ストリーム処理エンジンを用いたストリームデータのOLAP処理

中挾　晃介†　　北川　博之††　　天笠　俊之††

† 筑波大学システム情報工学研究科コンピュータサイエンス専攻　〒305–8573 茨城県つくば市天王台 1-1-1
†† 筑波大学システム情報系情報工学域　〒305–8573 茨城県つくば市天王台 1-1-1
E-mail: †nakabasami@kde.cs.tsukuba.ac.jp, ††{kitagawa,amagasa}@cs.tsukuba.ac.jp

あらまし　センサデータやログデータやマイクロブログ等，連続的に生成，配信されるストリームデータの増加に伴い，その処理を行うストリーム処理エンジンが開発されている．ストリームデータに対する分析としては様々なものがあるが，その一つに多次元分析に基づく OLAP 処理がある．これまでストリームデータの OLAP 処理については J. Han らの研究があるが，ストリーム処理エンジンの利用は想定されていない．このため，OLAP 処理のために個別のシステムを開発する必要があったり，他のストリームデータ応用との連携が十分にとれないなどの問題がある．本研究では，ストリーム処理エンジンが持つ集約計算機能を用いてストリームデータに対する OLAP 処理を実現するための手法とその処理の効率化手法について検討する．

キーワード　ストリーム処理エンジン，OLAP 処理

1. はじめに

近年，センサデータやログデータやマイクロブログ等，際限なく，連続的に生成され，配信されるようなストリームデータが増加してきている．このストリームデータを処理するストリーム処理エンジンが多数開発されている．ストリーム処理エンジンは，その専用問合せ言語で記述された問合せをエンジンに登録することにより，ストリームデータに対して登録した問合せを連続的に実行できる．一方，ストリームデータに対してより高いレベルの分析 (データの傾向や外れ値など) を行いたいというニーズが増加している．このような分析の代表例として，データを多次元のデータキューブとして捉えて分析を行う多次元データ分析 (OLAP 処理) がある．ストリームデータに対して OLAP 処理を行う研究として，Jaiwei Han らはストリームデータに対する OLAP 処理を容易にすることを目的とした Stream Cube と呼ばれるアーキテクチャを提案している [1]．ただし，ストリームデータを OLAP 処理することは可能となっているものの，ストリーム処理エンジンを生かした OLAP 処理については十分検討が行われていない．このため，OLAP 処理のために個別のシステムを一から開発する必要があったり，他のストリームデータ応用との連携が十分にとれないなどの問題がある．

そこで本研究では，ストリーム処理エンジンを用いた OLAP 処理を実現するための手法を検討する．具体的には，ストリーム処理エンジン側に連続的問合せを登録しておき，これらの登録しておいた連続的問合せから OLAP 処理の結果として必要なデータを得る．OLAP 処理では，分析対象のデータの次元 (属性) や次元内の階層のすべての組合せを頂点とする lattice を考える．lattice の各頂点は，OLAP 処理において集約する対象の次元の組合せに対応する．最も単純には，lattice 内の全ての頂点の次元の組合せの集約を実行すればよい．しかし一般には，lattice の全頂点数は膨大な数になることが多く，これらの対応する全ての問合せをストリーム処理エンジンに登録することは，処理性能上難しい場合が多い．また，必ずしも全ての頂点に対応する集約値を常時取得し続けなくても，ユーザ要求に応じて導出できれば良いという応用も多いと考えられる．そこで本研究では，lattice 内の一部の頂点のみを適切に選択し連続的問合せとしてストリーム処理エンジンに登録することで，効率的な OLAP 処理を実現することを目的とする．また，登録する連続的問合せからなる効率的な処理パイプラインの構成法についても検討する．

2. 基本事項

2.1 OLAP

OLAP [2] は，データウェアハウス内のデータに対して多次元的なデータ分析を行うために，データを多次元データモデルとして扱い処理する．OLAP は，その前処理として，分析の軸となる複数の次元表と，その分析対象のデータが格納された事実表から，データキューブと呼ばれる多次元データモデルを構築する．データキューブはメジャーと呼ばれる分析対象となる数値の集合と，それに属する複数の軸で構成される．このデータキューブに対して，ロールアップやドリルダウン等の操作を行うことができる．ここでは，データの例として TPC-D ベンチマーク [3] に準じたデータを用いる．今，商品テーブル，顧客テーブル，供給者テーブルの三つのテーブルと，ある期間に顧客が発注した部品の情報 (売上情報) を表すデータキューブを構築することを考える．この例では，商品，顧客，供給者が次元表となり，売上情報が事実表となる．また，売上情報の要素である売上額がメジャーとなる．次元表と事実表の関係は図 1 のようなスタースキーマと呼ばれる構造で表せる．これらの表から，図 2 のようなデータキューブが構築される．また，図 2 に対して顧客で Roll-up を行い，顧客の住所 (県) で集約を行った例，および，供給者で Drill-down を行い，供給者の中のどの部門かを示した例を図 3 に示す．

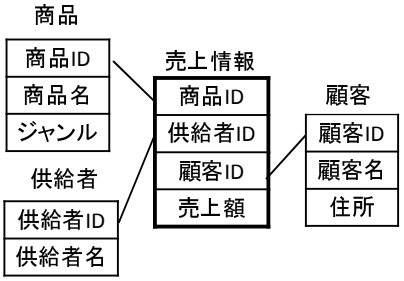

図 1　スタースキーマ

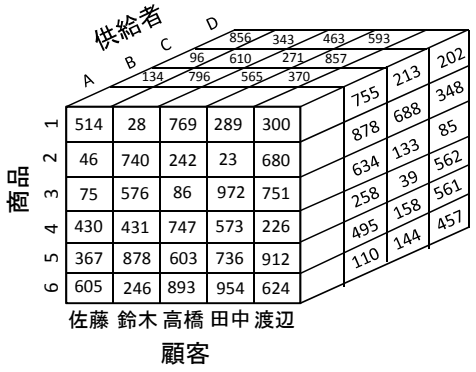

図 2　データキューブ

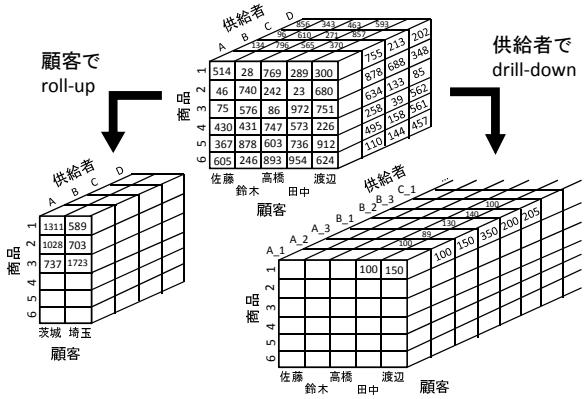

図 3　データキューブに Roll-up および Drill-down を行った例

静的なデータに対する OLAP 処理は，以上で述べたような手法で行われる．一方で，静的なデータとストリームデータの間のデータ処理における違いは，後者は，大量に生成され，入力から出力に向かい動的に流れ，急速に変化するということである．そのため，ストリームデータに対する OLAP 処理は，時々刻々と到着するデータを用いた最新の分析結果を示す必要があり，また，その効率化が望まれる．

2.2　ストリーム処理エンジン

ストリーム処理エンジンは，際限なく到達するストリームデータに対して長期間連続的に問合せ処理が可能なように開発されたものである．このようなストリーム処理エンジンには，STREAM [4]，Storm [5]，S4 [6]，Borealis [7] などが挙げられる．

ストリーム処理エンジンでは，ストリームデータに対し CQL 等の言語を用いて問合せを行う．このような言語で書かれた問合せ (連続的問合せ) をストリーム処理エンジンに登録することにより，ストリームデータに対して連続的に問合せを行うことができる．

本研究では，OLAP 処理の一部をストリーム処理エンジンで行うことにより，より効率的なストリームデータに対する OLAP 処理を行う．つまり，ストリーム処理エンジンの機能を用いて必要なデータの集約を行う．具体的には，データの次元，あるいは階層に対して集約を行う問合せの一部を連続的問合せとしてストリーム処理エンジンに登録する．ここで，登録する連続的問合せは，ユーザが常にその結果を欲しいような問合せや，連続的問合せ間の依存関係に着目した，OLAP 処理においてコアとなる連続的問合せ群である．これらに関しては，以降の章において詳述する．

例えば，商品ごとの直近一時間の売上を顧客の住所毎にモニタしたい場合は，以下のような連続的問合せをストリーム処理エンジンに登録する．

```
Select 商品 ID，顧客住所，Sum(売上額)
From 売上情報 [Range 1 hour]
Group By 商品 ID，顧客住所
```

3.　関 連 研 究

3.1　Stream Cube

Jiawei Han らは，ストリームデータの多次元分析を容易にする，Stream Cube と呼ばれるアーキテクチャを提案している [1]．この Stream Cube の計算のために三つの技術を考案している．

一つ目は，tilted time frame である．これは，ストリームデータ内の時間次元に着目し，最近のデータはより細かい粒度 (15 分毎等) で，時間が経ったデータは集約してデータの粒度を粗く (1 時間や 1 週間等) して保持することにより，総データ量を小さくする手法である．

二つ目は，two critical layers である．これは，データの次元やその階層全ての組合せを実体化するのではなく，minimal interesting layer と呼ばれる，分析者が興味のある最低限の次元の集約の組合せと，observation layer と呼ばれる，分析者がその次元の集約の組合せからドリルダウンするための層の二つの層のみを実体化することにより，空間的コストを削減する手法である．

三つめは，popular path approach である．これは，実体化した二層間の中間層を計算するための一つのパスを用意し，このパスに沿ったドリルダウンにより中間層を計算する手法である．このパスは，H-tree [8] と呼ばれるデータ構造を用いて格納する．

この StreamCube の手法は，ストリームデータに OLAP 処理を適用する手法であるが，一方で時間次元に対する集約の方法が事前に決められているため，時間に対する柔軟な問合せを書くことができない．また，実体化した二層間のパスをユーザが決める必要があるため，その決定に知識を要するという欠点

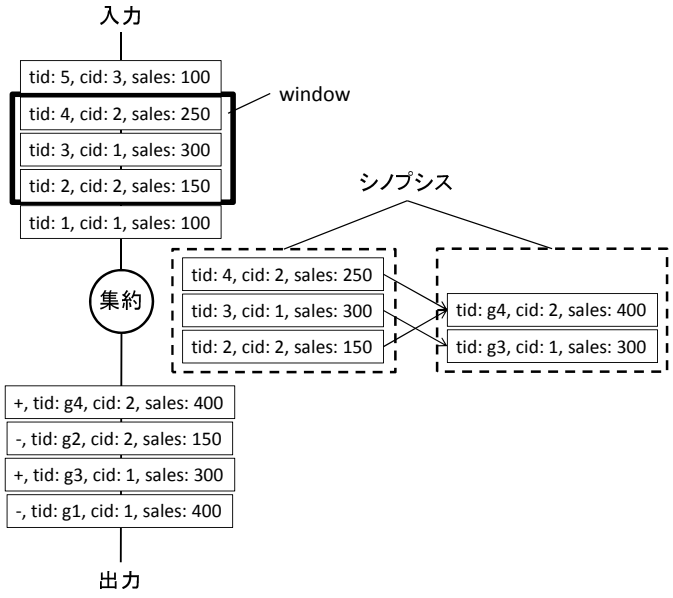

図 4　ストリームデータ処理における集約処理

がある．さらに，ストリーム処理エンジンの利用はまったく考慮されていない．

3.2　静的なデータに対するデータキューブの部分的実体化

OLAP 処理では，様々なレベルのデータ分析を行うため，集約演算が多用される．この集約問合せの応答時間を小さくする手法として問合せ結果を実体化することがあげられる．しかし，全問合せの実体化には非常に大きな空間的コストがかかる．また，問合せ間には，ある問合せの結果が，他の問合せの結果から導出することができるというような依存関係が存在する．そこで，Venky Harinarayan らは，そのような依存関係を表す lattice フレームワークと，実体化する一部の問合せを適切に選ぶためのアルゴリズムによって，問合せの応答時間を削減する手法を提案している [9].

lattice フレームワークは，ある問合せが，他の問合せの結果から答えられるといった依存関係を示す．例えば，ある問合せ a が，別の問合せ b の結果から導出できる場合，a から b に上に向かってパスが存在する．これにより，全ての次元によって集約される問合せを top とし，まったく集約されない問合せを bottom とする lattice が構築される．

Harinarayan らのアルゴリズムは，構築された lattice を元に，空間的コストを与えられた制約の範囲内とし，かつ，時間的コストが小さくなるように，実体化する問合せの組合せを選ぶアルゴリズムである．

4.　ストリームデータ処理における集約演算

従来のデータベース処理では，クエリ発行時に，DB に格納されたデータをバッチ処理などで一括処理を行うのに対し，ストリームデータ処理は，データ到着時に事前に登録したクエリを実行する，差分処理を行う．そのため，ストリームデータ処理では，連続的に到着する入力ストリームに対し，最も新しいタプルから一定の件数のタプルを演算対象として処理を行う．この件数を window 幅と呼び，この処理を window 演算と呼ぶ．また，ストリームデータ処理では，演算の結果を中間データとして保持しておき，新規の入力タプルが入ってきた時，差分計算によって問合せ結果を得る．この際，window 内に新たに到着したタプルによって生成されたタプルにプラス，window からはみ出したタプルに起因するタプルにマイナスの記号をつけ，出力とする．中間データを保持しておく領域をシノプシスと呼び，プラスが付与された出力をプラスタプル，マイナスが付与された出力をマイナスタプルと呼ぶ．

ここで，ストリームデータ処理においてどのように集約演算が実行されるかについて図 4 を用いて説明する．選択演算や結合演算等の他の演算の処理については，A. Arasu らの CQL についての論文 [10] を参照されたい．図 4 は，入力タプルが tid, cid，sales の三つの属性を持ち，cid を集約属性として sales を合計する例である．なお，window 幅はタプル 3 つ分であり，今，window 内にある演算対象データが tid の 1 から 3 から，2 から 4 にスライドされた状態とする．この状態から，ストリームデータ処理における集約演算を説明する．まず，指定の window 幅に含まれる入力ストリームに対して集約属性について集約演算を行う．この時，シノプシスには，演算対象となった元の入力データと集約結果のデータの両方を保持しておく．これは，元の入力データを保持していなければ，次の window 演算において正しく集約演算ができないためである．図 4 の例では，tid が 2 から 4 のタプルを cid について集約する．これにより，cid が 1 の集約結果が 300，cid が 2 の集約結果が 400 と計算され，これらの tid として新たに g3，g4 が振られる．ここで，先ほど tid が 1 から 3 のタプルにおいて計算した集約結果のタプルである g1，g2 が今回の集約演算の結果により置き換わる．よって，シノプシスに tid が 2 から 4 の元データのタプルおよび tid が g3 と g4 の集約結果のタプルが格納され，これら g3，g4 にプラス，直前の集約演算の結果である g1，g2 がマイナス付きで出力される．このように，集約演算の場合，window に新しく入ったタプルと追い出されたタプルのぞ

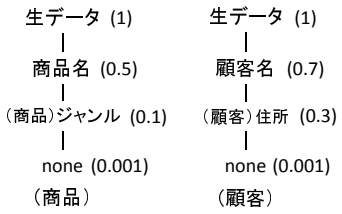

図 5　商品および顧客の lattice

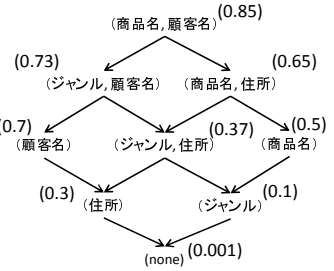

図 6　商品および顧客を統合した lattice

れぞれの集約属性の集約結果の更新により，プラスタプルとマイナスタプルの組が 2 つ，計 4 タプルが生成される．他にも，window に新しく入ったタプルと追い出されたタプルぞれぞれの集約属性が同じ場合は，プラスタプルとマイナスタプルの組が 1 つの計 2 タプルが生成され，新しく入ったタプルの集約属性の集約結果がシノプシスに無い場合と，タプルが追い出されることによってシノプシス内にそのデータの集約属性の集約結果が無くなる場合は計 3 タプルが生成される．

5. 提 案 手 法

5.1 処理の効率化

集約処理をストリーム処理エンジンで行う場合，データの次元，階層の全ての集約の組合せを連続的問合せとしてエンジンに登録することが考えられる．しかし，一般にその組合せの数は膨大になり，それら全てをエンジンに登録することは処理性能上難しい．また，ユーザが取得したい集約は，それらの全てではなく一部であることが大部分である．そのため，エンジンにどの集約問合せを登録するかを選択する．

また，各集約間には依存関係が存在する．そのため，選択された連続的問合せを独立に評価せずに，それらを依存関係に基づいて結びつけ，先に評価された集約結果を利用することで，処理を効率化できる．そこで，各集約の評価順を表す処理のパイプラインを構築する．

以上の二つの操作を行うために，まず，集約処理をストリーム処理エンジンで行う際のコストモデルを定義する．

5.2 コストモデル

本節では，次節以降において，ストリーム処理エンジンに登録する集約問合せを決めるアルゴリズムに用いるコストモデルについて説明する．

まず，以下のようなスキーマを持つデータ内の二つの次元から lattice を構成することを考える．

売上情報（商品 ID，商品名，ジャンル，顧客 ID，
　　顧客名，住所，売上額）

商品はその次元内の階層としてジャンルが存在し，顧客はその次元内の階層として住所が存在するため，商品と顧客の個々の lattice は，図 5 のようになる．上から下に向かって集約度は上がり，一番下の none は，たった一つの値に集約されることを意味している．この二つの lattice を一つの lattice に統合したものが図 6 である．なお，図 5，図 6 のそれぞれの集約の肩に乗る数字については後述する．

本手法において，処理にかかる全体のコストは lattice 内のそれぞれの集約問合せにおいてかかるコストの総和と考える．図 6 のような lattice 内には，ストリーム処理エンジンに登録する集約問合せと，ストリーム処理エンジンに直接集約問合せを登録せずに，必要に応じてストリーム処理エンジンに登録する集約問合せの結果から必要とする集約結果を導出する (以降，オンデマンドに評価すると言う) 集約問合せの二種類の問合せが存在すると考える．ここで前者を $Q_r = \{r_1, ..., r_n\}$ とし，後者を $Q_d = \{d_1, ..., d_m\}$ とする．これらを合わせた lattice 内の集約の全ての集合を Q とする．また，lattice 内のそれぞれの集約問合せ結果が参照されるタイミングは問合せごとに異なる．ストリーム処理エンジンに登録する集約問合せは，新しいタプルが到着する度に問合せが実行されるが，オンデマンドに評価する集約問合せはユーザからその集約結果が求められたときのみ実行されるので，両者の間には参照される頻度に相違がある．そこで，それぞれの集約が参照される頻度をコストに重み付けすることを考える．ここでは，Q の一つ一つの集約結果が単位時間あたりに参照される平均の頻度を P とし，エンジンに登録する集約問合せの平均参照頻度を $P(r_i)$，オンデマンドに評価する集約問合せの平均参照頻度を $P(d_i)$ として重み付けを行う．

ここからまず，ストリーム処理エンジンに登録した問合せに対する集約処理を行う単位時間あたりのコストを考える．これを C_1 とする．C_1 は，単位時間あたりにその集約問合せに到達するタプル数と集約計算自体の処理に応じたコストとなる．生データの単位時間あたりの入力レートを i とすると，それぞれの集約問合せには，$i \times w$ のコストがかかる．w について，4 節で説明したように，演算対象の window がスライドして集約結果の更新が行われると，プラスタプルとマイナスタプルが合わせて 2 個，3 個，4 個のいずれかの個数分生成される．そのため，w は 2，3，4 の値をとる．以降の例では，最悪のケースを考えて $w = 4$ としてコストを考える．集約計算自体の処理のコストを考えると，集約処理は，タプルが到着する度に集約属性に応じてハッシュを用いてタプルを振り分け集約すると考えるため，O(1) だけしかかからないとする．以上から，C_1 は，単位時間あたりにその集約問合せに到達するタプル数に応じたコストと考える．次に，オンデマンドに評価する集約問合せの単位時間あたりの処理コストを考える．これを C_2 とする．オンデマンドに評価する集約問合せは，ユーザからその集約が求められたときに限り，lattice 上で自分自身の上流にある Q_r 内の問合せの中で，そのシノプシスに保持する全タプル数が最も

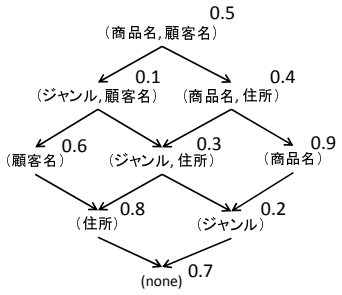

図 7　それぞれの集約問合せの平均参照頻度

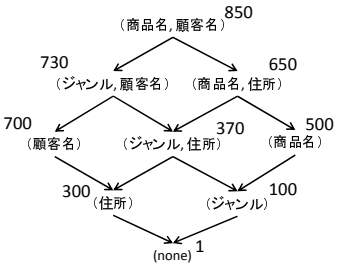

図 8　それぞれの集約結果のタプル数

少ない問合せを選び，その結果から必要とする集約結果を導出する．そのためコストは，その連続的問合せのシノプシス内のタプル数に依存する．従って C_2 は，その連続的問合せの集約結果タプル数に自分自身の問合せの平均参照頻度 (単位時間あたりに自分自身の問合せが参照される回数) をかけたものに応じたコストとなる．さらに，ストリーム処理エンジンに登録した連続的問合せがそのシノプシスに持つ全タプル数を空間コストとして考える．これを S_1 とする．

以上から，処理コスト C および空間コスト S を以下のように定義する．

$$C = \sum_{i=1}^{n} C_1(r_i) + \sum_{i=1}^{m} C_2(d_i)P(d_i) \quad (1)$$

$$S = \sum_{i=1}^{n} S_1(r_i) \quad (2)$$

これらの式を用いて，処理コストの計算を行う．

5.3　連続的問合せ群の決定するためのアルゴリズム

本節では，ストリーム処理エンジンに登録する集約問合せを決定するアルゴリズムについて説明する．このアルゴリズムについては，3.2 節で紹介した Harinarayan らのアルゴリズムを応用する．

まず，lattice のトップは必ずエンジンに登録する連続的問合せとして選ばれる．これは，ストリーム処理エンジン登録しない集約問合せは，必要に応じてストリーム処理エンジンに登録する集約問合せの結果から必要とする集約結果を導出するため，仮に lattice のトップがエンジンに登録されなかった場合，集約結果の導出元となる集約問合せが無いことから，集約問合せを実行できないためである．次に二つ目にエンジンに登録する集約問合せを選ぶ．この選び方は，先ほど選んだ lattice の top の集約を除いた集約問合せの中から，次にエンジンに登録する集約問合せを一つ選び，その集約問合せを連続的問合せとしてエンジンに登録した場合の処理コスト C を計算する．この計算を，lattice の top の集約を除いた全ての集約に対して行う．この結果，C の値が最も小さかったものを，次にエンジンに登録する集約問合せとして選ぶ．この後，三つ目にエンジンに登録する集約問合せを選ぶ．この場合も同様に，lattice の top と先ほど二つ目にエンジンに登録する集約問合せとして選ばれたものを除いた集約問合せの中から，C が最も小さくなるものを選ぶ．以降，この操作を繰り返し行う．この繰り返しを終了する条件としては，S の値を用いる．S の値にある上限 S_{max} を設け，ある一定の値を S が超えたときに上述のアルゴリズムを終了する．このアルゴリズムが終了した時点で，エンジンに連続的問合せとして登録する集約問合せの個数ごとの最小の C の値を比較し，これらの値の中で最も C の値が小さい個数個まで，エンジンに連続的問合せとして登録する．

以上のアルゴリズムを擬似コードとして表すと以下のようになる．

```
local_min_c = {};
nl = number of local_min_c;
sum_of_S = 0;
Q_r = {top aggregation};
for i=1 to all node except for top do begin
    if sum_of_S > S_max then exit for loop;
    for j = 1 to m do begin
        select that aggregation d in Q_d such
               that C is minimized;
        local_min_c = local_min_c union {(d,C)};
        nl++;
        sum_of_S += S;
    end;
end;
define number n that is nth element
       whose C is minimam in local_min_c;
for i=1 to n do begin
    extract d in nth (d,C) in local_min_c;
    Q_r = Q_r union {d};
end;
resulting Q_r is the continuous queries;
```

5.4　処理パイプラインの決定

本節では，前節のアルゴリズムによって決定した連続的問合せ群をどの順番で評価を行うかを表すパイプラインを構築する．

まず，lattice の top をパイプラインの root とする．次に，ストリーム処理エンジンに連続的問合せとして登録した集約問合せ間にエッジを張ることを考える．これは，自分の上流に存在する連続的問合せに向けてエッジを張る．ここで，経路の選択肢が複数あり，どちらか一方に決める必要が生じた場合は，それらの連続的問合せのシノプシス内にあるタプル数を比較し，

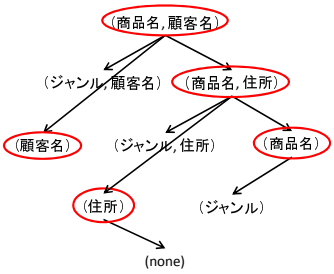

図 9　構築された処理木

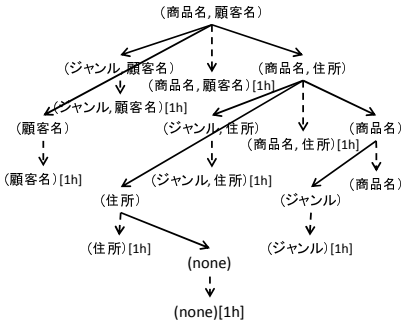

図 10　時間次元の集約を考慮した処理木

その数が少ない方の経路を選択するようにする．なぜなら，まず連続的問合せの入力レートは同一なので，入力レートでは比較できない．また，タプル数が少ない方が，より粗く集約されているため属性数も少なく，集約における処理が少なく済むためである．ストリーム処理エンジンに登録した連続的問合せ間にエッジを張った後，次に，連続的問合せとしてストリーム処理エンジンに登録されなかった集約問合せも含めてエッジを張ることを考える．この場合も同様に，自分の上流に存在する連続的問合せに向けてエッジを張る．ただし，経路の選択肢が複数あり，どちらか一方に決める必要が生じた場合は，その上流にある連続的問合せのシノプシス内にある集約結果のタプル数を比較し，その数が小さい方の経路を選択するようにする．以上で，処理のパイプラインが構築される．これにより，連続的問合せとして登録された問合せは毎回の演算で結果を出力し，連続的問合せとして選択されなかった集約問合せがユーザから要求された場合は，その祖先にある連続的問合せの結果から導出される．

5.5　提案手法のアルゴリズムを用いた具体例

ここで具体的な例を用いて前節のアルゴリズムを説明する．

前提として，まず，ストリームの入力レートを 200tuple/s とする．lattice 内のパイプラインの入力レートは 400tuple/s, 600tuple/s，800tuple/s がとりうるが，ここでは最悪のケースを考えて 800tuple/s とする．また，図 5，図 6 共に，生データを 1 としたときのそれぞれの集約の集約率がそれぞれの肩に乗る数字で表したようになると仮定する．さらに，図 7 に，それぞれの集約問合せの平均参照頻度を，それぞれの肩に乗る数字になると仮定する．なお，それぞれの集約問合せにおける window 幅は 1000 とする．すなわち例えば，商品名と住所で集約する問合せは以下のようになる．

```
Select 商品名，ジャンル，住所，Sum(売上額)
From 売上情報 [Rows 1000]
Group By 商品名，住所
```

図 6 で示した集約率と合わせて，それぞれの集約の結果タプル数は図 8 のようになると仮定する．この条件下において，C が最小となるようなエンジンに登録する連続的問合せを選ぶ．

まず，lattice のトップである，(商品名，顧客名) が選ばれる．次に二つ目にエンジンに登録する集約問合せを選ぶ．(ジャンル，顧客名) を選んだとき，コスト C は，以下のように計算される．

$$
\begin{aligned}
C \;=\; & 200 + 800 + (1000 + 850) \times 0.4 + (850 + 730) \times 0.6 \\
& + (850 + 730) \times 0.3 + (1000 + 850) \times 0.9 \\
& + (850 + 730) \times 0.8 + (850 + 730) \times 0.2 \\
& + (850 + 730) \times 0.7 \\
=\; & 7513
\end{aligned}
$$

同様に他の全ての集約について行うとそのコスト C は，以下の表のようになる．

表 1

(ジャンル，顧客名)	7513
(商品名，住所)	6645
(顧客名)	6840
(ジャンル，住所)	6774
(商品名)	6285
(住所)	6430
(ジャンル)	7400
(none)	7105

表 1 から，二つ目にストリーム処理エンジンに登録する集約問合せとしてもっともコストが小さいものは，(商品名) であるとわかる．よって，(商品名) を二つ目にストリーム処理エンジンに登録する集約問合せとして選択する．このように三つ目以降も選択を続け，結果的に，(住所)，(顧客名)，(商品名，住所)，(ジャンル，住所)，(none)，(ジャンル)，(ジャンル，顧客名) の順番でストリーム処理エンジンに登録されることとなる．

次に，これらをどの順番で評価を行うかを表すパイプラインを構築する．先に決まった集約問合せの中で，最初から 5 つを選んだところで S が S_{max} の値を超え，アルゴリズムが終了したとする．つまり，(商品名，顧客名)，(商品名)，(住所)，(顧客名)，(商品名，住所) を実際に連続的問合せとして選択することとし，それらを中心に処理パイプラインを構築することを考える．例えば，(ジャンル，顧客名) は，その上流にある連続的問合せである (商品名，顧客名) に向かってエッジを張る．また，(住所) からその上流にある連続的問合せに向かってエッジを張ることを考えると，(顧客名)，(ジャンル，住所) への二つの選択肢が考えられるが，それらのシノプシス内の集約結果タプル数が，(ジャンル，住所) の方が少ないため，こちらにエッジを張る．このように，前節で示した規則から，処理パイプラインが図 9 のように構築される．なお，図 9 中で円で囲まれたノードは，選択された連続的問合せ群である．

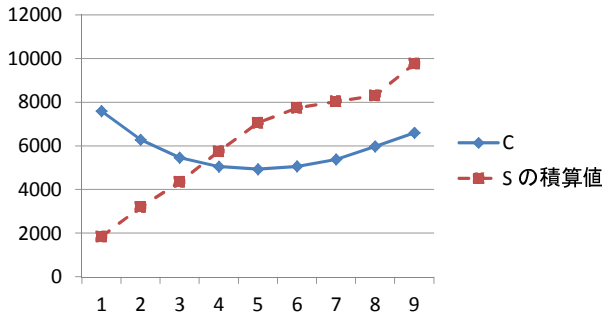

図 11　評 価 結 果

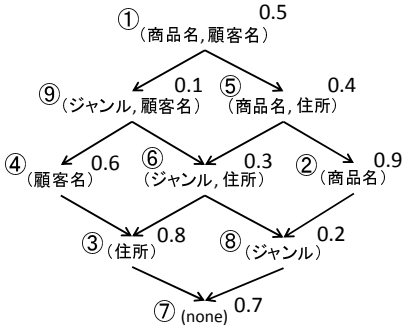

図 12　エンジンに登録される順序と参照頻度の関係

5.6　時間に対する集約

ストリームデータに対して OLAP 処理をし，データの傾向などの高いレベルの分析を行う場合，時間に対して集約を行いたいというニーズがある．ストリームデータを時間に対して集約処理を行うために，それぞれの集約問合せに加えて，ある広い一定の時間の window 幅で集約をする問合せも行うことを考える．すなわち，図 10 のように考える．図 10 の場合は，1000 タプルの window 幅が 1 時間の window 幅よりも大きいときであり，この場合は時間の window 幅による集約はその下流に置かれる．例えば，商品名と住所で集約する問合せでは以下のような集約も行う．

```
Select 商品名，ジャンル，住所，Sum(売上)
From 売上情報 [Range 1 hour]
Group By 商品名，住所
```

これにより，例えば商品名と住所の集約問合せにおいて，直近 30 分の集約が要求された場合においても，上記の問合せから導出できる．

6.　予備的評価

本章では，5.5 節における具体例において，本稿のアルゴリズムによって，どのように処理が効率化されているかを調べる．そのために，連続的問合せとして選択される個数を 1 から 9 の間で変えたときの処理コストの和 (実線)，および，登録した連続的問合せの集約結果のタプル数 (空間コスト) の積算値 (点線) を求める．この結果は，図 11 のようになる．ここで，x 軸は集約問合せをストリーム処理エンジンに連続的問合せとして登録する個数，y 軸はタプル数である．図 11 から，5.5 節における例の場合，ストリーム処理エンジンに連続的問合せとして 5 つ登録したときに最もコストが小さくなることがわかる．

また，図 12 は，それぞれの集約問合せがエンジンに登録される順番と、それぞれの集約問合せの平均参照頻度を表している．エンジンに登録される順番は丸付き数字で表しており，図 12 から，平均参照頻度が 0.9 である（商品名）が二番目に登録されるものとして選ばれ，平均参照頻度が 0.8 である（住所）が三番目に登録されるものとして選ばれることから，概ね参照頻度が高い集約問合せからエンジンに登録されている．

7.　まとめと今後の課題

本稿では，ストリーム処理エンジンを用いた OLAP 処理を実現の手法と，処理の効率化のためのアプローチを示した．また，本稿で提案したコストモデルにより，どのように処理コストが効率化されているかをシミュレーションし確認した．今後は，時間次元に対する集約を加えることにより，処理コストにどのような影響が出るかを再検討する．また，上述した手法を実際のストリーム処理エンジンに組込み，実験を行い，本手法の有用性を実際に評価する．

8.　謝　　辞

本研究の一部は，(株)KDDI 研究所と筑波大学の共同研究による．(CPE25113)

文　　献

[1] J. Han et al. “Stream Cube: An Architecture for Multi-Dimensional Analysis of Data Streams”, Distributed and Parallel Databases, 18(2):173-197, Sep. 20 2005.
[2] J. Han et al. “Data Mining: Concepts and Techniques”, Morgan Kaufmann, Third Edition, Jul. 2011.
[3] TPC-D, http://www.tpc.org/tpcd/default.asp.
[4] A. Arasu et al. “STREAM: The Stanford Data Stream Management System”, Technical Report, Department of Computer Science, Stanford University, http://ilpubs.stanford.edu:8090/641/, 2004.
[5] Storm, http://storm-project.net/.
[6] L. Neumeyer et al. “S4: Distributed Stream Computing Platform”, In Data Mining Workshops (ICDMW), 2010 IEEE conference on, pages 170-177, 13 Dec. 2010.
[7] D. J. Abadi et al. “The Design of the Borealis Stream Processing Engine”, In Second Biennial Conference on Innovative Data Systems Research (CIDR 2005), Asilomar, CA, Jan. 2005.
[8] J. Han et al. “Efficient Computation of Iceberg Cubes with Complex Measures”, In Proc. 2001 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD ’01), pages 1-12, Santa Barbara, CA, May 2001.
[9] V. Harinarayan et al. “Implementing Data Cubes Efficiently”, In Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD ’96), pages 205-216, ACM, Montreal, Canada, June 1996.
[10] A. Arasu et al. “The CQL continuous query language: semantic foundations and query execution”, The VLDB Journal, 15(2):121-142, 2006.