分散ストリーム処理システムにおける高信頼化方式の提案

塩川　浩昭[†]　北川　博之[†,††]　川島　英之[†,††]　渡辺　陽介[†††]

A High Availability Scheme for Distributed Stream Processing

Hiroaki SHIOKAWA[†], Hiroyuki KITAGAWA[†,††], Hideyuki KAWASHIMA[†,††], and Yosuke WATANABE[†††]

あらまし　近年，実世界から得られるストリームデータに対する問合せ要求が増大し，それらを実現するストリーム処理システムが研究開発されてきた．そして，地理的に離れた情報源の統合や負荷分散を実現させるために，ストリーム処理システムを分散配置させて利用する分散ストリーム処理システムが注目されている．分散ストリーム処理システムでは，複数のストリーム処理システムの入力と出力をつなぎ合わせることにより分散環境を構築するため，分散配置されたノードが一つでも停止してしまうと，システム全体が停止してしまうという問題がある．この問題を解決するため，本論文では，分散環境において高信頼化を実現する Semi-Active Standby 方式を提案する．本方式は，既存方式である Active Standby 方式，Upstream Backup 方式を統一化した方式であり，高信頼化におけるリカバリ時間とバンド幅使用率の調整を可能にする．本論文では，Semi-Active Standby 方式の動作特性の詳細について述べる．また，我々が開発したプロトタイプシステムで行った評価実験について述べる．

キーワード　ストリーム処理，高信頼化，分散コンピューティング

1. まえがき

近年，実世界センサデータや株取引データなどの絶えず情報を配信するストリーム型情報源が増大し，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，次々と配信される株取引データを監視し，株価が一定値以上であったら通知する，などは典型的な要求の一つである．このようなストリームデータに対する処理要求を実現するための基盤システムとして，ストリーム処理システム [1]～[3] が注目されており，我々の研究グループも StreamSpinner [4] というストリーム処理システムを開発している．

一方，地理的に離れた場所から提供される情報源を扱う場合や，ストリーム処理を行う特定のノードの負荷分散を行う場合には，ネットワーク上に分散した計算資源やネットワーク帯域・遅延を考慮した分散ストリーム処理システム [5], [6] を構築する必要がある．このような分散環境では，複数のノードが互いに連携し，情報源からユーザまでストリームデータを中継していくことで処理を実現する．そのため，分散環境を構成する一部のノードが故障などにより停止してしまうと，データが中継されずシステム全体が停止してしまうという問題がある．加えて，システムが停止している間もストリームデータは到着し続けるため，有限の計算資源を使い果たし大量のストリームデータが失われることになる．このように，分散ストリーム処理システムでは，ノードの停止によりシステム全体が停止しない，かつ，ノードの停止によりデータの欠損が生じないという性質を満たす高信頼化方式が求められている．

これまで，このような分散環境に対して，いくつかの高信頼化方式が提案されている [7]～[11]．代表的なものに，Hwang [8] らの研究による Active Standby 方式，Upstream Backup 方式，Passive Standby 方式がある．これらの高信頼化方式の共通アイデアは，分散環境を構成する各ノード上のストリーム処理システムに対してセカンダリと呼ばれるバックアップ用のストリーム処理システムを別ノード上に用意して事前

[†] 筑波大学大学院システム情報工学研究科，つくば市
Graduate School of System and Information Engineering, University of Tsukuba, Tsukuba-shi, 305–8573 Japan
[††] 筑波大学計算科学研究センター，つくば市
Center for Computational Sciences, University of Tsukuba, Tsukuba-shi, 305–8573 Japan
[†††] 東京工業大学学術国際情報センター，東京都
Global Scientific Information and Computing Center, Tokyo Institute of Technology, Tokyo, 152–8550 Japan

にバックアップデータを保持しておき，あるノードが停止した場合にはバックアップ用のノードが停止を検知して問合せ処理を引き継ぐというものである．事前に他のノードへとバックアップデータを送信しておくことで，システム停止に伴うデータの欠損を防ぐ仕組みをもつ．これにより，部分的な障害に対して問合せ処理が停止することなく，分散環境におけるシステムの高信頼化を実現している．

これらの高信頼化方式は，バンド幅オーバヘッド，リカバリ時間というトレードオフの関係にある二つの指標を用いて評価される [8]．バンド幅オーバヘッドは高信頼化におけるバックアップデータ通信に必要なバンド幅，リカバリ時間は障害回復に要する時間のことである．Active Standby 方式では，あるノードが次のノードへとデータを送信する際に，それと並列してバックアップ用のノードにも同じデータを送信するという方式をとっている．また，Passive Standby 方式では，問合せ処理を行っているノードの内部状態のスナップショットをバックアップ用のノードへと送信し続けるといった方式をとっている．これにより，Active Standby 方式と Passive Standby 方式ではバンド幅オーバヘッドを大きく犠牲にするが，リカバリ時間を小さくすることができるという特徴を有している．これに対して，Upstream Backup 方式では，あるノードに関するバックアップデータの蓄積は，そのノードよりも前に問合せ処理を実行するノードがすべて行い，ノードが停止するまではバックアップ用のノードへ一切通信を行わない．そのため，Upstream Backup 方式では，リカバリ時間を大きく犠牲にするが，バンド幅オーバヘッドを小さくできるという特徴を有している．このように，既存方式では，バンド幅オーバヘッド，リカバリ時間の片方を改善するために，もう一方を大幅に犠牲にするといったアプローチがとられている．

一般に，利用者が分散ストリーム処理システムを構築する際には，ネットワークの利用可能なバンド幅に上限が生じる場合やシステムの停止時間の上限が存在するなど，バンド幅オーバヘッドやリカバリ時間に対して制約が付くことは少なくない．リカバリ時間を一定時間以下にしつつ，バンド幅オーバヘッドはできる限り小さくしたいなどは，現実に起こり得る代表的な制約の一つである．ゆえに，現実的な環境において分散ストリーム処理システムの高信頼化を実現するためには，高信頼化方式が両指標を柔軟に調整できる必要がある．

そこで本論文では，Active Standby 方式と Upstream Backup 方式を一般化した高信頼化方式 Semi-Active Standby 方式を提案する．提案方式 Semi-Active Standby 方式は利用者の要求に合わせてバンド幅オーバヘッドとリカバリ時間を柔軟に調整することが可能である．本論文ではこれに加え，Semi-Active Standby 方式においてバックアップデータの通信に圧縮処理を用いることで，送信データ量の削減を可能にする方式についても考察する．更に，分散ストリーム処理システムのプロトタイプシステムを用いて提案方式の特性を評価し，提案方式の有効性を示す．

本論文では，**2.** にて，関連研究について説明する．**3.** では提案手法である Semi-Active Standby 方式について説明し，**4.** でその実験結果を示す．そして **5.** で，本論文における議論，**6.** で関連研究を示し，**7.** でまとめと今後の課題について述べる．

2. 従来手法

本章では，これまでに研究された分散ストリームシステムにおける高信頼化方式について述べる．

2.1 高信頼化方式の分類

既存の高信頼化方式は，分散ストリーム処理システムを構成する各マシン上のストリーム処理システムに対して，バックアップ用のストリーム処理システムを別マシン上に用意し，マシンが停止した場合にはバックアップ用のマシンが問合せ処理を引き継ぐというものである．Hwang らの研究 [8] では，高信頼化方式を以下のように分類している．ここでは，非故障時に問合せ処理を行うストリーム処理システム（ノード）をプライマリ，プライマリの停止時に問合せ処理を引き継ぐストリーム処理システム（ノード）をセカンダリと呼ぶ．また，ストリーム処理システム（ノード）にデータを送信するストリーム処理システム（ノード）を上流側ストリーム処理システム（ノード），処理結果を受信するストリーム処理システム（ノード）を下流側ストリーム処理システム（ノード）と呼ぶ．(図 1)

2.1.1 Active Standby 方式

Active Standby 方式 [1], [8], [9] では，上流側ストリーム処理システムからプライマリが受信するデータをすべてセカンダリも受け取り，セカンダリもプライマリに並列してプライマリと同じ問合せ処理を行う．プライマリが生存している場合には，セカンダリは処理結果を出力しない．プライマリが停止した場合には，

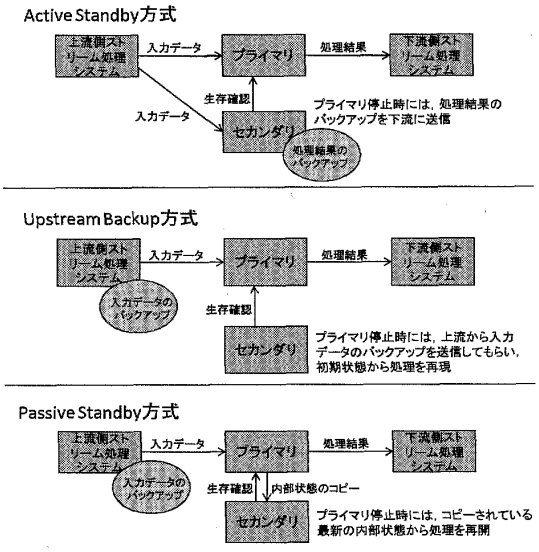


図 1　既存の高信頼化方式
Fig. 1　Existing high availability scheme.

セカンダリが自身の出力を下流側ストリーム処理システムに接続し処理結果の配信を行うことにより処理を引き継ぐ．

2.1.2　Upstream Backup 方式

Upstream Backup 方式 [6], [8] は，プライマリが生存している場合には，セカンダリは待機しながらプライマリの生存確認のみ行い，内部状態のコピーやプライマリと並列した問合せ処理は行わない．その代わりに，プライマリの上流側ストリーム処理システムは，プライマリへの入力データをバックアップしておく．プライマリが停止した場合は，セカンダリはデータをもたない初期状態から問合せ処理を開始し，上流側ストリーム処理システムにバックアップされているすべてのデータを再送してもらい，処理することで処理を引き継ぐ．

2.1.3　Passive Standby 方式

Passive Standby 方式 [8], [10] は，プライマリがメインで問合せ処理を行いながら，定期的にプライマリの内部状態をセカンダリへコピーする方式である．プライマリが生存している間は，セカンダリは問合せ処理は行わず，プライマリの生存確認を定期的に行う．プライマリが停止した場合，セカンダリは最新のコピーを用いて処理を再開する．このとき，コピー後にプライマリが処理したデータはセカンダリのコピーに反映されていないため，そのまま再開しては処理データが失われる．この損失を防ぐため，セカンダリはそれらのデータを処理再開時に上流側ストリーム処理システムから再送させる．

Passive Standby 方式は，Active Standby 方式・Upstream Backup 方式とは異なり，各プライマリの内部状態をセカンダリにコピーし復元するための特別な仕組みが必要となる．本論文では，そのような機構を用いない高信頼化方式のみを議論の対象とすることとし，以後，Passive Standby 方式を比較の対象としない．

2.2　特性比較

Hwang らの研究では，既存方式の特性比較のために，バンド幅オーバヘッドとリカバリ時間という二つの指標を用いている [8]．

[定義 1]　バンド幅オーバヘッド

分散ストリーム処理における通信のうち，問合せ処理のための送信（プライマリ・プライマリ間）に費やした総バンド幅を BW_{main}，バックアップデータの送信（プライマリ・セカンダリ間）に費やした総バンド幅を BW_{backup} とすると，バンド幅オーバヘッド BW は次のように定義される．

$$BW = BW_{backup}/BW_{main} \tag{1}$$

[定義 2]　リカバリ時間

プライマリ停止時に，セカンダリがプライマリ停止直前の状態を再現するまでに要する時間をリカバリ時間と定義する．

バンド幅オーバヘッドとリカバリ時間は互いにトレードオフの関係となる．この 2 指標を用いて既存方式の特性を比較する．Active Standby 方式では，プライマリへのデータ送信時は必ずセカンダリにもデータを送信することからバンド幅のオーバヘッドが，高信頼化方式を用いない場合の 2 倍と大きな値となる．しかし，セカンダリが常にプライマリと同じ状態で待機できるため，リカバリ時間が Upstream Backup 方式よりも小さくなる．Upstream Backup 方式では，障害発生時のみデータの送信を行うため，バンド幅オーバヘッドが 0 になる．しかし，セカンダリは初期状態から問合せ処理を開始するため，リカバリ時間が Active Standby 方式よりも大きくなる．

2.3　問題提起

既存方式は，バンド幅オーバヘッド，リカバリ時間のどちらか一方を犠牲にしなければ，高信頼化を保証することはできない方式であり，バンド幅オーバヘッドとリカバリ時間の調整が難しい．しかし，実際に利用者が分散環境を構築する際には，使用するマシンの性能やネットワークの帯域・遅延，アプリケーション

性質などから，バンド幅オーバヘッドやリカバリ時間に制約がつくことは少なくなく，利用環境に合わせたバンド幅オーバヘッド，リカバリ時間の調整は不可欠である．したがって，そのような調節を行える方式が求められている．

3. 提案手法 Semi-Active Standby 方式

本章では，前章で述べた従来手法の問題点を解決する提案手法 Semi-Active Standby 方式について述べる．

3.1 Semi-Active Standby 方式の概要

Active Standby 方式と Upstream Backup 方式に着目する．従来研究 [8] では，動作性質の違いから，両方式は全く異なる手法であるとされている．しかし，両方式における動作性質の違いは，非障害時にバックアップデータをどれだけ送信するかに依存しており，本質的に両方式は，上流側ノードがセカンダリへバックアップデータを送信するという共通のスキームをもつ方式である．そのため，バンド幅オーバヘッド，リカバリ時間の調整を可能にするためには，両方式のもつ共通スキームにおいて，バックアップデータ送信のタイミングを調節することが重要であると考えられる．

そこで本論文では，上流側ノードとセカンダリとの間で，一定間隔でバックアップデータを送信するアプローチをとる．定期的に送信するバックアップデータのデータ量を調整可能にすることで，バックアップデータの所在を調整可能にする．以下では，送信するデータのまとまりをバッチ，データの量をバッチサイズと呼ぶ．また，提案手法 Semi-Active Standby 方式は，バッチに対して圧縮処理を行うことで，送信するデータ量を削減し，バンド幅オーバヘッドを更に小さくすることができる．そこで本論文では，圧縮を用いない Semi-Active Standby 方式と圧縮を用いる圧縮 Semi-Active Standby 方式の二つを提案する．

3.2 システムモデル

Semi-Active Standby 方式では，従来研究 [8] のシステムモデルと同様に，分散環境を構築する各ノードに対してバックアップ用のノード（セカンダリ）を用意する．図 2 で，ノード N_u，N，N_d という三つのノードに分散したシステムモデルの例を表す．ノード N に対して，上流側にあるノードを N_u，下流側にあるノードを N_d とする．ここでは，N_u から N に送信されるストリームデータの流れを I_1，I_2 とし，N から N_d を O としている．また本論文では，従来研究 [8] と同様に，通信によるデータ順序の入換わりや，データの欠落が生じないことを前提とする．

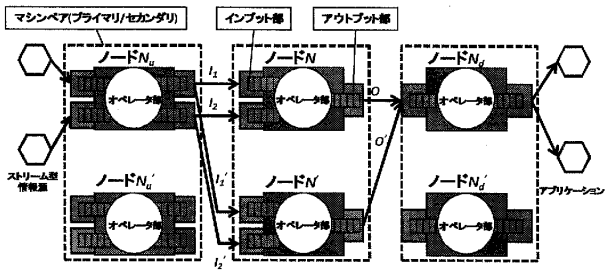


図 2 システムモデル（3 ノード）
Fig. 2 System model (3 nodes).

このシステムモデルでは，ノード N_u，N，N_d はプライマリで，ノード N_u'，N'，N_d' をセカンダリとする．セカンダリはプライマリに対し，定期的パケットを送信することで生存確認を行い，プライマリの停止を検知した場合には，セカンダリが処理の引継ぎを行う（図 2 では N が故障した場合，I_1，I_2，O をそれぞれ I_1'，I_2'，O' に切り換える）．

一方，本論文では従来研究 [8] とは異なり，ノード上のストリーム処理システムの構成要素を，インプット部，オペレータ部，アウトプット部の三つに抽象化して扱う．インプット部は，上流側ストリーム処理システムから配信されるデータを受け取りタプルに変換してインプットキューに蓄える．インプットキューに蓄えられたデータは随時，キューの先頭からオペレータ部へと入力される．また，インプット部は定期的に，上流側にあるプライマリ・セカンダリそれぞれのアウトプット部に対して Level-0-Ack を返す．Level-0-Ack は，インプット部が最後のどのデータを受け取ったかという情報を示す [8]．インプット部からオペレータ部へと入力されたデータは連続的問合せ [12] によって処理され，アウトプット部へと出力される．アウトプット部は，オペレータ部から受け取った処理結果を下流側ノードに送信し，バックアップをアウトプットキューに蓄える．このバックアップデータは下流側セカンダリから要求があったときのみ，アウトプットキューの先頭から送信される．また，プライマリのアウトプット部では，下流側ノードのインプット部から Level-0-Ack を受信するごとに Level-1-Ack を生成する．Level-1-Ack とは，下流側ノードのインプット部から受信した Level-0-Ack の情報を更に一段上流側ノードのアウトプット部に対して通知するものである [8]．

処理 1:Semi-Active Standby 方式（プライマリ）	
1:	if 上流側プライマリからデータを受信 then
2:	インプットキューに挿入
3:	if インプットキューにデータが存在 then
4:	データを取り出しオペレータ部へ渡す
5:	オペレータ部で処理
6:	処理結果を複製（処理結果 a，b を作成）
7:	処理結果 a をアウトプットキューへ挿入
8:	処理結果 b を下流側プライマリに送信
9:	if アウトプットキューのサイズ=バッチサイズ
10:	アウトプットキューの全データをバッチ化
11:	バッチを下流側セカンダリに送信
12:	if 現在時刻 - 前回 Level-0-Ack 送信時刻 ≥ Ack 送信間隔 then
13:	Level-0-Ack を上流側プライマリとセカンダリへ送信
14:	if Level-0-Ack を受信 then
15:	上流側プライマリに Level-1-Ack を送信
16:	if Level-1-Ack を受信 then
17:	Level-1-Ack よりも古いデータをアウトプットキューから削除
18:	if バックアップデータ送信要求受信 then
19:	アウトプットキューにあるデータをすべて下流側セカンダリに送信

処理 2:Semi-Active Standby 方式（セカンダリ）	
1:	if 現在時刻 - 前回の生存確認時刻 ≥ 生存確認間隔 then
2:	生存確認
3:	if プライマリが生存 then
4:	上流プライマリノードからバッチを受信
5:	バッチ内のデータをインプットキューに挿入
6:	if インプットキューにデータが存在 then
7:	データを取り出しオペレータ部へ渡す
8:	オペレータ部で処理
9:	処理結果をアウトプットキューに挿入
10:	if Level-0-Ack を受信 then
11:	Level-0-Ack よりも古いデータをインプットキュー・アウトプットキューから削除
12:	else if プライマリが停止 then
13:	下流プライマリノードと接続
14:	上流プライマリノードと接続
15:	if アウトプットキューにデータが存在 then
16:	アウトプットキューにあるデータをすべて下流のノードに送信
17:	上流プライマリノードにデータ再送要求送信
18:	バックアップデータ受信
19:	バックアップデータをインプットキューに挿入
20:	以後プライマリとして動作

3.3 Semi-Active Standby 方式

Semi-Active Standby 方式では，分散環境を構築する各プライマリ，セカンダリに対して，それぞれ処理 1，処理 2 に示した処理を適応する．本節では特に，図 2 において，ノード N の高信頼化を行う場合に着目し，本方式の仕組みをプライマリ N_u における処理（処理 1）とセカンダリ N' における処理（処理 2）に分けて説明する．

[プライマリ N_u における処理]

まず最初に，利用者がプライマリ N_u のアウトプット部に対してバッチサイズを指定する．バッチサイズを指定することにより，セカンダリ N' に対して 1 度に送信するバックアップデータのデータ量を決める．バッチサイズ指定後ストリームデータを受信した場合，順次インプットキューへ挿入する．インプットキューに挿入されたデータは逐次オペレータ部で処理され，処理結果を生成する．処理結果を生成した後，プライマリ N_u ではその処理結果の複製を行う．ここでは，複製した処理結果をそれぞれ処理結果 a，b とする．処理結果の複製を行った後，処理結果 a を自身のアウトプットキューに挿入し，処理結果 b を下流のプライマリ N へと送信する．このとき，自身のアウトプットキューのサイズがバッチサイズへと達していた場合には，アウトプットキュー内の全データをバッチとして下流のセカンダリ N' へと送信する．

プライマリ N_u よりも上流側にプライマリ・セカンダリが存在する場合，前述の処理の流れに並行して，プライマリ N_u は定期的に Level-0-Ack を生成し，上流側のプライマリと上流側のセカンダリへと送信する．これと同様に，下流側のプライマリ N から Level-0-Ack を受信した場合には，Level-0-Ack をもとに Level-1-Ack を生成し，上流側のプライマリへと送信する．以上の処理に加えてプライマリ N_u では，下流側のプライマリ N から Level-1-Ack を受信した場合，Level-1-Ack が示すデータよりも古いデータをアウトプットキューから削除する．

[セカンダリ N' における処理]

セカンダリ N' では，定期的にプライマリ N に対してパケットを送信することで，プライマリ N の生存確認を行う．プライマリ N が生存している場合，上流側のプライマリ N_u からバッチが送信されるごとに，受信したバッチをインプットキューに挿入し，順次オペレータ部で処理する．処理結果を生成した後，その処理結果をアウトプットキューに挿入する．

前述の処理の流れに並行して，セカンダリ N' では Level-0-Ack の受信と不要なバックアップデータの削除

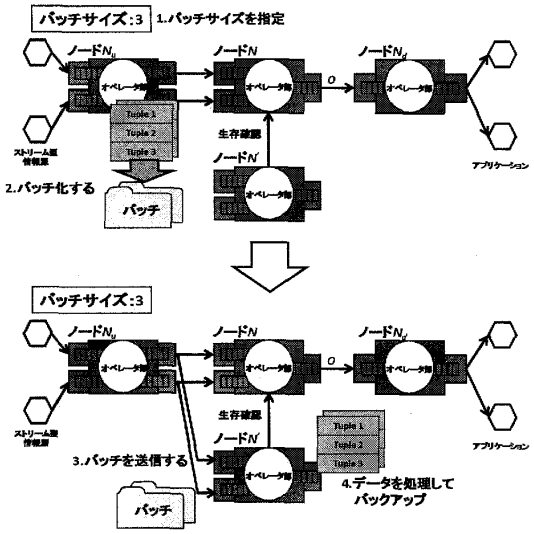


図 3 Semi-Active Standby 方式におけるデータのバックアップ
Fig. 3 Preservation for Semi-Active Standby.

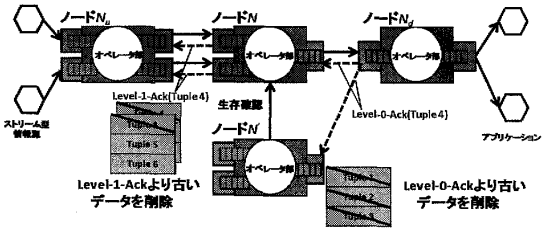


図 4 Semi-Active Standby 方式におけるバックアップデータの削除
Fig. 4 Elimination of backup data for Semi-Active Standby.

処理を行う．セカンダリ N' が下流側のプライマリ N_d から Level-0-Ack を受信した場合には，Level-0-Ack が示すデータよりも古いデータをインプットキュー及び，アウトプットキューから削除する．

[動作特性]

本方式では，バッチサイズを設けることにより，セカンダリ N' へと送信するデータ量を調整する．バッチサイズを小さくしていくと，バックアップデータの送信間隔は次第に短くなり，頻繁にデータをセカンダリ N' へと送信するようになる．これに対して，バッチサイズを大きくしていくと，バックアップは送信されず，プライマリ N_u に保持されるようになる．バッチサイズを変更することで，本方式では，既存方式 Active Standby 方式・Upstream Backup 方式の動作をも実現することが可能である．本方式では，バッチサイズを 1 とするときには Active Standby 方式と同様の動作をする．また，バッチサイズを理論上無限大にすることにより Upstream Backup 方式と同様の動作をする．既存方式 Active Standby 方式・Upstream Backup 方式は，本方式の特殊な場合であるとみなすことができる．

[動作例 1] 本方式の動作例を図 3，図 4 に示す．この動作例ではバッチサイズを 3 としている．図 3 では，ノード N_u のアウトプットキューのサイズが 3 となっているため，アウトプットキュー内部のデータをバッチ化し，まとめセカンダリへと送信している．バッチを受信したセカンダリは，オペレータ部で処理を行い，

処理 3:圧縮 Semi-Active Standby 方式（プライマリ）	
1:	if 上流側プライマリからデータを受信 then
2:	インプットキューに挿入
3:	if インプットキューにデータが存在 then
4:	データを取り出しオペレータ部へ渡す
5:	オペレータ部で処理
6:	処理結果を複製（処理結果 a，b を作成）
7:	処理結果 a をアウトプットキューへ挿入
8:	処理結果 b を下流側プライマリに送信
9:	if アウトプットキューのサイズ=バッチサイズ
10:	アウトプットキューの全データをバッチ化
11:	バッチを圧縮
12:	圧縮したバッチを下流側セカンダリに送信
13:	if 現在時刻 - 前回 Level-0-Ack 送信時刻 ≥ Ack 送信間隔 then
14:	Level-0-Ack を上流側プライマリとセカンダリへ送信
15:	if Level-0-Ack を受信 then
16:	上流側プライマリに Level-1-Ack を送信
17:	if Level-1-Ack を受信 then
18:	Level-1-Ack よりも古いデータをアウトプットキューから削除
19:	if バックアップデータ送信要求受信 then
20:	アウトプットキューにあるデータをすべて下流側セカンダリに送信

アウトプットキューに結果を挿入している．図 4 では，ノード N_d が上流側ノード N に対して Tuple 4 に関する Level-0-Ack を送信している．Level-0-Ack を受信したノード N は，ノード N_u に対して Level-1-Ack を送信する．ノード N' では Level-0-Ack に示されるデータ Tuple 4 よりも前のデータをすべて削除している．また，Level-1-Ack を受信したノード N_u でも，同様にして，Level-1-Ack に示されるデータ Tuple 4 よりも前のデータを削除している．

3.4 圧縮 Semi-Active Standby 方式

圧縮 Semi-Active Standby 方式のプライマリにおける処理を処理 3，セカンダリにおける処理を処理 4 に示す．

処理 4:圧縮 Semi-Active Standby 方式（セカンダリ）	
1:	if 現在時刻 - 前回の生存確認時刻 ≥ 生存確認間隔 then
2:	生存確認
3:	if プライマリが生存 then
4:	上流側プライマリから圧縮したバッチを受信
5:	圧縮したバッチを解凍
6:	解凍したバッチ内のデータをインプットキューに挿入
7:	if インプットキューにデータが存在 then
8:	データを取り出しオペレータ部へ渡す
9:	オペレータ部で処理
10:	処理結果をアウトプットキューに挿入
11:	if Level-0-Ack を受信 then
12:	Level-0-Ack よりも古いデータをインプットキュー・アウトプットキューから削除
13:	else if プライマリが停止 then
14:	下流側プライマリと接続
15:	上流側プライマリと接続
16:	if アウトプットキューにデータが存在 then
17:	アウトプットキューにあるデータをすべて下流側プライマリに送信
18:	上流側プライマリにバックアップデータ再送要求送信
19:	バックアップデータ受信
20:	バックアップデータをインプットキューに挿入
21:	以後プライマリとして動作

圧縮 Semi-Active Standby 方式のプライマリでは，処理 3 の 11～12 行目以外は Semi-Active Standby 方式と同様の動作を行う．本方式では，バッチを送信するまえに，バッチに対して圧縮を行う．

同様にして，圧縮 Semi-Active Standby 方式のセカンダリでは，処理 4 の 4～6 行目以外は Semi-Active Standby 方式と同様の動作を行う．本方式は，圧縮されたバッチを受信後，解凍を行い，インプットキューへと挿入する．

[動作例 2] 本方式の動作例を図 5 に示す．図 5 では，バッチサイズを 3 としている．ノード N_u のアウトプットキューのサイズがバッチサイズと等しくなり，バッチが生成される．その後，生成されたバッチに対して圧縮を行う．そして，圧縮したバッチをノード N' に送信し，N' で解凍する．解凍されたバッチは，インプットキューに挿入され，処理される．最後に処理結果がアウトプットキューに挿入される．

3.5 リカバリ処理

処理 1，2 や処理 3，4 で示されているように，Semi-Active Standby 方式では，プライマリの障害を発見した場合，セカンダリはまず上流側・下流側ノードと接続を行う．そして，セカンダリのアウトプットキュー内のデータをすべて下流側ノードへ送信し，上流側プライマリのアウトプットキュー内のデータをすべて再送してもらうことでリカバリ処理を行う．

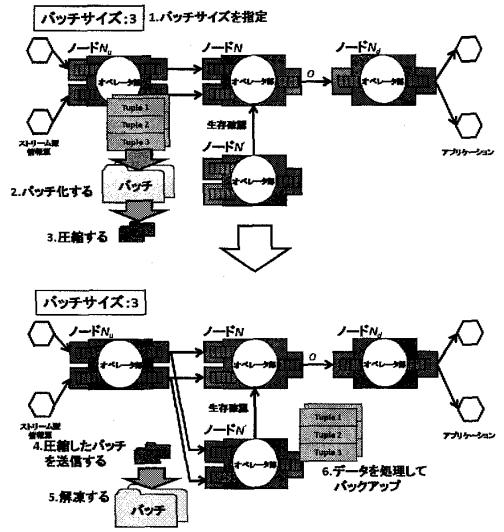


図 5 圧縮 Semi-Active Standby 方式におけるデータのバックアップ
Fig. 5 Preservation for Semi-Active Standby with compression.

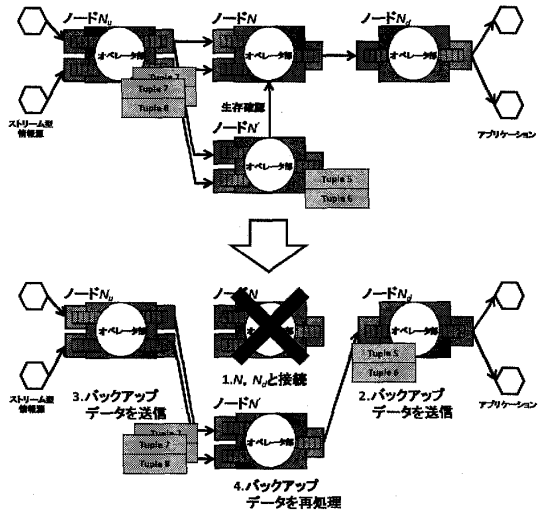


図 6 Semi-Active Standby 方式におけるリカバリ処理
Fig. 6 Data recovering for Semi-Active Standby.

[動作例 3] リカバリ処理の動作例を図 6 に示す．ノード N' がノード N の障害を発見し，上流側・下流側ノードと接続する．そして，ノード N' のアウトプットキュー内のデータ Tuple 5，6 をノード N_d へ送信し，ノード N_u のアウトプットキュー内のデータ Tuple 7，8 をノード N' に送信している．

4. 評価実験

本章ではプロトタイプシステムを用いたSemi-Active Standby方式の評価実験について述べる．

4.1 プロトタイプシステムの実装

本論文では，評価実験のために分散ストリーム処理システムのプロトタイプシステムを実装した．本システムは，処理モデルとしてCQLモデル[12]を採用しており，選択演算と射影演算，集約演算が実行可能である．加えて本システムでは，高信頼化方式として，Semi-Active Standby方式，圧縮Semi-Active Standby方式，Active Standby方式，Upstream Backup方式を有する．各高信頼化手法の通信プロトコルにはTCPを用いている．

実装にはJava1.6.0_13を用いており，プログラムの総行数は3000行程度である．各高信頼化方式については，Semi-Active Standby方式は49行，圧縮Semi-Active Standby方式は68行，Active Standby方式は40行，Upstream Backup方式は32行となっている．ただし，通信処理部分は行数に含まれていない．

4.2 実験環境

本論文で行った評価実験環境を図7に示す．本論文では，**4.1**で述べたプロトタイプシステムを4台（node-1～node-4）のマシンに分散配置した高信頼化環境を構築した．プロトタイプシステムが動作するマシンの性能はそれぞれOS：Linux2.6.22，CPU：Xeon2.4 GHz，Memory：2 GByteである．各マシンとも10 M/100 Mスイッチングハブに接続されており，互いに通信可能である．本実験ではnode-2がfail stopする場合を想定する．node-4はnode-2に対して250 ms間隔で生存確認用のパケットを送信し，node-2の生存を確認する．node-4がnode-2の停止を確認した場合には，障害回復処理が実行される．本実験では，各ノードが送信するLevel-0-Ackの送信間隔を250 msとした．また，node-1～node-4の各マシンはそれぞれ図2におけるN_u, N, N_d, N'に該当する．

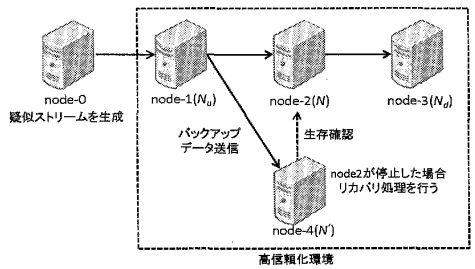


図7 実験環境
Fig. 7 Experimental environment.

図7のnode-0は，情報源として擬似ストリームデータを生成し，node-1へ送信する．node-0で生成される擬似ストリームデータは，実際のセンサデバイスから得られるストリームデータを想定し，Stream (id, timestamp, temperature, illuminance, x-acceleration, y-acceleration, z-acceleration) というスキーマをもつデータとした．idはタプルが生成された順番，timestampはタプルが生成された時刻を示すLong型の属性，temperatureは温度，illuminanceは光度，x-accelerationはx軸の加速度，y-accelerationはy軸の加速度，z-accelerationはz軸の加速度を示すDouble型の属性である．1タプル当りのデータサイズは56 Byte，データレートは1000（tuple/秒）である．

4.3 実験1：基本性能評価

提案手法Semi-Active Standby方式（SAS：Semi-Active Standby）及び，圧縮Semi-Active Standby方式（SASC：Semi-Active Standby with Compression）の性能評価を示す．本実験では，図7のnode-1～node-4に選択率1.0，window幅100 tuple，slide幅10 tupleの選択演算子を配置した．本実験では，ストリームデータ配信開始20秒後にnode-2を停止させたときのデータを測定する．この操作を各手法とも10回試行し，その平均値を実験結果とした．本実験では，Active Standby方式（AS），Upstream Backup方式（UB）に対しても同様の条件で実験を行い，比較を行った．なお，実験に使用したバッチサイズは1と20から500まで20ずつサイズを変化させた，計26種類である．

4.3.1 提案方式と従来方式の比較

Semi-Active Stadby方式と従来方式の性能比較を行う．本実験では，node-2におけるバンド幅オーバヘッド，リカバリ時間，スループット，CPU使用率を測定した．バンド幅オーバヘッド及びリカバリ時間は定義1，定義2に従う．具体的には，本実験では，バンド幅オーバヘッドをnode-1・node-2間の使用バンド幅に対するnode-1・node-4間の使用バンド幅の割合とし，リカバリ時間をnode-2停止時におけるnode-2の内部状態をnode-4が再現するまでに要する時間とした．スループットについては，node-1とnode-2の間及びnode-1とnode-4の間で1秒当りに発生する

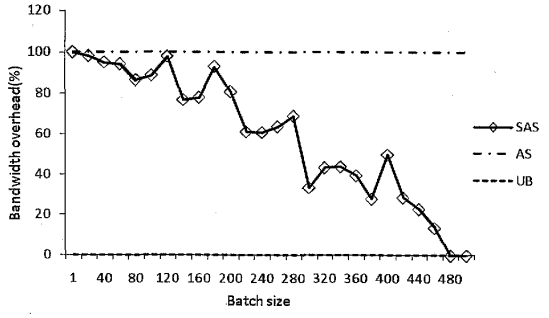


図 8　バンド幅オーバヘッドの比較
Fig. 8　A comparison of the bandwidth overhead among SAS, AS, and UB.

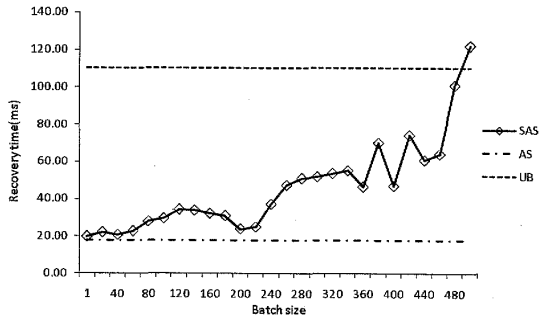


図 9　リカバリ時間の比較
Fig. 9　A comparison of the recovery time among SAS, AS, and UB.

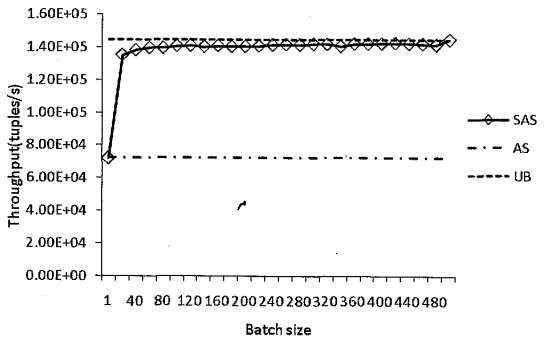


図 10　スループットの比較
Fig. 10　A comparison of the throughput among SAS, AS, and UB.

RTT（Round Trip Time）を測定し，その逆数をとることで算出した．CPU 使用率については，java 測定ツールである jconsole [13] を用いて測定した．

[実験結果]

実験結果を図 8，図 9，図 10，図 11 に示す．図 8，図 9 は，Semi-Active Standby 方式においてバッチサイズを変化させたときの，バンド幅オーバヘッド並びにリカバリ時間の推移である．この結果より，Semi-Active Standby 方式はバッチサイズを大きくなるに伴い，バンド幅オーバヘッドを減少させ，リカバリ時間を増大させていることが分かる．また，バッチサイズが 1 のときには Active Standby 方式とほぼ同様の実験結果を示し，バッチサイズを大きくするにつれて，Upstream Backup 方式の実験結果に近づいていることが分かる．バッチサイズが 1 のとき，Semi-Active Standby 方式は node-1 のアウトプットキューに存在しているすべてのデータを node-4 へと送信する．そのため，バンド幅オーバヘッドが Active Standby 方式と同じだけ必要となるが，node-2 と同じ状態を常に保ち続けることができるため，リカバリ時間が小さくなる．これに対して，バッチサイズを大きくした場合，node-2 はアウトプットキューにバッチサイズ分のデータが蓄積されるまで，バックアップデータの送信を待たなくてはならない．また，データの蓄積を待っている間に Level-1-Ack を受信した場合，アウトプットキューに蓄積されたデータは削除されるため，バックアップデータの送信待機時間が更に長くなることになる．これにより，node-1 のバックアップデータを送信する機会が減少し，バンド幅オーバヘッドが小さくなることになる．しかし，バッチサイズを大きくしたことにより，リカバリ処理のため，node-1 からデータを再送・再処理する必要が出てくる．そのため，Upstream Backup 方式と同様にリカバリ時間が増大することになる．このようなリカバリ時間の増大に伴うデータ遅延の発生は，ストリームデータを処理するアプリケーションのリアルタイム性を損なう原因となり得る．

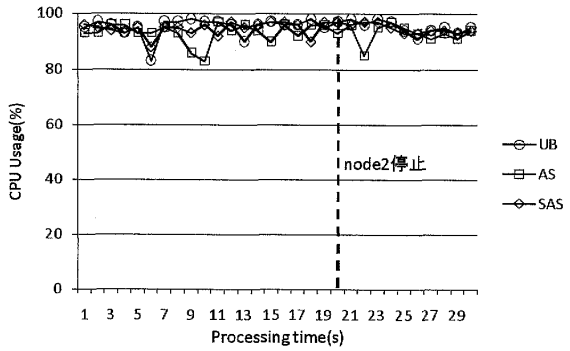


図 11　CPU 使用率の比較
Fig. 11　A comparison of the cpu usage among SAS, AS, and UB.

図 10 は各手法における node-1 のスループットを示したグラフである．この結果より，Active Standby 方式はスループットが一番小さくなり，Upstream Backup 方式は一番大きくなることが分かる．更に，Semi Acitve Standby 方式では，バッチサイズを 100 大きくしていくごとに約 10000（tuple/秒）スループッ

トが向上するという結果が得られた．Active Standby 方式では，大きなバンド幅オーバヘッドを伴いながら，常に node-2，node-4 に対してデータ送信を行う．そのため，一つのタプルがアウトプットキューにくるたびに，まず node-2 へ処理結果を送信し TCP Ack が返るのを待つ．そして，node-4 へと処理結果を送信し，TCP Ack が返るのを待つという処理を行う．これにより，node-4 に対して処理結果を送信するたびに TCP Ack の待ち時間が生じてしまい，他の方式に比べてスループットが低いことが考えられる．これは，高速データ処理が重要な要素となるストリーム処理システムにおいて，あまり望ましくない性質である．

図 11 は各手法における CPU 使用率を示したグラフである．この結果より，各手法とも CPU 使用率に大きな差がないことが分かる．しかしながら，どの手法も CPU 使用率が 90%以上と大きな値を示している．この原因は，各ストリーム処理システムのオペレータがインプットキューに入力されたデータを即時処理するために，常にインプットキューの監視を行うためであると考えられる．

本実験の結果より，Semi-Active Standby 方式はバッチサイズを変化させることで，バンド幅オーバヘッドとリカバリ時間の調整が可能であることを示した．Semi-Active Standby 方式は，実行環境に合わせた，リカバリ時間の調整やバンド幅オーバヘッドの調整によるスループットの向上が実現可能であるため，実環境上の分散ストリーム処理に置いて有用な方式であると考えられる．

4.3.2 圧縮の有無による比較

圧縮の有無による Semi-Active Standby 方式の性能比較を行う．本実験では，node-1 におけるバンド幅オーバヘッド，リカバリ時間，CPU 使用率を測定した．なお，本実験では，圧縮のために圧縮ライブラリ zlib [14] を用いた．zlib の平均データ圧縮率は 47%である．

[実験結果]

実験結果を図 12，図 13，図 14 に示す．図 12，図 13 は圧縮の有無によるバンド幅オーバヘッドとリカバリ時間の比較を示したグラフである．実験結果より，バンド幅オーバヘッドは，圧縮なしの Semi-Active Standby 方式に対して，約 47%程度にバンド幅オーバヘッドを小さくすることが分かる．この結果は，本実験に用いた圧縮ライブラリ zlib の平均データ圧縮率とほぼ同様の値を示した．これに対して，両方式におけるリカバリ時間の変動はほぼ同様の結果を示している．

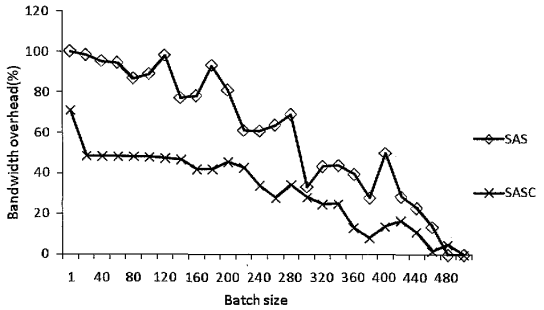


図 12 圧縮の有無によるバンド幅オーバヘッドの比較
Fig. 12 A comparison of the bandwidth overhead between SAS and SASC.

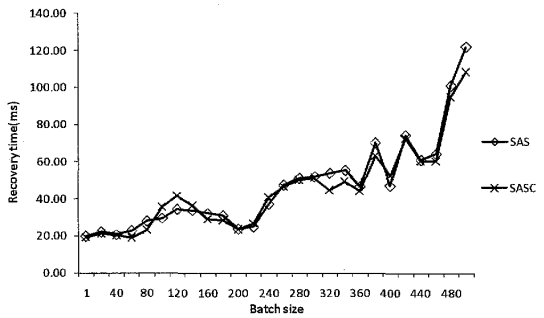


図 13 圧縮の有無によるリカバリ時間の比較
Fig. 13 A comparison of the recovery time between SAS and SASC.

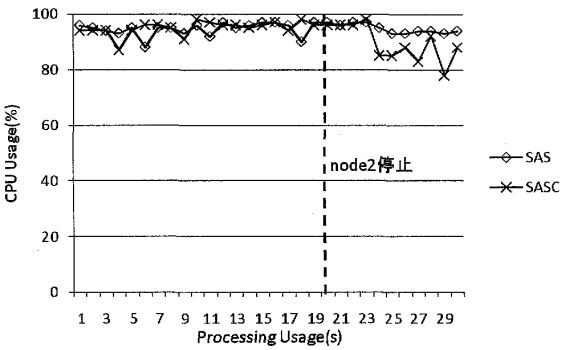


図 14 CPU 使用率の比較
Fig. 14 A comparison of the cpu usage between SAS and SASC.

図 14 は両方式における CPU 使用率を比較したグラフである．この結果より，両手法とも CPU 使用率 90%以上と大きな値を示すものの，手法間の CPU 使用率には大きな差がないことが分かる．

本実験の結果より，圧縮 Semi-Active Standby 方式はバンド幅オーバヘッドを削減することが可能であることを示した．

4.4 実験 2：実行環境の変化による性能評価

分散ストリーム処理の実行環境が変化した場合における，Semi-Active Standby 方式の性能評価を行う．

本実験で測定対象とした指標は，node-1 におけるバンド幅オーバヘッド，リカバリ時間，スループットである．本実験でも **4.3** と同様に，ストリームデータ配信開始 20 秒後，node-2 を停止させた場合の測定を 10 回試行し，その平均値を実験結果とした．比較対象は Active Standby 方式（AS），Upstream Backup 方式（UB）とし，実験に使用したバッチサイズは **4.3** と同様である．なお，圧縮 Semi-Active Standby 方式は，**4.3** で Semi-Active Standby 方式に類似した動作をすることが示されているため，本実験の比較対象とはしない．

4.4.1 オペレータの種類による比較

高信頼化対象である node-2 で実行されるオペレータの種類を変え，Semi-Active Standby 方式の性能評価を行う．本実験ではまず，図 7 の node-1 と node-3 に選択率 1.0，window 幅 100 tuple，slide 幅 10 tuple の選択演算子を配置する．そして，node-2 と node-4 に選択率 1.0，window 幅 100 tuple，slide 幅 10 tuple の選択演算子（Selection）を配置した場合と，window 幅 100 tuple，slide 幅 10 tuple の集約演算子（Aggregation）を配置した場合の 2 種類について比較実験を行った．なお，本実験では集約演算として，window 幅内にあるタプルのすべての属性を加算するという処理を行った．

[実験結果]

実験結果を図 15，図 16 に示す．図 15，図 16 は Semi-Active Standby 方式において選択演算，集約演算を実行したときのバンド幅オーバヘッドとリカバリ時間を比較を示したグラフである．実験結果より選択演算，集約演算ともにほぼ同様のバンド幅オーバヘッドを示すことが分かる．これに対して，バッチサイズが大きくなるにつれてリカバリ時間も増大していくということが分かった．また，集約演算は選択演算に対して約 3.1 倍のリカバリ時間を要すること分かる．これは，本実験で扱った集約演算の処理時間が，選択演算の約 3 倍であることが原因である．

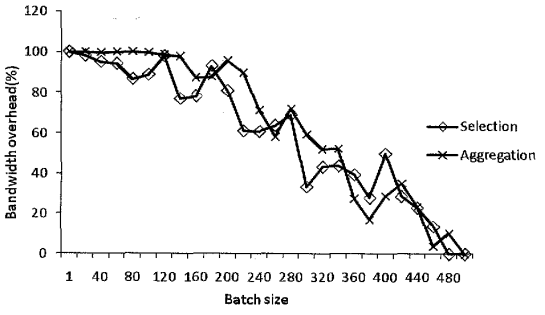


図 15　オペレータの変化によるバンド幅オーバヘッドの比較

Fig. 15　A comparison of the recovery time between selection and aggregation.

4.4.2 ネットワーク負荷の変化による比較

高信頼化環境に発生しているネットワーク負荷を変化させた場合における，Semi-Active Standby 方式の性能評価を行う．本実験では，図 7 のシステム上で，多数のクエリが実行されていることを想定し，ネットワーク全体に 40 MByte/s，60 MByte/s の負荷をかけた状況での比較実験を行った．本実験では **4.4.1** と同様に，node-1 と node-3 に選択率 1.0，window 幅 100 tuple，slide 幅 10 tuple の選択演算子，window 幅 100 tuple，slide 幅 10 tuple の集約演算子（Aggregation）を配置した．

[実験結果]

実験結果を図 17，図 18 に示す．図 17，図 18 は，ネットワークに負荷を発生させた場合における Semi-Active Standby 方式のバンド幅オーバヘッドとリカバリ時間を比較を示したグラフである．

まず，バンド幅オーバヘッドでは，各ネットワーク

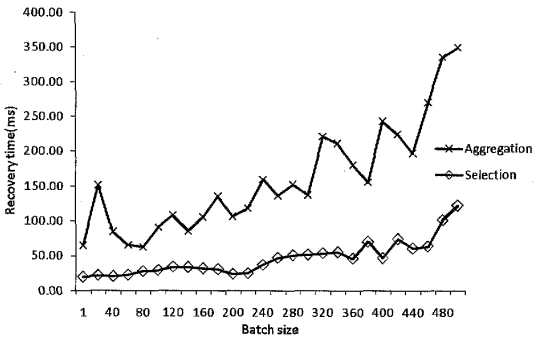


図 16　オペレータの変化によるリカバリ時間の比較

Fig. 16　A comparison of the recovery time between selection and aggregation.

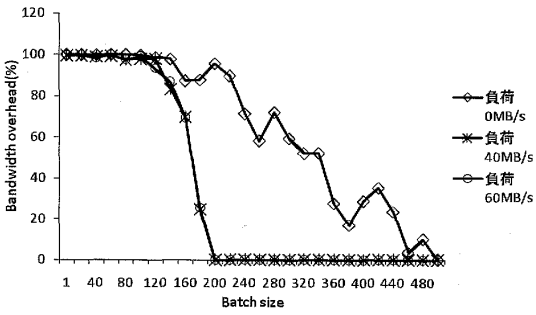


図 17　ネットワーク負荷を変化によるバンド幅オーバヘッドの比較

Fig. 17　A comparison of the recovery time among three network usages.

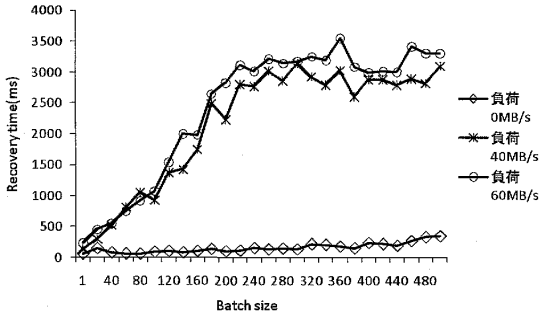


図 18 ネットワーク負荷を変化によるリカバリ時間の比較
Fig. 18 A comparison of the recovery time among three network usages.

負荷においてバッチサイズを大きくすることでバンド幅オーバヘッドが減少していくことが分かる．特に，ネットワーク負荷をかけた場合には，負荷をかけなかった場合と異なり，バッチサイズ 220 においてバンド幅オーバヘッドが 0%となっている．これは，ネットワークに負荷をかけたことで node-1 のスループットが低下したことが原因である．スループットの低下により，バッチサイズにデータ量が達するまでの時間が長くなる．そのため，Level-1-Ack によってアウトプットキューに保存されたデータが削除される可能性が高くなり，小さなバッチサイズでもバンド幅オーバヘッドが減少されることになると考えられる．

次に，リカバリ時間では，各ネットワーク負荷においてバッチサイズを大きくすることで，リカバリ時間が増大していくことが分かる．また，ネットワーク負荷が大きくなるにつれてリカバリ時間が増大していくことも分かる．負荷 60 MByte/s は負荷 40 MByte/s の約 1.2 倍，負荷 0 MByte/s の約 15 倍を示し，負荷 40 MByte/s では負荷 0 MByte/s のときの約 13.2 倍のリカバリ時間を示している．これは，リカバリ処理を行う際，ネットワーク負荷が大きいほど，node-1 のスループットが低くなることが原因であると考えられる．

5. 議　　論

データの性質が変化するようなストリームデータを扱う場合，本研究で提案している Semi-Active Standby 方式だけではバンド幅とリカバリ時間の制約を保障できない場合がある．

データの性質が変化する例として，ストリームデータのデータレートが増加する場合が挙げられる．データレートが増加する場合，事前に指定したバッチサイズ分のストリームデータがアウトプットキューにたまりやすくなる．そのため，データレートが増加する前に比べてバッチを送信する回数が多くなり，バンド幅の制約を保障できなくなる可能性が考えられる．また別の例として，1 タプル当りの処理時間をより多く必要とするようなデータが増える場合が挙げられる．このような場合，リカバリ処理時に必要となるタプルの再処理時間が大きくなる．そのため，リカバリ時間の制約を保障できなくなる可能性がある．

このように，データの性質が時々刻々と変わり得るストリームデータに対して Semi-Active Standby 方式を適応させるためには，ストリームデータの性質に応じて適応的にバッチサイズを変化させる仕組みについて考える必要がある．

6. 関 連 研 究

分散ストリーム処理システムの研究には，Borealis [5]，Medusa [6] がある．これらの分散ストリーム処理システムに関する研究では，**2.** で述べたような，Hwang ら [8] によるストリーム処理システムにおける高信頼化方式が利用されている．既に述べたように，本研究で提案した Semi-Active Standby 方式は，Active Standby 方式と Upstream Backup 方式を一般化したものである．

高信頼化対象となるサービスを複製することにより高信頼化を行う方式として，HYDRANET-FT [15]，Paxos [16]，Process-pairs [17]，Persistent queues [18] が挙げられる．これらの研究は，高信頼化対象となるサービスを複数のサーバに複製しておき，稼動中のサービスが停止してしまった場合には，サービスが複製された別のサーバに切り換えることによりサービスを継続させるという方式をとる．しかしながら，これらの高信頼化方式は，トランザクション処理とディスクの使用，クライアント・サーバ型のシステムモデルを前提としており，絶え間なく配信されるストリームデータをメモリ上で処理する並列データフロー型の分散ストリーム処理システムに適応させることは難しい．これに対して本研究では，分散ストリーム処理システムを対象とした高信頼化方式を提案している．

高信頼なアプリケーション構築のための高信頼化フレームワークを用いる方式として，Isis [19]，Horus，Ensemble [20]，Spread [21] が挙げられる．開発者がアプリケーションを開発する際にこれらのフレームワークを用いることで，高信頼なアプリケーションを

容易に構築可能にしたものである．しかし，これらの研究は，データベースやクライアント・サーバ型の分散システムを対象としており，本研究で扱う並列データフロー型のストリーム処理での適応を想定してない．

7. むすび

本研究では，上流側ノードとセカンダリノード間のバックアップデータ送信に対してバッチ処理を取り入れ，従来の高信頼化手法 Active Standby 方式と Upstream Backup 方式を統合する手法，Semi-Active Standby 方式を提案した．本論文では，バッチサイズを調整することによりリカバリ時間とバンド幅オーバヘッドの調節が可能であることを，プロトタイプシステムによる実験で示した．

また，本手法は，バッチ処理を用いたことにより，バックアップデータに対して圧縮を行うことを可能にする．本論文では，圧縮処理を用いることにより，バンド幅オーバヘッドを削減できることをプロトタイプシステムによる実験で示した．

今後の課題として，バンド幅オーバヘッドとリカバリ時間に対するコストモデルの提案や，コストモデルを用いたコスト最適化機構の構築などが挙げられる．

謝辞 本研究の一部は，科学研究費補助金（#21240005，#21013004），筑波大学 VBL 研究プロジェクト，情報処理推進機構（IPA）2009 年度下期未踏 IT 人材発掘・育成事業（未踏ユース）による．

文献

[1] S. Chandrasekaran, O. Cooper, A. Deshpande, M.J. Franklin, J.M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. Shah, "TelegraphCQ: Continuous dataflow processing for an uncertain world," Proc. CIDR, pp.269–280, 2003.

[2] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G.S. Manku, C. Olston, J. Rosenstein, and R. Varma, "Query processing, approximation, and resource management in a data stream management system," CIDR, pp.245–256, 2003.

[3] D.J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik, "Aurora: A new model and architecture for data stream management," VLDB J., vol.12, no.2, pp.120–139, 2003.

[4] StreamSpinner Project Team in University of Tsukuba, "StreamSpinner: A data stream processing system for emerging information sources," http://www.streamspinner.org/index_en.html

[5] Y. Ahmad, B. Berg, U. Cetintemel, M. Humphrey, J.-H. Hwang, A. Jhingran, A. Maskey, O. Papaemmanouil, A. Rasin, N. Tatbul, W. Xing, Y. Xing, and S. Zdonik, "Distributed operation in the borealis stream processing engine," SIGMOD '05: Proc. 2005 ACM SIGMOD International Conference on Management of Data, pp.882–884, New York, NY, USA, 2005.

[6] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing, and S. Zdonik, "Scalable distributed stream processing," CIDR 2003 — First Biennial Conference on Innovative Data Systems Research, p.23, Asilomar, CA, Jan. 2003.

[7] M.A. Shah, J.M. Hellerstein, and E. Brewer, "Highly available, fault-tolerant, parallel dataflows," SIGMOD '04: Proc. 2004 ACM SIGMOD International Conference on Management of Data, pp.827–838, New York, NY, USA, 2004.

[8] J.H. Hwang, M. Balazinska, A. Rasin, U. Çetintemel, M. Stonebraker, and S. Zdonik, "High-availability algorithms for distributed stream processing," Proc. 21st International Conference on Data Engineering 2005, ICDE 2005, pp.779–790, 2005.

[9] M. Balazinska, H. Balakrishnan, S.R. Madden, and M. Stonebraker, "Fault-tolerance in the borealis distributed stream processing system," ACM Trans. Database Syst., vol.33, no.1, pp.1–44, 2008.

[10] J.H. Hwang, Y. Xing, U. Çetintemel, and S. Zdonik, "A cooperative, self-configuring high-availability solution for stream processing," IEEE 23rd International Conference on Data Engineering 2007, ICDE 2007, pp.176–185, 2007.

[11] J.H. Hwang, U. Çetintemel, and S. Zdonik, "Fast and highly-available stream processing over wide area networks," Proc. 24th International Conference on Data Engineering (ICDE), pp.804–813, 2008.

[12] A. Arasu, S. Babu, and J. Widom, "The CQL continuous query language: Semantic foundations and query execution," VLDB J. International Journal on Very Large Data Bases, vol.15, no.2, pp.121–142, 2006.

[13] Sun Microsystems, "JavaTM プラットフォームの監視と管理," http://java.sun.com/j2se/1.5.0/ja/docs/ja/guide/management/

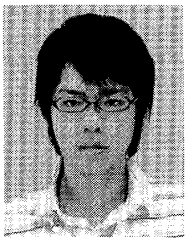
[14] G. Roelofs, "zlib: A massively spiffy yet delicately unobtrusive compression library," http://www.zlib.net/

[15] G. Shenoy, S.K. Satapati, and R. Bettati, "Hydranet-ft: Network support for dependable services," International Conference on Distributed Computing Systems, pp.699–706, 2000.

[16] B. Lampson, "The abcd's of paxos," PODC '01: Proc. Twentieth Annual ACM Symposium on Principles of Distributed Computing, p.13, New York, NY, USA, 2001.

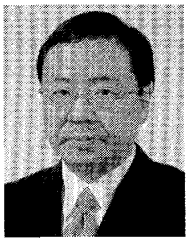
[17] J. Gray and A. Reuter, Transaction processing: Concepts and techniques (Morgan Kaufmann series in data management systems), Morgan Kaufmann, Oct. 1992.
[18] P.A. Bernstein, M. Hsu, and B. Mann, "Implementing recoverable requests using queues," SIGMOD Rec., vol.19, no.2, pp.112–122, 1990.
[19] K.P. Birman, "Isis: A system for fault-tolerant distributed computing," Technical Report, TR86-744, Cornell, Ithaca, NY, USA, 1986.
[20] K.P. Birman, R. Constable, M. Hayden, J. Hickey, C. Kreitz, R. Van Renesse, O. Rodeh, and W. Vogels, "The horus and ensemble projects: Accomplishments and limitations," Technical Report, TR99-1774, Cornell, Ithaca, NY, USA, 1999.
[21] Y. Amir and J. Stanton, "The spread wide area group communication system," Technical Report, CNDS-98-4, The Center for Networking and Distributed Systems, The Johns Hopkins University, 1998.

（平成 21 年 9 月 11 日受付，12 月 29 日再受付）



塩川　浩昭

2009 筑波大・第三学群・情報学類卒．現在，筑波大学大学院システム情報工学研究科に在学中．ストリーム処理システムにおける高信頼化に関する研究に従事．日本データベース学会学生員．

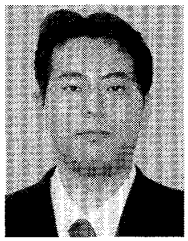


北川　博之　（正員：フェロー）

1978 東大・理・物理卒．1980 同大大学院理学系研究科修士課程了．日本電気（株）勤務の後，1988 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，並びに計算科学研究センター教授．理博．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，ACM，IEEE-CS，日本ソフトウェア科学会各会員．

川島　英之

1999 慶大・理工・電気卒．2005 同大大学院理工学研究科開放環境科学専攻後期博士課程了．同年，慶應義塾大学理工学部助手．2007 筑波大学大学院システム情報工学研究科講師，並びに計算科学研究センター講師．博士（工学）．センサデータ管理に関する研究に従事．情報処理学会，ACM 各会員．



渡辺　陽介

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了．同年，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勤務の後，2008 東京工業大学学術国際情報センター助教．博士（工学）．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．