データベースシステム 113－30
（1997. 7. 15）

アクティブデータベースにおける最適化を考慮したルール管理

宮崎 純　　横田 治夫
miyazaki@jaist.ac.jp　　yokota@jaist.ac.jp

北陸先端科学技術大学院大学 情報科学研究科
〒923-12 石川県能美郡辰口町旭台 1-1

あらまし アクティブデータベースにおけるルール条件の評価は非常に重い処理である．我々は，この処理を高速化するために，弁別ネットワークの並列処理とその最適化をバックグラウンドで行なう方式を提案してきた．本稿では，バックグラウンドで行なわれる最適化を考慮したアクティブルールの管理方法について述べる．また，複数の並行トランザクション間での，Δ-relation と弁別ネットワークを用いたルール条件の incremental なチェックについても言及する．

和文キーワード アクティブデータベース，ルール条件評価，弁別ネットワーク，最適化

Management of Rules and Their Background Optimization in an Active Database

Jun MIYAZAKI　　Haruo YOKOTA
miyazaki@jaist.ac.jp　　yokota@jaist.ac.jp

School of Information Science, Japan Advanced Institute of Science and Technology
1-1 Asahidai, Tatsunokuchi, Ishikawa 923-12, Japan

Abstract The evaluation of rule conditions is one of the heaviest job in active databases. To solve this problem, we have proposed a TREAT-based parallelized discrimination network and its background optimization scheme to conceal the optimization overhead. In this paper, we describe about a rule management method, taking account of the background optimization. We also consider an incremental condition evaluation method using Δ-relations and a discrimination network under multiple concurrent transactions.

英文 key words active database, rule condition evaluation, discrimination network, optimization

1 はじめに

データベースの分野で，最近注目されているアクティブデータベース [1] は，アクティブルールによりデータベース処理を能動的に起動することのできるシステムである．論理の拡張としての演繹データベースとは機構的に異なり，人工知能におけるルールベースシステムとデータベースシステムの融合といった側面を持つ．動的な経営判断や株の動向解析等の OLAP への利用も可能なことから，ビジネス分野でも注目されている．また，これまでのルールベースシステムとは異なり，頑健なデータベースに対してルール処理を適用することから，データの一貫性を硬く維持しなければならないという，ビジネス分野でのデータベースに対する要求にも適合する．

我々は，並列処理環境を前提とした PARallel Active Database Engine (Parade) と呼ぶ並列アクティブリレーショナルデータベースシステムのプロトタイプを構築している．アクティブデータベースでは，高負荷であるルール条件の評価がネックとなり性能が抑えられる．Parade では，このルール条件の評価を高速化するために TREAT ベースの弁別ネットワーク (P-TREAT) を用いて，これを並列処理することによりルール条件の評価の高速化を目指している．あるルール条件に対応する P-TREAT の処理は，リレーショナルデータベースの query tree に対応し，また，ルール条件の評価は何度も利用される．このため，Parade は並列環境を生かしてその query tree をバックグラウンドで楽観的に最適化する PODO 方式も提案している．本稿では，Parade における PODO による最適化を考慮した，アクティブルールの管理手法について述べる．さらに，並行トランザクション間での弁別ネットワークを，イベント駆動のアクティブルールを評価するための方法についても述べる．これらの処理が適切な排他制御によりシリアライザビリティが保たれることについても触れる．

2 Parade:並列アクティブデータベースプロトタイプ

並列アクティブデータベースプロトタイプ **Parade**(*PARallel Active Database Engine*) は，超並列計算機をターゲットとしたクライアント/サーバ型のアクティブデータベースである [2]．

Parade は並行細粒度プロセスを記述できる並列論理型言語 KL1 で記述され，KLIC[3] により C 言語へ変換することにより，超並列計算機 nCUBE2 や分散ワークステーション上で動作する．Parade は図 1のように，関係データベースにアクティブルール処理機構を統合した形で構成される．各要素は KL1 の並行プロセス集合から構成され，個々のプロセスは自然な形で同期がとられ，また並列に動作する．なお，アクティブルールはリレーションとして通常のデータと同様に管理される．

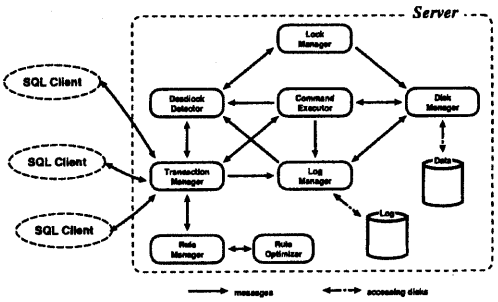

図 1: Parade のシステムアーキテクチャ

2.1 P-TREAT

弁別ネットワークは，データベースの変更の差分データのみを用いて効率良くルール条件を評価することができる．Parade ではストレージコストを下げるために，TREAT ネットワーク [4] のうち，ルール条件中の selection を施した結果を保存する α-memory を除いた P-TREAT(*Parade-TREAT*) を用いる[1]．また，ルール条件を満足するタップルの組を保存する *Conflict Set*(CS) は一時リレーションとしてトランザクション毎に持つ．これは，異なるトランザクション間で，並行に同一ルールをアクセスすることを可能とするためである．

差分データはデータベースに対する insert，update，delete 操作によって生じ，Δ-relation と呼ぶ特別なリレーションに入れられる．この Δ-relation を用いることにより，大きなベースリレーション

[1] TREAT の α-memory は永続的であるが，P-TREAT の α-memory は join 時に selection を施されて動的に生成された一時リレーションと見なすことができる．

にアクセスせずに，incremental に効率よく条件チェックを行なうことができる．このΔ-relation はトランザクション毎に一時リレーションとして扱う．そうすることにより，異なるトランザクション間の並行性を上げることができる．

基本的な P-TREAT のアルゴリズムは，あるΔ-relation に対応するルールに従って selection を施したリレーション$\sigma_i(\Delta R_i)$ に対して，次の通りに行なわれる．

$$
\begin{aligned}
CS_0 &= \sigma_1(R_1) \bowtie \cdots \bowtie \sigma_i(R_i) \bowtie \cdots \bowtie \sigma_n(R_n) \\
CS_{c+1} &= CS_c \bigcup \\
&\quad \sigma_1(R_1) \bowtie \cdots \bowtie \sigma_i(\Delta R_i) \bowtie \cdots \bowtie \sigma_n(R_n)
\end{aligned}
$$

以上のアルゴリズムにより，P-TREAT 内の各ルールは query tree の集合変換することができ，データベースの関係演算を利用することができる．Δ-relation はベースリレーションに比較して小さいので，異なる PE で並列にベースリレーションの読み出し，selection，build を行ない，Δ-relation を種として right-deep 木とハッシュ結合を組み合わせることにより，高い並列性を得ることができる．

2.2 P-TREAT の動的最適化

アクティブデータベースではアクションなどによりデータベース状態が変化するため，join の順序等の動的最適化は効率を維持するために有効である．

P-TREAT は，データデータベース状態の変化は緩やかであるという仮定の下，また，ルールに対応する query が反復して実行されることに着目して，次回以降の query に対する動的な最適化を，並列処理環境を利用して，バックグラウンドで行なう **PODO**(*Parallel Optmistic Dynamic Optimization*) 方式を用いる [5]．同一ルールが反復して実行されるのは，同一トランザクション内のルールのサイクルによるものだけでなく，異なるトランザクション間でも行なわれる．例えば，referential constraint のようなルールは，異なるトランザクション間で頻繁に使われる．

PODO には図 2 で示すように，条件チェックと最適化の時間関係を表す 3 種類の特有のタイミングモードがある．

- **tightly synchronous**: ルール条件チェックと最適化処理が同時に開始し，トランザクション終了時に最適化処理が終了しなければならない
- **loosely synchronous**: 次回のルール条件評価が開始するまでに最適化処理が終了しなければならない
- **asynchronous**: 条件チェックと最適化とは非同期に動作する

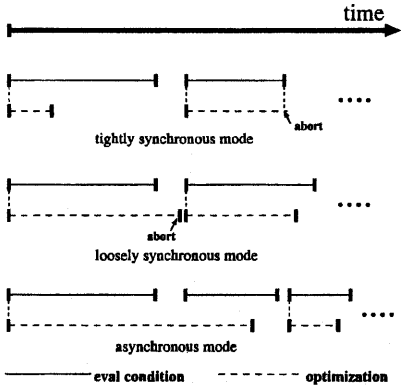

図 2: PODO のタイミングモード

Δ-relation を用いて join を行なう時，最適化のコストが join のコストに比較して無視できない場合があり，バックグラウンドで行なうことによる最適化のコストの隠蔽の意義は大きい．

異なるトランザクション間で同一のルールを最適化するのはコストが大きいため，オプティマイザに同一のルールの最適化のリクエストが来た時，一つのリクエストのみを処理し，その結果を他のリクエストの結果としても扱う．これにより冗長な最適化は行なわれない．

2.3 ネステッドトランザクション

あるトランザクション内でのルールの処理は，ネステッドトランザクション [6] により行なわれる．これは，ルール条件の評価やアクションの実行のアボートがトランザクション全体に影響を及ぼすことを避けるためである．ルールアクションの実行をサブトランザクションにより保護するだけでなく，ルール条件の評価も保護すべきである．それは，CS がそのトランザクション内で persistent であり，CS への結果の union 操作を含むためである．また，ネステッドトランザクショ

ンは，兄弟トランザクション間の並行実行も可能とする．

3 ルールの管理

3.1 ソースルールとコンパイルドルール

ソースルールは，ユーザにより定義されたSQL記述のECAルールである．一方，コンパイルドルールはソースルールを関係代数レベルのquery treeにまでコンパイルしたルールである．コンパイルドルールは，各Δ-relationごとに最新の最適なquery treeを持つ．このソースルールとコンパイルドルールの双方とも，データベース中にリレーションとして管理される(図3)．ソースルールが新たにinsertされたとき，ルールマネージャはソースルールの条件部分をΔ-relationごとにインデックスし，静的な最適化を行なった後，コンパイルドルールに登録する．また，ルール処理のためのプロセスネットワーク(図4)[7]に，新しくルール処理プロセスを追加する．このルール処理プロセスはシステム全体で1つだけ持ち，イベントにタグ付けされたトランザクション識別子に従って，適切なトランザクション上でルールをトリガする．

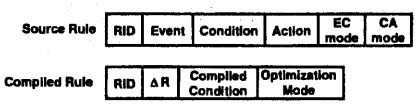

図 3: ソースルールとコンパイルドルール

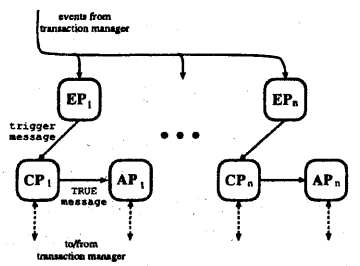

図 4: ECA プロセス

ソースルールとコンパイルドルールを分離する理由は，次の通りである．

- SQL レベルのルールをトリガのたびにコンパイルするのを避ける
- 常に最新のデータベース状態に応じたコンパイルドルールを維持する
- Δ-relation ごとにインデックスし，対応するコンパイルドルールを容易に検索可能とする
- ユーザに対して内部表現であるコンパイルドルールを隠蔽する

ある新しいトランザクションが開始する時，コンパイルドルールが一時リレーションとして各トランザクション毎に読み出される(図5)．これは，ルール条件の最適化が行なわれ，その結果新しいコンパイルドルールを更新するときに，同じルールを参照する他のトランザクションがブロックしないようにするためである．

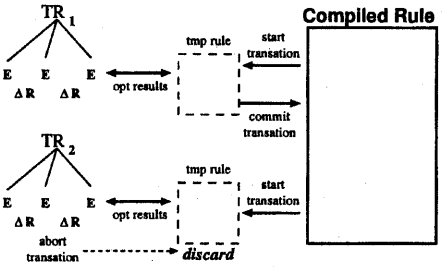

図 5: コンパイルドルールと一時ルール

3.2 Δ-relation のバージョニング

P-TREAT はΔ-relation によりルール条件の評価の高速化を行なうが，アクティブデータベースのルール評価がルールベースシステムと異なる点は，アクティブデータベースはイベントによりトリガされた時に特定のルールが評価されるのに対して，ルールベースシステムではあるデータが変更された時に，その変更により影響を受ける全てのルール条件の評価を行なう点である．すなわち，アクティブデータベースのルールはイベント駆動に対して，ルールベースシステムのルールはデータ駆動で条件が再評価される．

このため，アクティブデータベースでは各ルールごとに前回条件を評価した時のベースリレーションと，次回に評価する時のベースリレーショ

ンの差分を用いて，条件評価を行なわなければならない．そのためには，データベースイベントごとにΔ-relationをバージョニングし保存することにより解決できる．例えば，あるトランザクションが図6のようにイベントE_iによりデータベース状態が遷移し，その結果，差分ΔR_iが生じたとする．前回E_iであるルールがトリガされ，対応する条件が評価され，次にE_jにより同一のルールがトリガされ，対応する条件を評価するためには，E_iからE_jまでのΔ-relationの合成($\Delta R'$)の計算が必要となる．これは，

$$\Delta R' = \bigcup_{k=i}^{j-1} \Delta R_k$$

で計算できる．

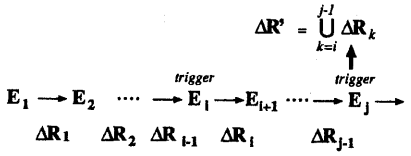

図 6: Δ-relationのバージョニング

3.3 ルール条件の評価と最適化

あるデータベースイベントに対してあるルールがトリガされ，対応する条件が評価される時，次の操作が行なわれる．

(1) サブトランザクションのスタート

(2) 適切なバージョン($i \sim j-1$)を持つΔRに対して共有ロックし，$\Delta R'$を合成

(3) そのルールに関連する他のリレーションに対して共有ロック

(4) 対応するCSに対して排他ロック

(5) 対応するコンパイルドルールに対して共有ロック

(6) (5)のコンパイルドルールを用いてjoin，CSとのunion

(7) サブトランザクションのコミット

なお，(3)の共有ロックはデータベースの更新と同様，プレディケートロックやタップル単位のロックにより，異なるトランザクション間の並行性を上げることができる．すなわち，データ操作が衝突しない限り，同一ルール条件の評価は同時に複数トランザクション間で実行できる．

次にPODOによる最適化とトランザクションの関係について述べる．各ルール条件の評価はサブトランザクションにより保護されており，PODOの各モードとトランザクションの関係について，明らかにしなければならない．

tightly synchronousモードでは，(6)の処理と同時にオプティマイザに最適化の要求を出す．ルールの評価がコミットする(7)において，最適化の結果が得られていないとき，オプティマイザにその最適化をアボートするよう要求する．最適化の結果が得られている時は，対応するコンパイルドルールを更新する．

loosely synchronousモードでは，(5)のコンパイルドルールを読む時に，オプティマイザに同一ルールの最も最近行なわれた最適化の結果を要求する．もし，結果が得られていなければ，対応するコンパイルドルールを更新し，それを(6)のjoinに採用する．そうでなければ，その最適化をアボートし，一時リレーション中のコンパイルドルールを採用する．それと同時に新しい最適化処理をオプティマイザに要求する．

asynchronousモードは，loosely synchronousモードとほぼ同様の処理を行なうが，継続中の最適化処理をアボートをしない．

3.4 トランザクションの終了時の処理

実行すべきルールが無くなる，ルール実行により強制的にトランザクションがコミットされる，もしくはトランザクションのアボートによりトランザクションは終了する．

もしトランザクションがコミットする時，Δ-relation，CSの一時リレーションは捨てられる．そのトランザクションに一時リレーションとして扱われていたコンパイルドルールは，オリジナルのコンパイルドルールリレーションに結果を反映した後，捨てられる．これにより，オリジナルのコンパイルドルールリレーションには，最新のコンパイルドルールが維持される．

もし，トランザクションがアボートする時は，Δ-relation，CS，コンパイルドルールの全ての一

時リレーションが捨てられる．なぜならば，トランザクションがアボートする時，ルールのアクション実行を含めて全てのデータベースに対する操作が取り消されため，データベース状態がそのトランザクション実行前に戻される，このため，トランザクションのコンパイルドルールが完全に無効となるからである (図5参照)．

4 排他制御

ベースリレーションへのアクセスは，プレディケートロックやタップル単位でのロックによる排他制御を通して行なうことにより，異なるトランザクション間でのinsert，update，deleteおよび，ルール実行のシリアライザビリティは保たれる．すなわち，あるトランザクションがプレディケート排他ロックを取り，ベースリレーションに変更を行なっている時は，他のトランザクションはそのロックの範囲内の操作，およびそれに対応するΔ-relationの作成は，そのロックを保持しているトランザクションが終了するまで待たされる．そのプレディケートロックを持つトランザクションがコミット (もしくはアボート) したならば，待たされていたトランザクションが動き始めるだけである．これに対してロックの範囲外のベースリレーションへのアクセスは，異なるトランザクション間で並行して変更，およびΔ-relationの生成が行なえる．当然，ロック待ちのないルール実行も並行して行なうことができる．

一方，同一トランザクション内では，サブトランザクション間でのベースリレーションに対する変更は，適切なロックにより異なるトランザクション間と同様，シリアライザビリティが保たれる．Δ-relationについては，同一トランザクション内において各バージョンの生成毎に排他制御が行なわれ，イベント駆動によるincrementalなルール条件の評価が行なえる．各々のサブトランザクションはコミット時にロックを親トランザクションに継承し，ロックにより待たされていた他の兄弟サブトランザクションは，親からロックを受け取り，そのサブトランザクションが動き出す．

5 まとめ

アクティブデータベースでは，アクティブルールの条件の評価がシステム全体の性能を落す原因となる．このため我々は並列処理環境を前提として，PODOと呼ぶ実際の条件の評価とは並列にバックグラウンドでルール条件の最適化を行なう方式を提案している．このとき，最適化によるルール条件部はコンパイル形式で維持される必要がある．

本稿では，PODO方式を最適化を考慮した場合に，このコンパイルドルールとユーザが定義するソースルールとを分けて管理する方法とその利点について触れた．また，複数のトランザクション間で並行性を大きく損なわずに，Δ-relationと弁別ネットワークを用いてincrementalにルール条件を評価する方法について述べた．イベント駆動で適切にルール条件の評価を行なうためには，トランザクション内でΔ-relationをバージョニングしルールがトリガされた時に，そのルール条件に必要なΔ-relationを合成することで可能となることを示した．さらに，以上の処理がプレディケートロックやタップル単位のロックにより並行性を損なわず，シリアライザビリティも保つことができることを示した．

参考文献

[1] J. Widom and S. Ceri: "Active Database Systems: Triggers and Rules For Advanced Database Processing", Morgan Kaufmann Publishers, INC., San Francisco, CA (1996).

[2] 横田, 宮崎, 土屋 : "並列アクティブデータベースエンジンParadeの並列処理", 文部省科学研究費重点領域研究『高度データベース』松江ワークショップ講演論文集, 第2巻, pp. 426–433 (1996).

[3] T. Fujise, T. Chikayama, K. Rokusawa and A. Nakase: "KLIC: A Portable Implementation of KL1", Proc. of FGCS '94, pp. 66–79 (1994).

[4] D. P. Miranker: "TREAT: A New and Efficient Match Algorithm for AI Production Systems", Morgan Kaufmann Publishers, INC., San Mateo, CA (1990).

[5] J. Miyazaki and H. Yokota: "An Optimization Technique of Discrimination Networks in Active Database Systems for Massively Parallel Processing", Proc. of CODAS '96, pp. 414–417 (1996).

[6] J. E. B. Moss: "Nested Transactions: An Approach to Reliable Distributed Computing", MIT Press Series in Information Systems, The MIT Press, Cambridge, MA (1985).

[7] 宮崎, 横田 : "並列アクティブデータベースParadeのルール処理機構", Proc. of DEWS '97 電子情報通信学会 (1997).