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

稲守　孝之† 　渡辺　陽介†† 　北川　博之†,††† 　天笠　俊之†,†††

† 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}@cs.tsukuba.ac.jp

あらまし　近年、ネットワークの進歩やデバイスの発達により、時々刻々と配信されるストリームと呼ばれるデータが増大している。そのため、大量のストリームに対する問合せ処理を実現するシステムの需要が高まっており、特に複数のノードから構成される分散型ストリーム処理システムに注目が集まっている。分散型ストリーム処理システムでは、個々のノードが割り当てられた処理を実行して互いに処理結果を受け渡すことで、全体として要求を実現する。我々の研究グループでも、我々が開発したストリーム処理エンジン StreamSpinner をベースに、分散型ストリーム処理システムの構築を行っている。本研究では、分散型ストリーム処理環境における利用者の利便性を向上させるための運用管理システムを提案する。提案システムは、各ノードの監視や問合せ要求の登録などを行うだけでなく、分散型ストリーム処理環境を利用したアプリケーションの作成を容易にするための仕組みを提供する。アプリケーションプログラムが登録されると、提案システムはその処理内容に応じて適切なノードへアプリケーションプログラムを配置し、問合せ処理に必要なストリームデータがそのノードへ集められるように分散環境中の各ノードに連携要求を発行する。本稿では、これらの機能について述べる。

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

A Proposal of Management System for Distributed Stream Processing Environments

Takayuki INAMORI†, Yousuke WATANABE††, Hiroyuki KITAGAWA†,†††, and Toshiyuki AMAGASA†,†††

† 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}@cs.tsukuba.ac.jp

Abstract　Rapid developments of network and device technologies bring the increase of data streams which deliver up-to–date information to users. 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. The systems consist of multiple nodes, and these nodes fulfill user requirements by collaborating each other. Our research group has been developing a distributed stream processing environment based on our stream processing engine named StreamSpinner. This paper proposes a management system for distributed stream processing environments to improve usability of the environments. The proposed system assists users to monitor multiple nodes and to register queries. And, it also provides an API to develop application programs which use distributed stream processing environments. When a new application program is registered, our system finds an appropriate node to execute the program and allocates it to the node. After that, the system distributes collaboration requests which make other nodes forward stream data needed by the program.

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

1. まえがき

近年、ネットワークの進歩やデバイス技術の発達により、時々刻々と逐次配信されるストリームと呼ばれる情報が増大している。ストリームの例としては、ニュースやデータ放送など web 上で配信されるものや、センサデータやカメラからの映像といった実世界情報などが挙げられる。Web 上のデータや DB に蓄積されているデータは利用者が自らアクセスするものであるのに対し、ストリームは情報源から能動的に利用者に向けて配信されるものである。このような性質を持つストリームに対して、フィルタリングや他のストリーム情報源・DB との統合といった高度利用要求が高まっている。

そこでこれらの要求に応える機能を備えたストリーム処理エンジン [1]〜[3] の研究・開発が進んでいる。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 では、利用者はストリームに対してフィルタリングや他のストリーム・DB との統合といった要求を定義することができる。ストリームに対する処理要求は SQL ライクな問合せ言語で記述され、StreamSpinner はストリームデータが到着する度に問合せ処理を実行する。

従来のストリーム処理エンジンでは、複数のストリームを単一のマシンで処理するような集中型処理が多く見られた。しかし地理的に分散したストリーム情報源・DB を扱う際、集中型ストリーム処理ではマシンやネットワークに過度の負荷が集中し、パフォーマンスを低下させるという問題が生じる。また、一つのハードウェアの故障がシステムの停止に直結するという問題も存在する。そこで、これらの問題を解決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。分散ストリーム処理とは、地理的に分散したストリーム情報源や DB を効率的に扱うために、複数のノードを配置し、互いに処理結果を送受信させることで目的の要求に応える処理のことである。我々の研究グループでも StreamSpinner をベースに分散ストリーム処理環境の構築を行っている。

本研究では、分散ストリーム処理環境における運用管理システム「ORINOCO」を提案する。分散ストリーム処理環境を構築するストリーム処理エンジンには StreamSpinner を使用する。ORINOCO システムを利用することで、利用者は分散ストリーム処理環境上で動作するアプリケーションを作成することができる。ORINOCO は、分散ストリーム処理環境中のどのノードに処理を割当てるかを決定する分散問合せ最適化機能を有する。

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 StreamSpinner を用いた分散ストリーム処理環境について述べる。その後４節で提案システムについて述べ、５節で利用者へのインタフェイスとなる WorkerPocketAPI について説明する。６節で関連研究、７節でまとめと今後の課題について述べる。

2. 分散ストリーム処理要求

分散ストリーム処理は、地理的に分散したストリーム情報源・DB のデータを効率的に扱うために有効な処理である。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を用いて以下に紹介する。

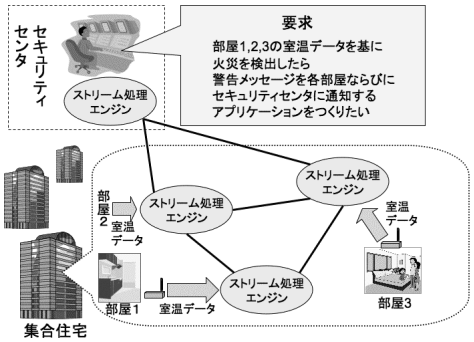

図 1　分散ストリーム処理の要求例：集合住宅の遠隔監視

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

図 1 に要求例の概要図を示す。この例では、集合住宅の部屋 1,2,3 に温度センサとストリーム処理エンジンが設置され、室温データを絶えず取得している。一方、地理的に離れた場所にセキュリティセンタが存在し、そこにもストリーム処理エンジンが設置されている。このような環境で、セキュリティセンタにいる管理者の要求として、「部屋 1,2,3 の室温データを基に火災を検出したら警告メッセージを各部屋ならびにセキュリティセンタに通知するアプリケーションを開発したい」というものが考えられる。本稿でのアプリケーションとは、複雑なアルゴリズムを処理するコンポーネントであり、この例では火災を検出するアルゴリズムなどである。

この要求を満たすためには、分散ストリーム処理環境において次の処理が必要となる。

- 火災検出に必要なデータを各ストリーム処理エンジンから収集する処理
- 室温データから火災検出を行うアプリケーション固有の処理
- アプリケーションから警告メッセージを配信する処理

本研究では上記の例を踏まえ、分散ストリーム処理環境の利便性を向上させるためには次のことが必要であると考えている。

- アプリケーションを開発する手段を提供すること
- 必要なストリームデータを容易に収集できること
- アプリケーションの処理結果を新たなストリームとして配信できること
- 分散ストリーム処理環境における個々のノードを意識する必要がないこと

本研究で提案する管理運用システム ORINOCO は、これらの機能を提供し、アプリケーション開発を容易にする。

3. 分散ストリーム処理環境

分散ストリーム処理環境を構成する最も重要なものはストリーム処理エンジンである。本研究では、ストリーム処理エンジンに我々の研究グループが開発している StreamSpinner を使用する。StreamSpinner を用いた分散ストリーム処理環境の概

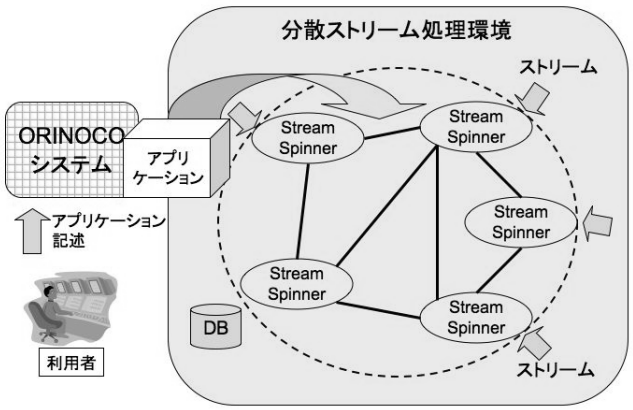

図 2　分散ストリーム処理環境の概要図

Fig. 2　Proposed Distributed Stream Processing Environment

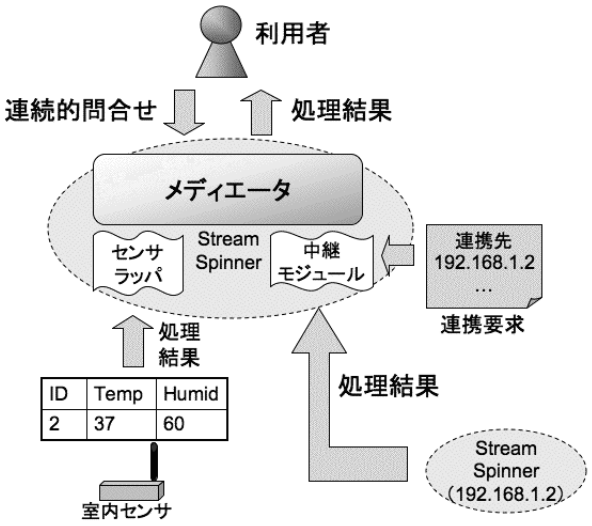

図 3　StreamSpinner の処理

Fig. 3　Processing of StreamSpinner

要図を図 2 に示す。StreamSpinner は、分散ストリーム処理環境中の各ノードに配置され、それぞれがストリーム情報源や DB に接続している。StreamSpinner については 3.1 で述べる。

この分散ストリーム処理環境を管理・運用するのが ORINOCO システムである。利用者は ORINOCO システムが提供する API を用いて分散ストリーム処理環境で動作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データ取得と処理実行に最適なノードに配置する機能を有している。アプリケーションは、各ノードから必要なデータを転送してもらい、送られてきたデータに対して任意の処理を行う。またアプリケーションは、処理結果を新たなストリームとして分散ストリーム環境中に配信する。

3.1　StreamSpinner

本研究における前提の技術である StreamSpinner について説明する。StreamSpinner での処理のアーキテクチャを図 3 に示す。

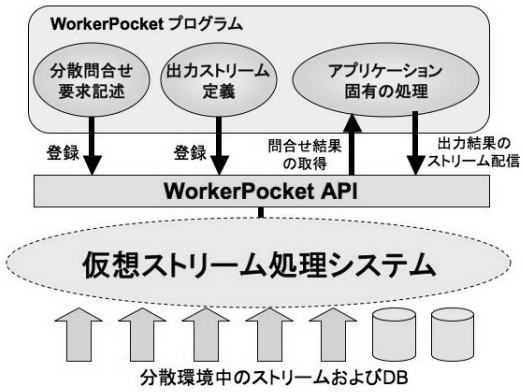

図 4　API の設計コンセプト

Fig. 4　Design Concept of API

StreamSpinner は各情報源に対してラッパというモジュールを持つ。ラッパの役割は各情報源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とである。StreamSpinner はすべてのデータをタプルの形式で表現することにより異種のストリームや DB との統合を実現している。

StreamSpinner には連続的問合せ要求を定義することができる。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。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 の処理結果を受信する中継モジュールを搭載している。StreamSpinner 同士を連携させるには、接続先の IP アドレスや転送してもらうデータの指定などの情報を含んだ連携要求を中継モジュールに渡す必要がある。

4.　分散ストリーム処理環境運用管理システム

本節では、StreamSpinner の連携機能を用いて構築した分散ストリーム処理環境を運用・管理するシステム「ORINOCO」の詳細について説明する。まず 4.1 で ORINOC が提供するプログラミングインタフェイスについて述べ、その後、4.2 で ORINOCO のシステムアーキテクチャを説明する。

4.1　アプリケーション記述

ORINOCO システムはアプリケーション記述のために、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アプリケーションを記述することができる。

WorkerPocketAPI の設計コンセプトの概念図を図 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。

アプリケーション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

```
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

```
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

- 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

利用者は、仮想ストリーム処理システムに対し、問合せ記述や出力ストリーム定義の登録を行い、また、取得した問合せ結果を基に実行される固有の処理を実装する。固有の処理は通常のプログラミング言語で記述し、問合せ結果が到着する度に呼び出される。分散問合せ要求記述については 4. 1. 1 で、出力ストリーム定義については 4. 1. 2 で述べる。アプリケーションの実装の具体的な説明は 5. 節で行う。以降では、WorkerPocketAPI を用いて作られたアプリケーションプログラムのことを単にアプリケーション、または WorkerPocket と呼ぶ。

4. 1. 1　分散問合せ要求記述

アプリケーション記述において、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。

4. 1. 2　出力ストリームのスキーマ記述

アプリケーション記述において、利用者はアプリケーション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 6 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 6 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。

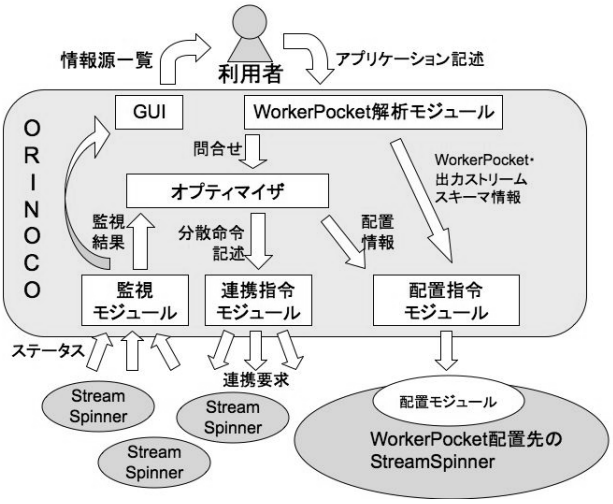

図 7　ORINOCO のアーキテクチャ
Fig. 7　Architecture of ORINOCO

4. 2　アーキテクチャ

ORINOCO のアーキテクチャを図 7 に示す。各モジュールの役割を以下に列挙する。

- WorkerPocket 解析モジュール

WorkerPocket 解析モジュールは、利用者から与えられたアプリケーション記述を解析し、WorkerPocket が利用するストリームデータを収集する分散問合せ要求記述と、WorkerPocket が出力するストリームのスキーマ情報を取得する。

- 監視モジュール

監視モジュールは、各ノード上の StreamSpinner が管理するストリーム情報源の情報と、ノードの負荷情報、ネットワーク帯域の負荷情報を監視する。

- オプティマイザ

オプティマイザは、監視モジュールから取得した監視情報と WorkerPocket 解析モジュールから取得した分散問合せ要求記述に基づいて、WorkerPocket を配置するノードを決定し、また問合せ処理に必要な演算の各ノードへの割当てプランを生成する。配置先と割当てプランの決定については 4. 3 で述べる。

- 連携指令モジュール

連携指令モジュールは、オプティマイザの決定に基づいて各 StreamSpinner に連携要求を発行し、WorkerPocket の処理に必要なストリームデータの収集を実現する。

- 配置指令モジュール

配置指令モジュールは、配置情報に基づいて WorkerPocket を目的のノードに配置するよう、配置先の StreamSpinner 内の配置モジュールに指令を出す。WorkerPocket の具体的な受渡しの仕組みについては 5. 節で説明する。

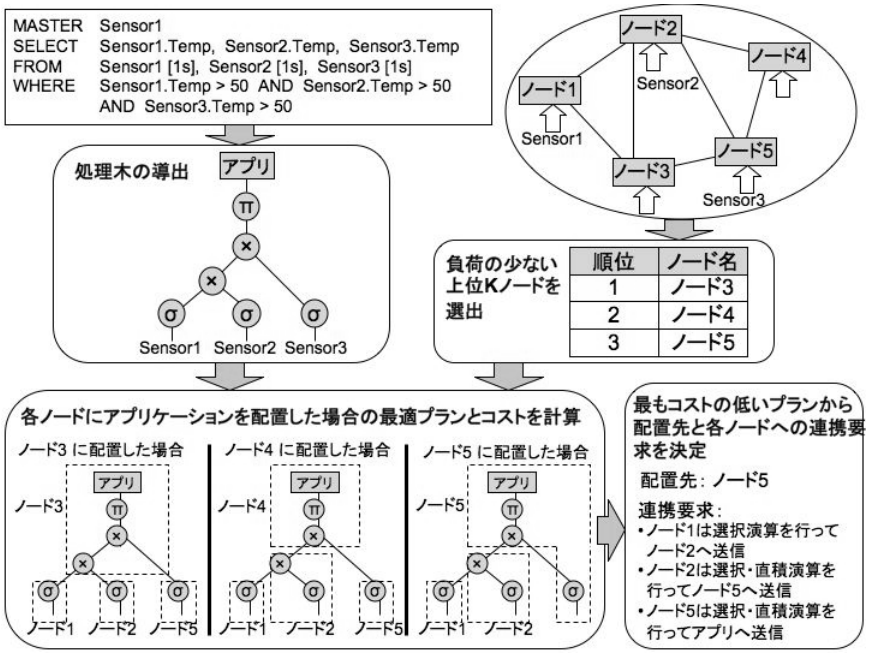

図 8　配置先と連携要求の決定

Fig. 8　Decision of WorkerPocket Allocation and Cooperation Requirement

- 配置モジュール

WorkerPocket 配置先の StreamSpinner に存在する配置モジュールは、WorkerPocket を当該ノードに配置し、StreamSpinner と接続させる。また、WorkerPocket の出力ストリームのスキーマ情報を基に、新たなストリームを配信するように StreamSpinner を設定する。

- GUI

GUI は、監視モジュールによって得られた情報源の一覧をスキーマ情報とともに利用者へ提供する。

4.3　配置先と連携要求の決定

ORINOCO システムは、利用者から与えられた分散問合せ要求に基づいて、WorkerPocket を配置するノードを決定し、また問合せ処理に必要なデータが転送されるように各ノードへ発行する連携要求を決定する。その際、特に考慮すべき事柄として次の３点が挙げられる。

- アプリケーション固有の処理は、通常の問合せ処理に比べて処理コストが高くなることが予想されるため、WorkerPocket は負荷の少ないノードに配置されることが望ましい。
- ネットワーク全体への負荷の観点から、WorkerPocket の配置されるノードへデータを送る転送コストは、小さいことが望ましい。
- 出力ストリームを利用する他のアプリケーションがある場合はその転送コストも考慮する必要がある。出力ストリームのレート情報の概算値は利用者から提供されることを想定している。

本研究では、これらの点を考慮して配置先と連携要求を決定する。手順を以下に示す。

（1） 問合せ記述を解析し、処理木を生成する。

（2） 各ノードにおける現在の負荷情報を収集し、最も負荷の軽いノード k 個を WorkerPocket 配置先の候補にする。

（3） k ノードそれぞれについて、配置した場合のネットワーク転送コストが最小となる演算の配置プランを導出する。なお、ネットワーク転送コストに基づく最適化は既存の手法 [8] を使用する。

（4） k 個の配置プランのうち最もコストの低いプランを選択し、アプリケーションの配置先の決定と各ノードへの連携要求の生成を行う。

図 8 は、図 5 の分散問合せ要求記述を含む WorkerPocket を、ノード１からノード５によって構成される分散ストリーム処理環境に配置する例である。ここでは、ノード３～５が候補として選ばれ、最終的にはノード５に配置するプランが選択されている。

特殊なケースとして、利用者が WorkerPocket が動作するノードを明示的に指定したいことも考えられる。その場合、ORINOCO システムは利用者が指定したノードを WorkerPocket の配置先として、連携要求の決定のみを行う。

5.　WorkerPocketAPI

WorkerPocketAPI を利用したアプリケーション記述では、Java のクラス「WorkerPocket」を継承したサブクラスを作成し、WorkerPocket クラスから引き継いだ abstract メソッドを実装する。以下は実装例を用いて説明する。

5.1　サンプルコード

図 9 のアプリケーションは、図 5 にある分散問合せによって各部屋の温度の異常値を取得する。その値を火災検出アルゴリズムにかけ、もし火災であると判別したなら警報メッセージを配信するというプログラムである。

10 行目からはじまる getQueries メソッドで、各部屋の温度

```
 1:Public class Example extends WorkerPocket
 2:{
 3:   // このWorkerPocket固有の名前を定義
 4:   public String getName(){
 5:      String name = "AlarmApp";
 6:      return name;
 7:   }
 8:
 9:   // 各部屋の危険域室温データを収集する問合せを登録
10:   public static String[] getQueries(){
11:      String[] queries = { "MASTER ... SELECT ... FROM ... WHERE ..."};
12:      return queries;
13:   }
14:
15:   // 出力ストリームを定義
16:   public static String[] getStreamDefinitions(){
17:      String[] stream_def =
18:         {"CREATE STREAM alarm ( message string )"};
19:      return stream_def;
20:   }
21:
22:   // 各タイミングで行う処理を実装
23:   public void init(){ 初期化処理... }
24:   public void start(){ アプリケーション開始時の処理... }
25:   public void init(){ アプリケーション終了時の処理... }
26:
27:   // データ到着時に実行されるメソッド
28:   public void dataDistributed( WPInputRowSet in_rs, int query_id ){
29:      // 入力ストリームデータを取得
30:      double temp1 = in_rs.getDouble( "Sensor1.Temp" );
31:      double temp2 = in_rs.getDouble( "Sensor2.Temp" );
32:      double temp3 = in_rs.getDouble( "Sensor3.Temp" );
33:
34:      // 火災判定
35:      boolean rc = detectFire( temp1, temp2, temp3 );
36:
37:      // 火災と判定されたら警報メッセージストリームを配信
38:      if( rc == true ){
39:         // 出力ストリームデータを作成
40:         WPOutputRowSet alarm_rs  = getWPOutputRowSet( "alarm" );
41:         alarm_rs.setString( 0, "[警報]火災が発生しました");
42:         // 配信
43:         alarm_rs.send();
44:      }
45:   }
46:   ...
47:}
```

図 9　WorkerPocket プログラムの実装例

Fig. 9　Example of WorkerPocket Program

の異常値を集める分散問合せ要求記述（図 5）を定義する。分散問合せ要求記述は複数記述することも可能である。複数記述した場合は、処理結果の通知時にクエリ ID によって識別される。16 行目から始まる getStreamDefinitions メソッドで、新たに配信するストリームのための CREATE STREAM 文（図 6 と同様）を定義する。23〜25 行目の init,start,stop メソッドでは、初期化時やアプリケーションの開始時と終了時に行う処理を実装する。28 行目からはじまる dataDistributed メソッドは、実際にデータが到着した際に呼び出されるメソッドである。30〜32 行目では、各部屋の異常温度データを変数 temp1〜3 に格納している。35 行目の detectFire メソッドは火災検出を行うこのアプリケーション固有の処理である。この detectFire メソッドの戻り値が true だったら（火災が検出されたと判別したら）、40〜43 行目で出力ストリームとして警報メッセージを作成しそれを配信する。

5.2　WorkerPocket の配置

WorkerPocketAPI の実装コードは、コンパイルされ Java の Class ファイル（WorkerPocket）となる。この Class ファイルをノードに配置するには Java の URLClassLoader クラスを使用する。URLClassLoader は、Class ファイルがある URL を与えると、そのファイルをロードして Java の実行環境で使用できるようにする機能を提供する。クラスファイルを受け取った後は、配置モジュールが WorkerPocket のインスタンスを生成する。

5.3　API に対する考察

提案システムが提供する API を用いた場合の利用者の利便性について考察する。利用者が API を用いずに分散問合せ処理と出力ストリームの配信を実現しようとした場合、次のような手順を踏まなければならない。分散問合せ処理を実現する場合、利用者は各ノードに対して問合せを発行し、また処理結果の受渡しを指定する連携要求を発行しなければならない。出力ストリームの配信を行う場合、利用者は StreamSpinner に直接データを渡すコードを記述しなければならない。API を用いなかった場合のこれらの作業は煩雑になることが多い。

それに対して提案 API を用いた場合は、仮想ストリーム処理システムに対する分散問合せ要求記述を与えるだけで必要なデータを収集することができ、出力ストリームを定義する場合にも StreamSpinner と直接通信するコードを記述する必要はない。これらのことから、提案システムにおける利用者の利便性の向上に対する貢献度は高いといえる。

6.　デモシステム

本研究で提案する ORINOCO システムのデモ実装を行った。デモの内容は図 1 の集合住宅の遠隔監視の内容に沿ったものとなっている。図 10 は、3 台のノードから構成される分散ストリーム処理環境を、ORINOCO システムが管理している様子を示している。各ノードでは温度センサからのストリームデータを毎秒提供している。ここで、利用者が図 9 のコードで作成した WorkerPocket をシステムに与えると、システムは分散問合せ要求から連携命令と配置先を決定し、それぞれ実行する。各ノードから 50 ℃より大きい温度データが到着すると、それらのデータを基に火災判定を行い、火災の場合には警告メッセージをストリームとして配信する。また、各ノードの連携状況は ORINOCO システムの GUI で利用者が確認できるようになっている (図 10 の左)。

7.　関 連 研 究

本研究の提案システムは、分散ストリーム処理環境においてストリームデータを扱うアプリケーションプログラムを実行することができる。グリッド・コンピューティング [9] における、Globus ツールキット [10] は分散環境の計算機資源を利用するためのミドルウェアであるが、ストリームデータに対する連続的な問合せ処理は考慮されていない。

分散データベースシステム [11] は、地理的に離れた情報源に対する問合せ処理を実現するためのものである。しかし、分散データベースシステムの問合せ処理は、問合せを実行して一度処理結果を生成すれば終了するが、分散ストリーム処理システムでは、ストリームとして送られてくるデータに対して、継続的な問合せ処理が行われる点が異なっている。

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、

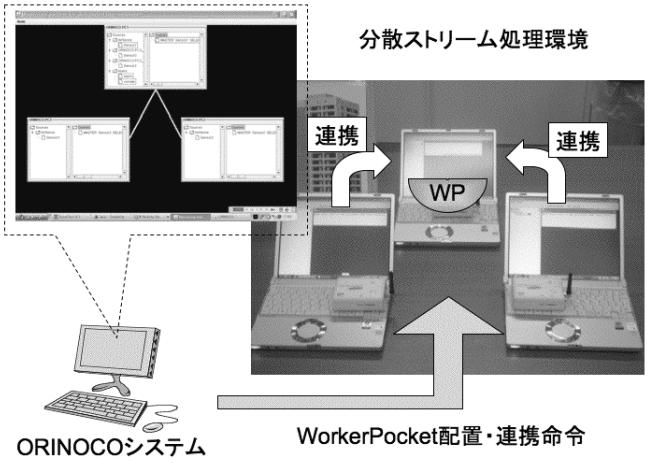

図 10　デモシステム
Fig. 10　Demonstration System

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]、STREAM [3]、TelegraphCQ [2] 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。演算の割り当て手法としては，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 [8] 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。これらの割り当て手法はすべて、ストリームデータの配送先であるアプリケーションが決まった場所で動作することを前提としている。本研究の提案したシステムは、演算だけでなく、アプリケーション自身の配置を行うことができるため、より柔軟な割り当てを考えることが可能となっている。

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。

8. あとがき

本研究では StreamSpinner を用いた分散ストリーム処理環境における運用管理システム ORINOCO の提案を行った。ORINOCO は、分散環境にアプリケーションを作成するフレームワークの提供と、その枠組みにおけるアプリケーションの配置と分散問合せ最適化を行う。現在、提案システムの実装作業を行っており、DEWS2007 においては実装システムのデモンストレーションを行う予定である。今後の課題としては、実装作業の完了後、分散問合せ最適化手法の提案を行い、提案システムの評価を行うことが挙げられる。評価する対象は、アプリケーション開発者が提案システムを使用することで軽減された作業量と分散問合せ最適化により改善された処理速度である。

謝辞　本研究は、科学研究費補助基盤研究 (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] P.Pietzuch, et al., ”Network-Aware Operator Placement for Stream Processing Systems”, Proc. ICDE, p.49, 2006.
[9] Open Grid Forum. http://www.ogf.org/
[10] The Globus Alliance. http://www.globus.org/
[11] M. T. Ozsu, et al., “Principles of Distributed Database Systems”, Prentice-Hall 1999.
[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.