到着率に動的に適応するストリームデータ処理スケジューリングの研究

18500073

平成18年度～平成19年度科学研究費補助金
（基盤研究（C））研究成果報告書

平成20年5月

研究代表者　有次正義
熊本大学大学院自然科学研究科教授

＜はしがき＞

平成 18 年度～平成 19 年度科学研究費補助金 (基盤研究（C)) 「到着率に動的に適応するストリームデータ処理スケジューリングの研究」（課題番号 18500073）にて行った研究成果を報告する．本研究課題では主に，センサーデバイスからのデータや株価変動のデータ，ネットワークトラフィックデータなどの，不定期に大量に発生するストリームデータに対する処理の効率化のためのスケジューリングアルゴリズムの開発と，ストリームデータの応用分野を広く調査・検討し，今後ますます重要となってくるであろうストリームデータ処理に関する研究の新しい方向性について検討した．

スケジューリングアルゴリズムの開発については，ストリームデータに対する共有ウィンドウ結合処理に焦点を絞った．これは，一般に結合処理の負荷が大きいためである．従来研究では，ストリームデータの，特に到着率がとても高い場合に有効なスケジューリングアルゴリズムはなかった．本研究では，放送サービスのためのスケジュール手法を調査し，それを基に，問合せ処理の成功率とスループットの両方のバランスを保ちながら，効率的に処理することが可能な動的スケジューリング手法を開発した．

ストリームデータ処理に関する研究の新しい方向性については，セキュリティやプライバシー，P2P システム，SMT プロセッサ処理，およびウェブキャッシングシステムについて検討した．プライバシーについては，多値属性分類を考えたときのプライバシーを保護することが可能なカウント演算について検討した．セキュリティについては，細粒度のデータベースアクセス制御のための計算手法について検討し，ルールベースの不整合チェック機構を設計し，実装手法を提案した．P2P システムについては，複製による負荷分散を可能にする P2P プロトコルを考案した．SMT プロセッサ処理については，プログラムフェーズを考慮したスレッドスケジューリング手法について検討した．ウェブキャッシングシステムについては，従来にはないグループの概念を導入し，これを使った P2P ネットワークにおける効率的なデータの配置について検討し，シミュレーションを使った予備的な実験結果を得た．

研究組織

研究代表者：有次正義（熊本大学大学院自然科学研究科教授）

交付決定額 (配分額)

(金額単位：円)

	直接経費	間接経費	合計
平成 18 年度	2,100,000	0	2,100,000
平成 19 年度	1,500,000	450,000	1,950,000
総計	3,600,000	450,000	4,050,000

研究発表

1. 雑誌論文

   (a) 佐治和弘，有次正義：複製による負荷分散を可能にした P2P プロトコルの提案，日本データベース学会 Letters, 5(2), pp.9–12, 2006.

   (b) 高見澤秀久，有次正義：プライバシーを保護するカウント演算の多値属性分類への適用について，日本データベース学会 Letters, 6(1), pp.33–36, 2007.

   (c) 多田直剛，有次正義：ExT: データ到着率の変化に適応する共有ウィンドウ結合の動的スケジューリングアルゴリズム，電子情報通信学会論文誌 D，J90-D(7), pp.1744-1754, 2007.

2. 学会発表

   (a) 佐治和弘，有次正義：複製による負荷分散を可能にした P2P プロトコルの評価，電子情報通信学会第 18 回データ工学ワークショップ (DEWS2007), 2007.

   (b) 高見澤秀久，有次正義：プライバシーを保護するカウント演算の多値属性分類への適用, 電子情報通信学会第 18 回データ工学ワークショップ (DEWS2007), 2007.

(c) 市川 雄二郎, 有次 正義：プログラムフェーズを考慮したSMTプロセッサ上でのスレッドスケジューリング手法, 電子情報通信学会第18回データ工学ワークショップ (DEWS2007), 2007.

(d) B. Purevjii, M. Aritsugi, S. Imai and Y. Kanamori: An Implementation Design of a Fine-Grained Database Access Control, 11th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems, Springer Lecture Notes in Artificial Intelligence 4693, pp.752–760, 2007.

(e) H. Takamizawa and M. Aritsugi: Applying Privacy Preserving Count Aggregate Queries to k-Classification, 11th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems, Springer Lecture Notes in Artificial Intelligence 4693, pp.761–768, 2007.

(f) 岩丸晃大，糸川剛，北須賀輝明，有次正義：グループの概念を取り入れたP2Pウェブキャッシングシステムに向けた考察，情報処理学会九州支部火の国情報シンポジウム2008，2008.

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

概要

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，スループットや問合せの成功率が高く，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．本稿では，データストリームに対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， MandF， adaptive MandF，および ExT を提案する．従来のスケジューリング手法とは異なり，実行されない問合せを考慮し，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．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．各手法について述べ，従来手法との比較実験により提案手法の有効性を示す．

第 1 章

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．また，これらの性能を比較するためのベンチマーク [4] も提案され，新たなデータ処理方法として注目を浴びている．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．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るため，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，処理システムには短期間に大量のデータが到着する可能性があり，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 [5]．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対象とする．

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かった特徴がある．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， MandF[7]， adaptive MandF[8]，および ExT[9] を提案する．従来のスケジューリング手法とは異なり，実行されない問合せを考慮し，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．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． MandF では，従来の 2 手法を切替えてスケジューリングを行っている． adaptive MandF では， MandF で切替えのために用いていた閾値の自動調節を行い，より状況の変化に適応したスケジューリングを行う． ExT では従来手法を用いずに，スループットと実行されない問合せを考慮した値を用いてスケジューリングを行う． 3 手法のうち， ExT が最も良い性能を示すスケジューリング手法である．

本稿の構成は次の通りである．まず 2章で，本研究の関連研究について述べる． 3章では，準備として研究の背景や共有ウィンドウ結合を説明し，従来のスケジューリング手法について述べる．続く第 4章で MandF，第 5章で adaptive MandF，第 6章で ExT を説明する．最後に第 7章で，まとめと今後の課題とする．

第 2 章

関連研究

2.1 データストリーム処理

本研究では，データストリームに対する共有ウィンドウ結合を対象としているが，他にも様々な処理手法が提案されている．データストリームの代表的な処理システムの 1 つとして， TelegraphCQ[3] が提案されている． TelegraphCQ では，処理内容によってモジュールを分け，その処理に対して最適化した手法を用いることができる．例えば CACQ[10] や PSoup[11] といったモジュールによって問合せのスケジューリングを行い，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することが可能になる．また， Flux[12] は scalability を考慮したモジュールで， shared nothing 環境における分散処理を可能としている．こういったモジュール間を Eddies[13] が繋ぐことで，処理内容に適したモジュールを選択することができる．

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．一方，実行されない問合せを生じないように，処理待ちのデータを二次記憶に保存して処理する処理方法が提案されている．また，各データアイテムのサイズが大きいデータストリームを対象とするシステムでは，二次記憶を用いた処理を考慮している． XJoin[14] は， non-blocking な結合を含むアルゴリズムである． Symmetric Hash Join を基にしており， 2 つの入力に対する処理を，並列に行うことで処理効率を高めている． MJoin[15] では， XJoin とは異なり，複数の入力を対象としたときのスループットを最大にする手法を提案している．どちらの手法も大きなメモリ空間を必要とする処理を考慮しており，二次記

憶を用いて，全ての処理が完了するアルゴリズムとなっている．また， Ding らが提案する MJoin[16] では XJoin を拡張し， *punctuation* を用いた処理方法を提案している．

PWJoin[17] は，データストリームの結合に *punctuation* を用いる． *punctuation* を用いた結合は value-based な結合であり，対象を時間で指定せず， *punctuation* がどのデータを対象とするかを明示的に指定する．一方，スライディングウィンドウ結合は time-based な結合であり，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．例えば，今後到着するデータの値が 10 以上であると *punctuation* が示すと， 10 未満の値を対象とする問合せは実行しなくても良いと分かる．本研究では，スライディングウィンドウを用いた結合を行う．

スライディングウィンドウを用いた結合方法に， Kang ら [18] や Golab ら [19] がある． Kang ら [18] は Symmetric Hash Join を拡張し， 2 つの入力に対するスライディングウィンドウ結合の手法を提案した． 2 つのストリームにおけるデータ到着率の差が大きいときも考慮しており，彼らの提案するコストモデルに基づいて，結合結果のタプル数が最大になるように処理する．また， Golab ら [19] は，複数の入力に対するスライディングウィンドウ結合の手法を提案している． Nested Loop 結合やハッシュ結合を用いたときの処理コストからコストモデルを作成し，処理コストが最小になるようにヒューリスティクスに問合せの実行順序を決める．

2.2 スループットと実行されない問合せの考慮

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のスケジューリングを行う．提案手法は，スループットと実行されない問合せ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においても良い性能を示す．

Viglas ら [20] は，データの到着率に基づいて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，スループットとともに，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．

[21, 22] は adaptive MandF と同様に，閾値の自動調節によって既存手法の拡張を行った． [21] 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． [22] では， [21] 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性質や到着パターンは時間とともに変化しうるので，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．

RxW[23] 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．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 [24, 25, 26]．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[24] 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る．したがって， Load Shedding との併用が可能であり，より多くのデータに対して問合せを実行することが可能になる．

第 3 章

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．データストリームはデータアイテムの無限の連鎖とみなすことができ，各データアイテムは，システムに入った時間を示すタイムスタンプと関連付けられている．データアイテムの到着の仕方は 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

LocationID	Value	Timestamp

図 3.1: 情報源から到着するデータアイテム

```
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 3.2: 問合せの例：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 3.3: 問合せの例：Q2

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 3.1〜 3.4は，文献 [6] と同じものである．

図 3.1は，モデルで用いる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．

図 3.2と図 3.3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 3.2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 3.3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．

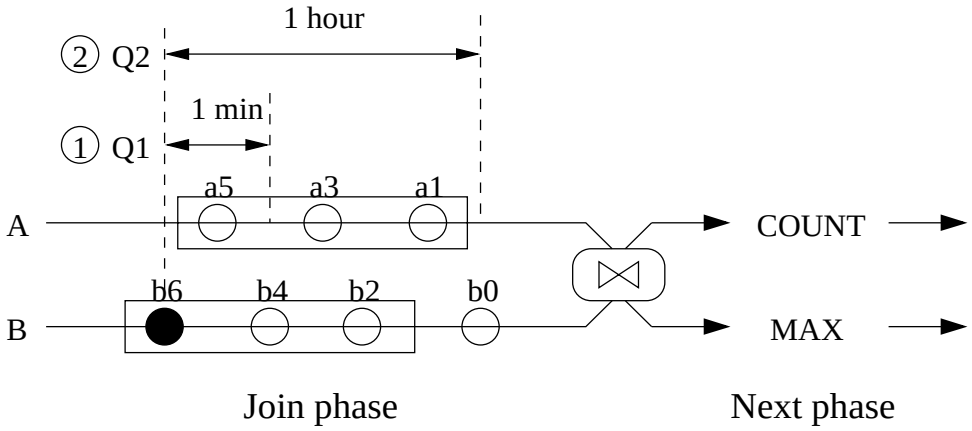

図 3.4: 共有ウィンドウ結合

3.2 共有ウィンドウ結合

[27] によると，データストリームは次の特性を持っている．

- rapid：データの早い到着
- unpredictable：データの予測つかない到着
- time-varying：時間とともに性質が変化

連続的問合せは上記のようなデータの到着ごとに実行されるため，短時間に繰返し実行される可能性が高い．同じデータストリームを対象とする問合せでは，問合せ間で処理結果を共有することで効率良く問合せを処理することができる．

図 3.4は，図 3.2と図 3.3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 3.2と図 3.3で示した問合せであり，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 3.1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．

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 3.4ではデータがシステムに到着すると，それに対して $Q1$ が先に実行され，続いて $Q2$ が実行される．図 3.4から，$Q2$ の処理結果は $Q1$ の処理結果を含んでいることがわかる．共有ウィンドウ結合における問合せの処理結果は，より大きいウィンドウを持つ問合せに含まれる．

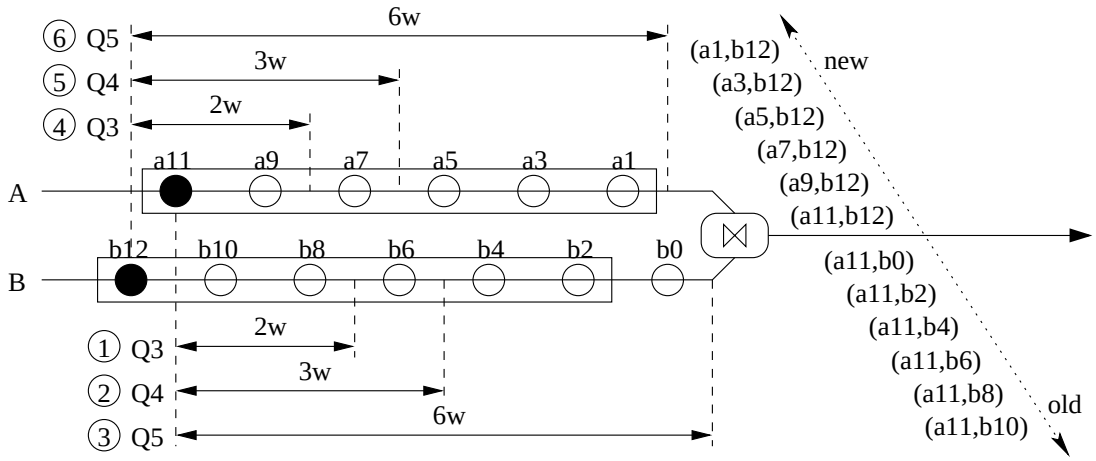

図 3.5: FCFS を用いた共有ウィンドウ結合

3.3 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 3.5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．

3.4 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．早く処理結果が返せる問合せを優先して処理することで，短期的なスループットを最大にしている．特に，短時間に大量のデータが到着する場合，よりスループットが高くなることが実験で示されている．ここでは，処理に必要なメモリが十分あることを前提としており，二次記憶を用いなくても，問合せが実行される前に対象のデータがメモリから消えることがないことを保証している．

問合せの実行優先度は，問合せの追加と削除があるときに求められる．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，問合せ間のウィンドウの差，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．

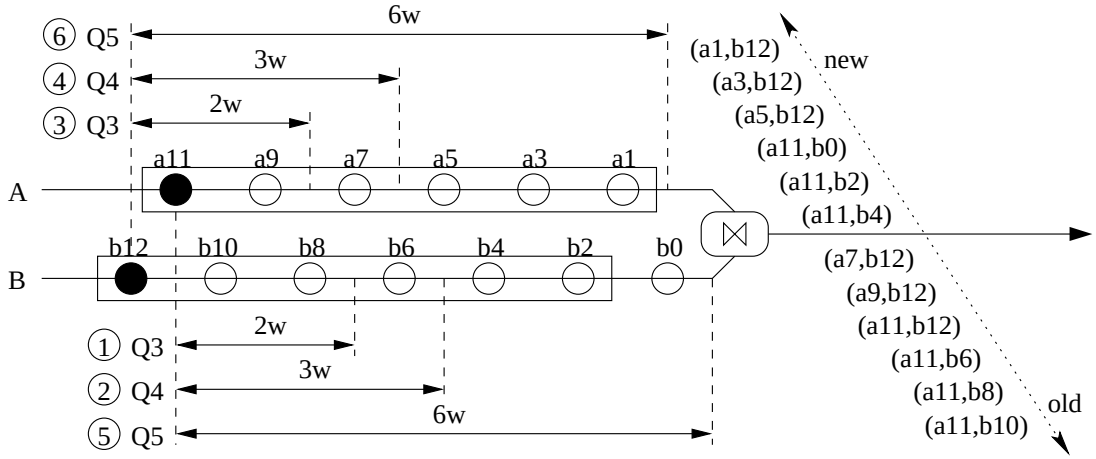

図 3.6: MQT を用いた共有ウィンドウ結合

表 3.1: MQT で用いる行列

—	w_3	w_4	w_5
0	$\frac{1}{2w}$	$\frac{2}{3w}$	$\frac{2}{3w}$
w_3	—	$\frac{1}{w}$	$\frac{1}{w}$
w_4	—	—	$\frac{1}{3w}$

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

図 3.6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，問合せの左の数字が示す実行順序となる．Q_4 の処理結果に Q_3 が含まれるため，全ての Q_4 と Q_3 が処理されたら Q_5 が実行される．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．

MQT のスケジューリングには，問合せ間のウィンドウの差，各問合せと同じウィンドウ幅を持つ問合せの数から求まる行列を用いる．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，問合せの追加や削除がある度に更新される．リストにある問合せの数を N とすると，MQT は N^2 の大きさの行列を用いて，スループットが最大になるように問合せの優先度を求める．例えば問合せが図 3.6のとき，行列は表 3.1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

に対して，既に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未処理の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．

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

各データの，未処理の問合せ全てに対して w_i と w_j の PW が求まる． $PW(0, w_j)$ は特殊な場合で，そのデータに対して 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，最も値の大きい問合せを実行する．

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 3.6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．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，次に実行する問合せを選択する計算量は $O(\log |s|)$ となる．この計算量は， FCFS における問合せ選択にかかる計算量と同じである．

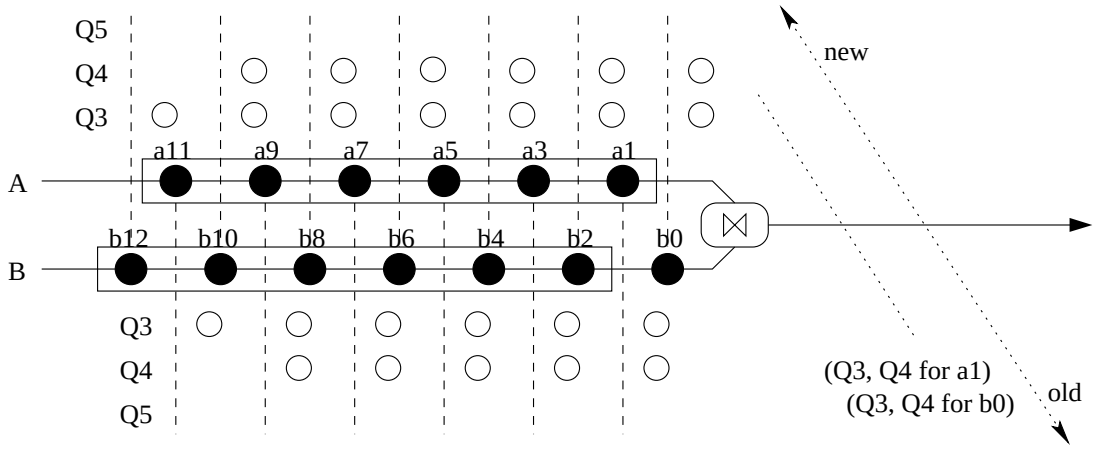

図 3.7: データ到着率が高いときの MQT

3.5 実行されない問合せの考慮

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 3.7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．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 3.7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．

図 3.8: スループットと問合せ成功率のトレードオフ

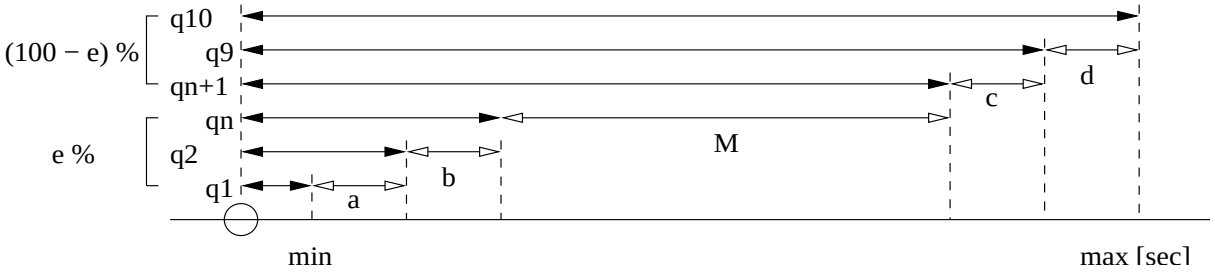

図 3.9: ウィンドウの分布

3.6 ウィンドウの分布と問合せの実行順序

スループットと問合せ成功率に影響を与える要因のうち，問合せのウィンドウの分布は最も重要な要因の 1 つである．例えば，ウィンドウの分布が図 3.8のようなとき，極端に長いウィンドウを持つ問合せを実行する代りに，短いウィンドウを持つ問合せを実行する方が良い．スループットを重視するのか，最も長いウィンドウを持つ問合せの処理を重視するかは，その時々の状況に適応して決める必要がある．

また， MQT ではウィンドウの分布によって問合せの実行順序が決まる．そのためウィンドウの分布によっては，優先して処理される問合せが生じる一方で，ほとんど実行されない問合せが生じる可能性がある．提案手法でも，ウィンドウの分布によって問合せの実行順序が変わってくる．本節では， 10 個の異なるウィンドウを持つ問合せを生成し，ウィンドウの分布によって， MQT の問合せ実行順序がどのようになるかを説明する．提案手法でも MQT と同様に，スループットを最大にする部分ではウィンドウ幅に基づいた判断をしているため，ウィンドウの分布によってどのような実行順序になるかを理解することは重要である．

図 3.9は，問合せのウィンドウの分布を表している．例えば $max = 3000$ としたときに，次の 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，

$$min = 1 < a < b < \cdots < M < \cdots < c < d \tag{3.1}$$

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

$$min = 500 \geq a \geq b \geq \cdots \geq M \geq \cdots \geq c \geq d \tag{3.2}$$

- $d < M < max$ のとき，

$$e = 20, 50, 80\text{で式(3.1)と式(3.2)}$$

1. MQT と FCFS 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 FCFS 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： 1-2,3-10

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

3.4で述べたように， MQT における問合せの優先度は，問合せ間のウィンドウ差，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．ウィンドウの分布をランダムに決めるとき，その分布は上記の 8 種類の分布で構成される．したがって，ウィンドウの分布と問合せの実行順序の関係は，上記の 8 種類の分布ついて調べることが重要である．

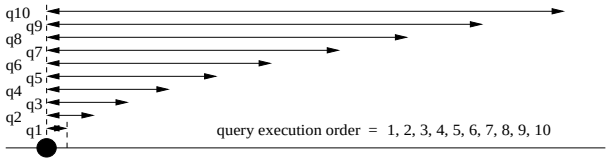

図 3.10: 1. の分布

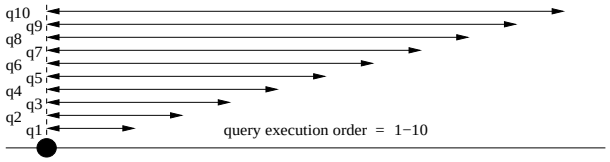

図 3.11: 2. の分布

図 3.10〜図 3.17は，生成された 8 種類の問合せのウィンドウの分布を表している．問合せの実行順序は，ある問合せを実行するとき，どの問合せまで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，問合せ 10 まで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 3.10と図 3.11について説明する．図 3.10は， MQT と FCFS のスループットの差が最大になる分布を示している． FCFS 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 FCFS 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 3.10，図 3.12，図 3.14，図 3.16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 3.11は， MQT と FCFS 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 FCFS 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 3.11，図 3.13，図 3.15，図 3.17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 3.12〜図 3.17は，図 3.10と図 3.11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 3.10と図 3.11の組合せのように， MQT と FCFS 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．

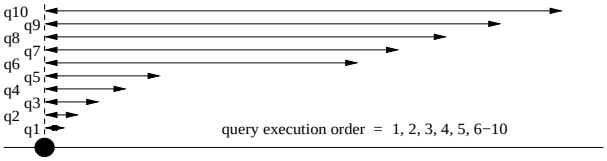

図 3.12: 3. の分布

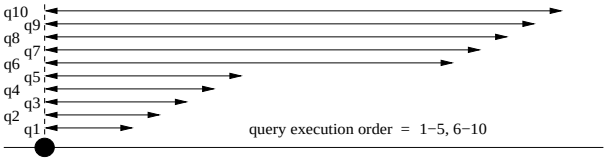

図 3.13: 4. の分布

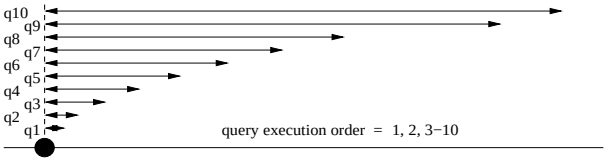

図 3.14: 5. の分布

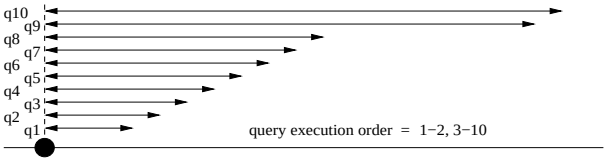

図 3.15: 6. の分布

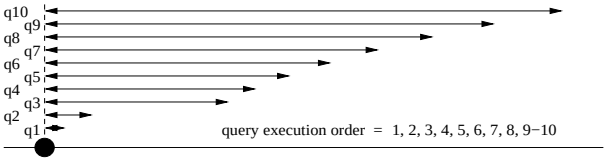

図 3.16: 7. の分布

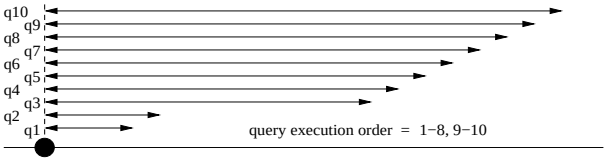

図 3.17: 8. の分布

第 4 章

MandF

本章では，データの到着率と問合せの処理状況に適応して問合せを動的にスケジューリングし，スループットと問合せ成功率が高くなるスケジューリング手法， MandF[7] を提案する．閾値を決め，単位時間ごとに求められる処理負荷と比較し，負荷が高ければ MQT，負荷が低ければ FCFS を用いてスケジューリングを行う．

4.1 アルゴリズム

4.1.1 スケジューリング手法を切替えるタイミング

問合せの動的スケジューリングを行うタイミングは，データ到着率を測定するときである．データ到着率は単位時間ごとに測定され，同時に，スケジューリングに MQT を用いるか， FCFS を用いるかが決定される．どちらの手法を用いてスケジューリングするかはフラグによって示され，処理システムは，問合せを 1 つ処理するごとにフラグを確認する．したがって，問合せの処理中にフラグが切替わっても，問合せの処理が途中で中断されることはない．

MandF では，データの到着率と問合せの処理状況から処理負荷を定義する．単位時間ごとに測定されるデータ到着率を基に，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，どれだけ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．

表 4.1: 行列から求まる問合せの重み

—	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
$\frac{2}{3w}$	$\frac{2}{3w}$	$\frac{1}{3w}$

4.1.2 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 3.8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良い可能性がある．これを考慮するため，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．

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 3.1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 4.1は，表 3.1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．

表 4.2: 実験に用いた計算機環境

CPU	Intel Pentium II 400MHz
メモリ	256MB
OS	Red Hat Linux 9 (2.4.20)
開発言語	Java (J2SE 1.4.1)

4.1.3 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．

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

例えば表 4.1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．

4.2 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．

4.2.1 実験環境

本システムの実験に用いた計算機環境は表 4.2の通りである．図 4.1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．本システムは，[6] でモデル化された環境を想定しており，2 つの情報源からデータが到着するものとする．ラッパーは情報源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．ここで求めた処理負荷と閾値を比較して，負荷が高いか低いかを判断する．負荷が低ければスケジューリングに MQT，負荷が高ければスケジューリングに FCFS を用いること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．

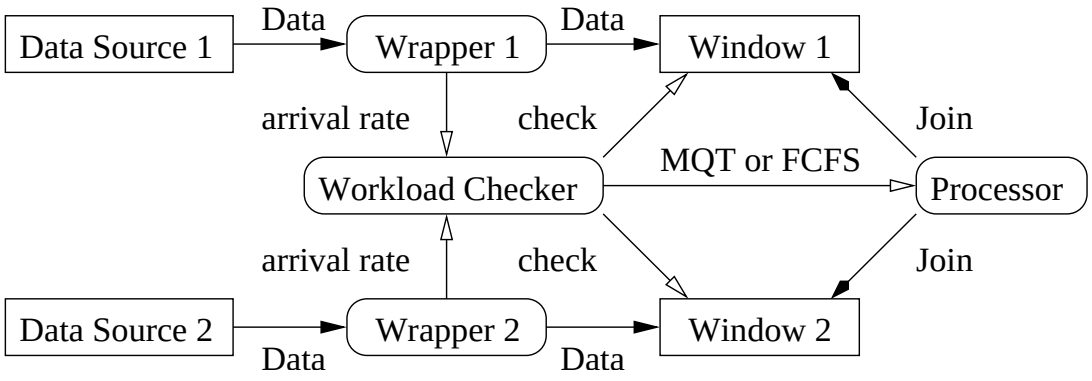

図 4.1: プロトタイプシステムの構成

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

図 4.2: 問合せのテンプレート

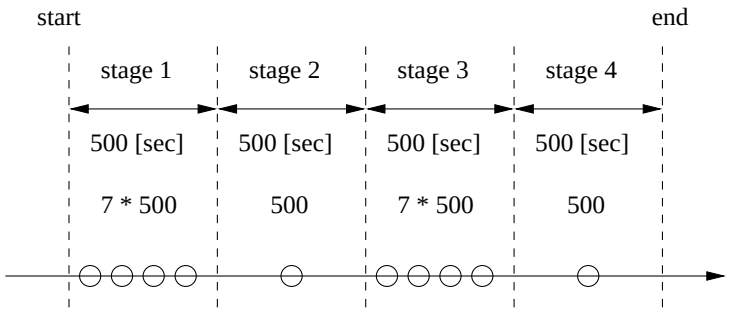

図 4.3: ステージごとの到着データ数

4.2.2 実験に用いる問合せとデータ

1 つのデータに対し，図 4.2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．システムに到着するデータは図 3.1の形式で人工的に生成した． 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 5000 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 80% のデータに対する問合せを全て処理済みとし，新しい 20% のデータに対する問合せを全て未処理とした．

データは図 4.3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．データの到着率は 0.5 秒ごとに測定する．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．

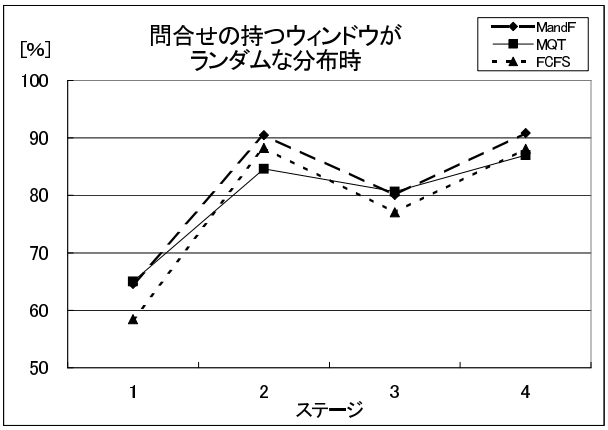

図 4.4: 処理された問合せ

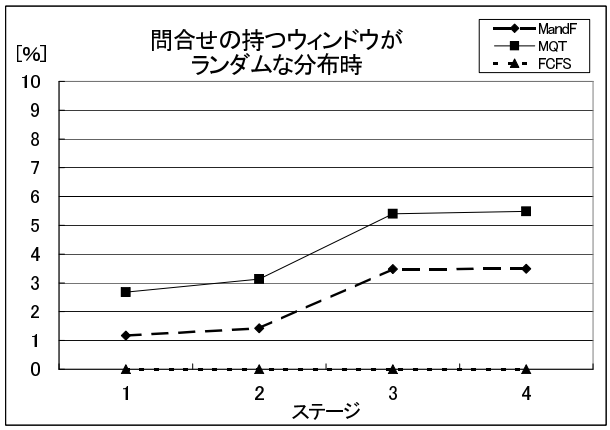

図 4.5: 実行できない問合せ

4.2.3 実験 1 : ウィンドウがランダムに分布

図 4.4と図 4.5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）， FCFS 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 4.3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 4.4と図 4.5より，実行できない問合せの割合は MQT のほぼ半分となり，処理された問合せの割合は大きいことが分かる．処理負荷が高いときに FCFS 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 4.4の処理された問合せの割合と，図 4.5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．

次に， MQT と FCFS を比較する．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，処理される問合せの割合の差が小さくなる．ここで， MQT では実行できない問合せがあるため，ステージ 2 とステージ 4 の時間が長くなると， FCFS の方が処理される問合せの割合が大きくなると考えられる．

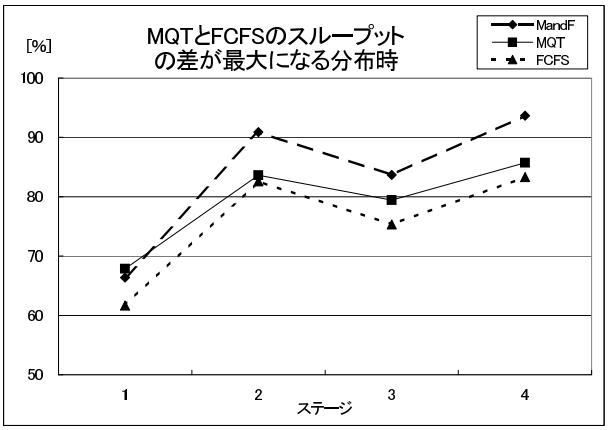

図 4.6: 処理された問合せ（P1）

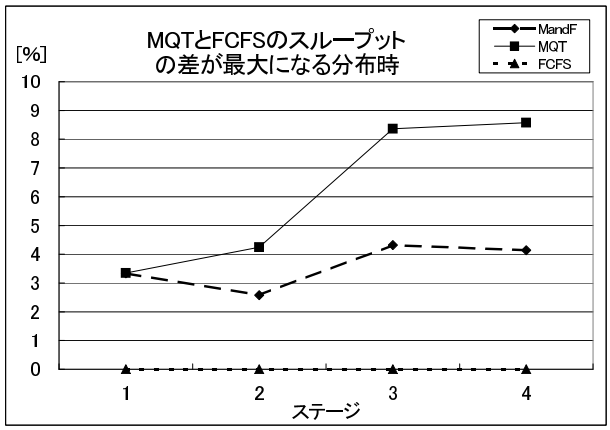

図 4.7: 実行できない問合せ（P1）

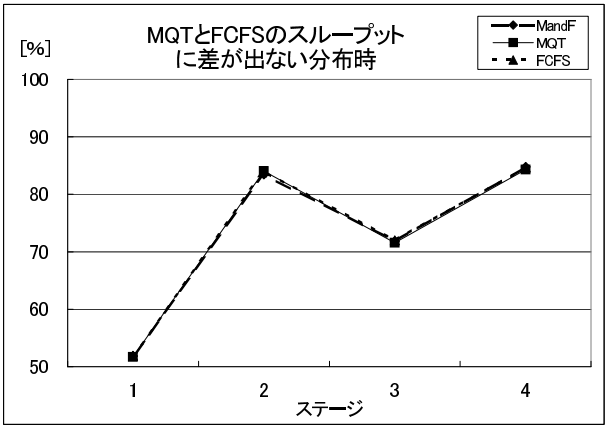

図 4.8: 処理された問合せ（P2）

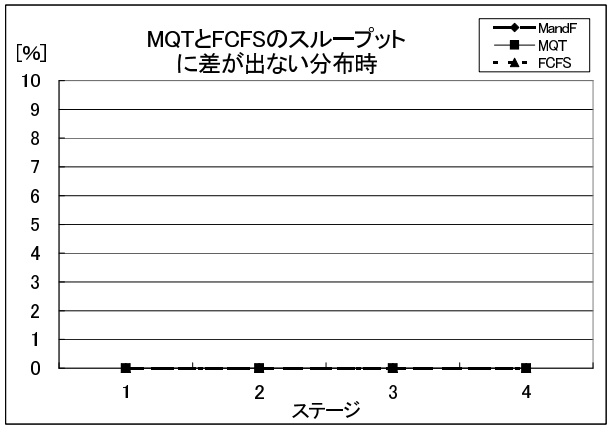

図 4.9: 実行できない問合せ（P2）

4.2.4 実験 2 : 極端な結果を示す分布

グラフの表題における P1 ～ P8 は，3.6で生成された 8 通りの分布の番号と一致している．8 種類のウィンドウの分布うち，P1 と P2 が両極端な結果を示す．

図 4.6と図 4.7において，MandF がステージ 1 で処理した問合せの割合は MQT よりもわずかに劣る．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 では，実行できない問合せを減らすために，FCFS と同じスケジューリングを行っているため，ステージ 2 からステージ 3 の傾きは FCFS と同じになっている．ステージ 4 ではステージ 2 と同じ理由の結果，提案手法が最も問合せを処理している．図 4.8と図 4.9から，ウィンドウの分布によっては 3 手法の問合せ実行順序が同じになり，同じパフォーマンスとなることが分かる．

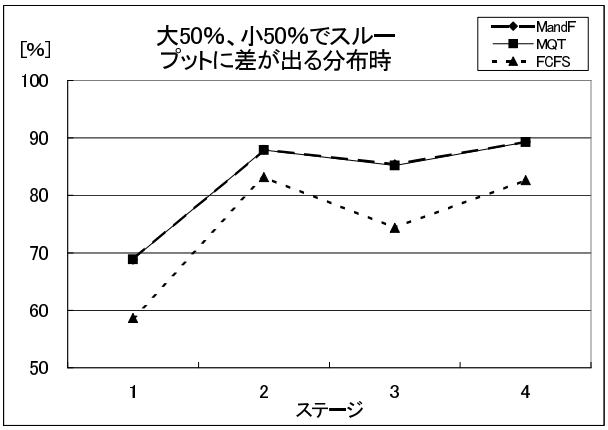

図 4.10: 処理された問合せ（P3）

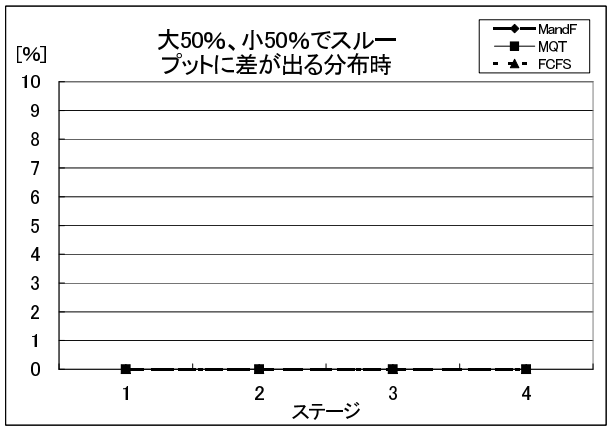

図 4.11: 実行できない問合せ（P3）

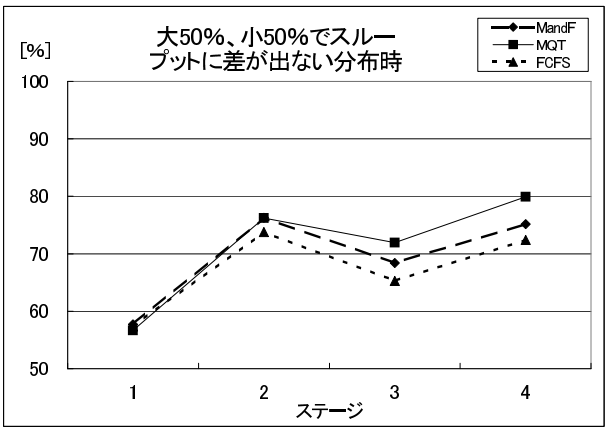

図 4.12: 処理された問合せ（P4）

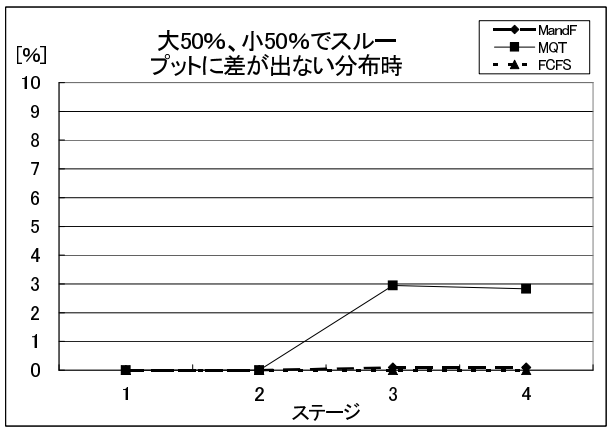

図 4.13: 実行できない問合せ（P4）

4.2.5 実験 3 : その他の分布

図 4.10〜図 4.21を通して，提案手法は従来手法よりも良い結果を示した．これは， MQT と FCFS の双方の利点を活かし，スループットを高くしつつ，実行されない問合せが少ないためと考えられる． 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 FCFS 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 4.10と図 4.11では，提案手法は短いウィンドウを持つ問合せを中心に処理しているので， FCFS よりもスループットが高く， MQT と同じスループットを実現している．

図 4.12と図 4.13では，提案手法は実行できない問合せがなく，なおかつ FCFS よりも高いスループットを実現している．

図 4.14と図 4.15では，提案手法は短いウィンドウを持つ問合せを中心に処理するので，FCFS よりもわずかにスループットは高く， MQT と同じスループットを実現している．

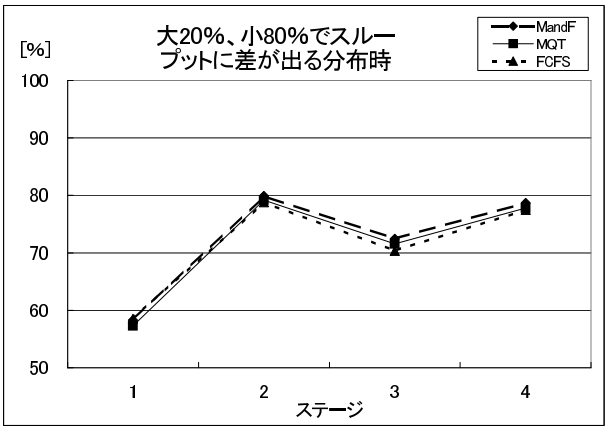

図 4.14: 処理された問合せ（P5）

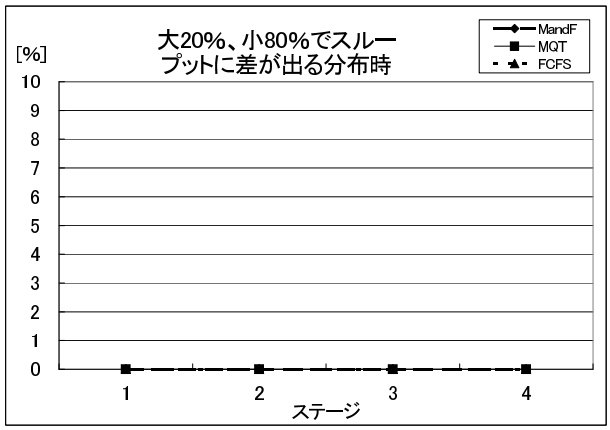

図 4.15: 実行できない問合せ（P5）

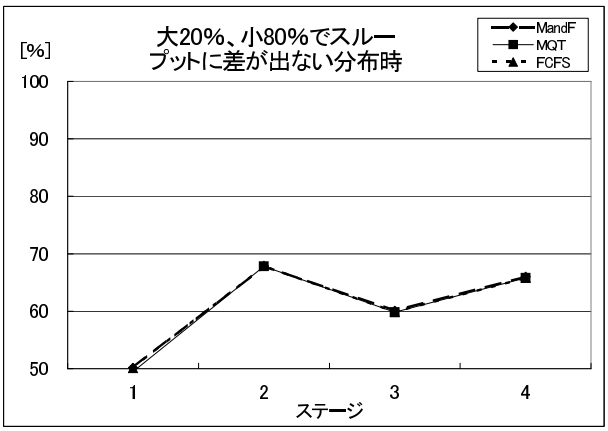

図 4.16: 処理された問合せ（P6）

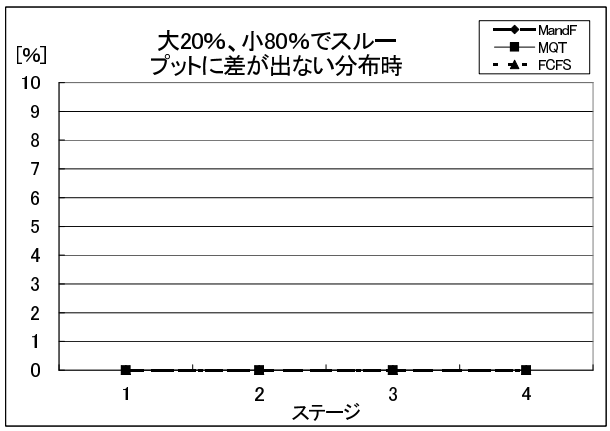

図 4.17: 実行できない問合せ（P6）

図 4.16と図 4.17では，問合せの実行順序が，図 4.8と図 4.9のときのスループットが向上しない FCFS よりのときと同じなので， 3 つの手法が同じパフォーマンスになった．

図 4.18と図 4.19では，提案手法は新しいデータが多いときにスループットは高くなるが，新しいデータが少ないときには，実行できない問合せの分が処理できない．処理された問合せの割合は MQT より大きく， FCFS より小さくなっている．

図 4.20と図 4.21では，提案手法は実行できない問合せがない分， MQT と比べて処理された問合せの割合が大きくなっている．また， FCFS よりも多くの問合せを処理している．

提案手法の利点は，図 4.6〜図 4.21を 3 つに分けることで見られる． 1 つ目は MQT と提案手法で実行できない問合せがある場合で，提案手法の方が MQT よりも実行できない問合せの数は少なく，処理された問合せの割合も大きかった． 2 つ目は MQT で実行できない問合せがあり，提案手法では問合せが全て処理された場合で，提案手法の方が FCFS よりも多くの問合せを処理した． 3 つ目は MQT で問合せが全て処理された場合で，提案手法は MQT と同じパフォーマンスを示した．

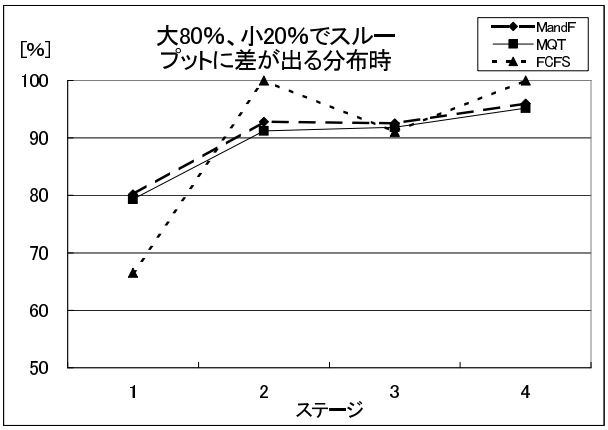

図 4.18: 処理された問合せ（P7）

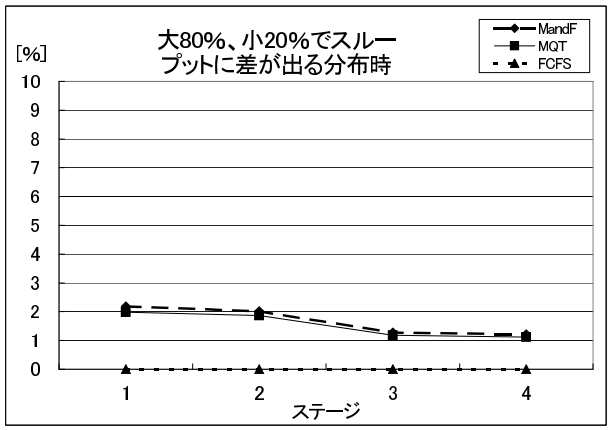

図 4.19: 実行できない問合せ（P7）

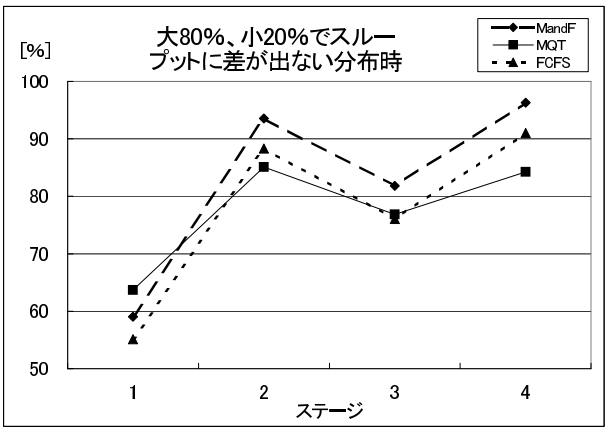

図 4.20: 処理された問合せ（P8）

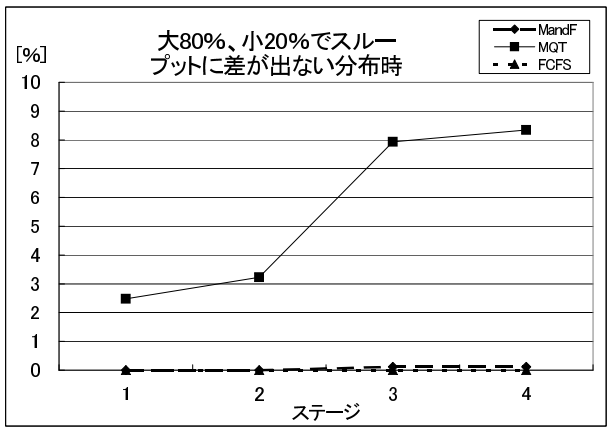

図 4.21: 実行できない問合せ（P8）

4.3 まとめ

本章では，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 FCFS 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．

第 5 章

adaptive MandF

本章では， MandF を拡張し，処理負荷の傾向に応じて閾値の自動調節をするスケジューリング手法， adaptive MandF[8] 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 FCFS を切替える．

5.1 MandF の課題

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，ストリームデータを処理するシステムでは，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．

MandF は処理負荷に基づいて MQT と FCFS を切替え，双方の利点が活きるように問合せをスケジューリングする．ただし，切替えに用いる閾値は静的な値であり，処理負荷が今後どのように変化するかには対応できていない．処理負荷が低い状況が続く場合や，高い状況が続く場合では MQT と FCFS のどちらを多用すべきかは変わってくる．したがって，負荷の傾向を考慮し，より良いタイミングで MQT と FCFS を切替えることが重要である．

5.2 アルゴリズム

5.2.1 MandF の切替アルゴリズム

MQT と FCFS の切替えの判断は，初期値で与える閾値 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．各問合せには重みが付いており，スループットが上がる問合せに大きい値が付いている．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 として用いている．以降，単位時間に到着するデータの数を到着率と呼ぶ.

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．

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

MandF では， β は静的な値である．負荷 S が閾値 α 以上のときに FCFS を用い，負荷 S が閾値 α 未満のときに MQT を用いる．

5.2.2 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．

β が 1 に近いときは， 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 α も大きくなり，実行できない問合せが生じても MQT により問合せをスケジューリングする．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．処理負荷の傾向が高いときは，より FCFS に切替わりやすい β に設定する方が良いと考えられる．

β が 0 に近いときは， FCFS 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 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， FCFS により問合せをスケジューリングする．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．処理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．

5.2.3 β 調節に用いる情報

β 調節のために，処理負荷の傾向とデータの到着率を用いる．これら 2 つの情報を用いて，次の処理負荷がどうなるのかを推定し，次に設定すべき β である β_{update} を決定する．処理負荷が高くても，データの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．したがって，処理負荷の傾向とデータの到着率の 2 つをもって，次に設定すべき β を求める必要がある．

5.2.4 β の調節アルゴリズム

β は，データの到着率が測定されるタイミングで更新される．まず，次に β が取る値として 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 を取るべきであり，次の式で求まる．

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

そして，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．

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

続いて，次に取りうるデータの到着率として 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$_{MQT}$， FCFS に切替わったときの到着率を 到着率$_{FCFS}$ とする． 到着率$_{next}$ が $\frac{(\text{到着率}_{MQT} + \text{到着率}_{FCFS})}{2}$ 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 の判断から，次に設定すべき β_{update} を次のように決定する．

1. $\beta_{next} > \beta_{now}$(負荷：大 → 大)
   必ず FCFS で処理するため， $\beta_{update} = \beta_{now}$

2. $\beta_{now} == 0$(負荷： FCFS 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： FCFS 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)
   FCFS 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$\beta_{update} = 0.03$

5.3 評価実験

adaptive MandF の有効性を示すため， 4.2.1のシステムを用いて評価実験を行った.

5.3.1 実験環境

4.2で行った実験と同じ環境で adaptive MandF を実行する． adaptive MandF 以外の結果は， 4.2で示した結果と同じである．実験環境は 4.2.1と同じであり，実験に用いる問合せとデータも 4.2.2と同じである．また， adaptive MandF で必要な β の初期値は 0.05 とした.

5.3.2 実験 1 : ウィンドウがランダムに分布

x 軸は，図 4.3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 5.1と図 5.2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 5.1の処理された問合せの割合と，図 5.2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.

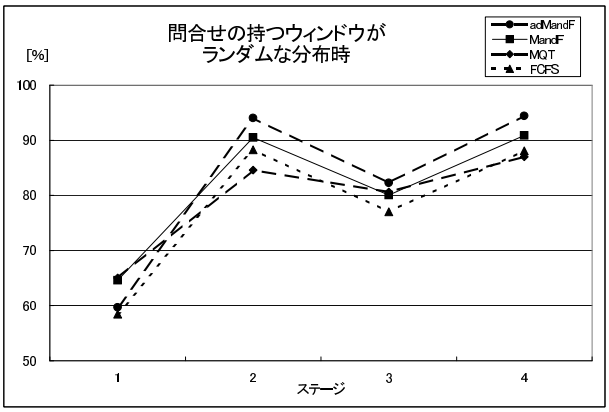

図 5.1: 処理された問合せ

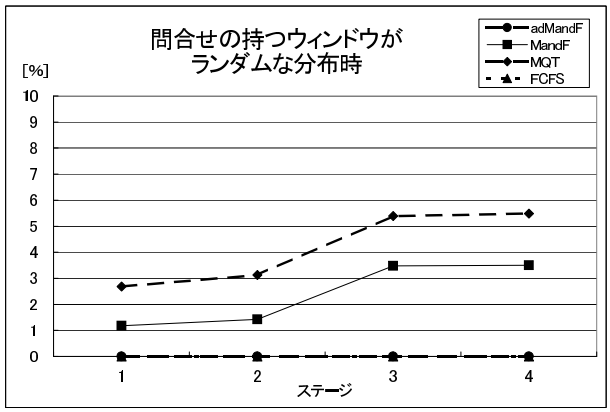

図 5.2: 実行できない問合せ

5.3.3 実験 2 : 様々なウィンドウの分布

グラフの表題における P1 ～ P8 は，3.6で生成された 8 通りの分布の番号と一致している．8 種類のウィンドウの分布うち，P1 と P2 が両極端な結果を示す．図 5.7～図 5.18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 FCFS 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 FCFS を多く用い，長いウィンドウを持つ問合せも多く処理した．しかし，提案手法は MQT を多く用い，実行できない問合せもなかったため，MQT よりも多く問合せを処理した．

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 FCFS よりもスループットが高くなるため，FCFS 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 FCFS と同じになる P2，P6 では，処理された問合せの割合は FCFS と同じになったが，問合せの実行順序が FCFS と異なる P3，P5 では，FCFS よりも多くの問合せを処理した．

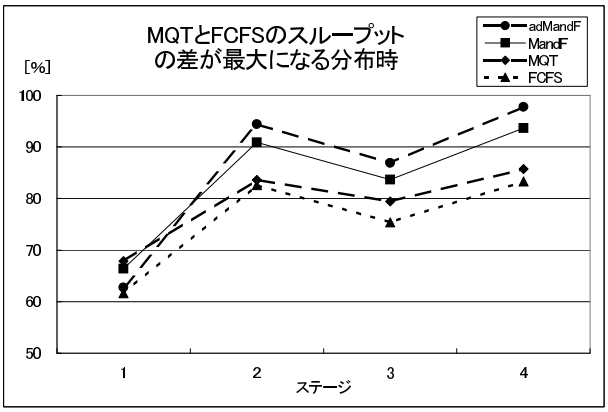

図 5.3: 処理された問合せ（P1）

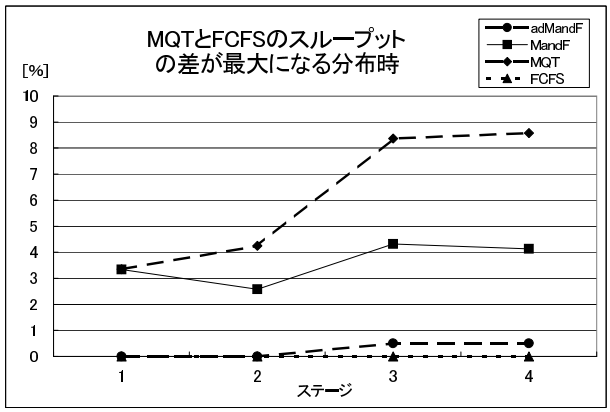

図 5.4: 実行できない問合せ（P1）

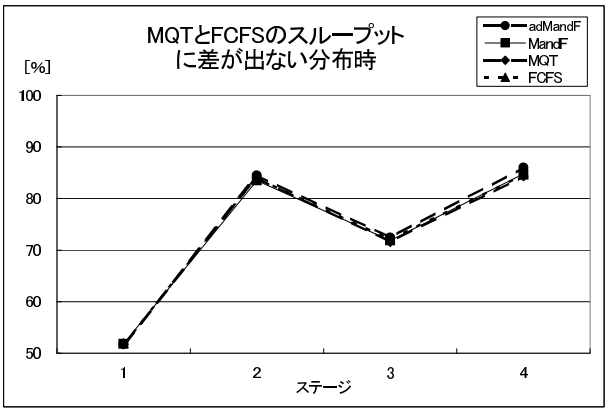

図 5.5: 処理された問合せ（P2）

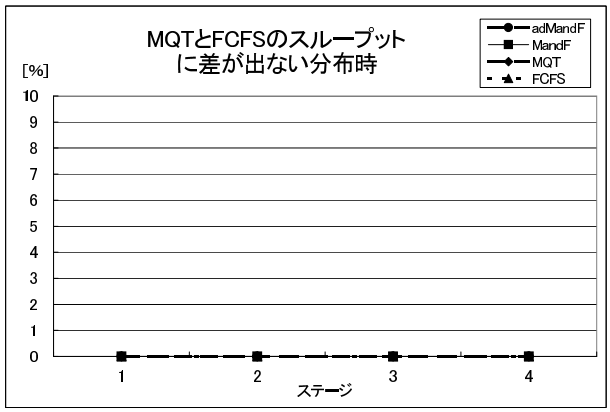

図 5.6: 実行できない問合せ（P2）

5.4 まとめ

本章では，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 FCFS 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，処理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．

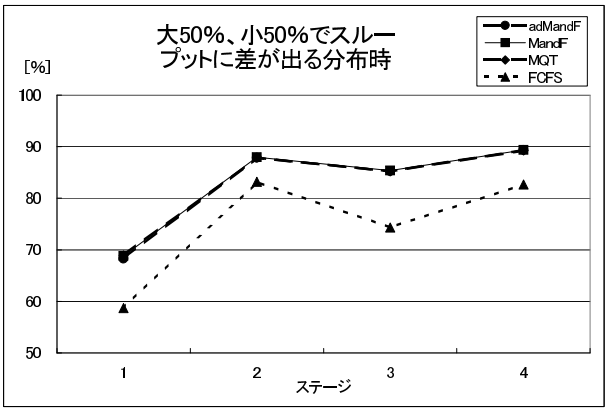

図 5.7: 処理された問合せ（P3）

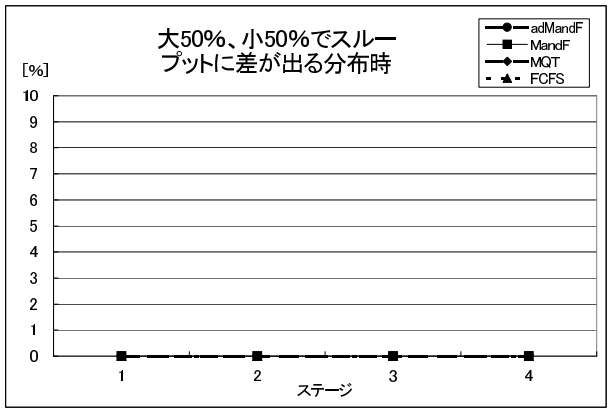

図 5.8: 実行できない問合せ（P3）

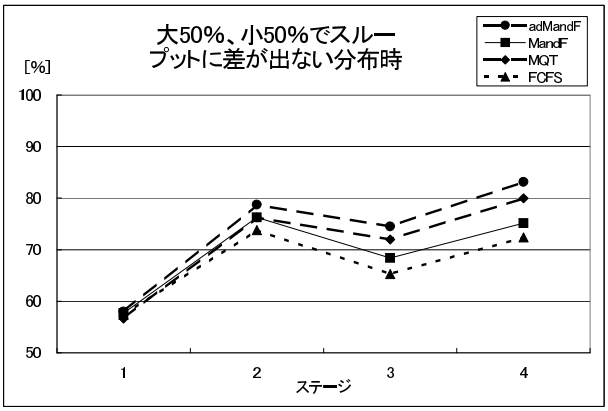

図 5.9: 処理された問合せ（P4）

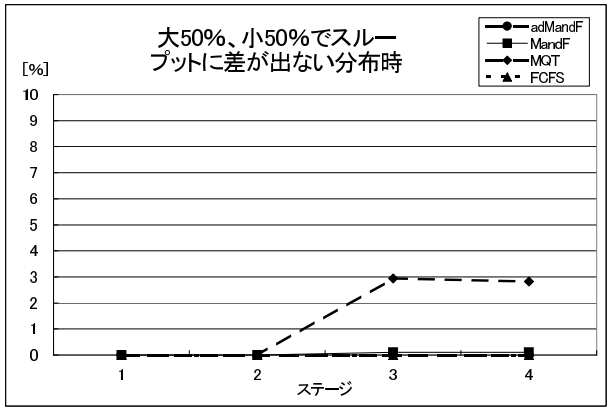

図 5.10: 実行できない問合せ（P4）

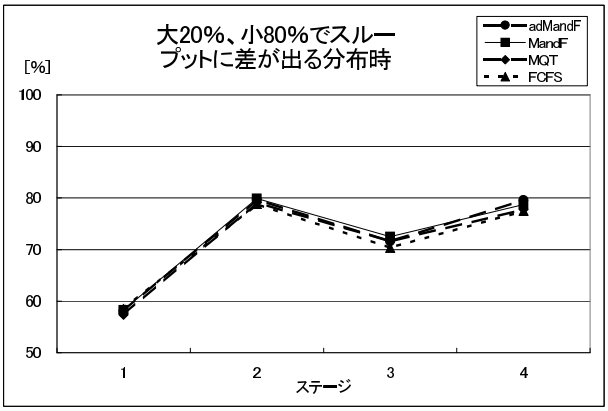

図 5.11: 処理された問合せ（P5）

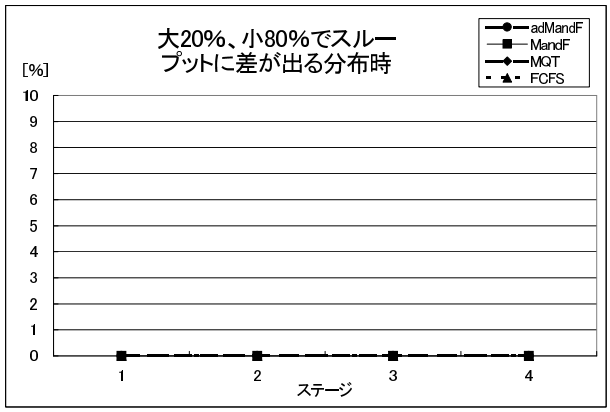

図 5.12: 実行できない問合せ（P5）

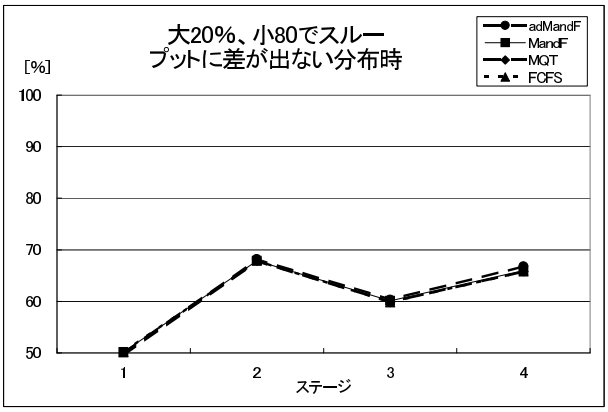

図 5.13: 処理された問合せ（P6）

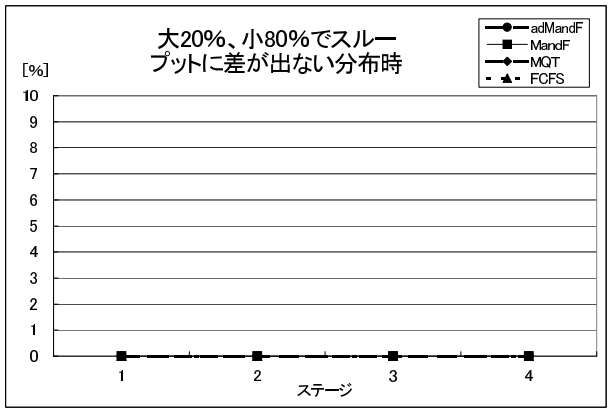

図 5.14: 実行できない問合せ（P6）

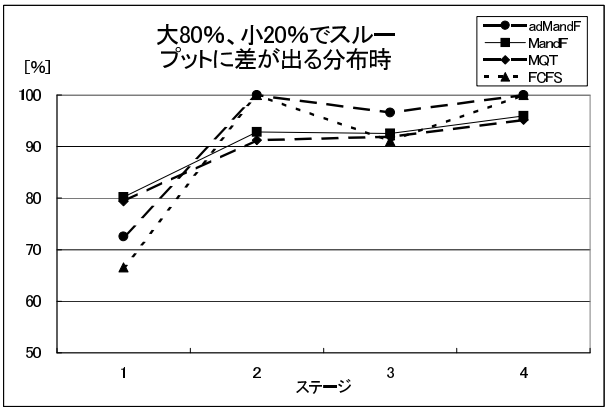

図 5.15: 処理された問合せ（P7）

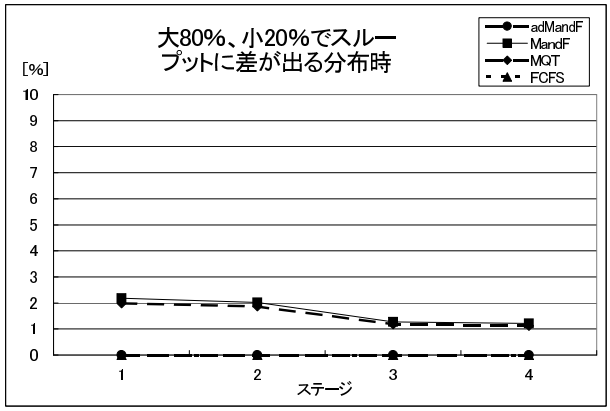

図 5.16: 実行できない問合せ（P7）

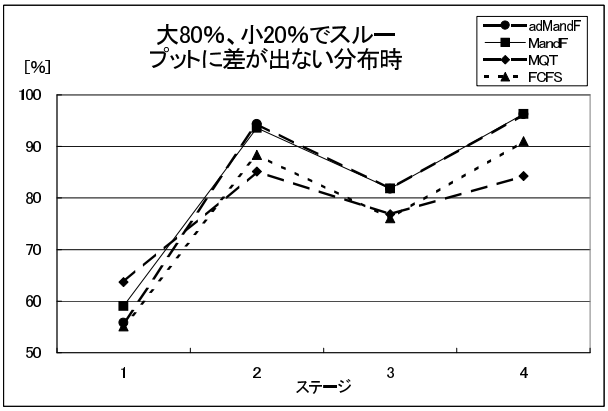

図 5.17: 処理された問合せ（P8）

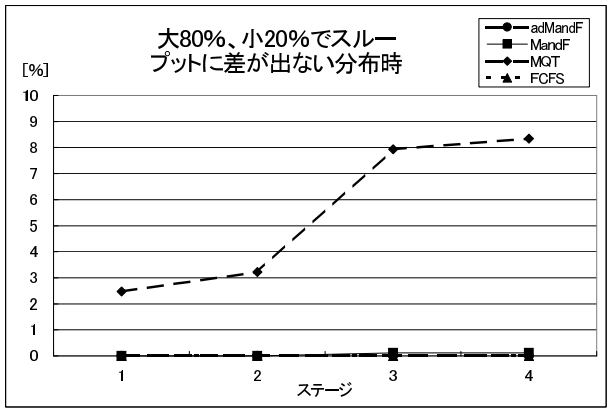

図 5.18: 実行できない問合せ（P8）

第 6 章

ExT

本節では，実行されない問合せを考慮し，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） [9] を提案する．

6.1 アルゴリズム

MandF や adaptive MandF とは異なり， ExT では MQT や FCFS などの従来手法は用いない．ただし ExT でも，その時々のデータの到着状況や，問合せの処理状況に適応して動的なスケジューリングを行う．また，問合せごとに問合せ成功率を指定できる点でも異なり，ユーザごとのサービスに偏りが少ないスケジューリングを行うことが可能である．

RxW[23] 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．

6.1.1 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 $S_i(0 \leq i \leq \infty)$ 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，s の各データを $s_i(0 \leq i \leq |s|-1)$ と表す．本研究では配列でバッファを管理するため，$|s|$ は配列の要素数である．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 とし，L の各問合せを $L_j(1 \leq j \leq N)$ と表す．このとき，$s_i(0 \leq i \leq |s|-1)$ に対する $L_j(1 \leq j \leq N)$ 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．問合せ成功率，e，t，et については，6.1.2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 $S_{interval}$ とし，$S_{interval}$ 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．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 と定義し，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 に対して定義する．

6.1.2 Naive ExT

ExT では E と T という値を定義し，2つを掛合わせた値を用いて実行する問合せに優先付けをする．E，T はそれぞれ小さい値ほど優先度が高くなるため，E×T 値が小さいほど問合せの優先度が上がる．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．

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

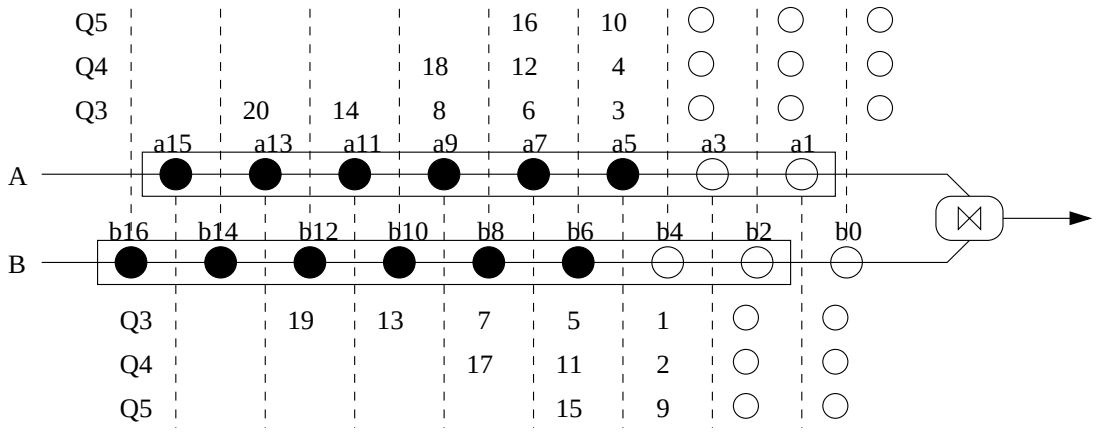

図 6.1: ExT を用いた共有ウィンドウ結合

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．

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

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

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，この値の逆数をスケジューリングに用いている．

処理に余裕があるかどうかは E に反映され，スループットが上がりやすいかどうかは T に反映される．したがって， E×T 値が小さい順に問合せを実行することで，それら両方を考慮したスケジューリングが可能になる．図 6.1は， 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． E×T 値が小さい順に 20 番目まで示している．

ExT における問合せ実行順序が最も極端な場合，データの到着率が 0 のときは T だけがスケジューリングに反映され， MQT と同じ実行順序になる．また， T が全て同じ値のときは E だけがスケジューリングに反映され， FCFS と同じ実行順序になる．問合せが失敗すると応答時間は ∞ であり，必ず問合せが成功すると仮定すると， ExT の平均応答時間は MQT と FCFS の間となる．

ExT ではユーザが要求する問合せ成功率を満たしつつ，そのときにスループットを最大にすることが目的である．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，ユーザから要求される問合せ成功率を考慮する．ここで，要求される問合せ成功率とは，後述の図 6.3の形式でユーザに指定されるものとする．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．

1. $base = 1;$ // 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;

   $while(base \leq N)\{$

   $MaxS = 0;$ // 最も高い問合せ成功率

   $Qno = 1;$ // $MaxS$の成功率の問合せ

   $for(int\ i = base;\ i \leq N;\ i{+}{+})\{$

   $if(\ (Q[*][L_i].s \geq MaxS)\{$

   $MaxS = Q[*][L_i].s;$

   $Qnum = i;$

   $\}$

   $\}$

   $for(;\ base \leq Qno;\ base{+}{+})\{$

   $Q[*][L_{base}].s = Q[*][L_{Qno}].s)\{$

   $\}$

$\}$

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． ExT では， 1 つ問合せを実行するごとに，次に実行する問合せを求める．問合せ成功率が要求を上回るときは 2. を行わず，要求を下回るときは 2. を行う．

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){

$Q[s_{oldest}][L_N].e, Q[s_{oldest}][L_N].t, Q[s_{oldest}][L_N].et$を求める;

$exec =$ E×T値が最小の問合せ;

}

}

2. $Q[s_{oldest}][L_N]$ が実行される実行順序を求める．

$plan[s_i][L_j] = Q[s_i][L_j]$をE×T値でソート;

$MAXtput = 0$;

for($plan[s_{oldest}][L_N]$と，これより前に実行される問合せ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($(plan[s_{oldest}][L_N].e \geq ptime)$ && $(tput \geq MAXtput)$){

$MAXtput = tput$;

$exec =$ 最初に実行する問合せ;

}

}

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．

6.1.3 Exaustive ExT

Exaustive 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．ユーザが要求する問合せ成功率を満たしている間は E×T が最小の問合せを実行すればよいが，要求される成功率を下回るときは，どのタイミングで $Q[s_{oldest}][L_N]$ を実行するかを素早く決定する必要がある． 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．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． E×T 値が小さい順に $Q[s_{oldest}][L_N]$ を比較しているので，問合せの実行順序はボトムアップに計算される．

以下に Exaustive ExT のアルゴリズムを示す．問合せ成功率が要求を上回るときは 2. を行わず，要求を下回るときは 2. を行う． 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;

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

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

   $\}$

1. の計算量は $O(|s| \times N)$， 2. の計算量は $O(1)$ であり，全体の計算量は $O(|s| \times N)$ 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．

6.1.4 Pruned ExT

Pruned ExT は実時間でのスケジューリングが可能であり，問合せの選択時間にバッファサイズが影響しないスケジューリング手法である． 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． Exaustive ExT との違いは 1. であり， E×T 値を求める問合せの数を減らしている．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， $Q[s_{oldest}][L_j]$ の E×T 値だけを求めればよい．

表 6.1: 実験に用いた計算機環境

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)

以下に Pruned ExT のアルゴリズムを示す．問合せ成功率が要求を上回るときは 2. を行わず，要求を下回るときは 2. を行う． Exaustive 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){

   $Q[s_{oldest}][L_j].e, Q[s_{oldest}][L_j].t, Q[s_{oldest}][L_j].et$を求める;

   $exec$ = E×T値が最小の問合せ;

   }

2. Exaustive 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，スケジューリングにかかる時間は問合せの処理時間に対して無視できるほど短くなっている．

6.2 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．

6.2.1 実験環境

本システムの実験に用いた計算機環境は表 6.1の通りである．図 6.2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う．

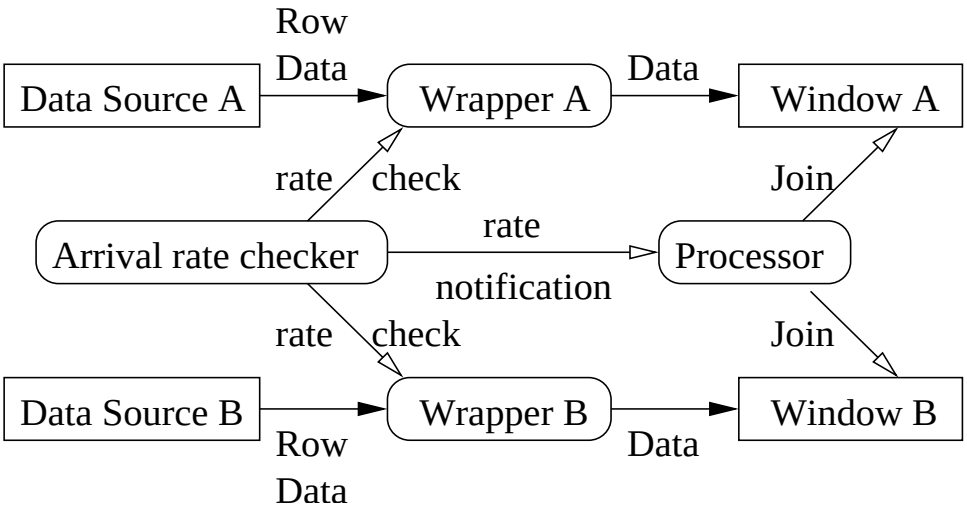

図 6.2: プロトタイプシステムの構成

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

図 6.3: 実験に用いる問合せのテンプレート

6.2.2 実験に用いる問合せとデータ

1 つのデータに対し，図 6.3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，問合せ成功率を指定するための SUCCESS 節を加えた．本実験では， SUCCESS は全ての問合せで 80% とした．システムに到着するデータは，図 3.1の形式で人工的に生成した． 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 100 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 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% のデータに対する問合せを全て未処理とした．また，データの到着率は 5 秒ごとに測定される．

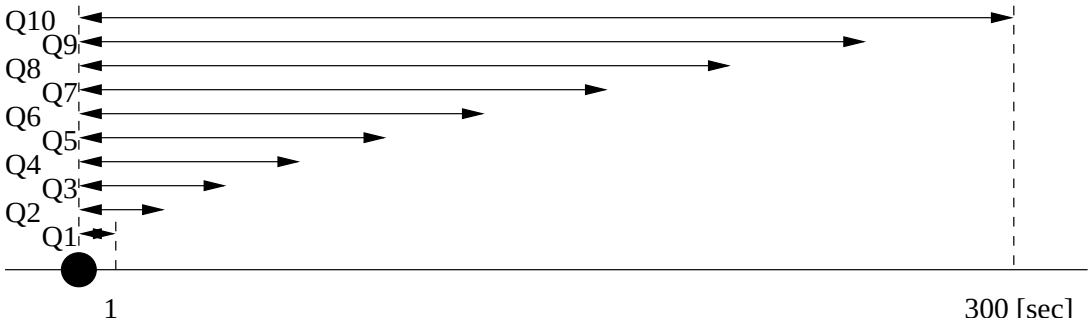

図 6.4: ウィンドウの分布

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 3.0，パレート分布の期待値を 42.0 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 14.0 となっている．

6.2.3 実験 1 : 問合せ成功数と失敗数

図 6.4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 の問合せの実行順序が最も異なる分布で，スループットの差が最大になる．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．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 6.5〜図 6.8は，提案手法である Pruned ExT と Exaustive ExT，従来手法である MQT と FCFS の 4 手法について， 300 秒の間に図 6.4の問合せを実行したときの実験結果を示している．図 6.5〜図 6.7においては，問合せ処理を行う 300 秒のうち，最初の 60 秒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， 60 秒から 300 秒までの結果を示した．指数分布とパレート分布を用いて発生させた 10 パターンのデータストリームを用い，各手法に対して， 10 回パターンの実験を行ったときの平均をとっている．まず問合せ成功数について述べ，続いて問合せ失敗数について述べる．

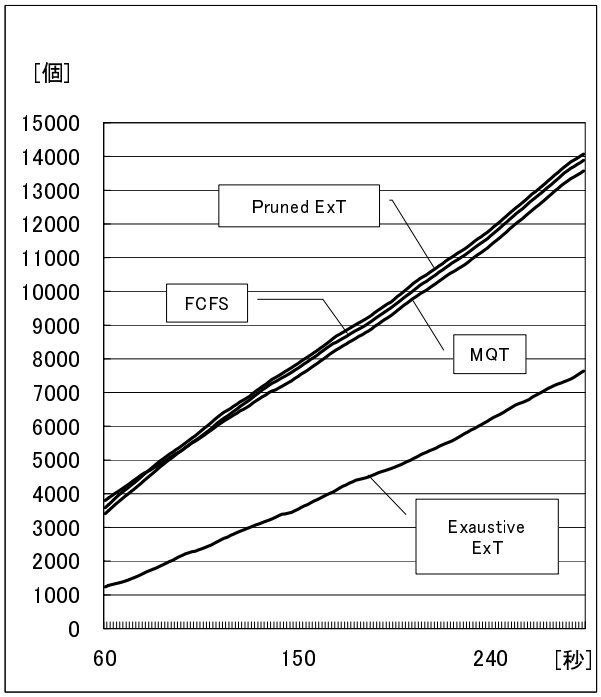

図 6.5: 問合せ成功数

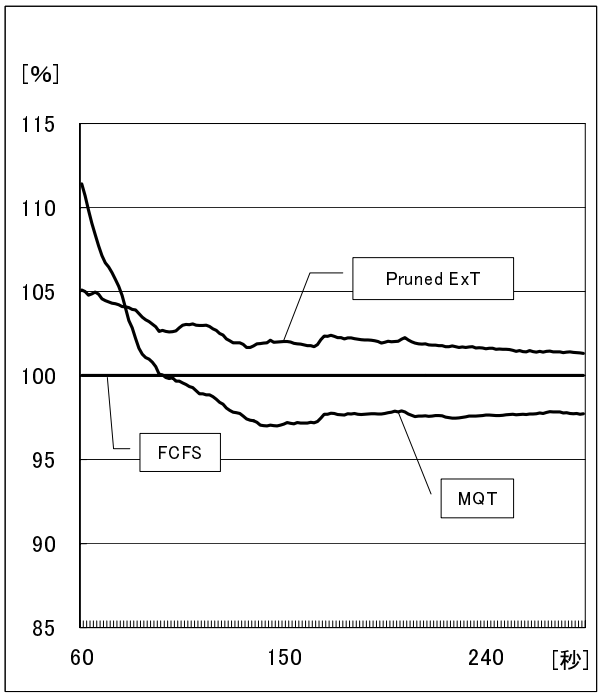

図 6.6: 問合せ成功数（対 FCFS）

図 6.5は，4 手法の問合せ成功数を示している．時間の経過にほぼ比例して，問合せが処理されていることがわかる． Exaustive ExT は問合せの選択に時間がかかるため，他の 3 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．問合せの選択にかかる時間については，6.2.4で詳しく述べる．図 6.6は，図 6.5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 90 秒経過したあたりから， Pruned ExT で最も多くの問合せが成功しており，続いて 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 3 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．問合せ処理状況の初期状態は， 99% が未処理となっているため，実験を開始して 90 秒までは Pruned ExT と MQT が FCFS に比べて多くの問合せを処理している．

図 6.7は，4 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 Exaustive ExT が最も多くの問合せを失敗しており，続いて MQT， FCFS， Pruned ExT の順に多く失敗している． 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

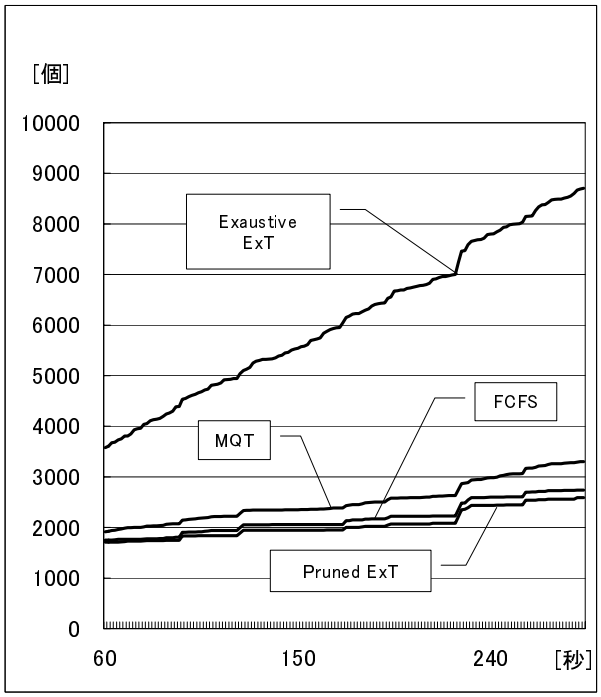

図 6.7: 問合せ失敗数

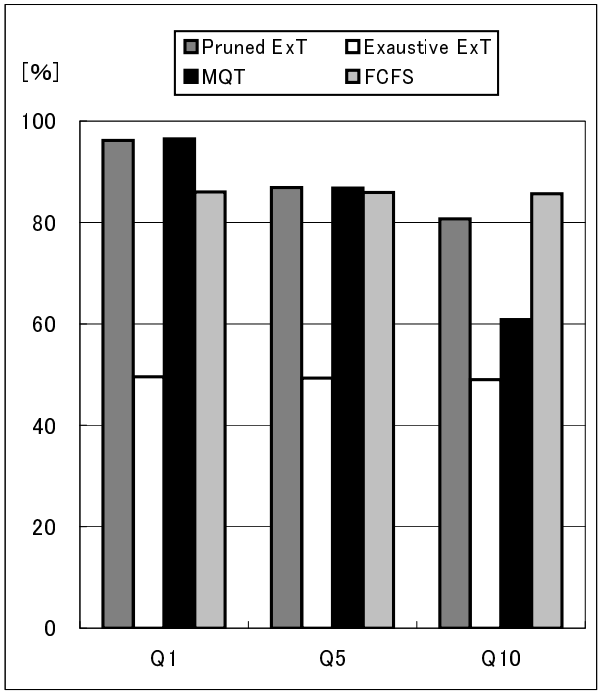

図 6.8: 問合せごとの成功率

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 6.8は， 300 秒が経過した時点の， 4 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 60% 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 6.5〜図 6.8を通して，提案手法は短期的なスループットが高く，問合せ成功率が高いために長期的な問合せ処理数も多いことが分かる．

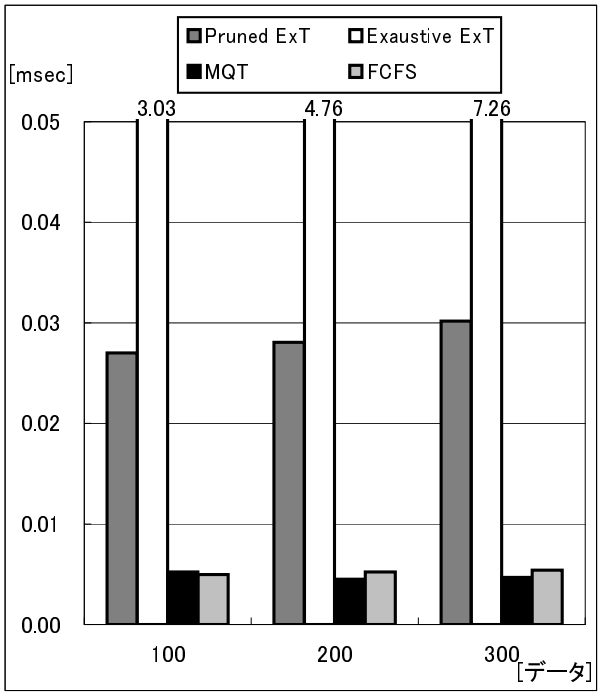

図 6.9: 1 回当たりの問合せ選択時間

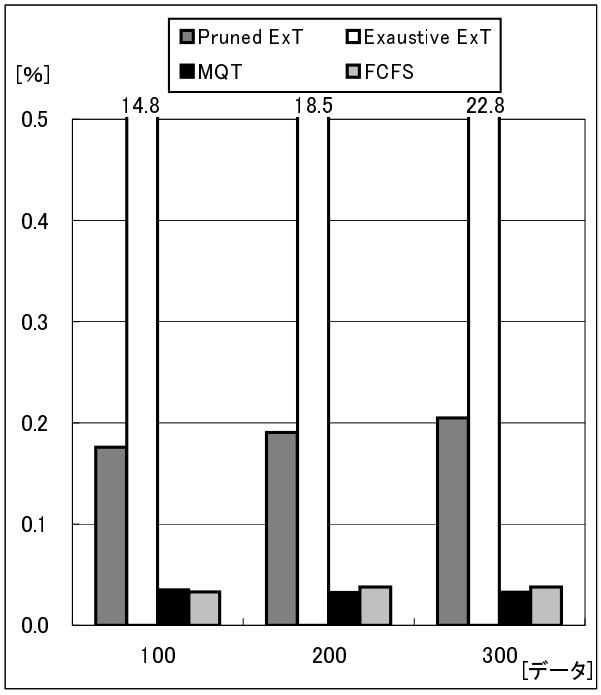

図 6.10: 問合せ選択／全体の処理

6.2.4 実験 2：問合せ選択時間の変化

図 6.9は，バッファサイズを変化させたときの，4 手法の問合せ選択時間を示している．300 秒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．Exaustive 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．4 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，Exaustive ExT に比べ，他の 3 手法はバッファサイズの影響が小さかった．

図 6.10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，結合用バッファの更新時間が含まれる．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，問合せ選択時間は無視できるほど小さいといえる．4 手法ともバッファの大きさに比例して，問合せ選択時間の割合は大きくなっている．図 6.9のときと同様に，Exaustive ExT 以外の 3 手法はバッファサイズの影響は小さかった．

6.2.5　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 3.9において $max = 300$ としたときの，図 6.4を含む 8 通りの分布を用いて実験を行った． 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．

6.3　まとめ

本章では，ユーザが要求する問合せ成功率を満たし，そのときにスループットを高くする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， ExT では問合せの成功率を考慮し，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．バッファサイズが変化しても，問合せ選択時間に影響しない手法であることも示した．

第 7 章

おわりに

本稿では，共有ウィンドウ結合のスケジューリング手法として MandF， adaptive MandF，ExT を提案した．従来のスケジューリング手法とは異なり，実行されない問合せを考慮し，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ことを実験で示した．

MandF では，従来手法の MQT と FCFS を処理負荷に基づいて切替えることで，双方の利点を用いたスケジューリングが可能であることを示した． adaptive MandF では MandF を拡張し，より良いタイミングで MQT と FCFS を切替えるための閾値の自動調節を行った．また， ExT では従来手法の MQT と FCFS を用いずに，実行されない問合せを考慮し，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るスケジューリング手法を提案した． ExT は実時間でのスケジューリングが可能で，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すことができた．

今後の課題として， Load Shedding との併用が考えられる．リソースの限界を越える程にデータが到着してしまうと，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．

参考文献

[1] Donald Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Greg Seidman, Michael Stonebraker, Nesime Tatbul, and Stanley B. Zdonik. Monitoring streams - a new class of data management applications. In *Proceedings of the 28th International Conference on Very Large Data Bases*, pp. 215–226, August 2002.

[2] Rajeev Motwani, Jennifer Widom, Arvind Arasu, Brian Babcock, Shivnath Babu, Mayur Datar, Gurmeet Manku, Chris Olston, Justin Rosenstein, and Rohit Varma. Query processing, resource management, and approximation in a data stream management system. In *Proceedings of the First Biennial Conference on Innovative Data Systems Research*, January 2003. Available at http://www-db.cs.wisc.edu/cidr/cidr2003/program/p22.pdf.

[3] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Sam Madden, Vijayshankar Raman, Fred Reiss, and Mehul Shah. TelegraphCQ: Continuous dataflow processing for an uncertain world. In *Proceedings of the First Biennial Conference on Innovative Data Systems Research*, January 2003. Available at http://www-db.cs.wisc.edu/cidr/cidr2003/program/p24.pdf.

[4] Arvind Arasu, Mitch Cherniack, Eduardo F. Galvez, David Maier, Anurag Maskey, Esther Ryvkina, Michael Stonebraker, and Richard Tibbetts. Linear road : A stream data management benchmark. In *Proceedings of the 30th International Conference on Very Large Data Bases*, pp. 480–491, September 2004.

[5] Nich Koudas and Divesh Srivastava. Data stream query processing. Available at http://icde2005.is.tsukuba.ac.jp/, April 2005.

[6] Moustafa A. Hammad, Michael J. Franklin, Walid G. Aref, and Ahmed K. Elmagarmid. Scheduling for shared window joins over data streams. In *Proceedings of the*

29th International Conference on Very Large Data Bases, pp. 297–308, September 2003.

[7] 多田直剛, 有次正義. 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, March 2005.

[8] 多田直剛, 有次正義. 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. 夏のデータベースワークショップ（DBWS 2005）, July 2005. Available at http://fw8.bookpark.ne.jp/cm/ipsj/select_signotes3.asp?category2=DBS&vol=2005.

[9] 多田直剛, 有次正義. 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. 電子情報通信学会第 17 回データ工学ワークショップ（DEWS 2006）, March 2006. To appear.

[10] Samuel Madden, Mehul Shah, Joseph M. Hellerstein, and Vijayshankar Raman. Continuously adaptive continuous queries over streams. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pp. 49–60, June 2002.

[11] Sirish Chandrasekaran and Michael J. Franklin. Streaming queries over streaming data. In *Proceedings of the 28th International Conference on Very Large Data Bases*, pp. 203–214, August 2002.

[12] Mehul A. Shah, Joseph M. Hellerstein, Sirish Chandrasekaran, and Michael J. Franklin. Flux : An adaptive partitioning operator for continuous query systems. In *Proceedings of the 19th International Conference on Data Engineering*, pp. 25–36, March 2003.

[13] Ron Avnur and Joseph M. Hellerstein. Eddies : Continuously adaptive query processing. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pp. 261–272, May 2000.

[14] Tolga Urhan and Michael J. Franklin. Xjoin : A reactively-scheduled pipelined join operator. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, pp. 27–33, March 2000.

[15] Stratis D. Viglas, Jeffrey F. Naughton, and Josef Burger. Maximizing the output rate of multi-way join queries over streaming information sources. In *Proceedings of the*

29th International Conference on Very Large Data Bases, pp. 285–296, September 2003.

[16] Luping Ding, Elke A. Rundensteiner, and George T. Heineman. Mjoin: A metadata-aware stream join operator. In *Proceedings of the 2nd International Workshop on Distributed Event-Based Systems (DEBS)*, June 2003.

[17] Luping Ding and Elke A. Rundensteiner. Evaluating window joins over punctuated streams. In *Proceedings of the 13th ACM Conference on Information and Knowledge Management*, pp. 98–107, November 2004.

[18] Jaewoo Kang, Jeffrey F. Naughton, and Stratis D. Viglas. Evaluating window joins over unbounded streams. In *Proceedings of the 19th International Conference on Data Engineering*, pp. 341–352, March 2003.

[19] Lukasz Golab and M. Tamer Özsu. Processing sliding window multi-joins in continuous queries over data streams. In *Proceedings of 29th International Conference on Very Large Data Bases*, pp. 500–511, September 2003.

[20] Stratis D. Viglas and Jeffrey F. Naughton. Rate-based query optimization for streaming information sources. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pp. 37–48, June 2002.

[21] 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手法. 電子情報通信学会論文誌, Vol. J87-D1, No. 10, pp. 873–886, October 2004.

[22] 渡辺陽介, 北川博之. 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, March 2005.

[23] Demet Aksoy and Michael Franklin. RxW: A scheduling approach for large-scale on-demand data broadcast. *IEEE/ACM Transactions on Networking*, Vol. 7, No. 6, pp. 846–860, December 1999.

[24] Brian Babcock, Mayur Datar, and Rajeev Motwani. Load shedding techniques for data stream systems. In *Proceedings of the 2003 Workshop on Management and Processing of Data Streams*, June 2003. Available at http://www.research.att.com/conf/mpds2003/schedule/babcockDM.pdf.

[25] Nesime Tatbul, Ugur Çetintemel, Stanley B. Zdonik, Mitch Cherniack, and Michael Stonebraker. Load shedding in a data stream manager. In *Proceedings of the 29th International Conference on Very Large Data Bases*, pp. 309–320, September 2003.

[26] Abhinandan Das, Johannes Gehrke, and Mirek Riedewald. Approximate join processing over data streams. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pp. 40–51, June 2003.

[27] Brian Babcock, Shiva Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proceedings of the 21st ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp. 1–16, June 2002.