2006－DBS－140（Ⅱ）（64）
2006／7／13

分散環境におけるストリーム処理の高信頼化

渡辺 陽介[†]，山田 真一[††]，北川 博之[††,†††]

概要

センサーデータやログ情報など，時間の経過に伴って変化する情報を提供し続ける，データストリームと呼ばれる情報源が増加し，データストリームに対する問合せ処理の需要が高まっている．現在，そのための基盤環境としてデータストリーム処理エンジンに注目が集まっている．データストリームのアプリケーションには，計算機の故障等による問合せ処理の停止やデータの消失が重大な問題につながるものも多い．そのため，データストリーム処理エンジンには長期に渡って安定したサービスを継続できる高い信頼性が求められる．本研究では，我々の研究グループがこれまで開発を行ってきたデータストリーム処理エンジン StreamSpinner をベースに，分散環境において信頼性の高い問合せ処理サービスを提供する，持続型データストリーム処理環境を実現する．提案システムでは，信頼性を高めるためのアプローチとして，既存の計算機資源仮想化技術を用いる．StreamSpinner を仮想マシン上で実行し，定期的に実行状態を保存・配布することにより，そのノードで障害が発生した場合にも，別のノード上で障害発生前の状態から問合せ処理サービスを再開させることが可能である．本稿では提案システムのアーキテクチャについて述べる．

Dependable Data Stream Processing in Distributed Environment

Yousuke Watanabe[†], Shinichi Yamada[††], Hiroyuki Kitagawa[††,†††]

Abstract

Data streams are information sources that continuously provide information changing over time. Examples of data streams are sensor data, log data and so on. Since demands for query processing over data streams are increasing, data stream processing engines become more and more important. For many stream applications, suspension of query processing and data loss caused by node failures and network failures are quite serious problems. Therefore, high availability of data stream processing engines is required. We propose a sustainable data stream processing system in distributed environments, based on a data stream processing engine StreamSpinner which we have developed. Our approach is based on collaboration of data stream processing engine and virtual machine technologies. StreamSpinner is executed on a virtual machine in a node, and the node periodically saves a snapshot of the running virtual machine. When the node crashes, other node recovers the virtual machine from the snapshot, and an alternative node continues query processing. This paper shows an architecture of our system.

1　まえがき

時間とともに変化する情報を絶えず提供する，データストリームと呼ばれる情報源の取り扱いは，その数と種類の増加に伴い年々重要になってきている．データストリームの例としては，オンラインで提供されるニュースや株価情報，実世界から得られるセンサーデータ，計算機上のサービスから得られるログ情報などがある．現在，我々の身近にはこれらストリームから得られるデータが溢れるようになっており，データストリームに対する問合せ処理への需要が高まっている．このような背景から，データストリーム処理エンジン [1, 4, 6] に関する研究が数多く行われている．

データストリーム処理エンジンでは，データストリームの性質に合わせた処理の必要性から，従来のデータベースシステムとは異なる仕組みによって実現されている．

- まず第 1 に，新しいデータが外部の情報源から次々と送られてくるため，到着したデータに対して繰り返し処理を適用していく必要が

† 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
†† 筑波大学システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba
††† 筑波大学計算科学研究センター
Center for Computational Scienses, University of Tsukuba

ある．このような処理の方式は連続的問合せ (Continuous Query)[2] と呼ばれており，利用者は問合せを事前に登録し，データの到着に応じて生成される処理結果を随時受け取る．一度登録された連続的問合せは，データストリーム処理エンジン上で長期に渡って実行され，処理結果を出し続ける．安定して処理結果を得るためにはシステムの信頼性が特に重要である．

- 第 2 に，高頻度で次々と到着する大量のデータを扱うため，データストリーム処理エンジンはメモリ上でのデータ処理に重点が置かれている．例えば，問合せ中でウインドウ指定を与えて処理対象データを制限することが可能である等，限られたメモリ内でディスクを介さずに問合せ処理を遂行するための機能が備わっている．これは，ディスクへのアクセスが前提であった従来のデータベースシステムとは異なっている．

- 第 3 に，地理的に離れた場所から提供されるデータストリームを扱う場合には，ネットワーク上に分散した複数の計算機資源やネットワーク帯域・遅延等を考慮し，それらを有効に活用する必要がある．そのため近年では，分散データストリーム処理エンジン [1] の研究が盛んに行われている．このような分散環境では，部分的なノード障害やネットワーク障害が発生するため，それらへの対処が重要となっている．

データストリーム処理の持つこれらの特徴は，従来のデータベースとは異なる新しいシステム高信頼化技術の確立を要求している．本研究では，分散環境において高い信頼性をもったデータストリーム処理を実現するための，持続型データストリーム処理環境を構築する．より具体的な目標は以下のようになる．

- **問合せ処理の継続**: 部分的なノード障害やネットワーク障害に対しても問合せ処理を停止することなく，継続的に動作するシステムを目指す．例えば，あるノード上で障害が発生して問合せ処理が継続できなくなった場合にも，別のノードが問合せ処理サービスを引き継ぐことにより，利用者に障害を意識させずに処理結果を提供し続ける．

- **障害によるデータへの影響の隠蔽**: 障害発生によって，利用者へ送信する前のデータを消失したり，また，障害回復後に重複したデータを送ることがないよう，データの整合性を維持するための機能を提供する．

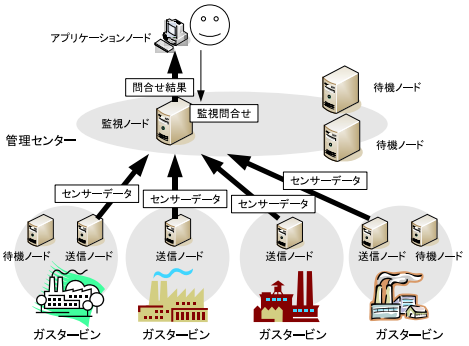

図 1: ガスタービン監視アプリケーション

高信頼化実現のアプローチとして本研究では，計算機資源の仮想化技術と分散データストリーム処理エンジンを連携させたシステムアーキテクチャを用いる．仮想化技術としてはサステーナブルサービスツールキット [9] を，分散ストリーム処理エンジンとしては我々の研究グループの StreamSpinner[11] を拡張したものを使用する．提案システムは，各ノードで仮想マシンを起動し，仮想マシン上でデータストリーム処理エンジンを走らせる．そして，仮想マシンのスナップショットを定期的にネットワーク上の別のノードへ配布することで，障害発生時にも別のノードが仮想マシンの実行を引き継ぎ，問合せ処理を続行することを可能とする．また，複数ノード上で動作するデータストリーム処理エンジン同士で，受信確認メッセージの交換とデータの再送を行うことで障害によるストリームデータの消失や重複を防ぐ．本稿では，これらの仕組みについて説明する．

本稿は以下の通りに構成されている．まず，2 節で信頼性の高いストリーム処理を必要とする具体的な例を示す．そして，3 節で本研究で提案する持続型ストリーム処理環境のアーキテクチャを説明する．ストリームデータの整合性を維持するための機能については 4 節で述べる．5 節では，システム実現に向けて行った予備実験の結果を示す．6 節で関連研究について述べ，最後に 7 節でまとめと今後の課題を述べる．

2　利用例 : ガスタービンの遠隔監視

持続型データストリーム処理環境の利用例として，ガスタービン監視アプリケーションを用いて説明する（図 1）．ここでは，数箇所の地方に分散した発電施設のガスタービンの状態を，遠隔地にある管理センターから監視するものとする．各ガスタービンには，温度や回転数などを計測する何種類ものセンサーが取り付けられており，絶えずガスタービンの状態を監視している．センサーデータは各発電施設のコンピュータ（送信ノード）によって取得され，データストリームとして管理センターのコンピュータ（監視ノード）へ逐次配信される．管理センター側の監視ノードでは，送られてきたセンサーデータに対して，問合せ処理を適用する．具体的な問合せ要求としては，たとえば，各センサーの値が正常な範囲から外れていないかを監視する問合せや，全ガスタービンによる発電量の合計を定期的に求める問合せなどがあげられる．また，各所には障害に備えて待機ノードが配置されている．

ガスタービン監視アプリケーションに何かの障害が発生し，監視を続けられない状態になってしまうと，ガスタービン本体に重大な問題が発生しても検知する事ができなくなる．そのままガスタービンの運転を続けることはリスクが高くなるため，運転中止を選択しなければならないが，停止による経済的損失は大きいものとなってしまう．このように，ガスタービン監視のようなシステムでは，障害に対しても監視処理を継続可能な持続的な問合せ処理が必要となる．

この利用例において起こりうる障害と，それに対する提案システムの動作について述べる．

- 管理センターの監視ノードで障害が発生する場合．このときはすべてのガスタービンに対する監視処理が停止してしまい，もっとも大きな損害を与える．提案システムは，監視ノードの役割を待機ノードへ引き継がせて，新たな監視ノードとすることが可能なため，このような障害に特に有効となる．本稿では主にこの場合についての提案システムの動作について説明する．

- ある発電施設の送信ノードで障害が発生する場合．このとき，障害の起きた場所以外にあるガスタービンに対する監視は続けることが可能である．ただし，障害の起きたガスタービン自身の監視の継続は，センサーに直接アクセス可能なノード自体がなくなってしまうために，提案システムでも続行不可能である．もし，センサーが複数のノードからアクセス可能な構成になっていれば，提案システムを用いて送信ノードの役割を別のノードに引き継がせることは可能である．

- 発電施設と管理センターの間のネットワーク経路上で障害が発生する場合．提案システムは，一定時間が経過するまで待ち，その間に通信が回復した場合には，データの再送によって対処する．障害が一定期間以上になった場合には，分断されたそれぞれのネットワークの待機ノードが新たな監視ノードとなり，各ネットワーク内で可能な問合せ処理のみを継続する．通信が回復した後は，並立した複数の監視ノードが合流し，各監視ノードで行われた処理の内容が同期される．

なお，本研究では各ノードで発生する障害の原因として，ハードウェアの故障と，ソフトウェアのバグによる故障，人間の操作ミスによる異常停止などを想定する．ただし，ソフトウェアのバグによる故障は，いつも必ず同じ処理を実行するとクラッシュする，というような再現性のあるものは除外する．

3　持続型データストリーム処理環境

本研究で提案する持続型データストリーム処理環境のアーキテクチャを示す．

はじめに，本システムの物理的な構成について述べる．本システムは分散環境において動作することを想定しており，複数台のノードによって構成される．本稿では，それぞれの役割に応じて送信ノード，監視ノード，待機ノード，アプリケーションノードと呼ぶ．送信ノードは実際にセンサーなどの情報源にアクセスし，データを本システム内で扱えるリレーションのタプルの形式に変換して，別のノードへデータを配信する．監視ノードはストリームデータに対して登録された問合せの評価を行い，利用者へ配信する．利用者は，アプリケーションノード上のプログラムから問合せを監視ノードに登録し，問合せ結果を逐次受け取る．待機ノードは監視ノードに障害が発生した場合にその代替候補となるノードである．ネットワーク障害などへの耐性を上げるため，監視ノードとは別のネットワークに置かれる．正常時には

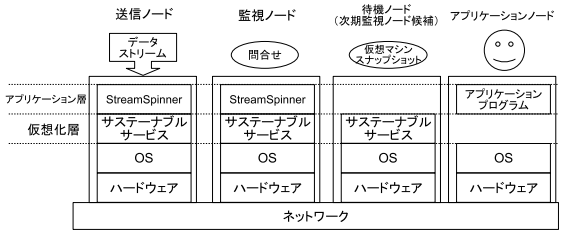

図 2: システムアーキテクチャ

待機ノードは監視ノードの状態を見張り，監視ノードの障害時には待機ノードの中の 1 つが次期の監視ノードとなる．

次に，各ノード上で動作するソフトウェアコンポーネントについて説明する．本システムは，通常のハードウェア，OS の上に，仮想化層があり，さらにその上でアプリケーションが実行される（図 2）．仮想化層は，仮想マシンなどの計算機仮想化技術により，アプリケーションと物理的なノードとの結びつきを切り離すことを目的とする．仮想化層のコンポーネントとして本研究では，3.2 節で述べるサステーナブルサービスツールキット [9] を用いている．アプリケーション層は，実際の問合せ処理を行うために必要なコンポーネント群で構成される．ここでは，我々の研究グループがこれまで開発を行ってきた StreamSpinner[11] を拡張して使用している．StreamSpinner 自身と本研究による拡張部分については 3.1 節で述べる．

StreamSpinner は主記憶上でのデータ処理を行うため，ノード障害時には主記憶上の処理結果は消失してしまう．そこで，サステーナブルサービスツールキットとの連携によって，信頼性の高いデータストリーム処理を実現する．

3.1 StreamSpinner

StreamSpinner[11] は我々の研究グループがこれまで開発を行ってきたデータストリーム処理エンジンである．本研究では StreamSpinner を拡張し，分散処理とデータ整合性の維持機能を追加している．StreamSpinner は，サステーナブルサービスツールキット上で動作する．

StreamSpinner の 1 ノードは，ラッパー，メディエータ，問合せ解析器，配信モジュール，中継モジュールからなる（図 3）．外部のデータストリームからのデータ取得はラッパーを介して行われる．ラッパーは新規データの到着を検知し，StreamSpinner の内部データ形式であるリレーションに変換する．そして，データ到着イベントをメディエータに通知する．メディエータでは，通知されたイベントに応じて，事前に登録された問合せの処理を実行する．問合せの評価は連続的問合せ処理 [8] の枠組みに基づいており，イベントの発生に応じて繰り返し問合せを評価する．1 回ごとの問合せ評価では，前回の実行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．生成された処理結果は，配信モジュールによって，利用者のアプリケーションノードや別の StreamSpinner のいるノードへ配信される．API モジュールは，アプリケーションノードから問合せを定義したり，処理結果を受信するためのインターフェースとして機能する．StreamSpinner では，データストリームに対する問合せ処理要求を 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．

複数のノード上での StreamSpinner 同士の連携について述べる．本システム上では，送信ノード，監視ノード上のアプリケーション層でそれぞれ StreamSpinner が稼動する．送信ノードの StreamSpinner はラッパーを用いて取得したデータを，監視ノードへ転送する．監視ノードの StreamSpinner は，転送されたデータに対して問合せ処理を行い，アプリケーションノードへ送信する．StreamSpinner 同士のデータ転送は，配信モジュールと中継モジュールの間，配信モジュールと API モジュールの間で行われる．中継モジュール，送信モジュール，API モジュールには，4 節で述べるデータの整合性を維持するための機能が組み込まれている．本研究では簡略化のため，問合せは監視ノード上でのみ処理されるものとし，分散環境における問合せ処理の最適化については考慮しないものとする．

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キット [9] は，仮想化技術を用いてソフトウェアサービスを複数の計算機上で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)[10] と，P2P ネットワーク上の分散ストレージシステム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の主記憶上のデータを含むすべての実行状態をスナップショッ

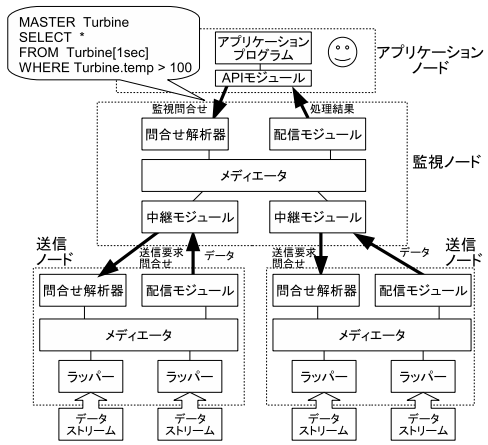

図 3: データストリーム処理エンジン StreamSpinner

トとして保存することができる．そのため，後からスナップショット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．提案システムでは，送信ノード，監視ノード，待機ノード上でサステーナブルサービスツールキットが動作する．以下にサステーナブルサービスツールキットによる，StreamSpinner の持続的実行の流れを示す（図 4）．

1. 仮想マシンの制御：監視ノードは SBUML を用いて仮想マシンを起動し，仮想マシン上で StreamSpinner による問合せ処理サービスを開始する．また，監視ノードは SBUML の機能により，定期的に仮想マシンのスナップショットを作成する．

2. スナップショットの共有仮想マシンのスナップショットは，複数台の待機ノードで構成される分散ストレージ上に置かれ，共有される．スナップショット全体のデータサイズは通常数 GB になるが，SBUML は差分的なスナップショット作成機能を持つため，毎回のスナップショットデータのサイズは数十〜百数十 MB である．

3. 障害の検知:障害の検知もサステーナブルサービスツールキットが行う．監視ノードは待機ノードに対して，定期的な生存メッセージ（ハートビートメッセージ）を送信する．待機ノードは，メッセージの到着を一定時間待ち受け，時間切れによって監視ノードの障害発生を検知する．

4. 仮想マシンの復元:サステーナブルサービスツールキットが障害を検知した時には，設定された

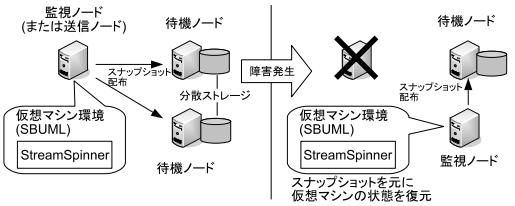

図 4: サステーナブルシステム

優先順位に基づいて待機ノードの 1 つが次期監視ノードとなり，分散ストレージ上のスナップショットデータを収集して，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．スナップショット作成時点に状態が戻ったことによってデータの一部が失われたり，また，再開の待ち時間中に溜まったデータが一気に届いたりする可能性があるため，StreamSpinner は回復処理（4.1 節）を行う．

5. 合流：ネットワーク障害によって分断されていた監視ノードとの通信が回復した場合は，複数の監視ノードが並立する状態となるため，監視ノードをひとつに合流させる．合流に際しては，監視ノード上の StreamSpinner 同士で，内部状態の整合性をとるための処理（4.2 節）が行われる．

4 ストリームデータの整合性維持

本研究において StreamSpinner に追加した，データの整合性維持のための機能について述べる．データストリームのアプリケーションには，部分的なデータ消失や重複の発生が問題にならないものもあるが，一般には障害による影響は可能な限り防がなくてはならない．例えばガスタービン監視アプリケーションでは，センサーデータの消失は重要な故障の兆候を見逃すことにもつながりかねない．本システムにおいてデータの整合性の維持が必要となるのは，仮想マシンが再開され，スナップショット作成時点に実行状態が戻された直後の回復処理と，ネットワーク分断の復旧後に複数の監視ノードを合流させる際の合流処理である．

4.1 仮想マシン再開時のデータの整合性維持

サステーナブルサービスツールキットによって，スナップショット作成時までに監視ノードに届いていたデータは完全に復旧できるが，それ以後に届いたデータは含まれないため，そのまま再開してしまうとデータの欠落が発生してしまう．本節では配信モジュール，中継モジュール，API モジュールに組み込んだデータ整合性を維持するための機能と，それを用いた回復処理について述べる．

配信モジュールは，送信するデータに対して必ずシーケンス番号を付加してから，別ノードにいる中継モジュールや API モジュールへの送信処理を行う．データストリームが複数ある場合は，シーケンス番号のカウンタはデータストリームの数だけ用意される．配信モジュールは，データの送信が正常に完了してもデータを直ちに廃棄せず，自身の管理するバッファにシーケンス番号とともに保持する．バッファに格納されたデータは，他のノードからの再送要求を処理するために利用される．配信モジュールからデータを受け取った別ノードの中継モジュールまたは API モジュールは，シーケンス番号を確認し，前回に受信したデータのシーケンス番号と比較する．正常に動作している場合は，前回受信したシーケンス番号に 1 を加えた値となる．監視ノード側でスナップショットが保存されると，中継モジュールはその時点のシーケンス番号を送信ノード側の配信モジュールへ通知する．スナップショット以前のデータは復旧可能であるので，送信ノード側配信モジュールは，受け取っ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．

次に，回復処理について述べる．障害の発生後に，監視ノードの仮想マシンが別ノードで再開されたとき，StreamSpinner はシーケンス番号を手がかりに回復処理を開始する．中継モジュールが記憶している各データストリームのシーケンス番号を使って，対応するすべての送信ノードへ再送要求を送る．送信ノードでは，配信モジュールのバッファの中から，要求されたシーケンス番号よりも新しいデータをすべて再送する．監視ノードは，全送信ノードからの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実行し，結果を API モジュールに送信する．このとき，回復処理で生成される問合せ結果には，障害前に一度 API モジュールに送信されている可能性がある．ただし，到着順番を考慮しながら問合せ処理をするため，処理結果に付加されるシーケンス番号は障害前に付加したものと同じになる．そのため API モジュールは，以前に受信したデータと同じまたは古いシーケンス番号のついたデータを受け取った場合，それを無視することができる．

4.2 ネットワーク分断と合流時におけるデータの整合性維持

ネットワーク分断直後，監視ノードは一部の送信ノードからのデータを受け取ることが出来ない状態に陥るため，一部の問合せの処理は継続不可能になる．StreamSpinner は現在実行中の問合せを分析し，(1) 必要なすべてのストリームデータが受信可能なもの，(2) 一部だけ受信不能なもの，(3) すべてのデータが受信不能なもの，に分類する．(1) の問合せについては問合せ処理を継続し，(2)，(3) の問合せについては問合せ処理を中断する．ただし，(2) の問合せについては，合流時に備えて受信可能なデータの蓄積だけは行う．

ネットワークが回復し，複数個の監視ノードの状態を合流させるときは，ひとつが次期監視ノードとなり，残りは合流ノードとなる．次期監視ノード上で (1) に分類されていた問合せはそのまま処理を継続する．(3) に分類されていた問合せのうち，合流ノードで (1) に分類されているものがあれば，そちらに問合せ処理の状態を同期する．(2) に分類された問合せについては，合流ノードで蓄積されたすべてのデータを次期監視ノードへ集め，データがすべてそろっていた場合には，回復のための問合せ処理を開始する．

実際には，分断されたネットワークが更に分断され，その後合流するなど，より複雑な場合が起こりうる．また，長期に渡ってネットワーク分断が継続する場合は，データを蓄積する記憶容量が足りなくなる可能性がある．そのような状況にも対応可能な合流処理の検討は今後の課題である．

5 予備実験

本稿で提案した持続型ストリーム処理環境は，現在実装に向けて準備を行っている段階である．本研究ではそれに先立って予備実験を行い，仮想化技術を用いてデータストリーム処理エンジンを復旧させるアプローチが現実的な時間で動作するのかを確認

表 1: ホスト OS 側実験環境

CPU	Pentium 4 2.8GHz
メモリ	1GB
OS	Vine Linux 3.2 (kernel 2.4.31)
仮想マシン環境	SBUML-core-2423-2um-1sb-082504d

表 2: ゲスト OS 側実験環境

ゲストOS	User-Mode Linux 2.4.19 + Vine Linux 3.2
割り当てメモリ	256MB
仮想ディスク	4GB (使用率50%)
StreamSpinner実行環境	JDK 1.5.0_07

した．

5.1 実験環境

本実験では，1 ノード上のみで SBUML を用いて仮想マシンを起動し，その上で StreamSpinner を実行する．まず，実験環境について説明する．SBUML を動かすホスト OS 側の環境は表 1 の通りである．また，SBUML が起動する仮想マシンのブートイメージおよび割り当て資源は表 2 の通りである．本研究では仮想マシンは，StreamSpinner の実行に最低限必要な動作環境よりも，多めに余裕を持たせた構成とした．

次に実験に用いたデータと問合せについて述べる．本実験では，図 5 の問合せを 10 個または 100 個登録した．ただし，図 5 は 2 種類のデータストリームに対する直積を毎分計算する問合せで，Clock_1minute は毎分定期的にアラームを配信するストリーム，Stream_1 と Stream_2 は毎分 60 タプルまたは 120 タプルを配信するストリームである．実験では，登録した問合せを 5 分間実行させ，到着するデータを次々処理させた．

5.2 スナップショット作成時間とデータサイズ

実験では，StreamSpinner が問合せを処理している間に，SBUML を用いて仮想マシンのスナップショットを作成し，その作成時間とスナップショットのデータサイズを計測した．測定は，毎分 60 タプル到着するストリームを 10 問合せが監視する場合，毎分 120 タプルで 10 問合せの場合，毎分 60 タプルで 100 問合せの場合の 3 つで，それぞれでスナップショットの作成を 8 回行い，それらの平均を求めた．実験結果は図 6 と図 7 である．スナップショットの作成時間はほぼ 28〜29 秒程度で，問合せ数や到着データ数にはそれほど影響されていない．スナップショットデータのサイズは 140〜160MB で，問合せ数や到着データ数に応じて増加することが確認できた．

```
MASTER Clock_1minute
SELECT *
FROM Stream_1[ 1minute ], Stream_2[ 1minute ]
```

図 5: 実験に用いた問合せ

5.3 仮想マシン復旧時間

5.2 節の実験で作成した仮想マシンのスナップショットを実際に復旧する場合にかかる時間を計測した．ここでも，8 回行ってその平均を求めた．実験結果は図 8 である．仮想マシンの復旧にかかる時間は 10〜20 秒程度，問合せ数や到着データ数によって若干変化することが確認できた．

本実験により，分単位で動作するストリーム処理に対しては，現状のままで十分実用になることがわかった．秒単位以下での処理が要求される場合については，工夫の余地があると思われる．ただし，今回の実験ではサイズの大きいブートイメージを用いて仮想マシンを起動したため，このような時間になったと考えられる．StreamSpinner の動作に必要最小限のイメージを作るだけでも，これらの時間の削減が期待できる．

6 関連研究

[7] では，クラスタマシンにおける信頼性向上のため，プロセスペアの概念を用いたアプローチを提案している．プライマリとセカンダリのノードでデータストリーム処理エンジンを走らせ，それぞれに同じ問合せ処理を行わせることで，冗長度をあげている．ただし，これはクラスタマシンにおける信頼性向上のための研究であり，本研究とは目的が異なっている．

広域に分散したデータストリーム処理エンジンにおける信頼性を扱った研究としては，[5] がある．彼らは途中経路上のノードが障害によって落ちた場合

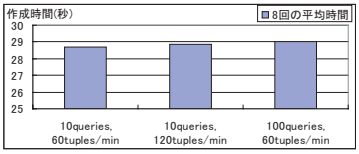

図 6: スナップショット作成時間

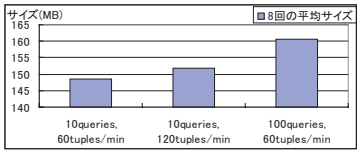

図 7: スナップショットサイズ

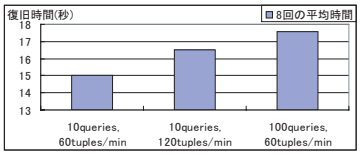

図 8: 仮想マシン復旧までの時間

に，複数のノードの連携によってデータを復旧するための方式として，Passive Standby, Upstream Backup, Active Standby の 3 つを提案している．また，[3] では，ネットワーク障害などで一部のノードからのデータが得られなくなった場合に，それ以外の正常ノードからのデータだけで仮の処理結果を生成・配信することを提案している．障害状態の間に生成された仮の処理結果は，ネットワーク障害が回復した後で，本来あるべきだった結果処理に置き換えられる．

これらに対して，提案システムは計算機仮想化技術とデータストリーム処理エンジンとの連携により，信頼性を向上させるという異なるアプローチをとっている．ストリーム処理エンジンを仮想マシン上で実行することで，実行中の状態のまま他のノードへ問合せ処理サービスを容易に移すことが可能である．また，本研究ではネットワーク分断・合流後の監視ノードの合流処理についても考慮している．

7 むすび

本研究では，信頼性の高いストリーム処理を実現するための持続型ストリーム処理環境のシステムアーキテクチャについて述べた．本研究のアプローチは，計算機資源仮想化技術との連携によって，問合せサービスを持続させるというものである．

今後の課題としては，本稿で述べた StreamSpinner への拡張機能を実装し，評価実験を行うことがあげられる．また，よりリアルタイム性の必要なアプリケーションのために，スナップショット作成や仮想マシン再開の時間を削減するための処理方式についても検討する．

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」，科学研究費補助金基盤研究 (A)(#18200005) による．

参考文献

[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,” Technical Report, http://dbpubs.stanford.edu/pub/2003-67 , 2003.

[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] R. Motwani, et al. “Query Processing, Resource Management, and Approximation in a Data Stream Management System,” Proc. CIDR, 2003.

[7] M. A. Shah, et al. “Highly-Available, Fault-Tolerant, Parallel Dataflows,” Proc. ACM SIGMOD, pp.827–838, 2004.

[8] 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.

[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] ScrapBook for User-Mode Linux. http://sbuml.sourceforge.net/

[11] StreamSpinner. http://www.streamspinner.org/