DBSJ Journal, Vol.8, No.4
March 2010

遅延演算を利用したストリームデータの再帰処理方法

Recursive Processing on Stream Data with Delay Operators

今木 常之♠　樫山 俊彦♠　西澤 格♠

Tsuneyuki IMAKI　Toshihiko KASHIYAMA
Itaru NISHIZAWA

株自動取引，車両位置情報，携帯電話操作ログなど，膨大な情報流を継続的に解析し，リアルタイムに社会行動に結びつける技術としてストリームデータ処理が注目されている．例えば株取引では投資行動は株価情報のみならず資金も鑑みて選択する必要があり，資金は投資行動自体によって変化する．このように，行動の選択に際しては行動自体に起因する行動者の状態変化を迅速に反映する必要がある．一方，このような再帰的処理は自己参照の矛盾を含むため，単純に相互再帰のクエリを定義するのみでは実行不可能である．本稿では，遅延演算を利用した自己参照の解消法，および遅延演算挿入位置を考慮したデータ処理順制御による，ストリームデータの再帰処理実現方法を提案する．

Automated trading system, car location system, and cell-phone system produce huge amount of time series data. Stream data processing is expected as a new technology which analyzes the data continuously to extract a value in social activity. In automated trading, a decision making must be performed not only on stock price information but also on the monetary resources of the investor. The resources could also be changed along with the investment behavior itself. In this manner, an actor must select the best action based on its state which the action itself will modify. Such behavior involves a self-reference which causes a paradox on the query languages of stream data processing. We propose an execution control method by inserting delay operators into recursive queries to resolve the paradox.

1. はじめに

株自動取引，携帯操作，センサデータなど，社会活動で継続的に発生する様々な情報がネットワークを流れ，その量は爆発的に増大している．これらの情報流から有意義なイベントの発生をいち早く抽出し利活用する技術として，ストリームデータ処理（以下，ストリーム処理）が注目されている．

ストリーム処理では関係データベースのクエリ言語SQLと同様の宣言型言語によりデータ処理を定義できるため，アプリケーションの開発が容易である点が一つの特長である[1]. ストリーム処理が提供するクエリ言語の多くはSQLの概念に，無限に続くデータ列から演算対象を切り出すウィンドウ演算を追加した形態をとっている[2][3][4][5]．ストリーム処理のユーザはこの演算を活用することで，イベント間の発生間隔，因果関係，統計値の時間変化といった，時間の流れに基くデータ処理を簡潔に記述できる．我々は，米Stanford大で提案された言語CQL（Continuous Query Language）[6]を採用している．CQLは関係演算の結果集合における変化分のみを正規化するストリーム化演算を備える．この演算は結果の一意性を保証しつつ意味のある変化のみを抽出する，あるいは後続の処理データ数を削減するといった効果を持つ．

ストリーム処理の有望な適用先として，株価の瞬間的な変化を捉えて発注を繰り返す株自動取引が挙げられる．発注の決定には資金や保有株などの情報も考慮する必要がある一方で，この情報は発注行為自体によって変化する．このような自己参照型の処理は，再帰的なクエリ定義が必要となる．

ストリーム処理において再帰クエリを実現するためには，以下に示す二つの課題が存在する．

- ウィンドウ演算を含む再帰的クエリ定義の論理的矛盾
- ハートビートを用いた時刻順制御におけるデッドロック

一つ目は，SQLとウィンドウ演算を組合せたデータ処理言語の意味論，即ちストリーム処理の概念自体に関わる課題である．二つ目は，ストリーム処理の一般的な実現方法であるハートビートを用いた時刻順制御，即ち実装の課題である．我々は前者に対して遅延演算を利用した矛盾解消法を提案する．また，後者に対して遅延演算の挿入位置に基くオペレータ処理順に従った時刻別実行法を提案する．方式の性能評価により再帰処理への適用可能性を確認した．

2. ストリームにおける再帰処理の効果と課題

事例に基いて再帰処理の効果を2.1節に示し，実現の課題を2.2節および2.3節に示す．

2.1 再帰処理の例および効果

株取引における投資行動は株価情報のみならず資金や保有株といった情報も判断材料にして決定する必要がある．株価情報が外部からのデータであるのに対し，資金や保有株の情報は内部で保持すべき状態である．発注行為による資金や保有株の増減はその後の投資判断に影響を持つため，状態変化は発注行為の発生と同時に反映する必要がある．即ち，投資判断の材料となった状態を投資行為自体の影響により変更する処理であり，再帰処理を構成することになる．

ストリーム処理では状態の保持にウィンドウ演算を用いる．従って上記のような再帰処理は，あるウィンドウ演算が保持する値に基いて生成された結果が原因となって，その値自身に変化が生じた場合に，変化後の値をそのウィンドウ演算への入力として戻す処理と言い換えることができる．

図 1 に簡略化した株自動取引処理のクエリ定義[1]を示す．この処理は以下のスキーマの3つのストリームを入力にとる．

株価(market)：　{銘柄番号(stock_id)，価格(price)}
初期資金(initial_resource)：　{金額(val)}
保有株(stock_stream)：
　　{銘柄番号(id)，株数(num)，買値(price)}

♠ 正会員　株式会社日立製作所中央研究所
{tsuneyuki.imaki.nn, toshihiko.kashiyama.ez, itaru.nishizawa.cw}@hitachi.com

[1] ストリーム化演算（ISTREAM）の記法は，米 Stanford 大より公開されている BNF（http://infolab.stanford.edu/stream/code/cql-spec.txt）に従っている．括弧内のクエリの結果を，[6]における Relation-to-Stream Operator の引数であるリレーション R にとった場合と等価である．

```
REGISTER QUERY buy_event
ISTREAM(
  SELECT stock.id, 1000 AS num, market.price
    FROM stock, resource, market [Now]
   WHERE stock.id = market.stock_id
     AND stock.num = 0
     AND market.price < 500
     AND resource.value > market.price * 1000)

REGISTER QUERY resource
SELECT * FROM resource_stream [Rows 1]

REGISTER QUERY resource_stream
ISTREAM(
  SELECT * FROM initial_resource [Now]
 UNION ALL
  SELECT resource.val
         - buy_event.price * buy_event.num AS val
    FROM resource, buy_event [Now])

REGISTER QUERY stock
SELECT * FROM stock_stream
        [Partition By stock_stream.id Rows 1]
```

図 1　株自動取引クエリの例
Fig.1　A Query Example of Automated Trading

取引処理全体は buy_event, resource, resource_stream, および stock の 4 つのクエリで構成される．クエリ stock と resource はそれぞれ保有株情報と資金額を保持するウィンドウ演算である．クエリ buy_event はこれらの状態と market ストリームの情報に基いて買い注文を出す投資ロジックである．ここでは，ある銘柄について保有株数が 0 で，価格が 500 を下回り，かつ千株買う資金があるならば購入するという単純なルールを想定する．クエリ resource_stream は，resource として保持された資金状態に対し発注に伴う資金の減少を反映させる．クエリ buy_event の出力がクエリ resource_stream を通ってクエリ resouce にフィードバックされており，全体として再帰処理を構成している．

ストリーム処理の内部で再帰処理をせず，クエリ resouce_stream の結果をストリーム処理の外部に位置するトランザクションシステムやデータベースに一旦出力し，クエリ resource に再入力するようにアプリケーションを組むことで，同様の処理を実現することは可能である．しかしながら，外部から再入力するまでのタイムラグにおいて，変化が反映される前の資金に照らして購買条件を満たす別の銘柄のデータが到来すると，結果的に資金を上回る余計な注文を出してしまう可能性がある．一方で再入力までストリーム処理を停止すれば，本来は可能であった他の注文の遅れにつながる．これに対し本例のように再帰処理を用いれば，処理の安全性と即時性を共に保証することが可能となる．

ストリーム処理の対象は，本例のような高レートデータのリアルタイム処理であるため，同様の効果が多くの応用に共通すると考えられる．

2.2 再帰処理の論理的矛盾

CQLの意味論において，**図**1に示したクエリは論理的矛盾を含んでいるため実際には実行不可能である．このことを，**図**2と**図**3を用いて示す．

図2は，**図**1に示したクエリのクエリグラフであり，角丸破線は各クエリに対応する部分グラフである．長円，角丸四角，および上向き五角形はオペレータであり，それぞれ関係演算，ウィンドウ演算，およびストリーム化演算を表す．台形はデータの入口となるscanオペレータである．オペレータ間の線はデータキューを示し，細線はリレーション，太線はストリームが流れる．データは図の下から上に流れるが，二本のキューは上から下に向かって閉ループを形成している．

また，**図**3は同クエリを実行した場合の，ストリームおよびリレーションにおけるデータの発生タイミングを表すタイムチャートである．左から右に向かって時間軸を表し，上から下の順にオペレータによってデータが処理される様子を表す．黒丸を起点とする線分はデータの生存期間を表す．白丸は生存期間の終了を表し，終了時刻丁度は生存期間に含めない．また，ストリーム（入力marketやストリーム化演算の結果buy_eventなど）は点データである[9]．右端に並んだ下向き矢印はオペレータを表し，　**図**2のA～Hを付したオペレータに対応する．各オペレータは，直上に位置するデータを処理して直下のデータを生成する．但し二項演算joinは，直上のデータと黒四角で指すデータの組合せを処理する．

時刻 t_0 にデータ{3000000}がストリームresource_streamに入ると，個数ウィンドウ[Rows 1]によって時刻 t_0 から生存開始するデータがリレーションresourceに生成される．その後，時刻 t_1 にデータ{a, 480}がストリームmarketに入り，時間ウィンドウ[Now]によって生存期間ε（物理的な時間単位に満たない仮想時間）が与えられる．[Now]は瞬間的なデータを関係演算の対象に変換する役割を持つ．ここでクエリbuy_eventの発注条件が充たされてデータ{a, 1000, 480}が生成され，クエリresource_stream は減額後の資金データ{2520000}を生成する．これによってリレーションresourceのデータ{3000000}が時刻 t_1 に消滅するのが矛盾となる．

図3の点線は，データ{2520000}発生以降のデータ消滅の様子を示している．resourceのデータ{3000000}が時刻 t_1 において存在しないことになるため，クエリbuy_eventの結果データ{a, 1000, 480}も存在せず，データ{2520000}も発生しないことになる．その結果resourceのデータ{3000000}は時刻 t_1 において存在することになり…以降無限に繰り返す．この動作は，自然言語で「時刻 t_1 における資金は3000000なので，同時刻における資金は2520000である．」と表現され，直感的な解釈においても論理的矛盾を含んでいる．

このような矛盾は，個数ウィンドウ演算の非単調性に起因する．ここでデータ集合Sと時刻付きデータ集合T からデータ集合R への関数F が以下の条件を充たすとき，Fを単調な演算，それ以外の場合を非単調な演算と呼ぶ[2].

$$S_1 \subset S_2, F(T, S_1) = R_1, F(T, S_2) = R_2 \quad \Rightarrow \quad R_1 \subseteq R_2$$

この定義に基き，CQLの各演算は表1のように分類される．非単調な演算が再帰処理に含まれる場合に矛盾が発生する．

表 1　演算の単調性
Table 1　Monotonicity of　the query operators

演算種別	単調性	演算
関係演算	単調	Filter, Join, Union, Projection,など
	非単調	集約演算(Avg, Sumなど), Exceptなど
ウィンドウ演算	単調	時間ウィンドウ
	非単調	個数ウィンドウ
ストリーム化演算	単調	IStream, RStream
	非単調	DStream

[2] ウィンドウ演算については，S はある時刻 t における時間軸上の点データの集合，R は時刻 t において出力リレーション上に生存するデータの集合である．T は時刻 t より前に入力した全データの集合を意味する．また，ストリーム化演算については，S はある時刻 t において生存する入力データの集合，R は時刻 t における出力データの集合である．T は時刻 t の直前に生存した入力データの集合を意味する．また，関係演算において，S と R は時刻 t において生存するデータの集合であり，T は空集合である．

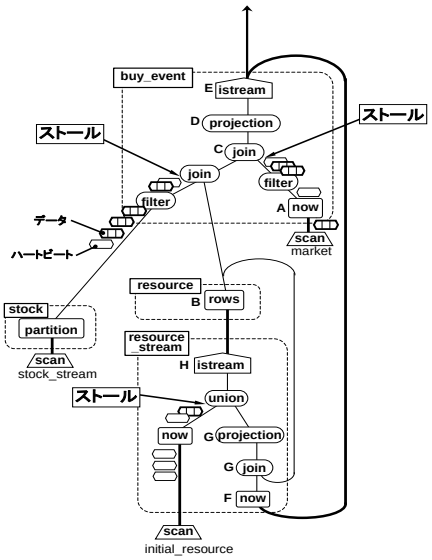

図 2　クエリグラフの例と二項演算におけるストール
Fig.2　A Query Graph and Stall at Binary Operators

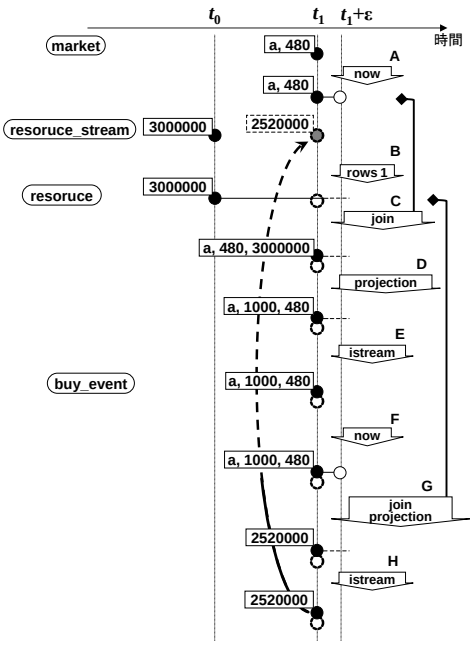

図 3　ウィンドウ状態の振動
Fig.3　Oscillation on the Window State

2.3 再帰処理の実行制御におけるストール

ストリーム処理では，複数のオペレータをキューで接続して構成したクエリグラフを用いて，パイプライン的にデータを処理する．ここで，データのタイムスタンプに基く処理順序を厳守するために，結合処理や集合演算などの二項オペレータにおいては，左右入力キューの先頭データを比較してタイムスタンプの早い方を先に処理するように制御する．このような制御方法において，片方のキューのみデータの到来が滞り，処理が進められない状態をストールと呼ぶ．これを回避するために，時刻の進行を知らせるハートビートをデータの入力から定期的に送信するハートビート法（以下HB法）[2][5]が広く利用されている．

一方，HB法での再帰処理の実現には課題があることを図2に示す．ハートビートはscanオペレータが入力データに混ぜて送信する．キュー上に位置する複数の六角形はscanオペレータから送出されたデータ（太線枠），またはハートビート（細線枠で中が空白）を表している．これらはグラフの閉ループ上に位置する3箇所の二項演算において，以下の理由でストールする．二項演算の処理は上述のように両入力にデータが揃わない限り進めることができない．一方，これらのオペレータが処理結果を出力しない限り，ループ内をデータやハートビートが流れることはない．その結果，これら二項演算の両入力は永久に揃わないことになるためストールする，いわゆるデッドロック状態に陥る．

3. 遅延演算を利用した再帰処理の実現方法

2章に示した課題を解決する遅延演算を利用した再帰処理の実現方法を提案する．3.1節では2.2節に示した論理矛盾の解決策として遅延演算を導入し，3.2節で2.3節に示した課題の解決策としてこの遅延演算に基く実行制御法を提案する．

3.1 遅延演算の挿入による論理的矛盾の解消

2.2節に示した論理矛盾は，ある時刻においてあるデータが存在することによって，その時刻にそのデータが消滅する，その結果，その時刻にそのデータが存在する…，といった関係が無限に続く構造になっていた．この問題は，データの存在と消滅がお互いに因果関係を持つのに対し，これを同時の事象として捉えることに起因する．この関係を解消する直接的な方法は，データの存在と消滅を異なる時刻の事象と捉えることである．そこで，ストリーム上のデータのタイムスタンプを指定時間だけ未来にずらす遅延演算を導入し，再帰クエリを定義する際には，クエリグラフの閉ループ中にこの演算が挿入されていることを制約として設けた[3]．

クエリ定義において遅延演算を挿入する記法は，下記クエリの下線部のように，ストリーム化演算が施されるクエリの末尾に“<”と“>”で括って任意の遅延時間を記入する．

```
REGISTER QUERY resource_stream
ISTREAM(SELECT * FROM initial_resource [Now]
 UNION ALL
  SELECT resource.val
         - buy_event.price * buy_event.num AS val
    FROM resource, buy_event [Now]
)<Now>
```

図4は，図3のタイムチャートの最後に遅延演算Delayを挿入した場合を示している（遅延幅は時間ウィンドウ[Now]と同じεとしている）．リレーションresourceにおける時刻t_1のデータ{3000000}によって，ストリームresource_stream上に時刻t_1+εのデータ{2520000}が発生する．ここで，リレーションresourceのデータが{3000000}から{2520000}に更新されるのは時刻t_1 +εとなるので，時刻t_1におけるデータ{3000000}は矛盾なく存在することになる．

なお，遅延演算はストリーム上の演算として定義し，リレーションの生存期間をずらす演算は定義しない．CQLの意味論においてリレーションに対する関係演算は，ある時刻に

[3] 本制約に違反するクエリに対しては，2.2 節に示した[Now]の生存期間と同じ ε の遅延演算を，クエリグラフ中に存在するストリーム化演算の任意の一つの出力位置に自動で挿入する，あるいはクエリ定義エラーとする．

おけるデータ集合に対する静的な関係（言い換えれば時刻に依存しない関係）を規定するものとして定義されており[6][9], 遅延演算はその性質を満たさないためである．また，実用上もストリーム上の遅延演算のみで十分である．

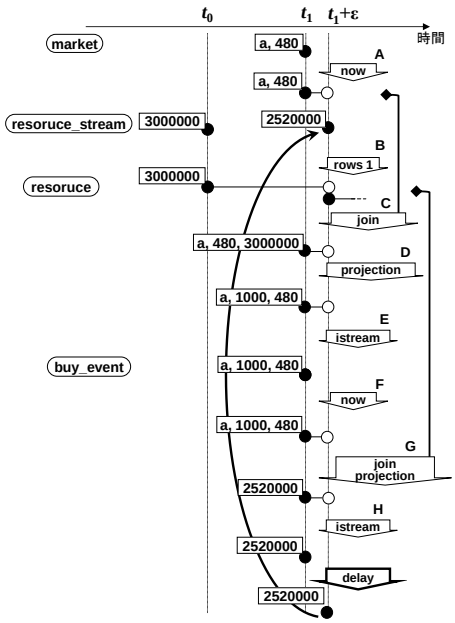

図 4　遅延演算を挿入した場合のタイムチャート
Fig.4　Time Chart with a Delay Operator

3.2 遅延演算を始点とする順序に基く時刻別実行法

2.3節に示した二項演算のストールは，あるオペレータの実行時に一方の入力キューが空の場合，他方のキューの先頭タプルよりも早い時刻のタプルがその空キューに到来する可能性を考慮する必要性により発生する．この問題に対応するため，時刻毎にクエリグラフ上の全オペレータの処理を完結することでこの可能性を排斥する，時刻別実行法を用いる．

時刻別に実行するためには，クエリグラフ上でのオペレータ間の入出力関係において出力側よりも入力側のオペレータを必ず先に実行する必要がある．例えば，ストリーム化演算では，ある時刻において生存する入力データの集合が確定しなければ，その時刻の出力データの集合も確定しない[6][9]．このような問題を回避するためにはクエリグラフの入力側から順に実行を完結していく必要がある．但し，再帰処理においては閉ループが形成されるため，このような順序は決定不可能である．そこで，再帰処理中に存在する遅延演算を始点として閉ループを切断して実行順序を決定可能にする．このような順序決定の根拠は次の通りである．ある時刻のデータが遅延演算に入ると，その出力は必ず同時刻よりも未来に進む．即ち，その時刻の処理は遅延演算で必ず行き止まりとなる．従って閉ループは，切断点の遅延演算を始点かつ終点とする一方向のグラフと見なすことができる．

図2のクエリグラフに対する適用例を図5に示す．各オペレータに付与された四角が実行順序を表す．ここで，実行順1～8以外のオペレータ群（図の灰色の領域）は閉ループを形成しているので，その中の遅延演算でループを切断する．その結果は図右のような一方向のグラフとなるので，入力から出力（図の下から上）に向かって昇順の実行順序を付与する．この順序を図左に戻すと，実行順9～19は図のように定まる．

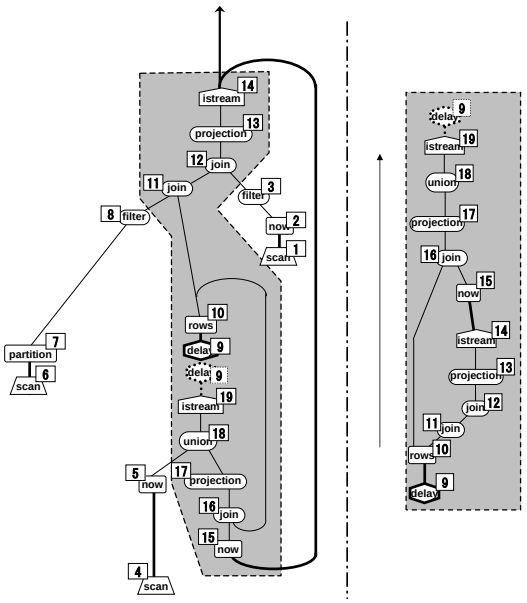

図 5　再帰クエリの実行順序
Fig.5　Execution Orders of Recursive Query

一般的には，以上のような遅延演算での閉ループの切断を再帰的に実施する．図6は多重に閉ループを含むグラフでの順序決定の例である．各円はオペレータ，太線は遅延演算を表す．まずaで切断すると，集合{c, d, e, m}および{f, g, h, i, j, k}がそれぞれ閉ループを形成している．これら閉ループをそれぞれ一つのオペレータとみなして順序を付け，更に各ループを切断していく．なお，閉ループ中に複数の遅延演算が存在する場合，切断点としてどれを選択するかは任意である．

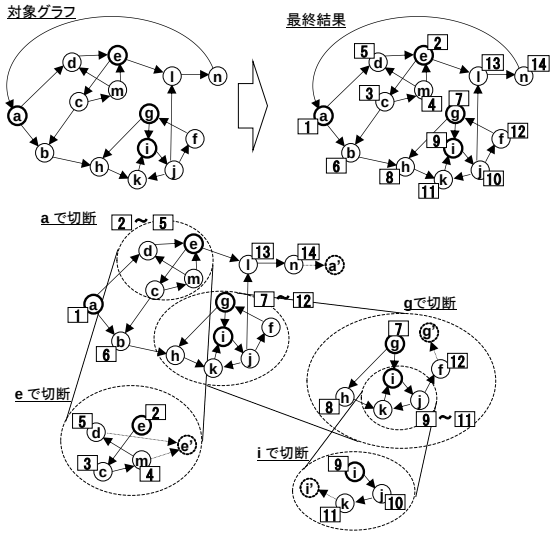

図 6　実行順序の決定方法
Fig.6　How to define execution orders

実行時刻は，タプルを自発的に生成するオペレータ（以下，発火点）について，直近に出力する可能性のあるタプルの時刻（以下，発火時刻）を比較して最も早い値に定める．発火点には，外部からの入力点であるscanオペレータの他，時限的な動作をする時間ウィンドウ，RStream，および遅延演算の三種類が存在する．例えば図5における発火点は，実行順

1, 2, 4, 5, 6, 9, 15のオペレータである．図4のタイムチャートでは，実行時刻t_1の処理が完了すると，2つの時間ウィンドウ[Now]および遅延演算の発火時刻が$t_1+\varepsilon$となり，これが最も早いため次の実行時刻に定まる．実行時刻を発火時刻とする，最も実行順の早い発火点が処理の起点となる．

ここで提案した，遅延演算の挿入位置に基いて決定したオペレータ実行順序に従う時刻別実行法を，以下ではDelay based Time-Splitting execution (DTS) 法と呼ぶ．

4. 評価

提案方式の評価として以下の二点を評価した．

- 従来の実行制御法（HB法）との性能比較
- 提案方式（DTS法）による再帰処理の性能

前者について4.2節で，後者について4.3節でそれぞれ示す．

4.1 実験環境と評価内容

本システムはJ2SE5.0ベースで実装している．評価に用いた実験環境のスペックはCPU: Intel® Core™2 Duo 2.33GHz, メモリ: 2GB，OS: Linux (Fedora 12) である．Java™のバージョンはJDK 6 Update 18，VMのオプションでヒープの最小・最大サイズに512MBを割り当て，それ以外のオプションはVMのデフォルト設定を用いた．

評価に用いたクエリとデータを図7に示す．入力データはidとvalの二つのカラムから成り，id=0～3で繰り返してvalが1からインクリメントする系列である．クエリはこのデータに対して，id別にvalの集計処理（avg）を実行する．本クエリのグラフは二重ループを構成している．6つのオペレータで構成される小クエリ（一点鎖線で囲われた部分）が複数組存在し，それを外側の再帰クエリ（all_avg_delay と uni_input）が一つに統合する．クエリグラフの複雑度（オペレータ数や接続関係）が性能に与える影響を評価するために，小クエリを1，2，4組と変化させた．各小クエリが互いに異なるidのデータを処理するため，小クエリの数に依らず，処理結果（クエリall_avg_delayの出力）は一致する．

また，再帰実行によって発生する性能への影響の調査，およびHB法との性能比較のために，このクエリの再帰ループを開放して得られる非再帰クエリも評価した．再帰で戻されるデータと同じ系列のデータを外部（図中のDUMMYの位置）から入力することで，各オペレータでの処理を再帰クエリと一致させている．

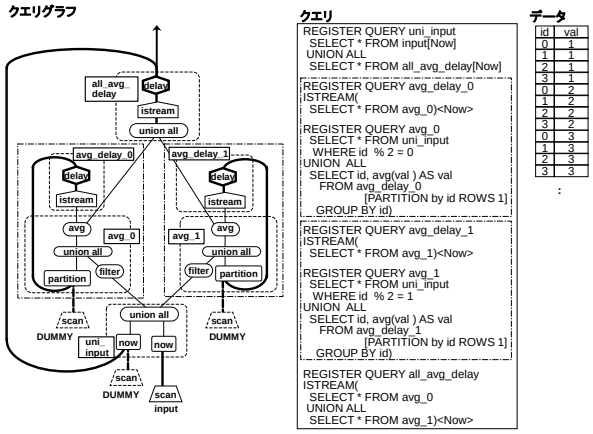

図 7　テストクエリとデータ

Fig.7　Query and Data for Evaluation

このクエリのストリーム（input）へ入力する2,000,000タプルの処理時間を測定した．非再帰クエリにおいてはinputの他にDUMMYにも相当するタプルを入力した．

4.2 従来方式との性能比較

図8に非再帰クエリの性能評価結果を示す．値は小クエリ数1のケースにおけるDTS法の結果を1.0とした処理時間の相対値である．横軸はHB法における一つのオペレータの連続実行回数の上限を表す．なお，HB法における実行順制御は，あるオペレータの実行において結果が生成された場合に，その出力先に位置するオペレータに処理を移す（複数ある場合はスタックしておく）方式とした．DTS法の値は水平線で示す．入力データのタイムスタンプの振り方として，(a)各タプルに異なる時刻を付与する場合，および(b)4タプルずつ同じ時刻を付与する場合の2パターンを評価した．

DTS法とHB法ともに，小クエリの個数が多くなるほど性能が劣化した．これはオペレータ数の増加に伴って空間的な局所性が減少し，キャッシュの利用効率が劣化したためと推測する．また，入力データのタイムスタンプの付与パターンを変更した(a)と(b)を比較すると，HB法ではほとんど性能差が無いのに対し，DTS法では(b)の方が高い性能を示す．これは，各オペレータが4つのタプルを連続して処理可能であるため，(a)に比べて(b)では処理の局所性が高まり，キャッシュの利用効率が改善したためと推測する．同様の推測は，HB法における連続実行回数と性能との関係からも可能である．

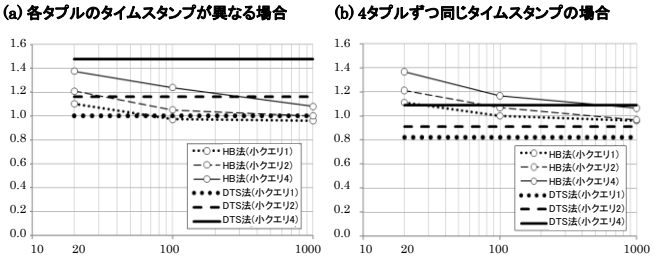

図 8　非再帰クエリの処理時間

Fig.8　Processing Time on Non-recursive Query

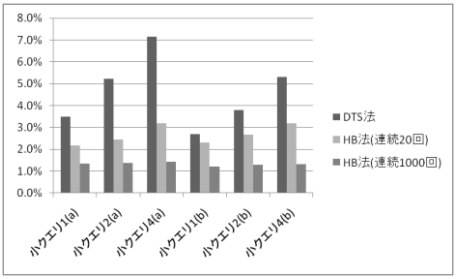

図 9　L1 データキャッシュのミス率

Fig.9　L1 Data Cache Misses

以上を確認するため，キャッシュのミス率を調査した結果を図9に示す．各棒は左から順にDTS法，HB法において連続実行回数の上限を20回とした場合，および1000回とした場合のL1データキャッシュのミス率である．DTS法，HB法ともに小クエリ数が多いほど高率，DTS法では(b)の方が低率，HB法では連続実行回数の多い方が低率といった傾向が，図8の性能グラフと一致した．

以上，DTS法をHB法と比較し，処理の局所性の低下によ

る性能劣化が確認された．但し，HB法において連続実行回数の上限を増やすことはレイテンシの増加に繋がるため，実運用では数十回程度に抑えることが想定される．その場合，DTS法はHB法と比べて性能的に遜色ないと考える．

4.3 提案方式による再帰処理の性能

図10に再帰クエリの性能評価結果を示す．値は4.2節と同じく相対値である．比較として，図8に示した非再帰クエリの値を再掲する．DTS法では再帰クエリと非再帰クエリでほぼ同程度の性能となることを確認した．再帰クエリでの性能が若干優れるのは，非再帰クエリに比べて発火点の数が減り実行時刻判定のコストが減少するためと考える．

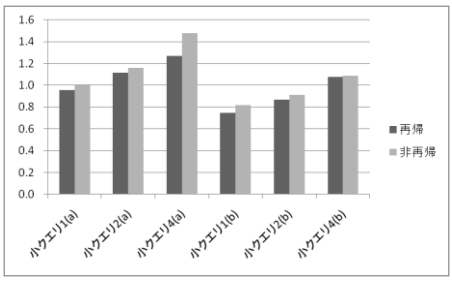

図 10　再帰クエリの処理時間
Fig.10　Processing Time on Recursive Query

以上の結果から，再帰クエリの実現方式としてDTS法が性能的にも実用性を備えると結論する．

5. 関連研究

ストリームデータ処理の実行制御に関する研究[7]は，スループット向上を目的に，オペレータ実行順に関する提案をしている．4.2 節に示した HB 法におけるスタックを用いた実行順制御方法と同様の制御である．閉ループを含むクエリグラフも対象としており，必要な場合はループ中の全オペレータをスタックに積むという方針をとる．一方，DTS 法では事前に決定した実行順に従ってループせずにグラフを辿る点が異なる．

ストリームデータの再帰処理方式に関する研究としては[8]が関連する．ハートビートと同様の概念である punctuation を再帰処理へ適用した場合に，Union，Join などの二項演算で punctuation の伝搬が永久に block するという課題（本稿の 2.3 節で示した課題と同じ）に対し，閉ループ内に特殊なオペレータ（Flying Fixed-Point）を挿入し，投機的な punctuation（以下 sp）をループ内で発生させる方式を提案している．sp を区切りとするデータシーケンスを単位として再帰ループを回すことで処理を進める．ループ内には同時に一つの sp しか流さないことを前提とする方式であるため，ループ中の分岐や多重ループを含むクエリを扱うことはできないと推測する．さらに，オペレータが強収束である必要がある，ウィンドウ演算のようなデータの生存期間を定めるオペレータを利用できない，などの制限が存在する点で DTS 法と異なる．一方で，[8]ではタプル間の out of order に対応可能である．DTS 法ではタプル間の out of order はストリーム処理の入り口で吸収されることを前提としており，クエリグラフ実行時の対応は今後の検討課題とする．

6. まとめと今後の課題

ストリームデータ処理における再帰処理の実現方法として，遅延演算の挿入による論理矛盾の解消方法を提案した．クエリ言語 CQL の意味論を変更せず，ストリーム化演算の有効性や利用可能な演算の種類を保存しつつ論理矛盾を解消することで，再帰処理においても CQL の利便性をそのまま保つことができる．また，実装方式として時刻別実行法を提案し，性能評価から再帰クエリの実行方式として適当であることを確かめた．一方で，処理の局所性が失われることによる性能劣化も確認された．今後の課題として，再帰ループ内とそれ以外で実行方式を使い分けるハイブリッド方式，および遅延演算の挿入位置の最適化を検討する．

【文献】

[1] Stonebraker, M., Çetintemel, U. and Zdonik, S.: “The 8 requirements of real-time stream processing”, SIGMOD Rec., Vol.34, No.4, pp.42-47 (2005).
[2] Arasu, A., Babcock, B., Babu, S., Cieslewicz, J., Ito, K., Motwani, R., Srivastava, U. and Widom, J.: “Stream: The stanford data stream management system”, Springer (2004).
[3] Abadi, D.J., Carney, D., Çetintemel, U., Cherniack, M., Convey, C., Lee, S., Stonebraker, M., Tatbul, N. and Zdonik, S.: “Aurora: a new model and architecture for data stream management”, The VLDB Journal, Vol.12, No.2, pp.120-139 (2003).
[4] Madden, S., Franklin, M.J., Hellerstein, J.M. and Hong, W.: “The Design of an Acquisitional Query Processor for Sensor Networks”, In ACM SIGMOD, ACM Press, pp.491-502 (2002).
[5] Johnson, T., Muthukrishnan, S., Shkapenyuk, V. and Spatscheck, O.: “A heartbeat mechanism and its application in Gigascope”, In VLDB, pp.1079-1088 (2005).
[6] Arasu, A., Babu, S. and Widom, J.: “The CQL Continuous Query Language: Semantic Foundations and Query Execution”, Technical report, Stanford (2003).
[7] Girod, L., Mei, Y., Rost, S., Thiagarajan, A., Balakrishnan, H. and Madden, S.: “XStream: a Signal-Oriented Data Stream Management System”, In ICDE, pp.1180-1189 (2008).
[8] Chandramouli, B., Goldstein, J. and Maier, D.: “On-the-fly Progress Detection in Iterative Stream Queries”, In VLDB, pp.241-252 (2009).
[9] 今木常之，西澤格: “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”, DBSJ Letters Vol.5, No.2, pp.65-68 (2006).

今木 常之　Tsuneyuki IMAKI
(株)日立製作所中央研究所主任研究員．1996 東京大学大学院工学系研究科修士課程修了．情報処理学会会員．

樫山 俊彦　Toshihiko KASHIYAMA
(株)日立製作所中央研究所研究員．2005 東京工業大学大学院情報理工学研究科修士課程修了．FIT 2004 船井ベストペーパー賞受賞．電子情報通信学会会員．

西澤 格　Itaru NISHIZAWA
(株)日立製作所中央研究所主任研究員．1996 東京大学大学院工学系研究科博士課程修了，工学博士．2002-2003 米スタンフォード大客員研究員．ACM，情報処理学会各会員．