問合せ最適化機構を備えたデータストリーム統合システムの開発

渡辺　陽介† 　北川　博之†† 　内山　大悟†††

† 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
††† 筑波大学 第三学群 情報学類　〒 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, †††uchi@kde.is.tsukuba.ac.jp

あらまし　近年，時々刻々と変化する情報をオンラインで逐次提供するデータストリームが増加し，ストリームデータに対する問合せ処理要求およびストリームデータと RDBMS 等の従来の情報源との統合利用要求が高まっている．ストリームデータに対する問合せ処理方式として連続的問合せがあり，現在，連続的問合せの処理システムが注目されている．本稿では，我々が提案するデータストリーム統合システムのアーキテクチャについて述べる．本システムの特徴は，複数の連続的問合せを効率的に実行するために，我々がこれまで提案してきた，実行パターン解析に基づく複数問合せ最適化方式を用いた最適化を行う点である．本システムが扱う連続的問合せは，ストリームデータと RDB の統合を行う，より一般的かつ実用的な処理を含んでおり，本稿ではそれらも対象とした最適化の処理について述べる．
キーワード　放送型サービスと DB, 最適化, 問合せ処理, 情報統合, 異種 DB, データストリーム

Development of a Data Stream Integration System with a Multiple Query Optimizer

Yousuke WATANABE†, Hiroyuki KITAGAWA††, and Daigo UCHIYAMA†††

† 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
††† College of Information Sciences, Third Cluster of Colleges, University of Tsukuba
E-mail: †watanabe@kde.is.tsukuba.ac.jp, ††kitagawa@is.tsukuba.ac.jp, †††uchi@kde.is.tsukuba.ac.jp

Abstract　Today, we can access a large number of data streams, which provide data changing over time. Demands for query processing on multiple streams and integrating streams and traditional RDBMSs have become quite important. Countiunous query provides a basic framework to process streams, and development of systems processing continuous queries is an vital research issue. In this paper, we describe an architecture of a data stream integration system. To process multiple continuous queries efficiently, our system performs multiple query optimization based on execution pattern analysis. We also show an extended query optimization method for integrating streams and RDBMSs.
Key words　Broadcasting services and DB, Optimization, Query Processing, Information Integration, Heterogeneous DB, Data Stream

1. ま え が き

天気予報，ニュース，株価情報，各種センサーデータなど最新の情報をオンラインで提供するデータストリームと呼ばれる新しい情報源が出現し，我々の手元には日夜大量のデータが送り届けられるようになった．それに伴って，ストリームデータに対する問合せ処理を実現するシステムへの関心が高まっており，到着情報のフィルタリングやストリームデータ同士の統合，ストリームデータと RDBMS に格納されたデータとの統合などの機能を実現するデータストリーム統合システムが求められている．また，ストリームデータは最新の情報を提供するというその性質上，情報の鮮度が重要視されることが多く，利用者に通知するまでの遅延時間が短いことが望まれる．今後，データストリームに対する問合せ処理が一般的となり，その利用者が増加していった場合，一つのシステムに対して大量の問合せ要求が発生することが予想される．このため，大量の問合せを効率よく処理可能な手法が求められている．

現在，ストリームに対する問合せ処理方式の一つとして，連

続的問合せ (Continuous Query) [2], [6] が注目されている．連続的問合せとは，新規に到着した情報に対して問合せ処理を適用し，前回の処理実行時からの差分にあたる結果を生成することを繰り返すものである．ストリームにおける連続的問合せでは，処理結果の条件や構造の指定を行うだけでなく，問合せが実行されるタイミングについても，新規情報の到着のたびに実行する，指定時刻に定期的に実行する等，利用者の要求に合わせたきめ細かな指定が可能であることが必要となる．

本稿では，我々の提案するデータストリーム統合システムのアーキテクチャについて述べる．本システムは，データストリームおよび RDBMS に対する連続的問合せの処理機能を提供する．RDBMS に格納されたデータは，利用者が問合せ要求を発行した結果として取得するという受動的なものであるのに対し，ストリームデータは情報源から逐次提供される能動的なものである．そのため，本システムは，情報の到着や時間の経過に応じて動作するイベント駆動処理の機能と，必要に応じて RDBMS からデータを抽出する機能，およびストリームデータと RDB のデータを統一的に扱う機能を有している．また，大量の連続的問合せを効率よく実行可能な処理方式を実現するために，本システムの問合せ最適化器は，これまで我々の研究グループが提案してきた，連続的問合せに対する複数問合せ最適化手法 [18]〜[20] を実装している．複数問合せ最適化とは，問合せ群の中から共通な演算を抽出し，処理結果を共有することで全体の処理効率を上げるという技術である．本システムの問合せ最適化器では，ストリームデータのみを処理対象としていたこれまでの手法を拡張しており，ストリームデータと RDB のデータとの統合処理を含んだ連続的問合せを扱うことが可能となっている．さらに本システムは，アプリケーションプログラムから連続的問合せを登録し，問合せ結果を受け取るための API を提供している．本 API はストリームデータと RDB のデータを統一的に扱うことが可能で，従来の RDBMS における API との親和性が高いものとなっている．本稿ではそれについても述べる．

本稿は次のような構成になっている．まず，2. で連続的問合せの記述法とそれを用いた利用例を示す．そして，3. で本システムの基本的なアーキテクチャについて述べる．4. では，本システムの問合せ最適化器で用いられている最適化手法について述べる．5. は本システムを用いた評価実験の結果を示す．そして，6. で関連研究について述べ，7. でまとめと今後の課題を述べる．

2. 連続的問合せと統合利用例

本節では，本データストリーム統合システム中で用いる連続的問合せの記述法と，それを用いた統合利用例を示す．

2.1 連続的問合せ

本システムでは，データストリームを仮想的なリレーションとしてモデル化し，到着したストリームデータの 1 配信単位をタイムスタンプの付加された 1 タプルとして扱う．そのため，連続的問合せの記述には以下のような SQL に基づいた記述法を用いる．

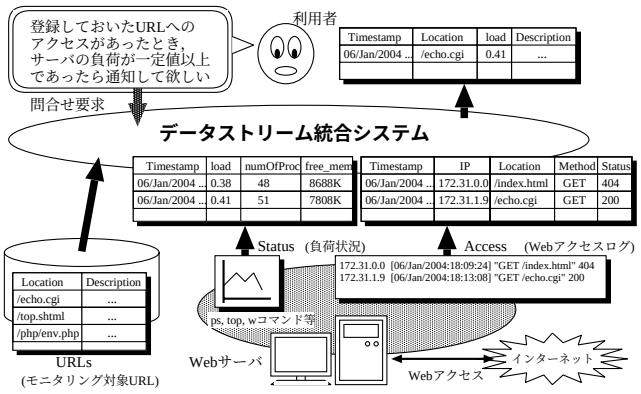

図 1　利　用　例

Fig. 1　Integration example

MASTER *master_source_1* ,…
SELECT *attr_1* ,…
FROM *source_1* { *[windowsize_1]* } , …
WHERE *conditions*

問合せの MASTER 節には，問合せ実行のきっかけとなるストリーム（**マスタ情報源**）を列挙する．マスタ情報源から新規情報が到着したとき，その問合せが実行される．マスタ情報源には任意のストリームが指定可能なほか，**クロックストリーム**を記述することができる．クロックストリームはシステムが提供するストリームで，指定された間隔でアラームを配信する．クロックストリームをマスタ情報源に指定することで，定期的に実行される問合せを記述することができる．問合せ記述の SELECT–FROM–WHERE の各節の意味は SQL とほぼ同様である．ただし，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．

また，本システムでは FROM 節の情報源記述の後に**ウインドウ** [2], [9] を指定することが可能である．ウインドウには処理対象とするデータの範囲を決めるための時間幅を与える．問合せの処理対象となるタプルは，到着してからの経過時間がウインドウ幅以内のものだけとなる．情報源ごとに異なった幅のウインドウを指定することが可能で，ウインドウ幅の指定を省略した際は，無限大のウインドウ幅を持つと解釈される．

連続的問合せが実行されたときに生成される処理結果は，前回の実行時以降に届いたデータを用いた差分の処理結果である．生成された処理結果は逐次利用者へ提供される．

2.2 統合利用例

データストリームと RDB の統合の具体的な説明のため，本システムを用いて Web サーバのモニタリングを行う例を示す（図 1）．この例における利用者は Web サーバの管理者で，自分が管理する Web サーバを監視し，サーバの負荷状況と HTTP アクセスの関係を分析したいと思っており，特に Web サーバに負荷をかける原因である CGI や SSI などによるプログラムの実行を含んだ特定 URL へのアクセスに関心があるとする．まず，Web サーバを監視するためのデータストリームとして，ここではサーバの負荷状況や使用メモリ量などを定期的に通知するストリーム「Status」と，Web サーバへの HTTP アクセスのログの出力ストリーム「Access」の 2 つを考える．Status は，5

```
MASTER Access
SELECT Access.Timestamp, Status.Load, Access.Location, URLs.Description
FROM Access [now], Status [5second], URLs
WHERE Access.Location = URLs.Location AND Status.Load >= 0.40
```

図 2　問 合 せ 例

Fig. 2　Example of continuous query

秒おきにデータを送信するものとする．また，利用者の興味のある CGI や SSL の処理を含んだ特定 URL のリストは，あらかじめテーブル「URLs」として RDBMS へ格納されているものとする．

利用者の要求を「登録しておいた URL へのアクセスが来たとき，サーバの負荷状況が一定値以上だったら通知して欲しい」としたとき，この要求は 2 種類のデータストリームと RDB のデータを統合するという連続的問合せとして記述可能である（図 2）．図 2 のウインドウ指定に現れる now は問合せ実行時刻そのものを表し，5second は 5 秒の時間幅を表している．元の要求から，問合せのマスタ情報源には Access が指定される．

3.　データストリーム統合システム

本研究で提案するデータストリーム統合システムのアーキテクチャおよび本システムが提供する API について説明する．

3.1　アーキテクチャ

本システムは Java によって実装されており，問合せ解析器，問合せ最適化器，ログマネジャ，メディエータ，ビューマネジャ，ラッパー，クロックストリームおよびローカル DBMS から構成される（図 3）．

まず，利用者から与えられた問合せは，問合せ解析器によって解析され，内部形式に変換された後，問合せ最適化器に渡される．問合せ最適化器は与えられた全問合せを管理しており，4. で述べる複数問合せ最適化手法によって問合せ集合から最適な実行プランの導出を行う．メディエータは実行プランに従って，リモートの RDBMS から必要なデータを取得するための SQL 問合せの発行や，ストリームから到着したデータの処理などを行う．メディエータにおいて生成された処理結果は直ちに利用者へ配信される．ログマネジャは，メディエータが処理したデータおよび各問合せの実行状況を蓄積し，問合せ最適化器の最適化処理に必要なログ情報を提供する．ビューマネジャはリモート RDBMS から取得した SQL の実行結果のキャッシングを行うためのもので，一度取得した結果をローカル DBMS に保持する．実行結果のキャッシングは，ストリームからデータが到着するたびにリモートの RDBMS へ問合せを発行することを避けるために必要となる．各情報源にはそれぞれ対応するラッパーが存在する．ラッパーの主な役割は情報源固有のデータ形式を本システムの内部形式に変換することである．ただし，ストリームラッパーには新規到着情報を検出し，メディエータに通知する機能があり，RDB ラッパーにはメディエータからの SQL 問合せ要求を RDBMS へ渡し，結果を取得する機能がある．クロックストリームはシステムが提供するストリームで，タイマーを用いて実装されており，定期的にアラームの通知を行う．アラームの通知間隔や通知時刻は任意に設定可能である．

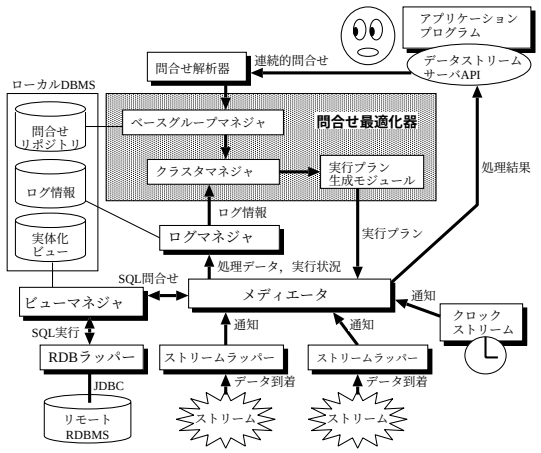

図 3　アーキテクチャ

Fig. 3　System architecture

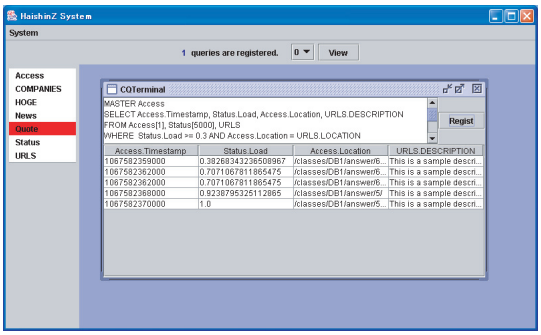

図 4　実 行 画 面

Fig. 4　Snapshot of our system

図 4 は，本システムの実行画面である．GUI から登録中の問合せ一覧や，データの到着状況が確認できるようになっている．

3.2　データストリームと RDB の統合

利用者から与えられた連続的問合せの実行プランに含まれる演算のうち，ストリームを扱わなければならない演算は，実際にデータが到着するまで処理を行うことができない．しかし，RDB に対する問合せ処理のみを行う演算は，あらかじめ計算して途中結果を取得しておくことが可能である．メディエータは実行プランを渡された時点で，事前に実行可能な演算を SQL に変換し，ビューマネジャにリモートの RDBMS からデータの取得を要求する．ビューマネジャは，取得した結果を実体化ビューとしてローカル DBMS に保存し，もしメディエータから再度同じ SQL によるデータ取得要求が来たときには，実体化ビューのデータを再利用する．ただし，リモート RDBMS 中のデータが更新・追加された場合は，更新情報を実体化ビューへ反映させる必要がある．変更は RDB ラッパーがトリガの機能を用いて検出する．

選択・結合・射影などの各演算子が，ストリームのタプルや RDB のタプルということを区別せずに，データを統一的に扱えるようにするため，本システム内部ではどのような情報源のタプルも必ずキューを介して取得される．演算子は，入力キューからタプル集合を操作するためのハンドラを取得し，ハンドラを使ってキュー内のタプルを 1 つずつ読み出す．演算子の処理

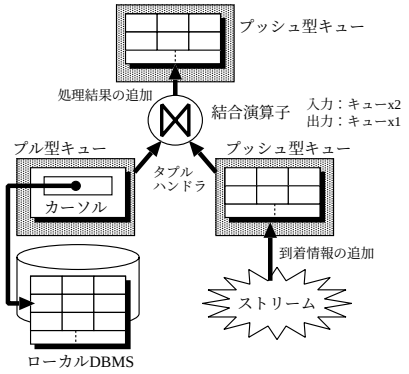

図 5　RDB とストリームの統合

Fig. 5　Integration of RDB and stream data

結果は，出力キューへ順次追加される．ストリームデータを扱うキューには，実際にキュー内部にタプル集合が保持され，新規情報が次々と追加される．それに対し，RDB データはプル型キューと呼ばれる特殊なキューによって提供される．プル型キューは，ローカル DBMS 中の実体化ビューと関連付けられており，内部にはタプル集合を持たず，実際は実体化ビューへのカーソルを保持している．演算子がハンドラを通じてタプルを取り出そうとすると，プル型キューは実体化ビュー上の対応するカーソル位置からタプルを読み出す．なお，RDB データとストリームデータを統合した結果はストリームデータとして扱われ，通常のキューへ格納される．

3.3　プログラミングインターフェース

本システムは，アプリケーションプログラムから問合せを登録し，処理結果を受信および利用するための API を提供している．本研究では，データストリーム統合システムを利用するための API に必要な機能を以下のように規定し，これらの点に基づいて本システムの API を設計した．

イベント通知機構　連続的問合せの処理結果は逐次生成される．そのため，アプリケーションへの処理結果の通知を能動的に行わなければならない．アプリケーション側が処理結果の通知に連動して動作できるよう，イベント駆動の機構を備えなければならない．

統一的なデータ操作　ストリームデータとその他の情報源のデータを統合利用する以上，それらに対する統一的な操作が必要である．本システムはデータストリームをリレーションとしてモデル化しており，また，ストリームデータと RDBMS のデータとの統合利用も想定しているため，問合せの処理結果をリレーションのタプルの集合として受信・操作できる必要がある．

透過的な接続管理　アプリケーションとデータストリーム統合システム間の通信は処理結果を通知するたびに発生する．多数のアプリケーションが問合せを登録し，処理結果の通知を待つという状況においては，すべてのアプリケーションとの間の接続を常時保持し続けることはリソースの制約上不可能である．しかし，プログラマにきめ細かい接続・切断の処理を記述させることは負担が大きいため，API 内部で透過的に接続を管理できることが望ましい．

名前	役割	主なメソッド
CQRowSet	問合せ結果を表す	setCommand(), start(), stop()
CQRowSetListener	イベントのリスナ	dataDistributed()
CQRowSetEvent	イベントを表すクラス	getSource()
CQRowSetMetaData	問合せ結果のメタデータ	getColumnName()

表 1　システム利用のためのクラス

Table 1　Classes for using data stream integration system

```
public class Example {
    public static void main(String[] args) ··· {
        CQRowSetListener ls = new myListener();
        CQRowSet rs = new DefaultCQRowSet();
        rs.setUrl("rmi://localhost/StreamServer");        // サーバの Location 指定
        rs.setCommand("MASTER Access SELECT ··· "); // 問合せのセット
        rs.addCQRowSetListener(ls);              // リスナのセット
        rs.start();              // 連続的問合せの実行開始
        ··· }
    public class myListener implements CQRowSetListener {
        public void dataDistributed(CQRowSetEvent e) ··· {
            CQRowSet rs = (CQRowSet)e.getSource();
            while(rs.next()){              // JDBC と同様のカーソル API による操作
                System.out.println(rs.getString("Location"));
            }
} } }
```

図 6　プログラムの記述例

Fig. 6　Sample program

表 1 に，本 API における主要なクラスを示す．CQRowSet クラスは問合せ結果を表す仮想的なリレーションで，問合せ結果に対する JDBC [10] ライクなアクセス手段を提供する．JDBC は Java で DBMS に接続するための標準的な API である．プログラマは，CQRowSet にパラメータとして問合せおよびシステムの URL をセットした後，start() メソッドを呼び出すことで問合せを登録することができる．CQRowSet がシステムから問合せ処理結果を受信すると，CQRowSet は CQRowSetListener インターフェースを実装したアプリケーションへイベントを通知する．イベント通知を受け取ったアプリケーションは，CQRowSet の提供するカーソル API を使って結果の取得を行うことができる．システムとの通信は CQRowSet の内部で行われており，プログラマからは隠蔽される．現在の実装では，通信機能には Java RMI を用いている．

図 6 は，本 API を用いた単純なプログラムの記述例である．問合せの処理結果が生成されると，CQRowSetListener インタフェースを実装した myListener クラスの dataDistributed メソッドが呼び出される．

4.　複数問合せ最適化手法

4.1　概　　要

複数問合せ最適化は，問合せ集合内に含まれる共通演算の処理結果を共有することで重複する処理を減らし，全体の処理効率を向上させる手法 [12], [15], [17] である．ただし，連続的問合せに複数問合せ最適化を適用する際に問題となるのは，マスタ情報源の異なる問合せ同士では，共通演算を含んでいても，

実行タイミングが離れすぎているために処理結果が共有できない場合があるということである．本研究では，共通演算の処理結果が共有できるケースとして，以下の 2 つを想定している．

case 1　MASTER 節に同一のマスタ情報源を与えられた問合せ同士が同じタイミングで実行されることは明らかである．もし，それらに共通演算が含まれているならば，生成される処理結果は共有可能である．

case 2　異なったマスタ情報源を与えられた問合せ同士であっても，非常に近いタイミングで実行されるものであれば，共通演算から生成される処理結果はかなり近いものとなる．この場合，処理結果の一部が共有可能である．

我々の最適化手法 [18]〜[20] はこれらの点を考慮し，前処理として問合せ集合に対して 2 段階のグループ化を行った後，生成されたグループごとに共通演算の共有化処理を適用する（図 7）．以下では，各問合せが問合せ解析器によりすでに内部形式に変換され，すべての可能な演算の実行順序のパスが導出されているものとして説明を行う．

4.2　ベースグループ

ベースグループは case 1 に相当する問合せの集合である．問合せ最適化器内のベースグループマネジャが，与えられた問合せの集合から以下の定義を満たす集合を生成する．

$$\forall Q_i, Q_j \in BG \left(\begin{array}{l} MS_{Q_i} = MS_{Q_j} \quad \wedge \quad REF_{Q_i} \cap REF_{Q_j} \neq \phi \\ \wedge \quad (\forall I \in (REF_{Q_i} \cap REF_{Q_j})(\frac{WIN_{Q_i}(I)}{WIN_{Q_j}(I)} \geq \theta_1)) \end{array} \right)$$

この式は，同一のベースグループに属する問合せは，同一のマスタ情報源をもち，さらに共通の情報源を閾値 θ_1 以上の割合で参照していなければならないことを表している．ただし，MS_Q, REF_Q, および $WIN_Q(I)$ は，それぞれ問合せ Q における，マスタ情報源集合，FROM 節で参照している情報源集合，および情報源 I におけるウインドウ幅を表す．ベースグループは問合せ記述を解析するだけで生成することができるため，情報の到着パターンの変化には無関係であり，後述する到着ログを用いた問合せのクラスタリングにおける最小単位となる．ベースグループを生成した時点でグループ内に共通演算が存在するならば，その演算については先に共有化の処理を行っておく．

4.3　到着ログを用いた問合せのクラスタリング

ベースグループの条件は厳しいので，一つのベースグループに属する問合せの数は平均的に少なく，このままでは共通演算の共有による恩恵が小さい．そこで，さらに case 2 の関係にあるより大きな問合せのグループを生成する．すでに問合せ群がベースグループの集合へ分類されているため，ここではベースグループ間で比較を行い，マスタ情報源の到着時刻が近く，かつ，処理結果が類似しているかどうかを調べる．

しかし，クロックストリームのような等間隔で到着するストリームを除き，ほとんどのストリームデータの到着タイミングを事前に知ることは困難である．そこで本研究では，ログマネジャが蓄積した到着データと問合せの実行状況に関するログを用いて，ベースグループ同士の実行タイミングおよび参照するデータの時間範囲の関係（**類似度**）を調べる．

以下でベースグループ同士の類似度について説明する．まず，ある期間 T におけるデータ到着ログを取得し，情報源 $I_i(1 \leqq i \leqq n)$ のタプルの到着時刻の集合 S_{I_i} を用意する．そして，ベースグループ $B_j(1 \leqq j \leqq m)$ 内の問合せ全てに対して，到着時刻情報を用いたシミュレーション実行を行い，そのとき参照された到着時刻の集合 USE_{I_iBj} を求める．このとき，実行タイミングが近いベースグループ同士では，参照するデータの範囲が重なることが多いため，$USE_{I_iQ_j}$ 中の要素が類似したものとなる．そこで，ベースグループ B_1,B_2 の類似度 $sim(B_1, B_2)$ を，以下の式により定義する．

$$sim(B_1, B_2) = \min_{I_i \in (REF_{B_1} \cap REF_{B_2})} \frac{|USE_{I_iB_1} \cap USE_{I_iB_2}|}{|USE_{I_iB_1} \cup USE_{I_iB_2}|}$$

この式は，B_1,B_2 中の問合せが共に使用する情報源において，参照する情報に最低でもどの程度共通部分があるかを表している．ただし，REF_B はベースグループ B 中の問合せの 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 $(REF_{B_1} \cap REF_{B_2} = \emptyset)$ の類似度は 0 とする．

全てのベースグループ同士で類似度を計算すると，m 個のベースグループに対して $m \times m$ の類似度行列が得られる．この類似度行列を用いて，通常の階層的クラスタリング [16] を行い，類似度の高いベースグループ同士のクラスタを生成する．ここでは，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_2 以上になるようにクラスタを生成する．

4.4　演算の共有

上記の処理により，クラスタ内の問合せに出現する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．あとは従来からの複数問合せ最適化手法 [15] を適用し，クラスタごとに最適な問合せ実行プランを導出する．ただし，ストリームにおいては演算のコスト見積りが困難であるため，最適な問合せ実行プランの候補となるパスが複数存在する場合の最適プランの選択法が従来とは異なる．以下には，手順の概要のみを示す．

（1）　クラスタ中に含まれる問合せの各演算から，重複して出現するものを検出する．探索は問合せ実行順序のパスをボトムアップに走査することで行う．重複する演算とは，ここでは各演算に指定された結合条件などのパラメターが一致するものと定義する．選択演算であれば選択条件，結合演算であれば結合条件，射影演算であれば属性のリストである．ただし，各演算のマスタ情報源の違いやウインドウの幅の違いは重複の判定基準には含めない．

（2）　重複する演算を発見したら，それらをマージする．マージされた演算のマスタ情報源は，元の演算のマスタ情報源の和集合とする．また，マージ後の演算の各情報源のウインドウ幅は，マージ前の演算における各情報源のウインドウ幅の最大値とする．

（3）　すべての共通演算をマージした後，候補となる実行順序のパスから最適なものを問合せ実行プランとして導出する．ここでは共有される演算の全体数が最大となるパスを最適なプランとする．もし，共有数最大となる候補が複数存在した場合には，狭いウインドウ幅を与えられた演算を優先的に処理するようなパスを選出する．これは中間結果の量を減らすための戦略で，ウインドウ幅が狭いほど，入力キューに蓄積されるタプ

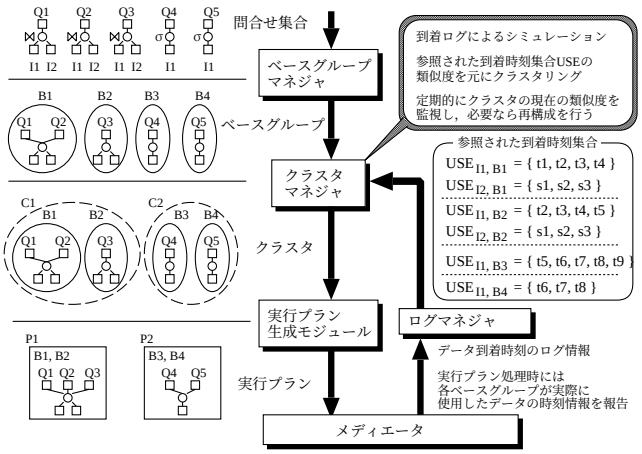

図 7　複数問合せ最適化処理の流れ

Fig. 7　Process of multiple query optimization

ル数が少ない傾向があるという理由による．

4.5　到着パターン変化への適応

一部のストリームでは，ある程度の時間が経つとデータの到着パターンが変化してしまうことがある．このとき，パターンの変化は類似度の変化として検出される．古くなったログ情報を用いて生成したクラスタでは，最適なパフォーマンスを出すことができないため，クラスタを再構成する必要がある．

メディエータは，各ベースグループが実際にどのデータを結果生成に用いたか，ログマネジャに逐次報告している．そのため，ログマネジャには常に最新の期間 T における $USE_{I_iB_j}$ が蓄積されており，クラスタマネジャは，この最新情報を元に，現在のクラスタ内のベースグループの類似度を定期的に再計算し，類似度が閾値よりも下回っていたらクラスタの再構成と実行プランの再導出を行う．

4.6　実体化ビューの更新

3.2 で述べたとおり，RDB に対する問合せ処理のみを行う部分は，あらかじめローカル DBMS 上に実体化ビューとして生成される．そのため，リモート DBMS 中のオリジナルのデータに変更が発生した場合には，更新情報を実体化ビューへ反映させる必要がある．本システムでは，オリジナルの RDB のテーブルへのデータの追加を新規データの到着とみなし，また，ビューを定義する問合せをオリジナルのテーブルをマスタ情報源とする連続的問合せとみなすことで，実体化ビューの更新をストリーム処理の一種として扱うことができる．よって，同じテーブルから生成された複数の実体化ビューが存在する場合には，それらのビュー定義問合せを上記の連続的問合せに対する複数問合せ最適化手法を用いて効率化することが可能である．この部分の処理は，従来の複数問合せ最適化を用いた実体化ビューの更新手法 [12] と論理的には等価であるが，本システムではストリームを処理するための枠組みをそのまま用いて更新処理を実現している．ただし，ストリームとして扱えるのは，オリジナルのテーブルに対する変更がデータの追加のみの場合だけで，削除や更新を含む複雑な場合は通常の実体化ビューの更新処理または実体化ビューの再構築を必要とする．

名前	到着間隔
Access	不定期
Status	5 秒
Clock30a	30 秒
Clock30b	30 秒 (Clock30a+15 秒)
Clock60a	60 秒
Clock60b	60 秒 (Clock60a+15 秒)
Clock60c	60 秒 (Clock60a+30 秒)
Clock60d	60 秒 (Clock60a+45 秒)

表 2　マスタ情報源

Table 2　Master information sources

CPU	UltraSparcII 296MHz x2
OS	Solaris9
メモリ	1GB
Java	J2SE1.4.1
DBMS	HSQLDB1.7.2rc1

表 3　実 験 環 境

Table 3　Experiment Environment

5.　評 価 実 験

本システムを用いて簡単な評価実験を行った．以下では，その結果について述べる．

まず，本研究で用いた実験データについて述べる．2. で述べた利用例に近い状況で実験を行うため，我々の研究室の Web サーバの 2004 年 2 月 18 日 15 時から 23 時までの 8 時間のアクセスログ (Access) と負荷データ (Status) をそれぞれファイルへ記録した．Access は時間内に発生した計 1600 件のアクセスを含んでおり，Status は 5 秒間隔で記録した負荷情報を含んでいる．実際の実験時には，ファイルに記録された到着時刻に合せて配信間隔を再現し，データの送信を行った．また，DBMS に格納された特定 URL リスト (URLs) には，Access 中に 1 回以上出現した URL 全 510 件を用いた．

本実験では，問合せとして人工的に生成したもの 500 件を用いた．500 件の問合せのすべてが，必ず Access, Status, および URLs に対する結合演算および射影演算を含むものとした．結合条件についてはどの問合せも同じものが与えられるものとし，射影演算によって残される属性のリストは問合せごとにランダムに選ばれるものとした．また，問合せ 500 件のうち 388 件が負荷データに対する選択条件 ($Status.Load > constant$) を含んでいる．選択条件中の定数は 0 以上 1 未満の範囲の少数からランダムで生成されるように設定した．各問合せのマスタ情報源は，表 2 中の情報源から 1 つをランダムに選択し，ウインドウの幅は Access および Status に対しては 1 秒，5 秒，30 秒，60 秒のうちのひとつ，URLs には無限大のウインドウを与えた．

本システムの実験環境は表 3 の通りである．ローカル DBMS として用いた HSQLDB [8] は，Java で実装されたオープンソースの DBMS で，テーブル中の全データを主記憶上に保持する機能を持つ．今回の評価実験ではリモート DBMS およびビューマネージャは用いず，ローカル DBMS に必要な全データがすでに蓄積されているものとして，全てのテーブルを HSQLDB の主記憶テーブルとして作成した．

実験では，8 時間分の実験データに対して 500 件の連続的問合せを実行し，データ到着からその到着に関連した問合せ処理を完了するまでの処理時間をすべて計測した．比較対象として，問合せ集合に対して複数問合せ最適化を行った場合と，それぞれを独立で実行した場合で計 2 回測定を行った．結果を図 8 に示す．図の横軸は，データ到着の発生した時刻を表し，縦軸は

そのイベントに連動する処理を完了するまでにかかった時間を表す．

データがそれほど到着していない状態では，両者の間で処理時間に劇的な差が出ていないが，複数問合せ最適化を行った場合のほうが平均的に 2 秒程度高速であった．それに対し，Access からデータの到着が頻発し，一時的に処理が大量発生する状態では，問合せを独立に実行すると処理が追いつかなくなってしまい，その後の処理にまで多大な遅延を及ぼしてしまうことがわかる．複数問合せ最適化を行った場合でも多少の遅延が発生したが，こちらはすぐに回復している．処理時間の平均は，複数問合せ最適化を行った場合が 1.9 秒で，問合せを独立に実行した場合が 67.9 秒であった．

複数問合せ最適化におけるグループ化によって，500 個の問合せからベースグループが 69 個生成され，それらがクラスタリングにより常時 15 から 30 個程度のクラスタへまとめられ，問合せ処理が行われた．

本実験から，実際の到着データに対して，本システムが効率的な問合せ処理を行えることが確認された．

6. 関 連 研 究

連続的問合せを用いたデータストリーム処理システムに関するこれまでの主な研究について述べる．

OpenCQ [11] 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，情報の到着によって起動する到着型問合せとタイマーによって起動するタイマー型問合せの両方の問合せがサポートされている．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 [5], [6] 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共有化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．

TelegraphCQ 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 [13] と PSoup [7] は TelegraphCQ のプロトタイプシステムである．CACQ では eddy [3] 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，その演算評価方式は，タプルが到着した時点で次々と適用可能な演算を評価し，処理結果を生成していくものである．しかし，本研究が想定するストリーム型情報源の処理環境のように，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では，生成しておいた処理結果が指定時刻には時間条件を満たさなくなってしまうことがあるため，処理結果が無駄になる可能性がある．このような状況では，到着したタプルをすぐに処理せず，指定されたタイミングになった時点でまとめて評価する方が余分な処理が発生せず効率が良い．PSoup も CACQ と同様に eddy を用いたシステムで，処理結果の共有や演算の評価方式については CACQ と同等である．PSoup の特徴は連続的問合せ自身のインクリメンタルな追加を問合せストリームとみなすことにより，問合せとデータを共にストリームとして対称的に扱うことができる点である．

Aurora [1] はストリームに対するモニタリングアプリケーションのためのシステムである．Aurora の最大の特徴はストリームデータ処理に QoS の概念を取り入れたことで，システムは利用者の与えた QoS 指定をもとに，クオリティの低下を最小に抑えるような演算子のスケジューリングを行う．また，ストリームの到着レートが一時的に高まってシステムの処理能力を超えた場合に，入力データを減らしてシステム負荷を軽減する，Load Shedding という概念を提案している．Aurora では，利用者は要求を連続的問合せとして宣言的に記述する代わりに，GUI を用いてワークフローを記述する．そのためのストリームに対する独自の代数演算を提案している．ただし，Aurora ではストリームと RDB との統合は考慮されていない．

STREAM [14] では，CQL [2] という SQL を拡張した連続的問合せ言語によって要求を記述する．CQL は 3 種類のウインドウをサポートしており，時間によるウインドウ，タプル数によるウインドウ，属性の値ごとにグループ化されたウインドウを宣言することができる．本研究では時間によるウインドウしか考慮していない．様々な種類のウインドウを持つ問合せ集合を最適化できるように，我々の最適化手法を拡張することについては今後の課題である．また，STREAM は，限られたリソース内で処理を行うために，ウインドウ幅の縮小やサンプリングレートの制限などを動的に行い，近似的な問合せ結果を返す機能を持つ．STREAM でも演算の共有が考慮されているが，本研究のような問合せの実行タイミングと参照範囲に着目した最適化までは行っていない．

本研究のデータストリーム処理システム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．

7. む す び

本稿では，我々の提案するデータストリーム統合システムのアーキテクチャの概要について述べた．本システムの特徴は，連続的問合せの実行タイミングと参照範囲に着目した複数問合

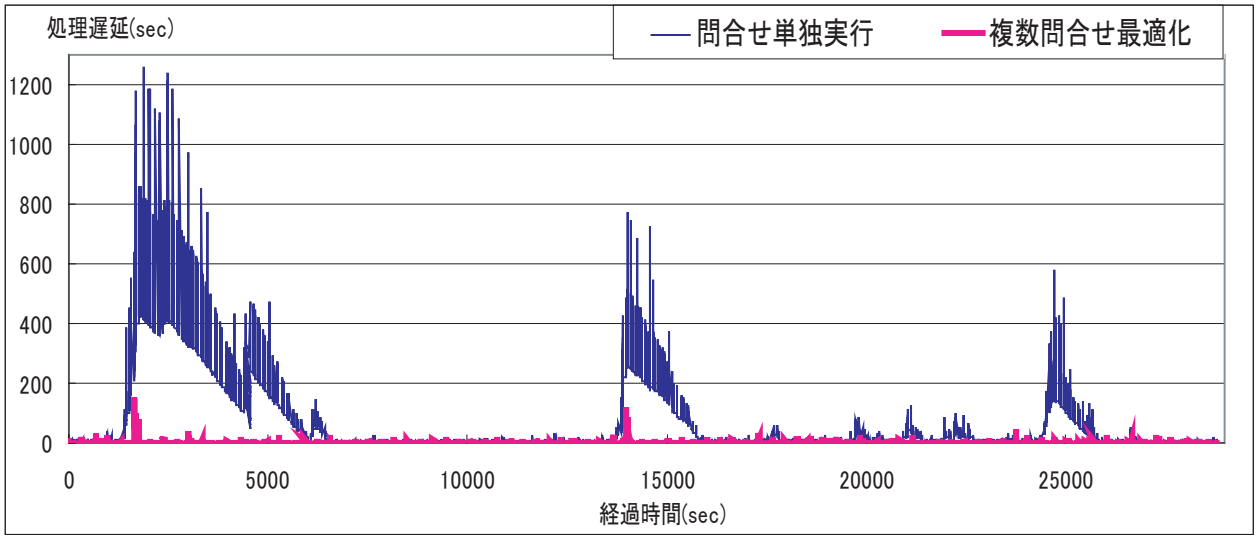

図 8　実 験 結 果

Fig. 8　Result of experiment

せ最適化を行う最適化器を備えている点である．本システムが提供する，連続的問合せの処理結果を利用したアプリケーションプログラムを作成するための API についても述べた．また，本システムの性能を測るため，実データを用いた評価実験を行い，その有効性を確認した．

今後の課題として，実装システムの機能の拡張が挙げられる．現在のシステムは扱えるデータ型などに制限があるため，実際の利用のためにはそれらの充実が求められる．また，本システムを用いたより詳細な評価実験がある．

謝　　辞

本研究の一部は，日本学術振興会特別研究員奨励費 (330)，科学研究費補助金特定領域研究 (2)(#15017207)，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] R. Avnur, and J. M. Hellerstein. “Eddies: Continuously Adaptive Query Processing,” Proc. ACM SIGMOD Conference, pp. 261–272, 2000.

[4] 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.

[5] J. Chen, D. J. DeWitt, and J. F. Naughton. “Design and Evaluation of Alternative Selection Placement Strategies in Optimizing Continuous Queries,” Proc. International Conference on Data Engineering, pp.345–356, 2002.

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

[7] S. Chandrasekaran, and M. J. Franklin. “PSoup: a system for streaming queries over streaming data,” VLDB Journal Vol.12, No.2, pp. 140–156, 2003.

[8] HSQLDB, http://hsqldb.sourceforge.net/

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

[10] JDBC Technology, http://java.sun.com/products/jdbc .

[11] 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.

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

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

[14] 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.

[15] 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.

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

[17] T. K. Sellis. “Multiple-Query Optimization,” ACM Trans. Database Systems, vol.13, no.1, pp. 23–52, 1988.

[18] Y. Watanabe, and H. Kitagawa. “A Multiple Continuous Query Optimization Method Based on Query Execution Pattern Analysis,” DASFAA 2004 (to appear).

[19] 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”，DEWS 2003.

[20] 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.