DEWS2006 3C-i11

索引分散管理システムにおける段階的再編成

野村　英憲† 　樋口　健† 　都司　達夫†

† 福井大学工学研究科　〒 910–8507 福井県福井市文京 3–9–1
E-mail: †{hidenori,higuchi,tsuji}@pear.fuis.fukui-u.ac.jp

あらまし　本論文では複合オブジェクトに対する索引分散管理システムにおけるオンライン再編成における段階的再編成の提案を行う．索引分散管理システム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散管理し，検索パス式に基づく問い合わせ処理を行う．しかし，検索要求の傾向の変化や大量の索引の更新が起こると，その後の処理の応答時間が悪化することがある．このような事態に対処するには索引の分割を再編成する必要がある．オンライン再編成は同システムにおいて他の処理への影響を抑えつつ索引分割の再編成を行う手法であり，段階的再編成は，オンライン再編成の処理を分割し，他の処理とともに連続的に処理することでオンライン再編成中における他の処理の応答時間の悪化を軽減することを目的とする手法である．
キーワード　オブジェクト指向データベース，オンライン再編成，マルチインデックス，索引分散管理システム，

Incremental Reorganization for Distributed Index System

Hidenori NOMURA†, Ken HIGUCHI†, and Tatsuo TSUJI†

† Graduate School of Engineering., University of Fukui　Bunkyo 3–9–1, Fukui-city, Fukui, 910–8507 Japan
E-mail: †{hidenori,higuchi,tsuji}@pear.fuis.fukui-u.ac.jp

Abstract　This paper concerns an incremental reorganization technique for on-line reorganization for distributed index system 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. However, if the tendency of retrieval queries changes or a large amount of index modification queries occurs, response time would be often degraded. In order to overcome such a situation, index reorganization is needed. On-line reorganization keeps the influence on other processing in this system. In our new system, reorganize operation is divided into small ones and these divided reorganize operations are processed successively with other operations. By scheduling other operations among divided reorganize operations, response time of other operations are improved.
Key words　object-oriented database, on-line reorganization, multi-index, distributed index system

1. はじめに

オブジェクト指向データベースでは，内部に複雑な参照関係を持つ複合オブジェクト集合を扱っており，それらを迅速に検索する手法が要求されている．その 1 つの解決法として, [1] [2] では索引付けを行った複合オブジェクトの索引をメッセージ通信非共有メモリ型並列計算機上に分散配置し，並列処理を行うことにより高速化を目指す研究が行われている．この方式では，データ本体と索引を分離することが可能であり，索引上の処理がデータ本体に影響を及ぼさないという利点がある．

しかし，これらの索引分散管理システムでは，システムを稼動し続けていくことにより，処理効率が悪化することがある．この現象は負荷の不均等による処理効率の悪化や，検索要求の傾向による最適な索引分割の違いが原因となっている．これらの処理効率の悪化を解消するためには索引の再分割及び再配置，つまり再編成を行うことが必要となる．しかし，この再編成処理は，非常に高コストな処理であるため，他の処理への影響を少なくすることが求められる．したがって，他の処理への影響を抑えつつ再編成処理を行うオンライン再編成が必要となってくる．データベースにおけるオンライン再編成に関しては，様々な研究がなされている [5] [6] [7] [8]．また，索引分散管理システムにおいてもオンライン再編成手法が提案されている [3] [4]．

しかしながら，オンライン再編成においても再編成処理中の他の処理の応答時間は著しく悪化する．この解消法として再編

成開始時に索引のスナップショットを生成し再編成処理と他の処理を並列に行い，再編成終了時に再編成後の索引を差し替える方法が考えられる．しかし，索引分散管理システムにおいてスナップショットを作成する場合，分割された各索引のスナップショットを同期させる必要がある．また，スナップショットを作成する間は更新処理を行うことがでず，処理の並列性が妨げられ，オフライン手法と大差なくなってしまう．

そこで本論文では，処理の並列性の高くすることが可能な [3] のオンライン再編成手法に段階的再編成を適用し，再編成処理中の他の処理の応答時間の改善を検証する．ただし，本論文では，索引の状態が非効率的な状態に陥り再編成が必要になったとき，他の処理への影響を少なく抑えながら再編成を行う手法の検証を目的としており，再編成が必要となるタイミングや，効率向上のための最適な索引分割については考慮しない．

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及び索引，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，$1 \leqq j < N$ 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_j の属性値として複数のオブジェクトを参照することができる．また，P の値を

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

とする．ここで，$o_1 o_2 \cdots o_N$ 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 $C_i (1 \leqq i \leqq n)$ 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．また，各 C_i 間の参照関係が，独立したオブジェクト同士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しない．

2.2 マルチインデックス

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 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，

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

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

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

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．

3. 索引分散管理システム

[2] における索引分散管理システムの概要について述べる．索引分散管理システムを構築する環境として，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，対象オブジェクトのクラスとそのインスタンスに ID を割り当て，クラスの ID を CID，インスタンスの ID を IID とし，この 2 つを対にしたものを OID として構成する．また，各プロセッサエレメント（PE）での処理は 1 プロセスの逐次処理とする．このような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．ただし，索引の分割には“ キー値が同じ索引要素は同じ PE で管理する ”という制限以外は加えない．したがって，IP_i が特定の一部の PE 集合に分割して格納される場合や，IP_i が全ての PE によって分割して格納される場合など，自由な索引分割が可能となる．

PE は HOST，検索 PE，DETECTOR の 3 種類に分類される．以下にそれぞれの PE の処理の概要を記す．また，図 1 は索引分散管理システムの処理の流れを表す．なお，索引分散管理システムは索引を管理するシステムであり，その索引のもとであるオブジェクトデータ本体を管理するシステムについては特定しない．

3.1 HOST

検索要求，更新要求などの外部からの処理要求を検索 PE, DETECTOR に送信し，その処理結果を集計し，最終的な処理の終了を判定するための PE である．このとき，外部からの処理要求に対して要求識別子（RID）を付加する．RID には自然数を用い，小さい順に処理要求に付加する．また，処理要求として送信したメッセージ総数の情報を DETECTOR に送信する．

3.2 検 索 PE

実際に索引を格納し，検索・更新処理などを処理するための PE である．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

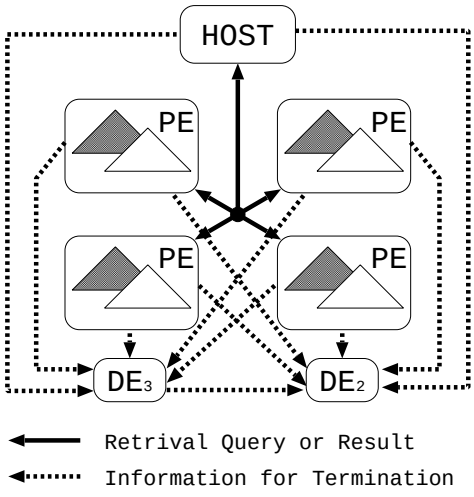

図 1　索引分散管理システム
Fig. 1　Distributed Index System

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 を自分が管理する IP の部分索引で検索する．

（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，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限り（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，(1) 及び（3）において検索要求の送信先 PE を特定する必要がある．つまり，索引要素がどの PE で格納されているかの情報が必要となる．本論文では，ハッシュ関数を用いて分割を決定し，それと同じハッシュ関数を用いて送信先 PE を決定する方法を用いている．また，後述の DETECTOR による終了判定を行うために，他の検索 PE（または HOST）からいくつのメッセージを受信し，それを処理した結果，いくつのメッセージを送信したかの情報を DETECTOR に送信する．

3.3 DETECTOR

DETECTOR（以下，DE）は HOST から発行された要求に対する処理が終了したか否かの判定を行うための PE である．DE は IP_i ごとに 1 つずつ用意され，検索 PE，HOST，DE からもたらされた IP_i に関するメッセージの送信数，受信数の情報を用いて，RID ごとに IP_i の処理が終了したかどうかを判定する．これは，検索パス式の検索処理の方向が一定であり，IP_i への検索要求は IP_{i+1} の検索結果として送信されたメッセージのみであることを利用する．つまり，IP_i に関する検索処理の終了は IP_{i+1} に関する処理がすべて終了し，かつ，IP_i への検索要求のメッセージの総数と IP_{i+1} の検索結果として送信されたメッセージ総数が等しいときに限られる．ただし，$i = N+1$ の場合，“ IP_{i+1} に関する処理 ”は HOST の検索要求の送信処理であり，“ IP_{i+1} の検索結果として送信されたメッセージ ”は HOST からの検索要求である．また，$i = 1$ の場合，”IP_i への検索要求のメッセージ ”は HOST への検索結果である．この性質を用いて検索の終了を判定する．

さらに，この終了判定を用いて索引更新の処理のタイミングを決定する．ここで，IP_i に関する DE を DE_i と表記することとする．

索引の更新に関しては以下の（1）〜（5）の手順で実行される．ただし，更新要求は IP_i に属する索引要素を更新対象とする．

（1） HOST は更新対象の索引要素を管理する検索 PE と DE_{N+1} に対し，更新要求を直接送信する．

（2） 更新要求を受信した検索 PE（更新対象 PE）は DE から更新許可が到着するまで，処理可能な要求に対して処理を行う．

（3） 更新要求の到着した DE（DE_j とする）は $i \neq j$ ならば DE_{j-1} に対し更新要求を転送する．$i = j$ ならば IP_i に関する処理が更新要求の RID より小さな RID を持つ処理要求すべてに対して終了した場合，更新対象 PE に更新許可を発行する・

（4） 更新許可を受信した更新対象 PE は更新処理を行い，その終了を DE に伝える．

（5） 更新終了のメッセージを受信した DE（DE_j とする）は DE_{j-1} に更新終了メッセージを送信する．ただし，$j = 2$ ならば HOST に対して送信を行う．

この更新方法により，更新要求より前に発行された処理要求はすべて更新処理前に実行され，更新要求より後に発行された処理要求はすべて更新処理後に実行されることが保証される．ここで問題となるのは（2）の“ 処理可能な要求 ”の決定である．[2] では以下の条件のどちらかを満たす処理要求に関しては“ 処理可能な要求 ”として処理を行う．

- 更新要求の RID より小さな RID を持つ処理要求．
- 更新対象の索引要素の属する IP_i に属さない索引要素に関する処理要求．

この終了判定及び索引更新手法により検索処理と更新処理が同時実行可能なシステムとなる．つまり，更新処理においては検索 PE における IP_i 単位で排他ロックが行われ，他の索引要素にはまったく無関係に処理が可能であり，検索 PE としてはアイドル状態に陥らずに何らかの処理が可能となる．ただし，索引分散管理システムにおける排他ロックは，実際に索引を排他ロックするのではなく，処理要求を保留として取り扱う．

4. オンライン再編成法

本論文におけるオンライン再編成は，以下に述べる [3] のオンライン再編成法に基づいている．

4.1 索引分割決定法と再編成

再編成とは，検索 PE に分散格納されている索引要素集合を再分割し，それらを検索 PE に再分配することである．したがって，検索 PE 間で索引要素の移動が生じ，それに対する処理が必要となる．

まず，システム全体として以下を前提とする．

- 索引の分割にはハッシュ関数を用いる．

- 各 PE は上記の分割決定に用いたハッシュ関数を保持し，それを用いてメッセージの送信先を決定する．
- 再編成要求は HOST が発行し，RID を付加する．
- 再編成要求はハッシュ関数の変更要求とし，すべての PE のハッシュ関数を変更する．
- 索引要素の移動は各検索 PE 間で直接通信を行って移動させる．

ここで，索引の分割にハッシュ関数を用いる理由としては，再編成前後における検索要求の送信先の変更をハッシュ関数の変更のみで行えることと，索引要素の移動においてハッシュ関数を用いることで移動先の検索 PE を特定可能であり，検索 PE 間で直接的に移動が可能であるためである．

4.2 ロックの取得

再編成によって生じる処理結果の正当性の喪失を防ぐため，以下のようなロックの取得を行う．

ここで，再編成によって移動を要求されている索引要素を“ 移動対象索引要素 ”，移動対象索引要素のキー値の OID が検索条件に含まれる検索要求または移動対象索引要素のキー値と更新対象の索引要素のキー値が等しい更新要求を“ 移動対象関連要求 ”，移動対象関連要求の処理対象をデータ値とする索引要素を“ 移動対象関連索引要素 ”と呼ぶことにする．

- 移動対象関連要求は HOST または検索 PE により送信される．
- HOST においては移動対象関連要求を保留する．
- 検索 PE においては移動対象関連索引要素を使用する可能性のある処理要求を保留する．

4.3 再編成処理の分割

索引要素の移動開始のタイミングは DE の終了判定情報を用いて以下のように決定する．

- 再編成要求以前に発行された要求に対する処理が終了後に索引要素移動を開始.
- 再編成要求以降に発行された要求に対する処理の開始前に索引要素移動を完了.

さらに，移動対象が索引全体に及ぶような大規模な再編成での再編成処理と他の処理との並列性を高めるために，再編成処理を IP_i 単位で分割し連続的に処理することで全体の再編成が完了する方法を採用している．

4.4 索引要素の移動

再編成処理においては他の処理とは別に各 DE がそれぞれ検索 PE 集合を管理し，各検索 PE がただ 1 つの DE により管理される．

以下に IP_i に関する再編成におけるすべての検索 PE に対し保留要求を送信する手順を示す．

（1） HOST は DE_{N+1} に再編成要求を送信.

（2） 再編成要求を受信した DE_j は自分の管理する検索 PE に保留要求を送信し，DE_{j-1} が存在するならば，再編成要求を送信する．

（3） 再編成要求を受信した DE_2 は DE_i に対してすべての検索 PE に保留要求が送信されたことを伝える．

以上の処理により DE_i は自らの終了判定情報を利用することで索引要素移動開始の許可を与えることができる．

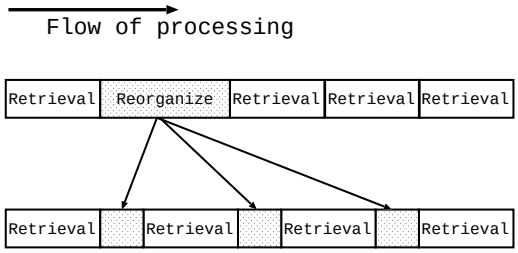

図 2 段階的再編成
Fig. 2 Incremental Reorganization

また，索引要素移動の終了は検索処理における終了判定と同様に DE において送受信情報をやり取りすることで判定を行う．

4.5 ロックの開放とハッシュ関数の変更

索引要素移動の終了が確認されたならば，4.2 において保留状態になっていた処理を行う．そのため，各 DE はそれぞれの管理下の PE 集合に対し，ロック開放の要求を送信する．また，同時にハッシュ関数を変更する．

5. 段階的再編成

4.3 では，再編成処理を IP_i 単位で分割することで，ロックの粒度を小さくし，他の処理との並列性を高めた．しかし，複数の処理要求がある場合には，RID 順に処理を行うため，再編成中に処理される他の要求はさほど多くはならない．そこで，本論文では再編成を完全に分割し，それぞれを他の要求として処理する段階的再編成を提案する．提案する段階的再編成では，再編成範囲を分割することで各再編成に必要なロック範囲が狭くなり，再編成処理中に他の処理を行うことが出来る確率が高まる．また，分割された再編成処理の間に他の要求を挿入することにより（図 2），挿入された要求に対する応答時間の改善が見込まれ，再編成処理と他の処理との並列性が高まる．

5.1 再編成範囲の分割

段階的再編成においては

（1） 再編成の分割数

（2） 分割した再編成間の他の処理数

の 2 点により他の処理の応答時間，再編成自体の処理時間が変化する．(1) の分割数については，少な過ぎる場合は 1 回の再編成あたりの再編成範囲が広くなるため，ロックの範囲が広くなり並列性が低くなる．したがって，再編成処理間に挿入した要求に対する応答時間が悪化することが予想される．一方，分割数が多過ぎる場合はロックの範囲は狭くなるが再編成処理の回数の増加により，ロック取得・開放の回数が増加し，再編成自体の処理時間が長くなり，全体的なスループットが悪化することが予想される．

(2) については，少な過ぎる場合は 1 つの分割された再編成処理が終わる前に次の再編成処理が始まるため，連続して再編成を行うことになり，従来のオンライン再編成と同じになってしまう上，再編成処理同士が衝突し，再編成処理自体の処理時間が長くなり，スループットが悪化することが予想される．しかし，(2) が多過ぎる場合は段階的再編成の途中段階におけ

る処理数が増えてしまう．ここで，その処理が検索要求だった場合，再編成後のハッシュ関数より効率的に不利な途中段階のハッシュ関数を使用しなければならないため，検索処理時間が悪化し，全体的なスループットの悪化を招くことが予想される．以上の点より，再編成処理の分割数と再編成間の他の処理数を適切に設定することが必要である．

5.2 分割手法

再編成処理の分割方法にはいくつかの方法が考えられるが，索引がハッシュ関数と索引のキー値に基づき分割されていることを考えると，分割と同様に索引のキー値に基づいて分割することが効率的である．索引のキー値の OID は 3. で述べたように CID と IID からなるので，これを用いた以下の分割手法が考えられる．

（1） 索引のキー値における一定範囲の CID 単位での分割

（2） 索引のキー値における一定範囲の IID 単位での分割

しかしながら，(1) の手法ではクラスの性質や属するインスタンスの状況により，分割後の各再編成範囲が異なる場合が考えられる．例えば，対象オブジェクトの各クラスのインスタンス数や参照関係の数がクラス毎に大きく異なる場合が挙げられる．この場合，一部の再編成の範囲のみが大きくなり，5.1 の (1) と同様に処理の並列性が低くなり，スループットの悪化が予想される．一方で，(2) の手法は対象オブジェクトの性質に関わらず分割後の各再編成範囲はほぼ均一であり，前述の問題は発生しにくい．よって，本論文における再編成処理の分割手法は，索引のキー値における一定範囲の IID 単位を分割単位として用いる．

6. 評価実験

提案した段階的再編成を従来の索引分散管理システムに実装し，評価を行った．提案手法との比較対象は再編成処理を分割しない従来のオンライン再編成とする．

6.1 実験条件

実験は以下の条件で行った．

6.1.1 対象オブジェクト

以下の複合オブジェクト集合を用意する．

- 索引の対象となる複合オブジェクトは $N = 6$ で各クラスのインスタンス数は 50000.
- 各オブジェクトは 0 から 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ランダムに決定され，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，評価を行う．

6.1.2 索引分散管理システム

実験に用いる索引分散管理システムは以下の条件からなる．

- 検索 PE 数は 15 個であり，DE 数は 5 個，HOST を 1 個用意する．
- 通信には MPI ライブラリ，索引要素の格納に 2 次記憶上の B^+ 木を使用する．
- 索引分割に用いるハッシュ関数は

  $Type1$ ： $(CID - 2) \times A + IID \bmod A$

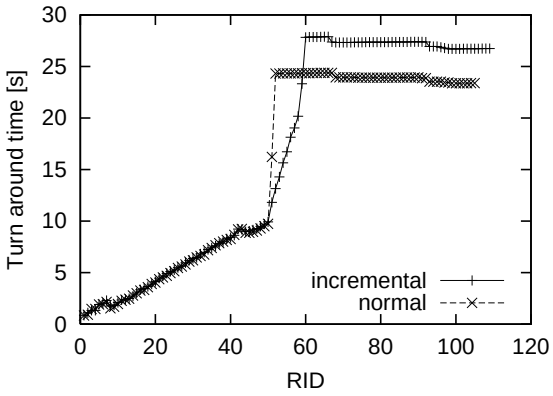

図 3 応答時間 （$K = 1$）

Fig. 3 Turn around time （$K = 1$）

$$A = \frac{\text{検索 } PE \text{ 数}}{N-1}$$

$Type2$ ： $OID \bmod$ [検索 PE 数]

の 2 種類を用意する．

6.1.3 その他条件

実験における条件を以下に挙げる．

（1） 段階的再編成の場合は再編成範囲の分割数を 5 とし，分割した再編成処理の間の他の処理の数を K とし，K を変化させ評価を行う．

（2） 開始時には Type1 のハッシュ関数を用い検索要求を 50 件処理する．その後，ハッシュ関数を Type1 から Type2 へ変更する再編成を行い，再び 50 件の検索要求を処理する．ただし，段階的再編成においては再編成処理の間に他の処理が入るので，この処理の数を考慮して通常のオンライン再編成と段階的再編成とで再編成処理以外の処理の数を等しくする．

（3） 各検索要求はランダムな 5000 個のキー値に対する要求とする．

以上の条件で Sun Fire E4900（コア数：24，メモリ：48GB）上に実装し，実験・評価を行う．ここで，(2) のハッシュ関数の変更は索引要素全体の 4/5 が再編成の対象となり，非常に大規模な再編成となる．また，HOST が各要求を受信した時刻から HOST がその要求の終了を判定した時刻までを応答時間とし，再編成処理中の応答時間を比較するため，再編成処理の前後に検索要求を処理している．

6.2 実験結果

実験結果を以下に示す．図 3，図 4，図 5，図 6，図 7，図 8 は $K = 1, 10, 15, 20, 25, 30$ とした場合の各処理の応答時間を表したものである．図 9，図 10，図 11，図 12，図 13，図 14 は検索処理の応答時間の分布を表したものである．

従来のオンライン再編成においては RID=51 が再編成要求であり，この処理が再編成要求以降の要求の応答時間に大きく影響を与え，RID=52 以降の処理の応答時間が非常に悪化していることがわかる．一方で，段階的再編成においては RID=51, 51+K, 51+2K, 51+3K, 51+4K が分割された再編成要求であり，各再編成要求の後に K 個の検索要求を処理しており，この検索要求に対する応答時間は，それ以前に行った再編成の対

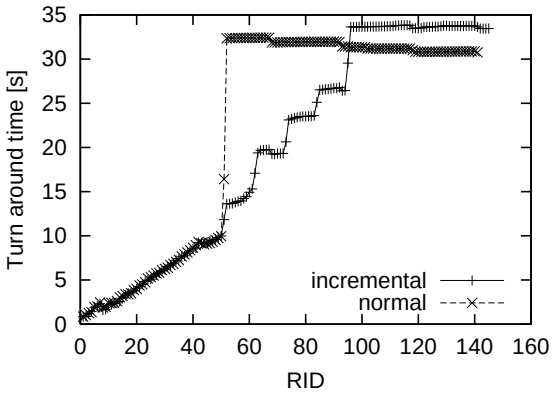

図 4　応答時間 （$K = 10$）
Fig. 4　Turn around time （$K = 10$）

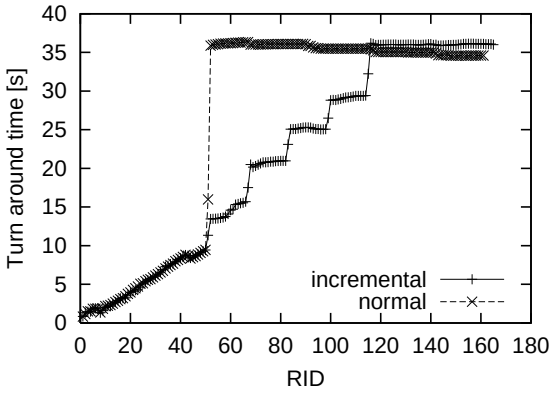

図 5　応答時間 （$K = 15$）
Fig. 5　Turn around time （$K = 15$）

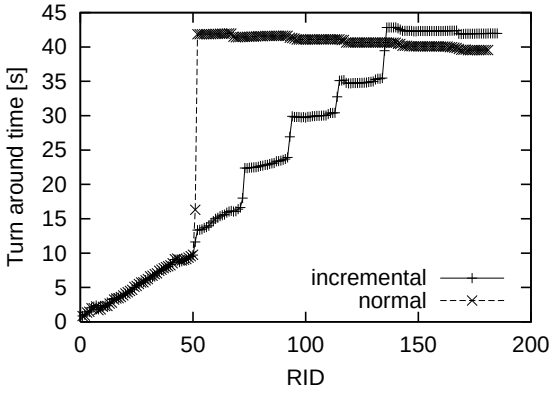

図 6　応答時間 （$K = 20$）
Fig. 6　Turn around time （$K = 20$）

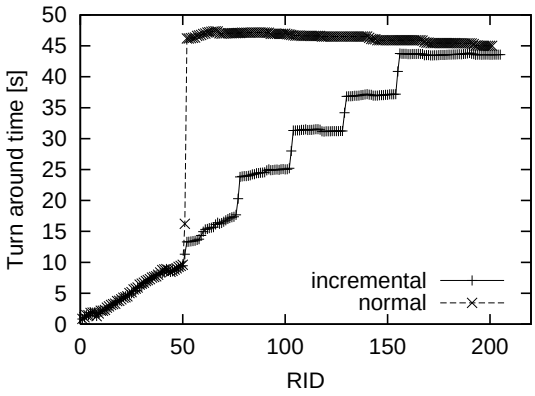

図 7　応答時間 （$K = 25$）
Fig. 7　Turn around time （$K = 25$）

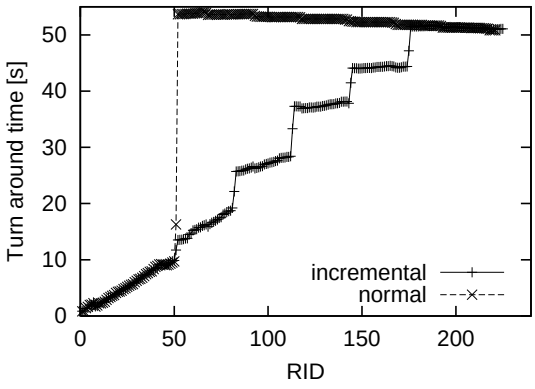

図 8　応答時間 （$K = 30$）
Fig. 8　Turn around time （$K = 30$）

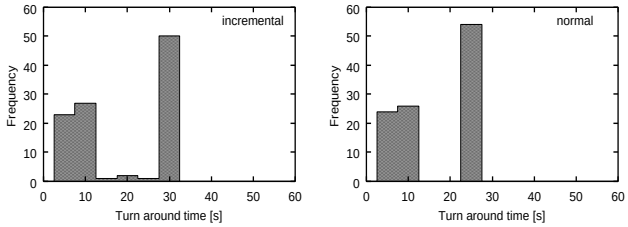

図 9　応答時間の分布 （$K = 1$）
左：段階的再編成　右：従来のオンライン再編成
Fig. 9　Histogram of turn around time （$K = 1$）
left：incremental　right：normal

象範囲によって応答時間の悪化の度合いが変わり，結果として階段状になっているのがわかる．全体の傾向としては分割した再編成処理の間に挿入した検索要求に対する応答時間が改善されていることがわかる．ただし，K が非常に小さい場合（$K = 1$）は処理の順序がほとんど変わらず，段階的再編成による応答時間の改善はほとんど見られていない．

表 1 は総処理時間を表している．K が小さい場合（$K = 1, 10$）は従来の再編成に対して，段階的再編成の総処理時間が悪化していることがわかる．これは，分割した再編成処理同士の間隔が狭いため各再編成によるロックが競合することにより再編成にかかる処理時間が悪化したことによるものと考えられる．K が十分大きいの場合（$K = 15$）には，総処理時間に大きな差は見られない．これは，段階的再編成を行うことで各処理の並列度が上がることにより，再編成回数の増加によるコストを十分に吸収したと考えられる．しかし，$K = 20$ の場合においては従来の再編成に対して，段階的再編成の総処理時間が悪化している．これは，K が大き過ぎるため 5.1 の（2）で述べたスループットの悪化が発生したことが原因と考えられる．ただし，$K = 25, 30$ の場合には，従来のオンライン再編成において再編成処理後の検索要求が多く，検索結果を集計してい

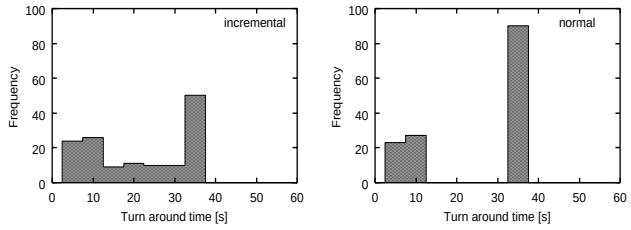

図 10　応答時間の分布 （$K = 10$）
左：段階的再編成　右：従来のオンライン再編成
Fig. 10　Histogram of turn around time （$K = 10$）
left：incremental　right：normal

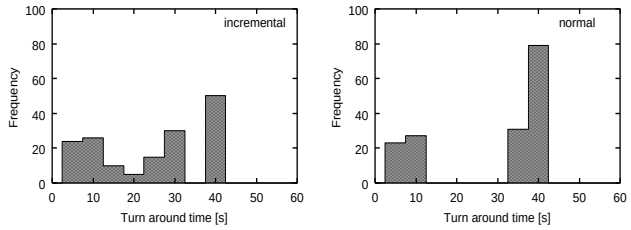

図 11　応答時間の分布 （$K = 15$）
左：段階的再編成　右：従来のオンライン再編成
Fig. 11　Histogram of turn around time （$K = 15$）
left：incremental　right：normal

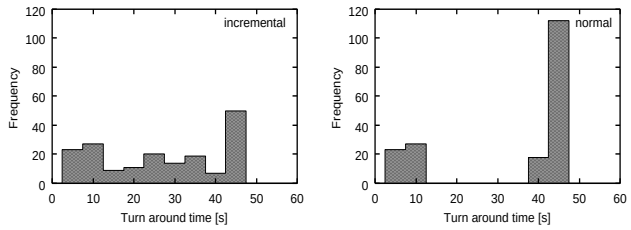

図 12　応答時間の分布 （$K = 20$）
左：段階的再編成　右：従来のオンライン再編成
Fig. 12　Histogram of turn around time （$K = 20$）
left：incremental　right：normal

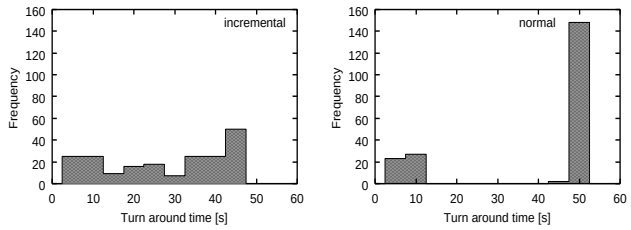

図 13　応答時間の分布 （$K = 25$）
左：段階的再編成　右：従来のオンライン再編成
Fig. 13　Histogram of turn around time （$K = 25$）
left：incremental　right：normal

る HOST に負荷が集中したために総処理時間が悪化したものであり，総処理時間の改善とは言えない状況である．

以上の結果より，段階的再編成を行うことで応答時間の悪化が改善されていることが確認でき，本手法の有効性が実証できた．ただし，K の設定によって従来手法と改善が見られない場合や，総処理時間が悪化する場合も見られ，適切な再編成の間隔の設定の必要である．

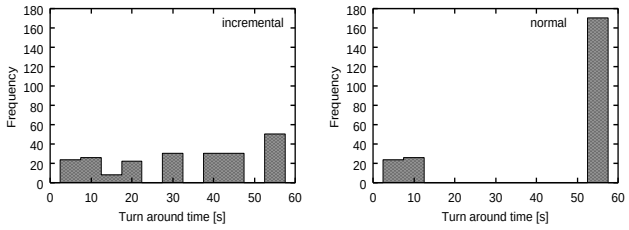

図 14　応答時間の分布 （$K = 30$）
左：段階的再編成　右：従来のオンライン再編成
Fig. 14　Histogram of turn around time （$K = 30$）
left：incremental　right：normal

表 1　総処理時間
Table 1　Total execution time

K	1		10	
type	incremental	normal	incremental	normal
総処理時間 [s]	29.42	25.97	36.74	33.98
K	15		20	
type	incremental	normal	incremental	normal
総処理時間 [s]	39.42	38.06	46.50	43.68
K	25		30	
type	incremental	normal	incremental	normal
総処理時間 [s]	47.84	46.01	55.88	55.97

7. ま と め

複合オブジェクトに対する索引分散管理システムにおけるオンライン再編成において，段階的再編成を提案し，オンライン再編成処理中の他の処理の応答時間の改善を検証した．従来のオンライン再編成に段階的再編成を適用し，分割した再編成処理の間の他の処理の数を適切に指定することで従来のオンライン再編成と同程度の総処理時間で再編成処理中の他の処理の応答時間を改善することができた．しかし，対象オブジェクトや並列計算機の特性によって異なる最適な再編成処理の分割数の決定，及び最適な分割した再編成処理の間の他の処理の数の決定が今後の課題となる．

文　献

[1] 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評価，信学論，Vol.J82-D-I，No.1，pp.58-69（2000）．
[2] 樋口　健，都司達夫，宝珍輝尚，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，情報処理学会論文誌：データベース，Vol.43，No.SIG12（TOD16），pp.64-79（2002）．
[3] 樋口　健，都司達夫，複合オブジェクトに対する索引分散管理システムにおけるオンライン再編成法，情報処理学会論文誌：データベース，Vol.45，No.SIG10（TOD23），pp.1-17（2004）．
[4] 田村弘行，樋口　健，都司達夫，複合オブジェクトに対する索引分散管理システムにおけるオンライン再編成法，信学技報，2005-DBS-135（5），pp.31-38（2005）．
[5] Achyutuni, K., Omiecinski, E. and Navathe, S., Two techniques for on-line index modification in shared nothing parallel database, Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, pp.124-136,（1996）．
[6] Sockut, G.H., Beavin, T.A. and Chang, C., A Method for

On-Line Reorganization of a Database, IBM System Journal, vol 36, no.3, pp.411-436 (1997).

[7] Zou, C. and Salzberg, B., Safely and Efficiently Updating References During On-line Reorganization, Proceedings of 24th International Conference on Very Large Data Bases, pp.512-522 (1998).

[8] Lakhamraju, M.K., Rastogi, R., Seshari, S. and Sudarshan, S., On-line Reorganization in Object databases, Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, pp.58-69 (2000).