ストリーム処理エンジンにおける複数書き込み最適化の提案

阿部 泰芽† 川島 英之†,‡ 北川 博之†,‡

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‡

1 はじめに

現在，各種センサーや GPS，ウェブカメラなどのデバイスの増加により，センシングデバイスから能動的に発信されるストリームデータが増加している．それに伴い，ストリームデータの永続化要求も高まってきている．ストリームデータを扱うアプリケーションの処理基盤としてこれまでに多くのストリーム処理エンジンが開発されてきたが，それらの多くは永続化要求に対応していない[1]，または対応しているものの性能評価が行われていない[2]など，ストリームデータの永続化に関してはあまり多くの議論はなされてこなかった．そこで，本研究ではストリーム処理エンジンおいてストリームデータを永続化する手法を提案する．

通常，ストリーム処理エンジンではシステムにデータが到着する度に主記憶上でクエリを評価する連続的問合せ[1]を行っている．しかし，全て主記憶上で処理を行う連続的問合せ処理に対し，ディスクアクセスを伴う永続化処理は低速であるため，永続化処理をストリームデータ処理と同期的に行うことは困難である．そのため，ストリームデータの永続化においては，永続化するデータ量を極力減らしディスクアクセスを減らすことが課題となる．そこで，本研究では複数の永続化要求における処理木の最適化を行う．ここで，本稿では射影演算子，選択演算子，及び永続化処理を行う書き込み演算子のみを扱う．この書き込み演算子の位置を最適に配置することでディスクアクセスを減らすことが可能であると考えられる．具体的には射影演算子同士，または選択演算子同士の重複関係に着目する．これらの演算子同士が重複関係にあるということは，永続化において同じデータをディスクに書き込んでいる事であり，永続化という点において冗長である．そのため，冗長な出力を持つ演算子同士を併合し，その結果を書き込み演算子により永続化する．さらに，元の永続化要求と整合するよう非同期的に書き戻し処理を行う．本手法を用いることにより，永続化処理において冗長なデータの書き込みを避けることが出来るため，ディスクアクセスを減少させ，永続化処理の高速化が可能となる．

A Proposal of Optimization Method of Multi Persistency Requirements for Stream Processing Engine

Taiga Abe†, Hideyuki Kawashima†‡, Hiroyuki Kitagawa†‡,

† Graduate School of Systems and Information Engineering, University of Tsukuba

‡ Center of Computer Sciences, University of Tsukuba

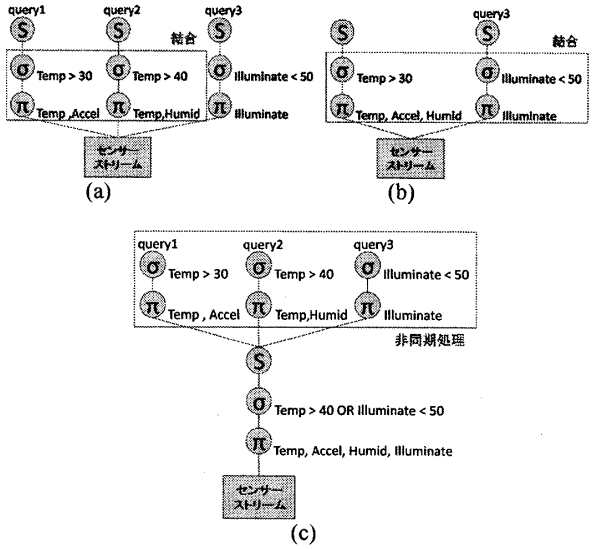

図 2.1　最適化例

2 複数書き込み最適化

ここでは，本稿で提案する複数書き込み最適化手法について説明する．本稿では射影演算子，選択演算子，及び書き込み演算子を対象とする．先に述べたように，永続化処理は連続的問合せ処理より低速なため，多くの永続化処理を同時に行うことは困難である．そのため，永続化処理においてはディスクアクセスを極力減らすことが重要となる．そこで，本研究では演算子同士の重複に注目し，永続化処理において重複したデータの書き込みを無くしディスクアクセスを最小化する手法を提案する．

2.1 概要

提案手法による最適化例を図 2.1 に示す．図中の S は書き込み演算，σ は選択演算，π は射影演算をそれぞれ表す．図 2.1(a)の例では，query1 の射影演算子は入力タプルの属性"Temp,Accel"を出力し，query2 の射影演算子は属性"Temp,Humid"を出力するため，属性"Temp"の分だけ重複した出力を持つ．また，query1 の選択演算子は入力タプルのうち属性"Temp"が 30 以上のタプルを出力し，query2 は 40 以上のタプルを出力するため，属性"Temp"の値が 40 以上のタプルが入力された場合に重複して出力する．重複したタプルは書き込み演算子 S によって永続化されるため，重複したデータ分だけ余計に永続化処理に時間がかかることになる．そこで，本研究では，図 2.1(b)のように重複した演算子同士を併合し，元の演算子の出力を全て含む出力を持つ演算子を作り，その結果を

書き込み演算子により永続化する．これにより，重複したデータを書き込む事がなくなるため，永続化処理の高速化が可能になる．また，query3 は他のクエリと重複した演算子を持たないため，演算子同士を併合しても書き込みデータ量を削減することはできないが，併合することによって演算子の数を減らすことができるため，連続的問合せ処理の高速化を図ることが可能である．さらに，このままでは元のクエリと整合しないため，図 2.1(c)のように元のクエリと整合するよう永続化した後に書き戻し処理を行う．この書き戻し処理はストリーム処理とは非同期的に行われる．併合された射影演算子は，属性"Temp,Humid,Accel,Illuminate"を射影し，併合された選択演算子は属性"Temp"が 30 以上か，"Illuminate"が 50 以下のタプルを選択する．併合された演算子は元の演算子の出力を全て含む出力を持つため，書き込み演算子による永続化時に，重複したデータを書き込む事を避けることができ，永続化処理の高速化が可能となる．

2.2 演算子の結合手順

重複した射影演算子同士，選択演算子同士の併合においては，併合された演算子の出力が元の演算子の出力を全て含むよう射影属性と選択条件を求める必要がある．以下にその手順を示す．

2.2.1 射影演算子の結合

併合された射影演算子の射影属性は以下のように求める．ここで，P_{new} を新しい演算子の射影属性集合，P_1~P_n を全クエリ中の n 個の射影演算子の射影属性集合を表す．

$$\{P_{new}\} = \{P_1\} \cup \{P_2\} \cup \ldots \cup \{P_n\}$$

新しい射影演算子の射影属性集合 P_{new} は元の射影演算子の射影属性を全て含むため，この出力を永続化することで重複した書き込みを避けることができる，

2.2.2 選択演算子の結合

結合された選択演算子の選択条件は以下のように求める．ここで，S_{new} は新しい演算子の選択条件，S_1~S_n は全クエリ中の n 個の選択演算子の選択条件を表す．

$$\{S_{new}\} = \{S_1\} \text{ OR } \{S_2\} \text{ OR } \ldots \text{ OR } \{S_n\}$$

上記のように，併合された選択演算子の選択条件は，元の選択演算子の選択条件を"OR"で繋ぐことで求められる．ただし，元の選択条件のうち，ある選択条件が別の選択条件を包含する場合には，包含する方の選択条件のみを用いる．図 2.1 の例では，query1 の選択条件が{Temp > 30}であり，query2 の選択条件が{Temp > 40}と，query1 の選択条件が query2 の選択条件を包含しているため，query1 の選択条件のみを用いている．

2.3 書き戻し手法

複数の演算子を併合すると，その出力結果を元の問合せに合うように分解して書き戻す必要がある．この書き戻しの手法には，書き戻し前のデータに索引を生成しておき，実際にユーザーからの検索を受けるまで書き戻し処理を行わないはどのアプローチが考えられるが，現在考案中であり，今後の課題とする．

表 3.1 実験環境

OS	Ubuntu 8.10
CPU	AMD Athlon(tm)64 X2 2.00GHz
メモリ	2GB
ファイルシステム	ext3
タプルサイズ	20(byte) (int×5:Time,Temp,Humid,Accel,Illuminate)
入力レート	100(tuple/sec)

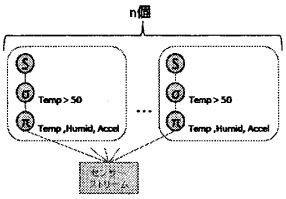

図 3.1 実験用処理木

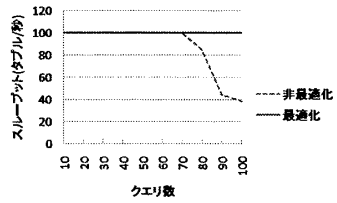

図 3.2 実験結果

3 シミュレーション

提案した最適化手法の有効性の検討のためシミュレーションを行った．実験環境を表 3.1 に示す．図 3.1 に示すように，クエリ数 n を 10~100 まで変化させ，その際のスループットを計測した．また，各タプルの値については 0~100 の間でランダムに決定した．結果を図 3.2 に示す．図より最適化を行わない場合，クエリ数が 70 を超えるとスループットが低下しているが，最適化を行った場合には低下していない．従って，提案手法が有用であることが示唆される．

4 まとめと今後の課題

本稿では，ストリーム処理エンジンにおいてストリームデータを効率的永続化する手法を代数的演算の観点から提案した．そして，シミュレーションによって提案手法の有用性を明らかにした．

今後の課題としては，システムの全容を明らかにすることが挙げられる．永続化したデータはユーザーが後から検索可能である必要があるため，物理的データ格納方式の検討や，永続化データへの問合せ言語の策定を行う．

謝辞

本研究の一部は，科学研究費補助金基盤研究(A)(#21240005)，若手研究(B)(#20700078)による．

参考文献

[1] 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 Info Lab. 2004.

[2] I.Botan, G.Alonso, P M.Fischer, D.Kossmann, and N.Tatbul. Flexible and Scalable Storage Management for Data-intensive Stream Processing. *ACM. EDBT*, Saint Petersburg, Russia, March 24-26, 2009