Vol.2012-DBS-155 No.19
2012/11/19

21. 次元拡張に対応したデータキューブの構築

茶木一興† 岡下慎太郎† 都司達夫† 樋口健†

データキューブは，分析対象のテーブルについて，その属性のすべての組合せに対して，ファクトデータを集計した結果を保持するデータ集合である．オンラインでの分析を可能とするために，データキューブは分析に先立って事前に構築される．通常，対象テーブルのスキーマは不変であることを前提としている．しかし，業務の拡大につれテーブルに新たな属性を追加する必要が生じた場合，対応するデータキューブの再構築には巨大なコストを伴う．本研究では，多次元データベースを使用してオンライ分析を行う MOLAP（Multidimensional OLAP）システムのためのデータキューブについて，属性の動的追加（次元拡張）に対して，再構築コストがほとんどかからないデータキューブを構築する方式を提案・実装し，評価する．このために，次元拡張に対して，高い適応性を有する経歴・パターン法によってタプルデータをエンコードする．

21. Construction of Datacubes Admitting Dynamic Dimension Extension

KAZUOKI CHANOKI† SHINTARO OKASHITA†
TATSUO TSUJI† KEN HIGUCHI†

For a relational table, a datacube can be used for its data analysis. It is a dataset generated by aggregating fact data for all the combinations of the table attributes. In order to enable online analysis, the related datacube is constructed prior to analyzing. Generally the table schema is assumed to be static.However, during the expansion of business, it often becomes necessary to add a new attribute to the table schema. In this case, the datacube corresponding to the new table should be reconstructed, which causes very large cost. This paper proposes a scheme of datacube construction in MOLAP environment, in which very low reconstruction cost is needed even if a new attribute is added and the dimension extension occurs. In order to realize the scheme, tuples in the analyzed table and the derived datacube are encoded using history-pattern encoding method we have proposed.

1. はじめに

高度情報化が進んだ近年，企業や組織において，保有している電子データを分析し“経営意志決定”や“経営戦略策定”に役立てることが盛んに行なわれている(ビジネス・インテリジェンス). この分析手法の一つとして OLAP(Online Analytical Processing)がよく利用されている．OLAP は，電子データを多次元的に集計し視覚化する分析手法であり，分析に用いるデータの形態により，多次元データベースを用いる Multi-dimensional OLAP (MOLAP)と関係データベースを用いる Relational OLAP (ROLAP)の二種類に大別される．

MOLAP では，例えば，一週間や一ヶ月などの一定期間ごとにフロントエンドの業務用関係データベースからバックエンドへデータをダンプし，多次元データベースへ集約する．多次元データの集計処理は非常に時間がかかる処理であるため，クライアントから分析に用いる集計データを要求される前に，テーブルの全ての属性の組合せに対して予め集計して，その集計結果を多次元データベースへ格納する．集約・集計によって多次元データベースへ格納したタプルをまとめて「データキューブ」と呼ぶ．クライアントからの集計データの要求には,データキューブを検索し，オンラインで集計結果を返す．

†福井大学
University of Fukui

一般的に，MOLAP で用いる多次元データベーステーブルは，各次元のカージナリティに応じた固定サイズの多次元配列で実現されるが，新しい属性値の割り当てによって配列サイズを超過した際，データキューブは再構築が必要となる．さらに，業務の拡大につれて，関係テーブルに新たな属性を動的に追加する（多次元データベーステーブルでは次元拡張に相当）ことが行われる．分析対象のテーブルに次元拡張が行われた場合も，データキューブを再構築する必要がある．分析対象のテーブルの次元が n 次元であり，属性の追加により n+1 次元に拡張された場合，集計結果の派生キューボイド(dependent cuboid)の数は 2^n 個から 2^{n+1} 個となり，この再構築は，高次元の場合に巨大なコストを必要とする．

本論文では，次元拡張に対して，再構築コストがほとんどかからないデータキューブの構築方法について提案・実装し，その結果について評価を行う．分析対象のテーブルとその派生キューボイドのタプルは我々が提案している経歴・パターン法でエンコードする．経歴パターン法は次元拡張に対して，有利な仕組みを提供する．

2. 経歴・パターン法

経歴・パターン法[1]は多次元データ集合のエンコーディング方式である．この手法は，エンコード対象のタプルを拡張可能配列[8]の一要素に対応させ，その座標に基づいて

エンコードを行う手法である．このとき，モデルとして用いる拡張可能配列の特質として，配列の次元サイズの超過や，次元拡張による動的な次元追加が行われた場合も，それまでにエンコードしたタプルを再エンコードする必要がないという利点を持つ．

2.1 タプルのエンコード

経歴・パターン法により，多次元データ集合のエンコードを行うには，まず，拡張可能配列の初期化処理を行う．続いて，タプル一件ずつに対し，添字タプルへのエンコード，およびパターンへのエンコードを行う．以後，エンコード対象の多次元データの次元を n とする．

まず，初期化処理では，各次元サイズが 1 の n 次元配列を用意する．ただし，配列要素の実体は確保せず，n 個の属性値・添字テーブルを用意する．そして，この多次元配列に経歴値として，0 を割り当て，さらにこの経歴値 0 に n 個の要素 0 を持つ境界ベクトル<0, …, 0>を割り当て，境界ベクトルテーブルへ記憶する．さらに，各次元に対して経歴値テーブルと呼ぶ 1 次元配列を用意し，各配列の先頭に経歴値 0 を記憶する．属性値・添字テーブル，経歴値，境界ベクトルテーブル，経歴値テーブルについては，後述する．なお，以後，多次元配列と記すものは全て本節で説明する拡張可能配列を指すものとする．

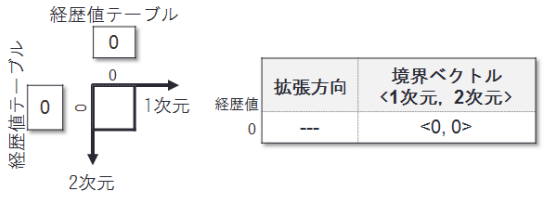


図 2.1. 経歴・パターン法の多次元空間(初期化後)

次に，添字タプルへのエンコード処理では，対象タプルの次元と多次元配列の次元を対応付け，各次元の属性値を対応した次元の添字へ変換することで，属性値タプルを添字タプルへ変換する．対応付けは，各次元に用意した属性値・添字テーブルへ<属性値，添字>対で記憶する．この操作により，タプルは多次元配列の一要素に対応する添字タプルで表される．新たな属性値の割り当てにより，当該次元の添字が配列の現次元サイズを超過した場合，その時点の多次元配列と同形の部分配列を用意し，超過した次元方向に付与することで，超過した次元のサイズを倍に拡張する．ここで付与した部分配列には，拡張の順番を示す経歴値 h を割り当て，当該次元の経歴値テーブルの末尾にこの h を記憶する．また，拡張後の多次元配列における次元 k のサイズを s_k とすると，経歴値 h に対し，拡張方向の次元と境界ベクトル<$b(s_1)$, …, $b(s_n)$>を割り当て，境界ベクトルテーブルの h の位置に記憶する．ここで，$b(k)$を 0 以上の整数 k を表すために必要な最小のビット数とする．境界ベクトルは，後述のパターンへのエンコードで使用する．

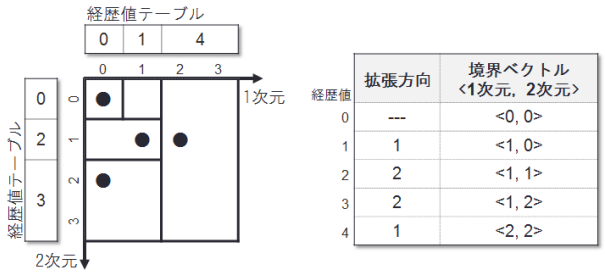


図 2.2. 経歴・パターン法の多次元空間

最後に，パターンへのエンコード処理では，得られた添字タプルを，そのタプルが属する部分配列の経歴値と，経歴値に対応する境界ベクトルを用いてエンコードする．これには，まず添字タプルの属する部分配列の経歴値を求める．添字タプルを$(i_1, i_2, \ldots, i_n)$，$h(k, p)$を次元 k $(1 \leqq k \leqq n)$ に対応した経歴値テーブルの p 番目の経歴値とすると，次元 k について $h(k, b(i_k))$の経歴値を調べ，一番大きい経歴値 $h_{\max}$ に対応する部分配列が添字タプル$(i_1, i_2, \ldots, i_n)$の属する部分配列である．

次に，$h_{\max}$ に対応した境界ベクトルを元に，添字タプルをパターンへエンコードする．この処理では，添字タプルにおける各次元の添字を，境界ベクトルの各次元で示されたビット数で表し，それらを連接して，パターンを作成する．例として，次元数を 3，添字タプルを(i_1, i_2, i_3)，境界ベクトルを<b_1, b_2, b_3>とすると，パターンは左シフト(<<)と論理和(+)のレジスタ命令のみを使って以下のように得られる．

パターン $= i_1$
パターン $= i_2 +$ (パターン$<<b_2$)
パターン $= i_3 +$ (パターン$<<b_3$)

こうして得られた<経歴値，パターン>対が，経歴・パターン法によるタプル一件のエンコード結果である．

2.2 次元拡張

次元数 n の多次元配列 A_n が次元拡張により次元数 n+1 の多次元配列 A_{n+1} へ動的に拡張された場合，A_n は A_{n+1} における次元 n+1 の添字 0 に対応した部分領域として扱われる．新たに得られた多次元配列 A_{n+1} の次元 n+1 のサイズは 1，それ以外のサイズは A_n と同一である．この性質から，過去に次元数 n 以下としてエンコードした全タプルも次元数 n+1 として扱う．

実際に，次元数 n の多次元配列 A_n に対して次元拡張を行った場合，まず，次元 n+1 用の属性値・添字テーブル，および経歴値テーブルを用意する．経歴値テーブルの先頭には，初期拡張同様に経歴値 0 を記憶する．さらに，既存の

全境界ベクトルの n+1 次元目に対してビット数 0 を追加する．このように境界ベクトルを修正することで，過去に n 次元以下としてエンコードした全タプルのパターン末尾に“次元 n+1 の添字 0 が 0bit で連結されている“として扱うことができる．ここで，次元数 n 以下の時点でエンコードした全タプルの n+1 次元目の添字を 0 とする．このように，過去にエンコードしたタプルは，境界ベクトルの修正のみで次元 n+1 の添字を 0 とすることにより，次元拡張による再エンコードの必要は無い．

次元拡張の例として，次元数 2，最大経歴値 4 の多次元配列に対し，次元拡張により次元数 3 へ拡張した場合について図 2.3 に示す．左図(a)が次元拡張前の多次元配列 A_n，右図(b)が次元拡張後の多次元配列 A_{n+1} を示す．

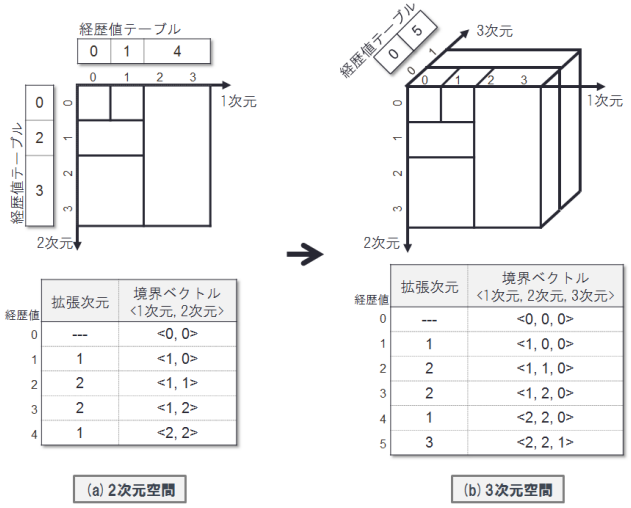


図 2.3. 次元拡張前後の多次元空間

3. MOLAP におけるデータキューブ構築

MOLAP では，フロントエンドの n 次元関係データベーステーブルより例えば一ヶ月ごとなどの定期的なタイミングでバックエンドの多次元データベースへタプル集合がダンプされる(図 3.1)．このタプル集合のうち，単一の次元を集計対象とし，この次元の属性値をファクトデータと呼ぶ．データキューブ構築における最初の処理では，得られたタプル集合のファクトデータを集約し，データキューブ構築の元となる n 次元テーブルを構築する．この n 次元テーブルを base cuboid と呼び，base cuboid 内のタプル一件を base cuboid cell と呼ぶ．最後に，得られた base cuboid の内，ファクトデータを除いた各次元についての組み合わせを集計条件とし，集計条件一つに対し一つテーブルを構築する．このテーブルを dependent cuboid と呼び，dependent cuboid 内のタプル一件を dependent cuboid cell と呼ぶ．ただし，集計条件として，全ての次元の組み合わせは base cuboid として区別するために，dependent cuboid は 2^n-1 個存在する．こうして得られた base cuboid および dependent cuboid がデータキューブである．なお，データキューブに含まれるタプルを総称して cell と呼ぶ．

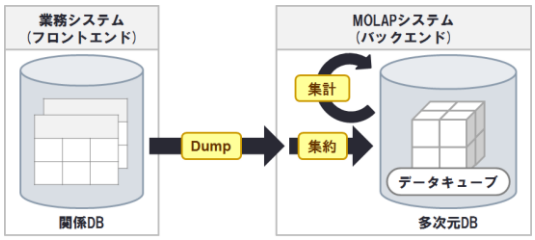


図 3.1. データキューブ構築

図 3.2 は base cuboid として，“店”，“商品”，の 2 次元データにファクトデータとして“売上”が集約されている．dependent cuboid は「店」「商品」「φ」の 2^2-1 = 3 個である．φ は店も商品も含まない条件であり，全 base cuboid cell を対象とする集計結果である．

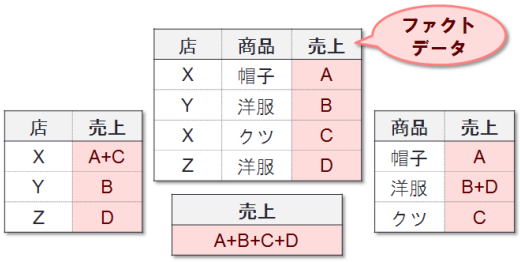


図 3.2.データキューブ内の各種テーブル例

フロントエンドの関係テーブルは，常に最新のタプルデータのみを保持しているが，MOLAP において，分析対象となるデータは過去に存在していたタプルデータもすべて含んでおり，当該データキューブはそれらの集計結果をすべて含む必要がある．

4. 次元拡張に対応したデータキューブ構築

本研究では，フロントエンドの関係データベーステーブルよりダンプされた多次元データ集合を経歴・パターン法によりエンコードした後，データキューブを構築する(図 4.1)．データキューブの全 cell がエンコードされるが，ファクトデータの次元はエンコード対象とはせずに，4.1.節で述べる B+木のデータ部に格納する．経歴・パターン法の利点により，多次元配列の次元サイズを超過しても再構築が必要ないデータキューブが構築される．

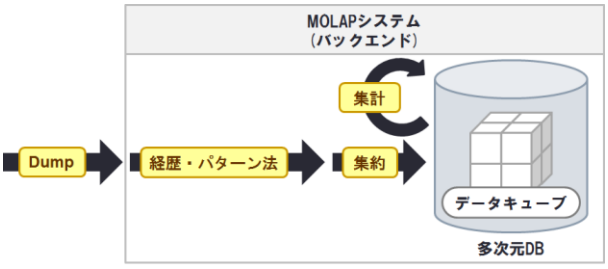


図 4.1.本研究における MOLAP システム

4.1 格納構造

前述のとおり，データキューブの全 cell は経歴・パターン法によりエンコードされるため，cell を表すデータは<経歴値，パターン>となる．この対を二次記憶上の B+木により記憶する．この B+木を RDT(Real Data Tree)と呼ぶ．B+木のキーは，経歴値を 1Byte，パターンを 8Byte として，経歴値，パターンの順にビットレベルで連接した 9Byte，データはファクトデータである．データ部のファクトデータは dependent cuboid の構築時に集計により更新される．

RDT のキーは経歴値を最上位バイト，つづいて，パターンを次元の昇順に上位バイトから連接しているため，キーは，経歴値によりグルーピングされる．このため，RDT のシーケンスセットを逐次アクセスすることで，経歴値の昇順にパターンを得ることが可能である．

4.2 単一配列によるデータキューブの構成

データキューブは，base cuboid と dependent cuboid の二種類の cuboid から構成される．これらの cuboid はすべて，Single Array Datacube Schema[7]という方法で単一の多次元配列へ割り当てる．これにより，2^n 個の cuboid をすべて単一の配列で管理することが可能になる．この方法では，多次元配列の次元添字 0 の領域へ当該次元の dependent cuboid を割り当てる．base cuboid はすべての次元添字が 0 でない領域に割り当てる(図 4.2). したがって, dependent cuboid cell はその添字タプルに少なくとも一つの 0 を含む．ここで，i_k を次元 k に対する添字タプルとすると, 0 を含まない添字タプル(i_1, i_2, i_3)の cell 一件に対し，集計先は以下 2^3-1=7 件の添字タプルとなる．

(i_1, i_2, 0)，(i_1, 0, i_3)，(i_1, 0, 0)，(0, i_2, i_3)，
(0, i_2, 0)，(0, 0, i_3)，(0, 0, 0)

なお，この方法を実現するため，経歴・パターン法によるエンコード時，各次元の添字 0 に対応した領域を dependent cuboid の格納領域として，予めリザーブしておく．

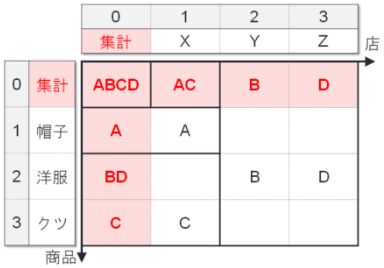


図 4.2.Single Array Datacube Schema

4.3 集計(dependent cuboid の構築)

集計処理には，Subarray Based Method[7]を用いる．この手法は，経歴・パターン法によりエンコードされた base cuboid について，Single Array Datacube Schema に基づく各次元添字 0 の領域へ dependent cuboid を構築する方法である．この手法はアルゴリズムの一部として，集計の中間結果を保持するためのバッファリングを行う．バッファにはメモリ上の B+木で実装される IRT(Intermediate Result Tree)を使用する. IRT のキーおよびデータは RDT と同様である．Subarray Based Method による dependent cuboid 構築(base cuboid 集計)の手順を以下に示し，そのフローチャートを図 4.3 に示す．

(1) 最大経歴値 H_{max} を求める．
(2) 経歴値 H_{max} から経歴値 0 まで以下(3)～(7)を繰り返す．処理中の経歴値を H_i とする．
(3) IRT 内の経歴値 H_i を持つ全 cell を経歴値の拡張方向に対し集計し，得られた dependent cuboid cell を IRT へ記憶する．
(4) RDT 内の経歴値 H_i を持つ全 cell を経歴値の拡張方向に対し集計し，得られた dependent cuboid cell を IRT へ記憶する．
(5) IRT 内の経歴値 H_i を持つ全 cell を経歴値の拡張方向以外に対し集計し，得られた dependent cuboid cell を IRT へ記憶する．
(6) RDT 内の経歴値 H_i を持つ全 cell を経歴値の拡張方向以外に対し集計し，得られた dependent cuboid cell を IRT へ記憶する．
(7) IRT 内の経歴値 H_i を持つ全 cell を RDT へフラッシュする．

なお，IRT へ記憶する dependent cuboid cell は，経歴・パターン法によりエンコード済みの base cuboid cell <経歴値，パターン>を添字タプルレベルまでデコードし，その添字タプルを元に集計先となる dependent cuboid cell の添字タプルを求め，得られた添字タプルをエンコードした結果である．例として，経歴値 H_i を持つ base cuboid cell をデコードして，得られた 0 を含まない添字タプルを(i_1, i_2, i_3)，境界ベクトルテーブルの経歴値 H_i に対応した要素が示す拡張次元を 3 次元方向とすると，添字タプル(i_1, i_2, i_3)の拡張方向で示された次元の添字を 0 に変更して得られた添字タプル(i_1, i_2, 0)が拡張方向への集計先である．また，拡張方向以外は対する集計先は, (i_1, 0, i_3), (0, i_2, i_3), (0, 0, i_3)である．

IRT の RDT へのフラッシュは，経歴値 H_i に対する集計が完了した時点で, 経歴値 H_i を持つ全 dependent cuboid cell に対して行なう．このフラッシュタイミングは，本システムのデータキューブにおける次の性質を利用している．

『 経歴値 H_i を持つ base cuboid cell に対するどの集計先 dependent cuboid cell も必ず H_i 以下の経歴値を持つ 』

この性質は，経歴・パターン法における最大経歴値 H_i 時点の多次元配列を考慮することで得られる．dependent cuboid cell は Single Array Datacube Schema により，各次元の添字 0 へ割り当てるため，現時点の多次元配列に属し得る全 base cuboid cell に対するどの集計先 dependent cuboid

cell も，配列サイズ拡張を起こさずに割り当て可能である．よって，それら dependent cuboid cell はデータキューブの最大経歴値 H_i 以下の経歴値を持つこととなり，上記の性質が得られる．この性質から，経歴値 H_i を持つ base cuboid cell を全て集計し終えた時点で，経歴値 H_i を持つ全 dependent cuboid cell の集計が完了したことが分かるため，先に示したタイミングでフラッシュを行なっている．このようなフラッシュにより，集計済セルをできるだけ早いタイミングで IRT から消去し，集計にかかる作業メモリ量を可能な限り小さく保っている．

前述のとおり，本システムのデータキューブは RDT に格納されており，キーは経歴値によりグルーピングされている．よって，Subarray Based Method によるアルゴリズムは B+木のシーケンスセットを最後尾から先頭まで逐次読み込むことで，集計に必要な全 cell の読み出しを迅速に行うことが可能である．

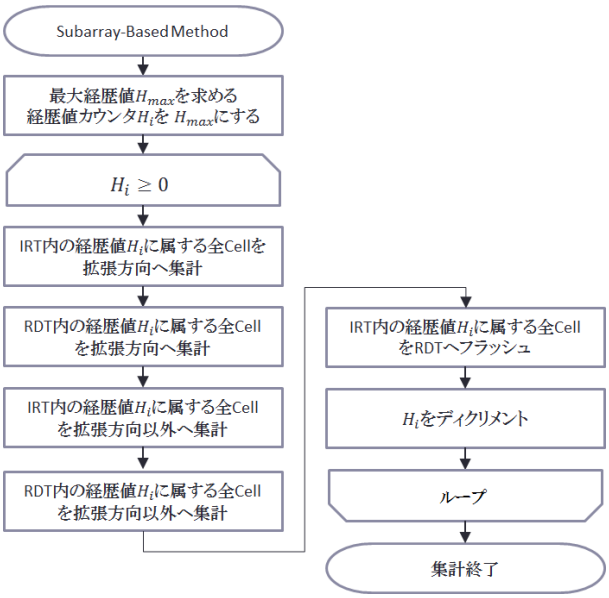


図 4.3. Subarray Based Method フローチャート

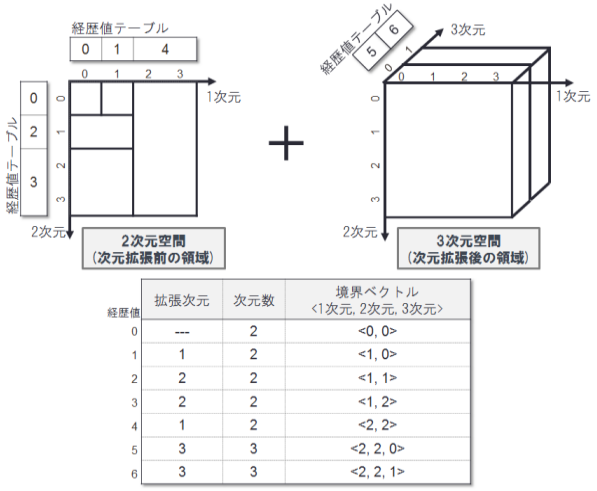
4.4 次元拡張を許容するデータキューブ

ここでは，業務内容の拡大により，フロントエンドの関係テーブルに新たな属性が追加された際，それに対応するデータキューブの次元拡張を行う方法について提案する．

前述のとおり，本システムにおけるデータキューブは，Single Array Datacube による構成であり，経歴・パターン法を用いてエンコードされている．ここで，データキューブを次元拡張するため，経歴・パターン法において，対応する次元拡張を行う．次元数 n の多次元配列 A_n は次元数 n+1 の多次元配列 A_{n+1} へ拡張され，A_n は A_{n+1} における次元 n+1 の添字 0 に対応した部分領域として扱われる．しかし，Single Array Datacube Schema では，各次元の添字 0 へ dependent cuboid cell を割り当てるため，次元拡張により追加された次元の添字 0 の利用について，競合が発生してしまう．

この問題を解決するため，次元拡張時に，A_{n+1} を A_n とは別に新たに確保することとする．A_{n+1} は，n+1 次元目のサイズを 1，それ以外の次元サイズを A_n と同一とする．この時点で確保した A_{n+1} の領域は，次元 n+1 における添字 0 に対応するため，全て集計添字である．このようにすることで，次元 n+1 に対する集計添字 0 の領域と，次元拡張前の多次元配列 A_n を共存させることができる．

上記の方法を実現するため，2.2 節で述べた経歴・パターン法における次元拡張の仕様を変更する．新たな次元拡張を，次元数 n，最大経歴値 H_{max} の多次元配列 A_n に対し行うと，まず，新たに次元 n+1 に対応した属性値・添字テーブルおよび経歴値テーブルを確保することまでは同様に行うが，経歴値テーブルの先頭には経歴値 H_{max}+1 を記憶する．続いて，経歴値 H_{max} に対応した n 次元の境界ベクトルの次元 n+1 の値を 0 としたものを経歴値 H_{max}+1 に割り当てる．経歴値 H_{max} 以下に属する部分配列が多次元配列 A_n，経歴値 H_{max}+1 以上に属する部分配列が多次元配列 A_{n+1} となる．



経歴値	拡張次元	次元数	境界ベクトル <1次元, 2次元, 3次元>
0	---	2	<0, 0>
1	1	2	<1, 0>
2	2	2	<1, 1>
3	2	2	<1, 2>
4	1	2	<2, 2>
5	3	3	<2, 2, 0>
6	3	3	<2, 2, 1>

図 4.4. 次元拡張前後の多次元空間

このとき，新たに確保した多次元配列 A_{n+1} とそれ以前に既に用意されている次元 n 以下多次元配列 Ai について，それらの属性値・添字テーブルや境界ベクトルテーブルは，全ての多次元配列で共有される．また，通常の次元拡張では，既存の境界ベクトルを全て修正することで，過去にエンコードしたタプルも次元数 n+1 であるものとして扱った．一方，新たな次元拡張では，既存の境界ベクトルを修正せずに，過去にエンコードしたタプルは，エンコード時点の次元数として扱う．また，このような仕様としたため，デコード時は，対象タプルの次元数を求めなければならない．そこで，境界ベクトルテーブルの新しい項目として，次元数を設け，デコード時に参照することでデコードを行う(図 4.4).

4.5 次元拡張後の集計

前節で本システムにおける次元拡張について述べた．しかし，4.3 節の集計アルゴリズム Subarray Based Method がこの次元拡張に対応していないという問題が存在する．次元数 n の多次元配列 A_n に対し，一度次元拡張を行い次元数 n+1 の多次元配列 A_{n+1} を追加したデータキューブに対して，Subarray Based Method による集計では，経歴値の大きい部分配列に属する base cuboid cell を，経歴値が同等以下の部分配列に属する dependent cuboid cell へ集計することを行うが，A_n は A_{n+1} よりも小さい経歴値に属するため，A_n の cell を次元 n+1 の集計添字 0 に対する集計が行えず集計漏れが発生してしまう．この問題を解決するため，通常の Subarray Based Method による集計後に“リフレクト処理”と呼ぶ追加集計を $A_i(i=1,\ldots,n)$について行い，A_i に再帰的に反映する．リフレクト処理の手順を以下のリストに，フローチャートを図 4.5 に示す．

次元拡張により最初に追加された次元を D_{ex0}(次元拡張が発生していない場合は未設定)，リフレクト済み最大次元を n_{ref}(過去にリフレクトが行われていない場合は未設定)，次元 N が拡張された時点の最大経歴値を H_{exn}，とする．

(1) n_{ref} が設定されている場合， n を n_{ref} に設定する．n_{ref} が未設定，かつ n_{ex0} が設定されている場合，n を n_{ex0} に設定する．どちらでもない場合，リフレクト処理を終了する．n から現最大次元まで(2)を繰り返す．現在処理中の次元を d とする．

(2) 経歴値 0 から H_{exd} に属する全 cell を d 方向へ集計する．

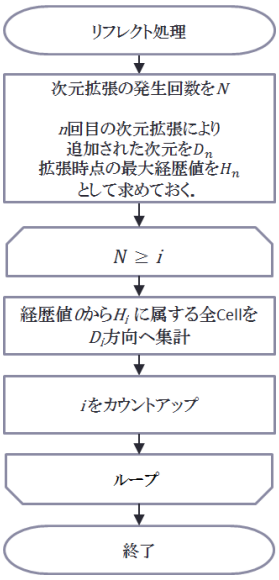


図 4.5. リフレクト処理のフローチャート

5. 差分構築

通常のデータキューブ構築は，ダンプの度にデータキューブの全 base cuboid cell を集計し直す必要がある．しかし，データキューブは巨大なデータ集合であるため，再集計処理には膨大なコストが伴ってしまう．そこで，差分構築を行うことで集計コストの軽減を行う．

差分構築では，データキューブを“デルタデータキューブ(以下デルタ)”と“オリジナルデータキューブ(以下オリジナル)”の二つに分割し，当該期間分の cell 集合についてデルタを構築する段階（プロパゲート）と，得られたデルタによりオリジナルを更新する段階（リフレッシュ）により最新のデータキューブを構築する．このような差分処理により，集計対象がデルタに構築された差分 cell のみと限定され，再集計コストが削減できる．

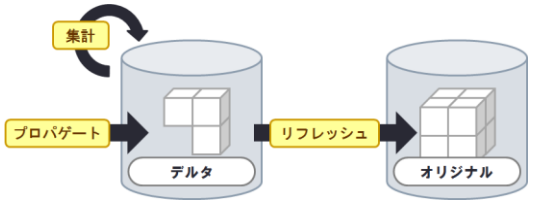


図 5.1.差分構築

5.1 プロパゲート処理とリフレッシュ処理

プロパゲート処理では，データキューブ構築における集約処理に対応する．よって，フロントエンドよりダンプされたデータ集合を経歴・パターン法によりエンコードを行った後，デルタへ集約する．集計処理は行わないため，デルタは base cuboid のみを保持する．

リフレッシュ処理では，デルタに構築された base cuboid をオリジナルの dependent cuboid に集計し，オリジナルの base cuboid および dependent cuboid を更新する．その後デルタを空にする．

5.2 次元拡張後の差分構築

差分構築では，オリジナルに対する処理はリフレッシュ処理のみであり，その処理は，前回集計からの差分 cell，および集計結果をオリジナルへ加算することである．オリジナルに格納された cell は二度と集計しないことから集計処理の軽減が可能である．しかし，次元数 n の多次元配列 A_n を次元拡張し，次元数 n+1 の多次元配列 A_{n+1} へ拡張した場合，A_n に属する全ての cell を A_{n+1} へリフレクトする必要がある．すなわち，次元拡張発生後のリフレッシュ処理後に，リフレクト処理を行う必要がある．なお，次元拡張により，次元数 n+1 へ拡張した後は，次元数 n 以下の cell が新たに集約されることは無い．よって，一度リフレクト処理を行うと，オリジナルは当該次元についてリフレクト済みとなるため，再度次元拡張が発生するまで，リフレクト処理を行う必要は無い．すなわち，リフレクト処理は，次元拡張後，最初の集計後に一度だけ行えば良い．

6. 関連研究

多次元データベースのための多次元配列に関する研究は数多く行われている（例えば 1)）．また，2) や 3) は拡張可能配列に関する最近の研究であるが，いずれも拡張可能配列のモデルが経歴・パターン法の場合とは，異なっており，エンコードの方式と特徴も異なる．本研究における，経歴・パターン法は，5) 6)で提案されている．

一方，データキューブの実装方式も数多く提案されている．その多くは ROLAP 向けの方式である．5) は base table のスキーマ変更を許したときの ROLAP データキューブのメンテナンスについて，述べている．また，6) は ROLAP データキューブの差分構築方式に関する研究である．

筆者等の知る限り，MOLAP のための多次元配列で実装されるデータベースの環境において，本論文におけるような次元拡張を許容するようなデータキューブを扱った研究は，少ないと考えられる．7)では MOLAP データキューブの差分構築方式について提案しているが，base table はスキーマ進化を認めていない．

7. 実験と考察

構築した MOLAP システムについて三つの計測実験を行った．実験環境として表 7.1 の環境を用いた．

表 7.1. 実験環境

OS	Red Hat Enterprise Linux5.6
CPU	Intel(R) Xeon CPU X7560@2.27GHz
Memory	128GB

7.1 実験 1

表 7.1 の条件について，集約・集計，次元拡張の実行時間を計測した．その結果を図 7.1 および図 7.2 に示す．なお，全ての結果は，同様の操作を 5 回行った平均である．

図 7.1~図 7.2 より，集約よりも集計に多大な処理時間がかかっていることが分かった．これは，MOLAP における集計が全集計条件に対して行われるため，base cuboid cell に比べ dependent cuboid cell 数が膨大となることから理解できる．また，次元拡張に要する時間はほぼ 0 であることが分かった．これは，次元拡張処理からデータキューブを再構築する必要が無くなり，新しい次元に対するメタ情報を用意するだけとなったため，処理時間が掛からなかったと考えられる．

表 7.1. 計測 1 の条件

タプル数	70000 件
各次元のデータ型	4Byte 整数
各次元カージナリティ	500

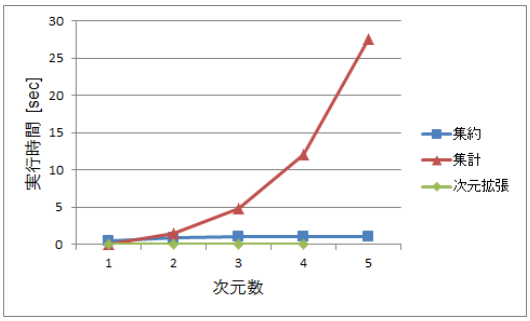


図 7.1.実験 1 の結果(1)

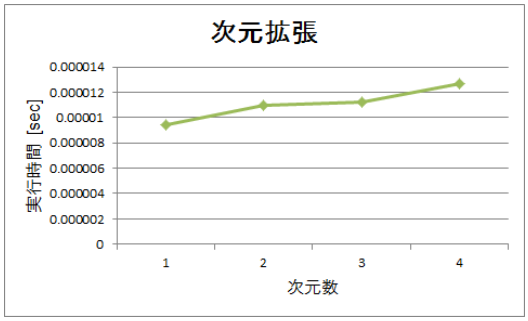


図 7.2. 実験 1 の結果(2)

7.2 実験 2

初期 1 次元から次元拡張により 5 次元へ拡張したデータキューブと，初期 5 次元のデータキューブについて，同規模の cell を集約・集計した．処理の内容を以下リスト 1 およびリスト 2 に，集約タプルの情報を表 7.2 および表 7.3 に，結果を図 7.3 に示す．なお，全ての結果は，同様の操作を 5 回行った平均である．

[リスト 1: 初期 1 次元のデータキューブ]

(1) 1 次元で初期化する．
(2) 次元 1~4 について，(3)~(4)を繰り返し行う．現在処理中の次元を *n* とする．終了したら(5)以降の処理を行う．
(3) 全次元を表 7.2 の条件とした *n* 次元のタプルを 70000 件集約する．
(4) 次元拡張で *n*+1 次元へ拡張する．
(5) 全次元を表 7.2 の条件とした *n* 次元のタプルを 70000 件集約する．
(6) 集計を行う(次元拡張を行なっているため，リフレクトが発生する)．

[リスト 2: 初期 5 次元のデータキューブ]

(1) 5 次元で初期化する．
(2) 次元 1~4 について，(3)を繰り返し行う．現在処理中の次元を *n* とする．終了したら(4)以降の処理を行う．
(3) *n* 個の次元を表 7.2 の条件，残りの次元を表 7.3 の条件としたタプルを 70000 件集約する．
(4) 全次元を表 7.2 の条件とした 5 次元のタプルを 70000 件集約する．

(5) 集計を行う(リフレクト発生無し).
リスト 1 およびリスト 2 の結果を比較すると，リフレクト処理を含むリスト 1 の処理に遅延が無いことが分かった．これは，リフレクト処理は本来行われるべき集計処理であるため, 構築時間がほぼ同等となると考えられる．

表 7.2. 実験 2 の次元条件(1)

データ型	4Byte 整数
カージナリティ	500

表 7.3. 実験 2 の次元条件(2)

データ型	4Byte 整数
カージナリティ	1

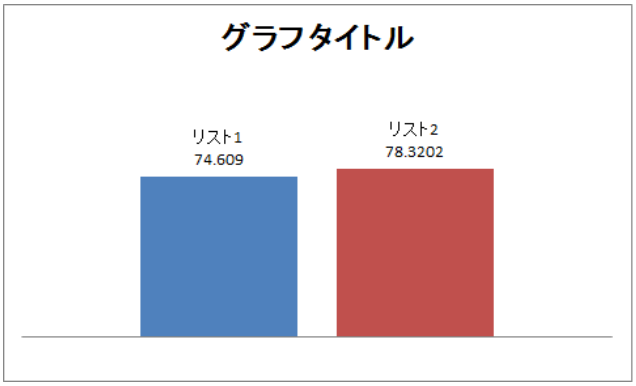


図 7.3. 実験 2 の結果

7.3 実験 3

同規模で構築済みのデータキューブ二つに対し，片方を差分構築有り，もう一方を差分構築無しとして，同規模のタプルを集約･集計した. その結果を図 7.4 に示す. なお，全ての結果は，同様の操作を 5 回行った平均である． 図 7.4 より，明らかに差分構築を行った方が，処理時間が短いことが分かった．差分構築無しの場合，構築済みの cell について集計を行うが, 差分構築有りの場合, 構築済み cell を集計し直す必要が無いため，明らかに差分構築有りの方が早い.

表 7.4. 実験 3 の条件

構築済み Cell 数	70000 件
集約件数	3500 件～28000 件 (3500 件刻み)
各次元のデータ型	4Byte 整数
各次元カージナリティ	500

8. おわりに

経歴・パターン法を用いた MOLAP 用のデータキューブの構築について，多次元配列の次元拡張時に再構築の必要がない方式を提案した．また実験 1 より次元拡張自体に処理時間が掛からないこと，実験 2 から次元拡張の有無で同規模データキューブの各種処理時間に変化が無いことが確認できた．以上から，本方式は次元拡張に伴うコストがほとんど発生しない方式であると言える．

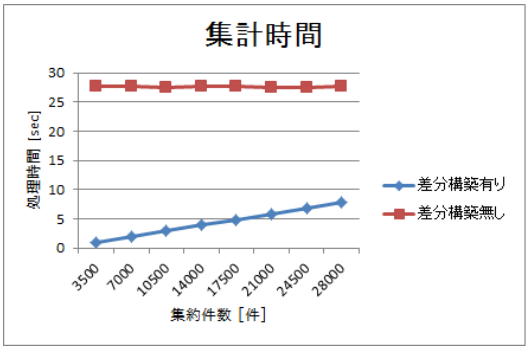


図 7.4 .計測 3 の結果

参考文献

1) Zhao, Y., Deshpande, P. M. and Naughton, J. F. : An array based algorithm for simultaneous multidimensional aggregate, *Proc of the ACM SIGMOD Conference*, pp.159–170, 1997.
2) Otoo, E. J., Rotem., D.: Efficient Storage Allocation of Large-Scale Extendible Multi-dimensional Scientific Datasets, *Proc. of SSDBM*, pp.179-183, 2006.
3) Tsuchida, T., Tsuji, T., Higuchi, K.: Implementing Vertical Splitting for Large Scale Multidimensional Datasets and Its Evaluations, *Proc. of DaWaK 2011*, pp. 208-223, 2011.
4) Jin, R. Yang, G., Vaidyanathan G., and Agrawal, K.: Communication and Memory Optimal Parallel Data Cube Construction, *IEEE Transactions On Parallel and Distributed Systems*, Vol.16, No. 12,pp.1105–1119, 2005.
5) Hurtado, C.A., Mendelzon, A.O. and Vaisman A.A.: Maintaining Data Cubes under Dimension Updates, *Proc. of the ICDE Conference*, pp.346–355, 1999.
6) Lee, K. Y. and Kim, M. H. : Efficient Incremental Maintenance of Data Cubes, *Proc. of the VLDB Conference*, pp.823–833, 2006.
7) Jin, D., Tsuji, T., Tsuchida, T., Higuchi, K.: An Incremental Maintenance Scheme of Data Cubes, *Proc. of DASFAA*, pp.682–685, 2008.
8) 都司，水野，松本，樋口：多次元データセットのコンパクトな実現方式の提案,日本データベース学会論文誌, Vol. 8, No. 3, pp. 1-6,2009,
9) 前田，水野，都司，樋口，“多次元データのコンパクトな実装とその性能評価”, 第 3 回データ工学と情報マネジメントに関するフォーラム，D6-2，2011.