複合オブジェクトに対する索引のオンライン更新が可能な分散管理システム

樋口 健[†] 都司達夫[†] 宝珍輝尚[†]

本論文では複合オブジェクトに対する索引の分散管理システムにおける同時実行制御方式の提案を行う．我々の索引の分散管理システムにおいてはオブジェクト間の参照関係をマルチインデックス手法で索引化し，その索引を非共有メモリ型並列計算機上で分割管理し，検索パス式に基づく問合せ処理を行う．そして，複数の要求を並列処理することで応答時間の短縮が見込まれる．従来の索引システムではオブジェクト更新時にオブジェクトと索引を同時にロックする必要がある．これにより更新時の排他ロックの範囲が広くなり，処理の並列性が低下してしまう．一方，本システムでは更新時のオブジェクトの排他ロックと索引の排他ロックを分離し，また，排他ロックする索引部分を縮小する．したがって処理の並列性への影響はほぼない．また，本システムにおいては同時実行制御における処理のアボートは存在しない．シミュレーションにより従来手法より我々の手法が優れていることを示す．

Distributed Index System for Complex Objects with On-line Modification

Ken Higuchi,[†] Tatsuo Tsuji[†] and Teruhisa Hochin[†]

This paper concerns a concurrency control scheme for distributed index systems for complex objects. In our index system, an index of references between objects is made using the multi-index technique. The index is divided and managed on a shared-nothing parallel computer, and our system retrieves the final result along the specified retrieval path expression. By processing queries in parallel, good response time would be expected. In the usual systems, exclusive-locks on both an object and the corresponding index part are required at the same time. Therefore, the locked index part tends to be large when an object is modified and the number of running processes in parallel would be decreased. On the contrary, our system separates the exclusive-locks on an object from that of the corresponding index part in processing a modification query. In addition, the locked index part is reduced. Therefore, the parallelism would not be significantly influenced. Moreover, in our system, the transaction aborting in the concurrency control never occurs. From the retrieval and modification simulation result, the performance of our scheme is proved to be better than the existing schemes.

1. はじめに

データベースが非常に大規模で，種々の高度な問題を扱い，停止してメンテナンスすることが難しくなっている現在では，オンラインでのデータ再構成は重要な問題である．この問題に対しては関係データベース，オブジェクト指向データベースともに様々な研究がなされている[2),6),11),16),17),20)]．文献 11)，16)，17) ではデータに対する索引に関する再構成の方式が提案されており，文献 20) では参照関係に関するデータの再構成も行っている．

また，データ集合に対して複数の処理を同時に行うための同時実行制御は処理の高速化の点から非常に重要な課題である[1),3)〜5),7),9),10),12),14),19)]．これらの研究においては，一貫性を保ちながらデータの検索とデータの更新の並列性を確保することが主眼である．データベースの再構成と同時実行制御は対象に対して一貫性を保ちながら複数の処理を同時に行うという点では同じであり，その処理の種類が再構成であるか更新であるかの違いであるということができる．

一方，文献 15)，18) では，メッセージ通信非共有メモリ型並列計算機上で，複合オブジェクト集合に対する大容量索引をクラスタリングして分散配置し，並列操作を行うことにより処理効率を向上させる手法が提案されている．これらの手法では検索パス式に基づ

† 福井大学工学部
Faculty of Engineering, Fukui University

き，参照関係に対してマルチインデックス手法[8),13)]を用いて索引を構成し，複数のプロセッサエレメントに索引を分割配置し，並列処理を行っている．

ここで，文献 15)，18) における索引の分散管理システムにおいて，データベース本体のデータの追加，削除，更新にともなう索引の更新を考えてみる．一般的にデータとそれに対応する索引の一貫性を保つためには，データ変更時にはデータとそれに対応する索引を同時にロックする必要がある．このような状況では同時に必要となる排他ロックの範囲が広くなり，排他ロックの競合が起きる確率が高くなる．したがってデッドロック回避のための処理のアボートが起きる確率も高くなり，全体の処理時間が増大する結果となってしまう．この状況を回避する 1 つの方法として索引の排他ロック部分を小さくする方法が考えられる．これは既存の二相ロック方式や時刻印順方式などの同時実行制御手法で可能である．しかし，索引におけるデータ一貫性は保証されるがオブジェクトのデータと索引間の一貫性を保証することはできない（詳細は 3.5 節で述べる）．つまり，既存のデータベース再構成手法や同時実行制御手法ではこの問題の解決にはならない．これらの問題点を考慮し，本論文では，データの排他ロックと索引の排他ロックを同時に行わなくても一貫性が保証されるシステムの構築を目的とする．これを実現する方法として索引においてマルチバージョン管理を行う方法が考えられる．しかし，バージョン管理のために空間的，時間的コストが増加してしまう．そこで，本論文では文献 15)，18) における索引の分散管理システムを基本とし，検索の各段階における処理の終了判定を行うことで索引の検索，更新のタイミングの制御を行うシステムを提案する．我々の方式では索引におけるロックの競合による処理のアボートは存在しない．また，索引の分散管理システムにおける我々の同時実行制御方式と，マルチバージョン管理手法をシミュレーションにより比較し，その有効性を検証する．

なお，本論文は複合オブジェクトに対するデータベースにおける索引の分散管理システムを対象としており，複合オブジェクトがデータベース内でどのように管理，格納されているかは問題としない．

2. 対象オブジェクトと索引

以下では対象とするオブジェクトおよび索引，検索の定義を行う．

2.1 複合オブジェクト

複合オブジェクトの検索パス式を

$$P = C_1 A_1 A_2 \cdots A_N$$

とし，N をこのパスの長さと定義する．ここで，A_1 はクラス C_1 の属性，A_j はクラス C_j の属性であり，$1 \leq j < N$ ならばその参照の定義域は C_{j+1} とする．ここで，各 A_j は単一オブジェクトとは限らず，オブジェクト集合も許す．つまり，C_i が属性 A_i の属性値として複数のオブジェクトを参照することが可能である．また，P の値を

$$o_1 o_2 \cdots o_{N+1}$$

とする．ここで，$o_1, o_2, \ldots, o_N$ はそれぞれ $C_1, C_2, \ldots, C_N$ のインスタンスであり，o_j $(1 < j \leq N)$ はオブジェクト o_{j-1} の属性 A_{j-1} の値である．また，o_{N+1} はオブジェクト o_N の属性 A_N の値であり，単純値またはオブジェクト ID（以下，OID）である．P の値の集合を O_P と表記する．

なお，本論文においては検索パス式は 1 つに限定し，クラス C_i $(1 \leq i \leq n)$ はそれぞれ異なるクラスとし，任意の異なる 2 つのクラスのインスタンス集合は共通部分を持たないものとする．また，各 C_i 間の参照関係が，独立したオブジェクトどうしの参照関係であるか，内包関係であるかは索引上では何ら違いはないため，オブジェクト間の関係の種類は特に限定はしない．

2.2 マルチインデックス

本論文では参照関係に対する索引にマルチインデックス手法[8),13)]を採用する．マルチインデックス手法はパスインデックス，入れ子インデックスなどの他の手法に比べ，検索コストが大きいという不利な点はあるが，更新コストが少ないという利点がある．また，マルチインデックス手法では同じ索引を用いて複数の検索パス式による検索が可能であるという利点がある．検索コストは並列処理による軽減が見込めることから本論文ではマルチインデックスを採用した．

複合オブジェクトに対するマルチインデックスとはオブジェクトの直接的な参照関係の逆関係をそのまま索引要素とするもので，検索はその索引要素を順に検索していくことで行われる（リバーストラバーサル）．

オブジェクト o_j の属性 A_j の値（集合値の場合はその要素）が o_{j+1} であるとき，

$$\langle o_j, o_{j+1} \rangle$$

を P の索引要素という．ここで o_{j+1} はキー値，o_j はデータ値となる．ただし，実際の索引要素のキー値，データ値には OID（または単純値）を用いることとする．さらに，IP をすべての索引要素の集合とし，クラス C_i のインスタンスの OID をキー値とする索引要素の集合を IP_i とする．また，A_N の値をキー値とする索引要素集合を IP_{N+1} とする．

マルチインデックスにおける検索要求 “要素 o_{N+1} を属性 A_N の値としているオブジェクトを検索パス式 P 上で参照している C_1 のインスタンスを求める” は

$o_1 o_2 \ldots o_N o_{N+1} \in O_P$

となる C_1 のインスタンス o_1 の集合を求めることであり，IP を用いて

$\langle o_N, o_{N+1}\rangle, \langle o_{N-1}, o_N\rangle, \ldots, \langle o_1, o_2\rangle$

のようにオブジェクトの被参照関係から求めることである．ここで，検索パス式 P 上で要素 o に対する検索結果のオブジェクト集合を $R_P(o)$ と表す．また，検索パス式 P と A_N の値の集合 S に対して

$R_P(S) = \cup_{o\in S} R_P(o)$

と定義する．

検索パス式を用いた検索においては $R_P(o)$ が一般の検索における単純値検索にあたり，$R_P(S)$ が範囲検索に対応する．一般的な範囲検索では $a \le m \le b$ などの条件で与えられ，$\cup_{a\le o\le b} R_P(o)$ を求める要求である．ここで，$a \le o \le b$ である A_N の値 o の集合 S を前処理で選択することにより，$\cup_{a\le o\le b} R_P(o) = R_P(S)$ となる．本論文における検索とは集合 S に対して範囲検索である $R_P(S)$ を求めることである．

マルチインデックスにおける索引の更新とは，オブジェクトの参照関係の変更にともなう索引要素の変更である．たとえば，o_1 が o_2 を参照し，それに対応する索引要素 $\langle o_1, o_2\rangle$ が存在するものとする．このとき，o_1 の o_2 への参照関係が o_3 へ更新された場合には，索引の更新処理は $\langle o_1, o_2\rangle$ の索引要素を削除し，$\langle o_1, o_3\rangle$ の索引要素を挿入することである．

以下は検索，更新の例である．

例 1 **図 1** は $N = 3$ で，o_1，o_2 はクラス C_1 のインスタンス，o_3，o_4 はクラス C_2 のインスタンス，o_5，o_6 はクラス C_3 のインスタンス，o_7，o_8 はクラス C_4 のインスタンスとなるオブジェクト集合である．ここで，C_1 の属性 A_1 の定義域は C_2 のインスタンス集合，C_2 の属性 A_2 の定義域は C_3 のインスタンス集合，C_3 の属性 A_3 の定義域は C_4 のインスタンス集合，検索パス式を $P = C_1A_1A_2A_3$ とする．

これに対する索引は以下のようになる．

$\{\langle o_1, o_4\rangle, \langle o_2, o_3\rangle, \langle o_4, o_6\rangle, \langle o_5, o_7\rangle, \langle o_6, o_7\rangle\}$

ここで検索 $R_P(\{o_7\})$ の処理は以下のようになる．

1. o_7 に対し，$\langle o_5, o_7\rangle, \langle o_6, o_7\rangle$ を用いて o_5，o_6 を導出．
2. o_5 からは導出なし．o_6 に対し，$\langle o_4, o_6\rangle$ を用いて o_4 を導出．
3. o_4 に対し，$\langle o_1, o_4\rangle$, を用いて o_1 を導出．
4. o_1 は C_1 のインスタンスなので $R_P(\{o_7\}) = \{o_1\}$ として検索処理は終了．

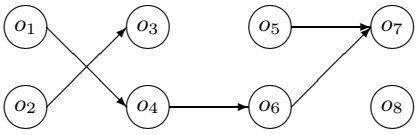

図 1 例 1 のオブジェクト集合
Fig. 1 The set of objects of example 1.

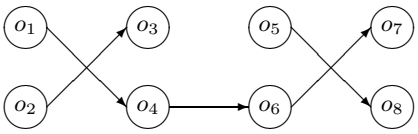

図 2 例 1 のオブジェクト集合（更新後）
Fig. 2 The set of objects of example 1 after modification.

同様にして $R_P(\{o_8\}) = \phi$，$R_P(\{o_7, o_8\}) = \{o_1\}$ である．ここで，図 1 の複合オブジェクト集合の参照関係が**図 2** のように更新され，o_5 から o_7 への参照関係が o_5 から o_8 への参照関係に変更された場合は $\langle o_5, o_7\rangle$ の削除と $\langle o_5, o_8\rangle$ の挿入を同時に行い，

$\{\langle o_1, o_4\rangle, \langle o_2, o_3\rangle, \langle o_4, o_6\rangle, \langle o_5, o_8\rangle, \langle o_6, o_7\rangle\}$

となる．

3. 索引の分散管理システム

前記マルチインデックス手法による索引を分散管理するシステムを提案する．我々の提案するシステムは索引システムであり，オブジェクトデータ本体に関するシステムについては特定しない．

3.1 並列処理環境と処理要求

索引の分散管理システムを構築する並列環境としてメッセージ通信非共有メモリ型並列計算機を仮定する．また，このような計算機のプロセッサエレメント（PE）群とは別に，検索，更新要求を発行し，検索結果を集計するための特別な PE として，HOST が存在すると仮定する．したがって検索時間は HOST から検索要求が発せられてから検索結果が HOST に到着するまでの時間である．また，HOST による処理要求の発行順はシステム全体に対する処理の直列スケジュールに基づいて決定されているものと仮定する．つまり，索引システムにおける処理の直列スケジュールが発行順と等価でない場合，システム全体の処理が直列化不可能となる可能性があるものとする．HOST から送られる検索，更新要求には固有の要求識別子（RID）がつけられているものとする．

RID は時刻印順方式における時刻印の一種と見ることができる．この RID を単なる時刻印ではなく，小

さい順につけた自然数とすることにより，RID を見ることでそれより以前に発行された RID をすべて知ることができる．この情報と 3.3 節の終了判定の情報により更新処理のタイミングを決定することになる．

また，実際のシステムにおいては複数のクライアントが問合せを行うため，HOST を 1 つに限定するのは制限ではあるが，HOST を RID 発行機構としてのみ使用し，PE への実際の問合せの通信をクライアント側が行うことによりこの制限は緩和される．

実際に索引要素を管理し，検索，更新要求を処理する PE を検索 PE と呼ぶ．また，各 PE での処理は 1 プロセスの逐次処理とする．

3.2 索引分割

3.1 節の並列処理環境において索引に対する処理を並列に行うために，IP を分割し，各 PE にそれぞれ格納して管理する分散システムを考える．分割された索引に対する検索処理は以下の 1.〜5. の手順で実行される．

1. HOST は検索条件中の A_N の値をキー値とする索引要素が格納されている検索 PE に対して検索要求を送信する．
2. 各検索 PE は検索要求が到着したならば，その検索条件の OID（または単純値）を自分が管理する索引で検索する．
3. 2. における検索結果がクラス C_1 のインスタンスの OID ならば HOST に検索結果として送信する．そうでない場合（他のクラスのインスタンスの OID）ならば，その OID をキー値とする索引要素を管理する検索 PE に対し，その OID を検索条件とする検索要求を送信する．
4. 各検索 PE に対して検索要求があるかぎり 2. と 3. を繰り返す．
5. すべての検索 PE の処理が終わり，HOST にすべての検索結果が到着したならば検索終了となる．

ここで問題となるのは 1. と 3. において検索要求を検索 PE に送信しなければならない点である．複数のオブジェクトから参照されるオブジェクトが存在する場合は同じキー値の索引要素が複数存在することになる．したがって，無制限に索引を分割した場合，同じキー値を持つ索引要素が複数の検索 PE に分散して管理される可能性がある．たとえば，例 1 においては $\langle o_5, o_7\rangle$ と $\langle o_6, o_7\rangle$ は別の検索 PE に分割管理されることもある．このような分割の場合，$R_P(\{o_7\})$ を求める検索要求に対して HOST は同じ検索要求を 2 つの検索 PE に送信しなければならない．このような重複した通信は検索途中の各検索 PE においても同様に起こる可能性があり，検索処理中のデータ通信量を確実に増大させる．また，更新処理においても処理コストは増加する．索引要素の削除やデータ値の更新の場合，その処理対象の索引要素がどこに存在するか知る必要がある．索引要素の位置を知るための索引が必要となり，索引更新のためのコストの増大を招く．したがって，これらの通信コストの増加，更新コストの増加を防ぐために，索引の分割に「同じキー値の索引要素は同じ検索 PE に格納する」という制限を加える．これにより，検索時に重複する通信は行う必要はない．また，一般的に索引要素数はキー値の数と同じかまたは多いため，この制限により，位置を知るための索引の規模は索引要素数と同程度からキー値の数と同程度に縮小される，

本論文においては上記分割の制限以外には制限は加えず，任意の分割方法で分割された索引を対象にする．

3.3 終了判定

文献 15)，18) では，3.1 節の並列計算機上に 3.2 節の制限に従って分割した索引を分散管理するシステムが提案されている．このようなシステム上で検索が行われる場合，検索結果の OID 集合の要素数は終了時にのみ確定可能であり，HOST は到着した検索結果のみから全体の処理の終了を知ることはできない．したがって，HOST に到着した検索結果以外の情報により処理の終了を判定するシステムが必要である．しかし，文献 15)，18) では処理の終了判定システムに関しては未解決であった．終了を判定する方法としては，各 PE を同期させ，ブロードキャスト通信などで検索が終了したかを問い合わせる方法が考えられるが，非常に通信コストがかかり，現実的ではない．そこで，DETECTOR と呼ばれる PE を導入し，検索の終了を確定する．

検索パス式 P における検索処理は各 IP_i に関する処理に分割して考えることができる．さらに，2.1 節の検索パス式の制限により，すべての検索要求は IP_i の順に処理される．つまり，検索要求である OID 値（または単純値）が HOST から与えられた場合，その値を検索要求として IP_{N+1} を用いた検索処理が行われ，その結果を次の検索要求として IP_N を用いた検索処理が行われる．同様に，各 IP_i を用いた検索処理が順に行われ，最終的に IP_2 を用いた検索処理の結果の C_1 の OID が最終的な検索結果として HOST に送信される．この IP_{N+1} から IP_2 への処理は一方向であり，かならず IP_i 順に行われる．したがって，IP_i に関する処理要求は IP_{i+1} の処理結果（$i = N+1$ ならば HOST からの要求）のみであり，IP_i に関する

処理結果は IP_{i-1} の処理要求（$i=2$ ならば HOST への結果出力）にのみ使われる．そこで，この検索処理の直列性を使って，IP_i 単位で終了判定を行うシステムを考える．

ここで説明を容易にするために各検索 PE を IP_i ごとに論理的に分割して考える．たとえば，検索 PE pe_1 が IP_2 と IP_3 の索引の一部をそれぞれ管理していた場合，pe_1 を論理的に pe_1^2，pe_1^3 と 2 つに分離して考え，pe_1^2 が IP_2 の索引の一部，pe_1^3 が IP_3 の索引の一部を管理しているものと見なす．この論理的に分割された検索 PE を分割 PE と呼ぶことにする．また，IP_i を管理する分割 PE の集合を $PES(IP_i)$ と表記する．定義により分割 PE は 1 つの IP_i に関する部分索引のみを管理していることになる．また，外部からの検索要求とシステム内部で生じる検索要求を区別するために，HOST と検索 PE 間，もしくは検索 PE どうしでメッセージとしてやりとりされる検索要求を内部検索要求と呼ぶことにする．

1 つの検索要求に対する IP_i に関する処理（つまり，$PES(IP_i)$ における処理）は，その検索要求に関する情報のみを考慮すれば，前述の検索処理の直列性により，以下の 2 点の条件が満たされれば終了と判定することができる．

(1) $PES(IP_{i+1})$（$i=N+1$ なら HOST からの内部検索要求の送信）の処理のすべてが終了．
(2) $PES(IP_{i+1})$（$i=N+1$ なら HOST）から $PES(IP_i)$ への内部検索要求の送信メッセージ総数と $PES(IP_i)$ が受信して処理したメッセージ総数が等しい．

ここで，$PES(IP_{i+1})$（$i=N+1$ なら HOST）から $PES(IP_i)$ への内部検索要求の送信メッセージ総数を $PES(IP_{i+1})$ の送信メッセージ総数または $PES(IP_i)$ の受信予定メッセージ総数と呼ぶ．また，$PES(IP_i)$ が受信して処理したメッセージ総数を $PES(IP_i)$ の処理メッセージ総数と呼ぶ．(1) において $PES(IP_{i+1})$（または HOST）の処理がすべて終了したということは，$PES(IP_{i+1})$ から $PES(IP_i)$ への内部検索要求の送信がすべて終了したことであり，$PES(IP_{i+1})$ の送信メッセージ総数が確定したこととなる．つまり，$PES(IP_i)$ の受信予定メッセージ総数が確定したことになり，$PES(IP_i)$ は受信予定メッセージ総数以上のメッセージを受信することはない．したがって (2) において $PES(IP_i)$ の受信予定メッセージ総数と $PES(IP_i)$ の処理メッセージ総数が等しいということは，$PES(IP_i)$ が受け取るべきメッセージはすべて受信し，すべて処理が終了したことを意味する．つまり，その検索要求に対する $PES(IP_i)$ の処理がすべて終了したことになる．そして，$PES(IP_2)$ に関する処理が終了し，$PES(IP_2)$ の送信メッセージ総数が HOST の受信メッセージ総数と等しくなれば検索結果がすべて HOST に到着したことになり，その検索要求に対する処理がすべて終了したと判定される．

これらの処理を行うために各 IP_i に対して DETECTOR と呼ばれる PE を用意し，終了判定に関する情報を処理させる．つまり，RID ごとに，自分が担当する IP_i に関する処理が終了したか否かの判定を行う．ここで，IP_i に関する終了判定用の DETECTOR を D_i とする．

D_i には各 RID ごとに 3 つのカウンタを用意し，その要求に関する情報を逐次更新することで終了判定を行う．

- $PES(IP_i)$ の受信予定メッセージ総数
- $PES(IP_i)$ の処理メッセージ総数
- $PES(IP_i)$ の送信メッセージ総数

また，HOST，各検索 PE，各 DETECTOR は，以下の処理を行う．

- HOST は検索要求を送信後，D_{N+1} にその送信メッセージ総数を知らせる．
- 各検索 PE は受信し，処理したメッセージの数とそれにより発生した送信メッセージ数を対応する DETECTOR に知らせる．
- 各 DETECTOR は検索 PE からの情報に基づいて各カウンタの値を更新する．
- D_i において，$PES(IP_i)$ の受信予定メッセージ総数と $PES(IP_i)$ の処理メッセージ総数が等しくなった場合は，その検索要求に対する $PES(IP_i)$ の処理が終了したものと判断し，D_{i-1}（$i=2$ の場合は HOST）に $PES(IP_i)$ の送信メッセージ総数を知らせる．
- D_i は D_{i+1} または HOST から送られてきた送信メッセージ総数の情報より $PES(IP_i)$ の受信予定メッセージ総数を定める．
- HOST は D_2 からの情報より HOST の受信予定メッセージ総数を定める．
- HOST は受信予定メッセージ総数と受信したメッセージ総数が等しくなった場合は検索が終了したと判定を下す．

これらの処理を各検索要求ごとに行い，各 DETECTOR が対応するカウンタを生成，更新することにより，ある検索要求に対する IP_i に関する処理が終了したか否かの判定が可能となる．

ここで注意すべきことは，上記判定においては，1

つの検索処理のある検索 PE における終了判定ではなく，各検索要求，各 IP_i 単位での終了判定であるということである．つまり，IP_i がどの検索 PE にどのように分割管理されていても終了判定が可能となり，索引分割に対して制限を加えることにはならない．一方，DETECTOR を導入することは，PE 数が有限な並列処理環境では検索 PE 数の減少を意味する．しかし，PE 数がある程度確保できる環境では，検索 PE が増加しても検索処理時間が短くなるとは限らない．なぜなら，検索 PE の増加は 1 つの PE が通信すべき送信先 PE の数の増加を意味し，通信量の増加につながり，全体としての処理時間が増加する可能性があるからである．この DETECTOR の増加にともなう検索 PE の減少による検索処理速度への影響に関しては，4.2 節で実験的評価を行う．

3.4 データへのロックと索引へのロックの分離

複合オブジェクトに対する索引を 3.2 節の制限に従って分割し，3.1 節の並列計算機上に格納し，3.3 節の DETECTOR を用いて終了判定を行うシステムを考える．このような索引システムを用いるデータベース上でオブジェクトのデータが変更される場合は以下の手順が考えられる．

1. データ本体への排他ロック取得
2. 索引への排他ロック取得
3. データの変更
4. 索引の更新
5. ロックの開放

このようなシステムの場合，データ本体への排他ロックと索引への排他ロックを同時に取得する必要がある．排他ロックの粒度に依存するとはいえ，同時に排他ロックする必要のある範囲が広くなり，ロック取得のための処理時間がかかり，デッドロック回避のための処理のアボートの可能性が高くなる．そこで，本論文では，データの追加，削除，更新時における，データ本体に対する排他ロックとそれにともなう索引に対する排他ロックを分離して処理するシステムを提案する．このシステムにおけるデータの変更手順は以下のようになる．

1. データ本体への排他ロック取得
2. データの変更
3. 索引の分散管理システムへの更新要求を送信
4. ロックの開放

ここで前述の手順と大きく異なるのは索引の更新は索引システムに任せ，更新要求を送信するのみで処理を終了することができる点である．これによりデータ本体と索引へのロックの範囲に変化はなくても，同時に必要となるロックの範囲は縮小する．また，索引の更新時におけるロックの競合は索引システム内だけの問題であり，データ本体の処理には関係しない．よって，処理の並列性が高まることが予想される．

一方，このようなシステムにおいては，索引の更新を待たずにデータに対する処理は終了してしまうため，索引システムにおける処理の直列スケジュールとデータ本体での処理の直列スケジュールが等価でない可能性がある．つまり，システム全体としての並列処理が直列化不可能となる可能性がある．そこで，我々の提案する索引システムでは，索引システムにおける処理が HOST による要求の発行順と等価な直列スケジュールになるように制御する方法を次節以降で提案する．この制御により，要求発行順はシステム全体の処理の直列スケジュールに基づいて決定されるとの仮定により，システム全体の処理が直列化可能であることが保証される．

3.5 データと索引の一貫性に関する問題点

3.4 節のデータと索引のロックを分離した手順で処理を行う我々の索引システムでは，システム全体の並列処理が直列化可能であることを保証するために，索引システムにおける処理が HOST による要求の発行順と等価な直列スケジュールになるように制御する必要がある．しかし，複合オブジェクトにおいて検索パス式を用いた検索においては，例 1 に見られるように検索の途中段階で複数のオブジェクトが関係する．このような検索処理に対してマルチインデックス手法を用い，索引を分割管理する場合，HOST が送信した処理要求順に処理対象の PE に到着するとは限らない．特に，検索処理は IP_i の順に検索が行われるため，IP_2 などの検索の後半部分の処理では，内部検索要求が到着する前にその後に HOST から送信された更新要求が到着し，単に到着順に要求を処理するようなシステムの場合，後に発行された更新要求が先に処理されてしまう可能性がある．この処理要求の追い抜きにより要求発行順と実際の索引上での処理順に差が生じ，不適切な検索結果が出力される場合がある．つまり，システム全体としての並列処理が直列化不可能となり，データと索引の一貫性に関する問題が起こる．以下にこの問題点と，既存の同時実行制御方法により解消可能か否かについて考察を行う．

例 1 における図 1 のオブジェクト集合と検索パス式 P に対する検索システムを考える．ここで，この索引要素を IP_i ごとに分割し，それぞれを 1 つの検索 PE に格納して処理を行うこととする．つまり，3 つの検索 PE，pe_1，pe_2，pe_3 がそれぞれ IP_2，IP_3，

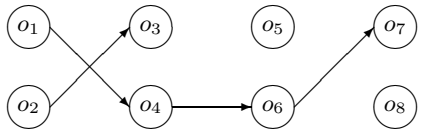

図 3 （2）の処理後のオブジェクト集合
Fig. 3 The set of objects after step (2).

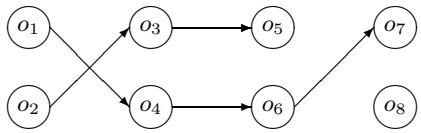

図 4 （3）の処理後のオブジェクト集合
Fig. 4 The set of objects after step (3).

IP_4 を管理し，検索，索引要素の更新を行っているものとする．このとき，以下の（1)〜(3）の処理要求がそれぞれ独立したトランザクションとして連続的に順番に索引システムに与えられ，これら要求が対応する検索 PE に直接送信されるものと仮定する．

(1) $R_P(\{o_7\})$ を求める検索要求
(2) o_5 から o_7 への参照関係が消去されることにともなう更新要求（**図 3** へ変更）
(3) o_3 から o_5 への参照関係が新しくできたことにともなう更新要求（**図 4** へ変更）

ここで，(1)，(2)，(3) の順に処理することが必要であるとする．つまり，(1)，(2)，(3) の処理要求（またはその要求を発行したトランザクション）のデータ本体に関する処理は (1)，(2)，(3) の順の直列スケジュールですでに終了しており，データ本体のみを処理対象とする他の要求に対する処理の影響によりこのスケジュール以外では等価となる直列スケジュールが存在しない場合を想定する．このような場合，索引システムにおける処理の直列スケジュールが (1)，(2)，(3) 以外の順である場合はシステム全体としての並列処理が直列化不可能となる．

これら検索，更新を同時に処理する場合，以下のような順序で実行される場合がある．

1. pe_3 で（1）の検索要求に対して $\langle o_5, o_7\rangle$，$\langle o_6, o_7\rangle$ を用いて o_5，o_6 を導出．
2. pe_3 で（2）の更新処理にともない $\langle o_5, o_7\rangle$ を削除．
3. pe_2 で（3）の更新処理にともない $\langle o_3, o_5\rangle$ を挿入．
4. 1. の結果である o_5，o_6 を pe_2 に検索要求として送信．
5. pe_2 で 4. の検索要求に対して $\langle o_3, o_5\rangle$，$\langle o_4, o_6\rangle$ を用いて o_3，o_4 を導出．
6. 5. の結果である o_3，o_4 を pe_1 に検索要求として送信．
7. pe_1 で 6. の検索要求に対して $\langle o_1, o_4\rangle$，$\langle o_2, o_3\rangle$ を用いて o_1，o_2 を導出．
8. （1）の検索結果として $\{o_1, o_2\}$ を出力．

ここで索引システムにおける処理の直列スケジュールに関して考える．(1) の処理を基準に考えてみると，(3) の処理が (2) の処理を追い抜いて処理されてしまい，索引システムにおける処理順は (3)，(1)，(2) である．これにより，検索結果は $R_P(\{o_7\}) = \{o_1, o_2\}$ となり，(1)，(2)，(3) の順に逐次処理された場合の検索結果である $R_P(\{o_7\}) = \{o_1\}$ とは異なり，誤った結果を出力したことになる．つまり，システム全体の処理が直列化不可能である．

このようなデータと索引の一貫性に関する問題点に関しては既存のデータベース再構成手法，同時実行制御手法では効率的に解消はできない．以下に代表的な解決方法をあげる．

(a) データベース再構成手法を用いる[11),16),17),20)]．
(b) 二相ロック方式を基本とする同時実行制御方法を用いる[3),9),10),12),14)]．
(c) tree-locking などの同時実行制御方法を用いる[1),4),5),19)]．
(d) 検索処理開始とともに処理要求が使うすべての索引要素をロックする．
(e) DAG-locking などを用いて更新時には影響を与える可能性のある部分をすべてロックする[5)]．
(f) 時刻印順方式を用いる[3),9),10),14)]．
(g) マルチバージョン管理を行う[7),10)]．
(h) 各オブジェクト o に対して，$R_P(o)$ の検索に関与するすべての索引要素を同じ検索 PE で管理する．

方式 (a) ではこの問題は解消されない．これらの再構成手法ではデータの位置や参照先のデータに再構成中でも正しく到達できることが主眼であり，検索パス式におけるような参照関係の連鎖である問合せに対して正しい結果が得られる保証はない．

方式 (b) ではこの問題は解消されない．なぜならば，3. および 5. の処理は同一検索 PE における処理であるが，二相ロック方式では，処理の衝突が起きて初めて機能するもので，衝突が起きない状況では何ら制御は行われない．したがって，この例のように 1. の処理中に 3. の処理が終了してしまう場合は pe_2 においてロックの競合などは起きず，到着した順に処理されてしまい，問題の解決にはならない．

方式 (c) ではこの問題は解消されない．なぜなら，これらの手法では木構造のデータ構造を対象とし，あ

るデータへのアクセスの道筋は 1 つであることを前提にし，それによって処理の追い抜きはない，もしくは追い抜き時にはそのことを察知できることが前提で議論されており，本システムのように，あるデータへのアクセスの道筋は 1 つではない場合，追い抜きはあっても，追い抜いたことを更新処理後でしか察知できない場合には対処不能である．

方式（d）を用いた場合,（1）の処理で使用される索引要素は（1）の処理が終了するまでは分からない．したがって，すべての索引をロックする必要があり,（1）の処理が終了するまで（2),（3）の処理は開始することができないことになる．よって，上記問題は発生しない．しかし，索引全体をロックするため，検索と更新の同時実行が行えない．つまり，更新処理の要求によりその前後の処理の並列度が非常に下がる．

方式（e）の場合，ロックの競合時に（1),（2),（3）の順に処理されるよう制御することで上記問題を解決することはできる．しかし，更新処理時にその更新により検索結果に影響するであろう索引要素をデータ値からキー値の方向にロックする必要があり，検索に用いる索引と逆方向の索引を用意する必要がある．しかもデータ更新時にはこれら索引を 2 つとも更新する必要があり，それぞれに DAG-lock が必要となる．よって，かなり広範囲の排他ロックが必要となり，ロックの競合が起きる確率が高くなり，処理がアボートする確率が高くなる．これら逆方向の索引の必要性やアボート処理の必要性により実用的とはいえない．

方式（f）の場合,（1）の要求は pe_2 においてアボートされ，再実行される．したがって,（1）の処理は（3）の処理後に行われ，正しい検索結果となる保証がない．また，全検索 PE に対して（1）のアボートの要求を送らなければならず，非常に高コストな処理が必要となる点などから実用的とはいえない．

方式（g）の場合，マルチバージョンを管理する時間的，空間的コストの問題，バージョンをいつまで保存すべきかの決定などの点で問題があるが，一貫性の問題に関しては解消される．

方法（h）は関連する索引要素が 1 つの検索 PE にまとめられることにより，処理の追い抜きを察知することができ，既存の同時実行制御手法を用いることで問題が解決される．しかし，この方法では分割に強い制限を設けることになる．つまり，最悪の場合は索引が分割不可能になったり，また，索引要素の更新にともない，索引を再分割したりする必要がある．したがって，更新にともない，分割に関する再計算を行わなければならず，更新中には他の処理はまったく行えず，更新に関するコストが非常に高く実用的ではない．

以上の結果より，既存の方式の中で最も効果的でデータと索引の一貫性が保証される方式としてマルチバージョン管理手法があげられる．しかし，マルチバージョン管理を用いる空間的，時間的コストは必要となる．

3.6 索引更新のための十分条件

3.4 節のシステムにおいて，3.5 節で述べた索引更新時のデータと索引の一貫性を保証するための十分条件に関して考察を行う．

データ本体を管理するシステムから索引システムへの索引の更新要求は以下の条件を満たすものとする．

- 更新対象の索引要素が格納されている PE の位置情報が付加されている．
- 索引の更新は索引要素の削除と挿入の 2 種類とする．
- 更新要求は 1 つの索引要素の削除または挿入とする．

索引の更新が起こるのはオブジェクトが生成，削除された場合，オブジェクトの参照関係が変更された場合が考えられる．いずれの操作も索引上では索引要素の削除，挿入に帰着し，これら操作が可能ならばデータ本体の変更を索引に反映することは可能である．また，索引要素が格納されている PE の位置は位置情報の管理システムを設けることにより知ることが可能となり，更新要求メッセージ内に付加することは可能である．以上のような更新要求に対して更新を行うこととする．

3.5 節のデータと索引の一貫性に関する問題を解消するために，本システムでは処理の直列スケジュールが要求発行順，つまり RID 順と等価になることが必要とされている．ここで，索引システムにおける処理の直列スケジュールを RID 順と等価にするための条件を考えてみる．検索処理どうしの順序関係がどのような順序であっても，それらの検索結果には影響はない．一方，更新と検索の順序関係により，その検索結果に影響を及ぼす可能性がある．また，更新処理どうしの順序関係により，更新処理後の検索結果に影響を及ぼす可能性がある．つまり，処理結果に影響を与えるのは更新と検索，更新と更新の順序関係であり，これらの順序関係が RID 順になっていれば索引システムにおける処理の直列スケジュールは RID 順と等価になる．したがって，以下の 4 つの条件が満たされれば 3.5 節のデータと索引の一貫性に関する問題は解消される．

（1）検索要求に対する処理は，それ以前に発行された

更新要求に対する処理の終了後に開始.

(2) 検索要求に対する処理は，それ以後に発行された更新要求に対する処理の開始前に終了.

(3) 更新要求に対する処理は，それ以前に発行された更新要求に対する処理の終了後に開始.

(4) 更新要求に対する処理は，それ以後に発行された更新要求に対する処理の開始前に終了.

ただし，本システムではマルチインデックス手法を採用しているため，上記の (1),(2) の条件をさらに限定可能である．たとえば，検索パス式 $P = C_1A_1 \ldots A_N$ において，更新対象の索引要素を $\langle o_{i-1}, o_i \rangle$ とし，o_i は C_i のインスタンスの OID とする．ここで，P を $P_1 = C_1A_1 \ldots A_{i-1}$ と $P_2 = C_iA_i \ldots A_N$ と 2 つに分割する．このとき，検索要求 S に対し，$R_P(S) = R_{P_1}(R_{P_2}(S))$ であり，変更前の $R_P(S)$ を R_1，変更後の $R_P(S)$ を R_2 とする．

さて，この R_1 と R_2 は以下の関係になる．

- $o_i \notin R_{P_2}(S)$ ならば，更新前と更新後では検索結果に変化はなく，$R_1 = R_2$
- $o_i \in R_{P_2}(S)$ ならば，
  - 更新が索引要素の挿入ならば $R_2 = R_1 \cup R_{P_1}(o_i)$
  - 更新が索引要素の削除ならば $R_1 = R_2 \cup R_{P_1}(o_i)$

つまり，更新処理により変更されるのは $R_{P_1}(o_i)$ に関するところのみであり，o_i 以外の $R_{P_2}(S)$ の要素集合 S' に関しては更新処理に関係せず，$R_1 \supseteq R_{P_1}(S')$ かつ $R_2 \supseteq R_{P_1}(S')$ である．よって，(1),(2) の守らなければならない更新要求と検索要求の処理順序は更新対象の索引要素のキー値に関係する検索のみとなる．したがって，上記 (1),(2) の条件は以下の条件 (1)',(2)' に限定される．

(1)' 内部検索要求に対する処理は，それ以前に発行された更新要求の中で，検索条件と同じ OID をキー値とする索引要素を更新対象とする更新要求に対する処理の終了後に開始.

(2)' 内部検索要求に対する処理は，それ以後に発行された更新要求の中で，検索条件と同じ OID をキー値とする索引要素を更新対象とする更新要求に対する処理の開始前に終了.

この (1)',(2)' と前述の (3),(4) の条件を満足する同時実行制御が可能ならば，索引システムに対する処理の直列スケジュールは RID 順と等価になり，データと索引の一貫性を保証しながらもデータと索引の処理を分離可能なシステムが構築可能となる．

3.7 索引更新方法

3.6 節の (1)', (2)', (3), (4) の条件を考慮し，本方式では DETECTOR の情報を利用することで分割 PE 単位でロックする方式を用いる．つまり，検索 PE で管理している部分索引をさらに IP_i で分割した部分索引単位でロックを行う方法である．また，本論文におけるロックとは索引を格納しているデータ構造のデータに対するロックというよりは，到着した検索，更新要求に対し，一時保留しておく行為を指す．そして，DETECTOR の終了判定情報を利用して，検索，更新のタイミングを決定し，現時点で処理不能な要求に対しては一時保留状態にして他の要求の処理を行う．これにより，更新対象の索引要素を管理する分割 PE (更新対象 PE) 以外では他の処理を同時に行うことが可能となる．また，更新対象 PE においても，RID を比較し，(1)', (2)' の条件を満たす要求に対しては処理を行うことも可能である．つまり，完全な排他ロックではなく，条件付き排他ロックとなる．

ここで，3.6 節の (1)', (2)' の条件を分割 PE 単位で考える．このとき，(1)', (2)' は以下の条件 (1)", (2)" となる．

(1)" 「検索対象の索引要素が分割 PE *ppe* に管理されている検索処理」は，「更新対象の索引要素が *ppe* に管理され，かつ，その検索要求以前に発行された更新要求」に対する処理の終了後に開始.

(2)" 「検索対象の索引要素が分割 PE *ppe* に管理されている検索処理」は，「更新対象の索引要素が *ppe* に管理され，かつ，その検索要求以後に発行された更新要求」に対する処理の開始前に終了.

上記 (1)" は 3.6 節の条件 (1)' の十分条件であり，(2)" は 3.6 節の条件 (2)' の十分条件である．したがって，(1)", (2)" と 3.6 節の (3), (4) を満足する処理タイミングであるならば，データと索引の一貫性を保証することができる．

そこで，これら条件を考慮し，本提案システムでは HOST，DETECTOR，検索 PE における処理を以下のようにする．なお，DETECTOR における処理は D_i における処理として記述する．

HOST

1. 各要求に対して，その RID として自然数を用い，小さい順に連続して付与する.
2. 更新対象 PE に更新要求を直接送信する.
3. 検索条件である OID をキー値とする索引要素が格納している PE に対し内部検索要求を送信する.

4. 発行した処理要求が更新要求であった場合，D_{N+1} に対し，その RID と更新対象 PE の情報を知らせる．
5. 発行した処理要求が検索要求であった場合，D_{N+1} に対し，その RID とその送信メッセージ総数を知らせる．
6. D_2 から受信した終了情報が検索要求に対するものであった場合，その検索要求に対する HOST の受信予定メッセージ総数を決定する．
7. D_2 から受信した終了情報が更新要求に対するものであった場合，その更新が終了したと判定する．
8. 検索要求に対する受信予定メッセージ総数とその検索要求に関する検索 PE からの受信メッセージ数が等しくなった場合，その検索が終了したと判定する．

検索 PE

1. 更新要求を受け取った検索 PE は DETECTOR からの更新許可が到着するまで更新を行わずに保留しておき，更新許可が到着した時点で更新を行う．また，更新終了後に対応する DETECTOR に対し更新終了の情報を送信する．
2. 内部検索要求を受け取った PE では以下の（a)〜（c）の条件のいずれかを満足する場合のみ，その内部検索要求に対する検索を行い，いずれの条件も満たさない場合はその処理を保留する．
   （a）もし保留中の更新要求が存在するならば，その RID より小さな RID を持つ内部検索要求である場合
   （b）もし保留中の更新要求が存在するならば，その更新対象のキー値が属する IP_i に属さない内部検索要求である場合
   （c）保留中の更新要求が存在しない場合
   ただし，保留中の更新要求に対する処理が終了した場合は保留中の内部検索要求に対し，保留すべきかどうか再度検討を行う．
3. 2. における検索結果がクラス C_1 のインスタンスの OID ならば HOST に検索結果として送信する．そうでない場合は，その OID をキー値とする索引要素を管理する検索 PE に対し，その OID を検索条件とする内部検索要求を送信する．
4. 2. において検索を行った場合，その RID，処理したメッセージの数，それにより発生した送信メッセージ数を対応する DETECTOR に知らせる．
5. 保留状態ではない処理可能な要求が存在する限り 2.〜4. の処理を繰り返す．

DETECTOR

1. $PES(IP_i)$ に対する処理がすでに終了した処理要求の RID を保持する．
2. 検索 PE からの検索要求に関する情報を受け取った場合，それに対応するカウンタを生成，更新を行う．
3. D_{i+1}（$i = N+1$ ならば HOST）から検索要求に関する情報を受け取った場合，その送信メッセージ総数の情報より，それに対応する $PES(IP_i)$ の受信予定メッセージ総数を定める．
4. 検索要求に対し，$PES(IP_i)$ の受信予定メッセージ総数と $PES(IP_i)$ の処理メッセージ総数が等しくなった場合は，その検索要求に対する $PES(IP_i)$ の処理が終了したものと判断し，D_{i-1}（$i = 2$ の場合は HOST）にその RID と $PES(IP_i)$ の送信メッセージ総数を知らせる．
5. D_{i+1}（$i = N+1$ ならば HOST）から更新要求に関する情報を受け取り，その更新対象が IP_i に属する場合，更新要求より RID の小さな要求の $PES(IP_i)$ における処理がすべて終了したとき，更新条件が満たされたとして更新対象が格納されている PE に更新許可のメッセージを送信する．
6. D_{i+1}（$i = N+1$ ならば HOST）から更新要求に関する情報を受け取り，その更新対象が IP_i に属さない場合，その更新要求に対する IP_i の処理は終了したものとして D_i の処理を行い，D_{i-1}（$i = 2$ ならば HOST）に更新要求に関する情報を送信する．
7. 検索 PE より更新終了の情報を受け取った場合，その更新要求に対する IP_i の処理は終了したものとして D_i の処理を行い，D_{i-1}（$i = 2$ ならば HOST）に更新要求に関する情報を送信する．

HOST の処理 1. により，2 つの処理要求に対し，RID を比較することで，どちらが先に出された要求であるかを判定することができる．また，RID を見ることにより，それ以前に出された処理要求の RID を知ることができるので，DETECTOR の保持する終了済みの処理要求の RID と比較することにより，更新処理をすることが可能かどうかを判定でき，DETECTOR の処理 5. が可能となる．更新許可を受け取り，検索 PE の処理 1. を行うことで（2)”,（4）の条件を満たすこととなる．また，すべての要求が HOST から順に送信されるという仮定より，HOST の処理 2. により，更新対象 PE に更新要求がそれ以降の要求より必ず先に到着することが保証される．したがって，検索 PE の処理 2. により,（1)” の条件を満たし，検索 PE

の処理 1. により（3）の条件を満たすことになる．つまり，本システムにおいては（1)",(2)",(3),(4）の条件をすべて満足する．

本システムでは上記手法を用いて同時実行制御を行う．ここで 1 つ注意すべきことは，上記処理においては処理が一時的に保留状態におかれることはあっても，処理自体がアボートされることはないという点である．つまり，二相ロック方式や時刻印順方式に見られるデッドロック回避のためのアボートによる処理効率の悪化を防げる．また，更新処理は更新対象 PE において RID 順に処理されるため，索引システム全体に対して RID 順に処理された場合と等しい処理結果になる．つまり，マルチバージョン管理方式と同じ結果であり，索引システムにおける処理は直列化可能であり，その直列スケジュールは RID 順となる．しかし，マルチバージョン管理方式のような，新しいバージョンの作成や，バージョンをどれだけ保存しなければならないかの決定などの処理の複雑さはない．以上の 2 点より，既存の手法の問題点を解消しつつ一貫性を保持する方式であるといえる．

しかし，この同時実行制御手法では次の 2 つの点で問題が残る．DETECTOR の処理 1. により DETECTOR は RID を保持しなければならないことと，検索 PE の処理 2. における排他ロックの粒度である．

DETECTOR の処理 1. により，DETECTOR は RID を保持する必要がある．単純に終了した処理の RID を保持する方法では，システムを長時間稼働させると保持する RID の数が大量になってしまう．そこで，終了した RID の集合 S を直接保持するのではなく，自然数 M と RID 集合 T を保持することにする．ここで，M は $i \in S\ (0 < i \leq M)$ かつ $M+1 \notin S$ となる自然数であり，$T = \{j \mid j \in S, j > M)$ とする．つまり，M 以下の RID と T に含まれる RID の処理は終了したことを意味する．これにより必要とする記憶容量は縮小され，長時間システムを稼働させても問題はなくなる．

また，検索 PE の処理 2. における排他ロックの粒度の観点からいえば，この（1)",(2)" の条件を用いると，3.6 節の（1)',(2)' の条件より排他ロックの範囲が大きくなる．検索 PE の処理 2. における検索要求に対する実行可能か否かの判定は,(1)",(2)" の条件に基づいて分割 PE 単位で制御され，RID どうしの比較と処理対象の属する IP_i どうしの比較によって決定される．一方，3.6 節の（1)',(2)' の条件に基づいて検索 PE の処理 2. を OID 単位の制御に変更した場合，RID どうしの比較と OID どうしの比較になる．ここで，通信回数の軽減のために内部検索要求を RID 単位や IP_i 単位で OID をまとめて送信することを考える．このとき，分割 PE 単位で制御する場合，1 つのメッセージに対して 1 回の比較で決定が可能である．一方，OID 単位で制御する場合は，内部検索要求中のすべての OID に対してこの比較を行う必要があり，比較回数の増加を招く．しかも，1 つの検索 PE は複数の分割 PE の集合であるので，1 つの分割 PE が排他ロックされていても，その他の分割 PE が処理可能ならば，検索 PE としては処理を実行することができ，排他ロックにより検索 PE がアイドル状態にはならない．これらの点を考慮し，本論文においては分割 PE 単位でのロック方法を採用した．

3.8 デッドロックの回避

3.7 節の更新方法を用いた場合，更新要求や内部検索要求が保留状態になる．したがって，デッドロックの可能性について考察する必要がある．

本システムにおいては各検索 PE は処理要求に対して逐次処理を行っている．また，各検索 PE は保留状態ではない処理要求が存在する限り処理を停止しない．したがって，検索 PE において処理要求が存在するのに処理が停止してしまうのは，保留状態にある検索，更新要求のみが存在する場合に限られる．また，HOST からの要求の送信と DETECTOR への受信予定メッセージ総数の送信は必ず終了する．したがって，保留状態となる処理は以下の 4 種類のみである．

1. 検索 PE において，3.7 節の検索 PE の処理 1. において保留された更新許可待ちの更新要求
2. 検索 PE において，3.7 節の検索 PE の処理 2. において保留された更新処理待ちの内部検索要求
3. DETECTOR において，検索 PE または他の DETECTOR からの終了判定情報の未到着による終了確定の保留
4. DETECTOR において，処理要求の終了が確定しないことによる更新許可の保留

また，4 種類の保留状態の直接的原因を考察すると以下のようになる．

- 1. の保留される更新要求は，その RID に関係する 4. の保留状態により発生する．
- 2. の保留される内部検索要求は，その RID より小さな RID を持つ 1. の保留状態の更新要求の存在により発生する．
- 3. の保留される終了確定は同じ RID の 1. または 2. の保留状態か，同じ RID の 3. の保留状態により発生する．
- 4. の保留される更新許可はその更新要求の RID

より小さな RID の 3. の保留により発生する．したがって，デッドロックが起こるとするならば，これら 4 種類の保留状態の連鎖による循環が形成される場合のみである．

ここで，1.〜4. の保留状態の連鎖によりデッドロックが生じ，すべての処理が停止してしまっている状況が存在すると仮定する．循環が形成された保留状態の m 個の処理をそれぞれ $Pr_1, Pr_2, \ldots, Pr_{m-1}, Pr_m$ $(m > 1)$ とし，Pr_i $(1 \leq i \leq m-1)$ の保留となる直接的原因は Pr_{i+1} であり，Pr_m の保留の直接的原因を Pr_1 とする．また，$RID(Pr_i)$ を Pr_i に関する RID とする．ここで，$RID(Pr_i)$ と $RID(Pr_{i+1})$ $(1 \leq i \leq m-1)$ について考える．Pr_i および Pr_{i+1} が 1.〜4. のどの状況であれ，上記の保留の直接的原因における RID の関係により $RID(Pr_i) \geq RID(Pr_{i+1})$ となる．同様に $RID(Pr_m) \geq RID(Pr_1)$ となる．つまり，$RID(Pr_1) \geq RID(Pr_2) \geq \cdots \geq RID(Pr_{m-1}) \geq RID(Pr_m) \geq RID(Pr_1)$ である．したがって，$RID(Pr_1) = RID(Pr_2) = \cdots = RID(Pr_{m-1}) = RID(Pr_m)$ となる．一方，保留の直接的原因の処理に関する RID と保留状態の処理に関する RID が等しい保留状態は 1. と 3. のみである．しかし，3. は 1. の保留の直接的原因とはなりえず，また，1. どうしでも直接的原因とはなりえないため，循環を形成するためにはすべての Pr_i は 3. の保留状態でなければならない．一方，同じ RID における終了判定の情報の送信はその処理要求が検索であっても更新であっても D_{i+1} から D_i への一方向である．つまり，$i \leq j$ である D_i から D_j へ送信されることはない．したがって，3. の保留状態の処理の連鎖が循環を形成することはない．つまり，いかなる場合においても保留状態の処理の循環はありえず，デッドロックは発生しない．よって，デッドロック回避のための処理のアボートも存在しない．

4. 計算機シミュレーション

計算機シミュレーションにより，本方式の有効性について検証を行う．本シミュレーションでは計算機上に実現した仮想的な並列計算機上でのシミュレーションであり，単位時間を基準としてイベントの発生時刻と終了時刻を決定することで総処理時間を計測するものである．また，実験結果は単位時間で表すものとする．

4.1 対象複合オブジェクトおよび索引

以下の条件の複合オブジェクト集合を 3 つ用意する．

- 索引の対象となる複合オブジェクトは $N = 8$ で各クラスのインスタンス数は 10,000.
- 各オブジェクトはたかだか 2 個のオブジェクトを参照する．ただし，参照数および参照先オブジェクトは等確率でランダムに決定し，2 つの場合は異なるオブジェクトを参照する．

上記の複合オブジェクト集合をマルチインデックス手法で索引化し，一様乱数に基づいて検索 PE に分割する．1 つの複合オブジェクト集合に対し，乱数系列を変更して 3 つの索引をそれぞれ作る．したがって，対象索引は 9 つとなり，それぞれに対してシミュレーションを行う．

4.2 実 験 1

DETECTOR の導入による検索 PE 数の減少にともなう検索時間の変化について考察を行うために，検索 PE 数と検索時間に関する計算機シミュレーションを行う．

4.2.1 検索シミュレーション

並列処理環境の特性値として以下のことを仮定した．

1. 検索用 PE 数は $8n$ であり，n を 1 から 20 まで変更する．
2. 通信パケットは最大 100 の OID を格納可能である．
3. C_s を通信初期化時間，C_t を 1 つの OID を送信するのに必要な時間，T_r を 1 つの索引要素を検索するのに必要な時間としたとき，その時間比を $C_s : C_t : T_r = 100 : 1 : 100$ とする．これらパラメータは C_t を 1 単位時間として設定する．
4. 要求発行間隔は 2500 とし，HOST は 20 要求を発行する．
5. 各オブジェクトが検索要求に含まれる確率は 0.01 とする．つまり，各クラスのインスタンス数が 10000 であるので，1 つの検索要求は平均 100 個の OID を検索対象とする範囲検索である．

以上の状況でのシミュレーションを各索引に対して 3 回ずつ行う．ここで，3. の時間比は IBM 製並列計算機 RS/6000SP 上で通信に MPI ライブラリ，索引要素の格納に B^+ 木を使用した場合の実測値を参考に決定した．また，4. の要求発行間隔は 1 つの検索要求に対する処理時間の 1/20 程度の短い間隔に設定している．

4.2.2 検索シミュレーション結果

図 5 は総処理時間の平均値を表している．図 5 より検索 PE 数が 8〜54 の範囲では PE 数が増加するにつれ検索時間が減少するが，54〜160 の範囲では PE 数の増加により総処理時間が増加することが分かる．これは，PE 数が増加することにより，各 PE の検索回数は減少するが，内部検索要求の送信先 PE 数の増加

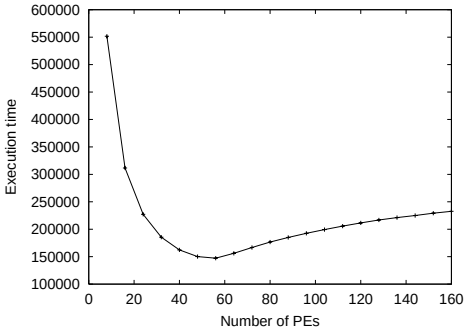

図 5 PE 数と総処理時間
Fig. 5 Number of PEs and execution time.

を招き，通信コストの増加を引き起こすことによるものである．最適な PE 数は索引分割方法，検索コスト，通信コスト，検索条件などにより変化するが，十分に PE が確保できる並列計算機ならば，DETECTOR の導入による検索 PE の減少にともなう処理時間の増加はほとんど問題にならないことが分かる．

4.3 実 験 2

3.5 節の議論によりデータと索引の一貫性を保持し，実用的なシステムとしてはマルチバージョン管理を行う手法があげられる．その他の手法では効率的な並列処理が行えない，またはデータと索引の一貫性を保証できず，比較対象としては適切ではない．そこで，本方式とマルチバージョン管理方式の 2 つのシステムを比較し，本方式の有効性を検証する．同時実行制御方式の比較を厳密に行うために，両方式とも索引分割，DETECTOR などの基本部分は同一のものを使用し，更新処理を行うタイミングの判定に両方式を用いたものを比較する．マルチバージョン管理方式では，更新処理にともなう新しいバージョンの作成は，更新要求が更新対象 PE に到着時に行われるものとした．ただし，マルチバージョン管理方式の時間的管理コストとして，検索 PE における 1 つの OID に対する検索時間，索引要素の削除（挿入）に必要な時間を一定割合で付加することとする．また，本シミュレーションは時間コストに関するシミュレーションであり，空間的コストは考慮していない．

4.3.1 検索シミュレーション

並列処理環境の特性値として以下のことを仮定した．

1. 検索 PE 数は 64，DETECTOR 数は 8 とする．
2. 通信パケットは最大 100 の OID を格納可能である．
3. C_s，C_t，T_r の時間比として
   case1（$C_s : C_t : T_r = 100 : 1 : 100$），
   case2（$C_s : C_t : T_r = 100 : 1 : 1000$），
   case3（$C_s : C_t : T_r = 100 : 1 : 10000$），
   の 3 つの場合を仮定する．また 1 つの索引要素を削除または挿入するのに必要な時間は T_r と等しいものとする．これらパラメータは C_t を 1 単位時間として設定する．
4. 要求発行間隔は case1 では 2500，case2 では 2500，case3 では 10000 とし，HOST は 20 要求を発行する．
5. 要求が更新である確率を 0 から 1 まで 0.1 きざみで変更して行う．
6. 各オブジェクトが検索要求に含まれる確率は 0.01 とする．
7. マルチバージョン管理方式の時間的管理コストはそれぞれ T_r の 0%，10%，20%，50%の増加の 4 つの場合について比較を行う．つまり，case1 の場合，それぞれ $Tr = 100$，$Tr = 110$，$Tr = 120$，$Tr = 150$ となる．

以上の設定でのシミュレーションを各索引に対して 3 回ずつ行う．ここで，3. の case1，2，3 は通信が高速な場合，低速な場合，索引の格納が一次記憶の場合，二次記憶の場合などの状況を想定して時間比を決定した．また，十分に大きな索引においては Tr と索引要素の削除または挿入にかかる時間は平均的にはほぼ等しくなることが予想されるため，索引要素の削除または挿入にかかる時間を Tr とした．4. の各要求発行間隔は 1 つの検索要求に対する処理時間の 1/20 程度の短い間隔に設定した．

4.3.2 検索シミュレーション結果

図 6，**図 7**，**図 8** は各 case における総処理時間の平均値である．pl は本方式による同時実行制御を行った場合，mvc-X は X%の管理コストでマルチバージョン管理を行った場合の結果である．

case1 の結果より，マルチバージョン管理方式の場合，管理コストによらずほぼ同じ処理時間で終了している．これは，case1 のような状況では，検索に対する処理時間が通信コストに大きく依存することを表している．実際に，1 つの検索要求において，通信回数と検索回数の比は約 2:1 であり，50%程度の検索時間の増大では全体の処理時間には影響はなかったものと考察する．一方，本方式とマルチバージョン管理方式との比較においては，ほぼ同じではあるがわずかに総処理時間が長くなった．これは，本方式においてロックの粒度が小さいとはいえ，到着した要求を保留する状況が発生するならば，検索 PE が何も処理をしない状況が生じ，それが全体の処理に影響を与えてしまうことがあることを示唆している．

一方，case2，case3 ではマルチバージョン管理方式

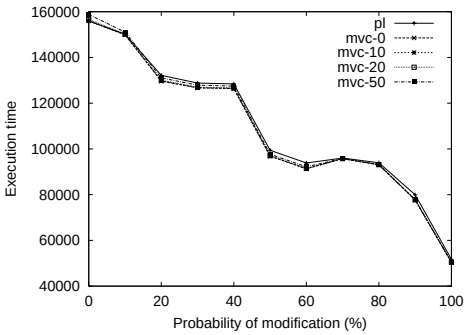

図 6 case1 における総処理時間
Fig. 6 Execution time for request of case1.

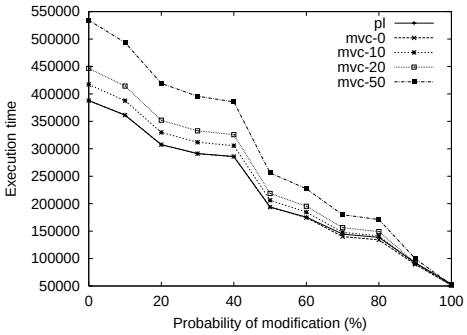

図 7 case2 における総処理時間
Fig. 7 Execution time for request of case2.

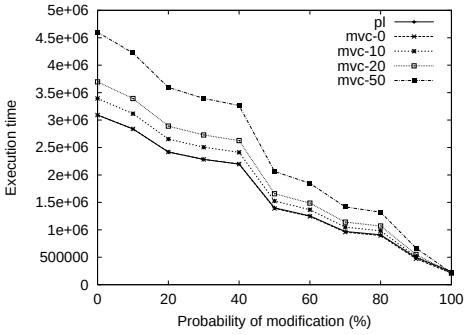

図 8 case3 における総処理時間
Fig. 8 Execution time for request of case3.

の場合，管理コストにより総処理時間に変化がみられる．これは，case1 とは違い，検索に対する処理時間が検索コストにも大きく依存することを表している．このような状況においても本方式の総処理時間はバージョン管理の時間コストが 0%のものとほぼ同じであった．これは，通信コストが検索コストより小さいことにより，検索 PE ではつねに複数の処理待ちの要求が存在する状態になり，更新処理が更新許可待ちの状態になる場合においても，処理可能な検索要求に対する処理が実行されることで，検索 PE がアイドル状態になることがほとんどなく，総処理時間においては差がなくなったものと考察する．したがって，バージョン管理の時間コストが増えれば増えるほど本方式とマルチバージョン管理方式では総処理時間において差が広がり，本方式の優位性が認められる．

結論として，シミュレーションで検証を行ったすべての場合において，本方式は，マルチバージョン管理において時間的管理コストが 0 のものとほぼ同じ総処理時間である．したがって，マルチバージョン管理における時間的管理コストの増加による総処理時間の増加の可能性や，バージョン管理のための空間コスト，処理の複雑化の点から，本方式はマルチバージョン管理手法より優れていると結論付けることができる．

5. ま と め

複合オブジェクトに対する索引を分散並列処理するシステムにおいて，データと索引の一貫性を保証しながらデータに対する処理と索引に対する処理を分離するための同時実行制御方式の提案を行った．本索引分散管理システムを用いることで，データの変更にともなう索引の変更は，システム障害などを除いて，索引システムのみで安全に確実に行われる．本方式は要求識別子と DETECTOR の終了判定情報を用いて更新のタイミングを決定し，分割 PE 単位で排他ロックを行う方法であり，デッドロック回避のための処理のアボートは存在せず，マルチバージョン管理システムと同等の処理結果を保証する．本方式とマルチバージョン管理手法による同時実行制御をシミュレーションで比較した結果，時間コストに関しては同等以上の処理能力であり，バージョン管理の空間コストと処理の複雑さを考慮するならば，本方式はマルチバージョン管理手法より優位であることを示した．

しかし，いくつかの点において本システムは汎用性が高いとはいいきれない．(1) 検索パス式や複合オブジェクトに制限がある．(2) 大量な処理要求が到着した場合に HOST の処理によるボトルネックが発生する．(3) 検索結果に基づいて更新を行う場合などの処理要求に対しては，システム全体での処理の直列化可能性について考慮する必要があり，単純に処理要求の発行順を決定することはできない．したがって，これらの制限を解除し，より一般的な処理要求に対応可能なシステムの開発が今後の課題である．

参 考 文 献

1) Bayer, R. and Schkolnick, M.: Concurrency of Operations on B-Trees, *Acta Informatica*,

Vol.9, pp.1–21 (1977).
2) Söderlund, L.: Concurrent Data Base Reorganization — Assessment of a Powerful Technique through Modeling, *Very Large Data Bases, 7th International Conference*, pp.499–509 (1981).
3) Bernstein, P.A. and Goodman, N.: Concurrency Control in Distributed Database Systems, *ACM Computing Surveys*, Vol.3, No.2, pp.185–221 (1981).
4) Lehman, P.L. and Yao, S.B.: Efficient Locking for Concurrent Operations on B-Trees, *ACM Trans. Database Syst.*, Vol.6, No.4, pp.650–670 (1981).
5) Bernstein, P.A., Hadzilacos, V. and Goodman, N.: *Concurrency Control and Recovery in Database Systems*, Addison-Wesley (1987).
6) Markowitz, V. and Makowsky, J.: Incremental Reorganization of Relational Databases, *Proc. 13th International Conference on Very Large Data Bases*, pp.127–135 (1987).
7) Aspnes, J., Fekete, A., Lynch, N.A., Merritt, M. and Weihl, W.E.: A Theory of Timestamp-Based Concurrency Control for Nested Transactions, *Proc. 14th International Conference on Very Large Data Bases*, pp.431–444 (1988).
8) Bertino, E. and Kim, E.: Indexing techniques for queries on nested object, *IEEE Trans. Knowledge and Data Eng.*, Vol.1, No.2, pp.196–214 (1989).
9) Carey, M.J. and Livny, M.: Parallelism and Concurrency Control Performance in Distributed Database Machines, *Proc. 1989 ACM SIGMOD International Conference on Management of Data*, pp.122–133 (1989).
10) Barghouti, N.S. and Kaiser, G.E.: Concurrency Control in Advanced Database Applications, *ACM Computing Surveys*, Vol.23, No.3, pp.269–315 (1991).
11) Salzberg, B. and Dimock, A.: Principles of Transaction-Based On-Line Reorganization, *Proc. 18th International Conference on Very Large Data Bases*, pp.511–520 (1992).
12) Muth, P., Rakow, T.C., Weikum, G., Brössler, P. and Hasse, C.: Semantic Concurrency Control in Object-Oriented Database Systems, *Proc. 9th International Conference on Data Engineering*, pp.233–242 (1993).
13) Bertino, E.: A survey of indexing techniques for object-oriented database systems, *Query Processing for Advanced Database*, Freytag, J.C., Maier, D and Vossen, G. (Eds.), pp.383–418, Morgan Kaufmam (1995).
14) Nørvåg, K., Sandstå, O. and Bratbergsengen, K.: Concurrency Control in Distributed Object-Oriented Database Systems, *Proc. 1st East-European Symposium on Advances in Databases and Information Systems* (ADBIS '97), pp.9–17 (1997).
15) 小倉一泰，都司達夫，ブレト アルベルト，宝珍輝尚：複合オブジェクトの索引に対する水平垂直分割の一方式，信学論，Vol.J80-D-I, pp.486–494 (1997).
16) Zou, C. and Salzberg, B.: Safely and Efficiently Updating References During On-line Reorganization, *Proc. 24th International Conference on Very Large Data Bases*, pp.512–522 (1998).
17) Achyutuni, K., Omiecinski, E. and Navathe, S.: Two techniques for on-line index modification in shared nothing parallel database, *Proc. 1996 ACM SIGMOD International Conference on Management of Data*, pp.124–136 (1996).
18) 樋口　健，小倉一泰，都司達夫，宝珍輝尚：複合オブジェクトに対する索引の分割を決定する確率アルゴリズムの実験的評価，信学論，Vol.J82-D-I, No.1, pp.14–23 (1999).
19) Haritsa, J.R. and Seshadri, S.: Real-Time Index Concurrency Control, *IEEE Trans. on Knowledge and Data Engineering*, Vol.12, No.3, pp.429–447 (2000).
20) Lakhamraju, M.K., Rastogi, R., Seshari, S. and Sudarshan, S.: On-line Reorganization in Object databases, *Proc. 2000 ACM SIGMOD International Conference on Management of Data*, pp.58–69 (2000).

(平成 14 年 6 月 19 日受付)
(平成 14 年 10 月 4 日採録)

(担当編集委員　掛下 哲郎)

樋口　健（正会員）

平成 4 年電気通信大学電子情報学科卒業，平成 9 年同大学大学院博士後期課程修了．平成 9 年 4 月より福井大学工学部情報工学科助手．現在，同情報・メディア工学科助手．博士（工学）．オートマトン，分散 OODBMS の研究に従事．電子情報通信学会会員．

都司 達夫（正会員）

昭和 48 年大阪大学基礎工学部電気工学科卒業．昭和 53 年同大学大学院博士課程修了．同年福井大学工学部情報工学科講師．現在，同情報・メディア工学科教授．工学博士．データベースシステム，プログラミング言語の研究に従事．著書 “Optimizing Schemes for Structured ProgrammingLanguage Processors”（Ellis Horwood）．電子情報通信学会，IEEE，日本情報考古学会各会員．

宝珍 輝尚（正会員）

昭和 57 年名古屋工業大学電気工学科卒業．昭和 59 年同大学大学院修士課程修了．同年，日本電信電話公社入社．NTT 情報通信網研究所を経て，平成 5 年 7 月より福井大学工学部情報工学科助手，現在，同情報・メディア工学科助教授．博士（工学）．マルチメディアデータの管理，印象に基づくマルチメディアデータ検索，科学技術データベース，拡張可能データベース管理システム，グラフィカルなデータベース問合せ，グラフに基づくデータモデルの研究に従事．電子情報通信学会，IEEE，ACM，日本情報考古学会，日本感性工学会各会員．