2005－DBS－137（Ⅱ）（49）
2005／7／14

負荷の傾向を考慮した 共有ウィンドウ結合の適応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　現在，連続的問合せを用いたデータストリームの処理が注目され，データストリームを効率良く処理するための問合せのスケジューリング手法が求められている．我々はこれまでに，共有ウィンドウ結合の動的なスケジューリング手法として，MandF を提案した．MandF は負荷に基づいて 2 つの手法を切替え，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ことで，処理できる問合せの数を多くする．負荷が高いか低いかの判定に閾値を用いるが，MandF での閾値は静的な値であった．本稿では，MandF を拡張し，負荷の傾向に応じて動的に閾値を調節する，adaptive MandF を提案する．状況に適したスケジューリング手法を用い，より多くの問合せを処理する．

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-Tendency-Based Adaptive 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　There is increasing interest in continuous query engines that process data stream. We have presented a scheduling method of shared window joins, called MandF, which chooses between two scheduling methods according to workload. In order to decide whether the workload is high or low, MandF compares the static threshold with measured workload. In this paper, we propose Adaptive MandF where the threshold is set dynamically. Adaptive MandF enables us to find which method is suited to the environment at a given time.

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なってきた．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情報源で，時間とともに新しい情報を提供する．例えば，各地に設置されたセンサーから処理システムに送られるセンサーデータは，各地の状況を刻々と伝えるデータストリームである．データストリームでは情報源から自発的なデータ送信が行われるため，処理システムでは次々と到着する大量のデータを扱わねばならない．したがって，データストリームに対する問合せの効率的な処理が求められる．

データストリームに対する問合せとして連続的問合せ [1] が注目されており，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．これらのシステムには問合せを効率良く処理するための仕組みが導入されており，例えば TelegraphCQ では，CACQ [4] や PSoup [5] といったモジュールを組込んでいる．これらのモジュールにより，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，問合せ間で処理結果を共有したりして問合せ処理の最適化を行っている．

連続的問合せの中で，結合処理に焦点を当てた研究がなされている [6]〜[8]．データストリームに対する結合処理を効率良く

行う方法の 1 つに共有ウィンドウ結合 [6] がある．これはウィンドウ結合を，処理結果を共有して行う方法である．共有ウィンドウ結合のスケジューリング手法として，問合せのスループットを最大にする MQT [6] や，データの到着順に問合せを実行する FIFO などがある．

我々はこれまでに，共有ウィンドウ結合の動的なスケジューリング手法として，MandF [9] を提案した．MandF では負荷に基づいて MQT と FIFO を切替え，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ことで，処理できる問合せの数を多くする．負荷が低いときには MQT を用いてスループットを上げ，負荷が高いときには FIFO を用いて実行できない問合せの数を少なくしている．

負荷が高いか低いかの判定は，初期値によって決まる閾値と，測定した負荷を比較して行っている．MandF での閾値は静的であり，閾値の動的な設定を行うことができなかった．本稿では，MandF を拡張し，負荷の傾向に応じて閾値を動的に調節する，adMandF(adaptive MandF) を提案する．これにより，より良いタイミングで MQT と FIFO を切替え，MandF よりも多くの問合せを処理させる．

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述べる．3. では，共有ウィンドウ結合と，そのスケジューリング手法を説明する．続く 4. で，我々が以前提案した MandF を説明し，5. で，今回提案する adMandF が用いる閾値の自動調節方法を説明する．6. では，以前実装した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. で，まとめと今後の課題とする．

2. 関 連 研 究

本研究は処理負荷に基づいた問合せのスケジューリングによって，効率良く結合処理を行っている．結合処理に関する他の研究では，情報源からの入力の特徴に基づいた手法や，出力される結果の量に基づいた手法が提案されている．例えば Mjoin [7] では，2 つ以上の情報源からの入力があるとき，結合を早く開始できる入力から処理するように問合せのスケジューリングを行う．また, [10] では，各問合せが 1 秒間にどれだけ入力を取ることができ，どれだけの出力タプルを生成するかが分かるとき，問合せを最適にスケジューリングする．何に基づいてスケジューリングすべきかは，データストリームの性質や処理システムの環境によって変わる．したがって，環境に適した手法を用いることが重要となる [8].

[11] では，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. [12] では, [11] で検討されなかった，クラスタ化の基準となる閾値の自動調節方法を提案している．データストリームの到着パターンは時間とともに変化しうるので，閾値の最適値も変化し続ける可能性がある．そこで，実行時の情報に基づき，最適と思われる値へ徐々に近づけるアプローチをとっている．本研究も同様に，閾値の自動調節によって既存手法の拡張を行った．実行時の情報に基づいて閾値を自動調節する点で，同様のアプローチをとっている．

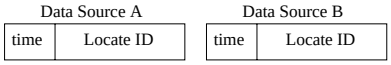

図 1　情報源 A，B から到着するデータ
Fig. 1　Data from data sources A and B

SELECT	*
FROM	Data Source A, Data Source B
WHERE	A.Locate ID = B.Locate ID
WINDOW	Time [sec]

図 2　問合せの例
Fig. 2　Example of queries

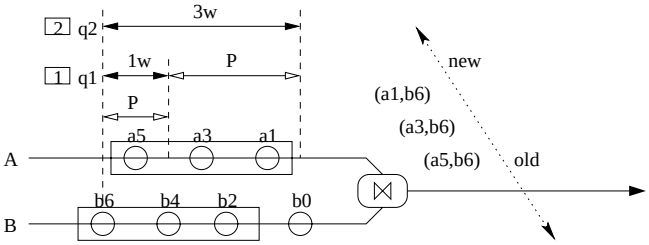

図 3　共有ウィンドウ結合による問合せの処理
Fig. 3　Processing shared window joins

3. 共有ウィンドウ結合のスケジューリング

連続的問合せは短時間に繰返し実行され，問合せ間で処理結果を共有できる場合が多い．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び，共有ウィンドウ結合によって効率良く結合処理が行える．

本研究では，情報源 A と B からそれぞれ図 1 のデータが到着し，それらのデータ 1 つ 1 つに対して図 2 の問合せが実行される．議論を簡単にするため，問合せの SELECT 節，FROM 節，WHERE 節は全て同一とした．

図 3 は，共有ウィンドウ結合による問合せの処理を示している．データを囲む四角は，メモリ上のバッファを示しており，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．1 つのデータに対して，2 つの問合せ q_1 と q_2 がウィンドウの小さい順に実行される．このとき，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，P の部分だけを処理する．

共有ウィンドウ結合のスケジューリング手法として，MQT [6] と FIFO [9] がある．MQT は共有ウィンドウ結合のスループットを最大にする．早く処理が終わる問合せから実行するように問合せのスケジューリングを行うため，新しいデータの到着のために実行されない問合せが生じる可能性がある．MQT では全てのデータに全ての問合せが実行できる負荷を前提としており，負荷が高いときは，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やすいという問題がある．

一方，FIFO は実行できない問合せが少なくする．データの到着順に問合せを実行するため，最も古いデータから処理が終わる．この為，負荷が低いときでもスループットを上げることはできない．

4. 負荷に基づいた動的スケジューリング

本節では，我々が提案した MandF [9] を説明する．

4.1 基本的アイデア

ストリームデータは時間とともに次々とシステムに到着し，システム内のバッファにおいて処理されるのを待つ．データによってバッファが満たされると，新しいデータの到着の度に古いデータがバッファから消える．問合せの実行前にバッファからデータが消えることは，問合せの対象が消えることを意味し，問合せの結果が生成されないということである．したがって，問合せの対象であるデータがバッファから消える前に，そのデータを処理する必要がある．

また，データストリームでは，データの到着パターンが時間とともに変化しうるため，処理システムにかかる負荷が変わる可能性がある．例えば，データの到着間隔が短ければ，古いデータがバッファから消える間隔も短くなるので，古いデータに対する問合せを優先して実行する方が良い可能性がある．逆に，データの到着間隔が長ければ，新しいデータに対する問合せのうち，早く処理が終わる問合せを先に実行する方が良い可能性がある．したがって，状況に合せた問合せのスケジューリングを行うべきである．

MandF は負荷に基づいて MQT と FIFO を切替え，双方の利点が活きるように問合せのスケジューリングを行う．これにより，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ことで，処理できる問合せの数を多くする．

4.2 切替アルゴリズム

MQT と FIFO の切替えの判断は，初期値によって決定される閾値 α と，単位時間ごとに測定される負荷 S の比較によって行われる．S が α 以上なら FIFO を用いて問合せのスケジューリングを行い，S が α 未満なら MQT を用いて問合せのスケジューリングを行う．

まず，負荷 S の測定方法を説明する．スループットが上がる問合せから実行できるように，各問合せには重みが付いている [9]．スループットが上がる問合せに大きい値が付いており，値は MQT 行列 [6] から求めることが出来る．単位時間に到着するデータの数を測定し，その個数分の問合せのうち，未処理である問合せの重みの和を負荷 S とする．以降，単位時間に到着するデータの数を到着率と呼ぶ．

次に閾値 α の設定方法を説明する．閾値 α は，到着率と閾値パラメータ β によって決定される．β は，1 データに対する問合せの重みの和のうち，どれだけ未処理であれば負荷が高いと判断するかの割合を 0 から 1 の範囲で与える初期値である．このとき閾値 α は以下の式となる．

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

閾値 α が大きくなるほど MQT を用いたスケジューリングになりやすく，逆に小さくなるほど FIFO を用いたスケジューリングになりやすい．MandF では，β は静的な値であり，閾値 α は β に依存した値になる．

5. 提 案 手 法

本節では，MandF を拡張し，負荷の傾向に応じて閾値を動的に調節する，adMandF(adaptive MandF) を提案する．閾値の調節は β の調節によって行い，MandF よりも，より良いタイミングで MQT と FIFO を切替える．

5.1 閾値パラメータ β の重要性

β によって，スループットを重視するか，実行できない問合せの減少を重視するかが決まる．β が 1 に近いとき，0 に近いときにはそれぞれ利点，欠点があり，状況に適した β を設定することが重要である．

5.1.1 β が 1 に近いとき

MQT によるスケジューリングが行われやすい閾値になる．式より，β が大きくなると閾値 α も大きくなり，実行できない問合せが生じても MQT により問合せのスケジューリングを行う．負荷が低いときは，実行できない問合せを生じることなくスループットが上がる利点がある．逆に，負荷が高いときは，実行できない問合せが生じやすいという欠点がある．負荷の傾向が高いときは，より FIFO に切替わりやすい β に設定する方が良いと考えられる．

5.1.2 β が 0 に近いとき

FIFO によるスケジューリングが行われやすい閾値になる．式より，β が小さくなると閾値 α も小さくなり，実行できない問合せが少なくなるように，FIFO により問合せのスケジューリングを行う．負荷が高いときは，実行できない問合せが少なくなる利点がある．逆に，負荷が低いときは，問合せの対象となるデータがバッファから消えるまでに余裕があるにも関わらず，その間に他の問合せを実行することができない欠点がある．負荷の傾向が低いときは，より MQT に切替わりやすい β に設定する方が良いと考えられる．

5.2 β 調節に用いる情報

β 調節のために，負荷の傾向と到着率の傾向を用いる．なぜなら，負荷が高くても，到着率が高いときと低いときでは設定すべき β の値は変化するためである．負荷の傾向として β_{next}，到着率の傾向として 到着率$_{next}$ を推定し，次に設定すべき β である β_{update} を決定する．

5.3 β の調節アルゴリズム

まず，負荷の傾向として β_{next} を推定する．前回の β である β_{prev} と，現時点で取るべきであった β である β_{now} を用いる．β_{now} は $S = \alpha$ となる β であり，次の式で求める．

$$\beta_{now} = \frac{S}{1\text{ データに対する問合せの重みの和} \cdot \text{到着率}}$$

右辺の分母に負荷 S があることから分かるように，負荷が 0 のときは β_{now} は 0 になる．つまり，β_{now} が 0 のとき，その時点で用いているスケジューリングの手法で，実行できない問合せが生じていないことを表している．

次に，次の式を用いて β_{next} を推定する．このとき，β_{next} は 0 から 1 の範囲を超えないようにする．

$$\beta_{next} = \beta_{prev} + \frac{(\beta_{now} - \beta_{prev})}{\text{到着率を求める時間間隔}}$$

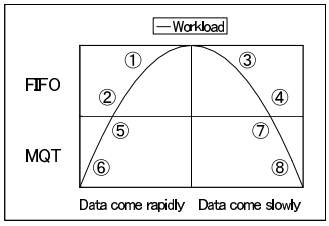

図 4　β_{update} の決定方法
Fig. 4　The method of deciding β_{update}

続いて，到着率の傾向として 到着率$_{next}$ を推定する．前回の 到着率 である 到着率$_{prev}$ と，現時点の到着率である 到着率$_{now}$ を用いる．次の式を用いて 到着率$_{next}$ を推定する．

$$\text{到着率}_{next} = \text{到着率}_{prev} + \frac{(\text{到着率}_{now} - \text{到着率}_{prev})}{\text{到着率を求める時間間隔}}$$

また，MQT に切替わったときの到着率を 到着率$_{MQT}$，FIFO に切替わったときの到着率を 到着率$_{FIFO}$ とする．到着率$_{next}$ が $\frac{(\text{到着率}_{MQT}+\text{到着率}_{FIFO})}{2}$ 以上のときは到着率が高いと判断し，それに満たないときは到着率が低いと判断する．

求めた β_{now}，β_{next}，到着率$_{next}$ の関係から，次に設定すべき β_{update} を決定する．図 4 は，β_{update} を決定するための場合分けを示している．図は 4 等分されており，左側が到着率が低いとき，右側が到着率が高いときである．図中の関数は負荷 S を示しており，負荷が高いと FIFO が実行されやすく，負荷が低いと MQT が実行されやすいことを表している．β_{update} を決定するときに用いていたスケジューリング手法によって上下が決まり，まず 4 個所のうち 1 個所に場合分けされる．

続いて，β_{now} と β_{next} の関係から，図中の番号を決定する．次の番号は，図中の番号と一致しており，決定された番号の方法に従って β_{update} を決定する．

（1） $\beta_{next} > \beta_{now}$(負荷：大 → 大)
　　必ず FIFO で処理するため，$\beta_{update} = \beta_{now}$
（2） $\beta_{now} == 0$(負荷：FIFO で未処理の問合せ無し)
　　MQT では対応できないので，$\beta_{update} = \beta_{now}$
（3） $\beta_{next} < \beta_{now}$(負荷：大 → 小)
　　MQT に切替えやすくするため，$\beta_{update} = \beta_{now}$
（4） $\beta_{now} == 0$(負荷：FIFO で未処理の問合せ無し)
　　MQT に切替えて，$\beta_{update} = 0.01$
（5） $\beta_{next} > \beta_{now}$(負荷：小 → 大)
　　FIFO に切替えやすくするため，$\beta_{update} = \beta_{now}$
（6） $\beta_{now} == 0$(負荷：MQT で未処理の問合せ無し)
　　MQT で対応できるので，$\beta_{update} = 0.02$
（7） $\beta_{next} < \beta_{now}$(負荷：小 → 小)
　　MQT で十分対応できるので，$\beta_{update} = \beta_{now}$
（8） $\beta_{now} == 0$(負荷：MQT で未処理の問合せ無し)
　　必ず MQT で処理するため，MQT$\beta_{update} = 0.03$

MQT，FIFO のどちらを用いても，実行できない問合せが生じないときは β_{now} が 0 になる．したがって，FIFO から MQT への切替えは，(4) のように明示的に行う必要がある．また，MQT を用いやすくするには，(6)，(8) のように β_{update} を設定する必要がある．

表 1　実験に用いた計算機環境
Table 1　Experiment environment

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

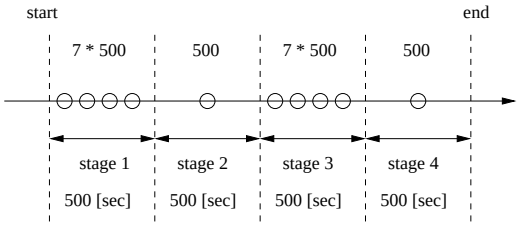

図 5　ステージごとの到着データ数
Fig. 5　Number of arrival data in each stage

6. 評価実験

提案手法の有効性を示すため, [9] で実装したプロトタイプシステムを用いて，評価実験を行った．

6.1 実験環境

実験に用いた計算機環境は表 1 の通りである．1 つのデータに対して 10 個の問合せが実行され，各データは図 1 の形式で，各問合せは図 2 の形式で人工的に生成した．

到着するデータを保存するバッファは情報源 A 用と情報源 B 用の 2 つがあり，それぞれの大きさはデータ 5000 個分である．各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 5 のように到着する．データの到着率は 0.5 秒ごとに測定され，このタイミングでスケジューリング手法の切替えの判定と，閾値パラメータ β の調整がされる．β の初期値は 0.05 とした．

6.2 実験に用いた問合せ

図 6 は，実験に用いる問合せのウィンドウの分布を示している．$min = 1$ かつ $max = 3000$ でランダムに発生した場合の他に，$max = 3000$ としたときに以下の 3 つの条件で 8 通りの分布を生成する．

- $b < \cdots < M < \cdots < c$ で $e = 50$ のとき，

$$min = 1 < a < b < \cdots < M < \cdots < c < d \qquad (1)$$

- $b \geqq \cdots \geqq M \geqq \cdots \geqq c$ で $e = 50$ のとき，

$$min = 500 \geqq a \geqq b \geqq \cdots \geqq M \geqq \cdots \geqq c \geqq d \quad (2)$$

- $d < M < max$ のとき，

$e = 20, 50, 80$ で式 (1) と式 (2)

生成された分布は次の通りである．問合せの実行順序は，例えば 1-8,9-10 なら，問合せ 1 から 8 を処理したら次のデータに対する問合せ 1 から 8 を処理し，全てのデータに対して処理が終わったら，問合せ 9 から 10 を処理することを示している．

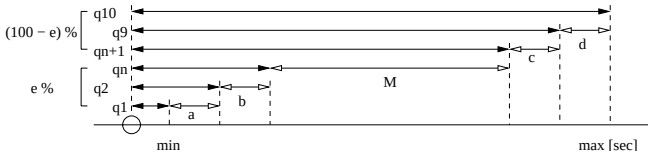

図 6　実験に用いる問合せのウィンドウ分布
Fig. 6　Windows distribution

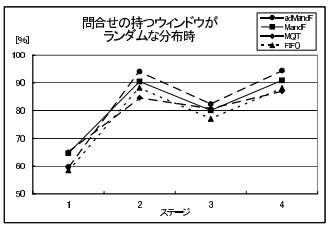

図 7　処理された問合せ
Fig. 7　Processed queries rate

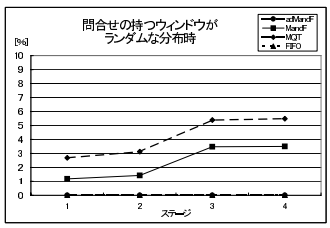

図 8　実行できない問合せ
Fig. 8　Dropped queries rate

（1） MQT と FIFO のスループットの差が最大になる分布
問合せの実行順序：1,2,3,4,5,6,7,8,9,10
（2） MQT と FIFO のスループットに差が出ない分布
問合せの実行順序：1-10
（3） 大 50%，小 50%でスループットに差が出る分布
問合せの実行順序：1,2,3,4,5,6-10
（4） 大 50%，小 50%でスループットに差が出ない分布
問合せの実行順序：1-5,6-10
（5） 大 20%，小 80%でスループットに差が出る分布
問合せの実行順序：1,2,3-10
（6） 大 20%，小 80%でスループットに差が出ない分布
問合せの実行順序：1-10
（7） 大 80%，小 20%でスループットに差が出る分布
問合せの実行順序：1,2,3,4,5,6,7,8,9-10
（8） 大 80%，小 20%でスループットに差が出ない分布
問合せの実行順序：1-8,9-10

6.3 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 通りの分布について実験を行った．

6.3.1 ウィンドウがランダムに分布

x 軸は，図 5 で示したステージを表している．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．注意すべきことは，各ステージごとの結果ではなく，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ことである．

図 7 と図 8 より，実行できない問合せがなくなり，その結果処理された問合せの割合が大きくなった．ステージ 1 でスループットが上がらなかった分，実行できない問合せを生じなかったことが，MandF に勝った要因と考えられる．図 7 の処理された問合せの割合と，図 8 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．

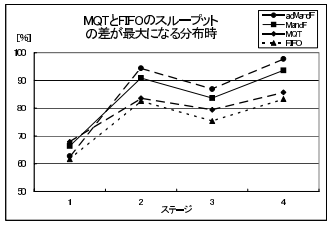

図 9　処理された問合せ（P1）
Fig. 9　Processed rate（P1）

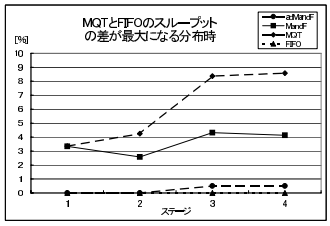

図 10　実行できない問合せ（P1）
Fig. 10　Dropped rate（P1）

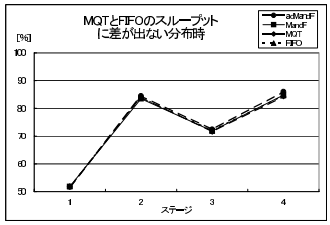

図 11　処理された問合せ（P2）
Fig. 11　Processed rate（P2）

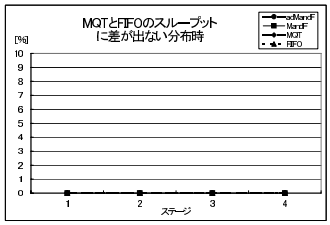

図 12　実行できない問合せ（P2）
Fig. 12　Dropped rate（P2）

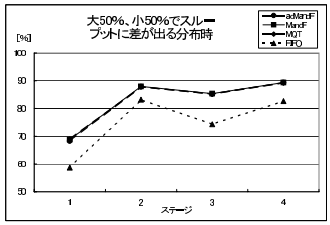

図 13　処理された問合せ（P3）
Fig. 13　Processed rate（P3）

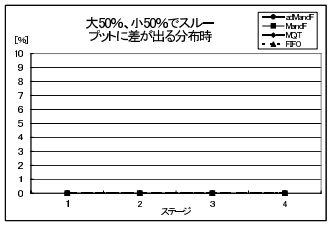

図 14　実行できない問合せ（P3）
Fig. 14　Dropped rate（P3）

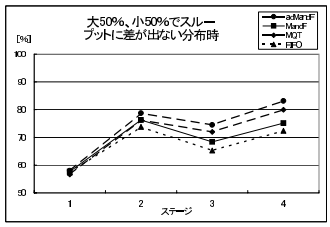

図 15　処理された問合せ（P4）
Fig. 15　Processed rate（P4）

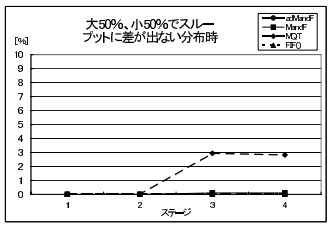

図 16　実行できない問合せ（P4）
Fig. 16　Dropped rate（P4）

6.3.2 生成した 8 通りの分布

グラフの表題における P1〜P8 は，6.2 で生成された 8 通りの分布の番号と一致している．図 9〜図 24 を通して，提案手法は他の 3 手法よりも多くの問合せを処理した．また，P1〜P8 の全ての場合において，提案手法は実行できない問合せを生じなかった．

P1，P4，P7，P8 のように MQT で実行できない問合せが生じるときは，提案手法は他の 3 手法よりも多くの問合せを処理した．P4，P7 では，MandF は MQT，あるいは FIFO よりも処理された問合せの割合は小さかったが，提案手法では割合が大きくなった．

P4 では，MQT が短いウィンドウを持つ問合せを中心に処理するのに対し，MandF は FIFO を多く用い，長いウィンドウを持つ問合せも多く処理していた．しかし，提案手法は MQT

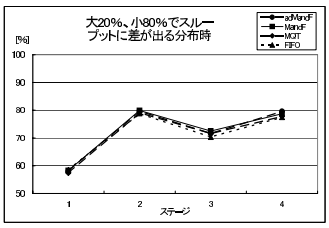

図 17　処理された問合せ（P5）
Fig. 17　Processed rate（P5）

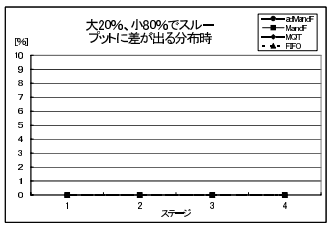

図 18　実行できない問合せ（P5）
Fig. 18　Dropped rate（P5）

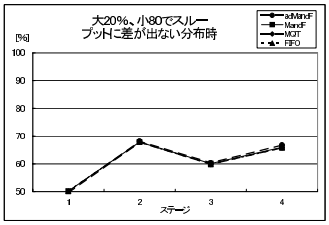

図 19　処理された問合せ（P6）
Fig. 19　Processed rate（P6）

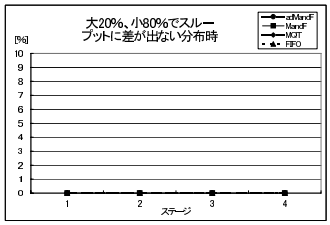

図 20　実行できない問合せ（P6）
Fig. 20　Dropped rate（P6）

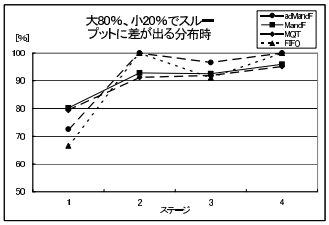

図 21　処理された問合せ（P7）
Fig. 21　Processed rate（P7）

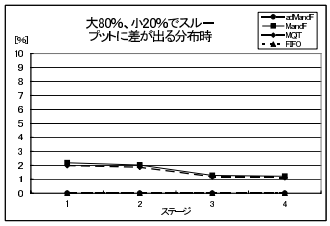

図 22　実行できない問合せ（P7）
Fig. 22　Dropped rate（P7）

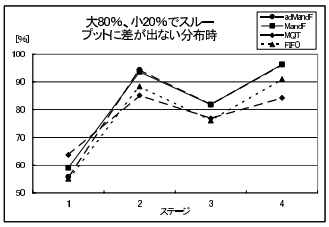

図 23　処理された問合せ（P8）
Fig. 23　Processed rate（P8）

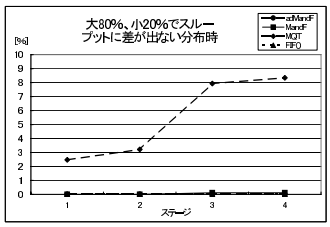

図 24　実行できない問合せ（P8）
Fig. 24　Dropped rate（P8）

を多く用い，加えて実行できない問合せがなかったため，MQT よりも多く問合せを処理したと考えられる．

P7 では，提案手法は実行できない問合せを生じなかったため，MandF や MQT よりも多くの問合せを処理できた．また，提案手法は MQT を用いる分だけ FIFO よりもスループットが高くなるため，FIFO よりも多くの問合せを処理したと考えられる．

P2，P3，P5，P6 のように MQT で実行できない問合せが生じないときは，提案手法は MQT と同じスループットで問合せを処理し，処理された問合せの割合が同じになった．問合せの実行順序が FIFO と同じになる P2，P6 では，処理された問合せの割合は FIFO と同じになったが，問合せの実行順序が FIFO と異なる P3，P5 では，FIFO よりも多くの問合せを処理した．

7. おわりに

本稿では，我々が以前提案した MandF を拡張し，より良いタイミングで MQT と FIFO を切替えるための，閾値の自動調節方法を導入した．今回提案した adMandF は，負荷の傾向とデータの到着率によって次に取るべき閾値を動的に決定し，その状況に適した閾値に設定することを可能にした．評価実験により，従来手法よりも多くの問合せを処理できることを示した．

今後の課題として，FIFO で実行できない問合せが生じる程の負荷で，MQT の方が実行できない問合せの数が少なくなる場合を考慮することが挙げられる．負荷が高いときの FIFO では，処理に時間のかかる 1 つの問合せを処理する間に，多くの問合せが実行できなくなる可能性があるためである．

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] N. Koudas and D. Srivastava: “Data stream query processing”, Advanced Technology Seminar 3 at the 21st International Conference on Data Engineering, Available at http://icde2005.is.tsukuba.ac.jp/ (2005).

[9] 多田, 有次：“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”, 電子情報通信学会第 16 回データ工学ワークショップ（DEWS 2005）, pp. 4C–o3 (2005).

[10] 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).

[11] 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).

[12] 渡辺, 北川：“セルフチューニングによる連続的問合せの適応型問合せ最適化”, 電子情報通信学会第 16 回データ工学ワークショップ（DEWS 2005）, pp. 4C–o1 (2005).