セルフチューニングによる連続的問合せの適応型問合せ最適化

渡辺　陽介† 　北川　博之†,††

† 筑波大学システム情報工学研究科
†† 筑波大学計算科学研究センター
〒 305-8573 茨城県つくば市天王台 1-1-1
E-mail: watanabe@kde.cs.tsukuba.ac.jp, kitagawa@cs.tsukuba.ac.jp

あらまし　現在，データストリームに対する問合せ処理方式として連続的問合せが注目されており，大量の連続的問合せを効率的に実行するための手法が求められている．我々の研究グループでは，これまでに連続的問合せに対する複数問合せ最適化方式を提案してきた．本方式では，実行パターンの類似する問合せ同士をクラスタリングし，クラスタごとに問合せ中の共通演算の共有化を行っている．過去の実験から，クラスタ分割の閾値を与えるパラメータによって，処理効率が大きく変化することが示されたが，最適なパラメータ値の設定法については検討されていなかった．本稿では，これまでの手法を拡張し，データストリームの特性変化に応じて最適なパラメータ値を自動的に推定する，適応的な複数問合せ最適化方式を提案する．

キーワード　ストリーム，最適化，問合せ処理

Self-Managing Query Optimization Method for Multiple Continuous Queries

Yousuke WATANABE† and Hiroyuki KITAGAWA†,††

† Graduate School of Systems and Information Engineering, University of Tsukuba
†† Center for Computational Sciences, University of Tsukuba
Tennohdai 1-1-1, Tsukuba, Ibaraki, 305-8573 Japan
E-mail: watanabe@kde.cs.tsukuba.ac.jp, kitagawa@cs.tsukuba.ac.jp

Abstract　Continuous query is widely noticed as a scheme for query processing over data streams, and efficient methods for processing multiple continuous queries are needed. Our research group has proposed a multiple query optimization method for continous queries. In our method, the system forms clusters of queries which have similar execution patterns, and derives query plans sharing the result of common operators. According to our previous experiments, we have shown that a parameter value in the clustering phase controls divisions of clusters and has a great impact on efficiency of query processing. However, the optimal parameter value needs to be decided by trial and error. This paper extends our previous work. The proposed method automatically estimates the optimal value, and iteratively adjusts it even if properties of underlying data streams dramatically change.

Key words　Stream, Optimization, Query Processing

1. はじめに

近年，ネットワークの発達により，ニュースや天気予報などの情報をオンラインで提供するサービスが増加している．また，無線センサーネットや GPS などのデバイス技術の進歩から，実世界からのセンサーデータが容易に利用可能となってきている．このような時々刻々と変化する情報を提供する情報源はデータストリームと呼ばれており，従来のデータベースとは異なる新しい情報源として注目されている．データストリームでは情報源の側から能動的な情報配信が行われ，利用者は次々と到着する情報を扱わなければならない．データストリームの数と種類は増え続けており，人手による取捨選択は非常に困難になってきている．そのため，データストリームに対する問合せ処理の重要性が高まっている．

データストリームへの問合せ処理の枠組みとして，連続的問合せ (Continuous Query) [1]〜[4], [8] が提案されている．連続的問合せは，システムにあらかじめ問合せを登録し，情報源から次々と到着するデータに対して差分的に処理結果を生成していくというものである．データ到着に応じた処理は従来のアクティブデータベースでも ECA ルール [9] によって実現できた

```
MASTER master_source_1 ,...
SELECT attr_1 ,...
FROM source_1 { [windowsize_1] } ,...
WHERE conditions
```

図 1　問合せ記述形式

Fig. 1　Syntax of continuous query

が，連続的問合せでは差分的な問合せ処理が行われるため，より軽量かつ高速に実行でき，データストリームの処理に適したものとなっている．現在，連続的問合せを処理するシステムの構築が重要となっており，特に，大勢の利用者からの問合せ要求を処理可能なシステムを実現するために，複数の連続的問合せを効率的に実行する手法が求められている．

我々の研究グループでは，これまでに連続的問合せを対象とした複数問合せ最適化方式を提案してきた [13], [15], [16]．複数問合せ最適化とは，与えられた問合せ群から，共通する演算を抽出し，それらの処理結果を共有することによって全体の処理効率を向上させる手法である．我々の提案した最適化手法では，連続的問合せの実行パターンの違いに着目し，例えば近いタイミングで実行されることの多い問合せ同士のような，実行パターンの類似する問合せのクラスタを生成することによって，クラスタごとに共有可能な演算を抽出する．過去に行った実験 [13], [15] により，問合せのクラスタリングにおけるクラスタ分割の閾値を決めるパラメータの値に応じて，問合せ処理の効率の改善度合いが大きく変化することを示した．

過去の研究によって最適化手法の特徴は明らかにされたが，実際に最適なパラメータ値を決定するための手法については，これまで具体的に検討されていなかった．最適なパラメータ値は，問合せの種類や，演算の選択率，データの傾向などの複雑な要素によって決まり，容易には推定できない．また，データストリームでは，到着するデータの性質やパターンが時間とともに変化しうるため，最適値が絶えず変化する可能性がある．よって，事前に最適なパラメータ値を設定するというアプローチは現実的でない．そこで本研究では，クラスタリングのパラメータの最適値を自動的に調節する手法を提案し，ストリームの特性が変化しても追随可能となるような適応的複数問合せ最適化手法を提案する．また，評価実験により，提案した手法の有効性の評価を行う．

本稿は以下のような構成となっている．まず 2. において，連続的問合せについて簡単に説明し，3. で我々の複数問合せ最適化手法にいて概説する．4. では，本稿が扱う問題を明確にするため，クラスタリングのパラメータが処理効率に与える影響について説明する．5. で，提案するパラメータの設定法について述べ，6. で評価実験の結果を示す．7. において関連研究について述べた後，8. でまとめと今後の課題を述べる．

2. 連続的問合せ

連続的問合せは，データストリームから到着したデータに対して連続的に問合せ処理を行い，前回評価時の処理結果との差分に相当する結果を生成するものである．我々のデータストリーム処理システム [13], [15], [16] では，データストリームからの配信単位をリレーションのタプルとして扱っており，問合せ要求は図 1 に示す SQL ライクな構文に基づいて与えられる．問合せ記述における MASTER 節は，その問合せの実行タイミングを指定するためのものである．実際のストリームの利用において，新規データの到着や時間の経過などのイベントに連動して連続的問合せを実行することは必要な機能であり，このようなイベント指定を含む連続的問合せは, [7], [14] などでも提案されている．FROM 節中では，タプルを保持する期間を指定するためのウインドウ幅 [6] が指定可能で，各タプルは到着時刻からウインドウ幅分だけ時間が経過するまで問合せ処理に使用される．

図 2 の問合せ Q1,Q2,Q3 は，同じ企業に関する株価情報 (Quote) とニュース情報 (News) のストリームを統合する問合せ例である．問合せ中の MASTER 節の記述から，Q1 は 12 時，Q2 はニュースの到着時点，Q3 は 0 時に実行される．ただし，Clock_12 や Clock_0 はタイマーからの入力を表すストリームであるとする．FROM 節中の “[9hour]” がウインドウの幅の指定で，ここでは 9 時間のウインドウを表す．

3. 連続的問合せに対するの複数問合せ最適化

ここでは，我々の研究グループがこれまでに提案した複数問合せ最適化手法 [13], [15], [16] についての概要を述べる．

3.1 基本的アイデア

イベント指定を含む連続的問合せでは，問合せごとに実行タイミングが異なっている．そのため，実行タイミングの大きく離れた問合せ同士では，共通演算であっても全く異なる結果を生成する可能性がある．例えば，図 2 の Q1 と Q3 では，生成される結果が全く異なり，共通部分がないことがわかる．我々は，共通演算が共有可能な中間結果を生成する場合として，以下の 2 つを想定している．

1 つ目は，確実に同時に実行されることが保証された問合せ同士の場合で，例えば全く同一の要求を記述した問合せ同士などである．実行タイミングが同時であるかは，問合せ記述の MASTER 節が同一であるかどうかを解析することにより検出できる．我々はこのような関係にある問合せの集合を**ベースグループ**と呼んでおり，複数問合せ最適化における基本単位としている．

2 つ目は，近いタイミングで実行されることが多い問合せの集合で，例えば図 2 では Q1 と Q2 のような問合せ同士である．実行タイミングが違っていても，短い間隔で続けて実行される問合せ同士ならば，共通演算が共有可能な結果を生成する場合が多い．ただし，タイマーのような到着時刻の明確なストリームが MASTER 節に指定されていない限りは，実際に問合せを実行してみるまで実行タイミングが近いかどうかを知ることはできない．問合せの実行パターンをオンラインで収集・分析することにより，実行タイミングの類似した問合せを検出する必要がある．

我々の最適化手法では，まず問合せ集合をベースグループへ分類し，さらにベースグループ中の各問合せの実行パターンを解析することにより，実行タイミングの類似したベースグルー

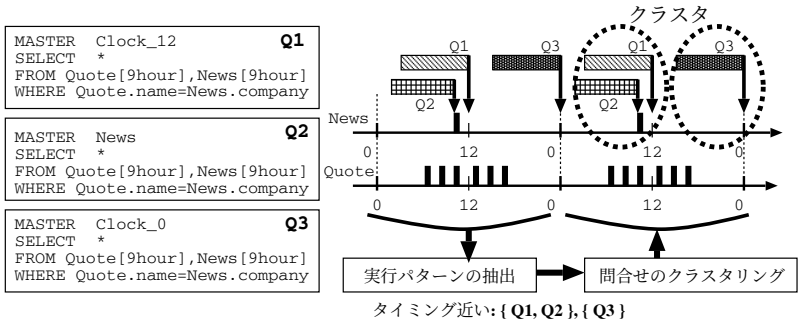

図 2　最適化手法の概要

Fig. 2　An example senario for explaining our optimization method

プの集合を生成している．

3.2　実行パターンに基づくクラスタリング

まず，問合せ単体の実行パターンについて説明する．問合せ Q のストリーム S_i 上における実行パターン $EP(Q,S_i)$ は以下のように与えられる．

$$EP(Q,S_i) = \bigcup_{m \in MS} \{s.TS \mid s \in S_i \wedge s.TS \in [m.TS - win,\ m.TS]\}$$

ただし，MS は問合せ Q の MASTER 節に記述されたストリームからの到着データの集合とし，win は問合せ Q における S_i のウインドウ幅を表すものとする．上記の通り，本研究における問合せの実行パターンとは，問合せが実行時に参照したストリーム上のタプルの到着時刻集合である．到着時刻集合の情報は実際の問合せ処理を行いながら収集される．

次に，ベースグループの実行パターンについて説明する．ベースグループ B のストリーム S_i 上における実行パターン $EP(B,S_i)$ は，以下の通り，ベースグループ中の各問合せの実行パターンの和集合で与えられるものとする．

$$EP(B,S_i) = \bigcup_{Q \in B} EP(Q,S_i)$$

必要な主記憶領域の削減と処理時間の短縮のために，実際の実行パターンの収集およびクラスタリングは，ベースグループ単位で行われる．

ベースグループのクラスタリングは，実行パターンの類似度に基づいて行われる．ベースグループ B_1, B_2 の実行パターンの類似度は以下の式により定義される．

$$similarity(B_1,B_2) = \min_{S_i \in (F_{B_1} \cap F_{B_2})} \frac{|EP(B_1,S_i) \cap EP(B_2,S_i)|}{|EP(B_1,S_i) \cup EP(B_2,S_i)|}$$

この式は，ベースグループの類似度は，各実行パターン内の要素の共通部分の割合として定義されることを表している．ただし，F_{B_k} はベースグループ B_k 中の各問合せが FROM 節で参照しているストリームの和集合である．

実行パターンの情報を一定期間収集し，類似度の計算を行った後，ベースグループのクラスタリングを行う．クラスタリングの手法には通常の階層的クラスタリング [11] を用いており，このときパラメータ θ により同一クラスタ内に含まれるベースグループ同士の類似度の最小値が決定される．θ の与える影響については，4. で詳しく述べる．図 2 は，{ Q1, Q2 } と { Q3 } という 2 つのクラスタを生成した例である．

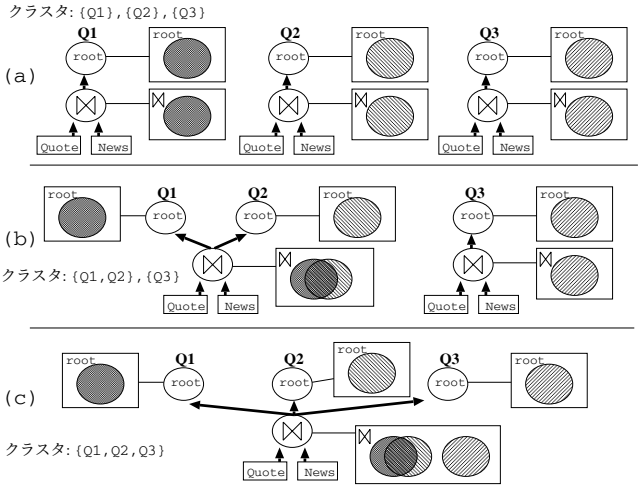

図 3　θ の違いによる問合せプランの変化

Fig. 3　Query plans with different θ

3.3　演算の共有

同じクラスタに含まれる問合せの共通演算は，共有可能な処理結果を生成することが保証されている．クラスタの生成後は，実際に問合せから共通演算を抽出し，中間結果を共有するような問合せプランの生成を行う．

共有された演算では，すべての問合せの要求を満たすような処理結果を生成する．例えば，図 2 の Q1, Q2 間で演算を共有した場合 (図 3(b))，共有された演算は Q1 と Q2 の両方の MASTER 節の指定に基づいて起動する．また，そのとき生成された中間結果は，演算を共有するすべての問合せの上位の演算の入力キューへ追加される．

4.　クラスタリングパラメータ θ の重要性

θ は，実行パターンの類似度に基づいてクラスタリングを行う際に，同一クラスタ内の要素間の類似度の最小値を与えるパラメータである．一般的なクラスタリングでは，θ を低く設定すると，類似度の低い要素同士を多く含む大きなクラスタが少数生成され，θ を高く設定すると，類似度の高い要素だけを含む小さなクラスタが多数生成される．図 3（a），(b)，(c) は，それぞれ，θ を高い値にした場合 (ex. $\theta = 1.0$)，中間的な値にした場合 (ex. $\theta = 0.5$)，低い値にした場合 (ex. $\theta = 0.0$) のクラスタ構成の例である．θ の値に応じて生成されるクラスタ

の構成が変化し，それによって演算の共有に影響を与えている．

演算の共有にはメリットとデメリットがあり，共有により削減される処理と，共有したことにより新たに追加される処理がある．

（1）演算の共有により削減される処理　問合せ集合内に複数個ある演算を共有化することで冗長な処理が減り，同じタプルを何度も処理する必要がなくなる．また，演算の入力キューも共有化されることにより，タプルの追加・削除などを行うキューの管理コストが削減される．一般に，1 つの演算をより多くの問合せで共有する方が有利であるため，問合せ数の多いクラスタを生成した方が共有によるメリットは高くなる．

（2）共有の副作用で追加される処理　複数の問合せから共有された演算では，すべての問合せの要求を満たすような中間結果を生成しなければならない．そのため，中間結果にはある問合せにとっては必要でも，他の問合せにとっては不必要であるようなタプルが含まれている可能性がある．よって，各問合せ中の上位の演算において，中間結果から自分の要求を満たさないタプルを調べ，廃棄する処理を行う必要がある．この処理は，問合せ同士で共有できない中間結果が生成されるほど増加するため，類似度の低いクラスタは不利となる．

クラスタの構成を変えることによって，これらの 2 種類の処理の量が変化する．ここでは図 3 を用いて説明する．図 3（a）では演算がまったく共有されないため，冗長な処理が発生している可能性があるが，各問合せにとって不必要な処理結果は生成されていない．図 3（b）では，Q1 と Q2 で結合演算を共有することにより，冗長な処理を削減している．ただし，お互いが必要とする中間結果の一部に共有できないものがあるため，上位の演算でその部分を廃棄しなければならない．図 3（c）では，さらに Q3 とも結合演算を共有している．しかし，Q3 の処理結果には Q1,Q2 と共有可能なものがないために，それぞれにとって不必要となる中間結果が大幅に増えている．

演算の共有によって処理効率を向上させるには，（1）によって削減される処理コストが，（2）によって増加する処理コストを上回らなければならない．処理効率を最も向上させるためには，最適な θ を設定することが極めて重要となる．5. では，適切な θ を設定するための我々のアプローチについて述べる．

5. 提 案 手 法

どの値が最適な θ となるのかは，問合せに含まれる演算の種類および個数，データストリームの性質，計算アルゴリズムなどの多くの要素に依存して決まるため，正確な値を推定することは容易ではない．最も確実な方法は，様々な θ の値をシミュレーション等によって試すという手法である．しかし，データストリームの性質が時間と共に変化してしまうような状況では，最適な θ が変動するため，事前に見積もった値は一切役に立たなくなってしまう可能性がある．そこで本研究では，問合せ処理に関する情報を実行時に収集しながら，θ の値を繰り返し調節し，徐々に最適値に近づけていくというアプローチをとる．

5.1 θ 調節のための基準情報

本研究では，マスタ情報源からタプルが到着してから，問合

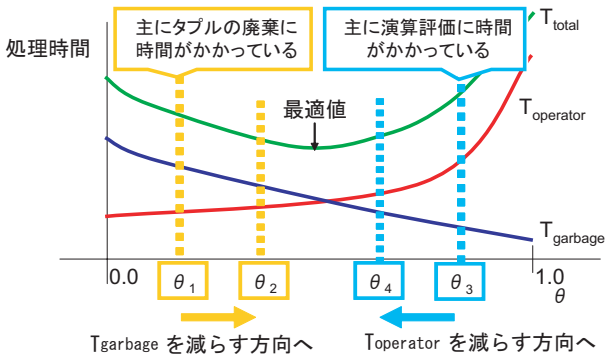

図 4　θ の自動調節

Fig. 4　Strategy for adjusting θ

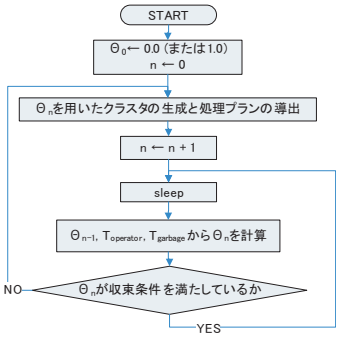

図 5　提案手法のフローチャート

Fig. 5　A flowchart diagram of our method

せ処理が完了するまでにかかる総処理時間の平均 T_{total} を最小化するような θ を最適値と定義する．実行時に収集する情報は，問合せ中の全演算の評価にかかる処理時間の平均 $T_{operator}$ と，中間結果内の不要タプルの検出および廃棄にかかる処理時間の平均 $T_{garbage}$ である．これらの平均処理時間は，現在時刻から期間 T まで過去の間に発生した処理から計算されるものとする．

実際にはその他の処理も存在するが，$T_{operator}$ や $T_garbage$ に比べて無視できる程度の時間である．よって，総処理時間 T_{total} は，$T_{operator}$ と $T_{garbage}$ の和にほぼ等しい．

5.2　調節アルゴリズム

パラメータ調節の基本的な考え方を以下に示す．提案手法では，$T_{operator}$ と $T_{garbage}$ の 2 つのうち，よりボトルネックになっている方を重点的に減らしていくことで，T_{total} を小さくしていく戦略をとる（図 4）．

- $T_{garbage}$ よりも $T_{operator}$ の方が長いときは，問合せ処理全体で演算評価にかかるコストが大きいということである．演算の総数を削減するために，より多くの問合せで 1 つの演算を共有するように，θ の値を減らす方向へ動かす．
- $T_{operator}$ に対して，$T_{garbage}$ の方が長いときは，演算の共有によるメリットよりも不要タプルの廃棄によるデメリットが勝っているということになる．互いに共有できない中間結果が生成されることを抑えるために，θ の値を高くする方向へ動かす．

実際には，以下の式により一定間隔で θ を繰り返し計算する．

CPU	UltraSparcII 296MHz x2
OS	Solaris9
メモリ	1GB
Java	J2SE1.4.1

表 1　実 験 環 境

Table 1　Experiment environment

```
MASTER   Master_i                    (i ∈ {0, ···, 29})
SELECT   *
FROM     Stream_1[w], Stream_[w]
WHERE    Stream_1.attr = Stream_2.attr
```

図 6　問合せテンプレート

Fig. 6　Template of queries

$$\theta_n = \theta_{n-1} + \frac{T_{garbage}}{T_{garbage} + T_{operator}} - \frac{T_{operator}}{T_{garbage} + T_{operator}}$$

θ_n は n 回目の更新後の値を表し，θ_{n-1} は前回の更新時のパラメータ値を表す．ただし，θ は [0, 1] の範囲を超えないものとする．また，問合せ処理開始時点でのパラメータ θ_0 は，0 または 1 とする．これは，開始時点では実行パターンの情報が収集されていないため，中間的な大きさのクラスタを生成できないことによる．

なお，本研究では θ が最適値と思われる値に収束したことを判定する条件を以下のように定める．

$$|\theta_n - \theta_{n-1}| < \varepsilon$$

$T_{operator}$ が極めて大きいときには，θ は 0 に収束し，$T_{garbage}$ が極めて大きいときは θ は 1 に収束する．$T_{operator}$ と $T_{garbage}$ が同程度である場合には，θ は両者がほぼ均衡するような点に収束する．一度収束条件を満たしても，データストリームの性質が変化し，θ の最適値が変わるような事態が発生した場合には，上記の収束条件を満たさなくなる．そのため，収束条件の評価を定期的に行い，条件を満たさなくなったときにはパラメータの再調整を開始する．

θ を更新した後は，その値を用いてベースグループのクラスタを再構成し，演算の共有をやりなおす．その部分の処理については，これまでに提案してきた複数問合せ最適化手法 [15], [16] におけるクラスタ再構成の処理と同様である．

図 5 に，提案手法による θ の更新の処理手順を示す．

6. 評 価 実 験

提案手法を用いて適切なパラメータ θ が推定可能であることを確認するため，評価実験を行った．

6.1 実 験 環 境

実験環境は，表 1 の通りである．実験データはすべて人工的に生成したもので，統合対象となる 2 種類のストリームと，異なる実行タイミングを与える 30 種類のストリームを用意した．各ストリームは 10 分おきに 1 度データを配信する．

問合せは図 6 のテンプレートを元に，2 種類のストリームの結合処理を行うものを 2000 個生成した．これら 2000 個の問合せは，30 種類のベースグループへ分類される．

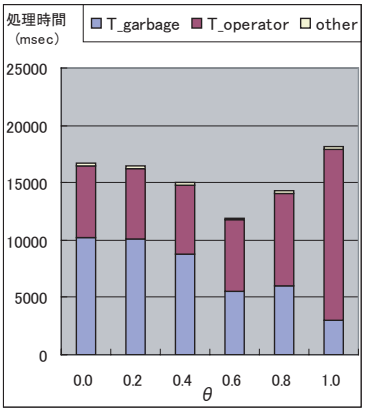

図 7　実験 1 における θ と処理時間の関係

Fig. 7　θ vs. processing times (exp.1)

パラメータの自動調節においては，θ の初期値を 0 とし，収束条件の ε を 0.1，θ の更新チェック間隔を 10 分とした．

6.2 変化の起きない状況における実験

まずはじめに，ストリームからの到着データのパターン変化がおきない状況において，提案手法によって θ を最適値に設定できるかどうかを調べた．ここでは，入れ子ループ結合によって結合演算の処理に時間がかかる場合（実験 1）と，ハッシュ結合を用いて結合演算の処理を高速化し，かつタプルの到着率を増やして大量の中間結果が生成されるようにした場合（実験 2）の 2 種類の実験を行った．どちらの場合でも，事前に最適な θ を知るために，いくつかの θ の値で実際に処理を行い，それぞれの処理時間を測定した．

6.2.1 実　験　1

はじめに，実験 1 の結果を示す．事前に調査した θ に対する処理時間の変化は図 7 の通りである．横軸は θ の値，縦軸は処理時間の平均値である．処理時間の内訳として $T_{garbage}$ と $T_{operator}$ およびそれ以外の処理にかかる時間を示してある．$\theta = 1.0$ では，$T_{operator}$ の割合が高く，また，$\theta = 0.0$ では，$T_{garbage}$ の割合が高くなっていることが確認できる．

提案手法を用いて θ の自動調節を行いながら，問合せ処理を行った結果を図 8 に示す．3 回の更新が発生し，θ の値が，0, 0.158, 0.432, 0.631 と変化した．事前に収集した図 7 のデータから，0.6 近辺は最適値と考えられる．θ の値の調節に伴って処理効率が向上していることも確認できる．

6.2.2 実　験　2

次に実験 2 の結果を示す．図 9 は，事前に調査した θ に対する処理時間の変化である．ハッシュ結合により結合演算のコストが下がったため，全体的に $T_{garbage}$ の割合が高くなっている．

θ の自動調節を行った結果を図 10 に示す．4 回の更新が行われ，θ の値が，0, 0.306, 0.586, 0.815, 0.991 と変化し，図 9 から最適値と思われる 1.0 近辺に落ち着いた．処理時間も改善が見られた．

6.3 パターン変化の発生する状況における実験

実験 1，2 により，データストリームのパターンや特性に変化のない状況では，提案手法を用いてパラメータ θ を適切な値

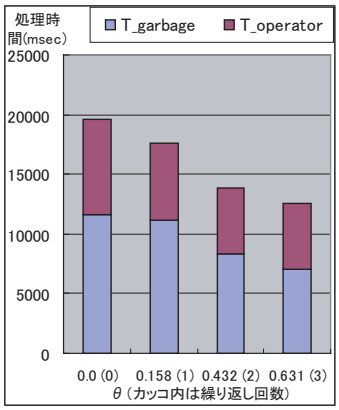

図 8　実験 1 における自動調整の結果

Fig. 8　Result of automatic adjustment(exp.1)

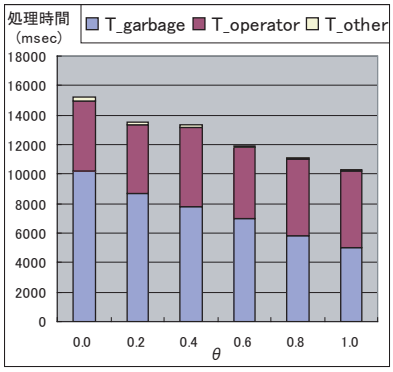

図 9　実験 2 における θ と処理時間の関係

Fig. 9　θ vs. processing times(exp.2)

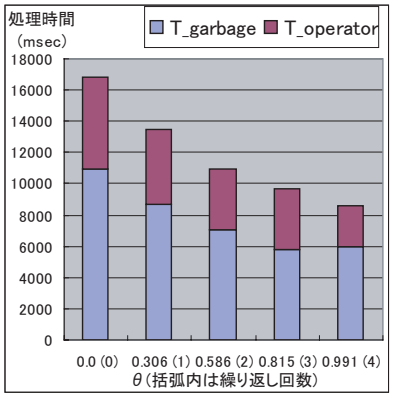

図 10　実験 2 における自動調整の結果

Fig. 10　Result of automatic adjustment(exp.2)

に設定できることが示された．ここからは，ストリームからのデータ到着のパターンや特性が変化する状況下での実験結果を示す．実験 3 は到着するデータの性質が変わる場合の実験，実験 4 は到着パターンの変化する場合の実験である．どちらの実験でも結合演算の処理には入れ子ループ結合を用いた．

6.3.1　実　験　3

この実験では，実験開始から 60 分経過後にストリームから到着するデータの特性が変化し，それにより問合せに含まれている結合演算の選択率が 0.025 から 0.075 に変化するものとした．選択率の増加により，生成されるタプル数が増加し，不要なタプルの廃棄処理にかかる時間 $T_{garbage}$ の割合が高くなることが予想される．演算の選択率が途中で変化した場合の提案手法の動きを調べた．

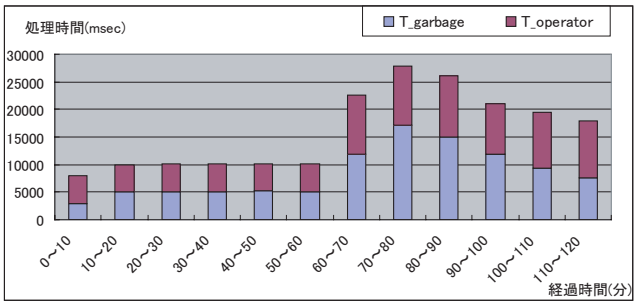

図 11　ストリームデータの性質変化に対するパラーメタ調節

Fig. 11　Result of automatic adjustment over changing properties of streams

チェック時刻	10	20	30	40	50	60
更新前 θ_{n-1}	0.000	0.000	0.000	0.000	0.000	0.000
見積値 θ_n	0.000	0.009	0.020	0.000	0.042	0.000
更新の有無	無	無	無	無	無	無

チェック時刻	70	80	90	100	110
更新前 θ_{n-1}	0.000	0.000	0.196	0.441	0.441
見積値 θ_n	0.012	0.196	0.441	0.524	0.340
更新の有無	無	有	有	無	無

表 2　実験 3 における θ の変化

Table 2　Change of θ(exp.3)

実験結果を図 11 に示す．また，この実験における θ の変化と毎回の更新チェック時における θ の見積値を表 11 に示す．実験開始から 60 分経過するまでは，$T_{operator}$ と $T_{garbage}$ はほぼ同程度であり，θ は 0 から動かない．60 分が経過し，到着データの性質が変化すると，$T_{operator}$ と $T_{garbage}$ のバランスが崩れ，徐々に θ が増加する．最終的には 0.441 で収束した．θ の変化に伴って，処理時間が減っていることが図 11 から確認できる．

6.3.2　実　験　4

実験 4 では，途中からストリームの到着パターンが変化し，それによって問合せの実行パターンが変化する状況における評価を行った．実行パターンの変化に応じてクラスタの再構成が必要となるが，提案手法がそれを正しく検出し，実行できるかを調べた．実験開始から 70 分経過した時点で，問合せの MASTER 節のストリームの到着パターンが変化し，それまでとは異なるパターンで到着するものとした．

実験結果を図 12 に示す．また，θ の調節結果と，毎回の更新チェック時の見積値を表 3 に示す．パターンの変化前 (70 分以前) は θ が 0.421 で収束している．パターン変化の発生した 70 分経過直後に，$T_{operator}$ と $T_{garbage}$ の均衡が崩れ，それにより θ の再調整およびクラスタの再構成が行われている．この場合は，θ を 1 回更新しただけですぐに収束している．

本実験から，提案手法は到着パターンの変化を最適な θ の変化として検出し，クラスタの再構成を行えることが確認できた．

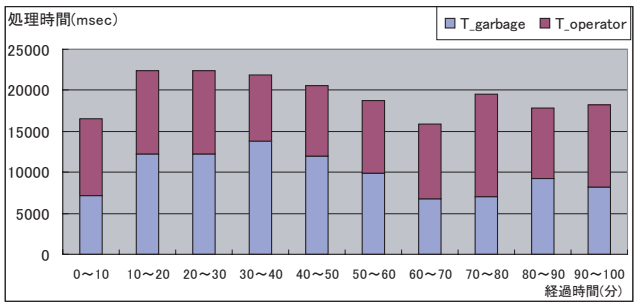

図 12　到着パターン変化に対するパラーメタ調節

Fig. 12　Result of automatic adjustment over changing arrival patterns

チェック時刻	10	20	30	40	50	60
更新前 θ_{n-1}	0.000	0.000	0.000	0.000	0.207	0.421
見積値 θ_n	0.000	0.089	0.081	0.207	0.421	0.454
更新の有無	無	無	無	有	有	無

チェック時刻	70	80	90
更新前 θ_{n-1}	0.421	0.097	0.097
見積値 θ_n	0.097	0.000	0.075
更新の有無	有	無	無

表 3　θ の変化（実験４）

Table 3　Change of θ(exp.3)

7. 関 連 研 究

従来の RDB における複数問合せ最適化 [10], [12] では，対象となる問合せはバッチなどでほぼ同時に実行されることが前提となっていた．そのため，実行タイミングの異なる問合せ間での演算の共有は全く考慮されていない．また，コスト見積りの方法も RDB とストリームでは異なっている．

データストリームに対する連続的問合せの処理システムとして，NiagaraCQ [4], STREAM [8], TelegraphCQ [3], Aurora [1] などがある．これらのシステム中でも，演算の共有は行われているが，我々の提案手法のように問合せの実行パターンに着目し，中間結果の共有度合いに応じて問合せを動的にグループ分けしているものはない．

複数問合せ最適化では，異なる選択条件を持つ問合せで演算を共有する場合，各条件の OR をとった制約のゆるい選択演算を共有するという手法をとるが，その副作用として不必要なタプルが発生することがある．Precision Sharing [5] はそのような，冗長な処理の削減と不必要なタプル生成の回避の問題を扱っている．本研究に共通する部分もあるが，実行タイミングの異なる問合せ間で演算を共有することを目的とした研究ではない．

本研究は，これまでに提案した連続的問合せの複数問合せ最適化手法 [13], [15], [16] におけるパラメータの設定の問題を解決するための手法を提案するものである．本手法により，ストリームの特性変化が起きた場合でも適切なパラメータを与えることができる．

8. まとめと今後の課題

本稿では，我々が以前提案した複数問合せ最適化手法を拡張し，クラスタリングのパラメータ θ を，自動的に調節するための手法を導入した．提案手法は，実行時に得られる情報を元に，徐々に θ を動かし，最適と思われる値へ近づけていくというアプローチである．また，評価実験により，提案手法を用いて問合せの処理時間を改善できることを示した．

今後の課題としては，より詳細な評価実験があげられる．具体的には，ストリームと RDB との統合を含む場合のような多様な問合せを用いた実験や，実データに対しても有効であるかの評価等である．また，パラメータの調節誤りがおきるケースについての検討などを行う予定である．

謝辞　本研究の一部は，日本学術振興会特別研究員奨励費 (15・330)，科学研究費補助金特定領域研究 (2)(#16016205)，基盤研究 (B)(#15300027) による．

文　　献

[1] D. J. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. “Aurora: a New Model and Architecture for Data Stream Management,” VLDB Journal Vol.12, No.2, pp. 120–139, 2003.

[2] A. Arasu, S. Babu and J. Widom. “The CQL Continuous Query Language: Semantic Foundations and Query Execution,” Technical Report, http://dbpubs.stanford.edu/pub/2003-67 , 2003.

[3] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. Shah. “TelegraphCQ: Continuous Dataflow Processing for an Uncertain World,” Proc. Conference on Innovative Data Systems Research 2003.

[4] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. “NiagaraCQ: A Scalable Continuous Query System for Internet Databases,” Proc. SIGMOD 2000, pp. 379–390.

[5] S. Krishnamurthy, M. J. Franklin, J. M. Hellerstein, and G. Jacobson. “The Case for Precision Sharing,” Proc. VLDB 2004, pp. 972–986.

[6] J. Kang, J. F. Naughton, and S. D. Viglas. “Evaluating Window Joins over Unbounded Streams,” International Conference on Data Engineering, 2003.

[7] L. Liu, C. Pu, and W. Tang. “Continual Queries for Internet Scale Event-Driven Information Delivery,” IEEE Trans. Knowledge and Data Engineering, vol.11, no.4, pp.610–628, 1999.

[8] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma. “Query Processing, Resource Management, and Approximation in a Data Stream Management System,” Proc. Conference on Innovative Data Systems Research 2003.

[9] N. W. Paton and O. Diaz. “Active Database Systems,” ACM Comp. Serv., 31(1), 1999.

[10] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe. “Efficient and Extensible Algorithms for Multi Query Optimization,” Proc. ACM SIGMOD Conference, pp. 249–260, 2000.

[11] G. Salton. “Automatic Information Organization and Retrieval,” McGraw-Hill Book Company, 1968.

[12] T. K. Sellis. “Multiple-Query Optimization,” ACM Transaction on Database Systems, 13(1), 1988, pp. 23–52.

[13] Y. Watanabe, and H. Kitagawa. “A Multiple Continuous Query Optimization Method Based on Query Execution Pattern Analysis,” Proc. DASFAA 2004, LNCS 2973, pp. 443-456, 2004.

[14] W. Xue, and Q. Luo. “Action-Oriented Query Processing for Pervasive Computing,” Online Proc. CIDR, pp. 305–316, 2005.

[15] 渡辺陽介，北川博之．“連続的問合せに対する複数問合せ最適化手法”, 電子情報通信学会論文誌，Vol. J-87-D-I, No. 10, pp. 873–886.

[16] 渡辺陽介，北川博之．“ストリームの動的特性変化を考慮した連続的問合せ最適化方式”，DBWS 2003.