価は非常にコストが高い [12] ので，コンテンツ毎の細粒度のルール制御を実現するために，この発火条件評価を高速化をする必要がある．

本研究では，イベント発生時のルール発火作業を短縮するために，各イベントに対して，部分的な条件評価を実行し，発火候補ルールを絞込む．条件評価済みのルール集合をノードとした弁別ネットワークを構成することで，不必要な条件評価とイベント発生時の条件評価処理自体の時間の短縮を行う．また，高機能ストレージとして我々が提案している自律ディスク [2]〜[4] に提案手法を実装し，その有用性を示す．

以下に本稿の構成を述べる．まず 2. でコンテンツへのルールの記述方法について述べる．次に 3. で，本稿で考察するルール処理方法について述べた後，4. で，提案するルール処理の効率化手法について述べる．5. では，提案手法の有効性を自律ディスク上の実験を通して考察を行う．6. で関連する研究について述べ，7. で全体のまとめと今後の課題について述べる．

2. コンテンツへの処理記述

コンテンツの状態やアクセス傾向に応じた処理を実現するために，格納されている全てのコンテンツに対して同一の処理を施すだけでは不十分な場合があり，異種コンテンツに対する処理の差別化を図る必要がある．その場合，コンテンツ毎の処理記述の付与は有用である．

2.1 メタデータと処理記述

本稿では，コンテンツ毎のメタデータに対して，そのコンテンツの特徴を生かした制御処理を ECA ルールによって記述することを考える．

2.1.1 メタデータ

コンテンツ毎に記述するメタデータにおいて，そのデータに関する情報は，POSIX1003.1 仕様や NFS バージョン 3 [13] などの標準的なファイルシステムがサポートしているファイル属性に加え，ユーザ等が拡張的に定義するコンテンツをより特徴化させる項目を追加することも考慮する．

さらに，我々は同様に ECA ルールで記述した制御処理をメタデータとして格納する．ルールを記述する際は，定義されているメタデータの種類やシステムが実行できるコマンドについて既知であることを前提とする．メタデータへのルール記述が，ユーザの手で直接記述することが困難である場合，アプリケーションを介して記述することも想定される．また，ユーザが記述した曖昧な表現を適切な数値や記述形式にコンバートするためのコンパイラを想定することも可能である．

2.1.2 ECA ルール

ECA ルールは，あるイベントに対して，アクションの発火条件とそのアクションの内容を宣言的に記述するものであり，Event・Condition・Action の 3 要素から構成される．各要素には，以下に示すような記述がなされる．

Event トリガされるルールの原因となるイベントを記述する．

Condition ルールがトリガされるたびにチェックされる条件を記述する．条件評価はアクションが実行される前に行われる．

Action トリガされたルールのうち，条件評価の結果が真であるときに実行 (発火) するイベントを記述する．

2.2 ECA ルールのコンテンツ管理への適用

ECA ルールを用いて記述した制御ルール群をコンテンツのメタデータの一部として付与する．以下に ECA ルールをスト

レージ管理に適用した場合に，各要素で記述できる項目について述べ，コンテンツへの処理記述例をいくつか示す．

Event には， insert，delete などの基本的な操作をはじめ，タイマ割り込みやアプリケーションで定義したイベントの記述が可能である．Condition には，コンテンツのメタデータとして記述された情報に対する制約条件や，検索結果の有無などを記述できるほか，イベント発生時に決定するパラメータについての条件も記述可能である．Action には，高機能ストレージが提供している insert，delete，update といった基本的なコマンドを記述可能である．ルールの下に，システムに触れられるコマンド群が存在するため，記述可能な処理の範囲は広い．

ルール記述例 1　Web のコンテンツなどの外部公開を行うルールである．例えば，その Web コンテンツで使用されている画像の著作権や契約の関係などで期間を限定して公開が許可されている場合，公開日になった時，閲覧権限の変更を行う場合などに利用される：

Event　2005/12/31 23:00 になった時

Condition　外部からの閲覧権限が与えられていない場合

Action　アクセス制限を解除し外部から閲覧できるようにする

ルール記述例 2　コンテンツ削除時にに関連するコンテンツも同時に削除するルールである．例えば，映像コンテンツが削除された場合に，それに関連する複数の音声コンテンツや字幕テキストなどでアクセス頻度が閾値以下のものも同時に削除する場合などに利用される：

Event　コンテンツが削除される時

Condition　削除対象のコンテンツに関連する音声コンテンツ，または字幕テキストでアクセス頻度が閾値以下のものが存在する場合

Action　関連コンテンツを削除する

このように，ユーザによるメタデータの定義とコンテンツの制御ルールの記述により，柔軟なコンテンツ管理が可能である．例えば，モバイル環境におけるユビキタスストレージシステムの研究 [14] が行われているが，メタデータの定義とルール記述を適切に行うことにより本記述方式の適用が可能である．

2.3　想定システム

柔軟なルール記述を行うにあたり，実装するシステムとして，ストレージ自身がそれらの内部での処理を行う命令群を持つような高機能ストレージを想定する．内部命令群の組み合わせで様々な処理を行うことが可能であれば，コンテンツ管理を行う上で柔軟な記述が可能となる．

2.3.1　自律ディスク

自律ディスク [2]〜[4] のように，実際にシステムに命令を出すストレージに近いレイヤーのコマンドとそれらを組み合わせることでルールとして実行できる高機能ストレージであれば，より効果的なストレージ管理が可能となる．自律ディスクはネットワーク環境でクラスタを構成することを前提としている．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．

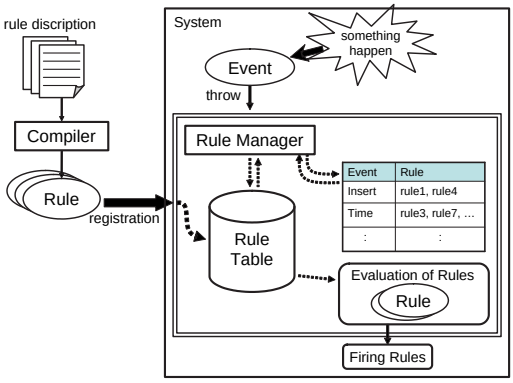

図 1　全走査によるルール処理方法

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．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，異種性といったストレージ管理を，ストレージシステム側で行うことが可能である．

自律ディスクはそれらの様々な機能を実現するために，ECA ルールによるルール記述を採用している．これはユーザレベルのシステムの柔軟性を取り入れるためである．ユーザは ECA ルールを記述することによって，様々な機構の動作戦略をユーザの目的に沿うように最適化できる．例えば，自律ディスクでは前述のとおり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クラスタを構成できるため，ルールに適切な記述を行うことで適切なコンテンツをクラスタ間で容易に移動することが可能である．また，システムによる自律的なストレージ管理も ECA ルールで記述されている．我々は，システムルールとコンテンツ制御ルールを同列に扱うことを目標としている．

3.　ルール処理

ECA ルールで記述されたルールを処理するためには，図 1 に示すように，システム側にルールの登録と実行を行うルールマネージャが実装されていることが前提となる．ルールの登録時には，ルールマネージャによりルールの登録作業が行われる．このとき，イベントとそのイベントに関連し発火する可能性のあるルールとのマッピングをあらかじめ行っておく．

イベント発生時には，ルール登録時に行われたマッピングを用いて発生したイベントに関連して発火する可能性のあるルール群を取得し，それぞれのルールについて発火条件であるコンディションの評価を行う．コンディションの評価結果が真となれば，ルールが発火され，アクションに記述された処理を実行する．

3.1　全走査によるルール処理方法の評価

ルールの全走査によるルール処理方法では，イベント発生時にイベントに関連した全てのルールのコンディション評価を行うため，コンテンツ更新時には，メタデータの更新以外に特別な処理は実行しない．

イベント発生時の発火ルール特定にかかる処理ステップ数 C_E

は，式 (1) となる．

$$C_E = f_E \cdot O(\|Rule_{(E)}\|) \quad (1)$$

ここで，f_E はイベント発生頻度を，$\|Rule_{(E)}\|$ は発生したイベントに関連して発火する可能性のあるルール数を表す．

コンテンツ更新処理ステップ数 C_M は式 (2) となる．

$$C_M = f_M \cdot \alpha \quad (2)$$

ここで，f_M はコンテンツの更新頻度を，α はメタデータの更新にかかる処理ステップ数を表す．

したがって，全走査によるルール処理における発火ルール特定のための処理ステップ数 C_T は，式 (3) となる．

$$C_T = C_E + C_M \quad (3)$$

ただし，我々の想定環境では，登録されている全ルール数を $\|Rule\|$，m を正数とすると，$\|Rule\| = m \cdot$(コンテンツ数) であり，格納コンテンツ数が増加するに従い，式 (1) の処理ステップ数は線形に増加するため，(コンテンツ数) = 20,000,000 といった現在の現実的な使用環境下においてルール処理時間の短縮が重要となる [15]．

3.2 全走査によるルール処理方法の問題点

コンテンツ毎に付与したメタデータからルールを実行する上で最もコストの高い処理は，大量のルールに対するコンディション評価である．あるイベントに関連するルールのコンディション評価は，そのイベントが発生するたびに関連する全てのルールに対して行われる．ストレージ内のコンテンツは，新たなコンテンツの挿入や不要なコンテンツの削除，格納されているコンテンツに対する更新処理等により変化する．しかし，ストレージ内のコンテンツの最新の状態を知るために，イベント発生の度に全てのルールを走査することはコストが高い．また，それらの中には発火しない多くのルールについてのコンディション評価処理も含まれている．

4. ルール処理の効率化

本稿では，大量のルールに対するコンディション評価の高速化のために，ルールを弁別ネットワークに構築することで事前に評価可能なコンディションについての評価をイベント発生前に実行し，発火候補ルールの絞込みによってイベント発生時のルール処理時間を短縮する手法を提案する．

4.1 コンディションの分割

コンディションを，イベント発生とは関係なく評価可能な部分（PEC：Previously Evaluable Condition）と，イベント発生時に渡されるパラメータに対する条件で構成されイベント発生時に評価すべき部分（REC：Runtime Evaluable Condition）に分割する．既に格納されているコンテンツに対する条件として PEC は存在し得る．また，ECA ルールのようにイベントの概念が存在する場合，イベント発生の際に初めて決定するパラメータが存在するため，REC が発生し得る．前述のルール記述例 1 における「外部からの閲覧権限が与えられていなかった場合」という条件は，PEC であり，ルール記述例 2 における「削除対象のコンテンツに関連する音声コンテンツ，または字幕テキストが存在する場合」という条件は，REC である．

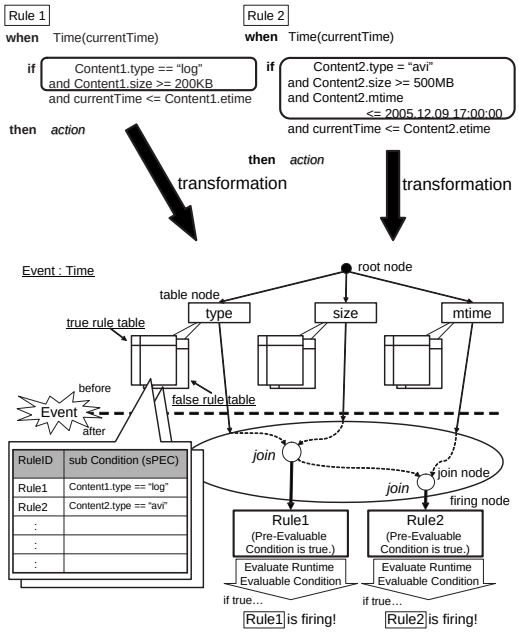

図 2　分割コンディションを用いた事前条件評価

また，PEC は，1 つまたは複数の条件式で構成されることから，PEC をさらに，最終更新時刻 (図 2 における mtime) 等の単一のメタデータ項目を含む部分コンディション sPEC（sub-PEC）に分割することができる．

そこで，両者をイベント発生時に評価するのではなく，コンテンツ更新時に sPEC の評価を行い，イベント発生時に PEC の評価が真となるルールを高速に選出することで，REC の評価を実行する発火候補ルールを削減し，ルール発火時間を短縮する．

4.2 分割コンディションを用いた事前条件評価手法

以下では，まず提案手法の概要について述べ，弁別ネットワークに変更が加えられるイベント発生時，ルール登録・削除時，メタデータ更新時の動作について述べる．

4.2.1 概　　要

本手法では，イベント発生時に発火候補ルール特定のために，登録されたルールを用いて弁別ネットワークを構築する．図 2 は弁別ネットワークと，そのイベント発生時の処理の概要を示したものである．弁別ネットワークは，ルートノード (root node)，テーブルノード (table node)，結合ノード (join node)，発火ノード (firing node) の各ノードと，これらを結ぶアークから構成される (図 2 を参照)．

ルートノードでは，登録されるルールのコンディション部分の分割が行われる．

テーブルノードは，メタデータの種類ごとに生成され，暫定発火候補ルールテーブル (true rule table) と発火対象外ルールテーブル (false rule table) を持ち，それぞれには sPEC の評価が真となるルールと，sPEC の評価結果が偽のルールが登録されている．

結合ノードでは，テーブルノードまたは他の結合ノードから

渡される 2 つの暫定発火候補ルール群の結合を行う．結合ノードは，ルール登録・削除時に生成・削除される．

発火ノードでは，結合ノードから渡されたルールの REC の評価とルールの発火が行われる．

4.2.2 イベント発生時の動作

弁別ネットワークはイベント別に生成されているため，発生したイベントに対応する弁別ネットワークにおいて，

（1） それぞれのテーブルノードから暫定発火候補ルールセットを結合ノードに渡す
（2） 結合ノードにおいて，2 つのルールセットの結合操作を行い，両セットに含まれるルール群を抽出する
（3） 抽出されたルール群を発火ノードに渡す
（4） 発火ノードにおいて，渡されたルール群について REC の評価を行い，評価結果が真となればアクションの処理を実行する

例えば，イベント発生時に図 2 の Rule1 が，テーブルノード”type” と”size” の結合を行う結合ノードにおいて抽出されたルール群に含まれていれば，発火ノードに Rule1 が渡され，REC の評価が真となれば発火される．Rule2 についても同様である．

4.2.3 ルール登録・削除時の動作

ルール登録時，

（1） ルートノードにおいて，ルールのコンディションが PEC と REC に分割し，PEC を sPEC に分解し，sPEC をその sPEC をコンディションに含むルールとマッピングし，sPEC に含まれるメタデータ項目により，該当するルールテーブルへ渡す
（2） テーブルノードにおいて，渡された sPEC の評価を行い，評価結果が真ならば，暫定発火候補ルールテーブルへ，偽ならば発火対象外ルールテーブルへ登録する
（3） イベント発生時にテーブルノードから渡されるルールセットを結合するための結合ノードを生成する

例えば，図 2 では，Rule1 登録時にはテーブルノード”type” と”size” にそれぞれ”Content1 のファイルタイプが log である” という sPEC と，” Content1 のファイルサイズが 200KB 以上である” という sPEC が渡され，それぞれについての評価が行われ，2 つのうちどちらかのテーブルに登録される．また，“type” と”size” の結合を行う結合ノードが無い場合は新たに生成する．

ルール削除時は，手順 1 は同様で，手順 2 において，渡された sPEC とルールのマッピングを登録されているテーブルから削除する．また，対応する結合ノードを削除する．

4.2.4 コンテンツ更新時の動作

更新されたコンテンツに関する条件の sPEC が存在した場合，その sPEC を含むテーブルノード中の暫定発火候補ルールテーブル・発火対象外ルールテーブルのどちらかにおいて，sPEC を再評価し，その結果によりテーブルへの再登録を行う．

これは，sPEC を 2 種類のテーブルで管理することで，更新の度に変更する sPEC をコンディションに含むルールの再構築を行う必要がなく，更新処理時間が削減できる．

4.3 分割コンディションを用いた事前条件評価手法の評価

提案手法について更新処理ステップ数及びイベント発生時の発火可能ルールの決定にかかる処理ステップ数について評価する．

イベント発生時，sPEC の評価が真であるルールの結合により PEC が真である発火候補ルールを選出し，発火ノードにおいて，REC の評価を行う．イベント発生時のルール処理ステップ数 $\widehat{C_E}$ は，式 (4) となる．

$$\widehat{C_E} = f_E \cdot (O(\|Rule_{(E)}\| \cdot k_{TRUE}) + S_{JOIN}(\|Rule_{(E)}\| \cdot k_{TRUE})) \quad (4)$$

ここで，$S_{JOIN}(\|Rule_{(E)}\| \cdot k_{TRUE})$ は結合操作の処理ステップ数を表し，$\|Rule_{(E)}\| \cdot k_{TRUE}$ の関数となる．k_{TRUE} は発生イベントに関連するルール内の発火候補ルールの割合を表す．

コンテンツの更新時，イベントごとの弁別ネットワークにおいて，テーブルノード内の sPEC に対する更新処理を行う．コンテンツ更新処理ステップ数 $\widehat{C_M}$ は式 (5) となる．

$$\widehat{C_M} = f_M \cdot (\alpha + O(\|Event\|)) \quad (5)$$

ここで，$\|Event\|$ はイベントの種類数を表す．また，第 2 項はテーブルノードが持つ発火候補ルールテーブルと発火対象外ルールテーブルには，ハッシュによるインデックスがあるため，その探査ステップ数を $O(1)$ とした．

したがって，提案手法における発火ルール特定のための処理ステップ数 $\widehat{C_T}$ は，式 (6) となる．

$$\widehat{C_T} = \widehat{C_E} + \widehat{C_M} \quad (6)$$

以上より，更新時には式 (2) と式 (5) から，全走査によるルール処理手法の方が処理ステップ数が小さく，イベント発生時には式 (1) と式 (4) から，提案手法の方が処理ステップ数が小さい．

ただし，更新操作以外の原因によるイベント発生の頻度を f_{other} とすると，$f_E > f_M$ $(\because f_E = f_M + f_{other})$ であり，更新頻度よりもイベント発生頻度の方が大きく，イベント発生時のルール処理ステップ数を減少させることが重要となる．また，ルール数の増加に伴い PEC の評価が真とならないルール数も増加するため，ルールの全走査による処理方法では，発火しないルールのコンディション評価を実行することになり，提案手法に比べ，ルール処理ステップ数は相対的に大きくなることが予想される．

これらより，本手法が，ルール数の増加による影響が少なく，コンテンツが大量に格納されている環境下において，ルールによるコンテンツ管理を実現する上で有用な手法であることが予想される．

5. 実　　験

本手法の有効性を示すために，高機能ストレージとして提案されている自律ディスク [2]〜[4] に提案手法を実装し実験を行う．

5.1 実験環境

実験は，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の模擬実装上で行う．これは Linux クラスタ上に Java を

表 1　ストレージノード諸元

CPU	AMD Athlon XP-M1800+ (1.53GHz)
Memory	PC2100 DDR SDRAM 1GB
Disk	TOSHIBA MK3019GAX (30GB，5400rpm，2.5inch)
OS	Linux 2.4.20
Java VM	Sun J2SE SDK 1.5.0_03 Server VM

```
when    time(currentTime)

    if          this.item == “video”
       and  o1.item == “audio”
       and  this.title == o1.title
       and  currentTime <= this.etime
       and  currentTime <= o1.etime

then    action
```

図 3　実験に用いるルール

用いて模擬実装されている．今回の実験では表 1 に示す構成の PC8 台と十分なバックボーン性能を持つネットワークスイッチを用いて，実験環境を構成した．

5.2　想定運用シナリオ

ここでは，提案手法によるイベント発生時のルール処理時間の比較と，提案手法を導入したことによるコンテンツ更新時の遅延時間の測定を行う．実験で用いるルールを設定するために，次のような状況を想定する．ストリーム形式の様々なデータが保存されている場合，できる限りそれらのデータを 1 つのコンテナフォーマットに格納して保存するような状況を考える．定期的な更新チェックのイベント発生時に，それぞれのデータの有効期限が過ぎていない同一タイトルの映像データ，音声データがあった場合，コンテナフォーマットにまとめるための編集用のストレージに移動させるなどのアクションを行うルールを設定する．図 3 にそのルールの例を示す．この例では，発火条件の上 3 つが PEC に相当し，その一つ一つが sPEC であり，下 2 つが REC に相当する部分である．

今回，ルールにおけるアクションの内容の違いによる処理時間のばらつきを避けるためにアクションの内容は特に記述しないものとした．

5.3　ルール処理時間の比較

イベント発生時のルール処理時間を，イベントに関連するして発火する可能性のあるルールのうち PEC の評価が真であるルールの割合を変化させて測定し，ルールの全走査による処理方法と提案手法についての比較を行った．

PEC 評価による絞込みの効果を示すために，REC の評価は，全て真とした．また，この比較を登録ルール数が 100，1000，3000，5000，8000，10000 の場合で実行し，登録ルール数によるルール処理時間の変動を測定した．ルールは 1 コンテンツに 1 つのルールが付与されているものとする．

図 4，5 は，イベント発生時に発火する可能性のあるルールに対し，PEC の評価が真であるルールの割合が 10 %，5 %の場合のルール処理時間を示している．

また，提案手法のルール処理時間はルールのコンディション

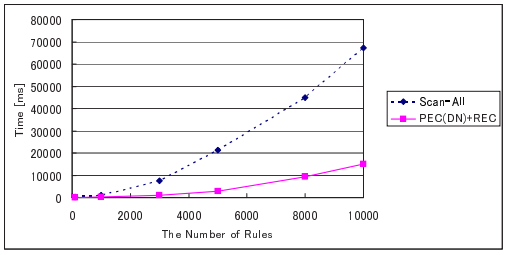

図 4　発火するルールの割合が 10 %の場合のルール処理時間の比較

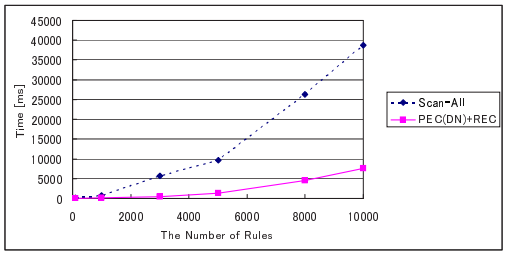

図 5　発火するルールの割合が 5 %の場合のルール処理時間の比較

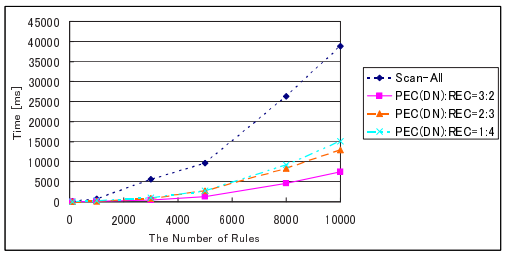

図 6　PEC,REC の比率を変化させた場合のルール処理時間の比較

内の PEC，REC の比率に依存するため，コンディション内の PEC，REC の比率を変化させたときのルール処理時間について測定を行った．

図 6 は，PEC の評価が真となるルールの割合が 5 %の場合に，PEC，REC の比率が 3:2，2:3，1:4 のルールについてのルール処理時間を示している．実験で用いたルールは図 3 のルールの PEC と REC の比率を意図的に変えたものを使用した．PEC:REC = 3:2 のルールは図 3 と同じものである．

5.4　コンテンツ更新処理時間の比較

提案手法を実装したことによるコンテンツ更新時の遅延時間を測定した．今回，測定を行った操作は，コンテンツの挿入・更新・削除の 3 つである．各操作が行われることにより，各コンテンツに付与されているメタデータの情報が変更されるため，弁別ネットワークの更新が必要になる．

図 7 は，1 つのクライアントから自律ディスクに対して逐次的に 1000 回のコンテンツの挿入・更新・削除をそれぞれ行ったときの 1 回当たりの平均処理時間を測定した結果を示している．

5.5　考　　察

5.3 と 5.4 で行った実験の考察を，発火ルール特定のための処理ステップ数に関する評価より得られた式 (1)〜(6) との比較を交えて行う．

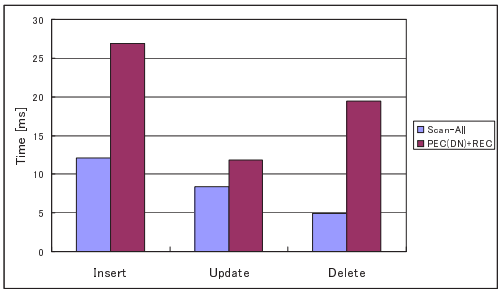

図 7　挿入・更新・削除操作の平均処理時間の比較

5.5.1　ルール処理時間に関する考察

5.3 の実験では，常に提案手法の方がイベント発生時のルール処理時間が短く，本手法が有効であることが分かった．

全走査によるルール処理手法では，ルール数の増加に伴いルール処理時間は増加している．これは，イベントに関連する全てのルールについてコンディションの評価を実行しているためである．また，この結果は式 (1) が示すイベント発生時のルール処理ステップ数がルール数 $\|Rule_{(E)}\|$ に比例することにも合致する．

提案手法でも，ルール数の増加に伴い処理時間は増加している．これは，全ルール数の増加により，k_{TRUE} に関わらず PEC の評価が真となるルール数自体が増加していることと，結合処理時間の増加が原因である．これらは，式 (4) の第 1 項がルール数 $\|Rule_{(E)}\|$ に比例し，第 2 項がルール数 $\|Rule_{(E)}\|$ の関数であることに合致する．

また，PEC，REC の比率を変化させた場合の実験では，REC 評価の高速化は行っていないため，コンディションに含まれる REC の増加により処理時間は増大する．しかし，処理時間の短縮の要因はルールの絞込みによる処理ルール数の減少によるところが大きいため，全走査によるルール処理時間を本手法のそれが上回ることはない．

5.5.2　コンテンツ更新処理時間に関する考察

更新操作では，提案手法の実装に伴う遅延時間は 3ms と最も小さかった．これは，sPEC を含むテーブルノード中の暫定発火候補ルールテーブル・発火対象外ルールテーブルにおいて，sPEC の再評価と，テーブルへの再登録を行うだけでよいためである．これは，全走査によるルール処理方法のコンテンツ更新処理時間が式 (2) で表されるのに対し，提案手法は式 (5) に表されるように第 2 項の処理が加わっていることにも合致する．

コンテンツの挿入・削除が行われる場合は，処理時間はそれぞれ 16ms,15ms 増加した．これらは，テーブルノードの更新だけでなく同時にルールの登録・削除，結合ノードの生成・削除も行われるためである．

6.　関 連 研 究

ストレージ中のデータをコンテンツ単位で扱う仕組みとして，ブロック単位で管理されていたファイルをより抽象度の高いオブジェクト単位で管理を行う OSD （Object-Based Storage Device）[16] が挙げられる．メタデータの付与や制御ルールの記述といった点において，本研究が提示するデータの管理方法との親和性がよい．

ECA ルールを用いたオブジェクト指向データベース管理の研究の一部には, [17]〜[19] があり, [17] では，ECA ルールを導入したデータベース管理とイベント検出手法が提案されている．また, [18], [19] では，オブジェクト指向データベースにおける ECA ルールによるデータベース管理システムのアーキテクチャと実装について述べられている．

ポリシーベースの ILM に関する研究としては [7]〜[9] などがある．[9] では，IBM が導入した大規模なメインフレームコンピュータシステムのためのポリシーベースの DBMS 手法である DFSMS （DataFacility Storage Management Subsystem）の概要について述べられている．[7] は格納データの処理についてのストレージ全体に適応されるポリシーを規定し，集中管理型システムによって ILM の自動化を実現させるための機能の提案と実装について述べられている．しかし，それぞれのポリシーは，ストレージに格納されている全てのコンテンツを対象としたものであるため，格納されている個々のコンテンツに関する状態やアクセス傾向を細かく反映した制御を行うことは難しい．[8] では, [7] の実際の実装と，ILM のためのデータ移動により通信帯域が占有されることによるユーザ操作への負荷の軽減のための通信レートの制御手法の提案と評価を行っている．本稿では言及していないが, [8] で述べられている通信レートの制御について検討することは必要である．

また，プロダクションシステム [20] などのルールベースシステムなどでは，ルールの条件評価を高速に行うために Rete アルゴリズム [21] が広く用いられている．大量のデータの中からルール条件に一致するデータ集合を高速に検索することが要求される．そのため，ルールベースシステム等に用いられるルールエンジンでは Rete アルゴリズムに基づいた Rete ネットワークを代表とする弁別ネットワークが利用されてきた．

Rete アルゴリズム [21] による条件評価を採用しているプロダクションシステムにおいて，ワーキングメモリを positive set と negative set に分割するといった提案 [22] はなされているが，実際にコンテンツ単位でのルールと組み合わせた場合の評価を行うことには意味がある．

また，Rete アルゴリズムは複雑なルールに対して大量のコンテンツが格納されている状況で，各コンテンツのメタデータに対するルールの条件評価の高速な実行に適しているが，本手法では，コンテンツ数以上にルールが存在するような状況で弁別ネットワークによりルールの条件評価を高速に行うことを目的としている．

我々のこれまでの研究において，コンテンツにメタデータとして時間情報と制御ルールを付与することでコンテンツ管理を実現するための手法 [23] を提案しているが，時間以外のイベントによるルールの起動は考慮されていなかった．

7.　まとめと今後の課題

細粒度のコンテンツ管理実現のために，個々のコンテンツに対してメタデータを付与し，各コンテンツに対する制御ルールを記述することで，ルールに基づいた柔軟なコンテンツ管理を

高機能ストレージ上で実現する手法を提案した．

また，コンテンツ毎にルールを記述した場合，ルールの発火条件の多様であり条件を満たさないルールも多く含まれる場合にも，イベント発生の度に実行される大量のルールのコンディション評価のために全てのルールを走査しなければならないのはコストが高い．そこで，本稿では，ルールのコンディションを PEC，REC に分割し，PEC 評価を高速に行うための弁別ネットワークを構築し，それを用いてコンテンツ更新時に PEC を評価することにより，イベント発生時の発火候補ルールの絞込みと評価すべき条件の削減により，ルール処理を高速化する手法を提案した．さらに，それらの手法を我々が提案する高機能ストレージである自律ディスクに実装し，実験により提案手法とルールの全走査による方法との比較を行った．

その結果，イベント発生時ルール処理時間は短縮され，提案手法の実装に伴うコンテンツ更新処理の遅延時間は未実装時と比較して許容範囲であることを確認した．また，実装するシステムとして自律ディスクを用いたが，高機能ストレージの性能を有するシステムであれば，同等の実装が可能である．

今後の課題として，今回はイベント発生時の発火候補ルールの絞込みによるルール処理時間の短縮に焦点を当てたため，イベントが発生したストレージ以外との通信が必要なルールについての検討や考察を行わなかったが，分散ストレージ環境におけるアクティブデータベースのルール処理に関する研究も行われているため [24]，ネットワークを介するルールについての検討が必要である．また，REC 評価の高速化についても検討すべきである．

また，大量のルールを自動記述する方法等の検討が必要がある．さらに，ルール自体の多様化により，ルールの実行により更なるイベントが発生するような連鎖反応的に ECA ルールが発生し，ストレージ資源の浪費によるユーザへの不利益が考えられる．このようなルールの停止性問題の解決や，矛盾する発火条件を持つルールの判定のためにルールの検証技術を導入が必要である．

謝　　辞

自律ディスクの実装コードに関して東京工業大学の吉原朋宏氏に助言を頂いた．ここに感謝の意を表します．また本研究の一部は，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 (SRCs)，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] Jim Gray (Microsoft Research). The Data Avalanche: Reducing Information Overload, November 2005. SIGMOD Chapter, U. Tokyo, Tokyo, Japan, 16 November 2005.

[2] 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.

[3] 鈴木裕通 , 横田治夫. 並列ディレクトリ構造 Fat-Btree における負荷分散の手法とその実装. 第 11 回データ工学ワークショップ論文集，DEWS2000，4B-4. 電子情報通信学会, Mar. 2000.

[4] 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.

[5] 合田和生 , 喜連川優. データベース再編成機構を有するストレージシステム. 情報処理学会論文誌データベース, Vol. 46, No. SIG(TOD 26), pp. 130–147, 2005.

[6] Christos Faloutsos Erik Riedel, Garth Gibson. Active Storage For Large-Scale Data Mining and Multimedia. In *In Proc. 24th Int. Conf. on Very Large Databases*, pp. 62–73, 1998.

[7] Mandis Beigi, Murthy V. Devarakonda, Rohit Jain, Marc Kaplan, David Pease, Jim Rubas, Upendra Sharma, and Akshat Verma. Policy-Based Information Lifecycle Management in a Large-Scale File System. In *POLICY*, pp. 139–148. IEEE Computer Society, 2005.

[8] Akshat Verma, David Pease, Upendra Sharma, Marc Kaplan, Jim Rubas, Rohit Jain, Murthy V. Devarakonda, and Mandis Beigi. An Architecture for Lifecycle Management in Very Large File Systems. In *MSST*, pp. 160–168. IEEE Computer Society, 2005.

[9] Lyn L. Ashton, Edward A. Baker, Arthur J. Bariska, Erika M. Dawson, Ruth L. Ferziger, Stanley M. Kissinger, Terri A. Menendez, Sanjay Shyam, Jimmy P. Strickland, Dave K. Thompson, Glenn R. Wilcock, and Mike W. Wood. Two decades of policy-based storage management for the IBM mainframe computer. *IBM Systems Journal*, Vol. 42, No. 2, pp. 302–321, 2003.

[10] Umeshwar Dayal. Active database systems. In *Proc. of 3rd Intn'l Conference on Data and Knowledge Bases*, pp. 150–169, 1988.

[11] Norman W. Paton and Oscar Díaz. Active database systems. *ACM Comput. Surv.*, Vol. 31, No. 1, pp. 63–103, 1999.

[12] Anoop.Gupta. *Parallelism in Production Systems*. Morgan Kaufmann, 1987.

[13] B. Callaghan, B. Pawlowski, and P. Staubach. *NFS Version 3 Protocol Specification*. Sun Microsystems, Inc., June 1995. http://ftp.riken.go.jp/pub/internet-doc/rfcs/rfc1813.txt.

[14] 藤原勤 , 宮崎純 , 植松俊亮. 自律ディスクによる広域分散ストレージ の静的な性能解析. 信学技報. DE2005-104(2005-07), pp. 227–232. 電子情報通信学会, Jul. 2005.

[15] Inc.Cluster File Systemss. "lustre". http://www.lustre.org/.

[16] Vishal Kher and Yongdae Kim. Decentralized Authentication Mechanisms for Object-based Storage Devices. In *IEEE Security in Storage Workshop*, pp. 1–10. IEEE Computer Society, 2003.

[17] Eman. Anwar, L. Maugis, and S. Chakravarthy. A new perspective on rule support for object-oriented databases. In *SIGMOD '93: Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pp. 99–108, New York, NY, USA, 1993. ACM Press.

[18] Sharma Chakravarthy, V. Krishnaprasad, Z. Tamizuddin, and R. H. Badani. ECA Rule Integration into an OODBMS: Architecture and Implementation. In Philip S. Yu and Arbee L. P. Chen, editors, *ICDE*, pp. 341–348. IEEE Computer Society, 1995.

[19] Sharma Chakravarthy. Sentinel: an object-oriented DBMS with event-based rules. *SIGMOD Rec.*, Vol. 26, No. 2, pp. 572–575, 1997.

[20] Toru Ishida. Parallel rule firing in production systems. *IEEE Trans. Knowl. Data Eng.*, Vol. 3, No. 1, pp. 11–17, 1991.

[21] Daniel P. Miranker. *TREAT : a new and efficient match algorithm for AI production systems*. Research notes in artificial intelligence. Pitman, Morgan Kaufmann, London, San Mateo, Calif, 1990.

[22] Fawzia Derrough Darche. Set differentiation: a method for the automatic generation of filtering algorithms. In *Knowledge-Based Software Engineering Conference, 1996., Proceedings of the 11th*, pp. 134–143. Digital Object Identifier 10.1109/KBSE.1996.552831, 1996.

[23] 山口宗慶 , 渡邊明嗣 , 小林大 , 田口亮 , 林直人 , 上原年博 , 横田治夫. 分散ストレージにおける情報ライフサイクルの効率的管理 . 信学技報. DE2004-78(2004-07), pp. 177–182. 電子情報通信学会, Jul. 2004.

[24] 宮崎純, 横田治夫. 並列アクティブデータベースにおける弁別ネットワークの楽観的な動的最適化方式. 電子情報通信学会論文誌, Vol. J82-D-I, No. 1, pp. 268–280, Jan. 1999.