MOLAP 用多次元配列構築のためのバッファリング方式

A Buffering Scheme for Constructing Multidimensional Arrays for MOLAP

土田 隼之♥　都司 達夫♦
樋口 健♦

Takayuki TSUCHIDA　Tatsuo TSUJI
Ken HIGUCHI

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その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データバッファリングの方式を提案して評価する．

In MOLAP systems, multidimensional arrays are employed to store fact tables dumped from the frontend RDB. On these fact tables, various kinds of statistical computations such as aggregate operations can be performed efficiently by utilizing the fast random accessing capability of arrays. This capability depends on that the size of an array is fixed in every dimension. In this paper, we provide an incremental construction scheme of multidimensional array for MOLAP, in which only the difference from the previous dump is dumped. The scheme has been implemented by using an implementation scheme of dynamic multidimensional datasets developed by our laboratory. In this paper, observing that the real time consistency of the dumped data with the corresponding frontend RDB table is not required, we propose and evaluate an input data buffering scheme for fast construction of clustered arrays.

1. はじめに

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．

♥ 学生会員　福井大学大学院工学研究科博士前期課程
tsuchida@pear.fuis.fukui-u.ac.jp
♦ 正会員　福井大学大学院工学研究科
{tsuji, higuchi}@pear.fuis.fukui-u.ac.jp

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．全体のパフォーマンスを確保するために，範囲検索の速度を向上させることが重要である．そのために，「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．この近傍性を保証するために，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．チャンク化されると，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．

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．本研究では，この特性を利用して，『二次記憶への実データの挿入と索引データの挿入を分離』することにより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．本論文ではこのための入力データのバッファリング機構の方式設計と実装，ならびに評価を行う．本方式の基本アイデアは[9]で提案されているが，ここでは，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]を用いた．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する．

2. HOMD の概要

2.1 多次元配列による関係テーブル実装方式

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．

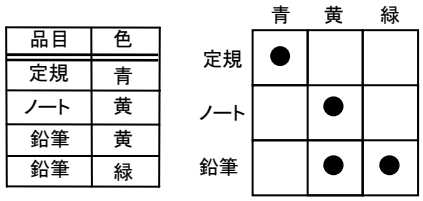

図 1 多次元配列を用いた関係テーブルの実装例
Fig.1 Implementation of a Relational Table Using a Multidimensional Array

2.2 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・削除に対して，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．

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．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．

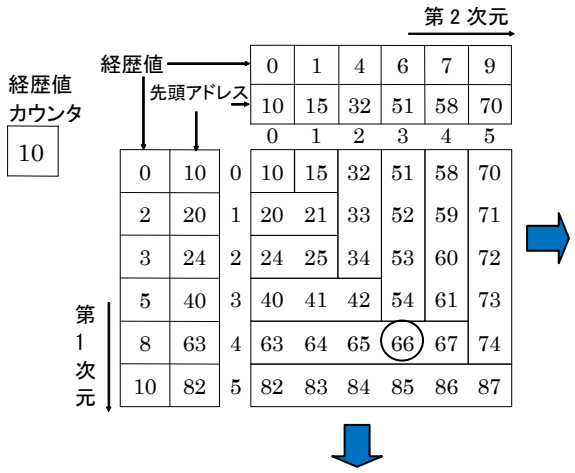

図 2 拡張可能配列
Fig.2 An Extendible Array

2.3 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．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．

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．また，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．図 3 に HOMD データ構造の例を示す．

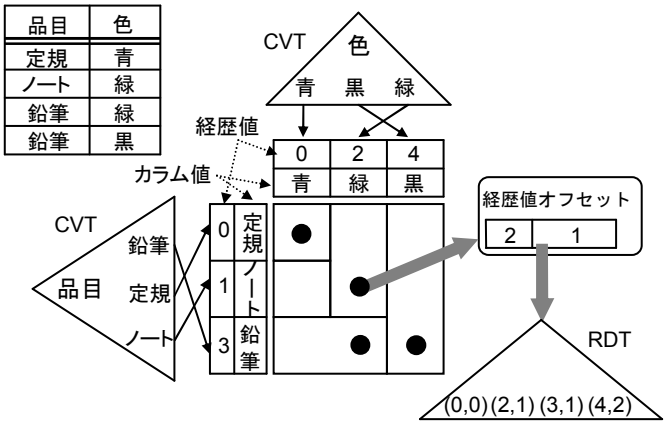

図 3 HOMD データ構造
Fig.3 A HOMD Data Structure

3. チャンク化とチャンク単位の拡張

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．また，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，各チャンクにチャンク番号と，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 HOMD の構造の例を示す．C-HOMD における各次元の 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．各次元の CVT にはカラム値に対応する 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，2.2 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”に大幅に減少する．

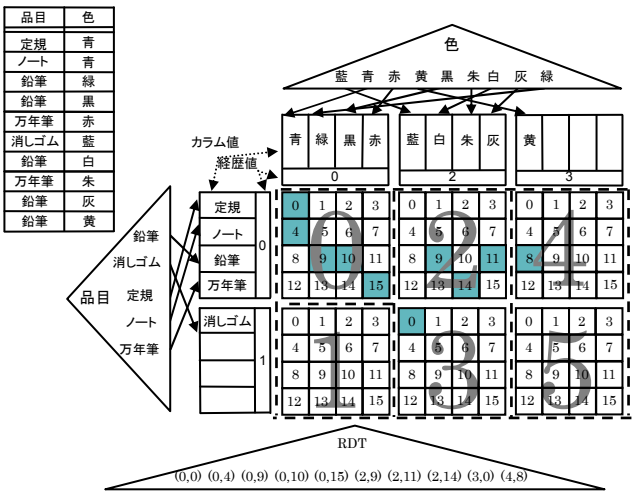

図 4 C-HOMD の構造
Fig.4 C-HOMD Data Structure

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 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ればよい．

4. カラム値の近傍性確保

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ので，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．図 4 では一つのシーケンスセットノードには 4 つの添字値が格納でき，たとえば，（藍，青，赤，黄）のノードのカラム値は 3 つのチャンクの添字にマッピングされる．一つの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いるが，このときには，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 では，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一つのシーケンスセットノードのカラム値集合は常に，一つのチャンクの添字集合にマッピングする．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．

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．

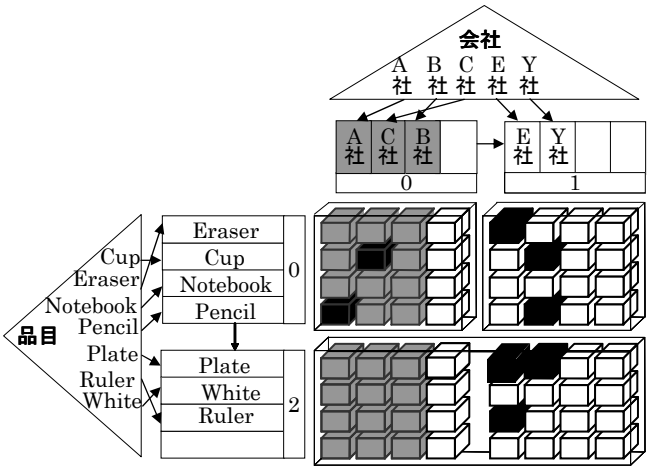

図 5 近傍性確保
Fig.5 Preserving Proximity

5. 入力データのバッファリング

4 節では，カラム値の近傍性を確保するための時間的コストが高いことを述べた．この節ではこのコストを回避するための入力データのバッファリングの方式を提案する．

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 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．

ここでは，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て，入力データのバッファリングを行うことによって，この問題を回避する．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はしない．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．このことに注目して，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 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．

以下に入力バッファリングの手順の概要を説明する．前回のダンプ時までに，図 6 のようにデータ（薄い灰色）が格納されているとし，バッファにはログデータとして，前回ダンプ時から現在までの，ある一定期間の増分データが入っているとする．この例でのチャンクの次元サイズは 4 である．

(1) カラム値の挿入と，CVT における最終位置決定

まず，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 へカラム値を追加する．次に，C 社が現れた時にも CVT へ追加するが，この時は，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 Y 社と Z 社のカラム値が移動するが，データ要素そのものの RDT への挿入は行われない．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 に挿入するが，RDT には挿入しない．

(2) データの追加（挿入）

(1) により，挿入すべきデータレコードの CVT 上での最終格納位置が確定するので，続いて，データ値の追加を行う．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，データ値として RDT に追加する．図 7 が処理を終えた状態で，薄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)．濃灰色の要素が新たに加えられた要素となる．

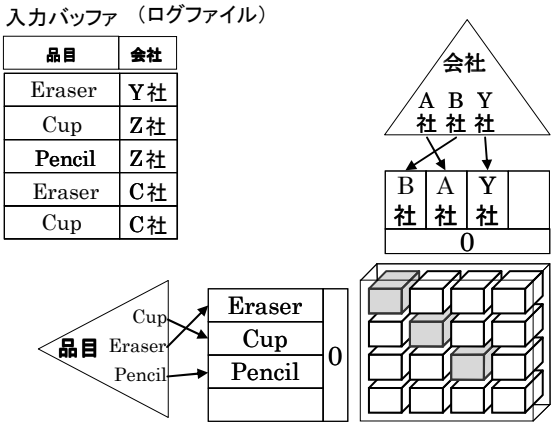

図 6 入力バッファリング （初期状態）
Fig.6 Input Buffering (Initial State)

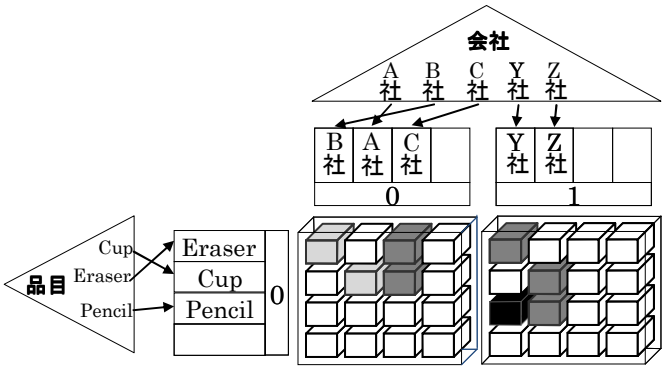

図 7 入力バッファリング （終了状態）
Fig.7 Input Buffering (Final State)

6. 入力バッファリング機構の実装

入力データは直前の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．データレコードの挿入処理は CVT への索引挿入と RDT へのデータ挿入の 2 段階に分かれる．各々の処理において，ログデータファイルをスキャンする．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．

6.1 索引挿入フェーズ

現在，索引挿入フェーズでは，ダンプ直前の RDT 内の既存データ要素は CVT のシーケンスセットノードの分割時に，逐次移動している．

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，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．新規カラムの挿入時には，CVT のシーケンスセット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．分割が発生した場合は，まず分割によって移動する要素を特定する．ここでは，バッファリングが開始される以前の既存要素については，逐次移動を行っているため，検索で要素を特定し次第，移動を行う．分割が発生しなかった場合は通常の挿入処理を行う．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分割が発生した場合は，次元が若い順に処理が行われる．

6.2 データレコード挿入フェーズ

索引挿入フェーズにおいて，ログデータファイルのレコード中の新たなカラム値に対応する最終的な位置(HOMD テーブルの添字)が各次元の CVT において確定する．ログデータファイルを再度，先頭レコードからスキャンする．各レコードを経歴オフセット法によりエンコードして対応する＜チャンク番号，オフセット＞対を求めて，ファクトデータを RDT の当該位置に格納する．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，集約値を更新して，ファクトデータとする．

7. 評価

以下では，実装したバッファ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．

7.1 カラム値のクラスタリングの効果

まず，近傍性確保の有無が範囲検索の速度にどの程度影響を与えるのか比較する．図 8 はすべて値の重複を許す 5 つのカラムからなるテーブルについて，チャンクの次元サイズ 300，カラム値の種類（カージナリティ）3,000 で，総レコード数を 50,000 から 300,000 まで変化させた時の範囲検索時間の推移を示している．連続 10 カラムの範囲検索では，その差は 2 倍程度に収まっているが，連続 100 カラムの範囲検索ではその差は 7 倍程度まで広がっている．これは，範囲検索を行う範囲が広ければ広いほど，近傍性を確保している場合に余分なチャンクへのアクセスが抑制され，ディスク読み取りの時間が削減されることを示している．この図より，近傍性の考慮が範囲検索の速度向上に有効なこと，ならびに，その有効性は検索範囲が広くなれば高まることがわかる．

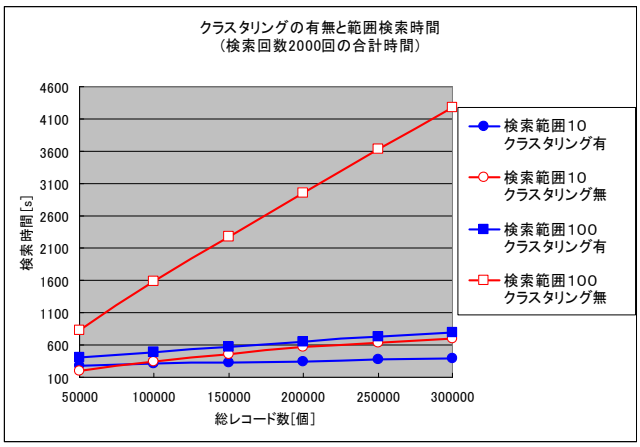

図 8 近傍性の有無と範囲検索時間
Fig.8 Proximity and Range Retrieval Time

7.2 入力バッファリングの効果

ここでは，提案した入力バッファリングの効果を測定・評価する．総レコード数 R のログデータファイルについて，バッファリングの回数を n 回にした場合，R/n 件のデータを一回のバッファリングで 6 節の手順によりダンプする．したがって，n を大きくとるほど，定期的ダンプの周期は短くなる．n=Rの時にはフロントエンドDBとの一貫性がほぼリアルタイムに維持されている状態である．ここでは次元数とチャンクの次元サイズは 7.1 節の場合と同じ条件で，総レコード数を 50 万件に固定して，カラムのカージナリティを各次元同一にしてカージナリティを変化させた時の挿入時間の推移を比較する．バッファリングの周期は総レコード数 50 万件を k レコードに分割して，一つの分割を 1 回のバッファリング単位としてログデータファイルからダンプする．

図 9 は，その結果であり，k の値が大きいほど，すなわちバッファリングの回数が少ないほどカージナリティの変化によらず，挿入時間は減少している．『索引挿入と，データ挿入を分離』したとしても，要素の繰り返し移動を防ぐことができるのは，同じダンプ周期における要素挿入に関してだけである．総レコード数 R が固定されている場合，バッファリング回数が多くなればなるほど，同一ダンプで挿入される要素の数が減少するがダンプ回数が増加する．このため，R 全体のダンプについてみれば，複数回移動しなければならない要素が多く発生し，バッファリング効果が弱まる．以上より，定期的ダンプの周期が長いほど，索引とデータの挿入を分離する本バッファリング機構は有効であることがわかる．

次に次元によりカージナリティが異なる，より一般的な入力データについての評価を行う．総レコード数，次元数およびチャンクの次元サイズはこれまでと同一の条件であるが，カージナリティを次元によりに変更して，バッファリングを行った場合の挿入時間の測定を行った．カラム値の標準のカージナリティを 3,000, 変更するカージナリティを x として，順に，(x, 3000, 3000, 3000, 3000),

(x, x, 3000, 3000, 3000),・・・, (x, x, x, x, x)

のように，次元によりカージナリティを変化させる．図 10 では，k=50,000，すなわち，バッファリング回数 10 回の結

果を示したが，xが 3,000 より小さい場合は 3,000 の次元が増えるほど挿入時間が増大し，3,000 より大きい場合は減少していることがわかる．図 10 のグラフの横軸の値 m は，カージナリティが 3,000 の次元が m 個で，それ以外の次元ではカージナリティが x であることを示しており，x を変化させている．以上の結果より，カージナリティの増大にともなって挿入時間が増大しているが，これは入力データのカラム値の種類が増加するにしたがって，チャンク分割が起こりやすくなり，それが挿入時間の増大をもたらしているためである．それに加えて，チャンクの分割が発生した時に，近傍性を保つように既に挿入されている二次記憶の要素を移動させる必要があるが，その時の要素特定にかかる時間が，カージナリティが増大するにつれ，大きくなることが挿入時間の増大をもたらしている．

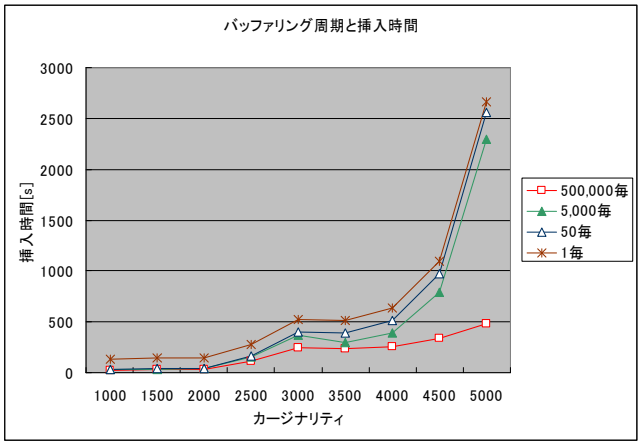

図 9 バッファリング回数の変化と挿入時間
Fig.9 Buffering Times and Insertion Time

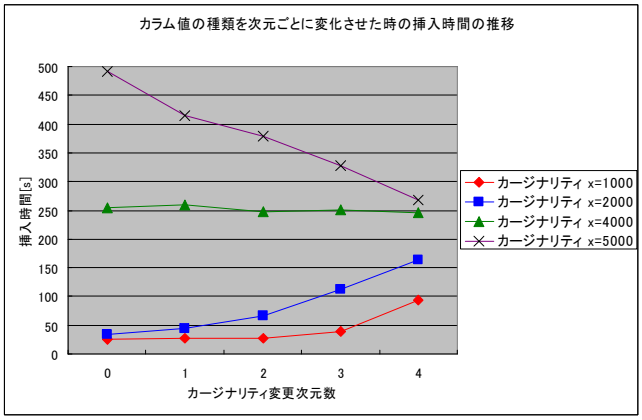

図 10 次元によりカージナリティが異なる時の挿入時間
Fig.10 Insertion Time for an Array of Different Dimension Cardinalities

8. まとめ

本論文では，実時間性が要求されないという特性を利用して，MOLAP の応答速度を向上させるための近傍性の確保を効率よく実施するためのバッファリング機構を我々が提案している多次元データの実装方式を用いて設計・実装し，近傍性の確保にかかる時間を削減した．ここでは，ログデータバッファから索引挿入するフェーズでは，必要に応じて，ダンプ前の既存データレコードはチャンク間で移動しており，時間コストが高くなっている．今後，このコストを削減する方式の開発が望まれる．

[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).

土田 隼之 Takayuki TSUCHIDA
福井大学大学院工学研究科博士前期課程在学中．2007 福井大学工学部情報・メディア工学科卒業．データウェアハウスの研究・開発に従事．日本データベース学会学生会員．

都司 達夫 Tatsuo TSUJI
福井大学大学院工学研究科教授．1978 大阪大学基礎工学研究科博士課程終了，工学博士．データベースに関する研究に従事．日本データベース学会正会員．

樋口 健 Ken HIGUCHI
福井大学大学院工学研究科准教授．1997 電気通信大学電気通信学研究科博士課程終了，工学博士．分散データベースの研究に従事．日本データベース学会正会員．