DEWS2006 3C-i6

チャンク化拡張可能配列による関係テーブルの実装

黒田　雅之† 　相羽　洋平† 　東　直樹† 　都司　達夫†† 　樋口　健††

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

あらまし　本論文では，チャンク単位で拡張を行う拡張可能配列による関係テーブルの一実装方式を提案し，コストモデルを用いて空間的コストと検索時の時間的コストの評価を行う．我々は HORT(History Offset implementation scheme for Relational Tables) と呼ぶ時間的・空間的コストの優れた関係テーブルの実装方式を提案している．HORT における問題点のひとつとして，経歴・オフセット空間のオーバフローの問題がある．ここでは，経歴・オフセット空間を有効に使用するために，拡張可能配列を配列要素単位ではなく，配列要素の正方部分配列からなるチャンクを単位として拡張を行うことによりこの問題の解決を試みる．評価実験として，コストモデルを用いて従来の関係テーブルの実装方式，配列要素単位の HORT，および提案する実装方式それぞれにおける空間的コストと検索時の時間的コストの比較評価を行う．

キーワード　関係データベース, 拡張可能配列，HORT, チャンク

An Implementation Scheme of Relational Tables by Extendible Chunk-Array

Masayuki KURODA†, Yohei AIBA†, Naoki AZUMA†, Tatsuo TSUJI††, and Ken HIGUCHI††

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

Abstract　In this paper, a new implementation scheme for relational tables is proposed, and evaluated. We are proposing an implementation scheme for relational tables named HORT(History Offset implementation scheme for Relational Tables) which exhibits good performance in space and time costs compared with conventional implementation. However, the problems of HORT include that its history-offset space will be overflows when a large scale relational table is stored. While an extendible array employed in the usual HORT is extended, a subarray of elements is dynamically allocated and attached to the existing extendible array, in the new scheme proposed here, in order to utilize the history-offset space efficiently, a subarray of chunks is allocated. Here, a chunk means a hyper-cube shaped set of array elements. Our new scheme exhibits good performance in space and time costs compared with the conventional implementation and the usual HORT implementation.

Key words　Relational Database, Extendible array, HORT, Chunk

1. ま え が き

企業や組織が保有する大量のデータを分析し，経営に寄与する意思決定や戦略立案の支援に利用することが盛んに行われている．そのため，大量のデータをユーザの問い合わせに応じてオンラインで高速に検索できるデータベースは必要不可欠である．我々は HORT(History Offset implementation scheme for Relational Tables) [1] [2] と呼ぶ新たな関係テーブルの実装方式を提案した．この方式では，関係テーブルのレコードが各カラムのカラム値の集合であることに注目し，MOLAP システムにおけるように多次元配列の要素位置の情報として関係テーブルを実装し，管理する [3] [4]．HORT では，新たなカラム値を持つレコードの追加による関係テーブルの拡張に対して，低コストで対応するために，従来の固定サイズの配列ではなく，拡張可能配列 [5]- [9] の概念をベースとして用いる．従来の拡張可能配列の実装に対して，HORT では経歴・オフセット法 [1] [2] と

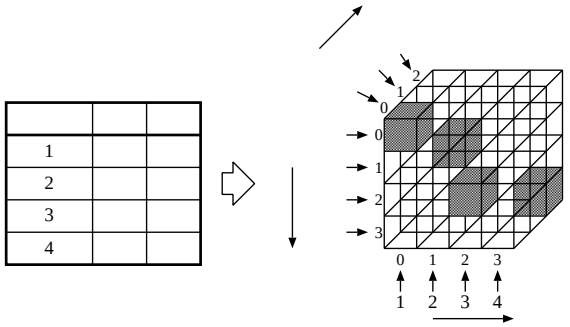

図 1　多次元配列を用いた関係テーブルの実装例
Fig. 1　Implementation scheme of relational table by multidimensional array

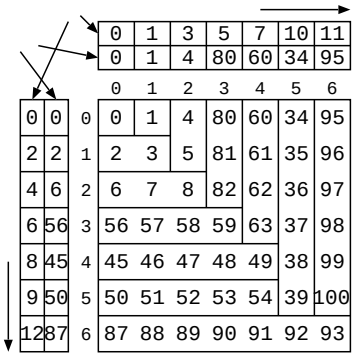

図 2　拡張可能配列の構造
Fig. 2　Physical structure of Extendible Array

呼ぶ手法を用いて配列要素の位置情報を圧縮し，拡張可能配列内の有効要素のみについて B^+-tree に格納することで疎配列問題を解消している．しかし，この方式には関係テーブルのサイズが増大するにつれ，経歴・オフセット空間が飽和し，新たなカラム値を持つレコードの追加が不可能になるという欠点がある．本研究では，n 次元拡張可能配列をチャンクと呼ばれる各次元等サイズの n 次元部分配列を単位として拡張することを提案し，この経歴・オフセット空間の狭小の問題の解決を目指す．

また，コストモデルを用いて，配列要素単位で拡張を行う HORT とチャンク単位で拡張を行う本方式について空間的コストならびに単一値指定検索における時間的コストを求め，レコードを入力順に記憶領域に配置する従来の関係テーブルの実装方式との比較評価を行う．

2. HORT

以下では，我々が提案している HORT(History Offset implementation scheme for Relational Tables) と呼ばれる拡張可能配列の概念を用いた関係テーブルの実装方式について説明する．

2.1 多次元配列を用いた関係テーブルの実装方式

従来の関係テーブルの実装方式においては，レコードは挿入順に逐次 2 次記憶上に配置されるため，次のような問題点がある．まず，レコードはカラム値の集合として 2 次記憶上に配置されるため，年齢や性別といったカラムにおいては同じカラム値が重複して数多く 2 次記憶上に格納される．また，あるカラム値を持つレコードの検索を行うには，全てのレコードを主記憶上に読み込み，カラム値をチェックする必要がある．そこで，これらの問題の対策として，多次元配列を用いて関係テーブルを実装することが考えられる．

図 1 に示すように，関係テーブルの各カラムを多次元配列の各次元に割り当て，カラムにおけるカラム値を対応次元の配列添字と対応付けることにより，関係テーブルの各レコードを配列の 1 要素として扱うことができる．この方式では，同一カラム内において，同じカラム値を持つレコードが複数あろうとも，カラム値そのものはただ 1 つ保持するだけでよいため，カラム値を保持するコストを削減することができる．また，レコードの検索や挿入，削除におけるレコードへのアクセスは，その多次元配列のアドレス関数を用いることにより，高速に行うことができる．しかし，このような多次元固定配列を用いて実装された関係テーブルに新たなカラム値を含むレコードを挿入するには，そのカラムが割り当てられている次元のサイズを 1 つ増加させた多次元配列の領域を確保し，全配列要素を再配置するといった高コストな処理が必要となる．また，多次元配列が大きなものとなるほどこのコストは増大する．そこで，任意次元方向に低コストで拡張を行うことができる拡張可能配列を用いることとする．

2.2 拡張可能配列を用いた関係テーブルの実装方式

拡張可能配列とは，配列の拡張が必要となった時に拡張差分の領域のみを確保し，現在確保している領域はそのまま使用することを可能としたデータ構造である．n 次元拡張可能配列 A は，図 2 に示すように経歴値カウンタと各次元に経歴値テーブル，アドレステーブル，n が 3 以上であれば係数テーブルの 2 または 3 種類の補助テーブルを持つ．配列拡張が行われるたびに現在の経歴値カウンタが 1 つインクリメントされ，その値が経歴値テーブルに順次記録される．ある次元方向への配列拡張は，その次元を除く $n-1$ 次元の配列断面に相当するサイズの連続する記憶領域を動的に確保し A に追加することによって行われる．この $n-1$ 次元の連続記憶領域は通常の固定配列であり，A の部分配列という．例えば，現在のサイズが $[s_1, s_2, s_3, s_4]$ の拡張可能配列において，次元 2 方向に 1 つ配列拡張を行う場合，サイズ $[s_1, s_3, s_4]$ の 3 次元部分配列 S が動的に確保され，この部分配列の先頭アドレスをアドレステーブルの該当スロットに記録する．A が 3 次元以上の場合には，部分配列内の要素のアドレスを計算するための 1 次関数の $n-2$ 個の係数からなる係数ベクトルを部分配列毎に係数テーブルに記録する．例えば，上記の部分配列 S 内の要素 $\langle i_1, i_3, i_4 \rangle$ のアドレスはよく知られているように 1 次関数 $s_1 s_3 i_4 + s_1 i_3 + i_1$ で求められる．この $(s_1 s_3, s_1)$ を S の係数ベクトルと呼ぶ．

図 2 において，次元 1 および次元 2 の経歴値テーブルを H_1，H_2，アドレステーブルを A_1，A_2 とすると，例えば配列要素 $\langle 3, 4 \rangle$ のアドレスの計算は次にように行われる．$H_1[3] < H_2[4]$ であるので，要素 $\langle 3, 4 \rangle$ は経歴値 $H_2[4] = 7$ の部分配列 S に含まれる．また，その先頭アドレスは $A_2[4] = 60$，要素 $\langle 3, 4 \rangle$ は

S 内では要素 $\langle 3 \rangle$ であるため，求めるアドレスは 63 となる．

また，例えば次元 1 の添字が $\langle 3 \rangle$ である要素は，経歴値 $H_1[3] = 8$ の部分配列内の全ての要素と，次元 2 に属す経歴値が $H_1[3]$ 以上の部分配列内の次元 1 の添字が $\langle 3 \rangle$ の要素である．

この拡張可能配列を用いて関係テーブルを実装することにより，新たなカラム値を含むレコードの追加による配列の拡張を低コストで処理することができる．

2.3 経歴・オフセット法

多次元固定配列ならびに多次元拡張可能配列を用いた関係テーブルの実装方式では，レコードの存在の有無を表現するために，配列領域全体を確保する必要がある．この領域は，実装する関係テーブルのカラム数や，カラム値の種類が多くなるほど巨大なものとなる．さらに，一般に関係テーブルを多次元配列を用いて実装すると，疎配列となるため記憶領域を浪費する．そこで，関係テーブルに存在しているレコードについてのみ配列上での位置情報を保持する．一般に，多次元配列内の要素の位置を示すには次元数だけの配列添字を必要とするが，HORT では拡張可能配列の次元数に依らず，要素が含まれる部分配列の経歴値と部分配列内オフセットの 2 つの値のみを用いて要素の位置を示す経歴・オフセット法を用いる．例えば，図 2 の配列要素 $\langle 3, 4 \rangle$ の位置を経歴・オフセット法を用いて表現すると，要素 $\langle 3, 4 \rangle$ が含まれる部分配列 S の経歴値は 7，要素 $\langle 3, 4 \rangle$ の S 内での位置は $\langle 3 \rangle$ であるため，経歴値が 7，オフセットが 3 となる．また，経歴値とオフセットの組から配列要素の組を求めるには，経歴値から要素が含まれる部分配列 S を特定し，オフセットを S の係数ベクトルで順次除算する．さらに，HORT では関係テーブルのレコードを表す配列の有効要素についてのみ，その位置情報を表すこの 2 つの値の組を RDT(Real Data Tree) と呼ぶ B$^+$-tree にキーとして挿入する．これにより，配列実体を確保する必要がなくなり，レコードの存在情報を記録する領域を抑えることができる．以後，実体を持たないこの拡張可能配列のことを論理拡張可能配列と呼ぶ．また，RDT 内では，キーの上位バイトを経歴値，下位バイトをオフセットとすることにより，経歴値，次いでオフセットの順でソートされる．したがって，同じ経歴値を持つキーは RDT のシーケンス・セット上に連続して配置される．

2.4 カラム値から配列添字への変換

関係テーブルを多次元配列で実装する場合，カラム値から配列添字への変換ならびに配列添字からカラム値への逆変換が必要となる．カラム値から配列添字への変換を高速に行うため，関係テーブルの各カラムに CVT(key-subscript ConVersion Tree) と呼ぶ B$^+$-tree を配置し，カラム値をキーとして，対応する配列添字をデータとして格納することにより高速に変換を行う．また，配列添字からカラム値への逆変換は，各次元の補助テーブルとしてカラム値テーブルを設け，対応するカラム値を記録することにより行う．以後，各次元の補助テーブル群のことを，HORT テーブルと呼ぶこととする．図 3 に HORT の構造の例を示す．

2.5 HORT におけるレコードの挿入と削除

HORT におけるレコードの挿入は，論理拡張可能配列の 1 要素の存在情報を RDT に挿入する操作である．まず，挿入対象のレコードの各カラムが対応次元の配列添字に対応付けられているかどうかを CVT 内を探索することにより調べる．もし，カラム値が配列添字と対応付けられていなければ，論理拡張可能配列を対応次元方向に 1 つ拡張し，そのカラム値と拡張部分の添字を CVT に格納する．次に，挿入対象のレコードを各次元の添字の組に変換，さらに，経歴・オフセット法により経歴値とオフセットの組に変換し，RDT に格納する．また，各カラム値が含まれるレコード数をカウントするために，HORT テーブルにレコードカウンタを追加し，この値を逐次書き換える．

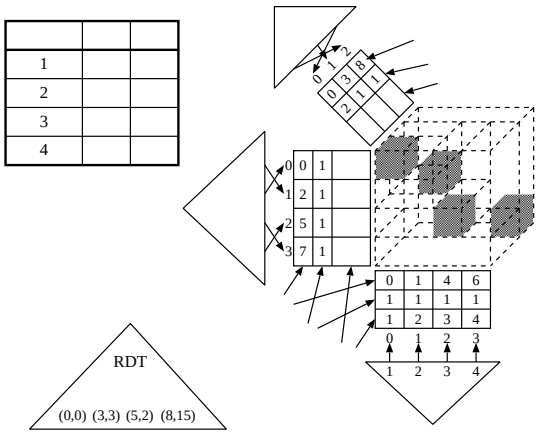

図 3　HORT の構造
Fig. 3　HORT structure

HORT におけるレコードの削除は，論理拡張可能配列の 1 要素の存在情報を RDT から削除する操作である．まず，削除対象のレコードの各カラム値を CVT を用いて配列添字に変換し，その組を経歴値とオフセットの組に変換，RDT から削除する．次いで，削除したレコードの各カラム値のレコードカウンタの値を 1 つデクリメントし，0 になれば，CVT からこのカラム値を削除する．また，この HORT テーブルにできた空きスロットは，以後の挿入で再利用する．

3. HORT におけるレコードの検索

HORT におけるレコードの検索は RDT に格納されている経歴値とオフセットの組に対して行われる．検索条件として全てのカラムの値が指定された場合は，指定されたカラム値の集合から経歴値とオフセットの組を求め，RDT 内にその組が存在するかどうかを調べることにより検索を行う．

以下では，より一般的な HORT における単一値指定検索の方法について述べる．

3.1 B$^+$-tree 内の探索方法

レコードの検索時に用いる 2 種類の B$^+$-tree，RDT と CVT 内の探索方法を次に挙げる．

EQUAL　B$^+$-tree をルートノードから辿ることにより，指定されたキーが B$^+$-tree 内に存在するかを調べる．CVT では，指定されたキー，つまりカラム値と対応付けられている配列添字を返す．また，そのキーにカレントポインタを設定する．

GTEQ　B$^+$-tree をルートノードから辿ることにより，指定さ

れたキーか，指定されたキー以上のキーのうち最小のキーを返す．CVT では，そのキーと対応付けられている配列添字をキーと共に返す．また，そのキーにカレントポインタを設定する．
SQGTEQ　カレントポインタが設定されているキーから，シーケンス・セット部を辿ることにより，指定されたキーか，指定されたキー以上のキーのうち最小のキーを返す．また，そのキーにカレントポインタを設定し直す．
NEXT　カレントポインタが設定されているキーの次に格納されているキーをシーケンス・セット部を辿ることにより求める．CVT では，そのキーと対応付けられている配列添字をキーと共に返す．また，そのキーにカレントポインタを設定し直す．

3.2 HORT における単一値指定検索

HORT における単一値指定検索では，まず，検索条件としてカラム値を指定されたカラム l の CVT を用いて，カラム値を対応する配列添字に変換する．この時，CVT 内に指定されたカラム値が存在しなければ，検索条件に合致したレコードは関係テーブル内に存在しない．次に，そのカラム値と対応付けられた部分配列と，その部分配列よりも大きな経歴値を持つ部分配列のうち，次元 l 以外の次元に属する部分配列内を検索する．これは，拡張可能配列において，ある添字を持つ要素はその添字に対応する部分配列内および，その部分配列よりも大きな経歴値を持つ他の次元に属する部分配列内のみに存在するという特性を利用している．以後，指定されたカラム値が対応付けられている部分配列を基部分配列と呼ぶ．

部分配列内の検索では，検索条件として指定されたカラム値が対応付けられている添字を持つ要素のみについて，その経歴値とオフセットの組が RDT に格納されているかどうかを調べればよいが，HORT により大規模な関係テーブルを実装した場合，RDT のサイズが非常に大きくなるため，RDT を主記憶上ではなく，2 次記憶上に配置すると仮定する．そのため，キーを指定して RDT をルートノードから辿る場合には RDT の高さ分だけのノード内探索と，新たにアクセスするノードの 2 次記憶上からの読み出しを行う必要がある．さらに，論理拡張可能配列の充填率は非常に低いため，指定したキーが RDT 内に存在する可能性は低い．一方，RDT 内では，キーは上位バイトより経歴値，オフセットの順で連接しているため，部分配列内のレコードは，RDT のシーケンス・セット部に連続して格納されている．そこで，キーを指定して RDT をルートノードから辿った場合と，シーケンス・セット部を逐次辿り，格納されている経歴値とオフセットの組を配列添字に逆変換することによって指定されたカラム値を持つレコードを探索する場合のコスト比較を行い，最適な RDT 内の探索方法を選択することとする．そのため，RDT の検索方法として，以下の 3 種類の方法を考える．これらは順に検索対象範囲を狭めるものであるが，検索対象範囲を狭めるほど計算コストは高くなる．

なお，検索条件として指定されたカラム値のレコードカウンタの値だけの検索対象レコードが見つかった時点で検索を終了する．

3.2.1 検索方法 1

まず，基部分配列の経歴値とオフセット 0 をキーとして RDT を GTEQ で探索し，基部分配列内の先頭レコードを取り出す．次に，NEXT で RDT のシーケンス・セット部を終端まで辿ることにより，基部分配列の経歴値以上の経歴値を持つ経歴値とオフセットの組を全て取り出す．これらの経歴値とオフセットの組から 2.3 で述べた経歴・オフセット法の逆変換を用いて，次元 l の配列添字を求めることにより，検索対象のレコードであるかどうかを判定する．この方法では検索対象のレコードが存在し得ない次元 l に属する基部分配列以外の部分配列内のレコードも RDT から取り出すこととなってしまう．

3.2.2 検索方法 2

まず，基部分配列の経歴値とオフセット 0 をキーとして RDT を GTEQ で探索し，基部分配列内の先頭レコードを取り出す．次に，NEXT で RDT のシーケンス・セット部を辿ることにより，基部分配列内のレコードを全て取り出す．この基部分配列内のレコードは全て検索対象のレコードである．続いて，次元 l 以外に属する部分配列のうち，基部分配列よりも大きな経歴値を持つ部分配列それぞれについて，部分配列内の全てのレコードを取り出す．ただし，次元 l 以外に属する部分配列内には非検索対象レコードも含まれているため，取り出した経歴値とオフセットの組から次元 l の配列添字を求め，検索対象のレコードであるかどうかを判定する必要がある．なお，1 つの部分配列内の検索を終了し，次の部分配列内の検索に移る際には，GTEQ で RDT をルートノードから辿った場合と，SQGTEQ でシーケンス・セット部を辿った場合のコスト比較を行い，コストが低い方法を用いて RDT を辿ることとする．

3.2.3 検索方法 3

まず，検索方法 2 と同様に，RDT から基部分配列内のレコードを全て取り出す．次に，次元 l 以外の次元に属する部分配列のうち，基部分配列より大きな経歴値を持つ部分配列それぞれについて検索を行うが，この時，検索対象の要素のオフセットが連続している区間を求め，それぞれについて検索を行う．

例として，n 次元論理拡張可能配列の次元 l のカラム値が指定されたとする．次元 m に属する検索対象要素を含むある部分配列 S のサイズが $[s_1, s_2, \cdots, s_n]$（但し $s_m = 1$）である時，検索対象要素のオフセットが連続している区間の長さは $\prod_{j=1}^{l-1} s_j$ で求められる．また，この区間は部分配列内に $\prod_{j=l+1}^{n} s_j$ 個存在する．

また，検索方法 2 と同様に，1 つの検索対象要素のオフセットの連続区間内の検索を終了し，次の検索対象要素のオフセットの連続区間の検索に移る際には，GTEQ を用いて RDT をルートノードから辿った場合と，SQGTEQ を用いて RDT のシーケンス・セット部を辿った場合のコスト比較を行い，コストが低い方法を用いて RDT を辿ることとする．検索方法 3 を用いることにより，RDT から取り出したレコードが検索対象のレコードであるかどうかを判定する必要がなくなる一方，部分配列 1 つにつき最大 $\prod_{j=l+1}^{n} s_j$ 回のコスト比較を行う必要がある．

4. 提案する実現モデル

以下では，HORT の問題点と，その問題点を解決するために

表 1　HORT における次元サイズとアドレス空間使用率
Table 1　Dimension size and address space utilization in HORT

次元数	1 辺のサイズ	空間使用率
2	2147483649	5.82×10^{-9}[%]
3	1431655766	3.07[%]
4	2642246	6.15×10^{-2}[%]
5	65536	1.53×10^{-3}[%]
6	7131	1.66×10^{-4}[%]
7	1625	3.78×10^{-5}[%]
8	565	1.32×10^{-5}[%]
9	256	5.96×10^{-6}[%]
10	137	3.00×10^{-6}[%]

表 2　C-HORT における 1 辺のサイズとアドレス空間使用率
Table 2　Dimension size and address space utilization in C-HORT

次元数	1 辺のサイズ	空間使用率
2	281474976710565	100.00[%]
3	4294529957	99.97[%]
4	16777216	100.00[%]
5	480663	99.69[%]
6	65284	97.69[%]
7	13405	98.06[%]
8	4096	100.00[%]
9	1612	92.38[%]
10	768	89.67[%]

提案するチャンク単位で拡張を行う HORT について説明する．

4.1　HORT の問題点

HORT では，関係テーブルのレコードを示す論理拡張可能配列上での有効要素の位置を，その要素が属する部分配列の経歴値と，その部分配列内オフセットの 2 つの値の組で表現している．そのため，HORT で実装する関係テーブルのカラムやそのカラム値の種類が多くなると，経歴値またはオフセットの値が格納する変数の取り得る値の上限を超えてしまう．しかし，オーバーフローを起こすのは最も経歴値が大きな部分配列についてのみであり，例えば，経歴値が 0 や 1 の部分配列のサイズは 1 であるなど，経歴値の小さな部分配列においては，オフセットを格納するための変数の取り得る値の領域ほとんどを使用していない．

表 1 に，経歴値に 32bit，オフセットに 64bit を割り当て，n 次元論理拡張可能配列をなるべく各次元のサイズを均等に最大まで拡張した時の 1 辺のサイズと，本来，経歴値とオフセットの組で表現できるアドレス空間，この例では 2^{96} の使用率を示す．この表を見ると，最もアドレス空間の使用率が高くなる 3 次元の時でも，全体の 3.07[%] しか使用できていないことがわかる．

この問題の解決策の 1 つとして，論理拡張可能配列を要素単位ではなく，ある一定要素単位の塊であるチャンク単位で拡張を行い，各チャンクに一意にふられたチャンク番号と，チャンク内オフセットの 2 つの値を用いて，論理拡張可能配列内でのレコードの位置情報を表現する．チャンクのサイズを，チャンク内オフセットに割り当てられた変数が取り得る最大値に近づけることで，経歴・オフセット空間の使用率を上げることができる．

要素単位で拡張を行った時と同様に，チャンク番号に 32bit，チャンク内オフセットに 64bit を割り当て，n 次元論理拡張可能配列をなるべく各次元のサイズを均等に最大まで拡張した時の 1 辺のサイズと，本来，チャンク番号とチャンク内オフセットの組が表現できる空間の使用率を表 2 に示す．表 2 に示すように，チャンク単位で論理拡張可能配列を拡張し，チャンク番号とチャンク内オフセットの 2 つの値を用いてレコードを表現することにより，経歴・オフセット空間の使用率が低いという問題を解決することができる．経歴・オフセット空間の使用率を上げることにより，従来の要素単位で拡張を行った時よりも 1 カラムあたりのカラム値の種類を増やすことができる．従って，経歴・オフセット空間のオーバーフローを遅らせることができる．図 4 にチャンク単位で拡張を行う HORT の構造の例を示し，以下でその概要と構造について説明する．以後，チャンク単位で拡張を行う HORT のことを C-HORT と呼び，チャンク単位で拡張される論理拡張可能配列の部分配列のことをチャンク部分配列と呼ぶこととする．

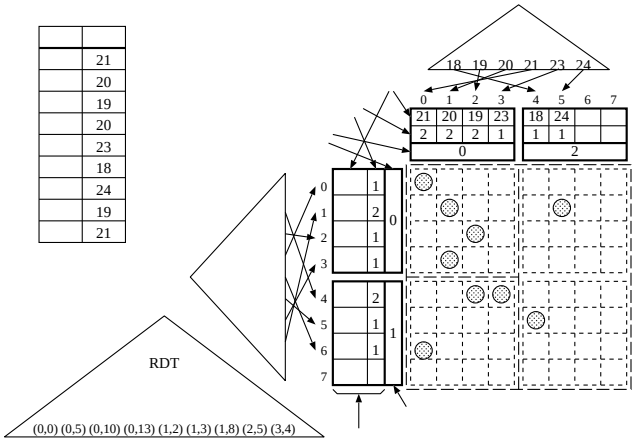

図 4　チャンク単位で拡張を行う HORT の構造
Fig. 4　C-HORT structure extended by chunk

4.2　C-HORT の構造

C-HORT における各次元の HORT テーブルは，チャンク部分配列の情報を記録するチャンク情報テーブルと，カラム値の情報を記録するためのカラム値情報テーブルの 2 つに分けられる．チャンク情報テーブルはチャンク部分配列ごとに作成され，経歴値やチャンク部分配列内の先頭チャンク番号とチャンク数，係数ベクトルが格納される．また，カラム値情報テーブルはカラム値ごとに作成され，カラム値とレコードカウンタが格納される．なお，全てのチャンクのサイズを同一とすることにより，チャンク内オフセットを求めるための係数ベクトルは C-HORT に唯一保持するだけでよい．また，HORT ではカラム値毎に保持していた部分配列の経歴値と係数ベクトルを，C-HORT ではチャンク部分配列毎に保持するだけでよくなるため，空間的コストを抑えることができる．

4.3 配列添字の組からチャンク番号とチャンク内オフセットへの変換

チャンク単位で拡張される n 次元論理拡張可能配列のある要素の添字が $\langle i_1, i_2, \cdots, i_n \rangle$，チャンクの各辺のサイズが $[c_1, c_2, \cdots, c_n]$ であったとすると，各次元のチャンク情報テーブルの添字は $\langle \lfloor i_1/c_1 \rfloor, \lfloor i_2/c_2 \rfloor, \cdots, \lfloor i_n/c_n \rfloor \rangle$ で求められる．最も大きな経歴値を持つチャンク部分配列内に目的の要素が含まれているので，この添字を基に各次元のチャンク情報テーブルにアクセスし， 経歴値を比較することで，目的の要素が含まれるチャンク部分配列を特定することができる．次に，そのチャンク部分配列の係数ベクトルと，各次元のチャンク情報テーブルの添字とチャンク部分配列内の先頭チャンク番号を用いて目的の要素が含まれているチャンクの番号を計算することができる．また，このチャンクにおける目的の要素の添字は $\langle i_1 \% c_1, i_2 \% c_2, \cdots, i_n \% c_n \rangle$ で求められるため，チャンクの係数ベクトルを用いてチャンク内オフセットを求めることができる．

また，チャンク番号とチャンク内オフセットの組をカラム値の組に逆変換する際には，まず，そのチャンクが含まれているチャンク部分配列の係数ベクトルを用いてチャンクのチャンク部分配列内での添字，すなわち各次元のチャンク情報テーブルの添字を求める．次に，チャンクの係数ベクトルを用いてチャンク内オフセットからチャンク内での要素の添字を求め，これらを基に各次元のカラム値情報テーブルにアクセスすることによりカラム値を求める．このように，配列添字の組からチャンク番号とオフセットの組への変換ならびに逆変換時の計算コストは，HORT において配列添字の組から経歴値とオフセットの組への変換ならびに逆変換時の計算コストよりも大きくなる．

4.4 C-HORT におけるレコードの挿入と削除

レコードの挿入を行う際には，まず，挿入対象のレコードの各カラム値が論理拡張可能配列の対応次元の添字と対応付けられているかどうかを，CVT 内を探索することによって調べる．もし，カラム値が配列添字と対応付けられていなければ，現在使用されていない配列添字と対応付ける．この時，使用されていない配列添字が無ければ，論理拡張可能配列を対応次元方向にチャンク単位で 1 つ拡張する．次に，挿入対象のレコードの各カラム値を各次元の配列添字の集合に変換し，チャンク番号とチャンク内オフセットの組に変換，それを RDT に格納する．また，各カラム値のレコードカウンタを 1 つインクリメントする．

また，レコードの削除を行う際には，HORT におけるレコードの削除処理と同様に，まず，削除対象のレコードの各カラム値を CVT を用いて配列添字に変換し，その組をチャンク番号とチャンク内オフセットの組に変換，その組を RDT から削除する．次いで，削除したレコードの各カラム値のレコードカウンタの値を 1 つデクリメントし，0 になれば，CVT からこのカラム値を削除する．

4.5 C-HORT におけるレコードの検索

C-HORT におけるレコードの検索は，RDT に格納されているチャンク番号とオフセットの組に対して行われる．HORT におけるレコードの検索と同様に，検索条件として，1 つのカラムにおいてカラム値が 1 つ指定された場合の単一値指定検索における検索手法を説明する．

まず，カラム値が指定されたカラム l の CVT を EQUAL で探索して論理拡張可能配列の添字に変換し，さらにカラム値をチャンク情報テーブルの添字とカラム値情報テーブルの添字に変換する．この時，カラム値が CVT 内に存在しなければ，指定されたカラム値を持つレコードは存在しない．次に，それらの添字を基に RDT 内の検索を行うが，その検索方法として以下の 3 種類の方法を考える．なお，検索条件として指定されたカラム値のレコードカウンタの値だけの検索対象レコードが見つかった時点で検索を終了する．

4.5.1 検索方法 1

まず，基チャンク部分配列の先頭チャンク番号とチャンク内オフセット 0 をキーとして RDT を GTEQ で探索し，基チャンク部分配列内の先頭レコードを取り出す．その後，NEXT で RDT のシーケンス・セット部を終端まで辿ることにより，基チャンク部分配列以降に拡張されたチャンク部分配列内のレコードを全て取り出す．取り出したレコードであるチャンク番号とチャンク内オフセットの組から，4.3 で説明したチャンク番号とチャンク内オフセットの逆変換方式を用いて，次元 l の配列添字を求めることにより，検索対象のレコードであるかどうかを判定する．

4.5.2 検索方法 2

まず，基チャンク部分配列内の先頭チャンク番号とチャンク内オフセット 0 をキーとして，RDT を GTEQ で探索し，基チャンク部分配列内の先頭レコードを取り出す．その後，NEXT で RDT のシーケンス・セット部を辿ることにより，基チャンク部分配列内のレコードを全て取り出す．ただし，HORT における検索時とは異なり，基チャンク部分配列内のレコード全てが検索対象のレコードではないため，チャンク番号とオフセットの組から次元 l の配列添字を求め，検索対象のレコードであるかどうかを判定する必要がある．

続いて，次元 l 以外の次元に属するチャンク部分配列のうち，基チャンク部分配列よりも大きな経歴値を持つチャンク部分配列それぞれについて，検索対象レコードが存在しうるチャンク内のレコードを全て取り出し，次元 l の配列添字を求め，検索対象のレコードであるかどうかを判定する．

例として，n 次元論理拡張可能配列の次元 l のカラム値が検索条件として指定されたとする．次元 m に属する検索対象要素を含むあるチャンク部分配列 CS の各辺のチャンク数が $[cn_1, cn_2, \cdots, cn_n]$（ただし $cn_m = 1$）である時，検索対象チャンクのチャンク番号が連続している区間の長さは $\prod_{j=1}^{l-1} cn_j$ で求められる．また，この区間はチャンク部分配列内に $\prod_{j=l+1}^{n} cn_j$ 個存在する．

なお，1 つのチャンク内の検索を終了し，次のチャンク内の検索に移る際には，GTEQ で RDT をルートノードから辿った場合と，SQGTEQ でシーケンス・セット部を辿った場合のコスト比較を行い，コストが低い方法を用いて RDT を辿ることとする．

4.5.3 検索方法 3

まず，基チャンク部分配列内の検索を行う．基チャンク部分配列内には，非検索対象要素が含まれているため，検索対象要素のオフセットが連続している区間を求め，検索を行う．例えば，n 次元論理拡張可能配列の次元 l のカラム値が指定された時，チャンクのサイズが $[c_1, c_2, \cdots, c_n]$ であれば，検索対象要素のオフセットが連続している区間の長さは $\prod_{j=1}^{l-1} c_j$ で求められる．また，この区間はチャンク内に $\prod_{j=l+1}^{n} c_j$ 個存在する．

次に，次元 l 以外の次元に属するチャンク部分配列のうち，基チャンク部分配列よりも大きな経歴値を持つチャンク部分配列それぞれについて検索を行う．C-HORT における検索方法 2 と同様に検索対象レコードが存在しうるチャンクを求め，それぞれのチャンク内において検索対象要素のオフセットが連続している区間を求めることにより検索を行う．なお，全てのチャンクのサイズは同じであるので，検索対象要素のオフセットが連続している区間の長さと，チャンク内に存在しているこの区間の数は全て同じである．

また，検索方法 2 と同様に，1 つの検索対象要素のオフセットの連続区間内の検索を終了し，次の検索対象要素のオフセットの連続区間の検索に移る際には，GTEQ で RDT のシーケンス・セット部を辿った場合と，SQGTEQ でルートノードから辿った場合のコスト比較を行い，コストが低い方法を用いる．

この方法を用いることにより，RDT から取り出したレコードが検索対象のレコードであるかどうかを判定する必要はなくなるが，チャンク 1 つにつき最大 $\prod_{j=l+1}^{n} c_j$ 回のコスト比較を行う必要がある．

5. 解析評価

以下では，HORT と C-HORT ならびに従来の関係テーブルの実装方式のコストモデルを作成し，関係テーブル R を実装した際の空間的コストと，単一値指定検索時の時間的コストの評価を行う．以後，従来の関係テーブルの実装方式を CI，HORT による実装方式を HI，C-HORT による実装方式を C-HI と記す．

5.1 コストモデル

以下では，コストモデルを作成するにあたって考慮したパラメータや，コストモデルを簡略化するための仮定について説明を行う．

5.1.1 パラメータ

コストモデルを作成するにあたって考慮したパラメータのうち，主なものを以下にまとめる．なお，サイズの単位は Byte とする．

NR: R 内のレコード数

n: R のカラム数

kl_i: カラム i におけるカラム値の長さ $(1 \leqq i \leqq n)$

L_i: カラム i におけるカラム値の種類

dp_i: カラム i における重複度．すなわち NR/L_i

NP: ディスクの 1 ページ内に格納する B^+-tree のノード数

C_i: チャンクの 1 辺の長さ

5.1.2 仮　　定

コストモデルを簡略化するために，以下のような仮定を行った．

（1） R のカラムの長さは全て同一とする．以後，カラム値の長さを kl とする．

（2） 全てのカラムの重複度は同一とする．以後，重複度を dp，カラムにおけるカラム値の種類を L とする．

（3） チャンクの各辺の長さは同一とする．以後，チャンクの 1 辺長を C とする．

（4） 論理拡張可能配列内でのレコードの分布は一様とする．また，論理拡張可能配列は各次元均等に拡張されることとする．

（5） RDT と CVT のノードのサイズはディスクの 1 ページ内にノードが NP 個入る最大サイズとし，各ノードの有効要素の割合は平均 69%とする．

5.2 空間的コストの評価

5.1 で述べたパラメータと仮定に基づいて構築したコストモデルを用いて，HI と C-HI，CI の空間的コストを比較する．

HI ならびに C-HI の空間的コストは，RDT のコスト + 各次元の CVT のコスト $\times n$+HORT テーブルのコスト $\times n$ で求められる．また，CI ではレコードは入力順に逐次二次記憶上に格納されるとすると，空間的コストは $kl \times n \times NR$ で求まる．

$NR = 1000000$, $n = 5$, $NP = 1$, $kl = 8, 16$, $C = 3000$ とし，dp を 20 から 1000 まで 20 刻みで変化させた時の空間的コストの変化を図 5 に示す．RDT のコストは NR のみに依存するため，dp が変化しても，RDT のコストは変化しない．HORT テーブルのコストは HORT テーブルの 1 つのセルのコスト $\times L \times n$ であるため，dp が小さい場合，すなわち，論理拡張可能配列が疎な場合，HORT テーブルの空間的コストは大きくなる．同様に，CVT のコストも dp に依存する．また，kl を 2 倍にすると CI の空間的コストも 2 倍になるのに対し，HI と C-HI ではカラム値を重複して保持せず，空間的コストの多くが RDT のコストであるため，kl を 2 倍にしても，空間的コストにさほど変化は見られない．C-HI は HI よりも空間的コストが低くなっている．C-HI と HI では，RDT ならびに CVT のサイズはほぼ同じであるため，この差は HORT テーブルのコストの差である．これは，HI では部分配列ごとに保持していた経歴値と係数ベクトルを C-HI ではチャンク部分配列ごとに保持していることによる．

5.3 時間的コストの評価

5.1 で説明したパラメータと仮定に基づいて構築したコストモデルを用いて，HI と C-HI，CI における単一値指定検索時の時間的コストを比較する．ここでは，検索時にアクセス必要なディスクアクセスページ数を基に時間的コストの比較を行う．また，HORT テーブルは主記憶上に展開されているものとする．したがって，HI ならびに C-HI の検索時の時間的コストは，CVT の探索コスト +RDT の探索コストで求められる．

HI ならびに C-HI における検索方法は，3.2 ならびに 4.5 で説明した検索方法 1〜3 を用いる．

$NR = 1000000$, $n = 6$, $NP = 1$, $kl = 8$, $C = 3000$ とし，dp を 20 から 1000 まで 20 刻みで変化させた時の検索時の時間的コストの変化を図 6 に示す．HI の検索方法 1 では，基部分

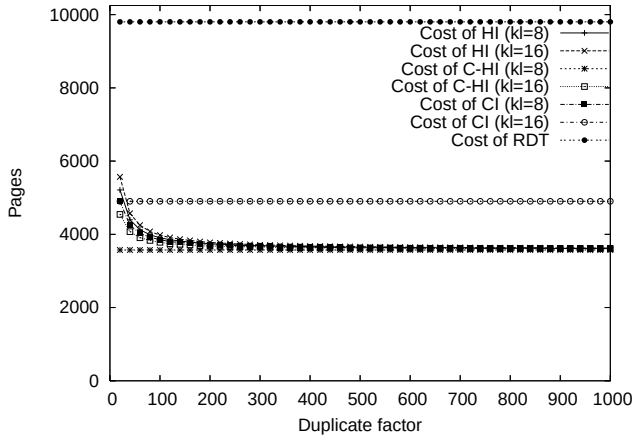

図 5　重複度を変化させた時の空間的コストの変化
Fig. 5　Storage cost with varing duplicate factor

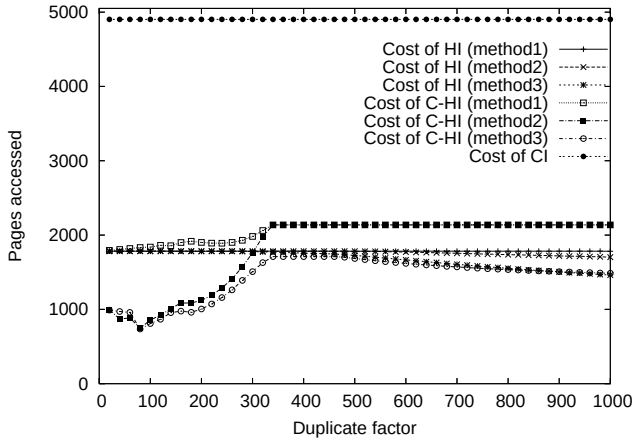

図 6　重複度を変化させた時の検索時の時間的コストの変化
Fig. 6　Retrieval cost with varing duplicate factor

配列以降の部分配列内の全てのレコードを RDT から取り出すため，重複度が変化しても，検索時の平均ディスクアクセス回数は変化しない．また，HI の検索方法 2，3 では，検索対象を絞り込んでいるにも関わらず，検索時の平均ディスクアクセス回数はそれほど減少していない．これは，RDT の 1 ノードをディスクの 1 ページに入る最大サイズに設定しているため位数が大きく，非検索対象レコードが 2 ノード以上にわたって存在する区間が希であったためだと考えられる．C-HI の検索方法 1，2 では，重複度が低い時に検索時の平均ディスクアクセス回数が減少しているが，重複度が 334 以上では全てのレコードが 1 つのチャンク内に納まるため，RDT のシーケンス・セット部の全ノードアクセスする必要がある．したがって，HI よりも検索時の時間的コストが大きくなってしまっている．しかし，C-HI の検索方法 3 では，チャンク内において検索対象要素が存在し得る区間に限定して検索を行うため，チャンクが 1 つになっても全てのノードにアクセスする必要がなく，検索時の時間的コストが抑えられている．また，CI では，全てのレコードを主記憶上にロードしてカラム値をチェックする必要があるため，HI ならびに C-HI の 3〜4 倍程度の時間的コストが必要となっている．

6. ま と め

本論文では，関係テーブルを効率よく実装し，高速な検索を可能とするために我々が提案している HORT と呼ばれる関係テーブルの実装方式における最も大きな問題点である経歴・オフセット空間の狭小の原因が，経歴・オフセット空間を有効に使用できていないということであることに着目し，拡張可能配列を要素単位ではなく，チャンク単位で拡張を行うことによりこの問題を解決した．従来の HORT では，拡張可能配列におけるレコードの位置情報をその要素が含まれる部分配列の経歴値と部分配列内オフセットの 2 つの値を用いて表現していたが，提案した方式では，要素が含まれるチャンク番号とチャンク内オフセットの 2 つの値を用いて表現している．

本方式の利点は，チャンク番号とチャンク内オフセットを用いてレコードを表現するため，より大規模な関係テーブルの実装が可能になったこと．HORT においてカラム値ごとに保持していた部分配列と経歴値と係数ベクトルを，チャンク部分配列ごとに保持するため，空間的コストが低くなること．チャンクを用いることで検索対象レコードの RDT 上での近傍性が高くなるため検索時のディスクアクセス回数が減少することである．また，本方式の欠点は，カラム値の組のチャンク番号とオフセットの組への変換，ならびに逆変換を行うための計算量が，従来の方式と比べ増えることである．

また，コストモデルを用いて従来の関係テーブルの実装方式と HORT，提案した実装方式の空間的コストと検索時の時間的コストの比較を行い，本方式が有効であることを示した．

文　献

[1] Masayuki Kuroda, Naoki Azuma, K. M. Azharul Hasan, Tatsuo Tsuji, Ken Higuchi, “An Implementation Scheme of Relational Tables”, Proc. of ICDE 2005 Workshop, p. 1244, 2005.

[2] K. M. Azharul Hasan, Masayuki Kuroda, Naoki Azuma, Tatsuo Tsuji, Ken Higuchi, “An Extendible Array Based Implementation of Relational Tables for Multi Dimensional Databases”, DaWaK 2005, LNCS 3589, pp. 233-242, 2005.

[3] Heum-Geun Kang, Chin-Wan Chung, “Exploiting Versions for On-line Data Warehouse Maintenance in MOLAP Servers”, Proc. of VLDB 2002, pp.742-753, 2002.

[4] T. Tsuji, A. Isshiki, T. Hochin, K. Higuchi, “An Implementation Scheme of Multidimensional Arrays for MOLAP”, Proc. of the 13th International Workshop on Database and Expert Systems Applications, pp.773-778, 2002.

[5] A. L. Rosenberg, “Allocating Storage for Extendible Array’s”, JACM, Vol. 21, pp.652-670, 1974.

[6] A. L. Rosenberg, L. J. Stockmeyer, “Hashing Schemes for Extendible Array’s”, JACM, Vol.24, pp.199-221, 1977.

[7] E. J. Otoo, T. H. Merrett, “A Storage Scheme for Extendible Array’s”, Computing, Vol.31, pp.1-9, 1983.

[8] A. Novacek, “Using Time Stamps for Storing and Addressing Extendible Arrays”, Computing, Vol.37 pp.303-313, 1986.

[9] 都司 達夫, 水野 剛, 宝珍 輝尚, 樋口 健, “拡張可能配列の遅延割付方式”, 電子情報通信学会論文誌 D-I, Vol.J86-D-I, No.5, pp.351-356, 2003.