車載システム向け
リアルタイム分散ストリーム処理

山口 晃広

車載システム向け
リアルタイム分散ストリーム処理

要旨

近年，自動車の機能として自動運転や車両衝突回避などの安全運転支援に注目が集まっている．このような安全運転支援システムは，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載システムに搭載し，複数のセンサデータを組み合わせて周辺環境を認識し，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．そのため，車載システムのデータ処理は複雑化している．

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処理が求められる．このリアルタイム処理要求を満たせないと，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を防ぐことができない．

一方，安全運転支援のセンサ情報処理では，車両や歩行者の追跡が行われる．しかし，センサデータには誤差がのるため，GPS や複数のセンサを統合して誤差を低減することで信頼性を上げるセンサデータフュージョンが必要がある．車載システムに搭載される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．特に車々間通信では，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，車々間通信から受信するデータの量は走行状況により増大する．このような状況では，リアルタイム処理要求を満たしながら，多くのデータを信頼性をあげて処理することが重要な課題となる．また，車々間通信から受信するデータでは，到着タイミングが通信遅延を無視しても 100 ミリ秒遅れる可能性があり，センサからデータを読み込む時刻と車々間通信を経由して車載システムにデータが到着する時刻には差が生じる．そのため，このような out-of-order な入力データへの対応も課題となる．

一方，車載システムは ECU（electronic control unit）と呼ばれる複数のノードが車載ネットワークで繋がれた分散システムであり，各ノードに様々な複数のセン

サが搭載されている．車載ネットワークはバスとゲートウェイで繋がれた複雑なネットワークである．また，それらの構成は車種によって異なる．そのため，自動車の製造メーカーではソフトウェアの開発コストを抑えることが課題となっており，ソフトウェアプラットフォームが必要である．

データの管理や処理を効率的に行うプラットフォームとしてはデータ管理システムの導入が有効である．特に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．データストリーム処理システムでは，ユーザがデータ処理を容易に記述できるクエリを提供する．これにより，センサやアプリケーションの追加や変更に対応しやすくなる．特に，車載システムではそれを構成する各ノードにセンサが分散して搭載されているため，そのデータ処理は複数のノードに分散する．そのため，分散ストリーム処理システムが必要となる．

しかしながら，従来のストリーム処理を直接用いて車載システムのデータ処理要求を満たすことは難しい．リアルタイムシステムの分野ではリアルタイムスケジューリングがリアルタイム処理要求を満たすのに有効であることが知られている．一方，ストリーム処理の分野においてもスケジューリングの研究は良く行われているが，平均時間やメモリ使用量の削減やスループットの向上を目的としており，リアルタイム処理要求を満たすことを目的としていない．そのため，ストリーム処理における従来のスケジューリング方式ではリアルタイム処理要求を満たすことは難しい．また，分散ストリーム処理ではオペレータと呼ばれる演算単位を各ノードに割り当てるオペレータの配置方式が良く研究されている．しかしながら，それらの従来方式はオーバーレイネットワークを想定しており，バスやゲートウェイで繋がれた車載ネットワークへ適用することは難しい．加えて，従来のストリーム処理では out-of-order な入力データを想定しておらず，全てのストリームキューを走査しようとするとオーバヘッドが増大する問題がある．このように，車載システムのデータ処理要求に合った分散ストリーム処理の技術開発が必要となる．

本研究の具体的な内容は以下の 3 つである．

1 つ目の研究として，車載システムのデータ処理要求からデータストリーム処理システムに必要な技術課題を整理し，それらを解決する分散ストリーム処理システム（Automotive Embedded Data Stream Management System; AEDSMS）を提案する．AEDSMS では，ソフトウェア開発時に位置透過で再利用性の高い高レベルなクエリから車種ごとにカスタマイズされた低レベルなクエリに変換する．ソフトウェア開発時にクエリを変換することで，クエリ実行時にはその変換を行わ

ずオペレータのノードへの配置が確定した状態となる．これにより，リアルタイム処理要求に必要となる予測可能性と車種展開におけるクエリの再利用性を高める．また，クエリが車種ごとに異なるノードやネットワーク構成に影響を受けない性質である位置透過性を実現するためのオペレータ配置方式においては，車載ネットワークに対してノード間でデータが流れる経路を正しく考慮してネットワーク使用量を見積もるようにすることで従来のオペレータ配置方式を適用可能にする．更に，各ノードに配置されたオペレータセットにデッドラインを適切に設定し，各ノード上の DSMS にリアルタイムスケジューリングを導入することで，分散ストリーム処理全体のリアルタイム処理要求を満たす．加えて，センサデータフュージョンを行うオペレータをデータストリーム処理システムに導入することでデータの信頼性を向上する．車々間通信を用いる車両衝突警告を AEDSMS のクエリとして作成し，AEDSMS におけるクエリ実行時の性能を評価する．その結果，デッドライン処理要求を満たすことを確認し，車両衝突事故率の低減及び車両情報の位置精度の向上を確認する．

2 つ目の研究として，各ノード上の DSMS におけるリアルタイムスケジューリング方式を提案する．車載システムのストリーム処理ではオペレータの出力が分岐する場合や車外からの入力データの到着が遅れてタイムアウトする場合など複雑なデータ処理に対応する必要がある．特に車々間通信のように大量のデータが様々なタイミングで入力されるストリーム処理ではスケジューリングのオーバーヘッドを削減することが必要となる．そこで，本研究ではオペレータの列をまとめてスケジューリングを行うことでスケジューリングのオーバヘッドを削減する方法を提案する．本提案では，オペレータの出力に分岐がある場合やタイムアウトがある場合に対応できるようにオペレータの列を構成するアルゴリズムを示す．基本性能評価により本方式がストリーム処理における従来のスケジューリング方式に比べてデッドライン処理要求を満たすことを確認する．アプリケーション性能評価により，車々間通信を用いた車両衝突警告のクエリを用いて，デッドライン処理要求を満たし車両衝突事故率を低減することを確認する．

3 つ目の研究として，通信遅延などの影響でセンサからデータを読み込む時刻と DSMS にデータが入力される時刻が異なる out-of-order な入力データに対して，各ノード上の DSMS のリアルタイムスケジューリングを効率的に行う方法を提案する．Out-of-order な入力データを考慮するには，ストリームキューは従来のように First-In, First-Out（FIFO）ではなく優先度キューが必要となる．特に特定のストリームキューで優先度キューの走査が不要であることを明らかにし，それに基づ

き効率的にリアルタイムスケジューリングを行う方式を示す．実験評価では，車々間通信を用いた車両衝突警告のクエリを用いて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含む従来方式と比較し，デッドライン処理要求を満たし車両衝突事故率を低減することを確認する．

以上の 3 点を柱として，入力データ量が大きくその到着タイミングも変動する車々間通信を用いた安全運転支援を主なアプリケーションの対象として，そのデータ処理要求を満たすストリーム処理方式を提案する．車々間通信を用いた車両衝突警告をストリーム処理により実現し，リアルタイム処理要求や車両衝突回避及び車両位置精度向などにおけるクエリ実行時の性能を評価する．以上の内容により，車々間通信を用いた安全運転支援システムにおける提案方式の有効性を確認する．

目 次

図 目 次

表 目 次

第1章　序論

1.1　研究背景

車載システムには，図 1.1 のように，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 [23, 49]．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 [8, 16]．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処理が求められる．このリアルタイム処理要求を満たせないと，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を防ぐことができない．

一方，安全運転支援のセンサ情報処理では，車両や歩行者の追跡が行われる．しかし，センサデータには誤差があるため，GPS や複数のセンサを統合して誤差を低減することで信頼性を上げるセンサデータフュージョンが必要となる．車載システムに搭載される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，図 1.1 のように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 [52]．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 [19]．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 [17]，車々間通信から受信するデータの量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

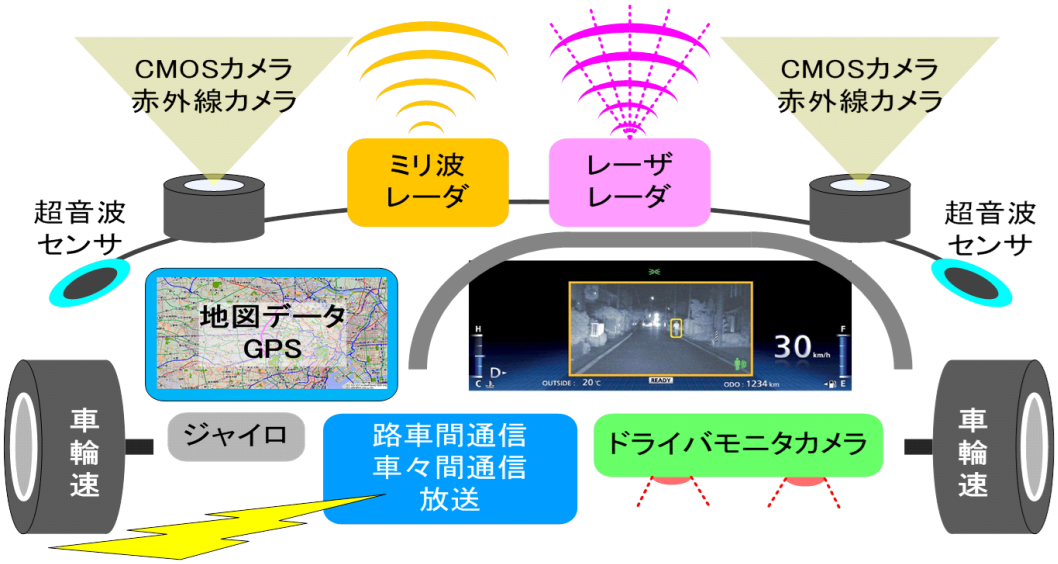


図 1.1: 車載システムに搭載されるセンサや通信機器

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，リアルタイム処理要求を満たしながら，多くのデータを信頼性をあげて処理することが重要な課題となる．また，車々間通信から受信するデータでは，到着タイミングが通信遅延を無視しても 100 ミリ秒遅れる可能性があり，センサからデータを読み込む時刻と車々間通信を経由して車載システムにデータが到着する時刻は異なる．そのため，このような out-of-order な入力データへの対応も課題となる．

一方，車載システムは ECU（electronic control unit）と呼ばれる複数のノードが車載ネットワークで繋がれた分散システムである．アプリケーションソフトウェアやセンサはそれらのノードに分散して配置されている．それらの構成は車種によって異なり近年の高級車では 70 個から 100 個程度のノードから構成される複雑な分散システムである．そのため，自動車の製造メーカーではソフトウェアの開発コストを抑えることが課題となっており，ソフトウェア開発がノードやネットワークの物理構成の違いに影響を受けにくいソフトウェアプラットフォームが必要である．

車載システムにおけるデータ量の増大やデータ処理の複雑化に対応するには，データの管理や処理を効率的に行うデータ中心のアーキテクチャが有効でありプラットフォームとしてはデータ管理システムの導入が有効である．特に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システムが好適である．図 1.2 に一般的なデータストリーム処理システムの概要を示

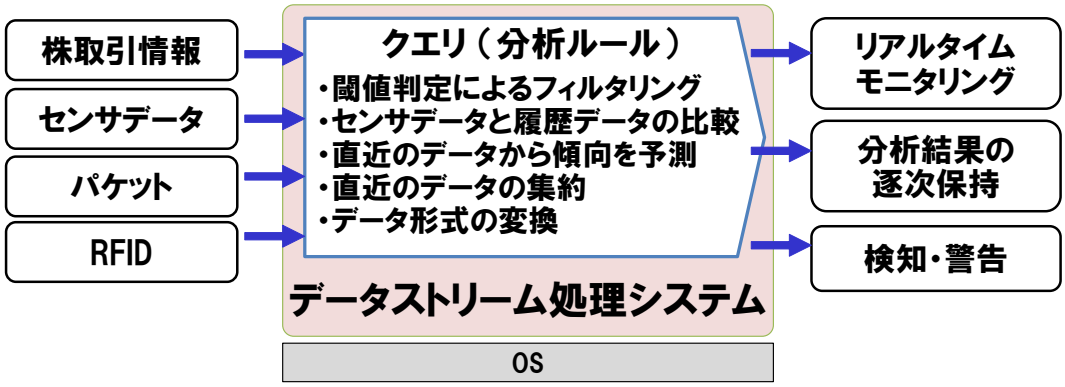


図 1.2: データストリーム処理システムの概要

す．データストリーム処理システムでは，データ処理の実行前にあらかじめデータ分析のルール（クエリ）を登録する．著者が 2011 年 4 月から 2015 年 3 月まで参加していた名古屋大学 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）の車載データ統合プロジェクトでは，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 [46, 51]，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 [48, 53]．

しかしながら，従来のデータストリーム処理システムを車載システムのデータ処理に適用することは難しい．車載システムは車載ネットワークで繋がれた分散システムであるので，分散ストリーム処理システムの導入が必要である．特に重要な課題としてデッドライン処理要求においては，従来のストリーム処理の研究で行われてきた平均遅延時間の削減やスループットの向上とは異なる技術が必要となる．

1.2　論文の概要

本研究では，特に入力データの量や到着タイミングが大きく変動する車々間通信を用いた安全運転支援におけるセンサ情報処理をアプリケーションの対象とする．従来のストリーム処理の研究とは異なるリアルタイム処理要求を中心に，車載システムのデータ処理要求を満たす分散ストリーム処理方式を提案する．本研究の位置づけを図 1.3 に示す．

１つ目の研究として，車載システム向けの分散ストリーム処理システム

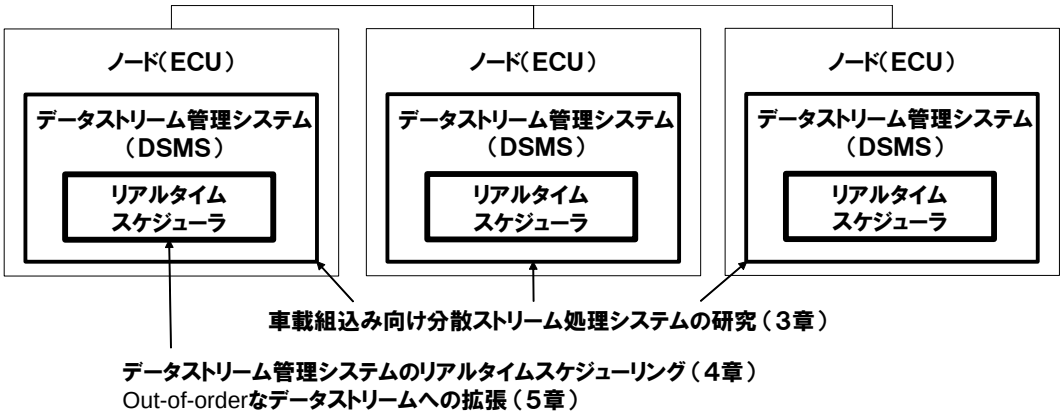


図 1.3: 本研究の位置づけ

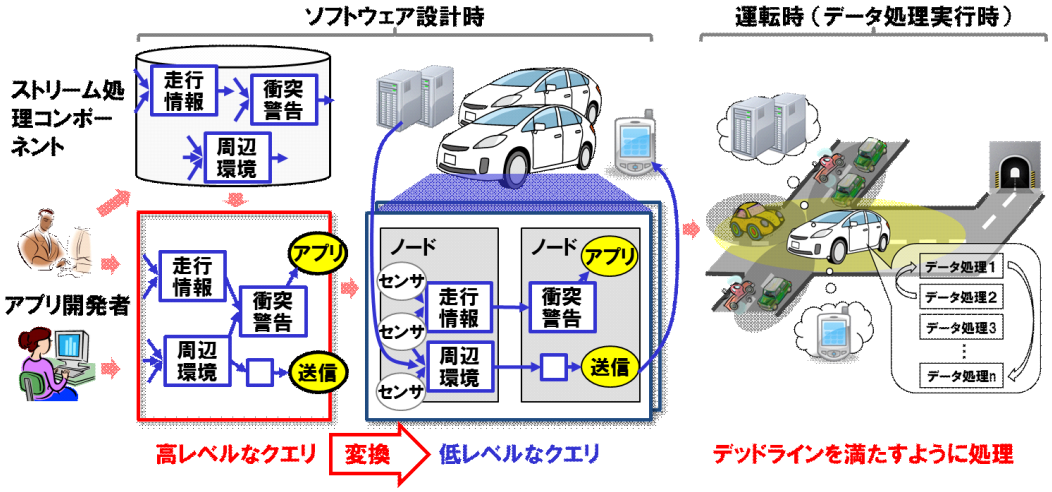


図 1.4: AEDSMS の概要

（AEDSMS）を提案する．図 1.4 に AEDSMS の概要を示す．AEDSMS ではソフトウェア開発時に位置透過で再利用性の高い高レベルなクエリから車種ごとにカスタマイズされた低レベルなクエリに変換する．これにより，リアルタイム処理要求に必要となる予測可能性と車種展開におけるクエリの再利用性を高める．また，複雑な車載ネットワークに対してリアルタイムシステムの分野で用いられるアーキテクチャグラフによりノード間のストリームの経路を正しく考慮することでオペレータを各ノードに適切に配置する．更に，各ノードに配置されたオペレータセットにデッドラインを適切に設定し，各ノード上の DSMS にリアルタイムスケジューリングを導入することで，分散ストリーム処理全体のリアルタイム処理要求を満たす．更に，センサデータフュージョンを行うオペレータをデータストリーム処理システムに導入することで車両情報の位置精度を向上する．

2つ目の研究として，各ノード上のDSMSにおけるリアルタイムスケジューリング方式を提案する．本方式は，車載システムのデータ処理はオペレータの出力が分岐する場合やタイムアウトを持つオペレータなどを含む複雑なデータ処理に対応できる．特に車々間通信のように大量のデータが様々なタイミングで入力されるストリーム処理ではスケジューリングのオーバーヘッドを削減することが必要となる．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自然に導入する方法を示すとともに，リアルタイム処理要求を満たしたままオーバーヘッドのみを削減する方法を示す.

3つ目の研究として，車々間通信など車外からの入力データがout-of-orderに到着する場合において，各ノード上のDSMSにおけるリアルタイムスケジューリングを効率的に行う方法を提案する．Out-of-orderな入力データを考慮することで，従来のストリーム処理方式とは異なりストリームキューはFIFOではなく優先度キューが必要であることを述べる．特に，特定のストリームキューで優先度キューの走査が不要であることを明らかにし，それに基づき効率的にリアルタイムスケジューリングを行う方法を示す.

1.3　論文の構成

本論文の構成は次の通りである．2章では，本研究の前提となる事項について，車載システムのデータ処理における要求を述べる．また，本研究全体で対象とするアプリケーションや前提事項を説明する．3章では，車載システムのデータ処理要求から既存のストリーム処理方式の課題を整理し，それらの課題を解決する分散ストリーム処理システムを提案する．4章では，各ノードにおけ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提案する．5章では，4章の拡張として特に入力データがout-of-orderな場合に効率的にリアルタイムスケジューリングを実現する方式を提案する．最後に6章で，本研究を総括して今後の展望について述べる.

第2章　分散ストリーム処理の車載システム導入における課題

本章では，本研究で主に想定する車々間通信を用いた車両衝突警告を説明する．次に車載システムのデータ処理要求を整理し，最後にストリーム処理による車載システムのデータ管理について述べる．

2.1　車々間通信を用いた車両衝突警告の例

まず，本研究の中心的な課題であるリアルタイム処理要求について，図 2.1 と図 2.2 を用いて説明する．図 2.1 は，見通しの悪い交差点における車々間通信を用いた車両衝突警告を表しており，以下のケースを想定する．自車両 A は，車両衝突警告により車両 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,F とは車々間通信できない．ETSI では，車両衝突警告に対する end-to-end デッドラインを 300 ミリ秒と提示しており [19]，ここでは，車両 E が車両 X を検知してから 300 ミリ秒以内に，自車両 A が車両 X の情報を受信し車両衝突警告を完了できなければ，自車両 A と車両 X は衝突する状況を想定する．

図 2.2 では，図 2.1 において自車両 A が現在時刻 t において車両 E からメッセージを受信し車両 B,C,D から受信したメッセージの処理を待っている状況を示している．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．タイムスタンプはセンサからデータが発生した時刻として付与される．説明を簡単にするために，各車両は同時にセンシングを行い，車両 E はその 95 ミリ秒後にそのデータを送信し，他の車両はセンシングの 5 ミリ秒後にそのデータを送信した場合を考える．各メッセージを処理するのに 30 ミリ秒かかると想定する．

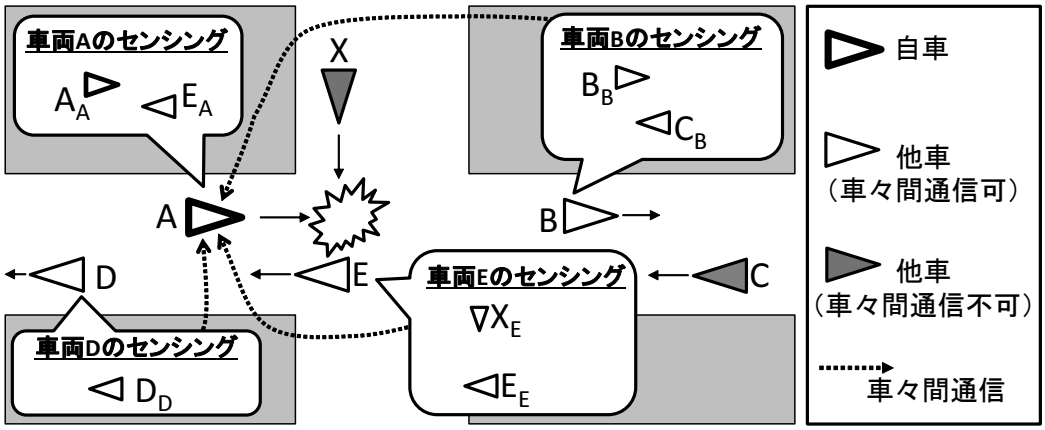


図 2.1: 車々間通信を用いた車両衝突警告

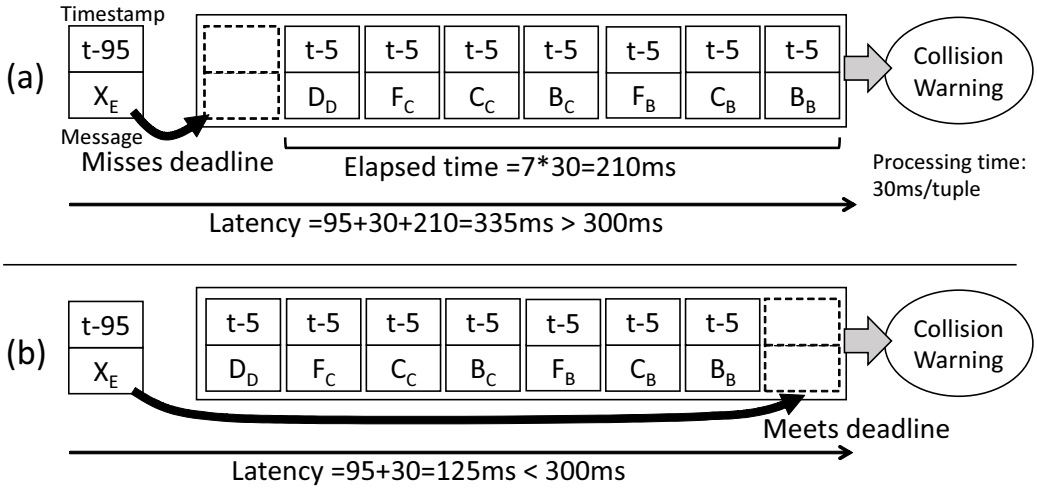


図 2.2: 車々間通信から得られるデータの処理順序

図 2.2(a) では first in first（FIFO）でメッセージを処理した場合を示している．X_E の前に処理することを待っているメッセージが 7 個あるため，X_E を処理する遅延時間は $95 + 30 + 7 \times 30 = 335$ ミリ秒である．その結果，300 ミリ秒の end-to-end デッドラインをミスしてしまい自車両は車両 X と衝突してしまう．このように車載システムのデータ処理では end-to-end デッドラインを満たすことが重要である．

他方，図 2.2(b) ではリアルタイムスケジューリングを用いる場合を示している．この場合 X_E のデッドラインが最も早いので X_E が最初に処理される．その結果，自車両 A は X_E と残りのメッセージを end-to-end デッドライン以内に処理することができ，車両 X との衝突を避けることが可能となる．このように車載システムのデータ処理にリアルタイムスケジューリングを適用することは車両の衝突を回

避することに有効である．

また，リアルタイム処理要求の他に以下のようなデータ処理要求もある．センサデータにはノイズがのるためデータ処理には信頼性の向上も必要である．例えば，自車両 A が車両 E の位置情報を知る場合に，自車両 A がセンシングした結果 E_A のみを用いるよりも車両 E 自身がセンシングした結果 E_E を車々間通信で受信しそれらの情報を融合することで，車両 E_E の位置情報の誤差を低減し信頼性を向上することができる．また，図 2.1 のような多くの車両が走行する交差点では，車々間通信から得られるデータ量も増大する．そのため，車々間通信を用いる場合には入力データ量の変動に対応する必要がある．加えて，この例では車両 E から 95 ミリ秒後に他の車両からは 5 ミリ秒後にデータを受信したように，車外との通信を用いる場合には各車両でセンシングしてから自車両が受信するまでに遅延時間が発生する．そのため，out-of-order な入力データの到着にも対応する必要がある．

データ処理の結果は，ブレーキやジャイロなどのアクチュエータの制御やカーナビゲーションのディスプレイへの表示に用いられる．これにより，車載システムにおける安全運転支援を行う．

2.2　車載システムにおけるデータ処理の要求

前節で述べたような本研究で対象とする車載システムのアプリケーションは以下の 3 ステップを含むデータ処理である．(1) 多数のノードに分散するセンサや通信機器からデータを取得し，(2) それらのデータを組み合わせて処理し，(3) 多くのノードに分散するアクチュエータを制御する．車載システムにおけるデータ処理の要求として，リアルタイム性における要求，信頼性における要求，入力データ量の変動における要求，out-of-order な入力データの到着がある．

ETSI では，協調 ITS を活用する車載システムや路側器などを対象に様々な仕様を策定し国際標準化を進めている．本研究におけるデータ処理の要求としては，ETSI の仕様に従い，車載システムのデータ管理において技術的な課題となるものを対象とする．以降で，それらの各要求について説明する．

2.2.1　リアルタイム性における要求

車載システムのデータ処理には車載システムのソフトウェアを設計する際に開発者により end-to-end デッドラインが設定される．End-to-end デッドラインとは

センサからデータを読みこんでからその処理が完了するまでの許容遅延時間である．例えば，車両の衝突警告の場合，end-to-end デッドラインは ETSI により 300 ミリ秒と定められている [19]．一方，車両の制御の場合，end-to-end デッドラインは数十ミリ秒から 100 ミリ秒程度が一般的である．この要求は安全運転支援システムにおける重要な要求である．

2.2.2　信頼性における要求

ETSI では協調 ITS の通信に用いるデータの定義を定めている．そのデータ定義では，車両の緯度，経度，速度といった連続値に対して，信頼性を付与することが要求される．信頼性とは観測値と真値との誤差であり，左右対称の 95%の信頼区間として大抵定義される [18]．特に，安全運転支援における車両や歩行者の追跡では位置情報の精度における信頼性を高めることが重要な課題となっている．また，ETSI の定める信頼性を含めたデータを車々間通信などで車外に配信できるように対応することが望ましい．

2.2.3　入力データ量の変動における要求

近年，運転支援システムにおいて，車々車間/路車間通信など，協調 ITS を利用する検討が進んでいる．例えば，交差点での衝突事故は，報告された車両事故の 50%を占め，交通事故の最も主要な 1 つである [20]．特に，見通しの悪い交差点では，カメラやレーダといった自律センサだけでは，周辺車両を十分に検知できず，協調 ITS の活用が望まれている．車々間/路車間通信は，100 ミリ秒周期で，半径 100 メートル程度から 1 キロメートル程度までの範囲にいる他車両へブロードキャストされる可能性がある [19]．その結果，車両台数の多い交差点などでは，車載システムに入力されるデータ量は大きく増加する．ETSI では，1 秒間に最大 1000 台の車両データを受信できることを仕様として要求している [19]．そのため，車載システムではこれらの大量のデータを効率的に処理することが要求される．

2.2.4　Out-of-order な入力データの到着

車々間/路車間通信は，100 ミリ秒周期で，半径 100 メートル程度から 1 キロメートル程度までの範囲にいる他車両へブロードキャストされる可能性がある [19]．車々

間通信を行う車両は，それぞれの車両のタイミングでメッセージをブロードキャストする．その結果，データがセンサから読みこまれてから車々間通信で配信されるまで最大で 100 ミリ秒の遅延が生じる．そのため，通信遅延を無視したとしても，ある車両が車々間通信でメッセージを受信するまで最大 100 ミリ秒まで遅延時間に違いが生じる．車両衝突警告の end-to-end デッドラインが 300 ミリ秒であることを考えるとこの遅延の差は無視できず，車々間通信から得られる入力データはデータ発生順に並んでいない out-of-order なデータである．

2.3　ストリーム処理による車載システムのデータ管理

車載システムではセンサの種類や数が増加しており，データ量やその処理においても大量化／複雑化している [11]．従来の車載システム開発のサプライヤは各アプリケーションプログラムとデータ処理を区別せず開発する．そのため，本来共通化できるデータ処理があっても重複して実装されてしまう．加えて，車載ネットワークやノードから構成される物理構成やセンサの配置などは車種によって異なる．従来のソフトウェア開発では，それらの構成が変わるとソフトウェアの大部分を修正する必要があり，その開発工数が増大する課題をかかえている．以降では，データストリーム処理システム，分散ストリーム処理システムを順に説明し，車載システムに向けたストリーム処理の適用について説明する．

2.3.1　データストリーム処理システム

データストリーム処理システムとは，センサデータやネットワークのパケットなど連続的に到着する入力データから現在を分析するのに適したデータ管理システムである [14, 24]．これまで，ネットワーク，株価，交通流などのモニタリングや異常検知などの分野に用いられてきた．従来のデータベース管理システムとは異なり，問い合わせ（クエリ）はデータを処理する前に登録しておき，入力データが到着するたびにメモリ上で問い合わせを実行する．図 2.3 のように，登録されたクエリは，オペレータと呼ばれる演算単位をストリームで繋いだデータフローとして表現される．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繋ぎ変えることで，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．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

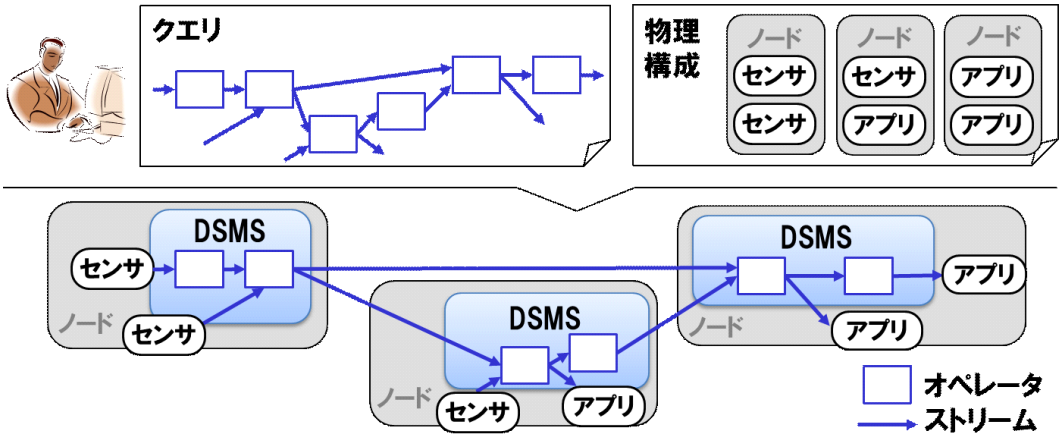


図 2.3: 分散ストリーム処理システム

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．

2.3.2 分散ストリーム処理システム

オペレータが複数のノードに配置される場合に対応するデータストリーム処理システムを分散ストリーム処理システムと呼ぶ．これまで，センサネットワークの分野で地理的に分散するセンサデータを処理する場合を対象に研究されてきた．図 2.3 のように，分散ストリーム処理システムでは，各ノードにデータストリーム管理システム（DSMS）が搭載され，各ノードでは DSMS がそのノードに配置されたオペレータセットを実行する [1, 26, 31]．また，オペレータの配置をユーザが指定せずシステムが適切に配置することで物理構成に依存しない位置透過な問い合わせを実現できる [26, 31]．

2.3.3 車載システムに向けたストリーム処理の適用

著者が参加していた車載データ統合プロジェクトでは，車載システムのデータを統合管理するアプローチによりソフトウェアの開発工数を削減する研究を行っている [46, 48, 51, 53]．従来の車載システムのソフトウェア構成と車載データ統合プロジェクトによるアプローチを図 2.4 と図 2.5 にそれぞれ示す．図 2.5 のように車載システムのデータ処理をアプリケーションプログラムから分離することで，複

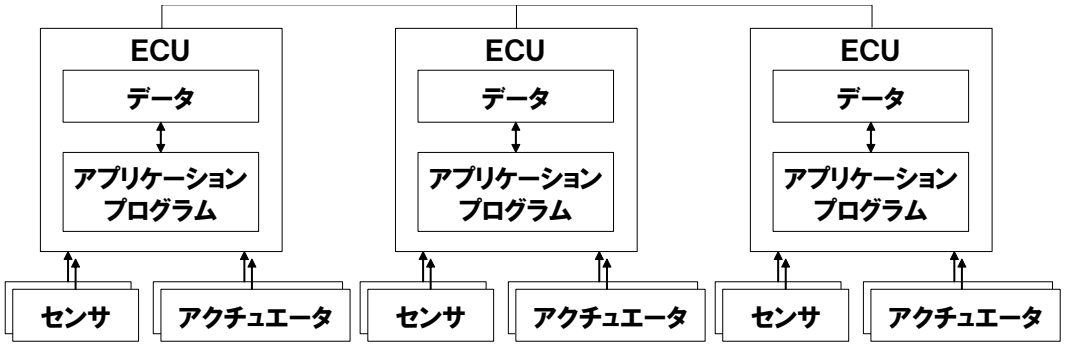


図 2.4: 従来の車載システムにおけるソフトウェア構成

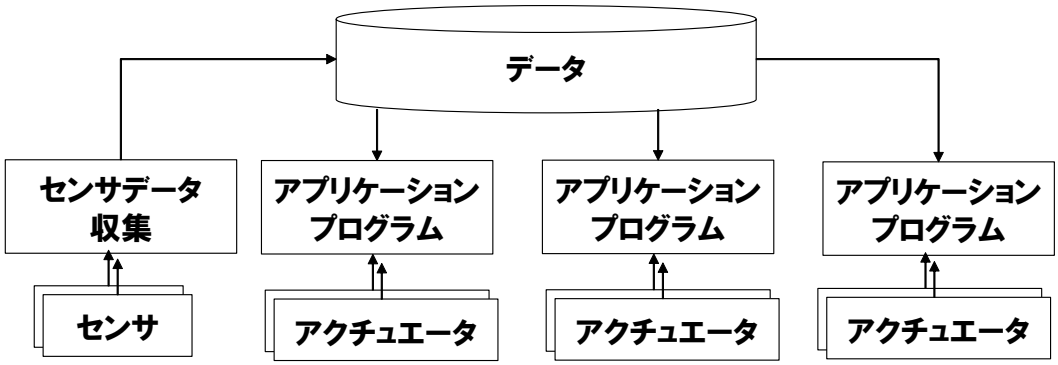


図 2.5: 車載システムのデータ統合によるアプローチ

数のアプリケーションで利用される再利用性の高いデータを定義し，アプリケーションを特定のセンサに依存させない．

車載データ統合プロジェクトでは，車載システムのデータ管理にデータストリーム処理システムを用いることで，低遅延なデータ処理とデータの統合管理によるソフトウェア開発の効率化を両立する [46, 48, 51, 53]．図 2.6 はデータストリーム処理システムを車載システムに導入した場合を示す．車載システムのデータ処理はクエリを用いて記述される．これにより，センサやアプリケーションの追加/変更に対して，オペレータと呼ばれる処理ブロックを追加/置換することにより，比較的容易に対応できる．

本研究では更に，複数のセンサデータを融合することで信頼性を高めて様々なアプリケーションにデータを提供する．加えて，物理的には複数のノードに分散するこれらのデータへのアクセスを位置透過にし，アプリケーションをノードや車載ネットワークの構成に依存させない．本研究では各ノードは ECU でありシングルプロセッサを想定する．

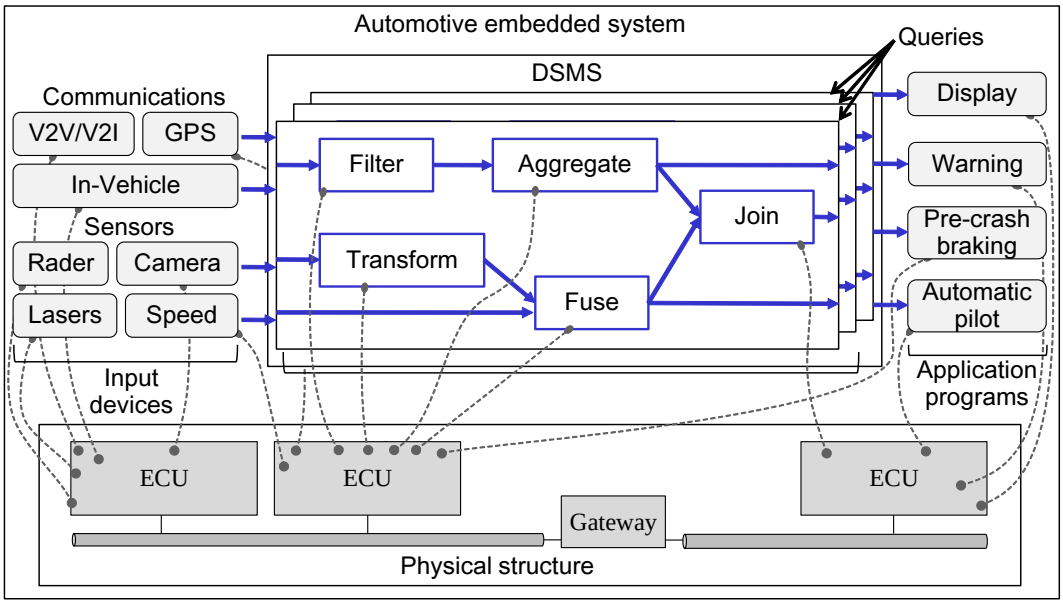


図 2.6: 車載システムへの分散ストリーム処理の導入

2.4 ストリーム処理モデル

本研究の全体を通して対象とするストリーム処理のモデルを図 2.7 を用いて説明する．このモデルは，Aurora [2] や Borealis [1]，SystemS [22] 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．本論文で想定するクエリは，**オペレータ**の入出力を**ストリーム**で繋いだデータフローとして表現される．センサなど，データストリーム処理システム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，データストリーム処理システムの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．ストリームは，**タプル**の列である．タプルは，時刻やセンサ値など複数の属性の値の組である．オペレータは，入力として繋がれたストリームからタプルを取り出して処理し，出力として繋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繋がっているオペレータを**出力オペレータ**と呼ぶ．

クエリは，オペレータ，ソース，シンクを頂点とし，それを繋ぐストリームを辺とみなすことで，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

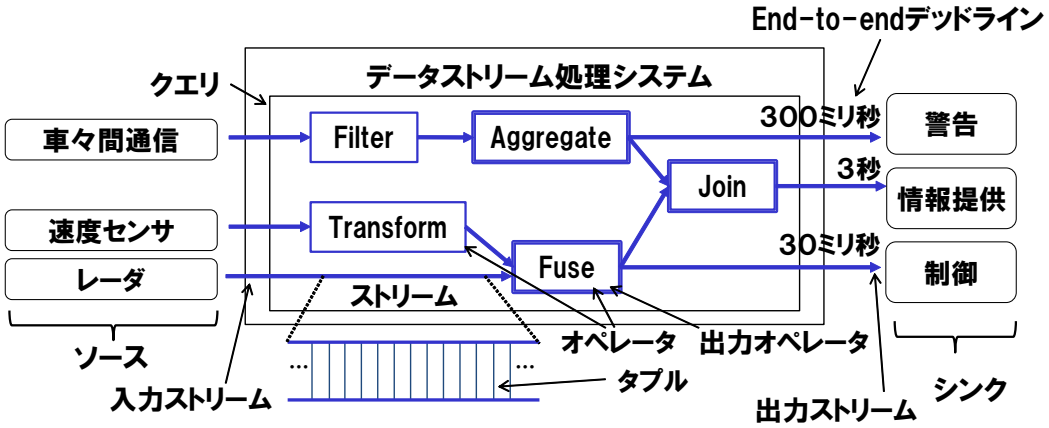


図 2.7: 本研究全体で想定するストリーム処理のモデル

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．

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許容時間である **end-to-end デッドライン**が指定されている場合を想定する．End-to-end デッドラインを持つ従来のモデルでは，クエリ（オペレータの出力が分岐する場合は出力ストリーム）に指定する方式 [30] と，クエリの入力ストリームに指定する方式 [45] がある．本研究では，複数のアプリケーションが異なる end-to-end デッドラインを持つ場合に対応するため，[30] 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．

リアルタイム処理要求は，タプルがそれらの出力ストリームに挿入される時に，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読み込まれた時刻でタイムスタンプ t_p を打刻する．リアルタイム処理要求とは，タプル p を出力ストリーム s に挿入する時刻が，式 (2.1) で定義される **絶対デッドライン** $d_{p,s}$ より遅れないことである．

$$d_{p,s} = l_s + t_p \tag{2.1}$$

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．リアルタイムスケジューリングとしては，式 (2.1) を優先度として式 (2.1) が小さいタプル p から順に処理する．このように絶対デッドラインを優先度とするスケジューリング方式を earliest

deadline first（**EDF**）という [9].

車々間通信から得られるデータは，データ量の変動が大きく，デッドラインを越えずに常に全てのデータを処理できるとはできない．そのため，このような入力ストリームに **load shedder** を設定する．Load shedder とは，設定した入力ストリームにおけるデータ量が許容範囲以上に増大した場合にデータを間引く機構である．ユーザは load shedder に許容できる単位時間あたりの最大入力データ量を指定する．データの間引き方には，(a) ランダムにタプルを間引く方法と (b) タプルの価値を評価し価値の低いタプルを間引く方法がある．本研究では特に記載のない限り (a) を用いる．タプルの価値を評価する場合，その評価関数はユーザにより定義される．load shedder によりデータを間引くことを **load shedding** という [2, 24]．Load shedding を行うことで，リアルタイム処理要求は満たしやすくなるが，車両衝突事故は発生しやすくなりデータの信頼性も低下しやすい．本研究では，load shedding によりデータを間引く影響を車両衝突回避やセンサデータフュージョンにおける効果により評価する．

第3章　車載組込み分散ストリーム処理システム

3.1　概要

データストリーム処理システムは車載システムのデータ処理のように，連続して到着するデータを低遅延に処理するのに適する．著者が参加していた車載データ統合プロジェクトでは，これまで車載システムにDSMSを適用することに向けてデータセントリックなアーキテクチャを研究してきた [33]．また，車載組込みシステムへの搭載に向けて少リソースで低遅延なデータ処理を実現するDSMS（eDSMS）を開発してきた [53]．しかしながら，車載システムのデータ処理には2章で述べたような要求があり，それを満たすようにストリーム処理を行うには新しい技術課題を解決する必要がある．

車載システムではクエリ実行時の最悪ケースを予測する予測可能性が求められる一方で，車種ごとの違いを吸収するような再利用性の高いクエリ記述が求められる．また，車載システムは車載ネットワークで繋がれた分散システムであるが，従来の分散ストリーム処理システムではバスとゲートウェイで繋がれた車載ネットワークに適用することは難しい．更に，ストリーム処理のリアルタイムスケジューリング方式は単一ノードのみが対象であり，分散ストリーム処理へのリアルタイムスケジューリング方式の拡張も必要となる．加えて，データストリーム処理システムでは，センサデータフュージョンによりデータの信頼性を向上する必要があるが，out-of-orderな入力データに対応する必要がある．従来のストリーム処理システムではこれらの課題を扱うことは難しい．

そこで，本研究ではこれらの課題を解決する分散ストリーム処理システムを提案する．主な貢献は以下の通りである．

(A) 車載システムのデータ処理要求からデータストリーム処理システムにおける技術課題を明らかにする．

(B) それらの課題を解決するため，以下の (a)–(d) を実現するデータストリーム

処理システム（AEDSMS）を開発する：

(a) 車載システムの設計時に，位置透過で高レベルなクエリから車種ごとにカスタマイズされた低レベルなクエリへ変換

(b) 車載ネットワーク上における分散ストリーム処理

(c) 分散ストリーム処理のリアルタイムスケジューリング

(d) Out-of-order な入力データに対応するセンサデータフュージョンのオペレータ

(C) ETSI の仕様に従って，実行時の安全運転支援システムにおけるフィージビリティを確認する．

本章の構成は以下のとおりである．まず，3.2 節で関連研究を紹介する．次に，3.3 節で車載システムにおける分散ストリーム処理の技術課題を抽出する．その後，3.4 節で提案方式を示し，3.5 節でその評価を行う．最後に，3.6 節では課題に対する解決を考察し，3.7 節でまとめを述べる．

3.2 関連研究

3.2.1 ストリーム処理を用いない車載ソフトウェアプラットフォーム

車載システムのソフトウェアの再利用性や生産性を向上する車載システムのソフトウェアプラットフォームとしては，AUTomotive Open System ARchitecture (AUTOSAR) がある [21]．そのコンソーシアムにおけるパートナーには多くの自動車メーカーが参画しており，商用の車載システムへの適用実績もある．しかし，AUTOSAR は 2.3 章で述べたデータ管理の問題を解決しない．

SAFESPOT プロジェクトでは，センサや車々間／路車間通信から得られたデータをリレーショナルデータベース（RDB）を用いて管理する車載システムのプラットフォームを開発しており，そのコンソーシアムのメンバーには Bosch や Volvo などの自動車メーカーが含まれ，実際の車載システムを用いた実証実験も行われた [42]．また，[34, 35] では車載システムのデータ管理におけるリアルタイム処理要求を満たす RDB を開発した．しかしながら，車載システムのデータは高頻度に更新されることや，運転支援では低遅延なデータ処理が求められることから，RDB によるデータ管理には適さない．

3.2.2　データストリーム処理システム

データストリーム処理システムはセンサデータやネットワークのパケットなどの連続的なデータを低遅延に管理することに適したデータ管理システムである．Oracle CEP や Esper などの複合イベント処理システムと同様に，STREAM, Aurora, Gigascope, InfoSphere Streams など，これまでのデータストリーム処理システムの多くは，汎用システム向けに開発されてきた [14]．近年，Spark Streaming や Storm などクラウドコンピューティングを用いたスケーラブルな並列／分散ストリーム処理システムが盛んに開発されている．しかしながら，これらのデータストリーム処理システムでは 2 章で述べた車載システムのデータ処理要求を満たすことは難しい．また，これらのソフトウェアのサイズは車載システムのリソース容量に対して大きすぎるため車載システムに搭載することも難しい [53].

センサネットワークの分野では，車載システムの分野と同様に組込みシステムに搭載するほど少ないリソース容量で動作するデータストリーム処理システムが開発されている．例えば，[38] では車載システム開発の診断やテストのために車載システムに搭載するストリーム処理方法が提案されている．また，Vehicle Data Stream Mining System (VEDAS) [28] や Minefleet [29] では，トラック運転手の異常検知などを目的として，車載システムやモバイルデバイスに搭載する分散ストリーム処理システムが提案された．StreamCars [7] は運転支援に向けたデータストリーム処理システムであり，本研究の目的と類似する．StreamCars ではセンサデータの信頼性を向上するためセンサデータフュージョンを行うオペレータを提供する．しかし，その性能や実現方法などは具体的には記述されていない．加えて，これらの従来研究では，車載ネットワーク上の分散ストリーム処理やリアルタイム処理要求について車載システムのデータ処理において重要な課題であるにも関わらず考慮していない．そのため，2 章で述べた車載システムのデータ処理要求を満たせない.

3.2.3　分散ストリーム処理のリアルタイムスケジューリング

リアルタイム要求を満たすために，デッドラインを満たす処理順序を決定するリアルタイムスケジューリングがリアルタイムシステムの分野でこれまでよく研究されてきている [9]．リアルタイムシステムやデータベースの分野では，ストリーム処理に適用する研究が幾らか行われている．[37, 44] は単一ノード上のストリーム処理の EDF スケジューリングを提案した．しかしながら，それらはクエリと

end-to-end デッドラインに 1 対 1 の仮定を暗黙的に置いているため，オペレータの出力が分岐するような場合に対応できない．[30] は rate monotonic と呼ばれる EDF とは別のリアルタイムスケジューリング方式に基づく方式を提案したが，[37, 44] と同様の理由でマルチクエリに対応できない．[45] では EDF に基づき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 デッドラインが全て同一であることが前提となる．[32] はマルチクエリに対応するとともに出力ストリームごとに end-to-end デッドラインが異なる場合にも対応した．この方法は，出力オペレータを含むオペレータの列に対してその end-to-end デッドラインに基づき EDF でスケジューリングする．しかし，分散ストリーム処理では同一のノードにそのオペレータ列が配置されるとは限らないのでこの方法を適用できない．それゆえ，従来の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分散ストリーム処理に適用することは難しい．

3.2.4　分散ストリーム処理のオペレータ配置

オペレータ配置問題は，遅延時間やスループットなどの性能に影響を与えるため，分散ストリーム処理の重要な技術課題である．それは，クエリを構成するオペレータとストリームを各ノードやネットワークに割り当てる組み合わせ問題となるので計算困難な問題である．これまで通信量の削減やノード間のロードバランスなどを主な目的として，様々なヒューリスティクスな方法が研究されてきた [31]．その後，混合整数計画問題（MILP）としてオペレータ配置を定式化する方式が提案された [26]．[26] では，分散ストリーム処理に特有な 1 つのストリームを複数のノードを経由するように割り当てることにも対応できる．しかしながら，これらの従来方式はオーバーレイネットワークを想定しており，3.3 節で述べるようにバスとゲートウェイからなる車載ネットワークに適用することは難しい．

3.3　車載システム向け分散ストリーム処理の技術課題

2 章で述べたデータ処理要求を解決するために，本節では従来のストリーム処理の課題とは異なる車載システムに特有の以下の課題を抽出する．

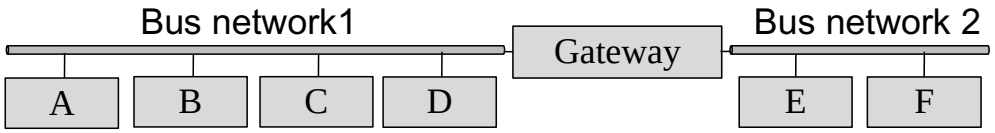


図 3.1: 車載ネットワークとノードの例

3.3.1　C1: クエリ実行時の予測可能性

従来分野では，データストリーム処理システムが稼働している最中にユーザがその場で様々なクエリを登録，削除，変更できることがデータストリーム処理システムを利用するメリットである．また，オペレータ配置や最適化などの方法として，実行時のパフォーマンスを向上するために，実行時の状況に応じて，オペレータを動的に移動する方法がこれまで研究されてきた [1].

一方，2 章で述べたように車載組込みシステムでは平均的に遅延時間を短くするだけでなく，デッドラインを超えないように処理を完了するリアルタイム要求を満たす必要がある．この要求は，これらのアプリケーションにデータを提供するデータストリーム処理システムにも求められ，データが生成されてからあらかじめ指定された時刻までに処理を完了し，アプリケーションプログラムに処理されたデータを配信する必要がある．このような違いから，実行時の AEDSMS ではソフトウェアの動作を検証しやすくし最悪ケースの遅延時間を正確に予測する予測可能性が従来分野よりも求められる．

3.3.2　C2: 車載ネットワーク上の分散ストリーム処理

車載ネットワークやノードの構成は車種ごとに異なる．そのため，車載システムにおける分散ストリーム処理ではオペレータをシステムが適切に配置するオペレータ配置機能が有効である．分散ストリーム処理におけるオペレータ配置はデータ処理の効率化やネットワーク使用量に大きな影響を与えることが知られており，ネットワーク使用量が削減されるようにオペレータを配置する方法がよく研究されてきた [31].

しかしながら，3.2.4 節で述べたようにそれらの方式ではシンプルなオーバーレイネットワークを想定している．一方，車載ネットワークは，主にバス型の複数のネットワークが複数のゲートウェイで繋がっている複合型である．この違いから，従来方式では，車載ネットワークの使用量を見積もることは難しい．

図 3.1 の例のように，ノード A,B,C,D とノード E,F がそれぞれ同一のバスネットワークに繋がれており，それら２つのネットワークが１つのゲートウェイで繋がれている場合を考える．ノードの A と B が通信した場合，直接通信していないノード C と D のネットワーク使用量も増加する．一方，ノード E と F のネットワーク使用量は増加しない．このように，車載ネットワーク上の分散ストリーム処理においては，ネットワークの構成からストリームがノード間をどのように経由するかを考慮する必要がある．

3.3.3　C3: 分散ストリーム処理のリアルタイムスケジューリング

2 章のリアルタイム処理要求で述べたように，車載システムの分散ストリーム処理ではアプリケーションごとに定められた end-to-end デッドラインを越えないようにデータを処理することが必要である．また，車載システムのデータ処理では，車両の位置情報といった共通のデータをを車両衝突警告やナビゲーションなどの複数のアプリケーションで利用するため，オペレータの出力が分岐する場合に対応する必要がある．

しかしながら，スケジューリング方法はストリーム処理の主要な研究テーマであるにも関わらずリアルタイムスケジューリングにおける研究はあまり行われていない．3.2.3 節で述べたように，オペレータの出力が分岐するような一般的なクエリに対しては，各ノードに配置されたオペレータの列に対して適切にデッドラインを設定する技術が無い．そのため，車載システムの分散ストリーム処理にリアルタイムスケジューリングを導入することは従来方式では難しく，各ノードのオペレータセットからタスクとそのデッドラインを適切に設定する技術が必要となる．

3.3.4　C4: センサフュージョンによる信頼性の向上

2 章の信頼性要求で述べたように，運転支援システムではセンサデータの信頼性を向上する必要がある．その一つの手段として，センサフュージョンが有効である．カルマンフィルタやパーティクルフィルタは車両や歩行者の位置精度の向上や追跡に良く用いられるセンサフュージョンのアルゴリズムである．しかしながら，センサフュージョンを実現するデータストリーム処理システムは殆どなく，その実現方法は明らかではない [1].

様々なセンサデータフュージョンのアルゴリズムを開発するには，データストリーム処理システムがその共通機能をオペレータとして提供することが有効である．センサフュージョンのアルゴリズムでは，同一時刻にセンサから読みこまれたデータを融合して，その出力結果を保持する必要がある．一方，2 章で述べたように車載システムのデータは out-of-order であり，古いセンサデータが先に到着する可能性がある．そのため，過去の一定期間の出力結果を保持／管理する必要がある．殆どのデータストリーム処理システムでは，オペレータの入力データを一定期間だけ保持／管理するためにウインドウの機構を提供するが，オペレータの出力結果を管理するウインドウは提供されていない．そのため，センサデータフュージョンの出力結果を保持するウインドウの機構が必要である．

3.3.5　C5: クエリ記述の再利用性

車載組込みシステムのデータ処理は自動車という製品に組み込まれるので，そのデータ処理を表すクエリは自動車の種類に影響を受ける．そのため，自動車の種類が異なると，データストリーム処理システムのクエリを作り直さなければならない．そのため，車種の違いに影響を受けにくい再利用性の高いクエリの記述が求められる．

3.4　車載組込み分散データストリーム管理システム

本節では，3.3 章で整理した課題を解決するため，車載システム向けの分散ストリーム処理システムである AEDSMS を提案する．

3.4.1　デザインコンセプト

AEDSMS の全体構成を図 3.2 に示す．AEDSMS では，ユーザは位置透過でモジュール化されたハイレベルなクエリ（HLQ）を記述する．そして，HLQ からモジュールを展開し各ノードにオペレータを配置することで各車種に適したローレベルなクエリ（LLQ）に変換する．

HLQ では，ユーザは Map, Filter, Union, Join といった Aurora のオペレータのサブセットを利用できる [2]．加えて，ユーザは C++のクラスを用いてオペレータの処理を実装するユーザ定義オペレータ（UDO）を利用できる．UDO では，C++

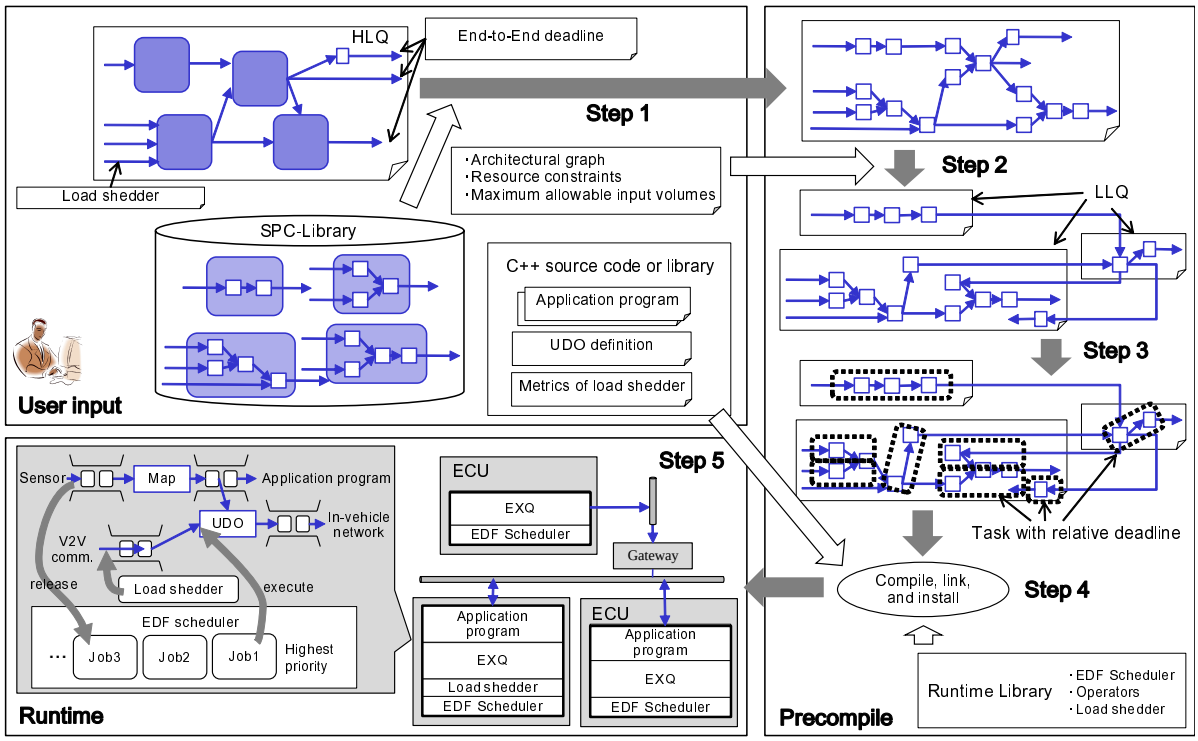


図 3.2: AEDSMS の全体構成

のクラス継承を用いることで段階的な拡張やカスタマイズが可能である．更に，AEDSMS では複数のオペレータを組み合わせてストリーム処理のコンポーネント（SPC）を定義することも可能であり，それらをライブラリ（SPC Library と呼ぶ）として保持する機構を持つ．ユーザは SPC-Library に保持された SPC をオペレータと同様に用いてクエリを記述できる．

異なるセンサから得られる複数のストリームを融合することでデータの信頼性を向上するため，AEDSMS ではセンサデータフュージョンオペレータ (SDFO) を利用できる．ユーザは C++のクラス継承により SDFO に対してセンサフュージョンの特定のアルゴリズムを実装できる．

従来の単一ノードにおけるタイムクリティカルなデータストリーム処理と同様に [32, 37, 44]，ユーザは分散ストリーム処理の出力ストリームに end-to-end デッドラインを設定できる．クエリ実行時には AEDSMS は EDF に基づいて end-to-end デッドラインを満たすようにオペレータを実行する．

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，自車両に搭載された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統合するように，クエリを設計する．この統合する

オペレータでセンサフュージョンを行うことでデータの信頼性を高めることができる．これにより，車々間通信など車外からのデータの到着が遅れた場合，タイムアウトして自車両に搭載されたセンサのみを用いた最低限の処理でリアルタイム要求を満たす．

車載システムに組み込まれるクエリは車種ごとにカスタマイズされるべきであるが，ユーザが記述するクエリは車種に依存するべきではない．これを解決するため，AEDSMS は物理構成に依存しにくい HLQ を提供し，それを各車種の物理構成に適した LLQ に変換する．そして，バイナリコードとして生成されたクエリプランのみを車載システムに搭載する．クエリプランを生成する順番は以下のとおりである．

I. High-Level Queries (HLQs)

HLQ はユーザにより記述されるクエリであり，ネットワークやノードなどの物理構成に依存しない位置透過なクエリである．それはオペレータと SPC を用いて記述される．

II. Low-Level Queries (LLQs)

LLQ は，ネットワークやノードなどの物理構成に従ってカスタマイズされており，特定の車種に依存する．LLQ は SPC を含まずオペレータのみから構成され，オペレータが配置されたノードごとに生成される．

III. Executable Queries (EXQs)

LLQ から生成された EXQ は車載システムに組み込まれたクエリプランである．バイナリコードであるためユーザは容易に編集できない．

3.4.2　ストリーム処理コンポーネント (SPC)

SPC の例を図 3.3 に示す．クエリを定義するのと同様に，SPC はオペレータとストリームから構成されるデータフローとして定義される．SPC の入出力はオペレータと同様でありストリームである．

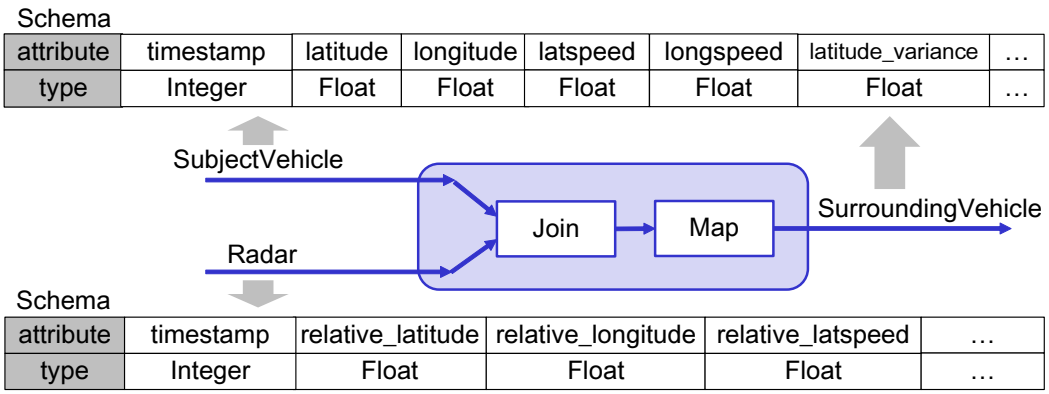


図 3.3: SPC の定義の例

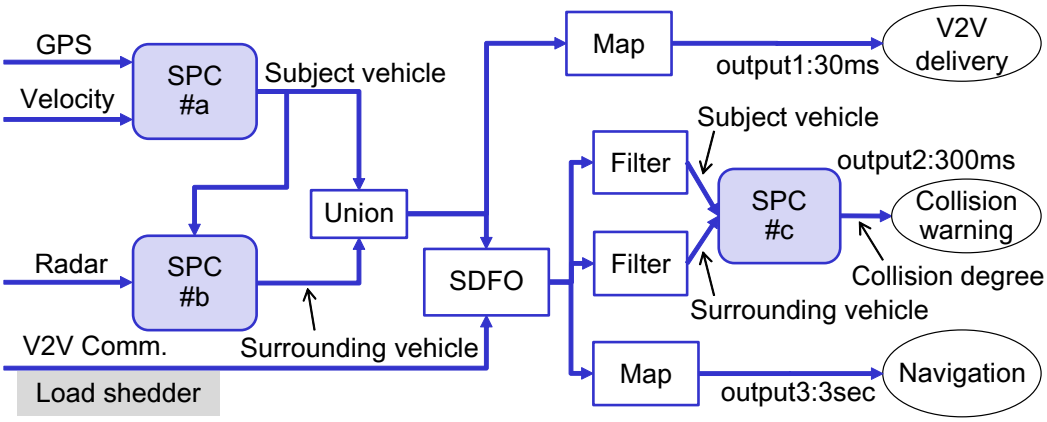


図 3.4: HLQ の例

図 3.3 では，SPC は周辺車両の位置・速度情報を出力する．レーダから得られる周辺車両の相対位置と相対速度を自車両の絶対位置と絶対速度をもとに絶対位置と相対速度に変換して出力する．レーダは，相対位置とそれらの信頼度（分散[1]）を属性として持つ．車両の走行データ（SubjectVehicle と SurroundingVehicle）は，データの発生源や対象車両を表す属性と，位置や速度とそれらの信頼度（分散）を属性として持つ．この SPC は，Join，Map という 2 つのオペレータから構成される．

3.4.3　HLQ の例

HLQ の例を図 3.4 に示す．SPC#b は，3.4.2 節の図 3.3 で説明した SPC 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，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処理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 [19] に合わせて 300 ミリ秒とした．

SPC#a では，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 [2]．SPC#b では，レーダから得られた周辺車両の相対位置/相対速度を自車の位置/速度と加算し，絶対位置/絶対速度に変換する．SPC#a と SPC#b の出力結果は Union オペレータによりマージされ，フォーマットを整形した後に output1 に出力される．またその結果と車々間通信から得られる車両情報を入力として，SDFO では，[39] 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．その結果は，2 つの Filter オペレータにより自車両と他車両の情報に分けられた後 SPC#c に入力され，そこで周辺車両と自車が衝突するまでの時間である Time To Collision（TTC）を計算する．SDFO の出力結果は Map オペレータでフォーマットを整形した後に output3 に出力される．車々間通信からの入力ストリームには高負荷時にデータを間引く load shedder を設定する．

3.4.4　AEDSMS の処理の流れ

AEDSMS の処理の流れは，図 3.2 のように 5 つのステップからなる．ステップ 1 から 4 までが，ソフトウェア開発時に静的にプリコンパイルされる処理である．ステージ 5 のみが，車載システム上におけるクエリ実行時の処理である．

[1]センサのノイズが正規分布に従うと仮定すると，この分散による信頼性の定義は，ETSI の定義する信頼区間に基づく信頼性に変換できる．

[2]位置や速度を含むタプルには，それらの分散も含まれる．

ステップ 1. SPC 展開

SPC-Library の SPC の定義を用いて HLQ の中の SPC を展開する．その結果，SPC がそれを構成するオペレータに置き換わる．このステップにより，クエリは SPC を含まずオペレータのみから構成される．

ステップ 2. オペレータ配置

オペレータを各ノードに配置する．オペレータに分解した単位で配置を決めることで，SPC の単位よりも柔軟に配置を決めることができる．また，従来のオペレータ配置方式とは異なり，ノードとネットワークの構成を組込みシステムの分野の設計探索でよく用いられるアーキテクチャグラフに変換する [36]．この処理は 3.4.5 節で後述する．その後，アーキテクチャグラフにオペレータを配置することで，3.2.4 節で述べた課題を解決する．これにより，オペレータ配置の既存方式を適用することが可能となる．

ステップ 3. タスク定義

LLQ からタスクを定義する．タスクとはスケジューリングの単位となるオペレータセットである．ここで end-to-end デッドラインから各タスクのデッドラインを導出する．この方法は 3.4.6 節で後述する．

ステップ 4. バイナリ生成

このステップは LLQ をソースコードに変換し，AEDSMS が特定の車載システムで動作するようにランタイムライブラリから必要最低限のモジュールを選択する．次に，コンパイルされたソースコードとアプリケーションプログラムをランタイムライブラリとリンクして EXQ を生成する．

ステップ 5. 車載システム上におけるクエリ実行

クエリ実行前にステップ 4 で生成された EXQ は該当するノードにインストールされる．各ノードではステップ 3 で設定されたデッドラインに基づき EDF でオペレータをスケジューリングする．各ノードにおける DSMS のスケジューリングは 4 章で後述する．これにより，分散ストリーム処理全体の end-to-end デッドラインを

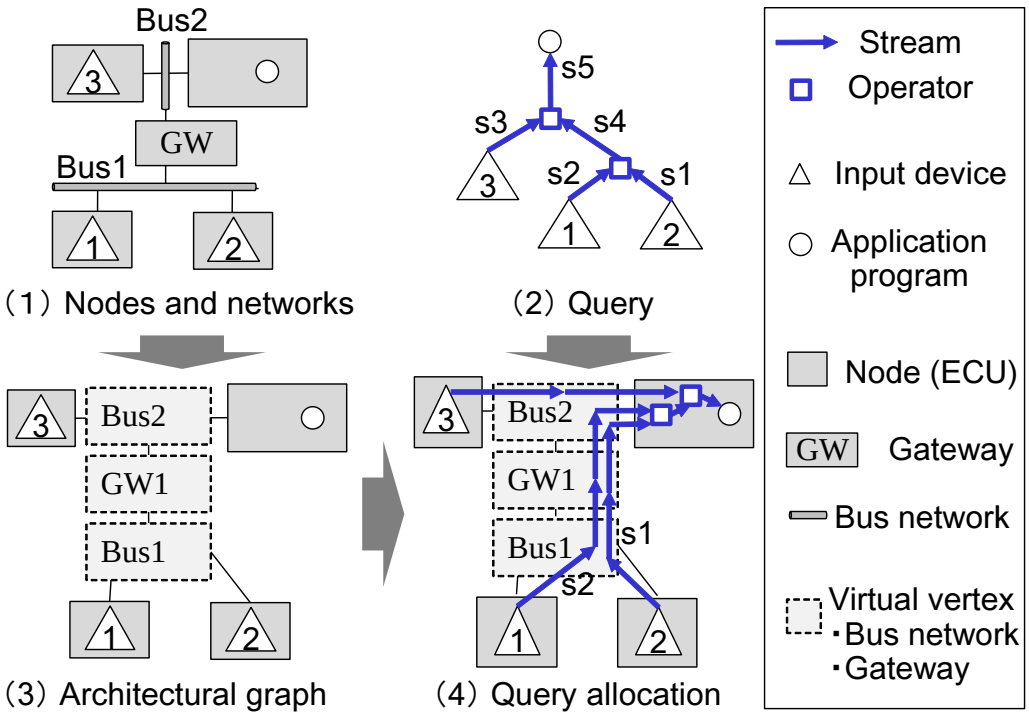


図 3.5: アーキテクチャグラフを用いたオペレータ配置

満たすリアルタイムスケジューリングを実現する．アプリケーションプログラムはコールバック関数を経由して出力ストリームのデータを利用できる．

3.4.5　アーキテクチャグラフを用いたオペレータ配置

図 3.5 ではアーキテクチャグラフを用いてどのようにオペレータを配置するかを説明する．アーキテクチャグラフでは頂点がノード，ネットワーク，ゲートウェイのいずれかである．ネットワークかゲートウェイによる頂点は仮想的な頂点としてオペレータ配置に用いられる．アーキテクチャグラフでは，ノードやゲートウェイがネットワークに接続している場合に（またその場合に限り），それらの頂点間に辺を持つ．

以下では，ノードとネットワークの構成が図 3.5(1) でありクエリが図 3.5(2) の場合で説明する．クエリは，オペレータを頂点，ストリームを辺とするグラフとみなせる．図 3.5(1) から生成されたアーキテクチャグラフを図 3.5(3) に示す．

アーキテクチャグラフに変換した後は，以下の制約を満たすようにオペレータを配置する．オペレータの配置には [27] の方法を用いる．

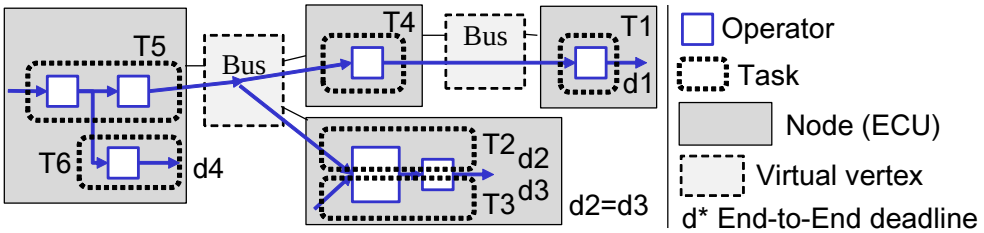


図 3.6: タスクとそのデッドラインの決定方法

- オペレータは,（ネットワークやゲートウェイを表す）仮想ノードには配置できない[3].
- ストリームは，頂点間に辺がある場合に限り異なるノード間に配置できる．
- 各ネットワークの使用量は，[27, 31] と同様にそのネットワークに対応する仮想ノードへ入力されるストリームのデータ量の総和として見積もる．例えば，Bus1 のネットワーク使用量は図 3.5(4) における 2 つのストリーム（s1, s2）の入力データ量の総和として見積もれる．

以上のようにアーキテクチャグラフを用いることで，異なるノード間を経由するストリームの経路を正しく表現できる．これにより，分散ストリーム処理の従来のオペレータ配置方式 [27, 31] を適用することができ，それらの従来方式と同様にネットワーク使用量を見積もれる．

3.4.6 DSMS のリアルタイムスケジューリング

DSMS に EDF スケジューリングを導入するには，各ノードでタスクを定義しそれに適切にデッドラインを設定する必要がある．AEDSMS は図 3.6 のようにオペレータ列をタスクとして定義する．タスク間に実行順序の依存関係がある場合，EDF*[13] を適用することでその依存関係を解消する．

EDF*では，先行するタスク τ の相対デッドライン d を，後続するタスク τ_i の計算時間を c_i として相対デッドラインを d_i を用いて以下のように計算する．

$$d = \min\{d_i - c_i; i = 1, 2, \cdots, n\}. \tag{3.1}$$

まず，後続するタスクが無い相対デッドラインは end-to-end デッドラインとして初期化される．次に，式 (3.1) を後続するタスクから先行するタスクへ再帰的に適

[3]ストリームは仮想ノードにも配置できる．

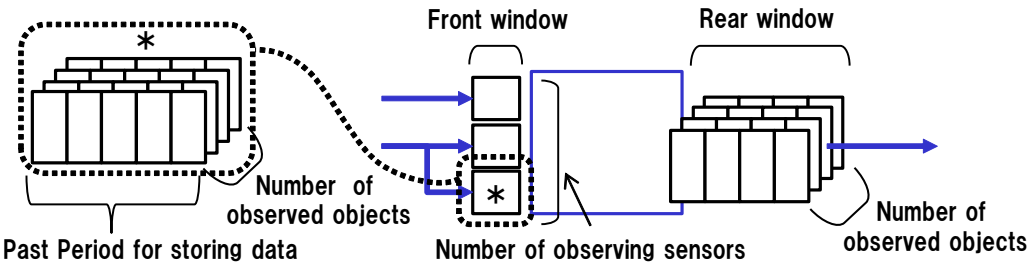


図 3.7: センサデータフュージョンオペレータ (SDFO) の概要

用することで，全てのタスクに適切に相対デッドラインを設定し，タスク間の実行順序の依存関係を解消する [13]．AEDSMS では，式 (3.1) を拡張して以下のようにオペレータの計算時間と同様に通信遅延を考慮する．

$$d = \min\{d_i - c_i - c_{\tau \to \tau_i}; i = 1, 2, \cdots, n\}, \tag{3.2}$$

ここで，$c_{\tau \to \tau_i}$ は τ と τ_i 間の通信遅延である．これにより，分散ストリーム処理の各ノードにタスクとそれらの相対デッドラインを割り当てることができる．

各ノード上でのスケジューリング方式は 4 章で後述する．また，オペレータ配置と各タスクの相対デッドラインをリアルタイム処理要求を満たすように同時に最適化する方法も提案している [47, 50]．

3.4.7　SDFO - センサデータフュージョンオペレータ

SDFO は，センサデータフュージョンのための特別なウインドウ構造を持つオペレータとして UDO をクラス継承して実装される．SDFO のウインドウ構成を図 3.7 に示す．3.3.4 節で前述したように，センサデータフュージョンでは過去の計算結果をフィードバックする場合が多い．そのため，SDFO は，入力を保持する前方ウインドウに加えて，出力を保持する後方ウインドウを持つ．一般に，観測している物体が同一である場合に限り，それらのセンサデータを融合できる．例えば，自車の周辺に車両 A と車両 B があった時に，車両 A の信頼性を向上する場合に，車両 A を観測するデータと融合することはできても，車両 B を観測するデータとは融合できない．そのため，後方ウインドウは，物体を区別するためグループごとに分かれており，各グループごとに過去一定時間のデータを保持し，SDFO の処理から利用される．

入力を保持する前方ウインドウに関しても，物体を区別するためグループごとにウインドウを分ける．センサデータフュージョンでは，融合するデータに多様性

が要求されるため，データの発生源が異なるデータ同士に限って一般には融合できる．例えば，同一のセンサから得られたデータ A と，それを加工して生成したデータ B とを融合することはできない．そのため，前方ウインドウは，グループに加えて発生源ごとにもウインドウを分ける．

基本的には，SDFO は全ての入力ストリームからデータが到着した時に，ウインドウに保持されたデータを融合する．しかし，遅延時間が保証されない車外から得られるデータを入力することもある．そのため，SDFO はタイムアウトの機能を持ち，最初のデータが到着してから指定時間以上経った場合にはウインドウに到着しているデータだけで処理する [4]．

SDFO はセンサデータフュージョンを容易に実現するためのフレームワークである．ユーザは，SDFO のクラスを継承して仮想関数を C++で上書きすることで具体的なセンサデータフュージョンのアルゴリズムを実装する．本研究では，運転支援で用いられる具体的なセンサデータフュージョンを 3 種類のカルマンフィルタとして実装した．実装した各アルゴリズムについては 3.5.2 節で後述する．

3.5　フィージビリティ評価

本節では，運転支援アプリケーションの具体的な事例を用いて，AEDSMS の実行時の性能を評価する．

3.5.1　評価方法

HLQ は，3.4.3 節で説明した図 3.4 を用いる．図 3.11（左）は本実験で用いる物理構成を表しており，図 3.11（右）は各ノードに搭載されるセンサや通信機器とアプリケーションプログラムを示す．各ノードにマッピングされた LLQ を図 3.11 に示す．車載ネットワークは 500kbps の CAN を想定し，その通信遅延は [15] の式 (1) を用いて算出した．車載システムの各ノードの評価には，カーナビゲーション相当のスペックを持つマシンとして，シングルプロセッサで，CPU:800MHz，メモリ:1024MB，OS:Linux (Fedora10) 32bit を用いた．

車外との通信を用いる車載システムでは，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 [19] 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

[4] もしセンサデータフュージョンを行うデータが無ければ，融合せずに出力する．

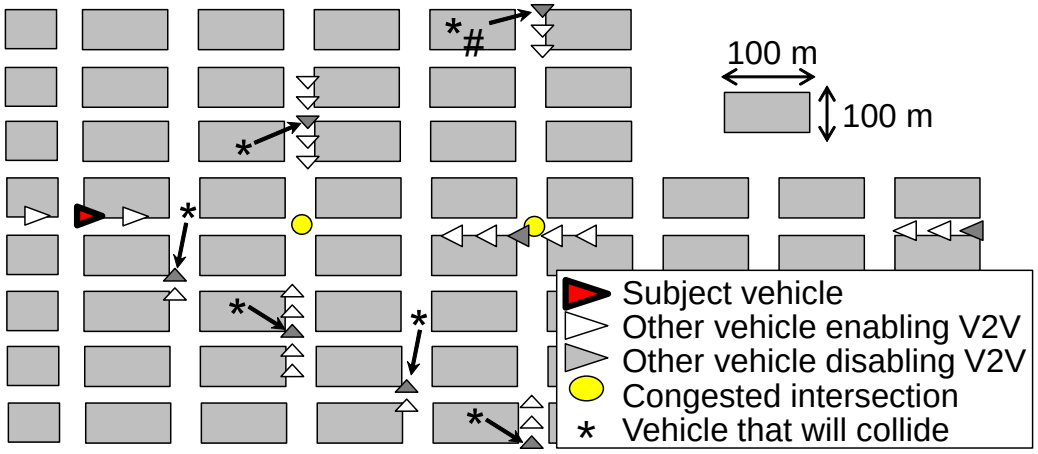


図 3.8: 本走行シナリオにおける車両の初期配置

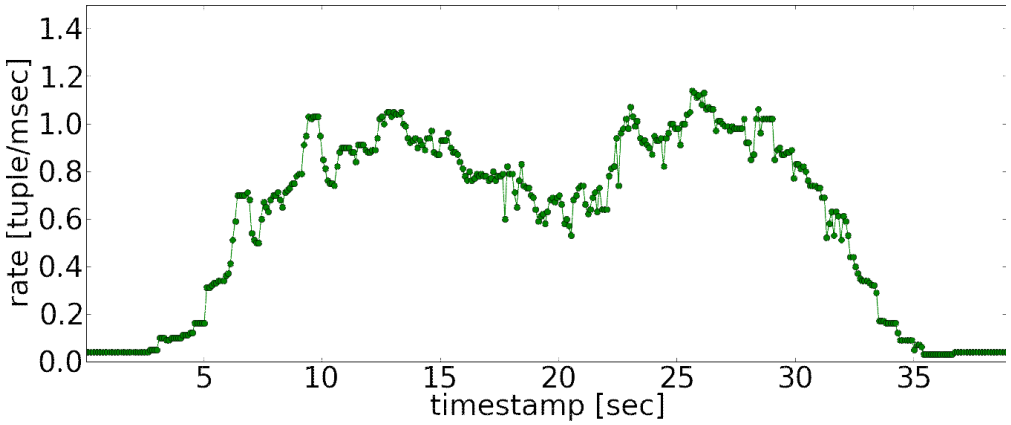
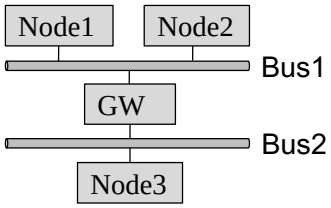


図 3.9: 本走行シナリオにおける車々間通信の入力データ量



Node (ECU)	Input devices	Application Programs
Node1	GPS, Velocity sensor	-
Node2	Radar	-
Node3	V2V communication	all

図 3.10: 物理構成

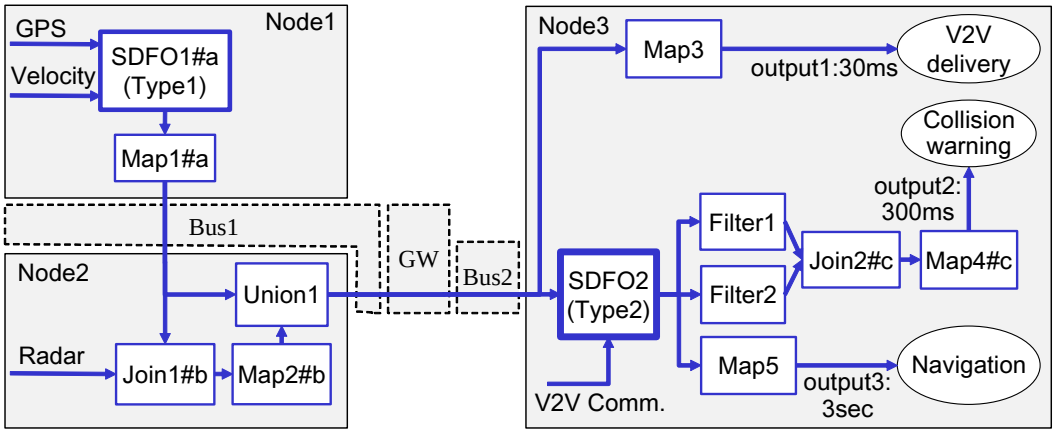


図 3.11: 物理構成にマッピングした LLQ

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[5] を用いて入力データを作成した．センサのノイズについては，Scenargie では模擬できないため正規分布に基づく人工的なノイズを付加した．各車両は，交差点間の距離が 100m の碁盤目の道路（マンハッタンモデル）に配置し，矢印の方向に時速 60km で直進した．

車々間通信は，100 ミリ秒周期で 200m 程度の範囲まで正しく受信できるように信号強度を設定した．また，レーダも同程度の距離まで正しく受信できるように信号強度を設定した．車々間通信の周波数は 700MHz としてレーダの周波数は 70GHz 以上を設定した．これにより，周波数の低い車々間通信の電波はレーダに比べて障害物に対して回折して届く．

車両の初期配置には図 3.8 を用いた．車々間通信器を搭載した車両と搭載していない車両がいるシナリオを想定する．Subject vehicle は，車々間通信器と AEDSMS を搭載した自車両を表す．他車両（Other vehicle enabling V2V）は，車々間通信器を搭載し自車両と車々間通信を行う．一方，他車両（Other vehicle disabling V2V）

[5]Scenargie: `https://www.spacetime-eng.com/`

表 3.1: センサフュージョン 1 回あたりの遅延時間

Type	Average	Standard deviation
1	1.84 ms	86.2 μs
2	11.0 ms	92.2 μs

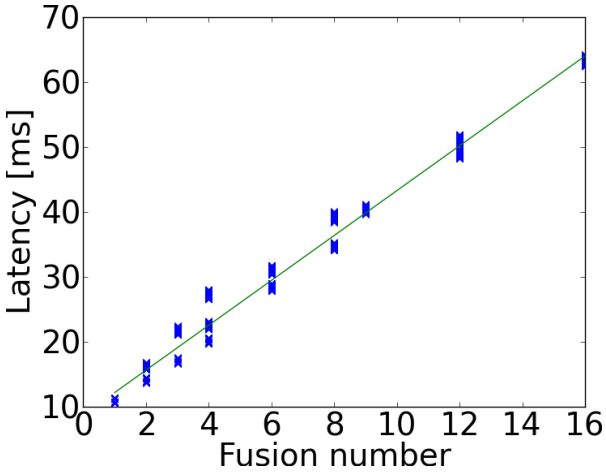


図 3.12: 車両情報を融合した回数と遅延時間の関係

は，車々間通信器を搭載していない．車両により混雑する 2 つの交差点では，自車両がその交差点に入ったときに車々間通信で受信する 1 秒あたりの車両の台数が約 1000 台分となり ETSI の定める最大車両台数になる．図 3.9 は，本シミュレーションで自車両が 1 秒あたりに受信する車両台数の推移を表しており，車両で混雑する交差点に自車両が近づいた時にそれぞれピークがあることが分かる．入力データ量が増加し処理待ちのデータが発生することで，その処理順序を決定するスケジューリング方法の違いによる効果の差が生じる．そのため，本シミュレーションの状況としては，スケジューリング方法による効果の差が明確に確認できるように，ETSI が定める最大車両台数の仕様に合うように車両の台数や配置を設定した．

3.5.2　SDFO の基本性能

3.5.1 で述べたクエリと走行シナリオで評価する前に，AEDSMS で新規に開発した特殊なオペレータである SDFO の単体性能を評価する．図 3.11 の LLQ では，以下の 3 種類のカルマンフィルタを SDFO で実装した [6]．

タイプ 1: 自車両に搭載したセンサによる状態推定 GPS と車速センサの観測値か

[6] 行列の計算には Eigen を利用した．

ら，車両の位置と車速の状態を推定する．[4] の方式をもとに実装した．このようなセンサデータフュージョンは，商用の自動車でもよく用いられている．

タイプ 2: 他車両の走行データの融合 [39] の方式をもとに，車両に搭載されたセンサまたは車々間通信から得られる車両の位置と速度を観測し，同一の車両同士を融合することで信頼性を高める．この場合，観測する車両台数に加えて車々間通信を行う車両台数も変化することから，SDFO のグループ数が変化するだけでなく，発生源の数も変化する．

各 SDFO のみのクエリの単一ノードにおける遅延時間を表 3.1 に示す．いずれの SDFO でも，センサデータフュージョンの回数が 1 回の場合の計算時間であり，各統計値は，一万個以上のサンプルを用いて計算されている．Type1 の遅延時間は，最悪ケースでも数ミリ秒のオーダーであるのに対して，それらを利用する図 3.4（または図 3.11）で要求されるデッドラインは最短でも 50 ミリ秒である．Type2 の SDFO の遅延時間は，最悪ケースでも数十ミリ秒のオーダーであるが，要求されるデッドラインは最短でも 300 ミリ秒である．以上の結果から，SDFO の融合回数が 1 回の場合における遅延時間は予測可能である．

図 3.12 では，Type2 の SDFO について，発生源とグループ数を変えて，遅延時間の変化をプロットした．また，融合回数を説明変数として遅延時間を予測した線形回帰の結果を図 3.12 の直線で示す．予測値と実測値の相対誤差における平均は，Typ2 で 1.64 ミリ秒であり十分に小さい．この測定結果から，SDFO の計算時間は，融合する回数から概ね予測できる．SDFO の入力データ量から融合する最大回数は見積もれるため，最悪ケースの計算時間は設計時に計算でき予測可能性は満たされる．

3.5.3　EDF スケジューラの導入効果

リアルタイム処理要求を定量的に評価するため，入力データ量を変化させたときの式 (3.3) のデッドラインミス率（DMR）を用いて性能を測る．

$$\frac{1}{\sum_{s\in\{s_1,\cdots,s_n\}}\alpha_s}\sum_{s\in\{s_1,\cdots,s_n\}}\alpha_s\frac{M_s}{N_s} \tag{3.3}$$

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処理要求の重要度を表し，

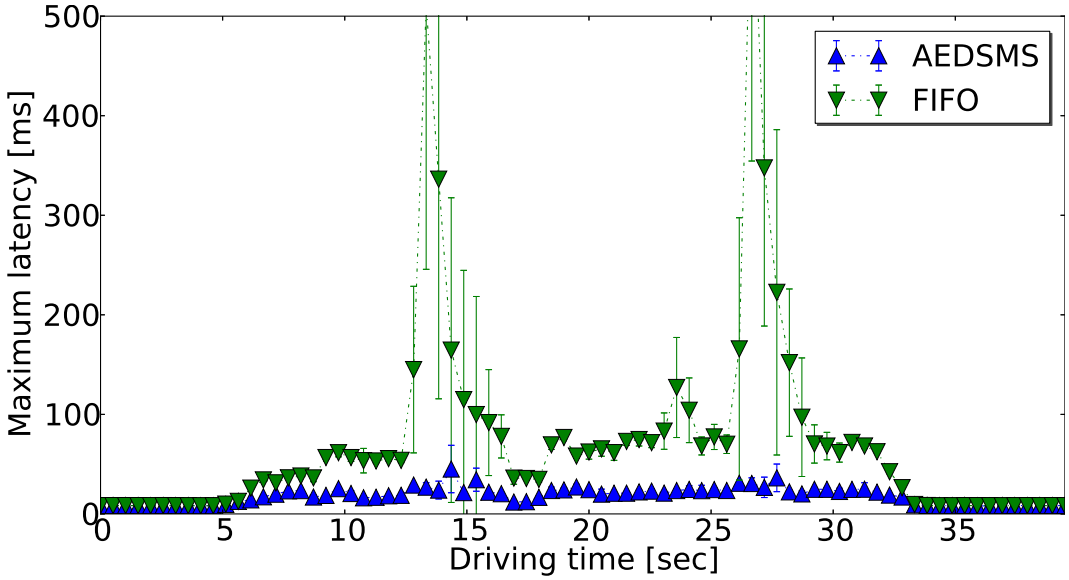


図 3.13: 走行時間における output1 の最大遅延時間の変化

そのデータを供給するアプリケーションプログラムに応じてユーザがクエリの登録時に決定する．

図 3.13 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 61%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．特に，車々間通信からの入力データ量は大きく SDFO2 の計算時間が長いため，図 3.13 に示すようにオペレータの実行順序の違いが最大遅延時間の違いに大きく影響した．

図 3.14 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．但し，式 (3.3) 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 output2 に対してはそれぞれ 1 とし，情報提供に用いる output3 については 0 とした．横軸は load shedder により制限する車々間通信から入力される 1 秒あたりの最大の入力データ量を表す．EDF スケジューリングを用いる提案方式（EDF）では，リアルタイム処理要求を満たして 1 秒あたり最大 800 タプルの入力データを処理できるが，リアルタイムスケジュー

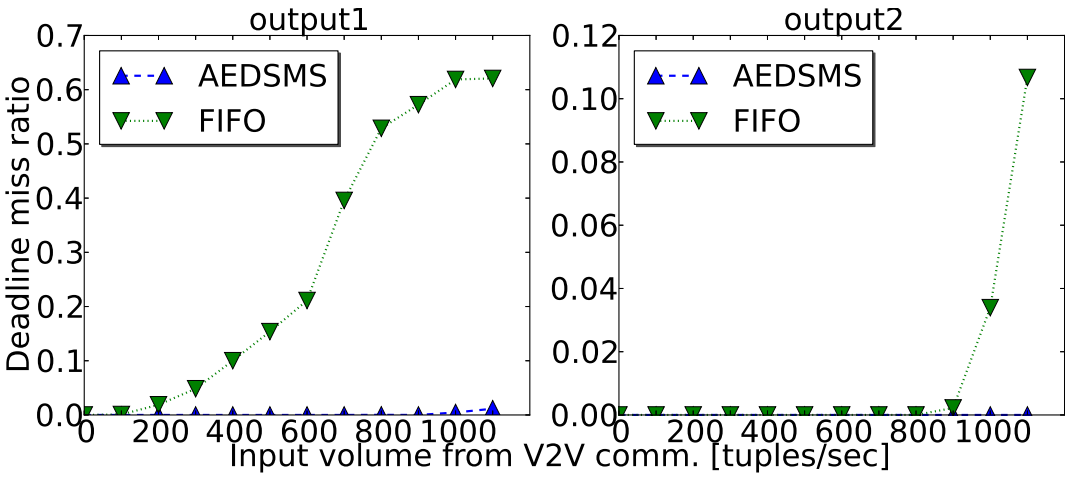


図 3.14: Output1 と output2 におけるデッドラインミス率

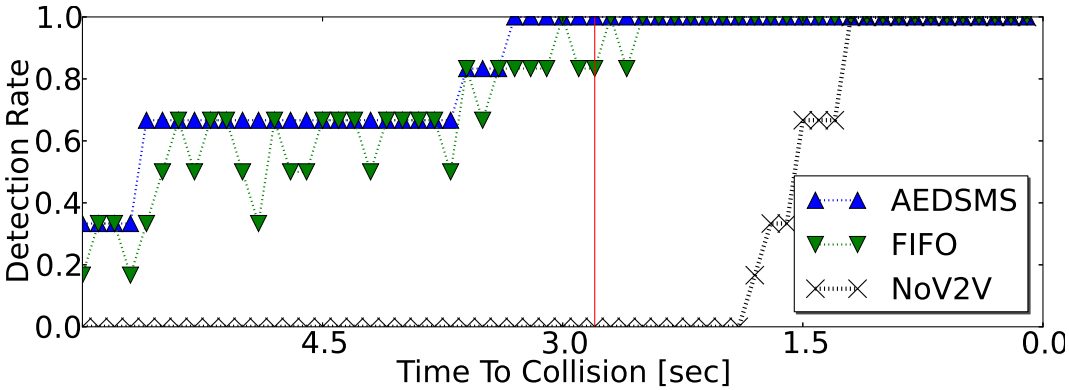


図 3.15: Output2 に出力される TTC に対する車両検出率の変化

リングを用いない従来方式（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処理要求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，車々間通信から入力できる 1 秒あたりの最大データ量を EDF と FIFO とでそれぞれ 800 個と 100 個とする．

3.5.4　車両衝突回避における効果

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，output2 から配信される周辺車両と自車両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．自車と衝突する可能性のある

車両は，図 3.8 の＊印の付いた 6 台の車両（車両＊）である．時速 60km で乾いた路面を走行する場合の停止距離から [40]，自車両の停止時間を 2.8 秒と算出する．自車両は，車両＊を最初に検出した時にブレーキをかけ始めるとして，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＊を最初に検出した時の 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 (3.4) で定義する．

$$M/N \tag{3.4}$$

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 3.15 は，ある試行において TTC を変化させた時，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 は車々間通信を全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，全ての車両＊を検出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，車載システムに搭載されたセンサのみを用いたストリーム処理がリアルタイム処理要求を満たして動作しているためである．図 3.15 から，FIFO では一時的に車両＊を検出できない時があり，EDF の方が検出率が良いことが分かる．これは，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 100 台から 800 台に増えたためである．

次に，図 3.15 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時の TTC（つまり TTC の最悪値）は，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 (3.4) の M は，EDF,FIFO,NoV2V で，それぞれ 0 台,50 台,600 台であった．そのため，式 (3.4) を適用すると，事故率は 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リアルタイムスケジューリングを導入することで車両の衝突事故を削減できることを示した．

3.5.5　センサデータフュージョンにおける効果

提案方式により，本ユースケースにおいて車両情報の位置精度がどの程度向上するかを確認する．図 3.16 は，output3 から配信される車両の走行情報の中で，あ

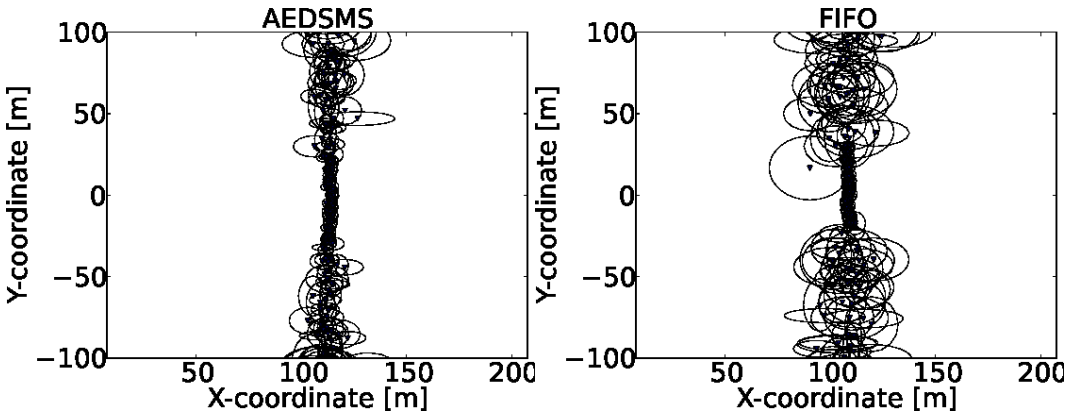


図 3.16: Output3 で得られる車両 # における位置精度の信頼区間（1 回の走行）

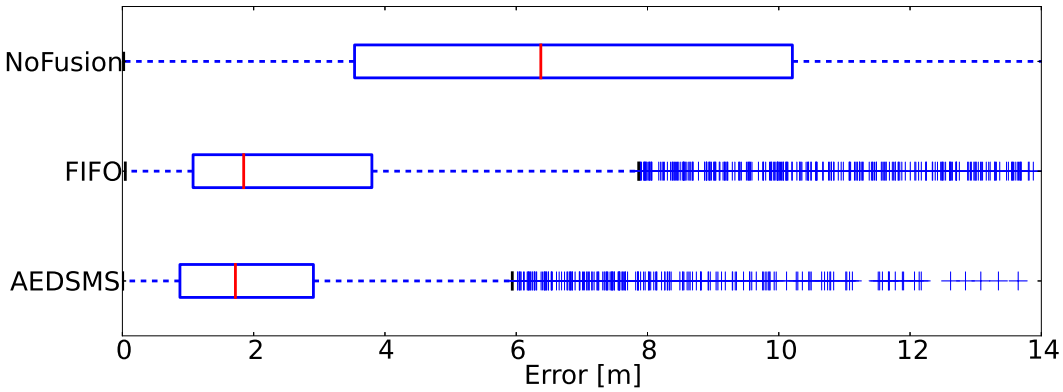


図 3.17: Output3 で得られる車両全体における位置精度の誤差

る一台の周辺車両（図 3.8 の車両＃）の位置情報をプロットした．図 3.16 の円内は，ETSI が走行データの信頼性として定めている左右対称の 95%の信頼区間である [17, 18]．図 3.16 により，提案方式が ETSI の定める信頼性と周辺車両の位置精度を向上することが，視覚的に分かる．これは EDF を用いることでデッドラインを満たしながら SDFO(Type2) でより多くの車両情報を融合できるためである．

図 3.17 は，output3 に出力される全車両の観測値と真値との誤差を箱髭図で表している．センサデータフュージョンをする場合としなかった場合とでは，誤差はそれぞれ 2.32 メートルと 7.30 メートルであり，センサデータフュージョンを行うことで平均 4.98 メートルの位置精度向上が確認された．それゆえ，AEDSMS においてセンサデータフュージョンの機能を提供することが車両の位置精度向上に有効であることが分かる．

EDF と FIFO とでは，誤差の平均はそれぞれ 2.32 メールと 3.37 メートルであり，EDF を適用することで 1.05 メートルの位置精度が向上した．図 3.17 の効果

も，EDF により車々間通信のデータを多く処理できるようになることで，SDFO2 でセンサフュージョンを行う車両台数が増加することが原因である．これにより，本シナリオでは AEDSMS のリアルタイムスケジューリングは位置精度を向上することにも有効であることが分かった．

3.6　考察

本節では，3.3 節で述べた技術課題をどのように解決したかをまとめる．

クエリ実行時の予測可能性 (C1)

図 3.2 に示すように，クエリをソフトウェア開発時にプリコンパイルするアーキテクチャによりこの課題を解決した．これにより，最悪ケースを見積もりそのケースで最適化することができる．加えて，標準のオペレータよりも計算時間がかかる SDFO の性能を評価し，3.5.2 節で SDFO の最悪ケースの計算時間を測定することで予測可能性を満たすことを確認した．

車載ネットワーク上の分散ストリーム処理 (C2)

従来の分散ストリーム処理システムでは，オペレータを各ノードに配置する際にバスやゲートウェイからなる車載ネットワーク上においてストリームのルートを正しく考慮してネットワーク使用量を見積もることができなかった．AEDSMS では，3.4.5 節で述べたように設計空間探索の分野で広く用いられるアーキテクチャグラフを介することでこれらの問題を解決した．

分散ストリーム処理のリアルタイムスケジューリング (C3)

ストリーム処理における従来のリアルタイムスケジューリング方式では，制限事項が多く特にオペレータの出力が分岐するようなクエリを持つ分散ストリーム処理に適用することが難しい．そのため，各ノードのオペレータ列に対してタスクを定義しそれらに相対デッドラインを適切に決定する方法を提案した．これにより，3.4.6 節で述べたように，分散ストリーム処理システムである AEDSMS に EDF に基づくリアルタイムスケジューリングを導入する．これにより，最悪遅延時間やデッドラインミス率（DMR）の削減に加えて，安全運転支援における車両衝突事故率の削減や車両情報の位置精度向上にも効果があることを 3.5 節の評価でそれぞれ確認した．

センサフュージョンによる信頼性の向上 (C4)

3.4.7 節で述べたように，AEDSMS ではセンサデータフュージョンオペレータ (SDFO) を用いることで信頼性を向上した．本研究では SDFO に 2 種類のカルマン

フィルタのアルゴリズムを実装し 3.5.2 節及び 3.5.5 節で車両情報の位置精度を向上することを確認した．

クエリ記述の再利用性 (C5)

AEDSMS では，SPC によるストリーム処理のモジュール化とオペレータ配置による位置透過性により，HLQ を用いた高レベルなクエリ記述を提供する．これにより，車種の変更などで物理構成が変わってもそのクエリ記述の修正を減らせる．

3.7　おわりに

自動運転など安全運転支援システムの普及にともない，分散ストリーム処理システムに基づく車載システムのデータを統合管理するソフトウェアプラットフォームはますます重要になっている．本研究では，車載ネットワークで繋がれた車載システムに導入するための分散ストリーム処理システムである AEDSMS を提案した．その際，車載システムのデータ処理要求から重要な 5 つの課題を明確化しそれらを解決した．フィージビリティ評価では，安全運転支援で重要な車両衝突警告を対象に，特に入力データ量が最も大きくなる車々間通信を用いた場合で評価した．車々間通信の入力データ量は ETSI の仕様に従ってシミュレーションにより最大化した．クエリ実行時における AEDSMS の性能を評価することで，AEDSMS を車載システムに導入する有効性を確認した．

第4章　単一ノード内のDSMSのリアルタイムスケジューリング

4.1　概要

車載システムのデータ処理では，リアルタイム処理要求を満たすことが重要である．一方，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 FIFO[12] や Path Capacity Strategy（PCS）[25]，スループットを向上する Min-Cost（MC）[10] 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，リアルタイム処理要求を満たすことを目的としておらず，この目的の達成には適切ではない．

本章では，このようなリアルタイム処理要求を満たすこと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良く用いられる 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場合や，オペレータの出力が分岐する場合，オペレータがタイムアウトを持つ場合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．

本章の構成は以下のとおりである．まず，4.2 節で本研究が解決する課題を述べ，4.3 節で関連研究を紹介する．次に，4.4 節で本章が想定するモデルを述べ，4.5 節で提案方式を示し，4.6 節でその評価を行う．最後に，4.7 節では課題に対する解決を考察し，4.8 節でまとめを述べる．

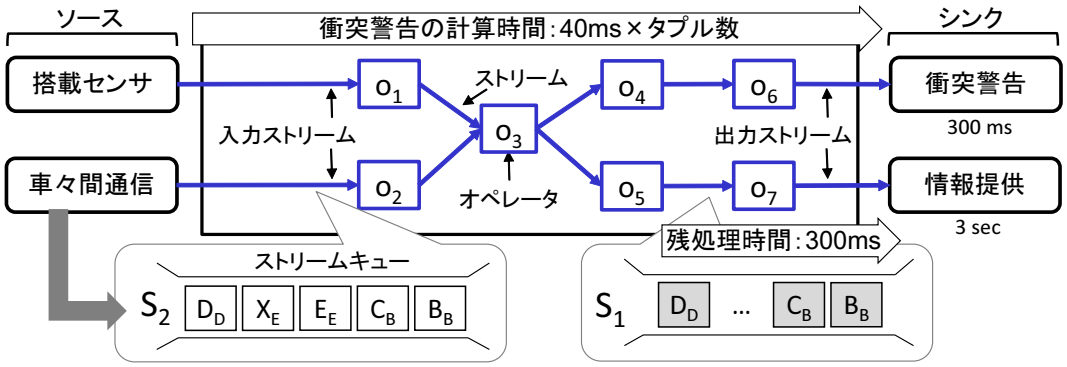


図 4.1: ユースケースシナリオにおける単純化されたストリーム処理

4.2 ストリーム処理のスケジューリングにおける課題

図 2.1 と同様に，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考える．2.1 節と同様に，ETSI の仕様 [19] 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．

図 4.1 は，図 2.1 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 の全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．

従来方式として例えば 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 の全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

2 章で述べたように，車載システムの安全運転支援におけるセンサ情報処理では，リアルタイム処理要求を満たすこと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 4.1 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場合やオペレータの出力が分岐する場合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処理要求を満たすには，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：オーバーロード時への対応

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 [19]．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 [17]，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一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 [3, 41]，リアルタイム処理要求を満たして車々間通信から得られるデータを全て処理することは難しい．そのため，リアルタイム処理要求を満たす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，リアルタイム処理要求を満た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ーヘッドの削減

スケジューリングにおけるオーバー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．リアルタイム処理要求を満たすためには，現在実行中の処理よりもデッドラインの早い処理（図 4.1 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．

4.3　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処理要求を満たすことを目的としておらず，4.2 節の I1 に対応できない．[6] 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 [5]．[10] 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ー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する 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．[12, 25] 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，4.2 節の I2 や I3 に対応することは難しい．[30] 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，4.2 節の 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．[45] では，EDF に基づき，タプルを入力する時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 デッドラインが全て同一であることが前提となり，4.2 節の 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直ちに結果を出力することを前提としており，4.2 節の I3 に対応できない．[10] 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

[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．

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，4.2 節の I2 に対応できない．

車載組込みシステム向け DSMS も幾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処理要求に対しては述べられていない．[38] 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[7] では，車両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，本研究の目的と類似する．しかしながら，これらでは 4.2 節の I1 に対応せず，本章の課題を解決できない．

4.4 本章で用いる記号

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．例えば，図 4.1 の $\mathrm{InTpl}(o_2)$ は $\{B_B, C_B, E_E, X_E, D_D\}$ となる．なお，オペレータ o の出次数を $\mathrm{ODeg}(o)$ と記述する．図 4.1 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 とパスで繋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 4.1 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．

4.5 ストリーム処理のリアルタイムスケジューリング

本節では，本章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，4.5.1 節でシステムの概要を説明したのち，基本的な方法を 4.5.2 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 4.5.3–4.5.4 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 4.5.5 節で述べる．

4.5.1 システムの概要

提案方式で想定するシステムは，3 章の AEDSMS を単一ノード上で動作させる場合に相当する．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

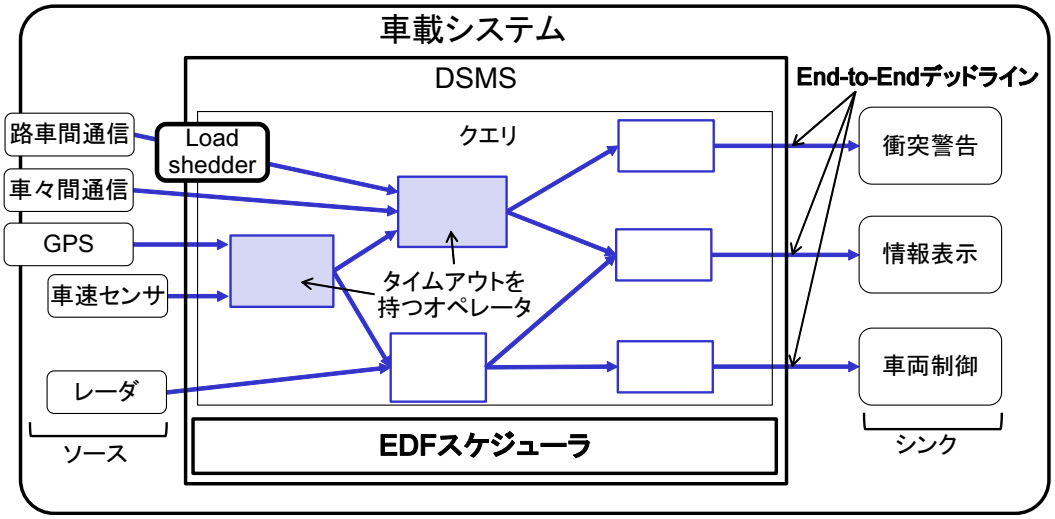


図 4.2: 提案方式によるシステムの概要

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 (2.1) 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処理要求を満たす．

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時に行う処理には，性能の高い計算機で十分な計算時間をかけられる．

4.5.2　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

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: 既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: $(o_1, o_2, \cdots, o_n) \leftarrow \mathrm{SucAllOp}(o)$ 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．

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 (2.1) 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方式である EDF*[13] の漸化式 (4.1) から導出される．

$$d_{p,o} = \min\{d_{p,o'} - c_{o'}; o' \in \mathrm{SucOp}(o)\} \tag{4.1}$$

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無い通常の 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 [13, 9]．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 (4.1) 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 が挿入される時に実行される．但し，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 (2.1) と (4.1) 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 [9] 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無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処理要求における指標が最も良くなる（小さくなる）ことが示されている [9]．なお，この定理ではスケジューリングのオーバー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，全ての $i = 1, 2, \cdots, n$ 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．

証明. 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，$L' := \max_{k=1,\cdots,n}\{\sum_{j=1,\cdots,k} c_{o_{i'_j}} - d_{p_{i'_k},o_{i'_k}}\}$ と置く [2]．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

[2] L または L' は，スケジューリング方式（A）または（B）の maximum lateness [9] である．

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． □

4.5.3 トレインスケジューリング

4.5.2 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，そのオーバーヘッドが大きい．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．4.5.4 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 (4.2) のように計算される．

$$D_{p,O} = d_{p,o_n} \tag{4.2}$$

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無くなった場合，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．但し，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）時に，中断したところから実行を再開する．

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ー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．但し，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，4.5.4 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．本章では，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時に実行される [3]．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，4.5.5 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みなすと，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 とパスで繋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無ければ (P, O) を実行（18 行目）する．

4.5.4　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，4.5.2 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．

[3]但し，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．

表 4.1: 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

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

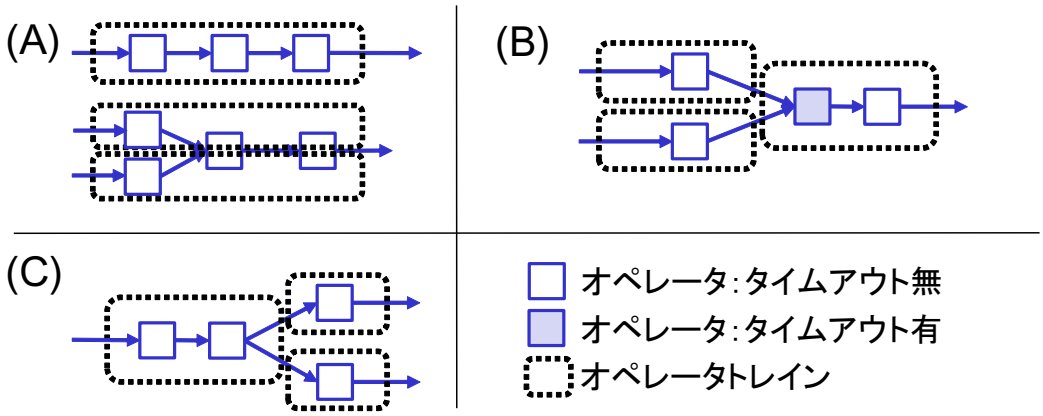


図 4.3: オペレータトレインのパターン

このオペレータトレインは，表 4.1 の条件を全て満たすオペレータ o と $o' \in \mathrm{PreOp}(o)$ を繋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 を繋げると，後述の定理 2 が適用されず，リアルタイム処理要求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良い．

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 4.3 に示す．(A) は最も基本的なパターンで全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

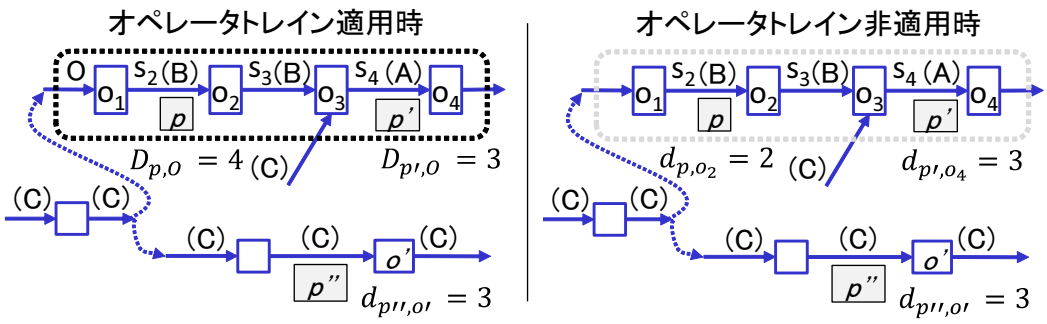


図 4.4: オペレータトレインの適用/非適用時における違い

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

この時，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．但し，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ーヘッドは増加しないことを前提とする．

証明のアイデア. まず，図 *4.4* 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 *4.4* の全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 *(4.1)* 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'' の処理は既に終わっている．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．

証明. 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ー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ーヘッドに変化が無いという前提で証明すれば十分である．

式 *(4.1)* と式 *(4.2)* 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時に，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 *(4.2)* から，オペレータトレイン適用時/非適用時に関わらず，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 で処理する時には，$(p'_1, o'_1), \cdots, (p'_m, o'_m)$ 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．

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． □

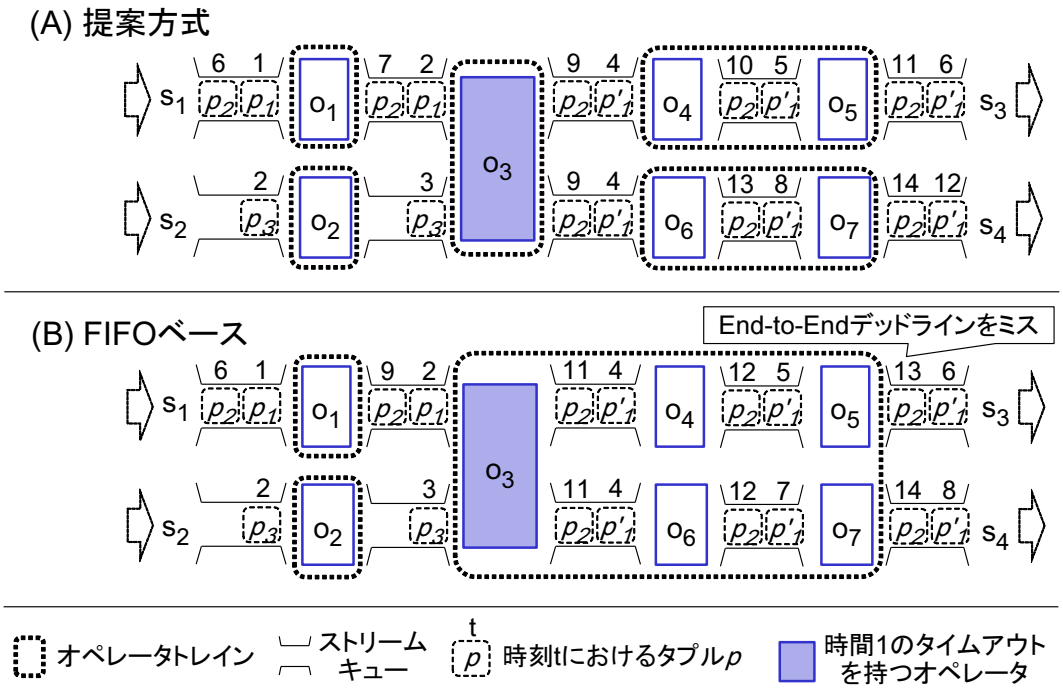


図 4.5: タイムアウトを含むスケジューリングの例

4.5.5　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 4.1 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全て揃った場合は，タイマーを直ちに解除する．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 4.5 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 4.5 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

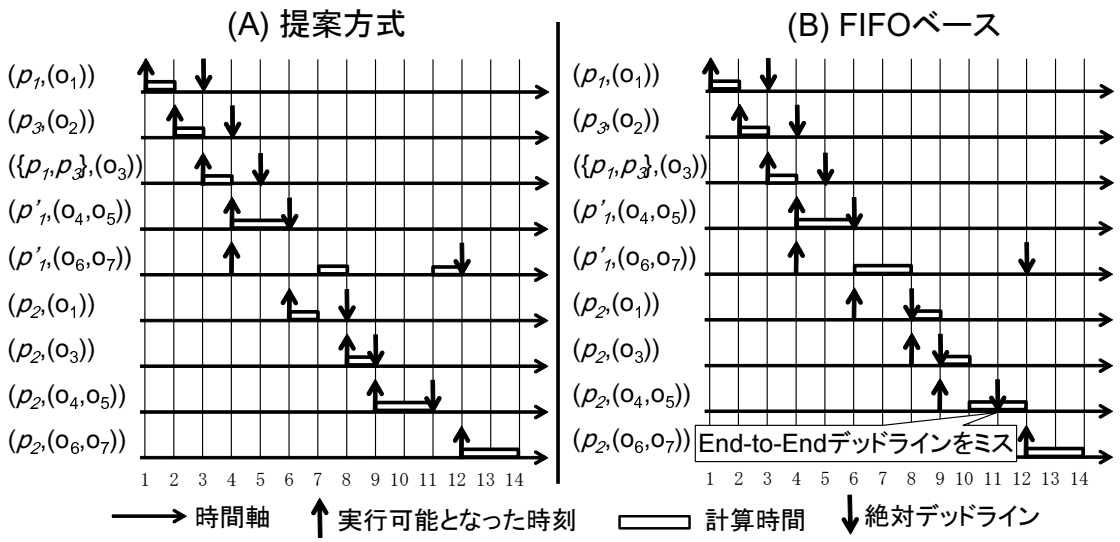


図 4.6: 時間軸に沿ったスケジューリングの比較

レータの計算時間は単純に全て 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 4.6 は，図 4.5 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 4.5(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，タイマーを解除して，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．

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 4.5(A) や図 4.6(A) のように全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 4.5(B) や図 4.6(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処理要求を満たすようにスケジューリングすることが可能となる．

4.6　実験評価

本節では，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，次に車載システム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．車載システムのノードとして，カーナビゲーション相当のスペックを持つマシンとして，シングルプロセッサで，CPU:800MHz，メモリ:1024MB，OS:Linux (Fedora10) 32bit を用いた．

4.6.1　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 4.7 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 (4.3) 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，

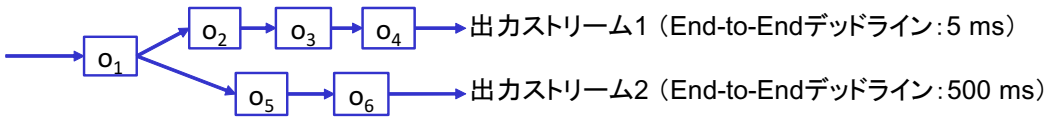


図 4.7: 基本性能評価に用いるクエリ

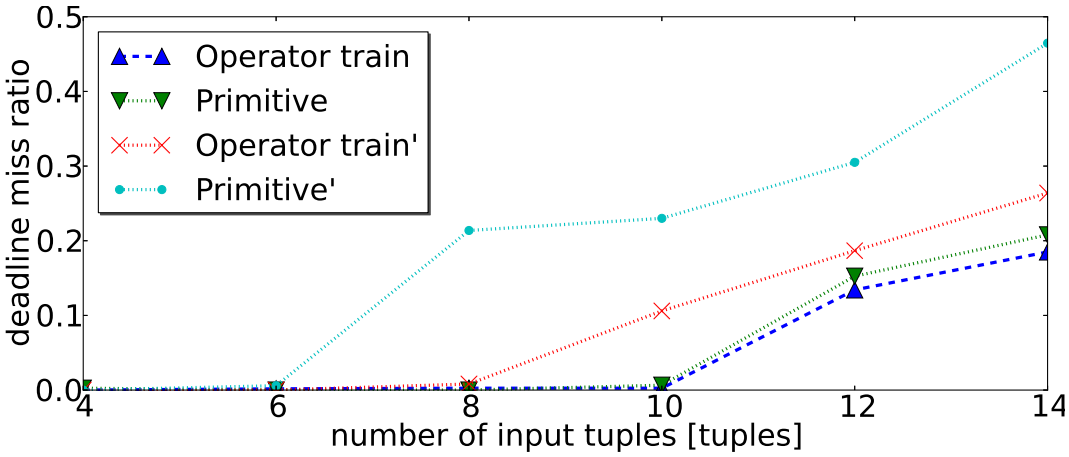


図 4.8: Input1 におけるオペレータトレインの効果

選択率は 1.0 である．以降の各実験では，少なくとも 100 回の試行を繰返した測定結果を用いる．

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．

オペレータトレイン

図 4.8 と図 4.9 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 4.8 のダッ

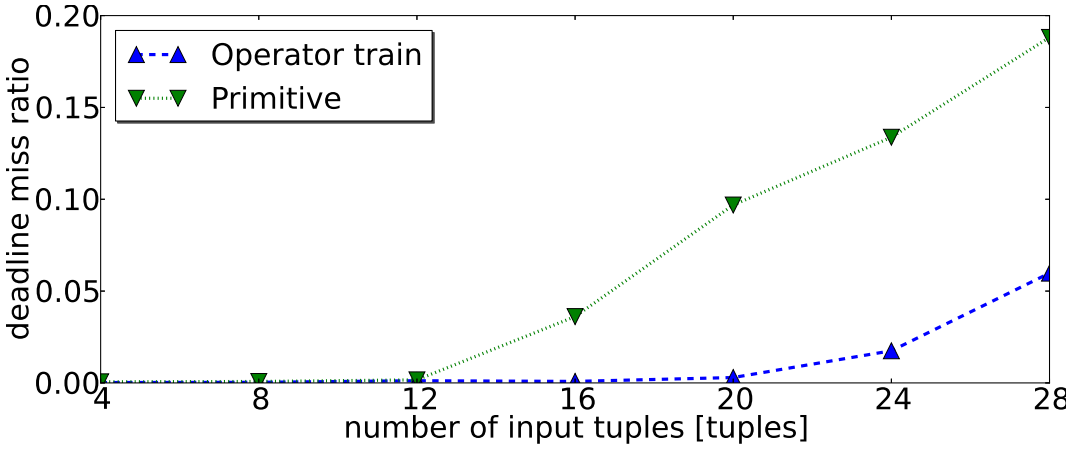


図 4.9: Input2 におけるオペレータトレインの効果

シュ記号の付いた方式ではタプルを全くバッチ化せず，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 4.7 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 4.8 と図 4.9 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ーヘッドが削減されたためである．また，図 4.8 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ー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．

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 個の時，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．この時，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ーヘッドは平均 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ー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．

リアルタイムスケジューリング

本節では，目的の異なるスケジューリング方式に対して，リアルタイム処理要求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

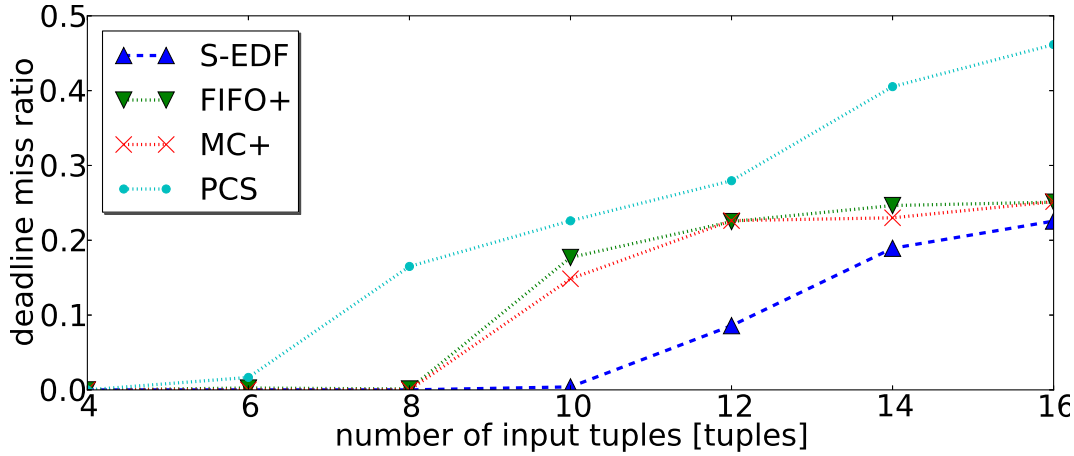


図 4.10: Input1 における従来方式との比較

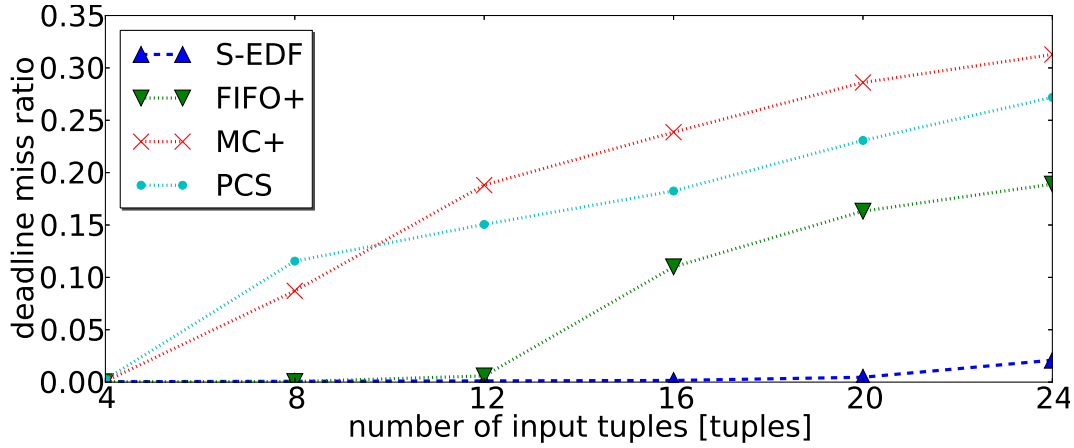


図 4.11: Input2 における従来方式との比較

述べる．

MC+ は，[10] 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．特に本章では，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繋がるパスから順に実行する．図 4.7 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，[12, 25] 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 4.7 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[5, 12] 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．特に本章では，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繋がるパスから順に実行する．図 4.7 のクエリでは，$o_1, o_2, \cdots, o_6$ となる．

図 4.10 と図 4.11 は，入力パターン input1 と input2 において，各方式のリアルタイム処理要求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+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．

4.6.2　アプリケーション性能評価

本節では，2 章で述べた車両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．

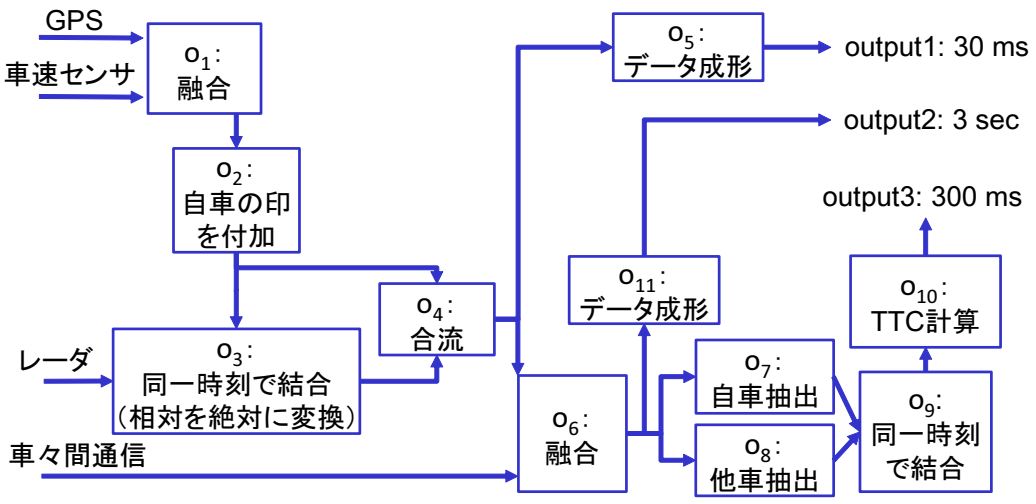


図 4.12: アプリケーション評価で用いるストリーム処理

表 4.2: 本評価におけるオペレータトレイン

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}	$d_8 = 3000 + t_p$

評価に用いるストリーム処理

図 4.12 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 4.2 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時における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，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処理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 [19] 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．

o_1 は，10 ミリ秒のタイムアウトを持ち，[4] 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 [4]．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，[39] 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．

評価方法

車外との通信を用いる車載システムでは，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 [19] 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[5] を用いて入力データを作成した．センサのノイズについては，Scenargie では模擬できないため正規分布に基づく人工的なノイズ

[4] 位置や速度を含むタプルには，それらの分散も含まれる．
[5] Scenargie: `https://www.spacetime-eng.com/`

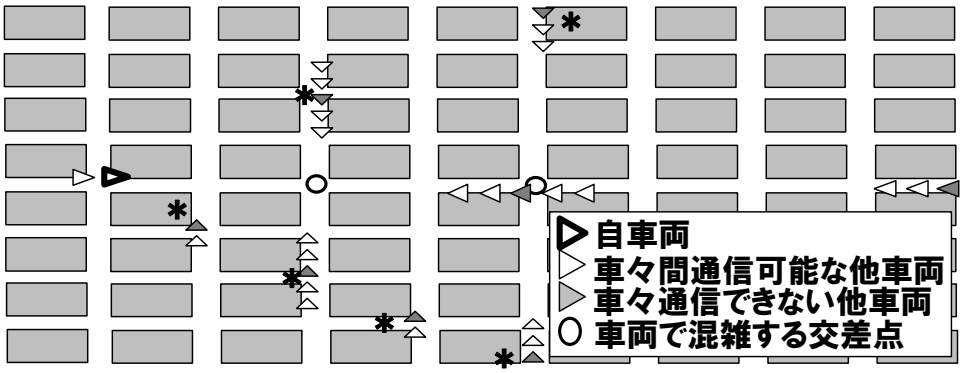


図 4.13: 車両の初期位置

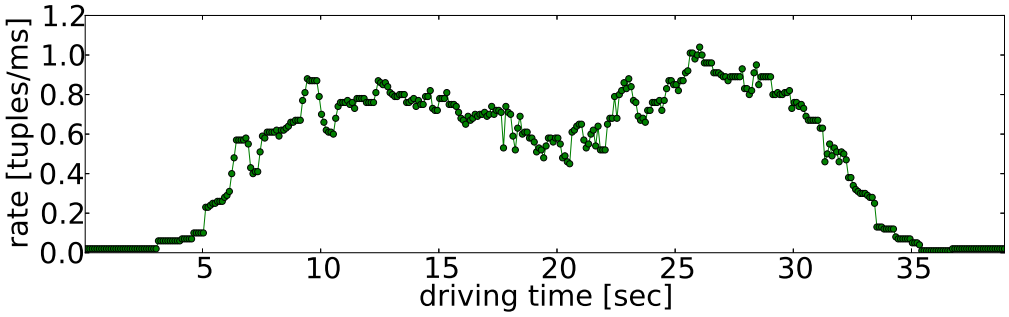


図 4.14: 車々間通信からの入力レートの推移

を付加した．各車両は，交差点間の距離が 100m の碁盤目の道路（マンハッタンモデル）に配置し，矢印の方向に時速 60km で直進した．

車々間通信は，100 ミリ秒周期で 200m 程度の範囲まで正しく受信できるように，信号強度を設定した．また，レーダも同程度の距離まで正しく受信できるように信号強度を設定した．車々間通信の周波数は 700MHz としてレーダの周波数は 70GHz 以上を設定した．これにより，周波数の低い車々間通信の電波はレーダに比べて障害物に対して回折して届く．

車両の初期配置には図 4.13 を用いた．車々間通信器を搭載した車両と搭載していない車両がいるシナリオを想定する．自車両では，車々間通信器と DSMS を搭載し提案方式によりストリーム処理のスケジューリングを行う．車々間通信可能な他車両は，車々間通信器を搭載し自車両と車々間通信を行う．車々間通信できない他車両は，車々間通信器を搭載していない．車両により混雑する 2 つの交差点では，自車両がその交差点に入ったときに車々間通信で受信する 1 秒あたりの車両の台数が約 1000 台分となり ETSI の定める最大車両台数になる．図 4.14 は，本

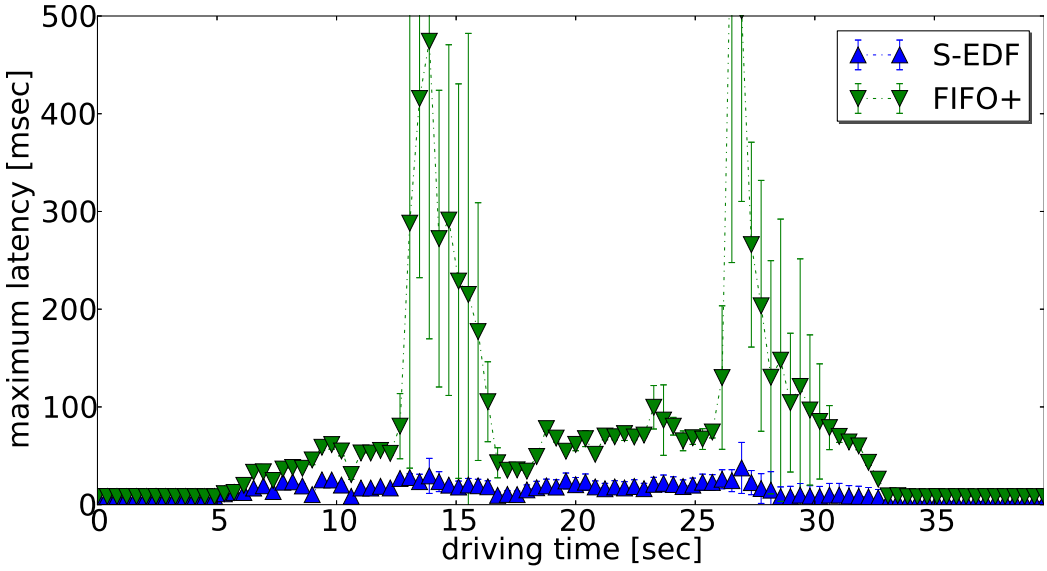


図 4.15: 走行時間に対する output1 の最大遅延時間の変化

シミュレーションで自車両が 1 秒あたりに受信する車両台数の推移を表しており，車両で混雑する交差点に自車両が近づいた時にそれぞれピークがあることが分かる． 入力データ量が増加し処理待ちのデータが発生することで，その処理順序を決定するスケジューリング方法の違いによる効果の差が生じる．そのため，本シミュレーションの状況としては，スケジューリング方法による効果の差が明確に確認できるように，ETSI が定める最大車両台数の仕様に合うように車両の台数や配置を設定した．

4.6.1 節の評価から，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 4.5(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．

リアルタイム処理要求

リアルタイム処理要求を定量的に評価するため，入力データ量を変化させたときの式 (4.3) のデッドラインミス率（DMR）を用いて性能を測る．

$$\frac{1}{\sum_{s\in\{s_1,\cdots,s_n\}}\alpha_s}\sum_{s\in\{s_1,\cdots,s_n\}}\alpha_s\frac{M_s}{N_s} \tag{4.3}$$

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処理要求の重要度を表し，

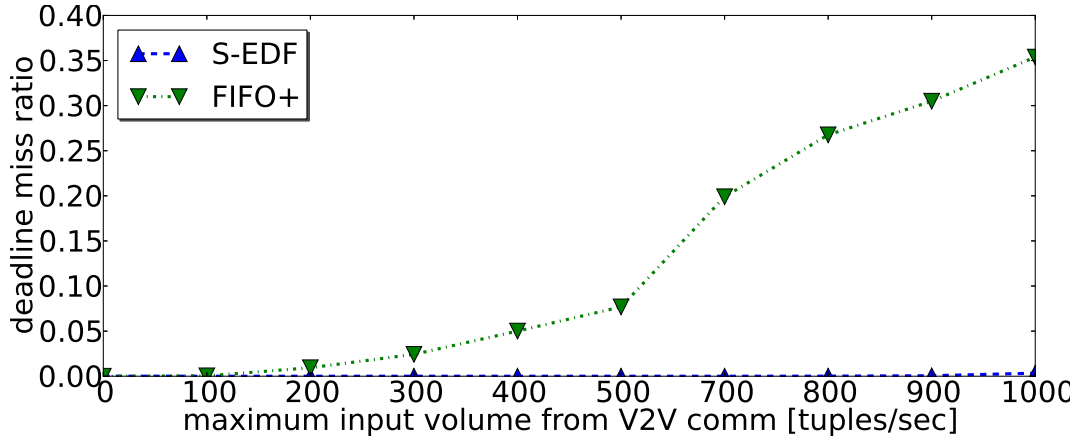


図 4.16: 本アプリケーションシナリオにおけるデッドラインミス率

そのデータを供給するアプリケーションプログラムに応じてユーザがクエリの登録時に決定する．

本アプリケーションシナリオにおいて，DMR を用いた評価の前に提案方式の直接的な効果を確認する．図 4.15 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 4.16 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．但し，式 (4.3) 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 4.16 のように，提案方式ではデッドラインミスを大きく削減できる．EDF では，リアルタイム処理要求を満たして 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，リアルタイム処理要求を満たし

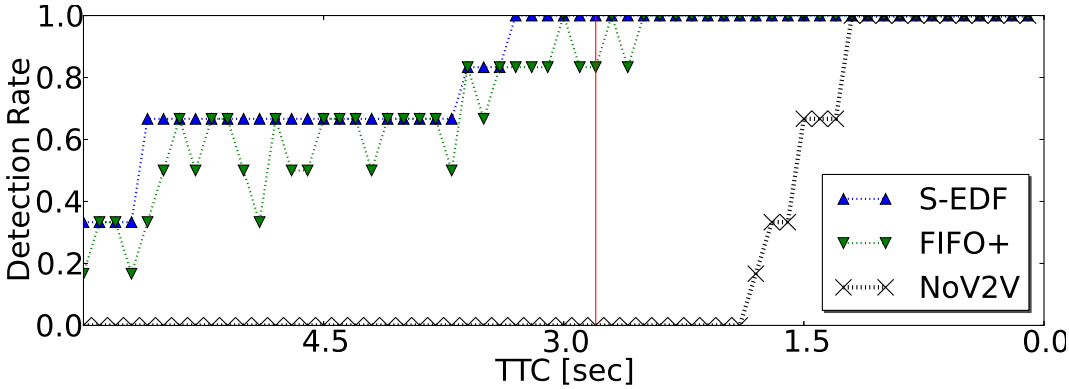


図 4.17: TTC に対する車両検出率の変化

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．

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．自車と衝突する可能性のある車両は，図 4.13 の＊印の付いた 6 台の車両（車両＊）である．時速 60km で乾いた路面を走行する場合の停止距離から [40]，自車両の停止時間を 2.8 秒と算出する．自車両は，車両＊を最初に検出した時にブレーキをかけ始めるとして，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＊を最初に検出した時の 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 (4.4) で定義する．

$$M/N \tag{4.4}$$

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 4.17 は，ある試行において TTC を変化させた時，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 は車々間通信を全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，全ての車両＊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処理要求を満たして動作しているためである．図 4.17 から，FIFO+ では一時的に車両＊を検出できない時があり，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 100 台から 800 台に増えたためである．

次に，図 4.17 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時の 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 (4.4) 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 (4.4) 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．

4.7　考察

本節では，4.2 節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

4.5 節の提案方式により本項目に対応した．4.6 節の評価では，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，リアルタイム処理要求を満たすことを目的とす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

4.5 節の提案方式は，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 4.7 と図 4.12）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，4.6 節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

4.5.5 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 4.12 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，4.6.2 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：オーバーロード時への対応

4.5.1 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．4.6.2 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 [19] 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ーヘッドの削減

4.5.4 節で述べたオペレータトレインの構成方法により，リアルタイム処理要求を満たすことを保証してオーバーヘッドを削減する方法を示した．また，4.5.3 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．4.6.1 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処理要求における性能改善を確認した．

4.8　おわりに

車載システム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処理要求を満たすため，本章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処理要求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単一ノードのストリーム処理として実現し評価した．その結果，本走行シナリオにおいて，リアルタイム処理要求に対して良好な 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，リアルタイム処理要求を満た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．

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: 既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: $(O_1, O_2, \cdots, O_n) \leftarrow \mathrm{SucAllTr}(O)$ 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 $\mathrm{PreOp}(o)$ が表 4.1 の条件を全て満たす **then**
5: **for all** $o' \in \mathrm{PreOp}(o)$ **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

第5章　単一ノード内のout-of-orderなデータストリームへの拡張

5.1　概要

4章とは異なり，本章ではセンサからデータが読み込まれる時刻とDSMSにそのデータがタプルとして入力される時刻を区別して扱う．2章で述べたように，車々間通信から得られるデータは100ミリ秒周期で配信されるため，ある車両のセンサからデータが読み込まれて別の車両が車々間通信でそのメッセージを受信するまで，通信遅延を無視したとしても最大100ミリ秒まで遅延時間に違いが生じる．一方，車両衝突警告のend-to-endデッドラインが300ミリ秒であることを考えるとこの遅延の差は無視できず，車々間通信から得られる入力データはデータ発生順に並んでいないout-of-orderなデータである．そのため，EDFでデータストリームを処理する場合，同じストリームキューにおいて前に挿入されたタプルよりも後から挿入されたタプルを先に処理しなければならない場合（preemptableなストリーム）が発生しうる．そのため，ストリームキューは従来のようにFIFOキューでは実現できず優先度キューが必要になる．

ストリーム処理における従来のスケジューリング方式ではpreemptableなストリームを考慮していない．優先度キューを用いずFIFOキューを用いた場合，デッドラインをミスする可能性がある．データストリームではデータは高頻度に流れるため，ストリームキューへのアクセス頻度も高い．その結果，優先度キューを用いた場合，キューの中を捜査する探索のコストがかかることでオーバーヘッドが増大するという問題がある．

本研究では，入力データの到着がout-of-orderな場合におけるストリーム処理の効率的なEDFスケジューリング方式（EDF scheduling for Preemptable data Streams; EDF-PStream）を示す．4章で示したオペレータトレインによるスケジューリング方法を用いることで，オペレータトレインを構成するストリームキューでは優先度キューの探索が不要なことを示す．ストリーム処理におけるリアルタイムスケ

ジューリングを含む従来のスケジューリング方式と比べて，提案方式がオーバーヘッドを削減することでデッドラインミス率を削減し，デッドライン処理要求を満たすことを確認する．

本章の構成は以下のとおりである．まず，5.2 節で関連研究を紹介し，5.3 節で本章の問題設定を述べる．次に，5.4 節で提案方式を示し，5.5 節でその評価を行う．最後に，5.6 節でまとめを述べる．

5.2　関連研究

ストリーム処理におけるリアルタイムスケジューリング方式はリアルタイムシステムとデータベースの分野で研究されている．[30] は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，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．[32, 37, 44, 45] は単一ノード上のストリーム処理の EDF スケジューリング方式を提案した．特に [32] では，マルチクエリと出力ストリームごとに異なる end-to-end デッドラインに対応した．しかし，いずれの方式においても out-of-order な入力データを考慮しておらず，そのようなデータ処理要求を持つデータストリームを効率的に処理できない．

5.3　問題設定

Out-of-order な入力データに対する EDF スケジューリングを扱う場合，センサからデータを読み込んだ時刻と車外との通信を経由して DSMS にデータが入力された時刻を区別する必要がある．そのため，センサからデータを読み込んだ時点のタイムスタンプ (sensor timestamp; sts) と DSMS にデータが入力された時点のタイムスタンプ (tuple timestamp; tts) を区別する．式 (2.1) のタイムスタンプ t_p はセンサからデータを読み込んだ時刻のため sts に相当する．

sts と tts を区別しない場合（つまり同一の場合），同一のストリームキューにおいて前に挿入されたタプルのタイムスタンプは後から挿入されるタプルのタイムスタンプより遅くなることはない．その結果，式 (2.1) より同一のストリームキューでは前に挿入されたタプルの方が必ず後に挿入されたタプルよりもデッドラインが早くなるので，ストリームキューは FIFO で実現することができた．しかしながら，sts と tts が異なる本研究では，同一のストリームキューに後で挿入されたタ

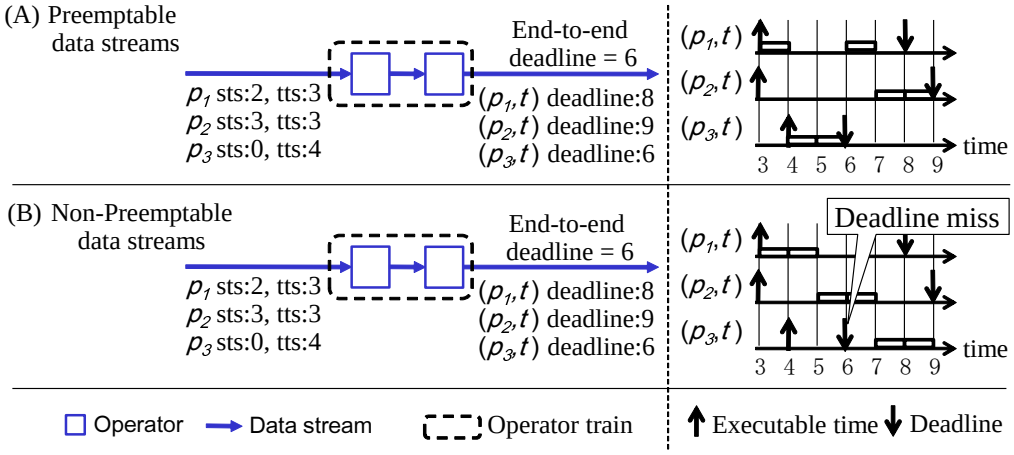


図 5.1: ストリームを preemptable にする場合としない場合における比較

プルの方が前に挿入されたタプルよりもデッドラインが早くなる場合が起こりうる．その結果，１つのデータストリームの中で後から挿入されたタプルが先に挿入されたタプルを追い越して処理される (preemptable stream) が発生し，従来のようにストリームキューを FIFO で実現できない．

本研究のストリームキューは優先度キューでなければならないが，優先度キューの実現には基本的に以下 (1)(2) のような方法が考えられる．(1) 各優先度に対応する複数の FIFO キューを用いる．(2) 優先度順に並んだリストとして実現し，データ挿入時にその挿入箇所を捜査する．本研究では，デッドラインが優先度のため優先度の数は多い．そのため (2) の方法で実現する．各ストリームキューでは優先度の順に並んでいるため，デキューの際はキューの先頭からタプルを取り出せばよい．

図 5.1 の単純な例を用いて，out-of-order な入力データに対して，従来方式のように FIFO キューではなく本研究のように優先度キューを用いることがリアルタイム処理要求を満たすことに必要であることを説明する．図 5.1 の (A) は優先度キューを用いた場合であり (B) は FIFO キューを用いた場合である．図 5.1 のオペレータトレインは 4 章で提案した Algorithm. 3 で構成したものである．各オペレータの計算時間は１であり，出力ストリームの end-to-end デッドラインは 6 とする．タプル p_1, p_2, p_3 が入力データとして時刻 3,3,4 に入力ストリームにそれぞれ挿入される．一方，各入力データがセンサから読みこまれた時刻 $\mathrm{sts}_{p_1}, \mathrm{sts}_{p_2}, \mathrm{sts}_{p_3}$ はそれぞれ $2, 3, 0$ である．図 5.1(B) のように，FIFO キューを用いた non-preemptable なストリームでは (p_3, t) が end-to-end デッドラインの 6 をミスしてしまう．一方，優

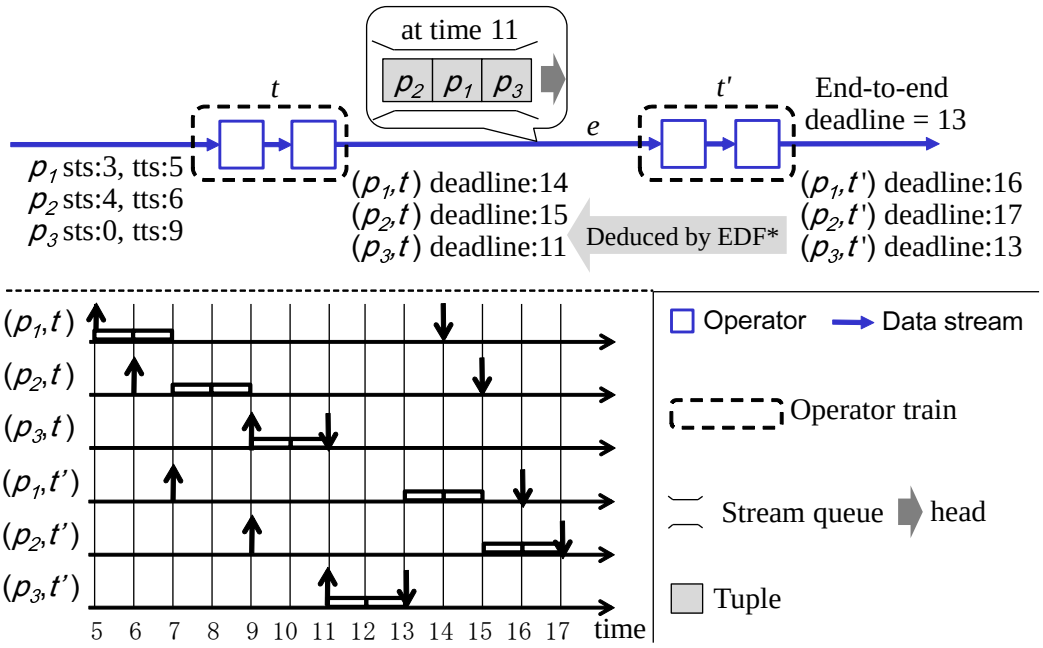


図 5.2: オペレータトレイン間における preemptable なストリームの例

先度キューを用いた preemptable なストリームでは $(p_1, t), (p_2, t), (p_3, t)$ はいずれも end-to-end デッドラインを満たす．このように，out-of-order な入力データを持つデータストリーム処理でリアルタイム処理要求を満たす場合には優先度キューを用いて preemptable なストリームに対応する必要がある．

図 5.2 の例を用いて，2 つのオペレータトレイン t, t' の間のストリーム（s_3）でストリームキューへのタプル挿入時に優先度キューの走査が必要であることを説明する．各オペレータの計算時間は 1 とし end-to-end デッドラインは 13 とする．タプル p_1, p_2, p_3 が入力データとして時刻 5,6,9 に入力ストリームにそれぞれ挿入される．$(p_1, t), (p_2, t), (p_3, t)$ のデッドラインは 4 章で説明したように EDF*[13] の漸化式 (4.1) から導出される．図 5.2 のように，$(p_1, t), (p_2, t), (p_3, t)$ の処理が完了する時刻はそれぞれ $7, 9, 11$ である．しかし各入力データがセンサから読みこまれた時刻 $\mathrm{sts}_{p_1}, \mathrm{sts}_{p_2}, \mathrm{sts}_{p_3}$ はそれぞれ $3, 4, 0$ である．その結果，ストリーム s_3 の優先度キューには，時刻 9 ではタプル p_2 がキューの末尾に挿入され，時刻 11 には p_3 がキューの先頭に挿入される．このように，オペレータトレインの間のストリームには out-of-order にタプルが挿入されるため優先度キューの走査が必要となる．

ストリーム処理の従来研究では，センサからデータが読み込まれる時刻 sts とタプルが DSMS に挿入される時刻 tts は等しいと暗黙的に仮定していたため，データストリームは preemptable ではなかった．しかし，一般的に他のノードからセンサ

データを受信する場合，通信遅延は発生する．2.2 節で述べたように車載システムにおいても，車々間通信を用いて車外からデータを受信する場合は sts と tts に乖離が発生し out-of-order な入力データの到着を無視できない．

5.4　オペレータトレインによるストリーム処理の効率化

本節では優先度キューの走査が必要な場合を明確にし，ストリーム処理にリアルタイムスケジューリングを効率的に適用する方法を述べる．5.3 節で述べたように，out-of-order な入力データを持つデータストリーム処理では preemptable なストリームとそれを実現するための優先度キューが必要となる．ストリーム処理ではタプルはストリームキューから頻繁に出し入れされる．そのため，タプルをキューに挿入する位置を頻繁に走査することはスケジューリングのオーバーヘッドに繋がる．そこで定理 3 を導出する．定理 3 により，オペレータトレインを構成するオペレータ間のストリーム（内部ストリーム）ではそのストリームキューの先頭に必ずタプルが挿入されることが示される．

定理 3. *Out-of-order* な入力データを考慮したストリーム処理の *EDF* スケジューリングにおいて，任意のオペレータトレイン t の任意の内部ストリームではタプルは必ず優先度キューの先頭に挿入される．

証明. 内部ストリームの優先度キューにタプルが既に存在しなければ定理は明らかに成り立つ．そのキューの中にタプルが既に存在する場合は以下のように背理法で証明できる．つまり，既にそのキューの先頭にタプル p があり新しく挿入されるタプル p' が先頭に挿入されなかった場合を考えて矛盾を導く．*5.3* 節で述べたように優先度キューにはデッドラインの早い順にタプルが挿入されるため，タプル p のデッドラインは p' より早いか同じである．一方，*EDF* スケジューリングのもとで先に処理された (p, t) を追い越して (p', t) が処理されていることから，タプル p' のデッドラインは p より早いはずである．これにより矛盾が導かれた．

優先度キューにおけるタプル挿入時の走査が必要か否かは以下のようにまとめられる．

系 1. *Out-of-order* な入力データを考慮したストリーム処理の *EDF* スケジューリングにおいて，オペレータトレインの内部ストリームであればタプル挿入時の走査は省略できるが *(case. 1)*，その他のストリームであればタプル挿入時の走査は省略できない *(case. 2)*.

証明. *Case. 1* の場合は定理 *3* から明らかである．*Case. 2* の場合は 図 *5.1* と図 *5.2* の例から明らかである．

系 1 から内部ストリームの割合を増やすことで優先度キューの走査を減らせることが分かる．オペレータトレインを行ってもリアルタイム処理要求を満たすことを保証する 4 章の定理 2 は，入力データが out-of-order であるかどうかに関係無く適用できる．そのため，本問題設定においても 4 章の Algorithm. 3 を同様に適用することができる．

以上の考察から，out-of-order な入力データを持つデータストリームのスケジューリング方式として EDF-PStream を提案する．EDF-PStream では，4 章と同様に Algorithm. 3 によりオペレータトレインを構成する．但し，ストリームキューを優先度キューとして実現し，オペレータトレインの内部ストリームのキューにタプルを挿入する際にはキューの走査を省く．以降では，本提案の有効性を評価する．

5.5　実験評価

本節では，車載システム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．車載システムの各ノードの評価には，カーナビゲーション相当のスペックを持つマシンとして，シングルプロセッサで，CPU:800MHz，メモリ:1024MB，OS:Linux (Fedora10) 32bit を用いた．

5.5.1　アプリケーションシナリオ

4 章と同様のアプリケーションとクエリを用いてクエリ実行時の性能を評価する．但し，5.3 節の問題設定の効果が表れるように，図 4.12 のオペレータ o_6 においてタプルを時刻順にソートせず，同時刻に挿入されるタプルは 1 タプルごとにスケジューリング対象として処理する．車外との通信を用いる車載システムでは，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 [19] 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[1] を用いて入力データを作成した．センサのノイズについては，Scenargie では模擬できないため

[1]Scenargie: `https://www.spacetime-eng.com/`

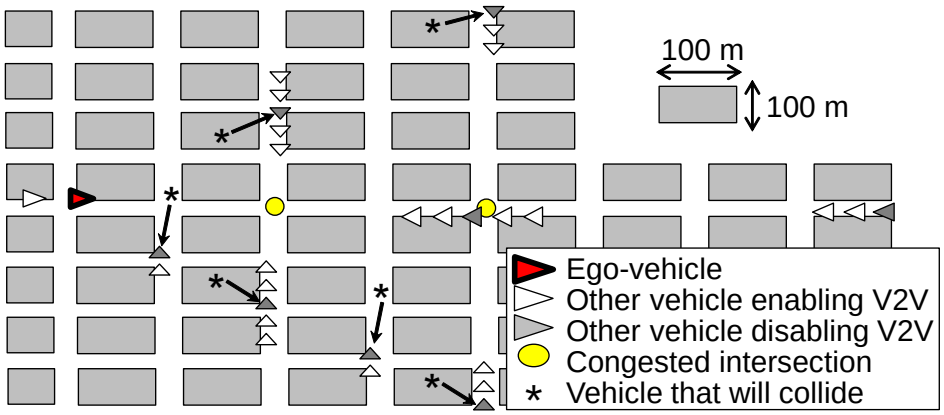


図 5.3: 車両の初期位置

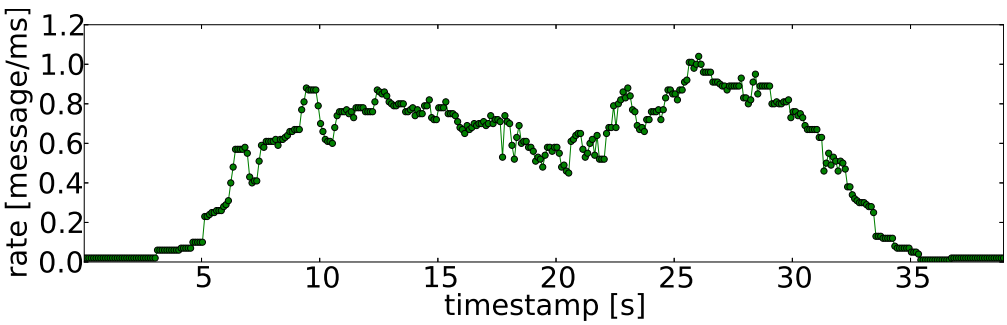


図 5.4: 車々間通信からの入力レートの推移

正規分布に基づく人工的なノイズを付加した．各車両は，交差点間の距離が 100m の碁盤目の道路（マンハッタンモデル）に配置し，矢印の方向に時速 60km で直進した．

車々間通信は，100 ミリ秒周期で 200m 程度の範囲まで正しく受信できるように，信号強度を設定した．また，レーダも同程度の距離まで正しく受信できるように信号強度を設定した．車々間通信の周波数は 700MHz としてレーダの周波数は 70GHz 以上を設定した．これにより，周波数の低い車々間通信の電波はレーダに比べて障害物に対して回折して届く．

車両の初期配置には図 5.3 を用いた．車々間通信器を搭載した車両と搭載していない車両がいるシナリオを想定する．Ego-vehicle は，車々間通信器と提案方式を実装した DSMS を搭載した自車両を表す．他車両（Other vehicle enabling V2V）は，車々間通信器を搭載し自車両と車々間通信を行う．一方，他車両（Other vehicle disabling V2V）は，車々間通信器を搭載していない．車両により混雑する 2 つの交差点では，自車両がその交差点に入ったときに車々間通信で受信する 1 秒あたりの車両の台数が約 1000 台分となり ETSI の定める最大車両台数になる．図 5.4 は，本シミュレーションで自車両が 1 秒あたりに受信する車両台数の推移を表しており，車両で混雑する交差点に自車両が近づいた時にそれぞれピークがあることが分かる．入力データ量が増加し処理待ちのデータが発生することで，その処理順序を決定するスケジューリング方法の違いによる効果の差が生じる．そのため，本シミュレーションの状況としては，スケジューリング方法による効果の差が明確に確認できるように，ETSI が定める最大車両台数の仕様に合うように車両の台数や配置を設定した．

5.5.2　ストリーム処理の従来スケジューリング方式との比較

ストリーム処理における従来のスケジューリング方式として以下と比較する．

EDF-PStream は本章による提案方式である．それは FIFO キューの代わりに優先度キューを用いオペレータトレインにより優先度キューの走査にかかるオーバーヘッドを削減する．

CMP-EDF は従来の EDF に基づくリアルタイムスケジューリング方式である [32, 37, 44, 43]．それはストリーム処理における従来スケジューリング方式の中で最も競合する方式である．それは EDF に基づきオペレータのプリエンプションは行うがストリーム内のタプルのプリエンプションは行わない．そのためストリー

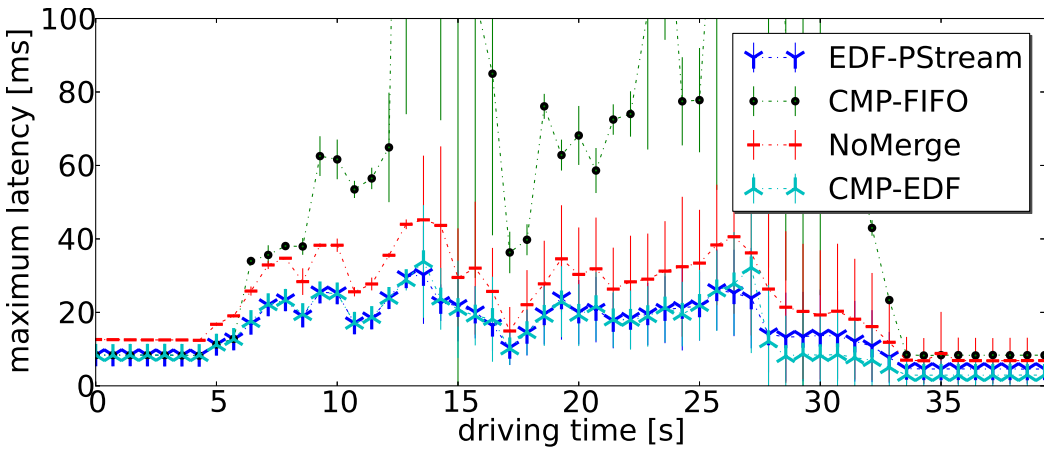


図 5.5: 走行時間に対する output1 の最大遅延時間の変化

ムキューを FIFO キューで実現できる．従来方式 [32, 37, 44, 43] ではオペレータの出力が分岐しタイムアウトを持つオペレータがある場合に対応できない．そのため，本提案方式と同様に Algorithm. 3 を適用しオペレータトレインを構成する．

CMP-FIFO は平均遅延時間を削減する FIFO[5, 12] 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．それはオペレータの実行順序が静的に決まるノンプリエンプティブなスケジューリング方式であり FIFO キューで実現される．それは平均遅延時間を削減する方式として知られている [5, 12]．

NoMerge は優先度キューを用いるが Algorithm. 3 を用いない比較方式である．それは全てのストリームキューを優先度キューとしてタプルの挿入位置を走査する．

5.5.3　リアルタイム処理要求における効果

リアルタイム処理要求を定量的に評価するため，入力データ量を変化させたときの式 (5.1) のデッドラインミス率（DMR）を用いて性能を測る．

$$\frac{1}{\sum_{s\in\{s_1,\cdots,s_n\}}\alpha_s}\sum_{s\in\{s_1,\cdots,s_n\}}\alpha_s\frac{M_s}{N_s} \tag{5.1}$$

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処理要求の重要度を表し，そのデータを供給するアプリケーションプログラムに応じてユーザがクエリの登録時に決定する．

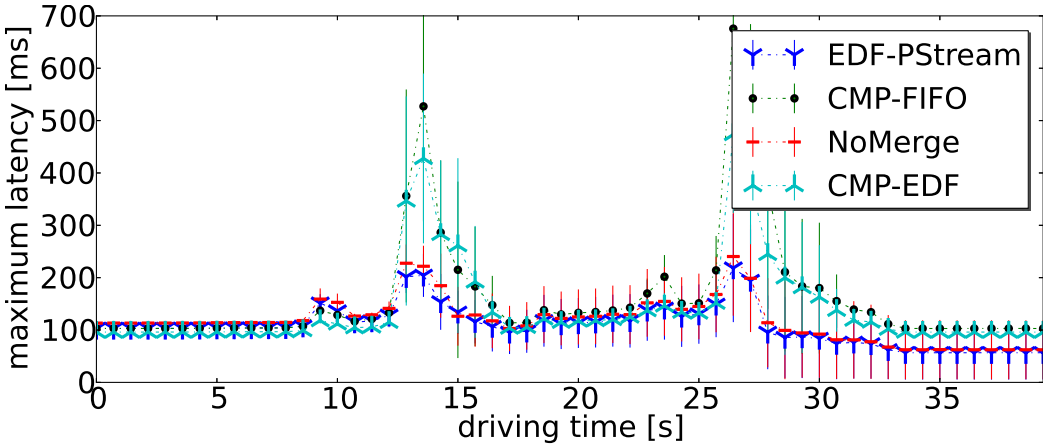


図 5.6: 走行時間に対する output3 の最大遅延時間の変化

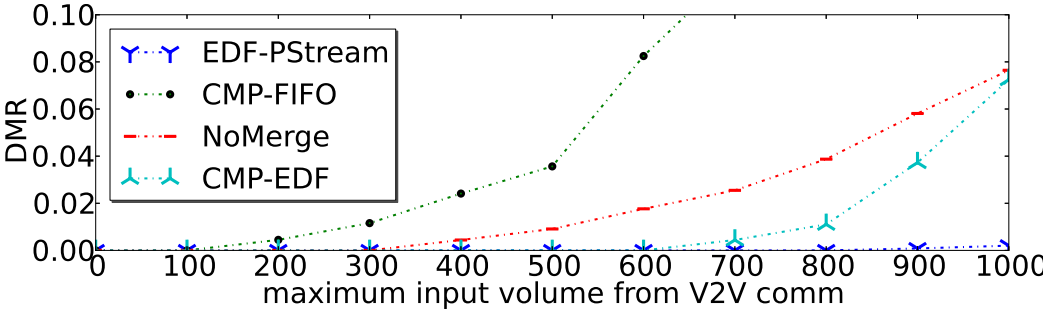


図 5.7: デッドラインミス率

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5.5は図5.6と，各方式においてend-to-endデッドラインが短いoutput1とoutput3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各出力ストリーム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20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．

図5.5よりEDF-PStream, CMP-EDF, NoMergeはFIFOと比べてoutput1の最大遅延時間を削減した．これは，EDF-PStream, CMP-EDF, NoMergeではEDFに基づくスケジューリングを行うことでoutput1に出力されるタプルを優先して処理したためである．図5.5では，EDF-PStreamやCMP-EDFよりもNoMergeの最大遅延時間が大きいことが分かる．これは，NoMergeでは全てのストリームに対して優先度キューの走査を行ったのに対して，他の方式では内部ストリームの優先度キューの走査を行わないことによるスケジューリングオーバーヘッドの軽減による効果である．図5.6より, EDF-PStreamとNoMergeはCMP-EDFやCMP-FIFOと比較して，output3の最大遅延時間を削減した．これは，車々間通信の通信遅延によりデッドラインの早いタプルが後からDSMSに到着した場合に，EDF-PStreamやNoMergeではそれらのタプルのデッドラインに従ってEDFによりスケジューリングしたためである．

図5.7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デッドラインミス率（DMR）の変化を表している．但し，式(5.1)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5.7では，デッドラインミス率の小さい順（リアルタイム処理要求を満たす順）に，本走行シナリオではEDF-PStream, CMP-EDF, NoMerge, CMP-FIFO, であった．それらがリアルタイム処理要求を満たしながら1秒間あたりに処理できるタプルの数はそれぞれ800, 600, 300, 100個であった．この効果はoutput1とoutput3の最大遅延時間の削減による．以降の評価では，5.5.2節の比較方式の中でリアルタイム処理要求をよく満たしたEDF-PStreamとCMP-EDFに着目し，それぞれの方式で1秒あたりに処理できるタプルの数をそれぞれ800個と600個として評価する．

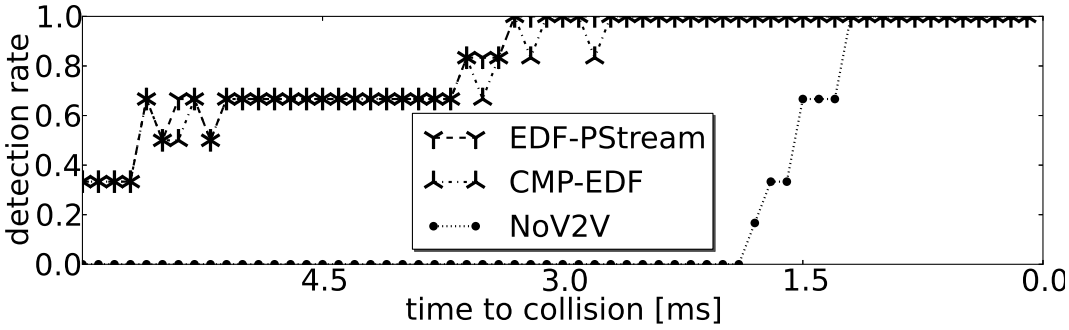


図 5.8: TTC に対する車両検出率の変化

5.5.4　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．自車と衝突する可能性のある車両は，図 5.3 の＊印の付いた 6 台の車両（車両＊）である．時速 60km で乾いた路面を走行する場合の停止距離から [40]，自車両の停止時間を 2.8 秒と算出する．自車両は，車両＊を最初に検出した時にブレーキをかけ始めるとして，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＊を最初に検出した時の 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 (5.2) で定義する．

$$M/N \tag{5.2}$$

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 5.8 は，ある試行において TTC を変化させた時，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 は車々間通信を全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，全ての車両＊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処理要求を満たして動作しているためである．図 5.8 から，CMP-EDF では一時的に車両＊を検出できない時があり，EDF-PStream の方が検出率が良い

ことが分かる．これは，EDF-PStream では CMP-EDF に比べて車々間通信からの最大入力量が 1 秒あたり 600 台から 800 台に増えたためである．これにより，式 (5.2) による事故率は，CMP-EDF では 2.8%であるのに対して EDF-PStream では 0%となった．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．

5.6　おわりに

デッドラインの早いタプルが DSMS に遅れて到着する状況では，データストリームを優先度キューによって管理し同一ストリーム内でタプル同士の追い越しが起こる preemptable なストリームの処理が求められる．本研究では，preemptable なストリームを EDF スケジューリングに基づき効率的に処理する方法を提案した．本提案では，内部ストリームでは挿入されるタプルの優先度が常に最も高いことに着目し，優先度キューの走査を削減することで処理を効率化する．実験評価では，車々間通信による入力データ量が ETSI の仕様で最大化する状況をシミュレートし車両衝突警告のクエリを用いて評価した．比較方式には，ストリーム処理における従来のリアルタイムスケジューリング方式を含む．その結果，提案方式が end-to-end デッドラインの短い出力ストリームにおける最大遅延時間やデッドラインミス率を最も削減することを示し，車両衝突事故を軽減できる可能性を示した．

第6章　結論

6.1　まとめ

本論文では，車載システムのデータ処理要求を満たす分散ストリーム処理を実現するため，以下の3つの研究を実施した．

1つ目は，車載システム向けの分散ストリーム処理システム（AEDSMS）を提案した．AEDSMSでは，SPCによるモジュール化とオペレータ配置による位置透過性を持つ高レベルなクエリ（HLQ）から，車種ごとの物理構成にカスタマイズされた低レベルなクエリ（LLQ）に変換する．これにより，リアルタイム処理要求に必要となる予測可能性と車種展開におけるクエリの再利用性を両立した．標準のオペレータよりも計算時間がかかるSDFOの最悪計算時間を予測できることも確認した．また，設計空間探索の分野で広く用いられるアーキテクチャグラフを介することで，車載ネットワーク上の分散ストリーム処理に従来のオペレータ配置方式を適用できることを示した．更に，各ノードに配置されたオペレータセットにデッドラインを適切に設定することで，各ノード上のDSMSにEDFスケジューリングを適用できることを示し，これにより分散ストリーム処理全体のリアルタイム処理要求を満たすことを示した．ETSIの仕様に従った車々間通信を用いた車両衝突警告のクエリをHLQとして作成し，デッドライン処理要求を満たすことに加えて，安全運転支援における車両衝突事故率の削減や車両情報の位置精度向上への有効性も確認した．また，センサデータフュージョンオペレータ(SDFO)により，車両情報の位置精度が向上することも確認した．

2つ目は，各ノード上のDSMSにおけるEDFスケジューリング方式を提案した．オペレータの出力が分岐する場合やタイムアウトを持つオペレータがある場合においても対応できるように，ストリーム処理にEDFスケジューリングを導入する方法を示した．また，オペレータトレインのアルゴリズムを提案することで，その方法によりデッドライン処理要求を満たしながらスケジューリングのオーバーヘッドを削減できることを示し，性能評価においてもその有効性を確認した．基本性能評価では，本方式がストリーム処理における従来のスケジューリング方式

である FIFO+,MC+,PCS と比較してデッドライン処理要求を満たすことを確認した．また，アプリケーション性能評価では，車々間通信を用いた車両衝突警告のクエリを用いて，デッドライン処理要求を満たし車両衝突事故率を低減することを確認した．

3 つ目は，各ノード上の DSMS における EDF スケジューリングにおいて特に通信遅延などの影響でセンサからデータを読み込む時刻と DSMS にデータが入力される時刻が異なる out-of-order な入力データを効率的に扱う方式を提案した．Out-of-order な入力データでは，リアルタイム処理要求を満たすにはストリームキューに優先度キューが必要となることを示した．特にオペレータトレインの内部ストリームに対しては優先度キューの走査が不要であることを明らかにしオーバーヘッドを削減して効率的に EDF スケジューリングを行う方式を示した．また，実験評価では，車々間通信を用いた車両衝突警告のクエリを用いて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含む従来方式と比較し，デッドライン処理要求を満たし車両衝突事故率を低減することを確認した．

以上 3 つの研究により，リルタイム処理要求を中心とした車載システムのデータ処理要求からストリーム処理の技術課題を整理し，それらを満たす分散ストリーム処理方式を提案した．また，ETSI の仕様に従った車々間通信を用いた車両衝突警告のクエリを作成し，それらの技術課題を解決することを示した．これにより，安全運転支援システムにおけるデータ管理のコスト低減に加えて，リアルタイム処理や信頼性などのデータ処理要求を満たすことで，車々間通信を用いた交通事故の低減にも大きく貢献できる．

6.2　今後の課題

今後の課題としては，実際の車載組込みシステムの環境に近づけた実装/評価が挙げられる．本研究では LINUX 上で提案方式の有効性を評価したが，リアルタイムＯＳ上で実装・評価することで，クエリ実行時のデッドライン処理要求をより満たしやすくなるとともに，リアルタイムシステムの理論をより広範囲に忠実に再現することができる．また，車載システムの実際のソフトウェアプラットフォームとしては AUTOSAR が用いられている．AUTOSAR にも ECU や車載ネットワークの構成に依存しにくい仕組みがあり，AEDSMS のオペレータ配置方式との統合やプラットフォーム間の機能分担なども今後の課題である．

加えて，実用化に向けたプラットフォームの拡充も必要となる．データ処理の開

発効率の向上には，データ処理を容易に記述するための高級なクエリ言語の提供に加えて，ライブラリの充実，他のデータベースやクラウドと容易に連携できる必要がある．また，自動運転を含む安全運転支援では，走行道路，周辺車両／歩行者の認識，経路計画の生成など高度なセンサ情報処理が要求される．対象とするアプリケーションを拡充し，これらのセンサ情報処理を本プラットフォームにより容易に開発できることを実際に確認することが今後の課題である．

最後に，本研究では車載システムの安全運転支援を主なアプリケーションの対象としたが，この成果はリアルタイム処理要求が必要となるロボットや航空機におけるセンサ情報処理にも適用可能である．今後，自動車に限らず様々な分野のセンサ情報処理への展開も期待される．

謝辞

本論文に関する研究活動を通じて，御指導と御助言を頂きました名古屋大学未来社会創造機構の高田広章教授，同大学大学院情報科学研究科情報システム学専攻の本田晋也准教授に，感謝致します．

名古屋大学大学院情報学研究科附属組込みシステム研究センターにおける車載データ統合プロジェクトにおいて，プロジェクトリーダとして御指導と御助言を頂きました，同志社大学理工学部情報システムデザイン学科の佐藤健哉教授に感謝致します．本研究を進めるにあたり，データベースの分野から御指導と御助言を頂きました，石川佳治教授，リアルタイムシステムの分野から御指導と御助言を頂きました，中本幸一教授，に感謝致します．車載データ統合プロジェクト及び車載データ統合アーキテクチャに基づく LDM の実装・評価に関するコンソーシアム型共同研究において，研究開発活動をともに行いました，研究員，常駐研究員の方々にも御協力，御支援を頂き，感謝致します．また，データストリーム処理の研究にあたり，多くの有益な御助言を賜りました，名古屋大学未来社会創造機構の渡辺陽介准教授にも感謝致します．

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へ出向し研究生活を通して成長する機会を与えて頂きました，東芝ソフトウェア技術センターの皆様，社会人ドクターのあいだ御協力，御支援を頂きました，東芝研究開発センター システム技術ラボラトリーの皆様に感謝致します．また，神戸大学大学院自然科学研究科数学専攻の修士研究を通して御指導頂きました，高山信毅教授，九州大学の鈴木昌和教授にも，感謝致します．

最後に，家族をはじめ長い間研究活動をサポートしてくれた方々に心から感謝致します．

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) の助成を頂きました．御礼申し上げます．

参考文献

[1] Daniel J. Abadi, Yanif Ahmad, Magdalena Balazinska, Mitch Cherniack, Jeong hyon Hwang, Wolfgang Lindner, Anurag S. Maskey, Er Rasin, Esther Ryvkina, Nesime Tatbul, Ying Xing, and Stan Zdonik. The Design of the Borealis Stream Processing Engine. In *Proc. of Second Biennial Conference on Innovative Data Systems Research*, pp. 277–289, 2005.

[2] Daniel J Abadi, Yanif Ahmad, Magdalena Balazinska, Jeong-hyon Hwang, Wolfgang Lindner, Anurag S Maskey, Alexander Rasin, Esther Ryvkina, Nesime Tatbul, Ying Xing, and Stan Zdonik. Aurora: a new model and architecture for data stream management. *The VLDB Journal*, Vol. 12, No. 2, pp. 120–139, 2003.

[3] Matthias Althoff, Daniel Althoff, Dirk Wollherr, and Martin Buss. Safety verification of autonomous vehicles for coordinated evasive maneuvers. In *Intelligent Vehicles Symposium*, pp. 1078–1083, 2010.

[4] S. Ammoun and F. Nashashibi. Real time trajectory prediction for collision risk estimation between vehicles. In *Proc. of the IEEE International Conference on Intelligent Computer Communication and Processing*, pp. 417–422, Aug 2009.

[5] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Dilys Thomas. Operator Scheduling in Data Stream Systems. *The VLDB Journal*, Vol. 13, No. 4, pp. 333–353, December 2004.

[6] Brian Babcock, Shivnath Babu, Rajeev Motwani, and Mayur Datar. Chain: operator scheduling for memory minimization in data stream systems. In *Proc. of the 2003 ACM SIGMOD international conference on Management of data*, pp. 253–264, 2003.

[7] A. Bolles, H. Appelrath, D. Geesen, M. Grawunder, M. Hannibal, J. Jacobi, F. Koster, and Daniela Nicklas. StreamCars: A new flexible architecture for driver assistance systems. In *Intelligent Vehicles Symposium*, pp. 252–257, June 2012.

[8] Martin Buehler, Karl Iagnemma, and Sanjiv Singh. *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic*. Springer, 1st edition, 2009.

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

[10] Don Carney, Uğur Çetintemel, Alex Rasin, Stan Zdonik, Mitch Cherniack, and Mike Stonebraker. Operator scheduling in a data stream manager. In *Proc. of the 29th International Conference on Very Large Data Bases*, pp. 838–849, 2003.

[11] L. Casparsson, A. Rajnak, K. Tindell, and P. Malmberg. Volcano a revolution in on-board communications. *Volvo Technology Report*, 1998.

[12] Sharma Chakravarthy and Qingchun Jiang. *Stream Data Processing: A Quality of Service Perspective Modeling, Scheduling, Load Shedding, and Complex Event Processing*. Springer, 1st edition, 2009.

[13] H. Chetto, M. Silly, and T. Bouchentouf. Dynamic Scheduling of Real-time Tasks Under Precedence Constraints. *Real-Time Syst.*, Vol. 2, No. 3, pp. 181–194, September 1990.

[14] Gianpaolo Cugola and Alessandro Margara. Processing Flows of Information: From Data Stream to Complex Event Processing. *ACM Comput. Surv.*, Vol. 44, No. 3, pp. 15:1–15:62, June 2012.

[15] Robert I. Davis, Alan Burns, Reinder J. Bril, and Johan J. Lukkien. Controller area network (can) schedulability analysis: Refuted, revisited and revised. *Real-Time Syst.*, Vol. 35, No. 3, pp. 239–272, April 2007.

[16] Guizzo Erico. How Google's Self-Driving Car Works, 2011.

[17] ETSI. Intelligent Transport Systems; Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service, March 2011.

[18] ETSI. Intelligent Transport Systems; Users and applications requirements; Part 2: Applications and facilities layer common data dictionary, August 2013.

[19] ETSI. Intelligent Transport Systems; V2X Applications; Part 3: Longitudinal Collision Risk Warning application requirements specification, November 2013.

[20] FHWA. Intersection Safety Issue Briefs, 2004.

[21] Ulrich Freund. Mulit-level System Integration Based on AUTOSAR. In *Proc. of the 30th International Conference on Software Engineering*, pp. 581–582, 2008.

[22] Bugra Gedik, Henrique Andrade, Kun-Lung Wu, Philip S. Yu, and Myungcheol Doo. SPADE: The System s Declarative Stream Processing Engine. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pp. 1123–1134, 2008.

[23] Erico Guizzo. Toyota's Semi-Autonomous Car Will Keep You Safe, 2013.

[24] Martin Hirzel, Robert Soulé, Scott Schneider, Buğra Gedik, and Robert Grimm. A Catalog of Stream Processing Optimizations. *ACM Comput. Surv.*, Vol. 46, No. 4, March 2014.

[25] Qingchun Jiang and Sharma Chakravarthy. Scheduling strategies for processing continuous queries over streams. In *BNCOD*, Vol. 3112 of *Lecture Notes in Computer Science*, pp. 16–30. Springer, 2004.

[26] Evangelia Kalyvianaki, Wolfram Wiesemann, Quang Hieu Vu, Daniel Kuhn, and Peter Pietzuch. SQPR: Stream Query Planning with Reuse. In *Proc. of the 2011 IEEE 27th International Conference on Data Engineering*, pp. 840–851. IEEE Computer Society, 2011.

[27] Evangelia Kalyvianaki, Wolfram Wiesemann, Quang Hieu Vu, Daniel Kuhn, and Peter Pietzuch. SQPR: Stream Query Planning with Reuse. In *Proc. of the 2011 IEEE 27th International Conference on Data Engineering*, pp. 840–851. IEEE Computer Society, 2011.

[28] Hillol Kargupta, Ruchita Bhargava, Kun Liu, Michael Powers, Patrick Blair, Samuel Bushra, James Dull, Kakali Sarkar, Martin Klein, Mitesh Vasa, and David Handy. VEDAS: A Mobile and Distributed Data Stream Mining System for Real-Time Vehicle Monitoring. In *Proc. of the SIAM International Conference on Data Mining*, pp. 300–311, 2004.

[29] Hillol Kargupta, Kakali Sarkar, and Michael Gilligan. MineFleet®: an overview of a widely adopted distributed vehicle performance data mining system. In *Proc. of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 37–46, 2010.

[30] Dhananjay Kulkarni, Chinya V. Ravishankar, and Mitch Cherniack. Real-time, Load-adaptive Processing of Continuous Queries over Data Streams. In *Proc. of the Second International Conference on Distributed Event-based Systems*, pp. 277–288. ACM, 2008.

[31] Geetika T. Lakshmanan, Ying Li, and Rob Strom. Placement Strategies for Internet-Scale Data Stream Systems. *IEEE Internet Computing*, Vol. 12, No. 6, pp. 50–60, November 2008.

[32] Xin Li, Zhiping Jia, Li Ma, Ruihua Zhang, and Haiyang Wang. Earliest Deadline Scheduling for Continuous Queries over Data Streams. In *Proc. of International Conference on Embedded Software and Systems*, pp. 57–64, May 2009.

[33] Yamada Masahiro, Sato Kenya, and Takada Hiroaki. Implementation and Evaluation of Data Management Methods for Vehicle Control Systems. In *Vehicular Technology Conference (VTC Fall)*, pp. 1–5, September 2011.

[34] D. Nystrom, A. Tesanovic, C. Norstrom, J. Hansson, and N.-E. Bankestad. Data management issues in vehicle control systems: a case study. In *Proc. of the Euromicro Conference on Real-Time Systems*, pp. 249–256, june 2002.

[35] Dag Nystrom, Aleksandra Tesanovic, Ra Tesanovic, Mikael Nolin, Christer Norstrom, and Jorgen Hansson. COMET: A Component-Based Real-Time Database for Automotive Systems. In *Proc. of the Workshop on Software Engineering for Automotive Systems*, pp. 1–8, 2004.

[36] Marcio F. S. Oliveira, Francisco A. Nascimento, Wolfgang Mueller, and Flávio R. Wagner. Design Space Abstraction and Metamodeling for Embedded Systems Design Space Exploration. In *Proc. of the 7th International Workshop on Model-Based Methodologies for Pervasive and Embedded Software*, pp. 29–36, 2010.

[37] Zhengyu Ou, Ge Yu, Yaxin Yu, Shanshan Wu, Xiaochun Yang, and Qingxu Deng. Tick Scheduling : A Deadline Based Optimal Task Scheduling Approach for Real-Time Data. In *Proc. of Advances in Web-Age Information Management*, pp. 725–730, 2005.

[38] Hendrik Schweppe, Armin Zimmermann Member, and Daniel Grill. Flexible On-Board Stream Processing for Automotive Sensor Data. *IEEE Trans. on Industrial Informatics*, Vol. 6, No. 1, pp. 81–92, 2010.

[39] Randall C. Smith and Peter Cheeseman. On the Representation and Estimation of Spatial Uncertainly. *Int. J. Rob. Res.*., Vol. 5, No. 4, pp. 56–68, December 1986.

[40] South Australia. *The Driver's Handbook*. Department for Transport, Energy and Infrastructure, 2005.

[41] Qing Tian, Teng Guo, Shuai Qiao, Yun Wei, and Wei wei Fei. Design of Intelligent Parking Management System Based on License Plate Recognition. *Journal of Multimedia*, Vol. 9, No. 6, pp. 774–780, 2014.

[42] G. Vivo, P. Dalmasso, and F. Vernacchia. The European Integrated Project SAFESPOT - How ADAS applications co-operate for the driving safety. In *Proc. of IEEE Conference on Intelligent Transportation Systems*, pp. 624–629, 2007.

[43] Yuan Wei, V. Prasad, S.H. Son, and J.A. Stankovic. Prediction-Based QoS Management for Real-Time Data Streams. In *Proc. of IEEE Real-Time Systems Symposium*, pp. 344–358, Dec 2006.

[44] Yuan Wei, S.H. Son, and J.A. Stankovic. RTSTREAM: real-time query processing for data streams. In *Proc. of the 9th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*, pp. 141–150, April 2006.

[45] Yongluan Zhou, Ji Wu, and Ahmed Khan Leghari. Multi-query scheduling for time-critical data stream applications. In *Proc. of the 25th International Conference on Scientific and Statistical Database Management*, pp. 15:1–15:12, 2013.

[46] 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, jun 2010.

[47] 山口晃広. 安全運転支援のためのストリーム型センサデータの分散処理機構. 進化適応型自動車運転支援システム「ドライバ・イン・ザ・ループ」研究拠点形成平成 26 年度成果報告会（第 2 回シンポジウム）, mar 2015.

[48] 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.

[49] 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, feb 2007.

[50] 山口晃広, 渡辺陽介, 佐藤健哉, 中本幸一, 高田広章. デッドラインを持つクエリプラン割り当てによる分散ストリーム処理のリアルタイムスケジューリング方式. 研究報告組込みシステム（EMB）, Vol. 2014, No. 25, pp. 1–8, nov 2014.

[51] 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, mar 2010.

[52] 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, jun 2011.

[53] 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, dec 2012.

研究業績

主論文に関連する研究業績

学術論文

- Akihiro Yamaguchi, Yousuke Watanabe, Kenya Sato, Yukikazu Nakamoto, Yoshiharu Ishikawa, Shinya Honda, and Hiroaki Takada, “In-Vehicle Distributed Time-critical Data Stream Management System for Advanced Driver Assistance”, Journal of Information Processing, Vol. 25, pp. 107–120, 2017/1.

- 山口晃広, 渡辺陽介, 佐藤健哉, 中本幸一, 高田広章, “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”, 情報処理学会論文誌（データベース） Vol. 8 No. 2, pp. 1–17, 2015/6.

国際会議論文（査読あり）

- Akihiro Yamaguchi, Yukikazu Nakamoto, Kenya Sato, Yousuke Watanabe and Hiroaki Takada, “EDF-PStream: Earliest deadline first scheduling of preemptable data streams - Issues related to automotive applications”, Proceedings of the 21st IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2015), pp. 257–267, 2015/8.

- Akihiro Yamaguchi, Yukikazu Nakamoto, Kenya Sato, Yoshiharu Ishikawa, Yousuke Watanabe, Shinya Honda, Hiroaki Takada, “AEDSMS: Automotive Embedded Data Stream Management System”, The 31st International Conference on Data Engineering (ICDE 2015), pp. 1292–1303, 2015/4.

研究会発表論文（査読なし）

- 高田広章, 石川佳治, 佐藤健哉, 中本幸一, 本田晋也, 山口晃広, “次世代車載連携アプリケーション向け分散処理プラットフォームの開発”, ICT イノベーションフォーラム 2015, 2015/10.
- 山口晃広, 佐藤健哉, 高田広章, “安全運転支援のためのストリーム型センサデータの分散処理機構”, 進化適応型自動車運転支援システム「ドライバ・イン・ザ・ループ」研究拠点形成 平成 26 年度成果報告会（第 2 回シンポジウム） 2015/3.
- 山口晃広，渡辺陽介，佐藤健哉，中本幸一，高田広章，“デッドラインを持つクエリプラン割り当てによる分散ストリーム処理のリアルタイムスケジューリング方式”，情報処理学会研究報告，Vol.2014-DBS-160, No.25, pp. 1–8, 2014/11.
- 山口晃広，渡辺陽介，佐藤健哉，中本幸一，高田広章，“デッドラインを持つクエリプラン割り当てによる分散ストリーム処理のリアルタイムスケジューリング方式”，WebDB フォーラム （ポスター） 2014/11.
- 山口晃広，“車載データ統合アーキテクチャに基づく LDM の実装・評価に関するコンソーシアム型共同研究”，第 4 回名古屋大学組込みシステム研究センターシンポジウム講演予稿集 pp. 67–80, 2014/10.
- 山口晃広，佐藤健哉，中本幸一，渡辺陽介，高田広章，“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”，情報処理学会研究報告，Vol. 2014-ITS-58, No.7, pp. 1–8, 2014/9.
- 高田広章, 石川佳治, 佐藤健哉, 中本幸一, 本田晋也, 山口晃広, “次世代車載連携アプリケーション向け分散処理プラットフォームの開発”, ICT イノベーションフォーラム 2015, 2015/10.
- 山口晃広, “安全運転支援のためのセンサデータのストリーム型処理機構”, 平成 26 年度私立大学戦略的研究基盤形成支援事業 進化適応型自動車運転支援システム「ドライバ・イン・ザ・ループ」研究拠点形成 第 3 回シンポジウム, 2015/9.

- 山口晃広, 佐藤健哉, 高田広章, “安全運転支援のためのストリーム型センサデータの分散処理機構”, 進化適応型自動車運転支援システム「ドライバ・イン・ザ・ループ」研究拠点形成 平成 26 年度成果報告会（第 2 回シンポジウム） 2015/3.

- 山口晃広，渡辺陽介，佐藤健哉，中本幸一，高田広章，“デッドラインを持つクエリプラン割り当てによる分散ストリーム処理のリアルタイムスケジューリング方式”，情報処理学会研究報告，Vol. 2014-DBS-160, No.25, pp. 1–8, 2014/11.

- 山口晃広，渡辺陽介，佐藤健哉，中本幸一，高田広章，“デッドラインを持つクエリプラン割り当てによる分散ストリーム処理のリアルタイムスケジューリング方式”，WebDB フォーラム （ポスター） 2014/11.

- 山口晃広，“車載データ統合アーキテクチャに基づく LDM の実装・評価に関するコンソーシアム型共同研究”，第 4 回名古屋大学組込みシステム研究センターシンポジウム講演予稿集 pp. 67–80, 2014/10.

- 山口晃広，佐藤健哉，中本幸一，渡辺陽介，高田広章，“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”，情報処理学会研究報告，Vol. 2014-ITS-58, No.7, pp. 1–8, 2014/9.

受賞等

- Akihiro Yamaguchi, Yousuke Watanabe, Kenya Sato, Yukikazu Nakamoto, Yoshiharu Ishikawa, Shinya Honda, and Hiroaki Takada, “In-Vehicle Distributed Time-critical Data Stream Management System for Advanced Driver Assistance”, WebDB フォーラム 2016 最優秀論文賞，2016/11.

- 山口晃広，佐藤健哉，中本幸一，渡辺陽介，高田広章，“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”，ITS 研究会奨励賞受賞，2014/9.

その他

- ［招待講演］ 山口晃広, “安全運転支援のためのセンサデータのストリーム型処理機構”, 平成 26 年度私立大学戦略的研究基盤形成支援事業 進化適応型自動車運転支援システム「ドライバ・イン・ザ・ループ」研究拠点形成 第 3 回シンポジウム, 2015/9.

その他の研究業績

学術論文

- Jaeyong Rho, Takuya Azumi, Akihiro Yamaguchi , Kenya Sato, and Nobuhiko Nishio, “Reservation-Based Scheduling for Automotive DSMS under High Overload Condition”, IPSJ Journal, Vol. 24, No. 5 pp. 751–761, 2016.
- Hideki Shimada, Akihiro Yamaguchi , Hiroaki Takada, Kenya Sato, “Implementation and Evaluation of Local Dynamic Map in Safety Driving Systems”, Journal of Transportations Technologies, Vol. 5, No. 2, pp. 102–112, 2015/4.
- Shingo Akiyama, Yukikazu Nakamoto, Akihiro Yamaguchi , Kenya Sato, Hiroaki Takada, “Vehicle Embedded Data Stream Processing Platform for Android Devices”, International Journal of Advanced Computer Science and Applications、Vol. 6 No. 2 pp. 285–294, 2015/3/1.
- 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 Vol. J95–D, No. 12, pp. 2031–2047, 2012/12.

紀要

- Kenya Sato, Hideki Shimada, Satoshi Katsunuma, Akihiro Yamaguchi , Masahiro Yamada, Shinya Honda, and Hiroaki Takada, “Stream LDM: Local Dynamic Map (LDM) with Stream Processing Technology”, The Science and Engineering Review of Doshisha University, Vol. 53, No. 3, pp. 28–35, 2012/10.

国際会議論文（査読あり）

- Yukikazu Nakamoto, Akihiro Yamaguchi , Kenya Sato, Shinya Honda, Hiroaki Takada, “Toward data-centric software architecture for automotive systems - Embedded data stream processing approach -”, The 11th IEEE International Conference on Autonomic and Trusted Computing, 2014/12.
- Jaeyong Rho, Akihiro Yamaguchi , Kenya Sato, Takuya Azumi and Nobuhiko Nishio, “ROP-EDF: Reservation-Based OP-EDF Scheduling for Automotive Data Stream Management System”, WiP session of the IEEE 11th International Conference on Embedded Software and Systems (ICESS2014), 2014/8.
- Yukikazu Nakamoto, Masanori Okamoto, Mohanmed Bhuiya, Akihiro Yamaguchi , Kenya Sato, Shinya Honda, and Hiroaki Takada, “Android Platform based on Vehicle Embedded Data Stream Processing”, the 10th IEEE International Conference on Ubiquitous Intelligence and Computing (UIC–2013), pp. 48–55, 2013/12.
- Kenya Sato, Erika Matsumoto, Hideki Shimada, Akihiro Yamaguchi , Shinya Honda, Hiroaki Takada, “A Proposal of Network Architecture for Narrowband V2X Communication”, The 20th ITS World Congress, pp. 1–10, 2013/10.

ワークショップ（査読あり）

- Akihiro Yamaguchi, Kenya Sato, Naoyuki Shiba, Shinya Honda, and Hiroaki Takada, “ADVISE: Autonomous Driving Vehicle for Individuality in a Stream Environment”, The 6th Biennial Workshop on Digital Signal Processing for In-Vehicle Systems, pp. 57–63, 2013/9.

研究会発表論文（査読なし）

- 中井将貴, 松原豊, 高田広章, 山口晃広, “データストリーム管理システムのための動作検証環境”, 情報処理学会研究報告, Vol. 2015–SLDM-170, No. 14, pp. 1–6, 2015/3.

- 島田秀輝, 山口晃広, 本田晋也, 高田広章, 佐藤健哉, “協調型安全運転支援のためのセンサデータ分散管理方式”, 第 52 回同志社大学理工学研究所研究発表会・2014 年度学内研究センター合同シンポジウム, 2014/12.
- 中井将貴, 松原豊, 高田広章, 山口晃広, “組込みデータストリーム管理システム eDSMS の可視化”, 第 16 回組込みシステム技術に関するサマーワークショップ（SWEST16）予稿集, 2014/9.
- 島田秀輝，山口晃広，本田晋也，高田広章，佐藤健哉，“組込み型データベースを利用した Local Dynamic Map の設計”，第 51 回 同志社大学理工学研究所研究発表会・2013 年度 学内研究センター合同シンポジウム，pp. 115–120, 2013/12.
- 中本幸一, 伊藤信一, 岡本昌訓, ムハンマドブイヤ, 山口晃広, 佐藤健哉, 本田晋也, 高田広章, “車載システム向けデータセントリックソフトウェアアーキテクチャとデータストリーム処理プラットフォーム”, 信学技法, 2013/12.
- 伊藤信一, 山口晃広, 佐藤健哉, 本田晋也, 高田広章, “ストリーム LDM における地図データのストリーム化機構の設計と評価”, 情報処理学会第 53 回高度交通システム研究会, No. 2, pp. 1–8, 2013/6.
- 山川達也, 鈴木結香子, 山口晃広, 島田秀輝, 佐藤健哉, “理想的ドライバモデルを目指した時系列センサデータの確率モデルに基づく運転支援システムの検討”, 第 76 回情報処理学会全国大会講演論文集, 2013/3.
- 勝沼聡, 熊谷康太, 山口晃広, 本田晋也, 佐藤健哉, 高田広章, “車載データ管理プラットフォームの運転支援アプリケーションへの適用”, 第 5 回データ工学と情報マネジメントに関するフォーラム, A5-4, pp. 1–8, 2013/3.
- 中井将貴, 佐藤健哉, 島田秀輝, 山口晃広, 高田広章, “車載データ統合アーキテクチャにおけるセンサ変更時のソフトウェア設計容易性の検討”, 第 75 回情報処理学会全国大会講演論文集, Vol. 3, pp. 161–162, 2013/3.
- 島田秀輝, 山口晃広, 本田晋也, 高田広章, 佐藤健哉, “協調型安全運転支援のための車載データ管理機構の実装と評価”, 第 50 回同志社大学理工学研究所発表会・2012 年度学内研究センター合同シンポジウム講演予稿集, pp. 140–145, 2012/12.

- 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”, 第 11 回情報科学技術フォーラム講演論文集, Vol. 2, pp. 7–14, 2012/9.
- 佐藤健哉，勝沼聡，山口晃広，島田秀輝，本田晋也，中本幸一，高田広章，“Cloudia：車載データ統合プラットフォーム 基本コンセプト”，第 155 回 SLDM・第 24 回 EMB 合同研究発表会 (ETNET2012), No. 18, pp. 1–6, 2012/3.
- 勝沼聡, 杉本明加, 山口晃広, 山田真大, 金榮柱, 本田晋也, 佐藤健哉, 高田広章, “車載システム向けストリームデータ処理の提案と評価”, 情報処理学会研究報告, No. 1, pp. 1–8, 2011/11.
- 勝沼聡, 山口晃広, 山田真大, 杉本明加, 金榮柱, 本田晋也, 佐藤健哉, 高田広章, “組込みシステム向けストリームデータ処理プラットフォーム 車載データ統合への適用”, 第 13 回組込みシステム技術に関するサマーワークショップ (SWEST13) 予稿集, pp. 98–101, 2011/9.
- 山口晃広, 山田真大, 勝沼聡, 本田晋也, 佐藤健哉, 高田広章, “車載 DSMS における静的クエリ最適化”, 第 4 回 Web とデータベースに関するフォーラム, 1G–2–4, pp. 1–9, 2011/11.

受賞等

- 伊藤信一, 山口晃広, 佐藤健哉, 本田晋也, 高田広章, “ストリーム LDM における地図データのストリーム化機構の設計と評価”, ITS 研究会奨励賞受賞，2013/6.
- 山川達也, 鈴木結香子, 山口晃広, 島田秀輝, 佐藤健哉, “理想的ドライバモデルを目指した時系列センサデータの確率モデルに基づく運転支援システムの検討”, 学生奨励賞受賞，2013/3.

その他

- ［技術交流会 話題提供］勝沼聡, 山口晃広，“車載データ統合アーキテクチャ研究の紹介”，組込みシステム開発技術研究会 (CEST), 第 144 回技術交流会, 2012/2.