Vol.2015-SLDM-170 No.14
Vol.2015-EMB-36 No.14
2015/3/6

データストリーム管理システムのための動作検証環境

中井 将貴[1]　松原　豊[1]　高田 広章[1]　山口 晃広[1]

概要：名古屋大学大学院情報科学研究科附属組込みシステム研究センター（NCES）では，車載システムのデータ管理を目的とした組込みデータストリーム管理システム（eDSMS）の開発及び研究を行っている．現在の eDSMS はブラックボックスになっており，オペレータやストリーム，データ処理の実行時間や動作順序を把握する手段が提供されていない．その結果，eDSMS のクエリ開発者が，eDSMS の動作が要求仕様と設計から逸脱していないことを容易に確認することができない．本研究では，eDSMS のクエリ開発者向けに，eDSMS の動作を容易に検証することができる環境を開発した．適用事例として，自動車の衝突警告とナビゲーション，車車間通信にデータ出力を模擬するクエリを開発し，要求仕様と設計からの逸脱の解析と，逸脱の原因特定を行った．解析の結果，衝突警告とナビゲーション間で共有しているストリームのデータ量が設計時に規定した最大データ量よりも高くなっている逸脱を発見した．提案する検証フローに沿った原因特定を行い，設計を見直した結果，設計からの逸脱がなくなり，正常にクエリが動作したことを確認した．以上より，提案した動作検証環境の有効性を確認することができた．

キーワード：データストリーム管理システム，解析，可視化，動作検証環境

1. はじめに

1.1 研究の背景・目的

近年，株価やセンサデータといったとめどなく流れるデータを処理する手法としてデータストリーム管理システム（DSMS）の研究が行われている．一般的なデータベース管理システム（DBMS）は，データを一度蓄積する構造を持ち，それに対してユーザが命令を発行する．DSMS の場合は，流れるデータをオペレータと呼ばれる演算子とデータの流れを決定するストリームの組み合わせにより逐次的に処理し出力する処理モデルをとる．

名古屋大学大学院情報科学研究科附属組込みシステム研究センター（NCES）では，DSMS を車載システムのデータ管理に使用することを目的として，組込みデータストリーム管理システム（eDSMS）の開発及び研究を行っている．NCES では eDSMS を実行するためのソースコード自動生成ツールや実行環境を総称してクラウディア（Cloudia）と呼んでいる．現在の eDSMS はブラックボックスになっており，オペレータやストリーム，データ処理の実行時間や動作順序を把握する手段が提供されていない．その結果，eDSMS の開発者が，eDSMS の動作が要求仕様と設計から逸脱していないことを容易に確認することが困難である．本研究では，要求仕様と設計からの逸脱を発見することと逸脱原因を特定することを目的とした動作検証環境を構築する．

1.2 関連研究

ログ解析という観点では，Simache らによるイベントログに基づく信頼性解析 [1] や森本らによるログ検索システムに関する研究 [2] がある．イベントログに基づく信頼性解析では，WindowsOS のイベントログを利用してボトルネックの特定，ディペンダビリティの定量化を行っている．ログ検索システムでは，ログ検索を正規表現を用いてることで簡単に検索ができる．本研究では，ユーザ自身がログを検索するのではなく，要求仕様書と設計書からログ解析を自動化し，逸脱した箇所を取り出すようにした．データストリーム処理の可視化という観点では，Mansmann らによる動的にストリームを可視化する StreamSqueeze ツール [3] がある．StreamSqueeze ツールでは，ニュースやシステムログを監視するために用いる．古いデータほどサイズが小さく表示され，新しいデータほどサイズが大きく表示される．そして，関係性があるデータ同士を同色で表示させる．このツールはあくまで，流れてきたデータをユーザにとってわかりやすく分割するものであり，デバッグ工程で使用するものではない．また，組込みシステムにおけるセンサデータのような非常に短い時間に大量のデータを処理しなければいけない場合，StreamSqueeze ツールではリアルタイムに処理ができない．

2. 組込みデータストリーム管理システム eDSMS

2.1 特徴

eDSMS とは，embedded Data Stream Mangement System の略であり DSMS を組込みシステム向けに対応させたものである．カメラやレーダといったセンサデータや車車間通信などのデータを逐次的に高速処理するためにストリーム処理を活用している．DSMS と比較した際の eDSMS の大きな特徴を以下に示す．

[1] 名古屋大学大学院情報科学研究科
Graduate School of Information Science, Nagoya University

- 動的ではなく静的にランタイムの生成を行う
- リアルタイム性を持つ

ここで述べるランタイムとは，実行環境のことである．従来の DSMS とは異なり動的ではなく静的にランタイムの生成を行う．静的にランタイムを生成することで，組込みシステムにおける限られたリソースを活用することができる．また，eDSMS のスケジューラはリアルタイム性を持つ．DSMS を組込みシステムに適用する上で，処理の時間制約を保障することは非常に重要である．

2.2 eDSMS 開発の流れ

本研究では，Cloudia を対象とする．XML 形式から C++ のソースコードに自動変換を行うツールを Cloudia ツールと呼ぶ．XML 形式は，自分でソースコードを記述するよりもわかりやすく，簡単に処理内容を記述することが可能である．また，自動変換されたソースコードをコンパイルするための環境を Cloudia ランタイムと呼ぶ．Cloudia ツールと Cloudia ランタイムの 2 つを総称して Cloudia と呼ぶ．下記に Cloudia を用いた eDSMS の開発の流れを示す．

(1) クエリの内容を XML 形式でクエリ設定ファイルとして記述
(2) クエリ設定ファイルを Cloudia ツールを用いて C++ 形式のソースコードに変換
(3) C++形式のソースコードを Cloudia ランタイム上でコンパイルし，実行ファイルを作成する

クエリ設定ファイルでは，オペレータの処理内容とストリームの流れ，オペレータの集合であるタスクの構成，タスクのデッドラインを記述する．

2.3 ストリーム

ストリームは，データ 1 件分に該当するタプルの流れる方向を決定する役割を持つ．ストリームごとに，キューと呼ばれるタプルを貯める構造を持つ．そして，ウィンドウと呼ばれる無限長のデータストリームを有限の長さに区切るための仕組みを持つ．

2.4 オペレータ

オペレータとは，ストリームを通して流れてきたタプルの処理を担当する演算子である．eDSMS におけるオペレータの動作内容は，Borealis[4] に準拠する．

2.5 タスク

eDSMS では，リアルタイム性を重視する観点からタスクの概念を持つ．eDSMS でのタスクとは，ユーザが定義するオペレータの集合である．ユーザは出力ストリームごとに相対デッドラインを定義する必要がある．タスク構成は，基本的にはクエリの入力から出力までを 1 本のパスとするのが効率的である．しかし，オペレータの出力が複数に分岐しストリームを共有する場合やデータの待ち合わせがある複数入力のオペレータ，データに待ち合わせがない複数入力のオペレータがある場合も存在する．そのような場合 eDSMS では，Xin らの発見した効率的なタスク構成を採用している [5].

2.6 コールバック

eDSMS のクエリで処理され，値が出力されることをコールバックと呼ぶ．出力された値をコールバック値と呼ぶ．また，各出力ごとにユーザは，コールバック名を付ける必要がある．クエリから出力されたコールバック値を使用して，衝突警告や前方車追従といったアプリケーションを開発することができる．

2.7 スケジューラ

eDSMS では，スケジューリングアルゴリズムとして Earliest Deadline First（EDF）アルゴリズムを採用している．EDF アルゴリズムは，デッドラインが最も近いタスクから実行する動的スケジューリング規則である [6]．ユーザが各出力ストリームごとに許容最大時間を指定する．センサからそのデータが発生した時間を示すタイムスタンプに許容最大時間を足し合わせたものをデッドラインとする．

また，待ち合わせがある複数入力のオペレータがある場合，タスクのデッドラインは，EDF*アルゴリズム [7] でデッドラインの計算がなされる．EDF*アルゴリズムでは，後続のタスクのデッドラインと計算時間から先行タスクのデッドラインを計算する手法である．後続タスクのデッドラインから，予測した後続タスクの計算時間を引くことでデッドラインを求める．

eDSMS ではタスクリストと呼ばれるキュー構造を持つ．タスクリストにはデッドラインが近い順にタスクが実行可能状態で並んでおり，FIFO 形式でタスクを順次実行する．タスクにデータが入力された際に，状況に応じて EDF か EDF*のどちらかの計算方法でデッドラインを求める．そして，そのタスクがタスクリストに入力され実行可能状態となる．これをタスク生成と呼ぶ．タスクリストにタスクが入力された際に，以前入力されたタスクとデッドラインを比較する．以前入力されたタスクの方がデッドラインが近ければタスクリスト内の順序は変わらない．しかし，後から入力されたタスクの方がデッドラインが近ければタスクリスト内でデッドラインが近い順にタスクの実行順序を入れ替える．これを，プリエンプションと呼ぶ．このスケジュール手法を採用することで，一般的な DSMS で使用されるリアルタイム性を持たない FIFO 形式と比較して，デッドラインミスが 0 になることが山口らの論文 [8] により証明されている．

3. データストリーム管理システムのための動作検証環境

3.1 概要

動作検証環境の全体図を図 1 に示す．本研究では，eDSMS のための動作検証環境を構築した．ユーザは，要求仕様書と設計書に基づいた逸脱ルールを記述することで解析結果を得ることができる．また，実行ログから可視化結果を得ることができる．想定する動作環境の使用方法を図 2 に示す．ユーザは，要求仕様書と設計書に基づいてクエリ設定ファイルと逸脱ルールファイルを記述する．実行ファイルから出力された実行ログに対して，作成した逸脱ルールファイルを用いて解析ツールを実行する．もし，逸脱がなければ実行ファイルは要求仕様と設計通りに作成されたものだと言える．しかし，逸脱があった場合は可視化ツールを使用して原因の特定を行う．

3.2 ログ出力の実装

eDSMS には，ログ出力の機能が存在しない．動作検証を実現するために，ログ出力の機能を追加する必要がある．

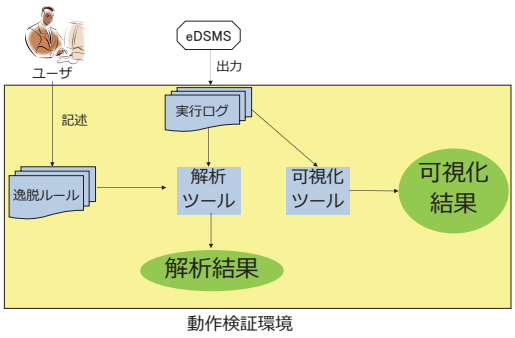

図 1　動作検証環境の全体図

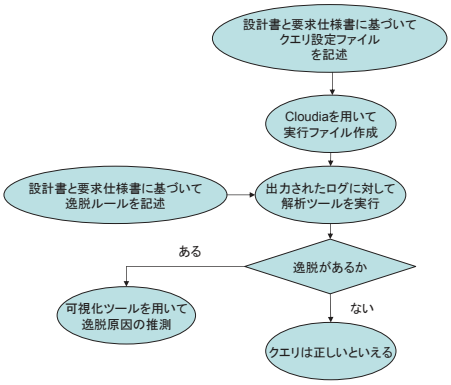

図 2　動作検証環境使用方法

ここでは，ログとして出力すべき動作内容について述べる．ログ出力時に一緒に出力される時間単位は，詳細に処理を追っていく必要があるという点から us 単位に設定した．なお，全てのログ出力コードは#ifdef により管理されており，ユーザが取得したいログのみを取得することが可能である．また，DSMS にはなく eDSMS 特有の概念は，デッドラインとタスクである．他の概念は一般的な DSMS にも利用することができる．以下では，ログファイルの内容を示す．

- callback.log

クエリの処理結果と出力時間が記述されるログファイルである．

- log_analysis.log

タスクリスト内のタスク数，タスクに対する入力データ，タスク生成，タスク開始，タスク終了，オペレータの開始，オペレータの終了が記述されるログである．

- que_data_size.log

クエリ上の各ストリームキューにデータがどれだけ入力されたか，取り出されたかを示すログである．

3.3　ログ可視化

出力したトレースログを用いて，eDSMS 動作の可視化とストリームキューのグラフ化を行った．

3.3.1　TLV を利用した eDSMS 動作の可視化

TLV（TracelogVIsualizer）とは，名古屋大学で開発された，各種 RTOS やシミュレータから出力されたトレースログを可視化し動作解析を容易にするツールである．本研究では，TLV を応用し eDSMS 動作の可視化を行った．TLV では可視化をするために，TLV 用に規定されたログ形式に変換する必要がある．eDSMS から出力されたログを，TLV 用に規定されたログに自動変換するツールを開発した．また，可視化に必要な情報を記述した可視化情報ファイルが必要である．可視化情報ファイルはクエリ作成時に使用したクエリ設定ファイルから自動変換できるようにした．本研究では，キュー操作の可視化とオペレータ動作の可視化をできるようにした．

生成された TLV 形式のログと可視化情報ファイルを利用して可視化を行うことができる．キュー操作の可視化には，キューの情報が記述された que_data_size.log を使用する．オペレータ動作の可視化には，オペレータの情報が記述された log_analysis.log を使用する．

3.3.2　ストリームキューのグラフ化

ストリームキューをグラフ化することで，キューに貯まっているデータ量を時系列順に直感的に理解することができる．ストリームキューのグラフ化には，que_data_size.log を使用する．ユーザは本研究で開発した該当箇所のログを取り出すツールを使用して，グラフ化したいストリームキューのデータ量が記述されたログを取り出す．もし全ての処理が見たい場合であれば，全ての処理を該当箇所として選択する必要がある．CSV ファイルとして，時系列順にストリーム名とストリームキューのデータ量が出力される．生成された CSV ファイルをグラフ化することで，ストリームキューのデータ量をグラフ化することができる．

3.4　ログ解析

実行ログには，que_data.log と log_analysis.log と callback.log を使用する．ユーザは要求仕様書と設計書に基づいた逸脱ルールを CSV 形式として記述する．ユーザは逸脱ルールとして，deadline_rule.csv と que_size_rule.csv と tasklist_rule.csv と value_rule.csv といった 4 つの CSV ファイルを記述する必要がある．deadline_rule.csv には出力ごとのデッドラインを記述する．que_size_rule.csv には，キューに入るデータ量の最小値と最大値を記述する．tasklist_rule.csv には，タスクリスト内のタスク数の最小値と最大値を記述する．value_rule.csv には出力値の最小と最大を記述する．作成した逸脱ルールに対して自動生成ツールを実行すると，逸脱ルールに基づいたログ解析実行ファイルが生成される．実行ログに対して生成されたログ解析実行ファイルを実行すると，逸脱ルールに対してどれだけ逸脱した動作が起きたか解析される．

4.　適用事例と評価

4.1　概要

本研究では，衝突警告，車車間通信，ナビゲーションのアプリケーションにデータ出力を模擬する安全運転支援クエリを開発した．この開発において，提案する動作検証環境を適用した．クエリから出力されたログを入力として解析ツールを実行し，解析ツールが正確に動作していることを確かめる．そして，検証フローを構築し，それに沿った検証を行えば逸脱原因を特定し，改善できることを確かめる．

4.2　動作環境

クエリの動作環境として以下を使用する．

- PC：Panasonic Let's note
- OS：Windows7 64bit
- 仮想マシン：VMware6.0.1

図 3　ユースケースにおける車両の初期位置

- 仮想マシン上の OS：Fedora10
- プロセッサ：Intel(R) Core(TM)　i5-3320M
- メモリ：8.00GB

4.3　ユースケース

ユースケースにおける車両の初期位置を，図 3 に示す．自車は初期位置から時速 60km で直線を走行していると想定し，他車も直線を時速 60km で走行する．自車に搭載されているセンサは，レーダセンサと自車速度センサ，GPS センサ，車車間通信センサ（V2V センサ）とする．どの車両から受け取ったデータか解釈するために，車両ごとに ID が割り振られている．自車が直線を走行している間に，受け取ったセンサデータをデータストリームに対する入力データとして扱う．車両が集中する交差点では，自車に入力されるセンサデータ数は増加する．なお，ユースケースと入力データは，ネットワークシミュレータである Scenargie[9] で作成した．

4.4　安全運転支援クエリ

図 4 に本研究に作成した安全運転支援クエリを示す．適用事例として開発した，衝突警告，車車間通信（V2V），ナビゲーションのアプリケーションにデータ出力を模擬する安全運転支援クエリに作成したデータを入力する．その際に，逸脱ルールファイルを作成する．デッドラインは，callback1 は 30[ms]，callback3 は 300[ms] と設定した．callback2 に該当するナビゲーションはリアルタイム性を必要としない処理であるため，十分時間をとった 3,000[ms] と設定した．また各ストリームのキューのデータ量は全て最小値は 0，最大値 301 と指定した．最大値を減らすと，車両が集中する交差点に自車が進入したときデータ量が急激に増加するため，データを処理しきれなくなる．また最大値を増やすと，入力するデータ量が増えて，データを入力する処理に時間がかかるため増えるためデッドラインミスが発生するおそれがある．図 4 の Join2 では衝突余裕時間（Time To Colusion) の値を計算している．本研究におけるユースケースの場合，車両は時速 60[km] で走行しているため，衝突余裕時間の値が 2.8 秒未満になった際に衝突警告をする必要がある [10]．つまり，安全運転支援クエリにおける衝突警告を正確に実現するためには，0 秒以上 2.8 秒未満の衝突余裕時間の値のみを callback3 より出力される必要がある．よって，callback3 の出力値のルールとして最小値 0，最大値 2.8 と設定した．他の出力は値に明確な逸脱ルールがないため，逸脱ルールを記述しなかった．

安全運転支援クエリから出力されたログを入力として，

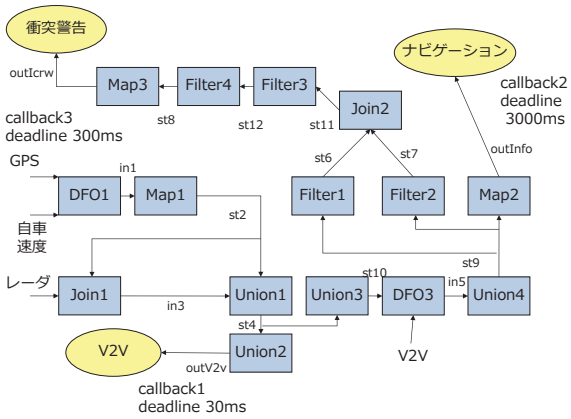

図 4　安全運転支援クエリ

```
[43908844] ENQUEUE st9 301
que_size_analysis
st9
min_count= 0
max_count= 1
succes_count= 2642
```

図 5　解析結果

逸脱ルールに基づいて解析ツールを実行した．その結果を図 5 に示す．結果から，ストリーム st9 におけるキューが最大量を超える逸脱を起こしていることがわかる．

4.5　逸脱原因の特定

キューのデータ量における逸脱が起きた時の原因特定までの検証フローを図 6 として提案する．逸脱ストリームとは，安全運転支援クエリであれば st9 のことである．まず，st9 のデータ量を，逸脱発生まで時系列に沿ってグラフ化する．次に，グラフから逸脱発生までの軌跡をたどる．例えば，急激にデータが増加し始めた箇所であったり，データが増加している期間に急にデータが減り始めた箇所といった特徴的な動きをする期間に印をつける．そして，周辺オペレータの可視化を行う．周辺オペレータは，ストリームに接するオペレータと定義した．安全運転支援クエリであれば，Union4 と Map2 と Filter1 と Filter2 が周辺オペレータの定義に該当する．周辺オペレータを可視化し，可視化結果を先に印をつけた逸脱ストリームのグラフと合成する．例えば，周辺オペレータで非常に処理時間がかかっているオペレータ A があり，処理している期間がデータ量増加期間と重なった場合，オペレータ A に原因があると推測できる．もし，周辺オペレータと逸脱ストリームのグラフを比較し不具合が見られなかった場合，周辺オペレータからさらにパスが 1 つ先のオペレータを可視化し，検証フローを繰り返す．周辺オペレータからさらに 1 つ先のパスとは，安全運転支援クエリであれば，Join2 と DFO3 である．このように，逸脱ストリームのキューのデータ量とオペレータの動きを比較していくことで，どの処理に原因があるか調べることができる．逸脱ストリームより出力側の処理が原因である場合，処理の優先度が他より低いことなどが原因として考えられる．また，逸脱ストリームに対してセンサ側の処理が原因である場合，センサの入力頻度が高いことなどが原因として考えられる．

Vol.2015-SLDM-170 No.14
Vol.2015-EMB-36 No.14
2015/3/6

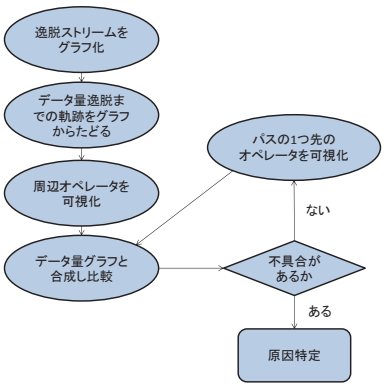

図 6　ストリームキューデータ量に逸脱が見つかった際の検証フロー

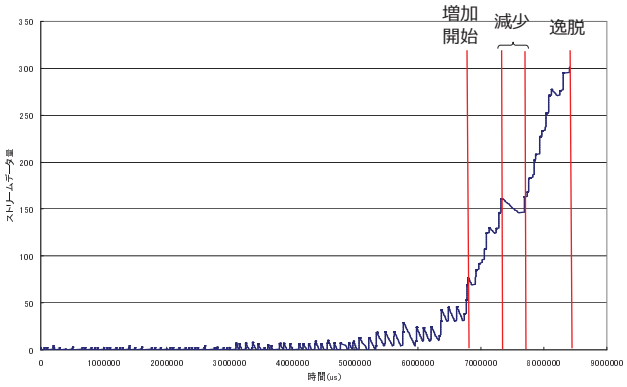

図 7　安全運転支援クエリ：st9 のデータストリームキューのグラフ

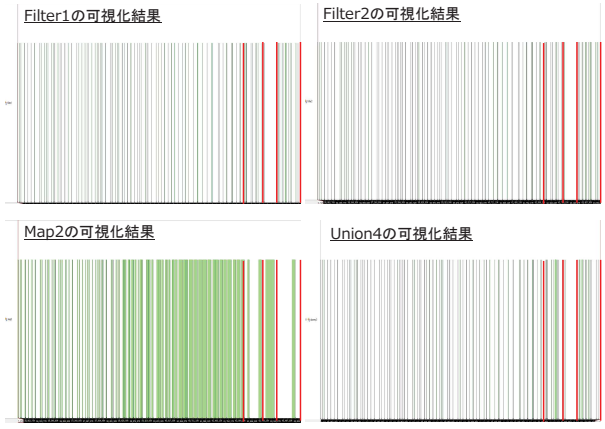

図 8　周辺オペレータの可視化結果

4.5.1　逸脱ストリームのグラフ化

安全運転支援クエリにおいて逸脱が発生した st9 をグラフ化する．グラフ化し，逸脱までの軌跡をたどる．およそ 6,800,000[us] からデータ量の増加が始まっている．およそ 7,400,000[us] からおよそ 7,800,000[us] までデータ量は減少している．およそ 7,800,000[us] からおよそ 8,500,000[us] までデータ量は増加し続けておよそ 8,500,000[us] でデータ量が 301 を超える逸脱が発生している．

4.5.2　周辺オペレータの可視化

オペレータ Union4，Filter1，Filter2，Map2 を可視化ツールを用いて可視化する．それぞれの可視化結果に，逸脱ストリームのグラフ化の際に検討したデータ量の増加とデータ量の減少の軌跡を印として付加する．

表 1　データ量の増減とオペレータ動作の対応表

	データ量増加	データ量減少
Map2	動作していない	動作している
Filter1	動作している	動作していない
Filter2	動作している	動作していない
Union4	動作している	動作していない

4.5.3　原因の特定

安全運転支援クエリにおける st9 のデータ量の増減に対応したオペレータの動作を表にまとめたものを表 1 に示す．st9 のデータ量が増加している間は，Map2 は動作しておらず，他の周辺オペレータは動作している．また，st9 のデータ量が減少している間は，Map2 は動作しており，他の周辺オペレータは動作していない．ストリームが共有される場合，共有するストリームの向かう先にある全てのオペレータにデータが入力され計算がされなければストリームキューのデータ量は減らない．よって，Map2 が動作していないことでストリームのデータ量が減少していないことが可視化結果からわかる．Map2 はリアルタイム性を持たないため，余裕を持ったデッドライン設定を行った callback2 に結果を出力するオペレータである．それに対して，Filter1 と Filter2 は，衝突警告という厳格なデッドラインが求められるため，短いデッドライン設定を行った callback1 に結果を出力するオペレータである．そこで，デッドラインに大きな差がある処理間でストリームを共有すると，デッドラインの大きい処理が後回しにされ，共有したストリームキューのデータ量があふれてしまうという原因を特定した．

4.6　原因特定結果を利用したクエリの再設計

デッドラインに差がある処理間でストリームを共有しているため，Map2 の処理が後回しにされ st9 からデータが十分に流れていない．st9 と Map2 間に，Filter1 と Filter2 と同じタイミングで処理され十分なストリームキューサイズを持ったオペレータを実装した．実装後のクエリを設計見直し後安全運転支援クエリと呼ぶ．設計見直し後安全運転支援クエリの全体図を図 9 に示す．新たに追加したオペレータは Union5 オペレータであり，処理は持たない．databox ストリームは，他のクエリよりも大きなサイズのストリームキューを持ち，データを貯める構造を持たせた．Union5 は Filter1 と Filter2 の前に処理されるように設定し，callback2 の出力に絡む処理が後回しにされないようにした．

4.7　設計見直し後安全運転支援クエリ：動作結果

設計見直し後安全運転支援クエリから出力されたトレースログに対してログ解析を行った結果，逸脱ルールからの逸脱は見られなかった．設計見直し後安全運転支援クエリの st9 のストリームキューグラフを図 10 に示す．図 7 と比較すると，図 10 では急激なデータ量の増加も見られない．よって，検証フローに沿った原因特定は正しかったと言える．

4.8　評価

安全運転支援クエリに対して解析ツールを実行した．安全運転支援クエリで逸脱箇所を特定することができた．よって，解析ツールの有効性が確認できた．

Vol.2015-SLDM-170 No.14
Vol.2015-EMB-36 No.14
2015/3/6

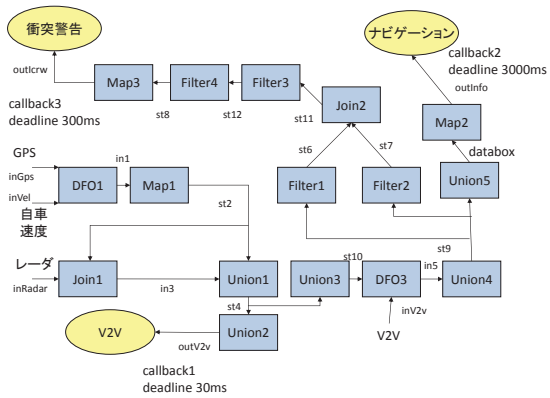

図 9　設計見直し後安全運転支援クエリ

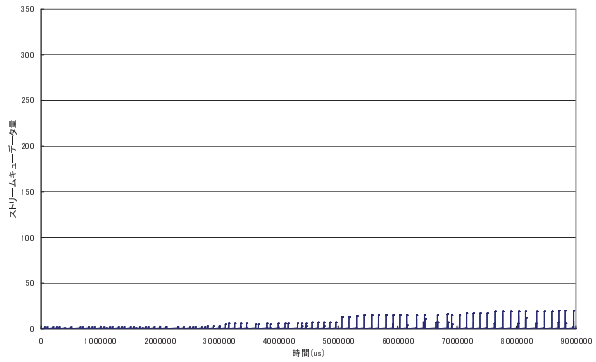

図 10　設計見直し後安全運転支援クエリ：st9 のデータストリームキューのグラフ

また，安全運転支援クエリで発見された逸脱を，検証フローに沿った検証を行うことで，原因を特定し改善した．よって，本研究で開発した可視化ツールと提案した検証フローの有効性を確かめることができた．以上より，動作検証環境の有効性は確認できた．

5. 終わりに

逸脱ルールからの逸脱が確認できたクエリに対して，提案した検証フローをもとに原因特定を行った．その結果，リアルタイム処理の必要があるオペレータとリアルタイム処理の必要がないオペレータ間でストリームを共有するとリアルタイム処理の必要がないオペレータの処理が後回しにされることが原因であると特定した．ボトルネックとなるオペレータの前に，データを貯めるオペレータを設置し逸脱が確認できたストリームにデータがたまり過ぎないように実装した．実装の結果，逸脱ルールからの逸脱は見られなくなり共有するストリームでのデータあふれを改善することができた．検証フローに沿った検証を行うことで，原因を特定し改善したことから可視化ツールと提案した検証フローの有効性を確認した．また，仮に検証フローが存在しない場合でも，オペレータ，ストリームといった 2 つを対象にした柔軟な可視化が行えるためユーザ自身で原因推測を行うことが可能である．

6. 今後の課題

解析ツールの課題は，汎用性がなく，簡単に逸脱ルール項目の追加ができないことである．本研究で作成した逸脱ルールの項目は理論的な裏づけがない．よって，今後 eDSMS のユーザが増えるにしたがって逸脱ルールの内容が変わっていく可能性がある．逸脱ルールの項目をユーザが簡単に増やすことができる解析ツールにすることが今後の課題である．

可視化ツールの課題は，TLV の可視化速度である．可視化ツールのベースとして使用した TLV は，大量のログを可視化する場合非常に時間がかかってしまう．しかし，今後ユースケース次第でさらに大量のログが出力される可能性がある．全体のログを短い時間で可視化するために，可視化速度を高速化する必要がある．

参考文献

[1] Event Log based Dependability Analysis of Windows NT and 2K Systems, Cristina Simache, Mohamed Kaaniche, and Ayda Saidane, : Dependable Computing, 2002. Proceedings. 2002 Pacific Rim International Symposium on(2002)

[2] リアルタイムシステムのための正規表現を用いたログ検索システムの構築とその評価，森本亮太，若林隆行，高田広章，情報処理学会研究報告. SLDM, [システム LSI 設計技術] (2002.03.04)

[3] StreamSqueeze: a dynamic stream visualization for monitoring of event data, Florian Mansmann ; Milos Krstajic ; Fabian Fischer ; Enrico Bertini, SPIE 8294, Visualization and Data Analysis(2012.01.25)

[4] The Design of the Borealis Stream Processing Engine, Daniel J. Abadi, Yanif Ahmad, Magdalena Balazinska, Ugur Cetintemel, Mitch Cherniack, Jeong-Hyon Hwang, Wolfgang Lindner, Anurag S. Maskey, Alexander Rasin, Esther Ryvkina, Nesime Tatbul, Ying Xing, and Stan Zdonik, 2nd Biennial Conference on Innovative Data Systems Research (CIDR'05), Asilomar, CA(2005.01)

[5] Earliest Deadline Scheduling for Continuous Queries over Data Streams, Xin Li ; Sch. of Comput. Sci. Technol., Shandong Univ., Jinan ; Zhiping Jia ; Li Ma ; Ruihua Zhang, Embedded Software and Systems, 2009. ICESS '09. International Conference on, pp.57-64(2009.05)

[6] Hard Real-time Computing Systems : Predictable Scheduling Algorithms And Applications(Real-Time Systems Series), Buttazzo, G. C, Springer (2004).

[7] Dynamic Scheduling of Real-time Tasks Under Precedence Con-straints, Chetto, H., Silly, M. and Bouchentouf, Real-Time Syst., Vol. 2, No. 3, pp. 181-194. (1990)

[8] 車々間通信を用いた安全運転支援のためのリアルタイムストリーム処理，山口晃広，佐藤健哉，中本幸一，渡辺陽介，高田広章，情報処理学会研究 報告. ITS, [高度交通システム] 2014-ITS-58(7), 1-8(2014.09.12)

[9] Scenargie を用いた ITS シミュレーション，大和田泰伯，情報処理学会シンポジウム論文集, 2008 2008(14), 233-234,(2008)

[10] South Australia. Department for Transport, E., Infrastructure, SA., T. and SA, S. A. G. T.: The Driver's Handbook, Department for Transport, Energy and Infrastructure (2005)