パケット充填率を考慮した複合オブジェクトの並列索引検索

堀江　武† 右近　拓弥† 都司　達夫†† 樋口　健††

† 福井大学大学院工学研究科　〒 910–8507 福井市文京 3–9–1
†† 福井大学工学部情報・メディア工学科　〒 910–8507 福井市文京 3–9–1
E-mail: †{horie,ukon,tsuji,higuchi}@pear.fuis.fukui-u.ac.jp

あらまし　複合オブジェクトの並列検索のため，マルチインデックスを用いて索引付けを行い，索引を分割して複数プロセッサ (PE) に割り当て，パイプラインによる索引検索処理と分岐履歴を用いた終了判定法で実現する．このとき索引の水平分割により PE 間通信で検索結果を格納するパケット充填率が低下し，生成パケット数の増大を引き起こす．このような充填率低下の問題を回避するためにパケット統合の方式とそれに基づく新たな並列検索終了判定方式を提案し，その評価を行う．

キーワード　インデックス, オブジェクト指向 DB, 性能評価, マルチインデックス, 並列処理

Parallel Retrieval for Complex Object Index with Improved Packet Filling Method

Takeshi HORIE†, Takuya UKON†, Tatsuo TSUJI††, and Ken HIGUCHI††

† Graduate School of Engineering, University of Fukui　Bunkyo 3–9–1, Fukui-shi, Fukui, 910–8507 Japan
†† Department of Information and Media Engineering, University of Fukui　Bunkyo 3–9–1, Fukui-shi, Fukui, 910–8507 Japan
E-mail: †{horie,ukon,tsuji,higuchi}@pear.fuis.fukui-u.ac.jp

Abstract　For parallel retrieval of complex object index, multiindexing technique is employed. By splitting the multiindex into subindices and placing each of them on a separate processor elements, pipelined parallel index retrieval scheme is designed with retrieval termination detection scheme based on branch history. But, index splitting causes the problem that the density of the packets involving the intermediate retrieval results become low and the number of the produced packets would increases. In this paper, we propose the schemes of integrating the packets for improving the packet density, and incorporate the new termination detecting algorithms that work under this integrating environment, and evaluates the proposed scheme.

Key words　index, OODB, performance, multi-index, parallel processing

1. はじめに

継承関係や入れ子の集約関係による複雑・大規模な複合オブジェクトを高速に検索するための手段として，高速な索引付けの研究がある．これらの索引技法には入れ子インデックス (nested index)，パスインデックス (path index)，マルチインデックス (multi index) などがある [1][2][3]．さらに，これらの索引技法に基づいた研究が数多く行われている（たとえば, [4][5][6][7]）．

オブジェクト指向データベースシステムの場合には，フラットな構造のみを扱う関係データベースシステムにくらべて，その索引構造は複合オブジェクト本体の構造を反映して，格段に複雑であり，構造マッチングやナビゲーションを必要とするために，索引検索のコストが高い．以上の研究のほとんどは単一マシン上の索引が前提となっているが，特に検索コストの高い複合オブジェクト索引の場合には並列検索の技法を開発することは固有の困難さはあるものの，実用速度を得る上で重要である．

複合オブジェクトの本体そのものを複数のマシンに分割して並列検索による効率向上をねらった研究も種々の視点から数多く行われている（たとえば, [8][9][10][11]）．一方, [12] や [13] では複合オブジェクト本体ではなく，その索引の分割と並列検索について論じている．特に，大規模・大容量の複合オブジェクト集合の検索の場合には索引検索の高速化手法を開発することは複合オブジェクトの検索に対して，効果が高いといえる. [12] では索引構造をハイパーキューブにマッピングした場合の並列検索アルゴリズムが提案されている．また, [13] では複合オブジェクトの検索パス式上の索引を複数マシンに分割して配置することにより，索引検索の効率向上を目的とした並列検索方式

と最適な索引の垂直水平分割の方式が提案されている．

以上の研究のほとんどは，一つの検索要求を処理する場合, [8] [9] や [13] に代表されるように，各マシンが自分の持ち分のオブジェクトや索引を検索して結果を関連する他のマシンとは独立に非同期に転送する方式を採用している．このような非同期並列検索の重要な問題点として，部分的な検索結果が到着する順序やタイミングが並列マシンの使用環境や検索対象オブジェクト集合（索引集合）によって異なり，予測できない．また，検索結果の総数も通常，前もって決定できない．このために，検索が全体として終了したことを検知する必要がある．すなわち，並列検索システムを構築するときには，検索最終段の各マシンから報告される検索結果がすべて出揃ったことを判定する効率的な検索終了判定アルゴリズムとその実現が必要となる．ところが，以上の研究においてはこの問題がほとんど取り上げられていない．並列検索の終了を効率よく判定することは，特に，並列検索プロセスが占有するメモリバッファなどの資源の開放を速やかに行うことにより，検索プロセスそのものを軽量化するために重要である．

[14] では，検索結果（オブジェクト ID(OID) の集合）がいくつかのパケットに分岐されて送出されてきたかを表す分岐履歴を各パケットに付与する方法によるアルゴリズムを提案している．しかしこのアルゴリズムは，実装時に生じるパケット充填率低下の問題を扱っていない．充填率の低下により，生成パケット数が増大し，(1) パケット通信コスト　(2) 入出力バッファの占有メモリ　(3) 終了判定負荷　の増大を引き起こし，結果的に検索速度とメモリコストを劣化させる．

本論文では，パケット充填率を上げることにより，これらの問題点を解決することを試みる．充填率を上げる際に必要となる新たな終了判定法を提案し，それに基づく索引並列検索システムを実装し，評価を行う．なお，ここではパイプライン並列性に対して適応性の高いマルチインデックスを採用している．

本文 2. では必要な諸定義を示し，3. では並列検索の終了判定について述べる．4. では，パケット充填率低下の問題について述べ，充填率向上のためのパケット統合方法および終了判定のアルゴリズムを提案する．また 5. ではパケット統合の導入に伴って解決すべき送受信の処理方法について述べる．更に，6. ではこれらの方法に基づき，SMP マシン上のマルチスレッド機能を使用した，性能評価用索引並列検索システムの実装と評価について述べ，提案した検索終了判定方法の有効性について述べる．最後に 7. で，まとめと今後の課題について述べる．

2. 索引の並列検索と検索終了判定

2.1 複合オブジェクトとパスの定義

ここでオブジェクトを 0 個以上の属性からなる集合値とする．個々のオブジェクトはオブジェクト識別子（以下，OID）で一意に識別される．属性は一意に識別され，その値は各属性ごとに定めた唯一の型に属し，単一値の他，集合値を認める．型はクラスまたは単純値型とする．クラスは，各クラスが定める属性集合を全て有すオブジェクトの集合とする．単純値は，OID，属性，型以外を表す任意の値とし，単純値型は，単純値の集合とする．またクラス C に属すオブジェクトを C のインスタンスと呼ぶ．クラス C（のインスタンス）が属性 A を備え，属性 A の型が C またはそれ以外のクラスであるような C のインスタンスを複合オブジェクトと呼ぶ．

次に複合オブジェクトのパスを

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

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，$1 \leqq j < N$ ならばその参照の定義域は C_{j+1} とする．

また，P の値を $o_j (1 \leqq j \leqq n+1)$ のリスト

$$(o_1, o_2, \cdots, o_{n+1})$$

とする．ここで $o_1, o_2, \cdots, o_n$ はそれぞれ $C_1, C_2, \cdots, C_n$ のインスタンスであり，$o_j (1 < j \leqq n)$ 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 の値である．

P の値集合を 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 $(1 \leqq i \leqq n)$ 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．各 C_j 間の属性に基づく参照関係は，それが独立したオブジェクト同士の通常の関連であるか集約に分類される関連であるかは，索引上では違いがないため区別しない．

2.2 マルチインデックス

マルチインデックス方式は，パスインデックス，入れ子インデックスなどの他の手法と比べて検索コストが大きいという不利な点はあるが，索引構造上，索引集合の分割単位が明瞭でパイプライン並列処理への適応が高く，また同じ索引を複数の検索パス式で利用できる利点から採用している．

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する行為となる．

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 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 を全ての索引要素の集合とする．

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 $a \leqq m \leqq b$ 等の条件で与えられ，$\cup_{a \leqq o \leqq b} R_P(o)$ を求める要求である．ここで，$a \leqq o \leqq b$ 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，$\cup_{a \leqq o \leqq b} R_P(o) = R_P(S)$ となる．

本研究での検索とは集合 S に対して範囲検索である $R_P(S)$ を求めることである．

2.3 索引分割と並列処理

全ての索引集合 IP は，キー値またはデータ値が属する検索パスに含まれる属性 A_j のクラス C_j によって分類できる．

初めに IP をこの特徴によって分割する．$1 \leqq j \leqq n$ に対して，

$$V_j = \{\langle o_j, o_{j+1} \rangle \mid \langle o_j, o_{j+1} \in IP,\ o_j \in C_j, o_{j+1} \in C_{j+1}\}$$

とする．ここでは V_j への分割を垂直分割と呼ぶ．次に各 V_j をさらに分割し，各々の分割要素を $HV_i^j (1 \leqq i \leqq m_j)$ とする．m_j は V_j あたりの分割数である．V_j から HV_i^j への分割を水平分割と呼ぶ．

HV_i^j は計算機リソースを抽象化した PE(Processor Element) PE_i^j に格納され，管理される．(図 1)

ここで各々の PE は他の PE とは独立して，非同期に並列動作する．検索手順は以下のようである．(図 1)．

1. 検索条件で指定されているキー値を含む V_n 上のすべての PE に検索要求を出す．
2. 各 PE の動作は以下の通りである．

2.1. 検索要求を受け取ったら自分の管理している索引要素集合について検索を行う．

2.2. 2.1. の検索結果のそれぞれのオブジェクトについて，そのオブジェクトをキーとする索引要素を管理している PE に対して検索結果を新たに検索キーとして検索要求を送る．

2.3. 自分が V_1 の一部を管理している場合，ここでの検索結果が求めたい検索結果の一部なので結果として返す．

3. 2.3. で得たすべての PE の検索結果の集合和を最終検索結果として返す．

2.4 検索終了判定問題

前節の検索手順のステップ 2.2 では，通信コストの削減のために，検索結果のオブジェクトの OID は通常，パケットに詰められ，次段の該当 PE に送信される．ここで，”次段の該当 PE” の決定は (1) パケット中の OID にその情報が含まれる場合，(2) 外部のオブジェクトの分散情報を参照する場合，(3)OID に関するハッシュ関数を計算する場合（ハッシュ・ジョイン [15]) などがあり得るが，その決定の方法についてはここでは，問わない．

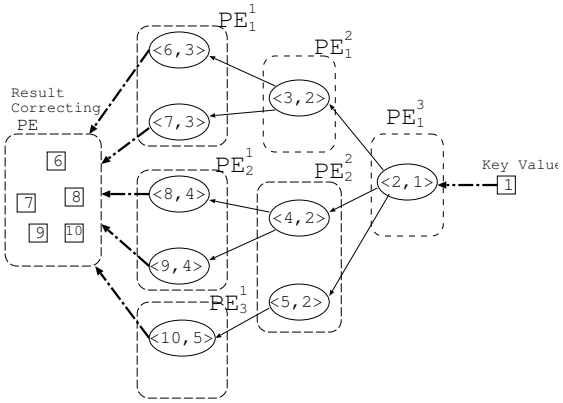

図 1　索引分割と並列検索
Fig. 1　Index partition and parallel retrieval

検索最終段の PE 集合から検索結果のパケットを集め終了判定を行う終了判定 PE を用意して，パケットに付与された” 分岐履歴” を検査することにより，すべての検索結果のパケットが揃ったかを判定する．各段の PE は次のような規則で分岐履歴を送信パケットに付与して次段の PE に送る．

3. パケットの分岐履歴と終了判定

3.1 分 岐 履 歴

[14] では，パケットに付与する分岐履歴の定義を以下のように行った（[定義 1]）．

(a) 1 段目の検索開始パケットの分岐履歴は空列 (ϵ)．

(b) 前段の PE から渡された検索結果のパケット P の分岐履歴が $\langle b_i, \ldots, b_1 \rangle$ であるとする．P 中の各 OID をキー値として検索を行う．検索結果は複数のパケットに順次格納され，それぞれ次段の該当 PE に送出される．最後のパケット以外については，分岐数がわからないので $unknown$ （未定）として，分岐履歴を $\langle unknown, b_i, \ldots, b_1 \rangle$ とする．最後のパケットが確定したときには分岐数 $p (p \geqq 1)$ が確定するので，最後のパケットは分岐履歴を $\langle p, b_i, \ldots, b_1 \rangle$ とする．

この定義は分岐の状況を直感的に表示し，分かり易い．しかし，分岐履歴の統合を考えるときには，受け取った分岐履歴の作業用履歴（終了判定ベクトル）への寄与が，より直接的に反映される分岐履歴を採用する方が都合がよい．そこで, [14] で命題や定理の証明の過程で導かれる「増分ベクトル」を分岐履歴として，直接，用いることにする．

［定義 2］ 増分ベクトルによるパケットの分岐履歴

(a) 1 段目の検索開始パケットの分岐履歴は空列 (ϵ)．

(b) 前段の PE から渡された検索結果のパケット P の分岐履歴が $\langle b_i, \cdots, b_1 \rangle$ であるとする．P 中の各 OID をキーとして検索を行う．検索結果のパケットを P’ とする．P’ が未確定パケットであるとき，P’ の分岐履歴を $\langle -1, 0, \cdots, 0 \rangle$ とする．P’ が確定パケットであり，その分岐数が q であるとき，$\langle q-1, b_i, \cdots, b_1 \rangle$ とする．

［例 1］ 図 2 に［定義 2］によるパケットの分岐履歴を示す．

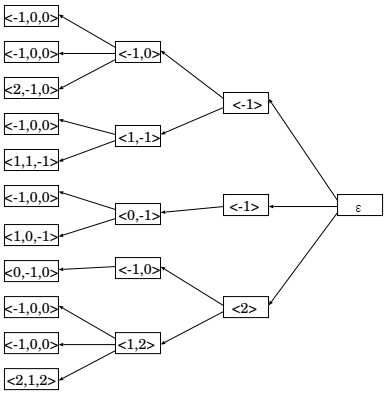

図 2　増分ベクトルによるパケットの分岐履歴
Fig. 2　Branch history of packets using incremental vectors

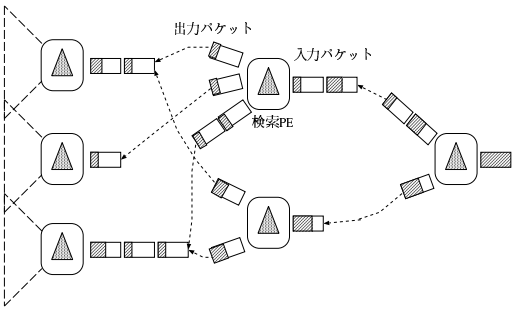

図 3　入出力パケット
Fig. 3　Input/Output packets

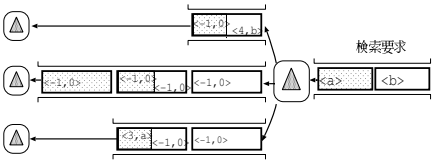

図 4　分岐履歴の混在
Fig. 4　Mixture of branch histories

3.2　検索終了判定

[14] では定義 1 による分岐履歴に対する終了判定アルゴリズムを示した．また，定義 1 の分岐履歴から導かれる定義 2 の増分ベクトルを考えることにより，その正しさを証明した．すなわち，次の定理が成り立つことを示した．

［定理 1］ 検索終了段から受け取るパケットの到着順に関わらず，それらの分岐履歴の各項ごとの和が全て 0 になったとき，かつその時に限り検索結果のパケットを全て受け取ったと判定できる．

［例 2］ 図 2 では 11 個のパケットの到着順に関わらず，全てのパケットを受け取ったときのみ，分岐履歴の各項ごとの総和は $\langle 0,0,0\rangle$ である．

4.　パケット統合

4.1　パケット充填率低下の問題

各 PE は前段の PE の各々から受け取る入力パケットの待ち行列を 1 本，また，検索結果を収容する出力パケットの待ち行列を次段の PE ごとに有しているものとする．各 PE は入力パケット中の OID をキーとして順次検索し，結果の OID を出力パケットに格納した後，分岐履歴を付与して，次段の該当 PE の出力待ち行列に置く．このときの問題は，索引集合 HV_i^j の全てのデータ値集合が HV_{i-1}^k のキー値集合の部分集合であることが成立する PE_i^j と PE_{i-1}^k 間以外では，検索結果が集合値のときパケットの分割が発生し，分割によって出力パケットの OID の充填率が低下しうることである（図 3）．充填率の低下は生成パケットの増大を意味し，次のような問題を引き起こす．
(1) PE 間のパケット通信コストの増大 (2) 入出力パケットの待ち行列の占有メモリの増大 (3) 終了判定 PE の判定負荷の増大

これは結果的に検索速度とメモリコストを劣化させる．

4.2　パケットの統合

4.1 で述べたパケットの充填率低下の問題を解決するために，異なる入力パケットの検索結果の出力パケットを必要ならば分割して図 4 に示すように単一のパケットに併合することにより，充填率を向上させることを考える．

検索結果の分割・併合そのものには問題はないが，併合に伴い，図 4 に見るように 1 つのパケットに分岐履歴が混在することになる．したがって，検索終了判定を正しく行うために，混在する複数の分岐履歴を統合して，単一の分岐履歴として，付け直す必要がある．以下では，

($S1$)　分岐履歴の和による統合
($S2$)　分岐履歴の破棄による統合
($S3$)　$S1$ と $S2$ の併用による統合
($S4$)　単一確定履歴への統合

の 4 つの方法を提案する．

2.4 の終了判定では，検索最終段から受け取った分岐履歴だけが集計 PE において演算されるのに対して，これらの方式は，充填率を高めるためのパケット併合の要求に従って，検索の途中段においても分岐履歴の演算が行われる．

なお，以下の 3 つの理由から，パケット充填率を上げるために，パケット統合アルゴリズムによってパケットを統合しても，［定理 1］における終了判定が正しく動作することが示せる．
(1) n 段目の PE における入力パケット集合の各 OID に対応する全ての検索結果の OID 集合はアルゴリズムによるパケット統合を行っても不変である．
(2) 後述するアルゴリズムによって分岐履歴の付け替えを行っても，定義 1 によるパケットの分岐履歴の付与規則は守られている．
(3) 分岐履歴のみの空のパケットが集計 PE に送信されることがある．これは [14] におけるハーフパスに相当するが，ハーフパスがある場合についても［定理 1］における終了判定が正しく動作することが証明されている．ここで，前段の i 段の PE から渡されたパケット P の分岐履歴が $\langle b_i,\cdots,b_1\rangle$ であるとする．P 中の各 OID o をキーとして検索を行い，その結果集合を $R(o)$ として，すべての検索結果の和集合，$\bigcup_{o\in P} R(o)$ を $\{o_0,o_1,\cdots,o_{q_1}\}$ とする．また，次段の PE の数を M とし，OID $o_j(0\leqq j<q)$ に対して o_j がキーとして格納されている次段の PE $m(0\leqq m<M)$ を $h(o_j)$ とする．また， m への出力パケッ

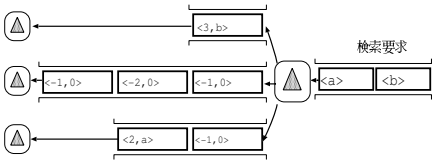

図 5　分岐履歴の和による統合 ($S1$)

Fig. 5　Sum of branch histories

トを P_m とする．以下，便宜上，各入出力パケットをそれらに付与されている分岐履歴で表すことにする．

4.2.1　分岐履歴の和による統合

1 つのパケットに混在する分岐履歴を加算することで，単一の分岐履歴とする方法である．図 4 に対して，本方式による統合を図 5 に示す．出力パケット数が 9 から 6 に減少しているのがわかる．またこのアルゴリズムを以下に示す．

```
merge() {
  int written_flag[M] = {0,...0};
  int k = 0; // 分岐数
  int j, m;
  for (j = 0; j < q; j++) {
    m = h(oj);
    Pm に oj を追加;
    if (written_flag[m] == 0) {
        Pm に i+1 項の未確定分岐履歴を加算;
        k = k+1;
        written_flag[m] = 1;
    }
    if (Pm が満杯) {
      if (j == q-1) {
        Pm に分岐履歴<k,bi,...,b1>を加算;
      }
      Pm を次段の PEm に送出後,Pm を空にする;
      written_flag[m] = 0;
  }}
  return;
}
```

4.2.2　分岐履歴の破棄による統合

図 4 の例に対して，本方式による統合を図 6 に示す．入力パケット a の検索結果が入力パケット b に先行して出力パケットを埋めたとする．後続の入力パケット b の検索結果は，入力パケット a の処理で埋められた満杯でない出力パケット，または新規パケットに書き込まれる．このとき，入力パケット b の処理で作成された新規パケットについてのみ分岐数を計上し，同出力パケットに分岐履歴を付与する．入力パケット a の検索結果と分岐履歴が付与された出力パケットは，後続の検索結果で充填しても，一旦付与された分岐履歴は更新しない．

このように統合する場合，入力パケット b の検索結果が全て先行する入力パケットの結果に詰め込まれ，入力パケット b の分岐履歴が紛失することがある．この場合，空のパケットを作成して紛失される分岐履歴を付与し，直接集計 PE に送出する．本方式のよるアルゴリズムの疑似コードを以下に示す．

```
int ready_flag[M] = {0,...,0};
merge() {
  int written_flag[M] = {0,...,0};
  int k = 0; // 分岐数
  int j, m;
  for (j = 0; j < q; j++) {
    m = h(oj);
    Pm に oj を追加;
    if (ready_flag[m] == 0) {
      if (written_flag[m] == 0) {
        Pm に i+1 項の未確定分岐履歴を加算;
        ready_flag[m] = 1;
        written_flag[m] = 1;
        k = k+1;
    }}
    Pm に i+1 項の未確定分岐履歴を加算;
    if (Pm が満杯) {
      if (j == q-1) {
        if (written_flag[m] == 0)
          分岐履歴が<k,bi,...,b1>である
              空パケットを集計 PE に送信;
        else
          Pm に分岐履歴<k,bi,...,b1>を加算;
      }
      Pm を次段の PEm に送出後,Pm を空にする;
      written_flag[m] = 0;
      ready_flag[m] = 0;
  }}
  return;
}
```

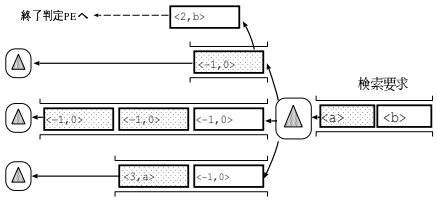

図 6　分岐履歴の破棄による統合 ($S2$)

Fig. 6　Neglect of branch histories

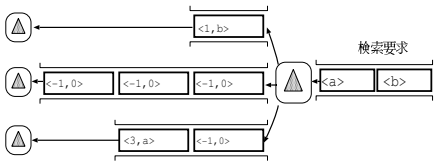

図 7　分岐履歴の破棄と和の併用 ($S3$)

Fig. 7　Combination of the two methods in 4.2.2 and 4.2.1

4.2.3　分岐履歴の破棄と和の併用による統合

4.2.2 および 4.2.1 のそれぞれの方式の欠点を軽減するために，これらの方式を併用する．すなわち，基本的に「分岐履歴の破棄」の方式を用い，4.2.2 において確定履歴が紛失する場合で未送信の出力パケットが存在する時は，既に付与されている分岐履歴との和をとり，分岐履歴を更新する．これにより，分岐履歴のみの空パケットの送信負荷増大，および分岐履歴の和による PE の計算負荷の増大を軽減する．

図 4 の例に対して，本方式による統合を図 7 に示す．

4.2.4　単一確定履歴への統合

以上で述べたパケット統合の方法は，いずれも 1 つの入力パケット P に対する検索が終了した時点で分岐数 m と分岐履歴 α をもとに確定履歴を付与している．この利点は，当該 PE が α を記憶する必要がないことである．

これに対して，入力パケットの待ち行列に現在到着しているか，または今後到着するパケットの最初の n 個に注目する．本

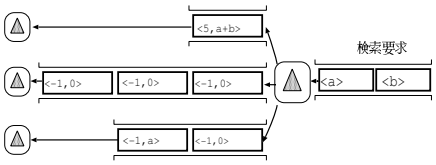

図 8　単一確定分岐履歴への統合 ($S4$)
Fig. 8　Merge into a single branch history

方法は PE 中でこれら n 個の入力パケットを順次検索し，検索結果を出力パケットに詰めて，次段の PE に送出する．このとき，出力パケットの充填率を 100%として，m 個の出力パケットが生成されたとする．最初の $1 \sim (m-1)$ 番目の出力パケットには未確定履歴を付与して未確定パケットとして送り出し，最後の m 番目の出力パケットにのみ確定履歴を付与して確定パケットとして送り出す．この確定パケットの確定履歴を決定するために，PE において各々の入力パケットに付与された分岐履歴を加算しながら，その累積値 α を記憶しておく．このとき，確定履歴は $\langle m-1, \alpha \rangle$ となる．図 8 は図 4 を本方式で併合した場合の例となる．

5. パケットの受送信と統合

5.1 断続的入力への対応

パケット統合を行うためには，満杯でない出力パケットの送信を待機する必要がある．PE が入力パケットを連続的に受信する保証はなく，また 3. の終了判定では分岐履歴と検索の結果以外に検索処理を制御するためのパケットは存在しない．このため，出力パケットの強制送出のタイミングとそのアルゴリズムが新たに必要となる．

この問題に対して，受信操作のタイムアウトを強制出力のタイミングとするアルゴリズムを適用する．待ち行列の操作では全て最大待機時間 T_w を設定し，タイムアウトによって失敗したとき，全ての送信待ちパケットを送出対象として選択し，出力する．これにより前段 PE から入力パケットが到着しない場合も確実に検索処理を継続できる．

5.2 要求の並列処理と連続的入力への対応

次に，複数の検索要求の並列処理に対して検索アルゴリズムを対応させる．まず，検索要求ごとに一意な識別子 (RID: Request ID) を与え，集計 PE では RID 別に集計する方法を適用する．このときパケットの統合は同一の RID が与えられた入力パケットの検索結果同士だけが統合可能である．

RID を用いた方法で問題となるのは，統合可能性を高めるために RID ごとに出力パケットバッファを複数用意して統合を行う場合，連続的な入力パケットの受信に対してバッファを出力するアルゴリズムが求められることである．単純な考えとして，バッファは無限個利用可能で異なる RID を持つパケットを連続的に受信するとき，5.1 のアルゴリズムだけでは受信が停滞するまで，最大で 出現した RID ×次段の分割数 だけバッファが作成され続けられるためにメモリを圧迫し，また送出されないことに伴って検索処理のターンアラウンドタイムの劣化を招く．

この問題に対して，出力するバッファの選択にタイムスタンプと LRU(Least Recently Used) アルゴリズムを適用し定期的な出力を行う．

出力バッファは，タプル $\langle$ 出力バッファ, タイムスタンプ $(t_s)\rangle$ により管理され，利用（書き込み）時にタイムスタンプが更新される．予め，5.1 の T_w とは別に，最大保持時間 T_k が設定される．検索 PE の入力 1 件の処理の流れ（受信〜索引検索〜出力）のいずれかの時点，または 5.1 の強制送出処理の送出対象選択時に，$T_k \leqq$ (現在時刻 $- t_s$) となった出力バッファを検索し，該当する出力バッファは履歴を確定してパケットを完成し送出する．出力バッファを有限個とする場合は，利用可能バッファが 0 で新たにバッファを要求する時，最も古い t_s を持つ出力バッファが選択されてパケットの完成と送出を行い，その操作によって得た利用可能数 1 を要求されたバッファに割り当てる．

充填率の向上は，5.2 で述べた最大保持時間 T_k を大きく取るほど最適の充填率に近付くが，T_k を大きくすればターンアラウンドタイムが劣化するので，これは検索時間とのトレードオフである．T_k の取り方についてはここでは議論しない．

6. 実装と性能評価

提案した方法に従って並列検索終了判定を行うアルゴリズムを組み込んだプロトタイプシステムを実装し，検索速度その他を評価した．

6.1 実 装 環 境

実験は，SMP 型のマルチプロセッサマシン Sun Fire 15K(UltraSPARC-III+ 32 CPU, 32GB Memory) 上で，POSIX Pthread ライブラリを用いたマルチスレッド並列環境で評価した．今後の議論では 1 つの PE が 1 つのスレッドに対応する．索引は B^+ 木をオンメモリで使用している．

RID は発行順に整列可能な値を付与し，発行の古いものの処理優先度が高いとする．検索 PE は受信の際にヒープを使用してパケットを RID で整列し，優先度順に処理を行うアルゴリズムを適用した．またパケット統合型の実装ではタプルを $\langle$ 出力バッファ, $t_s\rangle$ ではなく，$\langle RID, t_s\rangle$ として RID 単位で出力バッファを束ねて管理している．

6.2 評 価 環 境

分岐履歴による終了判定法に従ったパケット充填率を考慮しない方法と，提案したパケット統合方式による充填率を考慮した検索について，性能を評価した．

実験は，2 種類のクラス構造について単一の検索パス式 $P = C_1A_1A_2 \cdots A_n$ に対応するマルチインデックスを用意した．図 9 は A_n のインスタンス o について，各タイプにおける参照関係を示したものである．

・タイプ A: 各段の索引はキー値とデータ値が 1 対 1 で，全て線形変換であるタイプ．$A_n(\equiv C_{n+1})$ および $C_i(1 \leqq i \leqq n)$ のインスタンスはそれぞれ 30000 個を用意し，$n = 8$，各索引は各段で均等に水平分割した．

・タイプ B: 各段の索引は，キー値とデータ値が 1 対 2 で，全て線形変換であるタイプ．従って，検索条件の個数が k のとき検索結果が $k \times 2^n$ 個得られる．$A_n(\equiv C_{n+1})$ のインスタンスは

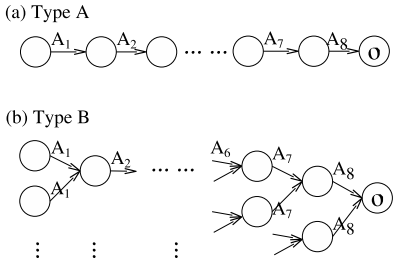

図 9　実験に使用した索引対象のインスタンス構造

Fig. 9　Instance diagram

3000 個を用意し，$n = 8$，各索引は各段で均等に水平分割した．

２つのタイプはそれぞれ入力条件数に対して最終出力結果数が等しい場合と多くなる場合に対応している．いずれも，最終段を除く各 PE は平均して入力パケット充填率以下の密度の出力パケットを次段に送信するため，パケット統合による性能向上が期待できる．

表 1 は各段あたりの水平分割数を示す．水平分割数は PE の検索負荷を均一化することを考慮しており，特にタイプ B では後段ほど多く確保している．

段	1	2	3	4	5	6	7	8
分割数	5	5	5	5	5	5	5	5

(a) Type A

段	1	2	3	4	5	6	7	8
分割数	2	2	2	2	4	4	8	8

(b) Type B

表 1　索 引 分 割

検索要求は単一の要求発行 PE が発行し，検索要求に対応するパケットを初段の検索 PE に送信する．発行は検索終了を待たず，逐次的に投入する．

バッファ容量，バッファ数，待機時間は，検索システムの最適化パラメタであり，パケット容量は 500 個の OID を格納できるサイズとした．パケット統合型の実装では，統合可能な出力パケットの出力バッファを RID 単位で組とし，この組を最大 10 個保持し，10000 個程度の索引検索の所用時間をタプルの最大保持時間とした．

実験では受信キューの同期処理に伴うスレッドのコンテキスト・スイッチ時間が主な通信コストである．パケット１件の処理に要する送受信コストが，満杯のパケットの索引検索時間に対して数分の１程度になるよう選択している．

6.3 性 能 評 価

初めにタイプ A に対する索引検索の実験結果を示す．図 10，図 11 は，それぞれ１検索要求あたりの検索条件オブジェクト数に対する検索時間と，システム全体のパケット通信量（各 PE のパケット受信回数の合計）を表すグラフである．

一見して，計算時間とパケット通信量に強い相関を見いだせる．そして充填率を考慮した４つの提案方法は従来型の考慮しない方法と比べて高速で，パケット統合によってパケット送受信回数を低減したことがわかる．

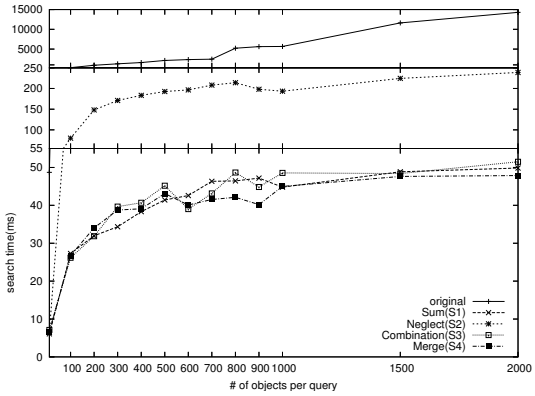

図 10　検索時間:タイプ A

Fig. 10　Retrieval time: Type A

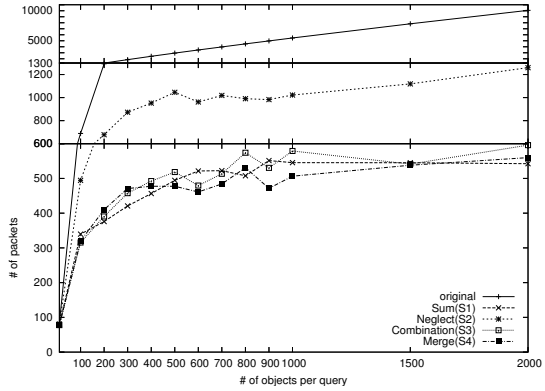

図 11　パケット数:タイプ A

Fig. 11　Received packets: Type A

４つの提案方式の内，分岐履歴の破棄による方法は最も遅く，他の３方式の間にはあまり差が現れていない．パケット統合のアルゴリズムは，(i) アルゴリズムの計算コスト，(ii) 空パケットの送信に違いがある．$S1$ と $S4$ の２方式は検索結果が空の場合を除き (ii) がない点が他の２つと異なり，$S2$ と $S3$ の２方式は (ii) を行う条件に違いがある．(i) は４方式全てで異なる．

$S1$ と $S4$ の結果には検索時間と通信量のいずれも，ほとんど差が見いだせないことから，２つの方式間の (i) は索引検索時間に占める割合は無視できる程度といえる．

$S2$ と $S3$ は受信パケット１件の処理の最後のアルゴリズムだけが異なる．$S3$ の目的である (ii) の減少は，図 11 の結果から機能しているといえる．また $S1$，$S4$ と $S3$ の通信量に差が見いだせないことから，この３つの方式の間では同程度にパケット統合を行ったといえる．そして３つの方式の間の検索時間に差が見いだせないことから，(i) は検索時間に占める割合は無視できる程度といえる．また各方式を通して，パケット統合に伴う (i) は通信量の減少によって得られる検索時間の改善幅で隠蔽されている．

次にタイプ B に対する索引検索の実験結果を示す．図 12，図 13 は，それぞれ１検索要求あたりの検索条件オブジェクト数に対する検索時間と，システム全体のパケット通信量を表すグラフである．

タイプ B ではタイプ A の実験結果と同傾向を示している．パ

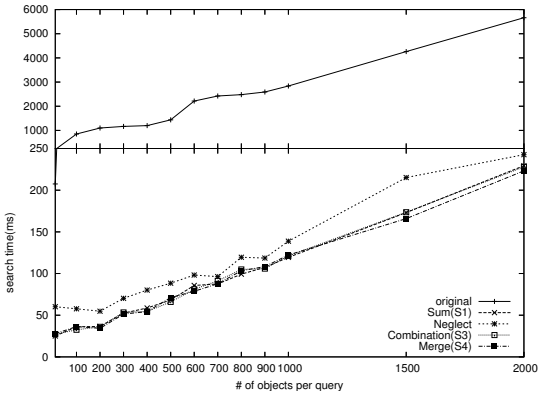

図 12　検索時間:タイプ B

Fig. 12　Retrieval time: Type B

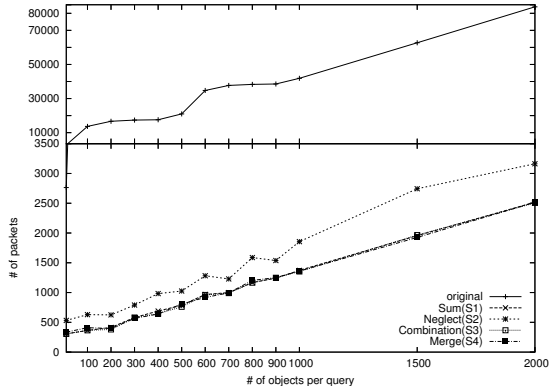

図 13　パケット数:タイプ B

Fig. 13　Received packets: Type B

ケット統合による影響がタイプ A よりも格段に大きくなり高速化されているが，これは統合しない時，水平分割に伴った索引検索結果パケットの分割が検索結果の増大に比例して増加するためである．

一般の複合オブジェクトのクラスは m 対 n の関係が積み重なった複雑な構造を持ち，タイプ A，B ともにその典型例ではない．しかし一般的な構造においても，索引の水平分割を行った場合各 PE の出力は入力条件数に対して出力数の違いと無関係に分割自体は発生しうるので，統合手法は有効に機能すると考えられる．そのときの性能は，実験のように送受信コストの方が大きい場合，検索結果 OID 数/検索条件 OID 数で表す比率が大きいパスの区間はタイプ A と同程度か，統合可能なパケットが存在しなくなることによって統合しない手法の性能に近くなり，高い比率の時，統合手法はタイプ B のような高い性能を示すと考えられる．

実験結果から，索引分割によって生じるパケット充填率低下の問題は，提案したパケット統合のアルゴリズムによって改善されるといえる．そしてパケット統合は空パケットの生成を極力行わない $S2$ 以外の 3 方式いずれかを選べばよいとわかる．

7. むすび

本論文では，複合オブジェクト索引の並列検索の問題において，検索要求パケットの充填率を上げることにより送信パケット数を減少させ，通信コストを軽減することを目的として，新たな並列索引検索終了の判定法を提案した．これらの終了判定法を組み込んだ並列検索システムを実装し，実際に検索実行時間の高速化が達成されることを示した．実験では，バッファリングに伴う保持時間を手動で選択し静的に設定しているが，今後最適な値に向けて動的に変更できるような方策が必要であると考えられる．

ここで提案したパケット統合と並列検索終了判定の方法は，一般に分割統治のアルゴリズム，特に分割委譲された各手続きにおいて発生するデータの量が未定であるような問題（例えば，ガーベジコレクションなど）において，広く適用可能と考えられる．

文　献

[1] E. Bertino and W. Kim: “Indexing techniques for queries on nested objects”, IEEE Trans. on Knowledge and Data Engineering, **1**, 2, pp. 196–214 (1989).

[2] E. Bertino: “A survey of indexing techniques for object-oriented database systems”, Query processing for Advanced Database (Eds. by J. C. Freytag, D. Maier and G. Vossen), Kaufmann, pp. 382–418 (1994).

[3] E. Bertino and P. Foscoli: “Index organizations for object-oriented database systems”, IEEE Trans. Knowl. Data Eng., **7**, 2, pp. 193–209 (1995).

[4] S. Chawathe, M. Chen and P. Yu: “On index selection schemes for nested object hierarchies”, Proc. of 20th VLDB, pp. 331–341 (1994).

[5] 原, 春本, 塚本, 西尾：“Asn.1 データベースシステムにおけるインデックス機構の比較”, 95-dbs-104, 情報処理学会研究会報告 (1995).

[6] C. S., B. E., B. M.H. and C. T.: “On the selection of optimal index configuration in oo databases”, Proc. of the 10th Int’l. Conf. on Data Engineering, pp. 526–537 (1994).

[7] C. Fung, K. Karlapalem and Q. Li: “Structural join index hierarchy: A mechanism for efficient complex object retrieval”, Proc. of the 5th Int’l Conf. on Foundations of Data Organization, pp. 127–136 (1998).

[8] T. A. K., S. S. Y. W. and L. H. X.: “Performance analysis of parallel object-oriented query proceesing algorithms”, Distributed and Parallel Databases, **2**, 1, pp. 59–100 (1994).

[9] T. A.K. and S. S.Y.W.: “Algorithms for asynchronous parallel processing of object-oriented databases”, IEEE Trans. on Knowledge and Data Engineering, **7**, 3, pp. 487–504 (1995).

[10] C. G.S. and G. A.: “Transforming complex methods in vertically partitioned oo databases”, Proc. of Int’l Conf. of Parallel and Distributed Computing and Networks, pp. 56–59 (1997).

[11] M. L., H. M., Y. T. and B. T.: “Parallel navigation in an a-netl based parallel oodbms”, Proc. of Int’l Symposium on High Performance Computing, pp. 305–316 (1997).

[12] N. M.: “Parallel processing for indexed structure of objects”, Proc. of the Int’l. Symposium on Next Generation Database Systems and Their Applications, pp. 56–61 (1993).

[13] 小倉, 都司, ブレト・アルベルト, 宝珍：“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”, 電気情報通信学会論文誌, **J80-D-I**, 6, pp. 486–494 (1997).

[14] 都司, 佛木, 樋口, 宝珍：“分岐履歴演算による複合オブジェクト索引の並列検索終了判定”, 電子情報通信学会論文誌 D-I, **J82-D-I**, 8, pp. 1059–1070 (1999).

[15] H. H., C. M.S. and Y. P.S.: “Parallel execution of hash joins in parallel databases”, IEEE Transactions on Parallel and Distributed Systems, **8**, 8, pp. 872–883 (1997).