社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2007－DBS－143（58）
2007／7／3

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

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 ネットワークやデバイス技術の進展に伴い、広域に配置されたセンサー等から得られる大規模時系列ストリームデータへの収集・監視要求が増加している。我々の研究グループでは、それらの要求を実現する分散ストリーム処理環境およびその管理ツールの開発を行っている。本環境は複数の処理ノードから構成されるため、利用者からの問合せ処理やアプリケーションプログラム本体の実行をどのノードで行うかが極めて重要となる。本論文では、提案システムにおける処理配置の最適化方式について述べる。提案手法は、ノードの負荷やネットワーク遅延だけでなく、アプリケーションプログラムの性質も考慮して適切な配置場所を決定する。

キーワード 分散ストリーム処理，放送型サービスと DB，並列・分散 DB

Optimization Method for Allocation of Applications in 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 1–1–1, 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 As technologies of network and sensor devices have been developed, a demand for collecting and monitoring large-scaled streaming data that are provided from distributed information sources is increasing. Our research group has been developing a management system for distributed stream processing environments. Since a distributed stream environment consists of many nodes, we have to decide which nodes are appropriate to allocate operators and application programs. In this paper, we propose an optimization method for task allocation in distributed stream processing environment. The method considers not only input rates and network latencies but also characteristics of application programs.

Key words Distributed Stream Processing, Broadcast Services and DB, Parallel and Distributed DB

1. はじめに

近年、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。

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、それらを管理・運用するためのORINOCOシステム[8]を開発している。ORINOCOシステムは、分散ストリーム処理環境の利用者に対して以下の機能を提供する。

- 問合せ言語：分散環境中に点在するストリームデータに対する収集・監視要求をSQLライクな問合せ言語によって記述可能である。
- API：分散ストリーム処理環境を利用するアプリケーションプログラムの作成を支援するための枠組みであり、問合せ結果の取得や、プログラムの出力を新たなストリームとして配信する機能を提供する。
- 問合せ最適化：問合せを演算単位に分割し、各演算をそれぞれ適したノードに割り当てるようにプランニングする。アプリケーションプログラム自身の配置場所も制御する。
- 各ノードの状態監視：問合せ最適化に必要となる、ネットワーク遅延やノード負荷情報等の収集。

ORINOCOシステムを使用することで、利用者は地理的に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任意の処理を実行するアプリケーションを作成することができる。ORINOCOシステムは、アプリケーションを適切なノードに配置し、そこにストリームデータが収集されるように各StreamSpinnerを連携させる。

我々は[8]で、ORINOCOシステムにおける演算およびアプリケーションプログラムの配置ノードを決めるための最適化方式を提案したが、この手法はネットワーク遅延やストリームからの入力レートなどのパラメータが定常的であることを仮定していた。より一般的なケースを扱うには、これらのパラメータの時間変化を考慮しなければならない。その場合、各処理に適した配置場所も時間変化するため、それに応じて演算およびアプリケーションプログラムの再配置が必要となる。だが、アプリケーションプログラムは利用者の記述した任意のプログラムコードであり、不用意に再配置を行うと正しく動作しなくなる可能性がある。アプリケーションプログラムは可能な限り初期配置から動かさないようにしなければならない。

そこで本研究で新たに提案する手法では、アプリケーションプログラムの初期配置に、各パラメータの時間変動の影響が少ないノードを選ぶ。それにより、パラメータが変動した場合にも、アプリケーションプログラムの配置は固定したまま、通常の演算を再配置するだけで処理コストをある程度抑えることができる。

本論文では、2.節で分散ストリーム処理について説明し、3.節で本研究で扱う分散ストリーム処理環境とそれを管理するORINOCOシステムについて述べる。その後、4.節でアプリケーション配置最適化手法を提案する。さらに5.節で関連研究を紹介し、最後に6.節でまとめと今後の課題について述べる。

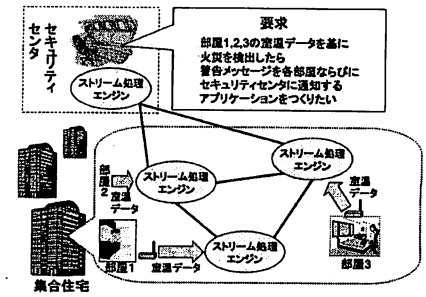


図1 分散ストリーム処理の要求例：集合住宅の遠隔監視

Fig. 1 Example of Distributed Stream Processing : Remote Monitoring of Cluster Housing

2. 分散ストリーム処理要求

分散ストリーム処理の要求例を、集合住宅の遠隔監視というシナリオで以下に紹介する。

図1はシナリオの概要図である。ここでは、集合住宅の各部屋に温度センサが取り付けられており、各々のセンサはストリーム処理エンジンに接続している。一方、集合住宅から地理的に離れた場所にストリーム処理エンジンを有するセキュリティセンタが存在する。このような環境において、セキュリティセンタ内の利用者が「部屋1,2,3の温度データを基に火災検知を行い、その結果を各部屋ならびにセキュリティセンタに知らせてほしい」という要求がある。この要求を実現するためには、分散ストリーム処理環境において以下のような処理が必要となる。

- 火災検知に必要な温度データを各ストリーム処理エンジンから収集する処理
- 収集したデータから火災検知を行う処理
- 火災検知の結果を各所に配信する処理

我々の研究グループでは、このような要求を実現する分散ストリーム処理環境管理システムORINOCOを開発中である。

3. 分散ストリーム処理環境

本研究で扱う分散ストリーム処理環境の概要を図2に示す。分散ストリーム処理環境の核となるストリーム処理エンジンには、我々の研究グループが開発しているStreamSpinnerを使用する。StreamSpinnerは各ノードに配置され、それぞれがストリーム情報源およびDBに接続している。StreamSpinnerは互いに処理結果を受け渡す連携機能を有しており、これにより分散環境を構築することができる。こうして構築された環境をORINOCOシステムが管理・運用する。ORINOCOシステムの目的は、分散ストリーム処理環境で利用者の要求した処理を環境への負荷を最小限にして実行させることである。利用者は処理を登録する場合、ORINOCOシステムにWorkerPocketプログラムを与える。WorkerPocketとは、分散ストリーム処理環境中で利用者の定義した処理を実行するアプリケーションである。ORINOCOシステムは、利用者が作成したWorkerPocketを最適

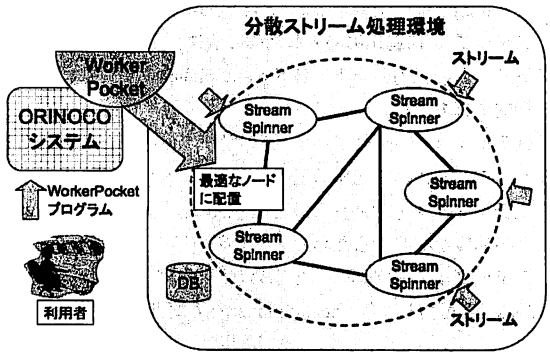


図2 分散ストリーム処理環境の概要図

Fig. 2 Proposed Distributed Stream Processing Environment

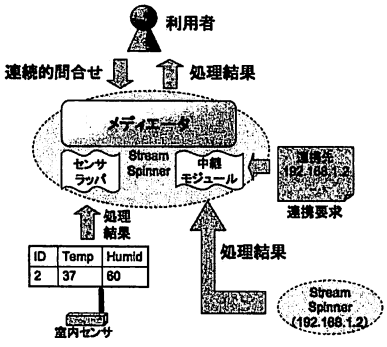


図3 StreamSpinner の処理

Fig. 3 Processing of StreamSpinner

なノードに配置するとともに、WorkerPocket にストリームデータが収集されるように各 StreamSpinner へ連携要求を送信する。

3.1 StreamSpinner

StreamSpinner のアーキテクチャを図3に示す。StreamSpinner は各情報源に対して固有のラッパというモジュールを持つ。ラッパは、各情報源が持つ固有のデータ形式をリレーションのタプル形式に変換する。こうすることにより、StreamSpinner は異種のストリームデータや DB のデータを統合することが可能となる。

利用者は、StreamSpinner に連続的問合せを定義することで処理結果を得ることができる。連続的問合せとは、ストリームデータ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。StreamSpinner は、リレーショナル代数における選択、射影、結合、直積演算を処理することができる。

StreamSpinner は、他の StreamSpinner の処理結果を受け取る中継モジュールを有している。中継モジュールに連携要求を与えることで連携が実現する。連携要求には、連携先の StreamSpinner の IP アドレスや配信データの指定情報などが記述される。

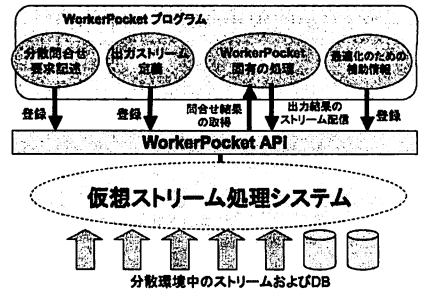


図4 API の設計コンセプト

Fig. 4 Design Concept of API

```
MASTER   Sensor1
SELECT   Sensor1.Temp, Sensor2.Temp, Sensor3.Temp
  FROM   Sensor1[1s], Sensor2[1s], Sensor3[1s]
 WHERE   Sensor1.Temp > 50 AND Sensor2.Temp > 50
         AND Sensor3.Temp > 50
```

図5 分散問合せ記述例

Fig. 5 Example of Query Description for Distributed Stream Processing

3.2 ORINOCO システム

StreamSpinner をベースに構築された分散ストリーム処理環境を管理・運用する ORINOCO システムについて述べる。

3.2.1 WorkerPocket 記述

ORINOCO システムは WorkerPocket プログラム記述のために、WorkerPocketAPI という Java のインタフェイスを用意している。WorkerPocketAPI を用いることで、利用者は分散ストリーム処理環境中で動作する WorkerPocket を記述することができる。

WorkerPocketAPI の概念図を図4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が実際にどのノードで実行されるかということを意識する必要がなくなるという利点がある。

WorkerPocket 記述における構成要素は次のとおりである。

- 処理に必要なストリームデータを収集する分散問合せ要求記述
- アプリケーション固有の処理
- 出力ストリームのスキーマ情報
- 問合せ最適化に使用する補助情報

利用者は WorkerPocket 記述において、分散ストリーム処理環境に存在する情報源に対しデータ収集を行う際、分散問合せ要求記述を使用する。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。

分散問合せ要求に対する処理方式は、StreamSpinner における問合せ処理方式と同様に、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

```
CREATE STREAM alarm ( message string )
```

図6 出力ストリーム定義の例
Fig. 6 Example of Output Stream Definition

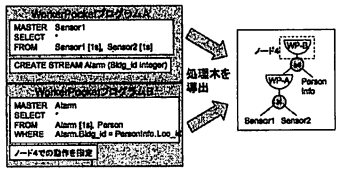


図7 処理木の導出例
Fig. 7 Example of Derivation of Processing Tree

てSQLライクな記述言語を提供する。例を図5に示す。図5の問合せの内容は、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。ウインドウサイズとは、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。図5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。

WorkerPocketでは、アプリケーション固有の処理はJavaで実装する。この部分には任意の処理を記述することが可能である。本稿ではプログラムコードは省略するが、詳しくは[8]を参照されたい。

WorkerPocket記述において、利用者はWorkerPocket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の例を図9に示す。出力ストリームの定義は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、その後に属性名と属性型のペアを指定する。図9の例では、string型のmessageという属性を持ったalarm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システムが問合せ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、WorkerPocket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、WorkerPocketの選択率の概算値である。これは、一タプルの入力に対しての平均出力タプル数のことである。

3.2.2 ORINOCOシステムの処理

WorkerPocketが与えられた際のORINOCOシステムの動作について説明する。

ステップ0: 最適化に必要な情報の収集、ORINOCOシステムは分:散環境中のネットワーク遅延、ストリームの入力レート、各StreamSpinnerの負荷情報などは事前に収集する。

ステップ1: 最適化に必要な情報の収集。ネットワーク遅延、ストリームの入力レート、各StreamSpinnerの負荷情報などを定期的に調査する。

ステップ2: WorketPocket記述の解析。分散問合せ記述を抽出し、WorkerPocketと演算を含む処理木に変換する。

ステップ3: 4節で述べる最適化手法により、WorketPocketおよび各演算を配置するためのノードを決定する。

ステップ4: 配置処理。上で求めた配置場所に基づいてWorkerPocketを配置し、各StreamSpinnerへの連携要求を送信する。

4. 最適化手法の提案

本節では、ORINOCOシステムにおける配置最適化手法について解説する。

4.1 目的とアプローチ

WorkerPocketは利用者が定義した処理を実行するアプリケーションであるため、一度配置すると動作ノードを変更させることは難しい。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、WorkerPocketを移動させなくても安定的な処理を実現するような配置プランを導出する。

本研究では、各ノード間のネットワーク遅延と、ストリーム型情報源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対して、WorkerPocketを動かさずに、演算の再配置のみでネットワーク転送コストを下げられるようなロバストな初期配置を決定することであるが、それ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既存手法を用いて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。

以下では4.2節でサンプルデータについて説明し、4.3節で初期配置を決めるための手順について述べる。4.4節では各サンプルデータに対する配置場所を求めるための既存手法について述べる。

4.2 状態サンプルデータ

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転送コスト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8中央の表にあるとおりである。$Lat_{m,n}$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。

4.3 初期配置プランの決定

提案手法の処理ステップを、図 8 の例を用いて解説する。

（1） 時刻 t_k におけるサンプルデータを取得する。サンプル取得間隔 $(t_{k+1} - t_k)$ は十分長いものとする。図 8 では $k = 100$ としている。

（2） WorkerPocket プログラムから処理木を導出する。例では、WorkerPocket-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、WorkerPocket-A、O_1、O_2 については初期配置を決める必要がある。

（3） k 個のサンプルデータそれぞれについて、4.4 節で解説する既存手法を適用し、各状態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 WorkerPocket 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。例では、WorkerPocket-A および演算 O_1, O_2 はノード 1 に配置することが決定する。

以上のような処理を行うことによって、WorkerPocket 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。

4.4 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、[9] で提案された手法を使用する。[9] の手法は、ネットワーク遅延情報と入力データレート情報を基に、ネットワーク転送コストを最小とする演算配置プランを導出する。このステップの処理においては、WorkerPocket も他の演算と同様に配置すべき対象として扱われる。ネットワーク遅延空間とは、d 個の適当なランドマークノードを選択し、全ノードから各ランドマークノードまでのネットワーク遅延を表現する d 次元の空間である。[9] の手法では、各演算の配置ノードを決定する際に式 1 のコスト関数を使用する。ここで L とは当該ノードと各接続先ノードとの間の回線の集合であり、回線 l において、$DR(l)$ は入力データレートを表し、$Lat(l)$ はネットワーク遅延を表す。

$$\sum_{l \in L} DR(l) \cdot Lat(l)^2 \qquad (1)$$

[9] 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 のコスト関数を用いてコスト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、各演算のコストの総和が閾値 F_t を下回ったらそのプランを出力する。F_t を上回ったら処理 (2) を再び実行する。

以上の手順により，各状態サンプルにおいて式 1 の値を最小化する配置場所が決定する.

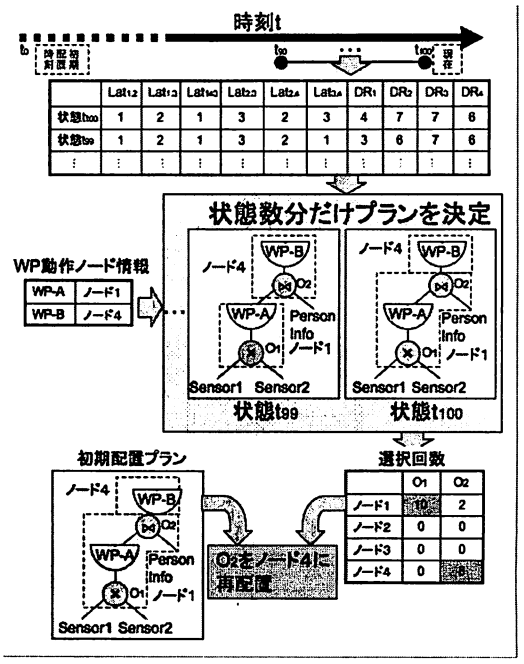


図 9 演算再配置の例

Fig. 9 Example of Operator Reallocation

4.5 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 WorkerPocket が配置され、StreamSpinner に連携要求が出される。前述の通り、WorkerPocket が稼動し始めた後は、各パラメータに変動が起こっても、WorkerPocket 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、変動に対するコストの調節を行う。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実行する。

（1） 分散ストリーム処理環境において、一定の時間間隔を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、WorkerPocket の動作ノード情報を基に、[9] の手法を用いて最適なプランを導出する。

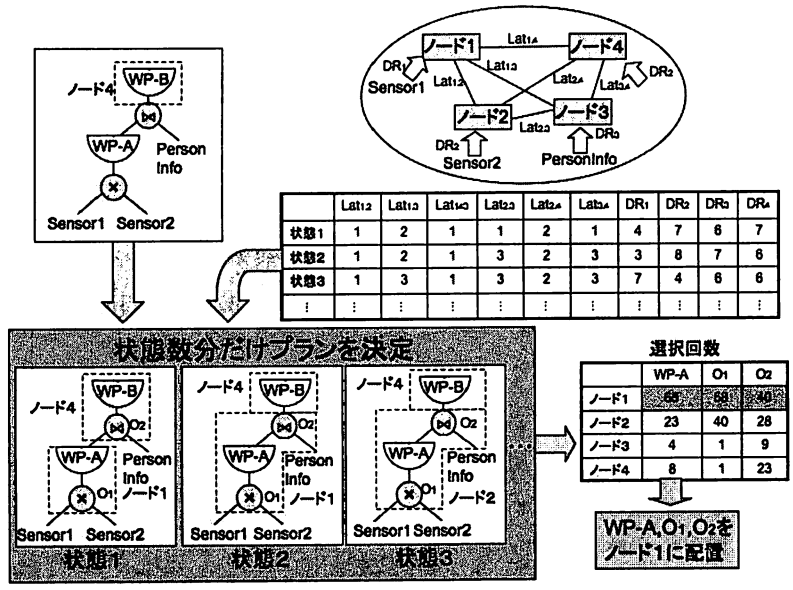
（3） i 個のプランから、各演算について配置ノードとなる回数が最も多かったノードを選択する。

（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。

以上の処理を行うことで、最近のパラメー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。

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



	Lat12	Lat13	Lat14	Lat23	Lat24	Lat34	DR1	DR2	DR3	DR4
状態1	1	2	1	1	2	1	4	7	6	7
状態2	1	2	1	3	2	3	3	8	7	6
状態3	1	3	1	3	2	3	7	4	6	6
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

図 8　初期配置プランの決定

Fig. 8　Decision of Initial Allocation Plan

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 [10] 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 [11] が提案されている。また、SBON [9] 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 [12]〜[14] があるが、これらは本研究とは目的が異なる。

6. おわりに

本研究では、分散ストリーム処理環境におけるアプリケーションの配置最適化手法を提案した。提案手法は、再配置が困難であるという WorkerPocket プログラムの性質を考慮し、ネットワーク遅延やストリームの入力レートなどのパラメータの時間変化が起きても処理効率の低下が発生しにくい場所へこれらを配置する。また、分散ストリーム処理環境を管理・運用する ORINOCO のデモシステムを構築した。

今後の課題としては、提案手法を ORINOCO システムに実装し、実験を通して手法の評価を行うことが挙げられる。

謝辞　本研究は、科学研究費補助基盤研究 (A)(#1820005), 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] P.Pietzuch, et al., "Network-Aware Operator Placement for Stream Processing Systems", Proc. ICDE, p.49, 2006.

[10] Y. Xing, et al., "Dynamic Load Distribution in the Borealis Stream Processor", Proc. ICDE, pp. 791–802, 2005.

[11] Y. Xing, et al., "Providing Resiliency to Load Variations in Distributed Stream Processing", Proc. VLDB, pp. 775–786, 2006.

[12] M. Balazinska, et al., "Fault-tolerance in the Borealis distributed stream processing system", Proc. ACM SIGMOD, pp.13–24, 2005.

[13] J. Hwang, et al., "High-Availability Algorithms for Distributed Stream Processing ", Proc. ICDE, pp. 779–790, 2005.

[14] 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.