

# ストリーム管理システムにおける複数永続化要求最適化手法

山田 真一<sup>†</sup> 渡辺 陽介<sup>††</sup> 北川 博之<sup>†,†††</sup> 天笠 俊之<sup>†,†††</sup>

<sup>†</sup> 筑波大学 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

<sup>††</sup> 科学技術振興機構 戦略的創造研究推進事業

<sup>†††</sup> 筑波大学 計算科学研究センター

E-mail: {snic,watanabe}@kde.cs.tsukuba.ac.jp, {kitagawa,amagasa}@cs.tsukuba.ac.jp

あらまし 近年，デバイス技術やネットワーク技術の発達に伴い，自発的に配信されるストリームデータに対する高度利用要求に注目が集まっている．ストリームに対する様々な処理要求を実現するために，我々はストリーム処理エンジンと DBMS を連携させることによって，ストリーム管理システム Harmonica の開発を行っている．一般に，高速なメモリ上で動作するストリーム処理に比べて，DBMS に対する処理にはディスクアクセスが発生してしまうため，ストリームデータに対する複数の蓄積要求を並行して処理し続けることは非常に困難である．本稿では，こうしたボトルネックを解決するために，ストリームデータに対する複数の蓄積処理を実現するための最適化手法を提案する．本稿で提案する最適化手法は，DBMS の書き込み性能を上回った蓄積処理が発生しないように，複数の蓄積要求から共通する演算を抽出し，蓄積処理を共有化することによって DBMS への書き込みコストの肥大化を抑制する．また本稿では，提案手法によって生成された処理プランに基づいて，Harmonica システムで行った実証実験についても述べる．  
キーワード データストリーム，連続的問合せ，DBMS，最適化

## An Optimization Method for Multiple Persistency Requirements on Stream Management System

Shinichi YAMADA<sup>†</sup>, Yousuke WATANABE<sup>††</sup>, Hiroyuki KITAGAWA<sup>†,†††</sup>,  
and Toshiyuki AMAGASA<sup>†,†††</sup>

<sup>†</sup> Graduate School of Systems and Information Engineering, University of Tsukuba

<sup>††</sup> Core Research for Evolutional Science and Technology, Japan Science and Technology Agency

<sup>†††</sup> Center for Computational Sciences, University of Tsukuba

E-mail: {snic,watanabe}@kde.cs.tsukuba.ac.jp, {kitagawa,amagasa}@cs.tsukuba.ac.jp

**Abstract** Today, the amount of data delivered as data streams has been increasing, and requirements over data streams have a great variety. We have been developing a stream management system called Harmonica by combining a stream processing engine and DBMSs to fulfill such requirements. Generally, throughputs of DBMSs are slower than those of stream processing engines with a high-speed memory. It is very difficult to process a huge number of continuous queries which request the system to store data into DBMSs. In order to overcome this bottleneck, we propose an optimization method for multiple persistency requirements. The main idea of our optimization is to reduce the writing costs of DBMSs by sharing common store-operators. Our optimization method creates feasible processing plans. In this paper, we present the algorithm of our method and the results of experiment by using Harmonica System.

**Key words** data stream, continuous query, DBMS, optimization

### 1. はじめに

近年，デバイス技術やネットワーク技術の発展・普及によって，自発的に最新の情報を配信するストリーム型情報源（ス

トリーム）が増加してきている．例えば，データ放送，株価，ニュースといった情報配信サービスや，温度や光などを取得するセンサ，マルチメディアデータを配信するカメラやマイクなどがストリームとして考えられる．従来，ストリームに対する

要求としては、データの鮮度を重視し、新規にデータが到着する度にメモリ上で問合せ処理を行う連続的問合せ要求 [2] が主流であったが、ディスクの大容量化・低価格化が進んだ結果、到着したストリームデータを加工して蓄積しておきたいといった要求や、新規到着データと履歴データを統合したいといった要求に注目が集まっている。このような背景から、ストリームデータに対する連続的問合せ処理だけでなく、ストリームデータの蓄積・活用をより容易に行える基盤システムが非常に重要となってきた。現在、ストリームデータを扱うための基盤システムとして、ストリーム処理エンジン [2], [4], [6], [8] があるが、それらはいずれも新規到着データに対する連続的問合せ処理に特化したシステムである。そこで我々は、ストリーム処理エンジンと DBMS を連携させることで、ストリームデータの蓄積・活用を実現する基盤システムを目指しており、プロトタイプシステム Harmonica [1] の開発を行っている。

一般に、高速なメモリ上で処理を行うストリーム処理に対して、ディスクアクセスの発生する蓄積処理は非常に遅い。ストリームデータが到着する度にフィルタリングして順次蓄積するといった処理をし続けるためには、DBMS の書き込み性能を超えない頻度で蓄積処理を実行しなければならない。書き込み性能を上回る数の処理を扱うことはできないため、通常では大量の蓄積要求を同時に実現することは困難である。このような問題に対して本研究では、DB の性能に基づいた複数蓄積要求の最適化手法を提案する。

本研究で提案する最適化手法の特徴は、類似した複数の要求によって蓄積される冗長なデータに着目し、それらを 1 つにまとめて蓄積することによって、蓄積されるデータ量の調整を実現しつつ、利用者に要求されたデータをきちんと蓄積することである。基本的なアプローチとしては、複数蓄積要求をそのまま処理するのではなく、複数蓄積要求に含まれる共通演算の処理結果のみを蓄積し、各要求に固有な残りの演算は、読み出し時にビューとして処理するプランを生成する。そこで提案手法では、書き込み性能が許す限りは元の要求のまま処理するが、書込むデータ量が性能を上回る場合には蓄積処理の共有化を必要最小限だけ行う。共有化が必要かどうかを判断するためには、処理プランの処理可能性を判定する必要がある。本研究では [1] で我々が提案した処理可能性判定手法を拡張したものをを用いている。本稿では、処理可能性判定方法について説明すると共に、提案する最適化手法のアルゴリズムについて述べる。また、提案手法に従って生成したプランを用いて、実際に Harmonica 上で処理を行った。その結果、提案手法では 1200 個もの蓄積要求に対しても処理可能なプランを生成することが可能であり、実際に Harmonica で継続的な蓄積処理が可能であることを確認した。本稿では、その評価実験の詳細についても述べる。

以下は、次のような構成になっている。まず、2 節で Harmonica の概要について述べる。3 節で処理可能性判定手法について説明した後、4 節で提案する最適化手法について述べる。5 節で実験について説明し、6 節で関連研究を紹介する。最後に、7 節でまとめと今後の課題について述べる。

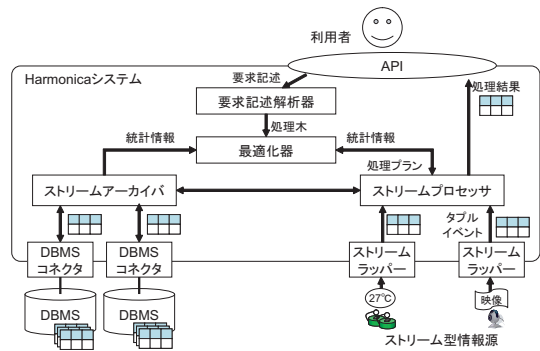


図 1 Harmonica のアーキテクチャ

Fig. 1 System architecture of Harmonica

```

MASTER Turbine
INSERT INTO Turbine_log
SELECT id, temp
FROM Turbine[1sec]
WHERE id = 1

```

図 2 要求記述例

Fig. 2 An example of query

## 2. プロトタイプシステム Harmonica

### 2.1 アーキテクチャ

Harmonica は到着データに対する連続的問合せ処理や、ストリームデータを DBMS へ書込む蓄積処理を実現する。Harmonica はストリーム処理エンジンと DBMS の連携を前提としており、リレーショナルデータモデルを採用している。

Harmonica はストリームプロセッサ (ストリーム処理エンジン)、ストリームアーカイバ、要求記述解析器、最適化器、DBMS コネクタ、ストリームラッパーによって構成されている (図 1)。利用者や応用プログラムからの要求は要求記述解析器によって処理木へと変換される。最適化器は、ストリームの配信間隔などの統計情報を元に、複数の処理木から最適な処理プランを生成する。具体的な生成方法については 4 節で説明する。ストリームラッパーは、情報源独自のデータ形式をテーブルに変換する役割を持っている。テーブルに変換されたデータはデータ到着イベントと共にストリームプロセッサに送られる。ストリームプロセッサでは最適化器で生成される処理プランに従って、ラッパーから通知されるイベントに連動して処理を行う。処理結果は直ちにストリームアーカイバや応用プログラムに送信される。ストリームアーカイバは、各 DBMS に対応した DBMS コネクタを通して、テーブルの書き込み・読み出しを行う。

現在は、我々の研究グループで開発を行っているストリーム処理エンジン StreamSpinner [2], [9] と、DBMS として MySQL や PostgreSQL を用いて構成されている。

### 2.2 要求記述言語

Harmonica ではストリームに対する各種要求を記述するために、SQL ライクな要求記述言語 HamQL を提供している。Harmonica では連続的問合せ [2] を処理する。連続的問合せとは、前回実行時から新規に到着したデータに対して繰り返し演算を評価し、前回実行時からの差分の生成を行う問合せ処理方

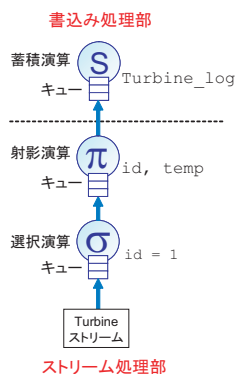


図 3 処理木の例  
Fig. 3 Processing tree

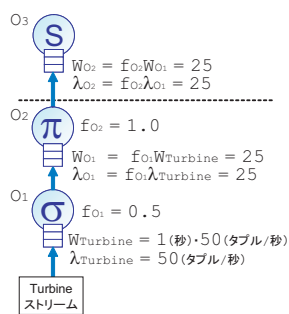


図 4 コストの見積り  
Fig. 4 An example of estimating costs

式である．利用者はマスタ情報源とウィンドウを指定することで連続的問合せ要求を記述する．マスタ情報源とは処理実行のきっかけを与えるストリームのことを言い、ウィンドウとは演算の適用対象となるタプルを選択するための時間幅のことを言う．

ここでは、ガスタービン発電所施設のモニタリングの例を使って、HamQL の記述例を説明する．ガスタービン発電所には温度や回転数などを計測するためのセンサが複数付属しており、毎秒取得したセンサデータが Turbine ストリームとして管理センタに届けられている．このとき例えば、管理者は管理センタにデータが到着する度に、ID が 1 のガスタービンに付けられた温度センサのデータのみを Turbine\_log テーブルに蓄積したいと考えている．このような要求は図 2 のように記述することができる．MASTER 節ではマスタ情報源を指定する．この例では、データが到着する度に処理を行いたいという要求なので、MASTER 節には Turbine を指定している．その他に関しては SQL とほぼ同様であるが、FROM 節では角括弧”[ ]”によってウィンドウを指定することができる．図 2 の要求では Turbine ストリームに対して、問合せの評価時刻を起点とした 1 秒幅のウィンドウを指定している．したがって図 2 の問合せの評価時には、最近 1 秒間に到着した Turbine ストリームのタプルのうち、まだ未評価のタプルが蓄積処理の対象となる．

HamQL では、図 2 のようなデータが到着するたびに DBMS へ連続的に書込んでほしいという要求を記述できる．本研究では、こういった連続的な蓄積要求が大量に与えられる状況を想定している．

### 3. 処理可能性判定

提案システムでは連続的問合せによる蓄積処理や新規到着データと蓄積データの統合活用を目的としているが、本稿では特に蓄積できずにデータが損なわれてしまう問題をより重大と考え、DBMS に対する書き込み処理に焦点を当てて最適化を行う．本稿で提案する最適化手法は、登録された複数蓄積要求の処理が DBMS の書き込み性能を上回る場合に、蓄積処理の共有化によって処理可能なプランを生成するものである．従って、登録された複数蓄積要求に対して最適化が必要かどうかを判断するには、それらが処理可能であるかを判定する必要がある．

本節では、複数蓄積要求に対する処理可能性判定手法について述べる．基本的な枠組みは Ayad らのコストモデル [5] をベースに蓄積処理判定用に拡張を行った [1] の手法に従う．しかし [1] で提案した判定手法では 1 つの蓄積要求に対する判定しか行うことができない．そこで、本稿ではさら [1] の判定手法を複数蓄積要求に対する判定ができるように拡張する．3.1 で判定手法の前提となるコストモデルについて述べ、3.2 で具体的な判定手法について説明する．

#### 3.1 コストモデル

ここでは、全てのストリームデータはほぼ一定間隔で到着し、時間によって頻度やデータ量が大幅に変化しないものとする．一般的にはストリームデータは、データの到着間隔や到着するデータ量が変動するものも含むが、実世界のモニタリングに利用されるセンサやカメラなど一定間隔でデータを配信する情報源もたくさん存在する．提案手法では、そうした一定間隔で配信されるストリームデータを対象としている．また、同じマスタ情報源からのデータ到着によって処理される要求の集合を対象とする．実世界のモニタリングなどでは周期的な監視要求が多く、こうした仮定を置いても十分実用的なアプリケーションが存在すると考えられる．

システムに登録された要求は、要求処理解析器によって処理木へと変換される．処理木には、その要求を処理するためにシステムが行なうべき演算の順番が定義されている．例えば図 2 の要求は、図 3 に示す処理木へと変換される．2 節で紹介したように、Harmonica はストリームプロセッサと DBMS を連携することで各種処理を行っている．従って図 3 に示す処理木は、ストリームプロセッサが扱うストリーム処理部と DBMS が扱う書き込み処理部に分けられている．それぞれの処理部は別々のモジュールによって処理されるので、本コストモデルでは、ストリーム処理にかかるコストと書き込み処理にかかるコストを分けて考える．

本コストモデルでは、まずストリームの到着レート(単位時間あたりに到着するタプル数)とウィンドウに含まれるタプル数の見積り値から、各演算の出力レート(単位時間あたりに出力するタプル数)をボトムアップに推定する．次に、推定した各演算の出力レートから全体の処理にかかるストリーム処理部のコストや書き込み処理部のコストの計算を行う．以下では、各処理部でのコストの推定方法について簡単に紹介する．

##### 3.1.1 ストリーム処理部のコスト

ストリーム処理部では、リレーショナルモデルの選択・射影・直積・結合演算を扱う．見積もるべき演算の出力レートを  $\lambda_o$ 、出力先のキューにおいてウィンドウに含まれるタプル数(出力ウィンドウタプル数)を  $W_o$  とする．見積もりに用いる情報は、演算の選択率  $f$ 、演算の入力レート(入力となる演算  $i$  の出力レート)  $\lambda_i$ 、入力キューにおいてウィンドウに含まれるタプル数(入力ウィンドウタプル数)  $W_i$  とする．選択演算の場合、出力レートは  $\lambda_o = f\lambda_i$ 、出力ウィンドウタプル数は  $W_o = fW_i$  で推定できる．また、単位時間当たりの選択演算の処理時間は  $\tau_\sigma \lambda_i$  と計算する． $\tau_\sigma$  は 1 タプルの選択処理にかかる時間でストリームプロセッサから取得する．本稿では、単位時間あたり

にかかるストリーム処理部での演算の処理時間のことを処理コストと呼ぶ。射影演算の場合は選択演算の  $f = 1$  の場合と等価とみなして推定する。

結合演算では2つ入力を考える。2つの入力ストリームの入力レートを  $\lambda_L, \lambda_R$ 、入力ウィンドウタブル数を  $W_L, W_R$  とすると、結合演算の出力レートは  $\lambda_o = f(\lambda_L W_R + \lambda_R W_L)$ 、出力ウィンドウタブル数は  $W_o = f W_L W_R$  と推定される。また、処理コストは  $\tau_M(\lambda_L + \lambda_R)$  によって計算できる。ここで  $\tau_M$  は1タブル当たりの結合演算にかかる処理時間を示している。直積演算の場合は選択率  $f = 1$  の結合演算として考える。

### 3.1.2 書込み処理部のコスト

書込み処理部では蓄積演算を処理する。ストリーム処理部で生成されたデータの入力レートを  $\lambda_i$ 、1タブルの蓄積処理にかかる時間を  $\tau_{DB}$  とすると、単位時間あたりにかかる書込み処理時間は  $\tau_{DB}\lambda_i$  と表すことができる。 $\tau_{DB}$  は書込みレート(単位時間あたりに書き込み可能な最大のタブル数)の逆数によって求める。本稿では蓄積要求に対して単位時間あたりにかかる書込み処理部での処理時間を書込みコストと呼ぶ。

書込みレートはディスクの書込み速度やDBMS内部の書込み方式の違いなどに依存するパラメータなので、一意に定めることは非常に難しい。さらに、書き込むタブルサイズの増加に反比例して書込みレートが低下する傾向がある[1]ので、本研究ではサイズの異なる複数のタブルで計測を行い、未計測のサイズの書込みレートについては最も近い計測済みのサイズ2点によって近似する。これらについては[1]で提案したものであるため詳細は省く。

### 3.2 処理プランの処理可能性

ここでは処理プランの処理可能性について述べる。処理プランは複数の要求から作られた処理木の集合であり、1つの処理プランには複数のストリーム処理部と書込み処理部が含まれている。3.1で述べたように、処理コストと書込みコストはそれぞれ別々に考えることができるため、処理可能性もストリーム処理部と書込み処理部で分けて考える。

まず、ストリーム処理部の処理可能性について定義する。連続的問合せを処理し続けるためには、データが到着してから次のデータが到着するまでの間に今回の到着データに対する処理が完了していることが必要である。したがって、次のように定義できる。

[定義 3.1] 処理プラン  $P$  に含まれるストリーム処理部の演算  $O_i (0 \leq i \leq n)$  の処理コストを  $c_i$  とするとき、以下の条件を満たす  $P$  をストリーム処理可能と定義する。

$$\sum_{i=0}^n c_i < 1 \quad (1)$$

式(1)はストリーム処理部の各演算の処理コストの和が単位時間内に納まらなければならないことを表している。 $\sum c_i$  を  $P$  全体の処理コストとし  $C_{SP}$  と表記する。

次に、書込み処理部の処理可能性について定義する。ストリーム処理の結果を逐次DBに書き込むためには、ストリーム処理部での処理結果が生成されたときに、前回の書込み処理部

での処理が完了していることが必要となる。したがって、書込み処理部での処理可能性も同様に定義可能である。

[定義 3.2] 処理プラン  $P$  に含まれる蓄積演算  $S_j (0 \leq j \leq m)$  の書込みコストを  $c_j$  とするとき、以下の条件を満たす  $P$  を書込み処理可能と定義する。

$$\sum_{j=0}^m c_j < 1 \quad (2)$$

以降  $\sum c_j$  を  $P$  の書込みコストとし  $C_{DB}$  と表記する。

以上より、処理可能性は次のように定義できる。

[定義 3.3] 処理プラン  $P$  がストリーム処理可能かつ書込み処理可能のとき、 $P$  を処理可能と定義する。

### 3.3 処理可能性判定の計算例

具体的な処理可能性判定例として、図4の処理プランを用いて説明する。Turbine ストリームとして毎秒50タブルが配信されているとする。また、選択演算の選択率  $f_{o1}$  を0.5、選択演算・射影演算の1タブルあたりの平均処理時間  $\tau_{o1}, \tau_{o2}$  をそれぞれ0.001sとする。

まず、与えられた統計情報からストリームの到着レートとウィンドウを設定する。図2の要求ではTurbine ストリームに対して1秒間の時間幅のウィンドウが指定されているので、時間幅と到着レートによって入力ウィンドウタブル数を見積もる ( $\lambda_{Turbine} = 50, W_{Turbine} = 1 \cdot 50 = 50$ )。次に、各演算の出力レートと出力ウィンドウタブル数を計算する ( $\lambda_{o1} = f_{o1}\lambda_{Turbine} = 0.5 \cdot 50 = 25, W_{o1} = f_{o1}W_{Turbine} = 0.5 \cdot 50 = 25, \lambda_{o2} = f_{o2}\lambda_{o1} = 25, W_{o2} = f_{o2}W_{o1} = 25$ )。最後に処理コストと書込みコストの計算を行い、処理可能性を判定する。それぞれの演算の処理コストは  $c_{o1} = \tau_{o1}\lambda_{Turbine} = 0.001 \cdot 50 = 0.05, c_{o2} = \tau_{o2}\lambda_{o1} = 0.001 \cdot 25 = 0.025$  であるので、このプランの処理コストは  $C_{SP} = c_{o1} + c_{o2} = 0.075 < 1$  となりストリーム処理可能である。また、DBMSの書込みレートが  $\lambda_{DB} = 40$  であったとすると、書込みコストは  $C_{DB} = \tau_{DB}\lambda_{o2} = \lambda_{o2}/\lambda_{DB} = 25/40 = 0.625 < 1$  となり、このプランは書込み処理可能であることがわかる。以上より、この処理プランは処理可能であると言える。

本研究で提案する最適化手法は、処理プランの書込みコストが  $C_{DB} > 1$  となってDBMSの性能を上回ってしまう場合に、蓄積演算の共有化によって  $C_{DB} < 1$  となる処理プランを導出する手法である。

## 4. 複数蓄積要求最適化手法

複数の蓄積要求に対する継続的な処理を実現するためには、蓄積するタブル数が書込みレートを超えない処理プランを生成する必要がある。蓄積するタブル数を減らす既存の方法としてはload shedding [4], [5] があるが、load shedding では重要なデータが失われてしまう可能性がある。可能な限りデータが失われることなくきちんと蓄積され、読み出し時には要求通りのタブルが取得できることが望ましい。提案手法では、複数の蓄積要求がシステムに登録されることによって蓄積処理が困難となる場合に、要求通りのタブルが読み出し時にきちんと取得で

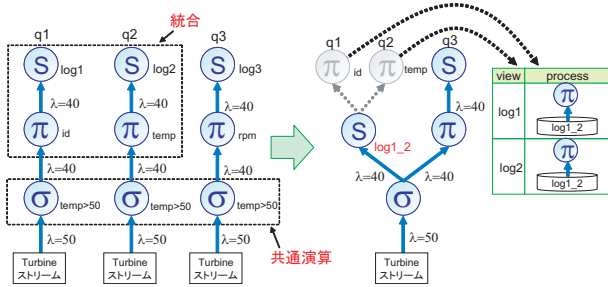


図 5 最適化例

Fig. 5 Example of optimization

きることを保証しつつ、生成されるタプル数を減少させるプランを生成する。4.1 で提案手法の基本概念について説明し、4.2 以降で最適化手順の詳細について述べる。

#### 4.1 基本概念

提案手法における最適化例を図 5 に示す。図 5 左では、 $q_1 \sim q_3$  の蓄積要求がある。Turbine ストリームの到着レートが毎秒 50 タプルで、 $q_1 \sim q_3$  それぞれで 40 タプルにフィルタリングされて DB へ蓄積されるとする。このプランに従って処理を行うと、毎秒合計で 120 タプルの蓄積処理が必要となる。こゝでもし、DBMS の書込みレートが毎秒 100 タプルだとすると、毎回の処理で 20 タプルずつ蓄積待ちタプルが増加し、いつかはキューがあふれてしまう。そのため、この処理プランは処理可能ではない。そこで、提案手法では処理プランに含まれる共通演算に注目する。図 5 左の例では、要求  $q_1 \sim q_3$  の選択演算が共通している。共通演算では、全く同じ内容のタプルが生成されるので、その処理結果さえ保存されていれば要求  $q_1 \sim q_3$  を満たすテーブルを後から生成することが可能である。よって、個々の要求に対応したテーブルへデータを蓄積する代わりに、共通性の高い中間結果を蓄積するというアプローチが考えられる。これにより、同じタプルの冗長なデータ書込みが減り、また、書込み処理の回数そのものを減らすことができる。

提案手法では、処理プランを書換えて、共通演算の上までそれぞれの蓄積演算を移動させる。そして移動させたそれぞれの蓄積演算を統合し、蓄積するタプルを共有化することで、DBMS に書込むタプル数を減少させる。図 5 右は、 $q_1$  と  $q_2$  の選択演算の先に蓄積演算を移動し、それぞれの蓄積データを共有している例である。図 5 右の処理プランの場合、蓄積タプル数は毎秒 80 タプルにまで減少し、DBMS への書込みレートを下回ったため、継続的な処理を実現することが可能となる。

もちろん提案手法の副作用として、統合した蓄積演算以降の処理は、共用テーブルに対するビュー生成の処理として読み出し時に行わなければならない。図 5 左の蓄積要求の  $\log_1, \log_2$  に対応するデータを図 5 右から取得するには、共用テーブル  $\log_{1,2}$  に対して残りの射影演算を適用する必要がある。共用テーブルから本来のデータを取得するコストをビュー処理コストと呼ぶ。

#### 4.2 最適プランの定義

書込み処理を減らすため蓄積演算の統合は必要だが、それによってビュー処理コストが増加しすぎると、DBMS からの読

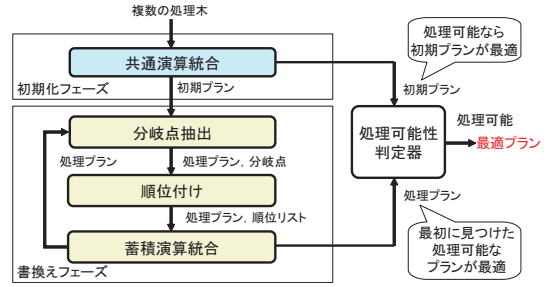


図 6 最適プラン導出の流れ

Fig. 6 Flow of optimization

出し時に多くの処理が発生する。提案システムでは蓄積データを後から別の処理要求で活用することを目的としているため、DBMS の読出しに多くの処理が発生するとシステムパフォーマンスが低下してしまう。そこで本研究では、処理可能かつビュー処理コストが最小となる処理プランを最適プランと定義する。しかし、ビュー処理コストは、共用テーブルから本来のデータを読み出すときに発生するコストであるが、データ読み出しがいつどのような頻度で発生するかを予測することは困難であるため、ビュー処理コストを正確に計算することはできない。そこで本手法では、ビュー処理に必要な総演算数をビュー処理コストとして扱う。最適化器では、システムに登録されている全ての処理木を入力として受け取り、最適プランを出力する。最適化器で出力されるプランは各要求の処理木をマージした DAG (非循環有効グラフ) と、読み出し処理時に行うビュー処理を示すビュー対応表で構成されている。ビュー対応表には、統合する前の蓄積演算が書込むはずだった本来のテーブル名とビューとして行う処理がペアで格納されている。

#### 4.3 最適プラン導出の流れ

本研究で提案するアルゴリズムについて説明する。ここでは、処理プラン内で共通演算から各要求固有の演算へと枝分かれている箇所を分岐点と呼び、また、分岐点から各要求の最後の演算までの経路を固有パスと呼ぶものとする。一般に、処理プランには分岐点が複数存在する可能性がある。最適化器では、ビュー処理コストが最小となる処理可能なプランを作成するために、分岐以降の演算数が最小となる分岐点から順に蓄積演算の統合を行い、新たな処理プランを構築する。そして、最初に見つけた処理可能な処理プランを最適プランとして出力する。

最適化器で行う最適プラン導出の流れを図 6 に示す。最適化器では、複数の処理木を入力として受け取り、最適プランを出力する。最適化器は内部に処理可能性判定器を持ち、初期化フェーズと書換えフェーズという 2 つのフェーズに分かれて最適プランの導出を行う。それぞれのフェーズの処理の概要について以下で述べる。

##### (1) 初期化フェーズ

まず、初期化フェーズでは共通演算統合モジュールで複数の処理木を元に初期プランを生成する。共通演算統合モジュールは、既存の複数問合せ最適化手法 [2] を利用して複数の処理木に共通して出現する部分木を検出し、ストリーム処理部での処理コストが最小となるように木の統合を行って DAG を生成する。

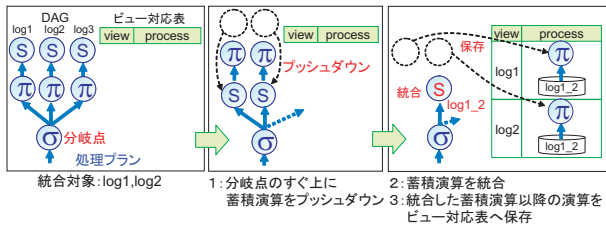


図 7 蓄積演算の統合例

Fig. 7 Integrating store operators

生成された DAG と空のビュー対応表が初期プランとなる．生成された初期プランは処理可能性判定器によって，3 節で示した方法によって処理可能性が判定される．初期プランのままでは処理可能ならビュー処理コストは 0 なので，そのプランを最適プランとして出力する．

## (2) 書換えフェーズ

初期プランが処理可能でない場合，書換えフェーズによって蓄積演算を統合し処理可能なプランを生成する．書換えフェーズでは，分岐点抽出・順位付け・蓄積演算統合という 3 つのステップを順に経由することで処理プランを更新する．処理プランが処理可能となるか，または全ての蓄積演算を統合し尽くすまで繰り返し更新が行われ，処理可能なものが見つかった場合には最適プランとして出力される．初期化フェーズから送られてくる初期プランは，最初の処理プランとして設定される．各ステップの処理内容を簡単に説明する．

### (2.1) 分岐点抽出ステップ

処理プランを受け取り，処理プランの中から着目すべき分岐点を 1 つ出力する．ビュー処理コストの低いプランを導出するため，分岐以降の合計演算数が最も少ない分岐点を選択する．分岐点が 1 つ選ばれる度に下の 2 ステップが実行され，処理プランが逐次更新されていく．一度評価を行った分岐点は再選択しない．未評価の分岐点が存在しなくなっても処理可能なプランにならない場合には，これらの蓄積要求に対する処理可能なプランは存在しないと結論を出す．

### (2.2) 順位付けステップ

ビュー処理コストの増加を抑えるため，分岐点以降の蓄積演算を一度にすべて統合するのではなく，段階的に 1 つずつ統合し，処理可能性を判定していく．このステップでは，統合のための優先順位を決定する．前ステップで選択された分岐点と処理プランを入力として受け取り，その分岐における統合候補となる固有パスの優先順位リストを出力する．具体的な順序付け手法は 4.5 で述べる．

### (2.3) 蓄積演算統合ステップ

順位付けステップの優先順位リストと処理プランを受け取り，更新した処理プランを出力する．ここでは，受け取った優先順位リストに従って順に蓄積演算を統合する．蓄積演算の統合方法については 4.4 で述べる．1 つ統合する度に処理可能性判定器で現在のプランの処理可能性が判定され，処理可能であればそのプランを最適プランとして出力する．受け取った優先順位リストの蓄積演算を全て統合しても，処理可能なプランになら

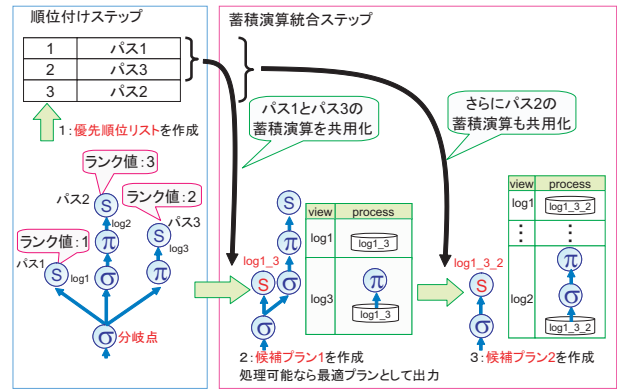


図 8 候補プランの生成例

Fig. 8 Example of generating candidate plans

ないときは，分岐点抽出ステップへ戻る．

## 4.4 蓄積演算の統合方法

蓄積演算の統合方法について具体的な例を挙げて説明する (図 7)．ここでは，テーブル log1 と log2 に蓄積を行う蓄積演算を統合するものとする．まず，それぞれの蓄積演算を分岐点 (= 選択演算) のすぐ上までプッシュダウンする (図 7 真ん中)．次に，それぞれの蓄積演算を統合して，現在の蓄積演算とは別のテーブルに蓄積する蓄積演算に置き換える．最後に，統合した蓄積演算以降の残りの演算をビュー対応表に保存する (図 7 右)．読み出しは本来のテーブル名で要求されるので，そのテーブル名をキーとしてビュー対応表を検索することで本来のテーブルの生成に必要な処理を知ることができる．以上の操作により蓄積演算が統合され，書込みコストを下げた新たな処理プランが生成される．

図 7 では一度も統合されていない蓄積演算を統合する例であるが，実際には蓄積演算の統合を繰り返すことによって，一度統合を行った蓄積演算をさらに統合する場合も存在する．その場合も，図 7 の場合と同様に蓄積演算を統合し，ビュー対応表にその蓄積演算の行が追加される．読み出しの際には，再帰的にビュー対応表のテーブル名を辿っていくことで元のテーブルを生成するまでの処理を復元することが可能である．

## 4.5 順位付け手法

ここでは，順位付けステップの優先順位リスト生成手順について説明する．順位付けステップは，分岐点抽出ステップによって選択された分岐点を基準に統合候補となる固有パスの優先順位リストを生成する．

基本的に分岐点から蓄積演算までに経由する演算が多いほど，ビュー処理コストが増加する．そこで本手法では，処理プランの分岐点から蓄積演算までの固有パスの長さをランク値として昇順にソートして順位付けを行う．ただし，提案手法では分岐先と蓄積演算の間に結合・直積演算が存在するパスを予め排除し，優先順位リストには加えない．これは，分岐点と蓄積演算の間に結合・直積演算を含む場合，蓄積演算を結合・直積演算の前に移動してしまうと，本来の要求のテーブルを復元するために，結合・直積演算のもう一方の入力となるテーブルも蓄積する必要があり，蓄積演算の数が増加するからである．

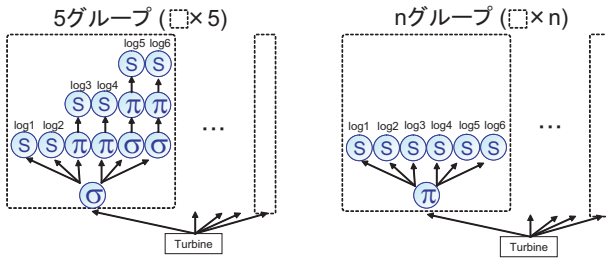


図 9 実験用初期プラン

Fig. 9 Initial plans for experiments

順位付けとプラン生成の例を図 8 を使って説明する．まず，分岐点から蓄積演算の固有パスの長さを計算し，各固有パスのランク値を設定する．ここからランク値の低い順に固有パスの順位付けを行うことで優先順位リストを作成し，蓄積演算統合ステップへと送られる．蓄積演算統合ステップでは 4.4 で述べた方法によって，優先順位リストに従ってまず  $\log_1$  と  $\log_3$  に書込む蓄積演算を統合する（候補プラン 1）．このとき，仮に候補プラン 1 が処理可能と判定されれば，候補プラン 1 が最適プランとなる．そうでなければ， $\log_1$ ， $\log_2$ ， $\log_3$  の全ての蓄積演算を統合した候補プラン 2 を作成する．候補プラン 2 まで作成しても処理可能なプランがなければ，候補プラン 2 を処理プランとして分岐点抽出ステップへと進む．

## 5. 実 験

実験の目的は，本手法によって得られた最適プランで実際に継続的な処理が行えるかの検証と本手法の性能評価である．

### 5.1 セットアップ

本実験は，CPU が Pentium D 3.00GHz，メモリが 2GB のマシンを使い，OS として Windows Vista Business，DBMS として MySQL を用いた．また，実験に使ったプログラムは JDK6 によって開発した．

本実験のために，擬似データを配信する Turbine ストリームを準備し，図 9 に示す 2 つの初期プランを作成した．初期化フェーズでは既存の手法を用いているだけなので，初期プランは人手で作成したものを利用する．図 9 左は，蓄積要求 30 個から構成される初期プランで，6 つの蓄積要求が 1 つの選択演算を共有しており，共有している選択演算によって 5 つのグループに分けられている．分岐点から先は選択演算の結果を蓄積する要求が 2 種類，異なる射影演算を行って蓄積する要求が 2 種類，さらに選択・射影と行って蓄積する要求が 2 種類ある．図 9 右は，6 つの蓄積要求が 1 つの射影演算を共有し，共有している射影演算によって  $n$  個のグループに分けられている．射影演算を共有している 6 つの蓄積演算はそれぞれのテーブルに結果を蓄積する要求となっている．射影・選択演算にかかる 1 タブル当たりの処理時間は 1ms と設定し，選択率はいずれも 1.0 とおいた．書き込みレートは実際に MySQL 上で計測したものを使った．

### 5.2 実験 1

本実験では，最適プラン導出に必要な処理プランの書換え回数を測定した．Turbine ストリームの到着レートを 2 タブル/秒

と設定し，初期プランとして図 9 左を用いた．また，MySQL の書き込みレートを実測した結果 35.9 タブル/秒であった．実験結果を図 10 に示す．横軸には書換え回数を取り，縦軸には各プランの書き込みコスト・処理コストが示されている．グラフより提案手法では，書き込み処理可能な条件である書き込みコストが 1 以下になるまで 13 回の書換えが行われ，その都度書き込みコストが下がっていることが確認できた．初期プランでは，蓄積演算が 30 個あるために毎秒 60 タブルの蓄積が必要である．しかし，提案手法では DB の書き込みレートに対して，毎秒 34 タブルの蓄積（蓄積演算数 17 個）が必要となるプランを生成した．また図 11 は，このときの実験用プログラムの出力である．図 11 の R はルートノードを示しており，要求の終点を表している．これらより提案手法では，全ての蓄積演算を統合するのではなく，必要最低限の書換えで処理可能となるプランが作成できていると考えられる．

### 5.3 実験 2

実験 2 では最適化したプランをプロトタイプシステム上で実行し，実際に継続的な処理ができるかどうかの検証を行った．Turbine ストリームの到着レートは 2 タブル/秒に設定し，初期プランとしては図 9 右のプランで蓄積演算数 30 個（グループ数 5）のものを用いた．また要求処理は毎秒実行されるようにマスタ情報源を設定した．図 12 に実験結果を示す．横軸は実行時間，縦軸は DBMS への書き込み待ち行列の長さである．最適化していないプランは，開始して約 210 秒後に待ち行列がおよそ 4600 にまで到達し，待ち行列があふれて継続的な処理を行うことができなかった．提案手法による最適化プランは，30 分間動かし続けても待ち行列の長さに見立った増加がなく継続的な処理が実現されていることが確認できる．さらに蓄積演算数 1200 個（グループ数 200）の場合の最適化プランに対しても検証を行った（図 13）．グラフではおよそ 5 時間分のデータしか表示していないが，実際に 24 時間動かし続けても待ち行列があふれることなく処理を継続できることを確認した．

### 5.4 実験 3

最後に提案アルゴリズムの性能について評価した．図 14 は，Turbine ストリームの到着レートを様々な値に設定したときの本手法が導出するプランの書き込みコストの変化を調査した結果である．初期プランとしては図 9 左に示す初期プランを用いた．図 14 は横軸が到着レート，縦軸が書き込みコストを表している．単位時間あたりの書き込み時間の見積もり値である書き込みコストが単位時間 (=1) を超えなければ，そのプランは書き込み処理可能である．図 14 より，Turbine ストリームの到着レートを様々な変化させても，DBMS の書き込み性能を超えないプランが導出されていることを確認した．

また，図 9 右の初期プランを用いることで最適プラン生成までの処理時間を計測した（図 15）．図 15 は蓄積演算数を 150～1200 個（グループ数は 25～200）まで増加させながら，それぞれ 5 回の平均の処理時間を計測したものである．提案手法では，1200 個もの蓄積要求に対してもおよそ 3 分で最適プランが導出可能である．

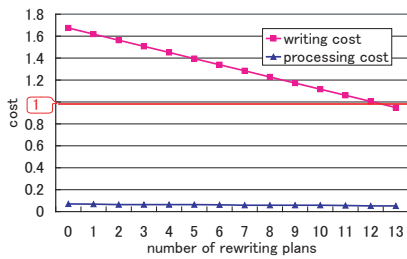


図 10 処理プランの書き換え回数

Fig. 10 The number of rewriting plans

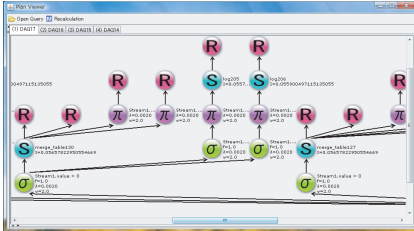


図 11 実験用プログラムの出力

Fig. 11 The output of experimental program

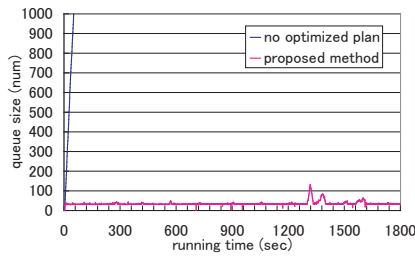


図 12 実行時間と書き込み待ち行列

Fig. 12 Processing time vs. store queue size

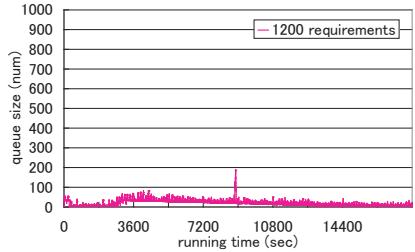


図 13 1200 要求の処理

Fig. 13 Processing 1200 requirements

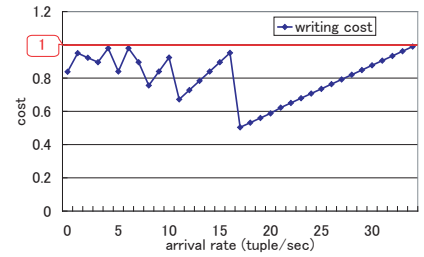


図 14 到着レートと書き込みコスト

Fig. 14 Arraival rate vs. writing cost

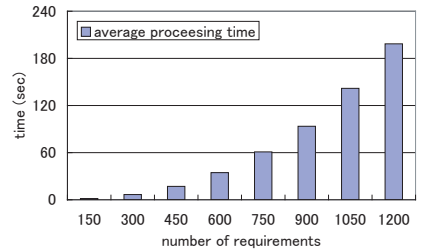


図 15 最適プラン導出時間

Fig. 15 The time of creating plan

## 6. 関連研究

大量の要求がシステムに登録されリソースを超えてしまうときに、それに対処する手法として load shedding [4], [5] がある。load shedding は処理するデータ量を減らすことで、計算機の性能を上回らないように問合せ処理を継続することが可能である。本手法では、蓄積するデータを共有することで、多くの蓄積要求を処理できることを目的としている。類似した要求が多く、到着レートやデータ量が大きく変動しないセンサなどのストリームに対して本手法は有効である。今後の拡張として、提案手法と load shedding を組合せて、動的なストリームに対しても複数蓄積要求最適化を実現することが考えられる。

STREAM [8] では、リレーションとストリーム処理を考慮しているが、到着データを逐次蓄積するといった機能については触れられていない。

KRAFT [3] はストリームに対する連続的問合せと蓄積処理を実現するシステムである。ストリームデータの高速な蓄積処理が可能であるが、複数の蓄積要求に対する最適化については触れられていない。

DBMS に対するビューを用いた最適化を行っている研究として [7] がある。[7] は頻りに結合問合せが発生する部分をあらかじめ実体化ビューとすることで、問合せ処理の高速化を目指している。しかし [7] は、ストリーム処理を目指したものではなく、蓄積要求が大量に発生したときの問題の解決を実現するものでもない。

## 7. おわりに

本研究では、ストリーム処理エンジンと DBMS を連携したデータベース管理システム Harmonica の開発を行っている。本稿では、Harmonica 上で大量の蓄積処理要求に対する永続化

処理を実現するための最適化手法について述べた。本稿で提案した最適化手法は、複数の蓄積処理を統合することによって、DBMS に書き込むタプル数を減らすことを目指したものである。本稿ではまた、Harmonica 上で実際に処理を行い提案手法の有効性を示した。今後の課題としては、動的に変化するストリームに対する拡張や、読出し処理に対するコストモデルの拡張があげられる。

謝辞 本研究は、科学研究費補助金基盤研究(A)(#18200005)、科学技術振興機構 CREST「自律連合型基盤システムの構築」による。

## 文 献

- [1] 山田真一, 渡辺陽介, 北川博之, 天笠俊之. "ストリーム管理システムにおける永続化要求の妥当性評価", 電子情報通信学会技術研究報告 Vol.106, No.149, pp.209-214.
- [2] 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-D-I, No.10, pp.873-886, 2004年10月.
- [3] 川島英之, 今井倫太, 遠山元道, 安西祐一郎. "センサデータベースシステム KRAFT の設計と実装", 情報処理学会論文誌, Vol.45 No.SIG14(TOD 24), pp.39-53, 2004年12月.
- [4] D. Abadi et al. "Aurora: A New Model and Architecture for Data Stream Management" VLDB Journal, (12)2:120-139, August 2003.
- [5] A. M. Ayad and J. F. Naughton. "Static Optimization of Conjunctive Queries with Sliding Windows Over Infinite Streams", Proc. of the ACM SIGMOD'04, pp.419-430, 2004.
- [6] S. Chandrasekaran et al. "TelegraphCQ: Continuous Dataflow Processing for an Uncertain World", Proc. Conference on Innovative Data Systems Research 2003.
- [7] H. Mistry et al. "Materialized View Selection and Maintenance Using Multi-Query Optimization", Proc. of the ACM SIGMOD'01, pp.307-318, May 2001.
- [8] R. Motwani et al. "Query Processing, Resource Management, and Approximation in a Data Stream Management System", Proc. Conference on Innovative Data Systems Research 2003.
- [9] StreamSpinner. <http://www.streamspinner.org/>