DEIM Forum 2014 D3-5

リアルタイム性を考慮した車載システム向けDSMSのクエリ処理効率化

勝沼　聡†*　本田　晋也†　高田　広章†

† 名古屋大学大学院情報科学研究科　〒 464–8601 名古屋市千種区不老町
* 日立製作所 中央研究所
E-mail: †{katsunuma,honda,hiro}@ertl.jp

あらまし　車載システムの複雑化に伴う開発コスト削減を目的とし，車載システムへの DSMS の適用を検討している．車載システムにおいて，従来の汎用システム向け DSMS のクエリ処理共有方式を適用した場合，優先度が逆転する時間（優先度逆転時間）が大きくなり，リアルタイム性が保つのが難しい．そこで本稿で提案するクエリコンテキスト共有方式では，優先度逆転時間を削減するために，アプリケーション毎に独立して決められた優先度でクエリを実行する．そして処理が等価なクエリ間で，クエリの処理過程で必要となるデータ（コンテキスト）を共有可能とし，あるクエリの実行がスケジューリングの関係上，中断され，他のクエリが実行された場合，前のクエリのコンテキストを引き継ぐことで処理時間を削減する．クエリコンテキスト共有方式を評価した結果，従来方式であるクエリ処理共有方式と異なり優先度逆転時間が低減され，また処理時間の増加も 2.13% に留まりクエリの処理効率化の効果を示した．
キーワード　データストリーム管理システム，クエリ処理効率化，リアルタイム，車載システム，RTOS

Realtime-Aware Efficient Query Processing for Automotive DSMS

Satoshi KATSUNUMA†*, Shinya HONDA†, and Hiroaki TAKADA†

† Graduate School of Information Science, Nagoya University Furo-cho, Chikusa-ku, Nagoya, 464–8601 Japan
* Central Research Lab., Hitachi Ltd.
E-mail: †{katsunuma,honda,hiro}@ertl.jp

1. はじめに

車載システムでは，アプリケーションの生産性向上に向け，AUTOSAR がデファクトとなり，プラットフォームの導入が進んでいる．AUTOSAR ではソフトウェアコンポーネントのインタフェースを共通化し，様々な ECU で共通のアプリケーションを動かせるようにする．

我々が提案している車載データ統合 PF [1] [2] [3] では，AUTOSAR 上で，様々な ECU で共通するデータ処理を定義し，そのデータ処理を様々なアプリケーションが利用可能とする．車載データ統合 PF ではデータ処理をデータストリーム管理システム（以下，DSMS）のクエリとして記述する．そして同一 ECU 上の複数のアプリケーションから呼ばれるクエリの処理を共有することで，リソース制約が厳しい車載システム上で処理量を低減することが期待されている．汎用システム向けの DSMS におけるクエリ処理共有方式 [7] [8] では，複数のアプリケーション向けのクエリの処理を一度だけ実行し，その処理結果を各アプリケーションに渡すことで処理量を低減している．

しかし車載システムではリアルタイム性が求められるため，優先度逆転が発生する DSMS のクエリ処理共有方式を適用することは難しい．車載システムでは RTOS が搭載され，アプリケーションは RTOS のタスクに割り当てる．そして各タスクをリアルタイム性の要件を満たすために，優先度を設定しプリエンティブなマルチタスク環境で動作させる．しかしクエリ処理共有方式では，そのクエリの処理に対して高い優先度を割り当てた場合には優先度逆転が発生し，クエリの処理が他のタスクの実行を妨げ，またクエリの処理に対して低い優先度を割り当てた場合には他のタスクがクエリの処理を妨げる．そのためリアルタイム性を保ちつつ処理を共有するのは難しい．

本稿ではリアルタイム性の要件を満たすため，車載システムのスケジュールの元で，クエリの処理を効率化するクエリコンテキスト共有方式を提案する．クエリコンテキスト共有方式では，クエリの処理を，その処理結果を利用するアプリケーションと同一タスクに割り当て，同一優先度で実行する．そして異なるタスクに割り当てられたクエリに対し処理が等価なクエリ間で，キューやウィンドウ，オペレータの計算に必要とする状態などクエリの処理過程で必要となるデータ（以下，コンテキスト）を共有し，タスク切替え時にクエリのコンテキストを用

いて，切替え先のクエリに処理を引き継ぐ．これにより優先度逆転の発生期間を，タスク間で共有するコンテキストへのアクセス時のみに抑制しつつ，クエリの処理をタスク間で引き継ぐことで処理時間を削減する．

本稿の構成は以下の通りである．まず 2 節で車載システムの要求仕様及び，車載システムへの DSMS 適用について述べ，3 節で従来方式であるクエリ処理共有方式の課題を述べる．そして 4 節で提案方式であるクエリコンテキスト共有方式を説明し，5 節でその評価について述べる．最後に 6 節でまとめと今後の課題を述べる．

2. 車載システムの要求仕様と DSMS 適用

本節では車載システムに求められる仕様と，車載システムへの DSMS 適用時の狙いについて述べる．

2.1 車載システム

車載システムにおけるスケジューリングの方法と，そのスケジューリングを実現するための RTOS の活用方法を述べる．

2.1.1 スケジューリングの方法

車載システムではリアルタイム性を高めるために，図 1(a) に示す，下記特性 1，2 を持つスケジューリングを行う．

- 特性 1. 優先度ベースのプリエンティブスケジューリング

車載システムは一般的に，一つの ECU に複数のタスクが割り当てられる，マルチタスクの構成になっており，各アプリケーションのリアルタイム性の要件に合わせて，アプリケーションを割り当てたタスクに対してあらかじめ優先度を設定する．そして高い優先度のタスクよりも低い優先度のタスクが実行される優先度逆転が発生する時間を小さくする，プリエンティブなスケジューリングを行う．

- 特性 2. 周期実行またはデッドラインを持つイベント駆動

車載システムでは各アプリケーションを周期実行または，デッドラインを持つイベント駆動で動作する．アプリケーションが扱う，車車間通信や自車のセンサのデータは一定の間隔で更新され，周期実行を行うアプリケーションではその周期で取得できるデータを入力とした処理を行い，処理が完了した場合には，次の周期まで待機する．また各周期の終わりをデッドラインとし，周期の間に処理が完了しなかった場合には，その周期でのアプリケーションの処理を中止する．イベント駆動のアプリケーションでは，デッドラインを設定し，イベントが発生してからデッドラインの時刻までに取得できるデータを入力とし処理する．そしてデッドラインの時刻を超えたらアプリケーションの処理を中止する．

2.1.2 RTOS の活用

車載システムでは，特性 1，2 を持つスケジューリングを行うために RTOS を活用し，一つの ECU に搭載される全てのアプリケーションに対して統一的なスケジューリングを行う．すなわち各アプリケーションを RTOS のタスクに割り当て，タスクに対し優先度や実行タイミング，デッドライン等を設定することにより，アプリケーションの処理の開始や，他のアプリケーションへの処理の切り替え，処理の中止を RTOS で制御する．なお車載システム向けの RTOS ではメモリ保護を用いることがほとんどなく，タスク間でデータを共有することが可能である．

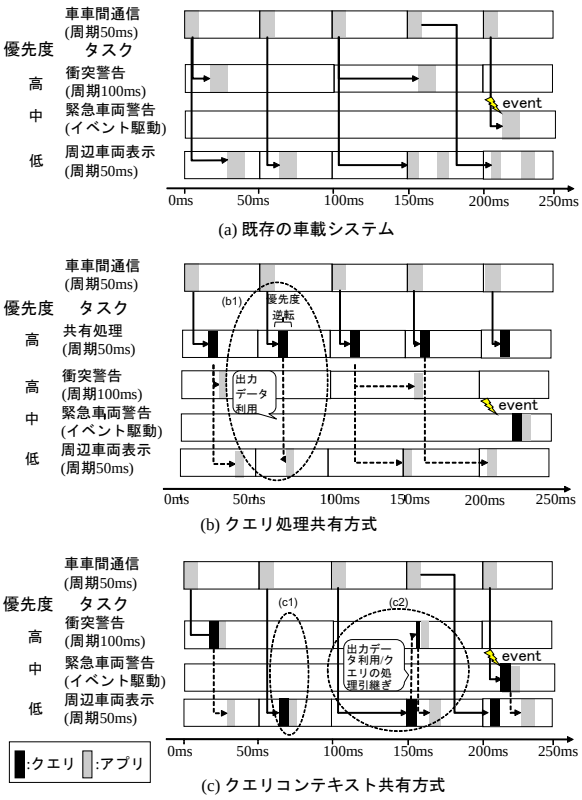

図 1 車載システムにおけるスケジューリングの例

2.2 車載システムへの DSMS 適用の狙い

車載データ管理 PF では，車載データ処理を DSMS のクエリとして記述し，車載システム向け DSMS（eDSMS）[2] [4] 上で実行する．そしてリソース要件が厳しい車載システムにおけるクエリの処理量を低減するために，DSMS を適用することで複数のアプリケーションが使用するクエリの処理効率化を目指している．図 2 に示す例では，車車間通信データから，走行している他車の平均速度等を算出する他車走行情報配信クエリを，衝突警告及び緊急車両警告，周辺車両表示の三つのアプリケーションが使用している．このような場合に複数アプリケーション向けの共通するクエリの処理を一度で行うなどの，クエリの処理効率化が求められる．

3. 従来手法の課題

クエリ処理効率化を実現するための，クエリ処理共有方式の概要と，車載システム適用時の課題を述べる．

3.1 クエリ処理共有方式

クエリ処理共有方式 [7] [8] では，複数のアプリケーションが共通に使用するクエリの処理を纏めて一度だけ実行することで，処理量の低減を行う．例えば渡辺らの方式 [8] では，クエリの処理を一つにするために，実行タイミングが異なるが実行時間に重なりを持つ複数のクエリに対して，各クエリの実行時間を合わせた時間を実行時間とする，クエリの処理を行う．

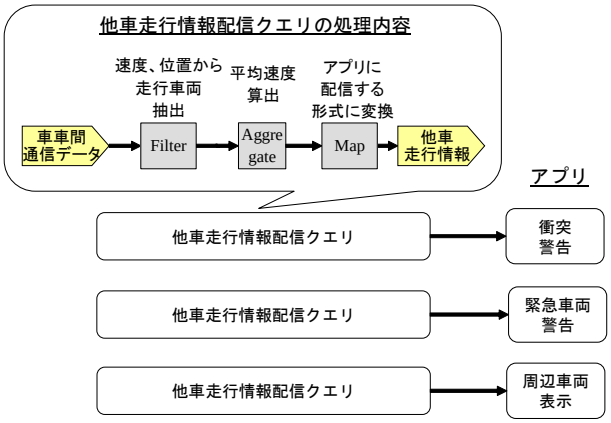

図 2　車載システムへの DSMS 適用

3.2　車載システム適用に向けた課題

車載システムでは特性 1 に示すように，処理内容が同じクエリであっても，その処理結果を利用するアプリケーションの優先度が異なる可能性があるため，クエリの処理を共有するためにいずれかのアプリケーションの優先度で実行する必要がある．しかし共有処理を割り当てるタスクに，いずれかのアプリケーションの優先度を設定したとしても優先度逆転が起こる可能性がある．

図 1(b) は衝突検知と周辺車両表示アプリケーションがクエリの処理を共有するため，クエリの処理をアプリケーションとは別のタスクに割り当てる場合である．この場合，各アプリケーションがクエリの処理結果を利用するため，クエリの処理を割り当てるタスクには，優先度が高い衝突検知アプリケーションの優先度を割り当て，また動作周期を，周期が短い周辺車両表示アプリケーションの周期 50ms で実行する必要がある．しかしその場合には，図 1(b1) に示すように優先度「低」の周辺車両表示アプリケーションのみがクエリの処理結果を利用するケースでもクエリの処理が優先度「高」で実行されるため，そのクエリの処理により優先度が「中」の緊急車両警告アプリケーションの実行が妨げられ，優先度逆転が起こりうる．

またクエリの処理を優先度が高い別のタスクとして実行する場合，クエリの処理終了後に，アプリケーションの処理を開始する．そのためアプリケーションで，クエリの処理結果を利用する前に事前に処理が必要である場合，その処理の開始が遅れる．またクエリで処理しつつ，その結果結果を順次，アプリケーションが受け取り，クエリの処理と並行して処理を実行することもできない．さらにクエリの処理を，アプリケーションと異なる新たなタスクに割り当てるため，その新たなタスクのスタック領域などが必要となりメモリ使用量が大きくなる．従ってクエリ処理共有方式を，車載システムに適用することは困難である．

4.　クエリコンテキスト共有方式

4.1　コンセプト

本稿では車載システムにおけるリアルタイム性を考慮し，車載システムのスケジューリングの元で，処理量を低減するためにクエリ処理効率化を行う，クエリコンテキスト共有方式を提案する．

- 車載システム向けスケジューリング

提案方式では RTOS を用いて車載システム向けのスケジューリングを実現する．車載システムでは，特性 1 に示すようにアプリケーションごとに優先度が異なるため，提案方式では処理が等価なクエリであっても図 1(c) に示すようにクエリを，クエリを利用するアプリケーションと同一のタスクに割り当て，同一優先度で処理を実行する．また特性 2 に示すようにデッドラインの時刻を過ぎた場合には，クエリの処理中であってもタスクを終了する必要があるため，次の周期でクエリの処理を継続できるよう，クエリのコンテキストを引き継ぐ．

- クエリの処理効率化

前述のように処理が等価なクエリであっても異なるタスクに割り当てられる．そこで処理が等価なクエリのコンテキストを共有し，スケジューリングの都合上，中断され，他のタスクが実行された際に，前のクエリの出力データを切り替わり先のアプリケーションが利用する．また前のクエリのコンテキストを利用して，前のクエリの処理を切り替わり先のタスクのクエリが引き継ぐ．例えば図 1(c1) では，周辺車両表示用のタスクにおけるクエリの出力データを，衝突警告アプリケーションが利用し，また，周辺車両表示用のタスクにおけるクエリの処理を，衝突警告用のタスクのクエリが引き継ぐ．

クエリコンテキスト共有方式では，優先度逆転の発生期間を，タスク間で共有するコンテキストへのアクセス時のみに抑制することで優先度逆転時間を削減する．またタスク切り替え時に，クエリの出力データを切り替え先のタスクのアプリケーションが利用し，またクエリの処理を切り替え先のタスクのクエリが引き継ぐことで処理時間を削減する．ただし提案方式ではクエリ処理共有方式と比較し，異なるタスクに割り当てられたクエリのコンテキストの共有が必要となるため，コンテキストを管理するための処理オーバヘッドが発生する．

4.2　構　　成

提案方式の構成を図 3 に示す．提案方式は RTOS（図 3（a））, eDSMS（図 3（b））, アプリケーション（図 3（c））から構成される．提案方式では，従来の eDSMS [4] と同様に，Filter, Aggregate, Map など Borealis [10] をベースとしたオペレータによりクエリの処理が行われるため，クエリのセマンティックスは従来の eDSMS に準じており，またクエリ内のオペレータのスケジューリングについても従来の eDSMS と同様である．しかしクエリ間及び，クエリとデータ入力部のスケジューリングの方法や，アプリケーションからのクエリの呼び出し方法は，下記に示すように従来の eDSMS とは異なる．

提案方式では RTOS にはあらかじめタスクが登録され，タスク管理テーブル（図 3（a1））を参照しタスクを呼び出す．各タスクではアプリケーションが DSMS スケジューラ（図 3（b1））を呼び出し，DSMS スケジューラがデータ入力部（図 3（b2））またはクエリ（図 3（b3））を呼び出す．データ入力部は入力データを入力キュー（図 3（b5））に格納し，クエリの処理により入力キューから入力データを取り出し，各オペレータで実行

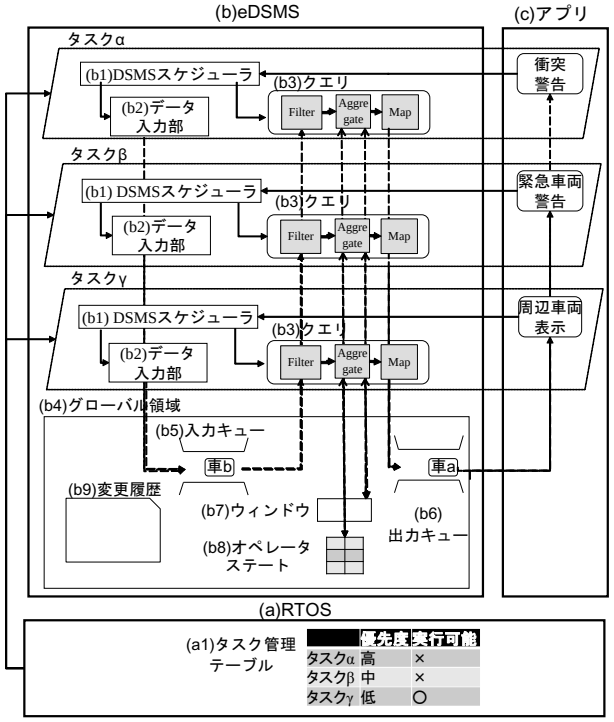

図 3　クエリコンテキスト共有方式の構成

後，出力データを出力キュー（図 3（b6））に格納する．そしてアプリケーションに制御が戻ると，出力キューから出力データを取り出し処理を行う．

また提案方式では，入力キュー，出力キューなどのキューや，ウィンドウ（図 3（b7））及び，Aggregate オペレータなど各オペレータの状態を保持するオペレータステート（図 3（b8））などにある全てのコンテキストを，異なるタスクからアクセスされるグローバル領域（図 3（b4））に格納し，異なるタスクのクエリ間でコンテキストを共有する．そしてクエリ処理効率化のために，タスクの切り替わり時に，出力キューに格納される，異なるタスクのクエリの出力データをアプリケーションが利用する．またクエリ実行中にタスクが切り替わった場合に，DSMS スケジューラでロールバックを行い，変更履歴（図 3（b9））に基づきコンテキストを入力データの実行前に戻し，切り替わったタスクで，コンテキストを引き継ぎクエリの処理を継続する．

4.3　基本的な動作

図 1(c1) に示す提案方式においてクエリのコンテキストの引継ぎがない場合の動作を，図 4 を用いて説明する．

4.3.1　アプリケーション構成とタスクへの割当て

まず具体例として挙げるアプリケーションの構成と，そのタスクへの割当てについて説明する．アプリケーションは衝突検知，緊急車両警告，周辺車両表示であり，図 3 に示すように，各アプリケーションは他車走行情報配信クエリと共にそれぞれタスク α，β，γ に割り当てる．そして図 1（c）に示すアプリケーションの優先度に合わせて，タスク α，β，γ の優先度をそれぞれ「高」,「中」,「低」とし，また実行タイミングは周期実行（周期 100ms），イベント駆動，周期実行（周期 50ms）とする．

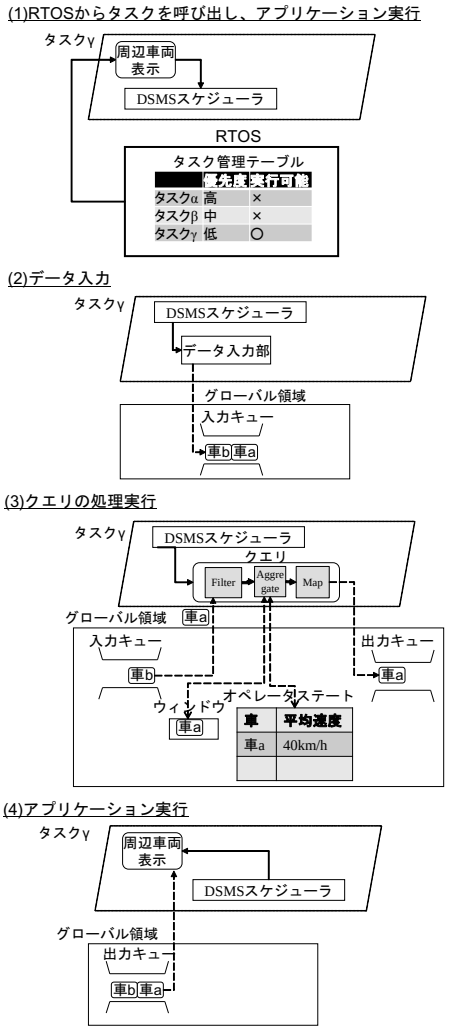

図 4　基本的な動作

4.3.2　アプリケーションの実行

RTOS からタスクが呼び出されると，そのタスクに割り当てられたアプリケーションが実行される．例えば図 4（1）では，RTOS からタスク γ が呼ばれ，タスク γ に割り当てられた周辺車両表示アプリケーションを実行する．そしてアプリケーションから DSMS スケジューラを呼び出すことで，クエリの処理が実行され，DSMS スケジューラの処理終了後に，アプリケーションで出力キューに格納された，クエリの出力データを取得する．例えば図 4（1）では周辺車両表示アプリケーションが DSMS スケジューラを呼び出し，その後，クエリの処理実行後に図 4(4) に示すように出力キューに格納されたデータ車 a，車 b を衝突検知アプリケーションが取り出し処理を行う．

4.3.3　DSMS スケジューラ

アプリケーションから呼ばれる DSMS スケジューラでは，データ入力部及び，クエリの処理を呼び出す．データ入力部は，図 4(2) に示すように入力データが入力キューに格納されていない場合に呼び出す．またクエリの処理は，図 4(3) に示すように入力データの入力キューへの格納後に呼び出す．

4.3.4　データ入力

DSMS スケジューラから呼び出されるデータ入力部では，車

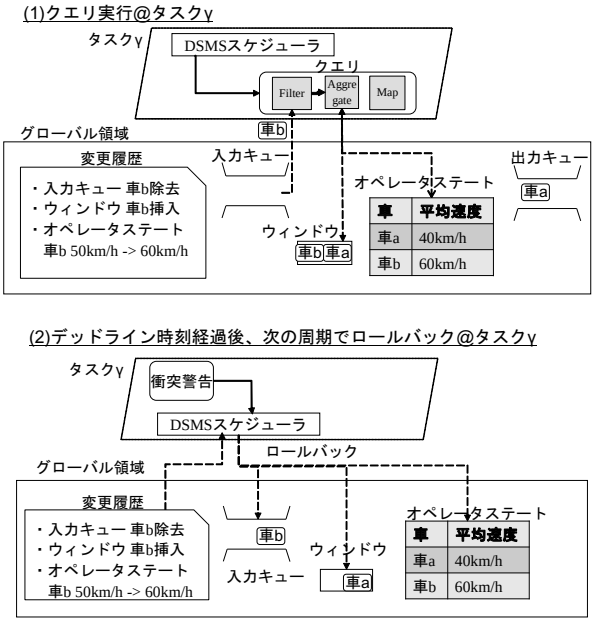

図 5　デッドライン時刻経過時のクエリ処理の中止

車間通信データなどの入力データを入力キューに格納する．例えば図 4(2) では，データ車 a，車 b を順に入力キューに格納する．

4.3.5　クエリの処理の実行

DSMS スケジューラからクエリの処理が呼び出されると，まずクエリの先頭のオペレータが入力キューから最新のデータを一つ読み込み，処理を実行する．そして先頭オペレータの処理終了後に，クエリの次のオペレータを処理し，そして続けてクエリの最後のオペレータまで処理をする．例えば図 4（3）ではデータ車 a をクエリで処理する場合を示す．まず先頭のオペレータ Filter が入力キューからデータ車 a を読み出し処理し，続いて次のオペレータ Aggregate で処理した後に，最後にオペレータ Map を処理する．そしてオペレータ Map は処理結果である出力データを出力キューに格納する．なお eDSMS におけるオペレータの処理方法の詳細は文献 [4] に記載する．

4.3.6　デッドライン時刻経過時のクエリ処理の中止

デッドラインまでにクエリの処理が完了しなかった場合に，そのタスクにおけるクエリの処理を中止する．その場合に，次の周期でクエリの処理を継続するために，クエリのコンテキストを引き継ぐ必要がある．

デッドライン時刻経過時においてクエリの処理を中止する処理を図 5 に示す．提案方式ではクエリの実行中に，クエリのコンテキストすなわちキュー，ウィンドウ，オペレータステートにあるデータの変更履歴を書き込む．例えば図 5（1）では，データ車 b に対してクエリを実行中，入力キューからデータ車 b を除去したことや，ウィンドウにデータ車 b を挿入したこと，Aggregate オペレータのオペレータステートの値を変更したことを変更履歴に記述する．そしてデータ車 b に対するクエリの処理を実行中にデッドラインの時刻が経過し，クエリの処理を中止する場合，図 5（2）に示すように次の周期でロールバックを行い，データ車 b を処理する前のコンテキストに戻す．これにより次の周期でコンテキストを引き継ぎ，クエリを処理可能とする．

4.4　クエリ処理効率化のためのタスク切替え時の動作

提案方式では図 1(c2) に示すように，処理が等価なクエリのコンテキストを共有することで，タスクが切り替わる際に（A）クエリの出力データを，切り替わり先のタスクのアプリケーションが利用する．また（B）クエリの処理を，切り替わり先のタスクのクエリが引き継ぐ．以下で (A)(B) の動作について説明する．

4.4.1　出力データの利用

タスクの切り替わり時に，他タスクのクエリの出力データを切り替え先のアプリケーションが利用する場合の動作を図 6 の (1)〜(3) に示す．クエリの処理の出力データはグローバル領域にある出力キューに格納されているため，タスク切り替え後，アプリケーションで出力キューにアクセスし，出力データを取得する．例えば図 6(1) ではタスク γ でデータ車 a が出力キューに格納されているため，図 6(2) でタスク γ からタスク α に切り替わった後に，て図 6(3) に示すようにタスク α に割り当てられた衝突警告アプリケーションが，出力キューからデータ車 a を取得する．

4.4.2　クエリの処理引継ぎ

タスク切り替わり時に，切り替わり先のタスクでクエリの処理を継続する場合の動作を図 6 に示す．4.3.6 節で述べたように，提案方式ではクエリの実行中にコンテキストの変更履歴を書き込む．そして変更履歴に基づき，切り替わり先のタスクでキュー，ウィンドウ，オペレータステートに保存されているコンテキストを，データを処理する前の状態に戻すことで，切り替わり先のタスクでクエリの処理を継続する．図 6（1）では，データ車 b に対してクエリを実行中，変更履歴を書き込み，図 6（2）に示すようにタスク γ からタスク α への切り替わり時に，図 6（4）に示すようにタスク α でロールバックを行い，データ車 b をクエリで処理する前のコンテキストに戻す．これにより図 6（5）に示すように，切り替わり先のタスクで再び，データ車 b をクエリの処理で実行可能とする．

また切り替え先のタスクでアプリケーションの実行終了後に，クエリの処理を実行中の切り替え元のタスクを呼び出す．その際にクエリのコンテキストが切り替え先のタスクで変更したにもかかわらず，切り替え元のタスクではクエリの処理を実行中であるため，そのままクエリの処理を行うとグローバル領域にあるコンテキストにアクセスすると処理が不正になる．そこで提案方式ではクエリ実行中にグローバル領域へアクセスする際にはタスクの切り替えを起こらないようにし，かつグローバル領域にアクセスする前にコンテキストが他のタスクのクエリにより更新されているか否かをチェックする．そしてコンテキストが更新されている場合には，クエリの処理が不正となるため，グローバル領域にアクセスする前にクエリの処理を中止し DSMS スケジューラに制御を戻す．例えば図 6（6）では再びタスク γ からタスク α に切り替わった際に，クエリの処理において Map オペレータがグローバル領域にアクセスする際に，タスク γ におけるクエリの処理によるコンテキストの更新を検出

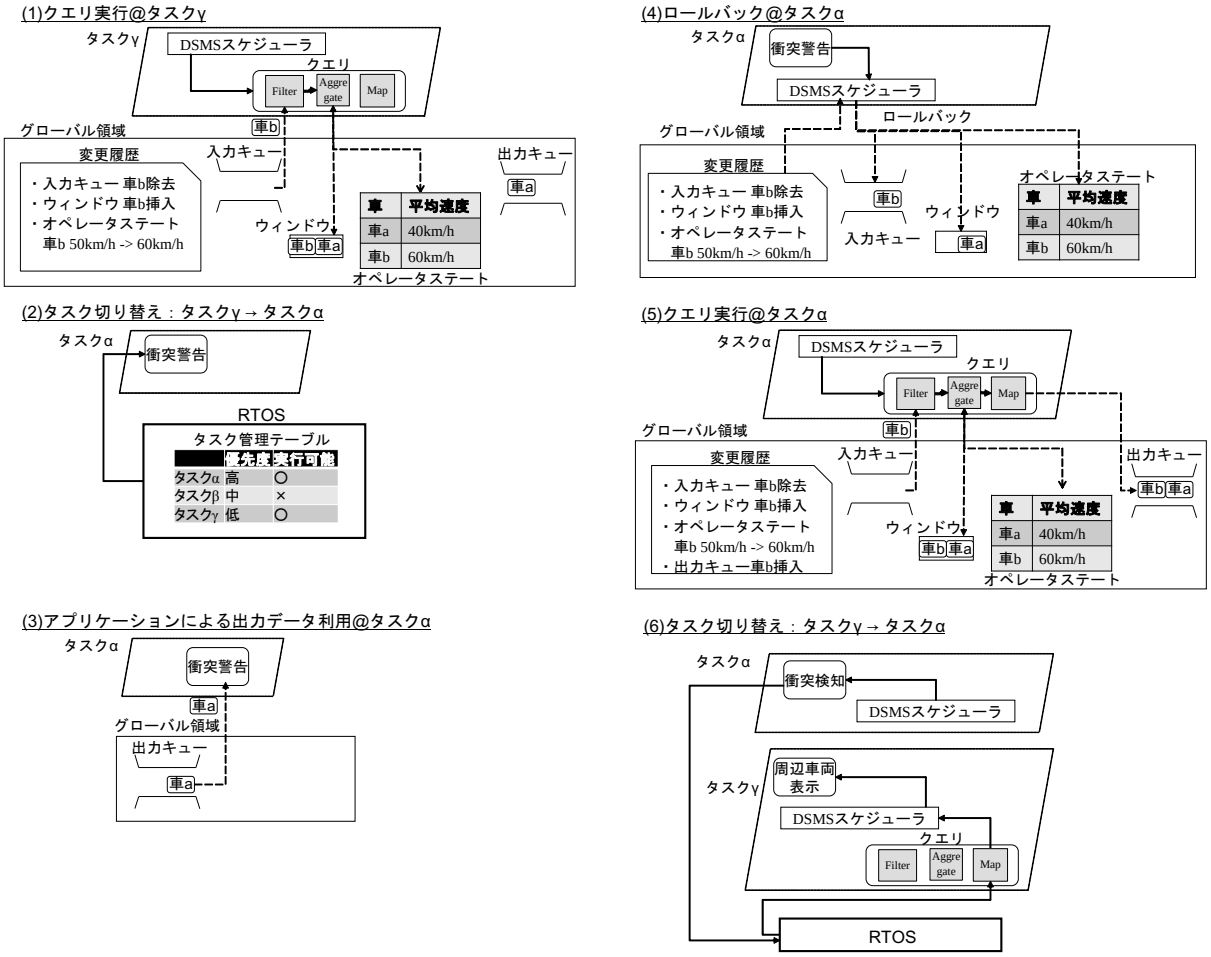

図 6　クエリ処理効率化のためのタスク切替え時の動作

するため，クエリの処理を中止し DSMS スケジューラに制御を戻す．

4.5　拡張方式：Non-preemptive section の導入

提案方式では 4.4.2 節で述べたように，タスク切り替え時にコンテキストを引き継ぐ際に，クエリの処理をロールバックし，切り替え先のタスクでその入力データに対してクエリの処理を再度，実行する必要がある．そのためタスク切り替え時に処理のオーバヘッドが発生する．

そこで本節では拡張方式として，その期間中はタスクの切り替えが起こらない Non-preemptive section(NPS) を導入し，クエリの処理を実行中，NPS とすることで，タスク切り替え後のロールバック及びクエリの処理の再実行を不要にする．拡張方式の動作を図 7 に示す．拡張方式では図 7（1）に示すように，タスク γ 実行中に，より優先度が高いタスク α が実行可能となった場合にも，クエリの処理を実行中，NPS となっているためタスク α への切り替えが発生せず，データ車 a をクエリで最後まで処理することができる．NPS は一つのデータをクエリの最後のオペレータまで処理すると終了し，タスクを切り替える．図 7（2）に示すようにデータ車 a の処理後にタスク γ からタスク α に切り替わり，タスク α ではロールバックやデータ車 a の再実行を行うことなく，図 7（3）に示すように続けてデータ車 b に対してクエリの処理を実行する．

拡張方式では NPS を RTOS の機能である優先度上限プロトコルにより実現する．優先度上限プロトコルでは，あらかじめ複数のタスクに対してリソースを登録することで，そのリソースを獲得したタスクは，リソースを登録したタスクの中の最も高い優先度のタスクと同じ優先度で実行することができる．拡張方式では同じクエリを割り当てたタスク α，β，γ に対してリソースを登録し NPS でリソースを獲得することにより，NPS で他のタスクに割り込まれることを回避する．

そのため拡張方式では NPS の期間中において優先度が高くなるため，その期間に優先度逆転が生じる可能性がある．例えばタスク γ は NPS の期間中ではタスク α と同じ優先度で実行されるため，NPS の期間である単一のデータをクエリの最初から最後まで処理する間，タスク γ よりも優先度が高いが，タスク α 以下の優先度のタスクの実行が妨げられる．

5.　評　　価

5.1　評価方法，環境

クエリコンテキスト共有方式をクエリ処理共有方式と比較し，優先度逆転時間，クエリの処理時間について評価を行った．

評価環境は，ZMP 社の RoboCar [9] を利用した．評価に用いたクエリは，図 8 に示す周辺車両情報を配信するクエリ（周辺車両情報配信クエリ）である．周辺車両情報配信クエリでは，

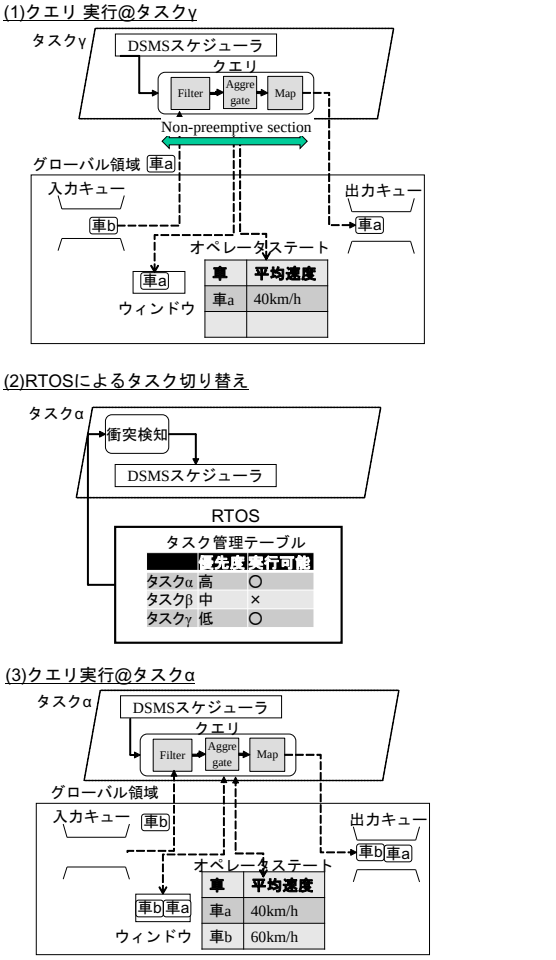

図 7　拡張方式：Non-preemptive section の導入

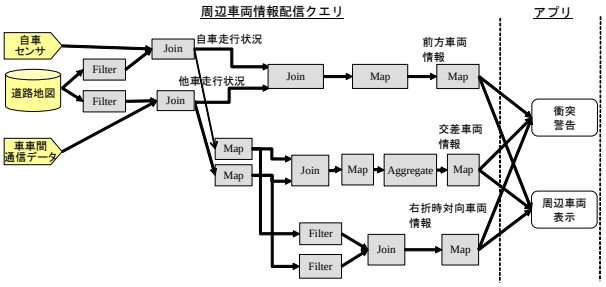

図 8　評価に用いたクエリ

自車センサ情報，車々間通信により受け取る他車センサ情報を入力源とし，道路地図情報も用いて前方車両及び，交差点における交差車両，右折時の対向車両の情報を算出する．そして算出した各種情報を衝突警告及び周辺車両表示アプリケーションに配信する．車々間通信データ，自車センサ情報はそれぞれ 50ms，20ms 周期で更新し，車々間通信データは最大で他車 90 台から受信する．また図 1 の例と同様に，衝突警告，周辺車両表示アプリケーションはいずれも周期実行を行い，動作周期はそれぞれ 100ms，50ms とする．優先度については，衝突警告アプリケーションの優先度は「高」，周辺車両表示アプリケーションの優先度は「低」とする．

5.2　クエリ実行方法

クエリ処理共有方式及び，クエリコンテキスト共有方式における実行方法について述べる．

- クエリ処理共有方式

クエリ処理共有方式では，図 1(b) に示すスケジューリング方法と同様に，衝突警告，周辺車両表示アプリケーションが利用するクエリの処理を，各アプリケーションとは別のタスクに割り当てる．そしてクエリを割り当てるタスクを，より優先度が高い衝突検知アプリケーションの優先度及び，動作周期が短い周辺車両表示アプリケーションの動作周期に合わせて，優先度「高」，動作周期 50ms で実行する．

- クエリコンテキスト共有方式

クエリコンテキスト共有方式ではクエリを，衝突警告及び周辺車両表示アプリケーションと同一タスクに割り当てる．そしてクエリ処理効率化のために，クエリの出力データを他のタスクのアプリケーションが利用する．またクエリのコンテキストを共有し，タスク切り替え時にクエリの処理を別のタスクのクエリが引き継ぐ．NPS を導入する拡張方式では，クエリ全体を一つの NPS の期間とし，一つのデータをクエリの処理で最初のオペレータから最後のオペレータまで実行する間，タスクの切り替えをしない．

表 1　評 価 結 果

項目	コンテキスト	NPS	処理共有	共有なし
優先度逆転時間	0 us	280 us	13,100 us	0 us
クエリ処理時間	13,380 us	13,100 us	13,100 us	26,200 us

5.3　優先度逆転時間

各方式の優先度逆転時間の評価結果を述べる．表 1 に，クエリコンテキスト共有方式（コンテキスト），クエリコンテキスト共有方式における NPS を導入した拡張方式（NPS），クエリ処理共有方式（処理共有），クエリに対して処理効率化のための共有を行わない方式（共有なし）の評価結果を示す．

クエリ処理共有方式でクエリの処理を実行する場合には，優先度が高い衝突警告アプリケーションと同一の優先度で実行する．従って図 1(b1) に示すように，衝突警告アプリケーションが実行されない周期では，優先度が低い周辺車両表示アプリケーション向けのクエリが高い優先度で実行されるため優先度逆転が生じる．同一周期で最大で他車 90 台分のデータがクエリにより処理され，その処理時間は最大で 13,100us となるため，優先度逆転時間の最大値は 13,100us となる．

一方，クエリコンテキスト共有方式ではクエリの処理はアプリケーションと同一の優先度で実行されるため，クエリの処理を実行中，優先度が逆転することはない．ただし 4.4.2 節で述べたように，グローバル領域の更新中はタスクの切り替えを禁止するため，優先度逆転が発生する可能性がある．上記に関する優先度逆転時間の評価については今後の課題とする．

また NPS を導入する方式では，周辺車両表示アプリケーション向けのクエリの処理を NPS で実行中，衝突警告アプリケーションと同様に優先度「高」で実行されるため，優先度逆転が生じる．NPS の期間は，一つの入力データを最初のオペレータから最後のオペレータまでクエリで処理する時間となるため，

優先度逆転時間は最大でも 280us に留まる．従ってクエリコンテキスト共有方式による優先度逆転時間の削減効果が示された．

5.4 処 理 時 間

各方式のクエリの処理時間の評価結果を述べる．図 1 に示すように，クエリ処理共有方式では，二つのアプリケーション向けのクエリの処理を共有することにより，共有化しない場合と比較しクエリの処理時間を半分に削減することができる．また NPS を導入するクエリコンテキスト共有方式でも同様にクエリの処理時間を半分に削減することができる．一方，NPS を導入しないクエリコンテキスト共有方式では，タスク切り替え時にクエリを処理中の場合に，ロールバックを行い切り替え先のタスクで，クエリ処理を再実行する．一つの入力データを最初のオペレータから最後のオペレータまでクエリで処理する時間は 280us であるため，クエリ処理共有方式と比較し，最大で 280us 処理時間が増加する．したがってクエリコンテキスト共有方式のクエリ処理共有方式と比較した場合のクエリの処理時間の増加は最大でも 2.13% に留まり，十分，小さいことが示された．

なお本評価におけるクエリの処理時間は，クエリ内のオペレータの処理時間であり，ロールバックや，変更履歴の書き込み，RTOS 内の処理で発生する処理オーバヘッドは考慮していない．上記の処理オーバヘッドを含めた処理時間の評価は今後の課題とする．

6. お わ り に

本稿では，車載システムのスケジューリングの元で，クエリの処理を効率化する，クエリコンテキスト共有方式を提案する．クエリコンテキスト共有方式では，アプリケーションに合わせて異なる優先度でクエリを実行しつつ，優先度の異なるタスクにおいて処理が等価なクエリのコンテキストを共有する．そしてクエリの実行を中断しタスクを切り替える際に，前のクエリの出力データをアプリケーションを利用し，また前のクエリの処理を引き継ぐ．クエリコンテキスト共有方式を評価した結果，クエリ処理共有方式と比較し優先度逆転時間を低減し，また処理時間の増加も 2.13% に留まり，クエリの処理効率化の効果を示した．今後の課題としては，処理時間等に関するより詳細な評価や，部分的に同一の処理内容を持つクエリのコンテキスト共有化などが挙げられる．

文　　献

[1] 山田真大，鎌田浩典，佐藤健哉，手嶋茂晴，高田広章: 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.42, No.2, 2010.

[2] 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, Dec, 2012.

[3] 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.

[4] 勝沼聡，本田晋也，佐藤健哉，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の静的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（TOD），vol.55, 2012.

[5] R. Motwani, J. Widom，A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma: Query Processing, Resource Management, and Approximation in a Data Stream Management System, Conference on Innovative Data Systems Research (CIDR), 2003.

[6] D. J. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik: Aurora: a new model and architecture for data stream management, The VLDB Journal, 2003.

[7] J. Chen, D. J. DeWitt, and J. F. Naughton: Design and Evaluation of Alternative Selection Placement Strategies in Optimizing Continuous Queries, ICDE 2002.

[8] 渡辺陽介, 北川博之: 連続的問合せに対する複数問合せ最適化手法, 電子情報通信学会論文誌，Vol. J87-D-I, No. 10, pp. 873-886, 2004.

[9] ZMP: RoboCarR 1/10 / ZMP RC-Z, http://www.zmp.co.jp/e-nuvo/jp/robocar-110.html.

[10] Borealis Distributed Stream Processing Engine, http://www.cs.brown.edu/research/borealis/public/.