

広域分散ストリーム処理環境における演算配置最適化手法の評価

稲守 孝之[†] 渡辺 陽介^{††} 北川 博之^{†,†††} 天笠 俊之^{†,†††} 川島 英之^{†,†††}

[†] 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

^{††} 科学技術振興機構 戦略的創造研究推進事業

^{†††} 筑波大学 計算科学研究センター

E-mail: [†]{tinamo,watanabe}@kde.cs.tsukuba.ac.jp, ^{††}{kitagawa,amagasa,kawasima}@cs.tsukuba.ac.jp

あらまし センシングデバイスやネットワークの技術発達などにより実世界センサデータや計算機の監視ログといった大規模ストリームデータが各所で取得可能となった昨今、これらに対する高度利用要求が高まっている。このような背景から分散ストリーム処理の研究・開発が行われている。本研究では、我々の研究グループで研究・開発を行っているストリーム処理エンジン StreamSpinner をベースに構築した分散ストリーム処理環境を管理する ORINOCO システムを構築した。ORINOCO システムは、分散ストリーム処理において標準演算処理とアプリケーション処理を融合するフレームワークを提供し、このフレームワークにおいてネットワークのデータ転送量を最小とする演算の配置最適化を行う。本論文では、評価実験を行い ORINOCO システムにおける配置最適化手法の有効性を検証する。
キーワード 分散ストリーム処理, 放送情報システム, 分散 DB

An Evaluation of Optimization Method for Allocation of Operators in Widely Distributed Stream Processing Environment

Takayuki INAMORI[†], Yousuke WATANABE^{††}, Hiroyuki KITAGAWA^{†,†††}, Toshiyuki AMAGASA^{†,†††},
and Hideyuki KAWASHIMA^{†,†††}

[†] Graduate School of Systems and Information Engineering, University of Tsukuba
Tennoudai, Tsukuba-shi, 305-8573 Japan

^{††} Japan Science and Technology Agency, Core Research for Evolutional Science and Technology (JST/CREST)

^{†††} Center for Computational Sciences, University of Tsukuba

E-mail: [†]{tinamo,watanabe}@kde.cs.tsukuba.ac.jp, ^{††}{kitagawa,amagasa,kawasima}@cs.tsukuba.ac.jp

Abstract Rapid developments of sensing device technologies and network bring the increase of data streams that deliver up-to-date information from wide area to users such as sensor data from real world and monitoring data of widely distributed computers. Since a demand for scalable query processing on a huge amount of data streams becomes quite important, distributed stream processing systems have been taken notice. We have developed a management system of distributed stream processing environment named ORINOCO. The distributed environment is based on StreamSpinner which is a stream processing engine our research group have developed. ORINOCO system has a framework which can integrate relational algebra operation and application processing. In this framework, ORINOCO also can minimize network traffic by allocating operators to proper nodes. In this paper, we evaluate our optimization method for allocation of operators based on experiments.

Key words Distributed Stream Processing, Broadcasting Information Services, Distributed DB

1. はじめに

近年、センシングデバイスやネットワークの発展に伴い、絶えず変動する情報を次々と配信するストリームデータと呼ばれる情報源が増大している。ストリームデータの例は、web 上で配信される株価データやニュースコンテンツ、センサデバイス

から配信される温度・光・音データ、カメラから配信される映像データなどである。web 上のデータや DB に格納されているデータは利用者が自ら取得するのに対し、ストリームデータは利用者に対して能動的かつ連続的に配信される。このような性質を持つストリームデータに対して、フィルタリングや他の情報源との統合のような問合せ処理要求が高まっている。

そこで、これらの要求を実現するストリーム処理エンジン [1] ~ [3] の研究・開発が行われている。我々の研究グループにおいても、StreamSpinner [4], [5] というストリーム処理エンジンを開発中である。StreamSpinner は、ストリームデータに対して連続的な問合せ処理を実行し、その結果を利用者に提供する。問合せ処理は、リレーショナル代数の選択、結合、直積、射影演算の組合せで実現される。StreamSpinner は、他の StreamSpinner の処理結果を受け取る連携機能を有する。

実世界情報などのストリームデータは、その情報源が地理的に分散していることが多々ある。そこで、分散した情報源から配信されるストリームデータを効率的に処理するために、分散ストリーム処理 [6], [7] の研究が行われている。我々の研究グループにおいても、StreamSpinner を構成単位とする分散ストリーム処理環境を構築した、本研究では、この処理環境を管理・運用するための ORINOCO システム [8] を開発した。ORINOCO システムを使用することで、分散ストリームを収集するための問合せ要求と、収集されたデータに対する任意のアプリケーション処理要求を定義することができる。

ORINOCO システムは、上記の問合せを演算単位に分割し、個々の演算およびアプリケーションプログラムを各ノードに配置した後、各ノードを連携させることで分散ストリーム処理を実現する。

我々は、演算・アプリケーションプログラムの配置最適化手法を [9] で提案した。この手法では、ノード間のネットワーク遅延とストリームデータの入力レートの時間的変化に着目し、データ転送量においてこれらの変化による影響が最小となるような配置プランを導出する。本研究では、この提案手法の有効性を実験を基に評価する。

本論文では、2. で ORINOCO を用いた分散ストリーム処理と配置最適化手法について解説する。そして 3. で、実験により提案手法を評価する。その後 4. で、実環境で動作するシステムについて紹介し、5. で関連研究を紹介し、最後に 6. でまとめと今後の課題について述べる。

2. ORINOCO システムを用いた分散ストリーム処理

本節では、ORINOCO システムを用いた分散ストリーム処理について概説する。

図 1 に示すのは、ORINOCO システムを用いた分散ストリーム処理の概要である。本研究では、分散ストリーム処理環境を構成するストリーム処理エンジンに、我々の研究グループで開発している StreamSpinner を使用した。StreamSpinner は、情報源から配信されるストリームデータを、リレーショナル代数のテーブル形式に変換して扱う。こうすることで、データに対しリレーショナル代数における選択、射影、結合、直積演算を行うことができ、異種のストリームデータの統合や、ストリームデータと DB に格納されたデータとの統合を可能とする。また StreamSpinner は、他の StreamSpinner の処理結果を受け取る連携機能を有する。

図 1 では、StreamSpinner が各ノードに分散配置され、近隣

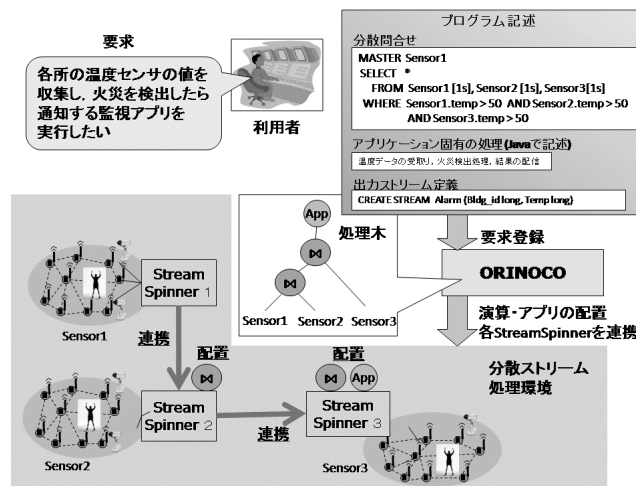


図 1 ORINOCO を用いた分散ストリーム処理
Fig. 1 Distributed Stream Processing with ORINOCO

のセンサと接続し、温度データを逐次取得している。ここでは、利用者は「各所の温度センサの値を収集し、火災を検出したら通知する監視アプリを実行したい」という要求を持っているものとする。この要求を満たすために必要な処理は以下の 3 項目である。

- (1) 温度データの収集処理
- (2) 火災を検出するアプリケーション処理
- (3) 検出結果の配信処理

上記 (1) ~ (3) の処理を実現するために、ORINOCO システムは、利用者が以下の 2 つの処理要求を定義できるようなフレームワークを有している。

- 分散問合せ処理要求：分散した情報源から配信されるストリームデータに対しての収集処理要求を、独自の SQL ライクな言語で定義することができる。
- アプリケーション処理要求：分散問合せによって収集されたデータに対する任意のアプリケーション処理を定義することができる。

利用者はこれら 2 つの処理要求をプログラム記述として ORINOCO システムに与える。プログラムを記述するには、ORINOCO システムが提供する API を使用する。API とプログラム記述については 2.1 で解説する。ORINOCO システムはこのプログラム記述から、リレーショナル代数演算とアプリケーション処理で構成される処理木を導出する (図 1 中央)。そして各演算およびアプリケーションプログラムをそれぞれノードに配置し、各 StreamSpinner を連携させることで分散ストリーム処理を実現する。配置ノードを決定する最適化手法は 2.2 で述べる。

2.1 API とプログラム記述

ORINOCO システムはプログラム記述のために、API を用意している。この API は Java のインタフェースである。API の概念図を図 2 に示す。利用者は分散ストリーム処理環境を複数のストリーム・DB が接続された一つの仮想ストリーム処理システムとして捉えることができる。こうすることで、利用者はアプリケーションプログラムが実際にどのノードで実行される

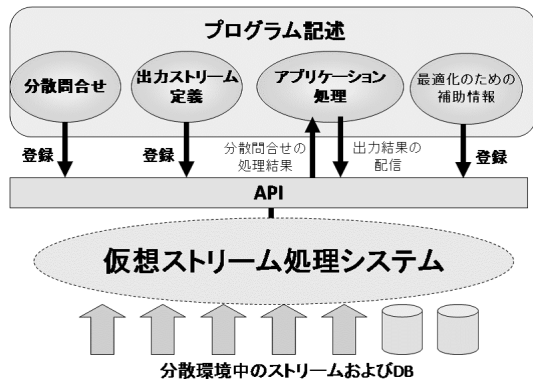


図2 API の設計コンセプト
Fig.2 Design Concept of API

```

1: public class Example extends WorkerPocket{
2:   // 分散問合せを登録
3:   public static String[] getQueries{
4:     String[] queries =
5:     { "MASTER Sensor1 " *
6:     "SELECT * "+
7:     "FROM Sensor1 [1s], Sensor2 [1s], Sensor3[1s]" +
8:     "WHERE Sensor1.temp > 50 AND Sensor2.temp > 50" +
9:     "AND Sensor3.temp > 50" };
10:    return queries;
11:  }

12: // アプリケーション処理の定義
13: public void dataDistributed( WPIInputRowSet in_rs, int ){
14:   // 火災検出処理
15:   ...
16:   // 火災結果配信処理
17:   ....
18: }

19: // 出力ストリーム定義を登録
20: public static String[] getStreamDefinitions(){
21:   String defs = {"CREATE STREAM alert ( Bldg_id long, Temp long)"};
22:   return defs;
23: }
24: ...
25: }

```

図3 プログラム記述例
Fig.3 Example of Program Description

かということ意識する必要がなくなるという利点がある。また分散問合せ要求記述においても、仮想ストリーム処理システムに対するものとして行う。プログラム記述における構成要素は図2にあるとおりである。

次に、プログラムの記述例を図3に示す。利用者は、WorkerPocket というクラスを継承したクラスを作成する(1行目)。分散問合せは5~9行目のように記述され getQueries メソッド(3行目)の戻り値として設定する。分散問合せ要求に対する処理方式は、連続的問合せというものである。連続的問合せとは、ストリームが到着する度に処理を実行し、前回までの処理結果との差分を結果として返す問合せ方式のことである。ORINOCOでは、分散問合せ要求の記述方式としてSQLライクな記述言語を提供する。この分散問合せの内容は、Sensor1 というストリームからデータが到着したとき、Sensor1,2,3の温度情報 Temp がすべて 50 を超えていたら、それらの温度情報を返してほしいというものである。MASTER 節では処理実行のタイミングをとるストリームを指定する。ここでは Sensor1 が指定されてい

るので、Sensor1 からデータが到着したら SELECT 以下の処理が実行される。SELECT-FROM-WHERE 節は、FROM 節でウィンドウサイズを指定できること以外は SQL のそれとほぼ同等である。ウィンドウサイズとは、処理実行時にどれだけ遡ったデータまでを処理対象とするかを指定する値である。この分散問合せでは、FROM 節で Sensor1,2,3 のウィンドウサイズを 1s と指定することで、処理実行時点から 1 秒前までのデータを処理対象にすることを要求している。

アプリケーション処理の定義は dataDistributed メソッド(13行目)内で行う。この部分には任意の処理を記述することが可能である。

プログラム記述において、利用者はアプリケーションプログラムの処理結果を出力ストリームとして定義することができる。出力ストリーム定義は21行目のようになり、CREATE STREAM 文で行う。CREATE STREAM は SQL の CREATE TABLE 文を模した構造となっており、出力ストリーム名を定義し、その後属性名と属性型のペアを指定する。ここでは、long 型の Bldg_id と同じく long 型の Temp という属性を持った alert という名前の出力ストリームを定義している。

上記の情報の他、利用者は ORINOCO システムが配置最適化に必要な二つの情報を与える必要がある。一つめは、アプリケーションプログラムが特定のノードで動作する必要があるか否かという情報であり、もし必要がある場合はそのノード名を与える。二つめは、アプリケーション処理の選択率の概算値である。これは、一タブルの入力に対しての平均出力タブル数のことである。

2.2 演算・アプリケーションプログラムの配置最適化

論文[9]で、演算・アプリケーションプログラムの配置最適化手法を提案した。以下でその概要を述べる。

2.2.1 目的とアプローチ

分散ストリーム処理では、各ノード間のネットワーク遅延や各ストリームの入力レートなどの値を基に、ネットワーク使用量が最小となるように演算の配置を行う。ここで、ネットワーク遅延や入力レートは時刻とともに変動するという特徴がある。これらの変動に対しては、演算を再配置することで常に安定的な処理を継続できると考えられる。しかし本研究で提案したフレームワークは、演算の他にアプリケーションプログラムも分散ストリーム環境に配置する。アプリケーションプログラムは、利用者が記述した任意の処理を実行する。例えば、火災検出アプリケーションでは、過去のセンサからの温度データの履歴を使用した処理が必要であり、プログラムの内部状態として常にこのデータを保持していることが考えられる。このようにプログラムが独自に管理しているデータが存在する場合、アプリケーションプログラムを再配置するとプログラムの内部状態が消えてしまう。そこで提案手法では、分散ストリーム処理環境において時間とともに変動するパラメータに着目し、アプリケーションプログラムを移動させなくてもネットワーク使用量を下げよう配置プランを導出する。ここでネットワーク使用量とは、単位時間あたりにネットワークに流れるデータの総量のことをいう。

本研究では、各ノード間のネットワーク遅延と、各ストリームからの入力レートを時間変動するパラメータとして考慮する。ただし、全く予測不能な変動ではなく、ある程度決まってきたいくつかのパターンの中で動くものと仮定する。

提案手法の目的は、各パラメータの時間変動による影響が小さいノードにアプリケーションを配置する初期配置プランを導出することである。こうすることで、初期配置後の各パラメータ時間変動に対しては、演算の再配置のみでネットワーク使用量を下げることができる。このためには、各パラメータの変動パターンを知る必要がある。変動パターンを把握するための最もシンプルなアプローチとして、本研究では各パラメータについての大量のサンプルデータを収集する。提案手法は、それら大量のサンプルデータを基に、最適な初期配置の決定を行う。より具体的には、サンプルデータごとにネットワーク使用量が最小となるように配置場所を計算させ、それを全サンプルデータに対して繰り返したときに、もっとも多く配置先として選択されたノードを初期配置場所とする。

2.2.2 サンプルデータ

提案手法では、各ノード間のネットワーク遅延と各ノードにおけるストリームデータの入力レートをサンプルデータとして利用する。サンプルデータは、問合せ処理を行う際のネットワーク使用量を見積もるために使用される。ORINOCOシステムは、サンプルデータを各ノードで動作する StreamSpinner から取得する。サンプルデータの例は、図4中央の表にありである。Lat_{m,n} はノード m, n 間のネットワーク遅延を表し、DR_n はノード n におけるストリームデータの入力レートを表す。なお、ネットワーク遅延の単位は秒であり、入力データレートの単位はタプル毎秒である。

2.2.3 初期配置プランの決定

提案手法の処理ステップを解説する。

- (1) 時刻 $t_1 \sim t_k$ におけるサンプルデータ k 個を取得する。サンプル取得間隔 ($t_{i+1} - t_i$) は十分長いものとする。
- (2) アプリケーション記述から処理木を導出する。
- (3) k 個のサンプルデータそれぞれについて、ネットワーク使用量が最小となるように各サンプルデータにおける配置プランを k 個導出する。
- (4) k 個の配置プランからそれぞれのアプリケーションプログラムおよび演算について、各ノードの配置候補となった回数を数え、最も多かったノードに配置する。

上記ステップの具体例を、図4を用いて説明する。

- (1) $k = 100$ としてサンプルデータを取得する(図4中央)。
- (2) プログラム記述から図4左上の処理木を導出する。この処理木において、App-B はノード4に配置することが利用者からの要求により指定されているが、App-A, O_1, O_2 については初期配置を決める必要がある。
- (3) 100 個のサンプルデータそれぞれについて、配置プランを 100 個導出する(図4左下)。
- (4) App-A および演算 O_1, O_2 は、いずれも 65 回, 58 回, 40 回と、配置ノードとしてノード1が最も多く選択されたので、これらすべてをノード1に配置する。

以上のような処理を行うことによって、アプリケーションプログラムが移動できない状況において、パラメータの時間変動に対してロバストなノードに配置することができる。

2.2.4 各状態における最適配置プラン

各状態における配置プラン決定手法を解説する。配置プラン決定手法は、[10] で提案された手法を使用する。[10] の手法は、ネットワーク遅延情報と入力レート情報を基に、ネットワーク使用量を最小とする演算配置プランを導出する。この手法を本研究で使用するには、アプリケーションプログラムも他の演算と同様に配置すべき対象として扱われる。[10] の手法では、各演算の配置ノードを決定する際に式1を用いてネットワーク使用量を見積もる。

$$\sum_{l \in L} DR(l) \cdot Lat(l) \quad (1)$$

ここで L とは、問合せ処理を実行する際に使用するすべての回線の集合であり、 $DR(l)$ は回線 l における入力データレートを表し、 $Lat(l)$ は l におけるネットワーク遅延を表す。[10] の手法における演算配置アルゴリズムの処理ステップを以下に示す。

- (1) 初期状態として、各演算を入力ストリームに最も近いノードに配置する。
- (2) 情報源に近い演算から順に、式1を用いてネットワーク使用量が最小となるノードに移動させる。
- (3) すべての演算を評価し終わった後、ネットワーク使用量を前回のプランと比較して、削減された量が閾値以下だったらそのプランを出力する。そうならなければ処理(2)を再び実行する。

以上の手順により、各サンプルデータにおいて式1の値を最小化する配置場所が決定する。

2.2.5 演算の再配置

初期配置が決まった後は、実際の処理を開始するために、ノードへアプリケーションプログラムが配置され、StreamSpinner に連携要求が出される。前述の通り、アプリケーションプログラムが稼働し始めた後は、各パラメータに変動が起こっても、アプリケーションプログラムを初期配置から動かすことはできない。そこで、移動可能な演算だけを再配置して、変動に対するネットワーク使用量の調節を行う。演算再配置の手順は以下の通りである。この処理を定期的に行う。

- (1) 分散ストリーム処理環境において、一定の時間間隔においてサンプルデータを取得する。
- (2) 現在から i 回前までの各サンプルについて、アプリケーションプログラムの動作ノード情報を基に、[10] の手法を用いて最適なプランを導出する。
- (3) i 個のプランから、各演算について配置ノードとなる回数が最も多かったノードを選択する。再配置後のプランのネットワーク使用量が、現在のネットワーク使用量よりも閾値以上低くなっていたら(4)を実行する。
- (4) (3)の結果と初期現在のプランを比較して、配置ノードが異なる演算があれば、その演算を(3)で選択したノードに再配置する。

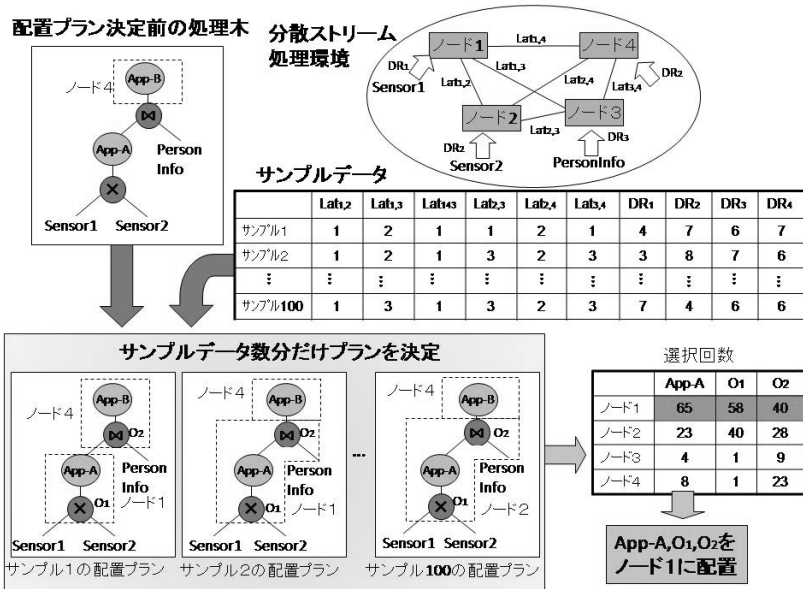


図4 初期配置プランの決定
Fig. 4 Decision of Initial Allocation Plan

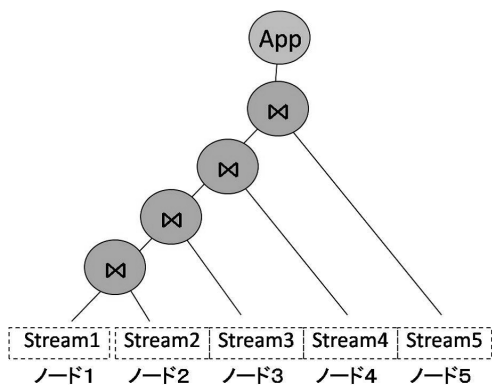


図5 実験における分散問合せを表現する処理木
Fig. 5 Processing Tree Representing Distributed Query on Experiment

以上の処理を行うことで、最近のパラメタの傾向に合った最適なプランに再構成することができる。

3. 評価実験

本節では、提案手法における初期配置と再配置の有効性を評価する実験を行う。

本実験の概要を述べる。本実験では、ノード1からノード5の全5ノードで構成された分散ストリーム処理環境に対して、Stream1からStream5までの5ストリームの結合演算処理を要求する分散問合せが与えられたものとする。Stream1からStream5は、それぞれノード1からノード5に接続している。分散問合せは図5のような処理木で表すことができる。すべての結合演算の選択率は0.01とする。ネットワーク遅延とストリームの入力レートは、人工的に生成したデータを用いる。実験では、ネットワーク遅延の単位を秒とし、各ストリームの入力レートの単位をタプル/秒とする。全ノード間のネットワーク遅延を0.64とし、全ストリームの入力レートを12.5とする。

以上のような前提の基で、分散ストリーム処理環境中で以下に示す状態1から状態5の5つの状態においてそれぞれ実験を行う。

- 状態1：ノード1と他のノードとの間のネットワーク遅延およびStream1の入力レートが変動
- 状態2：ノード2と他のノードとの間のネットワーク遅延およびStream2の入力レートが変動
- 状態3：ノード3と他のノードとの間のネットワーク遅延およびStream3の入力レートが変動
- 状態4：ノード4と他のノードとの間のネットワーク遅延およびStream4の入力レートが変動
- 状態5：ノード5と他のノードとの間のネットワーク遅延およびStream5の入力レートが変動

上記の変動において、すべてのネットワーク遅延は、70%の確率で0.5となり30%の確率で1.0となるものとし、すべてのストリームの入力レートは、70%の確率で5となり30%の確率で30となるものとする。本実験では、各パラメタにおいて70%で出現するデータをメジャーデータと呼び、30%で出現するデータをマイナーデータと呼ぶこととする。

これらの分布を基に生成されたサンプルデータを基に、提案手法を用いて初期配置プランを導出する。本実験では、配置プランの評価基準として、ネットワーク使用量を用いる。ネットワーク使用量とは、配置プランにおける各データ転送ルートのネットワーク遅延とデータ転送レートの積の総和である(2.2の式1)。各配置プランのネットワーク使用量は理論的に算出する。

各状態においてサンプルデータを50組生成し、以下の手法で配置プランを導出する。

- 提案手法：50のサンプルデータセットより2.2で述べた手順で初期配置プランを導出

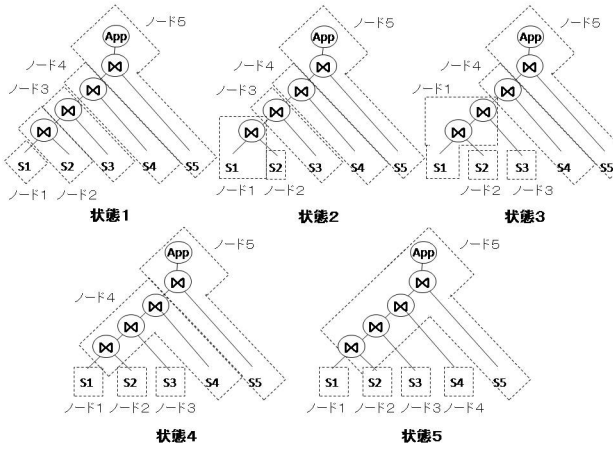


図6 各状態における初期配置プラン
Fig. 6 Initial Allocation Plan in Each Case

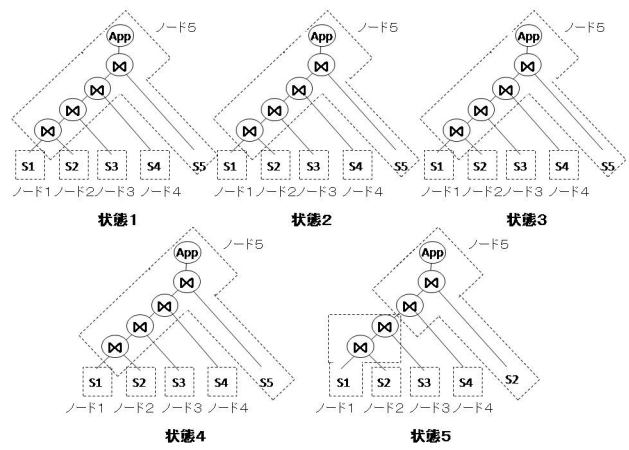


図8 各状態における再配置プラン
Fig. 8 Reallocation Plan in Each Case

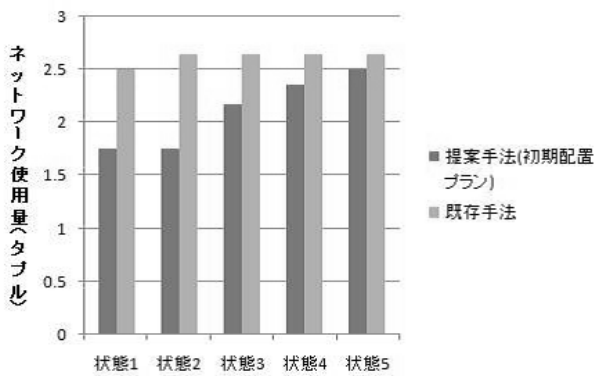


図7 メジャーデータにおける各配置プランのネットワーク使用量
Fig. 7 Network Usage of Each Allocation Plan with Major Data

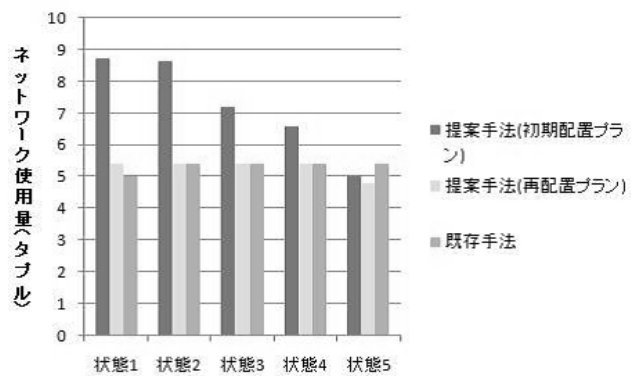


図9 マイナーデータにおける各配置プランのネットワーク使用量
Fig. 9 Network Usage of Each Allocation Plan with Minor Data

• 既存手法: [10] で提案され 2.2 内で述べた手順のみを用いて配置プランを導出。サンプルデータセットより計算したネットワーク遅延と入力レートの平均値を用いて最適化を行う。

各状態において、提案手法で導出した初期配置プランは図6のようになった。図6では、Stream1~Stream5をS1~S5というように略記している。また既存手法による配置プランは、すべての演算・アプリケーションプログラムをノード1に配置するようなプランになった。

実験 1. 初期配置プランの評価

上記で示した各配置プランにおいて、各パラメタがメジャーデータである場合のネットワーク使用量を計算した結果を図7に示す。各状態において、提案手法の初期配置プランによるネットワーク使用量は既存手法を下回った。状態1および状態2において、提案手法と既存手法のネットワーク使用量の差が大きかった理由としては、第一の結合演算の入力であるStream1およびStream2の入力レートが低く、その後の結合演算の出力レートに大きな影響を及ぼしているためだと考えられる。

実験 2. 再配置プランの評価

次に、上記の提案手法による初期配置プランを基に、各パラメタの値がマイナーデータに変化した場合を想定し、演算の再配置プランを導出する。それぞれの再配置プランは図8のとおりである。

提案手法の初期配置プラン・再配置プランおよび既存手法による配置プランにおいて、各パラメタがマイナーデータである場合のネットワーク使用量を図9に示す。各状態において、提案手法による初期配置プランは高いネットワーク使用量を示したが、再配置プランでは既存手法とほぼ同等になることが確認できた。この実験において、マイナーデータにおいても、アプリケーションプログラムを移動させずに演算の再配置のみを行うことでネットワーク使用量を軽減できることが確認された。状態1および状態2において提案手法の初期配置プランのネットワーク使用量が高いのは、第一の結合演算の入力であるStream1およびStream2の入力レートが高くなったためであると考えられる。このような事態を避けるには、入力レートが高くなると予想されるストリームが第一の結合演算の入力とならぬように、処理木における演算の組合せを調整する方法がある。

4. 実機上での運用

本研究では、ORINOCOシステムを用いて実際の分散ストリーム処理環境の管理・運用を行った。分散ストリーム処理環境は、情報爆発プロジェクトのInTrigger[11]を使用し、StreamSpinnerにより構築した(図10)。InTriggerは、日本国内で7拠点にまた

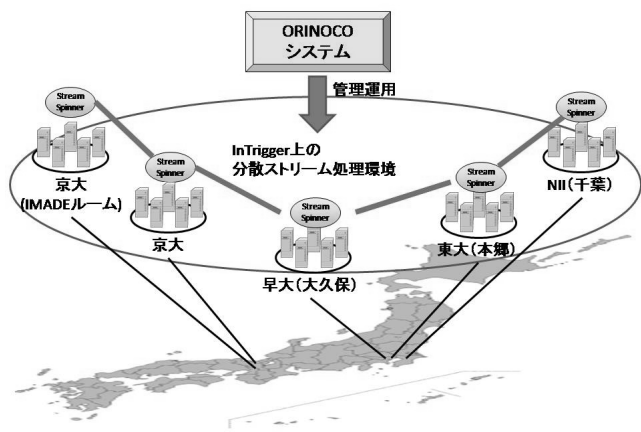


図 10 InTrigger 上の分散環境を管理する ORINOCO

Fig. 10 ORINOCO Managing Distributed Environment on InTrigger

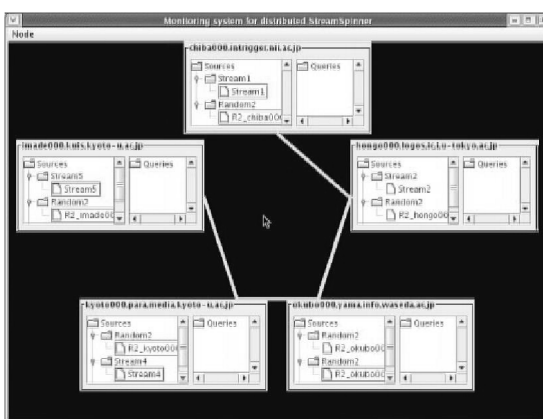


図 11 ORINOCO システムの実行画面

Fig. 11 A Screenshot of ORINOCO System

がり数百ノードを持つ、分散処理の研究などのために構築されたプラットフォームである。InTrigger 上で動作する ORINOCO システムの実行画面を図 11 に示す。図 11 では 5 ノードで構成される分散ストリーム処理環境を管理しており、利用者は各ノードの連携状態や管理するストリームを閲覧することができる。ここでは、図 5 の問合せ要求が登録され、アプリケーションプログラムは京大 (IMADE ルーム) に配置された。

5. 関連研究

ストリーム処理システムに関する研究について述べる。まず、1 台のマシンで動作する集中型のストリーム処理システムとして、Aurora [1]、TelegraphCQ [2]、STREAM [3] などがある。本研究における各ノードで動作する StreamSpinner は、これらに相当する機能をもつ。StreamSpinner の特徴はイベント駆動型の連続的問合せ処理や、複数問合せ最適化機能である。

分散型のストリーム処理システムとしては、Aurora をベースに実現された Medusa [6] と Borealis [7] がある。これらのシステムでは、利用者からの問合せ処理要求は複数の演算をつないだグラフとして与えられ、登録された演算はシステムによって適切なノードに割り当てられる。

演算の割当て手法としては、各ノードの負荷を平均的にする手法 [12] や、ストリームの到着レート変動に強くなるようにする手法 [13] が提案されている。また、SBON [10] は Borealis などの分散ストリーム処理システムと連携し、演算の配置場所を指示するためのシステムで、ネットワーク利用コストを最小化するように演算の割り当てを行っている。これらの割当て手法は、再配置可能な演算に対しての配置最適化を行っている。本研究の提案手法は、再配置不可能な WorkerPocket も考慮した配置最適化を行っている。

分散型ストリーム処理に関連するするほかの研究として、分散ストリーム処理システムの高信頼化を扱ったもの [14]~[16] があるが、これらは本研究とは目的が異なる。

6. おわりに

本論文では、本研究で提案した演算・アプリケーションプログラムの配置最適化手法の評価を実験に基づいて行った。

今後の課題は、実環境においての提案手法の有効性を評価する実験を行うことが挙げられる。

謝辞 本研究の一部は、科学研究費補助金特定領域研究 (# 19024006)、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] D. J. Abadi, et al., "Aurora: a new model and architecture for data stream management", VLDB Journal Vol.12, No.2, pp.120-139, 2003.
- [2] S. Chandrasekaran, et al., "TelegraphCQ: Continuous Dataflow Processing for an Uncertain World", Proc. CIDR, 2003.
- [3] R. Motwani, et al., "Query Processing, Resource Management, and Approximation in a Data Stream Management System", Proc. CIDR, 2003.
- [4] 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.
- [5] StreamSpinner. <http://www.streamspinner.org>
- [6] M. Cherniack, et al. "Scalable Distributed Stream Processing", Proc. CIDR, 2003.
- [7] Daniel J. Abadi, et al., "The Design of the Borealis Stream Processing Engine", Proc. CIDR, 2005.
- [8] 稲守孝之, 渡辺陽介, 北川博之, 天笠俊之. "分散ストリーム処理環境のための運用管理システムの提案" DEWS2007, 2007 年.
- [9] 稲守孝之, 渡辺陽介, 北川博之, 天笠俊之, 川島英之. "分散ストリーム処理環境におけるアプリケーション配置最適化手法" 夏のデータベースワークショップ DBWS2007, 2007 年.
- [10] P.Pietzuch, et al., "Network-Aware Operator Placement for Stream Processing Systems", Proc. ICDE, p.49, 2006.
- [11] InTrigger, <https://www.logos.ic.u-tokyo.ac.jp/intrigger/registration/>
- [12] Y. Xing, et al., "Dynamic Load Distribution in the Borealis Stream Processor", Proc. ICDE, pp. 791-802, 2005.
- [13] Y. Xing, et al., "Providing Resiliency to Load Variations in Distributed Stream Processing", Proc. VLDB, pp. 775-786, 2006.
- [14] M. Balazinska, et al., "Fault-tolerance in the Borealis distributed stream processing system", Proc. ACM SIGMOD, pp.13-24, 2005.
- [15] J. Hwang, et al., "High-Availability Algorithms for Distributed Stream Processing", Proc. ICDE, pp. 779-790, 2005.
- [16] 渡辺陽介, 山田真一, 北川博之. "分散環境におけるストリーム処理の高信頼化" 夏のデータベースワークショップ DBWS2006, 2006 年. 電子情報通信学会技術研究報告 Vol. 106, No. 149, pp. 203-208.