MOLAP 用多次元配列構築のためのバッファリング方式

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

‡福井大学工学部〒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テーブルのスナップショット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，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インクリメンタルに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構築する．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．

キーワード MOLAP バッファリング チャンク

A Buffering Scheme for Constructing Multidimensional Arrays for MOLAP

Takayuki TSUCHIDA, Tatsuo TSUJI, and Ken HIGUCHI

Graduate School of Engineering, University of Fukui Bunkyo 3–9–1, Fukui-city, Fukui, 910–8507 Japan

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

Abstract 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 employed multidimensional 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 multidimensional arrays.

Keyword MOLAP, Buffering, Chunk

1.まえがき

近年，業務用基幹データベースとは別に，意思決定をサポートするためにバックエンドで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]に必要な様々な要素を扱う必要があり，扱うデータ範囲が大変広い上に，処理の内容もその時々に応じて多岐にわたるため，処理が定型的な業務用データベースよりも応答速度は遅くなる．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．そのため，しばしば主要な作業である範囲検索の速度を向上させ，全体のパフォーマンスを底上げするために,「カラム値の距離が近傍である要素は，実データの 2 次記憶上の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』の考慮が行われている．MOLAP では一週間や一日に一度など，定期的にフロントエンドの業務用データベースの更新をバックエンドのデータベースに反映させるため，フロントエンドの更新を逐次反映させる必要は無い．本論文では，この特質を利用したデータバッファリング方式を提案することにより，近傍性を保証するための

コストの軽減を行った．

範囲検索を高速に行うには，データ反映の際にカラム値が近接している要素どうしは実データの物理配置においても近傍に配置するように考慮されることが望ましい．しばしば MOLAP で採用される配列全体を部分配列に分割するチャンク化では，多くの場合一つのチャンクが一つのディスクページに対応している．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を行うと，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 の持つ特性である『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』という点を利用して，『２次記憶への実データの挿入と索引データの挿入を分離』することにより高速に近傍性を保証するためのバッファリング機構の方式設計と実装，ならびに評価を行った．実装方式は我々が提案している HOMD(History Off implementation scheme for Multidimensional Datasets)[9][10]を用いた．この方式は，空間効率・検索速度共に優れている上に，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出来る．

2. HOMD の概要

まず，我々が提案している HOMD(History Offset implementation scheme for Multidimensional Datasets) と呼ばれる拡張可能配列[6][7][8]の概念を用いた多次元データの実装方式について説明する．以下では，多次元データとして特に関係テーブルを対象とする．

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 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．

2.2. 拡張可能配列による関係テーブル実装方式

HOMD は拡張可能配列の概念を用いている．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．拡張可能であるために，関係テーブルのレコードの動的な

Item	Color
Eraser	BLUE
Notebook	PINK
Fountain pen	CYAN
Loose-leaf	RED
Loose-leaf	CYAN
Eraser	RED

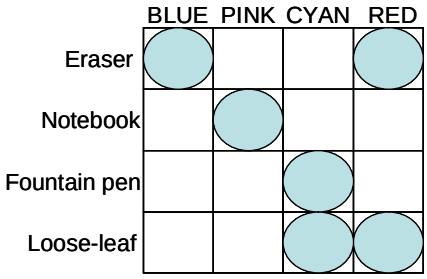

図 1 多次元配列を用いた関係テーブルの実装例

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することができる．

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，次元数 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[$s1$, $s2$, $s3$, $s4$] 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[$s1$, $s3$, $s4$] 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 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，上記の部分配列内の要素($i1$, $i3$, $i4$) のアドレスは 1 次関数 $s1s3i4 + s1i3 + i1$ で求められる．この($s1s3$, $s1$) 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(3,4) について，次のように行われる．まず，第 1 次元の添え字が 3 である部分配列の経歴値６と，第 2 次元の添え字が４である部分配列の経歴値８を比べ，６＜８であるので，要素(3,4)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(3,4)は部分配列内では要素(3)であるため，求めるアドレスは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することができる．

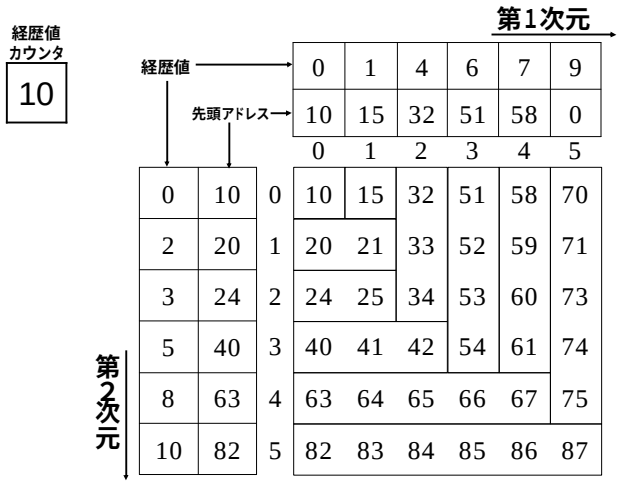

図 2 拡張可能配列の構造

2.3. 経歴・オフセット法と HOMD データ構造

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，配列領域全体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，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ことができる．例えば，図 2 の配列要素(3,4) の位置を経歴・オフセット法を用いて表現すると，要素(3,4) が含まれる部分配列の経歴値は８，要素(3,4)の部分配列内での位置は(3)であるため，経歴値が８，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．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，キーに対してファクトデータを必要としており，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を高速に行うため，関係テーブルの各カラムに CVT(key-subscript Con　Version 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，HOMD テーブルと呼ぶ．図 3 に HOMD データ構造の例を示す．
本研究においては，

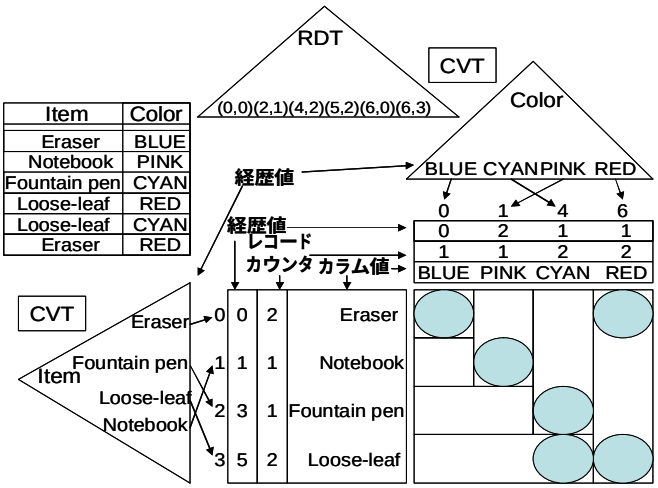

図 3 HOMD データ構造

3. チャンク単位で拡張を行う HOMD

ここでは，論理拡張可能配列として，ある一定サイズの部分配列である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．この配列では拡張された順番を示すチャンク番号をチャンク毎に付与する．また，拡張されていく部分配列に相当するチャンク部分配列毎に，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．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．以後，チャ

ンク単位で拡張を行う HOMD のことを C-HOMD［11］と呼ぶこととする．

3.3. C-HOMD の構造

図5 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，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-HOMD に唯一保持するだけでよい．また，HOMD ではカラム値ごとに保持していた部分配列の経歴値と係数ベクトルを，C-HOMD 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．

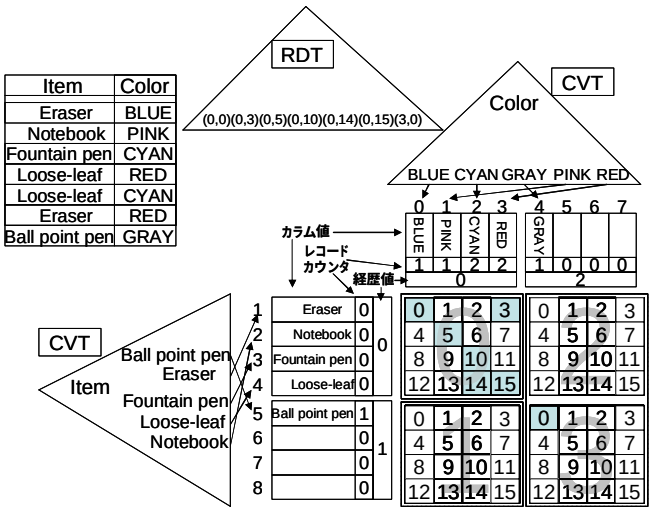

図 5 C-HOMD の構造

3.5. C-HOMD でのレコード検索

C-HOMD でのレコード検索は，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行われる．

3.5.1. 単一値指定検索

新規カラム値 A の挿入によって拡張されたチャンク部分配列のことをカラム値 A の基チャンク部分配列という (図５では，チャンク番号１のチャンク部分配列は Item である Ball point pen の基チャンク部分配列であるが，チャンク番号２・３のチャンク部分配列は基チャンク部分配列ではない) ．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 をルートノードから探索し，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，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，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．

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，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．図６に，検索対象が第 1 次元の添え字が５である場合の，論理拡張可能配列の検索範囲と RDT の探索範囲の例を示す．

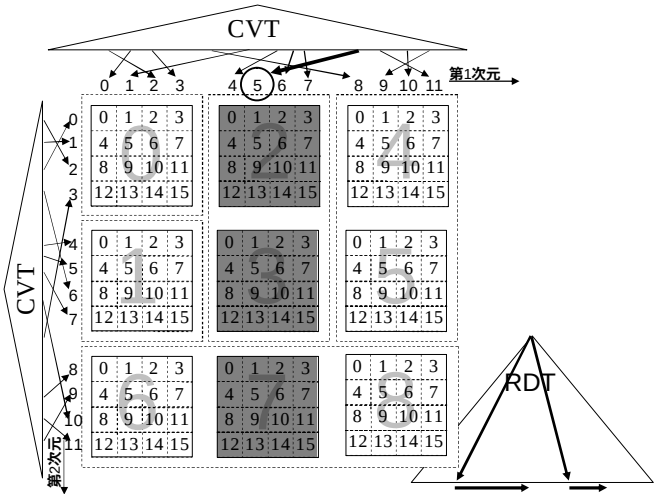

図６ 単一値指定検索の例

3.5.2 範囲検索

単一値指定検索において探索するべき基チャンク部分配列は一つしか存在し得ないが，範囲検索では範囲内のカラム値の数だけ基チャンク部分配列を調べなければならない可能性がある．また，それぞれの基チャンク部分配列を探索する際に,「その基チャンク部分配列に対応するカラム値」がどれであるか区別して，そのカラム値を持つ要素だけを検索結果に加えなければならない．つまり，カラム値 A の基チャンク部分配列ではカラム値 A について，カラム値 B と C の基チャンク部分配列ではカラム値 B と C についてチェックする必要がある．(※同一次元でのみ範囲検索を行っているのであれば，同一次元には重複するカラム値は存在できないため，全てのカラム値についてチェックしても問題は無い．) 他の変更点としては，それぞれの基チャンク部分配列は，検索次元から拡張されているため，単一値検索では検索対象の次元以外の次元に属するチャンク部分配列にだけ着目すればよかったが，範

囲検索では検索対象の次元にも探索するべき基チャンク部分配列が複数存在している場合があり，全ての次元に属するチャンク部分配列を要素が含まれているチャンク部分配列かどうか調べる必要がある．

4 カラム値の近傍性

HOMD の場合，新たなカラム値が入力される時間順に配列添字に時間順にマッピングされる．したがって，カラム値の順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１つのシーケンスセットのカラム値に対する添字集合が複数のチャンクにマッピングされる場合である．一つのチャンクは 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ために，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とが多いからである．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可能性がある．

例えば，図７は新たなカラム値の入力順に添字にマッピングされているが，カラム値 AAA, BBB, CCC の範囲検索を行ったとすると，４つ全てのチャンクを調べなければならないが，図８のカラム値の近傍性が考慮されている場合には，２つのチャンクを調べるだけで良い．

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非常に高い時間的コストを必要とする．すなわち，近傍性を確保するために，新たなカラム値の入力があれば，RDT 内の既存データを他のチャンクに移動する必要がある可能性が高いためである．本研究の目的は，いかにこの時間的コストを軽減するかということである．

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カラム値をキーとする索引の移動に従い，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移動させる．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８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 YYY である商品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，YYY

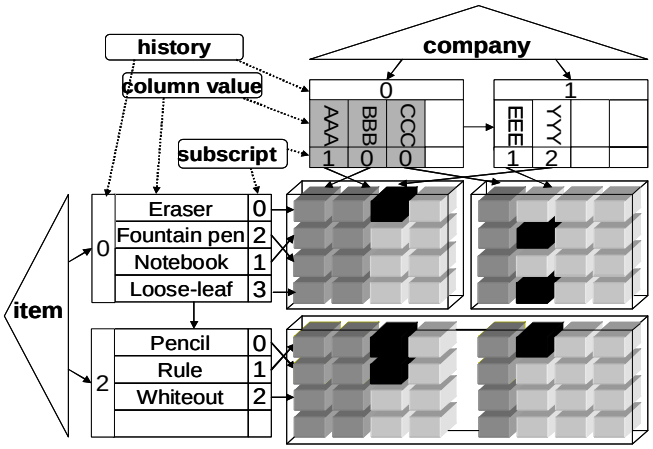

図７ 近傍性確保せず

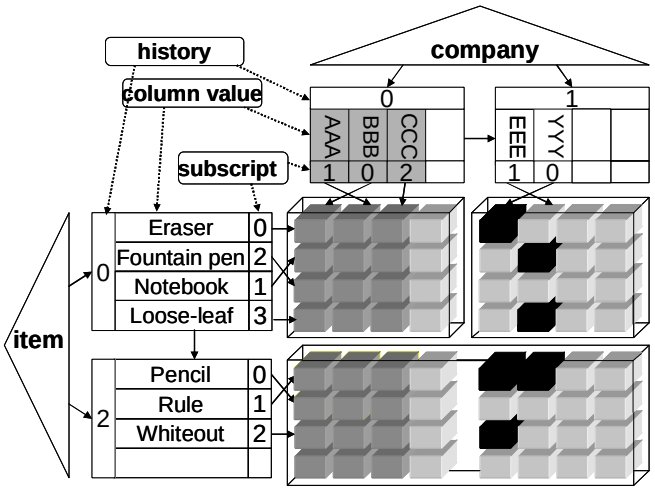

図８ 近傍性確保

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 YYY の商品データが，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 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 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 CVT 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２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

はデータ全体として一貫性がないが，ダンプ終了時には，索引とデータの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について説明する．前回のダンプ時までに，図 9 のようにデータが格納されているとし（黒の部分），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 ZZ)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，CC が現れた時，B+木へ追加するが，この時は，ノードが一杯になっているのでノードの分割が発生する．この時，新たなノードに YY と ZZ 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．データログ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，バッファを再度はじめからスキャンして，CVT をサーチして，添字を参照しながら，<経歴，オフセット>対に変換し，データ値として RDT に追加する．図 10 が処理を終えた状態で，黒の部分が最初から存在し移動していない要素．灰色の部分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(YY, pencil)．青色の部分が新たに加えられた要素となる．

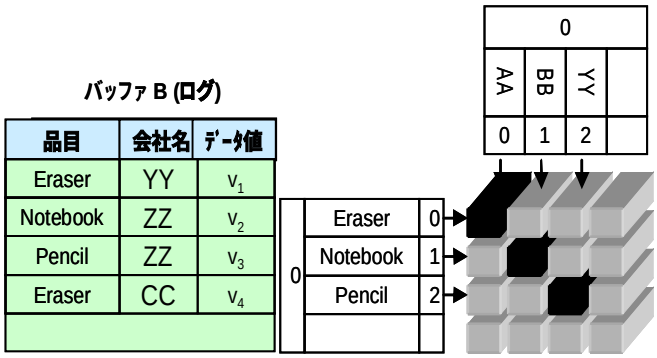

図９　入力バッファリング　（初期状態）

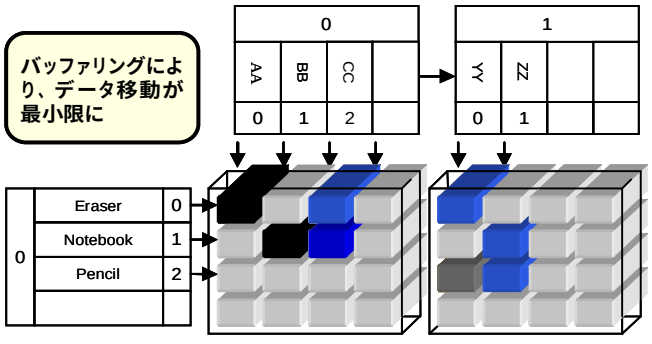

図１０　入力バッファリング　（終了状態）

６．入力バッファリング機構の実装

本節では５．で述べた入力バッファリング機構の実装について説明する．

入力データは直前の期間の追加データがすでにログファイルに格納されているものとして，この期間以前に HOMD データとして蓄積されている既存データに対して，ログファイルのレコードを追加して，最新の HOMD データを得ることとする．スクラッチから多次元配列データを構築する必要はなく，差分のみを集約するインクリメンタルダンプである．

CVT への索引挿入と RDT へのデータ挿入の 2 段階に処理は分かれる．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ず，挿入時間全体の１％程度である．

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．チャンク情報テーブルには，既に当該チャンクに挿入されているカラムの数が保持されており，その数を調べることにより，分割が発生するかどうかを判定することができる．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．各チャンク情報テーブルには，そのカラムの最大カラム値の要素が保存されているので，その要素をキーとして CVT にアクセスし，当該シーケンスセットノードを順次調べ，移動が必要なチャンクサイズの半分の個数の要素の配列添字を得る．この添字に従って，移動が必要な RDT 上の要素を特定する．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，逐次移動を行うため，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　データ要素挿入フェーズ

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位

置(論拡張可能チャンク配列の添字)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の実測値を比較評価する．実装環境は，Sun Fire E4900 (CPU：UltraSPARC IV，クロック速度：1050MHz，メモリ容量：48 GB，ディスク容量：RAID1 (73.4 GB×2) RAID5 (73.4 GB×12), OS：Solaris 9)を使用した．

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１１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，チャンクサイズ 300，カラム値の種類（カージナリティ）3000 で，総レコード数を 50000 から 300000 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てい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，その有効性は範囲検索する範囲が広くなればなるほど高まり，一定以上の条件では近傍性確保は不可欠であることがわかる．

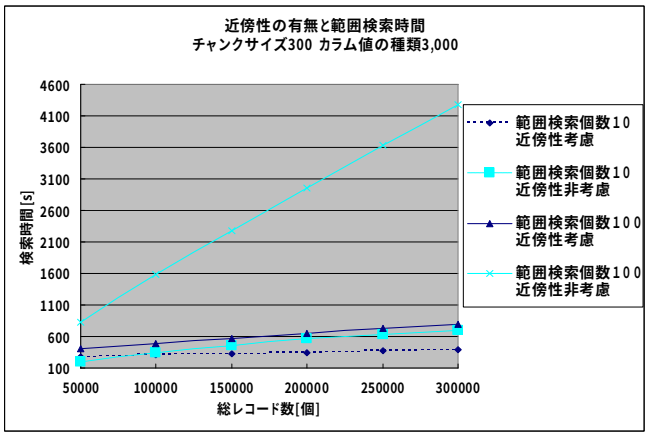

図１１ 近傍性の有無と範囲検索時間

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 の時にはリアルタイムダンプに相当する．ここでは次元数とチャンクサイズは 7.1 節の場合と同じ条件で，レコード数を 50 万件に固定して，バッファリングの回数を 10 回から 50 万回まで変化させた時の挿入時間の推移を比較する．図１２は，その結果であり，バッファリング回数１０回(B10)とバッファリング回数５０万回(B50 万回)ではおよそ 3 倍の差があることがわかる．これは，『索引への挿入と，データの挿入を分離』したとしても，余分な要素の移動を防ぐことが出来るのは，同じダンプにおける要素挿入に関してだけであるため，総レコード数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し，複数回移動しなければならない要素が多く発生するためである．また，要素移動を行う際にはカラム値情報を用いて移動する要素の経歴値とオフセットを特定しなければならないため，要素数が既に大量に蓄えられている状態で分割が発生すると，初期の要素数があまり２次記憶に存在していない状態での分割に比べて，要素の特定により多くの時間を必要とする．カーディナリティによっては分割が発生するタイミングが異なり，分割が発生する時期によっても挿入時間に変動が生じる．

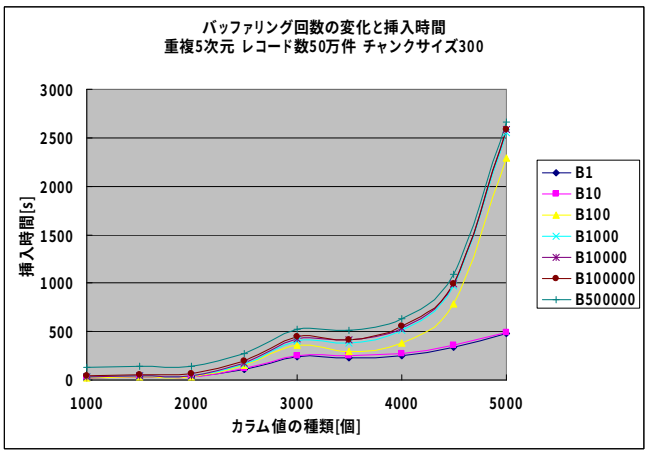

図１２ バッファリング回数の変化と挿入時間

今までは，カーディナリティが変化したとしても，「どの次元も同一のカーディナリティ」であるような性質を持つ入力データについて評価してきたが，次に次元ごとにカーディ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．

図１３では，次元数およびチャンクサイズはこれま

でと同一の条件であるが，カーディナリティを次元ごとに変更して近傍性を確保した場合の挿入時間の測定を行った．標準のカラム値を 3000，測定するカーディナリティを x として，順に，(x,3000,3000,3000,3000), (x,x,3000,3000,3000),・・・, (x,x,x,x,x)　のように次元毎のカーディナリティを変化させ測定を行った．バッファリング回数 10 回と，50 万回の結果を示したが，測定対象のカーディナリティが 3000 より小さい場合は，3000 の次元が増えるほど挿入時間が増大し，3000 より大きい場合は減少していることがわかる．カーディナリティが 5000 の場合に，挿入時間が特別増大しているのは，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り，その時の要素検索にかかる時間が，基チャンク部分配列だけの検索で済む場合と，基チャンク配列外も検索しなければならない場合とで大きく異なり，カーディナリティが増大するにつれ，基チャンク部分配列外の要素を検索する必要がある場合が加速度的に増加するからであると考えられる．チャンク分割で移動が発生する要素は，チャンクの後半部分であるため，近傍性が考慮されていればそれらの要素が必ず同じ基チャンク部分配列に属すことになり，高速化することが出来る．この測定を通して「次元ごとにカーディナリティが異なっても，バッファリング機構は有効である」ということがわかる．

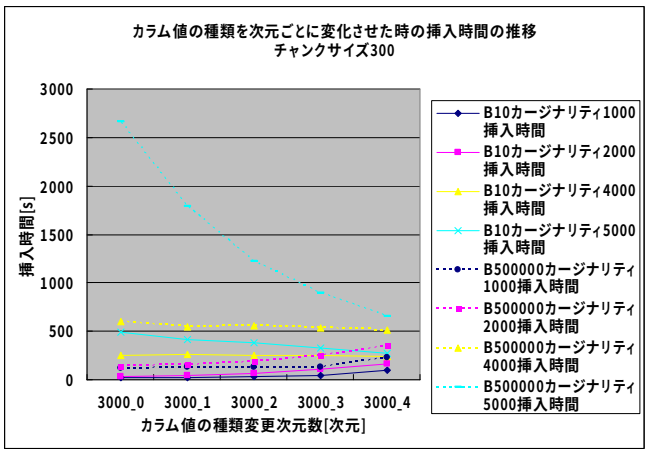

図１３ 次元ごとにカーディナリティを変化させた時の挿入時間

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．また，近傍性の確保が範囲検索にもたらす効力についても検証を行い，実際に範囲検索の時間短縮に大きな効果があることを示した．

文　　献

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

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

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

[4] Fredrik Bengtsson, Jingsen Chen, “Space-Efficient Range-Sum Queries in OLAP”, Proc. of DaWaK pp.87-96, 2004.

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

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

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

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

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

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

[11] 相羽，黒田，都司,樋口，”チャンク化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.