複合的ストリーム処理に対するトレーサビリティの研究

山田　真也† 　北川　博之†† 　SalmanAhmed Shaikh††† 　天笠　俊之†††† 　的野　晃整†††

† 筑波大学大学院 理工情報生命学術院 システム情報工学研究群　〒 305-8573 茨城県つくば市天王台 1 丁目 1-1
†† 筑波大学 国際統合睡眠医科学研究機構　〒 305–8575 茨城県つくば市天王台 1 丁目 1-1
††† 産業技術総合研究所 人工知能研究センター　〒 135-0064 東京都江東区青海 2-4-7
†††† 筑波大学 計算科学研究センター　〒 305–8577 茨城県つくば市天王台 1 丁目 1-1
E-mail: †yamada@kde.cs.tsukuba.ac.jp, ††kitagawa@cs.tsukuba.ac.jp,
†††{shaikh.salman,a.matono}@aist.go.jp, ††††amagasa@cs.tsukuba.ac.jp

あらまし　データ来歴（Lineage）とは分析結果の元になった入力データを提示することを指し，分析処理のトレーサビリティを保証するため活用されている．しかし，近年の AI や機械学習モデルを用いた処理（AI/ML 処理）を伴う分析を考えると Lineage だけでは問題がある．これまで著者らは，AI/ML 処理を伴う複合的データ分析処理では単に Lineage を提示するだけでなく，AI/ML 処理の分析結果の導出理由を説明する判断根拠も併せて提示する必要があることに着目し，拡張来歴について研究してきた．しかしながら従来の拡張来歴は静的なデータモデルである関係モデルに基づいており，近年広く活用されているストリーム処理に対して適用できないという問題があった．そこで本稿では，複合的ストリーム処理に対して拡張来歴を導出する枠組みについて説明する．

キーワード　拡張来歴，ストリーム処理，Apache flink，AI

1 はじめに

スマートフォンやセンサ等の IoT 機器の普及により，日々，膨大なストリームデータが生成されている．ストリームデータからリアルタイムに知識を抽出するために，ストリーム処理エンジンが広く用いられているが，そうして抽出されたデータや知識を信頼して活用するためには，データ分析のトレーサビリティが不可欠である．そのためには分析処理の利用者になぜその分析結果が得られたかを説明することが重要である．

分析結果のトレーサビリティはこれまで，データベースシステム，科学ワークフロー，ビッグデータ処理システムなど様々な領域で研究が行われてきている [1], [2], [3]．その中でも，データ来歴（Lineage）は分析結果のもとになった入力データを提示することを指し，特にデータベースの分野において研究が行われてきた [4], [5]．さらにストリーム処理を対象にした Lineage 導出システムの検討も始まっている [6], [7].

近年，社会で活用されているデータ分析処理は AI や機械学習モデルを用いた処理（AI/ML 処理）を伴うことが増えており，ストリーム処理においても同様の傾向がある．そうした複合的ストリーム処理では，AI/ML 処理の判断根拠が分析結果の導出理由を説明するために重要な情報であるため，従来の入力データのみを提示する Lineage だけでは十分にトレーサビリティを保証することができない問題がある．ここで以下のような例を考える．

［例 1］ 図 1 はストリーム処理を用いたリアルタイムレビュー分析を示している．この分析処理は 4 つの属性（製品 ID，顧客 ID，レビューコメント，タイムスタンプ）から構成されるタプルを受け取ると，タプルの検証，感情分類モデルの適用，製品ごとの Positive/Negative 数のカウントを行ったのち，最後に分析結果を次のシステムに送信する．

ここであるユーザが分析結果 $\langle 1, Negative, 3\rangle$ がなぜ導出されたのか知りたいと考えたとする．従来の Lineage を用いると，$\{\langle 1, C001,$ “Wrong one ...” $, 10000\rangle$, $\langle 1, C002,$ “This toy ...”, $10005\rangle$, $\langle 1, C003,$ “Some parts ...” $, 10010\rangle\}$ の 3 タプルをその分析結果の元になったデータとして提示することができる．しかしこの 3 つのタプルを提示するだけでは，なぜ感情分類モデルがそれらのタプルを Negative と判定したのか理解することができないという問題が生じる．つまりこのことは複合的ストリーム処理において，従来の Lineage だけではトレーサビリティを保証することができないという問題を表している．

この問題を解決するために本研究では，感情分類モデルがなぜそれらのタプルのコメントを Negative と判断したのか説明するために，AI/ML 処理の判断根拠を併せて提示するというアプローチをとる．具体的には「タイムスタンプが 10000 の入力タプルは 1 ワード目に “Wrong” という単語があったため Negative と判定し，10005 の入力タプルは 4 ワード目に “broken” という単語があったため Negative と判定し，10010 の入力タプルは 4–5 ワード目に “not arrived” があったため Negative と判定した」という判断根拠の情報を，従来の Lineage と併せて提示することで複合的ストリーム処理におけるトレーサビリティの保証を可能にする．

著者らはこれまで AI/ML 処理を伴う複合的データ分析処理に対して，通常の Lineage に加えて AI/ML 処理の判断根拠を提示する拡張来歴について研究してきた [8], [9]．しかしながら従来の拡張来歴は，関係データやファイルデータのような静的なデータモデルを前提としており，動的なストリーム処理に対

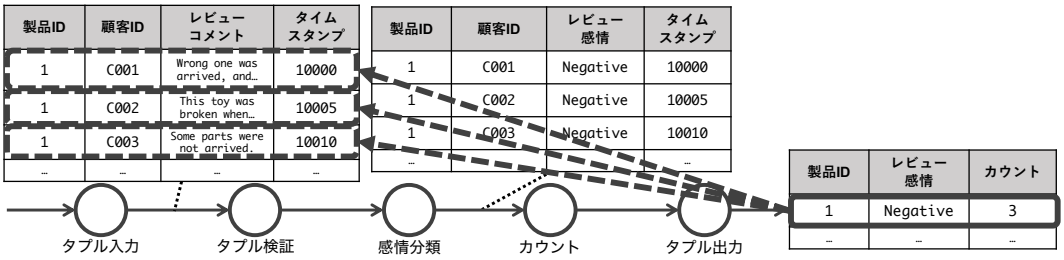

図 1　複合的ストリーム処理の例．感情分類モデルを用いたリアルタイムレビュー分析．点線で囲われているタプルは分析結果 $\langle 1, Negative, 3\rangle$ の Lineage を表す．

して適用することができないという問題があった．

そうした背景から著者らは複合的ストリーム処理のトレーサビリティを保証するため，ストリーム処理を対象に拡張来歴を導出する枠組みを提案した [10]．本稿ではその内容に基づき複合的ストリーム処理に対して拡張来歴を導出するシステムを説明するが，1) 導出手法をより理解しやすい例の使用，2) 導出アルゴリズムの詳細な説明の追加，3) Traversal 時間に関する実験データの追加，4) 従来の分析結果のより適切な統計的手法を用いた比較 の 4 点で拡張を行っている．本稿で説明するシステムは，ストリーム処理に対して Lineage 導出を行うフレームワーク GeneaLog をベースとした．Apache Flink 上で実行される複合的ストリーム処理に対して拡張来歴を導出するプロトタイプシステムの開発を行い，プロトタイプシステムを用いた評価実験を行った．実験では本稿で説明したシステムがレイテンシ／スループットの点で非常に小さいコストで拡張来歴を導出することを示す．

本稿の構成は以下の通りである．まず 2 節で関連研究について説明する．次に 3 節で本稿が使用するストリーム処理モデルと GeneaLog の説明を行い， 4 節でストリーム処理に対して拡張来歴を導出する手法を説明する．そして 5 節でプロトタイプシステムを用いた評価実験の結果を示す．最後に 6 節で結論と今後の課題を述べる．

2　関 連 研 究

2.1　データベース問い合わせにおける来歴研究

これまでデータベース問い合わせの来歴を導出する手法がいくつも提案されている [5], [11], [12]. [5], [11] は関係データモデルでモデル化された問い合わせに対して，タプルレベルの来歴を導出する研究である. [5] は Tracing Query と呼ばれる逆クエリを用いて分析結果の Lineage を求める手法であり, [11] は全ての入力タプルに識別子を事前に割り当て，識別子を分析結果まで継承することで元になったタプルを提示する手法である．より一般的な問い合わせに対する来歴研究も存在しており, [12] は XQuery に対して来歴を導出する枠組みを提案している．さらに MapReduce や Spark のようなビッグデータ処理システムにおける来歴を対象とするものも存在する [3], [13]．しかしながらこれらの枠組みは単に分析結果の元になった入力データを提示するもので，AI/ML 処理を伴う複合的データ分析処理に対して十分なトレーサビリティを保証することができない．こうした背景から著者らは [8], [9] で従来の Lineage に加えて AI/ML 処理の判断根拠を併せて提示する拡張来歴を提案した．しかしながら [8], [9] で提案した枠組みは静的データに対する問い合わせを対象としており，ストリーム処理には適用することができないという問題がある．

2.2　ストリーム処理における来歴研究

ストリーム処理に対して来歴を導出する枠組みの研究はまだ十分に行われていない．初期の研究として [14] は，出力ストリームにどの入力ストリームが貢献しているのかを提示することができるが，提示可能な来歴の粒度が非常に粗いという問題がある．そのためこの手法では，どのタプルが分析結果に貢献したのか提示することはできない. [15] はタプルのタイムスタンプに基づいて元になった入力タプルを提示する研究であるが，全てのオペレータに対して入力タプルと出力タプルの対応関係を表すルールを定義する必要がある．しかしながら，実行する分析処理に応じて多様なオペレータが組み合わせられる近年のストリーム処理において，全ての入出力関係をあらかじめ定義することは容易でない．

[6] と [7] はよりモダンなストリーム処理エンジンに対して Lineage を導出する研究である．Ariadne [6] は処理を記述する各オペレータを拡張し，導出されるタプルに元になったタプルを識別するアノテーションを付与することで Lineage を求める枠組みである．しかしながら [7] でも示されているように，タプルに付与されるアノテーションのサイズがタプルに貢献した入力タプルの数に比例するため，分析処理によっては非常に大きなオーバヘッドが発生することがある．そうした背景から GeneaLog [7] ではタプルにアノテーションを付与する際，アノテーションのサイズを，入力タプル数に依存しない固定サイズにする手法を提案した．さらに評価実験を通して GeneaLog が Ariadne より小さいオーバヘッドで Lineage を導出することを示した．このことから本稿の手法の実装は GeneaLog に基づいている．GeneaLog の概要については次節で説明する．

3　ストリーム処理モデルと Lineage

3.1　ストリーム処理モデル

ストリーム S は無限長のタプルのシーケンスで構成され，タプル t はタイムスタンプ τ と属性値をもつ ($t = \langle \tau, a_1, \ldots, a_m\rangle$)．ストリーム処理はノードがオペレータ，エッジがデータフロー

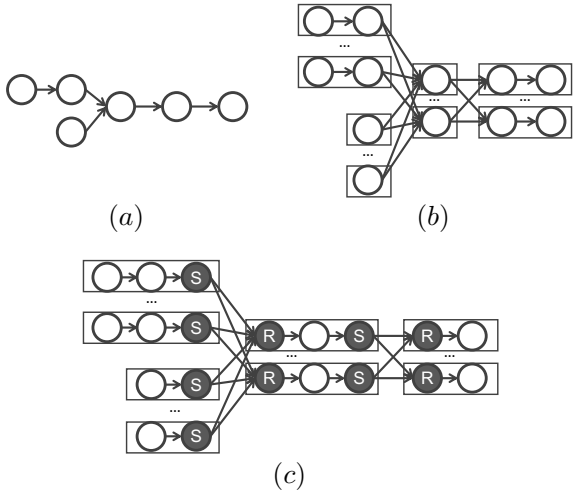

図 2　ストリーム処理フローの表記．(a) 論理表記．(b) 物理表記．(c) Send (S) / Receive (R) オペレータの導入．

の DAG 構造として表すことができ，オペレータは 1（または 2）入力ストリームを受け取ると 1 出力ストリームを出力する．

ストリーム処理を構成するオペレータはいくつも存在するが，それらは 3 種類に分類できる [7]．1 つ目は I/O で，具体的なオペレータしては Source と Sink オペレータがこれに該当する．これらのオペレータは，ストリーム処理に対して外部のシステムからデータを取り込む，または外部のシステムにストリーム処理結果を送信する役割を持つ．2 つ目は Transformer で，具体的には Map や Filter，Union，Aggregate，Join オペレータが該当する．ここで Aggregate や Join オペレータは，入力ストリームをウインドウに分割し，それぞれのウインドウごとに集約/結合処理を実行するようなオペレータを指している．3 つ目は Send と Receive オペレータで，これらはスレッド間でタプルをやり取りするために用いられる．

図 2(a) は，2 つの Source オペレータ，3 つの Transformer，1 つの Sink オペレータで構成されるストリーム処理を表している．この表記はストリーム処理を論理的に記述したもので，本稿ではストリーム処理の論理表記と呼ぶ．しかしストリーム処理が実際にストリーム処理エンジン上で実行されるとき，各オペレータは複数のオペレータインスタンスに展開され，それぞれが並列に処理を行う．図 2(b) は展開されたオペレータインスタンスを表しており，ストリーム処理の物理表記と呼ぶ．ストリーム処理エンジンではスレッドが処理の実行単位になり，図中の四角が 1 スレッドを指している．実行するストリーム処理が複数スレッドで実行される場合には，タプルをスレッド間でやり取りする必要がある．そのために Send オペレータがスレッドの最後のオペレータの後ろに挿入され，Receive オペレータがスレッドの先頭に挿入される (図 2(c))．Send と Receive オペレータはタプルをスレッド間でやり取りする際，タプルのシリアライズ／デシリアライズも併せて行う．

3.2　GeneaLog

本節で，本稿の拡張来歴導出手法や実装のベースになっている GeneaLog [7] を説明する．GeneaLog はストリーム処理に対して Lineage を導出する State-of-the-art の枠組みで，各オペレータが生成する全てのタプルにメタデータを追加する．そして Lineage を求める際にはこのメタデータを手がかりに，元になった入力タプルを導出する．Lineage を導出するためにはこのメタデータをセットするようにオペレータを拡張する必要があり，これを Operator instrumentation と呼ぶ．これから GeneaLog の概要を説明する．

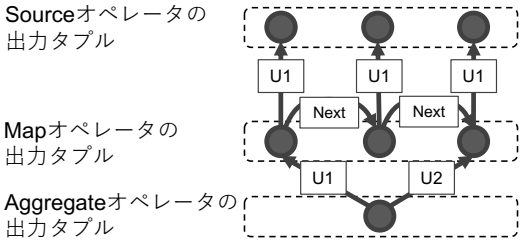

図 3　貢献グラフ

メタデータ: 全てのタプルに追加されるメタデータは，*Type*, *U1*, *U2*, *Next* の 4 つの属性で構成されるデータオブジェクトである．Type はメタデータが生成されたオペレータを識別する属性である．U1，U2，Next はポインタで，オペレータの出力タプルに貢献したオペレータの入力タプルを指す．

Operator Instrumentation: GeneaLog で拡張された処理フローの各オペレータは，単にタプルを処理するだけでなく，処理結果の Lineage が求められるようメタデータを適切にセットする必要がある．ここでは Aggregate オペレータを例にとって説明を行う．

[例 2]　ウインドウに含まれるタプル $t_{in_1}, \ldots, t_{in_n}$ を受け取ると t_{out} を返す Aggregate オペレータを考える．Lineage 導出のために拡張された Aggregate オペレータは，通常の集約結果を求めると同時に下記のように入力／出力タプルのメタデータをセットする．

（1）　$t_{out}.Type = Aggregation$

（2）　$t_{out}.U1 = t_{in_n}$，$t_{out}.U2 = t_{in_1}$

（3）　$t_{in_i}.Next = t_{in_{i+1}}$ $(i = 1, \ldots, n-1)$

他のオペレータに対する Operator instrumentation は，基本的に Type と元になった入力タプルに U1 と U2 ポインタをセットするよう拡張する．詳細は [10] を参照いただきたい．

貢献グラフを用いた Lineage 導出: Operator instrumentation で拡張されたストリーム処理の分析結果 t は，直前のオペレータにおける入力タプルを，t のメタデータに含まれるポインタで参照することができる．さらにその入力タプルもまた，直前のオペレータにおける入力タプルをポインタで参照できる．そのため Lineage はタプルのポインタを再帰的に辿ることで得ることができる．例えば図 1 に示したストリーム処理の分析結果は，図 3 のように，タプルがノード，ポインタがエッジのグラフ構造を得ることができる．これを貢献グラフと呼ぶ．図 3 の下のノードは分析結果 $(\langle 1, Negative, 3\rangle)$ を表し，そこからポインタでつながるノードは Map（感情分類）オペレータの出力タプル，一番上のノードは Source オペレータの出力タプ

ル（分析処理における入力タプル）を表している．そのため貢献グラフを用いて Lineage を求めるためには，辿ったタプルの Type が Source になるまで再帰的にグラフを辿ればよい．

クラスタ環境における Lineage 導出: クラスタ環境のように 1 つのストリーム処理が複数のスレッドにまたがって実行されるケースで Lineage を導出する際には，スレッド間通信についても考慮する必要がある．タプルが異なるスレッドに送られた場合，そのタプルの U1，U2 ポインタは入力タプルを指し示すポインタとして無効になるという問題がある．なぜならば U1 と U2 はタプルを送信したスレッドが持つメモリ領域を参照するポインタであるが，その領域はタプルを受け取ったスレッドから参照することができないからである．これを解決するために，GeneaLog では Explicit アプローチと Implicit アプローチの 2 つを提案しているが，本研究では Implicit アプローチを採用しており，以下の説明もそれに基づいている．

Send と Receive オペレータはスレッド間でタプルをやり取りするためのオペレータである．Implicit アプローチでは Send オペレータがタプルを送信するとき，まず送信タプルの Type を “REMOTE” にセットする．そして送信タプルの貢献グラフを辿り送信タプルの Lineage を求めた上で，送信タプルと Lineage をシリアライズして送信する．Receive オペレータではタプルと Lineage を受け取ると，下記で示すようにタプルのメタデータをセットし，貢献グラフを再構築する．

Receive オペレータが，タプル t_{out} とその入力タプル集合（Lineage） $t_{s_1}, \ldots, t_{s_n}$ を受け取る場合を考える．このとき，Receive オペレータは下記のように t_{out} のメタデータをセットし，貢献グラフを再構築する．そして t_{out} が次のオペレータに送られる．

（1） $t_{out}.U1 = t_{s_n}$，$t_{out}.U2 = t_{s_1}$

（2） $t_{s_i}.Next = t_{s_{i+1}}$ $(i = 1, \ldots, n-1)$

4 ストリーム処理における拡張来歴

4.1 定　　義

本節ではストリーム処理における拡張来歴を定義する．分析結果 t の拡張来歴とは，Source lineage と Reasoning lineage の 2 種類の Lineage のペアのことである．Source lineage とは分析結果 t の導出に貢献した入力タプルの集合のことであり，これは従来の Lineage と同じものを指す．一方，Reasoning lineage とは分析結果 t を導出したときの AI/ML 処理の判断根拠オブジェクト（Reasoning Lineage Object, RLO）の集合のことである．具体的には RLO は 4 つの属性で構成される：OPid，Input，Output，Reason．OPid は AI/ML 処理を実行し判断根拠を生成したオペレータを指す識別子で，Input と Output はそれぞれオペレータの入力タプルと出力タプルを指す．Reason は入力タプルからなぜ AI/ML 処理が出力タプルを導出したのかの説明を保存する属性である．例えば，図 1 に示したストリーム処理の分析結果 $\langle 1, Negative, 3\rangle$ の拡張来歴は図 4 になる．Reasoning lineage を用いることで，どのオペレータがどの入力データからどのような判断根拠で分析結果を導出したのかを提示することができるようになる．

Reasoning lineage を提示するためには，AI/ML 処理の設計者が AI/ML 処理結果の判断根拠をシステムに登録するための機能を提供している必要がある．そこでプロトタイプシステムの実装では，AI/ML 処理の設計者が AI/ML 処理結果の判断根拠を簡単に登録するためのインタフェースを提供しており，その詳細は 4.2 節で述べる．

4.2 拡張来歴の導出

本節では GeneaLog をベースに拡張来歴を導出するシステムの説明を 1) 追加メタデータ， 2) Operator instrumentation, 3) 拡張貢献グラフを用いた拡張来歴導出の 3 つに分けて行う．説明を簡単にするため，本稿では AI/ML 処理は Map オペレータのみで実行すると仮定しているが，実際には説明する手法はその他のオペレータで AI/ML 処理が実行されても拡張来歴を導出可能である．

追加メタデータ: 拡張来歴を導出するために，新しいメタデータ RL を従来のメタデータに追加する．RL は Reasoning lineage object 集合を指すポインタである．前述したように Reasoning lineage object は 4 つの属性 (OPid, Input, Output, Reason) で構成されるデータオブジェクトであり，Reasoning lineage の構成要素である．

Operator Instrumentation: RL は Map 処理で AI/ML 処理が実行されるとき，他のメタデータと同様に下記のようにセットされる．

Instrumented Map: 入力タプル t_{in} を受け取り，t_{in} に AI/ML 処理を適用したのち t_{out} を返す Map オペレータを考える．$t_{out}.v$ は AI/ML 処理によって導出された分析結果を表し，そのときの判断根拠を $REASON$ を表記する．このとき Map オペレータは下記のように出力タプルのメタデータをセットする．

（1） $t_{out}.Type = Map$

（2） $t_{out}.U1 = t_{in}$

（3） $t_{out}.RL = \{\langle Map, t_{in}, t_{out}.v, REASON\rangle\}$

実装システムは，AI/ML 処理の判断根拠を RL に登録するためのインタフェースをストリーム処理の開発者に提供している．このインタフェースを利用することで，ストリーム処理フローの開発者は次の 1 行を Map オペレータの関数定義に追加するだけで，AI/ML 処理実行時の判断根拠を登録することができる．

```
// t_in/t_out: Map オペレータの入力/出力タプル
// output/reason: AI/ML 処理の実行結果/判断根拠
Reason.registerReasonInfo("Map", t_in,
                          output, reason, t_out);
```

拡張来歴を求めるために，Send と Receive オペレータの拡張も併せて行う．Send オペレータは Lineage の代わりに拡張来歴を求め送信し，Receive オペレータは従来のメタデータのセットに加えて RL に Reasoning lineage object 集合を登録する処理を追加で行う．

$$\left[\left\{ \begin{array}{l} \langle\ 1,\ \text{C001},\ \text{“Wrong one ...”},\ 10000\ \rangle \\ \langle\ 1,\ \text{C002},\ \text{“This toy ...”},\ 10005\ \rangle \\ \langle\ 1,\ \text{C003},\ \text{“Some parts ...”},\ 10010\ \rangle \end{array} \right\}, \left\{ \begin{array}{l} \langle\ \text{Map},\ \langle\ 1,\ \text{C001},\ \text{“Wrong one ...”},\ 10000\ \rangle, \\ \quad \text{Negative},\ \text{“1st word is ‘Wrong’.”}\ \rangle \\ \langle\ \text{Map},\ \langle\ 1,\ \text{C002},\ \text{“This toy ...”},\ 10005\ \rangle, \\ \quad \text{Negative},\ \text{“4th word is ‘broken’.”}\ \rangle \\ \langle\ \text{Map},\ \langle\ 1,\ \text{C003},\ \text{“Some parts ...”},\ 10010\ \rangle, \\ \quad \text{Negative},\ \text{“4–5th words are ‘not arrived’.”}\ \rangle \end{array} \right\} \right]$$

図 4　分析結果 $\langle 1, Negative, 3\rangle$ の拡張来歴

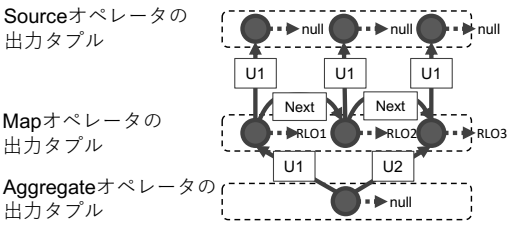

図 5　拡張来歴のための拡張貢献グラフ

拡張貢献グラフを用いた拡張来歴導出: メタデータとして RL を加えたことで従来の貢献グラフに RL が追加される．具体的には，図 1 の分析結果 $\langle 1, Negative, 3\rangle$ の貢献グラフは，図 5 のように従来の貢献グラフに RL ポインタが追加されたようなデータ構造になる．これを拡張貢献グラフと呼ぶ．

この拡張貢献グラフに対して，アルゴリズム 1 を適用することによって拡張来歴を求めることができる．このアルゴリズムは，RL が Reasoning lineage object を参照していない場合 null 値を取ることを仮定している．$enqueueIfNotVisited$ はタプル，キュー q，タプル集合 $visited$ を受け取ると，タプルが $visited$ に含まれているかを確認し，もしタプルが $visited$ に含まれていなければ（そのタプルにまだアクセスしたことがなければ），キューにタプルをエンキューする関数である．またタプルが既に含まれていた場合には何も行わず，そのまま処理がアルゴリズムに戻ってくる．このアルゴリズムは GeneaLog のものに基づいており，拡張来歴を求めるための拡張は 4 行目，5 行目である．ここではアクセスしたタプルが Reasoning lineage object を持っていればそれを RL に追加する処理を行なっている．この拡張によってアルゴリズムは拡張来歴を導出することができるようになる．

［例 3］ アルゴリズム 1 を用いて，図 1 の分析結果 $\langle 1, Negative, 3\rangle$ （対応する拡張貢献グラフは 図 5）の拡張来歴を求める様子を説明する．まず 1 行目で q を $\{\langle 1, Negative, 3\rangle\}$ で初期化し，他変数は空集合で初期化する．すると q には要素があるため，3 行目で $t = \langle 1, Negative, 3\rangle$ となる．t の RL は null であり $t.Type$ は Aggregate であるため 13 行目に移り，19 行目までの処理の間で q に $int_3 = \langle 1, C003, Negative, 10010\rangle$，$int_2 = \langle 1, C002, Negative, 10005\rangle$，$int_1 = \langle 1, C001, Negative, 10000\rangle$ がエンキューされる．次に 2 行目に戻ると $t = int_3$ となり，このタプルは RL が RLO3 を指しているため 5 行目で RL に RLO3 を追加する．その後 8 行目に制御が移り，$\langle 1, C003, ..., 10010\rangle$ が q にエンキューされる．int_2 と int_1 についても同様に行うと，RLO2 と RLO1 が RL に追加される．その時点で q には $\langle 1, C003, ..., 10010\rangle$，$\langle 1, C002, ..., 10005\rangle$，$\langle 1, C001, ..., 10000\rangle$ の 3 タプルがあり，それらの Type はすべて Source である．そのためこれらは 7 行目で SL に追加され，その後 q は空であるから SL と RL が返される．結果として拡張来歴が導出され，その内容は図 4 に示したものとなる．

アルゴリズム 1 拡張来歴の導出

入力: 分析結果のタプル t_{out}
出力: t_{out} の拡張来歴

```
1: キュー q を {t_out} で初期化し，visited と SL，RL を空集合で初期化.
2: while q に要素があるとき do
3:     t = q.dequeue()
4:     if t.RL ≠ null かつ t.RL.size > 0 then
5:         RL = RL ∪ {t.RL}
6:     if t.Type が Source then
7:         SL = SL ∪ {t}
8:     else if t.Type が Map then
9:         enqueueIfNotVisited(t.U1, q, visited)
10:    else if t.Type が Join then
11:        enqueueIfNotVisited(t.U1, q, visited)
12:        enqueueIfNotVisited(t.U2, q, visited)
13:    else if t.Type が Aggregate または REMOTE then
14:        t_tmp = t.U2
15:        while True do
16:            enqueueIfNotVisited(t_tmp, q, visited)
17:            if t_tmp == t.U1 then
18:                break
19:            t_tmp = t_tmp.Next
20: return [SL, RL]
```

5　実　　験

本節では Flink 上に実装したプロトタイプシステムを用いたクラスタ環境における性能評価の結果を述べる．

5.1　データセット

実験ではストリームデータとして，Amazon レビューデータ [16] に基づいた 2 種類のデータを使用した．Amazon レビューデータは Amazon の商品 ID とそのレビューコメントなどが記録されているデータセットである．事前の調査で Amazon レビューデータには商品の出現回数や入力タプルサイズの点で，データに偏りがあることがわかっていた．そこでデータの偏りによる実験結果への影響を排除するために，元の Amazon レビューデータを修正した人工データセットと元の Amazon レビューデータをそのまま使用するリアルデータセットを用意し

た．人工データセットでは商品 ID が一様分布に従い，全ての入力タプルのサイズが同じになるようにした．

5.2 複合的ストリーム処理フローと AI/ML 処理の実装

実験の複合的ストリーム処理フローは，レビューコメントを受け取るとそのコメントが Positive か Negative かを判定する感情判定 AI/ML 処理を活用する．実際にはリアルタイム処理フローとウインドウ処理フローの 2 つの処理フローを用いて実験を行った．どちらの処理フローも，Apache Kafka から入力タプルを受け取り，Kafka に分析結果を送信する．

リアルタイム処理フローは以下に示すオペレータで構成される：Source-Map-Filter-Map-Sink．入力タプルを受け取ると，そのコメントが Positive か Negative かを判定する分析を行う．はじめの Map オペレータは，文字列型入力タプルを Java オブジェクトのタプルに変換する．Filter オペレータは直前の Map で変換された際に欠損値を持つタプルを削除するが，実際にはここで削除されるタプルはほとんどなかった．2 つ目の Map オペレータは感情判定処理を実行し，タプルに Positive または Negative を割り当てる．

一方，ウインドウ処理フローは以下のオペレータで構成される：Source-Map-Filter-Map-Aggregate-Sink．リアルタイム処理フローの分析の内容に加えて，商品ごとに Positive/Negative なコメントの数をカウントする分析を行う．そのため Aggregate オペレータでは，商品 ID ごとにウインドウ内の Positive/Negative なコメントをカウントする処理を行う．

感情判定 AI/ML 処理に関して，今回の実験では 2 つの実装を用意した．まず 1 つ目は人工データを用いた実験で使用する，ダミーの感情判定モデルを用いた実装である．ダミーの感情判定モデルは，処理時間の変動を抑えることで安定した実験結果を得ることを目的にしており，入力データを受け取ると事前に定義した固定時間処理をアイドルしたあと，Positive の分析結果と固定長文字列を判断根拠として返す．2 つ目はリアルデータを用いた実験で使用する，IBM が提供している実際のモデル [17] を用いた実装である．

ここまでに説明したデータセットと処理フロー，感情判定モデルの実装を組み合わせて以下の実験を行った：人工データを用いた 2 つの処理フローの実験（ダミー感情判定モデル使用）と，リアルデータを用いた 2 つの処理フローの実験（IBM の感情判定モデル使用）．

5.3 比較手法と比較指標

各実験では 3 つの比較手法を評価した：1) Lineage を導出しないケース（Baseline, Base）, 2) GeneaLog を用いて Lineage のみ導出するケース（Lineage, Lin）, 3) 拡張来歴を導出するケース（SAL）．SAL が本稿で説明した手法である．

比較指標としてはレイテンシとスループットに加え，拡張来歴導出時の貢献グラフを辿るためにかかる時間（Traversal 時間）の 3 つを用いた．レイテンシと Traversal 時間は実験を 1 回実施した際に出力された分析結果の平均値を使用し，スループットは 5 回実験を行った時の平均値を使用している．なお実験はクラスタ環境で実行した．クラスタは 2 ノード Apache Flink クラスタ（1 ジョブマネージャー，1 タスクマネージャー）と 3 ノード Apache Kafka クラスタ（1 Zookeeper，2 ブローカノード）で構成され，各ノードは 128 GB メモリと 10 CPU コアを持つ．

5.4 人工データを用いた評価実験

人工データを用いた実験では入力データや処理フローの特性が性能に与える影響を検証するために，リアルタイム処理フローとウインドウ処理フローのそれぞれに対して，データの特性や処理パラメータの組み合わせを複数試すことで多角的な評価を行った．具体的には，以下の 3 つの特性を変化させた際のレイテンシとスループット，Traversal 時間を測定した．

- ダミー感情判定モデルの処理コスト：実行時のアイドル時間を変更．**Light**：アイドル無し，**Heavy**：200 ms.
- 入力タプル長：レビューコメントの長さを変更．**Short**：1 byte, **Long**：35,094 bytes.
- ウインドウサイズ：**Small**：1 秒，**Large**：100 秒.

3 つの特性の組み合わせを x/x/x のように表記し，それぞれが処理コスト，入力タプル長，ウインドウサイズに対応している．例えば H/L/S は，処理コスト Heavy，入力タプル長 Long，ウインドウサイズ Small を表す．またリアルタイム処理フローにおけるウインドウサイズは「-」で表記する．

図 6 は人工データを用いた評価実験のレイテンシと Traversal 時間を，図 7 はスループットを表している．図中の ** と * は，SAL を Baseline または Lineage と比較したときにそれぞれ有意差 1 % もしくは 5 % で統計的な差があることを示している．図 6 の Traversal 時間のバーにおける ** もしくは * の上には，SAL の Traversal 時間が Lineage の Traversal 時間と比較して何倍になったかを示している．なお Baseline 手法では Lineage を求めるための Traversal 時間が発生しないため，Traversal 時間は示していない．差の有無の確認には Welch の t 検定 [18] を用い，必要に応じてボンフェローニ補正を適用した結果に基づいている．

まずリアルタイム処理フローを用いた結果について説明する．レイテンシとスループットに関しては感情判定モデルの処理コストが大きいとき，拡張来歴の導出によって発生するオーバヘッドは非常に小さいことが示された．図 6(a)，7(a) の処理コストが大きい場合を見ると，レイテンシは H/S/- ，スループットは H/L/- のとき統計的な差があるもののその差は非常に小さい．実際 H/S/- の SAL のレイテンシは Baseline と比較して 1.0002 倍，Lineage と比較して 1.0001 倍であり，H/L/- の SAL のスループットは Baseline の 0.98 倍，Lineage の 0.99 倍である．またモデルの処理コストが大きいとき，レイテンシ全体に対する Traversal 時間の割合は非常に小さいことも示された．これらの理由はダミーの感情判定モデルの処理コストが大きい時，モデルの処理時間が全体の処理時間のほとんどを占め，拡張来歴を求めるための追加処理にかかるオーバヘッドが相対的に小さくなったからと考えられる．一方で処理コストが小さいときは，レイテンシやスループットの性能が大きく低下

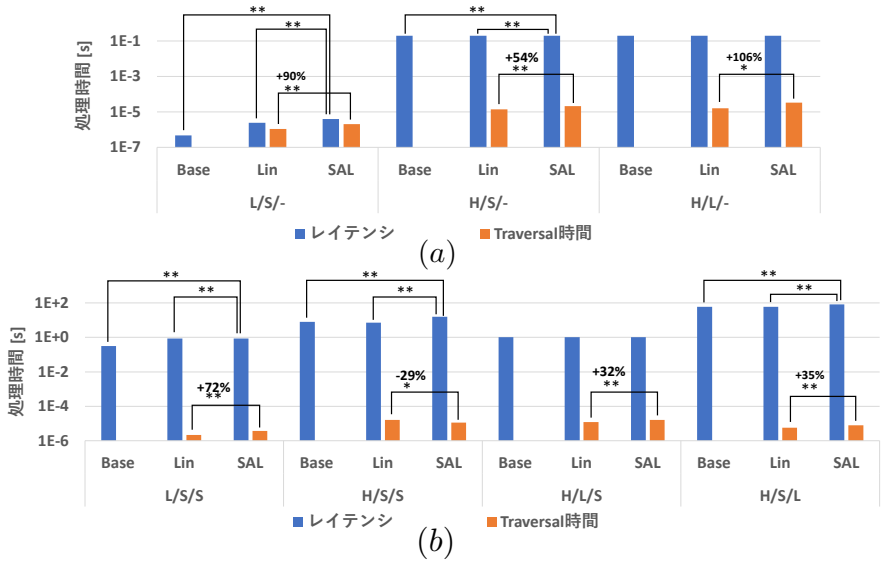

図 6　(a) リアルタイム処理 (b) ウインドウ処理 におけるレイテンシ／ Traversal 時間

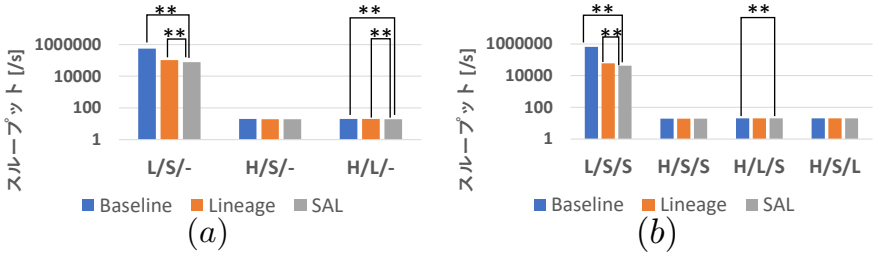

図 7　(a) リアルタイム処理 (b) ウインドウ処理 におけるスループット

した．SAL のレイテンシは Baseline の 8.4 倍，Lineage の 1.6 倍となり，スループットは Baseline の 0.14 倍，Lineage の 0.72 倍だった．この時の SAL の Traversal 時間は SAL のレイテンシのうち約 52%を占めており，このことは拡張来歴を求めるための追加処理が性能に大きく影響を与えることを示していると考えられる．加えて Traversal 時間に関しては，図 6(a) で示すように拡張来歴の追加情報を導出する SAL の方が Lineage と比べて，Traversal 時間が長くなることを示した．

次にウインドウ処理フローを用いた際の結果を説明する．レイテンシに関しては処理コストが大きい場合でも入力タプル長が長い場合，レイテンシへの影響は非常に小さいが，入力タプル長が短い場合は性能差が現れる結果となった．またスループットはリアルタイム処理フローの場合と同様，処理コストが大きい場合は拡張来歴を導出したとしても性能の低下がほとんど見られなかった（図 7(b) ）．H/L/S のケースで統計的な差は見られたが，この時の SAL のスループットは Baseline の 0.996 倍でその差は非常に小さい．一方，感情判定の処理コストが小さい場合は，SAL の L/S/S のレイテンシが Baseline の 2.8 倍，Lineage の 1.03 倍となった．Traversal 時間に関しては，処理コストの大小に関わらずレイテンシ全体に占める Traversal 時間の割合は非常に小さく，多くの場合で SAL の方が Lineage と比べて Traversal 時間が長くなることが示された．

人工データを用いた実験では，処理コストが小さい場合には

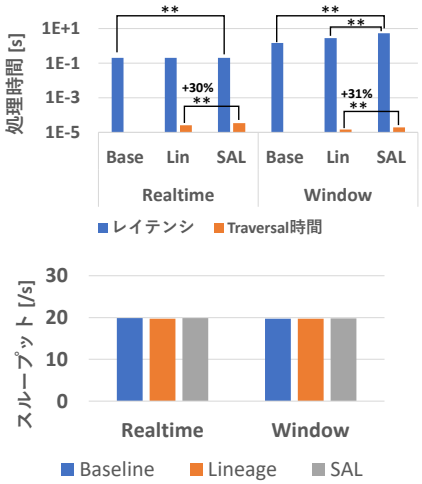

図 8　リアルデータを用いた性能評価

拡張来歴を求める際のオーバヘッドが大きくなる結果になった．しかし本研究で想定しているような AI/ML 処理の処理コストは一般に大きいため，実際に本研究が使用される状況においては拡張来歴を求める際のコストはそれほど問題にならないと考えられる．

5.5　リアルデータを用いた評価実験

リアルデータを用いた評価実験では，IBM が提供するネット

ワークベースの感情判定モデルと，元の Amazon レビューデータを使用した．感情判定モデルの処理コストは前節の処理コスト Heavy に相当し，元の Amazon レビューデータのタプルサイズには Short〜Long の範囲で様々なサイズのタプルが存在している．またウインドウサイズは 1 秒ウインドウを使用した．そのため本実験は，リアルタイム処理フローでは前節の H/S/- と H/L/-，ウインドウ処理フローでは H/S/S と H/L/S に対応している．

図 8 が実験の結果である．本実験は前節の処理コスト Heavy の場合に対応しているため，リアルタイム処理フローではレイテンシ，スループットの両方で非常に小さいコストで拡張来歴が導出でき，レイテンシに占める Traversal 時間は非常に小さいという結果になった．レイテンシで統計的な差は見られたものの，その時の SAL のレイテンシは Baseline の 0.996 倍 で，ほとんど影響ないことが示された．

またウインドウ処理フローにおけるスループットや Traversal 時間の傾向はリアルタイムの場合と同様となった．一方 SAL のレイテンシは SAL は Baseline の 3.6 倍，Lineage の 1.9 倍 になるという結果となった．これは元の Amazon レビューデータではサイズ小のタプルの割合が大きいため，前節の処理コスト大，タプルサイズ小の時と同様の傾向が現れたと考えられる．

6 結　　論

本稿では AI/ML 処理を伴う複合的ストリーム処理において，従来の Lineage に加えて AI/ML 処理の判断根拠を提示する，拡張来歴導出手法の説明を行った．さらにシステムを最新手法である GeneaLog に基づいて Apache Flink 上に実装し，偏りのない人工データとリアルデータを用いた評価実験の結果を示した．実験結果から，説明した手法は AI/ML 処理のコストが大きい場合には小さなコストで拡張来歴が導出可能である傾向を示した．今回説明した枠組みによって，AI/ML 処理を伴う複合的ストリーム処理の分析結果として得られるデータの信頼性を一層高められることが期待される．

今後の課題としては，より多様なデータセットや処理フローにおける評価実験の実施や，現実のストリーム処理アプリケーションへの適用，Apache Flink 以外のストリーム処理エンジンでの実装が挙げられる．さらに，導出された拡張来歴の情報をデータベース等を用いて永続化かつクエリ可能にするシステムの提案も重要なトピックである．

謝　　辞

本研究の一部は，JSPS 科研費 (JP19H04114，JP20K19806，JP22H03694)，JST CREST (JP-MJCR22M2)，NEDO (JPNP20006)，AMED (JP21zf0127005)，SKY 株式会社共同研究，JST 次世代研究者挑戦的研究プログラム (JPMJSP2124) による．

文　　献

[1] Melanie Herschel, Ralf Diestelkämper, and Houssem Ben Lahmar. A survey on provenance: What for? what form? what from? *The VLDB Journal*, 26(6):881–906, 2017.

[2] Susan B. Davidson and Juliana Freire. Provenance and scientific workflows: Challenges and opportunities. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, page 1345–1350, 2008.

[3] Sherif Akoush, Ripduman Sohan, and Andy Hopper. Hadoopprov: Towards provenance as a first class citizen in mapreduce. In *5th USENIX Workshop on the Theory and Practice of Provenance*, 2013.

[4] Y. Cui and J. Widom. Practical lineage tracing in data warehouses. In *Proceedings of 16th International Conference on Data Engineering*, pages 367–378, 2000.

[5] Yingwei Cui, Jennifer Widom, and Janet L. Wiener. Tracing the lineage of view data in a warehousing environment. *ACM Trans. Database Syst.*, 25(2):179–227, 2000.

[6] Boris Glavic, Kyumars Sheykh Esmaili, Peter Michael Fischer, and Nesime Tatbul. Ariadne: Managing fine-grained provenance on data streams. In *Proceedings of the 7th ACM International Conference on Distributed Event-Based Systems*, page 39–50, 2013.

[7] Dimitris Palyvos-Giannas, Vincenzo Gulisano, and Marina Papatriantafilou. Genealog: Fine-grained data streaming provenance in cyber-physical systems. *Parallel Computing*, 89, 2019.

[8] Masaya Yamada, Hiroyuki Kitagawa, Toshiyuki Amagasa, and Akiyoshi Matono. Augmented lineage: Traceability of data analysis including complex udfs. In *Database and Expert Systems Applications*, pages 65–77, 2021.

[9] Masaya Yamada, Hiroyuki Kitagawa, Toshiyuki Amagasa, and Akiyoshi Matono. Augmented lineage: traceability of data analysis including complex udf processing. *The VLDB Journal*, 2022.

[10] Masaya Yamada et al. Streaming augmented lineage: Traceability of complex stream data analysis. In *Information Integration and Web Intelligence*, pages 224–236, 2022.

[11] Todd J. Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *Proceedings of the Twenty-Sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, page 31–40, 2007.

[12] J. Nathan Foster, Todd J. Green, and Val Tannen. Annotated xml: Queries and provenance. In *Proceedings of the Twenty-Seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, page 271–280, 2008.

[13] Matteo Interlandi et al. Titian: Data provenance support in spark. In *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, volume 9, page 216–227, 2015.

[14] Nithya N. Vijayakumar and Beth Plale. Towards low overhead provenance tracking in near real-time stream filtering. In *Provenance and Annotation of Data*, pages 46–54, 2006.

[15] Min Wang et al. A time-and-value centric provenance model and architecture for medical event streams. In *Proceedings of the 1st ACM SIGMOBILE International Workshop on Systems and Networking Support for Healthcare and Assisted Living Environments*, page 95–100, 2007.

[16] Jianmo Ni, Jiacheng Li, and Julian McAuley. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 188–197, 2019.

[17] MLX. Text Sentiment Classifier. `https://www.ml-exchange.org/models/max-text-sentiment-classifier`.

[18] Bernard L Welch. The generalization of ‘student’s’ problem when several different population varlances are involved. *Biometrika*, 34(1-2):28–35, 1947.