DEIM Forum 2010 E1-5

# 分散ストリーム処理環境における適応的高信頼化手法

塩川　浩昭† 　北川　博之†,†† 　川島　英之†,††

† 筑波大学大学院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
†† 筑波大学計算科学研究センター　〒 305–8573 茨城県つくば市天王台 1–1–1
E-mail: †shion@kde.cs.tsukuba.ac.jp, ††{kitagawa,kawasima}@cs.tsukuba.ac.jp

**あらまし**　近年，センサや位置情報などの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ている．そして，そのような要求を実現するストリーム処理エンジン (SPE) がこれまで研究開発されてきた．SPE では，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，SPE を分散配置させる分散ストリーム処理エンジン (DSPE) の利用が一般的である．DSPE では，複数の SPE 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，分散環境中のノードが１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，我々が先行研究で提案した Semi-Active Standby 方式 (SAS) に関して，様々な環境下において評価・考察し，新たに適応型 Semi-Active Standby 方式 (A-SAS) を提案する．A-SAS は，ストリームデータの性質変化に対して，高信頼化によるバンド幅使用量，リカバリ時間を適応的に調整する．本稿では，A-SAS の詳細と，我々が開発したプロトタイプシステムを用いた SAS，A-SAS の評価実験結果について述べる．
**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

# An Adaptive High-Availability Method for Distributed Stream Processing Environments

Hiroaki SHIOKAWA†, Hiroyuki KITAGAWA†,††, and Hideyuki KAWASHIMA†,††

† Graduate School of Systems and Information Engineering, University of Tsukuba　Tennoudai 1–1–1, Tsukuba City, Ibaraki, 305–8573 Japan
†† Center for Computational Sciences, University of Tsukuba　Tennoudai 1–1–1, Tsukuba City, Ibaraki, 305–8573 Japan
E-mail: †shion@kde.cs.tsukuba.ac.jp, ††{kitagawa,kawasima}@cs.tsukuba.ac.jp

**Abstract**　Nowaday, distributed stream processing engines(DSPEs) have been studied to fulfill continuous query processing for data sources which are located at remote sites. Since they are built on cooperation of several stream processing engines(SPEs), node failures cause the failure of the whole system. This paper proposes a new high-availability scheme named Adaptive Semi-Active Standby (A-SAS) and gives considerations for our previous scheme named Semi-Active Standby(SAS). A-SAS enables adaptive tradeoff between recovery time and consumption of network bandwidth. Properties of the A-SAS are shown, and experimental results suggest effectiveness of SAS and A-SAS.
**Key words**　Stream processing, High availability, Distributed computing

## 1. ま え が き

近年，各種センサデバイスやデータ放送の様な，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処理要求が高まっている．例えば，次々と配信される温度センサデータを監視し，温度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．そして，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に，ストリーム処理エンジン (SPE) [1], [2] が研究開発されている．

SPE では，地理的に離れた場所に設置された情報源を扱う際や，ストリーム処理を行う特定のノードの負荷分散を行う際に，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エンジン (DSPE) [3] を利用する．DSPE では，複数の SPE が互いに連携し，情報源から利用者までストリームデータを中継する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 DSPE 全体が停止してしまうという問題がある．また，DSPE 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，DSPE 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．

一般に，利用者が DSPE を構築する際，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，バンド幅使用量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，バンド幅使用量はできる限り小さくしたいなどは，現実に起こり得る制約の 1 つである．したがって，現実的な環境において DSPE の高信頼化を実現するためには，高信頼化に伴うバンド幅使用量やリカバリ時間について調整できる必要がある．我々の先行研究 [4], [5] では，DSPE の高信頼化に伴う，バンド幅使用量とリカバリ時間の調整を可能にする高信頼化手法 Semi-Active Standby 方式 (SAS)(注1) を提案し，簡易的な実験環境での評価実験により，バンド幅使用量，リカバリ時間が調整可能であることを示す結果が得られた．本稿ではさらに，より複雑な実験環境下における SAS の評価を行う．

また，処理対象となるストリームデータは，データレートやタプルサイズなどのデータが持つ様々な性質が時々刻々と変化する可能性がある．そのため，DSPE 構築時にバンド幅使用量，リカバリ時間の制約を満たしているような場合でも，ストリームデータの性質変化により制約を満たせなくなることが考えられる．そこで本稿では，先に述べた SAS の評価実験に加え，バンド幅使用量とリカバリ時間を推定するための高信頼化コストモデルを提案する．また，コストモデルを高信頼化におけるパラメータの最適化処理に適用することで，先行研究で提案した SAS を拡張した適応型 Semi-Active Standby 方式 (A-SAS) の提案，評価を行う．

本稿では，2 節にて，関連研究について説明する．3 節では，我々の先行研究である SAS について説明する．4 節では，本稿の提案の 1 つである高信頼化コストモデル，5 節では，コストモデルを用いた A-SAS について述べる．そして，6 節にて，評価実験，7 節で議論を示し，8 節で本稿のまとめと今後の課題について述べる．

## 2. 関 連 研 究

本節では，これまでに研究された DSPE 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，分散環境を構成する各 SPE に対して，それぞれバックアップ用の SPE を別マシン上に用意する．動作中の SPE が停止した場合にはバックアップ用の SPE が問合せ処理を引き継ぐ．Hwang らの研究 [6] では，既存の高信頼化方式を以下のように分類している．ここでは，稼働時に問合せ処理を行う SPE 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 SPE を**セカンダリ**と呼ぶ．また，SPE にデータを送信する SPE を**上流側 SPE**，処理結果を受信するストリーム処理システムを**下流側 SPE** と呼ぶ．(図 1)

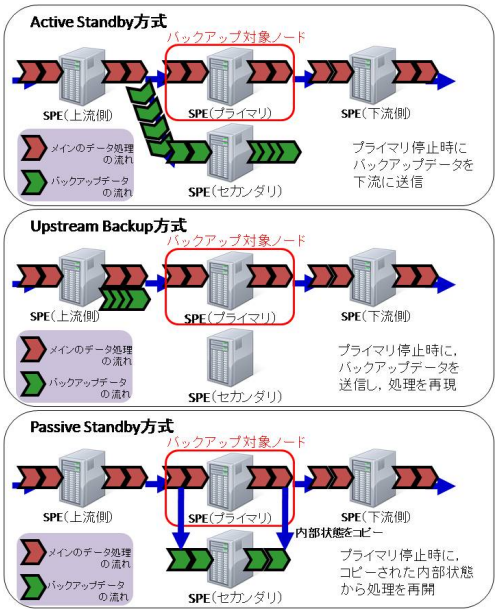



図 1　既存の高信頼化方式

#### 2.1.1 Active Standby 方式

Active Standby 方式 (AS) [6], [7] では，上流側 SPE からプライマリが受信するデータをセカンダリでも並列して受信し，セカンダリもプライマリと同様の問合せ処理を行う．プライマリが生存している間には，セカンダリは処理結果を出力せず，プライマリ停止後に，セカンダリが自身の出力を下流側 SPE 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 (UB) [6], [8] は，プライマリが生存している間には，セカンダリはプライマリの生存確認のみ行い，データのバックアップや問合せ処理は行わない．その代りに，プライマリの上流側 SPE は，プライマリへ送信した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持たない初期状態から問合せ処理を開始し，上流側 SPE にバックアップされているすべてのデータを再送，再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 (PS) [6] 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と内部状態のコピーを定期的に行う．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為，そのまま再開しては処理データが失われる．この損失を防ぐ為，セカンダリはそれらのデータを処理再開時に上流側 SPE から再送させる．

(注1)：Lazy Standby 方式から改名

PS は，AS・UB 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．本稿では，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，PS を比較の対象としない．

### 2.2 特性比較

Hwang らの研究では特性比較のために，バンド幅オーバヘッドとリカバリ時間という 2 つの指標を用いている [6]．

［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．AS 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 UB よりも小さくなる．UB 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 AS 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．実際に DSPE を構築する際には，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両指標の調整は不可欠である．これに対し，我々の先行研究 [4] では，新たな高信頼化手法 SAS を提案し，特定の環境下において，両指標が調整可能であることが示されている．しかし，複雑な環境下における動作特性の評価がなされていないため，様々な環境下における実験，考察を行う必要がある．

これに加え，既存方式 [6] や先行研究 [4] の SAS では，ストリームデータの性質変化が考慮されていない．そのため，DSPE 構築時に満たされていたバンド幅オーバヘッドやリカバリ時間に関する制約が，ストリームデータの性質変化により満たされなくなる可能性がある．ゆえに，バンド幅オーバヘッドとリカバリ時間の調節が可能であり，ストリームデータの性質変化に適応的に対応可能な方式を考える必要がある．

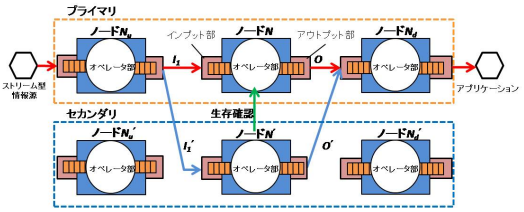



図 2 システムモデル（3 ノード）

## 3. 先行研究 Semi-Active Standby 方式

本節では，我々の先行研究である Semi-Active Standby 方式 (SAS) について述べる．

### 3.1 システムモデル

SAS では，従来研究 [6] 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セカンダリを用意する．3 つのノードに分散したシステムモデルの例を図 2 に示す．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 $I_1$ とし，$N$ から $N_d$ を $O$ としている．また本稿では，従来研究 [6] と同様に，通信によるデータ順序の入れ替わりや，データの欠落が生じないことを前提とする．

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，定期的に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き継ぎを行う（$N$ が故障した場合，$I_1$，$O$ をそれぞれ $I_1'$，$O'$ に切り替える）．

また，従来研究 [6] とは異なり，各 SPE 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 3 つに抽象化して扱う．インプット部は，上流側 SPE から配信されるデータを受け取りインプットキューに蓄える．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にどのデータを受け取ったかという情報を示す [6]．オペレータ部へと入力されたデータは連続的問合せ [9] 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，受け取った処理結果を下流側 SPE に送信し，バックアップをアウトプットキューに蓄える．また，プライマリのアウトプット部では，下流側 SPE のインプット部から Level-0-Ack を受信する毎に 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 の情報をさら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 [6]．

### 3.2 Semi-Active Standby 方式

SAS 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**アルゴリズム 1**，**アルゴリズム 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（**アルゴリズム 1**）とセカンダリ $N'$ における処理（**アルゴリズム 2**）に分けて説明する．

**アルゴリズム 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
```

**アルゴリズム 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全て下流
    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
```

**プライマリ $N_u$ における処理**

まず，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し，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，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，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 $N_u$ よりも上流側にノードが存在する場合，前述の処理に並列して，プライマリ $N_u$ は定期的に Level-0-Ack を生成し，上流側のノードに送信する．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．また，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，プライマリ $N$ に対して定期的にパケットを送信し，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毎に，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．処理結果生成後，その処理結果をアウトプットキューに挿入する．

前述の処理に並列して，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と，アウトプットキューから削除する．

**リカバリ処理**

SAS では，プライマリ停止後，セカンダリ $N'$ はまず上流側のプライマリ $N_u$，下流側のプライマリ $N_d$ と接続を行う．そして，セカンダリ $N'$ のアウトプットキュー内のデータを全て下流側のプライマリ $N_d$ へ送信，上流側プライマリ $N_u$ のアウトプットキュー内のデータを再送，再処理することでリカバリ処理を行う．

**動作特性**

SAS 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すると，バックアップデータの送信間隔は短くなり，頻繁にバックアップ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すると，バックアップデータは送信されず，プライマリ $N_u$ に保持される．SAS は，バッチサイズを変更することで，既存方式 AS・UB の動作を実現可能である．本方式では，バッチサイズを 1 とするときには AS と同様の動作をし，バッチサイズを理論上無限大にすることにより UB と同様の動作をする．既存方式 AS・UB は，本方式の特殊な場合であるとみなすことができる．

## 4. 高信頼化コストモデル

本節では，DSPE の高信頼化におけるバンド幅使用量，リカバリ時間に関するコストモデルについて述べる．

### 4.1 ストリームデータの性質変化

ストリームデータは，データレートや 1 タプル当りの処理時間などの性質が時々刻々と変化する場合がある．これらは，バンド幅の使用量やリカバリ時間を決定する要因であるため，このような性質の変化に伴い，高信頼化に必要なバンド幅使用量，リカバリ時間も時々刻々と変化する．しかし，既存手法 [6] の AS，UB や先行研究の SAS [4] は，性質変化に合わせた最適化機構を持たないため，利用者の要求するバンド幅やリカバリ時間の制約を満たせない可能性がある．そこで本節では，最適化処理に必要となる高信頼化コストモデルを提案する．

#### 4.1.1 バンド幅使用量に関するコストモデル

高信頼化に関わるバンド幅使用量は，単位時間あたりどれくらいのデータをセカンダリ $N'$ へと送信しているかにより決定する．そこで，現在の推定バンド幅使用量を $BW^*$，現在のバッチサイズを $B^*_{size}$，1 タプル辺りの平均データ量を $|tuple_i|$，単位時間あたりの平均バッチ送信回数を $B_{send}$ として，バンド幅使用量を以下のように定義する．

［定義 3］ バンド幅使用量のコストモデル

$$BW^* = B_{send} \sum_{i=1}^{B^*_{size}} |tuple_i| \tag{2}$$

このコストモデルでは，平均バッチ送信回数 $B_{send}$ と，1 タプル辺りの平均データ量を $|tuple_i|$ を実測することにより，バンド幅使用量を推定する．そのため，データレートの上昇，データサイズの増加というバンド幅に影響を与える要因をそれぞれ検知することができる．

#### 4.1.2 リカバリ時間に関するコストモデル

リカバリ時に必要となるリカバリ時間は，プライマリ $N_u$ のアウトプットキューにどれくらいの量のデータが存在しているかに依存する．本稿ではリカバリ時間が最大となる場合を想定し，プライマリ $N_u$ のアウトプットキューにはバッチサイズ分のデータが存在するものとする．これに基づき，現在の推定リカバリ時間を $R^*_{time}$，現在のバッチサイズを $B^*_{size}$，タプルの送信にかかる時間を $S_{time}$，タプルの再処理時間を $Rp_{time}$ として，リカバリ時間を以下のように定義する．

［定義 4］ リカバリ時間のコストモデル

$$R^*_{time} = B^*_{size}(S_{time} + Rp_{time}) \tag{3}$$

このコストモデルでは，タプル送信時間 $S_{time}$ と，タプル再処理時間 $Rp_{time}$ を実測することにより，リカバリ時間を推定する．そのため，ネットワーク遅延の増大や，再処理時間の増加というリカバリ時間に影響を与える要因をそれぞれ検知することができる．

## 5. 適応型 Semi-Active Standby 方式

本節では，3. 節で述べた SAS に対して，前節で示したコストモデルによるバッチサイズ最適化処理を拡張した適応型 Semi-Active Standby 方式 (A-SAS) について述べる．

**アルゴリズム 3:** バッチサイズ最適化処理

```
1: 現在の BW* と R*_time を推定
2: if BW* > BW borderline かつ R*_time < R_time borderline then
3:     バッチサイズを指定幅分インクリメント
4: else if BW* < BW borderline かつ R*_time > R_time borderline then
5:     バッチサイズを指定幅分デクリメント
6: else if BW* > BW borderline かつ R*_time > R_time borderline then
7:     バンド幅使用量とリカバリ時間のうち，優先度の高い方のみに最適化
8: else if BW* < BW borderline かつ R*_time < R_time borderline then
9:     最適化処理を終了する
```

### 5.1 適応型 Semi-Active Standby 方式の流れ

A-SAS の処理の流れは，コスト測定処理とバッチサイズ最適化処理の 2 つに分けることができる．以下で，それぞれについて説明する．

#### 5.1.1 コスト測定処理

前節で示したコストモデルを利用するために，A-SAS では，平均バッチ送信回数 $B_{send}$，1 タプル辺りの平均データ量を $|tuple_i|$，タプル送信時間 $S_{time}$，タプル再処理時間 $Rp_{time}$ を測定する必要がある．本稿では，DSPE における各プライマリ，セカンダリは対称的なマシン性能，ネットワーク性能を有するものとし，次のようにしてそれぞれの測定を行う．

**$B_{send}$，$|tuple_i|$ の測定**

平均バッチ送信回数 $B_{send}$，1 タプル辺りの平均データ量を $|tuple_i|$ では，まず，プライマリ $N_u$ において，バッチの送信回数，送信データ量を一定期間分測定する．そして，蓄積した値から平均値を算出することで $B_{send}$，$|tuple_i|$ を求める．

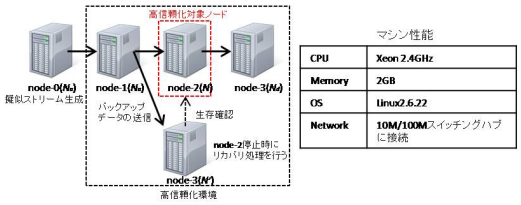
**$S_{time}$，$Rp_{time}$ の測定**

本研究では，プライマリとセカンダリが対称的な性能を有するものとしているため，リカバリ処理時に発生すると考えられるタプル送信時間 $S_{time}$，タプル再処理時間 $Rp_{time}$ を，プライマリ $N$ において測定することで，リカバリ時間の推定を行う．まず，プライマリ $N$ において，タプル送信時間，タプル再処理時間を一定期間分測定し，その平均値を求める．そして，プライマリ $N_u$ においてバッチサイズ最適化処理が実行される際に，プライマリ $N$ から各値を送信してもらうことにより，$S_{time}$，$Rp_{time}$ を求める．

#### 5.1.2 バッチサイズ最適化処理

A-SAS で用いる最適化処理の流れを**アルゴリズム 3** に示す．本節では，利用者により事前に設定されるバンド幅使用量，リカバリ時間の上限をそれぞれ $BWborderline$，$R_{time}borderline$ とする．

バッチサイズ最適化処理では，まずコストモデルを用いて，




マシン性能

| | |
|---|---|
| CPU | Xeon 2.4GHz |
| Memory | 2GB |
| OS | linux2.6.22 |
| Network | 10M/100Mスイッチングハブに接続 |

図 3　実 験 環 境

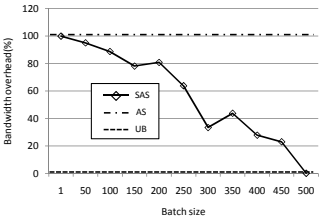



図 4　従来方式とのバンド幅オーバヘッド比較

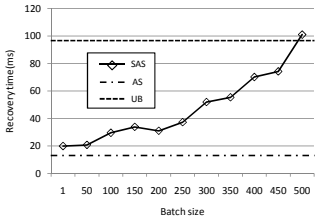



図 5　従来方式とのリカバリ時間比較

現在のバンド幅使用量 $BW^*$ とリカバリ時間 $R^*_{time}$ の推定を行う．コストモデルを用いる際に必要となる $B_{send}$ や $|tuple_i|$, $S_{time}$，$Rp_{time}$ の各値は，最適化処理前に測定を行い，コストモデルに適用する．現在のバンド幅使用量とリカバリ時間を推定した後，その結果を利用者が事前に設定した各指標の上限値と比較する．このとき，バンド幅利用量のみ上限値を超える場合には，SAS の性質に基づき，バッチサイズをインクリメントする．同様に，リカバリ時間のみ上限値を超える場合には，バッチサイズのデクリメントを行う．バッチサイズのインクリメント・デクリメント幅については利用者により事前に定義されるものとする．また，バンド幅使用量とリカバリ時間の上限値を両方共を超えるような場合には，事前に指定した優先度に従い，優先度の高い一方に関して最適化処理を行う．バッチサイズの調整処理終了後，再度，最初の処理へと戻り，バンド幅使用量とリカバリ時間が上限値を満たすまで最適化処理を繰り返す．最終的にバンド幅使用量とリカバリ時間が制約を満たした場合には，最適化処理を終了する．本方式では，以上の処理を定期的に実行することにより，バッチサイズの最適化を行う．

## 6. 評 価 実 験

本節では，SAS，及び，A-SAS の評価実験を行う．

### 6.1 プロトタイプシステム

本稿では，評価実験のために DSPE のプロトタイプシステムを実装した．本システムでは，処理モデルとして CQL モデル [9] を採用しており，選択演算，射影演算，集約演算が実行可能である．また，高信頼化手法として SAS，A-SAS，AS，UB を有し，各手法の通信プロトコルには TCP を用いている．

### 6.2 実 験 環 境

評価実験環境を図 3 に示す．本稿では，6.1 節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．本実験では node-2 が停止故障する場合を想定する．node-4 は node-2 に対して 250ms 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 250ms 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．また，図 3 の node-0 は，情報源として疑似ストリームデータを生成し，node-1 へ送信する．node-0 で生成される疑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，1 タプルあたりのデータサイズを 44byte とした．

### 6.3 SAS 性能評価

提案手法 SAS の性能評価を示す．本実験は先行研究 [4] とは異なり，オペレータの種類やネットワーク負荷を変化させた場合についても評価，考察を行う．本実験では，図 3 の node1〜node4 に選択率 1.0，window 幅 100tuples の選択演算を配置し，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 20 回試行し，その平均値を実験結果とした．本実験では，AS，UB に対しても同様の実験を行い，比較を行った．なお，本実験で測定を行った SAS のバッチサイズは，1 と 50 から 500 まで 50 ずつサイズを変化させた 11 種類である．また，実験で用いる人工ストリームデータのデータレートを 1000tuples/s とした．

#### 6.3.1 実験 1:SAS と従来方式の比較

SAS 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．

**実験結果**

実験結果を図 4，5 に示す．図 4 は SAS 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の推移，図 5 は，リカバリ時間の推移である．この結果より，SAS はバッチサイズが大きくなるのに伴い，バンド幅オーバヘッドは減少し，リカバリ時間は増大することがわかる．また，バッチサイズが 1 のときには AS 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，UB の実験結果に近づいていることがわかる．バッチサイズが 1 のとき，SAS は node-1 の全てのデータを node-4 へ送信するため，バンド幅オーバヘッドが AS と同じだけ必要となる．また，これにより，node-4 は node-2 と同じ内部状態を保つことができるため，リカバリ時間を小さく抑えることができる．バッチサイズを大きくした場合，バッチサイズ分のデータが node-1 のアウトプットキューに貯まるまでバックアップデータの送信を待たなければならない．バックアップデータの送信を待機することにより，Level-1-Ack が到着し，蓄積されたバックアップデータは定期的に削除される可能性が高くなる．そのため，バッチサイズが大きい程，node-1 におけるバックアップデータの送信感覚が長くなり，バンド幅オーバヘッドが減少することになる．しかし，バッチサイズを大きくしたことにより，リカバリ処理時に再送信・再処理する必要があるデータ量が増加するため，UB と同様にリカバリ時間が増大することになる．

#### 6.3.2 実験 2: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

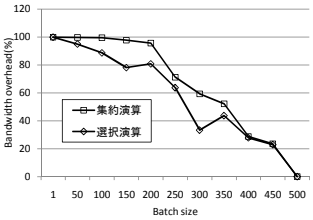



図 6　オペレータの違いによるバンド幅オーバヘッド比較

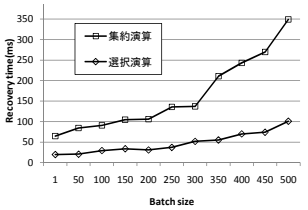



図 7　オペレータの違いによるリカバリ時間比較

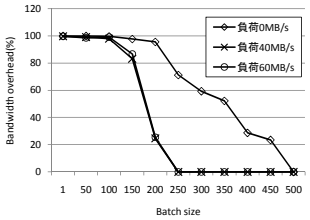



図 8　NW 負荷の変化によるバンド幅オーバヘッドの比較

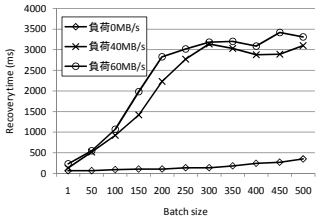



図 9　NW 負荷の変化によるリカバリ時間の比較

類を変え，SAS の性能評価を行う．本実験ではまず，図 3 の node-1 と node-3 に選択率 1.0，window 幅 100tuples 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 100tuples の選択演算子を配置した場合と，window 幅 100tuples，slide 幅 10tuples の集約演算子を配置した場合の 2 種類について比較実験を行った．本実験では集約演算として，window 幅内に存在する全タプルの属性値に対して，加算処理を実行した．

**実験結果**

実験結果を図 6，7 に示す．図 6 は，選択演算と集約演算を実行した時のバンド幅オーバヘッドの比較を，図 7 はリカバリ時間の比較を示したものである．まず，バンド幅オーバヘッドでは，両演算間で比較すると，選択演算の方が最大で 20%程小さい結果を示す事がわかる．これは，集約演算の方が選択演算よりも処理時間が大きい演算であること，そして，集約演算が状態を持つ演算であることが原因である．集約演算は，選択演算よりも処理時間が大きいため，node-1 から node-2 を経由して node-3 へとデータが送信されるまでに要する時間が，選択演算よりも大きくなる．そのため，Level-0-Ack や Level-1-Ack の返信時間も大きくなる．また，集約演算は window 幅内のタプルの状態に依存する演算であるため，あるデータが node-2 から node-3 へ送信されてから，削除可能と判定されるまでに，選択演算よりも長い時間を要する．ゆえに，あるデータが Level-0-Ack，Level-1-Ack として返信されるまでに要する時間も長くなる．したがって，node-1 にあるバックアップデータが削除されにくくなり，選択演算よりもバンド幅オーバヘッドが大きくなると考えられる．次に，リカバリ時間では，集約演算は選択演算に対し，約 3.1 倍のリカバリ時間を用することがわかる．これは，本実験で熱かった集約演算の処理時間が選択演算よりも約 3 倍大きいことが原因で有る．

**6.3.3**　実験 3: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，SAS の性能評価を行う．本実験では，図 3 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 40MB/s，60MB/s の負荷をかけた状況での比較実験を行った．なお，本実験では 6.3.2 節の集約演算子と同様の演算子配置を用いた．

**実験結果**

実験結果を図 8，9 に示す．図 8，9 はネットワークに負荷を発生させた場合の SAS のバンド幅オーバヘッドとリカバリ時間を比較したグラフである．まず，バンド幅オーバヘッドでは，負荷をかけた場合，負荷をかけない場合とは異なり，バッチサイズ 250 において，0%を示している．これは，負荷をかけたことにより，node-1 から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，蓄積されたバックアップデータが Level-1-Ack により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しやすくなると考えられる．次に，リカバリ時間では，負荷が大きくなるのに伴ない，リカバリ時間が大きく成ることがわかる．これは，リカバリ時間処理を行う際，負荷が大きいほど，node-1 のスループットが低下する事が原因であると考えられる．

### 6.4　A-SAS 性能評価

本実験では，本稿で提案した A-SAS の性能評価を行う．6.3 節と同様に選択演算を配置し，ストリームデータ配信開始 10 秒後にストリームデータの性質を変化させ，バンド幅使用量，リカバリ時間の変化を測定する．バンド幅使用量では，node-1，node-4 間の送信データ量を毎秒計測することで測定する．リカバリ時間では，人工ストリームデータ送信開始後，1，5，10，15，20，25，30 秒後に node-2 を停止させ，そのリカバリ処理に用する時間を測定する．データ測定では，この操作をそれぞれ 5 回試行し，その平均値を実験結果とした．また，AS，UB に対しても同様の実験を行い比較を行った．データ送信開始時における A-SAS のバッチサイズは 1，最適化時のバッチサイズ変化幅は 5，最適化処理実行間隔は 10ms とした．

**6.4.1**　実験 4:データレートの変化による評価

人工ストリームデータの送信レートを，配信開始から 10 秒間は 100tuples/s，10 秒後は 1000tuples/s と変化させ実験を行った．本実験で A-SAS に設定した，バンド幅使用量，リカバリ時間の上限値はそれぞれ 30KB/s，80ms である．また，本実験では，バンド幅，リカバリ時間ともに上限値を超えた場合，リカバリ時間を優先するものとした．

**実験結果**

実験結果を図 10，11 に示す．図 10，11 はそれぞれ，バンド幅使用量，リカバリ時間の比較を示したグラフである．まず，バンド幅使用量では，10 秒経過しデータレートが増加したとき，A-SAS では，最初の 2 秒間バンド幅の上限を超えてしまうが，その後，上限値を超えていないことがわかる．これに対し，AS では，データレート増加後，バンド幅の上限値を超えたままになっている．また，リカバリ時間では，データレート増加後でも，A-SAS は，上限値を超えていないのに対し，UB では上限

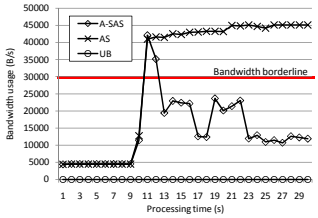



図 10　データレートの変化による バンド幅使用量の比較

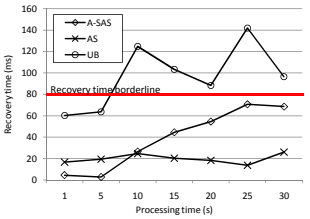



図 11　データレートの変化による リカバリ時間の比較

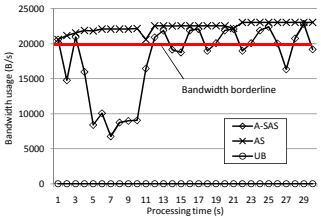



図 12　処理時間の変化による バンド幅オーバヘッドの比較

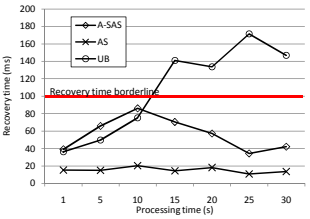



図 13　処理時間の変化による リカバリ時間の比較

値を超えてしまうことがわかる．A-SAS では，コスト見積りにより，バッチサイズが適宜調整されているためであると考えられる．従来方式には，本方式のような最適化機構がないため，上限値を超える結果となっている．

**6.4.2**　実験 5:1 タプルあたりの処理時間の変化による評価

人工ストリームデータ 1 タプルあたりの処理時間を，配信開始から 10 秒間は 0.05ms，10 秒後は 1ms と変化させ実験を行った．本実験で用いた人工ストリームデータの送信レートは 500tuples/s とし，A-SAS に設定したバンド幅使用量，リカバリ時間の上限値はそれぞれ 20KB/s，100ms である．また，本実験では，バンド幅，リカバリ時間ともに上限値を超えた場合，リカバリ時間を優先するものとした．

**実験結果**

実験結果を図 12，13 に示す．図 12，13 はそれぞれ，バンド幅使用量，リカバリ時間の比較を示したグラフである．バンド幅使用量では，処理時間が増加する前に関して，A-SAS は上限値を超えないように，バンド幅使用量を下げていることがわかる．しかし，処理時間増加後では，上限値を挟んで，値が上下していることがわかる．これに対して，AS では，データ送信開始時から上限値を超えたままとなっている．また，リカバリ時間では，処理時間増加後，A-SAS は，上限値を超えていないのに対し，UB では上限値を超えてしまうことがわかる．処理時間が増加する前では，バンド幅使用量の上限値，リカバリ時間の上限値の両制約共に満たせる環境にあったため，コスト見積りが働いてバンド幅に関してバッチサイズの調整が行われているが，処理時間増加後，両制約を共に満たせない状況となり，リカバリ時間に関して優先的にバッチサイズ調整が働いているものと考えられる．

## 7. 議　　論

本稿では，A-SAS において，プライマリとセカンダリが対称的なマシン性能，ネットワーク性能を有するに限定し，リカバリ時間の推定に必要となる各コストを測定している．しかし現実的な処理環境では，プライマリとセカンダリマシン性能が異なる場合や，一方のマシンのみに負荷がかかっている場合など，プライマリとセカンダリの関係が非対称となるような状況が存在する．このような場合，本稿で用いたコスト算出では，正確にバッチサイズを調整できない可能性があり，本稿とは異なるコスト測定アプローチが必要になると考えられる．

## 8. まとめと今後の課題

本稿では，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 AS と UB を統合する手法，SAS を提案した．また，状況に応じて適応的にバッチサイズを変化させる A-SAS を提案した．

今後の課題として，非対称的な環境におけるコスト算出や複数ストリームを考慮した手法の検討などが挙げられる．

**謝辞**　本研究の一部は，科学研究費補助金基盤研究 (A)(#21240005)，科学研究費補助金特定領域研究 (#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] Rajeev Motwani, et al., Query processing, approximation, and resource management in a data stream management system. In *CIDR*, pp. 245–256, 2003.

[2] D.J. Abadi, et al., Aurora: a new model and architecture for data stream management. *The VLDB Journal*, Vol. 12, No. 2, pp. 120–139, 2003.

[3] Yanif Ahmad, et al., Distributed operation in the borealis stream processing engine. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pp. 882–884, New York, NY, USA, 2005. ACM.

[4] 塩川浩昭, ほか，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. DEIM2009 論文集, 2009.

[5] 塩川浩昭, ほか，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．電子情報通信学会論文誌（情報爆発特集号）(to appear)

[6] J.H. Hwang, et al., High-availability algorithms for distributed stream processing. In *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, pp. 779–790, 2005.

[7] S. Chandrasekaran, et al., TelegraphCQ: Continuous dataflow processing for an uncertain world. In *Proc. CIDR*, pp. 269–280, 2003.

[8] Mitch Cherniack, et al., Scalable Distributed Stream Processing. In *CIDR 2003 - First Biennial Conference on Innovative Data Systems Research*, p. 23, Asilomar, CA, January 2003.

[9] A. Arasu, et al., The CQL continuous query language: Semantic foundations and query execution. *The VLDB Journal The International Journal on Very Large Data Bases*, Vol. 15, No. 2, pp. 121–142, 2006.