MOLAP のための多次元配列のクラスタリングとその評価

土田 隼之† 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
E-mail: †{tsuchida,tsuji,higuchi}@pear.fuis.fukui-u.ac.jp

あらまし MOLAP システムでは基幹 DB に蓄積された関係データベースがダンプされ，集約されたファクトテーブルが多次元配列に格納される．多次元配列の高速なランダムアクセス性を生かして，ファクトデータに対して集約演算をはじめ種々の統計処理を効率よく行うことができる．この高速アクセス性は配列の各次元サイズが固定であることに依っている．ここでは，前回ダンプした配列データを再配置することなしに，差分のみのダンプを行い，多次元配列を構築する．このために，我々が提案している動的な多次元データの実装方式によりシステムを実装する．本論文では，基幹 DB とのリアルタイムの一貫性を確保する必要がないことに注目して，クラスタリングされた多次元配列を高速に構築するための改良方式を提案して評価する．さらにこの多次元配列からクラスタリングされた固定サイズの多次元配列を効率よく構築する方法を提案して評価する．

キーワード MOLAP，バッファリング，チャンク

1. まえがき

近年，業務用基幹データベースとは別に，意思決定を支援するためにバックエンドでデータ分析用の多次元データベース[3]の運用が盛んになってきている．このような多次元データベースでは，オンライン分析[4]に必要な大量のデータに対する要求を迅速に処理する必要がある．

多次元データの格納に多次元配列[1][2]を使用する MOLAP[5][8]では瞬時の応答性は求められないが，使用者の思考を中断させない程度の速度は要求される．パフォーマンスを確保するために，特に範囲検索の速度向上が重要である．そのため，「カラム値の距離が互いに近傍にあるデータは，二次記憶上の物理距離においても近傍に位置していることを保証」する『近傍性』確保のためのクラスタリングが行われる．各次元において，この近傍性を保証するため，しばしば配列全体を部分配列に分割するチャンク化[1][2]が行われる．チャンク化では，同一のチャンク内データ要素は通常，同一ディスクページに格納される．ところが，論理的には連続しているデータ要素が入力順によっては複数のチャンクに納められる場合に，それらの要素への範囲検索では，多くのディスクアクセスが生じ，時間的コストが増大する．

一方 MOLAP 用のバックエンドデータベースには『一週間や一日に一度など定期的にフロントエンド DB からのダンプを行えば良く，フロントエンド DB とのリアルタイムの一致性が要求されない』という特性がある．[12]では，この特性を利用して，近傍性を保証するためのクラスタリングコストを軽減する方式を提案している．この方式の基本アイデアは[9]で提案されているが，[12]では，実装方式として我々が提案している HOMD (History Offset implementation scheme for Multidimensional Datasets)[10][11]を用いている．この方式では，通常の固定配列を用いた多次元データベース方式とは異なり，新規カラム値の追加に対して，それまでに存在している配列データを再配置することなく対応次元の配列サイズを拡張することができる．この拡張可能性に注目して，MOLAP 用の多次元配列データを差分構築している．しかし，[12]の方式では，差分構築前の元の多次元配列データを大量に移動する必要性が高く，差分構築のコストを悪化させていた．

本論文ではこの悪化を軽減し，差分構築のコストをさらに抑制するための方式を提案して評価する．さらにこの方式で得られた多次元配列からクラスタリングされた固定サイズの多次元配列を効率よく構築する方式を提案して評価する．

2. HOMD の概要

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 要素の位置として扱うことができる．

品目	色
定規	青
ノート	黄
鉛筆	黄
鉛筆	緑

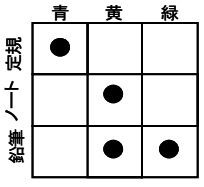

図 1 多次元配列を用いた関係テーブルの実装例

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

のはただ一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．

2.1. 拡張可能配列による関係テーブル実装方式

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[6]~[8]．拡張可能であるために，関係テーブルレコードの動的な追加に対して，配列要素の再配置なしに効率よく対処できる．

n 次元拡張可能配列は，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,係数テーブルの 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一つインクリメントされ，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記憶領域を動的に確保し拡張可能配列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，以下，部分配列という．

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 方向に一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．拡張可能配列が 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一次関数の *n-2* 個の係数からなる係数ベクトルを部分配列ごとに係数テーブルに記録する．例えば，上記の部分配列内の要素(i_1, i_3, i_4) のアドレスは一次関数$s_1s_3i_4 + s_1i_3 + i_1$ で求められる．この(s_1s_3, s_1) を係数ベクトルと呼ぶ．配列要素のアドレス計算は，例えば図 2 の配列要素(4,3)について，次のように行われる．

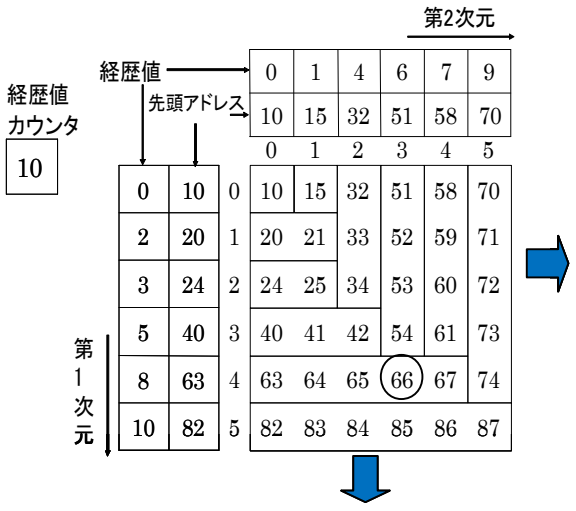

図 2 拡張可能配列

まず，第 1 次元の添字が 4 である部分配列の経歴値 8 と，第 2 次元の添字が 3 である部分配列の経歴値 6 を比べ，6<8 であるので要素(4,3)は，経歴値 8 の部分配列に含まれる．また，この部分配列の先頭アドレスは 63 であり,要素(4,3)は部分配列内の先頭要素からのオフセットは3であるため,求めるアドレスは63+3=66となる．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できる．

2.2. 経歴･オフセット法と HOMD データ構造

固定配列ならびに拡張可能配列を用いた関係テーブルの実装方式では，配列領域全体の記憶領域を確保する必要があるが，一般に関係テーブルを多次元配列を用いて実装すると，図 1 に見るように，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，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ことができる．例えば，図 2 の配列要素(4,3)の位置を経歴･オフセット法を用いて表現すると，要素(4,3)が含まれる部分配列の経歴値は 8，要素(4,3)の部分配列内でのオフセットは 3 であるため，<8,3>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を特定し，オフセットをその部分配列の係数ベクトルの各項の値で順次除算する．このような多次元データのエンコード法を経歴・オフセット法という．

HOMD は経歴・オフセット法を用いた多次元データの実装方式である．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，レコードの位置情報を表す 2 つの値の組を RDT(Real Data Tree) と呼ぶ B^+木 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，疎配列問題に対処できる．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. 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に配置される．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．関係テーブルのすべてのカラムが経歴・オフセット法によるエンコードの対象になるならば，RDT はキーのみを有する．しかしここでは OLAP を指向しておりファクトデータの存在を前提とし,これはキーに対するデータとして RDT のデータ部に格納される.図 3 に HOMD データ構造の例を示す.

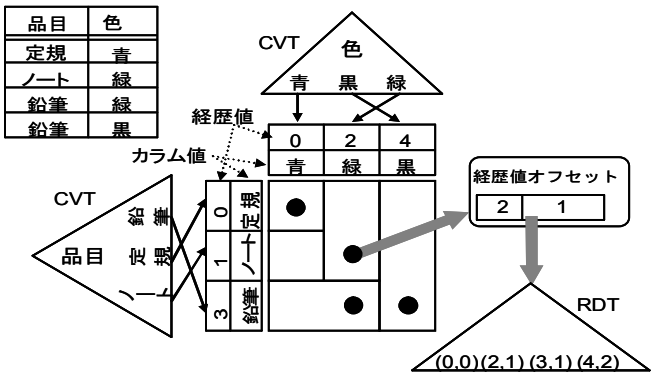

図 3 HOMD データ構造

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のために，関係テーブルの各カラムに CVT(key-subscript ConVersion Tree)と呼ぶ B^+木 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．配列添字からカラム値への逆変換のために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．以後，各次元の補助テーブル群のことを，HOMD テーブルと呼ぶ．

3. チャンク単位で拡張を行う HOMD

固定サイズの多次元配列を同じ次元の超立方体に分割，各々の分割を“チャンク”として二次記憶ページへの格納単位とすることが，しばしば行われる[1][2]．ここでは，HOMD における論理拡張可能配列をチャンク化する．今までのように，配列要素を単位として拡張を行うのではなく，チャンク単位で拡張を行うチャンク化拡張可能配列を採用する．この配列ではチャンクを一意的に識別するためのチャンク番号をチャンクごとに付与する．チャンク単位に拡張されるチャンク部分配列ごとに，その拡張の順番を示す経歴値とそのチャンク部分配列内の先頭チャンクの番号を持たせる．

関係テーブルをチャンク単位で拡張を行う拡張可能配列で表現すると，拡張可能配列内でのレコードの位置情報を，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する．以後，チャンク単位で拡張を行う HOMD のことを C-HOMD [11］と呼ぶ．

3.1. C-HOMD の構造

図 4 にチャンク単位で拡張を行う C-HOMD の構造例を示す．C-HOMD における各次元の C-HOMD 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，チャンク部分配列の経歴値，先頭チャンク番号および係数ベクトル等が格納される．また，カラム値情報テーブルはカラム値ごとに作成され，カラム値等が格納される．

チャンクの一辺サイズに相当する C-HOMD テーブルの連続部分をテーブルノードという．各次元の CVT にはカラム値に対応する C-HOMD テーブル中のカラム値情報テーブルの添字が格納され，RDT にはチャンク番号とチャンク内オフセットの対がキーとして格納される．このとき，チャンク番号はキーの上位バイト，オフセットは下位バイトに格納されるので，RDT のシーケンスセット上では，同じチャンク番号のレコードが連続して配置される．通常の HOMD で使用される，2.1 節で述べた，経歴値テーブルや係数テーブルは，C-HOMD では，チャンク部分配列に対して確保されるので通常の HOMD の場合より，”1/チャンクの一辺サイズ”に減少する．

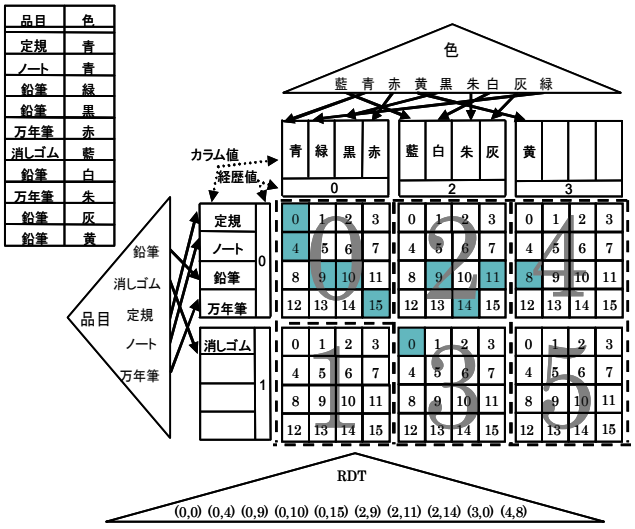

図 4 C-HOMD の構造

3.2. C-HOMD でのレコード検索

新規カラム値の挿入によって拡張されたチャンク部分配列をそのカラム値の「基チャンク部分配列」という (図 4 では，チャンク番号 1 を先頭とするチャンク部分配列は”品目”のカラム値“消しゴム”の基チャンク部分配列であるが，チャンク番号 2 および 4 をそれぞれ先頭とするチャンク部分配列はカラム値“消しゴム”の基チャンク部分配列ではない) ．

ある次元のカラム値 v を指定した場合の C-HOMD での単一値指定レコード検索は，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以下のように行われる．まず，v を当該次元の CVT で検索し，そのカラム値の C-HOMD テーブルでの添字を知り，v の基チャンク部分配列の先頭チャンク番号を引き当てる．その先頭チャンク番号とチャンク内オフセット 0 の対をキーとして，RDT をルートノードから探索し，

このキー値以上の最小のキーを検索する．この最小のキーは当該基チャンク部分配列内の先頭レコードである．その後，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順次アクセスする．シーケンスセット内のチャンク番号とオフセット対の内から逆変換により検索対象の次元の配列添字を求め，検索対象のレコードであるかどうかを判定する．この逆変換では，オフセットの値をチャンクのアドレス関数の係数で割り算することで，当該添字を求めることができる．

続いて，検索対象の次元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大きな経歴値を持つチャンク部分配列のそれぞれについて，検索対象レコードが存在し得るチャンク内のレコードを全て取り出し，検索対象次元の配列添字を求め，検索対象レコードであるかどうか判定する．

また，範囲検索では範囲内のカラム値をもつレコードに対する検索が，上記単一値指定検索とほぼ同様の手順で行われる．すなわち，範囲内のカラム値の基チャンク部分配列集合 B として，B の基チャンク部分配列の経歴値の最小値を h_m とすると，B の基チャンク部分配列をすべて調べた後，経歴値が h_m より大きい他の次元のチャンク部分配列を調べる．

4. C-HOMD におけるクラスタリング

HOMD や C-HOMD では，新たなカラム値は入力される時間順に論理拡張可能配列の次の添字位置にマッピングされる．したがって，CVT のシーケンスセットでのカラム値の順序がマッピングされた添字の値順と一致しないため，カラム値の範囲検索が不利になる．すなわち，CVT の一つのシーケンスセットノードのカラム値集合に対する添字集合は一般には，図 4 のように複数のチャンクにマッピングされる．また，テーブルノードはチャンクのその次元の一辺に 1 対 1 に対応する．従って，例えば，図 4 に示すように CVT のシーケンスセット上のカラム値（藍，青，赤，黄）は 3 つのチャンク添字にマッピングされる．同一チャンク中のレコードは RDT 上のシーケンスセットにおいて，連続して配置されており，シーケンスセット上のノードを不連続にアクセスする必要がある．これは範囲アクセスのパフォーマンスを極端に悪化させる．

図 4 は，近傍性を考慮しないマッピングであるのに対して，考慮しているマッピングではシーケンスセット上で連続するカラム値は可能な限り同一テーブルノードにマッピングする．例えば，図 5 のように，この近傍性が考慮されている場合には，カラム値｛A 社, B 社, C 社｝の範囲検索を行った場合には 2 つのチャンクを調べるだけで良い．しかし，近傍性が考慮されていない場合は 4 つ全てのチャンクを調べる必要があることがある．

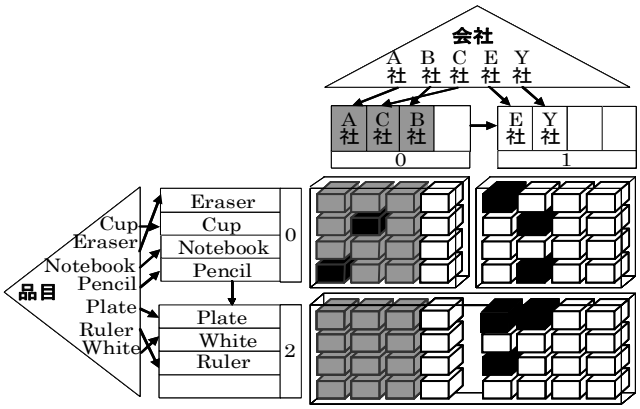

図 5 近傍性確保

C-HOMD の動的な拡張性を活かしながら，このようなカラム値についてのクラスタリングを行うには大きな時間的コストを必要とする．すなわち，新たなカラム値の入力によるシーケンスセットノードの分割が発生すれば近傍性を確保するために，RDT 内の既存データを他のチャンクに移動する必要性があるためである．

5. クラスタリングされた C-HOMD の差分構築

この節では 4 節で述べたカラム値の近傍性を確保するクラスタリングのための時間的コストを回避するための方式を提案する．[12]では，MOLAP システムにおける，効率的な「クラスタリングされた C-HOMD 差分構築のためのバッファリング機構」を提案した．

まず，[12]におけるバッファリング機構の概略について述べる．近傍性を考慮する場合，フロントエンドの RDB からのデータのダンプに要する時間的コストが非常に高くなる．レコードデータの挿入に伴う CVT のシーケンスセット内での C-HOMD テーブルに対する索引の付け替えに従い，RDT 内のデータ本体も移動させなければならない．通常，索引とデータの移動は両者の一貫性を確保するために同時に行われる．したがって，同一ダンプ中に同じデータ要素が複数回移動する可能性がある．例えば，図 5 において，以後，いくつかのデータレコードが追加され，近傍性を確保するために会社名が Y 社である品目データが別のチャンクに移動されたとする．引き続くデータレコードの挿入により，Y 社を含むシーケンスセットノードの分割が行われ，一度移動させられた Y 社の品目データが，再び分割によって別のチャンクに移動させられることがありうる．このように，近傍性を確保するためには，同じデータ要素が RDT 内で繰り返し，大量にチャンク間を移動する可能性がある．

[12]では実時間性が要求されないという MOLAP シ

ステムの特性を生かし，入力データのバッファリングを行うことにより，この問題を回避している．この問題は，一貫性を確保するために索引とデータの同時移動を行うことに起因しているが，MOLAP で使用する多次元配列においては，ダンプ途中での個々のデータレコード毎の一貫性の確保を必要としない．ダンプはバッチ処理であり，その間はデータの分析処理は通常行えない．このことに注目し，一回のダンプにおける索引の挿入とデータの挿入を分離する．つまり索引の最終的な位置が C-HOMD テーブル上で確定してから RDT へのデータの挿入を行うようにする．こうすることにより，データ要素の移動を繰り返すことなく，そのダンプにおける二次記憶上の最終位置にデータ要素を挿入することができる．ダンプ中は索引とデータの一貫性がないが,終了時には一貫性は保証されている．

[12]における方法では新規挿入される差分データ要素は，チャンク間移動なしにダンプできる．しかし，ダンプする以前の既存データ要素は差分データ要素のダンプにおける新規カラム値の出現に伴う分割に応じて，逐次移動を行う．このため，同一差分ダンプにおいて，複数回，同じデータ要素がチャンク間を移動する可能性がある.この問題は C-HOMD テーブル上における既存要素の最終位置が確定していない段階で既存要素に対応する実データの移動を行うのが原因である．

本論文では，差分ダンプ以前の既存データ要素に関しても新規差分要素のダンプと同様に最終位置が確定した後に既存データ要素のチャンク間移動を行うための方式を提案する．これにより差分ダンプ以前に既に存在している既存データの移動を既存要素毎に高々一回に抑制する．提案するアルゴリズムでは，既存データ要素の複数回の移動を防ぐため，MOVE と呼ぶ別途確保されるメモリ上のテンポラリデータ構造を利用する.既存データ要素の移動の必要性が生じたときには，直ちに対応するデータ要素を移動させるのではなく，代わりにこの MOVE に移動情報を登録または更新する．CVT 構築が終了し，既存データ要素に関しても C-HOMD テーブル上の最終位置が確定した後で，MOVE の格納情報にしたがって，既存データ要素のチャンク間移動を行う．

以下に提案方式によるクラスタリングされた C-HOMD 差分構築の手順概要を説明する．当該期間の追加データがすでにログデータファイルに格納されているものとして,この期間以前に C-HOMD データとして蓄積されている既存データに対して，ログデータファイルのレコードを追加・集約して，最新の C-HOMD データを得る差分ダンプとする．スクラッチから多次元配列データを構築する必要はない．

ログバッファ内の差分データレコードの挿入・集約処理は以下の３段階の処理に分かれる．第一，第三段階の処理において，ログデータファイルを先頭からスキャンする．ログデータファイル読み込みにかかる時間は，要素の挿入・移動にかかる時間に比べればほぼ問題にならない．

例として，前回のダンプ時までに，図 6 のようなデータ（灰色）が格納されているとする．この例でのチャンクの各次元サイズは 4 である．

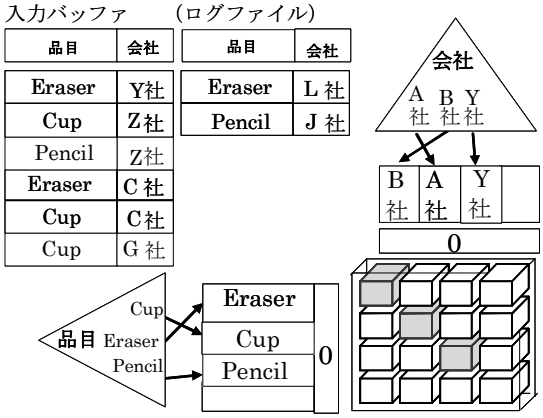

図 6 差分構築前　（初期状態）

(1)新規カラム値挿入と CVT における最終位置決定

入力バッファを先頭レコードからスキャンしていき，新たなカラム値(会社 Z 社)が現れたときには対応する CVT へカラム値を追加し，C-HOMD テーブルを更新する.新規挿入データ要素そのものの RDT への挿入は行わない．入力バッファ内の引き続くレコードデータについても CVT には挿入するが，RDT には挿入しない．次に，カラム値 C 社が現れて CVT へ追加するが,この時は C-HOMD の当該テーブルノードが満杯になっているのでノード分割が発生する．この時，新たなノードにカラム値 Y 社と Z 社を移動する．併せて対応する既存の<Y 社，Pencil>のデータ要素のチャンク間移動を行う必要があるが，実際に移動は行わず，上記データ構造 MOVE にただ記録する．同じ既存データ要素が複数回移動する必要があっても，MOVE の更新のみを行う．第一段階での C-HOMD を図 7 に示す．

(2) 一貫性確保

(1) の処理を実行後, CVT と C-HOMD テーブルは整合しているが，これらと RDT は未整合である．すなわち，RDT 中の移動すべき既存データ要素のキー値（<経歴値，オフセット>）が未更新のままである．新規挿入データ要素を実際に RDT へ追加する前に,既存データ要素を移動させて CVT，C-HOMD テーブル，ならびに RDT 間の一貫性を確保する.ダンプ以前に既に登録されているカラム値が移動する場合は MOVE を参

照して各次元の情報が格納されているか調べて，格納されていれば移動元添字情報と移動先添字情報を元にRDT 内での要素移動を行い，一貫性を確保する．

(3) 差分データ要素の追加と集約

(1) により，挿入すべきデータレコードの各カラムについて，CVT 上での最終格納位置が確定するので，続いて，差分データ要素の RDT への追加とファクトデータ値の集約を行う．これには，入力バッファを再度はじめからスキャンしながら，CVT をサーチして，各レコードの添字の組を得る．この組を＜チャンク番号，オフセット＞対に変換してキー値として，また，ファクトデータをデータ値として RDT に格納する．この当該位置にすでに，ファクトデータが格納されていれば，集約してファクトデータを更新する．

図 7 が処理を終えた状態で，灰色の要素が最初から存在し移動していない要素．黒色の要素が最初から存在したがノードの分割によって移動した要素(Y 社，Pencil)．青色の要素が新たに加えられた要素となる．

6. 既存データ要素の移動手段の実装

本節では，前節で概説したクラスタリングされた C-HOMD の差分構築の 3 つの段階のうち，段階(1)(2)に示した既存データ要素の移動手順の実装について，テンポラリデータ構造 MOVE の役割を中心に説明する．移動するデータ要素の添字情報を格納するために MOVE は次元ごとに確保される 2 種類のデータ構造からなる．(a)S-MOVE: 移動対象の移動元添字と移動先添字を格納する構造体集合，(b)C-MOVE: C-HOMD テーブル中のカラム値ごとに対応する S-MOVE の構造体の位置を格納するための一次元配列．例を図 7 に示す．

(1) 索引挿入

ログデータファイルをスキャンして，新たなカラム値について，CVT への挿入，C-HOMD テーブル中のチャンク情報テーブル，カラム値情報テーブルの更新を行う．新規カラム値の挿入時には，テーブルノード分割の発生を検査する．分割が発生したときには，当該次元方向に 1 チャンク分拡張する．その後，既存カラム値が移動対象かどうか調べ，移動対象である場合には，S-MOVE に既登録でなければ S-MOVE の構造体を割り付け，移動情報を登録し，C-MOVE の配列の当該要素にこの構造体の位置を登録する．既登録ならば，2 回目以上の移動であり，当該構造体の移動先添字のみを更新する．また，分割が発生しなかった場合は通常の索引挿入処理を行う．一つのレコードのカラム値の挿入によって複数の次元でテーブルノード分割が発生した場合は，次元の若い順に処理が行われる．

(2) 一貫性確保

CVT，C-HOMD テーブルと RDT の整合性を確保するために，既存データ要素の移動を行う．これには，RDT のシーケンスセットの先頭からシーケンシャルにアクセスしてデータ要素を取り出し，各データ要素の添字情報を取り出す．C-MOVE 配列の対応する要素に S-MOVE の構造体の位置が登録されていれば，移動が必要であり，この構造体の移動情報にしたがって，要素移動を行う．

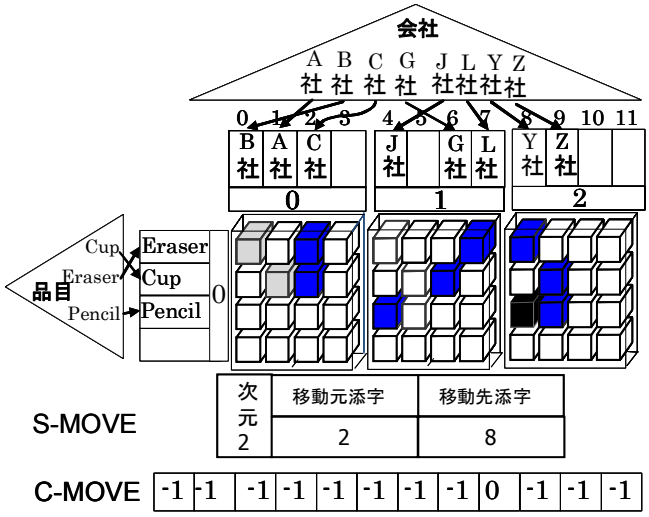

図 7 差分構築後　（終了状態）

7. C-HOMD からのクラスタリングされたチャンク化集約済み固定配列の構築

5, 6 節で提案したクラスタリング方式では，ノード分割の際にそのノードのカラム値上位半分を移動させている．これは，B^+木におけるノード管理のように，引き続く新規カラム値の挿入によりノード分割を多発させないためである．このために C-HOMD テーブルの充填率が低くなり，結果として，各次元においてチャンク数が増大し，C-HOMD 全体としてチャンク総数が増加する．これは検索時のアクセスチャンク数を増加させ，検索時間を増大させる可能性がある．

そこで，特に，検索速度が要求される場合には，通常の MOLAP システムにおけるように，各次元固定サイズの固定配列を構築して，負荷の高い検索に使用する．しかし，差分構築によらず，元のログデータに基づいてスクラッチから集約値を計算して格納する時間的コストは高い．この欠点を軽減するために，ここではデータ集約を行う元データとして 5, 6 節で述べたクラスタリングされた集約済みの C-HOMD を用いる．集約済みの C-HOMD から，カラム値をクラスタリングしたチャンク化固定配列(以下，C-FA (Chunked Fixed Array)と呼ぶ)を作成する．C-FA は図 8 に示す添字変換配列ならびにカラム値参照配列を準備することにより

C-HOMD の CVT を共有できる．添字変換配列は C-HOMD での添字を C-FA の添字に変換する．この変換により，C-FA のカラム値の順は変換された添字の順に一致するので，カラム値の近傍性が確保できる．C-FA の添字にはこの変換された添字が使用され，C-FA のチャンク番号とチャンク内オフセットの組が計算されて，C-FA の RDT に格納される．

C-HOMD の CVT にはすべてのカラム値がすでに出揃っており，C-HOMD からの C-FA の作成では，元のログデータの二度読みを行う必要が無い．また，C-HOMD の RDT にはすでに集約済みデータが格納されているため，集約する必要もない．

C-HOMD から C-FA を構築するためのおもな時間的コストは，C-HOMD の RDT に格納された，<チャンク番号，チャンク内オフセット> 対を上記の添字変換配列を使って，C-FA の<チャンク番号，チャンク内オフセット>対に変換するコストである．変換には，C-HOMD の RDT のシーケンスセット上のデータ要素のキー値である<チャンク番号，チャンク内オフセット>対を添字座標にデコードした上で，添字変換配列を使って，C-FA の<チャンク番号，チャンク内オフセット>対にエンコードした上で C-FA の RDT に集約済みデータ値とともに挿入する．

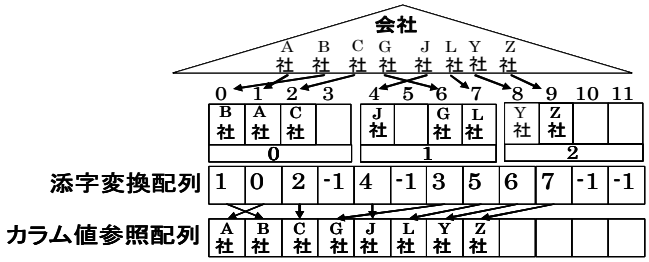

図 8 添字変換配列とカラム値参照配列

8. 評価

実装したクラスタリングシステムでの実測値を比較評価する．実測環境は，Sun Fire E4900 (CPU : UltraSPARC IV (1050MHz), メモリ容量 : 48 GB, ディスク容量 : 73.4 GB×12(RAID5) , OS : Solaris 9)である．

クラスタリングの効果については，すでに[12]において，クラスタリングにより範囲検索の速度が著しく向上すること，ならびに，検索範囲が広くなるほどクラスタリングが有効であることが報告されている．ここでは，まず，6 節で提案した差分構築方式の効果を測定・評価する．

8.1. 差分構築方式の評価

まず，ある時点において，C-HOMD に蓄積されている初期データに対して差分データのサイズを変化させたときの差分構築の時間を評価する．この初期データを 5 次元 200 万レコード，各次元のカージナリティを 1000 とする．また，差分データのレコード数を 2 万レコードずつ増加させながら，差分構築を繰り返す．差分 2 万レコード毎に各次元の新規カラム値のカージナリティが 5 または 10 増加するとし，初期・差分ともに入力データは一様分布とした．

このとき，初期の 200 万レコードの C-HOMD への差分レコードのみの追加・集約にかかる時間をそれぞれ測定した．また，同一の初期データ＋差分データを使用してクラスタリングされた固定配列 C-FA をスクラッチから構築する時間を同時に測定した．

図 9 はその結果である(F:初期データのカージナリティ，S:差分データのカージナリティ増分，CH:チャンクの一辺サイズ)．当然ながら差分データが増加すれば，差分構築に時間がかかる．また，S のカージナリティ増分が増加すると挿入時間が増加している．これは，カージナリティの増加に伴って初期の C-HOMD においてノード分割の頻度が増加し，移動要素が増加することに起因している．C-FA との比較では，（CH50，差分二十万レコード，カージナリティ 1100(1000+10×10)）の追加挿入においても差分構築の方が C-FA の構築よりも高速である．なお C-FA 構築時間は入力レコード数に単純に比例するが，C-HOMD の差分構築の場合は移動要素の量が挿入時間に大きく影響を与え，移動要素数はチャンクサイズとカージナリティの関係に依存するため，同一入力データであってもチャンクサイズによって挿入時間が異なる．

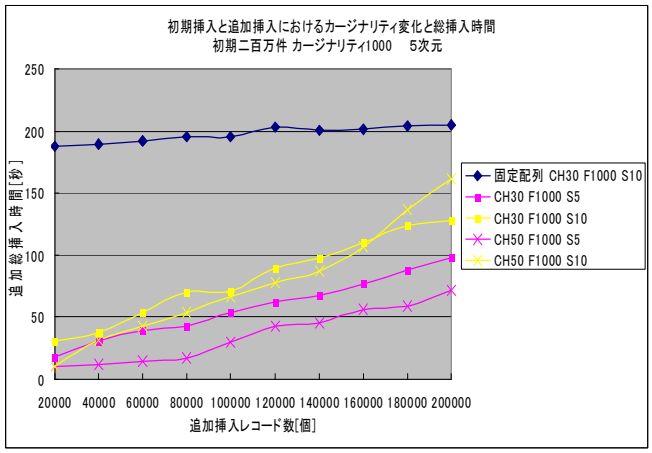

図 9 既存要素移動抑制　カージナリティ変化

そこで，チャンクサイズを変化させた時の差分構築時間の変化を調べた．結果を図 10 に示す．チャンクサイズが大きくなるほど，分割の周期が大きくなり，挿入時間曲線が緩やかになっている．また，カージナリティよりもチャンクサイズが大きい場合は分割が一切

発生しないため，移動要素がまったくないので，非常に高速に差分構築が可能になる．

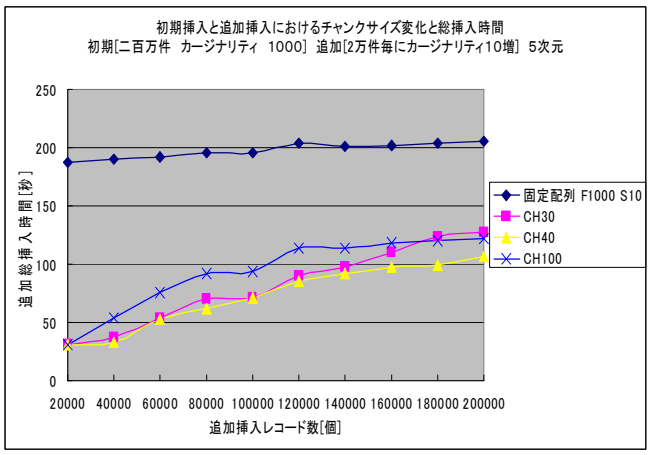

図 10 既存要素移動抑制 チャンクサイズ変化

8.2. C-HOMD からクラスタリングされた C-FA 構築

続いて 7 節で述べた C-HOMD からのクラスタリングされたチャンク化固定配列構築の効果を評価する．C-HOMD から固定配列を構築した場合，①重複レコードがすでに集約されているために挿入レコード総数が削減される，②C-HOMD の CVT を共用することができ，新たに CVT 構築を行う必要がない，というメリットがある．その検証を行うため，入力データの重複レコード割合を変化させて挿入時間を測定した．図 11 はその結果である．

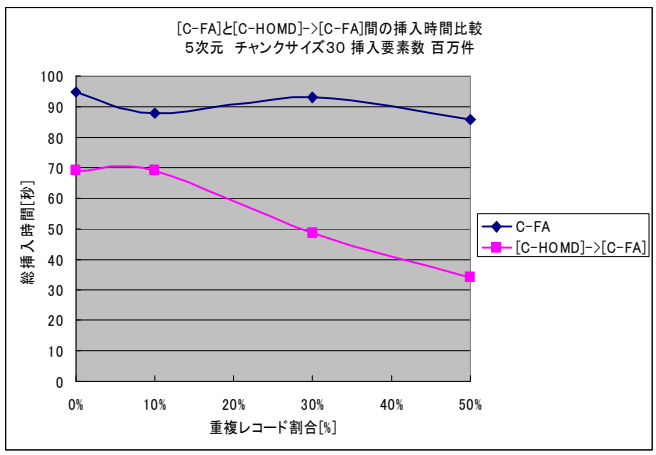

図 11 重複レコードの割合変化と挿入時間

①重複レコード集約の効果により，重複割合が大きくなるにしたがって挿入時間が減少し，重複割合 50%では 0%の半分程度になっていることがわかる．また，②CVT を共用できるために，重複割合が 0%の場合でも C-HOMD を経ずに直接固定配列を構築した場合と比べて高速である．

9. まとめ

本論文では，基幹 DB とのリアルタイムの一貫性を確保する必要がないことに注目して，MOLAP のためのクラスタリングされた多次元配列を高速に差分構築するための改良方式を提案して評価した．さらにこの多次元配列からクラスタリングされた固定サイズの多次元配列を効率よく構築する方式を提案して評価した．これらの評価の結果から提案方式が有効に機能し，高速に差分構築が可能なことを示した．

参 考 文 献

[1] Seamons, K. E. and Winslett, M.: “Physical schemas for large multidimensional arrays in scientific computing applications”, Proceedings of SSDBM, pp.218-227 (1994).

[2] Zhao, Y., Deshpande, P. M. and Naughton, J. F.: “An array based algorithm for simultaneous multidimensional aggregates”, Proceedings of ACM SIGMOD, pp.159-170 (1997).

[3] Pedersen, T. B. and Jensen, C. S.: “Multidimensional database technology”, IEEE Computer, 34(12), pp.40-46, (2001).

[4] Bengtsson, F. and Chen, J.: “Space-Efficient Range-Sum Queries in OLAP”, Proceedings of DaWaK pp.87-96 (2004).

[5] Kang, H. G. and Chung, C. W.: “Exploiting Versions for On-line Data Warehouse Maintenance in MOLAP Servers”, Proceedings of VLDB Conference, pp.742-753 (2002).

[6] Otoo, E. J. and Merrett, T. H.: “A Storage Scheme for Extendible Arrays”, Computing, Vol.31, pp.1-9 (1983).

[7] Novacek, A.: “Using Time Stamps for Storing and Addressing Extendible Arrays”, Computing, Vol.37 pp.303- 13 (1986).

[8] Rotem, D. and Zhao, J. L.: “Extendible Arrays for Statistical Databases and OLAP Applications”, Proceedings of SSDBM, pp.108-117 (1996).

[9] Tsuji, T., Hara, A. and Higuchi, K.: “An Extendible Multidimensional Array System for MOLAP”, Proceedings of ACM Applied Computing, pp.503-510 (2006).

[10] Azharul Hasan, K. M., Tsuji, T. and Higuchi, K.: “An Efficient Implementation for MOLAP Basic Data Structure and its Evaluation”, Proceedings of DASFAA, pp.288-299 (2007).

[11] 相羽，黒田，都司，樋口: “チャンク化拡張可能配列による関係テーブルの実装と評価”，電子情報通信学会，データ工学ワークショップ，A2-5 (2007).

[12]土田，都司，樋口: “MOLAP 用多次元配列構築のためのバッファリング方式”，電子情報通信学会，データ工学ワークショップ, D1-5 (2008).