確率モデルに基づく近似的な耐障害性の保証

高尾　大樹† 　石川　佳治† 　杉浦　健人†

† 名古屋大学大学院情報学研究科　〒 464–8603 愛知県名古屋市千種区不老町
Email: {takao,sugiura}@db.is.i.nagoya-u.ac.jp, ishikawa@i.nagoya-u.ac.jp

あらまし　分散データストリーム処理は様々な障害の影響を大きく受けるため，多くの処理システムではチェックポインティングによって耐障害性を保証している．しかし、そうしたシステムは障害時にチェックポイントからの全データを再処理しなければならず，復帰時間が大きくなるという課題がある．そこで本稿では，データストリームの集約処理における効率的な障害復帰のための，確率モデルに基づいた近似的な耐障害性の保証を提案する．確率モデルに基づき他のノードの内部状態から障害によって損失したノードの内部状態を推定することで，復帰処理でのオーバヘッドを削減する．また，より効率的に近似的耐障害性の保証を実現するためのノード分割の方法についても検討する．

キーワード　データストリーム処理，耐障害性，近似処理

1 はじめに

データの大規模化と意思決定の高速化を受け，分散データストリーム処理システムが多くのアプリケーションで利用されている．近年，Internet of Things（IoT）などの発展に伴い，センサデータやモバイル端末のログデータを始めとする大規模なデータストリームが得られるようになった．このようなデータストリームは，物流システムの異常検知やヘルスケアにおける行動モニタリング，さらには将来予測や行動推薦など様々な応用の可能性を持つ．一方，大規模なデータストリームの処理は簡単ではなく，Flink [1] や Spark Streaming [2] など，並列分散処理に基づく高スループットなデータ処理システムがそれらの処理基盤として用いられている．

分散データストリーム処理システムの構築において，耐障害性の保証は重要な要件の一つである．特に，障害の発生が処理の一貫性に与える影響の有無を，システムによってどの程度保証するかは重要な問題である．例えば，exactly-once セマンティクスを採用するシステムでは，障害発生の有無にかかわらず入力データが一度のみ処理されることを保証する．厳格な耐障害性を保証することでアプリケーション開発における実装コストを下げられるが，一方でシステムの処理性能は落ちてしまうため，耐障害性と処理性能のトレードオフをどのようにとるかは未だに課題である．

多くのシステムではチェックポインティングを用いて厳格な耐障害性を保証している [1–4]．チェックポインティングは，処理ノードの内部状態を任意の周期で外部ストレージなどに保存し，障害発生時には保存した内部状態を復元した後に必要な入力データを再送することでシステムを復旧する方式である．例として，6 つのセンサの計測値を入力とし，各センサ値の平均値 $X_1, ..., X_6$ を 2 つのワーカで並列に計算するという状況を考える．図 1 はチェックポイントの取得と障害からの復旧を表した図で，$t = t_e$ においてワーカ B に障害を検出した場合，最新のチェックポイントである $t = t_c$ でのワーカ A とワーカ B の内部状態 $X_1^{t_c}, ..., X_6^{t_c}$ を DB から送付する．そして，$t = t_c$ から $t = t_e$ までのワーカ A とワーカ B に対する入力データ $x_i^{t_c}, .., x_i^{t_e}$ $(1 \leqq i \leqq 6)$ を再処理することで障害発生前の状態を復元する．

チェックポインティングは頑健な耐障害性の保証が可能である一方で，以下の 2 つの課題がある．

一貫性のあるチェックポイント取得の難しさ　ストリーム処理では一般にフィルタ処理や集約処理などを連続して適用することで最終的な処理結果を得る．したがって，システムで exactly-once セマンティクスを保証しようとする場合，チェックポイントは一連の処理を行うノード群で一貫したものでなければならない．しかし，一貫性のあるチェックポイントを取得するには，それらのノードが同期して内部状態を保存するか，Flink のような複雑な非同期処理 [1] を行う必要がある．

障害復帰における再処理コスト　チェックポイントはある時点における内部状態のバックアップであるため，障害発生時の状態まで復元するにはチェックポイント以降の入力データの再送および再処理が必要となる．再処理のコストを削減するにはチェックポイントを取る頻度を増やす必要があるが，チェックポイントの取得により本来目的としている処理のスループットを犠牲にしてしまう例も報告されている [5] ため，得策ではない．

そこで本稿では，データストリームの集約処理における効率的な障害復帰のための，確率モデルに基づいた近似的な耐障害性の保証を提案する．本手法では，入力ストリームがセンシングデータなどである場合に，データ間に相関がある [6] ことを利用する．つまり，各キーの集約値の相関を確率モデルとして表すことで，あるキーの集約値から別のキーの集約値が推定できると仮定する．このような仮定の下，ある集約ノードで障害が発生したとき，他のノードの集約値から障害ノードの集約値を直接推定することで耐障害性を保証する．

例として，本手法での障害復帰処理の流れを図 2 に示す．本手法では，各キーの集約値の相関を多次元ガウス分布でモデル化する．また，ユーザから許容可能な誤差上限 ϵ と信頼度 $1 - \delta$

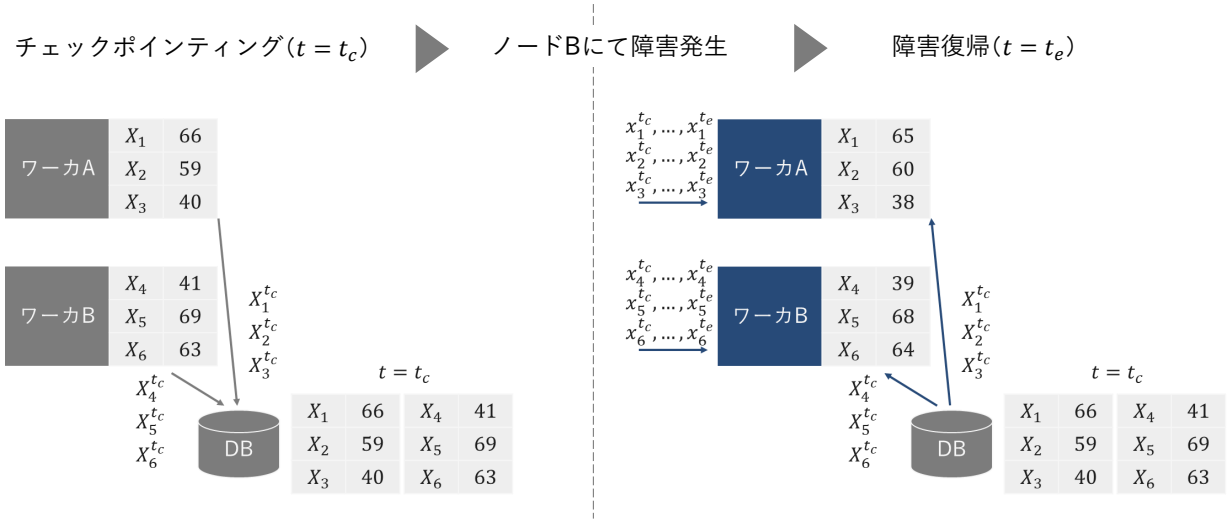

図 1　チェックポインティングによる障害復帰処理の流れ

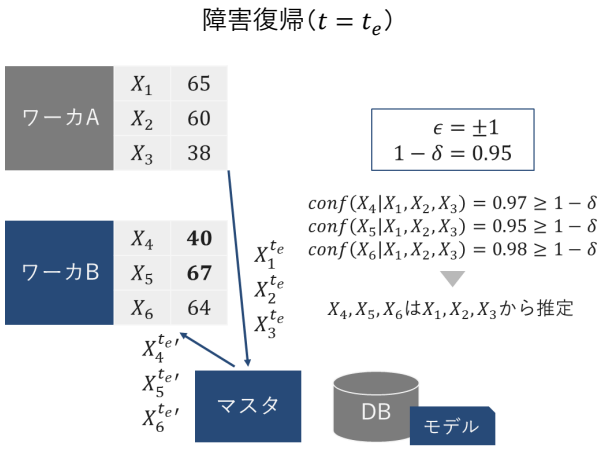

図 2　提案手法による障害復帰処理の流れ

を受け取り，各ワーカで計算された集約値が他のワーカ中の集約値から推定できるようにノード分割する．つまり，図 2 において，ワーカ A とワーカ B は互いの集約値を十分正確に推定でき，$t = t_e$ でワーカ B での障害の発生を検知した際にはワーカ A の集約値 $X_1^{t_e}, X_2^{t_e}, X_3^{t_e}$ からワーカ B の集約値 $X_4^{t_e}, X_5^{t_e}, X_6^{t_e}$ を近似的に復元する．このように，いくらかの誤差を許容した耐障害性を考えることで，定期的なチェックポイントの取得も障害復帰処理での入力データの再送も不要となる．

本稿での貢献は以下の通りである．

効率的な耐障害性の保証　データ間に相関がある状況下で，障害により損失した集約値を確率モデルに基づいて推定することで，近似的に耐障害性を保証する手法を提案する．本手法は，障害復帰に他の処理ノードの集約値を最大限活用することで，定期的なチェックポイントと入力データの再処理を省略し，処理性能の向上を図る．

新しいセマンティクスの提案　入力データを過不足なく一回処理した結果を近似的に復元する approximately-once セマンティクスに基づく本手法は，ユーザからの誤差要求だけでなく，処理結果に対する一定の一貫性の保証も可能である．一方，既存の近似的な耐障害性を保証する手法では，at-most-once セマンティクスや at-least-once セマンティクスに基づいており，誤差保証が十分でないものが多い．

適切なノード分割方法の検討　本稿では，他のワーカの集約値から損失した集約値を推定できるワーカをより多くするようなノード分割を確率モデルから決定する方法についても検討する．具体的には，誤差要求を制約条件としたノード分割の最適化問題としてこの問題を定式化し，これを解くための貪欲的なアプローチを提案する．

本稿の構成は以下の通りである．2 章では既存のストリーム処理システムでの耐障害性について概説する．次に，3 章では確率モデルに基づく近似的な耐障害性保証について，4 章では approximately-once セマンティクスに基づいた障害復帰について，それぞれ述べる．また，5 章では提案手法における効率的なノード分割についても検討する．提案手法による集約値復元の精度などについては，性能評価実験の結果を 6 章に示す．最後に，本研究のまとめと今後の課題について 7 章で述べる．

2　関 連 研 究

本章では，既存のデータストリーム処理エンジンの耐障害性について概説する．2.1 節ではチェックポインティングによる頑健な耐障害性保証について，2.2 節では近似的な耐障害性保証についてそれぞれ述べる．

2.1　頑健な耐障害性保証

誤差のない頑健な耐障害性として，多くのシステムが exactly-once セマンティクスを保証している [1–4]．これらシステムでは，入力されたデータを過不足なく一回処理した結果を復元するために，チェックポインティングによる手法が用いられている．しかし，頑健な耐障害性を実現するためにはチェックポインティング処理及び復帰処理においてファイルストレージや DB との多量のデータ通信が必要となり，処理性能の低下を招く恐れがある．

この問題に対していくつかのシステムでは，データの圧縮による通信量削減や，非同期なバックアップ処理によるシステム

全体での処理性能向上といった対策を取っている [1]．しかし，全ての内部状態や処理データをバックアップし，障害復帰時にはチェックポイント地点からのロールバック処理が必要となることには変わらない．センシングデータのような大規模データストリームに対するアプリケーションを念頭に置くと，頑健な耐障害性よりもスループットの向上や迅速な障害復帰が求められると考えられるため，この方針は最適ではない．

2.2 近似的な耐障害性保証

近似的に耐障害性を保証するシステムの多くは at-most-once セマンティクスや at-least-once セマンティクスに基づいている [7,8]．これらシステムは，2.1 節のシステムより前に開発されており，当初は頑健な耐障害性の保証が難しいことから近似的に耐障害性を保証していた．at-most-once セマンティクスを保証するシステムでは，内部状態や処理データのバックアップは取っておらず，障害発生時は待機ノードへの処理の移譲のみを行うため，データの損失の恐れがある．一方，at-least-once セマンティクスを保証するシステムでは，障害発生時に入力データの再送はできるが，同一データを複数回処理する可能性がある．これらのセマンティクスでは厳密な耐障害性を保証しないため exactly-once セマンティクスよりもスループットは上がるが，誤差上限などの保証もなく，結果の信頼性に問題がある．

そこで，Huang らは結果の信頼性を保証した近似的耐障害性手法を提案した [5]．この手法は，未処理データ数と内部状態の変量に対し閾値を設定し，閾値を超した場合にのみ DB にバックアップを取る．これによりバックアップするデータ量及び頻度を削減し，スループットを向上している．また，このバックアップデータを用いて障害復帰処理を行うことで，障害発生前の状態に完全に復元することはできないものの，復帰処理によって損失するデータの上限を閾値で保証することができる．この手法は，障害復帰時にある程度の誤差を許容する代わりに耐障害性保証のオーバヘッドを削減することから，最終結果に対してある程度の誤差を許容できるアプリケーションに適している．

しかし，この手法では SLA（Service Level Agreement）に適した閾値の設定が困難であるという問題点がある．例えば，未処理データ数に対して適切な閾値を設定するためには，一つのデータの損失が最終結果に与える影響の綿密な分析が必要である．この分析のためには実際のノード構成を考慮する必要があり，特に多段構成や並列処理の場合には複雑な計算が要求されることから，実際のアプリケーションにおいて多くの困難が伴う．

3 集約処理を対象とした近似的な耐障害性保証

本章では，データストリームの集約処理を対象とした，分散データストリーム処理システムの近似的な耐障害性保証について述べる．本手法では，データ間に存在する相関を多次元ガウス分布によってモデル化し，障害の影響があるノードの集約値をこのモデルに基づいて推定することで集約値を近似的に復元する．このような仕組みにより耐障害性を保証することで，定期的なチェックポインティングが必要なくなり，耐障害性保証のコストを大きく下げることができる．

図 3　DAG の 例

本手法の詳細について，まず，本手法において想定する問合せについて 3.1 節で述べる．また，3.2 節では確率モデルに基づいた集約値の推定手法について述べる．

3.1 想定する問合せ

本研究では，処理対象となる問合せが DAG（directed acyclic graph）によって表されると想定する．グラフの各ノードは演算に，エッジは演算の入出力となるストリームに対応する．例えば，図 3 は入力されたストリームを 2 種のフィルタにかけ，最終的に集約処理を行う問合せの DAG 表記である．

既存システムの多くは任意の DAG をサポートするが，一方で Structured Streaming [11] のように処理効率化のために対象とする DAG を限定するシステムもある．このシステムでは，結合演算の種類や条件に対して制限を加えており，集約演算は DAG 中に 1 つしか存在しないものとしている．しかし，様々な企業のサービスに対してこのシステムを実際に導入でき，限定された DAG でも基本的なアプリケーションの表現能力は十分にあるとされている．

そこで本手法では，ある時間窓において集約処理を一つ行う DAG を対象問合せとして想定する．この限定された DAG では，基本的に図 3 のように平均や総和などの集約演算が DAG の末尾に配置され，システムからの最終的な出力ストリームをはき出す．言い換えれば，システムに障害が起きたとしても，この集約値を適切に復元できればシステム全体として出力の信頼性を保証できる．

なお，本手法で扱う集約問合せはデータの各 key に対する集約処理である．つまり，時間窓に対して key 毎に窓内における平均値などを計算するとし，各 key 間の処理は独立とする．以下，DAG における演算処理を表すノードを論理ノード，各 key において論理ノードの処理を実際に行うノードを物理ノードと呼ぶ．

3.2 確率モデルに基づく集約値の推定

今回対象とする大規模なセンサデータには，データ間に相関があることが多い．センサネットワークの分野ではそうしたセンサデータ間の相関を利用し，効率的なデータ取得やクエリ処理を実現する研究が提案されている [6,10]．例えば，Deshpande ら [6] はセンサデータ間の相関を確率モデルで表現し，一部のセンサデータから他のセンサデータの値を推定することでセンサネットワークに対する効率的なクエリ処理を実現している．

本手法では，そうしたセンサデータ間の相関を障害復帰に利用することを提案する．つまり，センサデータの集約結果間に存在する相関を確率モデルによって表現できると仮定し，障害

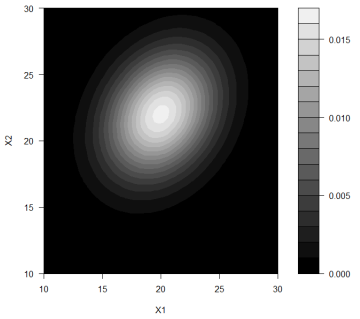

図 4　X_1 と X_2 の間の相関

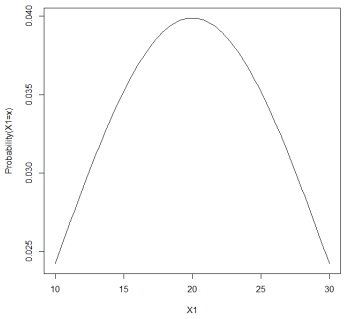

図 5　X_2 を考慮しない場合の X_1 の分布

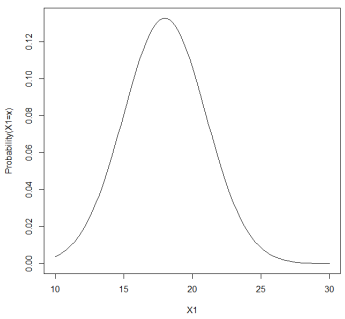

図 6　X_2 を考慮した場合の X_1 の分布

によって損失した物理ノードの出力値を確率モデルから推定する．

本研究では，Deshpande らの手法 [6] と同様に，各物理ノードにおける集約結果間の相関を多次元ガウス分布 $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ でモデル化する．なお，全 n 個の物理ノード，つまり全 n の key で計算される各集約値の確率変数を $\boldsymbol{X} = \{X_1, X_2, ..., X_n\}$ で，$\boldsymbol{\mu}, \Sigma$ でそれらの平均ベクトルと分散共分散行列を表すとする．このとき，いくつかの物理ノード $O \subseteq \boldsymbol{X}$ で集約結果ベクトル $\boldsymbol{o}$ が得られたとき，ある物理ノード $Y \in \boldsymbol{X} \setminus O$ の集約値を表すガウス分布 $\mathcal{N}(\mu_{Y|\boldsymbol{o}}, \Sigma_{Y|\boldsymbol{o}})$ は以下の式で計算できる．

$$\mu_{Y|\boldsymbol{o}} = \mu_Y + \Sigma_{YO}\Sigma_{OO}^{-1}(\boldsymbol{o} - \mu_O) \tag{1}$$

$$\Sigma_{Y|\boldsymbol{o}} = \Sigma_{YY} - \Sigma_{YO}\Sigma_{OO}^{-1}\Sigma_{OY} \tag{2}$$

なお，各記号の添字はベクトルないし行列から添え字に対応する次元のみを抽出することを示す．例えば，μ_Y は $\boldsymbol{\mu}$ から Y 次元目の値を抽出したもの（つまり物理ノード Y の期待値）であり，Σ_{YO} は Σ から Y 行 O 列の値を抽出したもの（つまり物理ノード Y に対する物理ノード集合 O の共分散ベクトル）である．

つまり，他の物理ノードにおける集約値を用いることで，障害が発生した物理ノードの集約値をガウス分布に基づき推定できる．例として，図 4 のような二次元ガウス分布を考える．X_2 の値がわからない場合，X_1 は図 5 のような事前ガウス分布 $\mathcal{N}(\mu_{X_1}, \Sigma_{X_1X_1})$ を持つ．一方で，$X_2 = 20$ であるとわかった場合，$O = \{X_2\}, \boldsymbol{o} = \langle 20 \rangle$ を式 (1) 及び式 (2) に代入することで，図 6 のような事後ガウス分布 $\mathcal{N}(\mu_{X_1|\boldsymbol{o}}, \Sigma_{X_1|\boldsymbol{o}})$ が得られる．つまり，X_2 の値を用いることで X_1 の分散が減少し，より正確な推定が可能となっている．X_1 を障害の発生した物理ノードとみなすと，これは無事な物理ノード X_2 の値を用いて推定精度を向上することで，事後ガウス分布の期待値 $\mu_{X_1|\boldsymbol{o}}$ を X_1 の近似的な復元値として扱えることを示す．

ただし，推定される値があくまで近似的に復元した値であり，信頼度の評価が必要となる点に注意する．推定値の信頼度を計算するために，本研究ではユーザが誤差上限 ϵ を与えると想定する．このとき，物理ノード集合 $O \subseteq \boldsymbol{X}$ の各集約値 $\boldsymbol{o}$ を用いた推定結果の信頼度 $R(Y|\boldsymbol{o})$ は，以下の式で計算できる．

$$R(Y|\boldsymbol{o}) = P(Y \in [\mu_{Y|\boldsymbol{o}} - \epsilon, \mu_{Y|\boldsymbol{o}} + \epsilon]|\boldsymbol{o}) \tag{3}$$

つまり，事後ガウス分布 $\mathcal{N}(\mu_{Y|\boldsymbol{o}}, \Sigma_{Y|\boldsymbol{o}})$ の確率密度関数 P に対して，Y の値が期待値 $\mu_{Y|\boldsymbol{o}}$ の上下 ϵ 以内となる確率を信頼度として扱う．

4　approximately-once セマンティクスの保証

本手法では，前章で述べた確率モデルに基づく集約値の推定を用いることで，入力データを過不足なく一回処理した際の集約結果の近似的な復元を保証する．以降，本研究ではこれを *approximately-once* セマンティクスによる耐障害性保証と呼ぶ．

本章では，approximately-once セマンティクスに基づく近似的耐障害性保証の仕組みについて述べる．まず，4.1 節では本手法で想定するシステム構成について述べる．次に，障害復帰の流れについて 4.2 節で述べる．

4.1　システム構成

本手法で想定するシステム構成を図 7 に示す．それぞれの構成要素と役割については以下の通りである．

- **ワーカ**：ワーカはスレッドなどの実際にプログラムを走らせる単位であり，DAG で表された問合せを処理する役割を持つ．ワーカにはいくつかの物理ノード，つまりデータの key が割り当てられ，割り当てられた key に対する処理を行う．自身の key を持つストリームが入力されると想定し，DAG 上における後段の処理を行うワーカもしくはシンクに対して対応する key の出力データを送信する．
- **ソース**：ソースはセンサ機器などからの入力データを管理し，DAG における先頭の処理を行う各ワーカに対して対応する key のデータを送信する．なお，Kafka [9] などのように，任意の時点からのデータ再送が可能なものをソースとして想定する．
- **シンク**：シンクは DAG の最手段の処理を行うワーカから最終的な出力データを受け取る．なお，本研究では DB のように冪等性のあるシステムをシンクとして想定する．
- **DB**：DB では集約結果間の相関関係を表した確率モデルを管理する．なお，本来は時間の経過に伴う確率モデルの更新などを考慮する必要があるが，今後の課題とし本稿では扱わない．つまり，本稿では確率モデルが不変であると想定し，DB は事前に学習したモデルを保持し続ける．

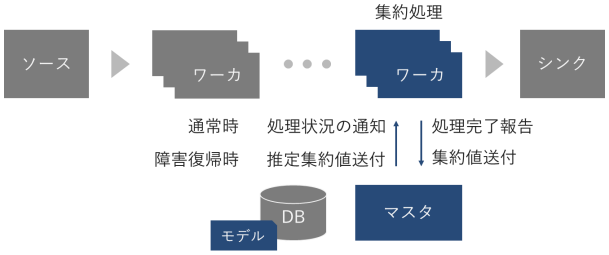

図 7　提案手法による耐障害性の保証

worker	key	time window [0, 10)	[5, 15)	[10, 20)	[15, 25)	[20, 25)	...
A	X_1	21	21	22	23	23	
	X_2	19	20	20	22	22	...
	X_3	20	20	21	22	23	
B	X_4	18	19	lost	lost	21	
	X_5	19	19	lost	lost	22	...
	X_6	21	22	lost	lost	23	

図 8　出力ストリームの例

- **マスタ**：マスタは各ワーカの処理の進行具合を監視すると共に，障害発生の把握や復帰のためのワーカ管理を行う．なお，本稿ではマスタはワーカの故障を非同期に必ず検知できると想定し，具体的な検知方法については議論しない．

4.2　障害復帰の流れ

前章で述べたように，集約処理を一つのみ含むような問合せを想定することで，DAG の最終段で行われる集約処理の結果を復元できればシステム全体の出力の正しさを保証できる．そこで本研究では，集約処理を行うワーカで障害が発生したとき，そのワーカの出力結果を他のワーカの出力結果から推定することでシステム全体の耐障害性を保証する．例えば，6 つのセンサ値について，窓幅 10，スライド幅 5 のスライディングウィンドウで平均値を計算する例を考える．このとき，得られる出力ストリームは図 8 のようになる．ただし，時刻 16 の時点でワーカ B に障害が発生し，時刻 19 で復帰したとする．障害により，ワーカ B は時間窓 $[10, 20)$ 及び $[15, 25)$ の集約値を出力できていない．一方，復帰以降の処理である時間窓 $[20, 25)$ に関しては正常な値を出力できる．つまり，時間窓 $[10, 20)$ 及び $[15, 25)$ の集約値をワーカ A の値から推定できれば，システム全体として正しい出力を保証できる．

approximately-once セマンティクスに基づく復帰処理の流れを以下に示す．

（1） 集約処理を行うワーカはシンクに出力した値をマスタにも送信する．なお，集約処理及びマスタへの集約値の送信は各ワーカで非同期に行われる点に注意する．

（2） マスタは図 8 のように各ワーカからの集約値を時間窓単位で保持する．また，全てのワーカで処理が完了した時間窓の集約値を破棄する．

（3） あるワーカの障害を検知した場合，そのワーカの処理を引き継ぐ代替ワーカを立て，復帰時刻以降の時間窓の処理を通常通り実行する．

（4） 障害により欠損した値を補完するために，マスタは DB が保持する確率モデル及び他のノードの集約値から欠損した値を推定し，代替ワーカに送信する．つまり，代替ワーカは通常の集約処理と推定された集約値のシンクへの出力を並行して行う．

（5） 障害により欠損が生じる時間窓への処理が終わった時点で復帰完了とみなし，正常運転へと移行する．

例として，図 8 の例における復帰処理を考える．まず，時間窓 $[5, 15)$ までの処理は正常に行われているため，マスタへの集約値の送信と処理完了による破棄が行われる．その後，時刻 16 でワーカ B において障害が発生したため，ワーカ A からの集約値はマスタに送信される一方，ワーカ B の集約値は送信されない．時刻 19 の時点でマスタはワーカ B の障害を検知し，代替ワーカ B’ を建て，時間窓 $[20, 25)$ 以降の処理を行うよう指定する．更に，ワーカ A から時間窓 $[10, 20)$ 及び $[15, 25)$ の集約値が送られてきた時点で，確率モデルを用いてワーカ B の集約値を推定し，代替ワーカ B’ に推定した値を送信する．代替ワーカ B’ はマスタから受け取った値を自身の集約値としてシンクに出力する．時間窓 $[15, 25)$ の出力が終わった時点で，それ以降代替ワーカ B’ は正常な処理が可能となるため，復帰が完了する．

5　効率的なワーカ割当の検討

本章では，効率的な耐障害性保証のための物理ノード（key）分割について検討する．本手法では損失した集約結果を推定することで障害復帰処理の効率化を図るが，この推定結果はユーザから与えられた誤差要求を保証する必要がある．そのため，効率的な耐障害性保証のためには，他のワーカの集約結果から誤差要求を満たした推定が可能なワーカをより多くするよう，物理ノードを各ワーカに割り当てなければならない．

5.1　最適化問題としての定式化

まず，最適なワーカ割当を求める問題を最適化問題として定める．

物理ノードの総数が n 個，ワーカの数が m 個であるとして，以下の条件を満たすワーカ割当 $\boldsymbol{A} = \{A_1, ..., A_m\}$ を考える．

$$\forall A_i \in \boldsymbol{A},\ A_i \subseteq \boldsymbol{X} \tag{4}$$

$$\forall A_i, A_j \in \boldsymbol{A}.\ A_i \cap A_j = \emptyset \tag{5}$$

$$\bigcup_{A_i \in \boldsymbol{A}} A_i = \boldsymbol{X} \tag{6}$$

ただし，今回は問題の簡単化のため，m が n の約数であり，各ワーカに対して同数（$|A_i| = \frac{n}{m}$）の物理ノードを割り当てる状況のみ考える．また，障害復帰処理が完了するまでは一つのノードでしか障害が発生しないものと仮定する．

物理ノードのワーカへの割当を考える上で重要な事実として，実際の観測値が推定精度に影響を与えないという点がある．式 (3) に示したように，推定値の信頼度 $R(Y|\boldsymbol{o})$ は事後ガウス分布 $\mathcal{N}(\mu_{Y|\boldsymbol{o}}, \Sigma_{Y|\boldsymbol{o}})$ の確率密度関数 P に対する範囲積分として計

算する．しかし，式 (2) に示したように，事後ガウス分布の分散 $\Sigma_{Y|\boldsymbol{o}}$ は観測値 $\boldsymbol{o}$ に依存しない．更に，復元後の値には期待値 $\mu_{Y|\boldsymbol{o}}$ を用いるとしているため，式 (3) の積分範囲は $\mu_{Y|\boldsymbol{o}}$ に依存しない．言い換えれば，期待値を 0 とした事後ガウス分布で範囲 $[0-\epsilon, 0+\epsilon]$ の積分を行えば同じ値が得られる．つまり，$R(Y|\boldsymbol{o})$ は実際には観測値 $\boldsymbol{o}$ に依存せず，推定に利用する物理ノード集合 O が決定した時点で計算できる．以降，このように計算する信頼度を $R(Y|O)$ と記述する．

以上の点を踏まえると，各ワーカ A_i の信頼度は以下のように計算できる．

$$R(A_i) = \min_{Y \in A_i} R(Y|\boldsymbol{X} \setminus A_i) \tag{7}$$

つまり，ワーカ A_i 内の各物理ノードの集約値 $Y \in A_i$ について，他のワーカの集約値から推定した際の最小の信頼度をワーカ A_i 全体の信頼度とみなす．

最後に，最適なワーカ割当 $\boldsymbol{A}^*$ を求める問題を，近似的に推定可能なワーカの数を最大化する最適化問題として定める．なお，ワーカが推定可能かどうかを判断するために，ユーザが信頼度のしきい値 $1-\delta$ を与えると想定する．このとき，本研究で求める最適なワーカ割当 $\boldsymbol{A}^*$ は以下の最適化問題の解である．

$$\begin{aligned} &\text{maximize} && |\boldsymbol{A}^*| \\ &\text{subject to} && \forall A_i \in \boldsymbol{A}^*,\ R(A_i) \geqq 1-\delta \\ & && (4) \wedge (5) \wedge (6) \end{aligned} \tag{8}$$

前述したとおり各 $R(A_i)$ の計算には観測値 $\boldsymbol{o}$ が不要であるため，最適なワーカ割当は事前に計算可能である．

なお，上記の最適化問題が示すように，他のワーカの集約値からの推定が不可能である（$R(A_i) < 1-\delta$ となる）ワーカも存在しうる．そのような場合は，既存のレプリケーションやチェックポインティングなどを用いて，そのような推定不可ワーカのみ頑健な耐障害性保証を行うものとする．

5.2 貪欲法に基づく解法

本稿では，貪欲法に基づくワーカ割当アルゴリズムを提案する．単純に考えて，式 (8) は可能なワーカ割当を全列挙することで最適解が求まる．しかし，そのような全探索による解法では計算量が物理ノード（key）数 n に対して指数関数的に増加し，現実的でない．そのため，貪欲法を用いて近似解を求める．

貪欲法のための指針として，ワーカへの物理ノード割当による信頼度の減少幅を用いる．本稿ではワーカは高々一つしか同時に故障しないと考えるため，推定時に使用できない物理ノードの集約値は同じワーカ内に割り当てられたもののみである．つまり，あるワーカ A_i に対して新たな物理ノード X_{new} を加えたとき，$X_j \in A_i$ の各信頼度は X_{new} を推定に使用できなくなった分だけ減少する．したがって，信頼度 $R(A_i)$ の減少が最も小さい割当を繰り返すことで，適切なワーカ割当の導出を目指す．

図 9 に貪欲法に基づくアルゴリズムを示す．前述したとおり，1 ワーカに割り当てるノードの上限は $\frac{n}{m}$ 個である．まず，3–9

Input: 集約値集合 $\boldsymbol{X}$, ワーカ数 m, 誤差上限 ϵ, 要求信頼度 $1-\delta$
Output: ワーカ割当 $\boldsymbol{A}$

```
1  A ← ∅
2  a_max ← n/m                          // 1 ワーカへの割当上限
   // 推定不可ノードを割当から切り分ける
3  X_alive ← {X_i | X_i ∈ X, R(X_i | X / X_i) ≧ 1 − δ}
4  A_dead ← ∅                           // 推定不可なノードを扱うワーカ
5  foreach X_i ∈ X \ X_alive do
6      A_dead ← A_dead ∪ {X_i}
7      if |A_dead| = a_max then
8          A ← A ∪ {A_dead}
9          A_dead ← ∅
   // 推定可能ノードをワーカに割り当てる
10 A_alive ← {∅_1, ..., ∅_{m−|A|−1}}    // 割当ワーカを初期化
11 X_alive を信頼度の降順にソート
12 foreach X_add ∈ X_alive do
       // 割当可能なワーカが存在するか確認
13     A_cand ← {A_i | A_i ∈ A_alive ∧ R(A_i ∪ {X_add}) ≧ 1 − δ}
14     if A_cand ≠ ∅ then                // 割当が可能な場合
           // 信頼度の減少が最も少ないワーカへ割り当て
15         A_i ← arg min_{A_i ∈ A_cand} R(A_i) − R(A_i ∪ {X_add})
16         A_i ← A_i ∪ {X_add}
17         if |A_i| = a_max then
18             A ← A ∪ {A_i}
19             A_alive ← A_alive \ {A_i}
20     else                              // 割当が不可能な場合
21         A_dead ← A_dead ∪ {X_add}
22         if |A_dead| = a_max then
23             A ← A ∪ {A_dead}
24             A_dead ← arg min_{A_i ∈ A_alive} R(A_i)
25             A_alive ← A_alive \ {A_dead}
```

図 9　貪欲法に基づく割当アルゴリズム

行目では確率モデルによる推定が不可能なノード，つまりいずれのノードの集約値を用いても信頼度がしきい値を満たさないノードを割当から切り分ける．言い換えれば，推定できないノードを扱うワーカ A_{dead} を用意し，推定不可ノードは全てそこへ入れる．10–25 行目では，残りの推定可能なノードをワーカへと実際に割り当てる．まず，現在の各割当 $A_i \in \boldsymbol{A}_{alive}$ に対し追加ノード X_{add} を加えたとき，要求信頼度を満たす割当が得られるか確認する（13 行目）．割当可能であれば，信頼度の減少が最も小さいワーカへ追加ノードを割り当てる（15–16 行目）．一方，割当不可能，つまり追加ノードが推定不可能なノードである場合は推定不可ワーカ A_{dead} へ割り当てる（21 行目）．この処理を全ての推定可能ノード $\boldsymbol{X}_{alive}$ に対して行う．なお，ノードをワーカへ割り当てる際，ワーカのノード数上限に至った場合は割当完了としてワーカ割当 $\boldsymbol{A}$ を更新する（17–19, 23–25 行目）．

6 実　　験

本章では，提案手法の損失集約結果の推定能力，及びノード

分割手法の性能について実験により評価する．人工データを用いた実験により，本手法は誤差要求を満たした高精度な推定が可能であることを示す．また，確率モデルを考慮してノード分割する本手法は，ランダムな分割に比べて損失集約結果の誤り率を大きく抑えられることを確認する．

実験に用いたデータセットや評価指標など実験設定について 6.1 章にまとめる．6.2 章では，損失集約結果の推定能力，及びノード分割手法の性能の評価実験の結果を述べる．

6.1 実験設定

提案手法の評価実験において使用するデータセット，及び評価方法について以下にまとめる．

6.1.1 データセット

入力データに対して適切なモデルが作成できたとき，提案手法が効率的なノード分割と誤差要求を満たした高精度な推定ができることを確認するため，人工データセットを用いて実験を行う．

本実験では平均が 0，分散が 1 であるような 120 次元の人工データセットを用いる．このデータセットは，それぞれ 12 次元ずつが 0.9 の共分散を持ち，それ以外の属性に対しては共分散が 0 となるような共分散行列から機械的に生成した 1000 のデータ系列から成る．つまり，各属性は一部の属性のみと強い相関を持つようなデータセットとなる強い相関を持つデータは集約値の推定の際に有用であることから，適切なノード分割ができている際には，これらの属性の集約値を考慮した高精度な推定が期待できる．

また，データ生成に用いた平均ベクトルと共分散行列をモデルとして使用することで，入力データに対するモデルを正しく学習できた場合での提案手法の性能評価を実現する．

6.1.2 評価指標

本実験では推定可能なワーカ数によってノード分割の性能を，推定の誤り率によって損失集約結果の推定能力をそれぞれ評価する．ただし，各ワーカ A_i のある真の集約結果 y が他のワーカの集約結果からの推定値 $\bar{y}$ の $\pm\epsilon$ の範囲に無い場合に誤りであるとする．よって，あるウィンドウ $W_i \in W$ での誤って推定した集約結果数を err_i とするとき，誤り率 err は次式の通り表される．

$$err = \frac{1}{|W||X|}\Sigma_i err_i \tag{9}$$

上式において，X は推定可能な集約値集合であるとする．これは，信頼度が $R(A_i) < 1-\delta$ となるワーカは，誤差要求を満たした推定が不可能であるため，今回の評価対象から除外するためである．

本実験で用いるクエリは，各属性に対するスライド幅 10 分，窓幅 30 分のスライディングウィンドウでの平均処理とする．また，パラメータとして誤差上限 ϵ，ノード数 m を変化させて実験を行う．なお，信頼度は実験を通じて $1-\delta = 0.95$ を使用する．

6.1.3 比較対象

本実験では，比較対象としてヒューリスティックに基づいた二つのノード分割を用いる．

- **h-best**：相関の高い属性をなるべく別のワーカに分散させたノード分割．各ワーカの集約値を推定する際，相関の高い属性の値をより多く参照できることから，本実験では最適なノード分割の代わりに使用する．
- **h-worst**：相関の高い属性をなるべく同じワーカにまとめたノード分割．各ワーカの集約値を推定する際，参照可能な相関の高い属性数が少なくなることから，本実験では最悪なノード分割の代わりに使用する．

6.2 実験結果

使用ワーカ数及び誤差上限を変化させた際の推定可能ワーカ数の推移を図 10，図 11 にそれぞれ示す．実験の結果，提案した貪欲的なノード分割手法は，最適な分割 h-best には及ばないものの，効率的なノード分割ができているといえる．また，使用ワーカ数に対して推定可能ワーカ数がスケールしている．

次に，使用ワーカ数及び誤差上限を変化させた際の誤り率の推移を図 10，図 11 にそれぞれ示す．提案したノード分割は誤り率についても，最適な分割 h-best には及ばないものの，非常に高精度に推定できているといえる．また，すべての設定において誤り率が 0.05 を超していないことから，誤差要求を満たした推定ができていると分かる．

本実験の結果から，提案手法により，効率的なノード分割及び，誤差要求を満たした確率モデルに基づく損失集約値推定が実現できているといえる．しかし，今回の実験ではある一つの特殊な条件下での人工データによる実験であるため，更なる調査実験が求められる．具体的には，(1) 実際のセンサデータを用いた実験により，実利用が可能かの調査や，(2) 人工データにより様々な条件下での実験を行うことで提案手法の詳細な性質の調査などが考えられる．今後は，これらの実験を行っていきたいと考えている．

7 まとめと今後の課題

本稿では，データストリームの集約処理を対象とした，確率的モデルに基づく効率的な耐障害性保証手法を提案した．本手法では，ノード障害などにより損失した集約値を他のノードの集約値から推定することで障害復帰するため，チェックポインティングや入力データの再処理が不要になる．

また，効率的な障害復帰のためのノード分割手法も提案した．本手法は，障害時に他のノードの集約結果から誤差要求を満たした推定ができるノードをより多くするようなノード分割を，貪欲的なアプローチに基づいて決定する．

提案手法の損失集約結果の推定能力，及びノード分割手法の性能について，人工データを用いた実験によって評価した．実験の結果，提案手法により，効率的なノード分割及び，誤差要求を満たした確率モデルに基づく損失集約値推定が実現できていることが分かった．

今後の課題としては，実データや異なる条件下での人工データを用いた実験により，提案手法の実利用性や詳細な性質の調

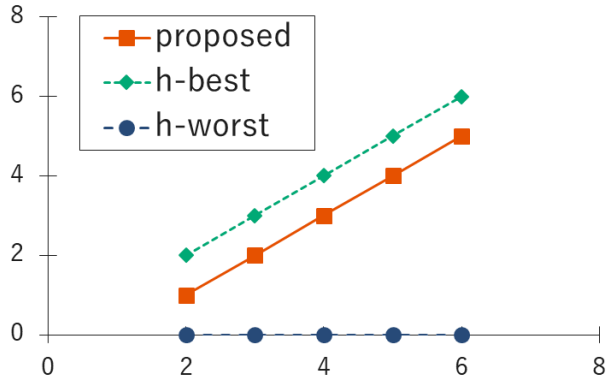

図 10　使用ワーカ数に対する推定可能ワーカ数の変化

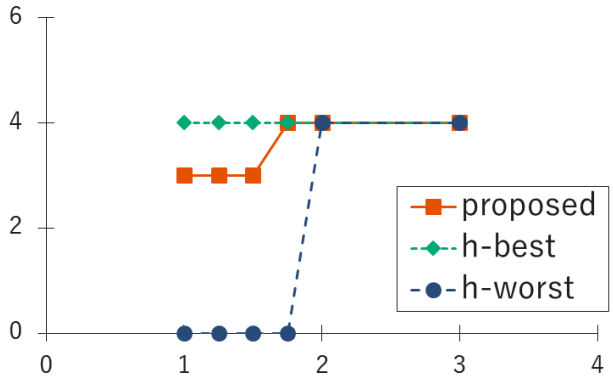

図 11　誤差上限に対する推定可能ワーカ数の変化

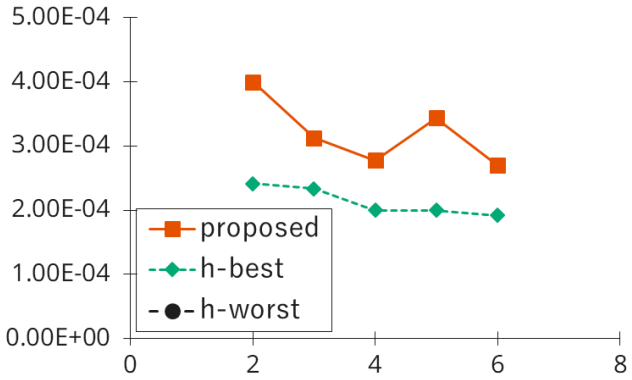

図 12　使用ワーカ数に対する誤り率の変化

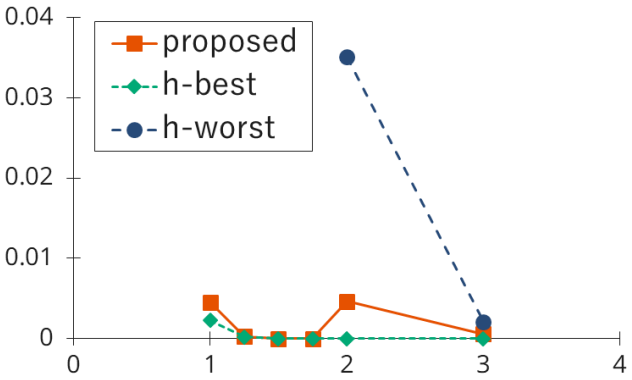

図 13　誤差上限に対する誤り率の変化

査が考えられる．また，今回はクエリ自体は単純に処理した場合の耐障害性保証について提案したが，クエリ処理にも近似的問合せ処理を用いた際の効率的な耐障害性保証についても今後検討していきたい．

謝　　辞

本研究は，JSPS 科研費（16H01722, 18H06461）の助成，および，国立研究開発法人新エネルギー・産業技術総合開発機構（NEDO）の委託業務による．

文　　献

[1] P. Carbone, S. Ewen, G. Fóra, S. Haridi, S. Richter, and K. Tzoumas, “State Management in Apach Flink,” in *Proc. VLDB Endowment*, vol. 10, pp. 1718–1729, 2017.

[2] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, “Discretized Streams: Fault-Tolerant Streaming Computation at Scale,” in *Proc. SOSP*, pp. 423–438, 2013.

[3] S. A. Noghabi, K. Paramasivam, Y. Pan, N. Ramesh, J. Bringhurst, I. Gupta, and R. H. Campbell, “Samza: Stateful Scalable Stream Processing at LinkedIn,” in *Proc. VLDB Endowment*, vol. 10, pp. 1634–1645, 2017.

[4] T. Akidau, A. Balikov, K. Bekiroğlu, S. Chernyak, J. Haberman, R. Lax, S. McVeety, D. Mills, P. Nordstrom, and S. Whittle, “MillWheel: Fault-Tolerant Stream Processing at Internet Scale,” in *Proc. VLDB Endowment*, vol. 6, pp. 1033–1044, 2013.

[5] Q. Huang and P. P. C. Lee, “Toward High-Performance Distributed Stream Processing via Approximate Fault Tolerance,” in *Proc. VLDB Endowment*, vol. 10, pp. 73–84, 2016.

[6] A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, and W. Hong, “Model-based approximate querying in sensor networks,” *The VLDB Journal*, vol. 14, no. 4, pp. 417–443, 2005.

[7] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, “S4: Distributed Stream Computing Platform,” in *KDCloud*, pp. 170–177, 2010.

[8] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, N. Bhagat, S. Mittal, and D. Ryaboy, “Storm @Twitter,” in *Proc. SIGMOD*, pp. 147–156, 2014.

[9] “Apache kafka.” https://kafka.apache.org/.

[10] D. Chu, A. Deshpande, J. M. Hellerstein, and W. Hong, “Approximate Data Collection in Sensor Networks using Probabilistic Models,” in *Proc. ICDE*, pp. 48–59, 2006.

[11] M. Armbrust, T. Das, J. Torres, B. Yavuz, S. Zhu, R. Xin, A. Ghodsi, I. Stoica, and M. Zaharia, “Structured Streaming: A Declarative API for Real-Time Applications in Apache Spark,” in *Proc. SIGMOD*, pp. 601–613, 2018.