Vol.2014-DBS-160 No.25
Vol.2014-OS-131 No.2
Vol.2014-EMB-35 No.25
2014/11/18

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

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

概要：近年，複数の機器（ノード）に分散するセンサなどから得られる連続的なデータを低遅延に処理し，アプリケーションに高度な情報を提供する分散ストリーム処理の重要性が高まっている．自動車を制御する場合など，これらの分散ストリーム処理では，センサからデータが発生してから処理が完了するまでの End-to-End のデッドラインを超えないリアルタイム制約が要求される場合も多い．しかし，従来のストリーム処理におけるスケジューリングやクエリプランのノード割り当ての方式は，平均的な遅延時間の削減などが主な目的として研究されており，分散ストリーム処理のリアルタイム制約に対応することは難しい．本研究では，リアルタイム制約に対応する分散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，(1) 各ノード上のローカルクエリプランにおけるサブデッドラインを数理計画モデルから決定し，(2) 動的なリアルタイムスケジューリングである Earliest Deadline First により各ノード上でローカルクエリプランを処理する．本提案の数理計画モデルでは，リアルタイム制約を数理計画の目的や制約として，ノードへ自由に割り当てられるオペレータに対して，その配置も同時に最適化できる．従来のストリーム処理の方式と比較し，提案方式がリアルタイム制約の維持に有効であることを確認した．

1. はじめに

近年，センサデータや，株価や外国為替レートの情報，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処理し，複数のアプリケーションに高度な情報を提供するデータストリーム処理が注目を集めている．データストリーム処理では，ユーザがクエリをデータ処理の実行前にあらかじめ登録しておく．登録されたクエリは，クエリプランとしてオペレータをストリームで繋いだデータフロー形式で表現される．クエリプランは，センサなどのソースからクエリプランの入力ストリームにデータがタプルとしてシステムに入力されることで実行され，その処理結果が出力ストリームに到着する．出力ストリームに到着したデータはアプリケーションプログラムなどのシンクに Push 型で配信される [1].

これらのストリーム処理では，データ処理の終端点となるソースやシンクが複数の機器（ノード）に分散していることが多いため，分散ストリーム処理の研究も広く行われてきている．分散ストリーム処理システムでは，クエリプランを分散するマシン（ノード）に分割して割り当て，各ノードのデータストリーム管理システム（DSMS）がそのローカルクエリプランを実行する [2], [3].

一方，自動車の衝突警告などでは，センサからデータが生成されてから，その処理が完了するまでの End-to-End デッドラインを超えないように処理するリアルタイム制約が要求される．リアルタイムシステムの分野では，シングルプロセッサの単一ノード上のスケジューリング方式として，Rate monotonic（RM）や Earliest deadline first（EDF）などが良く知られている [4]．また，分散リアルタイムシステムの分野では，複数のノードにまたがるタスクに対してリアルタイムスケジューリングを行う研究が進められているが，分散環境全体で最適なスケジューリングを行うことは難しい．その解決策の 1 つとして，複数のノードにまたがるタスクをサブタスクに分割し，タスクの End-to-End デッドラインから，各サブタスクのローカルなデッドライン（サブデッドライン）を決定するアプローチがある [5], [6]．これにより，適切なサブデッドラインを決めることで，従来の単一ノード上のスケジューリングが適用可能となり，分散システムにリアルタイムタイムスケジューリングを導入できる．

ストリーム処理の性能改善として，メモリ使用量や平均

[1] 名古屋大学大学院情報科学研究科附属組込みシステム研究センター
Center for Embedded Computing Systems, Nagoya University
[2] 同志社大学モビリティ研究センター
Mobility Research Center, Doshisha University
[3] 兵庫県立大学大学院応用情報科学研究科
Graduate School of Applied Informatics, University of Hyogo
[4] 名古屋大学 未来社会創造機構
Institute of Innovation for Future Society, Nagoya University

遅延時間の削減などを目的としたオペレータのスケジューリングは，多く研究されてきている [7]．また，分散ストリーム処理においては，クエリプランのノードへの割り当て方が性能に大きな影響を及ぼすため，通信量の削減やノード間のロードバランスを目的としたオペレータ配置問題も，よく研究されている [2]．しかし，これらの従来手法では，リアルタイム制約の維持を目的としておらず，この目的を達成することは難しい．

本稿では，デッドラインを持つクエリプランの割り当てを含む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を提案する．本提案では，分散リアルタイムシステムのサブデッドラインによるリアルタイムスケジューリングを分散ストリーム処理に適応する．また，従来の分散ストリーム処理のオペレータ配置に対してもリアルタイム制約に適応させ，1 つの数理計画モデルに統合することで，サブデッドラインの決定とオペレータの配置を同時に最適化する．そして，以上の手法から算出されるサブデッドラインを持つローカルクエリプランに対して，EDF スケジューリングを適用する．

本稿の構成は以下のとおりである．まず，2 章で関連研究を紹介し，3 章で本稿で想定する仮定を述べる．次に，4 章で本研究が解決する課題を述べ，5 章で提案手法を示し，6 章でその評価を行う．最後に，7 章でまとめを述べる．

2. 関連研究

これまでストリーム処理のスケジューリング方式としては，クエリプランのストリームキューのメモリ使用量を削減するようにオペレータをスケジューリングする方式 [8] や，平均的な遅延時間の削減やスループットなどの目的でオペレータをスケジューリングする方式 [9] など，様々な方式が研究されてきた．しかし，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は，自動車や航空機，ロボットの制御などで重要であるにもかかわらず，RM に基づく方式 [10] や EDF に基づく方式 [11] が少数研究されてきたのみであり，End-to-End デッドラインを持つ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は未だ研究されていない．

分散ストリーム処理のオペレータ配置問題は，クエリプランを構成するオペレータとストリームを各ノードに割り当てる組み合わせ問題となり計算困難な問題であり，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 [2]．その後，整数計画問題としてオペレータ配置を定式化する方式が提案された [3]．しかし，これらの従来方式はリアルタイム制約の維持を目的とせず，ローカルクエリプランのリアルタイムスケジューリングで必要なサブデッドラインを求める方法は存在しない．そのため，これらの従来方式で求めたローカルプランに EDF などのリアルタイムスケジューリングを適用できない．また，デッドラインを満たすようにオペレータを配置する方式も未だ研究されていない．

分散リアルタイムシステムの分野では，各ノードに配置されたサブタスクをリアルタイムスケジューリングするため，End-to-End デッドラインからサブデッドラインを決定する方法が幾つか研究されてきた．また，サブデッドラインの決定とサブタスクのノードへの配置を同時に最適化する手法も研究されている [12]．サブデッドラインの決定は，複数ノードをまたがるタスクの End-to-End のデッドラインを満たすことに加えて，ある 1 つのノードに複数搭載されたサブタスク全体でもデッドラインを満たすことが要求されるため，計算困難な問題である [13]．そのため，これらの方法はヒューリスティクスに基づくが，その中で凸計画問題として定式化する方法が提案された [6]．本研究では，これらの技術を融合することで，End-to-End デッドラインを持つ分散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実現する．

3. 本研究で想定するモデル

本研究では，複数のノードに，ソースやシンク，配置が固定されたオペレータが複数分散する一般的なストリーム処理において，各出力ストリームの End-to-End デッドラインが指定されるケースを扱う．また，オペレータの計算時間が通信遅延に対して十分に大きく通信遅延の影響がない理想的な状況を仮定する．各ノードのスケジューリングに EDF を用いるため，各ノードはシングルプロセッサを仮定し，オペレータの配置はクエリ実行時に動的に移動しないものとする．

クエリプランとしては，複数の入力を持つオペレータや，処理結果を複数のストリームに配信するオペレータを含むマルチクエリを対象とする．ただし議論を簡単にするために，複数入力のオペレータである場合には，ある入力が到着した時に，他の入力を待ち合わせてから処理するタイプに限定する．例えば，複数のセンサデータを入力してセンサフュージョンを行う場合，異なるセンサからの入力の到着を待って処理する必要があるため，本ケースに含まれる．対象外となるのは，ある入力が到着した時に他の入力が到着していなくても，直ちに結果を出力する場合である[*1]．

4. 分散ストリーム処理のリアルタイム制約における課題

一般にリアルタイム性が要求される処理において，デッドラインをミスすると事故などの損害を被る可能性がある．従来のストリーム処理におけるスケジューリング方式やオペレータ配置手法では，End-to-End デッドライン制

[*1] Aurora や Borealis の Union オペレータが相当する

Vol.2014-DBS-160 No.25
Vol.2014-OS-131 No.2
Vol.2014-EMB-35 No.25
2014/11/18

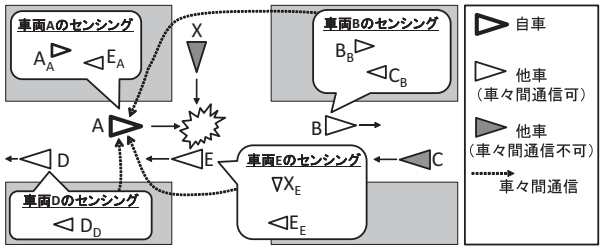

図 1 自動車の衝突警告におけるユースケースシナリオ
Fig. 1 Usecase scenario in collision warning of a vehicle.

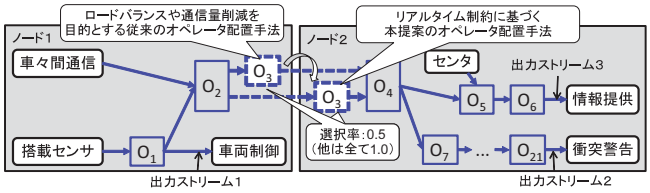

図 2 ユースケースを実現するクエリプランの割り当て
Fig. 2 Query allocation to implement the usecase scenario.

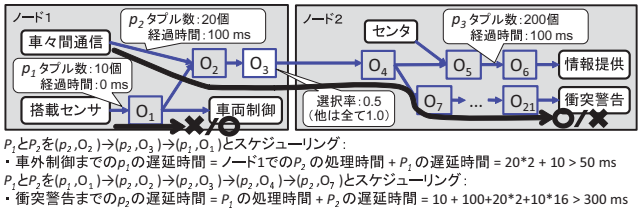

図 3 従来のクエリプラン割り当てにおけるスケジューリングの例
Fig. 3 Examples of scheduling in existing query allocation.

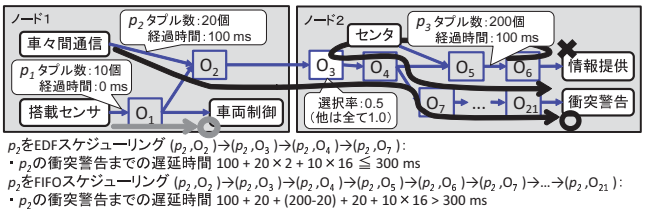

図 4 本提案のクエリプラン割り当てにおけるスケジューリングの例
Fig. 4 Examples of scheduling in proposal query allocation.

約の維持を目的としておらず，これらの方式ではこの目的を達成することが難しい．そのため，リアルタイム制約の維持に有効なオペレータ配置手法と，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が必要となる．以降では具体例として，自動車の衝突警告のアプリケーションを用いて，提案手法が解決する問題を説明する．

4.1 具体例：自動車の車々間通信による衝突警告

図 1 は，見通しの悪い交差点における車々間通信を用いた衝突警告のシナリオを表している．自車両 A は，車両 X を早期に発見してブレーキをかけなければ車両 X と衝突してしまうが，見通しが悪いためにレーダやカメラなどの搭載センサでは車両 X を早期に検知できない．一方，車両 E は，それに搭載されたセンサにより車両 X を早期に検知できる．車両 B,D,E は，自車両 A と車々間通信ができるが，車両 X,C とは車々間通信できない．そのため，車両 B,D,E はそれらの搭載センサでセンシングした車両の情報を車々間通信を用いてブロードキャストし，自車両 A はこれらのメッセージを受信して衝突警告を早期に処理することで衝突を回避できる．ここでは，欧州電気通信標準化機構（ETSI）の仕様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 [14]，車両 X のデータを車両 E から受信し 300 ミリ秒以内に処理しなければ，自車両 A と車両 X は衝突する状況を想定する．なお，図中の Y_Z は，車両 Y をセンシングした結果を車両 Z が送信するメッセージを表す．

図 2 は，クエリプランの割り当てにおいて，各ノードのロードバランスや通信量の削減を目的とする従来手法と，リアルタイム制約に基づく提案手法で，オペレータ O_3 の配置を決める場合をそれぞれ表している．ここで用いられるクエリプランは，3 つの出力ストリームを持つマルチクエリである．出力ストリーム 1 は，自車両の制御に用いられ，周辺車両の処理への入力として車々間通信でブロードキャストするための車両情報としても用いられる．その End-to-End デッドラインは最短で 30 ミリ秒とする．出力ストリーム 2 には，衝突警告の情報が配信され，End-to-End デッドラインは 300 ミリ秒である．出力ストリーム 3 は，情報提供などに用いられ，End-to-End デッドラインは長く 3 秒とする．各オペレータの計算時間は 1 タプルあたり 1 ミリ秒として，オペレータの選択率（入力するタプル数に対して出力するタプル数の割合）は，O_3 のみ 0.5 で，他のオペレータでは全て 1.0 とする．各ノードの CPU 容量は全て同一とする．ノード 1 の方がオペレータの数が少ないため計算負荷が軽く，ノード 1 に O_3 を配置した方がノード 1 と 2 の間のネットワークの通信量は削減されるため，従来のクエリ割り当て手法ではノード 1 に O_3 を配置する．一方，デッドラインとしてはノード 1 のローカルクエリプランの方が厳しいため，本提案のクエリ割り当て手法では，ノード 1 のローカルクエリプランがデッドランをミスしないようにノード 2 に O_3 を配置する．以降では，それぞれのクエリ割り当てに対して，現在時刻に搭載センサから 10 個のタプル p_1 が入力され，100 ミリ秒と 200 ミリ秒経過している古いタプル p_2 と p_3 がオペレータ O_2 と O_6 の入力ストリームキューに残っている場合を想定する．

まず，**図 3** のように，従来のオペレータ配置手法では，どのようにスケジューリングしても車両制御と衝突警告のデッドラインを同時に満たせない．例えばノード 1 において，まずオペレータ O_2,O_3 で p_2 を処理し，次に O_1 で p_1 を処理すると，10 ミリ秒と $(20 \times 2 =)40$ ミリ秒それぞれかかり，車両制御の End-to-End デッドラインであ

る 30 ミリ秒を p_1 がミスしてしまう．逆に，車両制御の End-to-End デッドラインを満たすように p_2 より p_1 を先に処理すると，p_2 の衝突警告までの遅延時間は，p_2 の処理を開始するまでの遅延時間 10 ミリ秒と，p_2 のもともとの経過時間 100 ミリ秒と，p_2 をノード 1–2 で処理する時間 $(20 \times 2 + 10 \times 16 =)$200 ミリ秒との和である 310 ミリ秒となり，p_2 が衝突警告の End-to-End デッドライン 300 ミリ秒をミスしてしまう．また，p_1 と p_2 のタプル単位で個別にスケジューリングを行っても，同様に車両制御か衝突警告のデッドラインをミスしてしまう．このように従来手法のオペレータ配置では，どのようなスケジューリングを行っても，車両制御の End-to-End デッドラインを満たすと，衝突警告の End-to-End デッドラインをミスしてしまい衝突回避が間に合わない．

次に，**図 4** のように，本提案のオペレータ配置手法では，各ノードのローカルな出力ストリームに適切なサブデッドラインを指定し EDF でスケジューリングすることで，車両制御と衝突警告の両方のデッドラインを満たすことができる．この場合，まずノード 1 で p_1 より p_2 を先に 20 ミリ秒で処理し，ノード 2 では p_3 よりも p_2 を優先してオペレータを $O_3, O_4, O_7, \cdots, O_{21}$ の順に $(20 \times 2 + 10 \times 16)$ ミリ秒で実行する．この EDF スケジューリングでの p_2 の遅延時間は，これらに p_2 のもともとの経過時間 100 ミリ秒を足した時間 300 ミリ秒であり，衝突警告の End-to-End デッドライン 300 ミリ秒に収まる．また，車両制御の遅延時間は，p_2 をノード 1 で処理する遅延時間 20 ミリ秒に，p_1 の処理時間 10 ミリ秒を足した 30 ミリ秒であり，車両制御の End-to-End デッドラインも満たす．一方，このオペレータ配置であっても，FIFO スケジューリングを用いると，p_2 がノード 2 に到着してもタイムスタンプの古い p_3 を先に処理してしまい，p_2 はノード 2 で p_3 を処理するまで $(200 - 20)$ ミリ秒待たされることとなり衝突警告の End-to-End デッドラインにミスしてしまい衝突回避が間に合わない．このように，リアルタイム制約を満たすには，オペレータの配置だけでなく各ノードでのリアルタイムスケジューリングが必要となる．

5. リアルタイム制約を満たすクエリプラン割り当てとスケジューリング方式

5.1 処理全体の流れ

提案方式の処理の流れは，**図 5** のように，クエリ登録時と実行時の処理に大きく分けられる．ユーザは，クエリの登録時に各出力ストリームに対して End-to-End デッドラインを指定する．このとき，単位時間あたりの入力量が規定値を超える入力ストリームに対して，その規定値を超えないように入力データをフィルタリングする load shedder を設定できる．load shedder には，単位時間あたりに許容できるタプルの最大入力量と，タプルの価値を評価するフィルタリング条件を登録できる．

クエリの登録時に，それらの情報と物理構造から[*2]，5.2 節で述べるように，ローカルクエリプランとそれらのサブデッドラインを決定する数理計画モデルを作成する．この段階では，ローカルクエリプランのタスクの単位はオペレータとするため，サブデッドラインは各オペレータに対して割り当てられる．これにより，各ノードにローカルクエリプランと各オペレータのサブデッドラインが登録される．その後，5.3 節で述べるように，各ノードの DSMS でスケジューリングする際のタスクの単位を修正しそれらのサブデッドラインを決定する．

クエリ実行時には，各ノード上の DSMS では，EDF スケジューラが，ローカルクエリプランから抽出されたタスクをスケジューリングする．これにより，リアルタイム制約を維持するようにタプルを処理できる．これについては，5.4 節で述べる．また，load shedder を設定した入力ストリームのキューに，登録した最大入力量を超えるデータが挿入される場合は，フィルタリングの条件に基いて最大入力量に収まる分だけ価値の低いデータを削除する．

5.2 ローカルクエリプランとサブデッドラインの決定

5.2.1 概要

本節では，オペレータ配置とそれらのサブデッドラインの決定方法を説明する．2 章で述べたように，オペレータ配置やサブデッドラインの決定は計算困難な問題である．本提案では，(1) 混合整数計画問題に基づく分散ストリーム処理のオペレータ配置手法 [6] と，(2) 凸計画問題に基づく分散リアルタイムシステムのサブデッドラインの決定手法 [3] を拡張して 1 つの数理計画モデルに統合し，オペレータの配置とサブデッドラインを同時に決定する．

しかし，一般に凸関数と整数の決定変数を同時に含む数理モデルを解くことは難しい．そのため，提案方式では統合した数理計画モデルを混合整数計画問題として定義し，求めたいオペレータの配置やサブデッドラインを決定変数とし，リアルタイム制約やクエリプランの割り当てを目的や制約として，それらを満たす最適な決定変数からオペレータの配置とそれらのサブデッドラインを決定する．整数計画問題の解法には，GLPK や CPLEX などの既存のソルバを用いることで，それらに実装された branch and cut などのアルゴリズムにより効率的に最適解やその決定変数を探索することができる [15].

統合する従来方式 (1)(2) との主な違いを以下にまとめる．

(1) 通信量の削減やロードバランスなどの目的から，リアルタイム制約に変更し，オペレータのサブデッドラインを同時に最適化するよう拡張した．具体的には後述

[*2] 物理構造とは，ノードやネットワークの構成，ソースやシンク，ノード配置が固定されたオペレータなどのノード情報である．

Vol.2014-DBS-160 No.25
Vol.2014-OS-131 No.2
Vol.2014-EMB-35 No.25
2014/11/18

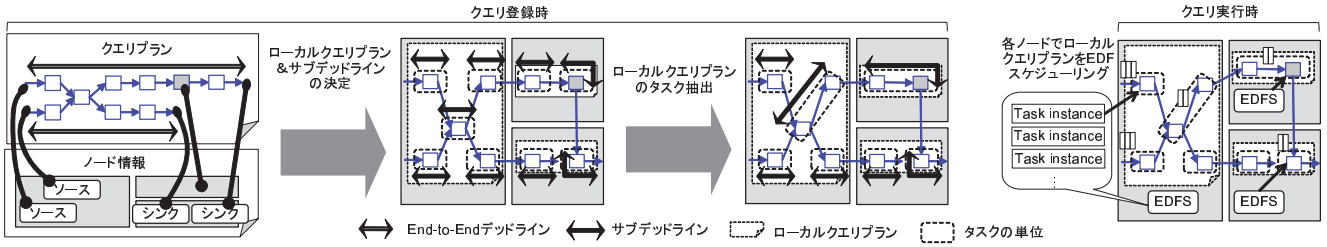

図 5 提案方式の処理の流れ
Fig. 5 Processing flow of the proposal method.

の式 (3)–(6) が該当する．

(2) 統合した数理計画モデルを整数計画問題とするため，凸関数の目的や制約が線形になるように変更した．具体的には後述の式 (3)(4)(6) が該当する．

5.2.2 数理計画モデルに用いる記号

まず，ノードやストリーム，オペレータにはインデックスを振っておく．N はノードの集合を表す．登録されたクエリプランに対して，O はオペレータの集合，S はストリームの集合，T は出力ストリームの集合とする．

オペレータ $o \in O$ に対して，$\mathrm{In}(o)$ または $\mathrm{Out}(o)$ はオペレータ o の入力または出力となるストリームの集合を表す．$\mathrm{Src}(n,s)$ は，ノード $n \in N$ にあるソースからストリーム $s \in S$ を供給できれば 1 を返し，無ければ 0 を返す．オペレータ $o \in O$ を配置できるノードの集合を $\mathrm{Node}(o)$ とする．出力ストリーム $s \in T$ に対して，$\mathrm{Pre}(s)$ は，その出力ストリーム s を提供するのに必要なオペレータの集合である．これは，クエリプランのグラフ構造をその出力ストリーム s から逆に辿ることができるオペレータの集合として抽出される．出力ストリーム $s \in T$ に対して，$\mathrm{Paths}(s)$ は，その出力ストリーム s が終点となるオペレータパスの集合である．

決定変数として，まずはオペレータ配置に関するものを述べる．$x_{n,n',s} \in \mathbb{B}$ はノード n からノード n' にストリーム s を流すか否かを表す．$z_{n,o} \in \mathbb{B}$ はノード n にオペレータ o を配置するか否かを表す．$y_{n,s} \in \mathbb{B}$ はノード n でストリーム s を利用可能か否かを表す．また，クエリプランで同一のストリームが複数ノードをまたいで流れるとき，ループしないようにするための特殊な変数として $l_{n,s} \in \mathbb{R}_+$ を用意する．次に，サブデッドラインの決定に関する決定変数を述べる．$s \in T, o \in \mathrm{Pre}(s)$ に対して $d_{s,o} \in \mathbb{R}_+$ はオペレータ o のサブデッドラインである．絶対値の付いた目的関数を線形にするために，$s \in T, o \in \mathrm{Pre}(s)$ に対して $h^+_{s,o}, h^-_{s,o} \in \mathbb{R}_+$ を用意する．

既知の情報として，$s \in T$ に対して d_s は End-to-End デッドラインである．$o \in O$ に対してオペレータ o の計算時間を c_o とする．c_o はクエリに入力されるデータ量に依存することが多いが，ここでは最適値を得るための入力データ量を固定してサブデッドラインとオペレータ配置を決める．そのため c_o は 5.2 節では固定値とする．なお，$s \in T$ に対して $c'_s = \sum_{o \in \mathrm{Pre}(s)} c_o$ と表記し，簡略化のため各ノードの CPU 容量は同一とする．

5.2.3 目的の定式化

サブデッドラインを直接最適化することは困難であるため [6]，分散リアルタイムシステムで用いられる方法の 1 つである，Normalized laxity ratio（NLR）を各サブデッドラインで偏りなく一定に近づけることを本最適化の目的とする．NLR とは，サブデッドラインとタスクの計算時間の差として定義される余裕時間をそのタスクの計算時間で正規化した値であり，全てのサブデッドラインで NLR を完全に一定とすると，各サブデッドライン $d'_{s,o}$ は式 (1) のように決定できる．

$$d'_{s,o} = c_o \left(\frac{d_s}{c'_s} \right), \forall s \in T, o \in \mathrm{Pre}(s) \qquad (1)$$

また，この最適化の目的は以下のように定式化できる．

$$\min \left\{ \sum_{s \in T, o \in \mathrm{Pre}(s)} \left| \frac{d_s - c'_s}{c'_s} - \frac{d_{s,o} - c_o}{c_o} \right| \right\} \qquad (2)$$

これを整数計画問題とするため，制約式 (4) を追加することで，目的式 (2) は式 (3) のように定式化できる．

$$\min \left\{ \sum_{s \in T, o \in \mathrm{Pre}(s)} \left(h^+_{s,o} + h^-_{s,o} \right) \right\} \qquad (3)$$

$$\frac{d_s - c'_s}{c'_s} - \frac{d_{s,o} - c_o}{c_o} = h^+_{s,o} + h^-_{s,o}, \forall s \in T, o \in \mathrm{Pre}(s) \quad (4)$$

5.2.4 リアルタイム制約の定式化

サブデッドラインが満たすべき End-to-End のデッドラインの制約としては，以下がある．

$$\sum_{o \in \{o_1, o_2 \cdots, o_k\}} d_{s,o} \leq d_s, \forall s \in T, \langle o_1, o_2, \cdots, o_k \rangle \in \mathrm{Paths}(s) \qquad (5)$$

加えて，各ノードに搭載されたローカルクエリプラン全体でデッドラインを満たす必要がある．もし各サブデッドラインの間でデッドラインの大小関係が分かっていれば，本提案ではシングルプロセッサの各ノードでローカルに EDF

スケジューリングを行うため，$\forall s \in T, o \in \mathrm{Pre}(s)$ に対して $d_{s,o}$ の昇順に (s,o) の組を $(s_1,o_1),(s_2,o_2),\cdots,(s_L,o_L)$ と並べることで，以下の式からデッドラインを満たすか分かる [4].

$$\sum_{l' \le l} z_{n,o_{l'}} \times c_{o_{l'}} \le d_{s_l,o_l}, 1 \le l \le L, \forall n \in N \qquad (6)$$

各サブデッドラインは本最適化問題の決定変数のため，大小関係はこの時点で正確には分からない．しかし，本最適化問題の目的である最小化が十分に行われているのであれば，$d'_{s,o}$ の大小関係が $d_{s,o}$ でも概ね成り立つと考えて，$d_{s,o}$ の代わりに $d'_{s,o}$ の大小関係を用いて式 (6) で各ノードのデッドライン制約を表す．

5.2.5 オペレータ配置制約の定式化

クエリプランをノードに割り当てる時には，オペレータとストリームの接続からなるクエリグラフの構造を正しく保つ必要がある．そこで，各ノードで利用可能なストリームに着目して決定変数 $x_{n,n',s}, z_{n,o}, y_{n,s}$ の間の依存関係を定式化する．

$$y_{n,s} \le \sum_{n \in N} x_{n,n',s} + \sum_{o \in O, \mathrm{Out}(o)=s} z_{n',o} + \mathrm{Src}(n',s), \quad \forall n' \in N, s \in S \qquad (7)$$

$$z_{n,o} \le y_{n,s}, \forall n \in N, o \in O, s \in \mathrm{In}(o) \qquad (8)$$

$$x_{n,n',s} \le y_{n,s}, \forall n, n' \in N, s \in S \qquad (9)$$

式 (7) は，ストリーム s をノード n' で利用するには，(A) ノード n からノード n' にストリーム s が流されるか，(B) ノード n' にあるオペレータよりストリーム s が出力されるか，(C) ノード n' にあるソースからストリーム s を入力できるかのいずれかであることを表しており，右辺の 1–3 項目がそれぞれ条件 (A)–(C) に対応している．式 (8) は，オペレータをノード n に配置するには，そのオペレータの入力するストリームがノード n で利用可能であることを表す．式 (9) は，ストリーム s をノード n から送信するには，ノード n でストリーム s が利用可能であることを表す．

ノードの配置を自由に決められないオペレータに対しては，式 (10) の制約を付ければよい．

$$z_{n,o} = 0, \forall o \in O, n \in N \setminus \mathrm{Node}(o) \qquad (10)$$

同様にストリームを自由に流せないノード間の制約なども容易に追加できる．

[3] と同様に，クエリプランでは 1 つのストリームであっても，複数のノードをリレーするような配置にも対応できる．このとき，1 つのストリームがノード間でループしないように制約式 (11) を導入する．

$$l_{n,s} \ge l_{n',s} + 1 - M \times (1 - x_{n,n',s}), \forall n, n' \in N, s \in S \qquad (11)$$

ここで M は，$M > |N| - 1$ を満たす大きな固定値とする．これは，ノード n のストリーム s に対してあるレベル $l_{n,s}$ を定義し，ストリームをリレーするとレベルが下がるようにし，ストリームのリレーをレベルが上がる方向に許さないようにしている，と解釈できる．

5.2.6 数理計画モデルの定式化

以上の議論から，本提案の数理計画モデルは，制約式 (4)–(11) のもとで式 (3) を目的とする整数計画問題として定義できる．これにより最適な決定変数 $d_{s,o}$，$z_{n,o}$，$x_{n,n',s}$ を得る．マルチクエリの場合では，異なる s に対して $d_{s,o}$ を複数持つオペレータ o が存在する．この場合，オペレータ o のサブデッドラインを $\min_{s \in T}\{d_{s,o}; o \in \mathrm{Pre}(s)\}$ とする．

5.3 ローカルクエリプランにおけるタスク抽出

本節では，各ノードのローカルクエリプランとオペレータのサブデッドラインが 5.2 節より分かっているもとで，各ローカルクエリプランからタスクの単位を抽出する方法を簡単に述べる．このタスクの単位は，各ノードの DSMS で EDF スケジューリングする際に用いられる．最初は，ローカルクエリプランの各オペレータが 1 つのタスクの単位である．次に，以下を繰り返すことで得られるオペレータパスの一部をタスクの単位とする．

- 出力するストリーム数が 1 のオペレータと入力するストリーム数が 1 のオペレータ同士をマージする．
- 出力するストリーム数が 2 以上のオペレータであっても，それらのストリームからパスとして辿れるローカルクエリプランの出力ストリームに対して End-to-End デッドラインが最小となれば，入力するストリーム数が 1 のオペレータとマージする．

タスクの単位 T のサブデッドライン d_T は，T を構成するオペレータのサブデッドラインの和として定義される．

5.4 各ノード上でのリアルタイムスケジューリング

ローカルクエリプランにおけるタスクの単位 T にタプル p が入力されたとき，EDF スケジューラはタスクインスタンス (p,T) を生成する．つまり，T を構成するあるオペレータの実行後に，他のタスクの単位 T' が入力とするストリームにタプル p が挿入されたとき，そのタスクインスタンス (p,T') を生成する．タスクインスタンスの絶対デッドライン $D_{(p,T)}$ を式 (12) から計算する．

$$D_{(p,T)} = r_{(p,T)} + d_T \qquad (12)$$

なお，$r_{(p,T)}$ は (p,T) の生成した時刻を表し，d_T は T のサブデッドラインを表す．タスクインスタンスを生成したとき，EDF スケジューラは，絶対デッドラインの最も早いタスクインスタンスに対して，タスク T を構成するオペレータを順に呼び出し，タプル p を処理する．このとき，現在実行しているタスクインスタンスの絶対デッドラインが最も早ければ，それを継続して処理する．

Vol.2014-DBS-160 No.25
Vol.2014-OS-131 No.2
Vol.2014-EMB-35 No.25
2014/11/18

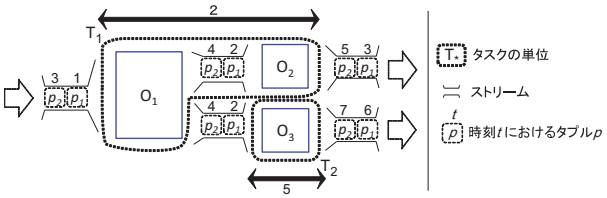

図 6 ローカルクエリプランの EDF スケジューリング
Fig. 6 EDF scheduling of a local query plan.

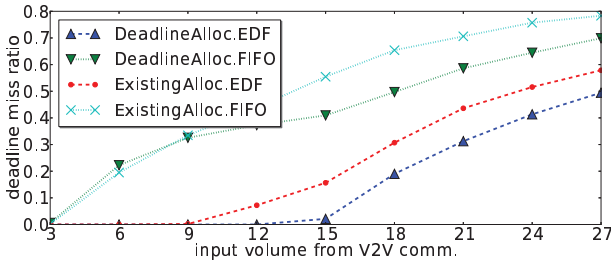

図 7 車々間通信からの入力量の変化に対するデッドラインミス率
Fig. 7 DMRs for changes of input volume from V2V comm.

図 6 のように，3 つのオペレータ O_1, O_2, O_3 からなるローカルクエリプランの簡単な例を用いて，スケジューリングする方法を説明する．ここでは，全てのオペレータの計算時間は 1 とし，タスクの単位 $T_1 = (O_1, O_2)$ のサブデッドラインを 2 とし，タスクの単位 $T_2 = (O_3)$ のサブデッドラインを 5 とする．時刻 1 に，タプル p_1 が O_1 の入力するストリームに挿入されると，タスクインスタンス (p_1, T_1) が生成される．時刻 2 では，O_3 の入力に p_1 が到着し，タスクインスタンス (p_1, T_2) が生成される．時刻 3 では，タスクインスタンス (p_1, T_1) の処理が完了する．また，O_1 の入力に新しいタプル p_2 が到着し，タスクインスタンス (p_2, T_1) が生成される．この時，先に生成されたタスクインスタンス (p_1, T_2) の絶対デッドライン $2 + 5$ よりも，後から生成された (p_2, T_1) の絶対デッドライン $3 + 1$ の方が早い．そのため，(p_2, T_1) を先に処理する．その後，時刻 5 において (p_2, T_1) の処理が完了した後で，(p_1, T_2) と (p_2, T_2) を順に処理する．これにより，全てのタスクインスタンスがサブデッドラインを満たして処理される．なお，これを FIFO でスケジューリングすると，p_2 が T_1 のサブデッドラインをミスする．

従来方式のようにタプルバッチングを用いて [9]，ローカルクエリプランの入力ストリームに挿入されるタプルをバッチして，その単位でタスクインスタンスを生成してスケジューリングすることで，スケジューリングのオーバーヘッドを削減できる．

6. 評価

6.1 評価方法

評価には図 2 のクエリを用いた．各出力ストリームの End-to-End デッドラインやオペレータの平均的な計算時間，自由に配置を決められるオペレータも 4 章で説明したとおりである．リアルタイム制約の評価には，全ての出力ストリームに挿入されたタプル数 X に対して，各出力ストリームの End-to-End デッドラインを満たしたタプル数 Y の割合 Y/X として，デッドラインミス率（DMR）を用いた．

評価項目は，（A）分散ストリーム処理のクエリ割り当て（オペレータ配置）方式と（B）ローカルクエリプランのスケジューリング方式の 2 つである．（A）については，従来方式としてネットワークの通信量の削減や各ノードの CPU 使用量のロードバランスを行う SQPR[3] を従来方式として比較評価し，提案方式と従来方式をそれぞれ DeadlineAlloc と ExistingAlloc と記述する．（B）については，デッドラインの無い FIFO スケジューリングを従来方式として比較評価し，提案方式と従来方式をそれぞれ EDF と FIFO と記述する．つまり，比較評価する組み合わせは，DeadlineAlloc.EDF，ExistingAlloc.FIFO，ExistingAlloc.EDF，DeadlineAlloc.FIFO の 4 種類となる．但し，ExistingAlloc では，ローカルクエリプランのサブデッドラインを決めることができないため，ローカルクエリプランに対して EDF スケジューリングを本来適用できない．ここでは，オペレータ配置手法の違いによる性能を比較評価するため，ExistingAlloc.EDF では DeadlineAlloc.EDF から算出されるサブデッドラインを用いた．なお，いずれの方式もタプルバッチングをしていない．

搭載センサとセンタからの入力タプル数はそれぞれ 10 個と 100 個で固定し，車々間通信からの入力タプル数を変化させて DMRs を測定し，各測定に対して 50 回の試行を行った．各ソースからの入力タイミングは，ノード 1 では車々間通信の直後に搭載センサからタプルが入力され，ノード 2 ではノード 1 からデータが受信される前にセンタからタプルが入力される．なお，5.2 節の数理計画モデルは，車々間通信から一度に 15 タプル入力された時のオペレータの計算時間で最適化された．評価マシンには，CPU が 700MHz でメモリが 512MB の 2 台の Raspberry Pi を用いた．

6.2 評価結果

図 7 より，提案方式は ExistingAlloc.FIFO や DeadlineAlloc.FIFO よりも良好であった．車々間通信の入力量が 15 タプルのときの DMRs では，提案方式が 2.11%に対して，ExistingAlloc.FIFO と DeadlineAlloc.FIFO が 55.5%と 40.9%であった．提案方式では，センタから入力されたタプルを O_5 と O_6 で処理するよりも先に，EDF と式 (12) に基づき，サブデッドラインの短い O_7–O_{21} で後から到着したタプルを先に実行する．これにより，End-to-End デッドラインが出力ストリーム 3 よりも短い，出力ストリーム

2 に対する遅延時間が削減されるためである．

また，図 7 より提案方式が ExistingAlloc.EDF よりも良好であった．車々間通信の入力量が 15 タプルのときの DMRs では，提案方式の 2.11%に対して，ExistingAlloc.EDF は 15.6%であった．提案方式では，オペレータ O_3 を End-to-End デッドラインが短い出力ストリームを含むノード 1 の代わりに，ノード 2 に配置する．これにより，車々間通信からの入力量が増えても，ノード 1 では O_3 の計算時間が増加しないことから，出力ストリーム 1 でデッドラインミスが増加しにくいためである．

7. まとめ

本稿では，End-to-End デッドラインの制約を持つ分散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提案した．本方式は，ノードに分散するローカルクエリプランのサブデッドラインを決定し，各ローカルクエリプランを EDF でスケジューリングする．また，提案方式では，サブデッドラインの決定とクエリプランのノードへの割り当てをリアルタイム制約に基づき同時に最適化する整数計画問題としてモデル化した．簡単なクエリに対して，ロードバランスや通信量の削減を目的とする従来のオペレータ配置手法や，ローカルクエリプランを FIFO でスケジューリングする従来方式と比較し，提案方式がリアルタイム制約の指標であるデッドラインミス率を削減することを確認した．

今後の課題として，評価するクエリプランのバリエーションを増やした有効性の確認や，分散リアルタイムシステムにおける従来方式との性能比較があげられる．また，自動車などのより具体的なアプリケーションに適用した場合の有効性の確認や，ノード数やオペレータ数が増加した場合における数理計画モデルのスケーラビリティの評価なども，今後も実施する．

謝辞：本研究の一部は SCOPE(121806015) と科研費 (25240007) の助成を受けたものである．

参考文献

[1] Golab, L. and Özsu, M. T.: Issues in Data Stream Management, *SIGMOD Rec.*, Vol. 32, No. 2, pp. 5–14 (2003).
[2] Lakshmanan, G. T., Li, Y. and Strom, R.: Placement Strategies for Internet-Scale Data Stream Systems, *IEEE Internet Computing*, Vol. 12, No. 6, pp. 50–60 (2008).
[3] Kalyvianaki, E., Wiesemann, W., Vu, Q. H., Kuhn, D. and Pietzuch, P.: SQPR: Stream Query Planning with Reuse, *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering*, IEEE Computer Society, pp. 840–851 (2011).
[4] Buttazzo, G. C.: *Hard Real-time Computing Systems: Predictable Scheduling Algorithms And Applications (Real-Time Systems Series)*, Springer-Verlag TELOS (2004).
[5] Liu, J.: *Real-Time Systems*, Prentice Hall (2000).
[6] Lee, J., Shin, I. and Easwaran, A.: Convex optimization framework for intermediate deadline assignment in soft and hard real-time distributed systems, *Journal of Systems and Software*, Vol. 85, No. 10, pp. 2331–2339 (2012).
[7] Chakravarthy, S. and Jiang, Q.: *Stream Data Processing: A Quality of Service Perspective Modeling, Scheduling, Load Shedding, and Complex Event Processing*, Springer Publishing Company, 1st edition (2009).
[8] Babcock, B., Babu, S., Motwani, R. and Datar, M.: Chain: operator scheduling for memory minimization in data stream systems, *Proc. of the 2003 ACM SIGMOD international conference on Management of data*, pp. 253–264 (2003).
[9] Carney, D., Çetintemel, U., Rasin, A., Zdonik, S., Cherniack, M. and Stonebraker, M.: Operator Scheduling in a Data Stream Manager, *Proc. of the 29th International Conference on Very Large Data Bases*, pp. 838–849 (2003).
[10] Kulkarni, D., Ravishankar, C. V. and Cherniack, M.: Real-time, Load-adaptive Processing of Continuous Queries over Data Streams, *Proceedings of the Second International Conference on Distributed Event-based Systems*, ACM, pp. 277–288 (2008).
[11] Zhou, Y., Wu, J. and Leghari, A. K.: Multi-query Scheduling for Time-critical Data Stream Applications, *Proceedings of the 25th International Conference on Scientific and Statistical Database Management*, New York, NY, USA, ACM, pp. 15:1–15:12 (2013).
[12] Jonsson, J. and Shin, K.: Deadline assignment in distributed hard real-time systems with relaxed locality constraints, *Distributed Computing Systems, 1997., Proceedings of the 17th International Conference on*, pp. 432–440 (1997).
[13] Baruah, S. K., Rosier, L. E. and Howell, R. R.: Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic, Real-time Tasks on One Processor, *Real-Time Syst.*, Vol. 2, No. 4, pp. 301–324 (1990).
[14] ETSI: Intelligent Transport Systems; V2X Applications; Part 3: Longitudinal Collision Risk Warning application requirements specification (2013).
[15] Wolsey, L.: *Integer Programming*, Wiley (1998).