

ストリームデータ処理におけるデータ生存期間管理方式

今木 常之 西澤 格

株式会社 日立製作所 中央研究所 〒185-8601 東京都国分寺市東恋ヶ窪 1-280

E-mail: tsuneyuki.imaki.nn@hitachi.com, itaru.nishizawa.cw@hitachi.com

あらまし 近年, RFID タグ, センサノードなどを情報源として生成される高レート of データを, 継続的にリアルタイム処理するストリームデータ処理が重要性を増している. ストリームデータ処理においては, 無限に継続するデータの中から処理対象を限定するため, データに生存期間の概念が必要となる. データの生存期間を管理する方式としては Negative Tuple 法と Direct 法が提案されており, Direct 法は処理スループットが Negative Tuple 法より優れる一方, 実行可能な問合せが限定されていた. 本稿では, Direct 法で実行できる処理の範囲を Negative Tuple 法と同等に広げるためのデータ生存期間管理方式を提案する. 本方式により Direct 法で実行可能となった基本クエリに対して評価実験を実施し, 1.4~2.0 倍のスループット向上を確認した.

キーワード ストリーム, ウィンドウ, 連続問合せ, CQL

Data Lifetime Management Method in Stream Data Processing

Tsuneyuki IMAKI and Itaru NISHIZAWA

Hitachi, Ltd., Central Research Laboratory, 1-280 Higashi-Koigakubo, Kokubunji-shi, Tokyo,
185-8601 Japan

E-mail: tsuneyuki.imaki.nn@hitachi.com, itaru.nishizawa.cw@hitachi.com

Abstract Emerging RFID technology and sensor network systems produce huge amount of data, and modern enterprise data processing systems must handle the huge amount of data in the real-time fashion. Stream data processing, which continuously processes the data in the real-time fashion, has proposed and accepted as a new data processing paradigm. Data lifetime notion is essential in the stream data processing because it requires extracting the target data from infinite data sequences using the notion. Negative Tuple Method (NT) and Direct Method (DT) have been proposed to manage the data lifetime. Although DT outperforms NT in the system throughput, it cannot be applied to some basic query processing operators due to its processing mechanism. This paper proposes a new data lifetime management method which extends DT to enable applying it to the basic query processing operators as the same as NT. Moreover, we have confirmed by experiments that our new data lifetime management method outperforms NT by 1.4-2.0 times in the system throughput.

Keyword Stream, Window, Continuous Query, CQL

1. はじめに

近年, RFID タグの読取情報, センサノードの監視情報, ITS, プローブカーによる渋滞情報, 株価情報など, 絶え間なく連続的にデータを生成する情報源が増加している. これらのデータは即時性に意味がある場合が多いため, 瞬時かつ継続的に処理することが求められている. このような要求が増大する中, 高レートデータの連続処理を目的としたストリームデータ処理が注目されている.

ストリームデータ処理では, RDBMS で広く利用されてきた SQL に類似の言語で処理内容を定義する.

これにより, C++, Java などの言語でアプリケーションを構築する場合に比べて開発コストを大幅に削減できるという重要な特長がある[13]. 一方で, 連続したデータの処理を目的とした場合, SQL を言語拡張する必要が指摘されている[4]. 例えば SQL の計算モデルである関係代数は, 有限のデータ集合を演算対象とする

ため、無限に連続するデータから処理対象とを絞り込むことが不可欠となる。これに対し、連続データから時間範囲を区切ってデータを切り出す、スライディングウィンドウの概念が提案されており、既存のストリームデータ処理システムの多くで採用されている[3, 9, 10, 11]。スライディングウィンドウはデータの生存期間を定める演算と捉えることができ、生存期間が重なるデータ集合に対して関係演算を定義する。

ウィンドウサイズの代表的な指定方法として、定数時間で定める方法と、同時に生存するデータの数を定める方法の二つが挙げられる。本稿では、前者の方法で定めるウィンドウ演算を時間ウィンドウ、後者を行数ウィンドウと呼ぶ。時間ウィンドウにおいては、そのデータが生成された時刻（データの出現時刻）に指定された定数時間を加えることで、生存期間の終了時刻（データの消滅時刻）が定まるため、演算実行の際に直ちに生成されるデータの生存期間を確定できる。一方、行数ウィンドウでは、あるデータの消滅時刻は後続のデータ（ウィンドウサイズが N の場合、そのデータから N 個あとに到着するデータ）の出現時刻により定まるため、演算実行の際には消滅時刻を確定できない。

一般に、ストリームデータ処理は実行木によって実現される。この実行木は、ウィンドウ演算、およびその演算結果である生存期間付きデータに対する関係演算を実現する、演算子オブジェクトをノードに持つ。演算子の間はデータを授受するキューで連結されており、演算子群はパイプライン的に動作する。このような処理様式を、本稿ではパイプライン型処理と呼ぶ。同様式において、行数ウィンドウのように演算実行の際に消滅時刻が確定しない演算子を実現する方式として、**Negative Tuple 法**(以下 NT) が提案されている[5]。NT では、一つのデータをその出現を表すオブジェクトと消滅を表すオブジェクトの2つによって別々に表現する。この方式は、一つのデータについて二度の処理が必要であるという点が非効率的である。これを回避するため、一つのオブジェクトに出現時刻と消滅時刻を共に含める **Direct 法**（以下 DT）が提案されている[7]。

DT では、関係代数の基本 5 演算のうち選択、射影、および和集合の 3 演算の処理について、NT の倍のスループットが期待できる。一方で、データの出現・消滅の両時刻が揃っている必要

があるため、演算実行の際に消滅時刻が確定しない行数ウィンドウに対応することができなかった。これに対し本研究では、消滅時刻が未確定のデータを扱うための DT の拡張方式を提案する。さらに、提案方式がデータ生存期間を正しく管理できることを示す。

次節以降の構成は以下の通りである。まず2節で関連研究について述べる。3節で、我々が前提とするデータモデルおよびデータの生存期間の概念について説明する。4節で、提案する DT の拡張方式を説明する。5節で、提案方式の実装に対する評価実験の結果を示す。最後に6節でまとめと今後の課題を述べる。

2. 関連研究

ウィンドウの概念を持つストリームデータ処理の実行系については多くの研究が存在する。[12]は、結合演算にかかるコストの評価式を、ウィンドウサイズと関連付けて導出している。[6]は、複数のクエリで一つの結合処理の結果を共有する方法に関し、単一の結合結果を複数の異なるウィンドウで切り分ける処理の実現方式を示している。ストリームデータ処理における結合処理では、2つの異なるデータソースから生存期間が重なるデータを抽出する処理が必要となる。本研究では、生存期間が未確定のデータについても同様の抽出を正しく行なうための拡張を行なう。

[5]は、時間ウィンドウを含む実行木の構成法として、NT と **Time Probing 法**のハイブリッド方式を提案している。**Time Probing 法**は、各演算子が実行木上での子ノードとなる演算子に対して、処理が完了した時刻を確認し、自ノード上に保持するデータの中から消滅させて良いデータを確定する方式である。但し、子ノードの処理完了時刻から消滅時刻を計算するため、扱えるのは時間ウィンドウのみである。[7]は、NT と DT のハイブリッド方式を提案している。演算実行の際に消滅時刻が確定されない演算を実行木上で **pop up** し、確定される演算を **push down** し、ルート側を NT、リーフ側を DT で構成することで、DT の適用範囲を最大化する。但し、行数ウィンドウはリーフノードに位置しないとクエリの意味が変わる可能性があるため、方式[7]は適用できない。以上のように、[5, 7] 両方式ともに NT と他の方式とのハイブリッドであるが、どちらも行数ウィンドウを入力とする

クエリには適用できない．さらに，ハイブリッドを効果的に機能させるために，実行木の最適化が必要となる．この最適化処理は，本研究では行数ウィンドウを含む全ての演算を DT で実現するため，不要である．

[8]は，multi-way の結合演算の実現方式を示している．本方式には NT も DT も適用可能である．但し，ウィンドウ演算の結果が結合演算に直接入力されることが前提となるため，結合演算は実行木のリーフに存在する必要がある．本研究では，全ての演算を DT で実現するため，結合演算の実行木上での存在位置を自由に定めることが可能である．

3. データモデル

本研究では STREAM[3]における SQL 拡張言語である CQL[1, 2]を利用する．CQL では，時刻を離散的な数値（正の整数）として扱う．これに対し，本研究では時刻の精度を高める目的で，時刻を連続値として（本稿では，正の実数として）扱えるよう変更する．

3.1. ストリームとリレーシオンの生存期間

ストリームデータ処理で扱うデータモデルを定義する（以下，タイムスタンプと時刻は同義とする）．

ストリームタプル： n 項組データ値 v とタイムスタンプ t の組 $\langle v, t \rangle$ ．ストリームタプル s のデータ値を $v(s)$ ，タイムスタンプを $t(s)$ と表す．

ストリーム： ストリームタプルを要素とする有限で非有界の順序集合であり， S と表す．またその要素を s_i ($1 \leq i \leq |S|$ ($= S$ の要素数)) と表す． S 上の要素は $t(s_i) \leq t(s_j)$ ($i < j$) を満たす．

リレーション： n 項組データ値 v ，出現時刻 ta ，消滅時刻 te の組 $\langle v, ta, te \rangle$ ．リレーション r のデータ値を $v(r)$ ，出現時刻を $ta(r)$ ，消滅時刻を $te(r)$ と表す．

S の最初の m 要素からなる部分順序集合 $\{s_i \mid 1 \leq i \leq m \wedge s_i \in S\}$ を $S(m)$ と表す．また，あるタイムスタンプ $t_o \geq t(s_m)$ に対し， $t(s_{m+1}) < t_o$ となる $m+1$ が存在しない場合（ストリーム S において，ストリームタプル s_m の後，時刻 t_o より前には他のストリームタプルが到来しない場合）， t_o を S における m 延長時刻と呼ぶ．

ストリームデータ処理では，ストリームをウィンドウ演算によって，生存期間を持つリレーシオンの集合に変換することで，関係演算の対象となるデータを規定する．但し，リアルタイムの継続処理であるため，時刻 t_o においては部分ストリーム $S(m)$ がウィンドウ演算の対象となる．

ストリーム $S(m)$ を， m 延長時刻 t_o においてウィンドウ演算 w で変換した結果である **リレーション集合**を $R(w, t_o, S(m))$ と表し，以下のように定義する．

$$R(w, t_o, S(m)) = \{r \mid r = w(t_o, s) \wedge s \in S(m)\}$$

但し w は，一つのストリームタプル s を一つのリレーション r に変換する，以下のような演算である．

時間ウィンドウ： ストリームタプルを，特定の生存期間 T を持つリレーションに変換

$$w(t_o, s) = \langle v(s), t(s), t(s) + T \rangle$$

行数ウィンドウ： ストリームタプルを，その N 番後ろのストリームタプルのタイムスタンプを消滅時刻とするリレーションに変換

$$w(t_o, s_i) = \langle v(s_i), t(s_i), t(s_{i+N}) \rangle \quad (1 \leq i \leq m-N)$$

$$w(t_o, s_i) = \langle v(s_i), t(s_i), nft(t_o) \rangle$$

$$(m-N+1 \leq i \leq m)$$

ここで， $nft(t_o)$ は時刻 t_o で消滅時刻が未確定であることを表す．なお，以降では t_o を延長時刻とする任意の m について， $R(w, t_o, S(m))$ を単に $R(t_o)$ と表す．

リレーション集合 $R(t_o)$ に対し，時刻 $0 < t < t_o$ における **n 項組データ値集合**を $Rv(t, R(t_o))$ （或いは，単に $Rv(t, R)$ ，または $Rv(t)$ ）と表し，以下のように定義する．

$$Rv(t, R(t_o)) = \{v(r) \mid r \in R(t_o) \wedge ta(r) \leq t \wedge t < te(r)\} \quad (0 < t < t_o)$$

以上の直感的な意味付けを図 1 に示す．ストリームタプルは時間軸上における点データであり，ウィンドウ演算により線分であるリレーションとなる．時刻 t に交わるリレーション r のデータ値 $v(r)$ の集合が $Rv(t)$ となる． $Rv(t)$ の定義より，リレーションの生存期間は左閉右開区間となる．即ち，出現時刻は生存期間に含み，消滅時刻は含まない．

3.2. リレーション集合の関係演算

n 項組データ値集合に対して定義される従来型の関係演算を Q としたとき、リレーション集合 $R_1(t_0)$ と $R_2(t_0)$ の間における関係演算を、以下のようなリレーション集合 $Rq(Q, R_1, R_2, t_0)$ を結果とする演算として定義する。

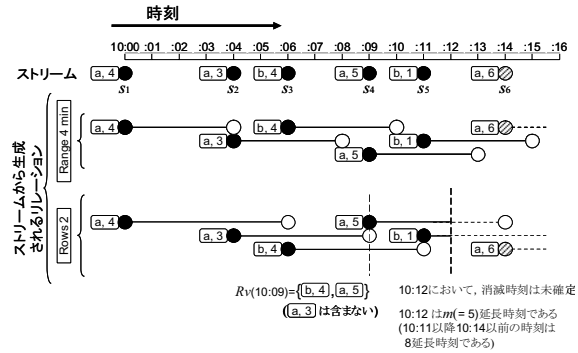


図 1: ウィンドウ演算によるリレーションの生成

時刻 $0 < t < t_0$ における n 項組データ値集合 $Rv(t, R_1(t_0))$ と $Rv(t, R_2(t_0))$ に対する関係演算 Q の結果である n 項組データ値集合を $Q(t, R_1, R_2)$ と表すと、全ての時刻 $0 < t < t_0$ において以下が成り立つ

$$Rv(t, Rq(Q, R_1, R_2, t_0)) = Q(t, R_1, R_2)$$

関係演算 Q が単項演算である場合も同様に定義できる。このようなリレーション集合 Rq は一意には定まらないが、ここで定義した関係演算においては、その何れも全て等価として扱える (3.3節参照)

3.3. リレーション集合の等価性

2 つのリレーション集合 $R(t_0)$ と $R'(t_0)$ に対し、全ての時刻 $0 < t < t_0$ において $Rv(t, R(t_0)) = Rv(t, R'(t_0))$ が成り立つならば、 $R(t_0)$ と $R'(t_0)$ はリレーション集合として**等価**であると呼ぶ。

前節で示した関係演算の定義より、

$$Rv(t, Rq(Q, R, t_0)) = Q(t, R) \quad (0 < t < t_0)$$

$$Rv(t, Rq(Q, R', t_0)) = Q(t, R') \quad (0 < t < t_0)$$

である。一方、 $R(t_0)$ と $R'(t_0)$ が等価であるならば

$$Rv(t, R(t_0)) = Rv(t, R'(t_0)) \quad (0 < t < t_0)$$

であるので

$$Q(t, R) = Q(t, R') \quad (0 < t < t_0)$$

が成り立つ。

従って、

$$Rv(t, Rq(Q, R, t_0)) = Rv(t, Rq(Q, R', t_0))$$

$$(0 < t < t_0)$$

が成り立つので、 $Rq(Q, R, t_0)$ と $Rq(Q, R', t_0)$ は等価である。即ち、関係演算においては、入力が等価であれば結果も等価であることが言える。

また、 $ta(r) = te(r)$ (生存期間 0) であるようなリレーション r は、 n 項組データ値集合の定義より、いかなる $0 < t < t_0$ における $Rv(t, R(t_0))$ にも含まれない。従って、リレーション集合の等価性に影響を与えない。本稿では、このようなリレーションを**ゴースト**と呼ぶ。

4. データ生存期間管理の提案手法

本研究で提案する DT の拡張方式を説明し、同方式が3節に示したデータモデルに従うことを示す。

4.1. 未確定消滅時刻の再評価法

1節に示したように DT の動作は、各演算子ノードが演算実行の際に、一つの結果リレーション毎にそれを表現する一つのオブジェクト (リレーションタプル) を生成し、実行木上で親の演算ノードにキューを介して送出するという動作をパイプライン的に繰り返す。リレーションタプルは、3.1節に示したリレーションの属性、 n 項組データ値、出現時刻および消滅時刻を保持する。但し、行数ウィンドウのように演算実行の際に結果リレーションの消滅時刻が確定しない演算については、具体的な時刻の値を付加することができない。SUM, AVG, INTERSECT, EXCEPT などこのような演算に分類される。

上記に分類される演算子 A について、演算実行の際には結果リレーションの消滅時刻が未確定でも、A がさらに別のタプルを処理することで確定する可能性がある。そこで、結果として送出するリレーションタプルに付加する消滅時刻として、具体値の代わりに**消滅時刻ホルダ**を付加しておく。これは、のちにリレーションの消滅時刻が確定したタイミングでその時刻を代入することが可能なホルダオブジェクトである。

一方、演算子 A の親ノードの演算子 B では、受理したリレーションタプルに消滅時刻ホルダが付いている場合は、同タプルを演算子 B の消

滅時刻未確定集合に保持し、消滅時刻の確定状況を監視する Observer を割当てて、のちに演算子 A によってホルダに確定時刻が代入された際には、同 Observer がコールバックされるので、その処理においてリレーションタプルを演算子 B の **消滅時刻確定集合**に移動する。

例として図 2では、状態 1 において演算子 A (行数ウィンドウ) にストリームタプル 1 が入力され、その結果であるリレーション 1 を演算子 B が消滅時刻未確定集合に格納し、さらに状態 2 において、ストリームタプル 2 が入力されてリレーション 1 の消滅時刻が確定する様子を示している。

但し演算子 B、が入力リレーションに対して高々一つの結果リレーションしか生成せず、生存期間が入力と結果で全く同じであるならば、入力の消滅時刻が未確定であっても、消滅時刻ホルダへの参照を出力リレーションにコピーするのみで構わない。選択、射影、和集合の 3 つがこのような演算に分類される。

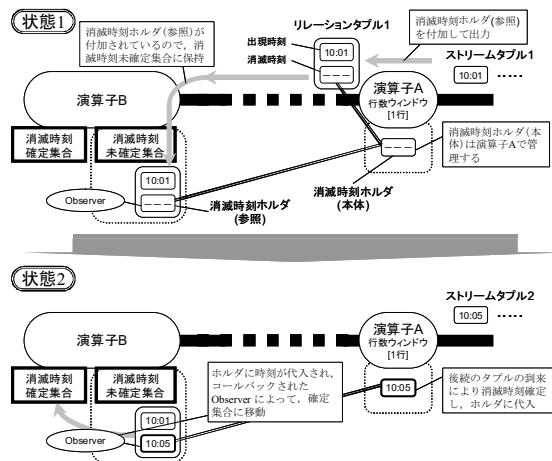


図 2: 消滅時刻ホルダによる消滅時刻の確定

4.2. 消滅時刻順の遅延整列法

消滅時刻確定集合にあるリレーションタプルのうち、入力リレーションタプルの出現時刻 (以下、入力時刻) より早い消滅時刻を持つタプル群は、当該入力タプルを処理する前にその演算子での処理対象から除く必要がある。DT では、この操作を効率的に行うために、消滅時刻確定集合にあるタプルを消滅時刻順に整列しておく。この整列を単純な挿入ソートで行うと、タプルを入力する度に集合のサイズに比例するコストがかかる。

一方、入力時刻以降の消滅時刻を持つタプルは処理対象として残るので、整列されている必要はない。このことを利用し、整列動作を必要になるまで遅延させる方式を提案する。具体的には以下のように動作する。

消滅時刻確定集合を整列済み集合と非整列集合に分ける。消滅時刻確定集合に移動されたタプルは非整列集合に保存し、同集合内で最早の消滅時刻を記憶しておく。入力時刻が同時刻を超えた時にはじめて整列し、整列済み集合とマージする。

4.3. 提案方式の正当性

以上説明した動作において、消滅時刻未確定のタプルは無条件に処理対象として残すことになる。このような処理の正当性は以下のように説明できる。

演算子 A が演算子 B の孫ノードであるとすると、A で生成したタプル T が消滅時刻未確定で B に保持されており、その時点での B の入力時刻を tb とする。一方、A がその時点で処理を完了している最新のタプルの出現時刻を ta とする。A がその時点で T の消滅時刻を確定できていないということは、T の消滅時刻が定まるとすれば、その時刻 tx について $tx \geq ta$ が成り立つ。一方、パイプライン型処理に従えば、B は A より後段に処理される演算子なので、 $ta \geq tb$ が成り立つ。従って $tx \geq tb$ が成り立つため、タプル T は演算子 B において、その入力タプルと生存期間が重なり、同時に処理対象となることが保証される。

但し $tx = tb$ である場合は余分なゴーストが生成されることになるが、4.1節で示したように結果の等価性には影響しないので、方式の正当性は保たれる。

5. 実験

未確定消滅時刻の再評価法の効果を確認するため、行数ウィンドウを入力とする選択、射影、和集合、結合の各演算について NT との比較評価を行なう。

5.1. 実験内容

図 3 に示すスキーマのストリーム STRu, STRb0, STRb1, および 4 つの基本クエリによってマイクロベンチマークを実施した。

ここで、WINu (単項演算のウィンドウサイズ指定) は、行数ウィンドウ (10 行～1,000,000 行) と時間ウィンドウ (10 msec ～ 1,000 sec) で変化させた。WINb (二項演算のウィンドウサイズ指定) は、行数ウィンドウ (5 行～500,000 行) と時間ウィンドウ (5 msec ～ 500 sec) で変化させた。また、データは図 4に示すストリームタブルの系列とした。

ストリームのスキーマ

STRu : (ca varchar, cb int, cc varchar)
 STRb0: (ca int, cb varchar, cc int)
 STRb1: (ca int, cb varchar, cc int)

クエリ

選択 select * from STRu[WINu] where cb > 3
 射影 select ca, cb from STRu[WINu]
 和集合 select * from STRb0[WINb]
 union all
 select * from STRb1[WINb]
 結合 select * from STRb0[WINb],
 STRb1[WINb]
 where STRb0.ca = STRb1.ca

図 3: 実験に用いたスキーマ及びクエリ

STRb0	STRb1
TimeStamp : ca, cb, cc	
14390: (532, s0, 0)	14390: (532, s1, 1)
14391: (533, s0, 0)	14391: (533, s1, 1)
14392: (534, s0, 0)	14392: (534, s1, 1)
14393: (535, s0, 0)	14393: (535, s1, 1)
14394: (536, s0, 0)	14394: (536, s1, 1)
14395: (537, s0, 0)	14395: (537, s1, 1)

図 4: テストデータ

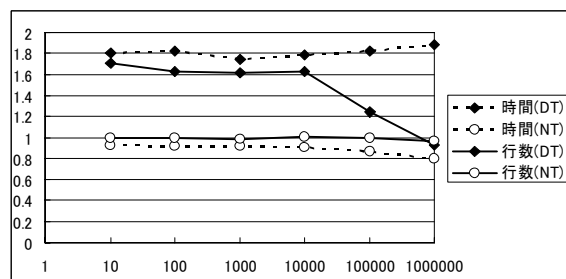
ストリームタブルには 1 msec 間隔のタイムスタンプを付けており、カラム STRb0.ca およびカラム STRb1.ca は 1 ずつ増える順番で並ぶ。STRb0 と STRb1 のストリームタブルが揃って流れるため、結合処理のクエリではウィンドウサイズに係わらず二つのタブルから一つの結果が生成される。また本実験の設定では 1 msec あたり 1 タブルがシステムに到来することになるので、N msec の時間ウィンドウは、N 行の行数ウィンドウと同じく常に N 個のタブルを切り出す。

実験は、Java を実装言語として、1GB のメモリを搭載した計算機にて VM に 756MB のメモリを割り当てて実行した。

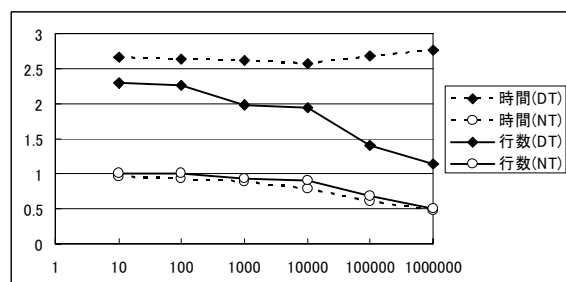
5.2. 測定結果

図 5に、NT と DT の性能比較をクエリ別に示す。縦軸はスループット、横軸はウィンドウが切出すタブル数を表す (例えば横軸 1,000 は、行数ウィンドウでは 1,000 行、時間ウィンドウでは 1 sec のウィンドウサイズでの結果。二項演算では両入力ウィンドウを合計したタブル数)。なお、スループットは 10 行の行数ウィンドウでの結果を 1 とした相対値で示す。

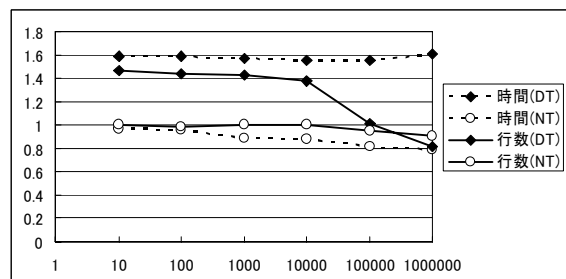
[選択]



[射影]



[和集合]



[結合]

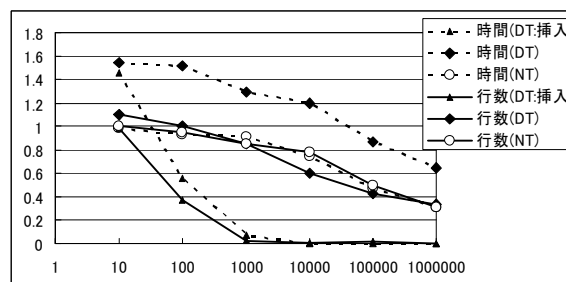


図 5: 基本クエリのスループット(相対値)

選択クエリ，射影クエリ，和集合クエリにおいて，10,000 行までの行数ウィンドウについては，DT は NT に対し 1.4 倍～2.0 倍のスループット改善の効果があった．一方，ウィンドウサイズが 10,000 行を超えると急激に性能が低下し，1,000,000 行においては DT と NT の性能はほぼ等しくなった．この原因を調べるため，選択クエリと和集合クエリに関し，ウィンドウサイズ 10,000 行とウィンドウサイズ 1,000,000 行の処理について Java VM の GC ログを取得し，全体処理時間に対し GC が占める割合を調べた．その結果，ウィンドウサイズ 10,000 行の処理においては，選択クエリで 1.1%，和集合クエリで 1.9%と非常に小さい割合であったが，1,000,000 行の処理では，選択クエリで 14.4%，和集合クエリで 21.5%と GC のコストが大幅に増大していることが確認された．また，10,000,000 行ではメモリアーバとなり実行不可能であった．

これに対し，時間ウィンドウではこのような傾向は確認されず性能が安定していた．また，NT での 1,000,000 行の処理における GC のコストも，選択クエリで 4.0%，和集合クエリで 8.8%であり，DT に比べて小さく性能も安定していた．このことから，行数ウィンドウで発生する消滅時刻ホルダの生成・回収コストが大きいことが予想される．本研究の提案手法は空間コストの観点で改善の余地があることが示された．但し，1,000,000 行を超えるウィンドウの応用における必要性についても併せて検討する．

結合クエリにおいては，時間ウィンドウで DT が NT に対して 1.6～2 倍程度の性能向上を示した．一方，行数ウィンドウについては，全てのウィンドウサイズに渡って DT と NT の性能はほぼ同じであった．時間ウィンドウの処理では，結合演算の結果リレーションの生存期間が演算実行の際に確定するため，消滅時刻が確定した一つのリレーシオンタプルを生成するだけで済む．一方，行数ウィンドウにおいては，消滅時刻ホルダを生成・付加するオーバーヘッドがかかる．このコストが，NT でのウィンドウ演算子におけるタプル管理コストと同程度になっていると予想される．

また，遅延整列を行なわない場合の結果（グラフ上での“DT:挿入”のライン）は，ウィンドウのタプル切り出し数が 100 の段階から性能が急激に悪化した．これに対し，遅延整列を行なう DT は NT と同じ性能の傾向を維持しており，遅

延整列の効果が確認された．

6. おわりに

本稿では，ストリームデータ処理におけるデータ生存期間管理の効率化手法として，消滅時刻が未確定であるデータの再評価方式を提案した．同方式により，行数ウィンドウを入力とする選択，射影，和集合演算の処理スループットの改善が可能となった．評価実験により，提案方式では従来の NT と比較して 1.4～2.0 倍のスループット改善効果が得られることを確認した．さらに，行数ウィンドウを入力とする結合演算においても，従来方式と同程度の性能を維持することを確認した．これにより，NT の性能改善方式として DT を単独で利用する可能性を示した．

今後の課題として，ウィンドウサイズが非常に大きい場合のメモリ管理コスト削減が挙げられる．

文 献

- [1] A.Arasu, S.Babu, and J.Widom, “CQL: A Language for Continuous Queries over Streams and Relations”, Proc. DBPL 2003, pp. 1-19, Potsdam, Germany, Sep.2003.
- [2] A.Arasu, S.Babu, and J.Widom, “The CQL Continuous Query Language: Semantic Foundations and Query Execution”, Technical Report, Stanford University, 2003.
- [3] A.Arasu, B.Babcock, S.Babu, J.Cieslewicz, M.Datar, K.Ito, R.Motwani, U.Srivastava, and J.Widom, “STREAM: The Stanford Data Stream Management System”, Technical Report, Stanford University, 2004.
- [4] Y. Law, H. Wang, and C. Zaniolo, “Query Languages and Data Models for Database Sequences and Data Streams”, Proc. VLDB 2004, pp. 492-503, Toronto, Canada, Aug.2004.
- [5] M.Hammad, W.Aref, M.Franklin, M.Mokbel, and A.Elmagarmid, “Efficient Execution of Sliding-Window Queries Over Data Streams”, Technical Report, Purdue University, 2003.
- [6] M.Hammad, M.Franklin, W.Aref, and A.Elmagarmid, “Scheduling for shared window joins over data streams”, Proc. VLDB 2003, pp. 297-308, Berlin, Germany, Sep.2003.
- [7] L.Golab, M.Özsu, “Update-Pattern-Aware Modeling and Processing of Continuous Queries”, Proc. SIGMOD 2005, pp. 658-669, Baltimore, Maryland, Jun.2005.
- [8] L.Golab, M.Özsu, “Processing Sliding Window Multi-Joins in Continuous Queries over Data Streams”, Proc. VLDB 2003, pp. 500-511, Berlin, Germany, Sep.2003.

- [9] C.Cranor, T.Johnson, O.Spatscheck, V.Shkapenyuk, "Gigascope: A Stream Database for Network Applications", Proc. SIGMOD 2003, pp. 647-651, San Diego, California, Jun.2003
- [10] D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik, "Aurora: a new model and architecture for data stream management", The VLDB Journal, vol.12, no.2, pp.120-139, Aug.2003.
- [11] S.Madden, M.Franklin, J.Hellerstein, W.Hong, "The Design of an Acquisitional Query Processor For Sensor Networks", Proc. SIGMOD 2003, pp. 491-502, San Diego, California, Jun.2003.
- [12] J. Kang, J. Naughton, and S. Viglas, "Evaluating Window Joins over Unbounded Streams", Proc. ICDE 2003, pp.341-352, Bangalore, India, Mar.2003.
- [13] M. Stonebraker, U. Cetintemel, S. Zdonik, "The 8 Requirements of Real-Time Stream Processing", SIGMOD Record, vol.34, no.4, pp.42-47, Dec.2005.