分散ストリーム処理環境における持続型問合せ処理方式

A Sustainable Query Processing Method in Distributed Stream Environment

渡辺 陽介♣ 北川 博之♦

Yousuke WATANABE Hiroyuki KITAGAWA

センサーなどから得られるストリームデータの増加により，それらに対する継続的な監視要求が高まっている．監視要求を実現するシステムでは，長期に渡って到着データの監視処理を続ける必要があり，障害によるシステム停止は極めて影響が大きい．我々の研究グループでは，高信頼なストリーム処理実現のため，仮想マシン技術を組合せた持続型ストリーム処理環境の構築を行っている．本環境では，仮想マシン上で問合せ処理を行い，仮想マシンの実行状態を定期的に別のマシンへ配布することで，１台が停止した場合にも，別の場所で処理を継続することが可能である．本稿では，持続型ストリーム処理環境の仕組みについて述べる．

A demand for monitoring stream data obtained from sensors is increasing. The systems that fulfill such requirements must continuously monitor data from information sources in long term, thus system failures bring serious damages to them. We have been developing a sustainable stream processing environment combining our stream processing engine and virtual machine technology. In our framework, a node executes our stream processing engine within a virtual machine. The node periodically saves a snapshot of the running virtual machine, and distributes the snapshot to other nodes. When the node crashes, another node recovers the virtual machine from the snapshot and takes over query processing. This paper shows the architecture of our system.

1. はじめに

センサーデータや計算機のログデータなど，時間とともに絶えず変化する情報を扱うストリーム型情報源が増加し，それらに対する継続的な監視等の処理要求が高まっている．例えば「温度センサーからのデータが一定値以上であったら通知してほしい」は典型的な要求のひとつである．このようなストリームデータに対する処理要求を扱う基盤システムとして，ストリーム処理エンジン[1],[4],[7],[12]が注目されている．ストリーム処理エンジンでは，連続的問合せ処理方式[2]を用いており，利用者から登録された問合せは，ストリーム処理エンジン上で実行され続け，随時処理結果を出力する．長期に渡ってストリームデータに対する監視を行うには，ストリーム処理エンジンが途中で停止することなく，安定動作し続けることが必要となる．そのため，ストリーム処理エンジンの高信頼化は非常に重要な課題となっている．

現在，いくつかのストリーム処理の高信頼化方式[3],[5],[6],[8]が提案されている．これらの基本的なアイデアは，分散環境中の複数のマシンそれぞれでストリーム処理エンジンを用意し，1台が故障した場合には別のストリーム処理エンジンが問合せ処理を引き継ぐというものである．ただし，これらのアプローチが対象としているのは，ストリーム処理エンジンだけで処理が実現され，ストリーム処理エンジン間で内部状態の引き継ぎが可能な場合のみである．

一般に，ストリーム処理エンジンが提供するデータ操作機能は，リレーショナル代数演算など基本的なものだけで，アプリケーションごとの固有の処理はストリーム処理エンジンの外側に別のプログラムとして実装するか，ユーザ定義関数などの拡張機能を用いてストリーム処理エンジンに組み込むことになる．障害時には外部プログラムの状態まで含めて別のノードへ引き継がねばならない．だが，ストリーム処理エンジン自身に任意の外部プログラムの状態を操作させることは一般に困難である．そのため，このような場合は既存のアプローチだけでは十分な障害対応ができない．

そこで，外部プログラムとの連携も考慮したアプローチとして，本研究では仮想マシン技術を用いた高信頼化方式を提案する．本方式では，仮想マシン上でストリーム処理エンジンと外部プログラムを実行する．そして，実行中の仮想マシンに対して，仮想マシンの状態スナップショットを定期的に作成する．スナップショットをネットワーク上の別のノードへ配布しておくことにより，障害発生時にも別のノードがスナップショットを元に仮想マシンを再開し，問合せ処理およびプログラム実行を続けることができる．本稿では提案方式を実装した持続型ストリーム処理環境[10]について述べる．

2. 関連研究

Hwang[5]らは，ストリーム処理の高信頼化手法をPassive Standby，Upstream Backup，Active Standbyの3種類に分類している（図1）．以下でそれぞれの特徴について説明する．ここでは，メインで問合せ処理を実行するストリーム処理エンジンをプライマリ，プライマリの故障時に問合せ処理を引き継ぐストリーム処理エンジンをセカンダリと呼ぶ．

Passive Standby方式[5],[6]は，プライマリが通常の問合せ処理を行う合間に，プライマリの内部状態を定期的にセカンダリへコピーする方式である．プライマリが正常に動作している間は，セカンダリは問合せ処理を行わず，プライマリの生存確認を定期的に行う．プライマリに障害が発生した場合には，コピーされた時点の内部状態でセカンダリが問合せ処理を再開する．コピー後に新規に届いたデータは欠落している可能性があるため，セカンダリが処理を再開したときに上流から再送してもらう必要がある．

Upstream Backup方式[5]では，セカンダリがプライマリの生存確認を行う点はPassive Standby方式と同じだが，こちらは内部状態のコピーを行わない．プライマリに障害が発生した場合には，データを持たない初期状態からセカンダリが問合せ処理を開始する．データについては上流にバックアップされているものを再送してもらうことによって復旧する．

Active Standby方式[3],[5],[8]では，セカンダリもプライマリと同様の問合せ処理を冗長に行う．上流からのデータをセカンダリでもプライマリと同様に受け取る．処理結果の

♣ 正会員 科学技術振興機構戦略的創造研究推進事業
watanabe@kde.cs.tsukuba.ac.jp
♦ 正会員 筑波大学システム情報工学研究科
kitagawa@cs.tsukuba.ac.jp

出力はプライマリだけが担当するが，プライマリに障害が発生した場合には，セカンダリが処理結果を代わりに送信する．

本研究の提案方式は，状態を定期的にコピーするという点においては，Passive Standby方式の一種と分類することもできる．ただし，我々の手法は外部プログラムを含むマシン環境全体の状態を回復できる点が異なる．

また，外部プログラムを含む高信頼化方式として，Active standby方式において外部プログラム自身も二重化することが考えうる．この方式の問題点は，障害回復後にプライマリを復帰させて再び冗長構成をとることができない点である．プライマリとセカンダリで状態同期が必要になったとき，外部プログラム間に状態同期の手段がなければ，完全な復帰が実現できない．提案手法では，仮想マシンのスナップショットがあれば，すぐに処理を引き継げる状態に復帰できる．

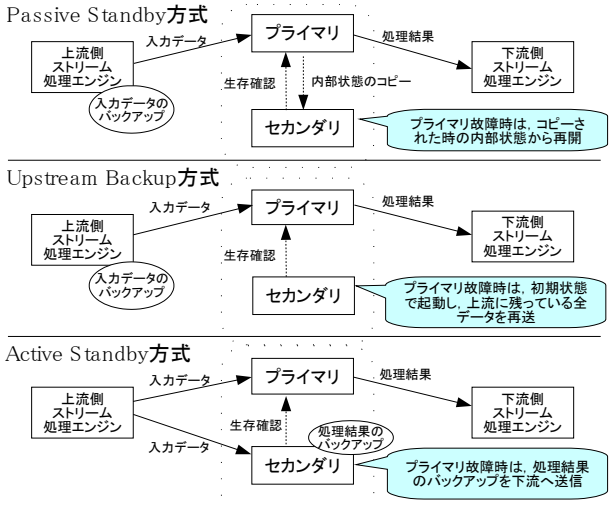

図1　既存の高信頼化手法

Fig.1　Existing High-Availability Approaches

3. 持続型ストリーム処理環境

本環境は，ストリーム処理エンジン StreamSpinner[12]と，仮想マシン管理を行うサステーナブルツールキット[9]から構成される(図 2)．まず StreamSpinner について述べ，次いでサステーナブルツールキットについて述べる．

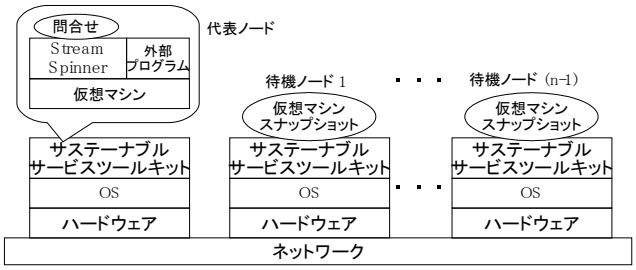

図 2　持続型ストリーム処理環境

Fig.2　Sustainable Stream Processing Environment

3.1　StreamSpinner

StreamSpinner は我々が開発したストリーム処理エンジンである．マシン 1 台でも動作するが，分散環境において複数のノード同士を連携させることが可能である．

1 ノード上の StreamSpinner の基本構成は，ラッパー，メディエータ，問合せ解析器，配信モジュール，中継モジュール，API モジュールである（図 3）．ストリーム型情報源からのデータはラッパーを介して取得される．ラッパーは新規データの到着を検知し，内部データ形式であるリレーションのタプルに変換，データ到着をイベントとしてメディエータに通知する．メディエータでは，イベントに応じて事前に登録された問合せの演算評価を実行する．メディエータにおける演算評価は連続的問合せ処理方式[2]に基づいており，前回評価後に到着した新規データから新たに生成可能な処理結果を差分的に生成していく．利用者は問合せ要求を SQL ライクな問合せ記述言語を用いて与えることができる．本研究では，射影，選択，直積，結合の各演算のほか，外部関数呼び出しを含む問合せを対象とする．メディエータで生成された処理結果は，配信モジュールによってアプリケーションプログラムや別のノードの StreamSpinner へ配信される．中継モジュールは，別の StreamSpinner で生成された処理結果を受け取るためのモジュールである．

StreamSpinner において外部プログラムを利用する方法は以下の 2 種類がある．

- アプリケーションプログラム：StreamSpinner は Java によるアプリケーション開発ができるようになっている．外部プログラムは API モジュールを用いて StreamSpinner へ問合せを登録し，生成されてくる処理結果を受信して，アプリケーション固有の処理を行う．
- 外部関数：メディエータ内部での演算評価において，外部の Java プログラムを直接呼び出すことが可能である．ただし，プログラム名と関数名，引数の型，戻り値の型などの情報は事前に登録しておく必要がある．問合せ中に関数名が記述されていた場合，メディエータは入力タプルから必要な属性値を抽出し，プログラムに渡す．プログラムの返した結果は，新たな属性値としてタプルに付加される．

これらの外部プログラムは，StreamSpinner と接続する部分に関しては規定の枠組みに従わなければならないが，プログラム内は任意の処理が記述可能である．

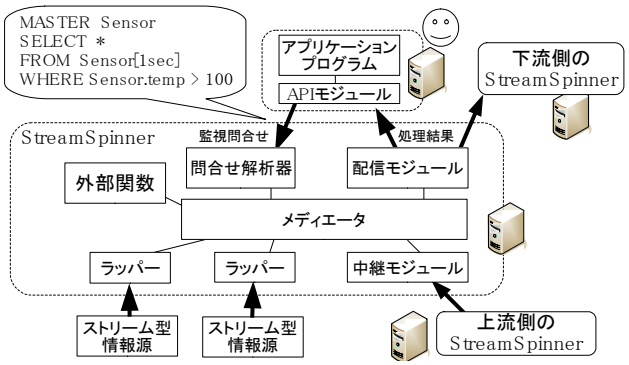

図 3　StreamSpinner のアーキテクチャ

Fig.3　Architecture of StreamSpinner

3.2　サステーナブルサービスツールキット

サステーナブルサービスツールキット[9]は，筑波大学 OSSS 研究室が開発した，分散環境中でソフトウェアサービスを持続的に動作させるためのソフトウェア群である．仮想マシン実行環境である ScrapBook for User-Mode Linux (SBUML)[11]と，P2P ネットワーク上の分散ストレージなどによって構成されている．SBUML は Linux マシン上でゲス

トOSとしてLinuxを起動することができ，起動中の仮想マシンに対して，主記憶上のデータを含む全実行状態をスナップショットとして保存することができる．また，スナップショットデータから，作成時点の状態で仮想マシンの実行を再開する事が可能である．

以下にサステーナブルサービスツールキットによる，StreamSpinner実行の流れを示す（図4）．

1. 仮想マシンの実行：サステーナブルサービスツールキットの動作するノード群は，決められた優先順位に基づいて代表ノードを一つ選出する．代表ノードがSBUMLを用いて仮想マシンを起動する．仮想マシン上ではStreamSpinnerや外部プログラムの処理が行われる．代表にならなかったノードは待機ノードとして代表ノードの生存確認を行う．
2. スナップショットの共有：サステーナブルサービスツールキットはSBUMLの機能を用いて，定期的に実行中の仮想マシンを一時停止し，スナップショットを作成する．作成されたスナップショットは，複数の待機ノードから構成される分散ストレージ上に置かれ，共有される．
3. 障害の検知：待機ノードは代表ノードに対して，定期的な生存確認メッセージを送信する．待機ノードは，メッセージの応答を一定時間待ち受け，時間切れによって代表ノードの障害発生を検知する．
4. 仮想マシンの復旧：障害が検知された時には，優先順位に基づいて待機ノードの1つが次期代表ノードとなり，分散ストレージ上のスナップショットを収集し，スナップショット作成時点の状態から仮想マシンを再開する．

仮想マシンの復旧によって，StreamSpinnerや外部プログラムの実行が継続される．ただし，内部状態はスナップショットを作成した時点に戻ってしまう．そのため，スナップショット作成後から仮想マシン復旧直前までの間に届いたデータに対して，次節で述べるようなデータ回復処理を行う．

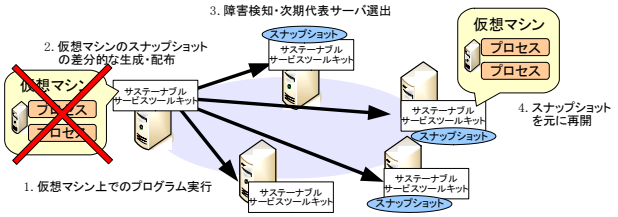

図4　サステーナブルサービスツールキット

Fig.4　Sustainable Service Toolkit

3.3　データ回復処理

以下では2つのStreamSpinner間でデータを送受信する例を用いて説明する．ここでは，ストリームデータを提供するStreamSpinnerを送信側と呼び，サステーナブルサービスツールキットの仮想マシン上で実行され，実際の問合せ処理を行うStreamSpinnerを受信側と呼ぶ．

1. 送信側は送信データに対してシーケンス番号を付加し，受信側に渡す．送信側は，データの送信が正常に完了してもデータを廃棄せず，自身の管理するバッファにシーケンス番号とともに保存する．バッファに格納されたデータは，再送要求を処理するために利用される．受信側ではデータのシーケンス番号を確認し，前回に受信したシーケンス番号と比較する．正常に動作している場合は，前回の番号に1を加えた値となる．前回の番号以下のデータが届いた場合は，重複データとして破棄する．
2. 受信側では，サステーナブルサービスツールキットによって定期的に仮想マシンのスナップショットが保存される．待機ノードへスナップショットの配布が完了するとStreamSpinnerへ通知が行われる．通知をうけた受信側StreamSpinnerは，その時点の最新シーケンス番号を送信側へ報告する．
3. 送信側では，報告されたシーケンス番号よりも古いデータをバッファから削除する．これは，スナップショット作成時点以前のデータはスナップショットに含まれており，再送の必要がなくなるためである．
4. 障害が発生し，仮想マシンが別のノードで再開されると，サステーナブルサービスツールキットは仮想マシン上の受信側StreamSpinnerに仮想マシンの再開を通知する．このとき受信側StreamSpinnerは，自分の中の最新のシーケンス番号を送信側へ送り，データの再送要求を出す．送信側では，バッファの中から，要求された番号よりも新しいデータを再送する．受信側は，すべての再送データがそろった段階で，データの到着順番を考慮しながら問合せ処理を再開する．

上記のデータ回復の仕組みは，仮想マシンのスナップショット作成や復旧に連動している点以外は，Passive Standby方式に基づいている．従って，提案手法はPassive Standbyと同等のデータ整合性維持の能力を持つ．

4. 評価実験

4.1　実験環境

本研究で行った評価実験の実験環境について述べる．今回は図5のようにマシンを5台用いた．5台のうち，node1はデータ送信，node2,3はサステーナブルサービスツールキットの実行，node4は処理結果受信を行う．node5は各処理時間を計測する．ストリームデータはLong型の属性を4つ含むような擬似ストリームを作り，node1から送信する．データの入力レートは自由に調節可能である．評価に用いた問合せは4種類で，ストリームデータをそのまま出力する問合せ，選択演算を含む問合せ，射影演算を行う問合せ，選択演算と射影演算を含む問合せである．

node1～node5

CPU	Xeon 3.6GHz
メモリ	2GB
HDD	32GB
LAN	1000BaseT
OS	Linux (2.6.18)
Java	JDK-1.5.0_10

仮想マシン

メモリ	96MB
HDD	340MB(89%使用)
OS	Linux (2.4.24)
Java	JDK-1.5.0_11

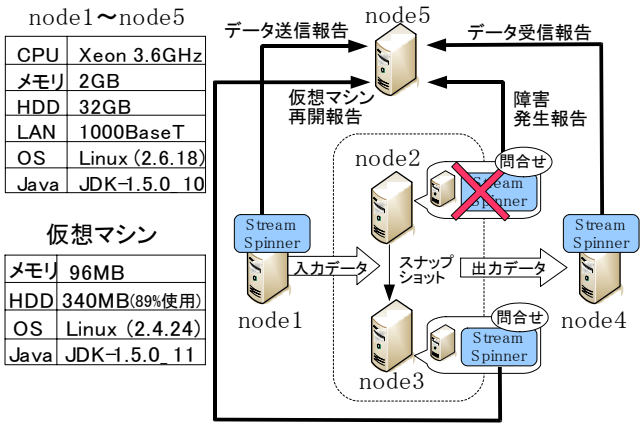

図5　実験環境

Fig.5　Experiment Environment

4.2　実験：仮想マシンのスナップショット作成

最初に，仮想マシンのスナップショット作成について実験

を行った. まず, 仮想マシン上で StreamSpinner を起動し, その状態で仮想マシンのスナップショットを作成する. 仮想マシンの割り当てメモリサイズ 96MB, HDD340MB に対し, スナップショットを 10 回作成したときの平均スナップショットサイズは 78MB 程度であった. サステーナブルサービスツールキットでは, これにさらに ZIP 圧縮を行った状態で配布を行う. ZIP 圧縮後のサイズは 28MB であった. さらに, 実行中の仮想マシンのスナップショットデータを保存するために必要な時間を測定した. 10 回スナップショットを作成して計測したところ, 平均 895 ミリ秒であった.

4.3 実験：回復時間

次に, 障害発生時点から仮想マシンを復旧するまでの時間と, そこからさらに StreamSpinner によるデータ回復処理が完了するまでの時間を調査した. node2 のサステーナブルサービスツールキットで StreamSpinner を起動し, node1 がデータを送信し始めてから約 40 秒後に, 強制終了コマンドを送信する. 終了コマンド送信から, node3 において仮想マシンが復旧されるまでの時間と, StreamSpinner が通常の処理に戻るまでの時間を 10 回測定した.

入力レートを毎秒 100 タプルと 150 タプルの 2 種類に設定し, 4 種類の問合せ（演算なし, 射影, 選択, 選択＋射影）を実行した場合の結果を図 6 に示す. グラフの横軸は入力レート（タプル/秒）および問合せ, 縦軸は障害発生からの経過時間(ミリ秒)である. 仮想マシンの復旧まではおよそ 7.8～8.0 秒程度, 仮想マシン復旧から StreamSpinner の回復までは 13～17 秒程度であった. 今回の実験では, 演算や入力レートの違いによる回復時間への影響は特に見られなかった.

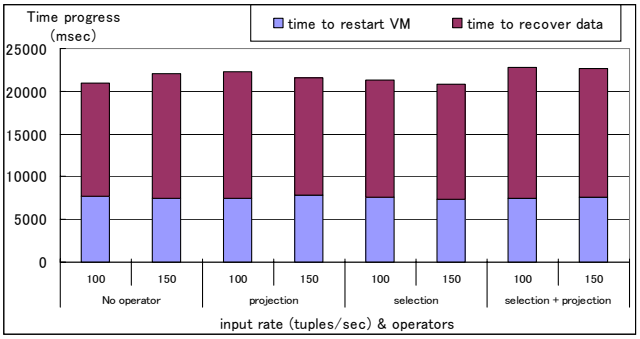

図 6 回復時間

Fig.6 Recovery Time

5. まとめと今後の課題

本稿では, 仮想マシンを用いた高信頼手法を実現する持続型ストリーム処理環境について述べた. 評価の結果, 十分実用になる時間で, 仮想マシンと問合せ処理を他のマシン上で再開させられることを示した.

今後の課題としては, 広域分散環境への対応があげられる. 現在はクラスタ環境などを対象としているが, 広域ネットワークではネットワーク分断による部分的なデータ不到着などが頻繁に発生する. そのような場合におけるデータ回復処理機能等の実装は今後行っていく予定である.

[謝辞]

本研究の一部は, 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. また, サステーナブルシステムおよび実験環境を提供していただいた筑波大学システム情報工学研究科 OSSS 研究室の皆様に感謝する.

[文献]

[1] D. J. Abadi, et al. “The Design of the Borealis Stream Processing Engine,” Proc. CIDR, pp.277--289, 2005.
[2] A. Arasu, et al. “The CQL Continuous Query Language: Semantic Foundations and Query Execution,” VLDB Journal, vol. 15, no. 2, 2006.
[3] M. Balazinska, et al. “Fault-tolerance in the Borealis distributed stream processing system,” Proc. ACM SIGMOD, pp.13--24, 2005.
[4] S. Chandrasekaran, et al. “TelegraphCQ: Continuous Dataflow Processing for an Uncertain World,” Proc. CIDR 2003.
[5] J. Hwang, et al. “High-Availability Algorithms for Distributed Stream Processing,” Proc. ICDE, pp. 779--790, 2005.
[6] J. Hwang, et al. “A Cooperative, Self-Configuring High-Availability Solution for Stream Processing,” Proc. ICDE, pp.176--185, 2007.
[7] R. Motwani, et al. “Query Processing, Resource Management, and Approximation in a Data Stream Management System,” Proc. CIDR, 2003.
[8] M. A. Shah, et al. “Highly-Available, Fault-Tolerant, Parallel Dataflows,” Proc. ACM SIGMOD, pp.827--838, 2004.
[9] 小磯知之, 阿部洋丈, 鈴木与範, Richard Potter, 池嶋俊, 加藤和彦. ”サステーナブルサービスを実現する基盤ソフトウェアの設計. ” 先進的計算基盤システムシンポジウム(SACSIS), 2006 年.
[10] 渡辺陽介, 北川博之. “仮想マシン技術を用いた持続型ストリーム処理環境の評価” 電子情報通信学会技術研究報告 Vol. 107, No. 131, pp. 339-344.
[11] ScrapBook for User-Mode Linux. http://sbuml.sourceforge.net/
[12] StreamSpinner. http://www.streamspinner.org/

渡辺 陽介 Yousuke WATANABE

2001 筑波大・第三学群情報学類卒. 2006 同大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. 情報処理学会, 日本データベース学会, ACM 各会員.

北川 博之 Hiroyuki KITAGAWA

1978 年東京大学理学部物理学科卒業. 1980 年同大学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, センサーデータベース, 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, 情報処理学会フェロー, 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.