DEWS2003 5-C-02

連続的問合せに対する複数問合せ最適化

渡辺　陽介† 　北川　博之††

† 筑波大学 システム情報工学研究科 〒 305–8573 茨城県つくば市天王台 1–1–1
†† 筑波大学 電子情報工学系 〒 305–8573 茨城県つくば市天王台 1–1–1
E-mail: †watanabe@kde.is.tsukuba.ac.jp, ††kitagawa@is.tsukuba.ac.jp

あらまし　今日，情報配信の形態としてデータ放送やプッシュ型情報サービスなどのデータストリームが注目されており，その数と種類が増加している．そのため，複数のストリーム型情報源の統合利用の重要性が高まっている．データストリームから必要なデータを抽出したり加工するための手段として，連続的問合せがある．多数のストリーム型情報源に対する多数の連続的問合せが与えられた際，その効率的実行が要求される．本稿では，そのためのアプローチとして，連続的問合せに対する複数問合せ最適化方式を提案する．本研究が想定する複数のデータストリームの統合を行う環境では，連続的問合せ中の演算においてウインドウなどの時間条件を用い，かつ利用者がその情報を必要とするタイミングで提供することが必要である．このような連続的問合せは，同一の演算であっても実行タイミングによって全く異なる結果を生成し得るため，従来のバッチ処理などを想定した複数問合せ最適化手法をそのまま適用することは困難である．本提案手法は，実行タイミングの違いによる問合せの参照範囲の違いを考慮し，参照範囲が近い同士の問合せをグループ化することにより効率的な実行処理プランを導出する．
キーワード　放送型サービスと DB, 問い合わせ処理, アクティブ DB, 情報統合

Multiple Query Optimization for Continuous Queries

Yousuke WATANABE† and Hiroyuki KITAGAWA††

† Graduate School of Systems and Information Engineering, University of Tsukuba, Tennohdai 1–1–1, Tsukuba, Ibaraki, 305–8573 Japan
†† Institute of Information Sciences and Electronics, University of Tsukuba, Tennohdai 1–1–1, Tsukuba, Ibaraki, 305–8573 Japan
E-mail: †watanabe@kde.is.tsukuba.ac.jp, ††kitagawa@is.tsukuba.ac.jp

Abstract　Recent development of network technologies has enabled us to use a variety of data streams, and a demand for integrating multiple data streams has been increasing. Continuous queries are often used to process data streams. When we have many continuous queries for many data streams, their efficient execution is an important research issue. In this paper, we propose a novel multiple query optimization scheme for this objective. In the data stream integration environments, we usually need to use temporal conditions as expressed by windows to obtain relevant information, and to provide information according to the required timing. In this situation, the same operator may generate completely different results when it is evaluated at different time instances. Therefore, we cannot directly apply existing multiple query optimization techniques. The proposed scheme forms clusters of continuous queries based on their execution time, and applies a groupwise multiple query optimization technique.
Key words　Broadcast services and DB, Query Processing, Active DB, Information Integration

1. ま え が き

近年，情報配信形態として，データ放送やプッシュ型情報源のようなデータストリームが注目されており，その数や種類が増加している．そのため，複数のストリーム型情報源の統合利用の重要性が高まっている．今後，多数のユーザがストリーム型情報源を利用するようになると，ストリーム型情報源の統合システムでは膨大な数の問合せが発生することが予想される．このため，大量の問合せ要求を効率的に処理可能なストリーム型情報源の統合方式が求められている．

ストリーム型情報源から到着するデータを処理する枠組みとして，連続的問合せ (Continuous Query) [7], [15] が注目されて

いる．連続的問合せは，ストリーム型情報源から到着した情報に対して繰り返し問合せ処理を実行し，前回の実行時からの差分となる結果を生成するものである．これまでにいくつかの連続的問合せの処理法 [3], [7], [10], [15] が提案されている．

本稿では，多数の連続的問合せを効率的に実行するための手段として，連続的問合せに対する複数問合せ最適化方式を提案する．大きく分類すると，連続的問合せには，情報源から情報が到着するとそれに連動して実行される到着型問合せと，タイマーによって一定間隔で定期的に実行されるタイマー型問合せの 2 種類がある．従来の連続的問合せに対する研究でもタイマー型問合せと到着型問合せは考慮されていたものの，それぞれに対して別個の実行方式や問合せ記述が用いられる等，統一的に扱われていない場合が多かった．本研究では，定期的にアラームを送信するような仮想的なストリーム型情報源であるクロックストリームと，マスタ情報源の概念を導入することにより，タイマー型問合せをクロックストリームからの情報の到着に連動して実行される到着型問合せとして統一的に扱う．一方，連続的問合せ中で記述可能な条件についても既存の研究における取り扱いは必ずしも同一ではない．本研究で想定する複数データストリームの統合環境では，時間的に近いデータ同士を結合するなどの時間条件を用いることが一般に必要である．本研究では，ストリーム型情報源に対する結合演算としてウインドウ結合 [6] を用いることを許し，それを含めた連続的問合せの複数問合せ最適化について検討する．

複数問合せ最適化とは，複数の問合せで共通な演算や中間結果を共有することを目的とするもので，これまでにもいくつかの研究が行われている [9], [12], [14]．しかし，リレーショナルデータベース等における従来の複数問合せ最適化は，バッチ型の問合せ処理など，あらかじめ同時に実行されることが分かっている問合せ群のみを対象としてきた．連続的問合せは，外部のイベントに連動して実行されるため，あらかじめ問合せの実行タイミングを完全に把握することは困難である．このため，既存の複数問合せ最適化方式をそのまま連続的問合せに適用することはできない．しかし一方，同じストリームからのデータ到着に連動して実行される連続的問合せ群や，短時間の間に続いて実行されるようなスケジュールを持つ連続的問合せ群の中では，演算の一部の共有化を図ることが可能である．本研究では，ストリーム型情報源からのデータ到着のパターンならびに各問合せの実行タイミングとその参照データ範囲に着目し，演算やその結果を共有できる可能性が高い問合せ同士をグループ化の対象とするような連続的問合せに対する複数問合せ最適化方法を提案する．

本稿の構成は次の通りである．まず 2. では，本研究の関連研究について説明する．3. では，本研究が対象とする連続的問合せについて述べる．そして，4. で提案する複数問合せ最適化手法について説明する．5. では提案手法の最初のステップである問合せのクラスタリングについて述べ，6. で次のステップであるクラスタ単位の複数問合せ最適化について述べる．

2. 関 連 研 究

連続的問合せに関するこれまでの主な研究について述べる．

Tapestry [15] は，追加のみ可能なリレーショナルデータベースに対する連続的問合せをサポートする．ストリーム型情報源も情報の追加のみしか発生しないリレーションとみなすことができる．Tapestry における連続的問合せは，WHERE 節に時刻の条件を含む SQL 問合せで，タイマー型である．OpenCQ [7] は分散異種情報源に対する情報統合システムである．OpenCQ の連続的問合せは，問合せ，発火条件，終了条件の 3 つから構成されており，発火条件が真になると，終了条件が満たされるまで問合せが繰り返し実行される．OpenCQ では，到着型とタイマー型の両方の問合せがサポートされている．Tapestry と OpenCQ においては，複数問合せ最適化に関する検討は行われていない．

NiagaraCQ [2], [3] は，到着型問合せとタイマー型問合せの両者を対象とした複数問合せ最適化手法を提案している．彼らの複数問合せ最適化方式は，連続的問合せがインクリメンタルに追加・削除される状況を想定しており，既に存在する問合せに対する実行プランをベースに処理の方法を決定することが特徴である．NiagaraCQ で対象とする連続的問合せは，ウインドウ結合のような時間条件を伴う演算や問合せの実行タイミングを考慮していない．そのため，上記のインクリメンタルな処理を除いては，基本的には従来の複数問合せ最適化が適用できる状況となっている．しかし，ストリーム型情報源に対する問合せにおいて時間条件を全く考慮しないのは非現実的であり，その場合には NiagaraCQ の最適化方式を直接適用することはできない．

CACQ [10] も大量の連続的問合せを処理することを目的としたシステムである．CACQ では eddy [1] を用いて，情報源の性質の変化やフィルタ条件の変化に対して演算の適用順序を変えるという，適応性の高い問合せ処理を実現している．しかし，CACQ は到着型問合せのみを対象としており，タイマー型問合せが考慮されていない．また，その演算評価方法は処理可能な演算を次々と評価する形で行われるため，ウインドウ結合のような時間条件を用いた演算があり，利用者が指定したタイミングで情報を提供する場合には，無駄な処理が発生することがある．

本研究の複数問合せ最適化方式が対象とする連続的問合せでは，クロックストリームとマスタ情報源の概念の導入により，タイマー型問合せと到着型問合せを統一的に扱っている．また，ウインドウ結合演算を含んだ問合せも考慮に入れており，各問合せの実行タイミングと参照範囲の重なり具合に着目した複数問合せ最適化を行う点が特徴である．

他にも連続的問合せに関する研究 [4], [8] はあるが，本研究の対象とは異なる．

連続的問合せ以外にも，ストリーム型情報源に対する問合せ処理を実現する枠組みとしては，トリガや ECA ルール [11] がある．連続的問合せと ECA ルールの比較については, [7] などで述べられている．TriggerMan [5] は，大量のトリガを実行す

るために，共通の構造を持った選択条件を効率よく評価可能な索引構造を提案している．

リレーショナルデータベースを対象とする複数問合せ最適化方式としては, [10], [12], [14] がある．これらは，問合せ中の演算に着目して，共通な演算の実行結果を共有するというものである．問合せ集合から共通な部分式を探索する処理はコストがかかるといわれているが, [12] では最適なプラン探索のためのヒューリスティクスに基づいたアルゴリズムが提案されている．本論文の提案方法における第 2 ステップのクラスタ単位の複数問合せ最適化では，基本的には [12] のアルゴリズムに基づく処理を行う．

3. 連続的問合せ

本節では，本研究が対象する連続的問合せについて述べる．

3.1 基本モデル

本研究ではストリーム型情報源がリレーションとしてモデル化されていることを想定する．情報源から送られてきたデータを，そのリレーションの 1 タプルとして扱い，情報の到着時刻 TS がタプルに付加されるものとする．

連続的問合せのセマンティックスについて述べる．連続的問合せ $CQ(E,M)$ は，リレーショナル演算式 E と情報源の集合 M から構成される．ただし，E は選択演算と結合演算のみを含む式とする．M に含まれる情報源をマスタ情報源と呼ぶ．リレーショナル演算式 E は，マスタ情報源のうちの 1 つから情報が届く度に実行される．

$$execution\_times_{CQ} = \bigcup_{I \in M} \{t | t \in I.TS\}$$

1 回の問合せの実行でユーザに返される結果は，前回までの実行結果の差分である．

$$\Delta E(t_i) = E(t_i) - \bigcup_{1 \leqq j \leqq i-1} E(t_{j-1})$$

ただし，$E(t)$ は時刻 t に演算 E を実行した時の結果を表し，$t_i \in execution\_times_{CQ}$ とする．問合せの実行によって生成された結果 ΔE は，ただちにユーザへ提供される．

3.2 問合せ形式

本研究では連続的問合せを，以下のような形式で記述する．

CREATE *query_name*
MASTER *master_source,...*
[**WINDOW SIZE** *window_size*]
SELECT–FROM–WHERE

CREATE 節には，問合せの識別子となる名前を記述する．MASTER 節は，マスタ情報源を指定する．3.1 で述べたとおり，連続的問合せはマスタ情報源の情報の到着時に実行される．WINDOW SIZE 節には，複数のストリーム型情報源間でウインドウ結合 [6] 等を実行する際のウインドウの範囲を記述する．ウインドウとウインドウ結合については 3.3 で述べる．SELECT–FROM–WHERE は SQL の構文と同様である．ただし本研究では，WHERE 節は AND で結合された条件のみとし，ネストした問合せや集約演算，OR などは考慮していない．

```
CREATE Example1
MASTER Quote
SELECT *
FROM Quote
WHERE Quote.name = 'A'
```

図 1 問 合 せ 1

```
CREATE Example2
MASTER Quote
WINDOW SIZE 6hour
SELECT *
FROM Quote, News
WHERE Quote.name=News.name
```

図 2 問 合 せ 2

連続的問合せの例を示す（図 1）．問合せ 1 は株価情報を配信するストリーム型情報源 Quote から，A 社に関する情報が到着したらそのデータを提供して欲しいという要求を表している．

3.3 ウインドウ結合

ストリーム型情報源の情報は，一般には時間の経過と共に重要度が低下し，また，離れた時期に届いたものほど関連が少ないことが多いため，非常に新しい情報と非常に古い情報を統合したいという要求は現実には少ない．さらに，ストリーム型情報源では，サービス提供中は逐次にタプルが到着するため，過去に到着したすべての情報を結合演算の処理対象とした場合，非常に大量のタプルを扱わなければならない．

このようなことを背景に，複数のストリーム型情報源に対する統合演算としてウインドウ結合が提案されている [6]．本研究における連続的問合せにおいても，ウインドウ結合を考慮の対象とする．ウインドウとは，結合演算の適用対象となるタプルを選択するための時間範囲のことである．ウインドウの大きさは問合せの WINDOW SIZE 節で与える．情報源ごとに違う大きさのウインドウを用いることも可能であるが，本研究では簡略化のため，1 つの問合せの中ではすべての情報源に対して同じ大きさとする．ウインドウの大きさを無限大にしたウインドウ結合は，これまでに到着したすべてのタプルを対象とする通常の結合演算に等しい．

論理的には，マスタ情報源のタプルの到着時刻 t を用いた以下の選択演算を各情報源 I に対して行った後，結合を計算することに相当する．

$$\sigma_{I.TS \in [t-windowsize,t]}(I)$$

単一リレーションを使用する問合せに対してウインドウが与えられた場合は，上記の選択演算として解釈する．

図 2 の問合せは，ウインドウ結合を含んだ問合せの例である．この要求は，Quote の情報が届いたときに，過去 6 時間以内に届いたストリーム型情報源 News の情報と Quote の情報から同じ企業に関するものを結合して提供して欲しいというものである．この要求のウインドウの大きさは 6 時間である．ただし，hour は 1 時間の長さを表す定数であるとする．

3.4 クロックストリーム

本研究では，クロックストリームの概念を導入することで，

```
CREATE Example3
MASTER Clock0
WINDOW SIZE 6hour
SELECT *
FROM Quote, News
WHERE Quote.name=News.name
```

図3　問 合 せ3

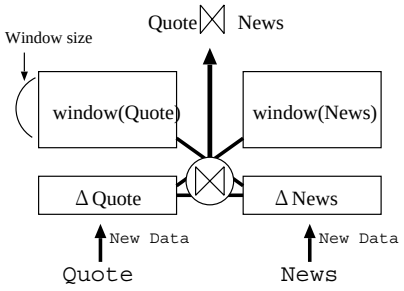

図4　Window Join

```
CREATE Example4
MASTER Clock12
WINDOW SIZE 8hour
SELECT *
FROM Stream1, Stream2, Stream3
```

図5　問 合 せ 4

```
CREATE Example5
MASTER Clock13
WINDOW SIZE 8hour
SELECT *
FROM Stream1, Stream2, Stream4
```

図6　問 合 せ 5

```
CREATE Example6
MASTER Clock0
WINDOW SIZE 8hour
SELECT *
FROM Stream1, Stream2, Stream5
```

図7　問 合 せ 6

従来のタイマー型問合せをも対象とする．クロックストリームは処理系が提供するストリーム型情報源であり，決まった時刻に周期的にアラームを送信する．クロックストリームをマスタ情報源として用いることで，問合せを定期的に実行することができる．

従来の連続的問合せでは，タイマー型問合せと到着型問合せは別の種類の問合せとして扱われていたが，本研究における連続的問合せではこれらを統一的に取り扱う．後述する複数問合せ最適化においても，タイマー型問合せと到着型問合せを分けて扱うことなく最適化することができる．

図3の例は，毎日0時に過去6時間分のQuoteとNewsの情報を統合して提供して欲しいというクロックストリームを用いた問合せである．ただし，Clock0は毎日0時にアラームを送信するクロックストリームである．

3.5　問合せの実行モデル

連続的問合せは，マスタ情報源のタプルが到着すると実行され，前回の実行からの差分がユーザへ提供される．ストリームから送られてきたタプルに対する差分を計算するため，到着したタプルはその情報源に対応したデルタリレーションへ一旦格納される．複数の問合せがあるときは問合せごとにデルタリレーションが用意され，到着したタプルはそれぞれのデルタリレーションに格納される．以下では，デルタリレーションを $\Delta(source\_name, query\_name)$ と表記する．マスタ情報源でない情報源のタプルは到着してもすぐには処理されず，デルタリレーションに蓄積されていく．問合せはデルタリレーション中のタプルに対して適用され，演算を適用し終わったデルタリレーションは毎回消費される．

ウインドウ結合を計算するためには，ウインドウの時間範囲に含まれるタプルをすべて保管しておく必要がある．そのための保管場所としてウインドウリレーションを用いる．以降では，ウインドウリレーションを $window(source\_name, query\_name)$ と表記する．

結合演算の計算は，各情報源のデルタリレーションとウインドウリレーションを用いて行われる．例えば，先程の問合せ2におけるウインドウ結合では，前回からの差分は次のようにして得られる．

$$\Delta(Quote, Example2) \bowtie window(News, Example2)$$
$$\bigcup window(Quote, Example2) \bowtie \Delta(News, Example2)$$
$$\bigcup \Delta(Quote, Example2) \bowtie \Delta(News, Example2)$$

処理を適用し終わったデルタリレーションのタプルは，ウインドウリレーションに移され，デルタリレーションからは削除される．また，ウインドウリレーションに含まれているタプルは時間が経つと古くなり，ウインドウの時間範囲から外れる．ウインドウリレーションに古くなった不要なタプルが存在するかどうかは，ウインドウ結合の実行直前にチェックされ，古くなったタプルがあればその時点で削除される．デルタリレーションに格納されているタプルでも，マスタ情報源の到着を待っている間にウインドウの時間範囲外へ出てしまうことがあるが，これもウインドウ結合の実行前に削除される．

4.　複数問合せ最適化手法

以降では，複数の連続的問合せが与えられたときの問合せ最適化方式について述べていく．まず，我々のアプローチの基本的アイデアを説明する．

ここでは，図5, 図6，図7のような3つの問合せの最適化を考える．例えば問合せ4は「毎日12時に過去8時間分のStream1とStream2とStream3を提供して欲しい」という要求である．マスタ情報源であるクロックストリームの違いから，問合せ4は毎日12時，問合せ5は毎日13時，問合せ6は毎日0時にそれぞれ実行される．これら3つの問合せは，共通する処理として $Stream1 \bowtie Stream2$ を含んでおり，この演算の計算結果を共有することで効率化が期待できる．

しかし，各問合せがウインドウ結合によって参照するデータの範囲を調べると，図8のような関係になっていることがわかる．問合せ4と問合せ5は，問合せの実行タイミングが近く，参照する範囲に重なりがあるが，問合せ6は実行タイミングが離れているため，ほかの2つの問合せの参照範囲と

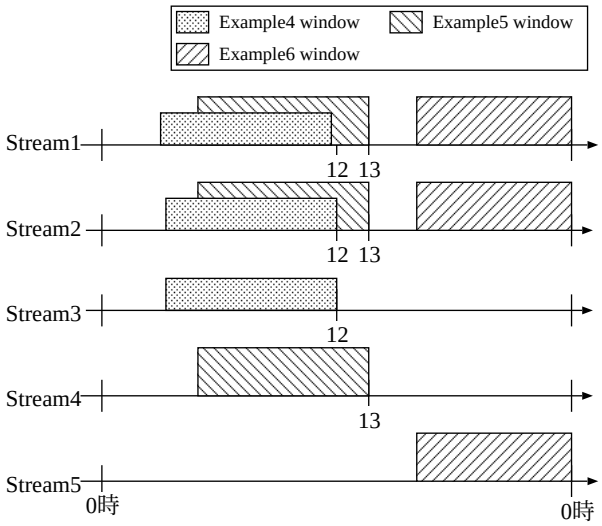

図 8　参照範囲の比較

の重なりが存在しない．このことから，問合せ 4 が生成する $Stream1 \bowtie Stream2$ の処理結果と問合せ 5 が生成する処理結果には同じものが含まれる可能性が高いが，問合せ 6 の生成する結果には他の 2 つと同じものが含まれる可能性が全くないことがいえる．

複数問合せ最適化の効果は，他の問合せが生成した結果をどの程度再利用できるかによって決まるため，問合せ 4 と問合せ 5 の演算を共有することは処理効率の向上を見込めるが，問合せ 6 と他の 2 つで演算を共有することでは処理効率の向上はありえない．逆に，再利用できない無駄な中間結果のスキャンが発生する分だけ，効率低下を招く可能性がある．よってこの場合，問合せ 4 と問合せ 5 だけ共有し，問合せ 6 を別にする方式が望ましい．

このように，実行タイミングが異なる問合せを最適化するためには，問合せが参照するデータの範囲を計算し，生成される結果がどの程度共有できそうかを調べることが必要である．本研究の連続的問合せの複数最適化方式は，この点を考慮し，次の 2 つのステップから構成される．

（1） 与えられた問合せの集合から，参照範囲の重なりが大きい問合せを集めたクラスタを生成する．

（2） クラスタごとに共有可能な演算を探し，最適な問合せ実行プランを導出する．

ステップ（1）の手順については 5. で，ステップ（2）の手順については 6. でより詳細に述べる．

5. 問合せのクラスタリング

5.1 参照範囲の類似度

はじめに，クラスタ作成の基準となる問合せ同士の参照範囲に基づく類似度を定義する．

クロックストリームのように到着時刻を正確に把握することができるマスタ情報源であれば，問合せの実行タイミングが特定できるので，4. の例のような方法でウインドウの重なり具合を調べられる．しかし一般に，外部のストリーム型情報源をマスタ情報源とする場合，その到着時刻は完全には予測できない

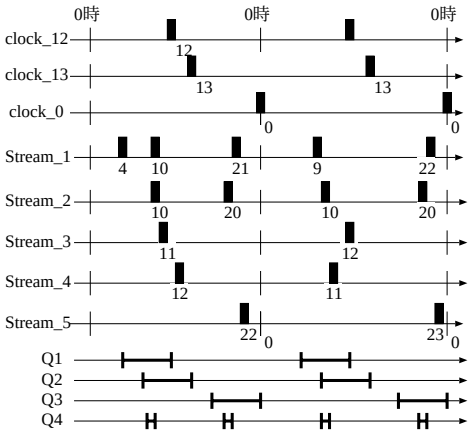

図 9　情報源のサンプル（サンプル期間=2day）

場合が多い．この場合，問合せ同士の参照範囲の重なりをあらかじめ正確に調べることは不可能である．そこで本研究では，各情報源からのデータ到着ログを用いて問合せの実行タイミングをシミュレートし，参照範囲の重なりを予測する．

まず，ある期間 T におけるデータ到着ログを解析し，期間内に届いた各情報源のタプルの到着時刻の集合を用意する．情報源 $I_i(1 \leqq i \leqq n)$ に対する到着時刻集合を S_i とする．次に，問合せ $Q_j(0 \leqq j \leqq m)$ をこれらの到着時刻情報に基づいて実行した場合に，Q_j が参照するタプルの到着時刻の集合を求める．これは Q_j のマスタ情報源の到着時刻集合の和集合 MS_j の各到着時刻について Q_j を実行したときに，各回の実行で Q_j のウインドウの参照範囲内に含まれた S_i の到着時刻の集合を計算することによって得られる．

$$USE_{ij} = \bigcup_{s \in MS_j} \{t | t \in S_i \wedge t \in [s - windowsize_j, s]\}$$

ただし，$windowsize_j$ は問合せ Q_j の WINDOW SIZE 節で指定されたウインドウの大きさであり，WINDOW SIZE 節を持たない選択演算のみの問合せの場合には 0 とする．

USE_{ij} を元に，問合せ Q_a,Q_b の参照範囲の類似度 $similarity(Q_a, Q_b)$ を計算する．本研究では，以下の式により類似度を求める．

$$similarity(Q_a, Q_b) = \min_{I_i \in Ref(Q_a) \cap Ref(Q_b)} \frac{|USE_{ia} \cap USE_{ib}|}{|USE_{ia} \cup USE_{ib}|}$$

この式は，Q_a,Q_b が共に使用する情報源において，最低でもどの程度参照範囲に共通部分があるかを表している．ただし，$Ref(Q)$ は問合せ Q の FROM 節に記述された情報源の集合を表すものとする．共通に利用する情報源がない場合 $(Ref(Q_a \cap Ref(Q_b) = \emptyset)$ の類似度は 0 とする．同じマスタ情報源を持ち，かつウインドウサイズが等しい問合せ同士の類似度は 1 になる．

クラスタ生成のため，全ての問合せのペアについて類似度を計算する．m 個の問合せに対しては，計算結果として $m \times m$ の対称行列 A（類似度行列）が得られる．例えば，図 9 のようなサンプルに対して表 1 の問合せ集合の類似度行列を計算した場合，図 10 のような結果が得られる．行列の (k,l) 成分は $similarity(Q_k, Q_l)$ の値に対応している．

ID	Master	Ref(Q)	Window
1	Clock12	Stream1,Stream2,Stream3	8hour
2	Clock13	Stream1,Stream2,Stream4	8hour
3	Clock0	Stream1,Stream2,Stream5	8hour
4	Stream2	Stream2	0
⋮	⋮	⋮	⋮

表 1　問合せ集合

$$\begin{bmatrix} 1.00 & 0.67 & 0.00 & 0.50 & \cdots \\ 0.67 & 1.00 & 0.00 & 0.50 & \\ 0.00 & 0.00 & 1.00 & 0.00 & \\ 0.50 & 0.50 & 0.00 & 1.00 & \\ \vdots & & & & \ddots \end{bmatrix}$$

図 10　類似度行列

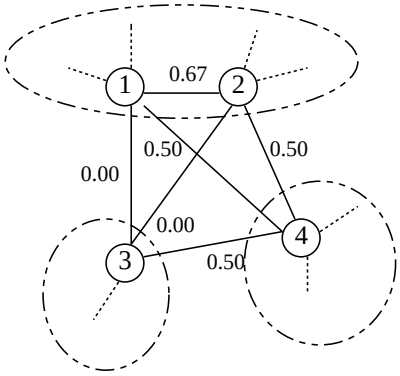

図 11　$\theta = 0.6$ のときのクラスタ

5.2　クラスタの生成と更新

計算された類似度行列から，類似度の高い問合せ同士のクラスタを生成する．ただし，クラスタ内に含まれる問合せの類似度の最小値が閾値 θ 以上になるようにクラスタを生成する．クラスタ生成のアルゴリズムは，通常のクラスタリング手法 [13] と同様であるので，詳細は省略する．

クラスタ生成のパラメタ θ を変化させることによって，クラスタの大きさを調整することができるが，これは，どの程度の演算結果の再利用性が保証されたら演算を共有するか，ということを決定する基準となっている．θ を 1 にした場合は，参照範囲が全く同一である問合せの共通部分式のみを共有することに相当する．また，θ を 0 にした場合は，巨大なクラスタが 1 つ生成される．これは類似度を気にせずに，問合せの共通部分式のみを調べて共有する従来の複数問合せ最適化の手法をそのまま当てはめた場合に相当する．

最初に与えられた問合せの集合が固定されたままで処理が続くということはなく，実際には，後から問合せの追加・削除が発生する．クラスタを常に最適に保つためには，問合せの追加削除のたびに毎回クラスタを再計算しなければならない．しかし，クラスタを更新すると最適な問合せプランも計算し直さなければならないので，毎回クラスタを更新することは非常にコストがかかる．そこで本研究では，問合せの追加・削除が一定回数以上行われた段階で，クラスタの再構成を行うものとする．新たに追加された問合せは，もっとも類似度の高い（θ 以上）問合せが属しているクラスタへ追加される．削除については，問合せそのものの削除のみ行い，クラスタ内の類似度の最小値の再計算は行わない．

また，ストリーム型情報源では，時間が経つと情報源の性質が変わることがある．そのような場合には，新たなログを用いてクラスタを再構成する必要がある．

6.　クラスタ内の複数問合せ最適化

5. で述べた通り，ユーザから与えられた問合せの集合は参照範囲の類似度が高いもの同士でクラスタ化される．クラスタ内の問合せの共通演算の処理結果は，ほぼ共有できることが保証されているため，あとは実際にクラスタ内の問合せから共通演算を探索し，最適な問合せプランをクラスタごとに導出する．この部分の処理には，従来の複数問合せ最適化 [12], [14] を適用する．ただし，コスト見積りはストリーム型情報源に対するログ情報に基づいて行う点が異なる．本節では概要のみ説明する．

例として，4. で示した問合せ 4（図 5），問合せ 5（図 6）の 2 つの問合せのみを含むクラスタに対する処理を説明する．この 2 つの問合せでは，$Stream1 \bowtie Stream2 \bowtie Stream3$ と $Stream1 \bowtie Stream2 \bowtie Stream4$ という処理を行っている．

はじめに，各問合せについて可能な実行プランをすべて探索する．それぞれの問合せの実行プランは図 12 のようになる．ただし，図 12 は実行プランを AND–OR DAG [12] によって表現している．AND–OR DAG 表現は，演算に対応する AND ノードと，AND ノードで生成された結果を表す OR ノードによって構成されるグラフである．図 12 では丸が AND ノード，四角が OR ノードを表している．各 AND ノードは自分自身が属する問合せのマスタ情報源の情報を保持している．また，実際の実行プランでは，結合演算のアルゴリズムの種類なども区別されなければならないが，ここでは簡略化のため論理的なレベルの DAG で説明する．

次に，すべての問合せの OR ノードの集合から，複数回出現しているノードを探し，一つの OR ノードにマージする．例におけるマージ結果を図 13 に示す．ここでは，$Stream1 \bowtie Stream2$ を表す OR ノードがマージされている．マージされた OR ノード以下の AND ノードでは，マスタ情報源の情報がマージされる．

マージした問合せの DAG から，全体の処理コストを最小にするような最適な実行プランを探す．ストリーム型情報源に対する演算の処理コストは，ログから得たデータ到着頻度の情報によって見積る．また，マージされた OR ノードの処理コストは，一回分として見積る．図 13 において，$Steam1 \bowtie (Stream2 \bowtie Stream3)$ のコストよりも，$(Stream1 \bowtie Stream2) \bowtie Stream3$ の方が処理コストが小さいと見積もられたと仮定すると，最適な問合せプランは図 13 中の太線になる．

これを元に導出される実行プランは図 14 のようになる．この場合，例えば 12 時に Clock12 のデータが届くと，$Stream1 \bowtie Stream2$ の結果が計算され，$\Delta(Temp, Example4)$ と $\Delta(Temp, Example5)$ に蓄積される．$Stream1 \bowtie Stream2$ のウイ

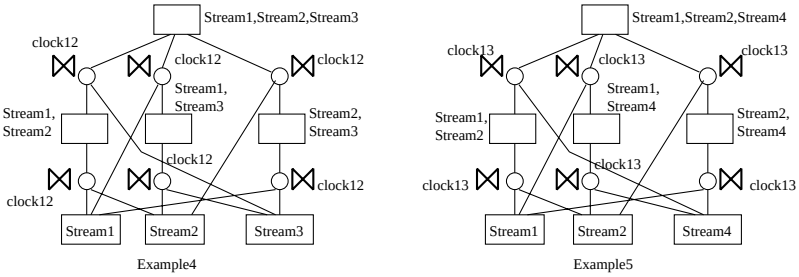

図 12　問合せの実行プラン探索後の DAG

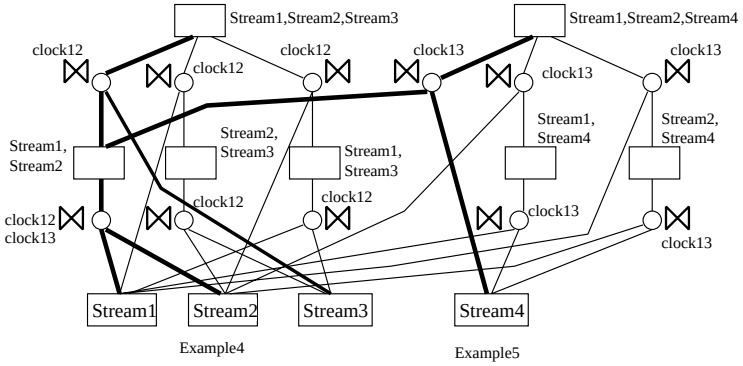

図 13　マージ後の DAG

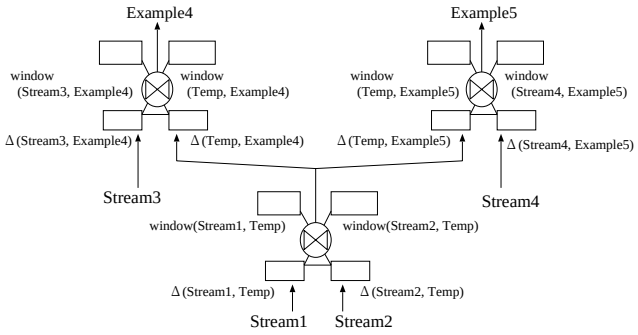

図 14　最適化された問合せ

ンドウ結合は，Clock12 のデータが到着した時点と Clock13 のデータが到着した時点の両方で実行されるが，実行されるたびにデルタリレーションは空になるので，重複したデータを生成することはない．

7. 他の最適化手法

6. で述べたようなクラスタごとに最適化アルゴリズムを適用するというアプローチ以外にも有効と思われるやり方がある．それは，全ての問合せを対象に最適化アルゴリズムを適用し，DAG 中の複数回出現する OR ノードをマージする際に，同じクラスタに属する問合せの OR ノードだけをマージしていくという手法である．ただし，この方法では従来の最適化手法に比べてマージの手順が複雑になることが予想される．また，大量の問合せを最適化アルゴリズムの入力として与えなければならないため，コストを最小にするプランを探索する処理に時間がかかってしまう．

CPU	UltraSPARC-II 296MHz × 2
メモリ	1GB
OS	Solaris 2.6
開発言語	Java (J2SE 1.4.0, J2EE 1.3.1)
使用 RDBMS	PostgreSQL 7.3.1

表 2　実 験 環 境

8. 評 価 実 験

本研究の提案手法の有効性を示すため，プロトタイプシステムを実装し，評価実験を行った．実装環境は表 2 の通りである．

本プロトタイプシステムは，まず与えられた連続的問合せの集合に対して複数問合せ最適化を適用し，問合せ実行プランを生成する．そして，ストリーム型情報源の情報が到着すると，問合せ実行プランに基づいて処理を行う．到着情報や中間結果の管理には RDBMS を用いており，ウインドウリレーションの更新処理等は SQL 問合せを発行することにより実現している．今回の評価実験では，統合結果を配信せず，ログに出力して処理を完了するものとした．

本実験で用いたストリーム型情報源はすべて擬似的に構築したものである．通常のストリーム型情報源として，10 秒おきにデータを配信するものを 2 種類用意した．クロックストリームは 10 種類用意し，それぞれ異なるタイミングで 5 分おきにアラームを配信するものとした．

実験対象となる連続的問合せの集合は，人工的に生成したものを用いた．各問合せの MASTER 節には，10 種類のクロックストリームからランダムに選んだ 1 つを与え，WINDOW SIZE 節には問合せごとに全て異なる値を与えた．今回の評価実験では，SELECT–FROM–WHERE 節の部分はすべて同一とした．

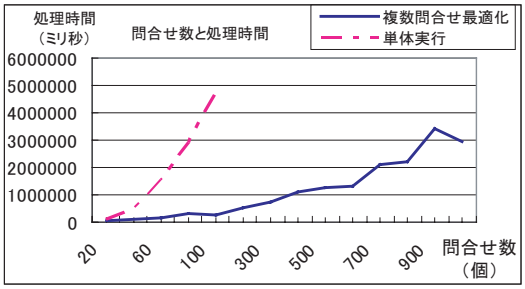

図 15　実験 1 の実験結果

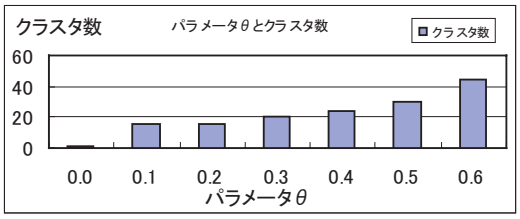

図 16　パラメータとクラスタ数

8.1 実　験　1

まず，すべての問合せを独立に実行した場合と，複数問合せ最適化を適用して実行した場合の比較を行った．本実験ではストリーム型情報源からの情報を 15 分間受信し，期間終了までに発生した処理の合計時間を測定した．ただし，ここでの複数問合せ最適化のクラスタリングのパラメータ θ は 0.5 に固定とした．問合せ数を変化させたとき，それぞれの場合の総処理時間の変化を図 15 に示す．図 15 から，問合せを独立に実行する場合よりも複数問合せ最適化を適用した場合の方が高速であることがわかる．

8.2 実　験　2

次に，複数問合せ最適化のクラスタリングのパラメータ θ を変化させたときの処理時間の変化を調べた．問合せの数を 200 個に固定し，θ を 0.0〜0.6 の間で変化させると，図 16 のように生成されるクラスタの数が変化する．それぞれの場合の処理時間の計測結果を図 17 に示す．図 17 より，$\theta = 0.2$ で問合せをクラスタリングした場合がもっとも速くなっていることがわかる．問合せの類似度を考慮しない方式である，$\theta = 0$ の場合よりも 66 秒高速に処理が行われていた．また，θ を高くしすぎると，逆に速度が低下している．これは，θ が高いとクラスタ数が増え，1 つのクラスタ内に含まれる問合せの数が少なくなるので，処理結果を共有する効果が上がりにくいためと考えられる．

9. む　す　び

本稿では，多数のストリーム型情報源に対する統合要求が大量に与えられる状況を想定し，連続的問合せの複数問合せ最適化方式を提案した．本研究が扱う連続的問合せは，ウインドウ結合のような時間条件を含んだ問合せがマスタ情報源からの情報到着に基づいて実行されるものであり，式の上では共通な演算であっても実行タイミングが異なれば共通性のない全く別の結果を生成する．我々が提案する手法は，実行タイミングの違

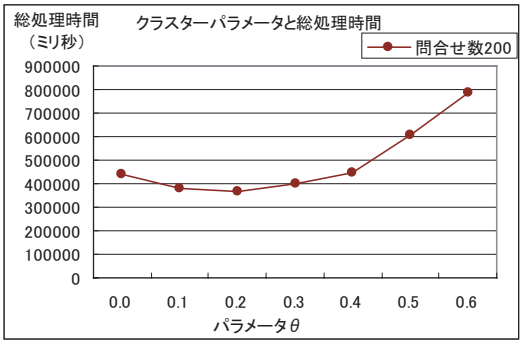

図 17　実験 2 の実験結果

いによる問合せの参照範囲の違いを考慮し，参照範囲の類似度が高いものをクラスタ化した後，クラスタ単位で共通演算の共有を行うというものである．

今後の課題として，多数のユーザから利用される実際の利用環境を想定し，実データを用いた評価実験などが挙げられる．また，今回の評価実験に用いたプロトタイプシステムは，まだ一部の種類の問合せに対する最適化しか行えないので，残りの機能を実装する予定である．

謝辞

本研究の一部は，日本学術振興会科学研究費基盤研究 (B)(12480067) による．

文　　献

[1] R. Avnur, and J. M. Hellerstein. ”Eddies: Continuously Adaptive Query Processing,” SIGMOD 2000, pp. 261–272.

[2] J. Chen, D. J. DeWitt, and J. F. Naughton. “Design and Evaluation of Alternative Selection Placement Strategies in Optimizing Continuous Queries,” ICDE 2002.

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

[4] S. Chandrasekaran, and M. J. Franklin. “Streaming Queries over Streaming Data,” VLDB2002.

[5] E. N. Hanson, C. Carnes, L. Huang, M. Konyala, and L. Noronha. “Scalable Trigger Processing,” ICDE 1999, pp. 266–275.

[6] J. Kang, J. F. Naughton, and S. D. Viglas. “Evaluating Window Joins over Unbounded Streams,” ICDE2003(to appear).

[7] L. Liu, C. Pu, and W. Tang. “Continual Queries for Internet Scale Event-Driven Information Delivery,” TKDE 1999, pp.610–628.

[8] S. Madden, and M. J. Franklin. “Fjording the Stream: An Architecture for Queries over Streaming Sensor Data,” ICDE 2002.

[9] H. Mistry, P. Roy, S. Sudarshan, and K. Ramamritham. “Materialized View Selection and Maintenance Using Multi-Query Optimization,” SIGMOD 2001, pp. 307–318.

[10] S. Madden, M. Shah, J. M. Hellerstein, and V. Raman. “Continuously Adaptive Continuous Queries over Streams,” SIGMOD 2002, pp. 49–60.

[11] N. W. Paton, and O. Diaz. “Active Database Systems,” ACM Computer Systems, 31(1), 1999, pp. 63–103.

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

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

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

[15] D. Terry, D. Goldberg, and D. Nichols. “Continuous Queries over Append-Only Databases,” SIGMOD 1992, pp. 321–330.