2005－DBS－135　(14)
2005／1／21

拡張チャンクによる多次元配列の圧縮格納方式

堅田晃平†　特日格勒†　都司達夫††　樋口　健††

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二次記憶領域に配置するときには不可避な問題である．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よ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．(1)の問題を解決するために chunk の圧縮を行うとき，chunk コンテナと呼ぶディスクページに複数の圧縮 chunk を詰め合わせる．我々は以前，この詰め合わせアルゴリズムを提案して，評価した．ここでは，拡張 chunk の考え方を提案して chunk の充填率が極端に低い状況にも対応できるコンテナ化アルゴリズムを提案して，シミュレーションにより評価する．

A Compression Scheme for Multidimensional Arrays Using Extended Chunks

Kohei KATADA†　TERIGELE†　Tatsuo TSUJI†† Ken HIGUCHI††

The implementation of multidimensional arrays used in MOLAP suffers from the problems; (1) the number of nonempty elements tends to be so small, (2) the time consumed in sequential access to array elements heavily depends on the dimension along which the elements are accessed. The second problem is inevitable in allocating a multidimensional array according to the predefined order of dimensions. In MOLAP, the whole multidimensional array is often partitioned into subarrays called chunks and stored in the secondary storage. In order to resolve the first problem, theses chunks are compressed and packed in a disk page called a chunk container. We have already proposed several chunk containerization algorithms and evaluated them. In this paper, by introducing the notion of the extended chunk, a new chunk containerization scheme is presented for against the situation where the filling up ratio of the chunks is extremely low, and the scheme is evaluated by simulation.

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，企業の意思決定支援に利用することが盛んに行われている[1]．ユーザは，このデータベース中のデータの任意の属性の組み合わせについて様々な集約結果を求め，多次元分析を行う[2][3]．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(On Line Analytical Processing)システムと呼ばれる[4]．OLAP はバックエンドのデータベースにより ROLAP と MOLAP[5][6]に分類できる．

MOLAP の対象データは通常バックエンドの多次元データベースが管理する大規模な多次元配列[7][8][9]に格納される．この多次元配列はフロントエンドの関係データベーステーブルの実装に使われるが，次のような問題点がある．

(1) テーブル中のレコードは配列要素の添字の組で表されるが，一般にレコード総数は配列の総要素数に比べて非常に少なく疎配列となる，

(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二次記憶領域に配置するときには不可避な問題である．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．MOLAP システムにおいて，多次元配列を

† 福井大学工学研究科
Graduate School of Engineering., Fukui University
†† 福井大学工学部
Faculty of Engineering., Fukui University

二次記憶に格納する上でよ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よく使われる．これにより特定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が，(1)の問題を解決するために chunk の圧縮を行うとき，chunk コンテナと呼ぶディスクページに複数の圧縮chunkを詰め合わせる．その時に圧縮する chunk の選択方法によっては新たな次元依存性が発生し得る．我々は以前，この圧縮chunkの詰め合わせアルゴリズムを提案して，評価した．MOLAP の対象となる多次元配列は極端に疎であることも多く，chunk の充填率が低い状況での取り扱いは十分ではなかった．本論文では，拡張 chunk の考え方を提案して chunk の充填率が極端に低い状況にも対応できるコンテナ化アルゴリズムを提案して，シミュレーションにより評価する．

2．chunkの圧縮とコンテナ化

検索時の次元依存性を解消するために MOLAP システムが管理する大規模配列は“chunk”[8][9]という単位に分割して取り扱われることが多い．chunk は各次元長が同一の配列であり，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1回の読み込みで1chunk 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 1 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，ページ読み込み回数の平均化を図ることができる．ただし，この平均化は密な配列の場合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格納するための圧縮を行う必要がある．この時圧縮された chunk の詰め合わせ方によっては, 更に次元依存性が発生し得る．

2.1 コンテナ化

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 1 ページに複数の圧縮 chunk を詰め合わせて配置する．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 を格納するための二次記憶の 1 ページを“chunk コンテナ”(以後，コンテナ)と呼ぶ．

ここで複数の chunk を組み合わせてコンテナに格納する際に，圧縮 chunk を特定の次元順に格納するとその格納次元方向に次元依存性が発生する．この圧縮 chunk 格納時の次元依存性も検索時の応答時間のばらつきを悪化させる原因であり，解消しなくてはならない．以下では，我々が提案してきた 2 種の chunk 詰め込み(コンテナ化)アルゴリズムを説明する．詳細は[6]を参照されたい．

[A]Dynamic order(d-order)

本アルゴリズムでは距離の近傍性を保証するためにドメインの概念を導入している．ドメイン D 内でコンテナ化を開始する chunk を特に基点 chunk と呼ぶ．一回の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コンテナ化の度に変更することで，複数の chunk を一つのコンテナに詰めた場合の次元依存性の解消を図っている．一回のコンテナ化が終われば，そのドメイン内の残りの chunk はそこで放棄する．その後，D を 1 chunk 分，あらかじめ定められた次元順に次元軸に沿って移動させた後，移動後のドメインについて基点 chunk よりコンテナ化を継続する．ここでコンテナ化を放棄された D 内の chunk はドメインの移動により，後のドメインのいずれかでコンテナ化されるので最終的に全てのchunkがコンテナ化されることが保証される．

コンテナ化に当たっては現在の優先次元方向へ chunk をコンテナバッファに付け加えてゆく．この時，現在処理しているドメイン D の次元の終端に至った場合，対象 chunk がすでに以前のドメインにおいてコンテナ化済みである場合，対象 chunk を付け加えるとコンテナバッファがあふれる場合，以上 3 つの場合は次の優先度の次元方向へ一つ分移動し，同様の処理を行う．全ての次元方向について処理し終えた後，コンテナバッファを二次記憶に書き込む．

[B]Hyper Rectangular(rect)

コンテナに収容される chunk 集合の形状を一般には超直方体として，近傍 chunk をコンテナに詰め合わせるが，優先次元方向を重視する方式と，コンテナ充填率を重視する方式の中間的な方式であり，超立方体の形状を維持しようとする．各次元の chunk 数の差が常に 1 以下の範囲内で超平面を拡張する．現在の超平面を処理できたら，次の

優先方向へ超立方体の形状を保つように拡張する．すでにコンテナ化済みのchunkが存在する場合は次の優先次元に切り替えて，コンテナ化を続行する．各次元方向のchunk数の差が1以下になるのでコンテナバッファに格納されるchunk集合の形状は超立方体に近い形に整えられる．

3. 拡張chunkによるコンテナ化

2節において提案されているchunk選択アルゴリズム d-order ではコンテナ化を行う範囲をドメインという有限の範囲に限定している．ドメイン内の chunk の充填率が低い場合に，100%に満たないコンテナが多数出力されると考えられる．また，ドメインを用いない rect においては chunk 選択を超平面で行っているため，コンテナ化が進むにつれて 1 度にチェックしなければならない chunk 数が増え，これらがすべて当該コンテナに納まらなければ，そこでコンテナ化を終了するか，他のより大きい超平面内のchunk集合を同様に試みる．したがって，この場合もコンテナ充填率100%を目指すことが難しくなる．

以上の点から多次元配列全体が低充填率 chunk で構成されていた場合におけるコンテナ化アルゴリズムの見直しが必要であると考えられる．そこで現状の chunk の特性を生かしつつ chunk 内要素数を増加させる拡張chunkの考え方を提案し，拡張chunkによるコンテナ化アルゴリズムを提案する．

3.1 拡張chunk

2節における多次元配列のchunkを以下では基本chunkと呼ぶ．拡張chunkは基本chunkと同じような形状をもちつつ，二次記憶の１ページにおける充填率を 100%に近づけるために各次元長を増大させた chunk である．拡張 chunk の次元長はその拡張chunkが多次元配列内でどの位置にあり，どのようなサイズであるかの確認を容易にするために全て基本 chunk の次元長の 2^n となるように決定している．基本 chunk は n=0 の時の拡張chunkである．nを拡張chunkのrank値と呼ぶ．したがって，基本chunkのrank値は0である．

拡張chunkのrank値の決定は次のように行われるchunkの拡張処理を行うために拡張領域を決定する．その領域は常に現在の拡張chunkの次元長の2倍の領域，つまり現在の拡張chunkを 2^{dim}(dimは次元数)個収容できる領域であり，現在の拡張chunkのrank値をnとすると，この拡張領域はランク値が n+1 の拡張 chunk となる．ここでこのランク値 n+1 の拡張 chunk の情報を代表して持つ rank 値 n の拡張 chunk を 1 つ決定し，その拡張 chunk を基点拡張 chunk と呼ぶ．拡張できるか否かの判定は拡張領域内の有効要素数によって図1のように判断され，その数が二次記憶の1ページに収まる数ならばその領域をrank値n+1の拡張chunkとして決定し，1ページに収まらない数ならばその拡張領域を 2^{dim} 個の rank 値 n の拡張chunkの集まりであるとしてrank値を決定する．1 回の拡張処理が終わればその領域をあらかじめ決められた次元順に次元軸にそって領域サイズ分移動する．こうすることにより，多次元配列内の全ての基本chunkの拡張処理を漏れなく行うことが保証される．

以上のrank値決定処理をn = 1 から最大rank値まで繰り返す．その最大rank値は $p \cdot 2^x \leqq y$（pは基本chunkの次元長，yはコンテナ化を行う多次元配列の全次元の中で最も小さい次元の長さ）を満たす範囲，つまり $x \leqq \log_2 y - \log_2 p$ を満たす最大の rank 値 x である．

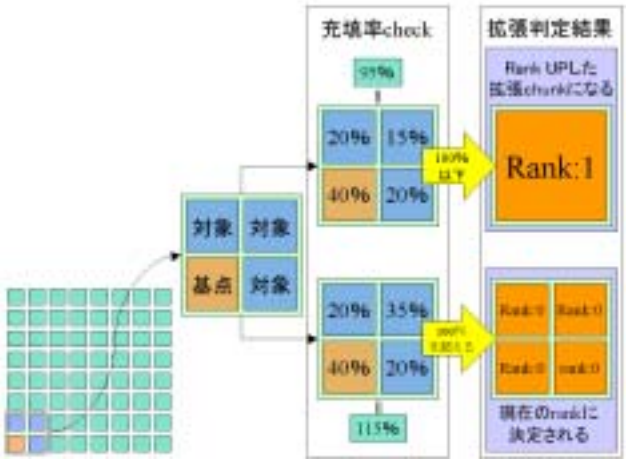

図１　rank値０からの拡張

3.2 拡張chunkの結合

上述した手順によって多次元配列を拡張 chunk の集まりとして構成することができるが，拡張chunkには柔軟性が不足しており，それ単一では二次記憶の１ページにおける充填率 100%を目指す

ことは難しい．そこで拡張 chunk 同士を結合することでページの充填率 100%を目指す．但し，ここでもその結合方法によって次元依存性が発生する可能性が含まれているため，結合拡張 chunk の選択方法を考える必要がある．今回は有効要素数を考慮した近傍性優先となる結合方法を用いる．結合後の拡張 chunk 群はコンテナとして，二次記憶中に格納される．

最初に基本 chunk で構成される多次元配列の(0,0,…,0)の基本 chunk を含む拡張 chunk を結合の核となる基点拡張 chunk に決定し，同時に基点拡張chunkの移動のための各次元の優先度を与える．この優先度とは別に結合における各次元の優先度をコンテナ化の度にランダムに決定する．これはコンテナ化に際して常に各次元に一定の優先度を与えるとそれによって次元依存性を発生させると考えられるからである．結合は以下の手順で行われる．

基点拡張 chunk に結合する際，ランダムに決定した優先度の高い次元から順に隣接する基本 chunk を選択し，その基本 chunk を含む拡張 chunk を併せて現在，コンテナに含まれる有効要素の総数を計算する．有効要素総数が 1 ページ以内であり，かつその拡張 chunk がいまだ他の拡張 chunk と結合していなければ結合可能と判断して，コンテナに収容する．結合できるものから順に結合して行きどの次元にも結合を進めることができなくなれば 1 回のコンテナ化が終了する．

コンテナ化が終了すれば基点拡張 chunk をあらかじめ決められた次元の次元軸に沿って移動し，上記コンテナ化を継続する．基点拡張 chunk を多次元配列の最後の拡張chunkまで進めることで全ての拡張chunkを漏れなく結合できることが保証される．

4． シミュレーションによる評価

前節で提案した chunk 配列のコンテナ化方式をシミュレーションにより評価する．その評価にあたって 2 節の d-order，rect の 2 種のコンテナ化アルゴリズムを比較対象とする．

4.1 シミュレーション条件

一つの次元方向に 18 個の chunk を持ち，各次元長が等しい 5 次元 chunk 配列をコンテナ化対象とする．その配列に対して配列中の chunk のデータ密度分布を 0.01%～0.1%の範囲を 0.01%刻みで変化させる．この条件下で，拡張 chunk，ドメインの各次元の長さ(以下ドメイン長)3 である d-order，ドメイン長 7 である d-order，rect の四種のコンテナ化を行う．ここで d-order に関して二種類のケースを考えているのはドメイン内に存在する全chunkをコンテナに詰め込んだ時コンテナに含まれる有効要素数が二次記憶における 1 ページに収まるケースと収まらないケースの二種類のデータを取るためであり，chunk の充填率が 0.01%時において最も有効であると思われるドメイン長が 7 であり，chunk の充填率が 0.1%の時にドメイン内の全chunkをコンテナに詰め合わせてそのコンテナの充填率が 100%を超えないのはドメイン長が 3 以下の時であるのでドメイン長 7 と 3 を採用している．上記四種類のコンテナ化を行った結果，それぞれの方式によって生成されたコンテナ総数の比較を行う．また，3 次元スライス時における平均コンテナ読み込み回数と次元依存性によって生じるそのばらつきを示す標準偏差の平均読み込み回数に対する割合をみる．ここで，n 次元配列の i 次元スライス(i=1,2,…,n-1)とは，n－i 個の次元の値を固定して i 次元分を可変にしたときにできる i 次元超平面である．この i 次元長平面の数はmを各次元の次元長であるとすると ${}_5C_i \times m^{(5-i)}$であり，平均コンテナ読み込み回数と標準偏差はこのような超平面内のコンテナ数の平均値とその標準偏差である．平均値に対して標準偏差の割合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．

次に，一つの次元方向に 2~32 個の chunk を持ち，各次元長が等しい 5 次元 chunk 配列をコンテナ化対象とする．配列中の chunk のデータ密度分布はある次元に対して正規分布を持ち，他の次元については一様な密度分布であるとしている．これは部分的に限りなく 0 に近い充填率とそうでない部分の両方を実現するためである．この条件下で拡張 chunk，d-order，rect の三種のコンテナ化を行い，それぞれの方式によって生成されたコンテナ総数の全 chunk 数に対する割合を見る．これにより，コンテナの充填率の平均を得る．また，1 つの次元方向に 18 個の chunk を持つ 5 次元の多

次元chunk配列においてはコンテナ充填率別コンテナ数と 3 次元スライス時における平均コンテナ読み込み回数と次元依存性によって生じるそのばらつきを示す標準偏差を測定する.

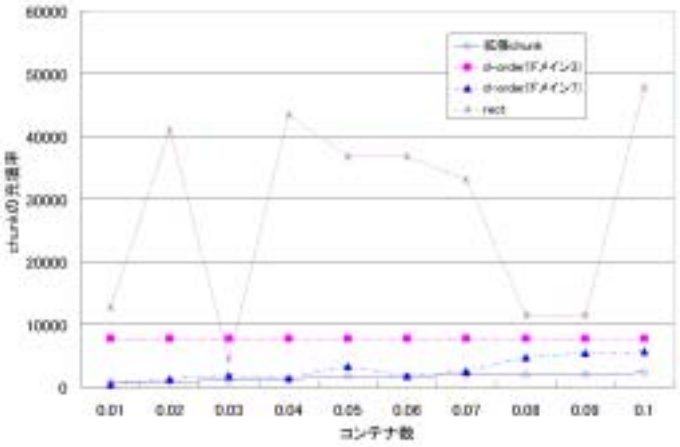

図 2　chunk の充填率別コンテナ総数

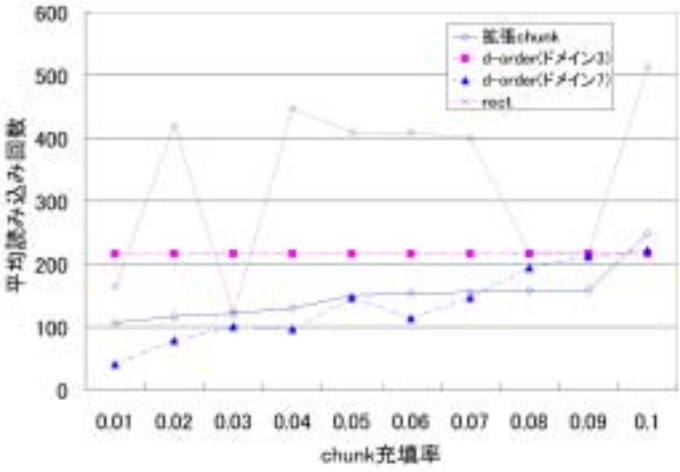

図 3　コンテナの充填率別 3 次元スライスの平均コンテナ読み込み回数

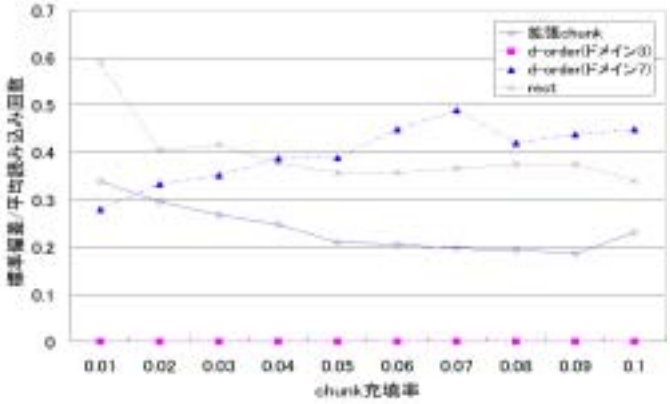

図 4　コンテナ充填率別 3 次元スライスの標準偏差./平均読み込み回数

シミュレータの入力として，各 chunk のデータ密度を格納した float 型の 1 次元配列を用いた. 各 chunk はすでに[9]で提案されている chunk-offset や bitmap などの圧縮方式にて圧縮されているとし, シミュレーションでは chunk を処理単位とした. また, 出力として, コンテナ化の結果, chunk が格納されるコンテナ番号を整数型の 1 次元配列に chunk 毎に記録した.

4.2　実験結果と考察

上記の二種のシミュレーション条件のもと実験を行った.

4.2.1　chunk の充填率を変化させた時

一つの次元方向に 18 個の chunk を持ち，各次元長が等しい 5 次元 chunk 配列をコンテナ化対象とし，その配列中の chunk のデータ密度分布を 0.01%～0.1%の範囲を 0.01%刻みで変化させて実験を行った.

図 2 に充填率変化に伴う全 chunk 数に対する総コンテナ数の割合のグラフを示す. chunk 充填率が 0.01%の時にドメイン長 7 の d-order に劣っているが全体として見ると他のアルゴリズムには見られないような安定して低いコンテナ数で落ち着いているのが見られる. ここでドメイン長 3 の d-order は全充填率においてまったく同じ値であるが, ドメイン内の全 chunk をコンテナに詰め合わせてもその有効要素数が二次記憶の 1 ページに満たないため, 多次元 chunk 配列がドメインサイズに分割され，その 1 つの分割が 1 つのコンテナに格納される. したがって分割の数だけコンテナが存在するのでコンテナ数は変化しない.

次に図 3 に，3 次元スライス時における平均コンテナ読み込み回数のグラフを示す. ドメイン長 7 の d-order と比べると拡張 chunk を用いたときの平均読み込み回数が若干多いが, 多少 chunk の平均充填率が変化してもほとんど変わらないという安定性を見せている. また，chunk の充填率が 0.1%の時に跳ね上がっている. これは平均充填率 0.1%の chunk の集合により充填率 100%のコンテナができたとするとそのコンテナは 1000 個の chunk で構成されており, 0.09%の時は 1111 個の chunk で構成されている. 一方 rank 値 2 の拡張 chunk 集合により 1 つのコンテナを構成するには

4^5=1024 個の基本 chunk を含めなければならない．つまり，chunk の平均充填率が 0.1%の時は rank 値 2 の拡張 chunk が作られることは無く 0.09%の時は rank 値 2 の拡張 chunk が作られる点にある．0.1%の時には rank 値 2 の拡張 chunk ができないために 1 つのコンテナでカバーできる chunk 集合のサイズが小さくなるため，総コンテナ数が増加するので平均コンテナ読み込み回数も増加したと考えられる．

次に図 4 に 3 次元スライス時における平均コンテナ読み込み回数に対する標準偏差の割合のグラフをしめす．標準偏差自体の特性上，標準偏差値の元となる平均値を関連付けせずに評価することは難しいので平均コンテナ読み込み回数に対する標準偏差の割合を用いた．この場合においても拡張 chunk を用いた場合は安定して低い(振れ幅の少ない)値になる．0.01%の時の値が上昇しているが，極端に低い充填率であるために，総コンテナ数も極端に少ない状況での偏差であり，重大な欠点ではないと考えられる．

また，ドメイン長 7 の d-order に関しては chunk の平均充填率が増加すればするほど標準偏差の割合は増加している．これは広いドメインがあだとなり多次元配列内において論理的に特定次元に伸びたコンテナが生成されやすくなったため次元依存性が発生したのだと思われる．

上記 3 つの測定をまとめると拡張 chunk は安定してよい結果が出ていると思われる．一方 rect は反対に，その長所，短所が極端に表れている．d-order 二種に関してはドメイン長 3 の場合は読み込み回数に対する標準偏差の割合は良好であるが，総コンテナ数が多いため記憶領域を多く必要とする，ドメイン長 7 の場合は最適な充填率においては優れているが充填率が増加するに従って平均読み込み回数に対する標準偏差の割合が増加している．

4.2.2 次元長を変化させた時

一つの次元方向に 2~32 個の chunk を持つ 5 次元配列で，その 5 次元目に正規分布の形となるように各 chunk に充填率を与え実験を行った．測定結果を図 5 に示す．多次元配列のサイズが小さい間は振動が見られるが，拡張 chunk による場合には，次元長 10 のサイズあたりからは安定して全

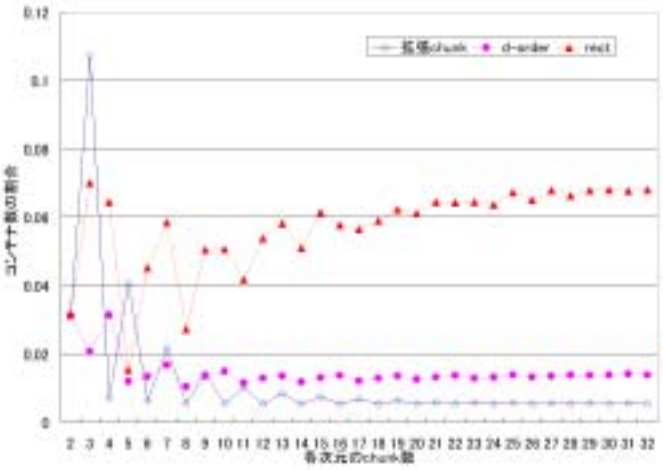

図 5 次元長別全 chunk に対する総コンテナ数の割合

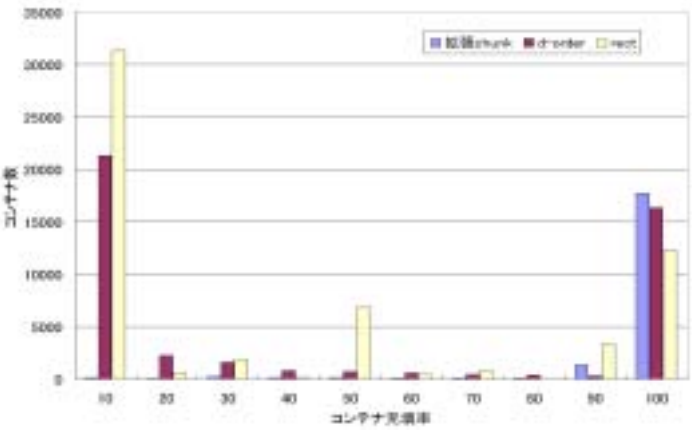

図６　コンテナ充填率別コンテナ数

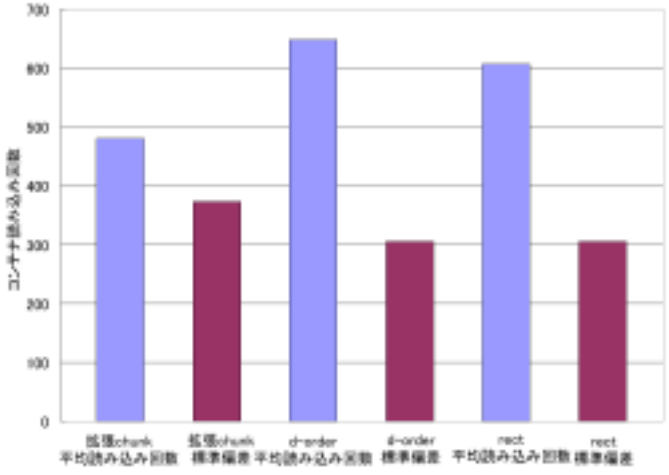

図 7　3 次元スライス時における
平均読み込み回数と標準偏差

chunk 数の約 1%の量のコンテナ数で全領域をカバーできている．一方 d-order は約 2%，rect は

3~6%ほどである．このことから拡張 chunk を用いた場合の方が他の方式より総コンテナ数を減少させることができるのではないかと考えられる．振動部に関して次元長が偶数時は谷になり，奇数時は山になっており，rect も似通った形を取っている．拡張 chunk については各次元の長さが 2^n となるように決定しているため chunk 単位とした多次元配列の各次元の長さが奇数になると多次元配列の端の部分に存在する chunk は rank 値 0 の拡張 chunk となる．多次元配列が小さければ，この rank 値 0 の拡張 chunk が割合として大きくなるためであると考えられる．

次に 1 つの次元方向に 18 個の chunk を持つ 5 次元多次元配列で，5 次元目に正規分布の形をとるように chunk に充填率を与えた場合にコンテナ化を行い，その結果のコンテナ内充填率（二次記憶における 1 ページに対する有効要素数の割合）別コンテナ数と 3 次元スライス時における平均コンテナ読み込み回数と次元依存性により生じるそのばらつきを示す標準偏差を測定する．

コンテナ充填率別コンテナ数のグラフを図 6 に示す．図.6 はコンテナ充填率が 10%単位で棒グラフが形成されている．図 6 に示した充填率別コンテナ数のグラフから見て取れるのは d-order, rect ともに 10%の棒グラフが最も高い数値を示している．これは 3 節で考察したようにドメインの影響と急激なコンテナ対象領域の増加の影響が出ているのではないかと考えられる．それに対し，拡張 chunk は多少低充填率コンテナが残るが多くは充填率 80%を越えるコンテナになっているのがわかる．

平均読み込み回数と標準偏差のグラフを図 7 に示す．拡張 chunk は平均読み込み回数においては最も優れているが標準偏差においては最も劣っていることがわかれる．この理由として考えられるのは chunk の充填率の分布に正規分布を取ったことにあると考えられる．拡張 chunk 形成時において正規分布の頂点部分の chunk 充填率が高く壁を作るが如くに多次元配列を半分にわけてしまい，その頂点部分の拡張 chunk が全て rank 値 0 になり，逆にそれ以外の場所の拡張 chunk は rank 値 1~2 を形成し，さらにその rank 値 1~2 の拡張 chunk は互いに近傍にあるため結合時に同一コンテナに格納される．また，rank 値 0 同士の chunk が集まり，コンテナを形成する．したがって，1 つのコンテナがカバーしている論理的な領域が大きいものと小さいものの両極端に偏ってしまい，かつその小さい領域が同一スライス上に乗っているので平均読み込み回数が減少しているにもかかわらずスライス時の最大読み込み回数が減少していないのではないかと思われる．これを解消するには結合時において有効要素数のみではなく，隣接拡張 chunk の rank 値のチェックを行う必要があると考えられる．

6.むすび

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った．多次元配列データベースで一般的である低充填率状態の多次元配列に注目したコンテナ化法である拡張 chunk に関して，一部不利な状況も見られたが d-order, rect に対して有利な点も多く見られた．今後の課題として次元依存性のさらなる解消，さらに，コンテナ化を行った後，二次記憶に格納するコスト面にも着目しなければならないと思われる．

文　献

[1] M.Abbey, I.Abramson, and L.Bames, SQL Server 7 Data Warehousing, McGraw-Hill, 1999

[2] J.Gray, A.Bosworth, A.Layman, and H.Pirahesh,”Data cube:A relational aggregation operator generalizing group-by, cross-tab, and sub-totals, ”J. Data Mining and Knowledge Discovery, vol.1,no.1, pp29-53, 1997

[3] V.Harinarayan, A.Rajaraman, and J.D. Ullman,”Implementing data cubes efficienty, ”Proc. ACM SIGMOD Conf. on Management of Data, pp.205-216,1999

[4] E.F.Codd, S.B.Codd, and C.T.Salley,“Beyond decision support,” Computerworld, vol27, no.30, pp.87-89,1993

[5] Heum-Geun Kang, Chin-Wan Chung, ”Exploiting Versions for On-line Data Warehouse Maintenance in MOLAP

Servers", Proc. of
VLDB 2002, pp.742-753, 2002.
[6] 都司達夫，一色淳夫，樋口　健，宝珍輝尚，“MOLAPのための多次元配列の実現方式とその性能評価”電子情報通信学会 vol.J87-D-I no.2, 2004
[7] K.E. Seamons and M. Winslett, “Physical schemas for large multidimensional array-s in scientific computing applications,”Proc. Scientific and Statistical Database Manage-ment, pp.218-227,1994
[8] S.Sarawagi and M. StoneBraker, “Efficie-nt organization of large multidimensional arrays,” Pro. ICDE, pp.328-336, 1994
[9] Y.Zhao, P.M. Deshpande, and J.F.Naughton, “Array-based algorithm for simultaneous multidimensional aggregates,”Proc. ACM SIGMOD Conf. on Management of Data, pp.159-170, 1997