ストリーム処理エンジンにおける効率的な来歴管理

川島　英之[†,††]　　北川　博之[†,††]　　寺島　裕貴[†††]

† 筑波大学大学院システム情報工学研究科　〒 305–8573 茨城県つくば市天王台 1–1–1
†† 筑波大学計算科学研究センター　〒 305–8573 茨城県つくば市天王台 1–1–1
††† 筑波大学第三学群情報学類　〒 305–8573 茨城県つくば市天王台 1–1–1
E-mail: †{kawasima,kitagawa}@cs.tsukuba.ac.jp, ††yterajima@kde.tsukuba.cs.ac.jp

あらまし　本論文は，データストリームにおける来歴管理技法を示す．ストリーム処理エンジンの出力を受け取ったアプリケーションからの根拠を問い合せる要求に応えるため，ストリーム処理エンジンに到着したタプルストリームの内，その出力の全来歴を永続化する．まず，データストリームの来歴となりうるデータは，実行木の出力タプルの来歴タプルであることを示す．ストリーム処理環境では，実行木の出力タプル以外のデータはユーザに提供されない．ユーザに提供されないタプルは，ユーザに感知されない．感知されないタプルは意味を持たない．次に，頻繁に到着するデータストリームを一括してディスクに転送する方式を提案する．結果集合を成すタプルの全来歴タプルを連続領域にマーシャリングし，それをディスクへ一括転送することで，処理時間の長いディスクアクセスを減らすことができる．そして，プロトタイプ SPE を作成して，ウィンドウ演算と選択演算から構成される実行木について，ウィンドウ幅を変えて来歴タプル永続化に関する処理時間を計測した．実験の結果，来歴タプル永続化処理には多大な時間を要すること，およびその原因はディスクアクセス回数であることを示した．

キーワード　データストリーム，来歴

Efficient Lineage Management on Stream Processing Engines

Hideyuki KAWASHIMA[†,††], Hiroyuki KITAGAWA[†,††], and Yuki TERAJIMA[†††]

† Graduate School of Systems and Information Engineering, University of Tsukuba1-1-1 Tennodai, Tsukuba, 305-8573
†† Center for Computational Sciences, University of Tsukuba, Tennoudai 1–1–1,Tsukuba City, Ibaraki, 305-8573, Japan
††† College of Information Science, University of Tsukuba Tennoudai 1–1–1,Tsukuba City, Ibaraki, 305-8573, Japan
E-mail: †{kawasima,kitagawa}@cs.tsukuba.ac.jp, ††yterajima@kde.tsukuba.cs.ac.jp

Abstract　This paper proposes lineage management techniques for data streams. The techniques provide persistence with lineages of tuples output to applications. Firstly, we show that lineages of data streams are limited to lineages of output tuples of operator trees. Secondly, we propose to transfer lineage tuples to a disk in a lump. We developed a prototype SPE and evaluated the proposal techniques.

Key words　Data Stream, Lineage

1. はじめに

近年のセンサデバイス技術の進展に伴い，センサデータの生成量ならびにその利用例は増加の一途を辿っている．センサデバイスは実世界の状況を検知するために設置される為，それらが出力するセンサデータはその目的を達成する為に使用される．センサデータの特徴は，短周期で継続的に生成されることである．この特徴より，センサデータはストリーム処理されることが多い．それに従い，本論文ではセンサデータにはストリーム処理が施されることを想定する．ストリーム処理の内容としてSTREAM [11] が規定する演算の一部を想定する．

センサデータがストリーム処理されて得られた出力について，その来歴 (Lineage) を知りたい欲求は自然に生まれる．そのような例として，道路に設置された車の速度を測る速度センサのデータと，画像センサと超音波センサにより道路の状態を常に監視して事故を検知する [16]（以降では事故ストリームと略記）から事故状態のデータを結合することで各道路の車の平均速度と事故状態を知らせるアプリケーションを考える．

図 1.1 はセンサの設置例である．速度センサはスピードガンを利用して，自動車の速度を測定する．これは通称オービスと

呼ばれる自動速度違反取締装置で採用されている．この速度センサが国道 6 号線，125 号線，246 号線（図 1,2 では R6，R125，R246 と略記）に一定間隔で設置されているとする．事故ストリームは画像センサと超音波センサを共に利用したアルゴリズムにより事故を検出する．

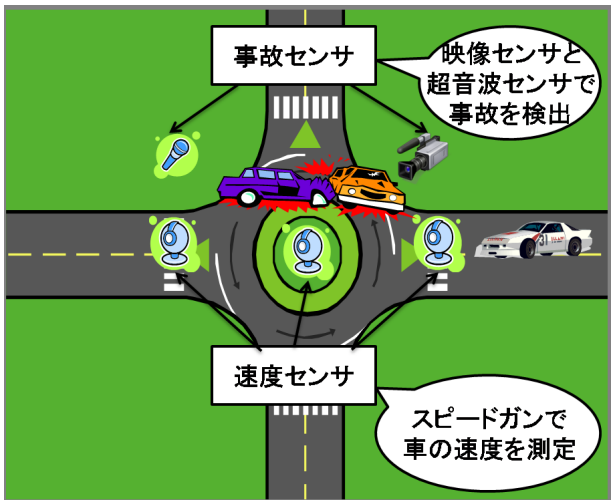

図 1　道 路 情 報

速度

ID	道路	S_id	時速	時刻
1	R6	1	60	11:00
2	R6	2	40	11:00
3	R6	3	80	11:00
4	R246	4	10	11:00
…	…	…	…	…

事故情報

ID	道路	事故	時刻
24	R246	事故	11:00
25	R125	通常	11:00
26	R6	通常	11:00

平均速度

ID	道路	平均速度	時刻
12	R125	20	11:00
13	R6	70	11:00
14	R246	50	11:00

図 2　入力データと平均速度

図 1.2 左の速度リレーションは速度センサのストリームから作られている．ID，道路，s_id，速度，時刻を属性とする（左から来歴の識別子，道路の名称，センサの識別子，測定した自動車の速度，測定時刻）．車が速度センサを通過する度に自動車の速度を測定し，データを生成する．図 1.2 中央の事故情報リレーションは事故ストリームから作られている．ID，道路，事故，時刻を属性とする（左から来歴の識別子，道路の名称，事故の有無，時刻）．事故ストリームは毎秒その道路の状況のデータ（事故が発生しているかいないか）を生成する．以上二つのデータから道路ごとの自動車の平均速度と事故の有無の情報を取得する．はじめに，速度センサの時速の道路ごとの平均値を求める．図 1.2 右の事故情報リレーションはこれにより作られる．平均速度リレーションと事故情報リレーションを道路で結合することにより，道路ごとの自動車の平均速度と事故の有無の情報を得ることができる．今，この出力に，おかしいデータがあるとする．自動車の平均速度は法定速度と同程度であるのに道路で事故が発生しているというタプルが存在するとする．この時考えられる理由は，ある速度センサが壊れているために平均速度の値に異常が起きた，ある地点に法定速度を遥かに超えた速度で走っている自動車がある，事故ストリームが誤って事故として検出した，等である．どの理由が正しいのかを判断するためには，出力タプルがどの入力データを来歴とするのかを知る必要がある．前二つの事例に関しては個々の速度センサの測定した速度を調べることで判断ができる．事故ストリームの事例では事故ストリームの情報と実際の状態を比較することで事故センサが誤りを起こしたかを確認することができる．しかし，ストリーム処理エンジンでは来歴である入力データ（以降では来歴データと略記) の保存はされない．従って，将来の来歴追跡要求に備えて，来歴データを保存しておく必要がある．

そこで本研究では，ストリーム処理エンジンにおいて，来歴データを永続化する方式を 2 つ提案する．第一の方式は，実行木への入力タプル全てを永続化する．第二の方式は，実行木 (Operator Tree) の根に到着したタプルの来歴タプルを永続化する手法である．

来歴管理はデータウェアハウス，科学的ワークフロー，確率的データベースにおいて研究されてきたが，ストリーム処理エンジン内部で来歴を管理する研究は筆者の知る限り，本論文が初めてである．

本論文の構成は次の通りである．2 節で関連研究について述べる．3 節で本研究のモデルを述べる．4 節でシステムの設計と実装について述べる．5 節で実験の評価について述べる．最後に 6 節では本研究をまとめ，今後の課題を述べる．

2. 関 連 研 究

2.1 ストリーム処理エンジン

データストリーム研究はおおまかに 3 世代に分けることができる．第一世代 SPE に関する研究では，関係データモデルを拡張して，ストリームを扱うためのデータモデルが提案された．そして同モデル上における性能向上とメモリ使用量の制御が研究された．このようなモデル化が行われた理由は，汎用的である関係データモデルを拡張することで，データストリーム処理の汎用的なモデルを構築しようとしたからである [2], [4], [7]. 第二世代 SPE に関する研究では，第一世代 SPE で提案されたデータモデルに対して，分散処理を導入することで，高性能化および耐故障化に関する研究が推進された．この研究動向は現在もまだ継続されている [12]．そして第三世代 SPE に関する研究では，第一世代，第二世代とは一線を画し，明確なアプリケーションを想定した，専門用途システムの構築が研究されている．用途例には，RFID，ネットワークトラフィック，そして動物の監視など，幅広いアプリケーションが含まれる [6], [9], [10].

本論文では，第一世代の SPE である STREAM システムのモデル [3] を基礎とする．

2.1.1 来　　歴

データ来歴とはデータの由来を表す．データ来歴は 2 種類に大別される．データがどこから来たかを表す起源来歴 (Where Provenance) と，どのような処理を経て来たかを表す処理来歴 (Why Provenance) である．本研究では起源来歴のみを対象とする．本論文の残りでは，起源来歴を略して来歴と記す．

来歴に関する研究には，データウェアハウスでのビュー構築

に関する来歴を扱う研究[8]や，確率的データベースにおける計算の安全性と効率的を満たすために来歴を用いる研究[5],[14]，科学ワークフローにおけるユーザ指向の問合せを来歴を利用することで実行ログから便利な問合せを提供したり，来歴によりデータ依存関係の軌跡を取ることで回答の正確性を向上させる研究[17]，データセットの管理に来歴を用いることでその検索、解析、計算結果のアクセスを行えるようにする研究[18]，センサデータのインデックス化とその探索に来歴を用いる研究[19]，来歴をそのまま保存すると保存する量が増えすぎてしまうので，来歴保存を効率的に行う研究[20]がある．[20]は少ないコストで保存する来歴のサイズを最大 20 倍減少させることに成功した．

本研究の来歴モデルは ULDB[5],[14] における来歴データベース (LDB) に基づく．LDB では，各タプルに識別子 (ID) 及び来歴関数 λ が与えられる．λ は入力をタプル ID とし，出力を来歴であるタプルの多重集合とする．

2.1.2 高速データ永続化

DBMS におけるトランザクション処理の高性能化のため，高速データ永続化には多くの研究がされてきた．[13] では 2 台のリモートメモリを永続的デバイスとみなすことで高速化を実現している．リモートメモリとはリモートホストのメモリのことであり，頻繁に到着するトランザクションを処理する場合には，ディスク帯域よりも TCP を用いたネットワーク帯域の方が広いことに着目した技法である．この技法はトランザクション処理を高速化するが，ストリームデータに着目した技法ではなかった．

SPE におけるデータストリームの高速永続化技法としては [15] がある．この研究では，実行木の根ノードの入力キュー内のリレーションを一括書込処理により，高いスループットを実現している．この研究は実行木の出力結果を永続化することを対象にしており，その来歴を永続化することは考えていない．また，実験内容として単純な選択演算のみを対象にしている．

3. 提　　案

3.1 研究課題

通常の DBMS はデータ永続化処理をトランザクショナルに行う．つまり INSERT 命令毎に永続化処理を行う．一方 SPE の場合には，全ての入力タプルを永続化する必要はない．なぜなら，結果として出ていったタプルだけが人間の目に触れる可能性があるからであり，結果として出ていかないタプルは，ユーザあるいはアプリケーションにとっては存在しないと見なされ得るからである．そこで本研究では，結果として出ていくタプルの来歴タプルを全て永続化することを研究課題と設定する．このとき注意されなければならないことは，来歴タプルが永続化されるまで，結果タプルは出力されてはならないことである．なぜなら，永続化前のタプルが人目に触れたとして，その来歴タプルを求められたときに，突発的事故等により来歴タプルが失われることで検索不能になる事態を防ぐためである．文献 [15] においても，そのような方式は高速であるものの無意味であると触れられている．

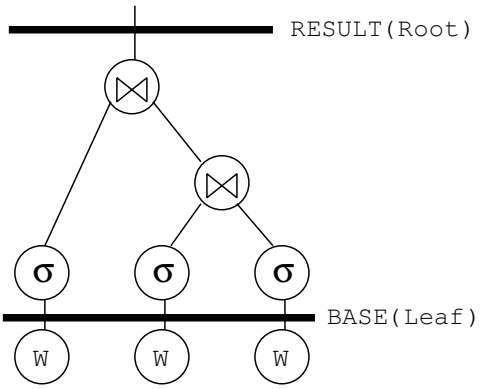

図 3　方式概略

3.2 前　　提

本論文では下記の前提を設定する．第一の前提として，入力ノードであるウィンドウ演算の出力タプルが実行木を上って全ての終点ノードから出力されるまで，次のウィンドウ演算は実行されないと定める．第二の前提として，実行木の終点ノードには必ず新規増分タプルストリーム出力演算子(注1)が置かれると定める．第三の前提として，来歴タプル集合はウィンドウ演算の入力タプル集合に含まれると定める．第四の前提として，ウィンドウ演算子は行数ウィンドウ (tuple based window) のみ有すると定める．第五の前提として，来歴の永続化が終了する迄，実行木は結果タプル集合を出力しないと定める．

3.3 永続化技法

来歴を永続化するための技法を二方式提案する．一つは実行木の葉でタプルを永続化する方式であり，もう一つは実行木の根でタプルを永続化する方式である．これを図 3 に示す．いずれの方式にも長短があるため，実行木の形態および入力タプルに応じて有利な方式は変わる．いずれにせよ，処理時間が長くなりすぎると，ストリームが頻繁に大量のタプルを流す場合に，要求性能を実現できず，SPE がその目的を果たせない場合が生じうる．

各方式について説明する前に，全方式に共通の部分を記しておく．システムにデータが到着すると，同じタプルが二つ生成される．ひとつは実行木へ投入され，もうひとつは起源タプルとして保持される．起源タプルの永続化を実行木の処理のどの時点で行うかにより，二方式は分けられる．永続化処理の実行は実行木を処理するスレッドと別のスレッドで実行される．即ちマルチコア環境において，永続化実行中も問合せ処理は実行される．実行木用スレッドは処理を終えても，永続化スレッドの処理が終わるまでは，結果を出力しない．ウィンドウ結合を含め，あるタプルが実行木で使われているかどうかは，ウィンドウ演算のセマンティクスにより判定される．各ノードのシノプシスが保存するタプル長はユーザからの問合せにより自動的に決定するため，この管理は容易である．ウィンドウ演算の入力タプルは，その結果タプルが実行木で使われているか，あるいは，それが永続化されるまでは破棄されない．

(注1)：STREAM における ISTREAM 演算子

Algorithm 1 葉タプル永続化方式

```
St; { タプルストリーム }
O; { 実行木 }
永続化フラグ ⇐ false; { 永続化が完了しているか判定 }

{ 以下は実行木スレッドの動作 }
while |St| ≠ 0 do
  Operate(St, O); { 問合せ処理の実行 }
  while 永続化フラグ = false do
    Wait 永続化; { 永続化終了まで待つ }
  end while
  問合せ結果を出力;
end while

{ 以下は永続化スレッドの動作 }
St; { タプルストリーム }
Buf; { 起源タプルのバッファ}
while |St| ≠ 0 do
  Buf ⇐ St { 起源タプルをバッファに保持 }
  永続化処理実行; { 永続化 }
  永続化フラグ ⇐ true; { 永続化完了 }
end while
```

Algorithm 2 根タプル永続化方式

```
St; { タプルストリーム }
O; { 実行木 }
Buf; { 起源タプルのバッファ}
永続化フラグ ⇐ false; { 永続化が完了しているか判定 }
Root フラグ ⇐ false; { 実行木のタプルが根まで到達しているか }

{ 以下は実行木スレッドの動作 }
while |St| ≠ 0 do
  Operate(St, O); { 問合せ処理の実行 }
  Root フラグ ⇐ true; { 根に到達 }
  永続化指示; { 永続化スレッドに永続化を指示 }
  while 永続化フラグ ⇐ false do
    Wait; { 永続化終了まで待つ }
  end while
  問合せ結果を出力;
end while

{ 以下は永続化スレッドの動作 }
while |St| ≠ 0 do
  Buf ⇐ St; { 起源タプルをバッファに保持 }
  while Root フラグ ⇐ false do
    Wait; { 実行木の出力が決定するまで待機 }
  end while
  永続化処理実行; { 永続化 }
  永続化フラグ ⇐ true; { 永続化完了 }
end while
```

3.3.1 葉タプル永続化方式

この方式は，起源タプル全てを永続化する．この方式のアルゴリズムを以下に記す．

本方式は，起源タプルのコピーを即座に永続化する．同時に，起源タプルのオリジナルを実行木スレッドが処理する．この処理はマルチコア環境において並列に実行される．なお，出力に使用されないタプルの起源タプルは，後で非同期的にディスクから削除される．

図 4 はこの方式の例を示している．ストリームからウィンドウ演算が入力を受けて，その後，選択演算と結合演算が実行される．ウィンドウ演算の入力を起源タプルとし，すぐさま永続化を行っている．

本方式の利点は，実行木のコストが高い場合には，最速となることである．なぜなら，永続化処理と実行木処理が並列に行われるからである．実行木のコストが高い場合は，直積が複数ある場合，あるいは選択演算・結合演算によるタプル数の減衰率が低い場合である．

本方式の欠点は，実行木からの出力が少ない場合には，大量の不要なタプルを永続化してしまうことである．このような場合，それらの大量の不要なタプルを削除する必要がある．削除に際してはシステムに負荷をかけない為に非同期的な実行方式が用いられる．

3.3.2 根タプル永続化方式

本方式は，実行木が出力する全てのタプルの来歴タプル集合を永続化する．すなわち，実行木へ投入されたタプルの内，根まで残ったタプルの来歴である起源タプルのみが永続化のためにディスクへ書き込まれる．残りのタプルの内，ウィンドウ結合で用いられる可能性があるタプル以外は破棄される．

アルゴリズム 2 では，処理が根に到達するまで永続化スレッドは永続化処理を行わない．処理が根に到達すると，実行木スレッドは永続化スレッドに対して，タプル群を渡す．永続化スレッドは受け取ったタプル群の来歴タプルを起源タプル内から同定して，それらをマーシャリングして永続化を実行する．

図 5 は本方式の挙動を示している．図 4 と同様に実行木の演算は行われていくが，起源タプルの永続化は出力が決定した後に行われる．

本方式の利点は，実行木からの出力が一つもない場合には，永続化作業を行わなくてよい為に最速になることである，あるいは，実行木において相当数のタプルが減らされる場合には，高い性能を示すことである．それに加えて，不要なタプルが永続化されることがないために，ディスクへの書き込み量には無駄がない．それゆえアルゴリズム 1 のように，後で非同期的にディスクアクセスをする必要がない．

本方式の欠点は，実行木の出力が決定してから来歴タプル永続化を行うため，根で永続化を必ず待つ時間が発生することである．特に，永続化するタプルの数が多い場合には，ディスク書込時間および書込確認時間が長くなるため，待ち時間が延びる．

4. 設計と実装

4.1 設　　計

4.1.1 実行木の設計

提案手法を評価するために，プロトタイプ SPE を作成した．

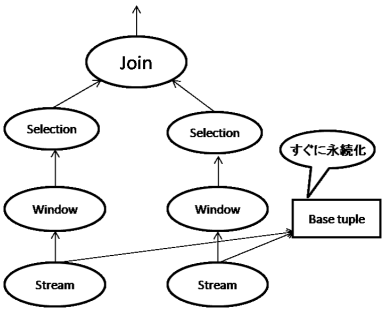

図 4　葉タプル永続化方式

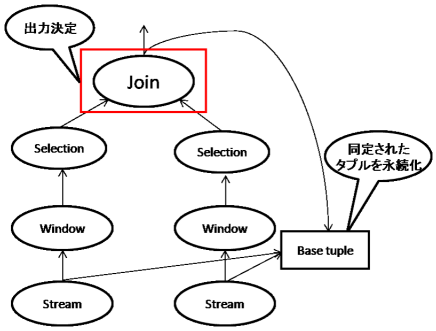

図 5　根タプル永続化方式

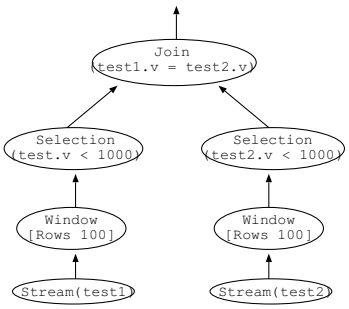

図 6　生成された実行木

```
(J[test1.v = test2.v]((S[test1.v < 1000](W[test1, 100])), (S[test2.v <
1000](W[test2, 100]))
```

図 7　SPE の入力例

```
test1 (id int) (v double)
test2 (id int) (v double)
```

図 8　ストリーム定義

この SPE は代数式を入力し，その代数式から実行木 (Operator Tree) を作成する．例えば二つのストリーム test1, test2 があるとき，図 7 の代数式の意味は次のようになる．

- test1, test2 に対して行数 100 のウィンドウ演算子を適用する．
- 各ウィンドウ演算子の出力に対して，1000 以下の値を有するタプルのみを選択演算により残す．
- 選択演算の出力を等結合する．

本 SPE は図 7 から図 6 に示されるような実行木を生成する．この状態から書き換えは行われない．即ち実行木内のノード位置移動による問合せ最適化処理は本研究の対象外である．

実行木の入力はタプルストリームであり，これはウィンドウ演算子によりリレーションに変換される．ウィンドウ演算子は必ず実行木の葉に置かれるため，実行木内にはストリームは存在せず，データは全てリレーションとして扱われる．オペレータの実行順序は，下から上である．同じ高さに存在する演算子については，それらの間に実行順序の優先度は存在しない．即ちどの順序で実行されても構わない．この規則に従ってオペレータ実行順序を定める．実行順序は実行キューにより管理され，連続的問合せが終了するまで順序が変わることはない．実行木の出力はタプルストリームになる．本研究では CQL における Istream [3] のみを対象とする．すなわち，本研究における実行木は新しい入力に関する結果のみを出力する．

ストリームの定義は，名前とデータ型の組を記述することで行われる．例えば図 8 では二つのストリーム，test1 と test2 を定義している．いずれも int 型属性 id と double 型属性 v を有する．本 SPE が実現した型は，int 型，double 型，そして固定長テキスト型である．

4.1.2　並行/並列処理

各ストリーム生成処理，実行木処理，そして永続化処理は全て異なるスレッドにより実現される．それゆえこれら複数のスレッドは並行/並列処理される．

4.2　実　　装

実装は C 言語により行った．排他制御のために pthread ライブラリの mutex を使用した．総行数は 1900 行程度である．タプル形式はスキーマにより決定されるため，データ領域は void 型とし，全属性のサイズの総和長の連続領域により実現した．同連続領域は属性サイズにより区切られる．

来歴はモデル的にはタプル ID の集合だが，本実装では各導出タプルから基底タプルへのポインタ配列により来歴を実装した．

来歴の永続化処理に際してはマーシャリングを行い高速化を実現した．すなわち，結果集合中の各タプルについて，ポインタ配列中のポインタを辿り，来歴タプルを発見する．そしてその来歴タプルの ID とデータを連続領域にコピーする．この作業を全ての来歴タプルについて実行し，write および fsync システムコールを一度だけ呼び出してディスクに書込みを行った．ディスクアクセスコストは高価であるため，このような方式を採用した．

結合演算の結果，複数の導出タプルが同一の起源タプルを来歴タプルとすることがある．この場合，同一タプルが二度以上は永続化処理をされてはならない．換言すれば，永続化処理は冪等 (idempotent) である必要がある．そこで，永続化処理したタプルのメモリ番地を保存しておき，永続化処理前にそれを参照することで冪等化を実現した．

表 1　実 験 条 件

条件内容	条件値
入力タプルストリーム数	10000
sleep 時間	なし
入力データ	全て同じ

表 2　実 験 環 境

環境内容	条件値
CPU	Intel(R) Core(TM)2 Quad CPU
コア数	4
コア周波数	2.66 GHz
RAM 容量	4GB

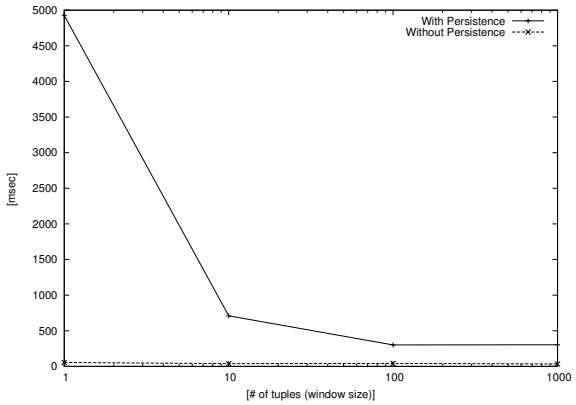

図 9　処理時間の比較

5. 評　　価

5.1 実 験 内 容

提案手法の葉タプル永続化方式と根タプル永続化方式の性能を実験により測定した．来歴永続化を行わない場合の処理時間を測定し，根タプル永続化方式との比較を行った．また，葉タプル永続化方式と根タプル永続化方式の比較を行った．

実験条件を表 1 に示す．我々の実験環境においては fsync システムコールの処理時間が速い場合には僅か数百マイクロ秒と非常に高速だったため，永続化に伴うコストを明確するために，ストリームを生成するスレッドを含め，全スレッドの sleep 時間を零に設定した．実際には全スレッドにおける sleep 関数をコメントアウトして実験を行った．

実験に用いた問合せは `S[test1.v $<$ 1000](W[test1, 100])` であり，ストリームのスキーマ `test1 (id int) (v double)` である．

5.2 実 験 結 果

5.2.1　来歴永続化のコスト

図 9 に実験結果を示す．図 9 は横軸 (対数) にウィンドウサイズ（行数ウィンドウ），縦軸に処理時間を表す．図より次のことが見て取れる．

- 来歴を行わない場合 (“Without Persistence”)，スループットは 56 万タプル/秒である．
- 来歴永続化のコストは大きい．
- 行数ウィンドウ幅が 1 の場合，“With Persistence” の性能は悪い．

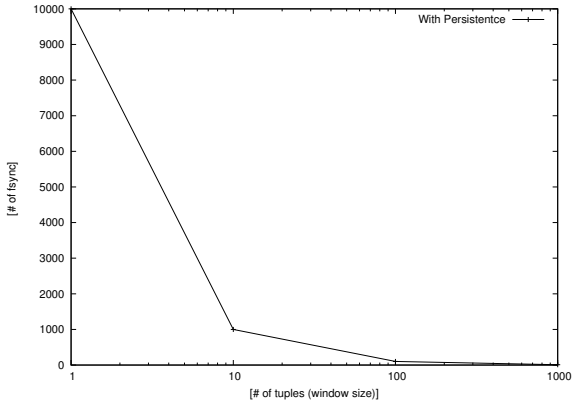

図 10　同期書込回数

来歴永続化については，行数ウィンドウ幅と性能の間に密接な関係があるように見える．この関係は，ディスクへの同期書込回数によりもたらされる．

図 10 にウィンドウサイズと同期書込回数の関係を示す．ディスクへの書き込みはコストが高いため，この回数が多い程，性能は大きく下落する．特にウィンドウサイズが 1 の場合，多くのタプルが到着していようとも，必ず一つずつしか処理されないために，ディスク書き込み回数が極めて多くなり，そのために性能が極めて悪化する．ウィンドウサイズが 1 である場合，処理実態は通常のトランザクションと類似する．即ち，通常のトランザクション処理を用いると，非常に性能が劣化することが推察される．

本研究では対象外としたが，ログファイルのデータからデータベースのデータを再構築する処理が通常は必要になる．この処理は別スレッドにより実現され，他のディスク領域へのアクセスを行うため，排他制御処理とランダムディスクアクセスを引き起こし，更に性能を劣化させる可能性がある．

5.2.2　プロトタイプ SPE の基本性能

上記のスキーマ，問合せに関するプロトタイプ SPE のスループットを測定した．これを図 11 に示す．ウィンドウ演算と選択演算により構成される単純な実行木の場合には，プロトタイプ SPE は最大で 56 万 TPS もの性能を達成している．一方，来歴を永続化する場合，最悪の場合には 2000TPS まで性能が下落する．すなわち 280 倍程度の性能差がある．

5.2.3　葉方式と根方式の比較

2 つの提案手法である，葉方式と根方式を実験条件を変えて比較した．以降の図において，RESULT は根方式を表し，BASE は葉方式を表す．選択率の変動は，選択条件の指定により実現した．入力データストリームにはデータとして 100 の値を持たせ，選択率を 0%，10%, 100%にするために，選択条件をそれぞれ `v > 1000`，`v < 100`，`v < 1000` とした．

図 12,13,14 より，実行木中に選択演算が 1 つしか存在しない場合は，選択率が 1.0 であろうとも根方式は葉方式を上回る性能を示した．

一方，図 1516 に示すように，実行木のコストが高い場合には，葉方式は根方式を上回る性能を示した．図 15 はウィンド

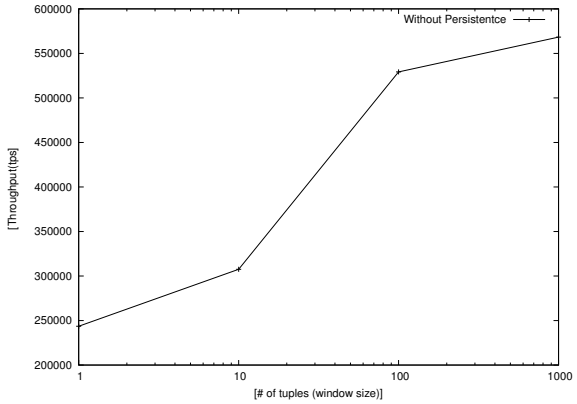

図 11　スループット

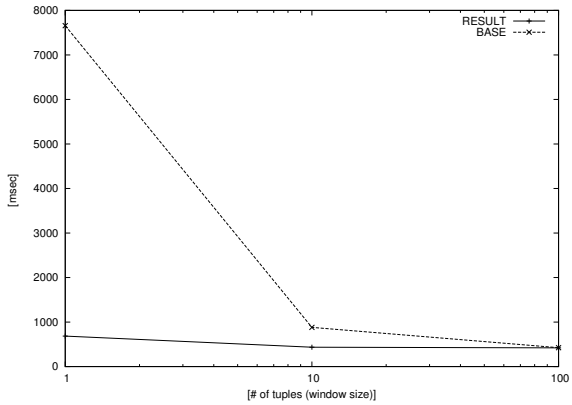

図 12　1 回の選択演算 (選択率=0)

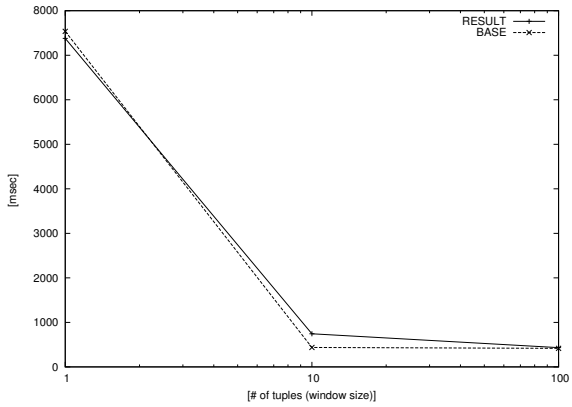

図 13　1 回の選択演算 (選択率=0.1)

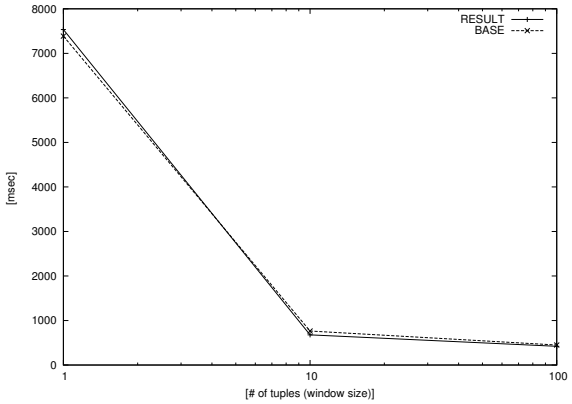

図 14　1 回の選択演算 (選択率=1.0)

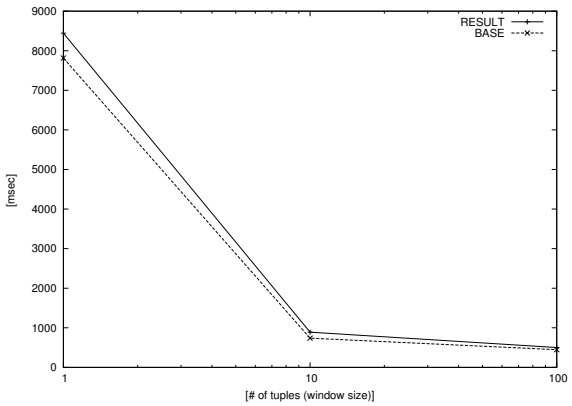

図 15　300 回の選択演算 (選択率=1.0)

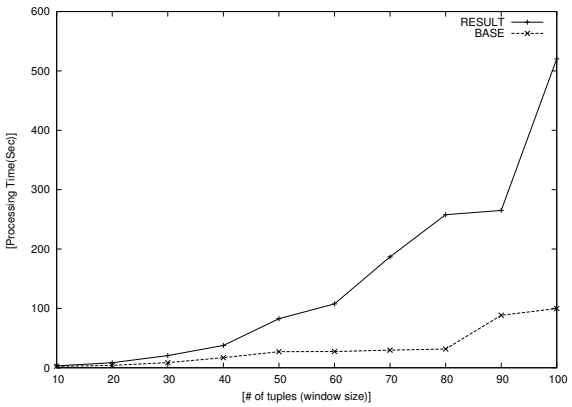

図 16　直　　積

ウ演算後に選択演算が 300 個設置された場合であり，図 16 は直積が二つあるような場合である．この場合には 3 つのウィンドウ演算子があるため，ウィンドウ演算子のタプル数が 10 であれば，永続化すべきタプル数は 30 であるのに根では 10000 のタプルが生成されてしまい，永続化必要性確認処理の為に長い時間が必要とされる．

本研究では永続化処理をディスクへの書き込みとした．ディスクへの書き込みコストは高いため，基底法は特殊な条件でしか高い性能を示さなかった．しかし今後 MRAM を代表とする不揮発高速メモリが使用可能になった場合，両者の関係は変わると考えられる．

6. 結　　論

本研究の目的は，データストリームにおける来歴管理技法を示すことであった．

まず，データストリームの来歴となりうるデータは，実行木の出力タプルの来歴タプルであることを示した．ストリーム処理環境では，実行木の出力タプル以外のデータはユーザに提供されない．ユーザに提供されないタプルは，ユーザに感知されない．感知されないタプルは意味を持たない．意味を持たないタプルを形成するタプルもやはり意味を持たない．本論文ではそのように考えた．

次に，頻繁に到着するデータストリームを一括してディスクに転送する方式を提案した．結果集合を成すタプルの全来歴タプルを連続領域にマーシャリングし，それをディスクへ一括転送することで，処理時間の長いディスクアクセスを減らすことができた．

そして，プロトタイプ SPE を作成して，ウィンドウ演算と選択演算から構成される実行木について，ウィンドウ幅を変えて来歴タプル永続化に関する処理時間を計測した．実験の結果，来歴タプル永続化処理には多大な時間を要すること，およびその原因はディスクアクセス回数であることがわかった．

6.1 今後の課題

- 演算子に関する課題

本研究ではウィンドウ演算と選択演算のみを対象にした．今後はウィンドウ結合，射影演算，集約演算，Model based User View も対象にする．

- 効率化に関する課題

本論文で述べた技法は，来歴タプルの永続化を実現するものの，基本的な方式であり効率の悪いものであった．来歴タプル永続化により，最悪で 280 倍程度の性能劣化が生じることを 5.2.2 節で述べた．そこで効率を改善する技法案を 2 つ述べる．

– 標本乱択

この方式は，ウィンドウ演算子を実行した後，一旦実行木の動作を停止する．そしてその出力から一部のタプルを乱択して，それらを用いて来歴タプル数が最小になる枝集合を推測する．それから実行木の動作を再開し，推測された点において永続化を実行する方式である．推測精度を高めることが基本的に重要な課題となろう．

– 圧縮処理

この方式では，永続化されるタプル集合は圧縮化後，一括してディスクへ転送される．圧縮化の理由はディスク帯域の効率化である．

謝辞 本研究の一部は科学研究費補助金基盤研究 (# 18200005)，筑波大学 VBL 研究プロジェクトによる．

文　献

[1] D. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and Stan Zdonik. The design of the borealis stream processing engine. In *Proc. of the Conference on Innovative Data Systems Research*, 2005.

[2] Daniel J. Abadi, Donald Carney, Ugur Cetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, and Stanley B. Zdonik. Aurora: a new model and architecture for data stream management. *VLDB Journal*, Vol. 12, No. 2, 2007.

[3] Arvind Arasu, Shivnath Babu, and Jennifer Widom. The cql continuous query language: Semantic foundations and query execution. *VLDB Journal*, Vol. 15, No. 2, 2006.

[4] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and Issues in Data Stream Systems. In *ACM Symposium on Principles of Database Systems*, 2002.

[5] O. Benjelloun, A. Das Sarma, A. Halevy, and J. Widom. Uldbs: Databases with uncertainty and lineage. In *Proc. of International Conference of Very Large Databases*, pp. 953–964, 2006.

[6] Lars Brenna, Alan Demers, Johannes Gehrke, Mingsheng Hong, Joel Ossher, Biswanath Panda, Mirek Riedewald, Mohit Thatte, and Walker White. Cayuga: a high-performance event processing engine. In *Proc. of the 2007 ACM SIGMOD International Conference on Management of Data*, pp. 1100–1102, 2007.

[7] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellestein, Wei Hong, Sailesh Krishnamurthy, Sam Madden, Vijayshankar Raman, Fred Reiss, and Mehul Shah. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In *Proc. of the Conference on Innovative Data Systems Research*, 2003.

[8] Yingwei Cui, Jennifer Widom, and Janet L. Wiener. Tracing the lineage of view data in a warehousing environment. *ACM Transactions on Database Systems*, Vol. 25, pp. 179–227, 2000.

[9] Yanlei Diao, Neil Immerman, and Daniel Gyllstrom. Sase+: An agile language for kleene closure over event streams. In *UMass Technical Report 07-03*, 2007.

[10] Lewis Girod, Yuan Mei, Ryan Newton, Stanislav Rost, Arvind Thiagarajan, Hari Balakrishnan, and Samuel Madden. The case for a signal-oriented data stream management system. In *Proc. of the Conference on Innovative Data Systems Research*, pp. 397–406, 2007.

[11] The STREAM Group. Stream: The stanford stream data manager. *IEEE Data Engineering Bulletin*, Vol. 26, No. 1, 2003.

[12] Jeong-Hyon Hwang, Magdalena Balazinska, Alexander Rasin, Ugur Cetintemel, Michael Stonebraker, and Stan Zdonik. High-availability algorithms for distributed stream processing. In *Prof. of the 21st International Conference on Data Engineering*, 2005

[13] Hideyuki Kawashima, Michita Imai, and Yuichiro Anzai. Providing persistence for sensor data streams by remote wal. In *Proc. DAWAK, pp. 524–533*, 2006.

[14] A. Das Sarma, M. Theobald, and J. Widom. Exploiting lineage for confidence computation in uncertain and probabilistic databases. In *Proc. of IEEE International Conference on Data Engineering*, 2008.

[15] 樫山俊彦, 花井知広, 田中美智子, 今木常之, 西澤格. ストリームデータ処理におけるデータベースアーカイブ処理高速化の提案と評価. 日本データベース学会 Letters, Vol. 6, No. 2, pp.37–40, 2007.

[16] 上條俊介. センサー融合による交通事象認識システムの構築. 情報処理学会研究報告. ITS, [高度交通システム] Vol.2006, No.103(20060928) pp. 81–86

[17] Shawn Bowers, Timothy McPhillips, Bertram Lud ¨ ascher, Shirley Cohen, and Susan B. Davidson. A Model for User-Oriented Data Provenance in Pipelined Scientific Workflows. *Lecture Notes in Computer Science, Volume 4145*, Provenance and Annotation of Data, 2006, pp. 133-147.

[18] Leon J. Osterweil, Lori A. Clarke, Aaron M. Ellison, Rodion Podorozhny, Rodion Podorozhny, Alexander Wise, Emery Boose, and Julian Hadley. Experience in Using a Process Language to Define Scientific Workflow and Generate Dataset Provenance. In *Proc. SIGSOFT 2008/FSE-16*, 2008.

[19] Jonathan Ledlie, Chaki Ng, David A. Holland, Kiran-Kumar Muniswamy-Reddy, Uri Braun, and Margo Seltzer. Provenance-Aware Sensor Data Storage In *Proc. icdew,pp.1189, 21st International Conference on Data Engineering Workshops (ICDEW'05)*, 2005.

[20] Adriane P. Chapman, H.V. Jagadish, and Prakash Ramanan. Efficient Provenance Storage. In *Proc. SIGMOD'08*, 2008.