データベースシステム 104−14
（1995. 7. 18）

メッセージ通信に基づく並列トランザクション管理

横田治夫　　荒井竜一†
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 15
yokota@jaist.ac.jp

高機能な並列トランザクション管理システムを構成するためのモデルの提案を行ない、並列論理型言語 KL1 でシステムを記述する場合の実現方法を示す。まず、高機能化のため、データベースの各項目は自律的に動作するオブジェクトとして扱う。と同時に、ネストが可能なトランザクション、各アクティブルールの発火制御、ロックテーブルのエントリーもそれぞれ自律的なオブジェクトとする。これらのオブジェクトの集合間でどのようなメッセージ通信が必要なのか解析を行ない、デッドロック検出の実現方法やログの取り扱いについても考察する。更に、KL1 の永続プロセスと論理変数を用いたメッセージ通信によってスマートに実現できることを示す。

Parallel Transaction Management using Message Communication

Haruo Yokota　　Ryuichi Arai†
School of Information Science
Japan Advanced Institute of Science and Technology, Hokuriku
15 Asahidai, Tatsunokuchi, Nomi, Ishikawa, 923-12 Japan
yokota@jaist.ac.jp

We propose a model for an intelligent parallel transaction management system, and an implementation method of the system using a parallel logic programming language, KL1. Items in a database, nested transactions themselves, firing controls of active rules, and entries of a lock table are treated as sets of autonomous objects. We analyze messages among these objects, and mechanisms for detecting deadlocks and logging. KL1 is suitable for representing the system, since a perpetual process of KL1 directly corresponds to an object, and logical variables shared among the processes can be used for passing messages.

†現在の所属：日本テレコム

1 はじめに

社会の情報化が進み、あらゆるものに関するデータがデジタル化されてデータベースとして格納され、アクセスされつつある今日、オンライントランザクション処理の重要性が増していることは言うまでもないと思われる。米国情報スーパーハイウェイ構想に代表されるように、マルチメディアデータがネットワークを行き交い、高度利用のためにマルチメディアのデータがデータベース化 [1] されれば、その機能拡張と高性能化の要求は更に高まることは間違いない。

データベース処理の高機能化と言う観点からすると、操作対象を単なる関係データベースのタプルから、オブジェクト指向データベースのオブジェクトに移行させる方向が有効であろう [2][3]。抽象データ型として、データとそのデータに対する操作を組にすることにより、柔軟性が増す。と同時に、格納された内容の変更に伴い動的に何らかのアクションを起こすアクティブルールの発火機能も採り入れるというアプローチ [4] を取ることにより更に高機能化を図ることができるようになる。この時、アクティブルールを発火させるためには、トランザクションをネストさせる [5] ことも重要となってくる。

例えば、動画サーバとしてマルチメディアデータベースをとらえた場合、扱うデータは、動画データそのものの他に、課金情報などの一般のデータも当然対象となる。動画データは、受けとる側の能力によって、その画質や転送速度が異なるかも知れない。それらの調整は、抽象データ型のメソッドに組み込まれているかもしれない。あるいは、受けとる側の能力に関する項目が変更されることによりアクティブルールで対応がとられるかも知れない。課金情報にしても、デポジットが減ってきた場合の注意喚起をアクティブルールで対応することもできよう。場合によっては、統計データなどからアクティブルールによりアクセス制限を行なう場合があるかも知れない。そのような場合には、アクティブルールがネストされたトランザクションを走らせているようにとらえることができる。

このように高機能化していくと、当然性能的に問題が発生してくる。そもそも、オンライントランザクション処理に対する性能要求は、そのリアルタイム性から非常に高いものである。その要求に対応するためには、高並列化は避けて通れない問題である。現在、専用並列マシン等も存在するが、上に述べたような高機能化はこれからの課題であろう。ここで扱うモデルでは、基本的にはオブジェクト単位で並列に実行できることが望ましく、アクティブルールの発火等も並列化されるべきである。ネステッドトランザクションは、分散環境での利用を前提に検討されてきたが [6][7]、当然そのまま並列環境でも有効な手法である。その場合に排他制御が並列化のネックとなってはならない。

システムの記述に関しては、高並列化を考えた場合、従来の逐次型言語で記述することは、同期処理の繁雑さを考えるとあまり好ましいとは言えない。並列論理型言語を用いると、オブジェクトを永続プロセスで実現し、そのオブジェクト間のメッセージ通信を論理変数への代入の形で記述することができ、プロセッサ間の同期処理を気にする必要がなくなる。第五世代コンピュータプロジェクトで並列推論マシン用 [8] に開発された並列論理型言語 KL1[9] は、シンタクス的にもシンプルで、C 言語にコンパイルして汎用並列マシンでも実行可能なポータブルな処理系 [10] も用意され、利用し易い状況になっている。

KL1 に関係したデータベース関係の研究としては、ICOT で行なわれてきた、Kappa[11] と *Quixote*[12] がある。Kappa は非正規関係の検索を主眼としており、アクティブルールやネステッドトランザクション等は扱っていない。また、*Quixote* は演繹機能に重点を置いた知識表現言語という位置付けになっている。

本報告では、データベースの各項目を自律的に動作するオブジェクトとして格納すると同時に、アクティブルールの発火機能や、ロック機能も同様に自律的に並列に動作するオブジェクトして取り扱う並列ネステッドトランザクションシステムのモデルについて提案を行う。次に、システムを並列論理型言語 KL1 で記述する場合の構成方法を検討し、並列推論マシン上での簡単な実験について報告を行なう。

2 システムモデル

2.1 システムの構成要素

システムの基本的な構成要素を、

T: トランザクションオブジェクト集合

D: データオブジェクト集合

L: ロックオブジェクト集合

R: ルールオブジェクト集合

とし、システム S を、

$$S =< T, D, L, R >,$$

のように表現する。

トランザクションオブジェクト集合　具体的には、T はトランザクションを識別する識別子(TRID)を持ち、ネステッドトランザクションの入れ子構造（木構造）を管理するオブジェクトの有限集合である。ここで、ネストされた各子トランザクションも1つのオブジェクトとしてみなす。時刻 τ に n 個のトランザクションオブジェクトが有るとすると、t_i^τ $(1 \leq i \leq n)$ を1つのトランザクションオブジェクトとして、時刻 τ のトランザクションオブジェクト集合 T^τ を、

$$T^\tau = \{t_1^\tau, \ldots, t_n^\tau\}$$

のように表現する。

データオブジェクト集合　データベースとして格納され、実際に利用者からアクセスされる対象のデータをそれぞれ管理するオブジェクトの有限集合がデータオブジェクト集合 D である。データの実体は、不揮発性（二次）記憶上にあり、その写像としてデータオブジェクトが存在するものとする。ここでは、トランザクションのコミットまでは、データオブジェクトは一時的な状態を取るものとして、時刻 τ のデータオブジェクト集合 D^τ を、

$$D^\tau = \{d_1(s_1)^\tau, \ldots . d_m(s_m)^\tau\}$$

と表現することにする。ここで、変数 s_j は、データオブジェクトとして保持している値が、二次記憶上の実体と同じ恒久的なものである場合には p という値を、コミット前の一時的なものである場合には t という値を取るものとする。なお、以降の議論で、恒久的／一時的という情報が重要でない場合には、$d_j(s_j)$ の代わりに d_j を用いる。

ロックオブジェクト集合　自律的に動作するデータオブジェクトの同時アクセス制御には、ロックテーブルを用いることとするが、ロック管理が逐次的で全体のボトルネックとなることは好ましくない。そこで、ロックテーブルをエントリー毎に分割し、個々のエントリー毎にロックを管理する分散ロックを前提とする。L^τ を時刻 τ における分割されたロックテーブルのエントリーを管理するロックオブジェクトの有限集合とし、

$$L^\tau = \{l_1^\tau, \ldots, l_m^\tau\}$$

と表す。l_j は、データオブジェクト d_j に対応したロックオブジェクトである。分散ロックでは、デッドロック検出が問題となるが、ロックオブジェクト（分割されたロックエントリー）間の通信により、デッドロックの検出を行なうことが可能である。

ルールオブジェクト集合　データオブジェクトの変更等のトリガーにより、発火するアクティブルールを管理するアクティブルールオブジェクトの有限集合 R^τ を、

$$R^\tau = \{r_1^\tau, \ldots, r_k^\tau\}$$

のように表す。なおここでは、アクティブルールは、トランザクション、データ、ロックエントリーとは異なり、時系列で変化しないことを前提とするが、アクティブルールオブジェクトとしては、自律的に動作するオブジェクトとし、時刻により状態が変わるものとする。アクティブルール自体は、データ同様不揮発性（二次）記憶上にあるものとする。将来的には、アクティブルールの更新もデータと同様の枠組で行なえることが望ましいが、そのためにはデータの更新との同期等を考慮する必要が生じるため、まずシステム動作中はルールの変更はないものとして議論を進める。

2.2　集合間の関係

T、D、L、R、いずれかのオブジェクト集合を O とする。

$$O \in \{T, D, L, R\}$$

このとき、O の要素 $\varepsilon \in O$ は、メッセージを受けて自律的に動くオブジェクトである。ここで、要素 ε_x から要素 ε_y へのメッセージを、$\mu(\varepsilon_x, \varepsilon_y)$ で表すことにする。時刻 τ における各オブジェクトの内部状態 ε_x^τ は、メッセージ$\mu(\varepsilon_y, \varepsilon_z)$ を受けとると、そのメッセージとその時点の内部状態を入力とするそのオブジェクトが持つ状態変化の関数 f_{ε_x} によって新たな内部状態 $\varepsilon_x^{\tau+1}$ に変化する‡。

$$\varepsilon_x^{\tau+1} = f_{\varepsilon_x}(\varepsilon_x^\tau, \mu(\varepsilon_y, \varepsilon_x))$$

‡この、$O^{\tau+1}$ の定義から分かるように、τ は、状態がメッセージを受けることにより変化するシステム全体の論理的な時刻を表している。

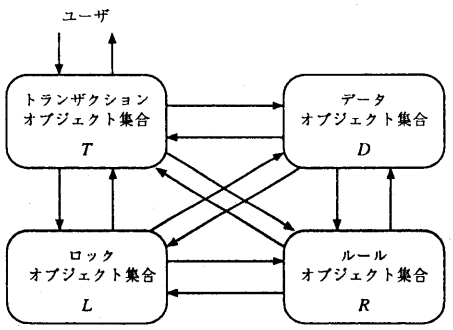

図 1: 集合間のメッセージ通信パターン

また、必要な場合にはメッセージ生成関数 f_{ε_m} を用いて新たなメッセージ $\mu(\varepsilon_x, \varepsilon_z)$ を生成する。

$$\mu(\varepsilon_x, \varepsilon_z) = f_{\varepsilon_m}(\varepsilon_x^\tau, \mu(\varepsilon_y, \varepsilon_x))$$

場合によっては、1 対 1 の要素間だけでなく、ある要素から、ある集合全体に対して、メッセージを送る必要が生じる場合がある。実現する場合には、ブロードキャストメッセージあるいは、マルチキャストメッセージのようなものである。その場合は、

$$\mu(\varepsilon_x, O) = f_{\varepsilon_m}(\varepsilon_x^\tau, \mu(\varepsilon_y, \varepsilon_x))$$

と表現できる。逆に、集合全体の内容に従って、メッセージを生成するような場合は、O の中のある一つの要素 ε_x が最終的な合意を取って、

$$\mu(\varepsilon_x, \varepsilon_z) = f_{\varepsilon_m}(O^\tau, \mu(\varepsilon_y, \varepsilon_x))$$

のようにメッセージを生成する。

4 つのオブジェクト集合の要素間で取り得る全てのメッセージの組合せは、次の 12 種類である。

$$\mu(t_x, d_y)\ \mu(t_x, l_y)\ \mu(t_x, r_y)\ \mu(d_x, l_y)\ \mu(d_x, r_y)\ \mu(l_x, r_y)$$
$$\mu(d_x, t_y)\ \mu(l_x, t_y)\ \mu(r_x, t_y)\ \mu(l_x, d_y)\ \mu(r_x, d_y)\ \mu(r_x, l_y)$$
$$t_{\{x/y\}} \in T,\quad d_{\{x/y\}} \in D,\quad l_{\{x/y\}} \in L,\quad r_{\{x/y\}} \in R$$

これは、図 1 に示すように、T、D、L、R、を頂点とする有向完全グラフの各辺に対応する。実際に、それぞれの要素の動作を考慮すると、必要なメッセージの通信の種類は上記のものに比べて減ることになる。

この他に、同一集合内や、ユーザから／へのメッセージがある。ユーザとのやりとりは、必ずトランザクションオブジェクトを経由することとして、$\mu(user, t_x)$ と $\mu(t_x, user)$ を前提とする。

2.3 各オブジェクトの動作

トランザクションオブジェクト　各トランザクションオブジェクトは、ユーザからのメッセージもしくは、ルールオブジェクトからのメッセージにより、状態が変化すると考えるのが素直である。よって、

$$t_x^{\tau+1} = f_{t_*}(t_x^\tau, \mu(\{user/r_y\}, t_x))$$

となる。具体的には、トランザクションおよびサブトランザクションの開始／コミット／終了／中断といった要求により、その状態が変化する。要求によっては、新たなメッセージを、データオブジェクトやロックオブジェクトに発送する。具体的には、まずアクセスするデータに対応するロックオブジェクトに、

$$\mu(t_x, l_z) = f_{t_{m1}}(t_x^\tau, \mu(\{user/r_y\}, t_x))$$

というメッセージでロックを要求し、ロックが獲得できた場合には、データオブジェクトに、

$$\mu(t_x, d_z) = f_{t_{m2}}(t_x^\tau, \mu(\{user/r_y\}, t_x), \mu(l_z, t_x))$$

のように、ロックが獲得できた旨ともともとの要求内容を伝える。

データオブジェクト　データオブジェクトは、トランザクションオブジェクトよりメッセージを受けとると、まず、現在のデータオブジェクトの状態をルールオブジェクト集合にメッセージとして送り、その返答を待つ。どのアクティブルールが発火するか、この時点では分からないので、ルールオブジェクト集合全体にメッセージをブロードキャストすることになる。

$$\mu(d_x, R) = f_{d_{m1}}(d_x^\tau, \mu(t_y, d_x))$$

ルールオブジェクト集合では、いずれか 1 つのオブジェクトが最終的な判断を下すことになる。その返答が許可であった場合には、データオブジェクトは自分の状態を変化させ、トランザクションオブジェクトにレスポンスを返す。

$$d_x^{\tau+1} = f_{d_*}(d_x^\tau, \mu(t_y, d_x), \mu(r_z, d_x))$$

$$\mu(d_x, t_y) = f_{d_{m2}}(d_x^\tau, \mu(t_y, d_x), \mu(r_z, d_x))$$

当該トランザクションがコミットされるまでは、このデータオブジェクトは、$d_x(t)^{\tau+1}$ という一時的な状態を保つ。トランザクションオブジェクトよりコミットが知らされた時点で、対応する二次記憶の内容が変更

され、$d_x(p)^{\tau+1}$ という恒久的な状態になる。コミットの手続きは、二相コミットに従う。また、トランザクションが中断（アボート）された場合には、$d_x(t)^{\tau+1}$ の状態は破棄され、前の二次記憶と同じ状態 $d_x(p)^{\tau}$ に戻される。

ロックオブジェクト　上述したように、トランザクションオブジェクトは、データオブジェクトをアクセスする前にロックオブジェクトにロックの獲得を要求してくる。ロックオブジェクトは、データオブジェクトに対応したロック管理テーブルの1エントリーとみなすことができる。状態は、対応するデータオブジェクトを要求するトランザクションの待ちキューである。待ちキューが空の場合には、状態を変更して（キューの先頭につないで）、要求元に対して獲得できた旨のメッセージを知らせる。

$$l_x^{\tau+1} = f_{l_*}(L^{\tau}, \mu(t_y, l_x))$$

$$\mu(l_x, t_y) = f_{l_{m1}}(L^{\tau}, \mu(t_y, l_x))$$

もし、空でない場合には、同様に状態を変更するが、獲得できるまでメッセージは送らない。あるトランザクションのロック解除により、それまで待ちキューの二番目にいたロック待ちトランザクションにロック獲得のメッセージを送出する。

$$\mu(l_x, t_z) = f_{l_{m2}}(L^{\tau}, \mu(t_y, l_x))$$

ここで、ロックによってトランザクションが待ち状態に入ることから、デッドロックの可能性があることに注意する。デッドロック発生時にはロックオブジェクトからロールバック指示を出すため、状態遷移やメッセージ生成は、当該ロックオブジェクトのキューだけでなく、ロックオブジェクト全体の状態に依存する。デッドロック検出のためにはロックオブジェクト間でメッセージをやりとりする必要が生じるが、具体的なデッドロック検出のメカニズムは次章で述べる。排他ロックと共有ロックは、キューの先頭に共有ロックが来ている間は、排他ロックの前に共有ロックを入れていくことで対応する。ロック施錠／解除は二相ロックに従うが、フェーズの管理はロックオブジェクトではなくトランザクションオブジェクトが担当する。

ルールオブジェクト　ルールオブジェクト集合には、トランザクションオブジェクトから送られてきたメッセージを基にデータオブジェクトが作成したメッセージが送られてくる。ここで注意すべきことは、データオブジェクトはどのルールオブジェクトが対応するかの知識は持たない点である。つまり、対象を特定せずに、ルールオブジェクト集合全体に対するメッセージとして発生される。各ルールオブジェクトは、そのメッセージを受けとると、自律的に発火すべきかどうか判断を行ない、最終的にはいずれか1つのルールオブジェクトが集合全体としての合意を取って、元のデータオブジェクトに返答メッセージを返す。

$$\mu(r_y, d_x) = f_{r_{m1}}(R^{\tau}, \mu(d_x, R))$$

ルールによっては、新たなサブトランザクションを作成する必要が生じるものがあるかも知れない。その場合には、トランザクションオブジェクト集合にその旨を伝えるメッセージを発して、子どものトランザクションとして生成させる。

$$\mu(r_x, t_z) = f_{r_{m2}}(R^{\tau}, \mu(d_y, R))$$

そのようにして生成された子トランザクションと呼び出し元のトランザクションの関係は、メッセージの推移関係を見ることによって判定できる。つまり、$\mu(d_y, R)$、$\mu(t_z, d_y)$ の定義から、

$$\mu(r_x, t_z) = f_{r_{m2}}(R^{\tau}, f_{d_m}(d_y^{\tau}, f_{t_{m2}}(t_w^{\tau},$$

$$\mu(\{user/r_v\}, t_w), \mu(l_u, t_w))))$$

のように、t_z が t_w から再帰的に呼ばれていることが分かる。この呼び出し関係は、TRID をメッセージの中に含ませることで示すことができ、その親子関係はトランザクションオブジェクトの中で管理される。また、ロックオブジェクトへのメッセージにもいれて、階層的なロック管理にも用いる。一方、ルールオブジェクトからの返答がない限り親のトランザクションは終了できないので、孤児 (orphan) トランザクションは生成されない。

2.4　ログの取り扱い

トランザクション管理におけるログの取り扱いは、Redo-Undo 機能やシステム性能の上で重要な位置を占める。ここで提案するモデルでは、データオブジェクト集合が対象となるリソースの実際の管理を行なっていることから、データオブジェクトからログを生成することにする。

ログには、対象データ自身のログシーケンシャル番号 (LSN) と、トランザクションの LSN が必要になる

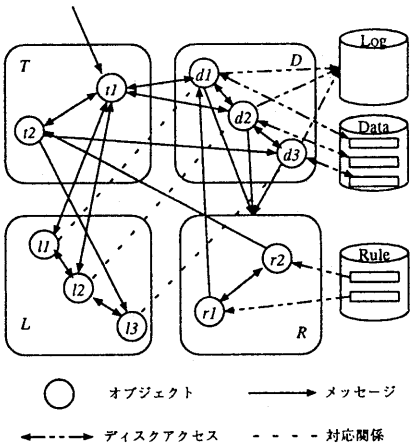

図 2: システムの構成

が、データの LSN はデータオブジェクトがその状態として保持しており、トランザクションの LSN はトランザクションオブジェクトが保持した値をメッセージに含ませて渡すことができる。つまり、ログレコード LR を、

$$LR = f_{d_i}(d_x(p)^\tau, \mu(t_y, d_x))$$

とすると、$\mu(t_x, d_y)$ の定義から、

$$LR = f_{d_i}(d_x(p)^\tau, f_{t_{m2}}(t_y^\tau, \mu(\{user/r_z\}, t_y), \mu(l_w, t_y)))$$

となり、$d_x(p)^\tau$ と t_y^τ が持つ LSN からログを作ることができる。ここで、データオブジェクトは、コミットするまで一時的な値を保持していることから、データオブジェクト側でログを書き出すタイミングは、トランザクションオブジェクト側でコミットトランザクションによる二相コミットで合意が取れた後、恒久的な状態になるデータオブジェクト $d_x(p)^\tau$ のログを書き出すことになる。

2.5 システムの構成

以上の各オブジェクトの動作とログの扱いをまとめると、図2 に示すようなメッセージ通信によって、システムの動作を実現できることが分かる。同一集合内でのメッセージを除くと、前述した 12 種類のメッセージの中で、$\mu(t_x, d_y)$、$\mu(d_x, t_y)$、$\mu(t_x, l_y)$、$\mu(l_x, t_y)$、$\mu(d_x, R)$、$\mu(r_y, d_x)$、$\mu(r_x, t_z)$ の 7 種類が使われている。

なお、ここで示した、各オブジェクトの動作やメッセージの種類は、システムの 1 つの構成例によるものに過ぎない。例えば、ここでは、データオブジェクトの処理の並列性を高めるため、各データオブジェクトがルールオブジェクト集合に対してルールのチェックを依頼するような形態を取っているが、トランザクションオブジェクトから依頼することも可能である。ここで提案する構成と違う構成方法を取った場合との比較については、今後の研究項目としたい。

3 並列論理型言語による実現

前章では、システムを自律的なオブジェクトとその間のメッセージによって記述するモデルを示した。オブジェクト間では、メッセージのみにより通信が行なわれ、一般のトランザクション管理システムのようにトランザクション管理テーブルやロック管理テーブルのような共有テーブルのようなものは存在しない。このため、オブジェクトが確実に自律的に動作し、メッセージによるデータフローが実現されれば、システムは自然な並列性を抽出することができる。

ここでは、並列論理型言語 KL1 によってシステムを記述する場合の実現手法 [13] について述べる。

3.1 永続プロセスによるオブジェクト

各オブジェクトは、KL1 の述語の再帰呼びだしによる永続（パーベチュアル）プロセスとして実現され、状態は再帰呼びだしにおける論理変数の内容という形で扱われる。また、オブジェクト間のメッセージパッシングは論理変数への代入と言う形で行なわれる。このため、前章で述べた状態とメッセージをほぼそのままに、容易に実現するこができる。

各集合のオブジェクトを生成させるためにマネジャを用意すると、トランザクションオブジェクト集合と、ロックオブジェクト集合とデータオブジェクト集合の間の通信は、次のように記述できる。

```
transaction_set(TR_msgs) :- |
    tr_manager(TR_msgs,LK_msgs,DT_msgs),
    lock_set(LK_msgs),
    data_set(DT_msgs).
```

トランザクションマネジャの記述を簡略化して表すと、新たなトランザクションオブジェクトの生成とそこへ

のメッセージ通信は、永続プロセスを用いて次のように記述できる。

```
tr_manager([(Trid,start(Ans))|Rest],Lm,Dm) :- |
    tr_obj(Trid,1,Rest,Lm1,Dm1), Ans = ok,
    merge(Lm1,Lm2,Lm), merge(Dm1,Dm2,Dm),
    tr_manager(Rest,Lm2,Dm2).
tr_manager([],Lm,Dm) :- |
    Lm = [], Dm = [].
otherwise.
tr_manager([Msg|Rest],Lm,Dm) :- |
    tr_manager(Rest,Lm,Dm).
```

各トランザクションオブジェクト内では、次のようにして、自分宛のメッセージの判断と、それによる新たなメッセージの生成を行なうことができる。

```
tr_obj(Id,1,[(Id,read(X,Ans))|Rest],Lm,Dm) :- |
    Lm = [share_lock(X,Ans1)|Lm1],
    check(Ans1,read(X,Ans2),Ans2,Ans,Dm,Dm1),
    tr_obj(Id,1,Rest,Lm1,Dm1).
otherwise.
tr_obj(Id1,Phase,[(Id2,Req)|Rest],Lm,Dm) :- |
    tr_ojb(Id1,Phase,Rest,Lm,Dm).

check(ok,Req,Ai,Ao,Dm,Dm1) :- |
    Dm = [Req|Dm1], Ao = Ai.
check(ng,Req,Ai,Ao,Dm,Dm1) :- |
    Dm = Dm1, Ao = ng.
```

ロックオブジェクトへのメッセージに含まれる変数Ans1は、ロックが取れたかどうかを示すロックオブジェクトからトランザクションオブジェクトへのメッセージを返すための変数であり、同様にAnsは、データオブジェクトからトランザクションオブジェクトへのメッセージを返すための変数である。各メッセージは、変数上でリストの要素として扱われるため、メッセージの追い越しがなく、そのための複雑な機構を必要としない。

同様にして、他の処理や他のオブジェクト集合に関してもスマートに記述することができる。

3.2 並列デッドロック検出

本報告で提案しているような分散ロックでは、デッドロック検出が1つの課題となる。検出方法としては、大きく分けて、タイムアウトによる方法とトランザクション間のWFG(Wait-for-Graph)を利用する方法がある。タイムアウトによる方法は、実現が簡単であるが、デッドロックが実際に発生していない場合でもトランザクションを中断することがあり、処理を高機能化してトランザクションの実行時間が一定でなくなった場合には、あまり適さない。一般のWFGによる分散デッドロック検出は、全体の情報を整理をしてループを検出するために、何らかの集中制御が必要になる。しかし、ここでは、各オブジェクトが自律的に動作することを前提としており、集中的な管理は並列効果の上からも適さない。そこで、各ロックオブジェクト間でメッセージを通信しながら、ダミーリストを入れたWFGを利用して並列にデッドロックを検出する方法[14]を用いることにする。

各ロックオブジェクトは、実際に対応するオブジェクトに対して要求を出しているトランザクションだけでなく、そこから推移的に得られる待ち関係をダミーの要求としてキューに入れておく。そうすることによって、各ロックオブジェクトは、自分のキューを見るだけで、全体のトランザクション間の待ち関係を把握することができ、ループ検出によってデッドロックを発見できる。当然、ダミーの要求は、実際にはロックを必要としないので、普通の要求とは区別しておく必要がある。ダミーリストは、同じトランザクションに属するロックオブジェクトがキューの前後関係をメッセージとして転送し合うことで作成することができる。

3.3 並列ルール発火機構

アクティブルールの発火に関しては、トランザクションオブジェクトからルールオブジェクト集合全体に対して、ブロードキャストが必要なことを述べた。KL1では、複数のプロセスで論理変数を共有することにより、ブロードキャストを容易に実現することが可能となる。つまり、ルールオブジェクトを生成する際に同じ変数を持ち回るようにしておけば、その変数にメッセージを次々と流すことによって、すべてのルールオブジェクトで受けとることができる。ルールオブジェクトは、並列にそれらのメッセージに対応するルールの発火をチェックし、結果を論理和を取りながらマージしていくことにより、最後に結果を出したルールオブジェクトが、結果メッセージをトランザクションオブジェクトに返すことになる。

実はこのようなブロードキャストによる発火チェックはコストが高いため、将来的には、Reteアルゴリズム[15]を並列化した手法[16]の利用を検討したい。

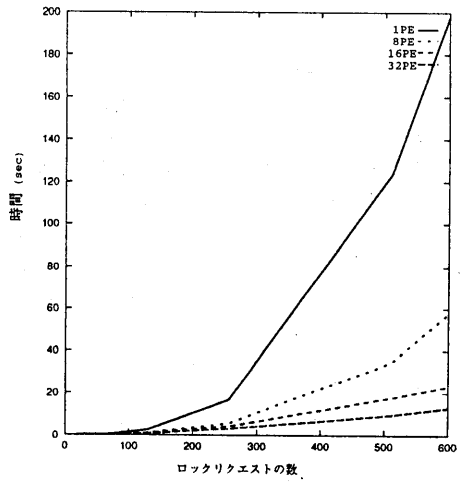

図 3: デッドロック検出に要する時間

4 おわりに

自律的に動作するオブジェクト集合間のメッセージ通信という統一的な枠組で、アクティブルールも扱える並列ネステッドトランザクション管理システムを実現する方法について提案を行なった。並列論理型言語 KL1 によって記述することにより、スマートに実現することが可能である。現在、並列推論マシン PIM/m 上で初歩的な実験を行なっているところである。図 3 に並列デッドロック検出に関する実験の結果を示す。

本研究はこれから検討を要する項目も多い。本文中で述べた検討項目以外にも、例えば、現在の排他／共有ロックのみでなく、意向排他 (intention exclusive) ／意向共有 (intention share) ロック等の導入や、索引とオブジェクトとの同期、データの分散配置、キャッシング等についても、今後検討していく必要がある。

参考文献

[1] 特集. マルチメディアデータベースシステム. 情報処理学会誌, 28(6), 1987.

[2] 特集. オブジェクト指向データベースシステム. 情報処理学会誌, 32(5), 1991.

[3] 牧之内顕文. 次世代データベースシステム. 電子情報通信学会誌, 78(1):65–75, 1995.

[4] 石川博. アクティブデータベース. 情報処理学会誌, 35(2):120–129, 1994.

[5] J. E. Moss. *Nested Transactions.* Series in Information Systems. MIT Press, 1985.

[6] Spector Alfred Z, Randy F.Pausch, and Gregory Bruell. Camelot:A Flexible,Distributed Transaction Processing System. Tech. Rep. CMU-CS-87-129, CMU, 1987.

[7] Barbara Liskov. Distributed Programming In Argus. *Communications of the ACM*, 31(3):300–312, 1988.

[8] Kazuo Taki. Parallel Inference Machine PIM. In *Proc. of FGCS'92*, pages 50–72. ICOT, 1992.

[9] Kazunori Ueda and Takasi Chikayama. Design of the Kernel Language for the Parallel Inference Machine. *The Computer Journal*, 33(6):494–500, 1990.

[10] Tetsuro Fujie et al. KLIC: A Protable Implementation of KL1. In *Proc. of FGCS'94*, pages 66–79. ICOT, 1994.

[11] Moto Kawamura and Toru Kawamura. Parallel Database Management System: Kappa. In *Proc. of FGCS'94*, pages 100–105. ICOT, 1994.

[12] Hiroshi Tsuda and Kazumasa Yokota. Knowledge Representation Language *Quixote*. In *Proc. of FGCS'94*, pages 106–116. ICOT, 1994.

[13] 荒井竜一. 並列論理型言語を用いたトランザクション管理に関する研究. Master's thesis, 北陸先端科学技術大学院大学, 1995.

[14] 横田治夫、野口泰夫、武理一郎. メッセージ通信によるトランザクション処理の並列化. In 並列処理シンポジウム *JSSP '91*, pages 325–332, 1991.

[15] C.L.Forgy. RETE: A fast alogorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19(1):17–37, 1982.

[16] 宮崎 純、横田 治夫. 二つの負荷分散機構を組み合わせたプロダクションシステムの並列化. 電子情報通信学会論文誌 *D-I*, J78-D-I(5):455–466, 1995.