MOLAPのための多次元配列の実現方式

一色淳夫† 鈴木程之∗ 都司達夫‡ 宝珍輝尚‡ 樋口健‡

† 福井大学大学院工学研究科, ∗ 福井大学大学院工学研究科 (現，日本電装 (株)),‡ 福井大学工学部

An Implementation Scheme of Multidimensional Arrays for MOLAP

†Atsushi Isshiki †Noriyuki Suzuki ‡Tatsuo TSUJI ‡Teruhisa HOCHIN ‡Ken HIGUCHI

†Graduate School of Engineering, Fukui University ‡Dept. of Information Science, Fukui University

あらまし MOLAP において使用される多次元配列は，一般に (1) 疎配列となる，(2) 配列要素の逐次検索速度が検索する配列次元に依存する，という問題点がある．(2) は多次元配列の要素をあらかじめ決められた次元順に線形に記憶空間に配置する時には不可避の問題であると言える．MOLAP システムにおいて，多次元配列を二次記憶に格納する上で良く使われる手法として, chunk と呼ばれる一様な大きさに配列全体を分割して，配置する手法がある. これにより, 検索速度の次元依存性を軽減することができる. このような chunk を単位とする二次記憶割り付けの研究ではいずれも，圧縮されたチャンクの二次記憶割付は従来通り配列要素レベルの次元順に準じている．圧縮に伴うチャンクのパッキング（コンテナ化）の視点はなく，圧縮チャンクレベルにおける次元依存性の回避は行われていない．その結果，チャンク圧縮を行って，二次記憶に割り付ける際には配列要素レベルの検索に置いても次元依存性が発生し得る．本研究ではこの点に着目して，チャンクコンテナの考え方を導入し，圧縮チャンクをコンテナに詰めるためのパッキングアルゴリズムを工夫することにより，配列圧縮時に発生する次元依存性の問題をも解決しようとしている.

1 はじめに

基幹系システムからのデータのスナップショットを取り, 大規模なデータベースに格納して, それを分析することにより，企業の意志決定支援に利用することが盛んに行われている. ユーザは, このデータベース中の任意の属性の組合せについて様々な集約結果を求め多次元分析を行う. これをサポートするためのデータキューブ演算が提案され [2] , データキューブを効率良く計算する手法の研究が行われている [9]〜[12] . システムにはユーザからの問い合わせを, 思考を妨げないようオンラインで答えられる速度が必要であり, この要求を満たすシステムが OLAP と呼ばれる [1].OLAP はバックエンドの構成により ROLAP, MOLAP に分類できる.

MOLAP (Multi-dimensional OLAP) は OLAP が提供する多次元分析機能をよりダイレクトにサポートする.MOLAP システムはバックエンドに多次元データベースを利用し, 集約演算結果をあらかじめ計算し、多次元配列として管理する. 集約演算が事前に行ってあるため応答時間は短いが, 事前計算時に想定した問い合わせにしか答えられず柔軟性では劣る. また, 事前計算のバッチ処理に非常に時間がかかる. MOLAP において使用される多次元配列は，一般に (1) 疎配列となる，(2) 集約演算や検索の速度が配列の次元に依存する，という問題点がある．(2) は多次元配列の要素を通常良く行われるようにあらかじめ決められた次元順に線形に記憶空間に配置する時には不可避の問題であると言える．このことは，決められた次元順とは異なる次元順に配列要素を検索する際には検索速度の低下を引き起こす．また，配列をスライス [13] して集約演算などを行う場合，スライスする次元に依存して応答性にばらつきが生じることになる.

多次元配列を二次記憶に格納する上で良く使われる手法が, chunk と呼ばれる一様な大きさに分割し管理する手法である [3] . これにより, 次元方向への検索速度の次元依存性を軽減することができる. [5] では, 大規模多次元配列を簡単なプログラミングで二次記憶に格納でき, 入出力性能を改善させるライブラリの提供を行っている. chunk に分割した多次元配列を分メモリ型プロセッサに割り当てて管理を行っている．[3] では一様なサイズであった chunk を [4] では, 任意のタイリングを行い変化させている. タイリングは, 問い合わせやデータの関連性, 階層レベルから任意で決定し, よりアプリケーションの要求に沿うように決定する. [6] では、関係データベースに chunk の概念を導入しており, "chunked file" と呼ぶ構成をとり chunk をキャッシュを行う際の管理単位とする. これにより, テーブルレベルやクエリーレベルでキャッシングを行う場合に比べ, 効果的なキャッシュの利用ができる.

[7], [8] にあるように, 多次元データベースを用いた MOLAP システムでは chunk は基本となるデータ管理単位で

ある. [7] では chunk 単位で BESS と呼ぶ方法でデータ圧縮を行うことでデータベースサイズを削減する. また, さらに並列化することによる性能向上を目指している.[8] でも chunk 単位でデータ圧縮を chunk-offset compression と呼ばれる方法で行っている. データキューブ演算を MOLAP - 多次元データベース上で行い, 配列のどの方向からスライス断面を取り出しても，そのコストも余り変わらないという特徴を生かし, より少ない必要領域で行うことを提案している.

上述の chunk を単位とする二次記憶割り付けの研究はいずれも，圧縮されたチャンクの二次記憶割付は従来通り配列要素レベルの次元順に準じている．また，圧縮に伴うチャンクのパッキング（コンテナ化）の視点はなく，圧縮チャンクレベルにおける次元依存性の回避は行われていない．その結果，チャンク圧縮を行って，二次記憶に割り付ける際には配列要素レベルの検索に置いても次元依存性が発生し得る．本研究ではこの点に着目して，チャンクコンテナの考え方を導入し，圧縮チャンクをコンテナに詰めるためのパッキングアルゴリズムを工夫することにより，圧縮時に発生する次元依存性の問題をも解決しようとしている．

本論文では, MOLAP のデータ管理システムである多次元データベースを構築する際に問題となる (1) と (2) の欠点を軽減するための方式の提案と評価を行なう．ここでは，配列全体を論理ブロックとしての chunk の集合として管理する．(1) の疎配列問題に対しては，chunk の充填率に依存して，chunk-offset compression と bitmap compression の手法を混用して chunk 単位で圧縮を行う．さらに，圧縮された chunk をディスクの 1 ページに収まるように複数まとめ，chunk コンテナとしてディスクの 1 回の読み込みで複数の chunk を得るようにしている．OLAP システムではスライス操作が頻繁に行なわれるため，隣接したデータは一度の読み込みで得られることが望ましい．そこで，圧縮チャンクレベルでの次元依存性の問題を解決するために，いくつかのコンテナ化のアルゴリズムを提案した後，シミュレーションによりその有効性を検証する．

2　疎配列に対する対策

MOLAP のデータは，多次元データベースが管理する大規模な配列に格納される．データの属性は配列の次元に対応しており，データに n 個の属性が存在し，i 番目の属性の値の数を c_i とすると，この配列は，サイズが $(c_1, c_2, ..., c_n)$ の n 次元配列である．データまたはそれへの参照はその各属性の値に応じて，この n 次元配列の 1 つの要素に格納される．ところが，各属性値のすべての組合せについてデータが存在する（配列が密）場合は少なく，通常は配列の要素数に比較してデータの数はかなり少ない．したがって，データの充填率は低く無駄な記憶領域を消費することとなる．これは，配列要素の高速アクセスのために，アドレス関数により配列要素を線形に配置するためである．データを RDB のテーブルのレコードとして管理する ROLAP（Relational OLAP）の場合は，このような充填率の低下の問題は存在しない．

このような無駄な配列要素を多く含む領域を” 疎領域” といい，疎領域が比較的多い配列を” 疎配列” という．疎領域は属性の数（配列の次元数）が増すにつれて，増大する．また，格納するデータが集約演算による階層構造の下位であるほど (詳細な区分のデータであるほど)，増大すると考えられる．上位階層は複数の下位階層の集計データを保持するため，逆に上位階層ほど疎領域は少ない．データが配列中のほぼすべての要素に格納されている場合を，“密” な配列と呼ぶ．この場合，多次元データベースはほとんど無駄な領域を消費しない．

この疎配列に対する対策として，後述の chunk 単位でのデータ圧縮を行う．圧縮方法の詳細は次の節で述べるが，データの存在する配列要素のみを格納し，データベースサイズの低減を行う．

3　検索時の次元依存性に対する対策

多次元配列はメモリ上では，次元の順に連続した領域として確保されている．この領域をそのままの順で二次記憶に格納したとする．格納次元順に沿った検索を行う場合は，二次記憶へのアクセスは最小限に抑えられるが，格納次元順に沿わないような検索をする場合，一回のページ読み込みで得られる有効データが少なく，多くの二次記憶へのアクセスが必要となる．

このような検索時の次元依存性は，MOLAP では大きな問題である．OLAP の目的は，対話的な多次元分析環境の提供であり，ユーザからの要求に対して応答性を損なわないことが期待される．また，ユーザのオンライン分析に必要なスライス要求などは，前もって特定できない．分析視点を変更することでレスポンスが大幅に悪化するようなシステムは好ましくない．すべての視点からのスライス要求に対し，一定範囲内の時間でレスポンスを返すことが望まれる．

3.1　chunk とその圧縮格納

MOLAP システムは検索時の次元依存性を解消するため，管理する大規模な配列は文献 [3] の “chunk” という単位に分割して取り扱う．chunk もまた n 次元の配列であり，その大きさは二次記憶の 1 ページ以内とする．大きさを 1 ページ以内とすることで，一回の読み込みで 1 chunk

を取り出すことができる．扱う単位を chunk とすることで，どの次元軸のスライス要求に対しても一回のページ読み込みで複数の隣接データを得ることができ，平均ページアクセス数の平均化を図ることができることが示せる．

ただし，この平均化は有効要素で満たされる密な chunk が配列の大部分を占めるときにはそのまま有効であるが，疎配列の場合には，有効データのみを格納するため圧縮を行う必要がある．本研究では，chunk データの圧縮を行いながら，上述のような次元依存性を解消するための方策を提案することを目的としている．データを圧縮する方法として，chunk-offset 圧縮，bit-map を利用した圧縮を考える．無圧縮の場合も含めて，これらの方法を図 1 に示す．

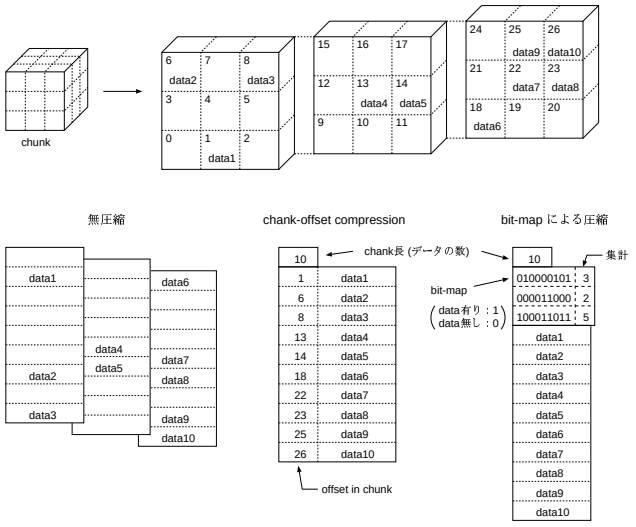

図 1: chunk の圧縮

chunk-offset 圧縮は，chunk 中に存在するデータを chunk 内のオフセット値とデータ値のペアとして格納する．無圧縮の場合と比べ chunk 内オフセット値を保持する必要があるが，chunk の大きさはあらかじめ決まっており，オフセット値を表現するためのビット長はそれほど大きくない．したがって格納データが少ない場合，有効な方法である．bit-map を利用した圧縮は，chunk 内の全要素を表すことのできる bit-map を用意し，データが存在する場合は 1，データが存在しない場合は 0，として記録する．この方法では，後方のオフセットのデータにアクセスするには，数多くの bit を数えなくてはならない．そこで，bit-map のあるサイズ毎にそこまでに立っている bit の集計値を保存しておくことで対処する．この手法では，bit-map と bit 集計値を保存する必要があるため，格納データが少ない場合は不利である．しかし， 格納データが多くなっても増えるのはデータの格納スペースだけであり，中程度のデータ密度では有効であると思われる．データ密度が高い場合は，無圧縮のまま格納する．これは，圧縮のためにはメタデータが必要であり，データ密度が高いと無圧縮時より記憶域を消費することがあること，また，圧縮していない分，高速にアクセスが可能であることによる．

これらの圧縮法をどのデータ密度で使用するかは，速度とデータベースサイズの関係から，ユーザが決定することになる．各圧縮方法での，データの充填率と必要サイズの関係を図 2 にグラフ化した．ディスクのページサイズを 8,192 byte とし，chunk は無圧縮状態でページサイズと同じ，格納データは実際のデータレコードへの参照として 8 byte とする．chunk-offset 圧縮における offset は 2 byte，bit-map に合計 144 byte，chunk のメタ情報として格納データ数を保存するため 2 byte 確保している．システムの利用可能なリソースにもよるが，0〜20%ぐらいで Chunk-Offset，〜70%で bit-map，それ以上で無圧縮の利用が有効であることがわかる．

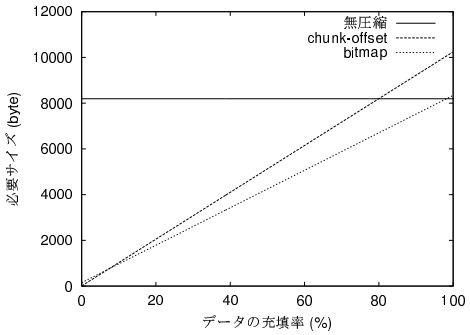

図 2: 圧縮比

3.2 コンテナ化

chunk はその要素が有効であるか否かに関わらず，二次記憶の 1 ページ分の論理サイズを占めるものとする．chunk の全要素が有効データで埋まっている無圧縮の状態では，二次記憶における chunk の物理サイズも 1 ページである．chunk が疎である場合，圧縮された chunk は 1 ページより小さくなる．ここではそのページ上に複数の圧縮された chunk を配置するものとする．これにより、一度の読み込みで複数の chunk を読み込むことができる．このまとめた複数の圧縮 chunk を収容するための二次記憶の 1 ページを “chunk コンテナ” （以後，単にコンテナ）と呼ぶ．chunk は論理的な大きさの単位であるのに対して，コンテナは二次記憶中の物理的な大きさの単位であることに注意されたい．

ここで，どの chunk を組合わせてコンテナに格納するかという問題が生じる．配列のセルを chunk に分割した際と同様に，この圧縮 chunk を配列の順に格納すると，その格納次元方向に依存性が発生する．この圧縮 chunk 格納時の次元依存性も，スライス時に応答時間の標準偏差を

悪化させる原因であり，解消しなくてはならない．この問題を解決するには，コンテナを構成する際に方向性がなくなるように chunk を選ぶ必要がある．

コンテナの充填率を 100% に近づけると全体のコンテナ数が減少して，総読み込み回数が減少すると考えられる．OLAP ではスライスとダイス操作 [13] が処理の中心となり，スライスでは，隣接した chunk 内の配列要素を必要とする．コンテナ化にあたっては，より近い chunk 同士を同一コンテナに格納することにより，検索のスライス要求時に読み込み回数が減少すると考えられる．したがって，なるべく隣接した chunk 同士でコンテナ内の充填率を上げる必要がある．

4 コンテナ化の方式

上述の主旨に沿うための，いくつかの種類のコンテナ化方式を以下に提案する（図 3）．コンテナ化を行う上で基点となる chunk は，3 次元配列なら (0, 0, 0) から始める．(0, 0, 0) を基点とするコンテナの作成が終わったら，配列の順に基点を移動してゆく．つまり，$10 \times 10 \times 10$ の chunk 配列なら，(0, 0, 0)，(1, 0, 0)，(2, 0,0),... (0, 1, 0),... (9, 9, 9) と移動する．

[A] 配列順 (配列順)

(chunk 化しない）通常の配列と同じように，定められた次元の順に沿って順番に chunk をコンテナに格納する．充填率が 100% を越えないところまでを 1 つのコンテナとして扱い，chunk レベルでの次元依存性の解消を考慮しない．その他の方式との比較対象とする．

[B] 特定次元優先 (PRI)

コンテナ化の度に各次元方向の優先順位をランダムに決定し，最優先次元方向の chunk から優先的にコンテナバッファに詰める．各次元の優先順位をチャンク化の度に変更することで，複数の chunk を 1 つのコンテナに詰めた場合の次元依存性の解消を図る．ここで，1 回のコンテナ化において，コンテナ化の対象となり得る chunk は，各次元方向で一定個数の範囲 R 内とする．できるだけ距離的に近い chunk をまとめてコンテナ化するほうが，スライス時の平均読み込み回数の点で有利であると考えられるためである．

優先次元方向へ chunk をコンテナバッファに付け加えていった時，(a)R のその次元での終端に至った場合，(b) 対象 chunk がすでにコンテナ化済みであり，他のコンテナに含まれる場合，(c) 対象 chunk を付け加えるとコンテナバッファの充填率が 100%を越える場合 は次の優先度の次元方向へ 1 つ分拡張し同様の処理を行う．(c) の場合においても，その時点でコンテナ化を行うことはしない．最も優先度の低い次元について，処理が終わった時点，すなわち R 内のすべての chunk を処理した時点で，コンテナバッファ上の chunk 集合を二次記憶に書き出し，コンテナ化を行う．続いて，R の起点移動が行われ，同じアルゴリズムにより，新たなコンテナ化を進める．

もし，R 内の chunk の充填率が低く，R のすべての chunk がコンテナ化された場合には，充填率の低いコンテナが生成されることとなる．逆に，R 内の chunk の充填率が高い場合には，100%近い充填率のコンテナが生成されるが，R 内の他の chunk はコンテナ化されないままである．ただし,R の最後の chunk がコンテナに収まりきらない場合は、その chunk についてはコンテナ化を行わないこのためにコンテナの充填率が低くなる場合もある。この場合，これらの chunk は起点移動による新たな範囲 R に対するコンテナ化でカバーされることになる．

[C] 超直方体 (rect-x)

コンテナ化の度にランダムに優先次元を選択するのは同様である．ただし，現在の形 (n 次元の超直方体) から優先方向へ 1 chunk 分だけ拡張してゆく．優先方向への拡張ができたら，次の優先方向へ超直方体の形状を保つように拡張を行う．周辺 chunk を優先し，全体としては n 次元の超直方体となる．以下の 3 種類を測定する．

● rect-1 は優先次元方向重視であり，初めに決めた優先次元方向の順でのみコンテナ化を行なう．拡張の予定範囲内にすでにコンテナ化済の chunk が含まれたり，コンテナの充填率が 100%を越えるためその方向へ拡張できなくなったら，それまでの chunk 集合をコンテナに格納する．

● rect-2 はコンテナ内 chunk 充填率重視であり，rect-1 とほぼ同様であるが，優先次元の方向へ拡張できなくなったら，次の優先次元方向でチェックをおこない，拡張できるようならその方向へ拡張する．そのため，rect-1 より多くの chunk をコンテナに格納できるが，ある方向へ偏って伸びた 2×5 のようなコンテナができることがある．

● rect-3 は整った超立方体を形成する．rect-2 の偏った方向へ伸びる問題を防ぐため，各次元方向で伸ばした chunk 数の差が 1 以下になるようにし，コンテナの形状を整える．

[D] 特定領域内でランダム (rand-x)

特定領域でランダムに chunk を選んで，その chunk をコンテナに加えられるなら加える．加えられない場合は，それまでの chunk 集合をコンテナとして格納する．領域にまだコンテナ化されていない chunk が残っていれば，同じ領域内で次のコンテナを作成する．したがって，領域内で作成される最後のコンテナは，充填率が低い．

● rand-1 は小さな領域 (各次元方向へ 2 chunk) でコンテナ化を行う．

● rand-2 は大きな領域 (各次元方向へ 3 chunk) でコンテナ化を行う．

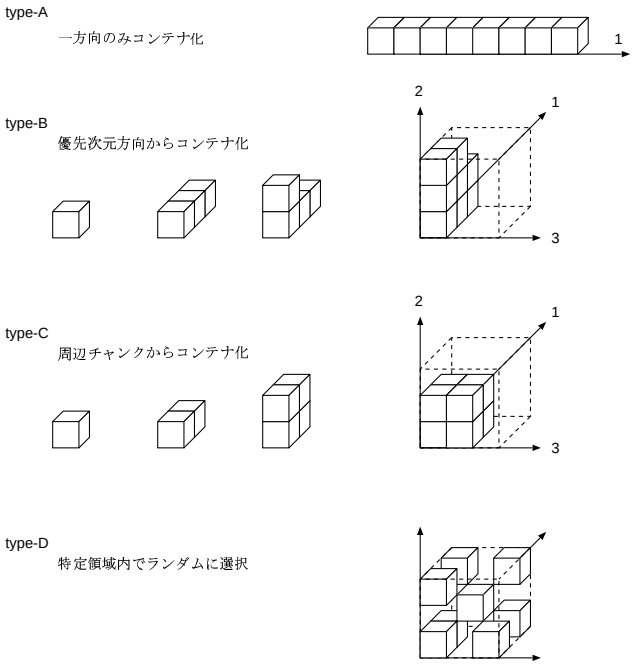

図 3: chunk のコンテナ化

5 評価

5.1 シミュレーション条件

1 つの次元方向に 36 個の chunk を持つ 5 次元配列を用意し，全 chunk が一様なデータ密度であり，圧縮時に 10% になると仮定する．全体の chunk 数は，$36 \times 36 \times 36 \times 36 \times 36 = 60,466,176$ 個である．この 5 次元の chunk 配列に対し前述のコンテナ化を行い，各次元数でのスライス時のコンテナ読み込み回数とばらつきを標準偏差で測定する．ここで，n 次元配列の i 次元スライスとは，$n-i$ 個の次元の値を固定して n 次元分を可変にした時にできる i 次元超平面である．平均読み込み回数と標準偏差値はこのような超平面中のコンテナの数の平均と標準偏差である。固定分のバリエーションが ${}_nC_i \times 36^{(5-n)}$ できるので, このような超平面の数は ${}_nC_i \times 36^{(5-n)}$ である．標準偏差値が低いほど次元依存性が解消されていることを示している．

5.2 実験結果と考察

図 4 は，コンテナ化を行った際のコンテナ総数である．rect1 のコンテナ数が他の方式と比べて多いことが分かる．図 5 は，コンテナ内充填率に対応するコンテナの数である．rect1 の 10%，20%，40%，の充填率のコンテナが多くなっている．rect1 は，優先次元方向へコンテナを拡張できないとコンテナ化を打ち切り，それまでの chunk だけでコンテナを形成するので，総コンテナ数を増加させている．

図 6 から 図 13 は，各次元軸に垂直にスライスを行った際の平均 chunk 読み込み数とその標準偏差である．4 次元のスライス要求があった場合，総コンテナ数の多さから rect1 の読み込み回数が多い．次が rand2，配列順，の順である．rand2 は $3 \times 3 \times 3 \times 3 \times 3 = 243$ の領域のなかからランダムにコンテナ化を行なう．rand1 は $2 \times 2 \times 2 \times 2 \times 2 = 32$ の中から選択する．rand2 が rand1 と比べて読み込み回数が多い原因は，選択領域が広いため同一スライス平面上の chunk が，同じコンテナに少ないためである．rand1, rand2 は，コンテナ化を行なう chunk を乱数で選択しているため，標準偏差は非常に低く，どのようなスライスを行なっても読み込み回数に差はない．

配列順は総コンテナ数が最も少ないが，平均読み込み回数は良くない．この原因は，次元依存性にある．配列順のスライス時の読読み込み回数は，コンテナすべてが要求されたスライス平面に乗るか，もしくはコンテナ内の 1 つの chunk しか利用できないために，非常に極端である．その結果，標準偏差も他の方式と比べ遥かに大きい．

平均読み込み回数が少ないのは rect2，rect3，PRI である．rect2 は超立方体を保ったまま，なるべく多くの chunk を取り込もうとする．その結果，90% (3×3)，100% (2×5) の様な形状をとることも多く，標準偏差がやや大きい．rect3 は各次元方向の大きさの差が 1 以下となる 整った形状の超立方体を作る．あえて 100% のコンテナを目指さないが，逆に多くのコンテナを 80% の充填率にすることができ，平均読み込み回数，標準偏差ともに良い結果が得られている．PRI は，逆に 100% の充填率を目指してコンテナ化を行なう．その結果，rect3 と比べて 10% 〜 90% のコンテナも多い．しかし，100% のコンテナも多いので，rect3 と遜色のない結果が得られている．この PRI と rect3 が平均読み込み回数，標準偏差から見て有効なコンテナ化方式であると思われる．

6 おわりに

MOLAP を構築する際に問題となる，疎配列の問題と，次元依存性を軽減する実装方式の検討を行なった．シ特定の次元方向を優先する PRI の 2 つの 方式が，平均読み込み回数，標準偏差の点で優れていることを確認した．以上のことから，chunk のコンテナ化を行なうことで MOLAP の，スライス時平均読み込み回数と，スライスの取り方による読み込み回数のばらつきが解消されることを明らかにした．ここでは，データ値は多次元配列において，一様に分布し，各次元のサイズも同一としたが，これらの制限を緩和した場合の検証も必要である．また，コンテナ化を行い二次記憶に格納するコストの検証も必要であると思われる．

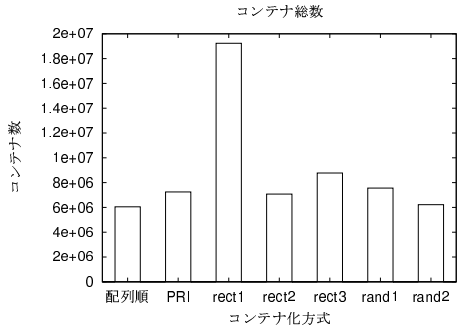

図 4: コンテナ総数

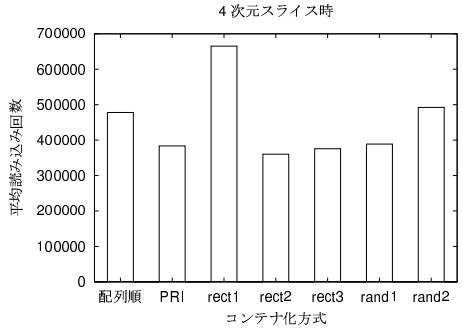

図 6: 4 次元スライス時の平均読 込 数

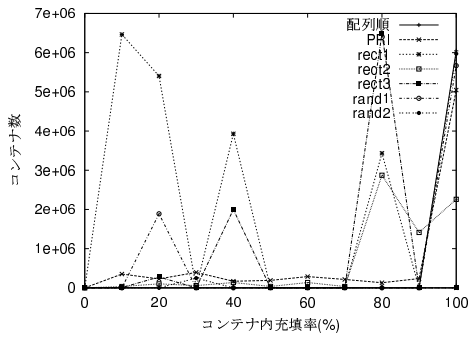

図 5: コンテナ内充填率

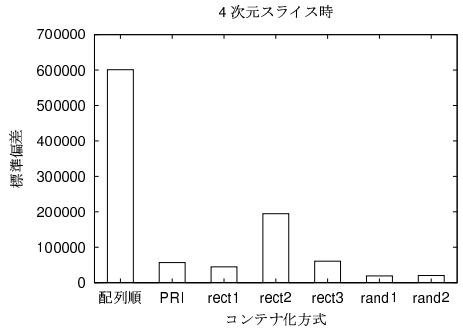

図 7: 4 次元スライス時の標準偏差

参考文献

[1] E.F.Codd, S.B.Codd, C.T.Salley, *"Beyond decision support"*, Computerworld, vol.27, no.30, July 1993.

[2] J.Gray, A.Bosworth, A.Layman, and H.Pirahesh, *"Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals"*, Technical Report, Microsoft, Nov. 1995.

[3] S. Sarawagi, M. Stonebraker, *"Efficient Organization of Large Multidimensional Arrays"*, Proc. of ICDE, pp.328–336, 1994.

[4] Paula Furtado, Peter Baumann, *"Storage of Multidimensional Arrays Based on Arbitrary Tiling"*, 1999 IEEE

[5] Kent E. Seamons, Marianne Winslett, *"Physical Schemas for Large Multidimensional Arrays in Scientific Computing Applications"*, 1994 IEEE

[6] P.M.Deshpande, K.Ramasamy, A.Shukla, and J.F.Naughton, *"Caching multidimensional queries using chunks"*, ACM SIGMOD, p.259-270, Seattle, WA, june 1998.

[7] Sanjay Goil, Alok Choudhary, *"A Parallel Scalable Inforastructure for OLAP and Data Mining"*, International Data Engineering and Applications Symposium (IDEAS'99), Montreal, August 1999.

[8] Yihong Zhao, Prasad M.Deshpande, Jeffry F.Naughton, *"An Array-Based Algorithm for Simultaneous Multidimensional Aggregates"*, SIGMOD '97 AZ,USA p.159-170

[9] 松澤 裕史, 福田 剛志, "複数の集約演算のための並列アルゴリズム", 電子情報通信学会論文誌 D-1 Vol. J82-D-1 No.1 pp98-110, January 1999.

[10] Sameet Agrawal, Rakesh Agrawel, Prasad M. Deshpande, Ashish Gupta, Jeffrey F. Naughton, Raghu Ramakrishnan, and Sunita Sarawagi, "On the Computation of Multidimensional Aggregates", Proc. of the 22nd VLDB Conference, Mumbai(Bombay), India, Sept. 1996.

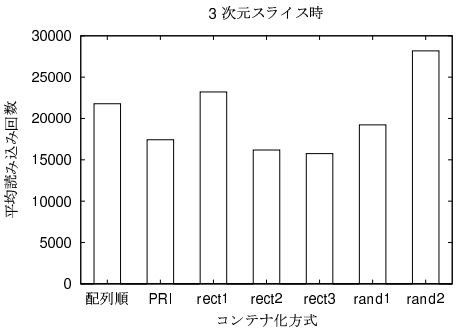

図 8: 3 次元スライス時の平均読 込 数

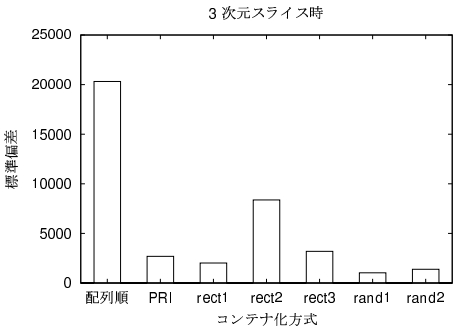

図 9: 3 次元スライス時の標準偏差

[11] Sunita Sarawagi, Rakesh Agrawal, Ashish Gupta, *"On Computing the Data Cube"*, Research Report RH10026, IBM Almaden Research Center, 1996.

[12] Theodore Johnson, Dennis Shasha, "Hierarchically Split Cube Forests for Decision Support: description and tuned desgn", Working Paper, 1996.

[13] M.Abbey et.al., *"SQL Server 7 Data Warehousing"*, McGraw-Hill, 1999.

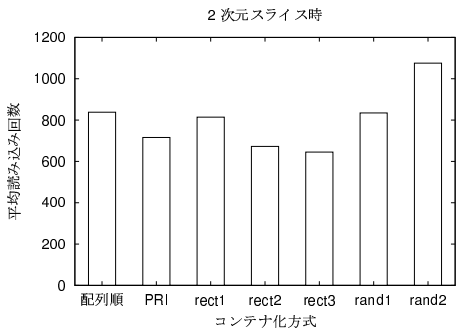

図 10: 2 次元スライス時の平均読 込 数

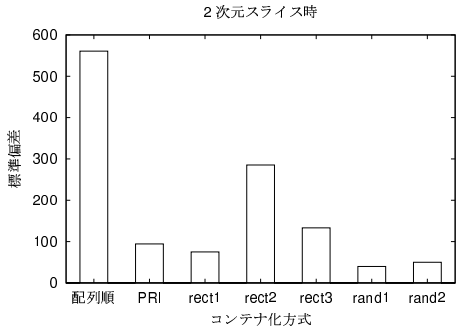

図 11: 2 次元スライス時の標準偏差

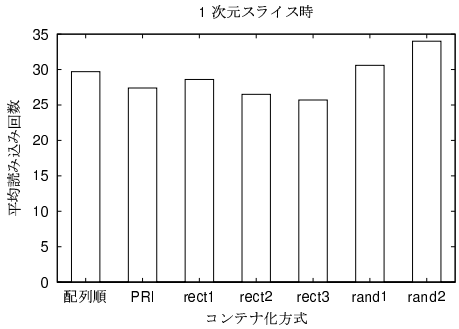

図 12: 1 次元スライス時の平均読 込 数

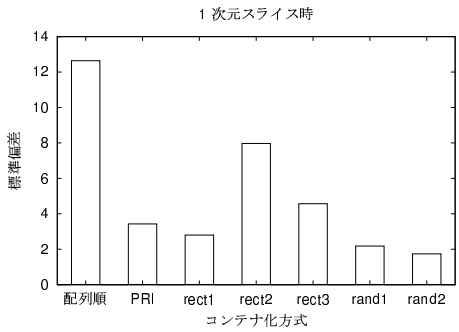

図 13: 1 次元スライス時の標準偏差