DBSJ Japanese Journal
Vol. 14-J, Article No. 3, March 2016

StreamOLAPにおける問合せの最適化手法

Cost-based Optimization of Stream OLAP

中挾 晃介♡ 北川 博之♢ Salman Ahmed SHAIKH♠ 天笠 俊之♣

Kosuke NAKABASAMI Hiroyuki KITAGAWA
Salman Ahmed SHAIKH Toshiyuki AMAGASA

近年，ストリームデータの増加に伴いストリーム処理エンジン (SPE) 等の各種技術が開発されている．ストリーム処理においては，フィルタリングの他，より高度な集約分析が重要である．静的データに対する代表的集約分析の手法として OLAP がある．OLAP をストリームデータに対して適用する提案がこれまでにもあるが，SPE の利用を考慮しておらず，また，問合せに数多くの制約があるという課題がある．本研究では，SPE とストリーム OLAP エンジンを連携させたシステムアーキテクチャを示し，さらに，コストモデルに基づくストリーム OLAP 処理の最適化手法を提案する．具体的には，提案アーキテクチャにおいて，OLAP 問合せ結果の効率的な導出を実現する最適化アルゴリズムを提案し，実験により提案手法の有効性を示す．

Due to the increase of stream data sources, many Stream Processing Engines (SPEs) have been developed and deployed. Besides simple data filtering, demands for more sophisticated analysis of streams such as multi-level aggregation and summarization are increasing. OLAP is one of the common methods for systematic analysis of static data and has been studied intensively. Some existing works proposed application of OLAP to stream data. However, they lack some required features and do not consider use of SPEs for stream OLAP. Moreover, cost-based analysis to improve efficiency of stream OLAP has not been studied before. In this paper we present a stream OLAP architecture consisting of an SPE and an OLAP engine as a general framework for stream OLAP. Then, we propose a cost-based optimization scheme. To the best of our knowledge, this is the first proposal for cost-based optimization of stream OLAP. Moreover, we have implemented a runnable prototype system, and evaluated the proposed optimization scheme. The experimental results prove the effectiveness of the proposal.

♡ 正会員　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
nakabasami@kde.cs.tsukuba.ac.jp
♢ 正会員　筑波大学システム情報系情報工学域
kitagawa@cs.tsukuba.ac.jp
♠ 正会員　筑波大学計算科学研究センター
salman@kde.cs.tsukuba.ac.jp
♣ 正会員　筑波大学システム情報系情報工学域
amagasa@cs.tsukuba.ac.jp

1. はじめに

近年，センサデータやマイクロブログ等，連続的に生成され，配信されるようなストリームデータの増加に伴い，このストリームデータを集約し，そのデータの傾向などの分析を行いたいというニーズが増加している．この代表例として，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があり，その一つに OLAP 処理 [1] がある．OLAP 処理が対象とするデータの次元には階層が存在し，どの階層で集約演算をするかによって異なる粒度の分析を行うことができる．OLAP 処理では，分析対象のデータの次元や次元内の階層の全ての組合せを頂点とする lattice を考える．この lattice の頂点にあたる集約問合せ (以下，OLAP 問合せ) の結果をユーザの要求に応じて出力することで，OLAP 処理を実現している．

一方，このストリームデータを処理する基盤としてストリーム処理エンジン (SPE) が多数開発されている．このような SPE には，STREAM [2]，Storm [3]，S4 [4]，Borealis [5] などが挙げられる．SPE に SQL ライクな問合せである CQL [6] で書かれた問合せを登録することで，ストリームデータに対して連続的な問合せを行うことができる．

ストリームデータに対して OLAP 処理を行う研究として，Jiawei 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 [7]．ただし，この手法は SPE の利用を想定しておらず，一部の問合せの結果を返すことができないなどの制約が存在する．

本研究では，SPE と OLAP エンジンからなる，ストリームデータに対する OLAP 処理のためのアーキテクチャを提案する．以降，ストリームデータに対する OLAP 処理のことを StreamOLAP と書く．StreamOLAP を実現する最も単純な方法として，lattice 内の全ての頂点に対応した CQL の問合せを SPE に連続的問合せとして登録し，これらの結果から，OLAP 問合せの結果を実体化しておくことが考えられる．しかし一般には，lattice の全頂点数は膨大な数になり，この全頂点に対応する連続的問合せを SPE に登録することは処理性能上難しく，また，これらの全ての結果を実体化することはメモリ制約から現実的ではない．また，必ずしも全ての頂点に対応する問合せの結果を実体化する必要はなく，ユーザ要求に応じて導出することが可能である．

そこで本研究では，コストモデルに基づいた最適化アルゴリズムを提案する．これは，lattice 内の全 OLAP 問合せを，SPE に連続的問合せとして登録し集約結果を実体化しておく問合せと，ユーザ要求に応じて実体化した問合せ結果から自身の集約結果を導く問合せ (On-demand Query) の二種類に分類する．この分類を，与えられたメモリ制約の範囲内で，処理コストが最小になるように行うアルゴリズムである．

2. 基本事項

2.1 OLAP

OLAP [1] 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を用意し，これらの表のスキーマ間の関係を表すスタースキーマを構築する．このスタースキーマの情報から，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，ユーザは，データのより詳

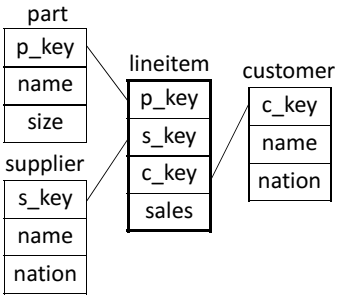


図 1: スタースキーマ

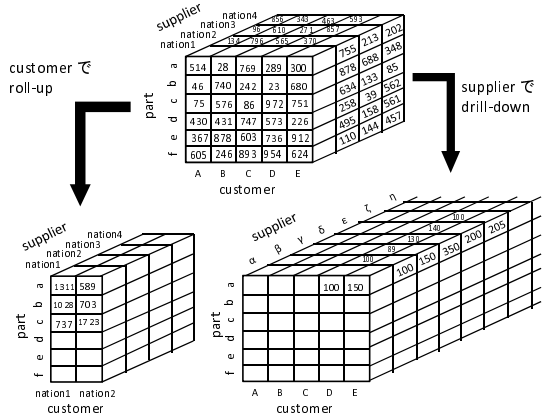


図 2: データキューブおよびキューブに対して drill-down と roll-up を行う例

細な粒度の集約結果を求める drill-down や，逆により粗い粒度の集約結果を求める roll-up 等の操作を対話的に行うことができる．ここでは，データの例として TPC-H ベンチマークに準じたデータを用いる．今，customer，part，supplier の三つを軸とし，ある顧客が発注した売上情報，lineitem 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，customer，part，supplier が次元表となり，lineitem が事実表となる．また，lineitem の要素である sales 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で表せる．

これらの表から，図 2 の上のようなデータキューブが構築される．また，図 2 のように，データキューブに対して customer で roll-up を行い，customer の nation で集約を行ったり，supplier で drill-down を行い，supplier を name で細分化を行うことが可能である．

2.2 ストリーム処理エンジン (SPE)

ストリーム処理エンジン (SPE) 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 SPE には，STREAM [2]，Storm [3]，S4 [4]，Borealis [5] などが挙げられる．

SPE では，ストリームデータに対し CQL [6] 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 SPE に一つ以上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，ストリームデータに対する OLAP 処理に SPE を利用することで，StreamOLAP を実現する．すなわち，SPE のストリームデータに対する集約演算機能を用いて必要なデータの集約を行う．具体的には，OLAP 問合せにおけるいくつかの次元階層に対する集約を，連続的問合せとして SPE に登録する．

例えば，part ごとの直近一時間の売上 (sales) の合計を customer の地域 (nation) 毎にモニタしたい場合は，以下のような連続的問合せを SPE に登録する．

```
Select part_name, customer_nation, Sum(sales)
From lineitem [Range 1 hour]
Group By part_name, customer_nation
```

3. 関連研究

OLAP 処理における最適化についての研究は，以下に述べるようなものがある．

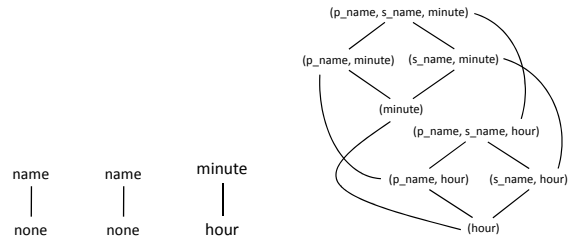
Aouiche らは，実体化する View と適切なインデックスの選択を併用し，データウェアハウス内のデータアクセスを効率化している [8]．Talebi らは，適切な View とインデックスの選択の最適化により，OLAP 処理の効率化をしている [9]．Santos らは，OLAP 処理の効率化のために，インデックスの選択の他に partitioning により OLAP 処理の最適化を行っている [10]．Harinarayan らは，OLAP 処理において用いる様々な粒度の集約問合せに対して，lattice フレームワークを構築してそれらの依存関係を表し，その中で，どの集約問合せを実体化し，問合せにおけるコストを小さくするかを決めるアルゴリズムを提案している [11]．Joslyn らは，lattice 内の最適な View の実体化を行うために，データキューブ内の情報理論的な統計的尺度を用いている [12]．これらの研究は静的に蓄積されたデータを対象としており，ストリームデータを対象としたものではない．

ストリームデータを対象とした OLAP としては，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 [7]．Stream Cube は，実体化する OLAP 問合せを減らして空間コストを削減するために，より過去のデータほど粗い粒度で集約し，最近のデータほど細かい粒度で集約し実体化した結果を保持する．さらに，空間コストを削減するために，ユーザが要求すると思われる最も細かい粒度と最も粗い粒度の OLAP 問合せ，およびその二つの間のパス中に存在する OLAP 問合せの結果のみを実体化する．ただし，過去のデータに対して細かい粒度で問い合わせるといった，時間次元に対する柔軟な問合せができない．また，実体化した一部のキューブよりも細かい粒度での OLAP 問合せが要求された場合，その問合せ結果を返すことができない．さらに，SPE の利用を想定していない．Rui Zhang らは，ストリームデータである IP トラフィックデータに対し，SPE の一つである Gigascope を用いて高速に複数の集約を行う手法を提案している [13]．具体的には，集約問合せの結果をハッシュテーブルに保持し，また，粒度の細かい一部の集約問合せの結果を，粒度の粗い集約問合せの結果を導くために用いることで，高速な集約処理を実現している．ただし，この研究は Gigascope を用いることを前提としており，対象データについても IP トラフィックデータに限定している．本研究では，コストモデルに基づいた Stream OLAP の最適化手法を提案する．さらに本研究では，時間属性についても一つの次元として扱い lattice を構築することで，任意の時間粒度で集約結果を取得できる．また，lattice 内の OLAP 問合せ処理を SPE と連携して行う．

4. StreamOLAP における OLAP 問合せ

4.1 ストリームデータに対する OLAP 処理

本研究では，リレーショナルタプルがデータストリームとして流れてくるものと仮定する．StreamOLAP の詳しい説明の前に，まず，データの次元とその階層情報からどのように lattice を構築するかについて説明する．例として，part，supplier，time の次元がそれぞれ図 3(a)， 3(b)， 3(c) のような階層を持つとする．lattice の root にあたる頂点は，それぞれの次元の階層の中で最も粒度が細かい階層の集約となる．すなわち，(p_name，s_name，minute) による集約となる．その後，(p_name，s_name，hour)



(a) part (b) supplier (c) time (d) their lattice

図 3: part, supplier time 次元の階層構造とそれらを組み合わせた lattice

, (p_name, none, minute) , (p_name, none, hour) というように，それぞれの次元の階層を組合せた集約を lattice に追加していく．これを全ての組合せについて行うと，図 3(d) のような lattice が構築される．

lattice 内の各頂点の間には，ある頂点の集約結果が，エッジで繋がっている上位に存在する頂点の集約結果から導出出来るという依存関係が存在する．言い換えると，ある OLAP 問合せの結果は，それよりも細かい粒度の集約の OLAP 問合せの結果を用いて導くことができる．例えば，図 3(d) の lattice において，(p_name, minute) の OLAP 問合せの結果は，その上位にある (p_name, s_name, minute) の OLAP 問合せの結果から導出できる．

以上までは，静的なデータに対する OLAP 問合せ処理と同様である．しかし，本研究では，ストリームデータに対して OLAP を行うことを考える．

本研究ではスタースキーマにおける事実表に相当するデータがストリームとなることを想定する．ストリームデータは，静的なデータと異なり連続的に無限に流れてくる．そのため，時間次元について，OLAP の対象となる時間幅が無限長になる．しかし，実際は問合せの対象を無限長の時間幅とすることは不可能である．そこで，本研究では，ストリームデータに対して OLAP 問合せをするために，ユーザが興味のある，問合せの対象としたい時間幅 (Interval of Interest (IoI)) を設定する．IoI は，ユーザが問合せ結果を得たい最大時間幅とする．なお，各時間次元において同じ階層の粒度で集約を行う OLAP 問合せの IoI の時間幅は全て同じであることを前提とする．

4.2 ストリームデータに対する OLAP 処理

StreamOLAP のアーキテクチャの全体像を図 4 に示す．StreamOLAP は，SPE と OLAP エンジンという二つの処理エンジンから構成される．SPE は，ストリームデータ情報源から連続的に到着するストリームデータに対して，特定の粒度の階層における集約結果を生成する．OLAP エンジンは，SPE において生成された集約結果を，IoI で指定した時間だけメモリ領域に保持しておく．また，ユーザが，結果を保持していない OLAP 問合せの集約結果を求めたときは，保持してある集約結果から求められた問合せの集約結果を導く．以降で，それぞれのエンジンの役割の詳細を説明する．

4.2.1 ストリーム処理エンジン (SPE)

SPE は，ストリームデータ情報源から送られるタプルストリームを受け取り，CQL の問合せで指定した階層の粒度における集約結果のタプルを生成する．

例えば，図 4 のような，三つの次元 (part, supplier, time) と一つの計算対象となる属性 (sales) の四つの属性からなるタプルストリームを考える．それぞれの次元の階層構造は図 3 で示したものと同じである．

OLAP エンジンは，図 4 で示したような集約演算や結合演算などの CQL [6] 問合せを SPE に登録する．図 4 で示した CQL 問合せは，(s_name, minute) の OLAP 問合せの結果を作るために，直近一分間において，s_name と minute で集約したときの，sales の合計値を求める問合せである．

SPE では，ストリームデータに対する連続的問合せを実行するために，入ってきた最新のタプルから指定幅分のタプルを演算対象とする．この演算対象とするデータ幅を window 幅とよぶ．図 4 は，最新のタプルから過去 1 分間のタプルを集約演算の対象としている．ここで，window 幅を 1 分間にしている理由は，OLAP 問合せ (s_name, minute) の時間粒度が “分” であるためである．問合せ中の RSTREAM [6] は，指定した時間毎に window 内に存在する全てのタプルに対して演算を行った問合せ結果を出力するという演算子である．図 4 の場合，s_name と minute での集約の結果を 1 分毎に出力することになる．したがって，CQL 問合せの集約結果は 1 分毎に生成される．生成された集約結果は再びタプルストリームとして，送信される．

StreamOLAP では，既に述べたように，事実表にあたるタプルが，ストリームデータ情報源から連続的に生成される．次元表のタプルは静的であり，問合せ処理において事実表のタプルと結合するためにのみ使用される．そこで，集約演算を行うときに，事実表のタプルと結合させるために，次元表のタプルをあらかじめ SPE 内に保持しておく．

4.2.2 OLAP エンジン

OLAP エンジンは，a) Registered Query と，b) On-demand Query の二種類の問合せ方式を用いて OLAP 問合せ結果を生成する．

Registered Query Registered Query は，その OLAP 問合せに対応する次元階層の粒度で集約演算を行う CQL の問合せを SPE に登録しておく．CQL 問合せの結果は OLAP エンジンに送られる．OLAP エンジンがこれらの結果タプルを受けとると，それらをメモリに保持しておく．このメモリ領域をデータバッファと呼ぶ．データバッファは，実際には CQL 問合せで生成された集約結果タプルを格納する配列である．OLAP エンジンは最新の集約結果タプルから IoI の時間幅分だけデータバッファに保持しておく．IoI の時間幅は，ユーザからの要求に応えるために，時間次元の各階層において，ユーザが興味のある時間分に設定する．ユーザから Registered Query に対して OLAP 問合せの結果を要求された場合は，OLAP エンジンは，データバッファから直接結果を返す．

図 4 では，(s_name, minute) の問合せが Registered Query であると仮定している．ここで，IoI の長さは 1 時間に設定されると仮定する．これにより，データバッファは直近 1 時間分のデータを保持することになる．図 4 では，現時点でデータバッファは 9:00 から 9:59 までの集約結果タプルを保持している．ここにタイムスタンプが 10:00 の集約結果タプルが挿入されるとき，9:00 の集約結果タプルは IoI から外れるため，データバッファから削除される．

On-demand Query 一般に OLAP 問合せにあたる各 lattice の頂点の総数は膨大な数になることが多いため，それら全てを Registered Query として対応する CQL 問合せを SPE に登録することは処理性能上難しく，また，データバッファにかかるメモリ領域も膨大なものとなり，不可能である．また，ユーザは常に全ての OLAP 問合せの結果を必要とするわけではなく，それらの中の一部であることが多い．

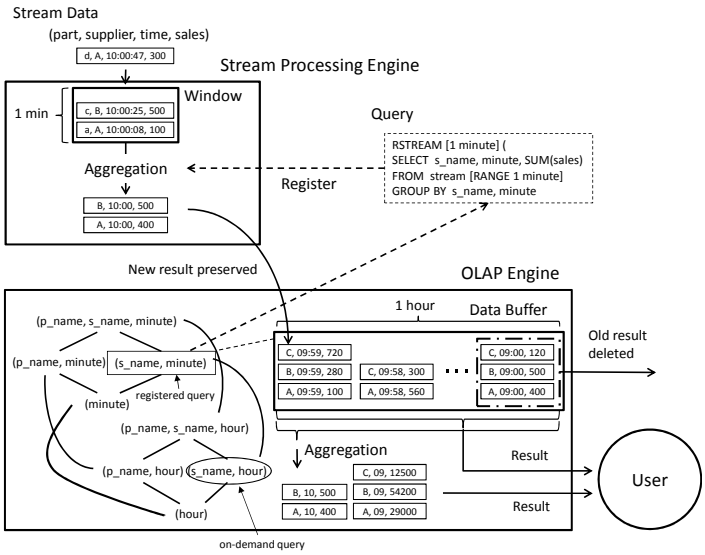


図 4: Stream OLAP アーキテクチャとその処理の流れ

そこで，オンデマンドに評価する問合せ処理手法を導入し，この手法による問合せを On-demand Query とする．これは，一部の OLAP 問合せのみを Registered Query とし結果を実体化しておき，それ以外の OLAP 問合せは，ユーザから集約結果を要求された時点で，OLAP 問合せ間の依存関係に基づいて，lattice 内においてその上位にある (より細かい粒度の) Registered Query の結果が格納されているデータバッファからデータを取得し，そのデータに対して集約演算を行うことで自身の集約結果を導く問合せである．

例えば図 4 において，(s_name, hour) の OLAP 問合せを On-demand Query であると仮定する．この問合せは，より細かい粒度の Registered Query の結果から自身の集約結果を導ける，すなわち，(s_name, minute) の問合せ結果を使って導くことができる．

ただし，もし，“minute” に対応する IoI が 30 分で，“hour” に対応する IoI が 1 時間であった場合，(s_name, hour) は，(s_name, minute) のデータバッファから自身の集約結果を作り出すことができない．そこで，lattice において，時間階層が一つ細かくなったときに，粗い方の時間階層の IoI が一つ細かい粒度の時間階層の IoI よりも短くなった場合，その時間階層における最も細かい粒度の集約の OLAP 問合せは必ず Registered Query とする．図 4 の例の場合，(p_name, s_name, minute) および，(p_name, s_name, hour) は，Registered Query とする．これにより，任意の On-demand Query において，自身の結果を導くための Registered Query のデータバッファが必ず存在することを保証する．

5. 問合せの最適化手法

5.1 問題の定式化

本節では，コストモデルに基づいて，OLAP 問合せを，Registered Query と On-demand Query に分類する最適化手法を考える．ここでは，OLAP 問合せ q_i の問合せ結果が参照される (ユーザが問合せ結果を要求する) 頻度 p_i を考慮する．これは，OLAP 問合せ q_i の最新の集約結果が単位時間当りに p_i 回要求されるということである．提案する最適化手法は，各問合せ q_i を Registered Query とするか，On-demand Query とするかを，実体化した結果を保持しておくための空間コストの制限の中で，問合せの処理コストが最小となるように決めるアルゴリズムである．

ただし，一般に，このような最適な組合せを求める問題は組合せ最適化問題となるため，我々はこの問題に対して貪欲アルゴリズムを用いる．次節から，このアルゴリズムにおいて用いられる処理コストと格納コストを説明する．

5.2 コストモデル

本節では，処理コストと格納コストを二種類の問合せそれぞれに対して説明する．

5.2.1 処理コスト

Registered Query Registered Query q_i は，データバッファ内から直接集約結果を得るため，OLAP 問合せの結果を取得するコストは無視できる．しかし，データバッファの管理，すなわち，データバッファへのタプルの挿入と削除にコストがかかる．実際の実装では，データバッファ内の最も古い集約結果タプルに，最新のものを上書きするようになっている．ここで，C_R を 1 タプルの上書きにかかる処理時間とする．また，各 OLAP 問合せ q_i に対して，登録される CQL 問合せの window サイズを w_i，その CQL 問合せの集約結果の平均タプル数を k_i とする．w_i 単位時間毎に，データバッファ内の，IoI の期間から外れる古いタプル k_i は，最新のタプル k_i によって上書きされる．よって，Registered Query q_i の単位時間あたりの処理コストは以下の式で表される．

$$C_R \frac{k_i}{w_i} \tag{1}$$

On-demand Query On-demand Query q_i は，Registered Query q_r の中から，より細かい粒度の集約を行う，自身の結果を導くことができる問合せ q_{r_i} を選び，そのデータバッファのタプルを用いる．このデータバッファ内の全タプルが集約対象となる．ここで，集約のコストは集約対象となるタプル数に比例すると仮定し，集約演算処理における 1 タプルあたりの処理時間を C_O とする．また，q_{r_i} の IoI の期間は I_{r_i} とする．q_{r_i} のデータバッファには平均で $\frac{I_{r_i} \cdot k_{r_i}}{w_{r_i}}$ 個のタプルが含まれているので，各 q_i の処理には，$C_O \cdot \frac{I_{r_i} \cdot k_{r_i}}{w_{r_i}}$ のコストがかかる．ただし，q_{r_i} に対応する CQL 問合せは，w_{r_i} 単位時間毎に，平均 k_{r_i} 個のタプルを生成するとする．以上から，On-demand Query q_i の単位時間あたりの参照回数を p_i としたときの q_i の単位時間あたりの処理コストは以下の式で表される．

$$C_O \cdot \frac{I_{r_i} \cdot k_{r_i} \cdot p_i}{w_{r_i}} \quad (2)$$

On-demand Query q_i が自身の結果を導くために用いる Registered Query q_{r_i} は，Registered Query q_r の中で処理コストが最小となるものが選ばれる．

以上から，全 OLAP 問合せの内，Registered Query が n 個，On-demand Query が m 個あるときの StreamOLAP における単位時間あたりの全体の処理コスト C は以下の式で表せる．

$$C = C_R \sum_{i=0}^{n} \frac{k_i}{w_i} + C_O \sum_{i=0}^{m} \frac{I_{r_i} \cdot k_{r_i} \cdot p_i}{w_{r_i}} \quad (3)$$

なお，本研究では，SPE 内部における集約演算にかかる処理コストは考えない．これは，OLAP エンジンのプロセスと SPE のプロセスが分かれており，このコストは本研究の対象外としているためである．

5.2.1 格納コスト

格納コストは主にデータバッファのサイズに依存する．ある Registered Query q_i のデータバッファ内の 1 タプルのサイズを s_i とするとき，全体の格納コスト S は以下の式で表される．

$$S = \sum_{i=0}^{n} \frac{I_i \cdot k_i \cdot s_i}{w_i} \quad (4)$$

5.3 最適化アルゴリズム

本節では，lattice 内の頂点にあたる OLAP 問合せを，Registered Query と On-demand Query に分ける最適化アルゴリズムを説明する．

このアルゴリズムの入力は，lattice 内の頂点の集合 Q とする．lattice 内の root にあたる頂点 (最も細かい粒度の集約) r を On-demand Query とすると，自身の結果を導く元の Registered Query がないため，r は常に Registered Query とする．出力は，アルゴリズムの結果，Registered Query と分類された OLAP 問合せの lattice の頂点の集合 Q_{out} である．Q_r と Q_o は，それぞれ Registered Query と On-demand Query の集合を表す．初期段階では，Q_r は lattice の root r のみを含み，その他の頂点は全て Q_o に含まれる．

まず現時点での処理コスト C_{pre} を計算する．全体の処理コスト $processCost(Q_r, Q_o)$ は，式 (3) によって計算される．次に，Q_o の中から 2 番目に Registered Query とする頂点を選ぶ．この頂点を仮に Registered Query とした時の Registered Query の集合を Q'_r，On-demand Query の集合を Q'_o とする．この時の処理コスト C を計算し，C と C_{pre} の差 $Benefit$ を求める．この処理を，Q_o の全頂点に対して一つずつ行い，その結果，$Benefit$ が最大となった頂点を Q_r に入れる．また，Q_r に頂点を入れた順番とその時の処理コスト C を配列 $minSeqC$ に加え，覚えておく．さらに，Q_r に頂点を加えると同時に，格納コスト S についても計算する．この格納コスト $storageCost(Q_r)$ は，式 (4) によって計算される．以上の Registered Query を選択していく処理を，格納コストの閾値 S_{max} を超えるまで繰り返す．ここで，繰り返しが終わった時点では閾値をちょうど超えた段階での頂点も Q_r に加わっているため，以降の処理では，最後に加えられた頂点は，Q_o の頂点のままとする．繰り返しが終了した後，Registered Query の OLAP 問合せとして選ばれた頂点の配列 $minSeqC$ の中で，最初の頂点 (root r) から，最も C の値が小さかった頂点までを Q_{out} とする．以上のアルゴリズムの擬似コードを Algorithm1 に示す．

Algorithm 1: Optimization Algorithm

```
Input: a set Q of vertices of the lattice, the root vertex r
       of the lattice (r ∈ Q)
Output: a set Q_out of vertices (Q_out ⊆ Q)
begin
    minSeqC ← newArray()
    S ← 0, Q_r ← {r}, Q_o ← Q\Q_r
    while sumS > S_max do
        C_pre ← processCost(Q_r, Q_o)
        Benefit_max ← −∞, C_min ← +∞, q_min ← null
        for q ∈ Q_o do
            Q'_r ← Q_r ∪ {q}, Q'_o ← Q\Q'_r
            C ← processCost(Q'_r, Q'_o)
            Benefit ← C_pre − C
            if Benefit > Benefit_max then
                Benefit_max ← Benefit
                C_min ← C, q_min ← q
            end
        end
        minSeqC.add((q_min, C))
        Q_r ← q_min, S ← spaceCost(Q_r)
    end
    min_cost ← minimum cost in minSeqC, Q_out ← {r}
    for i = 0 to minSeqC.length() do
        if minSeqC[i].C = min_cost then
            Q_out ← Q_out ∪ {minSeqC[i].d}
            break
        end
        Q_out ← Q_out ∪ {minSeqC[i].d}
    end
end
```

5.4 アルゴリズムの適用例

前節において説明した最適化アルゴリズムの適用例を示す．この例で用いるデータセットは，part，supplier，time，sales の 4 つの属性からなるものとする．ここで，part，supplier，time は次元であり，sales が事実表のメジャーである．各次元は，図 3 のような階層を持ち，lattice 内の各頂点のパラメータ値は表 1 に示してあるようなものであると仮定する．

表 1: 各頂点における OLAP 問合せの各種パラメータ値

Aggregate Dimension	w_i	k_i	p_i	s_i
p_name, s_name, minute	1	3000	0.7	56
p_name, minute	1	2300	0.4	36
s_name, minute	1	2500	0.8	36
minute	1	60	0.2	16
p_name, s_name, hour	60	2500	0.1	56
p_name, hour	60	2000	0.9	36
s_name, hour	60	2300	0.6	36
hour	60	10	0.3	16

この例では，I (IoI の時間幅) は，全 OLAP 問合せにおいて 3600 とし，格納コストの閾値 S_{max} を 1,000,000,000 と設定する．このような条件下で，どの OLAP 問合せ (lattice の頂点) を Registered Query とするかを決めていく．最初に root である，(p_name, s_name, minute) が選ばれる．次に，二番目に Registered Query とする頂点を Q_o の中から選ぶ処理を行う．仮

に，(p_name, minute) を Registered Query として選んだ場合，処理コスト C は以下のように計算される．ここでは簡単のために，$C_R = 2C_O$ と仮定する．Registered Query である (p_name, s_name, minute) のデータバッファは，On-demand Query である (s_name, minute)，(p_name, s_name, hour)，(s_name, hour) のために使われる．また，Registered Query である (p_name, minute) のデータバッファは，その他の On-demand Query のために使われる．

$$\begin{aligned} C &= 3000 \times 2 / 1 + 2300 \times 2 / 1 + 3000 \times 3600 \times 0.8 / 1 \\ &+ 2300 \times 3600 \times 0.2 / 1 + 3000 \times 3600 \times 0.1 / 1 \\ &+ 2300 \times 3600 \times 0.9 / 1 + 3000 \times 3600 \times 0.6 / 1 \\ &+ 2300 \times 3600 \times 0.3 / 1 \\ &= 27802600 \end{aligned}$$

このように，他の全ての On-demand Query の頂点について，Registered Query としたときの処理コスト C を計算する．この結果を表 2 に示す．

表 2: アルゴリズムにおける繰り返し 1 周目の C の結果

p_name, minute	27802600
s_name, minute	25031000
minute	30310920
p_name, s_name, hour	15396083
p_name, hour	22722066
s_name, hour	25967476
hour	32406000

表 2 から，C の値が最小となる頂点は，(p_name, s_name, hour) であることがわかる．よって，この頂点が root の次の Registered Query として選ばれる．以降，同様に Registered Query とする頂点を選んでいくと，(s_name, minute)，(p_name, minute) の順に Registered Query として選ばれていく．ここで，(p_name, minute) を Registered Query とした時点で，格納コスト S の値が 1,235,280,000 となり，閾値 S_{max} を超えるため，ここで，Registered Query の選択を終了する．

最後に，今まで Registered Query として選ばれた頂点が Registered Query として選ばれた時点での処理コストの値を比較する．この例の場合，(p_name, s_name, minute)，(p_name, s_name, hour)，(s_name, minute) はそれぞれ，35,646,000, 15,396,083，6,401,083 となる．本アルゴリズムは，この中で最も小さい値の頂点までを最終的な Registered Query とするので，これら三つの頂点全てが Registered Query となる．

6. 評価実験

6.1 実験概要

本実験は主に，StreamOLAP にかかる処理時間と消費メモリ量の測定を行う．4.2 節で説明したように，提案する StreamOLAP のアーキテクチャは SPE と OLAP エンジンから成る．実験では，SPE として，商用の SPE である日立製作所 (株) が開発した uCosminexus Stream Data Platform [14] を使用する．OLAP エンジンは，我々が一から実装を行った．実装における言語は Java を利用している．我々のプロトタイプのエンジンは，Registered Query に対するデータバッファ管理と，OLAP 問合せ結果の生成，On-demand Query に対する OLAP 問合せの結果の生成を行うことができる．本章では，このプロトタイプを使用した時の問合せの処理時間と消費メモリ量の測定を行う．

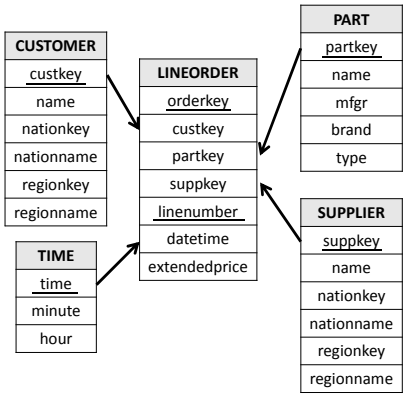
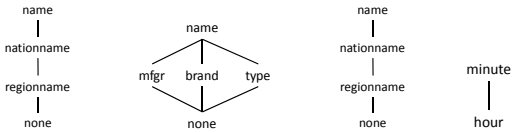


図 5: 実験に用いるデータセットのスタースキーマ



(a) customer (b) part (c) supplier (d) time

図 6: 実験に用いるデータセットの次元とその階層

実験に用いるデータセットは，データベースのベンチマークの一つである TPC-H [15] で用いるデータセットを本実験用に加工したものを用いる．この加工の方法に関しては，Patrick O'Neil らの論文 [16] を参考にした．実験では，TPC-H のデータセットにおける，lineitem および order の二つのテーブルを結合した "lineorder" テーブルを事実表として用いる．よって "lineorder" テーブルは，ストリームデータとなる．実際は "lineorder" テーブルは 6018 個の有限個のタプルを含むテーブルであるため，これらを SPE に循環させて流すことにより，擬似的にストリームデータを作成している．この事実表とその他四つの次元表のスタースキーマを図 5 に示す．また，それぞれの次元の階層を図 6 に示す．

提案手法の有効性を評価するために，Registered Query と On-demand Query への割当てを以下の 2 つの手法で行った場合と比較を行う．

Frequency-based: Registered Query q_i を，各 OLAP 問合せの参照頻度 p_i が高い順に選ぶ．

Random: Registered Query をランダムに選ぶ．

また，各 OLAP 問合せ q_i における参照頻度 p_i について，以下の 6 通りの与え方を行う．

Rand: 各 OLAP 問合せ q_i に対して参照頻度をランダムに与える．すなわち，[0, 1] の範囲で，p_i の値をランダムに割り当てる．(実験における単位時間は "分" なので，$p_i = 1$ は，1 分毎に問合せの結果が要求されることを意味する．)

AllHigh: 全 OLAP 問合せに対して高い参照頻度を与える．すなわち，[0.8, 1] の範囲で，p_i の値をランダムに割り当てる．

AllLow: 全 OLAP 問合せに対して低い参照頻度を与える．すなわち，[0, 0.2] の範囲で，p_i の値をランダムに割り当てる．

CoarseHigh: より粗い粒度の集約を行う OLAP 問合せに対し，より高い参照頻度を与える．すなわち，最も粗い粒度の OLAP 問合せに $p_i = 1$ を割り当て，他の問合せは，最も粗い粒度の問合せから lattice 内において 1 つ集約の粒度が細かくなる毎に，参照頻度を 0.01 ずつ下げながら割り当てる．

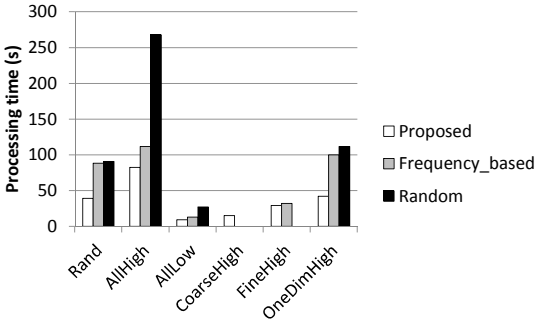


図 7: 実験結果 (I (1))

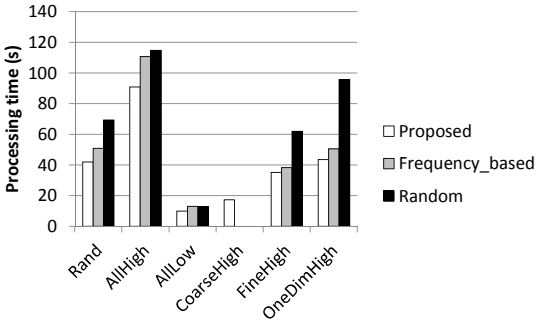


図 8: 実験結果 (I (2))

FineHigh: より細かい粒度の集約を行う OLAP 問合せに対し，より高い参照頻度を与える．すなわち，root に $p_i = 1$ を割り当て，他の問合せは，root から 1 つ集約の粒度が粗くなる毎に，参照頻度を 0.01 ずつ下げながら割り当てる．

OneDimHigh: root に最も高い参照頻度を割り当て，ある一つの次元を除いた次元の集約の粒度が粗くなるに従い，より低い参照頻度を与える．例えば，図 3 において，時間次元を粒度のレベルと参照頻度に関係のない次元とすると，(p_name, s_name, minute) と (p_name, s_name, hour) には，$p_i = 1.0$ が割り当てられ，(p_name, minute)，(s_name, minute)，(p_name, hour)，(s_name, hour) には，$p_i = 0.8$ が，(minute) と (hour) には，$p_i = 0.6$ が割り当てられる．本実験では，“customer” を粒度のレベルと参照頻度に関係のない次元とする．

I (IoI の時間幅) の割り当て方について，以下の 2 通りで行う．なお，I の単位は分である．

I (1): 全 OLAP 問合せにおいて，$I = 60$ を割り当てる．

I (2): “分” の問合せに $I = 60$，“時” に $I = 240$ を割り当てる．

また本実験では，式 (3) の Registered Query と On-demand Query における 1 タプル当りの処理時間 C_R，C_O について，$C_R = 5C_O$ の重み付けで実験を行う．これは，Registered Query と On-demand Query における 1 タプル当りの処理時間を測定した結果，2 倍から 10 倍程度の範囲で Registered Query の処理時間が長くかかったためである．

実験環境は以下の通りである．

- OS: CentOS 6.5 (Linux 2.6.32-431.el6.x86_64)
- メモリ: 31.1GB
- CPU: Intel (R) Core (TM) i7-3770
- クロック数: 3.40GHz

実験では，StreamOLAP を，“lineorder” ストリームを 10 分間流すことで評価する．ストリームデータの到着レートは 500tuple/s，格納コストの閾値 S_{max} は 1GB に設定する．

6.2 実験結果

図 7 に I (1) のときの，図 8 に I (2) のときの Frequency_based および Random の手法と提案手法の処理コストを比較したグラフを示す．x 軸は，6 通りの参照頻度の与え方を示している．y 軸は，Registered Query におけるデータバッファの更新 (タプルの上書き) にかかる処理時間と，On-demand Query における集約結果の構築にかかる処理時間を加算した総処理時間を示している．両図における白，灰，黒色の棒グラフはそれぞれ，Proposed, Frequency_based，Random の場合の処理時間を示している．これらの結果から，提案手法が Frequency_based，Random の場合よりも処理時間を短くできていることが分かる．なお，両図において CoarseHigh，FineHigh の結果の一部は存在しない．これは，Registered Query の数が多く，SPE の処理が追いつかなかったためである．

次に，空間コストについて比較を行う．結果は，いずれの手法においても消費されたメモリ量は等しくなった．これは，格納コストの閾値を全ての場合において 1GB としたためである．

7. まとめと今後の課題

本研究では，SPE と OLAP エンジンからなる StreamOLAP アーキテクチャを提案した．SPE は，連続的に流れてくるストリームデータに対する集約結果を生成し，OLAP エンジンは，OLAP 問合せに対する結果の生成を効率的に行うために，集約結果を実体化しメモリ領域に保持しておく．また，このアーキテクチャに基づき，各 OLAP 問合せに対する結果の返し方を決めるためのコストモデルに基づいた最適化アルゴリズムを提案した．ここでは，OLAP 問合せの結果を，SPE の集約演算によって問合せ結果を実体化しておくものと，ユーザ要求に応じてオンデマンドに，他の実体化された問合せ結果を集約処理することで自身の結果を導くものの二種類に分類する．このような環境において，メモリサイズの上限を超えない範囲で，問合せにかかる処理コストを最小限にする最適化アルゴリズムを示した．最後に，評価実験により，提案手法の最適化アルゴリズムを用いることにより，各問合せの頻度順やランダムに分ける方法よりも計算にかかる時間を抑えられていることを示した．

今後の課題としては，SPE から受け取る集約結果が膨大であったり，IoI の時間の長さが非常に長い場合への対応がある．この場合，データバッファに必要となるメモリ領域がメモリの最大サイズを超えてしまうことが考えられる．この問題に対処するためには，OLAP 問合せの結果をディスクやフラッシュメモリに保持すると共に，効率的に問合せ処理できるような手法を考える必要がある．

[謝辞]

この研究の一部は，文科省「実社会ビッグデータ利活用のためのデータ統合・解析技術の研究開発」による．

[文献]

[1] S. Chaudhuri et al., “An Overview of Data Warehousing and OLAP Technology”, ACM SIGMOD Record, 26 (1), pages 65-74, March 1997.

[2] 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.

[3] Storm, http://storm-project.net/.

[4] L. Neumeyer et al., “S4: Distributed Stream Computing Platform”, In Data Mining Workshops (ICDMW), 2010 IEEE conference on, pages 170-177, 13 Dec. 2010.

[5] 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.

[6] A. Arasu et al., "The CQL continuous query language: semantic foundations and query execution", The VLDB Journal, 15 (2):121-142, 2006.

[7] 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.

[8] K. Aouiche et al., "Data Mining-based Materialized View and Index Selection in Data Warehouses", Journal of Intelligent Information Systems, Vol. 33, Issue 1, pages 65-93, August, 2009.

[9] Z. A. Talebi et al., "Exact and Inexact Methods for Selecting Views and Indexes for OLAP Performance Improvement", Proc. 11th International Conference on Extending Database Technology (EDBT 2008), pages 311-322, Nantes, France, March, 2008.

[10] R. J. Santos et al., "PIN: A Partitioning & Indexing Optimization Method for OLAP", Proc. 9th International Conference on Enterprise Information Systems (ICEIS 2007), Funchal, Madeira, Portugal, June, 2007.

[11] V. Harinarayan et al., "Implementing Data Cubes Efficiently", Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD 1996), pages 205-216, ACM, Montreal, Canada, June 1996.

[12] C. Joslyn et al., "View Discovery in OLAP Databases through Statistical Combinatorial Optimization", Scientific and Statistical Database Management Lecture Notes in Computer Science, Vol.5566, 2009, pages 37-55.

[13] R. Zhang et al., "Multiple Aggregations Over Data Streams", Proc. 2005 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD 2005), pages 299-310, ACM, Baltimore, Maryland, USA, June 14-16, 2005.

[14] uCosminexus stream data platform, http://www.hitachi.co.jp/Prod/comp/soft1/cosminexus/sdp/ .

[15] TPC-H, http://www.tpc.org/tpch/ .

[16] P. O'Neil et al., "The Star Schema Benchmark and Augmented Fact Table Indexing", First TPC Technology Conference (TPCTC 2009), Lyon, France, Aug. 24-28 2009.

[17] 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.

中挟 晃介 Kosuke NAKABASAMI
2013 年筑波大学情報学群情報科学類卒業．2015 年同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博士前期課程修了．ストリームデータ処理，OLAP 等の研究に従事．日本データベース学会，情報処理学会，各会員．

北川 博之 Hiroyuki KITAGAWA
1978 年東京大学理学部物理学科卒業．1980 年同大学理学系研究科修士課程修了．日本電気（株）勤務の後，1988 年筑波大学電子・情報工学系講師．同助教授を経て，現在，筑波大学システム情報系教授，ならびに計算科学研究センター教授．理学博士（東京大学）．データベース，情報統合，データマイニング，情報検索等の研究に従事．日本データベース学会会長，情報処理学会フェロー，電子情報通信学会フェロー，ACM，IEEE，日本ソフトウェア科学会，各会員．

Salman Ahmed SHAIKH
2005 年パキスタン Mehran University of Engineering and Technology 卒業，2008 年同大学 Communication Systems and Networks 修士課程修了，2014 年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博士後期課程修了．博士（工学）．現在，筑波大学計算科学研究センター研究員．ストリーム処理，ビッグデータ処理，トランザクション処理，不確実データ，データマイニング等の研究に従事．日本データベース学会 (DBSJ)，International Association of Computer Science and Information Technology (IACSIT)，Pakistan Engineering Council (PEC)，各会員．

天笠 俊之 Toshiyuki AMAGASA
筑波大学システム情報系准教授．データ工学，データベース，Web マイニング等の研究に従事．日本データベース学会，情報処理学会，ACM 各会員．電子情報通信学会，IEEE 各シニア会員．