データストリーム管理システムHarmonicaの設計と実装

山田真一[†,☆] 渡辺陽介[††]
北川博之[†,†††] 天笠俊之[†,†††]

近年，センサデータなど実世界から得られるストリームデータが増加し，それらに対する高度利用要求に注目が集まっている．ストリームデータに対する要求としては，従来は新規到着データに対する連続的な問合せ要求が主流であったが，ストレージの大容量化や低価格化が進んだ結果，ストリームデータの蓄積要求や，新規到着データと蓄積データとの統合要求が重要となってきている．そこで本論文では，ストリームデータに対する各種要求を実現するデータストリーム管理システム Harmonica を提案する．Harmonica は以下の 3 つの特徴を持っている．(1) 各種処理を実現するアーキテクチャと要求記述．Harmonica は，ストリームデータに対する問合せ処理システムであるストリーム処理エンジンと既存の DBMS を連携させることで，ストリームデータに対する各種要求を処理する．また，それらの連携要求を定義する要求記述言語を提供する．(2) 要求処理可能性の判定機能．Harmonica は，各種要求に対して連続的な処理の実現可能性を事前に判定する．(3) 複数蓄積要求の最適化機能．Harmonica は，システムの実行環境や性能に応じて複数の蓄積要求に対する最適化を行う．本論文では，(1)，(2)，(3) に示す機能の詳細について述べる．また，我々が開発したプロトタイプシステムで行った性能評価実験について述べる．

Design and Implementation of a Data Stream Management System Harmonica

Shinichi Yamada,[†,☆] Yousuke Watanabe,[††]
Hiroyuki Kitagawa[†,†††] and Toshiyuki Amagasa[†,†††]

Today, the amount of data delivered as data streams has been increasing, and a variety of processing requirements over data streams are emerging. They include archiving and querying streams as historical data, processing continuous queries over incoming data, and integrating historical data and incoming data. We have developed a data stream management system Harmonica to fulfill such requirements. Harmonica provides the following features. (1) Harmonica integrates a stream processing engine and DBMSs and provides an original query language to specify different query and storage requirements. (2) Harmonica detects requirements which exceed system capacity. (3) Harmonica creates feasible processing plans for multiple persistency requirements. In this paper, we describe the above three features in details and present the performance of our prototype system.

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．たとえば，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たびにメモリ上で問合せ処理を行う連続的問合せ要求[1)~6)]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のパターンを比較するといった統合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

† 筑波大学大学院システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba
†† 科学技術振興機構戦略的創造研究推進事業
Core Research for Evolutional Science and Technology, Japan Science and Technology Agency
††† 筑波大学計算科学研究センター
Center for Computational Sciences, University of Tsukuba
☆ 現在，NTT コムウェア株式会社
Presently with NTT COMWARE CORPORATION

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ている．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)~5),7)]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ため，ストリームデータの蓄積や蓄積データを扱う処理には適していない．また，ストリームデータの蓄積処理や検索処理を扱う基盤システムとしてDBMSがあるが，DBMS単体では連続的問合せを処理することができない．そこで本論文では，ストリームに対する各種要求を実現するデータストリーム管理システムHarmonicaを提案する．

本論文で提案するHarmonicaは以下の3つの特徴を持っている．(1) ストリーム処理と蓄積処理の連携のためのアーキテクチャと要求記述言語．(2) 要求処理可能性の判定．(3) 複数蓄積要求の最適化．

(1) に関して，ストリームデータに対する各種処理の実現には，(a) 新規到着データに対する連続的な処理，(b) 処理結果の蓄積，(c) 新規到着データと蓄積データの統合の各機能が必要である．(a)，(b) の機能に関しては，すでに基盤システムとして実績のあるストリーム処理エンジンやDBMSが存在するため，本研究ではストリーム処理エンジンとDBMSを独立したモジュールとして扱い，それらを連携させることによって (c) の機能を実現する．また，それらの連携を実現するためには，新規到着データと蓄積データを透過的に扱う要求記述言語が必要である．従来の連続的問合せ記述[1),6)]では，最新のストリームデータのみを問合せ処理の対象としているため，新規到着データと蓄積データのパターンの比較などの問合せを行うことができない．こうした処理を実現するためには，任意の時刻に到着したデータをまとめる記述力，まとめたデータ列どうしを比較する処理，時間やデータの到着に応じた繰返し処理が記述できることが必要である．Harmonicaは，SQLをベースとしてこれらの要件を満たすように拡張を行った要求記述言語を提供する．

(2) に関して，一般に，高速なメモリ上で動作するストリームの問合せ処理に対して，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たびにフィルタリングして順次蓄積するといった連続的な蓄積要求を大量に処理し続けるためには，ストリームデータの到着頻度や配信データ量，蓄積に用いるDBMSの書き込み性能を考慮に入れて，データが損なわれることのないような蓄積要求を発行することが必要である．しかし，利用者がストリームやDBMSの特徴を正確に判断することは困難であるため，DBMSの書き込み性能を超えた蓄積要求をシステムに登録してしまう可能性がある．DBMSの書き込み性能を超えた要求がシステムに登録されると，書き込みできないデータが消えたり，書き込み待ちデータの増加のために継続的な処理が不可能となったりする．このような問題に対処するため，Harmonicaは利用者の蓄積要求が処理可能かを登録時に判定する．本研究では，ストリームの到着レートを考慮したコストモデルからストリーム処理結果の出力レートを求め，処理結果がその出力レートでDBMSに書き込み可能か検証することによって処理可能性を判定する．

(3) に関して，DBMSでは，DBMSの書き込み性能を上回る数の処理を扱えないため，通常ではストリームの処理結果を蓄積するといった要求を大量かつ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論文では，DBMS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．本手法の特徴は，類似した複数の要求によって蓄積されるデータのうち冗長な部分に着目し，それらを1つにまとめて蓄積する点である．これにより蓄積されるデータ量を調整しつつ，要求されたデータを蓄積することが可能となる．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．本手法では，書き込み性能が許す限りはもとの要求のまま処理するが，書き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．

本論文では (1)，(2)，(3) の機能の詳細について述べるとともに，Harmonicaの性能について評価を行った実験について述べる．

以下は，次のような構成になっている．まず，2章でストリームの利用例について述べ，3章でシステムアーキテクチャ・要求記述について述べる．4章で処理可能性判定手法について説明し，5章で最適化手法について説明する．6章で性能評価実験について説明し，7章で関連研究を紹介する．最後に，8章でまとめと今後の課題について述べる．

2. Harmonicaの利用例

Harmonicaの利用例として，各種センサを使ってガスタービン発電所のモニタリングをする場合について述べる（**図1**）．ガスタービンには，温度や燃料の投入量，タービンの回転数などを逐次管理するために大量のセンサが取り付けられている．複数のガスタービンから取得された各センサデータがストリームデー

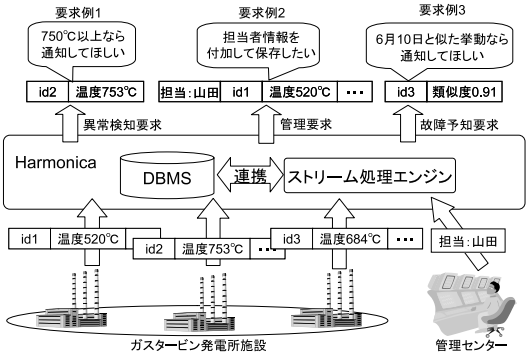

図 1 Harmonica の利用例
Fig. 1 An example scenario using Harmonica.

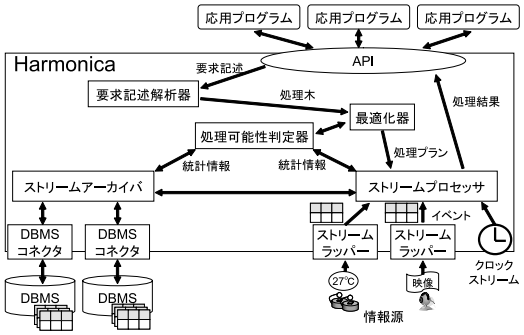

図 2 Harmonica のアーキテクチャ
Fig. 2 Architecture of Harmonica.

タとして管理センタに毎秒配信されている．また，管理センタにはガスタービンをモニタリングしている担当者が存在するものとする．このような状況では以下に示すような要求例が存在する．

要求例 1 ガスタービンを異常な温度のまま放置すると，ガスタービン発電所施設すべてを巻き込んだ大事故になる可能性がある．管理者は，ガスタービンから得られるセンサデータが異常な温度になっていたら，ただちに対策を講じる必要がある．そこでたとえば「温度センサのデータが 750 度以上になったら，タービンの ID とその温度を通知してほしい」といった異常検知要求が考えられる．この要求を実現するためには，毎秒到着するセンサデータに対して，温度によるフィルタリング処理を逐次行う必要がある．

要求例 2 将来何か問題が発生したときの原因究明のためには，日々の運用ログをとる必要がある．そこで「配信された各種センサデータに担当者の情報を付加して保存したい」といった管理要求が考えられる．この要求を実現するためには，毎秒到着するセンサデータとガスタービンを現在監視している担当者の情報を統合して，DBMS に書き込む機能が必要である．

要求例 3 ガスタービンには部品の劣化などで故障するという問題がある．劣化した部品を早期に交換することができれば，ガスタービンの故障を未然に防ぐことができる．そこで「各センサデータが過去に故障したときのデータと類似した挙動をしていたら通知してほしい」といった要求が考えられる．このような要求を実現するためには，あらかじめ故障時のデータを蓄積しておき，最近のセンサデータ列と故障時のデータ列を使って，データ列どうしの類似度を定期的に計算することが必要である．

要求例 1 の異常検知要求のように，到着するデータに対して連続的に問合せ処理を行うといった要求は，従来のストリーム処理エンジンで処理することができる．しかし，ストリーム処理エンジンはストリームデータの蓄積用に作られたシステムではないため，要求例 2 の管理要求や要求例 3 の故障予知要求のように，データの蓄積をともなう要求については DBMS との連携が必要となる．本論文で提案する Harmonica 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，要求例 1，2，3 の処理を実現する．具体的なアーキテクチャや要求記述例については 3 章で述べる．

3. Harmonica システム

3.1 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 へ書き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 のアーキテクチャを**図 2** に示す．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，処理可能性判定器，最適化器，DBMS コネクタ，ストリームラッパによって構成されている．利用者や応用プログラムは，SQL ライクな要求記述言語 HamQL（Harmonica Query Language）によって Harmonica に要求を登録する．HamQL については 3.2 節で述べる．利用者や応用プログラムから登録された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，処理可能性判定器を使い，複数の処理木から最適と考えられる処理プランを生成する．処理可能性判定器は，ストリームアーカイバとストリームプロセッサから各種統計情報を取得し，処理プランの処理可能性を判定する．処理可能性判定器の判定手法については 4 章で，最適化器による処理プランの生成方法については 5 章で説明する．ストリームラッパは，各種スト

リーム型情報源から配信されるデータをリレーションのタプル形式へと変換する役割を持っている．タプル形式に変換されたデータはデータ到着イベントととも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，ストリームラッパやクロックストリーム☆から通知されるイベントに連動して処理を行う．ストリームプロセッサで生成された処理結果は，ただ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，データの書き込みを行ったり，DBMS から必要なデータの読み出しを行ったりする．

本論文で開発したプロトタイプシステムは，ストリームプロセッサとして我々の研究グループで開発しているストリーム処理エンジン StreamSpinner[7] を用いており，DBMS として MySQL や PostgreSQL, SQL Server の利用を想定している．

3.2 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 を提供する．Harmonica では，DBMS に対して行うような**即時問合せ**と**連続的問合せ**[1),2),4)~7)] の 2 種類の要求を想定している．即時問合せとは，要求を与えたときに 1 度だけ処理される要求のことである．また，連続的問合せとはストリーム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，新規到着データを用いて生成される処理結果を逐次配信する処理方式である．利用者は**マスタ情報源**と**ウインドウ**を指定することで連続的問合せを記述する．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いい，ウイ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いう．ストリームでは逐次タプルが到着するため，過去に到着したすべてのデータを演算の処理対象とした場合，非常に大量のタプルを処理しなければならない．そこでウインドウを用いて演算の適用対象を限定することで処理の効率化を目指している．

図 3 に HamQL の構文を示す．HamQL は MASTER 節，INSERT 節，SELECT-FROM-WHERE 節，GROUP BY 節に分けることができる．MASTER 節はストリームに対して連続的問合せ処理を行う場合に，マスタ情報源を指定する．MASTER 節が記述されないときには，即時問合せと解釈される．INSERT 節では，SELECT 節以降の処理によって生成されたタプルを蓄積するテーブル名を指定する．INSERT 節が記述されない場合は，処理結果は直接利用者へと配信される．SELECT-FROM-WHERE 節と GROUP BY 節は SQL のそれとほぼ同等の意味を持つ．ただし，FROM 節では角括弧 “[]” によってウインドウを定義できるように拡張を行っている．ウインドウはウインドウサイズと**起点**を指定して定義する．起点とは，要求が評価されるときにどの時刻をウインドウの開始時刻とするかを設定するものである．起点の値としては，具体的な数値のほかに，要求の評価時刻を表す **now** を指定することができる．起点は省略することもでき，省略された場合は起点に now が指定されたものと解釈される．ウインドウの大きさと起点を駆使することで，新規データと評価時刻から 1 カ月前のデータの比較といった，処理対象とする 2 つのデータが時刻によってどちらも変化する処理が記述可能である．

```
{MASTER       master_source1, …}
{INSERT INTO  table_name}
 SELECT       attribute1, …
 FROM         source1[window_size,start_time], …
{WHERE        condision1 …}
{GROUP BY     key1, …}
```

図 3 HamQL の構文
Fig. 3 Syntax of HamQL.

```
MASTER Turbine
SELECT Turbine.timestamp, Turbine.id, Turbine.temp
FROM Turbine[1s,now]
WHERE Turbine.temp >= 750
```

図 4 要求例 1
Fig. 4 Example of query 1.

以下では 2 章の要求例をもとに HamQL の具体的な記述例を説明する．ここでは，ガスタービンから取得したセンサデータが毎秒 Turbine ストリームとして，また，管理者の情報が 3 時間ごとに Monitor ストリームとしてシステムに配信されているものとする．

記述例 1 **図 4** に要求例 1 の記述例を示す．要求例 1 では，Turbine ストリームから到着した温度データのうち 750 度以上のデータだけをフィルタリングすればよい．そこで，MASTER 節には Turbine ストリームを指定し，WHERE 節で温度の値が 750 度以上のタプルのみを選択している．また，図 4 では要求の評価時刻を起点とした 1 秒幅のウインドウを Turbine に対して定義している．したがって図 4 の問合せの評価時には，評価時刻から 1 秒前までに到着した Turbine ストリームのタプルのうち，未評価のタプルがフィルタリング処理の対象となる．

☆ クロックストリームとは，定期的にアラームを送信する仮想的なストリームのことで Harmonica が提供する．

```
MASTER Turbine
INSERT INTO Turbine_log
SELECT Turbine.*, Monitor.name
FROM Turbine[1s,now], Monitor[3h,now]
WHERE Turbine.id = Monitor.turbine_id
```

図 5 要求例 2
Fig. 5 Example of query 2.

```
MASTER Clock60s
SELECT max(Turbine.timestamp),
       dist(array(Turbine.temp),array(Turbine_log.temp))
FROM ( SELECT max(Turbine.timestamp),
              array(Turbine.temp)
       FROM Turbine[60s,now]
       WHERE Turbine.id = 1          ),   } 副問合せ1
     ( SELECT array(Turbine_log.temp)
       FROM Turbine_log[60s,10800]
       WHERE Turbine_log.id = 1      )   } 副問合せ2
```

図 6 要求例 3
Fig. 6 Example of query 3.

記述例 2 要求例 2 の記述例を**図 5** に示す．図 4 との主な違いは INSERT 節が追加されたことである．図 5 の要求によって，Turbine ストリームからデータが到着するたびに，Turbine の id と Monitor の turbine_id が等しいタプルどうしが結合され，Turbine_log テーブルに蓄積するといった処理が行われる．

記述例 3 **図 6** は要求例 3 の記述例である．図 6 は，Turbine ストリームから配信される新規データと Turbine_log テーブルに蓄積したデータとの時系列的な距離を計算して毎分配信するといった処理を実現する．ここでは ID1 のガスタービンのデータに対して距離の計算を行う．また，故障予知に用いるセンサデータとして温度を利用する．距離の計算には新規データと蓄積データそれぞれ 1 分間分のデータを使用しており，要求では 1 分間幅のウインドウを定義することによって計算対象のデータを指定している．距離を毎分計算するために，MASTER 節には 1 分ごとにアラームを送信するクロックストリーム Clock60s を指定している．新規データと蓄積データの距離の計算は，2 つの配列型データに対してユーザ定義関数 dist() を適用させることで行う．dist() は 2 つの配列型の属性を引数として受け取って，2 つのデータ列間の線形距離の計算を行う．距離の計算に使う配列型データは，FROM 節で副問合せを使って生成する．図 6 に示す 2 つの副問合せのうち，副問合せ 1 が Turbine ストリームの到着データを配列型に変換する副問合せであり，副問合せ 2 が Turbine_log テーブルの蓄積データを配列型に変換する副問合せである．2 つの副問合せにおいて，Turbine ストリームと Turbine_log テーブルから距離計算の対象とするタプル集合をそれぞれウインドウによって定義する．Turbine ストリームには，要求の評価時刻を起点として 1 分間幅のウインドウを定義し，Turbine_log テーブルには，起点を時刻 10,800 で固定した 1 分間幅のウインドウを定義している．それぞれのウインドウに含まれるタプル集合に対して関数 array() を適用することによって，タプル集合の温度の値を配列化する．array() は本研究で導入した関数である．また，副問合せ 1 では距離を計算した時刻を取得するために，関数 max() を timestamp 属性に対して適用している．関数 max() はタプル集合の中から最大値を取得する関数である．

以上のように，HamQL では本研究で目的とするストリームに対する各種処理要求が記述可能である．

4. 処理可能性の判定手法

Harmonica では連続的問合せによる蓄積処理や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，本章では特に蓄積できずにデータが損なわれてしまう問題に焦点を当てる．一般に DBMS に対する蓄積処理はストリーム処理と比べて非常に遅いので，連続的問合せによる蓄積要求を処理し続けるためには，ストリームデータの到着レートや DBMS の書き込み性能を考慮して蓄積要求を記述する必要がある．しかし，利用者がそれらの特徴を正確に判断することはできないため，書き込み性能を超える蓄積要求を利用者がシステムに登録してしまう可能性がある．書き込み性能を超えた蓄積要求では，蓄積処理が間に合わずに処理が破綻するといった問題が生じる．Harmonica ではこのような問題に対して，利用者の蓄積要求が処理可能であるかを事前に判定する処理可能性判定器を提供する．本章では，その判定手法について述べる．

本研究で提案する判定手法は，ストリームの到着レートからストリームプロセッサの処理コストと処理結果の生成レートを推定する．そして，その処理コストで連続的問合せが処理可能であるか，その生成レートで処理結果を DBMS に書き込み可能であるかを判定する．本研究では，処理コストや処理結果の生成レートの推定に文献 8) や文献 9) で提案されているコストモデルを利用する．ただし，文献 8) や文献 9) のコストモデルでは蓄積処理に対するコストについては考えられていない．そこで本研究では，これらのコストモデルを新たに蓄積処理用に拡張し，DBMS に処理結果が書き込み可能かどうかの判定を行う．

以降では，4.1 節で処理可能性の定義について述べ，4.2 節でコストモデルについて述べる．また，4.3 節で具体的な判定例について述べる．

4.1 処理プランの処理可能性の定義

処理プランの処理可能性の定義について述べる．こ

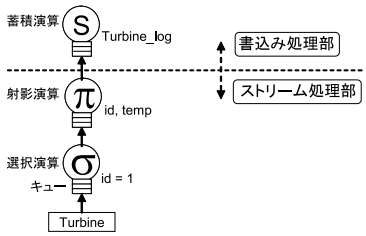

図 7 処理木
Fig. 7 Processing tree.

こでは，すべ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多く存在する．この定義は，そうした一定間隔で配信されるストリームデータを意図したものである．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を考慮の対象とする．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，こうした仮定をお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う演算の順番が定義されている．処理木の例を**図 7** に示す．図 7 は，ID1 のタービンのセンサデータから温度のデータのみを Turbine_log テーブルへと蓄積する処理木である．図 7 の円で示す 3 つのノードは演算を表し，内側の文字が σ は選択演算，π は射影演算，S は蓄積演算である．各演算のノードには，連続的問合せの処理時にウインドウの時間条件に合うタプルを一時保存しておくためのキューが付属している．処理木のリーフノードは情報源を表しており，図 7 のリーフノードには Turbine ストリームが示されている．

3 章で述べたように，Harmonica はストリームプロセッサと DBMS を連携させることで各種処理を行っている．したがって図 7 に示す処理木は，蓄積演算の前後で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き込み処理部が含まれている．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，提案手法では，ストリーム処理部と書き込み処理部にそれぞれ分けて，処理プランの処理可能性を考える．

まず，ストリーム処理部の処理可能性について定義する．ストリーム処理部では，各演算が処理木の下から順に処理されることを想定している．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現在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 1 処理プラン P に含まれるストリーム処理部の演算 o_j（$0 \leq j \leq n$）において，単位時間内に到着するタプルに対する処理時間を c_j 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．

$$\sum_{j=0}^{n} c_j \leq 1 \tag{1}$$

式 (1) はストリーム処理部の各演算において，単位時間内に到着するタプルに対する処理時間の合計が単位時間内に収まらなければならないことを表している．以降，c_j を演算 o_j の**処理コスト**と呼ぶ．また，$\sum c_j$ を P 全体の処理コストとし $C_{P,SP}$ と表記する．

次に，書き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 DBMS 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，前回の書き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，書き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 2 処理プラン P に含まれる蓄積演算 S_k（$0 \leq k \leq m$）において，単位時間内に到着するタプルに対する処理時間を s_k とするとき，以下の条件を満たす P を**書き込み処理可能**と定義する．

$$\sum_{k=0}^{m} s_k \leq 1 \tag{2}$$

以降，s_k を蓄積演算 S_k の**書き込みコスト**と呼び，$\sum s_k$ を P 全体の書き込みコストとし $C_{P,DB}$ 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 処理プラン P がストリーム処理可能かつ書き込み処理可能のとき，P を**処理可能**と定義する．

4.2 コストモデル

4.1 節で述べたように，処理プランの処理可能性を判定するためには，処理プランの処理コスト $C_{P,SP}$ と書き込みコスト $C_{P,DB}$ をそれぞれ推定する必要がある．本節では，$C_{P,SP}$ と $C_{P,DB}$ の推定に用いるコストモデルについて説明する．推定に用いる変数を**表 1** に示す．

本コストモデルでは，まずストリームの**到着レート**

表 1 処理可能性判定に用いる変数
Table 1 Variables used in estimating whether queires are feasible.

変数	役割
λ_o	演算 o の出力レート（タプル/単位時間）
W_o	演算 o の出力キューに含まれるタプル数
f_o	演算 o のタプルの選択率
τ_σ	選択・射影演算の 1 タプルの処理時間
$\tau_\bowtie$	結合・直積演算の 1 タプルの処理時間
τ_{DB}	1 タプルの蓄積にかかる時間
c_j	ストリーム処理部の演算 o_j の処理時間（/単位時間）
s_k	蓄積演算 S_k の処理時間（/単位時間）
$C_{P,SP}$	処理プランの処理コスト
$C_{P,DB}$	処理プランの書き込みコスト

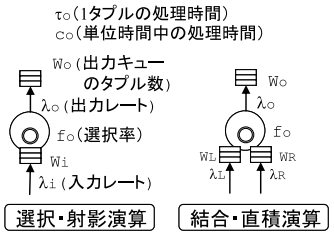

図 8 演算の処理コスト
Fig. 8 Processing costs.

（単位時間あたりに到着するタプル数）とウイ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）と**出力キュー**（次の出力先のキュー）に含まれ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 $C_{P,SP}$ や $C_{P,DB}$ の計算を行う．以下では，各コストの推定方法について述べる．

4.2.1 処理プランの処理コストの推定

処理プランの処理コスト $C_{P,SP}$ は，4.1 節で述べたようにストリーム処理部の各演算の処理コストの合計である．ここでは，ストリーム処理部の各演算の処理コストの推定方法について説明する．本研究では，ストリーム処理部の演算としてリレーショナルモデルの選択・射影・直積・結合演算を扱う．

まず，選択演算の場合について説明する（**図 8** の左）．ここでは，演算 i の処理結果が演算 o に入力されるとする．見積もるべき演算 o の出力レートを λ_o，出力キューに含まれる時間平均のタプル数を W_o とする．選択演算の場合，演算 o の**入力レート**（単位時間あたりに演算 i から入力されるタプル数）λ_i，演算 o のキューに含まれるタプル数 W_i，演算 o の選択率 f_o を用いて以下のように推定できる．

$$\lambda_o = f_o\lambda_i$$
$$W_o = f_oW_i$$

ここで求めた λ_o と W_o は，次の演算の処理コストの推定に使われる．射影演算の場合は，演算の前後でタプル数に変化がないので，選択率 $f_o = 1$ の選択演算と考えることができる．また，演算 o の処理コスト c_o は次のように計算可能である．

$$c_o = \tau_\sigma\lambda_i$$

τ_σ は 1 タプルの選択処理にかかる時間であり，ストリームプロセッサから取得する．

次に，結合演算の場合について説明する（図 8 の右）．結合演算は 2 つのリレーションから 1 つのリレーションを出力する演算である．ここでは，結合演算 o は演算 L と演算 R の処理結果を入力として受け取るものとする．2 つの入力となる演算の入力レートをそれぞれ λ_L，λ_R，それぞれの入力キューに含まれるタプル数を W_L，W_R とする．まず，演算 L について考えると，単位時間の間に λ_L のタプルが到着する．結合演算 o において，結合対象となる R 側のキューには W_R 個のタプルが存在するので，単位時間に L 側の入力によって出力されるタプルは $f_o\lambda_LW_R$ となる．演算 R についても同様に考えることができ，単位時間に R 側の入力によって出力されるタプルは $f_o\lambda_RW_L$ となる．したがって，結合演算 o の出力レート λ_o は次の式で推定できる．

$$\lambda_o = f_o(\lambda_LW_R + \lambda_RW_L)$$

また，結合演算 o の出力キューに含まれるタプル数 W_o は，ウインドウ W_L の 1 タプルに対して f_oW_R 個のタプルが出力されるため，次の式で推定する．

$$W_o = f_oW_LW_R$$

選択演算や射影演算の場合と同様に，ここで求めた λ_o と W_o は，次の演算の処理コストの推定に使われる．結合演算の処理コスト c_o も出力レートと同様に考えると，L の入力タプルに対する処理コストは $\tau_\bowtie\lambda_L$，R の入力タプルに対する処理コストは $\tau_\bowtie\lambda_R$ と推定できるので，結合演算の処理コスト c_o は以下の式で計算できる．

$$c_o = \tau_\bowtie(\lambda_L + \lambda_R)$$

ここで $\tau_\bowtie$ は 1 タプルあ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 $f_o = 1$ の結合演算として考える．

4.2.2 処理プランの書き込みコストの推定

処理プランの書き込みコスト $C_{P,DB}$ は，定義 2 に示したとおり書き込み処理部の蓄積演算の書き込みコストの合計によって計算する．ストリーム処理部で生成されたデータの入力レートを λ_i，1 タプルの蓄積処理にかかる時間を τ_{DB} とすると，蓄積演算 o の書き込みコスト s_o は以下のように推定する．

$$s_o = \tau_{DB}\lambda_i$$

τ_{DB} は**書き込みレート**（単位時間あたりに書き込み可能な最大のタプル数）の逆数によって求める．

書き込みレートはディスクの書き込み速度やDBMS内部の書き込み方式の違いなどに依存するパラメータなので，利用者が一意に定めることは非常に難しい．そこで本研究では，システムで利用するDBMSを使って，単位時間あたりに書き込み可能なタプル数を実際に計測することによって，書き込みレートを推定する．一般に書き込む1タプルあたりのデータ量が増加すると，書き込みレートが低下していくと考えられることから，本研究では書き込みレート λ_{Write} の推定は以下の式で行う．

$$\lambda_{Write} \simeq rate\_estimate(tuple\_size(S_o))$$

S_o は蓄積演算 o が蓄積するテーブルのスキーマである．また，$tuple\_size()$ はスキーマ S_o から1タプルあたりのデータ量を計算する関数であり，$rate\_estimate()$ は1タプルあたりのデータ量から書き込みレートを推定する関数である．スキーマ S_o に対するデータ量はDBMSから取得する．

また，1タプルあたりのデータ量からの書き込みレートの推定については次のように行う．まず，書き込みレートを推定するためのサンプルとして，事前にデータ量の異なる n 種類のタプルに対して実際に書き込みレートを計測する．そして，未計測のデータ量のタプルに対する書き込みレートについて，最も近い計測済みの2点で直線近似することで推定する．

4.3 処理可能性判定の計算例

具体的な処理可能性の判定例として，**図9**のプランを用いて説明する．ただし，Turbineストリームのウインドウは1秒，選択演算の選択率 f_{o1} は0.5とする．また，Turbineストリームから毎秒50タプルが到着し，選択・射影演算の1タプルあたりの平均処理時間 c_{o1}，c_{o2} は0.001 sとする．

まず，与えられた統計情報から最初の選択演算の入力レート $\lambda_{Turbine}$ とキューに含まれるタプル数 $W_{Turbine}$ を推定する．今Turbineストリームが毎秒50タプル到着するので，入力レートは $\lambda_{Turbine} = 50$ である．また，選択演算のキューに含まれるタプル数は，Turbineストリームのウインドウの大きさとTurbineストリームの到着レートの積によって見積もることができる．図9の処理では，Turbineストリームに対して1秒間のウインドウが指定されているので，選択演算のキューに含まれるタプル数 $W_{Turbine}$ は $W_{Turbine} = 1 \cdot 50 = 50$ と計算できる．次に，$\lambda_{Turbine}$ と $W_{Turbine}$ をもとにして，各演算の出力レートと次の演算の入力キューに含まれるタプル数をボトムアップに推定する．

$$\lambda_{o1} = f_{o1}\lambda_{Turbine} = 0.5 \cdot 50 = 25$$
$$W_{o1} = f_{o1}W_{Turbine} = 0.5 \cdot 50 = 25$$
$$\lambda_{o2} = f_{o2}\lambda_{o1} = 1 \cdot 25 = 25$$
$$W_{o2} = f_{o2}W_{o1} = 1 \cdot 25 = 25$$

各演算の出力レートの計算後，処理コストの計算を行い，ストリーム処理部の処理可能性を判定する．

$$\begin{aligned} C_{P,SP} &= c_{o1} + c_{o2} \\ &= \tau_{o1}\lambda_{Turbine} + \tau_{o2}\lambda_{o1} \\ &= 0.001 \cdot 50 + 0.001 \cdot 25 = 0.075 \end{aligned}$$

上式より，この処理プランの処理コストは $C_{P,SP} \leq 1$ なので，この処理プランはストリーム処理可能である．また，このプランで生成されるタプルサイズに対する書き込みレートが $\lambda_{Wrtie} = 40$ であったとすると，書き込みコストは以下のように計算できる．

$$\begin{aligned} C_{P,DB} &= \tau_{DB}\lambda_{o2} \\ &= \lambda_{o2}/\lambda_{Write} = 25/40 = 0.625 \end{aligned}$$

上式より，この処理プランの書き込みコストは $C_{P,DB} \leq 1$ となり書き込み処理可能であることが分かる．この処理プランはストリーム処理可能であり，かつ書き込み処理可能であるので，処理可能であるといえる．

ここで仮に，図9と同じ処理で蓄積テーブル名だけが違う要求が同時にシステムに登録されていたとする．すると，これら2要求に対する処理プランの処理コストは $C_{P,SP} = 0.075 + 0.075 = 0.15$，書き込みコストは $C_{P,DB} = 0.625 + 0.625 = 1.25$ となり書き込み処理可能ではなくなる．この場合，DBMSの書き込み性能を超える蓄積が要求されるため，蓄積処理が間に合わず破綻する．しかし，この2要求は蓄積するタプルが完全に一致しているので，どちらかの結果だけで代用が可能である．Harmonicaの最適化器は，処理プランの書き込みコストが $C_{P,DB} > 1$ となってしまう場合に，蓄積するタプルを共用することで書き込みコストが $C_{P,DB} \leq 1$ となる処理プランを導出する．

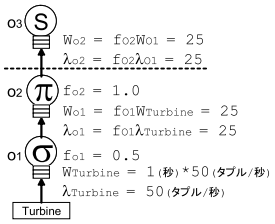

図9 コスト見積り
Fig. 9 Example of estimating costs.

5. 複数蓄積要求の最適化手法

図 5 のような，ストリーム処理の結果を逐次 DBMS へ蓄積する要求が大量にシステムに登録されると，DBMS の書き込み性能を容易に上回ってしまい，継続的な蓄積処理が困難となる．このような問題に対して Harmonica は，DBMS の性能に基づいた複数蓄積要求に対する最適化機能を提供する．

複数の蓄積要求に対する継続的な処理の実現には，蓄積するタプル数が書き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．蓄積するタプル数を減らす既存の手法として load shedding [2),10)] 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．ストリームデータの蓄積処理では，要求されたデータがきちんと蓄積され，読み出し時には要求どおりのタプルが取得できることが望ましいと考えられる．そこで Harmonica の最適化器では，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，要求どおりのタプルが読み出し時にきちんと取得で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．

以下では，5.1 節で提案手法の基本概念，5.2 節で最適化の方針について述べる．5.3 節で最適化手順について説明し，5.4 節で最適化の例について説明する．

5.1 基本概念

最適化の概要を**図 10** に示す．図 10 左では，q1〜q3 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，q1〜q3 それぞれで 40 タプルにフィルタリングされて DBMS へ蓄積されるものとする．このプランで処理すると，毎秒合計で 120 タプルの蓄積処理が必要となる．仮に DBMS の書き込みレートが毎秒 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共通に含まれる演算に注目する．図 10 左の例では，要求 q1〜q3 の選択演算が共通している．共通の演算では，まったく同じ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 q1〜q3 を満たす結果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き込みが減り，蓄積処理の回数そのものを減らすことができる．

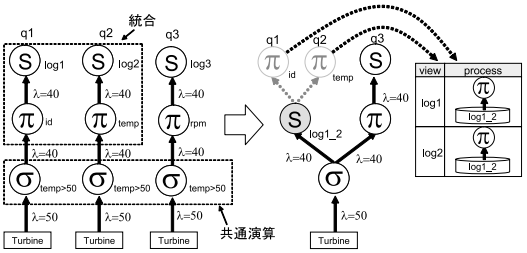

図 10 最適化の概要
Fig. 10 Summary of optimization.

提案手法では，処理プランを書き換えて，共通する演算の上までそれぞれの蓄積演算を移動させる．そして移動させた蓄積演算どうしを統合し，蓄積するタプルを共有化することで，DBMS に書き込むタプル数を減少させる．図 10 右は，q1 と q2 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 10 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 の書き込みレートを下回ったため，継続的な処理が可能となる．

もちろん提案手法の代償として，読み出し時には共用テーブルから本来求められている結果を導出するためのビュー処理を行わなければならない．図 10 左の蓄積要求の log1，log2 に対応するデータを図 10 右から取得するには，共用テーブル log1_2 に対して残りの射影演算を行う必要がある．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．

5.2 最適プランの定義

書き込み処理を減らすため蓄積演算の統合は必要であるものの，それによってビュー処理コストが増加しすぎると，DBMS からの読み出し時に多くの処理が発生する．Harmonica 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 の読み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する．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，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すべての処理木を入力として受け取り，処理可能かつビュー処理コストが低いプランを出力する．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**（非巡回有向グラフ）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**で構成される．ビュー対応表に

は，統合する前の蓄積演算が書き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る．

5.3 処理可能プラン導出手順

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している箇所を**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，ビュー処理コストの低い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出力する．ただし，分岐以降の固有パス上に結合・直積演算が含まれる場合は，その固有パス上の蓄積演算は統合の対象とはしない．結合・直積演算は 2 つの入力されたタプルを結合する演算であるため，結合・直積演算を行う前に蓄積してしまうと，タプ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タプルも蓄積する必要があり，統合による蓄積演算数の減少が見込めないからである．

最適化器による処理可能プラン導出手順を**図 11** 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，4 つのステップに分かれて処理可能プランの導出を行う．それぞれのステップの概要について以下で説明する．全体を通した具体的な最適化例については 5.4 節で述べる．

ステップ 1：共通演算統合　共通演算統合ステップでは複数の処理木を入力として受け取り，そこから初期プランを生成する．初期プランの生成には，既存の複数問合せ最適化手法[1),11)~13)] を利用する．複数問合せ最適化手法により，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．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，その処理可能性が判定される．初期プランが処理可能ならビュー処理コストは 0 なので，そのプランを出力する．初期プランが処理可能でない場合，以下に続く分岐点抽出・順位付け・蓄積演算統合という 3 ステップを順に経由して処理プランの更新を行う．

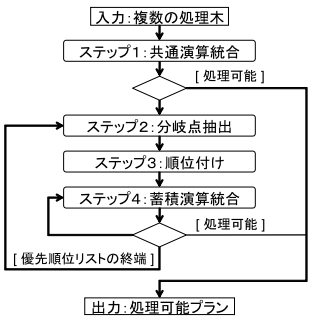

図 11 処理可能プラン導出手順
Fig. 11 Flow of optimization.

ステップ 2：分岐点抽出　ここでは，処理プランの中から着目すべき分岐点を 1 つ選択する．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．ただし，蓄積演算と分岐点の間に結合・直積演算が存在する場合，そのパスについては演算数のカウントを行わない．分岐点が 1 つ選ばれるたびに下の 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．

ステップ 3：順位付け　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 1 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をもとに，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生成する．一般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，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，分岐点抽出ステップと同様に，分岐先と蓄積演算の間に結合・直積演算が存在するパスは優先順位リストには加えない．

ステップ 4：蓄積演算統合　ここでは，順位付けステップの優先順位リストに従って順に蓄積演算を統合する．具体的な統合方法としては，まず優先順位リストの先頭にある蓄積演算を 2 つ選択する．次に，それぞれの蓄積演算を分岐点のすぐ上までプッシュダウンし，蓄積演算どうしを統合する．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へと保存する．蓄積演算統合ステップでは，蓄積演算が 1 つ統合されるたびに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出力する．受け取った優先順位リストの蓄積演算をすべて統合しても，処理可能なプランにならないときは，分岐点抽出ステップへ戻る．

以上のステップが処理プランが処理可能となるか，すべての蓄積演算を統合するまで行われる．処理可能となった場合にはそのプランが出力され，すべての蓄積演算を統合しても処理可能とならない場合，空のプ

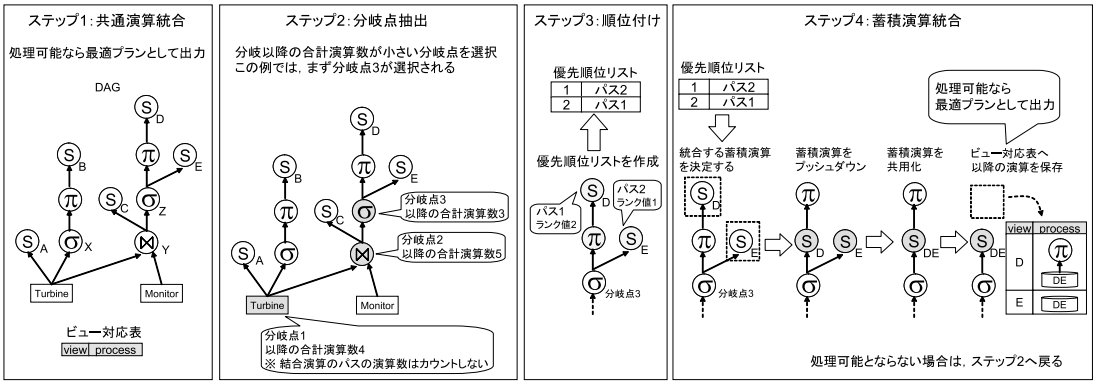

図 12 最適化例
Fig. 12 Example of optimization.

ランを出力して終了する．

5.4 最適化例

最適化手順の具体例について述べる（**図 12**）．ここでは，Turbine ストリームと Monitor ストリームが Harmonica に配信されており，Harmonica には 5 つの異なる蓄積要求が登録されるものとする．5 つの蓄積要求のうち，2 つが Turbine ストリームのみに対する蓄積要求であり，3 つが Turbine ストリームと Monitor ストリームを結合した結果に対する蓄積要求である．円で表される演算ノードのうち，⋈ で表されるノードは結合演算を表している．また，蓄積演算の脇に記述されている文字は，蓄積するテーブル名を示しており，選択演算と結合演算の脇の文字は，それぞれ選択条件，結合条件を表している．

まず，共通演算統合ステップにおいて，5 つの蓄積要求の処理木に含まれる部分木を統合し，初期プランとなる DAG と空のビュー対応表を生成する．初期プランの DAG は既存の手法によって生成するため，ここでは詳細は省略する．初期プランが処理可能な場合は，初期プランを出力し最適化処理は終了する．初期プランが処理可能でない場合は，分岐点抽出ステップへと移行する．

分岐点抽出ステップでは，着目する分岐点を 1 つ選択するために分岐以降の合計演算数を計算する．初期プランには合計 3 つの分岐点が存在する．分岐点 1 に関しては，分岐以降の演算数は 10 であるが，そのうち 6 つは結合演算以後の演算であるため計算から除外され，演算数は 4 となる．また，分岐点 2 以降の演算数が 5，分岐点 3 以降の演算数が 3 となるので，ここでは演算数が最小となる分岐点 3 が選択され，順位付けステップへと処理が移行する．

順位付けステップでは，固有パスの長さをランク値として昇順にソートした優先順位リストを作成する．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が作成される．ここでは，分岐点 3 以降の 2 つのパスのうち，パス 1 のランク値が 2，パス 2 のランク値が 1 なので，パス 2，パス 1 の順で優先順位リストが作成される．その後，蓄積演算統合ステップへと処理が移行する．

蓄積演算統合ステップでは，優先順位リストに従って蓄積演算が統合される．図 12 では，分岐点 3 以降の蓄積演算は 2 つしか存在しないので，必然的にその 2 つの蓄積演算が統合される．まず，それぞれの蓄積演算を分岐点 3 のすぐ上までプッシュダウンする．次に，蓄積演算を統合して，現在の蓄積演算とは別のテーブル（図 12 の例では DE）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．読み出しは本来のテーブル名（D および E）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タプルの生成に必要な処理を知ることができる．蓄積演算の統合を繰り返すことによって，1 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，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たどっていくことで元のタプルを生成するまでの処理を復元することが可能である．以上のように蓄積演算が統合され，書き込みコストを下げた新たな処理プランが生成される．

優先順位リストに従って着目している分岐点以降の蓄積演算をすべて統合しても処理可能なプランとならない場合，再び分岐点抽出ステップへと移行し，まだ

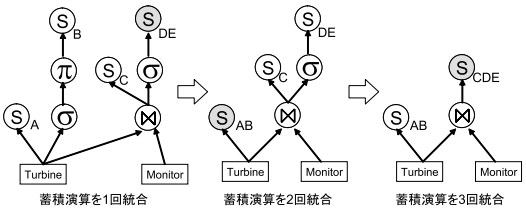

図 13 処理プランの遷移
Fig. 13 Transition of processing plan.

着目していない分岐点について蓄積演算の統合が行われる．このような統合処理が処理可能なプランとなるまで繰り返し行われる．

最後に，図 12 で紹介した処理プランがこの先どのように更新されるかを**図 13** に示す．図 12 で作成した初期プランは，最終的には 3 回蓄積演算の統合が可能である．図 12 から分かるように，蓄積演算の統合回数を増やしていくことによって，蓄積演算の総数が少ない処理プランが生成されていることが確認できる．この例の初期プランで 3 回蓄積演算を統合しても処理可能とならない場合，統合できる蓄積演算が存在しなくなるため，空の処理プランを出力して終了する．

5.5 考　　察

提案アルゴリズムによって導出される処理プランについて議論する．まず，処理可能性に関しては，提案アルゴリズムは最悪の場合でも統合可能な全蓄積演算の統合まで試みるため，処理可能プランが存在する場合には必ずそれを導出することが保証される．ビュー処理コストに関しては，低くはなるが最小になることは保証されない．これは提案アルゴリズムが全処理プランを探索していないことによる．具体的にはステップ 2 の分岐点抽出が，分岐点以降の総演算数の少ない順に適用されるためで，演算数の多い分岐点の蓄積演算統合は，演算数の少ない他の分岐点の蓄積演算統合が済んだ後にのみ行われる．よって，演算数の多い分岐点の蓄積演算統合を先に行うような処理プランなどは探索対象にならない．

厳密に最適プランを導出するには，処理可能性を満たすすべての処理プランを列挙したうえで，ビュー処理コストが最小となる処理プランを選べばよい．ただし，大量の処理要求があった場合には探索範囲が膨大となり，一般に非常に多くの計算コストを必要とする．提案アルゴリズムの処理可能プラン導出までの計算量としては，ステップ 1 の計算量は，既存の手法を用いているだけなので，初期プラン作成に用いる複数問合せ最適化手法に依存する．残りのステップは，蓄積演算を 1 つずつ統合するだけなので，蓄積演算が n 個ある場合，たかだか $O(n)$ で処理可能プランが導出可能である．

表 2 実験環境
Table 2 Experimental environment.

OS	Windows Vista Business
CPU	Pentium（R）D 3.00 GHz
Memory	2 GB
DBMS	MySQL 5.0
Java	JDK6

表 3 処理可能性判定の妥当性
Table 3 Relevance of feasibility validator.

要求の演算	推定値（タプル/秒）	実測値（タプル/秒）
蓄積	34	33
選択 蓄積	68	66
射影 蓄積	35	34
選択 射影 蓄積	70	68
結合 射影 蓄積	3	3

6. 性能評価実験

本研究で行った評価実験について述べる．本実験の主な目的は，処理可能性判定器と最適化器の妥当性検証と性能評価である．実験は**表 2** に示す環境で行った．

6.1 実　験　1

実験 1 では，処理可能性判定手法の妥当性を検証した．妥当性検証のために，到着レートを任意に変更可能な疑似 Turbine ストリームと 5 種類の異なる蓄積要求を用意した．まず，疑似ストリームの到着レートを変更しながら，それぞれの蓄積要求で個別に処理可能性の判定を行い，処理可能となる到着レートの境界を求めた．処理可能性判定に必要な書き込みレートは，実際に MySQL 上で計測したものを用いた．次に，開発したプロトタイプシステムで処理を行い，実際の処理可能な境界値を求めた．ここでは，プロトタイプシステムで 5 分以上処理を行い，待ち行列の長さに目立った増加傾向がない場合を処理可能と判断した．

実験結果を**表 3** に示す．「要求の演算」の列には，各要求を構成する演算が示されている．また，推定値は提案手法によって求めた境界値を示し，実測値はプロトタイプシステムで処理できた境界値を示す．表 3 より，今回実験を行った 5 種類の蓄積要求に対しては，どれも高い精度で処理可能性の判定ができていることが確認できる．

また，書き込みに用いるディスクキャッシュの影響を調査するために，ディスクキャッシュを無効にして同様の実験を行った（**表 4**）．表 4 より，ディスクキャッシュが無効であっても，提案手法による処理可能性の

表 4 処理可能性判定の妥当性（キャッシュなし）
Table 4 Relevance of feasibility validator (no cache).

要求の演算	推定値（タプル/秒）	実測値（タプル/秒）
蓄積	31	30
選択 蓄積	62	64
射影 蓄積	31	30
選択 射影 蓄積	62	64
結合 射影 蓄積	3	3

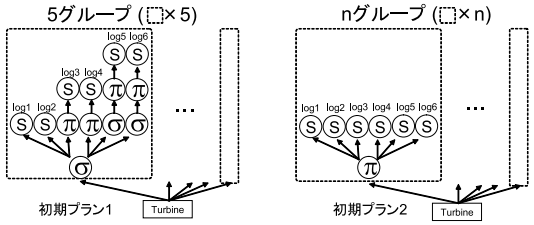

図 14 実験用初期プラン
Fig. 14 Initial plans for experiments.

判定が高精度で行えることが確認できる．

5 つの蓄積要求のうち，4 つの蓄積要求は推定値と実測値に多少ずれがあるが，DBMS に対して書き込みレートの計測を行う際に，サンプリングする点を増やすことで判定の精度をさらに高められると考えられる．また，どうしても多少の誤差は生じうるため，実際は推定した境界値をそのまま使うのではなく，誤差分を考慮してある程度のマージンを確保した値を用いるのが良いと考えられる．提案する処理可能性判定手法は，ディスクキャッシュの有無に依存せずに使用可能であることが本実験によって確認されたので，以降ではディスクキャッシュが有効な場合のみを対象として実験を行う．

6.2 実 験 2

実験 2 では，最適化器の有効性と性能を評価した．

6.2.1 初期プラン

共通演算統合ステップでは，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．本実験では，**図 14** に示す 2 つの初期プランを作成した．初期プラン 1 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するというグループを 5 個含んでいる．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．初期プラン 2 は，6 つの蓄積要求が 1 つの射影演算を共有するグループ n 個で構成されている（n は可変）．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 タプルあたりの処理時間は 1 ms と設定し，選択率はいずれも 1.0 となるように選択演算の条件を設定した．

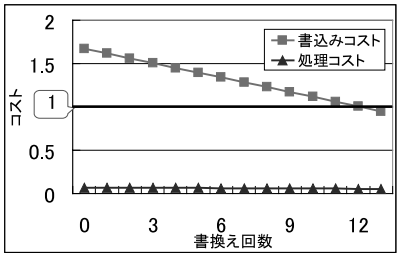

図 15 処理プランの書き換え回数
Fig. 15 Number of rewriting plans.

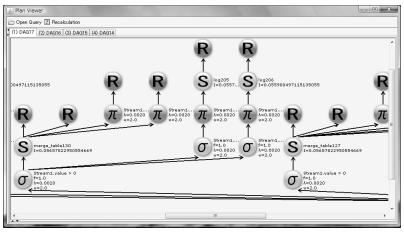

図 16 実験用プログラムの出力
Fig. 16 An output of experimental program.

6.2.2 処理プランの書き換え回数

本実験では，最適化器が処理可能プランを導出するまでに必要な処理プランの書き換え回数を測定した．Turbine ストリームの到着レートを 2 タプル/秒と設定し，初期プラン 1 を用いた．また，MySQL の書き込みレートを実測した結果 35.9 タプル/秒であった．実験結果を**図 15** に示す．横軸には書き換え回数をとり，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，書き込み処理可能の条件である書き込みコストが 1 以下になるまで 13 回の書き換えが行われ，そのつど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 タプルの蓄積が必要である．しかし，提案手法では DBMS の書き込みレートに対して，毎秒 34 タプルの蓄積（蓄積演算数 17 個）が必要となるプランを生成した．また**図 16** は，このときの実験用プログラムの出力である．図 16 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，すべての蓄積演算を統合するのではなく，必要最低限の書き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．

6.2.3 有効性の検証

ここでは最適化し終わったプランをプロトタイプシ

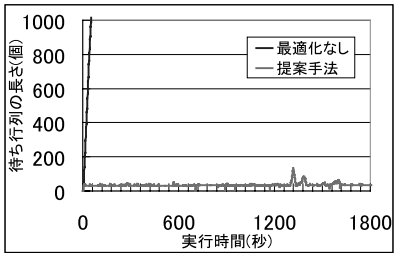

図 17 実行時間と書き込み待ち行列
Fig. 17 Processing time vs. store queue size.

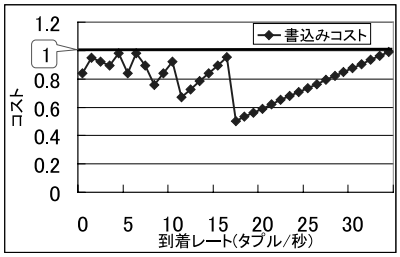

図 19 到着レートと書き込みコスト
Fig. 19 Arraival rate vs. writing cost.

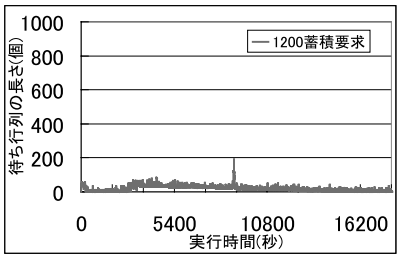

図 18 1,200 蓄積要求の処理
Fig. 18 Processing 1,200 requirements.

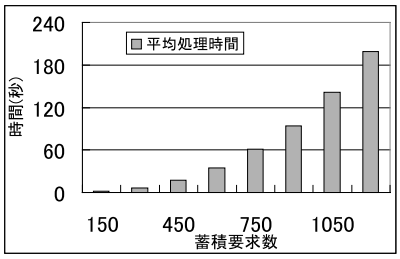

図 20 処理可能プラン導出時間
Fig. 20 Elapsed time for plan generation.

ステムに渡して処理を実行し，実際に継続的な処理が可能か検証した．Turbine ストリームの到着レートは 2 タプル/秒に設定し，初期プランとして蓄積演算数 30 個（グループ数 5）の初期プラン 2 を用いた．**図 17** 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．開始して急激に待ち行列の長さが増加している方が，最適化を行っていないプランの実行結果である．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 4,600 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を行ったプランでは，30 分間動かし続けても待ち行列の長さに目立った増加がなく，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 1,200 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 18**）．タイミングの関係で一時的に待ち行列の長さが増加することもあったが，システムを破綻させることはなかった．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．

6.2.4 最適化器の性能

最後に最適化器の性能について評価した．**図 19** 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最適化器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，図 14 の初期プラン 1 を用いた．図 19 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 19 より，ストリームの到着レートを様々に変化させても，それに応じて DBMS の書き込み性能を超えないプランが導出できることを確認した．

また，初期プラン 2 を用いて処理可能プラン生成までの処理時間を計測した（**図 20**）．図 20 は蓄積演算数を 150〜1,200 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，1,200 個もの蓄積要求に対してもおよそ 3 分で処理可能プランが導出可能である．

7. 関 連 研 究

ストリームデータに対する問合せ処理システムとして，Aurora[2] がある．Aurora はストリームの新規到着データに対して問合せを実現するストリーム処理エンジンである．Aurora ではストリーム処理が可能か判定し，処理が困難であるときには処理するデータ量を自動的に減らす load shedding 機能を提供している．対して Harmonica では，到着データに対する蓄積処理や到着データと蓄積データの統合処理も可能である．また，蓄積データを共有することによって，シ

ステムの性能に応じた最適化を実現する．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 Harmonica は有効である．今後の拡張として，Harmonica の最適化手法と load shedding を組み合わせて，動的なストリームに対しても複数蓄積要求の最適化を実現することが考えられる．

KRAFT[14] もまた，ストリームに対する問合せ処理を実現するシステムである．KRAFT は，データの鮮度と永続性の両立，類似検索，周期的監視という 3 つの特徴を持ち，新規到着データに対する連続的問合せだけでなく，ストリームデータの蓄積も考慮したセンサデータベースである．センサ型というデータ型を導入して，センサ型どうしのユークリッド距離やダイナミックタイムワーピング距離の計算が可能である．センサ型はセンサデータがシステムに到着した時間とデータの組から構成されている．また，すべての機能を備えた単一のシステムとして実装されており，高速な蓄積処理を行うことも可能である．Harmonica では，DBMS との連携のために SQL で使われているデータ型のみを利用することによって，ストリームデータ列どうしの類似度の計算を行う．類似度の計算はユーザ定義関数によって実現することが可能である．また，Harmonica はメディエータ・ラッパー型で構成されており，既存の DBMS をそのまま利用することができる．さらに，KRAFT の高速な蓄積処理と Harmonica で提供する最適化手法はそれぞれ独立に適用可能であるため，Harmonica で利用する DBMS として KRAFT を利用するといった拡張も考えられる．

その他のストリーム処理エンジンとして，Borealis[3] や TelegraphCQ[4]，STREAM[5]，NiagaraCQ[12),13)] などがあるが，いずれも蓄積に関する機能については触れられていない．

また，Harmonica の最適化に関連する研究として文献 15) がある．文献 15) では，DBMS に対するビューを用いた最適化手法を提案しており，頻繁に結合処理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，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を解決するものでもない．

8. おわりに

本論文では，ストリーム処理エンジンと DBMS を連携させたデータベース管理システム Harmonica の設計と実装について述べた．Harmonica は，ストリームに対する連続的問合せ処理，蓄積処理，新規到着データと蓄積データの統合処理を実現する．Harmonica はまた，複数の蓄積要求がシステムに登録されることによって DBMS の書き込み性能を超えてしまう場合に，複数の蓄積要求に含まれるタプルの共通性に着目し最適化を行う．本論文ではまた，プロトタイプシステムを開発し，Harmonica の有効性について示した．

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，読み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 本研究の一部は，日本学術振興会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．

参考文献

1) 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).
2) 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).
3) Abadi, D.J., Ahmad, Y., Balazinska, M., Çetintemel, U., Cherniack, M., Hwang, J.-H., Lindner, W., Maskey, A.S., Rasin, A., Ryvkina, E., Tatbul, N., Xing, Y. and Zdonik, S.: The Design of the Borealis Stream Processing Engine, *Proc. CIDR*, pp.277–289 (2005).
4) Chandrasekaran, S., Cooper, O., Deshpande, A., Franklin, M.J., Hellerstein, J.M., Hong, W., Krishnamurthy, S., Madden, S., Raman, V., Reiss, F. and Shah, M.A.: TelegraphCQ: Continuous Dataflow Processing for an Uncertain World, *Proc. CIDR* (2003).
5) Motwani, R., Widom, J., Arasu, A., Babcock, B., Babu, S., Datar, M., Manku, G.S., Olston, C., Rosenstein, J. and Varma, R.: Query Processing, Approximation, and Resource Management in a Data Stream Management System, *Proc. CIDR* (2003).
6) Arasu, A., Babu, S. and Widom, J.: The CQL continuous query language: Semantic foundations and query execution, *The VLDB Journal*, Vol.15, No.2, pp.121–142 (2006).
7) StreamSpinner. http://www.streamspinner.org
8) Ayad, A.M. and Naughton, J.F.: Static optimization of conjunctive queries with sliding windows over infinite streams, *Proc. ACM SIGMOD*, pp.419–430 (2004).
9) Viglas, S.D. and Naughton, J.F.: Rate-based

query optimization for streaming information sources, *Proc.ACM SIGMOD*, pp.37–48 (2002).
10) Tatbul, N., Çetintemel, U., Zdonik, S., Cherniack, M. and Stonebraker, M.: Load Shedding in a Data Stream Manager, *Proc. VLDB*, pp.309–320 (2003).
11) Roy, P., Seshadri, S., Sudarshan, S. and Bhobe, S.: Efficient and extensible algorithms for multi query optimization, *Proc. ACM SIGMOD*, pp.249–260 (2000).
12) Chen, J., DeWitt, D.J. and Naughton, J.F.: Design and Evaluation of Alternative Selection Placement Strategies in Optimizing Continuous Queries, *Proc. ICDE*, pp.345–356 (2002).
13) Chen, J., DeWitt, D.J., Tian, F. and Wang, Y.: NiagaraCQ: A scalable continuous query system for Internet databases, *Proc. ACM SIGMOD*, pp.379–390 (2000).
14) 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：データベース，Vol.45, No.SIG14(TOD 24), pp.39–53 (2004).
15) Mistry, H., Roy, P., Sudarshan, S. and Ramamritham, K.: Materialized View Selection and Maintenance Using Multi-Query Optimization, *Proc. ACM SIGMOD*, pp.307–318 (2001).

(平成 19 年 3 月 20 日受付)
(平成 19 年 7 月 4 日採録)

(担当編集委員　井上 潮)

山田　真一

2005 年筑波大学第三学群情報学類卒業．2007 年同大学大学院博士前期課程システム情報工学研究科修了．ストリームデータの蓄積・活用に関する研究に従事．修士（工学）．現在は NTT コムウェア（株）に勤務．日本データベース学会会員．

渡辺　陽介（正会員）

2001 年筑波大学第三学群情報学類卒業．2006 年同大学大学院博士課程システム情報工学研究科修了．博士（工学）．現在は科学技術振興機構戦略的創造研究推進事業における研究員として，自律連合システムにおけるデータ・インターオペラビリティに関する研究活動に従事．日本データベース学会，ACM 各会員．

北川　博之（フェロー）

1978 年東京大学理学部物理学科卒業．1980 年同大学大学院理学系研究科修士課程修了．日本電気（株）勤務の後，1988 年筑波大学電子・情報工学系講師．同助教授を経て，現在，筑波大学大学院システム情報工学研究科教授，ならびに計算科学研究センター教授．理学博士（東京大学）．異種情報源統合，XML とデータベース，データマイニング，センサデータベース，WWW データ管理等の研究に従事．著書『データベースシステム』（昭晃堂），“The Unnormalized Relational Data Model”（共著，Springer-Verlag）等．日本データベース学会理事，電子情報通信学会フェロー，ACM，IEEE-CS，日本ソフトウェア科学会各会員．

天笠　俊之（正会員）

1994 年群馬大学工学部情報工学科卒業．1999 年同大学大学院工学研究科修了．奈良先端科学技術大学院大学情報科学研究科助手を経て，2005 年から筑波大学大学院システム情報工学研究科講師（筑波大学計算科学研究センター講師を兼任）．博士（工学）．XML データベース，e サイエンスにおけるデータベース応用等の研究に従事．日本データベース学会，電子情報通信学会，ACM，IEEE Computer Society 各会員．