高信頼なデータストリーム処理システムにおける リカバリ時間短縮手法の提案

長野　恭子† 糸川　剛† 北須賀輝明† 有次　正義†

† 熊本大学大学院自然科学研究科 〒 860-8555 熊本市黒髪 2 丁目 39 番 1 号
E-mail: †kyoko@dbms.cs.kumamoto-u.ac.jp, ††{itokawa,kitasuka,aritsugi}@cs.kumamoto-u.ac.jp

あらまし　近年，継続的に発生するデータの処理を行うストリーム処理システムの需要が高まっており，このようなシステムは分散環境でも構築されている．分散環境においては 1 台の計算機の故障が，システム全体に影響を及ぼさないように高信頼化の技術を組み込む必要がある．そこで，プライマリの計算機の他にバックアップの計算機を配置する高信頼化手法が盛んに研究されている．高信頼化の既存手法では，故障が発生していない通常処理の場合，バックアップの計算機はアイドル状態となっている．本研究ではバックアップの計算機にも通常の処理を振り分ける手法を提案する．これによりリカバリ時に再処理するタプル数を削減し，リカバリ時間の短縮を実現する．
キーワード　ストリーム処理システム，高信頼化

An Approach to Reducing Recovery Cost for High-Availability Stream Processing Systems

Kyoko NAGANO†, Tsuyoshi ITOKAWA†, Teruaki KITASUKA†, and Masayoshi ARITSUGI†

† Graduate School of Science and Technology, Kumamoto University
2-39-1 Kurokami, Kumamoto, Kumamoto 860-8555, Japan
E-mail: †kyoko@dbms.cs.kumamoto-u.ac.jp, ††{itokawa,kitasuka,aritsugi}@cs.kumamoto-u.ac.jp

Abstract　Recently, a requirement for stream processing systems which deal with data continually generated has been increasing. Such systems are often developed in distributed environments. Since the failure of a computer may cause damages to all of the systems in distributed environments, high-availability mechanisms must be involved in distributed stream processing systems. One of the high-availability mechanisms is to provide a backup computer for a primary computer. When a primary computer fails, the backup computer takes over the operation of the primary computer. It is important to reduce the cost for the recovery. In this paper, we propose an approach to realize short recovery from fails by making use of an idle backup computer.
Key words　Stream processing system, High-availability

1. はじめに

近年，センサを用いた交通状況の監視システムや，株価のモニタリングといった金融アプリケーションなど，リアルタイムで処理を行うシステムが増加している．このように継続的に発生するストリームデータの処理を行うシステムとして，ストリーム処理システムに対する需要が高まっている．また，データソースと処理部が地理的に離れた場所にあるこれらのアプリケーションを, 分散環境で実現する分散ストリーム処理システムの研究が行われている [1], [2].

分散環境においては，1 台の計算機で故障が発生しても全体へ影響を及ぼすことなく，システムが稼動し続けなければならないため，高信頼化の技術を組み込む必要がある．そこで主に処理を行う計算機と，故障が発生した場合に処理を引き継ぐバックアップの計算機の 2 台を用意するという高信頼化手法の研究が行われている [3]〜[5].

Hwang ら [3] は高信頼化のアルゴリズムとして，Passive Standby,Upstream Backup,Active Standby の 3 手法を示している．これらの中で，Upstream Backup に注目する．Upstream Backup は高信頼化のために生じるオーバヘッドがほとんど無いという利点があるが，障害回復のためのリカバリ時間が長いという欠点がある．また，故障が発生していない間，バックアップのために用意してある計算機はアイドル状態で，処理を行っていない．そこで，本研究では，このアイドル状態であるバッ

クアップの計算機にも処理を振り分ける手法を提案する．2 台の計算機を用いることで，故障が発生した後のリカバリ処理において，再処理するべきタプルの数を削減することができ，リカバリ時間が短縮できることを示す．

本稿の構成は以下の通りである．まず 2 章で分散ストリーム処理システムに関連する研究を述べて，高信頼化手法の従来研究を詳しく説明する．3 章では，提案手法を説明し，4 章で提案手法の有効性を検討するため評価実験を行い，考察する．5 章でまとめと今後の課題について示す．

2. 関 連 研 究

現在，Brandeis/Brown/MIT の合同プロジェクトである Aurora [6] や，UC Barkeley による TelegraphCQ [7] などの分散ストリーム処理システムが開発されている．これらに関連した研究が幅広く行われている．

分散ストリーム処理システムは，データの処理部の各演算を複数の計算機へと割り当てている．そこで，各演算をどの計算機へと配置するかというアルゴリズムが必要となる．Repantis と Kalogeraki [8] は，各計算機間の物理的な距離によって発生する通信の遅延と，各計算機の処理能力を考慮して処理に必要な演算を計算機へと配置するアルゴリズムを提案している．また，分散ストリーム処理システムにおける高信頼化の技術に関する研究も盛んに行われており，いくつか手法が提案されている [3]〜[5]．これらは，分散ストリーム処理システムを構成する各計算機とは別に，バックアップ用の計算機を配置することで，ある計算機の故障時にバックアップが処理を引き継ぐという共通した方式を持っている．

これらの関連研究の中でも，1 つの計算機の故障がシステム全体に影響を及ぼさないための高信頼化技術は，分散環境において重要である．そこで本研究では，この高信頼化手法に焦点を当てる．

2.1 従来の高信頼化手法

Hwang ら [3] によって，Passive Standby，Upstream Backup，Active Standby という 3 つの高信頼化手法が示されている．ここではまず，本研究で扱う分散ストリーム処理システムのモデルを示し，高信頼化のための手法として提案された 3 手法について詳しく説明する．

2.1.1 システムモデル

図 1 に分散ストリーム処理システムの例を示す．システム上を流れるストリームデータは，リアルタイムに処理されるタプルのシーケンスである．システムの演算処理が上流ノード N_u，ノード N，下流ノード N_d の 3 つのノードに分散している．また，ストリームデータの流れる方向は実線の矢印で描かれており，上流ノード N_u からノード N へとタプルが流れる．ノード N でタプルの処理を行い，下流ノード N_d へと送信する．

このようなシステムにおいて，処理を行うノード N をプライマリと言い，プライマリに接続されるバックアップのためのノード N' をセカンダリと言う．セカンダリは，プライマリで故障が発生したことを検知し，処理を引き継ぐ役割を持つ．故障時のデータの流れは破線の矢印で表されている．また，各

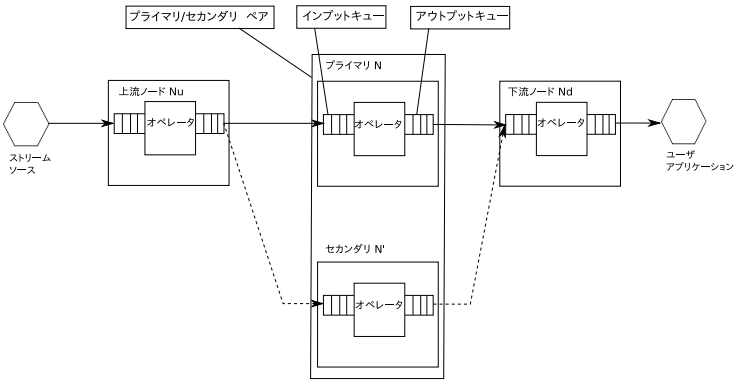


図 1　分散ストリーム処理システムの例 [3]

ノードは入力と出力のキューを持っている．

故障を発見するために，セカンダリは定期的に生存確認のパケットをプライマリへ送信する．それを受信したプライマリは応答を返す．故障が発生した場合，プライマリは応答を返さない．セカンダリで生存確認がタイムアウトとなったら，セカンダリはプライマリで故障が発生したと判断して上流ノードにリクエストを送信し，セカンダリが処理を引き継ぐ．なお，生存確認のパケットを送る間隔を “生存確認送信間隔” と定義する．

システムの基本的な処理の流れは上記のようになっているが，プライマリとセカンダリの利用方法によって高信頼化手法は 3 つに分類されている．この高信頼化手法について以下に説明する．

2.1.2 Passive Standby

Passive Standby は，プライマリ N が完全にタプルを処理するまで上流ノード N_u の出力キュー内に，タプルを保存しておく方式である．プライマリはタプルの処理を行うと同時に，その内部状態の変化を知らせるメッセージをセカンダリ N' へ送信する．この時の内部状態をチェックポイントと言い，このメッセージをチェックポイントメッセージと言う．セカンダリはチェックポイントメッセージを受信すると，上流ノードへどのタプルまで処理が完了したかを知らせ，上流ノードは出力キューから該当タプルを削除する．

プライマリが故障した場合，セカンダリが上流ノードにプライマリの故障を通知することで，上流ノードは出力キュー内にあるタプルをセカンダリへ再送信する．セカンダリはチェックポイントメッセージによってプライマリの内部状態を復元し，最新のチェックポイントから処理を再開する．

2.1.3 Upstream Backup

Upstream Backup も上流ノードの出力キュー内に，タプルを保存する方式である．しかし，プライマリは内部状態をセカンダリへ送信せず，上流ノードの出力キューからタプルを削除するために，Level-0 ACK と Level-1 ACK の 2 つのメッセージを用いる．

プライマリ N に上流ノード N_u からタプル t_1 が届き，それを処理して下流ノード N_d にタプル t_1' を出力した状況を考える．下流ノードは，プライマリから受信したタプル t_1' の識別情報を付加した Level-0 ACK をプライマリに送信する．なお，

Level-0 ACK を送信する間隔を “ACK 送信間隔” と定義し，これは一定とする．Level-0 ACK を受け取ったプライマリは，その中の識別情報を参照してタプル t_1' を生成したタプル t_1 を見つけ，t_1 の識別情報を付加した Level-1 ACK を上流ノードへ送信する．この Level-1 ACK を受信すると，上流ノードはタプル t_1 を出力キューから削除する．

プライマリで故障が発生した場合，セカンダリは上流ノードにプライマリの故障を通知し，上流ノードは出力キュー内に保存されていたタプルをセカンダリへ再送信する．セカンダリはそれらを受信し，タプルの再処理を行い処理を引き継ぐ．

2.1.4 Active Standby

Active Standby はプライマリ N とセカンダリ N' の両方が上流ノードからタプルを受信し，並列に処理を行う．プライマリで故障が発生していない場合はセカンダリもタプルの処理を行うが，下流ノードに出力せず，セカンダリの出力キュー内にタプルを保存しておく．Upstream Backup と同様に，プライマリに上流ノード N_u からタプル t_1 が届き，それを処理して下流ノード N_d にタプル t_1' を出力した状況を考える．下流ノードはプライマリから受信したタプル t_1' の識別情報を付加した Level-0 ACK をプライマリに送信する．また，セカンダリではプライマリと同時に処理が行われ，タプル t_1' が出力キューに保存されている．Level-0 ACK を受信したプライマリは，タプル t_1' が下流ノードまで到達したことをセカンダリへ知らせる．すると，セカンダリは出力キューからタプル t_1' を削除する．

プライマリで故障が発生した場合，セカンダリは出力キュー内のデータを下流ノード送信し，処理を引き継ぐ．

また，Active Standby を WAN 環境に応用した手法も提案されている [4]．この場合，セカンダリも下流ノードに処理を行ったタプルを送信する．下流ノードはプライマリとセカンダリからタプルを受信することとなるが，下流ノードでは最も速く到達した方のタプルを用いて処理を継続するという方式をとる．セカンダリも下流ノードにタプルを送信しているため，出力キューから古いタプルを削除する必要がなく，Level-0 ACK は不要となる．計算機の故障やネットワークの障害がより頻繁に起こることが想定される環境においては，ストリームデータだけでなく Level-0 ACK も，該当するノードに到達しない可能性が大きい．したがって WAN 環境に適した手法と言える．

2.2 従来手法の特性と問題点

Hwang ら [3] はこれらの手法の特性をバンド幅オーバヘッドとリカバリ時間という 2 つの指標を用いて評価している．全データ転送に用いたバンド幅のうち，高信頼化のために用いたバンド幅をバンド幅オーバヘッドという．また，データの処理中に故障が発生した場合，データの損失が生じる．故障を検知してから損失したデータを再処理し，故障前の状態へ復旧するまでの時間をリカバリ時間という．

Active Standby は，プライマリとセカンダリの両方にタプルを送信するため，バンド幅オーバヘッドはほぼ 100%となる．一方，セカンダリにおいて並列でタプルを処理しているため，リカバリ時間は非常に少ない．Upstream Backup では，故障時のみセカンダリへデータを送信するため，バンド幅オーバヘッドはほとんどかからない．一方で，上流ノードからタプルの再送信を行い，セカンダリで再処理を行うためリカバリ時間は長くなる．Passive Standby はセカンダリがプライマリの内部状態を保持しており，故障時にはその状態から再処理を行う．したがってリカバリ時間は Upstream Backup よりも短くなる．しかしプライマリがセカンダリへ内部状態を送信する際には多くのバンド幅を消費してしまうため，そのオーバヘッドは大きくなる．

上記のことから，Upstream Backup は 3 手法の中ではバンド幅オーバヘッドが最も少ないという利点をもっている．したがって，システムを構築する際に高信頼化のために用いるバンド幅の制限を考慮する必要がない．しかし，故障が発生した際に，リカバリ時間が長くかかってしまうという問題が挙げられる．したがって上流ノードの出力キュー内にあるタプルの数が多ければ，その分セカンダリで再処理を行わなければならないため，リカバリに時間がかかることになる．

ここで，故障が発生していない場合の各手法のセカンダリの動作について注目する．Active Standby では上流ノードから常にストリームデータが送信され，セカンダリは待機している．Passive Standby では，セカンダリはプライマリの内部状態を知らせるメッセージを受信しており，メッセージを受信した時点までで処理が完了しているタプルを，上流ノードへ知らせるという働きを持っている．すなわち，これらの 2 手法は故障時でない場合もセカンダリは動作していることになる．一方，Upstream Backup では故障が発生していない間，セカンダリはプライマリに対して生存確認を行うのみで実際に動作はしておらず，アイドル状態となり待機している．

そこで，このセカンダリにも処理を行わせることによって，リカバリ時に再処理を行うタプル数の削減を図る．本研究ではこのようにして，Upstream Backup の唯一の欠点であるリカバリ時間を短縮させる手法を提案する．

3. 提 案 手 法

高信頼化のための従来手法である Upstream Backup では，リカバリ時間が長いという問題点があった．また，プライマリとセカンダリの計算機を配置するが，通常はプライマリだけで処理を行い，故障時のみセカンダリを利用する．そこで，通常はアイドル状態にあるセカンダリにも処理を行わせることで，故障発生時に再処理を行うタプルの数を削減し，リカバリ時間を削減する手法を提案する．本章では，提案手法の流れを説明し，その動作例を示す．

3.1 動　　作

提案手法を，通常時の処理の流れ，生存確認の方法，上流ノードが保持する出力キュー内のタプルの削除と再送信に関して順に説明する．また，本手法のもととなるシステムの構成図を図 2 に示す．

（1） **データの流れ**

提案手法においては，上流ノードからプライマリとセカンダリへ，ストリームデータを切り替えて送信する．図 2 ではこのデータの流れを実線の矢印で示す．上流ノードは送信したタ

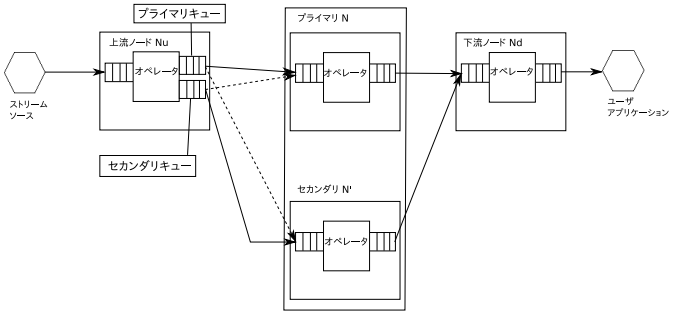


図 2　提案手法のシステム構成図

プルを保存する出力キューとして，プライマリ用とセカンダリ用の 2 つのキューを持つ．そして，プライマリに送信したタプルはプライマリ用の出力キューに，セカンダリに送信したタプルはセカンダリ用の出力キューに分割して保存する．プライマリもしくはセカンダリは，上流ノードからのデータを受信すると処理を行い，出力タプルを下流ノードへと送信する．下流ノードは，プライマリかセカンダリのどちらからデータを受信したか判断して，該当するノードへ Level-0 ACK を返す．このとき，Level-0 ACK を返す “ACK 送信間隔” は Upstream Backup と同様に一定間隔とする．Level-0 ACK を受信したプライマリもしくはセカンダリは，出力が下流ノードで受信され処理が完了したことを上流ノードに知らせるために，Level-1 ACK を上流ノードへ送信する．上流ノードは，プライマリ，セカンダリのどちらから Level-1 ACK を受信したか判断して，その出力キューから該当するタプルを削除する．

プライマリもしくはセカンダリにおいて故障が発生した場合，後述の生存確認により故障を検知し，生存しているノードは上流ノードに故障発生を知らせるメッセージを送信する．これにより，上流ノードは故障したノードの出力キュー内のタプルを生存しているノードへと送信し，再処理を行うことによって，リカバリを実現する．

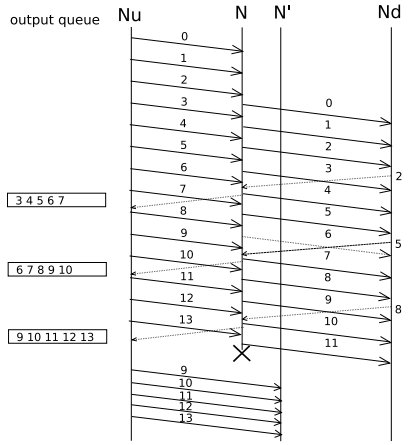
（2） **生存確認方法**

従来の手法において，セカンダリからプライマリへ，一方向に定期的にパケットを送信し，生存確認を行っていた．セカンダリがパケットを送信し，それを受け取ったプライマリは応答を返す．一定期間応答が返らずにタイムアウトとなった場合，プライマリに故障が発生したと判断する．

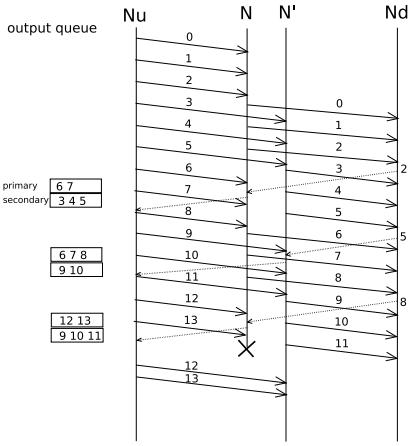
本手法では，プライマリとセカンダリの両方を処理に用いるため，双方向で定期的に生存確認を行う．従来のセカンダリからプライマリへの生存確認に加え，プライマリからセカンダリへの生存確認も行い，どちらのノードで故障が発生した場合でも検知できるようにする．

（3） **キュー内のデータ再送信**

故障が発生し，生存しているノードがそれを検知すると，上流ノードにタプルの再送信のリクエストを送信する．その時，上流ノードは出力キュー内のタプルを再送信しなければならない．本手法では，タプルを保存しておくための出力キューをプライマリ用のものと，セカンダリ用のものの 2 つに分割している．そのため，故障が発生した側のキュー内のタプルを再送信



(a) 従来手法 Upstream Backup



(b) 提案手法

図 3　ストリームデータの流れと出力キュー

する仕組みが必要となる．プライマリもしくはセカンダリが故障を検知して，再送信のリクエスト送信する．すると上流ノードは，リクエストを受け取ったノードで故障が発生したことを識別し，故障したノードの出力キュー内にあるタプルのみ再送信を行う．図 2 ではこのリカバリ処理時のタプルの流れを破線の矢印で示す．

3.2 動 作 例

図 3(a) に従来手法である Upstream Backup，図 3(b) に提案手法それぞれにおけるストリームデータの流れと，上流ノードの出力キュー内のタプルを示す．各タプルの送受信は実線の矢印で表しており，タプルを識別するための ID として，タプルにシーケンス番号が割り振られている．また，点線の矢印は Level-0 ACK，Level-1 ACK の各 ACK と該当するタプルのシーケンス番号を表している．左側には上流ノードのもつ出力キューを示している．

従来手法では，プライマリに故障が発生したとき，上流ノードのキュー内には (9,10,11,12,13) の 5 つのタプルがあり，これらを全てセカンダリに送信しなおして再処理を行う．提案手法

ではプライマリにおいて故障が発生したとき，セカンダリ側に送信していたデータ (9,10,11) はセカンダリで正常に処理が行われている．そのため再送信する必要がなく，プライマリ側のキュー内にあるデータ (12,13) のみを再送信する．このように，再送信しなければならないタプルの数が減るので，リカバリ時間の短縮につながると考えられる．

リカバリ時に，故障が発生していないノードへ送信したタプルは再送信しないため，下流ノードが受信した出力結果のタプルの順序が保たれない場合が生じる．演算の種類によってはタプルの順序の変更にセンシティブなものがあるが，その場合，必要に応じて下流ノードにおいてタプルのソートを行わなければならない．したがって，ソート処理が必要な演算では，下流ノードにおけるコストが増えてしまうことが考えられる．

4. 実　　験

本章では，提案手法の有効性を検証するために行うシミュレーションについて延べ，実験結果について考察する．

4.1 シミュレーションの概要

シミュレーションを行う環境として，図 2 に示すような上流ノード，プライマリ，セカンダリ，下流ノードで構成されるシステムを仮定する．また，各ノード間の通信には TCP/IP プロトコルを用いた．このシステム上を流れるタプルは，ID 部とデータ部で構成される．ID はタプルが生成された順番を示し，データ部は擬似的な数値データとする．また，演算は入力に対して特定の値のみを出力するフィルタ演算を行うと仮定する．フィルタ演算はタプルの順序が変更されても出力結果には影響しないため，下流ノードにおいてソートを行う必要がない．

次にシミュレーションの流れを説明する．システムが定常状態となるように，一定時間ストリームデータを送信し続ける．プライマリもしくはセカンダリの動作を止めることで故障を発生させ，もう一方が故障を知らせるメッセージを上流ノードに送信すると，リカバリ処理を開始する．ネットワークシミュレータ ns-3 [9] を用いてこの環境を構築し，システムの振舞いを測定する．

4.2 検 証 項 目

従来手法との特性比較に用いる指標として，リカバリ時に再送信を行うタプルの数とリカバリ時間を測定する．リカバリ時間は，上流ノードがメッセージを受信した時刻から，生存しているノードでタプルの再処理が完了するまでの時間とする．

その際に，パラメータとしてストリームデータの入力レートを変化させる．Ack 送信間隔が一定であれば，入力レートの増加によって，上流ノードの出力キュー内に保存されるタプル数も増加する．この出力キュー内のタプル数と，リカバリ時間を測定することで本手法の有効性を検証する．シミュレーションに用いた計算機は CPU Intel(R) Core(TM)2 Duo 3.00GHz, メモリ 4GB，OS Ubuntu 9.04 である．また各パラメータを表 1 に示す．なお今回は，1 回のシミュレーションの時間を 1 分間とする．システムが定常状態となる時間を 30 秒間とし，これ以降のランダムな時刻に故障が発生するものと仮定する．このシミュレーションを 10 回試行した平均値を実験結果とする．

表 1　シミュレーションに用いたパラメータ

ACK 送信間隔	25ms
生存確認送信間隔	100ms
タプル (ID 部，データ部)	(8,50)byte
ネットワークバンド幅	16Mbps
ネットワーク遅延	5ms
演算時間	10μs

4.3 実 験 結 果

シミュレーション結果を図 4 と 5 に示す．図 4 は入力レートと故障時に再処理するタプル数の関係を示している．これより，従来手法に比べ，提案手法の方は再処理しなければならないタプル数が減っていることがわかる．しかし，入力レートが低い間，このタプル数は従来手法と提案手法での差がほとんどない．入力レートが低い時は，新たにタプルが送信されるまでに上流ノードは Level-1 ACK を受信し，出力キューからタプルが削除されていることが原因であると考えられる．したがって，入力レートが高いほど，リカバリ時に再処理するタプル数を削減できることがわかる．また図 5 は，入力レートとリカバリ時間の関係を示している．これより，再処理するタプル数を削減できれば，リカバリ時間も短縮されていることがわかる．

入力レートが 1000[tuples/s] 以下では図 4 において提案手法と従来手法の差は少ないが，図 5 のリカバリ時間では 2 つの手法で大きく差がでている．この原因は，システムの実装時に用いた TCP プロトコルが挙げられる．RFC896 [10] によると，TCP プロトコルには，送信側がバッファにデータを一旦取り込んでまとめて送信を行う Nagle アルゴリズムが使われている．従来手法では通常処理の際にセカンダリが用いられておらず，リカバリ処理を行う際に初めて利用される．つまり，これまで通信が行われていない相手と通信が開始されるため，Nagle アルゴリズムが働き，上流ノードで送信バッファにデータを一旦取り込む間で遅延が生じる．一方提案手法では，セカンダリとの通信が既に行われているためこのアルゴリズムが働かない．したがってリカバリ時間にはこのような差が生じたと考えられる．

シミュレーションを行った結果，通常の処理をプライマリだけでなくセカンダリにも振り分け，タプルを保存するキューを分割することで，再処理するタプル数が削減できたことが確認できた．また，システムの実装上では TCP の仕様によってリカバリ時間に影響が出ることがわかった．結果として，提案手法によりリカバリ時間の短縮が可能であると言える．

5. おわりに

本研究では，分散ストリーム処理システムの高信頼化手法である Upstream Backup に対してリカバリ時間の短縮のため，通常処理時にアイドル状態となっているセカンダリノードにも処理を振り分ける手法を提案した．上流ノード，プライマリ，セカンダリ，下流ノードから構成されるシステムでシミュレーションを行った．これにより，実際にリカバリ時に再送信するタプルの数を削減することができ，リカバリ時間の短縮が可能

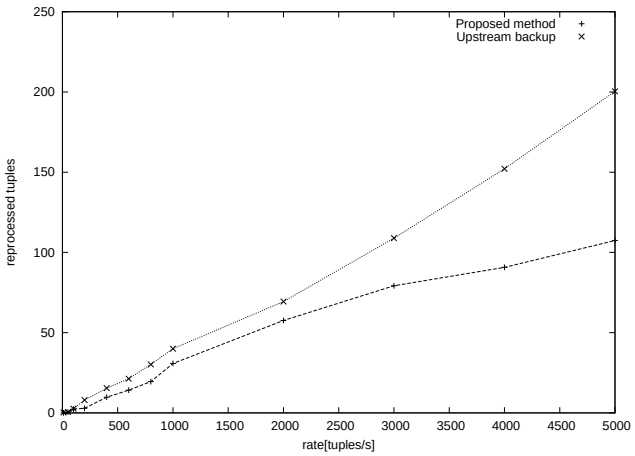


図 4　入力レートと再処理するタプル数

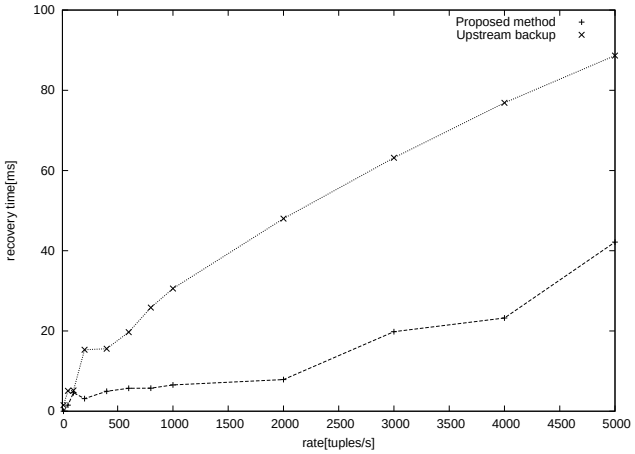


図 5　入力レートとリカバリ時間

となることがわかった．

今後の課題として，本手法について今回はリカバリ時間のみを議論したが，プライマリとセカンダリに処理を振り分けた場合の，各ノードにおけるパフォーマンスに関しても評価を行う必要があると考えられる．提案手法では，プライマリとセカンダリでのタプルの処理を行うパフォーマンスは向上すると考えられる．しかし，上流ノードにおいてはキューを分割して保存する必要があり，下流ノードにおいては必要に応じてソート処理をしなければならない．これらの原因によるパフォーマンスの低下も議論する必要がある．演算に関しても，今回はフィルタ演算のみで実験を行った．しかし，より多くの種類での演算で実験を行う必要がある．その際に Join や Aggregate 演算では，ソートを行うことによるペナルティが考えられる．またシミュレーションでは，プライマリとセカンダリへ送信するストリームデータの切り替え間隔は一定として実験を行った．しかし，処理能力の異なる 2 つの計算機をプライマリとセカンダリに用いた場合を想定すると，計算機のリソースを考慮した上で処理の振り分けを行う仕組みが必要となる．

実際に分散ストリーム処理システムのインフラ構築を行うことを考えると，リカバリ時間，パフォーマンス，消費電力等の指標を示すことによって，各計算機の性能やコストを考慮して，アプリケーションの仕様に適したシステムを設計することが可能になると考えられる．

文　献

[1] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *PODS*, pp. 1–16, 2002.

[2] Rajeev Motwani, Jennifer Widom, Arvind Arasu, Brian Babcock, Shivnath Babu, Mayur Datar, Gurmeet Singh Manku, Chris Olston, Justin Rosenstein, and Rohit Varma. Query processing, approximation, and resource management in a data stream management system. In *CIDR*, 2003. Available at http://www.cidrdb.org/cidr2003.

[3] Jeong-Hyon Hwang, Magdalena Balazinska, Alex Rasin, Ugur Çetintemel, Michael Stonebraker, and Stanley B. Zdonik. High-availability algorithms for distributed stream processing. In *ICDE*, pp. 779–790, 2005.

[4] Jeong-Hyon Hwang, Sanghoon Cha, Ugur Çetintemel, and Stanley B. Zdonik. Borealis-r: a replication-transparent stream processing system for wide-area monitoring applications. In *SIGMOD Conference*, pp. 1303–1306, 2008.

[5] 塩川浩昭, 渡辺陽介, 北川博之, 川島英之. 分散ストリーム処理システムにおける高信頼化手法の提案. In *DEIM Forum*, 2009.

[6] Daniel J. Abadi, Donald Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, and Stanley B. Zdonik. Aurora: a new model and architecture for data stream management. *VLDB J.*, Vol. 12, No. 2, pp. 120–139, 2003.

[7] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel Madden, Vijayshankar Raman, Frederick Reiss, and Mehul A. Shah. TelegraphCQ: Continuous dataflow processing for an uncertain world. In *CIDR*, 2003.

[8] Thomas Repantis and Vana Kalogeraki. Replica placement for high availability in distributed stream processing systems. In *DEBS*, pp. 181–192, 2008.

[9] *The ns-3 network simulator*. http://www.nsnam.org/.

[10] *RFC896 Congestion control in IP/TCP internetworks*. http://www.faqs.org/rfcs/rfc896.html.