処理負荷に基づいた共有ウィンドウ結合の動的スケジューリング

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　近年，データストリームの処理に高い関心が集まっている．データが予測不可能に到着する環境では，処理システムに高い負荷がかかる可能性がある．本稿では，処理負荷に基づいた結合処理の動的スケジューリングの手法を提案する．共有ウィンドウ結合のスループットを最大にする手法を用い，処理負荷が高いときは，スループットを高くしつつ実行できない問合せの数を少なくする．実験により，我々の提案手法の有効性を検証する．
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

Workload-Based Dynamic Scheduling for Shared Window Joins

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　Recently, there has been a growing interest in processes on data stream. If stream data arrive continuously in an unpredictable rate, the workload of a system could sometimes become high. In this paper, we propose a workload based dynamic scheduling method for join queries. Our method attempts to maximize the throughput of join queries by exploiting a technique of shared window joins as well as to reduce the number of dropped queries caused by high workload. Some experimental results show that our scheduling method can work well in many cases.
Key words　Stream, Continuous Query, Window join, Scheduling

1. はじめに

近年，データストリームを効率良く処理するための研究が盛んに行われている．特に，連続的問合せ（Continuous Query）[1] を用いたデータストリームの処理が注目を浴びており，これまでにも連続的問合せを処理するシステムとして STREAM [2] や TelegraphCQ [3] が提案されている．これらでは，従来のリレーショナルデータベースで用いられていた選択や射影，結合などの処理を，実際にデータストリームの処理に適用し，システムとして実装している．

TelegraphCQ では CACQ [4] や PSoup [5] を用いて，問合せの最適化を行っている．これらでは，情報源の性質の変化に合せて問合せの実行順序を変更したり，問合せ間で処理結果を共有したりしている．他にも，連続的問合せの最適化手法が提案されている [6]〜[9]．これらでは，スループットが高くなるように問合せの実行順序を変更したり，問合せ間で結果を共有できるものをクラスタ化して処理したりしている．

連続的問合せの中で，本研究では結合処理に焦点を当てる．選択や射影，結合は，リレーションに対する従来の問合せにおいて基本的な処理である．これらは同様に，連続的問合せにおいても基本的な処理である．また，結合は選択や射影と比較して処理コストがかかるため，結合を効率良く行うことは重要であるといえる．これまでにも，データストリームの結合に焦点をおいた研究が数多くなされている [6], [7], [10]〜[15].

連続的問合せの結合処理で用いられる方法の 1 つにウィンドウ結合 [10] がある．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結合対象とする結合方法である．ウィンドウ結合の処理結果を共有して処理することを，共有ウィンドウ結合と呼ぶ．連続的問合せは短時間に繰返し実行され，問合せ間で結果を共有できる場合が多い．したがって共有ウィンドウ結合により，ウィンドウ結合を効率良く処理することができる．

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の手法として，MQT（Maximum Query Throughput）[6] が提案されている．MQT は，最も早く処理が終わる問合せから実行するように問合せをスケジューリングすることで，問合せのスループットを最大にする．そのため，時間のかかる問合せの処理を後回しにしている．この手法では，全てのデータに全ての問合せが実行できる処理負荷を前提としており，処理負荷が高いときに，問合せが実行されないままバッファから消えるデータが生じる．つまり，実行できない問合せが生じてしまう．

本稿では，処理負荷に基づいて問合せを動的にスケジューリングし，スループットが高く，高い処理負荷でも実行できない問合せの数を少なくする手法を提案する．具体的には，共有ウィンドウ結合のスループットを最大にする MQT と，実行できない問合せの数を少なくする手法を，処理負荷に基づいて使い分ける．測定した処理負荷と，後で説明する閾値を比較しながら，その時に適したスケジューリングの手法を用いることで，双方の利点が活きるようにする．

本稿の構成は次の通りである．まず 2. で，本研究の関連研究について説明する．3. では，共有ウィンドウ結合と，そのスケジューリングの手法を説明する．ここで，MQT を用いたときの，処理負荷が高いときに生じる問題についても述べる．続く 4. で，提案する動的スケジューリングの手法について説明する．5. 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 6. で，まとめと今後の課題とする．

2. 関 連 研 究

情報源からの入力を考慮した結合処理の手法として，Mjoin [7] が提案されている．この手法は 2 つ以上の情報源があるときに，情報源からの入力をみて，結合を早く開始できる入力から処理を行うように問合せのスケジューリングを行う．本研究では，処理負荷に基づいて問合せのスケジューリングを行う．情報源からの入力だけではなく，その時点でシステムにどれだけ未処理の問合せが残っているかも考慮している点で，Mjoin とは異なる．

情報源からの入力と，入力に対する各問合せの出力を考慮した問合せのスケジューリングの手法として, [8] が提案されている．各問合せが 1 秒間にどれだけ入力を取ることができて，どれだけの出力タプルを生成するかが分かるとき，問合せの実行順序を最適にする．Mjoin と同様に，処理負荷を考慮に入れたスケジューリングは行われていない．

MQT を用いたシステムに [9] がある. [9] では，問合せ間のウィンドウの重なり具合を考慮し，問合せ間で処理結果を多く共有できるように問合せをクラスタ化する手法を提案している．本研究と同様に，このシステムでは共有ウィンドウ結合をするときに MQT を用いている．しかし，処理負荷について言及していない点で，本研究とは異なる．また，本研究が共有ウィンドウ結合そのものの性能向上を目指している点でも異なっている．

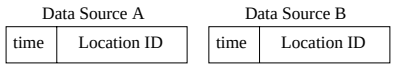

図 1　情報源 A，B から到着するデータ
Fig. 1　Data from data sources A and B

SELECT	*
FROM	Data Source A, Data Source B
WHERE	A.Location ID = B.Location ID
WINDOW	2 or 3 or 6 [sec]

図 2　問合せの例
Fig. 2　Example of queries

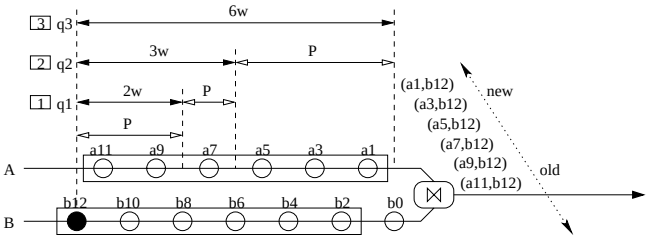

図 3　共有ウィンドウ結合による問合せの処理
Fig. 3　Processing shared window joins

3. 共有ウィンドウ結合のスケジューリング

本節では，まず共有ウィンドウ結合による問合せの処理を説明する．続いて問合せのスループットを最大にする MQT の動作と，処理負荷が高いときに生じる MQT の問題について述べる．その後，処理負荷が高いときに MQT の代りに用いる，実行できない問合せの数を少なくする FIFO について説明する．

3.1 共有ウィンドウ結合による問合せの処理

共有ウィンドウ結合とは，ウィンドウ結合の処理結果を問合せ間で共有する処理である．情報源 A と B からそれぞれ図 1 のデータが到着し，データ 1 つ 1 つに対して図 2 の問合せが 3 つ実行される状況を考える．各問合せの SELECT 節と FROM 節，WHERE 節は全て同一で，WINDOW 節はそれぞれ 2 秒，3 秒，6 秒である．図 3 はこの状況における，共有ウィンドウ結合による問合せの処理を示している．白丸は全ての問合せが実行されたデータを示し，黒丸は未処理の問合せが残っているデータを示している．データを囲む四角は，メモリ上のバッファを示している．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．

共有ウィンドウ結合を行うために，処理システムはウィンドウの小さい順に並んだ問合せのリストを持っている．このリストは問合せの追加，削除のときにのみ更新される．処理システムはリストに従って，ウィンドウの小さい順に問合せを実行する．図 3 において，問合せの実行順序は四角で囲った番号の順である．データ b_{12} に問合せが q_1，q_2，q_3 の順で実行され，このとき，白い矢印で囲まれた P の部分を処理する．つまり，前の問合せのウィンドウと重なる部分の処理結果を共有し，前の問合せのウィンドウとの差分だけを新たに処理する．

表 1　MQT で用いる行列
Table 1　MQT matrix

—	w_1	w_2	w_3
0	$\frac{1}{2w}$	$\frac{2}{3w}$	$\frac{2}{3w}$
w_1	—	$\frac{1}{w}$	$\frac{1}{w}$
w_2	—	—	$\frac{1}{3w}$

$$MQT(PW(0,w_2)) = max\{\frac{C_{01}}{pw_{01}}, \frac{C_{02}}{pw_{02}}\} = max\{\frac{1}{2w}, \frac{2}{3w}\} = \frac{2}{3w}$$

3.2　MQT による問合せのスケジューリング

3.2.1　MQT の動作

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手法である．処理システムはウィンドウの小さい順に並んだ問合せのリストを持っており，あるデータに対してウィンドウの小さい順に問合せを実行する．リストにある問合せの数，つまり 1 つのデータに実行される問合せの数を N とすると，MQT は N^2 の大きさの行列を用いて，どの問合せを処理したらスループットが最大になるかを判断する．例えば問合せが図 2 のとき，行列は表 1 となる．

行列は次のようにして作成される．ウィンドウの小さい順に並んだ問合せのリストにおいて，各問合せを $q_1 \cdots q_N$ とし，それぞれが持つウィンドウを $w_1 \cdots w_N$ とする．あるデータに対する現在処理が終了している問合せのウィンドウを w_i，同じデータに対する次に実行可能な問合せのウィンドウを w_j とすると，w_i と w_j の差分 PW（Partial Window）は以下の式となる．

$$PW(w_i, w_j) = \{pw_{ik} \mid pw_{ik} = w_k - w_i, \text{for } i+1 \leqq k \leqq j\}$$

あるデータに対して実行可能な問合せ全てについて行列を参照するので，実行可能な問合せの数だけ w_i と w_j の PW が求まる．$PW(0, w_j)$ は特殊な場合で，問合せの対象となるデータに 1 つも問合せが実行されていないことを表す．

次にウィンドウ w_i を持つ問合せの処理までに処理される問合せの数を C_i とする．またウィンドウ w_i を持つ問合せの数を $Queries(w_i)$ とすると，C_i は以下の式となる．

$$C_i = \sum_{l=1}^{i} Queries(w_l)$$

ウィンドウ w_i とウィンドウ w_j の差分を pw_{ij} とすると，pw_{ij} の間に処理される問合せの数 C_{ij} は以下の式となる．

$$C_{ij} = C_j - C_i$$

pw_{ij} を処理する時間は，pw_{ij} の長さに比例する．すなわち，pw_{ij} を処理する問合せのスループットの評価に $\frac{C_{ij}}{pw_{ij}}$ を用いることができる．あるデータに対して，$PW(w_i, w_j)$ のうちのどの pw_{ij} も処理可能であるとき，そのデータに対する問合せの最大のスループットは以下の式となる．

$$MQT(PW(w_i, w_j)) = max\{\frac{C_{ij}}{pw_{ij}} \mid pw_{ij} \in PW(w_i, w_j)\}$$

この値は 2 つのウィンドウから計算されるもので，その値は N^2 の大きさの行列に保存される．この行列は，問合せの追加，削除のときにのみ更新される．MQT は，処理を待っている全てのデータについて行列を参照し，最も値の大きかった問合せを実行する．

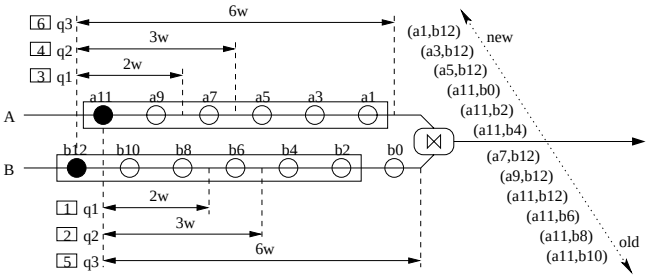

図 4　MQT によるスケジューリング
Fig. 4　Scheduling by MQT

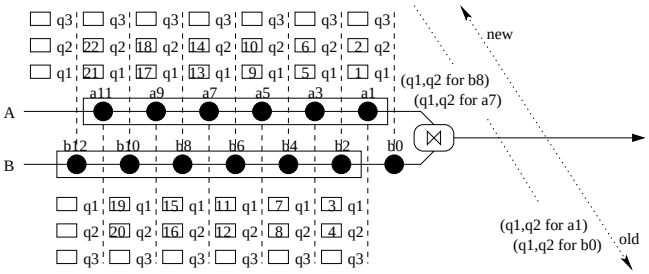

図 5　処理負荷が高いときに生じる MQT の問題
Fig. 5　The problem of MQT under heavy workload

図 4 は，以下の順でデータが到着，処理されるときの，MQT による問合せの処理を示している．

（1）　a_{11} が到着．
（2）　a_{11} に対する q_1 を実行．
（3）　b_{12} が到着．
（4）　a_{11} に対する実行可能な問合せと，b_{12} に対する実行可能な問合せについて行列を参照．
（5）　a_{11} に対する q_2 の値（行列の w_1 行 w_2 列の値 $\frac{1}{w}$）が最大なので，a_{11} に対する q_2 を実行．
（6）　同様にして行列を参照しながら実行する問合せを決定．

3.2.2　MQT の問題点

MQT はスループットを最大にするために，処理に時間のかかる問合せの処理を後回しにする．処理負荷によっては実行されないままバッファから消えるデータが生じ，その結果，処理される問合せの数が減る．

図 5 は，図 4 において次々と新しいデータが到着したときに生じる問題を示している．新しいデータが次々と到着すると，それらに対して問合せ q_1 と q_2 が優先して実行され，q_3 はいつまでも実行されない．到着したデータを一時的に保存しておくバッファの大きさは有限であるので，q_3 の実行待ちの間に，問合せの対象となるデータ b_{12} がバッファから消えてしまう．したがって，この後に処理負荷が低くなっても，b_{12} に対する問合せ q_3 を実行することは不可能になる．

[1] によると，ストリームデータの特性は次の通りである．

- rapid：早い到着
- unpredictable：予測つかない
- time-varying：時間によって変化

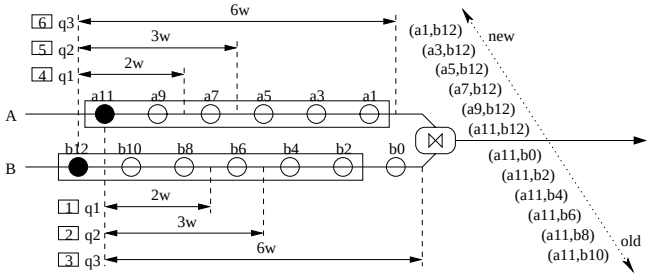

図 6　FIFO による問合せ処理
Fig. 6　Scheduling by FIFO

ストリームデータは早く到着するため，図 5 のような状況は十分起こりうる．したがって処理負荷が高いときは，MQT を用いない別のスケジューリングの手法を用いる方が良い．また，ストリームデータを処理するシステムの処理負荷は予測がつかず，時間によって変化する．すなわち効率的な処理をするためには，処理負荷を測定しながら，その時点の処理負荷に合ったスケジューリングの手法を用いることが必要不可欠である．図 5 では，データ b_{12} がバッファから消える前に問合せ q_3 を実行していれば，その問合せが実行できないという問題は発生しない場合があり得る．

3.3　FIFO による問合せのスケジューリング

システムの処理負荷が高いときに，MQT の代りに用いるスケジューリングの手法を説明する．実行できない問合せの数を少なくするために，データの到着順（FIFO）に問合せを実行する．図 6 は，FIFO による問合せの処理を示している．MQT と同様に，処理システムはウィンドウの小さい順に並んだ問合せのリストを持っており，先に到着したデータに対してウィンドウの小さい順に問合せを実行する．

4.　処理負荷に基づいた動的スケジューリング

本節では，処理負荷に基づいて問合せを動的にスケジューリングし，スループットが高く，高い処理負荷でも実行できない問合せの数を少なくする手法を提案する．まず提案手法の動作を説明し，その後，問合せのウィンドウの分布が与える問合せの実行順序への影響について述べる．

4.1　動的スケジューリングのタイミング

問合せの動的スケジューリングを行うタイミングは，処理負荷を測定するときである．処理負荷はある単位時間ごとに求められ，MQT を用いるか，FIFO を用いるかが決定される．システムは，どちらを用いて問合せを処理するかを示すフラグに従って処理する．したがって，問合せの処理中にスケジューリングの手法を切替えても，問合せの処理が途中で中断されることはない．

処理負荷を求めるために，単位時間に到着するデータの数を測定する．以降では，これをデータの到着率と呼ぶ．測定したデータの到着率を基に，バッファの半分の位置から，到着したデータの個数分のデータについて，どの問合せが未処理であるかを調べる．問合せがどれだけ未処理であるかによって，処理負荷が高いか低いかを判断する．

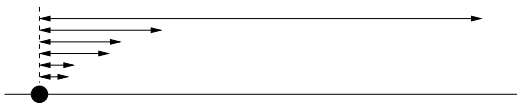

図 7　スループットと溢れる問合せのトレードオフ
Fig. 7　Trade-off between throughput and dropped query

表 2　行列から求まる問合せの重み
Table 2　Weight derived from the matrix

—	w_1	w_2	w_3
0	$\frac{1}{2w}$	$\frac{2}{3w}$	$\frac{2}{3w}$
w_1	—	$\frac{1}{w}$	$\frac{1}{w}$
w_2	—	—	$\frac{1}{3w}$

$\Rightarrow$

w_1	w_2	w_3
$\frac{2}{3w}$	$\frac{2}{3w}$	$\frac{1}{3w}$

4.2　問合せの重み付け

問合せがどれだけ未処理であるかは，未処理の問合せの数で判定せず，問合せに重みを付け，その重みを用いて判定する．図 7 は，問合せに重みを付けることで，スループットと，実行できない問合せの数のトレードオフが考えられることを示している．最も長いウィンドウだけが極端に長いとき，これを処理するよりも，他のデータに対する短いウィンドウを処理する方が効率が良い場合があり得る．MQT と FIFO の双方の利点を活かすには，問合せの重み付けが効果的である．

提案手法は，スループットが高く，かつ実行できない問合せの数を少なくすることが目標である．したがって，それら両方を考慮できる重み付けをする．本研究ではスループットに注目し，先に処理したらスループットが上がる問合せに大きい重みを付けた．これは，ある問合せを処理しなかったら，どれだけもったいないかの尺度になる．

問合せの重みを求めるために，MQT で求めた行列を利用する．MQT で求めた行列では，先に処理することでスループットが上がる問合せに大きい値が付いている．したがって，この行列を問合せの重み付けに用いることができる．MQT の行列は，まずどの問合せまで実行し，次にどこまで実行するかを示していると見ることができる．例えば，表 1 で作成した行列では，まずウィンドウ w_2 を持つ問合せまで実行し，次にウィンドウ w_3 を持つ問合せを実行する．

表 2 は，表 1 で作成した行列から求まる問合せの重みを示している．まず w_1 を持つ問合せの重みについて説明する．行列から，w_1 は単独で処理されることはなく，w_2 を処理する過程で，共に処理されることが判る．したがって，w_1 を持つ問合せの重みは，w_2 を持つ問合せの重みと同じ値になる．w_2 を持つ問合せの重みを求めると，w_2 は w_1 と共に処理されるので，0 から w_2 までを処理した値 $\frac{2}{3w}$ となる．w_3 を持つ問合せの重みを求めると，w_3 は w_2 を処理した後に処理されるので，w_2 から w_3 までを処理した値 $\frac{1}{3w}$ となる．

単位時間ごとに，バッファの半分の位置から，到着したデータの個数分のデータについて，未処理である問合せの重みの和を求める．未処理である問合せの重みの和を S とし，次に説明する閾値と比較することで，処理負荷が高いか低いかを判断する．

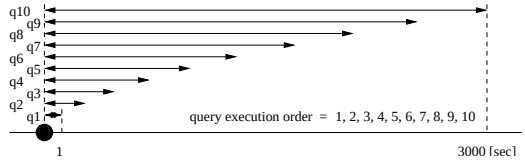

図 8　MQT と FIFO のスループットの差が最大になる分布
Fig. 8　MQT gets the best advantage to FIFO

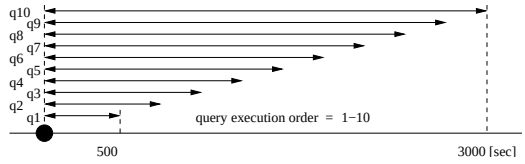

図 9　MQT と FIFO のスループットに差が出ない分布
Fig. 9　Query execution order of MQT is the same as FIFO

4.3　閾値の決定

処理負荷が高いか低いかを判断するために，閾値 α を決定する．閾値 α が高いほどスループットを重視したスケジューリングとなり，低いほど実行できない問合せの数を重視したスケジューリングとなる．

閾値 α を次のように決定する．まず，1 データに対する問合せの重みの和のうち，どれだけの問合せが未処理であれば負荷が高いと判断するかの割合を β（1 以下）とする．これは予め初期値として設定しておく．β が決まるとき，閾値 α は以下の式となる．

$\alpha = 1$ データに対する問合せの重みの和 $* \beta *$ 到着率

例えば表 2 において，1 データに対する問合せの重みの和は $\frac{2}{3w} + \frac{2}{3w} + \frac{1}{3w} = \frac{5}{3w}$ である．したがって $\beta = 0.2$ とすると，ウィンドウ w_3 を持つ問合せが実行されなかったら，処理負荷が高いと判断する．

4.4　ウィンドウの分布と問合せの実行順序

スループットと，実行できない問合せの数に影響を与える要因のうち，ウィンドウの分布は最も重要な要因の 1 つである．MQT ではウィンドウの分布によって問合せの実行順序が変わる．また，提案手法ではウィンドウの分布が図 7 のようなとき，大きいウィンドウを持つ問合せを実行するかどうかに影響する．

ランダムにウィンドウを決めたとき，そのウィンドウの分布は図 8〜図 15 の 8 種類で構成される．それぞれの問合せの実行順序は図中に示した．ここでは，ある問合せを実行するときに，どの問合せも共に処理するかを示している．例えば 1,2,3-10 なら，問合せ 3 を処理するときは，問合せ 4，5，6，7，8，9，10 も処理することを示している．

まず，最も両極端な動作となる図 8 と図 9 について説明する．図 8 は，MQT と FIFO のスループットの差が最大になる分布を示している．FIFO のように，1 つのデータに対して最も長いウィンドウを持つ問合せまで処理するのとは異なり，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．問合せの実行待ちデータが複数あるときの問合せの実行順序を，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，という順にすると，MQT と FIFO のスループットの差が最大になる．本研究では全てのウィンドウが異なるものとし，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．図 8，図 10，図 12，図 14 でも，各問合せ間のウィンドウ差は徐々に大きくなっている．

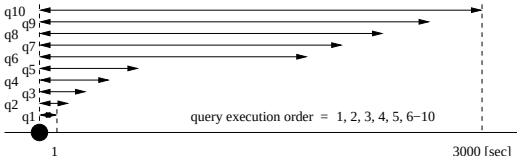

図 10　大 50%，小 50%でスループットに差が出る分布
Fig. 10　Large 50% and small 50% that MQT favors

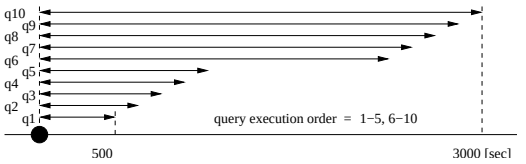

図 11　大 50%，小 50%でスループットに差が出ない分布
Fig. 11　Large 50% and small 50% that MQT doesn't favors

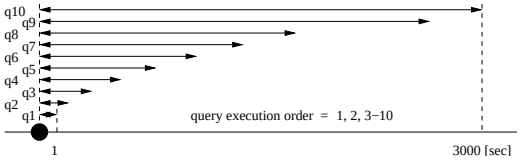

図 12　大 20%，小 80%でスループットに差が出る分布
Fig. 12　Large 20% and small 80% that MQT favors

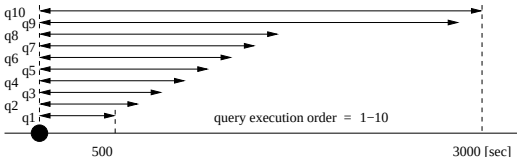

図 13　大 20%，小 80%でスループットに差が出ない分布
Fig. 13　Large 20% and small 80% that MQT doesn't favors

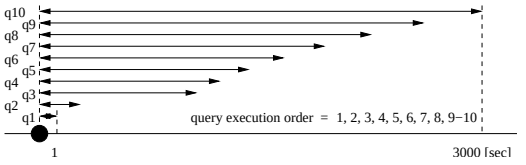

図 14　大 80%，小 20%でスループットに差が出る分布
Fig. 14　Large 80% and small 20% that MQT favors

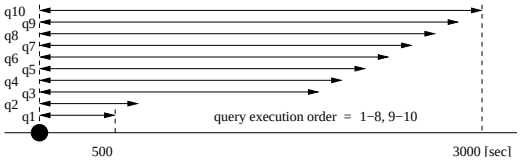

図 15　大 80%，小 20%でスループットに差が出ない分布
Fig. 15　Large 80% and small 20% that MQT doesn't favors

表 3　実験に用いた計算機環境
Table 3　Experiment environment

CPU	Intel PentiumII 400MHz
メモリ	256MB
OS	Red Hat Linux 9
開発言語	Java(J2SE 1.4.1)

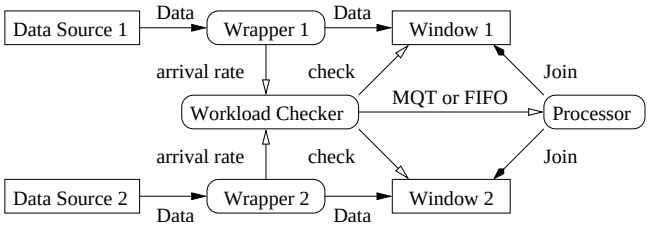

図 16　プロトタイプシステムの構成
Fig. 16　Prototype system architecture

図 9 は，MQT と FIFO のスループットに差が出ない分布を示している．問合せの実行待ちデータが複数あるときの問合せの実行順序を，最も古いデータに対する q_1〜q_{10}が 処理し終わったら次のデータ，そのデータに対する q_1〜q_{10}が 処理し終わったら次のデータ，という順にすると，MQT と FIFO のスループットに差が出ない．本研究では全てのウィンドウが異なるものとしているので，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大きくないときにこの分布となる．したがって，問合せ q_1 は各問合せのウィンドウ差よりも大きくない．図 9，図 11，図 13，図 15 では，大きいウィンドウを持つ問合せ群と小さいウィンドウを持つ問合せ群の境界を除いて，各問合せ間のウィンドウ差は一定になっている．

図 10〜図 15 は，図 8 と図 9 を元にし，大きいウィンドウを持つ問合せ群と，小さいウィンドウを持つ問合せ群に分け，その間には大きいウィンドウ差があるものとした．20:80，50:50，80:20 の比率で分け，それぞれ図 8 と図 9 の組合せのように，MQT と FIFO のスループットに差が出る分布と，差が出ない分布の組合せとなっている．

5. 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．

5.1 実 験 環 境

本システムの実験に用いた計算機環境は表 3 の通りである．図 16 が示すように，本システムはプロセッサ，ラッパー，ワークロードチェッカで構成される．ラッパーはデータソースから到着するデータを受取り，バッファにデータを入れる．同時にデータの到着率を測定し，ワークロードチェッカに通知する．ワークロードチェッカは到着率の通知を受け，未処理である問合せの重みの和を求める．これと閾値を比較して負荷が高いか低いかを判断し，問合せのスケジューリングに MQT を用いるか，FIFO を用いるかをプロセッサに通知する．プロセッサは，ワークロードチェッカの通知に従ったスケジューリングの手法を用いて問合せを処理する．

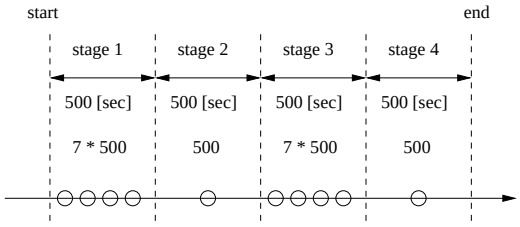

図 17　ステージごとの到着データ数
Fig. 17　Number of arrival data in each stage

1 つのデータには問合せが 10 個実行される．問合せは図 2 の形式で人工的に生成した．SELECT 節と FROM 節，WHERE 節は全て同一とし，WINDOW 節は 1 秒から 3000 秒とする．全てのウィンドウは異なるものとした．システムに到着するデータは図 1 の形式で人工的に生成した．time にはデータの生成時刻が入り，Location ID には整数 1 が入っている．これにより，WINDOW 節で指定された時間の範囲に含まれる，情報源 A のデータと情報源 B のデータが全て結合される．

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，バッファを 1 つずつ用意した．各バッファの大きさはデータ 5000 個分とした．また，実験を開始するときのバッファの初期状態は，古い 80%のデータに対する問合せは全て処理済みとし，新しい 20%のデータに対する問合せは全て未処理とした．

データは図 17 のように到着する．各ステージを 500 秒とし，ステージ 2 とステージ 4 に 500 個，ステージ 1 とステージ 3 に 7 倍の 3500 個が到着する．

データの到着率を 0.5 秒ごとに測定し，このタイミングで問合せの動的スケジューリングを行う．また，1 データに対する問合せの重みの和のうち，どれだけの問合せが未処理であれば負荷が高いと判断するかの割合 β は 0.05 とした．

5.2 実 験 結 果

問合せのウィンドウがランダムに分布するときと，それを構成する 8 種類の分布について実験を行った．8 種類の分布の実験結果については，特に両極端な結果を示す図 8 と図 9 について詳しく述べる．

5.2.1　ウィンドウがランダムに分布

図 18 と図 19 は，ウィンドウがランダムに分布するときの実験結果を示している．MQT，MandF（提案手法），FIFO を用いて問合せのスケジューリングを行い，それぞれ 5 回実験したときの平均を示した．x 軸は，図 17 で示したステージを表している．ステージ 1 とステージ 3 では，ステージ 2 とステージ 4 よりも新しいデータが多く到着する状況である．y 軸は，ステージ 1 からステージ 4 を通して到着したデータに対する問合せのうち，処理された問合せの割合と，実行できない問合せの割合を表している．ここで注意すべきことは，各ステージごとの結果ではないことである．例えばステージ 4 の結果は，ステージ 1 からステージ 3 で到着したデータに対する結果も含んでいる．

図 18 と図 19 より，実行できない問合せの割合は MQT のほ

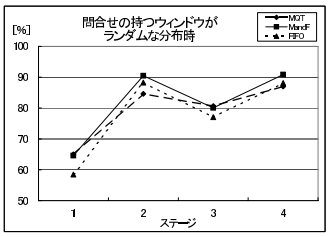

図 18　処理された問合せ
Fig. 18　Processed queries rate

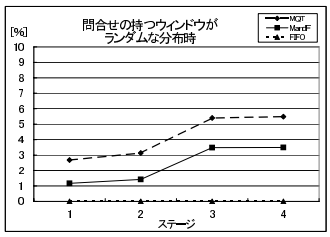

図 19　実行できない問合せ
Fig. 19　Dropped queries rate

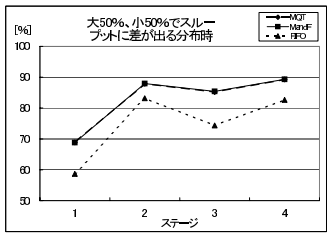

図 24　処理された問合せ（P3）
Fig. 24　Processed rate（P3）

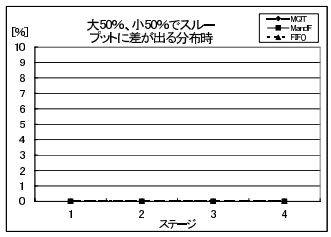

図 25　実行できない問合せ（P3）
Fig. 25　Dropped rate（P3）

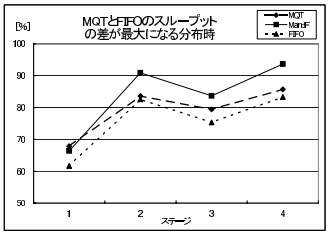

図 20　処理された問合せ（P1）
Fig. 20　Processed rate（P1）

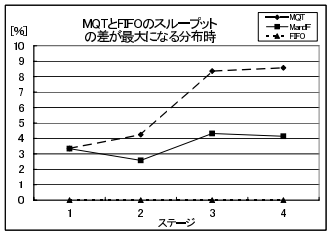

図 21　実行できない問合せ（P1）
Fig. 21　Dropped rate（P1）

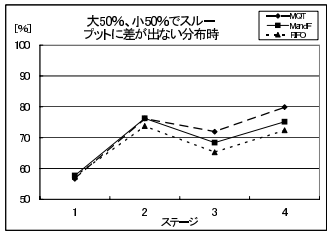

図 26　処理された問合せ（P4）
Fig. 26　Processed rate（P4）

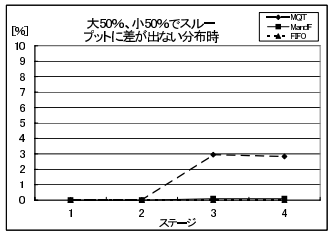

図 27　実行できない問合せ（P4）
Fig. 27　Dropped rate（P4）

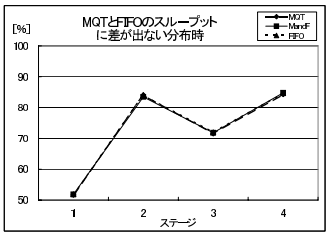

図 22　処理された問合せ（P2）
Fig. 22　Processed rate（P2）

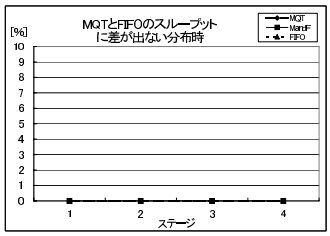

図 23　実行できない問合せ（P2）
Fig. 23　Dropped rate（P2）

ほ半分となり，処理された問合せの割合は大きいことが判る．処理負荷が高いときに FIFO を用い，処理負荷が低いときに MQT を用いることで，双方の良さが反映した結果となった．図 18 の処理された問合せの割合と，図 19 の実行できない問合せの割合を足して 100%にならないのは，バッファに残っているデータに対する未処理の問合せがあるためである．

5.2.2　両極端な結果を示す分布のとき

8 種類のウィンドウの分布うち，図 8 と図 9 が両極端な結果を示す．ここではその 2 つの結果について詳しく述べる．グラフの表題における P1〜P8 は，ウィンドウの分布がそれぞれ図 8〜図 15 であることを示している．

図 20 と図 21 において，まず提案手法を従来手法と比較する．ステージ 1 で処理された問合せの割合は，MQT よりわずかに劣っている．これは，実行できない問合せを減らすために，MQT に比べて長いウィンドウを持つ問合せを処理したためである．ステージ 2 では MQT に比べ，提案手法の方が問合せを多く処理している．これは，ステージ 1 で短い問合せを処理していない分と，実行できない問合せが少ない分を処理しているためである．ステージ 3 で処理された問合せの割合が MQT に近づいているのは，実行できない問合せを減らすために，FIFO と同じスケジューリングを行っているためである．そのため，ステージ 2 からステージ 3 の傾きは FIFO と同じになっている．ステージ 4 ではステージ 2 と同じ理由で，提案手法が最も問合せを処理している．

次に，MQT と FIFO を比較する．以降では様々なウィンドウの分布で実験を行っているが，部分的にはこれと同じ分布をしているので，この分布における MQT と FIFO の結果の違いを理解することは非常に重要である．処理負荷が高いステージ 1 とステージ 3 に比べ，処理負荷が低いステージ 2 とステージ 4 では，処理された問合せの割合の差は小さくなっている．ステージ 1 とステージ 3 では新しいデータが次々と到着するので，MQT は小さいウィンドウを持つ問合せを中心に実行する．しかしステージ 2 とステージ 4 では新しいデータは少なく，またステージ 1 とステージ 3 で既に小さいウィンドウを持つ問合せを実行してしまったので，ステージ 2 とステージ 4 では大きいウィンドウを持つ問合せを中心に実行する．したがって処理された問合せの割合は小さくなる．ここで，実行できない問合せの割合を見ると，MQT では実行できない問合せがあることがわかる．このため，ステージ 2 とステージ 4 の時間が長くなると，FIFO の方が処理された問合せの割合が大きくなると考えられる．

図 22 と図 23 から，3 つの手法が同じパフォーマンスであることが確認できる．ウィンドウの分布によっては問合せの実行順序が同じになるため，このような結果となる．

5.2.3　その他の分布

図 24〜図 35 を通して，提案手法は MQT と FIFO の双方の利点を活かした結果となっている．MQT と提案手法の両方で実行されない問合せがある場合，提案手法の方が処理した問合せの割合が大きかった．また，提案手法で問合せが全て処理された場合は，提案手法の方が FIFO よりも多くの問合せを処理した．MQT で全て問合せが処理された場合，提案手法は MQT と同じパフォーマンスを示している．

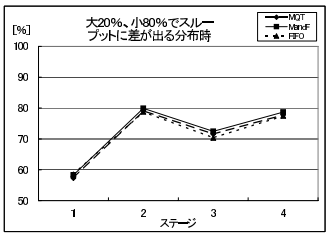

図 28　処理された問合せ（P5）
Fig. 28　Processed rate（P5）

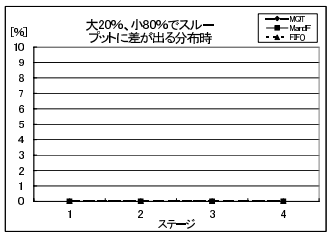

図 29　実行できない問合せ（P5）
Fig. 29　Dropped rate（P5）

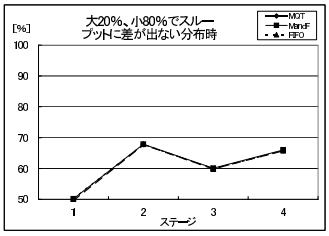

図 30　処理された問合せ（P6）
Fig. 30　Processed rate（P6）

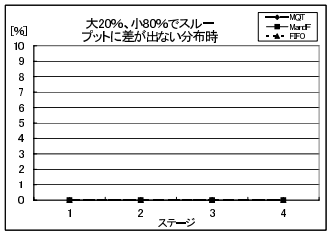

図 31　実行できない問合せ（P6）
Fig. 31　Dropped rate（P6）

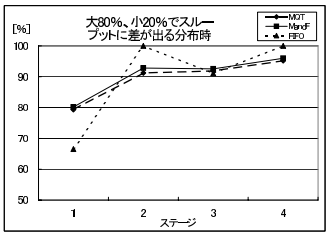

図 32　処理された問合せ（P7）
Fig. 32　Processed rate（P7）

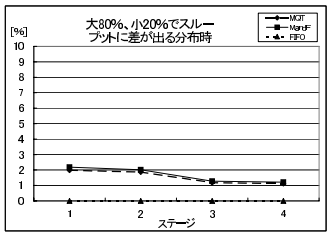

図 33　実行できない問合せ（P7）
Fig. 33　Dropped rate（P7）

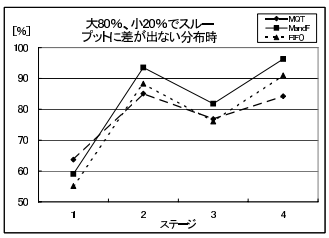

図 34　処理された問合せ（P8）
Fig. 34　Processed rate（P8）

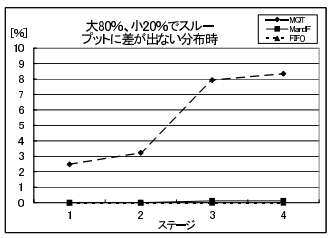

図 35　実行できない問合せ（P8）
Fig. 35　Dropped rate（P8）

6. おわりに

本稿では，共有ウィンドウ結合のスループットを最大にする MQT と，実行できない問合せの数を少なくする FIFO を処理負荷に基づいて使い分け，双方の利点が活きる手法を提案した．スループットが高く，かつ実行できない問合せの数が少ないため，9 種類の分布のうち 7 種類の分布で，他の 2 手法よりも多くの問合せを処理することができた．

今後の課題として，提案手法の理論的な分析が挙げられる．データの到着率，ウィンドウの大きさ，システムの処理能力などが分かるとき，どの手法が適しているかを求められれば，より良い動的スケジューリングが可能であると考えられる．

文　献

[1] B. Babcock, S. Babu, M. Datar, R. Motwani and J. Widom: “Models and issues in data stream systems”, Proceedings of the 21st ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pp. 1–16 (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, pp. 245–256 (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 processiong for an uncertain world”, Proceedings of the First Biennial Conference on Innovative Data Systems Research, pp. 269–280 (2003).

[4] S. Madden, M. Shah, J. M. Hellerstein and V. Raman: “Continuously adaptive continuous queries over streams”, Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, pp. 49–60 (2002).

[5] S. Chandrasekaran and M. J. Franklin: “Streaming queries over streaming data”, Proceedings of the 28th International Conference on Very Large Data Bases, pp. 203–214 (2002).

[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] S. D. Viglas, J. F. Naughton and J. Burger: “Maximizing the output rate of multi-way join queries over streaming information sources”, Proceedings of the 29th International Conference on Very Large Data Bases, pp. 285–296 (2003).

[8] 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).

[9] 渡辺, 北川：“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, **J87-D1**, 10, pp. 873–886 (2004).

[10] J. Kang, J. F. Naughton and S. D. Viglas: “Evaluating window joins over unbounded streams”, Proceedings of the 19th International Conference on Data Engineering, pp. 341–352 (2003).

[11] L. Ding and E. A. Rundensteiner: “Evaluating window joins over punctuated streams”, Proceedings of the 30th ACM Conference on Information and Knowledge Management, pp. 98–107 (2004).

[12] 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).

[13] A. Das, J. Gehrke and M. Riedewald: “Approximate join processing over data streams”, Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, pp. 40–51 (2003).

[14] T. Urhan and M. J. Franklin: “Xjoin : A reactively-scheduled pipelined join operator”, Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, pp. 27–33 (2000).

[15] L. Ding, E. A. Rundensteiner and G. T. Heineman: “Mjoin : A metadata-aware stream join operator”, Proceedings of the 2nd International Workshop on Distributed Event-Based Systems, pp. 1–8 (2003).