情報処理学会論文誌　データベース　Vol.8　No.2　1–17 (June 2015)

車載組込みシステム向けデータストリーム処理のリアルタイムスケジューリング方式

山口 晃広[1,a)]　渡辺 陽介[2]　佐藤 健哉[1,3]　中本 幸一[1,4]　高田 広章[1,2]

受付日 2014年12月21日, 採録日 2015年3月18日

概要：近年，自動車では自動運転や衝突回避などが研究開発され，複数のセンサに加えて車々間通信など入力データの到着タイミングやデータ量が変動する車外からのデータ活用が進められている．このようなセンサ情報処理では，センサからデータが発生してから処理が完了するまでの End-to-End のデッドラインをミスしないリアルタイム制約が要求される．データストリーム処理では，低遅延なデータ処理を実現しながら，クエリにより複雑なセンサ情報処理の開発効率を高めることができ，これまで平均遅延時間の削減など様々な目的に応じたスケジューリング方式がネットワークや金融サービスなどの分野で多く研究されてきた．しかし，ストリーム処理で用いられる従来方式は，リアルタイム制約の維持を目的としておらず，この目的の達成には適切ではない．本論文では，リアルタイムスケジューリングのアルゴリズムである Earliest Deadline First に基づくストリーム処理のスケジューリング方式を，これらのセンサ情報処理に適用可能な方式として実現する．これにより，データ量が増加しても優先度の高いデータ処理を遅らせずに処理できる．車々間通信からの入力データ量が増加する場合における車両衝突警告を想定して，提案方式を評価した．その結果，従来方式と比較して，デッドラインミスを削減し，リアルタイム制約を維持しながら車々間通信からの入力データを多く処理することで車両衝突事故の削減に有効であることを確認した．

キーワード：データストリーム処理，データストリーム管理システム，リアルタイムスケジューリング，Earliest Deadline First，車載システム，安全運転支援システム

Real-time Scheduling Method for Automotive Embedded Data Stream Processing

Akihiro Yamaguchi[1,a)]　Yousuke Watanabe[2]　Kenya Sato[1,3]　Yukikazu Nakamoto[1,4]
Hiroaki Takada[1,2]

Received: December 21, 2014, Accepted: March 18, 2015

Abstract: Recent automotive systems use a variety of sensor data and communications from outside the vehicle to promote autonomous and safe driving. Such sensor data processing requires to maintain real-time constraints, which require to meet End-to-End deadlines between when data is read from a sensor and when it is processed. Data stream processing eases to design the complicated data processing by query description, and process data at a low latency. Scheduling of stream processing has well been studied according to various purposes such as reduction of average latency, in fields such as networks and financial services. However, the existing methods used in stream processing are not intended to maintain the real-time constraints, and are not appropriate to achieve this purpose. In this paper, we propose scheduling methods of stream processing, based on Earliest Deadline First, which is an algorithm of real-time scheduling, so that the method can be applied to the automotive sensor data processing. By the method, higher priority data can be processed without delaying when increasing the data volume. We evaluated the method by assuming the vehicle collision warning in case to increase the input volume of data from vehicle-to-vehicle (V2V) communications. As a result, we confirmed that the proposed method reduced the deadline miss and the vehicle crash by processing more input data from V2V communications while meeting the deadlines, comparing to the existing methods.

Keywords: data streams, DSMS, real-time scheduling, EDF, automotive systems, ADAS

1. はじめに

自動車には，車速センサ，レーダ，レーザ，カメラなど，車両の状態や周辺状況を監視するセンサを車両に搭載した安全運転支援システムが普及し始めている [1], [2]．加えて，Google に代表されるように複数のセンサを車両に搭載し，これらのセンサ情報を融合し高度な制御を行うことで，ドライバが操作することなく市街地を自動走行する研究も行われている [3], [4]．これらのシステムは，複数のセンサにより車両の周辺環境を認識し，その情報をもとに車両の制御やドライバへの警告を行うセンサ情報処理として実現される．

このような自動運転も含めた安全運転支援におけるセンサ情報処理には，平均的に低遅延なだけでなく，センサからデータが発生してからその処理が完了するまでの End-to-End の遅延時間があらかじめ定められた許容時間（End-to-End デッドライン）を超えないリアルタイム制約が求められる．このリアルタイム制約が満たせないと，車両の前方に危険な事象が発生したときドライバへ警告を行うなどの遅延時間を保証できず，車両が衝突するなどの事故が発生しやすい．

前述のように車両に搭載されるセンサ（搭載センサ）のみを用いた安全運転支援では，見通しの悪い交差点からの急な飛び出しなど，走行中に搭載センサで監視できない状況への対応が難しい．このような問題に対して，近年，搭載センサに加えて車々間通信や路車間通信など車外との通信を利用して相互に情報を交換することで，より安全な走行を目指す協調 ITS の活用が進められている [5]．協調 ITS の標準化を進めている欧州標準化機構（ETSI）では，協調 ITS を活用した衝突警告などの仕様の策定を進めている [6]．しかし，車々間通信では，各車両が通常 100 ミリ秒周期でブロードキャストするため [7]，車々間通信から受信するデータの量や到着タイミングは走行状況により変わる．特に，走行する車両で混雑する大規模な交差点などでは一度に大量のデータが送られてくるため，一時的に処理が間に合わない可能性もある．このような状況では，リアルタイム制約を維持して，多くのデータを処理することが重要な課題となる．

[1] 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
Center for Embedded Computing Systems, Nagoya University, Nagoya, Aichi 464–8603, Japan
[2] 名古屋大学未来社会創造機構
Institute of Innovation for Future Society, Nagoya University, Nagoya, Aichi 464–8601, Japan
[3] 同志社大学モビリティ研究センター
Mobility Research Center, Doshisha University, Kyotanabe, Kyoto 610–0321, Japan
[4] 兵庫県立大学大学院応用情報科学研究科
Graduate School of Applied Informatics, University of Hyogo, Kobe, Hyogo 650–0047, Japan
[a)] yamagut@nces.is.nagoya-u.ac.jp

データレートの変化する連続的なデータを低遅延に処理するには，データストリーム処理（ストリーム処理）が好適である．ストリーム処理では，クエリはデータ処理の実行前にあらかじめ登録される．登録されたクエリは，オペレータの入力や出力をストリームでつないだデータフロー形式で表現される．このクエリの形式により，一部のオペレータを変更しストリームをつなぎ変えることで，センサやアプリケーションの追加や変更に対応しやすい [8], [9]．また，ストリーム処理では，データをタプルの無限列であるストリームとして表現し，キューを用いてタプルの入出力を実現することで，データレートの増減にも対応しやすい．これまで我々は，自動車のセンサ情報処理を効率的に開発するためのソフトウェアプラットフォームとしてデータストリーム管理システム（DSMS）を採用し [10]，車載システムに組み込んでそのセンサ情報を低遅延に処理するための DSMS（車載組込みシステム向け DSMS）を研究開発してきた [11]．

ストリーム処理の性能改善の研究として，平均的な遅延時間を削減する First In First Out（FIFO）[12] や Path Capacity Strategy（PCS）[13]，スループットを向上させる Min-Cost（MC）[14] など，様々な目的に応じたスケジューリング方式がこれまで多く研究されてきている．しかしながら，これらの従来方式では，リアルタイム制約の維持を目的としておらず，この目的の達成には適切ではない．

本論文では，このようなリアルタイム制約の維持が求められるセンサ情報処理を対象としたストリーム処理のスケジューリング方式を提案する．提案方式では，リアルタイムスケジューリングの分野でよく用いられる Earliest Deadline First（EDF）をストリーム処理のスケジューリングに適用する．これにより，デッドラインの早いタプルを優先して処理し，データ量が増加した場合でも優先度の高いタプルを遅らさずに処理できる．また，本方式は，各出力に対して異なる End-to-End デッドラインを持つマルチクエリや，タイムアウトを持つオペレータにも対応できるため，安全運転支援におけるセンサ情報処理にも適用可能である．

本論文の構成は以下のとおりである．まず，2 章で本研究が解決する課題を述べ，3 章で関連研究を紹介する．次に，4 章で本論文が想定するモデルを述べ，5 章で提案方式を示し，6 章でその評価を行う．最後に，7 章では課題に対する解決を考察し，8 章でまとめと今後の課題を述べる．

2. ストリーム処理のスケジューリング方式における課題

図 1 は，見通しの悪い交差点における車々間通信を用いた衝突警告のシナリオを表している．自車両 A は，車両 X を早期に発見してブレーキをかけなければ車両 X と衝突してしまうが，見通しが悪いためにレーダやカメラなどの搭

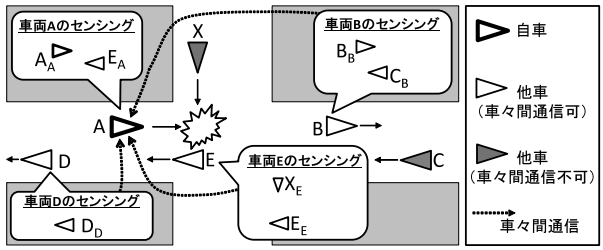

図 1　本論文で用いるユースケースシナリオ
Fig. 1　Usecase scenario in this paper.

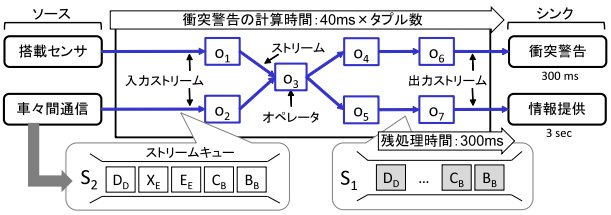

図 2　ユースケースシナリオにおける単純化されたストリーム処理
Fig. 2　Simplified stream processing for the usecase scenario.

載センサでは車両 X を早期に検知できない．一方，車両 E は，それに搭載されたセンサにより車両 X を早期に検知できる．車両 B，D，E は，自車両 A と車々間通信ができるが，車両 X，C とは車々間通信できない．そのため，車両 B，D，E はそれらの搭載センサでセンシングした車両の情報を車々間通信を用いてブロードキャストし，自車両 A はこれらのメッセージを受信して衝突警告を早期に処理することで衝突を回避できる．ここでは，ETSI の仕様 [6] に合わせて衝突警告に対する End-to-End デッドラインを 300 ミリ秒とし，車両 X のデータを車両 E から受信し 300 ミリ秒以内に処理しなければ，自車両 A と車両 X は衝突する状況を想定する．なお，図中の Y_Z は，車両 Y をセンシングした結果を車両 Z が送信するメッセージを表す．

図 2 は，図 1 の衝突警告と情報提供の 2 つのアプリケーションを実現するための簡単化したストリーム処理であり，2 つの出力ストリームを持つ．衝突警告を配信する出力ストリームの End-to-End デッドラインは 300 ミリ秒とする．一方，情報提供の出力ストリームの End-to-End デッドラインは 3 秒と，十分長く設定する．ここでは議論を簡単にするため，各オペレータの選択率や計算時間は同一であり，現時刻において，情報提供のみに用いられる古いタプルの列（S_1）がキューに残っており，車々間通信から得られたメッセージ B_B，C_B，E_E，X_E，D_D がタプルの列（S_2）として到着した状況を仮定する．また，S_1 のすべての処理を完了するまでの残りの処理時間を 300 ミリ秒と仮定し，S_2 の 1 タプルあたりが衝突警告の処理にかかる計算時間を 40 ミリ秒と仮定する．

従来方式としてたとえば FIFO でスケジューリングする場合，先に到着した S_1 を S_2 よりも先に処理する．その結果，X_E を含む S_2 の処理は衝突警告の End-to-End デッドラインである 300 ミリ秒に間に合わず，自車両 A は車両 X と衝突してしまう．一方，S_2 の衝突警告を処理した後で，S_1 を処理するようにスケジューリングする場合，S_1，S_2 のすべてのタプルが各出力ストリームの End-to-End デッドラインをミスせずに処理されることとなり，自車両 A は車両 X との衝突を回避できる．

このような課題を解決するため，以下の項目 I1–5 への対応ができるスケジューリング方式が必要となる．

I1：リアルタイムスケジューリングの導入
自動車の安全運転支援におけるセンサ情報処理では，リアルタイム制約の維持が重要な課題となる．これは，先ほどの例のように，デッドラインの早い順に処理順序を決定するリアルタイムスケジューリングアルゴリズムである EDF に基づき，ストリーム処理をスケジューリングすることで実現される．そのため，ストリーム処理にリアルタイムスケジューリングを行う方式が必要となる．

I2：異なるデッドラインを持つマルチクエリへの対応
安全運転支援におけるセンサ情報処理では，共通のセンサから得られる自車両や周辺車両の位置や速度といったデータなどは衝突警告や情報提供など複数のアプリケーションで利用されうる．そのため，出力を複数に分岐するオペレータ（図 2 の例では O_3）が含まれるクエリ（マルチクエリ）に対応する必要がある．また，車両制御や衝突警告，情報提供など，アプリケーションによって End-to-End デッドラインは異なる．そのため，各出力ストリームの End-to-End デッドラインが異なるマルチクエリへの対応が必要となる．

I3：タイムアウトを持つオペレータへの対応
車々間通信や路車間通信など車外との通信は，自動車の走行状況によって一時的に利用できない場合があり，一時的に特定の入力ストリームにデータが入力されない状況が発生する可能性がある．そのため，このような状況でリアルタイム制約を維持するためには，搭載センサなど確実に利用可能な入力のみを用いる処理と多重化・冗長化してクエリを設計する．そのため，多重化や冗長化した処理を統合するオペレータでは，ある入力にタプルが到着した後，一定時間待っても他の入力にタプルが到着しない場合，到着したタプルのみを用いて処理を実行するように，タイムアウトを行う必要がある．

I4：オーバロード時への対応
特に車々間通信からの入力データ量は増加する可能性が高く，ETSI では 1 秒間に最大 1000 メッセージを受信できることを要求している [6]．車々間通信のメッセージとして用いられる CAM（Cooperative Awareness Message）

の場合，1 メッセージに車両の 1 台分の情報が含まれるため [7]，1 秒間に最大 1000 台の車両情報が受信される想定となる．しかしながら，このようなセンサ情報処理には車両 1 台につき数十ミリ秒の計算時間がかかる場合があるため [15], [16]，リアルタイム制約を満たして車々間通信から得られるデータをすべて処理することは難しい．そのため，リアルタイム制約を維持するように車外からの入力データをフィルタリングすることが必要となり，リアルタイム制約を維持しながら多くのタプルを処理する必要がある．

I5：スケジューリングにおけるオーバヘッドの削減
スケジューリングにおけるオーバヘッドを削減するため，多くのスケジューリング方式では，実行順序が確定するオペレータの列（オペレータトレイン）をクエリ実行前にあらかじめ構成し，オペレータ単位ではなくオペレータトレインに対してスケジューリングを行う．この場合，あるオペレータトレインの実行中に，それよりも優先度の高いオペレータトレインが実行可能となった場合には，実行中のオペレータトレインを一時中断するプリエンプティブなスケジューリングと，そのような一時中断が発生しないノンプリエンプティブなスケジューリングに分類できる．リアルタイム制約を維持するためには，現在実行中の処理よりもデッドラインの早い処理（図 2 の例では S_2 の処理）が発生したときには，その処理に切り替えるべきである．そのため，本方式でも，オペレータトレインを構成し，プリエンプティブにスケジューリングする必要がある．

3. 関連研究

これまでストリーム処理のスケジューリングでは，メモリ使用量の削減や平均遅延時間の削減，スループットの向上など，様々な目的を達成するための方式が提案されてきた．しかし，これらの従来方式はリアルタイム制約を目的としておらず，2 章の I1 に対応できない．文献 [17] では，クエリの単位をオペレータトレインとして，クエリを構成するオペレータの計算時間や選択率といった統計情報に基づき，ストリームキューのメモリ使用量を削減する，プリエンプティブなスケジューリング方式を提案した．また，FIFO スケジューリングを平均的な遅延時間を削減する方式のベースラインとし，メモリ使用量を削減しながら遅延時間の削減を FIFO に近づける方法なども提案されている [18]．文献 [14] では，オペレータやタプルをバッチ化することで，スケジューリングのオーバヘッドを削減する方式が提案されており，superbox と呼ばれるオペレータトレインに対して，スループットを向上させる MC や，平均遅延時間やメモリ使用量を削減する，プリエンプティブな方式を提案している．文献 [12], [13] では，オペレータの計算時間や選択率を与えられたもとで，シングルプロセッサ上で，平均または合計遅延時間を最小にする PCS という方式を提案している[*1]．PCS では，オペレータパスをオペレータトレインとして，単位時間あたりに各オペレータパスで消費するタプル数（プロセッシングキャパシティ）が大きいオペレータパスからプリエンプティブにスケジューリングする．PCS はマルチクエリにも対応できるが [12]，クエリを構成する各オペレータがオペレータパスに沿って処理されるという前提があるため，タイムアウトを持つオペレータへの対応方法は自明ではない．

一方，リアルタイムスケジューリングをストリーム処理に適用する方式が少数提案されている．しかし，これらの方式では適用範囲に制限があり，2 章の I2 や I3 に対応することは難しい．文献 [19] は，静的でプリエンプティブなリアルタイムスケジューリングアルゴリズムとして知られる Rate monotonic に基づく方式を提案している．しかし，この方式は 1 つのクエリに 1 つのデッドラインを指定するため，2 章の I2 に対応できない．また，周期的な入力データに限定するため，車々間通信など車外からの非周期な入力データに適さない．文献 [20] では，EDF に基づき，タプルを入力するときにそのデッドラインを決定する動的でプリエンプティブなリアルタイムスケジューリング方式を提案しており，マルチクエリにも対応できる．しかし，この方式では各出力ストリームの End-to-End デッドラインがすべて同一であることが前提となり，2 章の I2 に対応できない．また，クエリを構成する各オペレータはタプルが入力されるとただちに結果を出力することを前提としており，2 章の I3 に対応できない．文献 [14] では，遅延時間に対して徐々に価値が下がるようなソフトリアルタイムを想定して，各出力ストリームの遅延時間に対して QoS を指定し，それに基づきオペレータ単位でスケジューリングする方式も提案されている．しかし，この方式では各オペレータの出力するストリームが 1 つである必要があるため，2 章の I2 に対応できない．

車載組込みシステム向け DSMS もいくつか提案されているが，安全運転支援におけるセンサ情報処理で重要となるリアルタイム制約について述べられていない．文献 [21] では，車載システムを診断や検査するための DSMS が提案された．StreamCars [22] では，自動車の衝突警告などの安全運転支援のアプリケーションを対象とした DSMS を提案しており，我々の研究目的と類似する．しかしながら，これらでは 2 章の I1 に対応せず，本論文の課題を解決できない．

4. 本研究で想定するモデル

本章では，本研究で対象とするストリーム処理のモデルを説明する．このモデルは Aurora [23] や Borealis [24],

[*1] PCS では，出力ストリームにタプルが挿入されるまでの End-to-End の遅延時間だけでなく，選択率が 1 より小さいオペレータでタプルがドロップされた場合そこまでの遅延時間も含む．

SystemS [25] などデータフローとしてクエリを表すストリーム処理と同様である．End-to-End デッドラインを追加する従来のモデルでは，クエリ（マルチクエリの場合は出力ストリーム）に指定する方式 [19] と，クエリの入力ストリームに指定する方式 [20] がある．本研究のモデルでは，2 章の I2 に対応するため，文献 [19] と同様，各出力ストリームに End-to-End デッドラインを指定する．

本論文で想定するクエリは，図 2 のように，オペレータの入出力をストリームでつないだデータフローとして表現される．センサなど，DSMS の外部からデータを提供する入力源をソースと呼び，アプリケーションプログラムなど，DSMS 外部にある処理結果の配信先をシンクと呼ぶ．ソースからの入力やシンクへの出力もストリームであり，これらをそれぞれ入力ストリームまたは出力ストリームと呼ぶ．オペレータは，入力としてつながれたストリームからタプルを取り出して処理し，出力としてつながれたストリームへタプルを流す．オペレータは複数のストリームにその処理結果を配信できるマルチクエリを対象とする．オペレータの種類には，ユーザがパラメータのみを指定する基本的な演算（結合，射影，合流など）と，ユーザがプログラミング言語で処理を記述するものがある．特に，出力ストリームに直接つながっているオペレータを出力オペレータと呼ぶ．クエリの実行時に，オペレータへ入力するストリームに含まれるタプルなど，オペレータ o が入力に用いることのできるタプルの集合を $\mathrm{InTpl}(o)$ と記述する．たとえば，図 2 の $\mathrm{InTpl}(o_2)$ は $\{B_B, C_B, E_E, X_E, D_D\}$ となる．

クエリは，オペレータ，ソース，シンクを頂点とし，それをつなぐストリームを辺と見なすことで，グラフ（クエリグラフ）として表現される．本論文で想定するクエリグラフは有向非巡回グラフ（DAG）でありループを含まない．オペレータの入次数や出次数は 1 以上で，ソースの入次数は 0 で出次数は 1 以上，シンクの入次数は 1 で出次数は 0 とする[*2]．なお，オペレータ o の出次数を $\mathrm{ODeg}(o)$ と記述する．図 2 の例では，$\mathrm{ODeg}(o_3) = 2$ であり，他のオペレータの出次数は 1 である．あるオペレータ o と隣接する後続のオペレータの集合を $\mathrm{SucOp}(o)$ と記述する．逆に，あるオペレータ o と隣接する先行のオペレータの集合を $\mathrm{PreOp}(o)$ と記述する．あるオペレータ o とパスでつながる後続のオペレータと o 自身を含めた集合を $\mathrm{SucAllOp}(o)$ と記述する．図 2 の例では，$\mathrm{SucOp}(o_3) = \{o_4, o_5\}$，$\mathrm{PreOp}(o_4) = \{o_3\}$，$\mathrm{SucAllOp}(o_3) = \{o_3, o_4, \cdots, o_7\}$ などとなる．

各出力ストリームには，センサからデータが発生してから，そのデータがタプルとしてそこに挿入されるまでの End-to-End デッドラインが指定されている場合を想定する．リアルタイム制約は，タプルがそれらの出力ストリームに挿入されるときに，その End-to-End デッドラインをミスしないこととする．各タプル p にはセンサからデータが発生した時刻でタイムスタンプ t_p を打刻する．リアルタイム制約とは，タプル p を出力ストリーム s に挿入する時刻が，式 (1) で定義される絶対デッドライン $d_{p,s}$ より遅れないことと同値である．

$$d_{p,s} = l_s + t_p \tag{1}$$

なお，l_s はその出力ストリーム s に指定された End-to-End デッドラインである．各オペレータからタプルが出力されるとき，そのタプルのタイムスタンプは入力に用いたタプルのタイムスタンプの中で最古のものを用いる．本論文では EDF に基づいてリアルタイムスケジューリングを行う．EDF はシングルプロセッサが前提のアルゴリズムであるため，本論文でもシングルプロセッサを仮定して議論する．

5. ストリーム処理のリアルタイムスケジューリング方式

本章では，本論文で提案するストリーム処理のリアルタイムスケジューリング方式を説明する．まず，5.1 節でシステムの概要を説明したのち，基本的な方法を 5.2 節で述べる．次にその拡張として，効率的なデータ処理を実現するための方法を 5.3–5.4 節で述べ，タイムアウトを持つオペレータを含む場合への対応を 5.5 節で述べる．

5.1 システムの概要

提案方式に基づくシステムの概要を**図 3** に示す．クエリの各出力ストリームに対して，ユーザは End-to-End デッドラインを指定する．これにより，タプルが入力されたとき，各出力ストリームに対して式 (1) からそのタプルの絶対デッドラインが求まる．リアルタイムスケジューリングを行う EDF スケジューラは，この絶対デッドラインをミスしないようにオペレータを実行しタプルを処理する．

車外との通信は一時的に切断される可能性があるため，これらを入力に用いる処理は，搭載センサのみを用いる処理と，冗長化しタイムアウトを持つオペレータでそれらを

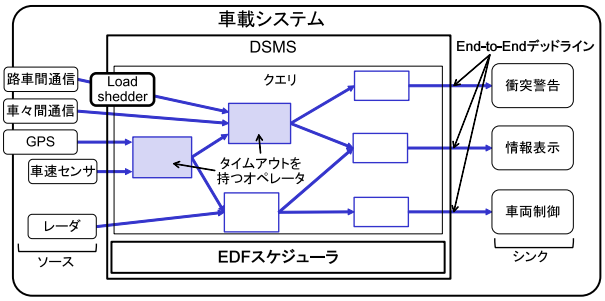

図 3　提案方式によるシステムの概要
Fig. 3　Overview of the system by the proposal method.

[*2] 一般のクエリグラフではシンクの入次数は 1 以上であるが，本論文では議論を簡単にするため入次数を 1 に限定する．シンクの入次数が 2 以上の場合には，計算時間が非常に小さく選択率が 1 のダミーオペレータをそのシンクの入力ストリームに挟むことで，シンクの入次数が 1 のクエリグラフに変換できる．

Algorithm 1 スケジューリング方法（基本）

Require: オペレータ o でタプル p の処理が実行可能となる．
Require: すでにスケジューリングされ実行待ちとなっているタプルとオペレータとその絶対デッドラインの組からなる集合を S とし，$(p, o, d_{p,o}) \notin S$ である．
1: $(o_1, o_2, \cdots, o_n) \leftarrow$ SucAllOp(o) をトポロジカルソートの逆順に並べたリスト
2: **for** $i = 1$ **to** n **do**
3:　**if** o_i はある出力ストリーム s の出力オペレータである **then**
4:　　$d_{p,o_i} \leftarrow t_p + l_s$
5:　**else**
6:　　$d_{p,o_i} \leftarrow \min\{d_{p,o'} - c_{o'}; o' \in \mathrm{SucOp}(o_i)\}$
7:　**end if**
8: **end for**
9: S に $(p, o, d_{p,o})$ を追加する．
10: $(\hat{p}, \hat{o}) \leftarrow \underset{(p',o',d_{p',o'}) \in S}{\arg\min} \{d_{p',o'}\}$
11: タプル $\hat{p}$ を処理するように，オペレータ $\hat{o}$ を実行する．

統合するように，クエリを設計する．これにより，車外からのデータの到着が遅れた場合，タイムアウトして搭載センサのみを用いた最低限の処理でリアルタイム制約を維持する．

車々間通信のようにフィルタリングが必要な入力ストリームに対して，ユーザは load shedder を設定できる．Load shedder には，単位時間あたりに許容できるタプルの最大入力量と，タプルの価値を評価するフィルタリングの条件を登録できる．Load shedder は，設定された入力ストリームの入力データ量が最大値を超えない分だけ，価値の低いデータを削除する．

安全運転支援を対象とした車載システムでは，安全性などの観点から，クエリはソフトウェアの開発時に組み込まれることを想定する．そのため，データ処理を実行する前のクエリを登録するときに行う処理には，性能の高い計算機で十分な計算時間をかけられる．

5.2　プリミティブなスケジューリング

本節では，まず基本的な方法として，タイムアウトを持つオペレータを含まない場合で，オペレータトレインを行わずオペレータ単位でスケジューリングする方法を述べる．この場合，スケジューリングするタイミングは，オペレータへ入力するストリームにタプルが挿入されたときである．EDF に基づく本方式では，各オペレータ o のタプル $p \in \mathrm{InTpl}(o)$ に対して，(p, o) の絶対デッドラインの早い順に p を o で処理する．

次に，(p, o) の絶対デッドライン $d_{p,o}$ を算出する方法を述べる．o が出力オペレータの場合，ユーザがその出力ストリーム s に End-to-End デッドライン l_s を指定しているため，式 (1) から $d_{p,o} = t_p + l_s$ と算出できる．o がそれ以外のオペレータの場合には，リアルタイムスケジューリングの既存手法である EDF* [26] の漸化式 (2) から導出される．

$$d_{p,o} = \min\{d_{p,o'} - c_{o'}; o' \in \mathrm{SucOp}(o)\} \quad (2)$$

なお，$c_{o'}$ はオペレータ o' が入力したタプルを処理する計算時間である．一般のシングルプロセッサのリアルタイムシステムにおいて，EDF*では，順序依存を持つタスク間で，後続のタスクのデッドラインと計算時間から，先行のタスクのデッドラインを導出することが可能であり，EDF*で求めたデッドラインは順序依存のない通常の EDF における適切なデッドラインとなることが知られている [26], [27]．本方式では，出力ストリーム側から入力ストリーム側へ式 (2) を再帰的に適用することで，各オペレータにおけるタプルの絶対デッドラインを適切に設定する．

アルゴリズム 1 に，本方式の処理をまとめる．これは，あるオペレータ o へ入力するストリームにタプル p が挿入されるときに実行される．ただし，1 行目の処理はクエリ登録時にあらかじめ実施しておく．1 行目のトポロジカルソートは，クエリグラフが DAG のため可能であり，その逆順でソートした末尾のオペレータ o_n が o と一致する．2–8 行目で，出力ストリーム側のオペレータから，式 (1) と (2) を再帰的に用いることで，o における p の絶対デッドラインを計算する．9 行目で，$(p, o, d_{p,o})$ をスケジューリングの対象として EDF スケジューラに登録する．10 行目で，最もデッドラインの早いタプルとそれを入力するオペレータのペアを見つけ，11 行目でそれを実行するようにスケジューリングする．タプル p の処理をオペレータ o が完了した時点で，スケジューリングの対象から $(p, o, d_{p,o})$ を消去する．

リアルタイムスケジューリングの分野で知られる Jackson's rule [27] から，本節の方式は定理 1 の意味で最適なスケジューリング方式である．Jackson's rule では，実行順序に依存関係のないスケジューリング対象が与えられたとき，それらを絶対デッドラインの早い順にスケジューリングすれば，maximum lateness というリアルタイム制約の指標が最も良くなる（小さくなる）ことが示されている [27]．なお，この定理ではスケジューリングのオーバヘッドを想定しない．

定理 1. タイムアウトを持つオペレータが含まれない場合で，オペレータとタプルのある n 個のペアからなる集合 $S_n = \{(p_i, o_i); p_i \in \mathrm{InTpl}(o_i), i = 1, \cdots, n\}$ の実行順序を決定するとき，すべての $i = 1, 2, \cdots, n$ におけるペア (p_i, o_i) に対して，絶対デッドライン d_{p_i,o_i} と計算時間 c_{o_i} がスケジューリング方式によらず同一であれば，本方式で絶対デッドラインをミスする場合には，他のスケジューリング方式でも絶対デッドラインを必ずミスする．

略証. 以下（A），（B）を仮定しても一般性を失わない．

（A）　$(p_1, o_1), (p_2, o_2), \cdots, (p_n, o_n)$ は本方式の実行順である．

（B）　他の方式は $i = i'_1, i'_2, \cdots, i'_n$ の順に (p_i, o_i) を実行

する．$(i'_1, i'_2, \cdots, i'_n)$ とは $(1, 2, \cdots, n)$ の並べ替えである．

$L := \max_{k=1,\cdots,n}\{\sum_{i=1,\cdots,k} c_{o_i} - d_{p_k,o_k}\}$ と，$L' := \max_{k=1,\cdots,n}\{\sum_{j=1,\cdots,k} c_{o_{i'_j}} - d_{p_{i'_k},o_{i'_k}}\}$ とおく[*3]．S_n の実行順序には依存関係がないため，S_n に Jackson's rule を適用できて，$L \leq L'$ が成り立つ．スケジューリング方式（A）がデッドラインをミスすることは $0 < L$ と同値であり，$0 < L'$ ならばスケジューリング方式（B）はデッドラインをミスする．そのため，本方式で S_n がデッドラインをミスすれば，他のスケジューリング方式でも S_n は必ずデッドラインをミスする．□

5.3　トレインスケジューリング

5.2 節で述べた EDF スケジューラは，絶対デッドラインに基づき，各オペレータの単位でスケジューリングしていたため，そのオーバヘッドが大きい．そのため，スケジューラを介さずに連続して実行するオペレータの列（オペレータトレイン）をクエリ登録時に構成し，その単位でスケジューリングすることを考える．オペレータトレインは，オペレータパスの連結する一部として構成される．なお，オペレータパスとは，クエリグラフにおいて，あるソースからあるシンクへのパスから，そのソースとシンクを除いたパスのことである．5.4 節では，定理 1 の最適性を維持するオペレータトレインの構成方法を述べる．オペレータ $o_1, o_2, \cdots, o_n$ の列からなるオペレータトレイン O に対して，あるタプル p を処理するときの絶対デッドラインは，$D_{p,O}$ または $D_{p,(o_1,o_2,\cdots,o_n)}$ と記述され，末尾のオペレータ o_n の絶対デッドラインとして式 (3) のように計算される．

$$D_{p,O} = d_{p,o_n} \tag{3}$$

オペレータトレインの中では，先頭のオペレータから順に後続のオペレータを実行していく．もし，オペレータトレインの途中のオペレータで処理するタプルがなくなった場合，そのオペレータトレインの実行は完了する．

あるオペレータトレインの実行中に，それよりも優先度の高い（つまり，絶対デッドラインの早い）オペレータトレインが実行可能となった場合には，実行中のオペレータトレインはプリエンプションされる．ただし，オペレータの実行中にはプリエンプションは発生せず，オペレータの実行が完了したのち発生する．プリエンプションが発生してオペレータトレインの処理が途中で中断した場合には，オペレータトレインがどこまで実行されたかを覚えておき，そのオペレータトレインの優先度が最も高くなった（つまり，最も絶対デッドラインが早くなった）ときに，中断したところから実行を再開する．

[*3] L または L' は，スケジューリング方式（A）または（B）の maximum lateness [27] である．

複数のタプルをバッチ化して，その単位でスケジューリングすることでも，スケジューリングのオーバヘッドを削減できる．バッチ化したタプルのタイムスタンプは，構成するタプルのタイムスタンプの中で最古のものを用いる．ただし，スケジューリングの精度が悪くなり，バッチ化が完成するまでの遅延時間も発生するため，タプルのバッチ化にはトレードオフがある．また，5.4 節のオペレータトレインとは異なり，タプルをバッチ化すると定理 1 の最適性は保証されない．本論文では，タプルをバッチ化する構成方法はユーザがアプリケーションに応じて決めるものとし，その具体的な方法を議論しないが，後述の “タプル” を “バッチ化したタプル” に読み替えることで，タプルをバッチ化した場合にも提案方式を適用できる．

アルゴリズム 2 に，本方式の処理をまとめる．タイムアウトによる実行ではない場合，オペレータトレインを構成する先頭のオペレータへ入力するストリームにタプルが挿入されたときに実行される[*4]．この場合，アルゴリズム 2 のタプル集合 P は，要素数が 1 個のそのタプルとなる．しかし，5.5 節で後述するように，タイムアウトにより実行される場合は，複数のタプルを入力に用いる場合があるため，オペレータトレインはタプルの集合とペアでスケジューリングする．オペレータトレイン O におけるタプル集合 P の絶対デッドライン $\hat{D}_{P,O}$ は，オペレータトレイン O における，タプル集合 P の中で最古のタイムスタンプを持つタプルの絶対デッドラインであり，$\hat{D}_{P,O} := \min_{p \in P}\{D_{p,O}\}$ と表せる．オペレータトレイン，ソース，シンクを頂点として，オペレータトレイン間のストリームを辺と見なすと，それはクエリグラフと同様に DAG となる．ここで，あるオペレータトレイン O と隣接する後続のオペレータトレインを $\mathrm{SucTr}(O)$ と記述し，あるオペレータトレイン O とパスでつながっている後続のオペレータトレインと O 自身を含めた集合を $\mathrm{SucAllTr}(O)$ と記述する．1–9 行目で $D_{P,O}$ を計算する．なお，c_O はオペレータトレインの計算時間である．11–19 行目では，オペレータの実行順序を決定する．現在実行中の処理がある場合，12 行目で優先度を比較し，現在実行中の処理よりも優先度が高ければ，それをプリエンプションして (P, O) を実行し（15 行目），そうでなければ現在実行中の処理を継続する（13 行目）．現在実行中の処理がなければ (P, O) を実行（18 行目）する．

5.4　オペレータトレインの構成方法

本節では，クエリからオペレータトレインを構成する方法を述べる．特に，本節の方法で構成したオペレータトレインは，5.2 節の方法で End-to-End デッドラインをミスしないならば，このオペレータトレインでスケジューリングしてもミスしないことが後述の定理 2 により保証される．

[*4] ただし，1 行目の処理はクエリ登録時にあらかじめ実施しておく．

Algorithm 2 スケジューリング方法（拡張）

Require: オペレータトレイン O でタプルの集合 P が実行可能.
Require: すでにスケジューリングされているタプル集合とオペレータトレインと $\hat{D}_{P,O}$ のペアからなる集合を S とし，$(P, O, \hat{D}_{P,O}) \notin S$.
1: $(O_1, O_2, \cdots, O_n) \leftarrow$ SucAllTr(O) をトポロジカルソートの逆順に並べたリスト
2: $p = \arg\min_{p \in P}\{t_p; t_p$ はタプル p のタイムスタンプ $\}$
3: **for** $i = 1$ **to** n **do**
4:　**if** O_i の末尾はある出力ストリーム s の出力オペレータである **then**
5:　　$\hat{D}_{P,O_i} \leftarrow t_p + l_s$
6:　**else**
7:　　$\hat{D}_{P,O_i} \leftarrow \min\{\hat{D}_{p,\hat{O}} - c_{\hat{O}}; \hat{O} \in \mathrm{SucTr}(O_i)\}$
8:　**end if**
9: **end for**
10: S に $(P, O, \hat{D}_{P,O})$ を追加
11: **if** あるオペレータトレイン $(P', O', \hat{D}_{P',O'})$ を現在実行中 **then**
12:　**if** $\hat{D}_{P',O'} \leq \hat{D}_{P,O}$ **then**
13:　　O' の実行を継続
14:　**else**
15:　　O' の実行をプリエンプトし，P を処理するよう O を実行
16:　**end if**
17: **else**
18:　P を処理するよう O を実行
19: **end if**

Algorithm 3 オペレータトレインの構成方法

Require: クエリを構成するオペレータを $o_1, o_2, \cdots, o_n$ とする.
1: $L \leftarrow ((o_1), \cdots, (o_n))$
2: **for all** $O \in L$ **do**
3:　$o \leftarrow O$ の先頭のオペレータ
4:　**if** o と PreOp(o) が表 1 の条件をすべて満たす **then**
5:　　**for all** $o' \in$ PreOp(o) **do**
6:　　**for all** $O' \in L$ such that O' の末尾は o' である **do**
7:　　O の先頭に O' を追加し，O' を L から削除
8:　　**end for**
9:　　**end for**
10:　　goto 2
11:　**end if**
12: **end for**
13: **return** L

表 1　オペレータ o と PreOp(o) におけるオペレータトレインの条件
Table 1　Conditions of operator train between o and PreOp(o).

条件 1	オペレータ o はタイムアウトを持たない
条件 2	$\forall o' \in$ PreOp(o) に対して，ODeg(o') = 1

このオペレータトレインは，**表 1** の条件をすべて満たすオペレータ o と $o' \in$ PreOp(o) をつなぐことで構成される．条件 1 は，o がタイムアウトを持つ場合には，o の実行は，タイムアウトによりスケジューラから呼び出され，スケジューラの介在を必要とするためである．条件 2 は，複数のストリームに出力する o' と o をつなげると，後述の定理 2 が適用されず，リアルタイム制約を満たせなくなる場合があるためである．オペレータトレインを構成する方法をアルゴリズム 3 にまとめる．アルゴリズム 3 はクエリの登録時に実行すればよい．

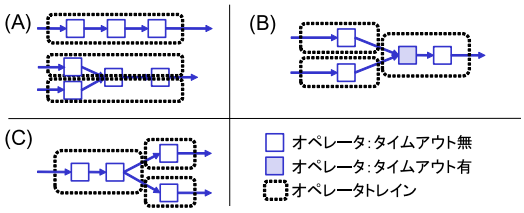

図 4　オペレータトレインのパターン
Fig. 4　Patterns of operator train.

アルゴリズム 3 により構成されるオペレータトレインの典型的なパターンを**図 4** に示す．(A) は最も基本的なパターンですべてのオペレータパス全体がオペレータトレインとなる．(B) はタイムアウトを持つオペレータが含まれる場合に，その直前でオペレータトレインが分割される例である．(C) は複数のストリームに出力するオペレータの直後では，オペレータトレインが分割される例である．

本節の以降では，この方式により構成したオペレータトレインの性質を示すために，クエリを構成する任意の 1 つのオペレータトレインに着目し，それを構成した場合（適用時）と構成しなかった場合（非適用時）とでスケジューリングの違いを調べる．オペレータトレインの長さが 1 の場合は，オペレータトレインを行っても単一のオペレータと変わらないため，自明であり，以降では着目したオペレータトレインの長さが 2 以上の場合を仮定する．本節の以降では，このオペレータトレインを O または $\langle o_1, s_2, o_2, s_3, \cdots, s_{n+1}, o_{n+1}\rangle$ と記述する．なお，s_i と o_i はクエリを構成するストリームとオペレータを表し，オペレータトレインの長さは $n+1$ である．後述するように，オペレータトレイン適用時と非適用時で，ストリームの種類によりタプルの処理順序に違いが生ずる場合がある．そのため，ストリームを以下のようなクラス (A)–(C) に分類する．

(A)　s_{n+1}（これに属するストリームは 1 つ存在する）

(B)　$s_2, \cdots, s_n$（n=1 ならば，これに属するストリームは存在しない）

(C)　クエリの中で，クラス A でも B でもないストリーム

このとき，次の定理 2 が成り立つ．

定理 2. 本方式を用いて着目した任意のオペレータトレイン $\langle o_1, s_2, o_2, s_3, \cdots, s_{n+1}, o_{n+1}\rangle$ をスケジューリングする場合（オペレータトレイン適用時），それを個別のオペレータ $o_1, o_2, \cdots, o_n$ としてスケジューリングした場合（非適用時）と比較して，クエリの各出力ストリームにタプルが挿入される時刻は遅れない．ただし，クエリはタイムアウトを持つオペレータを含まず，オペレータトレイン適用時では非適用時に比べてスケジューリングのオーバヘッドは増加しないことを前提とする．

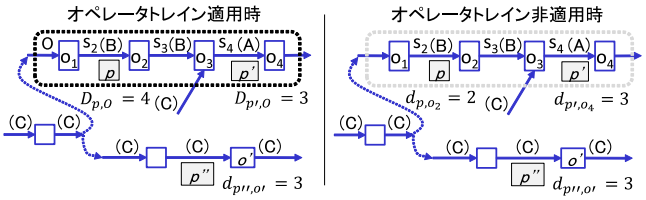

図 5　オペレータトレインの適用/非適用時における違い
Fig. 5　Difference in with/without operator train.

証明のアイデア. まず，**図 5** の例を用いて証明の基本的なアイデアを説明する．この例では，注目するオペレータトレインは，$O=\langle o_1,s_2,o_2,s_3,o_3,s_4,o_4\rangle$ であり，あるタプル p がクラス B の s_2 に含まれ，別のタプル p' と p'' がクラス A や C のあるストリームに含まれる場合を考える．また議論を簡単にするため，図 5 のすべてのオペレータの計算時間や選択率は 1 とし，$D_{p,O}=4$，$D_{p',O}=d_{p',o_4}=3$，$d_{p'',o'}=3$ とする．$D_{p,O}=4$ より，式 (2) を o_3 と o_2 と順に適用することで，$d_{p,o_3}=3$ と $d_{p,o_2}=2$ となる．その結果，オペレータトレイン適用時では $D_{p',O}<D_{p,O}, d_{p'',o'}<D_{p,O}$ なので p'，p'' を p よりも先に実行する．それに対して，非適用時では $d_{p,o_2}<d_{p',o_4}, d_{p,o_2}<d_{p'',o'}$ なので p'，p'' を p よりも後に実行する．このように適用時にはクラス B のストリームに含まれる p の処理が後回しとなる．しかし，適用/非適用時にかかわらず，p が s_4 に挿入されて o_4 を実行するときには p' や p'' の処理はすでに終わっている．そのため，オペレータ o_4 で p を処理する時刻は変わらず，オペレータトレイン適用時に処理が後回しとなった p でも，出力ストリームに挿入される時刻は遅れない．

略証. オペレータトレイン適用時では非適用時に比べてスケジューリングのオーバヘッドは増加しないという前提から，適用時と非適用時でそのオーバヘッドに変化がないという前提で証明すれば十分である．

式 (2) と式 (3) から，任意のタプル p に対して $d_{p,o_1}<d_{p,o_2}<\cdots<d_{p,o_n}=D_{p,O}$ が成り立つ．そのため，本方式でスケジューリングすると，クラス B のストリームに含まれるタプルは，オペレータトレイン非適用時よりも，後で処理されるか変わらない．これにより，クラス A や C のストリームに含まれるタプルは，クラス B のタプルよりも早く処理されるか変わらない．ここで，先に処理されたタプルとそれを処理したオペレータのある m 個の組の集合を $S=\{(p'_i,o'_i);i=1,\cdots,m\}$ とする．$S=\emptyset$ ならば，定理は成り立つ．

そのため，以降では $S\neq\emptyset$ を仮定し，クラス B のストリームに含まれる任意のタプル p が処理されてクラス A のストリームに挿入されたときに，p を o_n で実行可能となる時刻がオペレータトレイン適用時に遅れないことを確認する．式 (3) から，オペレータトレイン適用時/非適用時にかかわらず，p を o_n で処理する優先度は変わらない．その結果，オペレータトレイン非適用時でも，p を o_n で処理するときには，$(p'_1,o'_1),\cdots,(p'_m,o'_m)$ の処理は先に終わっているはずである．このことから，p を o_n で処理する時刻がオペレータトレイン適用時でも遅れないことが分かる．

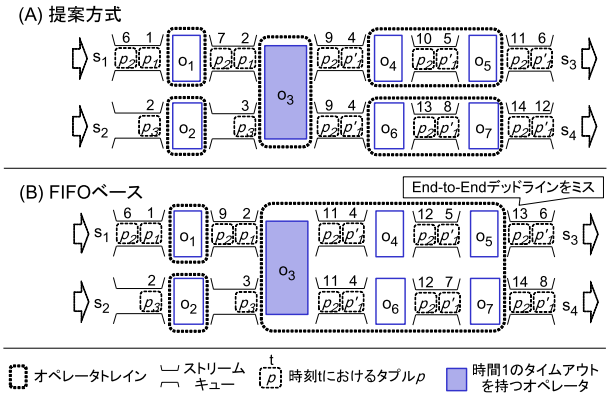

図 6　タイムアウトを含むスケジューリングの例
Fig. 6　Example of scheduling with timeout.

出力ストリームは，o_n の出力するストリームである可能性はあるが，$o_1,\cdots,o_{n-1}$ の出力するストリームではない．以上から，クラス B のストリームに含まれる任意のタプル p が，出力ストリームに挿入される時刻はオペレータトレイン適用時に遅れないことが示される．クラス A やクラス C のストリームに含まれるタプル $p'_1,\cdots,p'_m$ は，オペレータトレイン適用時に早く処理されることはあっても遅れて処理されることはないため，$p'_1,\cdots,p'_m$ も出力ストリームに挿入される時刻はオペレータトレイン適用時に遅れない．□

5.5　タイムアウトを含めたスケジューラの動作

本節では，タイムアウトを持つオペレータを含む場合における EDF スケジューラの動作を説明する．オペレータトレインを構成する表 1 の条件から，タイムアウトを持つオペレータは必ずオペレータトレインの先頭の要素となる．タイムアウトを持つオペレータが先頭となるオペレータトレインの実行は，タイムアウトが発生したときに入力されたタプルのみで処理を実行する．タイムアウトするオペレータの入力がすべて揃った場合は，タイマをただちに解除する．いずれの場合でも，EDF スケジューラから実行可能な状態にすることで，タイムアウトを持たないオペレータと同様に扱われる．本節の場合でもアルゴリズム 2 をそのまま適用できる．

以降では，**図 6** を例に用いて，タイムアウトを持つオペレータを含む場合のスケジューリング手順を述べる．図 6 の o_3 は時間 1 のタイムアウトを持つオペレータであり，その他のオペレータはタイムアウトを持たない．出力ストリーム s_3 と s_4 の End-to-End デッドラインはそれぞれ 5 と 11 とする．タプル p_1 と p_2 が時刻 1，6 に入力ストリーム s_1 に挿入され（$t_{p_1}=1$，$t_{p_2}=6$），タイムスタンプ 2 を持つタプル p_3 が時刻 2 に入力ストリーム s_2 に挿入される

($t_{p_3}=2$) 場合を考える．各オペレータの計算時間は単純にすべて 1 とし，タプル上の数字はそこにタプルが到着した時刻を表している．**図 7** は，図 6 の例を時間軸に沿って表しており，提案方式におけるオペレータトレインの実行順序に対して，FIFO ベースの従来方式と提案方式とを比較している．

図 6 (A) で，提案方式を説明する．

時刻 1：入力ストリーム s_1 に p_1 が挿入され，スケジューラは o_1 を実行し p_1 を処理する．$D_{p_1,(o_4,o_5)}=1+5=6$ と $D_{p_1,(o_6,o_7)}=1+11=12$ から $D_{p_1,(o_3)}=\min\{6-2,12-2\}=4$ となり $D_{p_1,(o_1)}=4-1=3$ となる．

時刻 2：(p_1,o_1) の実行が完了し，o_3 の入力に p_1 が到着することで，o_3 のタイムアウトの時刻を $2+1=3$ と設定する．また，その時刻に入力ストリーム s_2 に p_3 が挿入されるため，(p_3,o_2) を実行する．

時刻 3：(p_3,o_2) の実行が完了し，o_3 が入力する 2 つのタプルが到着したため，タイマを解除して，o_3 で $\{p_1,p_3\}$ を実行する．

時刻 4：o_3 は $\{p_1,p_3\}$ を処理することで p_1' を 2 つのストリームに挿入する．このように複数のタプルを結合する場合，結合されたタプルのタイムスタンプは，結合するタプルの中で最古（最小）のタイムスタンプである．$D_{p_1',(o_4,o_5)}=6$ と $D_{p_1',(o_6,o_7)}=12$ なので $(p_1',(o_4,o_5))$ が先に実行される．

時刻 5：$(p_1',(o_4,o_5))$ を実行し (p_1',o_4) の実行まで完了する．

時刻 6：$(p_1',(o_4,o_5))$ の実行が完了し，s_1 に p_2 が挿入される．$D_{p_2,(o_1)}=8$ で $D_{p_1',(o_6,o_7)}=12$ なので，待たされていた $(p_1',(o_6,o_7))$ よりも $(p_2,(o_1))$ が先に実行される．

時刻 7：p_2 が o_3 の入力に到着し，o_3 のタイムアウトの時刻を $7+1=8$ と設定する．また，待たされていた $(p_1',(o_6,o_7))$ が実行される．

時刻 8：o_6 から p_1' が出力される．また，o_3 のタイムアウトが発生し，$(p_2,(o_3))$ がスケジューリング可能となる．$D_{p_2,(o_3)}=9$ と $D_{p_1',(o_6,o_7)}=12$ なので，実行中の $(p_1',(o_6,o_7))$ をプリエンプションして $(p_2,(o_3))$ を先に実行する．

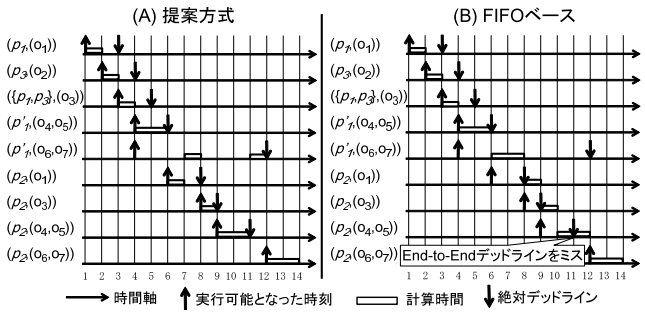

図 7　時間軸に沿ったスケジューリングの比較
Fig. 7　Comparison of scheduling along the time axis.

時刻 9–10：$D_{p_2,(o_4,o_5)}=11$ なので $(p_1',(o_6,o_7))$ を続けてプリエンプションして新たに実行可能となった $(p_2,(o_4,o_5))$ を先に実行する．

時刻 11–14：待たされていた $(p_1',(o_6,o_7))$ と $(p_2,(o_6,o_7))$ を実行する．

以上のようにスケジューリングすることで，図 6 (A) や図 7 (A) のようにすべてのタプルがデッドラインをミスせずに処理される．一方，従来の FIFO ベースのスケジューリング方式では，図 6 (B) や図 7 (B) のように，時刻 6 で先に入力されて処理されていた絶対デッドラインの遅い $(p_1',(o_6,o_7))$ を先に実行してしまう．その結果，絶対デッドラインの早い $(p_2,(o_1))$ や $(p_2,(o_3))$ しいては $(p_2,(o_4,o_5))$ の処理が間に合わず，p_2 は出力ストリーム o_1 の End-to-End デッドラインをミスしてしまう．このように，提案方式ではタイムアウトを含むクエリに対してもリアルタイム制約を維持するようにスケジューリングすることが可能となる．

6. 評価

本章では，データ処理の実行時における性能を評価する．まず簡単なクエリを用いて基本的な性質を確認し，次に自動車の具体的なアプリケーションを用いて本方式の有効性を確認する．評価マシンには，CPU：800 MHz，メモリ：1024 MB，OS：Linux (Fedora10) 32 bit を用いた．

本研究では 2 章の I4 のケースを扱うため，入力データ量を変化させたときの式 (4) のデッドラインミス率（DMR）を用いて，リアルタイム制約の性能を測る．

$$\frac{1}{\sum_{s\in\{s_1,\cdots,s_n\}}\alpha_s}\sum_{s\in\{s_1,\cdots,s_n\}}\alpha_s\frac{M_s}{N_s} \tag{4}$$

なお，$s_1,\cdots,s_n$ はクエリの出力ストリームであり，N_s は出力ストリーム s に挿入されたタプル数で，M_s はその中でデッドラインをミスしたタプル数である．α_s は 0 以上の実数で各出力ストリーム s におけるリアルタイム制約の維持に対する重要度を表す．

6.1 基本性能評価

本節では，異なる End-to-End デッドラインを含む**図 8** のマルチクエリを用いる．出力ストリーム 1 には 5 ミリ秒と短いデッドラインを指定し，出力ストリーム 2 には十分長い 500 ミリ秒を指定する．式 (4) における出力ストリーム 1 と 2 の重要度 α_s は 1 とする．各オペレータは，計算時間が平均 100 マイクロ秒であり，選択率は 1.0 である．以降の各実験では，少なくとも 100 回の試行を繰り返した

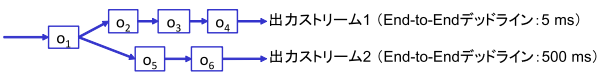

図 8　基本性能評価に用いるクエリ
Fig. 8　Query used in the basic performance evaluation.

測定結果を用いる．

スケジューリング方式による違いを確認するため，スケジューリングする対象が複数存在する状況が発生するように入力データ量を変動させて評価する．入力データ量の変動するパターンとしては，短期間でデータ量が急激に増加する場合（input1）と，長期間でデータ量が徐々に増加する場合（input2）で実験する．input1 では，ある時刻とその 500 マイクロ秒後に，指定した数のタプルを入力ストリームに一度に挿入する．input2 では，指定した数のタプルを，400 マイクロ秒間隔で 1 タプルずつ入力ストリームに挿入する．いずれの入力パターンにおいても，入力データ量（つまり，指定されたタプル数）を変化させて性能を測定する．

6.1.1　オペレータトレイン

図 9 と**図 10** では，入力データのパターン input1 と input2 において，アルゴリズム 3 によりオペレータトレインを用いた場合（Operator train）と用いない場合（Primitive）とで，入力データ量を変化させて DMR を比較した．図 9 のダッシュ記号の付いた方式ではタプルをまったくバッチ化せず，付いていない方式では同時刻に入力ストリームに一度に挿入されるタプルをバッチ化している．図 8 のクエリにおけるオペレータトレインは，(o_1)，(o_2, o_3, o_4)，(o_5, o_6) である．

図 9 と図 10 から，オペレータトレインにより DMR が削減されることが分かる．これは，オペレータトレインにより，スケジューリングのオーバヘッドが削減されたためである．また，図 9 からタプルをバッチ化することでオペレータトレインの性能向上の効果が減少している．これは，複数のタプルをバッチ化して 1 つのタプルのようにスケジューリングすることで，スケジューリングにかかるオーバヘッドが減少したためである．以降では，Operator train を提案方式として用い，いずれの方式に対しても本節と同様の方法でタプルをバッチ化する．

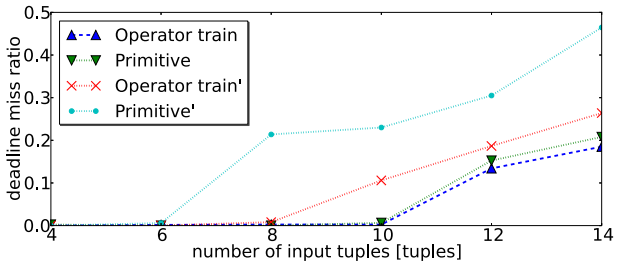

図 9　input1 におけるオペレータトレインの効果
Fig. 9　Effect on operator train in input1.

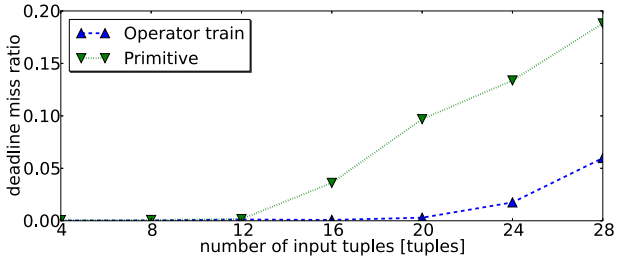

図 10　input2 におけるオペレータトレインの効果
Fig. 10　Effect on operator train in input2.

プリエンプションは，あるオペレータトレインの処理途中で，他のオペレータトレインが処理される場合に発生する．input1 と input2 のケースにおいて，input2 で入力されるタプル数が 28 個のとき，スケジューリングする対象が最も増えるため，プリエンプションが最も多く発生する．このとき，プリエンプションを行った場合のスケジューリングのオーバヘッドは平均 2.6 マイクロ秒であり，全出力ストリームの遅延時間は平均 7.8 ミリ秒であった．この結果から，プリエンプションのオーバヘッドはこのケースでは無視できる程度に小さいことが分かった．

6.1.2　リアルタイムスケジューリング

本項では，目的の異なるスケジューリング方式に対して，リアルタイム制約への有効性を比較評価する．提案方式（S-EDF）と比較する従来方式を以下で述べる．

MC+ は，文献 [14] と同様に，スループットの向上を目的とする従来方式である．MC+では，クエリグラフでトポロジカルソートを行った順番でオペレータをスケジューリングする．新しいタプルが入力ストリームに挿入されるたびに先頭から実行し直し，オペレータの呼び出し回数を削減する．特に本論文では，オペレータから出力するストリームが複数に分岐する場合には，End-to-End デッドラインが短い出力ストリームにつながるパスから順に実行する．図 8 のクエリでは，静的に決定されたオペレータの実行順は $o_1, o_2, \cdots, o_6$ である．

PCS は，文献 [12], [13] と同様に，平均遅延時間を削減することを目的とし，マルチクエリにも対応する従来方式である．静的に計算されるプロセッシングキャパシティの大きいオペレータパスから順に実行する．現在実行中のオペレータパスよりもプロセッシングキャパシティの大きいオペレータパスが実行可能となれば，実行を切り替える．オペレータの出力が複数に分岐する場合には，分岐した各パスをボトムアップにスケジューリングする [12]．図 8 のクエリでは，o_1，o_2，o_5，o_3，o_6，o_4 となる．

FIFO+ は，平均遅延時間を削減する FIFO [12], [18] と同様の方式である．入力ストリームに挿入されたタプルから順に処理するようにオペレータをノンプリエンプティブにスケジューリングする．特に本論文では，オペレータの出力が複数に分岐する場合には，End-to-End デッドラインが短い出力ストリームにつながるパスから順に実行する．図 8 のクエリでは，$o_1, o_2, \cdots, o_6$ となる．

図 11 と**図 12** は，入力パターン input1 と input2 において，各方式のリアルタイム制約の維持における性能を比較した結果である．いずれの入力パターンにおいても，提案方式である S-EDF が従来方式である MC+，PCS，FIFO+

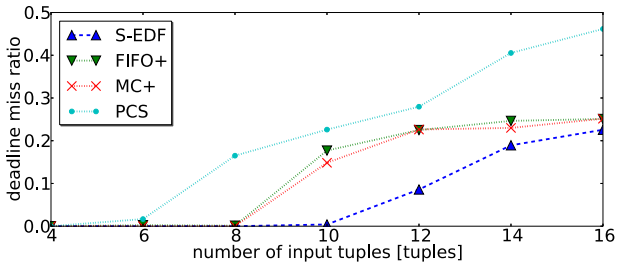

図 11　input1 における従来方式との比較
Fig. 11　Comparison with the existing methods in input1.

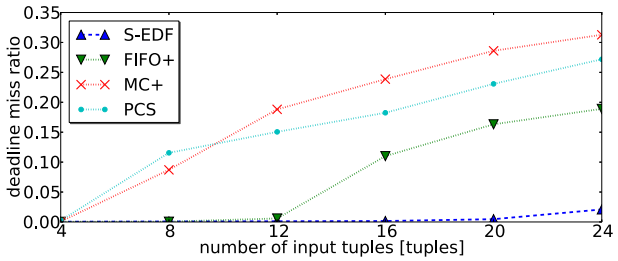

図 12　input2 における従来方式との比較
Fig. 12　Comparison with the existing methods in input2.

よりも良好な結果を示した．これは，EDF に基づき，プリエンプティブなリアルタイムスケジューリングを行った結果である．

平均遅延時間の削減を目的とする従来方式である FIFO+と PCS を比較すると，FIFO+の方が性能が良い．これは，オペレータの出力が分岐するときに，FIFO+では End-to-End の短い o_2，o_3，o_4 を o_5，o_6 よりも先に実行するようにスケジューリングするため，出力ストリーム 1 の End-to-End デッドラインをミスしにくくなったためである．平均遅延時間の削減が目的の FIFO+と，スループットの向上が目的の MC+とでは，input1 ではほぼ同様の性能だが，input2 では MC+の性能が悪い．これは，input2 ではストリームキューに徐々に溜まった古いタプルが MC+では優先して処理されず，それらのタプルが出力ストリーム 1 に到着したときには絶対デッドラインをミスしてしまうためである．以降では従来方式として，いずれの入力パターンでも性能が良好な FIFO+を用いる．

6.2　アプリケーション性能評価

本節では，2 章で述べた自動車の衝突警告を含む具体的なアプリケーションを用いて，本方式の有効性を確認する．

6.2.1　評価に用いるストリーム処理

図 13 は，本評価で用いるマルチクエリであり，o_1–o_{11} はそれを構成するオペレータである．アルゴリズム 3 により構成されるオペレータトレインは**表 2** のようになる．d_j，c_j（$j = 1, \cdots, 7$）は，あるタプル p がクエリに入力されたときの各オペレータトレインの絶対デッドラインと計算時間である．

出力ストリームとしては，自車両の GPS や車速センサ，

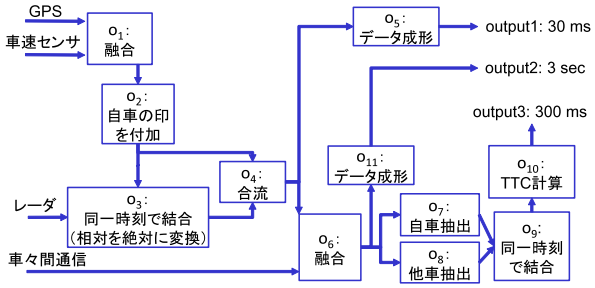

図 13　アプリケーション評価で用いるストリーム処理
Fig. 13　Stream processing for the application evaluation.

表 2　本評価におけるオペレータトレイン
Table 2　Operator train for the evaluation.

オペレータトレイン	オペレータの列	絶対デッドライン (ms)
O_1	o_1, o_2	$d_1 = \min\{d_2 - c_2, d_3 - c_3\}$
O_2	o_3, o_4	$d_2 = \min\{d_4 - c_4, d_5 - c_5\}$
O_3	o_4	$d_3 = \min\{d_4 - c_4, d_5 - c_5\}$
O_4	o_5	$d_4 = 30 + t_p$
O_5	o_6	$d_5 = \min\{d_6 - c_6, d_7 - c_7, d_8 - c_8\}$
O_6	o_7, o_9, o_{10}	$d_6 = 300 + t_p$
O_7	o_8, o_9, o_{10}	$d_7 = 300 + t_p$
O_8	o_{11}	$d_8 = 3000 + t_p$

レーダでセンシングしたデータから得られる車両情報を配信する output1 と，その情報を車々間通信から得られる車両情報と融合した結果を配信する output2 と，自車と周辺車両との衝突警告を行うための情報を配信する output3 がある．output1 は車々間通信可能な周辺車両における衝突警告などのアプリケーションの入力や自車の車両制御に用いられることを想定し，その End-to-End デッドラインは最も短く 30 ミリ秒を設定した．output2 は情報提供やログ出力などリアルタイム制約が求められないアプリケーションを対象として十分に長い 3 秒を設定した．output3 は ETSI の仕様 [6] に合わせて 300 ミリ秒とした．

GPS や車速センサから得られるタプルは自車の位置や速度を含み，レーダから得られるタプルは相対位置や相対速度を含む．車々間通信から得られるタプルは，自車両が output1 からブロードキャストしたデータに相当し，自車と周辺車両の位置や速度が含まれる．この車々間通信からの入力ストリームには load shedder を設定する[*5]．

o_1 は，10 ミリ秒のタイムアウトを持ち，文献 [28] と同様の方法で，自車両における位置と速度を用いたカルマンフィルタが実装されている[*6]．o_2 は，o_1 から出力された自車両の位置/速度からなるタプルに，自車の搭載センサで生成された印を付けた結果を o_3 と o_4 に配信する．o_3 は，レーダから得られた周辺車両の相対位置/相対速度を自車

[*5] 本実験ではデータのフィルタリングはランダムに行い，その入力データ量の最大値は 6.2.3 項で決める．
[*6] 位置や速度を含むタプルには，それらの分散も含まれる．

の位置/速度と加算し，絶対位置/絶対速度に変換する．o_4 は，o_2 と o_3 の出力結果を単純に合流し，o_5 や o_6 の入力にそれを流す．o_5 は，output1 を利用するアプリケーション用にデータを整形する．o_6 は，100 ミリ秒のタイムアウトを持ち，文献 [29] と同様の方法で，搭載センサと車々間通信から得られる車両の位置情報をセンサフュージョンで融合するよう実装されている．o_6 の出力結果は，o_{11} で表示用に成形されて ouput2 に出力される．o_7 と o_8 も，o_6 の出力結果を入力し，自車と周辺車のタプルをそれぞれ抽出する．o_9 は，同一のタイムスタンプを持つ自車と周辺車の情報を結合する．o_{10} は，o_9 の結合結果から，自車と交差する周辺車両に対して Time To Collision（TTC）を計算し，その結果を output3 に出力する．

6.2.2　評価方法

車外との通信を用いる自動車のアプリケーションでは，車々間通信から入力されるデータ量が多く，その変動も大きい．本評価では，ETSI の要求 [6] である 1 秒あたり最大 1000 台分の車両情報を車々間通信から受信するケースを扱う．そのため，多数の車両の車々間通信の模擬や，電波の特性や強度，車両のモビリティなども模擬できるネットワークシミュレータ Scenargie[*7]を用いて入力データを作成した．各車両は，交差点間の距離が 100 m の碁盤目の道路（マンハッタンモデル）に**図 14** のように初期配置し，矢印の方向に時速 60 km で直進した．**図 15** は，車々間通信からの入力レートの推移を表しており，図 14 の円でマークした 2 つの交差点付近で増加し 1 秒あたり約 1000 台分の車両情報がタプルとして入力される．

車々間通信やレーダは，100 ミリ秒周期で 200 m 程度の範囲まで到達するように，電波強度を設定した．車々間通信は 700 MHz とし，障害物があってもある程度屈折して電波が届く．レーダは 70 GHz 以上の高周波を設定し，障害物でほとんど屈折しない．6.1.2 項の評価から，ここでの比較対象には従来方式の中で最も性能が良好な FIFO+を用いる．また，図 6 (B) のように，FIFO+はタイムアウトを持つオペレータを含む場合にも適用できる．

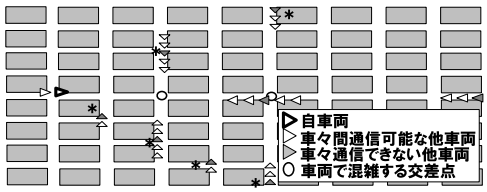

図 14　車両の初期位置
Fig. 14　Initial positions of vehicles.

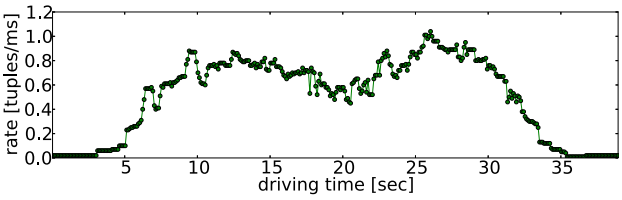

図 15　車々間通信からの入力レートの推移
Fig. 15　Input rate from V2V communications.

[*7] Scenargie: https://www.spacetime-eng.com/

6.2.3　リアルタイム制約

本アプリケーションシナリオにおいて，提案方式の直接的な効果を確認する．**図 16** は，各方式において End-to-End デッドラインが最も短い output1 における最大遅延時間の推移を表している．この遅延時間は，センサからデータが発生してから，output1 にそのタプルが挿入されるまでの時間として測定した．エラーバーは同一の走行シナリオで 100 回試行した場合の標準偏差を表し，グラフはその平均を表す．提案方式により，従来方式の FIFO+と比べて output1 の最大遅延時間を平均 65%と大きく削減できた．これは，提案方式では，GPS や車速センサ，レーダから入力されたタプルの絶対デッドラインが早ければ，車々間通信から大量に入力されるタプルよりもそれらを優先して処理するためである．一方，FIFO+や他の従来方式では，絶対デッドラインの早いタプルが別にあっても，計算時間の大きい車々間通信からの入力データを処理してしまうことがあるため，このように最大遅延時間に大きな差が生じる．

図 17 は，load shedder により車々間通信から 1 秒あたりの最大入力データ量を推移させたときの DMR の変化を表している．ただし，式 (4) における重要度 α_s は，車両制御や衝突警告に用いる output1 や output3 に対してはそれぞれ 1 とし，情報提供に用いる output2 については 0 とした．図 17 のように，提案方式ではデッドラインミスを

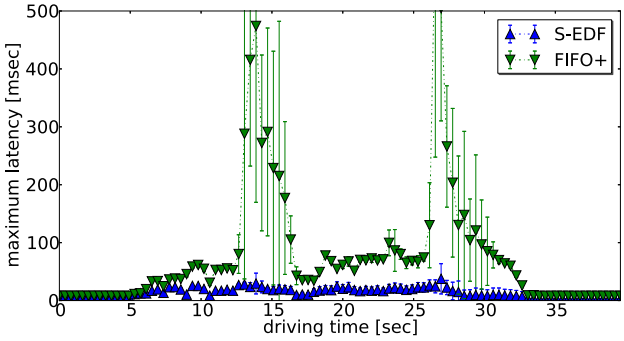

図 16　走行時間における最大遅延時間の変化
Fig. 16　Changes in maximum latency with driving time.

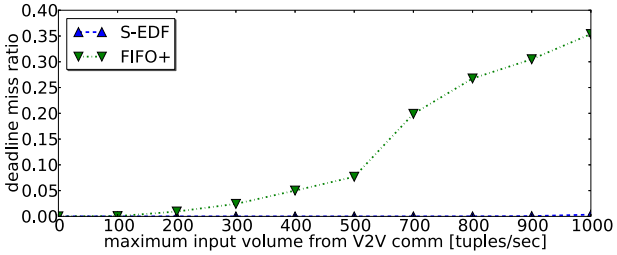

図 17　本アプリケーションシナリオにおけるデッドラインミス率
Fig. 17　Deadline miss ratio of the application scenario.

大きく削減できる．EDF では，リアルタイム制約を満たして 1 秒あたり最大 800 タプルの入力データを処理できるが，FIFO では最大 100 タプルしか処理できない．これにより，提案方式では，リアルタイム制約を満たしながらより多くの車々間通信からの入力データを処理できる．これは，End-to-End デッドラインの短い output1 や output3 の最大遅延時間の削減による．以降では，車々間通信からの 1 秒あたりの最大入力データ量を S-EDF と FIFO+とでそれぞれ 800 個と 100 個とする．

6.2.4　車両衝突警告への影響

本シナリオにおける衝突警告への提案方式の有効性を確認するため，output3 から配信される周辺車両と自車が衝突するまでの時間である Time To Collision（TTC）を用いて車両衝突事故への影響を確認する．自車と衝突する周辺車両は，図 14 の＊印の付いた車々間通信できない 6 台の車両（車両＊）である．ここでは，時速 60 km で乾いた路面を走行する場合の停止距離から [30]，自車両の停止時間を 2.8 秒と算出する．本節では，車両＊を最初に検出したとき，TTC が停止時間の 2.8 秒よりも短ければ衝突し，長ければ衝突を回避できるとする．つまり，車両＊との衝突回避の閾値は，その車両＊を最初に検出したときの TTC が 2.8 秒の時点である．ここで，停止しない場合に衝突する車両の台数 N のうち，衝突を回避できなかった台数 M の割合として，出会い頭衝突の事故率を式 (5) で定義する．

$$M/N \tag{5}$$

本実験では，同一の走行シナリオで 100 回試行した場合で事故率を測定した．つまり，$N = 6 \times 100$ であり，試行 t で衝突する車両＊の台数を M_t とすると $M = \sum_{t=1,\cdots,100} M_t$ である．

図 18 は，ある試行において TTC を変化させたとき，6 台の車両＊の内検出できた台数の割合（検出率）を表している．横軸は車両＊との TTC で，縦軸がその TTC における検出率を表しており，衝突回避の閾値に線を引いた．NoV2V は車々間通信をまったく利用しない場合である．NoV2V では，事故率は 100%であり衝突を回避できなかったが，すべての車両＊を衝突前には検知できているのが分かる．これは，車々間通信を利用できない場合でも，搭載センサのみを用いたストリーム処理がリアルタイム制約を満たして動作しているためである．図 18 から，FIFO+では一時的に車両＊を検出できないときがあり，S-EDF の方が検出率が良いことが分かる．これは，S-EDF では FIFO+に比べて車々間通信からの最大入力量が 1 秒あたり 10 台から 80 台に増えたためである．

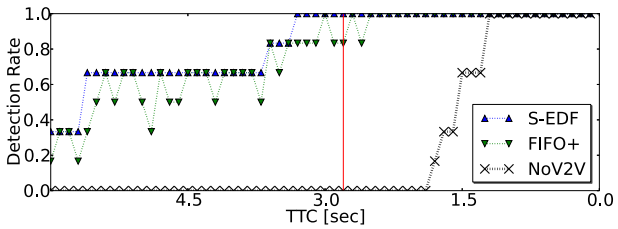

図 18　TTC に対する検出率の変化
Fig. 18　Changes in detection rate with TTC.

次に，図 18 のような試行を 100 回試行した測定結果を述べる．車両＊の検出が最も遅れたときの TTC（つまり TTC の最悪値）は，S-EDF，FIFO+，NoV2V で，それぞれ 3.2 秒，2.4 秒，1.2 秒であった．また，衝突した車両＊の台数である式 (5) の M は，S-EDF，FIFO+，NoV2V で，それぞれ 0 台，50 台，600 台であった．そのため，式 (5) を適用すると，事故率は S-EDF，FIFO+，NoV2V で 0%，8.3%，100%となる．これにより，本シナリオにおいて，提案方式が車両の衝突事故を削減できることを示した．

7.　考察

本章では，2 章であげた課題の対応項目 I1–5 に対して，本研究でどのように解決したかをまとめる．

I1：リアルタイムスケジューリングの導入
5 章の提案方式により本項目に対応した．6 章の評価では，リアルタイムスケジューリングを行わないストリーム処理における従来方式である FIFO+，MC+，PCS と比較することで，リアルタイム制約の維持における提案方式の有効性をデッドラインミス率の削減から確認した．また，衝突警告アプリケーションにおいて提案方式により車両衝突事故を削減できることも確認した．

I2：異なるデッドラインを持つマルチクエリへの対応
5 章の提案方式は，各出力ストリームに異なる End-to-End デッドラインを指定することが可能であり，マルチクエリを想定した方式であることから，本項目に対応している．また，評価で用いたクエリ（図 8 と図 13）は，いずれも異なる End-to-End デッドラインを持つマルチクエリであるため，6 章の評価で本項目における提案方式の有効性を確認できた．

I3：タイムアウトを持つオペレータへの対応
5.5 節の方法で，タイムアウトを持つオペレータに対応した．また，図 13 の車両衝突警告を行うクエリはタイムアウトを持つオペレータ（o_1 と o_6）を含み，6.2 節ではそれを用いて提案方式の有効性を評価している．

I4：オーバロード時への対応
5.1 節で述べたように，load shedder を特定の入力ストリームに設定して対応する．6.2 節の評価では，車々間通信からの入力データ量が ETSI の仕様 [6] で最大となるケースをシミュレーションし，リアルタイムスケジューリングを

用いながら load shedding を行うことで，そのシミュレーション環境における提案方式の有効性を示した．

I5：スケジューリングにおけるオーバヘッドの削減
5.4 節で述べたオペレータトレインの構成方法により，リアルタイム制約の維持を保証してオーバヘッドを削減する方法を示した．また，5.3 節では，それをプリエンプティブにスケジューリング方法を示した．6.1 節の評価では，オペレータトレインを行うことでリアルタイム制約における性能改善を確認した．

8. おわりに

自動車の安全運転支援におけるセンサ情報処理で重要なリアルタイム制約を維持するため，本論文では，ストリーム処理における EDF ベースのリアルタイムスケジューリング方式を提案した．本研究では，車々間通信など車外からのデータを活用する場合を対象とし，提案方式は，それらのセンサ情報処理で必要となる，各出力ストリームに様々な End-to-End デッドラインを持ち，タイムアウトを持つオペレータを含むマルチクエリに適用可能である．本評価では，ストリーム処理における目的の異なる従来のスケジューリング方式と比較しリアルタイム制約の維持における性能向上を確認した．また，車々間通信を用いた衝突警告を含むセンサ情報処理をストリーム処理で実現し，ETSI の仕様に合わせて車々間通信から受信するデータ量を最大に近づけてシミュレーションにより評価した．その結果，本走行シナリオにおいて，提案方式が，リアルタイム制約の維持に良好な FIFO ベースの従来方式である FIFO+と比較して，最大遅延時間やデッドラインミス率を削減し，リアルタイム制約を維持して車々間通信から入力データをより多く処理することで，特定のシミュレーション環境のもと，車両の衝突事故を削減することを確認した．今後の課題としては，効率的にデータ処理を行うタプルのバッチ化の構成方法，マルチコアやメニーコアへの対応，実際の車載システムにおける評価があげられる．

謝辞　本研究の一部は総務省戦略的情報通信研究開発推進事業 SCOPE（121806015）と科研費（25240007）の助成を受けたものである．

参考文献

[1] Guizzo, E.: Toyota's Semi-Autonomous Car Will Keep You Safe, IEEE Spectrum (online), available from ⟨http://spectrum.ieee.org/automaton/robotics/artificial-intelligence/toyota-semi-autonomous-lexus-car-will-keep-you-safe⟩ (accessed 2014-11-21).
[2] 藤田浩一，宇佐美祐之，山田幸則，所　節夫：衝突危険性のセンシング技術，自動車技術，Vol.61, No.2, pp.62–67 (2007).
[3] Erico, G.: How Google's Self-Driving Car Works, IEEE Spectrum (online), available from ⟨http://spectrum.ieee.org/automaton/robotics/artificial-intelligence/how-google-self-driving-car-works⟩ (accessed 2014-11-21).
[4] Buehler, M., Iagnemma, K. and Singh, S.: *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic*, 1st edition, Springer (2009).
[5] 国土交通省自動車交通局先進安全自動車推進検討会：先進安全自動車（ASV）推進計画報告書 (2011).
[6] ETSI: Intelligent Transport Systems; V2X Applications; Part 3: Longitudinal Collision Risk Warning application requirements specification (2013).
[7] ETSI: Intelligent Transport Systems; Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service (2011).
[8] 山田真大，鎌田浩典，佐藤健哉：データストリーム管理機構を利用した車載データ統合モデルの提案と評価，自動車技術会論文集，Vol.41, No.2, pp.419–424 (2010).
[9] 山口晃広，本田晋也，佐藤健哉，高田広章：車載システム向けデータストリーム管理システムにおけるクエリ自動構築手法，情報科学技術フォーラム講演論文集，Vol.11, No.2, pp.7–14 (2012).
[10] 佐藤健哉：自動車走行環境認識のためのセンサデータ処理機構，データ工学研究会信学技報，Vol.110, No.107, pp.51–56 (2010).
[11] 勝沼　聡，山口晃広，熊谷康太，本田晋也，佐藤健哉，高田広章：車載組込みシステム向けデータストリーム管理システムの開発，電子情報通信学会論文誌．D，情報・システム，Vol.95, No.12, pp.2031–2047 (2012).
[12] Chakravarthy, S. and Jiang, Q.: *Stream Data Processing: A Quality of Service Perspective Modeling, Scheduling, Load Shedding, and Complex Event Processing*, 1st edition, Springer (2009).
[13] Jiang, Q. and Chakravarthy, S.: Scheduling Strategies for Processing Continuous Queries over Streams., *BNCOD*, Lecture Notes in Computer Science, Vol.3112, pp.16–30, Springer (2004).
[14] Carney, D., Çetintemel, U., Rasin, A., Zdonik, S., Cherniack, M. and Stonebraker, M.: Operator Scheduling in a Data Stream Manager, *Proc. 29th International Conference on Very Large Data Bases*, pp.838–849 (2003).
[15] Althoff, M., Althoff, D., Wollherr, D. and Buss, M.: Safety verification of autonomous vehicles for coordinated evasive maneuvers, *Intelligent Vehicles Symposium*, pp.1078–1083 (2010).
[16] Tian, Q., Guo, T., Qiao, S., Wei, Y. and wei Fei, W.: Design of Intelligent Parking Management System Based on License Plate Recognition, *Journal of Multimedia*, Vol.9, No.6, pp.774–780 (2014).
[17] Babcock, B., Babu, S., Motwani, R. and Datar, M.: Chain: Operator scheduling for memory minimization in data stream systems, *Proc. 2003 ACM SIGMOD international conference on Management of data*, pp.253–264 (2003).
[18] Babcock, B., Babu, S., Datar, M., Motwani, R. and Thomas, D.: Operator Scheduling in Data Stream Systems, *The VLDB Journal*, Vol.13, No.4, pp.333–353 (2004).
[19] Kulkarni, D., Ravishankar, C.V. and Cherniack, M.: Real-time, Load-adaptive Processing of Continuous Queries over Data Streams, *Proc. 2nd International Conference on Distributed Event-based Systems*, ACM, pp.277–288 (2008).
[20] Zhou, Y., Wu, J. and Leghari, A.K.: Multi-query Scheduling for Time-critical Data Stream Applications,

Proc. 25th International Conference on Scientific and Statistical Database Management, pp.15:1–15:12 (2013).
[21] Schweppe, H., Member, A.Z. and Grill, D.: Flexible On-Board Stream Processing for Automotive Sensor Data, *IEEE Trans. Industrial Informatics*, Vol.6, No.1, pp.81–92 (2010).
[22] Bolles, A., Appelrath, H., Geesen, D., Grawunder, M., Hannibal, M., Jacobi, J., Koster, F. and Nicklas, D.: StreamCars: A new flexible architecture for driver assistance systems, *Intelligent Vehicles Symposium*, pp.252–257 (2012).
[23] Abadi, D.J., Ahmad, Y., Balazinska, M., Hwang, J.-H., Lindner, W., Maskey, A.S., Rasin, A., Ryvkina, E., Tatbul, N., Xing, Y. and Zdonik, S.: Aurora: A new model and architecture for data stream management, *The VLDB Journal*, Vol.12, No.2, pp.120–139 (2003).
[24] Abadi, D.J., Ahmad, Y., Balazinska, M., Cherniack, M., Hyon Hwang, J., Lindner, W., Maskey, A.S., Rasin, E., Ryvkina, E., Tatbul, N., Xing, Y. and Zdonik, S.: The Design of the Borealis Stream Processing Engine, *Proc. 2nd Biennial Conference on Innovative Data Systems Research*, pp.277–289 (2005).
[25] Gedik, B., Andrade, H., Wu, K.-L., Yu, P.S. and Doo, M.: SPADE: The System S Declarative Stream Processing Engine, *Proc. 2008 ACM SIGMOD International Conference on Management of Data*, pp.1123–1134 (2008).
[26] Chetto, H., Silly, M. and Bouchentouf, T.: Dynamic Scheduling of Real-time Tasks Under Precedence Constraints, *Real-Time Syst.*, Vol.2, No.3, pp.181–194 (1990).
[27] Buttazzo, G.C.: *Hard Real-time Computing Systems: Predictable Scheduling Algorithms And Applications* (*Real-Time Systems Series*), Springer (2004).
[28] Ammoun, S. and Nashashibi, F.: Real time trajectory prediction for collision risk estimation between vehicles, *Proc. IEEE International Conference on Intelligent Computer Communication and Processing*, pp.417–422 (2009).
[29] Smith, R.C. and Cheeseman, P.: On the Representation and Estimation of Spatial Uncertainly, *Int. J. Rob. Res.*, Vol.5, No.4, pp.56–68 (1986).
[30] South Australia: *The Driver's Handbook*, Department for Transport, Energy and Infrastructure (2005).

山口 晃広　（正会員）

2006 年神戸大学大学院自然科学研究科数学専攻修士課程修了．同年（株）東芝に入社．2011～2015 年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研究員．同年（株）東芝に復職．データストリーム処理の研究に従事．

渡辺 陽介　（正会員）

2006 年筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻修了．博士（工学）．2014 年より名古屋大学未来社会創造機構特任准教授として，情報統合，データストリーム処理等の研究に従事．電子情報通信学会，ACM，日本データベース学会各会員．

佐藤 健哉　（正会員）

同志社大学大学院理工学研究科情報工学専攻教授．1986 年大阪大学大学院工学研究科電子工学専攻修士課程修了．同年住友電気工業情報電子研究所入社．1991～1994 年スタンフォード大学計算機科学科客員研究員．2000 年奈良先端科学技術大学院大学情報科学研究科博士後期課程修了．米国 AMI-C, Inc. チーフテクノロジストを経て，2004 年より現職．同志社大学モビリティ研究センター長，および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特任教授兼務．博士（工学）．IEEE-CS，ACM，自動車技術会各会員．

中本 幸一　（正会員）

1982 年大阪大学大学院基礎工学研究科前期課程修了．同年 NEC 入社．1990～1991 年 Cornell 大学計算機科学科客員研究員．1997 年大阪大学大学院基礎工学研究科後期課程入学．2000 年単位取得退学．博士（工学）．2004 年より現職．2006 年より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特任教授，組込みソフトウェア，分散システム，ソフトウェア開発環境に興味を持つ．電子情報通信学会，日本ソフトウェア科学会，システム制御情報学会，IEEE Coomputer Society，ACM，USENINX 各会員．

高田 広章　（正会員）

名古屋大学未来社会創造機構教授．同大学大学院情報科学研究科教授・附属組込みシステム研究センター長を兼務．1988 年東京大学大学院理学系研究科情報科学専攻修士課程修了．同専攻助手，豊橋技術科学大学情報工学系助教授等を経て，2003 年より名古屋大学大学院情報科学研究科情報システム学専攻教授．2014 年より現職．リアルタイム OS，リアルタイムスケジューリング理論，組込みシステム開発技術等の研究に従事．オープンソースのリアルタイム OS 等を開発する TOPPERS プロジェクトを主宰．博士（理学）．IEEE，ACM，情報処理学会，電子情報通信学会，日本ソフトウェア科学会，自動車技術会各会員．

（担当編集委員　浅井 達哉）