タプルストリームからのデータキューブの構築

嶋田 鉄兵† 佐々原 秀男† 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

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

あらまし 本論文ではネットワーク上の関連サイトで MOLAP 用のデータキューブを分散構築するための方式を提案する.データ発生源から送信されるタプルデータを直接受信するサーバはベースキューボイドを構築しながら，それをタプルストリームとして下位のサイトに送信する．下位のサイトは受信したタプルデータを集約して，上位のキューボイドより，1 次元低いキューボイドを構築しながら，それをタプルストリームとしてさらに下位のサイトに送信する．本方式では，筆者等が提案している多次元データのエンコード法である経歴・オフセット法による実装方式を用いることにより，効率よくタプルストリームを処理し，各サイトにおいて，リアルタイムでキューボイドを構築する．本文では本方式におけるタプルストリームの処理方法，およびデータキューブの分散構築について述べたあとで通信量を評価する.

キーワード タプルストリーム，データキューブ，タプルエンコーディング，集約演算

Distributed Construction for Data Cube from Tuple Stream

Teppei SHIMADA† Hideo SASAHARA† Tatsuo TSUJI† and Ken HIGUCHI†

†Graduate School of Engineering, University of Fukui 3-9-1 Bunkyo, Fukui-shi, Fukui, 910-8507 Japan

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

Abstract We propose a distributed construction scheme for MOLAP data cube in related sites on the network. The server that directly receives tuple data sent from data generation source constructs the base cuboid, and sends them to downstream sites as a tuple stream. The downstream site receives the tuple data and aggregates them, and constructs its own cuboid that is one dimension lower than the upstream cuboid, then sends the cuboid data to downstream sites. In our scheme, using the implementation scheme of multidimensional datasets based on the history-offset encoding we are proposing, the tuple stream can be processed efficiently to construct cuboids in real-time on each site. In this paper, we describe our tuple stream processing scheme and distributed data cube construction, then evaluate the required communication cost.

Keyword tuple stream, data cube, tuple encoding, aggregation

1. はじめに

複数の単純型データのタプルからなるデータを一般的に多次元データという．ファクトデータが付随する n 次元データ集合 S の任意の次元の組み合わせについて，ファクトデータを集約して得られる 2^n-1 個の多次元データセットの集合を S_{dc} とすると，S をベースキューボイド，$s \in S_{dc}$ を S をベースキューボイドとするキューボイドという（図 1）．また，ベースキューボイドからデータキューブを構築する演算をデータキューブ演算[1]という．データキューブは多次元分析や多次元データマイニングなどのデータウェアハウスにおける主要な機能において，ベースキューボイドデータを種々の視点から分析・マイニングするために不可欠である．しかし，データキューブ演算は計算インテンシブ・データインテンシブな演算であるため，これらの機能をオンラインでサポートするために，通常は事前計算によってデータキューブが構築される．

本論文では，筆者等が提案している多次元データのエンコード法である経歴・オフセット法による実装方式を用いることにより，ネットワーク上のノード集合において，効率よくタプルストリームを処理し，リアルタイムにキューボイドを構築する方式を提案する．まず，経歴・オフセット法による多次元データの実装方式である HOMD[9][10]を用いて，多次元データをエンコードした後，タプルストリームとして転送する方式を提案する．多次元データを圧縮エンコードすることにより，データストリームとして，効率よく構造転送を行う．即時性の要求を満たすために，データを高速に送信し，受信側で，それを効率よくデコードし，集約演算を行う．次に，この方式に基づいて，ネットワーク上のノード集合におけるデータキューブのキューボイドをメンテナンスする方式を提案する．本方式

では，タプルストリームから受信したデータをリアルタイムに当該キューボイドに反映しながら，バックグラウンドでは，ネットワーク上のクライアントノードキューボイドの MOLAP 操作要求に応えることができる．

本文では本方式におけるタプルストリームの処理を述べた後，データキューブの分散構築について述べ，さらに必要な通信量を評価する．

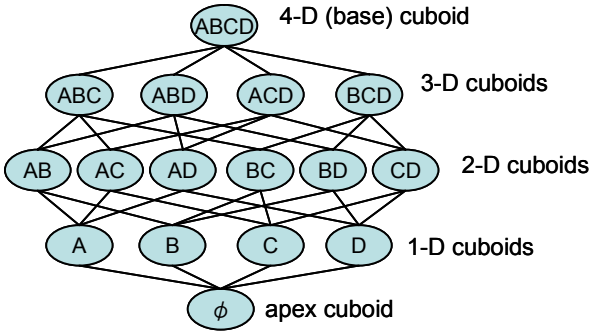

図 1　4 次元データキューブ

2. 経歴･オフセット法

ここでは，多次元データの代表例として関係テーブルを取り上げ，筆者等が提案している多次元データのエンコード方式である経歴･オフセット法について概説する．詳しくは[9][10]を参照されたい．

2.1. 拡張可能配列

プログラミング言語で用いられる従来の固定サイズ配列は，実行時に動的に配列の各次元サイズの変更を行うことができない．これに対して，拡張可能配列[7][8]では，配列サイズの動的変更に対して，すでに確保している領域はそのまま使用し，新たな領域を差分領域として，必要な分だけ付加することができる（図 2）．以下，メモリの動的割り付け環境に適合した拡張可能配列の実現モデル[8]に基づいて述べる．

一般に n 次元拡張可能配列は，n-1 次元の固定サイズの部分配列の集合で表される．拡張可能配列では，各部分配列の先頭アドレスおよび，部分配列が何番目の拡張の際に確保されたかを表す経歴値をそれぞれ，アドレステーブル（図 2 では L_1，L_2），経歴値テーブル（H_1，H_2）に格納している．また，部分配列の要素へ高速にアクセスできるように，n が 3 以上の場合には，各部分配列のアドレス関数を計算するための n-2 個の係数値を係数ベクトルとして，各部分配列ごとに係数テーブルへ記録している．n が 2 以下の場合には係数テーブルを持つ必要がない．2 次元拡張可能配列の場合，部分配列は 1 次元であり，座標 (i_1, i_2) の部分配列内オフセットは添字 i_1 または i_2 である．図 2 では，2 次元拡張可能配列について第 1 次元方向にサイズを一つ拡張している．まず，第 2 次元方向のサイズが 3 であるので，サイズ 3 の 1 次元部分配列を確保する．次に，現在の経歴値を記録している経歴値カウンタを 1 つカウントアップして，新たに確保した部分配列の経歴値とする．また，新たに確保した部分配列の先頭アドレスを拡張次元方向のアドレステーブルに記録する．

要素 $(i_1, i_2, \ldots, i_n)$ が与えられたとき，各次元の添字に対応する経歴値のうち，最大経歴値の次元を m とすると，次元 m のアドレステーブルの i_m番目の先頭アドレスの部分配列が，与えられた要素を含む．その部分配列の係数ベクトルを知って，要素のアドレスが計算できる．例えば，図 2 の (3, 2) 要素の場合，$H_1[3] > H_2[2]$ であるから，この要素は経歴値 5 の部分配列に含まれ，その先頭番地は $L_1[3]$である．したがって，求めるアドレスは $L_1[3]+2=78$ と計算される．このように補助テーブルのわずかな記憶コストの増加のみで，シンプルな方法で動的に拡張が可能な配列の要素のアドレス計算が可能であることは注目に値する．

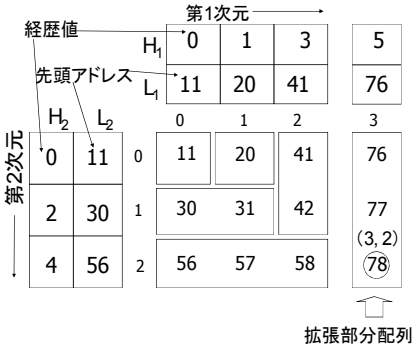

図 2　拡張可能配列

2.2. 拡張可能配列を用いた多次元データの表現

関係テーブルの各属性を従来の多次元配列の各次元に，属性値を配列の添字に対応付けることによって，関係テーブルのタプルを，配列要素の位置で表現する

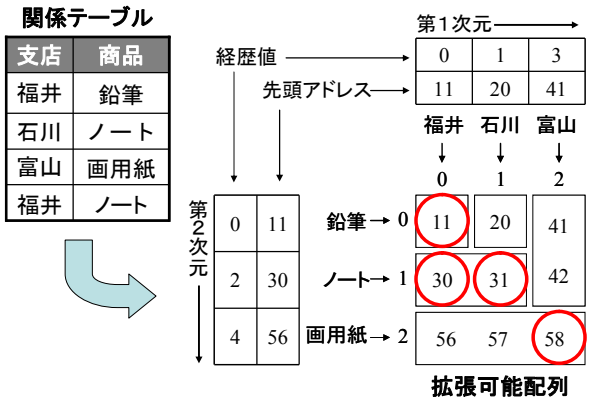

図 3　拡張可能配列を用いた関係テーブルの表現

ことができる．2.1 節で述べた拡張可能配列を用いれば，関係テーブルに対して，動的なタプルの追加に対応することができる．図 3 に，拡張可能配列を用いた関係テーブルの表現例を示す．

2.3. 経歴･オフセット法を用いた多次元データの実装

2.2 節で述べたように，多次元配列を用いて関係テーブルを表現することは可能であるが，通常，関係テーブルのタプルに対応しない配列要素が多く存在してしまうため，多数の使用しない配列要素をメモリ上に確保しなければならないという過疎性の問題がある．そこで，存在するタプルに対応しない無駄な配列要素を保持しないために，有効な配列要素が属する部分配列の経歴値とその部分配列内のオフセットを配列要素の位置情報として保持する．この位置情報は対応するタプルそのものを表すことになる．このような多次元データセットのエンコーディングの方法を，経歴･オフセット法 (history-offset encoding) と呼ぶ．この方法では，配列の次元数 *n* に依存することなく，小さな固定長で *n* 次元タプルを表現することができる．図 4 に，経歴･オフセット法を用いた図 3 の関係テーブルの実装を示す．

図 4 では，関係テーブルの属性値から配列の添字に高速に変換できるよう，配列の各次元において添字変換用の B^+木（CVT : key-subscript ConVersion Tree と呼ぶ）を利用している．また，経歴値とオフセットの対を B^+木（RDT : Real Data Tree と呼ぶ）を用いて管理している．このように，経歴･オフセット法を用いて多次元データを実装するための方式および，そのデータ構造を HOMD (History-Offset implementation for Multidimensional Datasets) と呼んでいる．

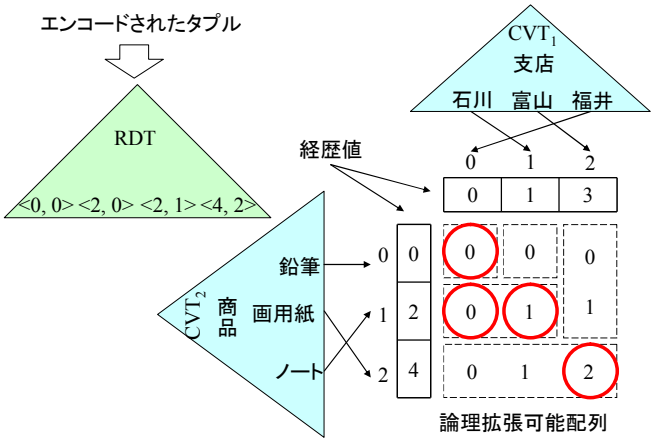

図 4　HOMD による関係テーブルの実装例

3. タプルストリームの処理

本節では，前節の経歴・オフセット法を用いたタプルストリームの処理方式を説明する．

3.1. 概要

本方式では，データ発生源から受信したタプルストリームを取り込んでデータキューブを形成する．データキューブのキューボイド集合はネットワーク上の複数のノードにおいて，分散して形成され，保守される．キューボイドは 2.3 節で述べた HOMD 構造によって構成される．ベースキューボイドを保守するサーバノードは，データ発生源から受信する n 次元データを集約し，常に最新のベースキューボイドを維持し続けると同時に，データ発生源から受信した n 次元タプルを＜経歴値，オフセット＞対にエンコードした後，受信したファクトデータとともに，n-1 次元キューボイドを保守する下位のサーバノードへそのまま即時に送信する．データを受信した下位のノードは受け取った＜経歴値，オフセット＞対から n 次元タプルにデコードし，当該の n-1 次元のタプルにエンコードした後，それを自身の HOMD 構造に集約すると同時に，受信したファクトデータとともにさらに下位のノードへそのまま即時に送信する．

次節で説明するように，タプルやファクトデータ以外にも，下位のノードが保守するキューボイドのための HOMD 構造の構成に必要なデータを必要に応じて送信する．下位のノードではこれらのデータを受信し，自身のキューボイドを最新の構造に更新する．タプルデータの＜経歴値，オフセット＞対へのエンコードや，逆にタプルデータへのデコードは，この HOMD 構造によるキューボイドを参照することにより行われる．

3.2. 送信データ

送信ノードから受信ノードへ送信するデータとして，以下の 5 種類のデータを定義する．データを送信するに当たり，通信コストを下げるため，最低限のデータのみを送信し，受信側で，その情報にしたがって，HOMD 構造を保守する．⑤については，以下の 4 節で述べる．

① メタ情報：タプルを定義するスキーマ情報．最初に一度だけ送信する．
② デコードデータ：属性値とその次元番号の組．新たな属性値が出現した場合のみ送信する．
③ タプルデータ：経歴・オフセット法によりエンコードされた＜経歴値，オフセット＞対．
④ ファクトデータ：タプルに付随する集約対象となるデータ．タプルデータと同時に送信する．
⑤ 確定フラグ：当該キューボイドが確定済みであることを示すフラグデータ．このフラグデータを受

け取った，下位のキューボイドノードは自身が管理する当該のキューボイドを確定させた後，下位のキューボイドノードにさらに確定フラグを送信する必要がある．

以下に，送信パケットの例を示す．なお，図は連続した 1 つのパケットを表す．

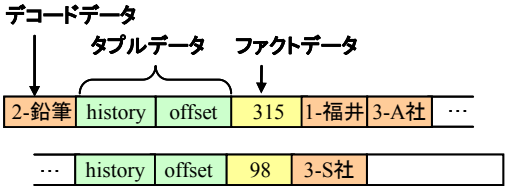

図 5　送信パケットの例

3.3. 送信処理

送信ノードのうち，データ発生源からデータを受信する最上位のノードでは，以下のようにベースキューボイドを構築し，必要データを下位ノードに送信する．

1. メタ情報（①）により，多次元データセット（テーブル）を定義する．また，送信パケットにメタ情報を詰め込む．
2. データ発生源から受信したタプルデータをエンコード（③）して HOMD 構造に登録する．既登録ならば，RDT のデータ部の集約値とタプルに付随するファクトデータ（④）を集約してデータ部の集約値を更新する．このとき，タプルデータの各属性値が当該 CVT に登録されていなければ，拡張可能配列の当該次元を 1 つ拡張する．
3. 2 で挿入したタプルから，新しく出現した属性値とその次元（② (デコードデータ)），および下位のノードに送信する 1 次元少ないタプルをエンコードした結果の＜経歴値，オフセット＞対（③）と付随するファクトデータ（④）を送信パケットに挿入する．
4. パケットが満杯あるいはそれに近い状態になった場合，下位ノードへ送信する．以下，手順 2～4 を繰り返す．

最上位ノード以外のノードの送信処理については，次節で述べる．

3.4. 受信･集約処理

以下，送信ノードのキューボイドの次元を n 次元，受信ノードのキューボイドの次元を n-1 次元とする．

1. 最初にメタ情報を含むパケットを受信し，そのメタ情報から HOMD を定義する．ここでは，受信データ用の n 次元 HOMD（Hn）と集約用の n-1 次元 HOMD（Hn-1）を定義する．また，下位ノード用の送信パケットに Hn-1 のメタ情報を送信パケットに詰める．
2. n 次元データからなるパケットを受信する（HOMD がすでにある場合）．
3. 受信パケットの次のデータがデコードデータであれば，その次元における属性値を Hn の当該次元の CVT に挿入する．また，下位のノード中の Hn-1 のためのデコードデータを形成し，送信パケットへ詰める．
4. 受信パケットの次のデータが n 次元タプルデータ（＜経歴値，オフセット＞対）とそれに付随するファクトデータであれば，以下の処理を行う．タプルデータとファクトデータを Hn-1 の RDT に挿入あるいは集約する．受信パケット中の n 次元タプルデータのオフセットは受信ノードで再計算され，n-1 次元のオフセットに変更される．また，この n-1 次元のオフセットが算出された時点で，経歴値，ファクトデータと共に送信パケットに詰める．
5. 送信パケットが満杯あるいはそれに近い状態になった場合，n-2 次元のキューボイドを保守する下位ノードへ送信する．以下，2～5 を繰り返す．

Hn の CVT および HOMD テーブルは受信した n 次元タプルデータのデコードのためのデータ構造として維持され，それらは Hn-1 において共有される．RDT については，Hn-1 についてのみ保持する．

図 6 に受信・集約処理の例を示す．表 1 は支店・商品・メーカの 3 つの属性，およびファクトデータ（売上額）をもつ．例では，3 次元ベースキューボイドを保守する最上位ノードからデータを受信し，支店・商品の 2 次元キューボイドを生成する．図 6 において，実線部が実際に個々の HOMD で保持する部分であり，破線部は保持しない．下位ノードの受信側ではまず，初回にテーブル定義用のメタ情報を受信し，デコード用 HOMD（上位と同じ 3 次元 HOMD （図 6 左)），および集約用の 2 次元 HOMD （図 6 右）を定義する．その後，上位ノードから送信されたパケットを受信し，デコードデータおよびタプルデータ，ファクトデータを取り出す．ここで，デコードデータはデコード用 HOMD の CVT に挿入するが，タプルデータ・ファクトデータはデコード用 HOMD では保持しない．タプルデータ＜経歴値，オフセット＞対のうちオフセットは，支店，商品の 2 次元の組に対するオフセットに変換し，受信した経歴値・ファクトデータとともに集約用 HOMD の RDT に挿入・集約する．

表 1　支店別の売り上げ情報（例）

支店	商品	メーカー	売上額(円／日)
福井	鉛筆	A社	315
石川	ノート	B社	1580
富山	画用紙	C社	840
福井	鉛筆	B社	735

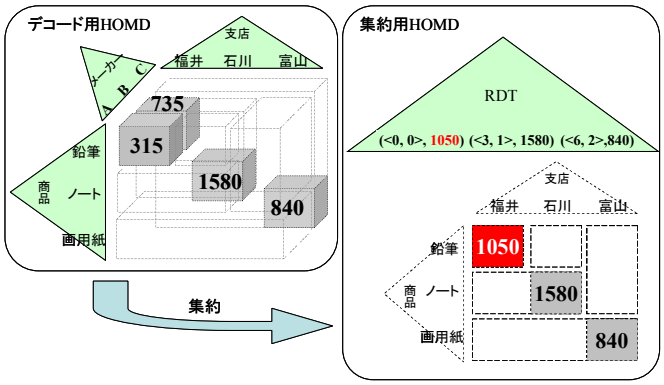

図 6　受信・集約処理

4. データキューブの確定処理

一般に，オンライン分析のためのデータは1日ごとや1週間ごとなど定期的にフロントエンドのOLTP DBシステムからデータがまとめて送信され，MOLAP処理に利用される．そこでは，OLTP DBのテーブル状態をリアルタイムに反映する必要はないが，各キューボイドのメンテナンスを担当するノードでは，データを適切なタイミングで確定し，分散したキューボイド集合が全体として，一貫性のあるデータキューブを提供できる必要がある．

本方式では，タプルストリームから受信したデータをリアルタイムに当該キューボイドに反映しながら，バックグラウンドでは，ネットワーク上のクライアントノードのMOLAP操作要求に応えることができる．

図7に本方式における各ノードのタプルストリーム処理の流れを示す．実時間で発生するデータ発生源からのデータはベースキューボイドノードが直接受信し，遅延なしに実時間で集約する．前節で述べたように，図7において上位ノードからのタプルを集約するノードは，3.2節の①~④の4種類のデータを上位ノードから受信し，自身のキューボイドを最新データで更新するリフレッシュ動作を行う．リフレッシュ動作の後，ノードは必要なデータをパケットに詰め，満杯になれば時間遅延なしに下位ノードへ送信する．これらの動作は他のキューボイドを保守するノードと同期することなく非同期に行われる．一方で，ネットワーク上の関連ノードに分散して構築されるキューボイド集合をデータキューブとして矛盾のない一貫した状態で使用できることを保障する必要がある．ここでは，次のような方式を採用する．

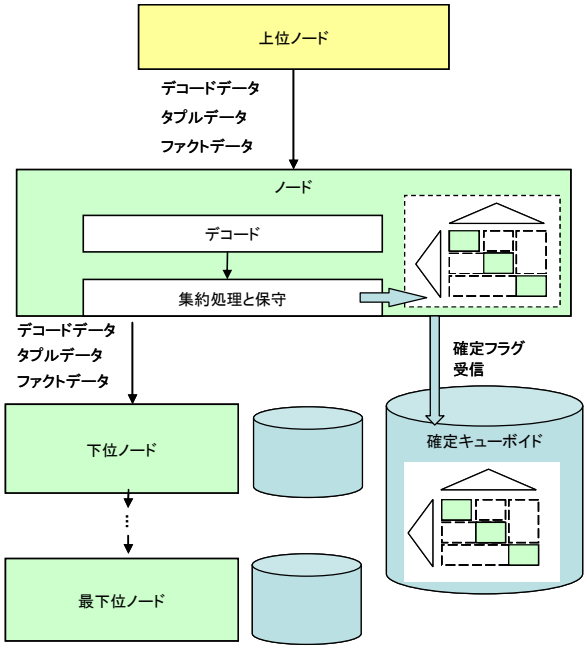

図7　タプルストリーム処理の流れ

上述の定期的なデータキューブの確定時には，ベースキューボイドノードが送信パケットデータの最後尾に確定フラグ（3.2節の⑤）を付与して，下位のノードに送信すると同時に，その時点での当該HOMD構造のコピーをバルクコピーにて，バックグラウンドで2次記憶上に作成する．この確定フラグは最下位キューボイド (apex cuboid) まで，引き継がれて送信されるが，他のキューボイドノードも同様に，確定フラグを受信したならば，当該HOMD構造のコピーを2次記憶上に作成する．各ノードにおける確定時のこれらのキューボイドのコピーの各々は，日付をIDとするバージョン番号で管理される．キューボイドノードが確定フラグを受け取って，当該キューボイドを確定しても，下位のキューボイドに必要データを送信する必要がないときには，最下位キューボイドに直接確定フラグを送信する．

キューボイドノードは確定フラグによって当該バージョンのキューボイドを確定した後，ネットワーク上のクライアントノードからの当該キューボイドに対するMOLAP操作要求を処理して結果をクライアントに送信できる．確定した時点では，ベースキューボイドからこの確定キューボイドに至るパス上の同じバージョンのキューボイドは確定しているので，同様にMOLAP操作が可能である．しかし，他のパス上の同じバージョンのキューボイドは確定しているとは限らな

い．だが，クライアントに対しては，データキューブ上でドリルダウンやロールアップの処理を繰り返して任意のキューボイド上で分析処理を可能にする必要がある．ここでは一貫した状態のデータキューブのバージョンを保証するために,次のような方策を採用する．

最下位ノードはすべての関連キューボイドから確定フラグを受け取った時点で，データキューブ全体が確定するので，そのバージョン番号を自身に登録して管理する．クライアントは，特定のキューボイドにアクセスしたいときには，直接，当該のキューボイドノードにバージョン番号を添えて処理要求を送信する．受け取ったノードはバージョン番号のキューボイドが確定済みであれば,要求を処理して,結果を返信する．未確定であれば，クライアントの指定に応じて，エラーを返す(非同期処理)か，確定後に要求を処理して，結果を返信する(同期処理)．他方，すべてのキューボイドが確定済みであることを確認してからデータキューブに対して，MOLAP操作を行いたいときには，当該バージョンのデータキューブが確定済みかどうかの問い合わせを最下位ノードに発行して，確定していることを確認してから，個々のキューボイドノードに操作要求を発行する．

最下位ノードの負荷は最も少ないので，ここでは最下位ノードが確定データキューブのバージョン情報を管理している．また，同時に，各キューボイドのノードへの割り当てなどのメタ情報も管理しており，クライアントの求めに応じて，アクセスしたいキューボイドがどのノードに割り当てられているかを通知する．

上位ノードからの3.2節①～④のデータの受信と処理をフォアグラウンドでリアルタイムに行いながら，これらのMOLAP操作要求はバックグラウンドで処理される．フォアグラウンド処理とバックグラウンド処理を並列に行うことにより，フォアグラウンド処理の実時間性を犠牲にしないために，ノードシステムはマルチコアシステムを想定する．

5. 評価

キューボイド間の通信データ量について解析評価を行う．ここでは，以下の2方式に対して比較評価を実施する．

- 経歴・オフセット法によりエンコードしてタプルを送信する方式（提案方式）
- タプルをエンコードせずにそのまま送信する方式（非エンコード方式）

5.1. 評価条件

以下の条件で評価を行う．

- タプル総数：1,000,000 タプル
- 属性数（次元数）：5
- パケットサイズ：8192 [byte]
- ファクトデータのサイズ：4 [byte]（int型）

実験では，ベースキューボイドと4次元キューボイドの間の通信量を比較する．実験内容は次の通りである．

(1) タプルストリームにおける各属性の属性値の重複率を，0～95[％]まで5[％]刻み，および99[%]として変化させた場合の，各方式の送信パケット総数を計算して比較する．ここで，属性値の平均サイズ（平均属性値長）は4, 16 [byte]の2種類とし，この2つの場合について評価を行う．

(2) 各属性の平均属性値長に対し，提案方式と非エンコードのパケット総数が等しくなる場合の属性値の重複率（以下，平衡重複率という）を計算する．平均属性値長は1,4,8,10,16 [byte]，および20～100[byte]まで5[byte]刻みで変化させる．

5.2. 評価結果

(1)に対する評価結果を5.2.1節，(2)に対する結果を5.2.2節に示す．

5.2.1. 重複率に対するパケット総数

図8に結果を示す．平均属性値長 l に関わらず，非エンコード方式ではすべての重複率に対して一定の送信パケット総数である．一方，提案方式では，属性値の重複率が上がるほどパケット総数が減少する．これは，提案方式において属性値が重複した場合には，その分をデコードデータから省くことができるため，送信するデータ量が減少するからである．重複率とパケット数について注目すると，重複率が高いほど，デコードデータの量を抑えることのできる提案方式のほうが非エンコード方式よりも少ないパケット総数で通信が行えることがわかる．ただし，提案方式のパケット総数が非エンコード方式のパケット総数を下回る重複率は，属性値のサイズによって異なっている．

平均属性値長l = 4[byte]の場合，提案方式のパケット総数は，属性値の重複率が68.1[%]を越えるときに，非エンコード方式のパケット総数を下回る．一方，l = 16[byte]の場合では，提案方式のパケット総数は，重複率が19.6[%]を越えるときに，非エンコード方式のパケット総数を下回る．つまり，属性値のサイズが大きいほど，より小さな属性値の重複率で，提案方式が非エンコード方式よりも少ないパケット量で通信を行えることがいえる．

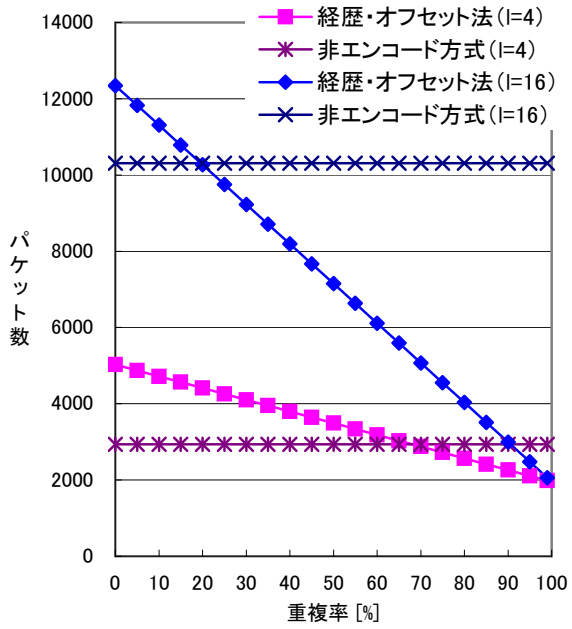

図8　重複率に対する通信量の変化

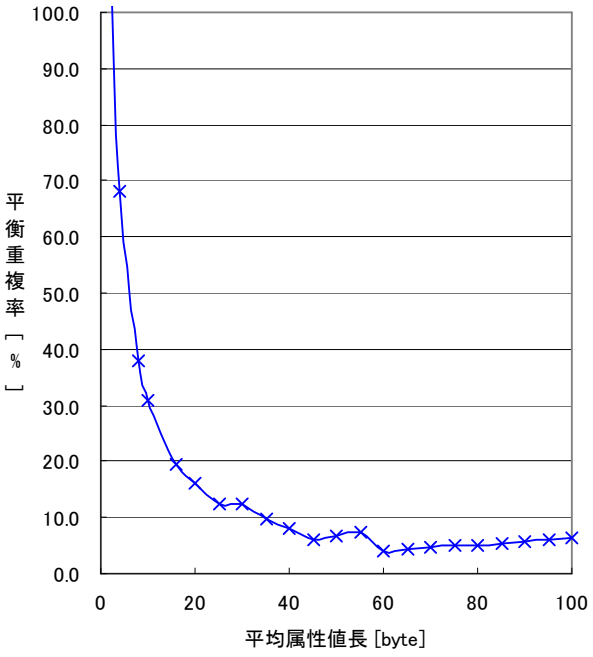

図 9　平均属性値長に対する平衡重複率の変化

5.2.2. 平均属性値長に対する平衡重複率

図9に結果を示す．結果において，重複率1%の場合は平衡重複率が100[%]を超えている．この場合は，経歴・オフセット法を用いた提案方式のパケット総数が非エンコード方式のパケット総数を下回ることがないことを示している．それ以外は100[%]以内に収まっており，いずれかの重複率で，提案方式のパケット総数が非エンコード方式のパケット総数を下回る.図より，平均属性値長が大きいほど，両方式のパケット総数が等しくなる重複率(平衡重複率)が低いことがわかる．特に，平均属性値長が1～20[byte]の範囲では，属性値長の変化に対して平衡重複率の変化が急激であることがわかる．

これらの結果から以下のようにまとめられる．提案した経歴・オフセット法を用いたストリーム処理方式は，属性値の重複がある場合にデコードデータのサイズを抑えることができるため通信量を抑制することができる．また提案方式は，各属性の属性値のサイズが大きく，かつタプルストリームにおける属性値の重複率が高いほど，非エンコード方式に対して通信にかかるコストをより抑えることができると考えられる．

6. 関連研究

事前計算の高速化を図るために，主に 2 つの方式が検討されている．一つはデータキューブの並列構築方式（例えば[2][3]）であり，もう一つは差分構築方式（例えば[4][11]）である．後者は，スクラッチからデータキューブを構築することを回避するためにオリジナルのデータキューブを直前の期間に構築される差分データキューブでリフレッシュすることによって，最新のデータキューブを高速に構築する．Relational OLAP（ROLAP）のための差分構築の研究はあるが，多次元配列を用いた Multidimensional OLAP（MOLAP）のための研究は，筆者等の知る限りでは[11]のみである．これは従来のように固定サイズの多次元配列を用いる場合には，効率良い差分構築が困難であるためであると考えられる，[11]では動的に増大する多次元データの効率良い実装方式のために経歴・オフセット法と呼ばれている多次元データのエンコード方式を採用して，差分構築に対応している．

データキューブをネットワーク上の関連ノード集合において分散構築するための研究も見られる．[5]ではネットワーク上で格子状に配置されたセンサノード集合において，Prefix SUM による集約データからなる分散データキューブを構築している．また，[6]では同様に Prefix SUM による分散データキューブの集約値の更新プロトコルを提案している．[11]では MOLAP のための多次元配列を使用したデータキューブの差分構築が扱われているが，分散構築は扱っておらず，タプルストリームからの構築である本方式とは異なる．

7. まとめ

本論文ではネットワーク上の関連サイトでMOLAP用のデータキューブを分散構築するための方式を提案し，通信量を評価した．本方式では，上位ノードからのストリームデータの受信と最新キューボイドの保守

をリアルタイムで行いながら，確定したデータキューブに対するMOLAP操作要求はバックグラウンドで処理される．

参 考 文 献

[1] Jim Gray, Surajit Chaudhuri, Adam Bosworth, et al., Data Cube: A Relational Aggregation Operator, Generalizing Group-By, Cross-Tab, and Sub-Totals, Journal of Data Mining and Knowledge Discover, 1, pp.29-53, 1997.

[2] Ying Chen, Frank Dehne, Todd Eavis, A. R. Chaplin, Parallel ROLAP Data Cube Construction on Shared-Nothing Multiprocessors, Distributed and parallel Databases, Vol. 15, pp.219-236, 2004.

[3] S. Muto, M. Kitsuregawa, A Dynamic Load Balancing Strategy for Parallel Datacube Computation, Proc. of DOLAP 1999, pp. 67-72, 1999.

[4] K. Y. Lee, K. M. H. Kim, Efficient Incremental Maintenance of Data Cubes, Proc. of VLDB2006, pp.823-833, 2006.

[5] Lok Hang Lee, Man Hon Wong, Aggregate Sum Retrieval in Sensor Network by Distributed Prefix Sum Data Cube, Proceedings of the 19th International Conference on Advanced Information Networking and Applications, vol.1, pp. 331- 336.

[6] Dan Wu, Chi Hong Cheong, Man Hon Wong Supporting asynchronous update for distributed data cubes, Journal of Network and Computer Applications, 32, pp.889-900, 2009.

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

[8] 都司達夫，水野剛，宝珍輝尚，樋口健，拡張可能配列の遅延割付け方式，電子情報通信学会論文誌 D-I, Vol.J86-D-I, No.5, pp.351-356, 2003.

[9] K. M. A. Hasan, M. Kuroda, N. Azuma, T. Tsuji, K. Higuchi, An Extendible Array Based Implementation of Relational Tables for Multi Dimensional Databases, Proc. of DaWaK 2005, pp. 233-242, 2005.

[10] T. Tsuji, M. Kuroda, K. Higuchi, History offset implementation scheme for large scale multidimensional data sets, Proc. of ACM Symposium on Applied computing, pp.1021-1028, 2008.

[11] J. Dong, T. Tsuji, K. Higuchi, An Incremental Maintenance Scheme of Data Cubes and Its Evaluation, IPSJ Online Transaction, Vol.2, pp.36-48, 2009.