ExT：予測不可能なデータ到着率における共有ウィンドウ結合のスケジューリング手法

多田　直剛†　　有次　正義††

† 群馬大学大学院工学研究科情報工学専攻　〒 376-8515 群馬県桐生市天神町 1-5-1
†† 群馬大学工学部情報工学科　〒 376-8515 群馬県桐生市天神町 1-5-1
E-mail: †naotake@dbms.cs.gunma-u.ac.jp, ††aritsugi@cs.gunma-u.ac.jp

あらまし　現在，連続的問合せを用いたデータストリーム処理の研究がますます盛んになっている．データストリームを効率良く処理するためには，スループットが高く，状況の変化に適応するスケジューリング手法が必要である．我々は，短期的なスループットだけでなく，長期的な問合せ処理数においても良い性能を示すスケジューリング手法，ExT を提案する．従来のスケジューリング手法とは異なり，スループットだけでなく，問合せの成功率を考慮する．また，その時々のデータの到着状況や，問合せの処理状況に基づき，状況の変化に適応した動的なスケジューリングを行う．このとき，実時間での動的スケジューリングが可能であるという特徴もある．ExT の持つ利点を述べ，実験により提案手法の有効性を示す．

キーワード　ストリーム，連続的問合せ，ウィンドウ結合，スケジューリング

ExT : A Scheduling Approach for Shared Window Joins at Unpredictable Data Arrival Rates

Naotake TADA† and Masayoshi ARITSUGI††

† Department of Computer Science, Graduate School of Engineering, Gunma University
1-5-1 Tenjin-cho, Kiryu, Gunma 376-8515, Japan
†† Department of Computer Science, Faculty of Engineering, Gunma University
1-5-1 Tenjin-cho, Kiryu, Gunma 376-8515, Japan
E-mail: †naotake@dbms.cs.gunma-u.ac.jp, ††aritsugi@cs.gunma-u.ac.jp

Abstract　There has been a growing interest in efficient processing on continuous queries over data streams. In order to efficiently process data streams, it is necessary to have scheduling algorithms that can provide high throughput and low number of unexecutable queries, and can scale in terms of data arrival rate and buffer sizes. We propose an algorithm, called ExT, that attempts to provide good performance balanced between the throughput of queries and the process rate of each query. Unlike previous work on scheduling for continuous queries, ExT makes scheduling decisions based on the current processing state. That is, ExT can adapt to changes in the data arrival rate and intensity of the workload. Moreover, query selection time is short. We describe the performance advantages of ExT, and report some experimental results showing that ExT can work well over a range of workload settings.

Key words　Stream, Continuous query, Window join, Scheduling

1. はじめに

近年，データストリームの処理に多くの関心が集まっており，Aurora [1]，STREAM [2]，TelegraphCQ [3] といった，データストリーム処理システムが提案されている．データストリームは従来のリレーショナルデータベースとは異なる性質の情報源で，時間の経過とともに新しい情報を提供し続ける．例えば，各地の状況や変化を刻々と伝えるセンサーはデータストリームである．リレーショナルデータベースとは異なり，データストリームでは情報源から自発的にデータが送信され，時間の経過とともにデータの到着パターンが変化する可能性がある．したがって，処理システムでは予測不可能に到着する大量のデータを扱わねばならず，データストリームを効率良く処理することが非常に重要となる．

データストリームを処理する枠組みとして連続的問合せが注目を浴びている．中でも，ウィンドウ結合を用いた問合せに関する研究が数多くなされている [4]〜[6]．ウィンドウ結合とは，ウィンドウで指定した時間範囲に含まれるデータを対象とする結合方法で，データストリーム同士を結合するときに広く用いられている方法である．特に，問合せ間で処理結果を共有してウィンドウ結合を行うことを共有ウィンドウ結合と言い，共有ウィンドウ結合に焦点を当てたスケジューリング手法も提案されている [6]．本研究でも，共有ウィンドウ結合を行う．

データストリームに対する処理では，従来のリレーショナルデータベースに対する処理には無い特徴がある．本研究で対象とするデータストリームの処理では，処理システムに到着したデータはメモリ上のバッファに一時的に保存され，新しいデータが到着する度に古いデータから消えていく．したがって，データが予測不可能に到着する環境では一度に大量のデータが到着する可能性があり，問合せが実行される前にデータが消えてしまう可能性がある．問合せの失敗は処理結果に影響するため，問合せの成功率を考慮したスケジューリングが必要となる．データストリームにおける問合せのスケジューリングでは，スループットや問合せの成功率が高く，状況の変化に適応できることが求められる．

本稿では，短期的なスループットだけでなく，長期的な問合せ処理数においても良い性能を示すスケジューリング手法，ExT を提案する．従来のスケジューリング手法とは異なり，スループットだけでなく，問合せの成功率を考慮する．また，その時々のデータの到着状況や，問合せの処理状況に基づき，状況の変化に適応した動的なスケジューリングを行う．本稿では，3 種類のスケジューリング手法を提案する．素朴な方法では実時間でスケジューリングが難しいが，問合せの選択時間を短くすることで，実時間でのスケジューリングを可能としている．

本稿の構成は次の通りである．まず 2 節で，本研究の関連研究について述べる．3 節では，準備として研究の背景や共有ウィンドウ結合を説明し，4 節で，従来のスケジューリング手法について述べる．続く 5 節で，提案手法である ExT を説明する．6 節では，プロトタイプシステムを用いた評価実験について述べる．最後に 7 節で，まとめと今後の課題とする．

2. 関 連 研 究

MQT [6] では，共有ウィンドウ結合のスループットを最大にするスケジューリング手法を提案している．本研究でも，共有ウィンドウ結合を対象としたスケジューリングを行う．提案手法は，スループットと問合せの成功率の両方を考慮することで，短期的な問合せ処理数だけでなく，長期的な問合せ処理数も増加する．

Viglas ら [4] は，データの到着率に基づいた最適化を行い，問合せ処理のスループットを最大にする手法を提案している．我々は，スループットを最大にするだけではなく，問合せの成功率も考慮したスケジューリングを行う．また，データの到着率だけではなく，問合せの処理状況にも適応して動的なスケジューリングを行う．

LocationID	Value	Timestamp

図 1　情報源から到着するデータアイテム
Fig. 1　A data item from data sources

RxW [7] は，データ配信のスケジューリング手法を提案している．人気のあるデータを優先して配信すると，人気のないデータはいつまでも配信されない可能性がある．RxW では，人気のあるデータと人気のないデータの配信の両方を考慮するため，データごとの要求数と要求されてからの待ち時間を掛合せた値をスケジューリングに用いる．この値が大きい順にデータを配信することで，スループットを高くしつつ，*starvation* が起きないようにデータを配信する．本研究ではスループットと問合せ成功率の両方を考慮するため，RxW の成果を用いて，スループットを考慮した値と問合せ成功率を考慮した値を掛合せた値をスケジューリングに用いる．

問合せ成功率の低下に対処する 1 つの手法として，Load Shedding が提案されている [2], [8]．この手法は，問合せの処理結果の正確さを保ちつつ，処理にかかる負荷を軽減するというものである．例えば STREAM [2] では，入力データをランダムにサンプリングすることで，システムが扱うデータ量を減らしている．我々の目的は，問合せの実行順序によって問合せ成功率の低下に対処することであり，より多くのデータに対して問合せを実行することを目標としている．したがって，提案手法と Load Shedding は併用可能であり，互いに競合する手法ではないと言える．

3. 準　　備

3.1 背景・環境

本研究では，データストリームに対して共有ウィンドウ結合を行うシステムを考える．データストリームの各データアイテムは，システムに入った時間と関連付けられている．データアイテムの到着の仕方には 2 通りあり，バースト的に到着する場合と，規則的に到着する場合がある．バースト的な到着とは，データアイテムの到着間隔が不規則で，一度に複数のデータアイテムが短期間にまとまって到着することをいう．バースト的なデータストリームの例には，ネットワークモニタリングストリーム，電話の発信履歴，イベント駆動のセンサーなどがある．対照的に，規則的なデータストリームの例には，定期的なポーリングによって駆動するプル型のセンサーがある．

例えば，クーリングシステムによって冷却されるデータセンターを考える．このようなデータセンターでは，部屋の温度や湿度を監視するためにセンサーが設置され，制御システムはこれらのセンサーを監視する. [6] ではこの例を，温度センサーと湿度センサーからの 2 つのデータストリームを備えるシステムとしてモデル化しており，本研究でも同じモデルを使用する．説明に用いる図 1〜4 は，文献 [6] と同じものである．

図 1 は，データアイテムの属性を示している．LocationID はセンサーの設置場所，Value はセンサーが読む値，Timestamp はデータアイテムがシステムに入った時間を示す．

```
SELECT      COUNT (DISTINCT A.LocationID)
FROM        Temperature A, Humidity B
WHERE       A.LocationID = B.LocationID and
            A.Value > Threshold_t and B.Value > Threshold_h
WINDOW      1 min;
```

図 2　問合せの例：Q1
Fig. 2　An example of queries : Q1

```
SELECT      A.LocationID, MAX (A.Value), MAX (B.Value)
FROM        Temperature A, Humidity B
WHERE       A.LocationID = B.LocationID
GROUP BY    A.LocationID
WINDOW      1 hour;
```

図 3　問合せの例：Q2
Fig. 3　An example of queries : Q2

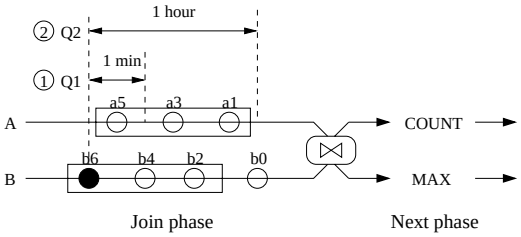

図 4　共有ウィンドウ結合
Fig. 4　The shared execution of two windows

図 2 と図 3 は，スライディングウィンドウを用いた連続的問合せの例を示している．図 2 の問合せは，温度と湿度の値が閾値を上回る場所の数を，最近の 1 分以内のデータについて求めることを表している．問合せの構文における WINDOW 節は，現在から，過去の特定された時間までにセンサーから到着したデータに対して，問合せを実行することを示している．また，図 3 の問合せは，場所ごとの最高温度と最高湿度を，最近の 1 時間以内のデータについて連続的に報告することを表している．これらのような問合せが，システムにデータが到着する度に連続的に実行される．

本研究では，バッファの更新は問合せ処理と排他的に行なう．つまり，実行中の問合せが処理されるまでは対象のデータがバッファに必ず残っており，実行中の問合せが中断されることはない．また，データストリームの各データアイテムを保存するバッファは配列で実装されている．

3.2　共有ウィンドウ結合

図 4 は，図 2 と図 3 の問合せの例を用いた共有ウィンドウ結合を表している．Q_1 と Q_2 は，それぞれ図 2 と図 3 で示した問合せであり，問合せの左の数字が示す順に実行される．情報源 A と B からそれぞれ図 1 で示したデータアイテムが左から到着する．白い丸は全ての問合せが実行されたデータを表し，黒い丸は実行されていない問合せが残っているデータを表している．データを囲む四角は，メモリ上のバッファを表している．バッファにデータがある間に問合せが実行され，新しいデータが到着するごとに古いデータが消える．

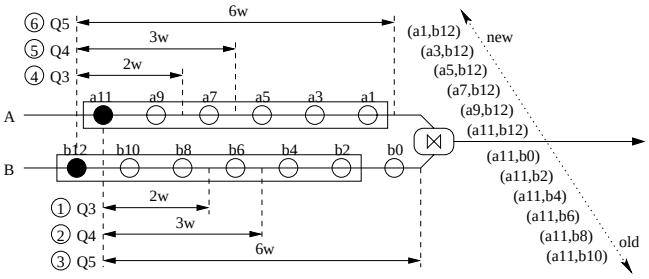

図 5　FCFS を用いた共有ウィンドウ結合
Fig. 5　Scheduling the shared window joins using FCFS

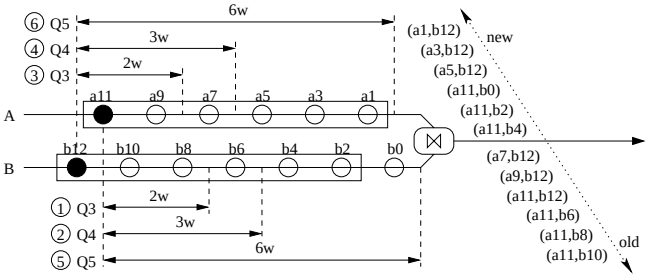

図 6　MQT を用いた共有ウィンドウ結合
Fig. 6　Scheduling the shared window joins using MQT

4.　従 来 手 法

4.1　FCFS

FCFS（First Come First Served）は，データの到着順に問合せを実行し，どの問合せも平等に実行するスケジューリング手法である．図 5 は，a_{11} と b_{12} がほぼ同時に到着したときの，FCFS による問合せのスケジューリングを表している．問合せの左の数字は，問合せの実行順序を示している．

次に実行する問合せを選択するには，実行されていない問合せが残っている，最も古いデータが見つかればよい．バッファを配列で実装すると，このデータは二分探索で見つかる．ここで，バッファ内のデータ数を $|s|$ とすると，次に実行する問合せを選択する計算量は $O(\log |s|)$ となる．

4.2　MQT

MQT（Maximum Query Throughput）[6] は，共有ウィンドウ結合のスループットを最大にするスケジューリング手法である．早く処理が終わる問合せを優先して処理することで，スループットを最大にしている．問合せごとに実行優先度を求め，各データに対する実行待ちの問合せのうち，優先度の高い問合せから実行する．優先度は，各問合せのウィンドウの大きさ，各問合せと同じウィンドウ幅を持つ問合せの数から求められる．

図 6 は，a_{11} と b_{12} がほぼ同時に到着したときの，MQT による問合せのスケジューリングを表している．ここでは，問合せの優先度は $Q_4 > Q_3 > Q_5$ となっており，Q_4 の処理結果に Q_3 が含まれるため，問合せの左の数字が示す実行順序となる．1 つの問合せが各データに対して実行されるため，実行待ちの問合せの優先度が同じになる可能性がある．同じ優先度の場合，古いデータに対する問合せから実行される．例えば，b_{12} の Q_4 は必ず $a11$ の Q_4 の後に実行される．

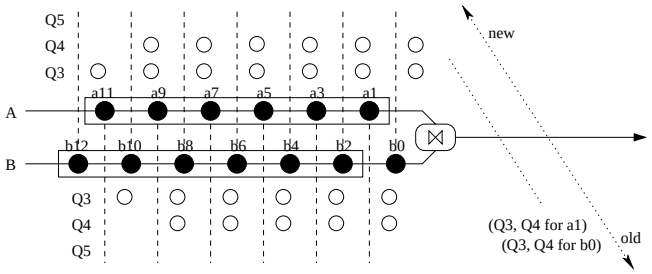

図 7　データ到着率が高いときの MQT
Fig. 7　MQT at high data arrival rates

次に実行する問合せを選択するには，次に実行すべき問合せが実行されていないデータが見つかればよい．図 6 では，全てのデータに対する Q_4 が実行された後に Q_5 が実行される．したがって，最も新しいデータに対する Q_4 が実行されていれば，次に実行すべき問合せは Q_5 となり，Q_5 が実行されていないデータを見つけることになる．バッファを配列で実装すると，このデータは二分探索で見つけることが出来る．ここで，バッファ内のデータ数を $|s|$ とすると，実行する問合せを選択する計算量は $O(\log |s|)$ となる．これは FCFS の計算量と同じである．

4.3 問題定義

どのような問合せ処理システムにおいても，同時に扱うことができるデータ数や問合せ数は，CPU やメモリといったリソースに制限される．ストリーム処理システムでは，システムに到着したデータはメモリ上のバッファに一時的に保存され，バッファに存在するデータに対して問合せが実行される．つまり，データ到着率が高い状況が続くと，問合せが実行される前にデータがメモリから消える可能性がある．

図 7 は，データ到着率が高いときの，MQT による問合せのスケジューリングを表している．点線の横の丸は，実行された問合せを表している．例えば，b_0 に対する Q_3 と Q_4 は既に実行済みである．b_0 に対する Q_5 については，Q_5 が実行される前にバッファから b_0 が消えたため，b_0 に対する Q_5 は失敗となる．問合せが失敗しない間は，スループットが最大になる MQT が多くの問合せを処理するが，問合せが失敗しづらい FCFS の方が最終的に多くの問合せを処理する可能性がある．つまり，スケジューリングに MQT を用いることで短期的な問合せ処理数は増加するが，長期的に見れば，問合せが失敗しづらい FCFS を用いる方が問合せ処理数が増加する可能性がある．

また図 7 より，Q_5 が主に失敗することがわかる．つまり，MQT では問合せ成功率に偏りが生じてしまう．これは，処理に時間のかかる問合せを発行したユーザへのサービスが極端に悪く，処理が速く済む問合せを発行したユーザへのサービスを優先していることになる．このようなサービスは不平等であり，改善が必要と言える．問合せのスケジューリングにおいて，スループットは考慮すべき重要な要素の 1 つである．ただし，ストリーム処理システムにおいては，スループットだけではなく，問合せごとの成功率も考慮することが非常に重要となる．

5. 提案手法

本節では，短期的なスループットだけでなく，長期的な問合せ処理数においても良い性能を示すスケジューリング手法，ExT（Executable interval × Throughput decision factor）を提案する．

RxW [7] では，人気のあるデータと人気のないデータの配信の両方を考慮するため，データごとに要求数と要求されてからの待ち時間を掛合せた値をスケジューリングに用いる．本研究は，問合せ成功率とスループットの両方を考慮する．RxW と同様に，問合せ成功率を考慮した値である問合せ実行可能期間（Executable interval）と，スループットを考慮した値であるスループット決定要素（Throughput decision factor）をそれぞれ求め，2 つを掛合わせた値を用いてスケジューリングを行う．

まず，アルゴリズムの説明に必要な変数を定義し，続いて 3 種類の ExT を提案する．これら 3 つのスケジューリング手法の違いは問合せ選択時間の長さだけであり，求まるスケジューリングは全て同じになっている．

5.1 変数定義

あるデータストリームを S とすると，$|S|$ は ∞ であり，S の各データは $S_i(0 \leqq i \leqq \infty)$ と表すことができる．また，S に割当てられるバッファを s とすると，s の各データは $s_i(0 \leqq i \leqq |s|-1)$ と表すことができる．本研究では配列でバッファを実装するため，$|s|$ は配列の要素数に相当する．例えば，s_0 はバッファ内で最も古いデータを示し，$s_{|s|-1}$ はバッファ内で最も新しいデータを示す．特に，問合せが全て実行されていないデータのうち，最も古いデータを s_{oldest}，最も新しいデータを s_{newest} と表す．

N 種類の問合せをウィンドウの大きさで昇順に並べたリストを L，L の各問合せを $L_j(1 \leqq j \leqq N)$ と表す．このとき，$s_i(0 \leqq i \leqq |s|-1)$ に対する $L_j(1 \leqq j \leqq N)$ を $Q[s_i][L_j]$ と表す．例えば，L_j を実行可能な，最も古いデータに対する問合せは $Q[s_{oldest}][L_j]$ となる．問合せ情報は，ウィンドウ幅を表す win，同じウィンドウ幅を持つ問合せの数を表す $qnum$，問合せの実行可能期間を表す e，スループット決定要素を表す t，E×T 値を表す et，同じウィンドウ幅を持つ問合せのうち最も高い問合せ成功率を表す s を持っている．e，t，et，問合せ成功率については 5.2 で後述する．例えば，$Q[s_{oldest}][L_N].e$ は，s において最も長いウィンドウを持つ問合せが実行されていない，最も古いデータに対する問合せの実行可能期間を表す．

また，データ到着率を以下のように定義する．S の到着率を測定する間隔を $S_{interval}$，$S_{interval}$ 内に到着したデータの個数を S_{num} とすると，S のデータ到着率は次の式で定義される．

$$S_{rate} = \frac{S_{num}}{S_{interval}}$$

ExT の説明の為に，3.1 で述べたモデルを用いる．以上の定義において，温度センサーが発するデータストリームを A，A に割当てられるバッファを a と定義する．同様に，湿度センサーに対しても B と b を定義する．データ到着率も，A と B に対してそれぞれ定義する．

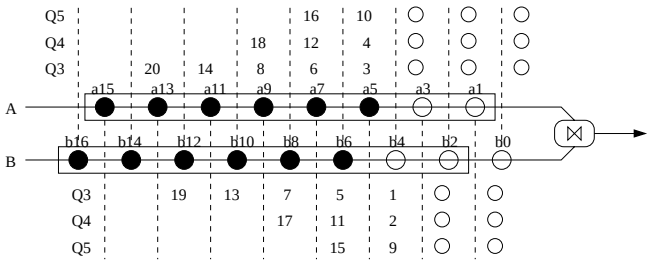

図 8　ExT を用いた共有ウィンドウ結合

Fig. 8　Scheduling the shared window joins using ExT

5.2　Naive ExT

ExT では，E と T という値を定義する．E と T は，それぞれ小さい値ほど優先度を高くし，実行待ちの問合せについて E×T 値を求め，値が小さい順に問合せを実行する．

まず，E を定義する．E は問合せの実行可能期間を表し，次の式で定義される．

$S_{rate} = 0$ のとき, $Q[s_i][*].e = 1$

$S_{rate} \neq 0$ のとき, $Q[s_i][*].e = \dfrac{\text{バッファ内の位置}}{\text{データ到着率}} = \dfrac{(i-1)}{S_{rate}}$

E はデータの到着率やバッファ内の位置によって値が決まるため，データの到着状況や問合せの処理状況を反映する．E が小さい順に問合せを実行すれば，データの到着順に問合せを実行することになる．

次に，T を定義する．T はスループット決定要素を表し，次の式で定義される．

$j = 0$ のとき, $Q[*][L_j].t = \dfrac{Q[*][L_j].win}{Q[*][L_j].qnum}$

$j \neq 0$ のとき, $Q[*][L_j].t = \dfrac{Q[*][L_j].win - Q[*][L_{j-1}].win}{Q[*][L_j].qnum}$

T は処理するウィンドウが短いほど小さい値になり，複数の問合せが同じウィンドウ幅を持っているほど小さい値になる．T が小さい順に問合せを実行すれば，スループットが最大になるように問合せを実行する．従来手法の MQT では，この値の逆数を用いてスケジューリングを行っている．

図 8 は，各問合せの成功率を満たしたときの，ExT による問合せのスケジューリングを表している．点線の横の丸は実行された問合せを表しており，数字は問合せの実行順序を表している．ここでは，E×T 値が小さい順に，20 番目までの実行順序を示している．

上記の優先付けでは，極端に長いウィンドウを持つ問合せの場合，E が小さいために処理に余裕がなくても，T が大き過ぎるために優先度が低くなる可能性がある．したがって，問合せが失敗し，ユーザが要求する問合せ成功率を下回る可能性がある．これに対処するため，要求を下回るときは問合せが必ず成功しつつ，そのときにスループットが高くなるようにスケジューリングを行う．

本研究では，ユーザに後述の図 10 の形式で問合せ成功率を指定させる．これにより，処理が後回しにされる問合せの成功率を保証する．以下に，各問合せが満たす成功率を求めるアルゴリズムを示す．

```
base = 1; // 最も短いウィンドウを持つ問合せ
N = ウィンドウ幅の異なる問合せの数;
while(base ≦ N){
    MaxS = 0; // 最も高い問合せ成功率
    Qno = 1; // MaxS の成功率の問合せ
    for(int i = base; i ≦ N; i++){
        if( Q[*][L_i].s ≧ MaxS){
            MaxS = Q[*][L_i].s;
            Qn0 = i;
        }
    }
    for(; base ≦ Qno; base++){
        Q[*][L_base].s = Q[*][L_Qno].s
    }
}
```

共有ウィンドウ結合における問合せの処理結果は，より長いウィンドウを持つ問合せに含まれることになる．したがって，要求される最も高い問合せ成功率を満たすことで，それよりも短いウィンドウを持つ全ての問合せで要求される成功率も満たすことができる．このとき，短いウィンドウを持つ問合せの成功率は，要求される最も高い問合せ成功率となる．要求される最も高い問合せ成功率の問合せよりも長いウィンドウを持つ問合せがある場合，以上で述べた方法で，残りの問合せに対して満たすべき成功率を求める．

以下に Naive ExT のアルゴリズムを示す．

（1） E×T 値を求める．

```
exec = null; // 次に実行する問合せ
for(未処理のデータに対する){
    for(実行待ちの問合せ){
        Q[s_i][L_j].e を求める;
        Q[s_i][L_j].t を求める;
        Q[s_i][L_j].et を求める;
        exec = E×T値が最小の問合せ;
    }
}
```

（2） $Q[s_{oldest}][L_N]$ が実行されるように修正する．

```
if(要求された成功率を下回る){
    plan[s_i][L_j] = Q[s_i][L_j] をE×T値でソート;
    MAXtput = 0;
    for(plan[s_oldest][L_N] より前に実行される問合せと,
          plan[s_oldest][L_N] を入れ替えた実行順序){
        ptime = plan[s_oldest][L_N] の実行までの処理時間;
        tput = plan[s_oldest][L_N] の実行までの問合せ処理数;
        if( (plan[s_oldest][L_N].e ≧ ptime) &&
            (tput ≧ MAXtput) ){
            MAXtput = tput;
            exec = plan[s_i][L_j];
        }
    }
}
```

（1）の計算量は $O(|s| \cdot N)$,（2）の計算量は $O(|s| \log |s|) + O(|s| \cdot N)$ であり，全体の計算量は $O(|s| \cdot N) + O(|s| \log |s|)$ である．従来手法の MQT と FCFS の計算量は $O(\log |s|)$ であり，これらに比べて Naive ExT の計算量は非常に大きくなっている．Naive ExT ではスケジューリングに時間がかかり過ぎてしまうので，求められる問合せの実行順序を変えずに，計算量だけを減らす工夫が必要となる．

5.3 Exhaustive ExT

Exhaustive ExT では，Naive ExT に動的計画法を用いることで計算量を減らす．ExT におけるスケジューリングの最適解は，$Q[s_{oldest}][L_N]$ を実行するデータがバッファから消える直前まで，E×T 値が小さい順に問合せを実行し，その後に $Q[s_{oldest}][L_N]$ を実行することである．本研究では，1 つ問合せが実行されるごとに次に実行する問合せを選択する．つまり，$Q[s_{oldest}][L_N]$ が，E×T 値が最小の問合せの次に実行できるかどうかを再帰的に判定することで最適解が求まる．

以下に Exhaustive ExT のアルゴリズムを示す．Naive ExT との違いは，（2）を次のように処理する点である．

（1） Naive ExT と同じ手順.

（2） E×T 値が最小の問合せの次に，

$Q[s_{oldest}][L_N]$ を実行できるか判定.

$ptime$ = E×T値が最小の問合せの処理時間;

$if(Q[s_{oldest}][L_N].e > ptime)$ {

$exec = Q[s_{oldest}][L_N]$;

}

（1）の計算量は $O(|s| \cdot N)$,（2）の計算量は $O(1)$ であり，全体の計算量は $O(|s| \cdot N)$ である．実験結果を見ると，スケジューリングにかかる時間が全体の処理時間に対して無視できないため，さらに計算量を減らす工夫が必要となる．

5.4 Pruned ExT

Pruned ExT では，共有ウィンドウ結合の特徴を利用して計算量を減らす．共有ウィンドウ結合では，同じ種類の問合せは必ず古いデータから実行されなければならない．したがって，全ての問合せに対してではなく，$Q[s_{oldest}][L_j]$ の E×T 値だけを求めればよい．

以下に Pruned ExT のアルゴリズムを示す．Exhaustive ExT との違いは，（1）を次のように処理する点である．

（1） $Q[s_{oldest}][L_j]$ の E×T 値を求める.

$exec = null$; // 次に実行する問合せ

for(L に存在する各問合せ){

$Q[s_{oldest}][L_N].e$ を求める;

$Q[s_{oldest}][L_N].t$ を求める;

$Q[s_{oldest}][L_N].et$ を求める;

$exec$ = E×T値が最小の問合せ;

}

（2） Exhaustive ExT と同じ手順.

（1）の計算量は $O(N)$,（2）の計算量は $O(1)$ であり，全体の計算量は $O(N)$ である．実験結果で示すが，問合せの選択時間は処理時間に対して無視できるほど短くなっている．

表 1 実験に用いた計算機環境

Table 1 Experiment environment

CPU	Intel Pentium III 1400MHz
メモリ	512MB
OS	Red Hat Linux 9 (2.4.20)
開発言語	Java (J2SE 1.4.2)

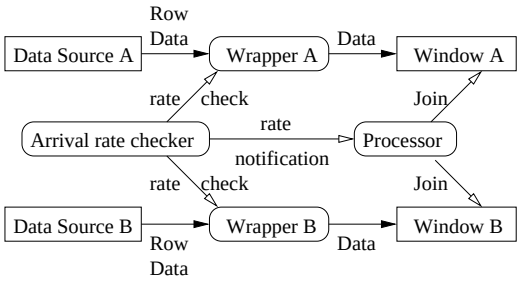

図 9 プロトタイプシステムの構成

Fig. 9 Prototype system architecture

SELECT	*
FROM	Data Source A, Data Source B
WHERE	A.LocationID = B.LocationID
WINDOW	[sec]
SUCCESS	80 %

図 10 実験に用いる問合せのテンプレート

Fig. 10 Query template used by experiment

6. 評価実験

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．

6.1 実験環境

本システムの実験に用いた計算機環境は表 1 の通りである．図 9 が示すように，ラッパーは情報源から到着するデータを受取り，システムに到着した時刻をデータに付加してバッファに入れる．到着率チェッカは，到着したデータ数を単位時間ごとに集計し，プロセッサに到着率を通知する．プロセッサは，通知を用いてスケジューリングを行い，問合せを実行する．

1 つのデータに対し，図 10 の形式の問合せが 10 個実行される．議論を簡単にするため，SELECT 節，FROM 節，WHERE 節は全て同一とした．WINDOW 節は秒単位で指定する．本研究では SQL 構文を拡張し，SUCCESS 節で問合せ成功率を指定する．今回は，全ての問合せが 80%の場合と，100%の場合で実験した．データは図 1 の形式で人工的に生成する．Value には任意の整数が入り，Timestamp にはデータがシステムに到着した時刻が入る．LocationID は全て整数 1 とした．

本実験では，データは 2 つの情報源から到着する．したがって，それぞれの情報源からのデータに対し，データ 10000 個分のバッファを 1 つずつ用意した．実験を開始するときのバッファの初期状態は，古い 0.1%のデータに対する問合せを全て処理済みとし，新しい 99.9%のデータに対する問合せを全て未処理とした．また，データ到着率は 60 秒ごとに測定する．

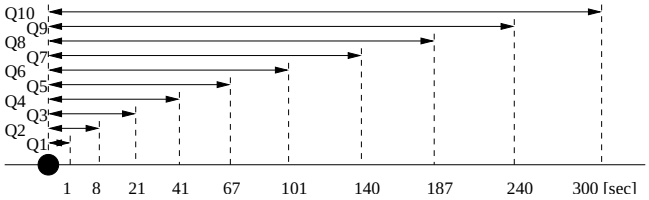

図 11　ウィンドウの分布
Fig. 11　Distribution of windows

本実験では文献 [6] と同様に，データが予測不可能に到着するように，指数分布とパレート分布を組合せてバースト的なデータストリームを発生させる．パレート分布は ON，OFF の期間によってパケットが送信されるネットワークトラフィックのシミュレーションで用いられる．ON の期間にデータがバースト的に発生し，OFF の期間はデータが発生しない. [6] では，ON と OFF の間隔はパラメタ $\lambda = 1.0$ の指数分布に従い，1 回に発生されるデータの数はパラメタ $\alpha = 1.05$ のパレート分布に従う．ここで，指数分布の期待値は 1.0，パレート分布の期待値は 21.0 である．本実験では，バッファの大きさとデータ到着率の関係に合せ，指数分布の期待値を 100.0，パレート分布の期待値を 840.0 になるように修正した．したがって，1 秒間に到着するデータ数の期待値は 8.4 となっている．

6.2　実験 1：問合せ成功数と失敗数

図 11 は，実験に用いた問合せのウィンドウの分布を表している．この分布は，MQT と FCFS のスループットの差が最大になる分布である．問合せ間のウィンドウ差が，1 つ前の問合せ間のウィンドウ差よりも大きいときにこの分布となる．したがって，問合せ Q_1 は各問合せ間のウィンドウ差よりも小さくなっている．MQT では，全てのデータに対する問合せのうち，最も短いウィンドウを持つ問合せから処理する．つまり，全てのデータに対する Q_1 が処理し終わったら Q_2，全てのデータに対する Q_2 が処理し終わったら Q_3，という順で問合せを実行する．

図 12〜図 15 は，問合せ成功率が 80%と 100%の Pruned ExT，成功率が 80%の Exhaustive ExT，従来手法である MQT と FCFS について，166 分の間に図 11 の問合せを処理したときの実験結果を示している．図 12〜図 14 においては，問合せ処理を行う 166 分のうち，最初の 33 分はデータの到着状況によって処理結果が大きく異なるため，33 分から 166 分までの結果を示した．それぞれ，5 回ずつ実験を行ったときの平均をとっている．

図 12 は，各手法の問合せ成功数を示している．Exhaustive ExT は問合せの選択に時間がかかるため，他の手法に比べて成功した問合せ数がおよそ半分になっている．問合せ選択時間については，6. 3 で詳しく述べる．図 13 は，図 12 において FCFS を基準としたときの，各手法の問合せ成功数を示している．実験を開始して 100 分経過したあたりから，Pruned ExT-80 で最も多くの問合せが成功しており，続いて Pruned ExT-100, FCFS，MQT の順に多く成功している．データがバースト的に到着すると，FCFS 以外の手法ではウィンドウの短い問合せを中心に実行できる．したがって，データが到着した直後の短期間だけを見れば，FCFS よりもその分だけ多くの問合せが処理されている．

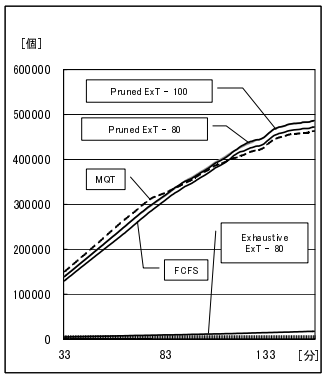

図 12　問合せ成功数
Fig. 12　# of processed queries

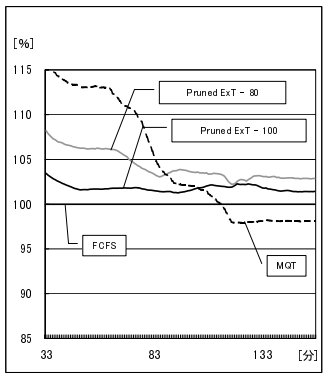

図 13　問合せ成功数（対 FCFS）
Fig. 13　Ratio to FCFS

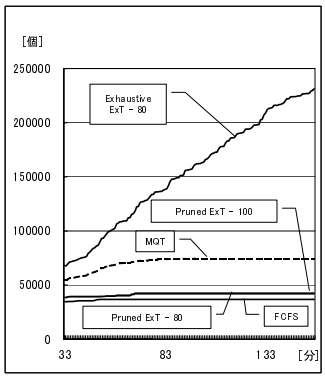

図 14　問合せ失敗数
Fig. 14　# of unprocessed queries

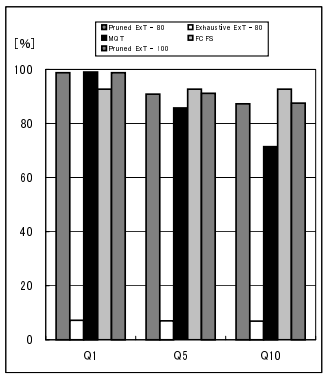

図 15　問合せごとの成功率
Fig. 15　Processed querites rate

図 14 は，各手法の問合せ失敗数を示している．問合せの選択に時間がかかる Exhaustive ExT が最も多くの問合せを失敗している．MQT はウィンドウの短い問合せを優先して実行するため，ウィンドウの長い問合せが多く失敗する．結果的に，FCFS や Pruned ExT よりも多くの問合せが失敗している．FCFS ではウィンドウの長い問合せを実行している間に，多くの問合せが失敗している可能性がある．一方 Pruned ExT では，問合せ成功率を満たしている間はスループットを重視したスケジューリングとなる．つまり，ウィンドウの長い問合せが失敗する代りに他の問合せが成功するため，FCFS よりも Pruned ExT の方が失敗した問合せが少なくなっている．

図 15 は，166 分が経過した時点の，各手法の問合せごとの成功率を示している．問合せ Q_1 から問合せ Q_{10} の 10 個の問合せの代表として，Q_1，Q_5，Q_{10} を選択した．Pruned ExT, FCFS ともに，要求された成功率 80%を全て満たしている．逆に MQT では，Q_{10} の成功率が 70%ほどであった．Q_1，Q_5，Q_{10} の成功率の差から分かるように，MQT によるサービスは，ウィンドウが短い問合せほど優先される．特に，Q_{10} に対するサービスが極端に悪くなっていることが分かる．Pruned ExT では，MQT のようにスループットを向上させ，なおかつ全ての問合せで要求される 80%を満たす結果となった．

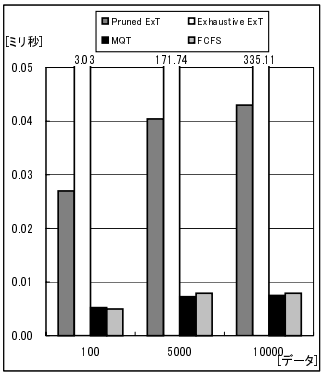

図 16　1 回の問合せ選択時間
Fig. 16　Query selection time

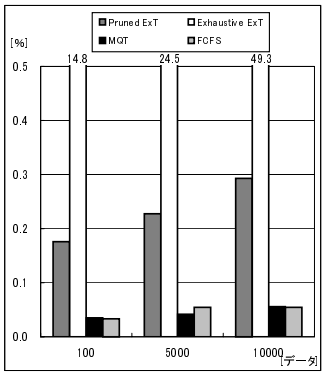

図 17　問合せ選択／全体の処理
Fig. 17　Ratio to process time

今回のデータ到着状況，問合せの処理能力では，Pruned ExT-80 と Pruned ExT-100 における実行されない問合せの数に差が出なかった．そのため，スループットを重視した分，Pruned ExT-80 の方が良い結果となったと考えられる．

6.3　実験 2：問合せ選択時間の変化

図 16 は，バッファサイズを変化させたときの，各手法の問合せ選択時間を示している．166 分の間にかかった問合せ選択時間を，問合せ選択回数で割った値を示した．Exhaustive ExT の選択時間が極端に長く，続いて Pruned ExT が長くなっている．MQT と FCFS はほぼ同じ時間で，Pruned ExT よりも短かった．各手法ともバッファの大きさに比例して選択時間が長くなるが，Exhaustive ExT に比べ，他の手法はバッファサイズの影響が小さかった．

図 17 は，バッファサイズを変化させたときの，全体の処理時間に占める問合せ選択時間の割合を示している．全体の処理時間には，結合時のバッファ走査や，結合用バッファを最新の状態にするための時間が含まれている．MQT や FCFS に比べて Pruned ExT の方が問合せ選択時間は長いが，全体の処理時間から見ると，問合せの選択にかかる時間は無視できるほど小さいといえる．各手法ともバッファの大きさに比例して，全体の処理時間に占める問合せ選択時間の割合は大きくなっている．図 16 のときと同様に，Exhaustive ExT 以外の手法はバッファサイズの影響は小さかった．

6.4　実験 3：様々なウィンドウの分布

スループットや問合せ成功率には，ウィンドウの分布が影響する．図 11 の分布以外に，短いウィンドウが 80%を占める分布，長いウィンドウが 80%を占める分布，短いウィンドウと長いウィンドウが 50%ずつを占める分布が考えられる．図 11 では問合せ間のウィンドウ差が徐々に長くなっているが，4 種類の分布に対して，問合せ間のウィンドウ差が短い場合も考えられる．本実験では，図 11 の分布を含めた 8 種類の分布について実験を行った．Pruned ExT の問合せ実行順序は，分布によっては FCFS と同じ，あるいはほぼ同じになる．どの分布でも，Pruned ExT の性能は MQT と FCFS を下回ることはなかった．

7. おわりに

本稿では，スループットと問合せ成功率の両方を考慮したスケジューリング手法，ExT を提案した．従来手法との比較実験で示したように，短期的なスループットだけでなく，長期的な問合せ処理数においても良い結果となった．また，ExT では実時間での，状況の変化に適応した動的なスケジューリングが可能である．バッファの大きさが変化しても，問合せ選択時間に影響しない手法であることも示した．

今後の課題として，Load Shedding との併用が考えられる．リソースの制限を越える程にデータが到着すると，ユーザが要求する成功率を満たすことができない場合がある．Load Shedding との連携により，あらゆるデータ到着率においても，スループットを高くしつつ，要求される問合せ成功率を満たすことができると考えられる．

謝　辞

本システムの評価にご協力頂いた，同研究室の広木伯哉さんに心より感謝致します．

文　献

[1] D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul and S. B. Zdonik: “Monitoring streams - a new class of data management applications”, Proceedings of the 28th International Conference on Very Large Data Bases, pp. 215–226 (2002).

[2] 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”, Proceedings of the First Biennial Conference on Innovative Data Systems Research (2003). Available at http://www-db.cs.wisc.edu/cidr/cidr2003/program/p22.pdf.

[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”, Proceedings of the First Biennial Conference on Innovative Data Systems Research (2003). Available at http://www-db.cs.wisc.edu/cidr/cidr2003/program/p24.pdf.

[4] S. D. Viglas and J. F. Naughton: “Rate-based query optimization for streaming information sources”, Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, pp. 37–48 (2002).

[5] M. A. Hammad, W. G. Aref and A. K. Elmagarmid: “Stream window join: Tracking moving object in sensor-network databases”, Proceedings of the 15th International Conference on Scientific and Statistical Database Management, pp. 75–84 (2003).

[6] M. A. Hammad, M. J. Franklin, W. G. Aref and A. K. Elmagarmid: “Scheduling for shared window joins over data streams”, Proceedings of the 29th International Conference on Very Large Data Bases, pp. 297–308 (2003).

[7] D. Aksoy and M. Franklin: “RxW : A scheduling approach for large-scale on-demand data broadcast”, IEEE/ACM Transactions on Networking, **7**, 6, pp. 846–860 (1999).

[8] N. Tatbul, U. Çetintemel, S. B. Zdonik, M. Cherniack and M. Stonebraker: “Load shedding in a data stream manager”, Proceedings of the 29th International Conference on Very Large Data Bases, pp. 309–320 (2003).