

クラウド型自動運転を指向した 並列ストリーム型ダイナミックマップ

鈴木 有也^{1,a)} 佐々木 健吾¹ 佐藤 健哉² 高田 広章³

受付日 2017年3月30日, 採録日 2017年10月3日

概要: クラウドから車両群を集中制御するクラウド型自動運転において, 重要な技術要素の1つに, 複数の車両とその周辺環境から得られる様々なデータを管理し ITS アプリケーションに提供するダイナミックマップ (DM) があげられる. クラウド型自動運転においては, 扱う車両台数が増加したときの低遅延維持が DM に求められる. 本研究では, 並列実行により車両台数増加時に低遅延を維持可能な並列ストリーム型 DM を提案する. N 台の車両の位置と速度を把握し衝突危険時にブレーキを制御する ITS アプリケーション (N = 1,000) を構築して評価したところ, 1 コア使用時では, アプリケーションの性能要件を満たせないのに対して, 8 コア使用時では 3.25 倍の処理性能向上により低遅延を維持でき, 性能要件を満たしうることを確認した.

キーワード: クラウド, 自動運転, Dynamic Map (DM), ストリーム処理, ITS

Parallel Stream-based Dynamic Map toward Cloud-based Automated Driving

NAOYA SUZUKI^{1,a)} KENGO SASAKI¹ KENYA SATO² HIROAKI TAKADA³

Received: March 30, 2017, Accepted: October 3, 2017

Abstract: In cloud-based automated driving that realizes a centralized control system for vehicles from the cloud, one of the key technologies is DM (Dynamic Map) that manages various data from vehicles and their surroundings distributed widely and provides their data for ITS applications. To realize cloud-based automated driving, one of the requirements of DM is to keep low latency when the vehicles increase. We propose *parallel stream-based DM* that can keep low latency when the vehicles increase. We evaluate the system by a collision avoidance application that controls brakes of N vehicles by checking their positions and speeds (N = 1,000). The result shows that the system using one core cannot meet the performance requirement of the application, while using eight cores improves 3.25 times as fast as using one core and can meet the requirement. We confirm that the system can keep low latency and meet the requirement by increasing processors or cores.

Keywords: cloud, automated driving, dynamic map (DM), stream processing, ITS

¹ 株式会社豊田中央研究所
Toyota Cenral R&D Laboratories, Inc., Nagakute, Aichi
480-1192, Japan

² 同志社大学モビリティ研究センター
Mobility Research Center, Doshisha University, Kyotanabe,
Kyoto 610-0321, Japan

³ 名古屋大学未来社会創造機構
Institute of Innovation for Future Society, Nagoya University,
Nagoya, Aichi 464-8601, Japan

a) naoya@mosk.tytlabs.co.jp

1. はじめに

近年, 国内外において自動運転に関する研究開発が精力的に進められている. 内閣府は, 2014 年度より戦略的イノベーション創造プログラム (SIP) を創設し, 研究課題の1つに自動走行システムをあげている [1]. また, Google は, ステアリング, アクセルペダル, およびブレーキペダルがない自動運転車 [2] の公道試験走行を米国内で開始し

ている。

本研究では、自動運転の1つの形態として、クラウド型自動運転の実現を目指している。クラウド型自動運転とは、車載のカメラ、レーダ、LIDAR等を用いて、車両が自律的に周囲の環境を認識し目的地まで走行する自律型の自動運転[2]とは異なり、クラウド上のデータ管理機構に複数の車両の位置・速度等のデータを集約し、クラウド上のITSアプリケーション（以降、アプリケーションと略す）がそのデータ管理機構の情報を用いて車両群を集中制御する自動運転の形態であり、Connected vehicle cloud[3]、Passenger vehicles[4]等の発展形である。クラウド型自動運転によって、多数の車両の経路と速度を集中制御することで、たとえば、車両の流れを信号で制御する従来の交差点ではなく、車両を停止させず互いに他車両をすり抜けて走行するような交差点[5]を実現できる。また、このような交差点の実現を見越し、信号交差点と比較してどれほどの交通流への効果が見込めるかを数値化する理論的なフレームワークも研究されている[6]。

クラウド型自動運転を実現するうえで最も重要な技術要素の1つに、クラウド上のデータ管理機構があげられる。LDM (Local Dynamic Map) [7]は、近年、ITS分野において、安全運転支援・自動運転のためのプラットフォームとして注目されており、クラウド型自動運転のデータ管理機構に適用できる。LDMは、図1に示すように、主に自車周辺を対象として、交通の安全にかかわる静的および動的な要素（車両情報・天候状態・交通状態・詳細地図等）を格納するデータの集合体である。LDMは、4階層のデータ構造を具備し、Type 1の静的データ（道路等）、Type 2の準静的データ（道路標識等）、Type 3の準動的データ（渋滞等）、およびType 4の動的データ（車両等）を格納する。各階層は、適切な頻度によって更新される。特に、Type 4の動的データに関しては、低遅延の更新が求められる。アプリケーションは、LDMのデータを用いて、安全運転支援・自動運転のための情報提供と車両制御を行うことができる。本研究では、LDMのデータ保有範囲を都市レベルにまで広げ、これをダイナミックマップ (Dynamic Map,

以降、DMと略す)と呼ぶ。LDMとDMは、データ保有範囲が異なるのみで本質的な概念は同一であるため、以降、本稿ではDMに統一する。

クラウド型自動運転による多数の車両の集中制御を考えた場合、DMの重要な要件の1つとしてあげられるのが、車両台数が増加したときの低遅延維持である。ここでいう遅延とは、DMに車両等のデータが入力され、それを用いてアプリケーションが処理を行い、結果が出力されるまでの処理時間のことをいう。たとえば、複数の大規模な交差点で、前述の互いにすり抜けて走行する交差点の実現を考えるならば、数百台~数千台の制御処理を低遅延で行う必要が出てくる。低遅延を実現するにあたっては、リアルタイムにストリーム（非同期に到着する無限個のデータ列）を処理する技術であるストリーム処理を適用することが解決方法の1つとして報告されている[8],[9]。すなわち、道路等の静的データは、PostgreSQL、あるいはSQLiteのような蓄積型データベースで扱い、車両等の動的データは、ストリーム処理で扱うことにより、DMに対するデータの抽出・選択の手続き（以降、DMクエリと呼ぶ）を低遅延で処理することができ、蓄積型データベースのみを用いるDM（以降、蓄積型DMと略す）と比較して遅延を大幅に低減できる。本稿では、このような方式のDMをストリーム型DMと呼ぶ。

しかしながら、従来のストリーム型DM[9],[10]は、クラウドに配置するのではなく、車載ECU (Electronic Control Unit) に配置することを目的としており、車両等の処理すべき動的データの規模を、車載センサ・車車間通信で取得できる範囲であるたかだか数百台規模に設定している。それゆえに、単一の実行主体（プロセッサ・コア等）で逐次処理するように設計されており、それをクラウド向けに用いた場合には、入力される車両台数が増加したときに低遅延を維持しにくいという問題点が生じる。なぜなら、都市レベルの広範囲に存在する動的データがクラウド上のストリームDMに入力されたなら、地理的に分散し互いに独立して処理可能なDMクエリの処理に対しても逐次処理を行わざるをえないからである。

本研究では、DMをクラウドに配置することをねらい、DMクエリの一部を並列実行することで車両台数が増加しても低遅延を維持できる並列ストリーム型DMを提案する。以降、従来のストリーム型DMを単にストリーム型DMと呼び、新たに提案する並列ストリーム型DMと区別する。並列ストリーム型DMの最大の特徴は、アプリケーション開発者が記述したDMクエリから、並列処理に適するDMクエリを機械的に生成でき、かつそれが低遅延に実行できる点である。本稿では、LDMリアルタイムシミュレータ[11]を拡張したDMリアルタイムシミュレータを用いて、並列ストリーム型DMとN:N衝突回避アプリケーション（N台の車両の位置と速度を把握し衝突危険時にプ

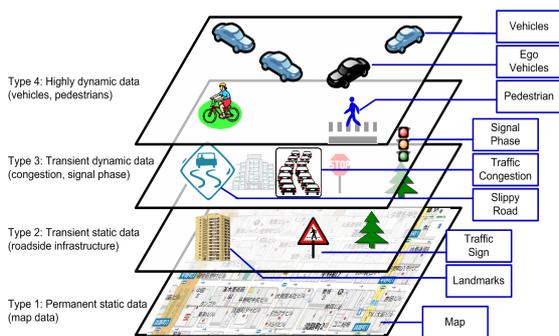


図1 LDM (Local Dynamic Map)

Fig. 1 LDM (Local Dynamic Map).

レーキを制御するアプリケーション) を実装し、並列ストリーム型 DM の有効性を評価する。

本稿の構成は、以下のとおりである。2 章でストリーム型 DM について概略を説明し、3 章で本研究が提案する並列ストリーム型 DM について述べる。4 章で評価を行い、5 章で関連研究を示す。6 章でまとめと今後の展開を述べる。

2. ストリーム型 DM

ストリーム型 DM は、図 2 に示すように、データストリーム管理システム (DSMS : Data Stream Management System) によるストリーム処理を基本とし、DM の動的データを蓄積型データベースに格納せずにストリームとして処理するモデルである。DSMS を用いると、車両データ等の動的データを、データフローで表現された DM クエリにストリームとして入力することで、アプリケーションに必要なデータを低遅延で生成できる。道路等の静的データと車両データ等の動的データで演算を行うことも可能である。その際には、道路等の静的データは、蓄積型データベースに格納されているため、ストリームに変換して DSMS に入力することによって動的データと演算を行う [8]。これにより、たとえば、マップマッチング (地図上のどの道路を車両が走行しているかの演算) を低遅延で行うことができ、その結果から、ある車両が次に到達する交差点を求めてアプリケーションにわたすといったことが可能になる。ストリーム型 DM を用いた場合、マップマッチングの平均遅延時間を、蓄積型 DM と比較して、車両台数 50 台のときに約 1/17 に低減できることが報告されている [8]。よって、ストリーム型 DM は、低遅延が要求されるアプリケーションを DM 上で動作させる有効な手段といえる。しかしながら、前述したように、従来のストリーム型 DM は、車載 ECU への搭載を目的とし、自転車とかかわりのあるたかだか数百台規模の車両データの逐次処理を前提としているため、そのままクラウド向けに用いた場合、入力する車両台数が増加すると低遅延を維持しにくいという問題点がある。

たとえば、多数の車両を集中制御するアプリケーションの一例として、4 章で取り上げるような、半径数 [km] に

分布する N 台の車両の位置と速度を把握し衝突危険時にブレーキを制御する衝突回避アプリケーションを考える。人間と同等の反応時間内にブレーキ制御を開始するためには、190 [ms] 以内の応答時間 [12] が要件としてあげられる。一方、文献 [11] において、車両 1 台とその周辺の車両 50 台の衝突予測をストリーム型 DM で実行した場合、平均 5.98 [ms] の処理遅延がかかることが報告されており、車両 1 台あたりの衝突予測の処理遅延が 0.12 [ms] と算出できる。このストリーム DM を用いると、道路上を 1,000 台が一列に直進している各車両に対して衝突予測を行う場合は、1,000 回の衝突予測の処理遅延を 120 [ms] と見積もることができ、応答時間の要件を満たすことができる。しかしながら、上記のような道路が、地理的に別の場所にもう 1 本存在する場合、2 倍の車両に対して逐次処理で衝突予測をする必要があり、処理遅延の見積もりが 240 [ms] となって応答時間の要件を満たすことが困難になる。このように、従来のストリーム DM では、地理的に分散した、並列実行可能な複数の独立した処理に対しても逐次処理を行わざるをえないため、入力する車両台数が増加するにつれて、応答時間が増加し、低遅延を維持しにくいという問題点がある。

3. 並列ストリーム型 DM

ストリーム型 DM の問題点を解決する手段の 1 つとして並列化があげられる。たとえば、2 章で述べた衝突回避アプリケーションが、地理的に別の地域にある複数の各道路において車両の衝突予測を行う場合、ある道路の衝突予測処理に必要な車両データが、他の道路の衝突予測処理に非依存であるなら、道路ごとに並列に衝突予測処理を行うことによって、大幅な処理遅延の低減が期待できる。このような並列処理に適した DM クエリを機械的に生成でき、並列実行により低遅延処理を実現するのが、本研究が提案する並列ストリーム型 DM である。本章では、並列ストリーム型 DM の特徴である、並列処理に適した DM クエリの機械的な生成手法と低遅延処理に関して説明する。はじめに、並列ストリーム型 DM の前提とコンセプトについて述べた後に、本手法の全体像を述べる。

3.1 前提

一般的に、DM の配置場所のバリエーションとしては、ITS 集中管理システムをはじめとし、携帯端末、車両、および路側器への配置が考えられている [13]。本研究では、前提として、並列ストリーム型 DM の配置場所を、ITS 集中管理システムの類であるエッジクラウドとする。近年では、クラウドとの通信遅延を低減させるために、欧州の標準化機構である ETSI が、モバイルエッジコンピューティングの規格化を進めており、次世代移動通信システム 5G におけるキーテクノロジーとして位置づけている [14]。モ

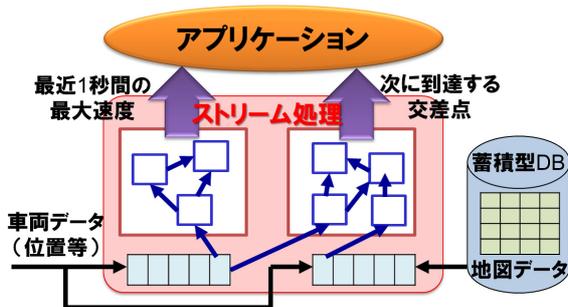


図 2 ストリーム型 DM
Fig. 2 Stream-based DM.

パイルエッジコンピューティングとは、IT サーバをネットワークエッジ（例、通信端末の基地局等）に配置し、エッジクラウドという小規模クラウドをそこに構築することで、IP バックボーンの先にある一般的なクラウドと通信する場合と比較して、通信端末との通信時間を大幅に低減する技術である。本研究では、並列ストリーム型 DM とそのアプリケーションを、エッジクラウドのような、通信端末との通信遅延が低いクラウドに配置することを想定する。

並列ストリーム型 DM を車両に配置することも選択肢としてありうる。すなわち、車載センサ（カメラ・LIDAR・レーダ等）、あるいは車車間通信を用いることで、周囲の車両・人等を認識し、個々の車両に DM を構築することは可能と考える。しかしながら、車載センサ、あるいは車車間通信を用いると、自車両周辺（たとえば、車車間通信であれば自車両周辺の約 300 [m]）の環境認識となり、構築できる DM の地理的範囲が限定されてしまうという問題が生じる。一方で、クラウド（エッジクラウド）に DM を構築した場合、車載センサ、あるいは車車間通信よりも広範囲（セル半径が大きいもので数 [km]）の情報によって DM を構築できるため、より広範囲を対象として DM クエリを発行できるという利点が生まれる。将来的に、たとえば、半径数 [km] の範囲に存在する複数の交差点の混雑度を用いて、各車両の目的地に最も早く到着できる最適な経路と速度で車両の走行を制御するといった用途等に用いるのであれば、クラウドに並列ストリーム型 DM を配置するのが妥当と考える。

3.2 コンセプト

並列ストリーム型 DM は、ストリーム型 DM に対して以下の 2 点の拡張を施している。第 1 の拡張は、データフローの並列実行を中心とした既存の並列ストリーム処理の仕組み [15], [16], [17] を追加することによって低遅延に DM クエリを実行できるよう改良したことである。第 2 の拡張は、DM クエリに特有の地理的なデータ非依存性を活用し、アプリケーション開発者が記述した DM クエリから、並列処理に適する DM クエリを機械的に生成するクエリ展開機能を追加したことである。本研究の最大の特徴は、従来の並列ストリーム処理基盤が有しない、後者のクエリ展開機能にある。

一般的に並列化を行うためには、計算すべき対象から並列性を抽出する必要がある。DM クエリから並列性を抽出するにあたって着目すべき点は、DM クエリが、ある地理的範囲（交差点、合流点、区域等）に対して発行されることが多く、地理的なデータ非依存性をともなうことが多いために、並列性を容易に抽出できる点である。地理的なデータ非依存性とは、ある DM クエリを実行する際に必要とする入力データが地理的範囲において互いに非依存であることをいう。入力データが互いに非依存ということは、それ

を処理する DM クエリを、地理的範囲ごとに複数に展開して、それぞれの地理的範囲において独立して実行できることを意味する。よって、そのように展開した DM クエリそれぞれに実行主体を割り当てることで処理速度の向上が期待でき、車両等の処理台数が増加したとしても実行主体を増加させることで低遅延の維持が可能となる。

たとえば、DM から現在の車両の挙動を抽出し、衝突の危険性があれば警告を車両に送信するアプリケーションを考える。例として、交差点で停止している前方車両への追突警告をあげるなら、DM クエリは以下ようになる。

- (1) 交差点の半径 r [m] 以内の停止車両を抽出
- (2) (1) の車両の後方を走行する車両を抽出
- (3) (2) の車両のうち前車との衝突余裕時間が t [s] 以内の車両を抽出
- (4) 警告を (3) の車両に対して送信

DM 上に登録されているすべての交差点において (1)~(3) を実行することで、交差点で停止している前方車両への追突危険のあるすべての車両に対して警告が送信される。もし、DM に複数の交差点が存在すれば、(1)~(3) は各交差点において独立した DM クエリとして実行できる。なぜなら、(1)~(3) の DM クエリを各交差点において発行する場合、入力である車両データが交差点において互いに独立しており依存関係がないからである。これはすなわち、「交差点」が N 個存在すれば、 N 個の独立した DM クエリに展開できることを意味する。このように展開した N 個の各 DM クエリに N 個以下の実行主体を割り当てることで並列実行による性能向上が得られ、車両台数が増加したとしても低遅延を維持できる。本研究では、上記のような並列処理向けの DM クエリの展開を機械的に行えるようにする。

3.3 クエリ展開の全体像

前述のコンセプトを具現化したクエリ展開の全体像を図 3 に示す。アプリケーション開発者が記述する DM クエリは、ストリームの処理ロジックの実体であるオペレータと発行ポイントの接続で構成される。発行ポイントとは、その後続のオペレータ群（以降、サブクエリと呼ぶ）を実行すべき地理的範囲を示す。発行ポイントの例としては、交差点、合流点、道路列等があげられる。発行ポイントの後続のオペレータ群が、発行ポイントにおいて展開される。展開するオペレータは単独であっても複数であってもよい。図 3 では、発行ポイント 1 と 2 の直後にそれぞれ A と B という単独のオペレータが配置されているので、発行ポイント 1 と 2 で、それぞれ A と B を実行する。

地図には、発行ポイントをあらかじめ明示しておく。図 3 では、発行ポイント 1 を交差点とし、3 個の発行ポイント (1-1~1-3) を定義している。発行ポイント 2 は、道路列（地理的な順に並べた道路 ID の列）で、同様に 3 個の発行

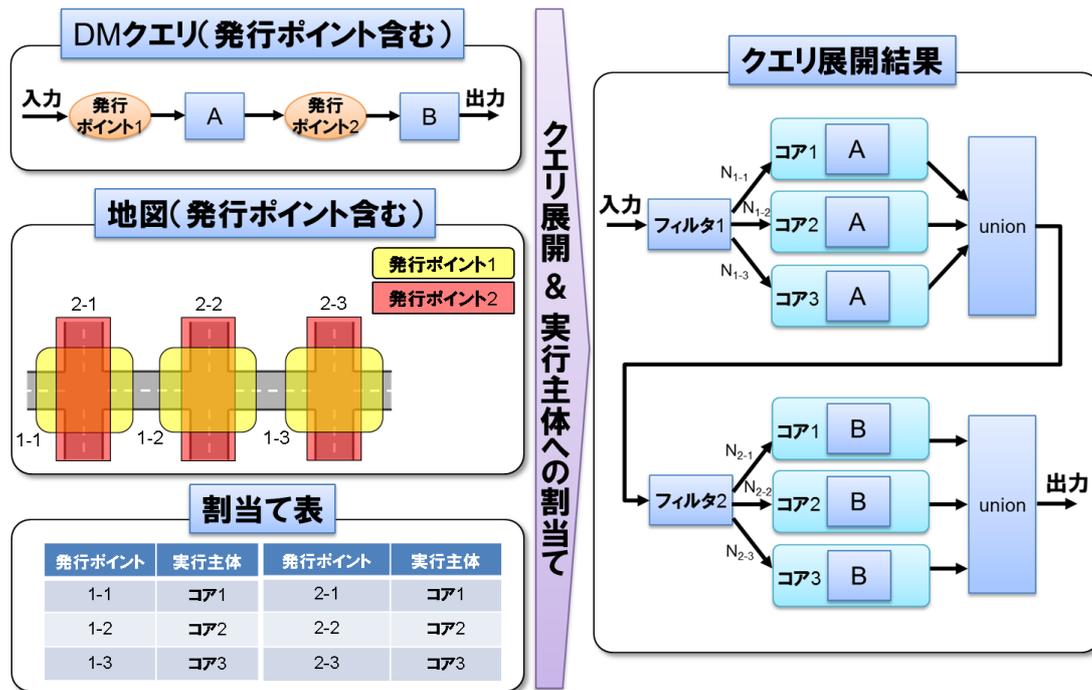


図 3 クエリ展開の全体像
Fig. 3 Overall perspective of the query expansion.

ポイント (2-1~2-3) を定義している。

割当て表は、発行ポイントと実行主体を対応づける。すなわち、各発行ポイントで展開したサブクエリをどの実行主体で実行するかの対応づけを行う。図 3 では、発行ポイント 1-1~1-3 に対して、それぞれコア 1~3 を対応づけている。発行ポイント 2-1~2-3 についても同様である。

DM クエリ、地図上の発行ポイント、ならびに割当て表が定義されると、DM クエリを並列処理向けに展開できる。その結果を図 3 の右に示す。フィルタ 1 において、入力データが 1-1, 1-2, または 1-3 のどこに属するかをチェックし、それぞれを担当するオペレータ A に送信する (N_{1-1} , N_{1-2} , N_{1-3})。オペレータ A は、それぞれコア 1~3 で実行され、送信されてきたデータを処理してフィルタ 2 に集約する。フィルタ 2 でもフィルタ 1 と同様に、入力データが 2-1, 2-2, または 2-3 のどこに属するかをチェックし、それぞれを担当するオペレータ B に送信する (N_{2-1} , N_{2-2} , N_{2-3})。最後に結果を集約して出力する。発行ポイントが多くなるほど、展開されるサブクエリの個数が増加するため、実行主体の個数増加に対応できる。

もし、アプリケーション開発者が、複数の DM クエリを並列ストリーム型 DM に対して与えるのであれば、それらは並列に実行可能であることが自明である。一方、並列ストリーム型 DM は、図 3 に示すように、アプリケーション開発者が与える DM クエリが単一であってもかまわない。そのときに、並列ストリーム型 DM は、DM クエリに含まれる発行ポイントによって機械的に DM クエリを展開し、効率的に並列実行できるように変換する。無論、アプ

リケーション開発者が、上記の展開後の DM クエリを自力で書き下すことは可能である。しかしながら、発行ポイントが何千、何万とあった場合に、もしくは発行ポイントが頻繁に変更される場合に、それは現実的とはいえない。並列ストリーム型 DM は、アプリケーション開発者の負担軽減と低遅延処理を同時にねらう。

並列ストリーム型 DM の制限は、実行主体の個数以上の発行ポイントを地図に包含させなければ、並列処理による処理遅延の低減効果が減少することである。たとえば、交差点に対して発行する DM クエリがあるとして、実行主体が N 個ある場合、N 個以上の交差点に対して当該クエリを発行しなければ N 並列で実行できず、処理遅延の低減効果が減少する。よって、N 並列で実行するためには、N 個以上の発行ポイントを地図に包含させることが条件となる。

4. 評価

本章では、並列ストリーム型 DM を評価する。はじめに、評価アプリケーションについて詳細を説明した後、評価システムの構成、評価項目、評価結果、評価のまとめの順で述べる。

4.1 評価アプリケーション

4.1.1 概要

本評価では、地理的なデータ非依存性を用いた並列化の有効性を示すことに焦点を当てるために、車両台数の増加に対して処理量の増加が顕著となる N:N 衝突回避アプリケーションを用いる。本アプリケーションの概要を図 4 に

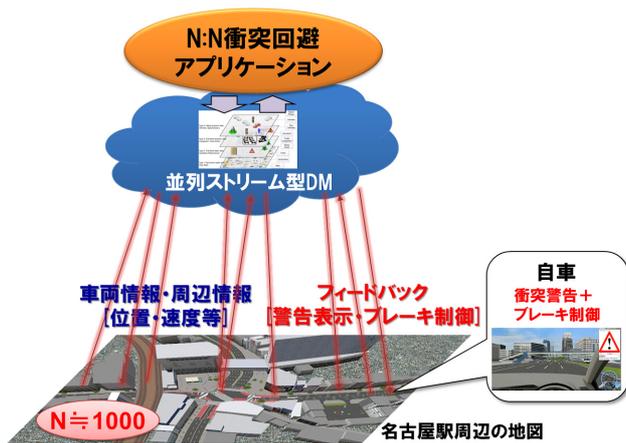


図 4 N : N 衝突回避アプリケーション
 Fig. 4 N : N collision avoidance application.

示す。本アプリケーションは、クラウドに配置され、N 台の車両から 100 [ms] 周期で車両データを受信し、各車両の位置関係を把握して、各車両とそれ以外の車両との衝突余裕時間（以降、TTC : Time To Collision と略す）が、しきい値 t_1 以下になれば、衝突の可能性があるとして当該車両に警告を表示し、さらに TTC が、しきい値 t_2 以下になれば、当該車両にフルブレーキをかけて衝突の回避を試みるアプリケーションである。前述したように、並列ストリーム型 DM と本アプリケーションは、N 台の車両情報を収集して並列で処理するために、エッジクラウドのような、通信端末との通信遅延が低いクラウドに配置することを想定する。地図には、例として名古屋駅周辺を使用する。

エッジクラウド上で衝突回避を行うメリットは、従来の車載センサ・車車間通信では認識することが困難な車両に対して衝突回避が可能になる点である。たとえば、交差点において高層ビル等の大きな建造物が存在する場合、あるいは大雨・濃霧等の天候不良が発生した場合、車載センサ（カメラ、レーダ等）では、自車から認識できない交差車両・前方車両が発生する可能性が高くなる。また、車車間通信を用いた場合では、車両密度が高くなればなるほど、CSMA/CA 方式の特性と隠れ端末の影響によりパケット到達率が低下し、衝突防止等のサービスを提供可能なパケット到達率を満たすことが困難になるという報告がある [18]。エッジクラウドによる衝突回避は、これらの状況が発生したときにも有効に機能しうる点が最大のメリットである。その他のメリットとしては、衝突回避のために必要となる、レーンまで含めた最新の詳細地図を、すべての車両に配信する必要がなく、エッジクラウドで保持すれば十分な点があげられる。各車両に対して詳細地図を配信するための通信を行わずに済み、かつ地図更新のタイムラグによって車両が保持する詳細地図と実際が異なることが原因で衝突回避の精度が低下することを防止できる。

本アプリケーションは、以下のように動作する。N 台の

車両から車両データが約 100 [ms] 周期でセンシングされ、並列ストリーム型 DM を介して本アプリケーションへとわたされる。本アプリケーションでは、並列ストリーム型 DM が提供する DM オペレータを用いてマップマッチング等の処理を行った後に、N 台それぞれの TTC を計算し、 t_1 未満であれば警告表示を、 t_2 未満であればフルブレーキ制御をそれぞれの車両にフィードバックする。本評価では、例として $t_1 = 4.5$ [s]、 $t_2 = 2.5$ [s] に設定する。

並列ストリーム型 DM が、無線通信とエッジクラウドの使用を前提とする以上、厳密に処理時間のデッドラインを守ることは困難と考える。よって、この点に関しては、アプリケーションにおいて、デッドライン制約が満たされない場合に例外処理、あるいは縮退処理を実施し、システム全体としてクリティカルな状態にならないような設計を施す、もしくは、処理時間の変動を許容するアプリケーションを実行するといった対応が必要となる。本評価では、N : N 衝突回避アプリケーションを、処理時間の変動を許容するアプリケーションとして後者に位置づける。すなわち、現在の自動ブレーキシステムと同様で、どんな状況でも確実に衝突を回避するというのではなく、あくまでも運転者の衝突回避行動を支援するアプリケーションという位置づけで評価を行う。

本アプリケーションの性能要件は、以下のように設定する。人間の単純反応時間（ものを見て動作反応するまでの時間）は、視覚の場合、190 [ms] とされている [12]。また、次世代移動通信 5G の基地局にエッジクラウドを構築して並列ストリーム型 DM と本アプリケーションを配置することを想定し、本稿では、車両からクラウドまでの片道の通信時間を 14 [ms] とする。この値は、5G における車両からクラウドまでの片道の無線伝送遅延である 1 [ms] [19] と、基地局におけるユーザデータを扱うプロトコルスタック (U-plane) の処理遅延である 13 [ms] [20] (LTE の場合を参考) の合計である。これらより、本アプリケーションの性能要件を、人間と同等以上となる $190 - 14 \times 2 = 162$ [ms] 以内の遅延時間と定める。また、電波状況によって、あるいは基地局への接続端末数によって、無線伝送遅延が変化することを考慮し、評価結果において処理遅延が 162 [ms] 未満である場合、162 [ms] とその値との差が無線通信のばらつきを許容できる範囲とする。

4.1.2 内部構成

本アプリケーションの内部構成を図 5 に示す。本アプリケーションは、衝突警告部と衝突回避部から構成される。衝突警告部は、N 台分の車両データが入力され、マップマッチングオペレータでマップマッチングを行った後に、TTCLessThan オペレータで前後車両の TTC がしきい値に達したか否かを判定し、警告する必要がある場合は当該車両に警告をフィードバックする。また、求めた TTC に応じてイベントを発行する。TTC が t_1 以下、かつ t_2 より上で

●発行ポイントを含んだDMクエリ

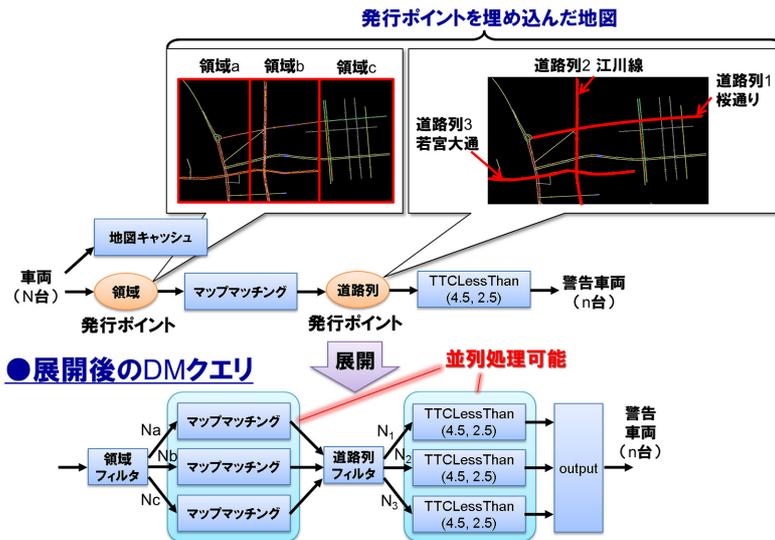


図 6 発行ポイントによるクエリ展開

Fig. 6 The expansion of query by query running points.

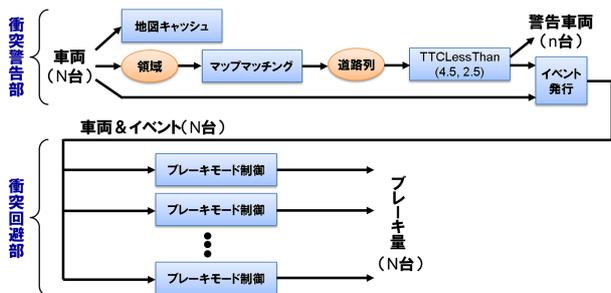


図 5 N : N 衝突回避アプリケーションの内部構成

Fig. 5 The inner mechanism of the N : N collision avoidance application.

あればブレーキ補助イベントを発行し、後段の衝突回避部に対して、ブレーキ補助モード（ブレーキ量 1.5 倍）に移行するように指示する。 t_2 以下であればフルブレーキイベントを発行し、緊急ブレーキモード（フルブレーキ）に移行するように指示する。図 5 において、TTCLessThan オペレータの引数には、 t_1 と t_2 を指定し、本評価では、前述したようにそれぞれ 4.5 [s] と 2.5 [s] に設定する。後段の衝突回避部において、各ブレーキモードにおけるブレーキ量を N 台分算出し、車両がそれらを受信することでブレーキ制御が行われる。展開されて並列に実行される部分は、発行ポイントを含む衝突警告部であり、具体的には、マップマッチングオペレータと TTCLessThan オペレータである。ブレーキモード制御クエリに関しては、すべてを単一の実行主体（本評価ではコア）で逐次実行する。

4.1.3 発行ポイントの展開

本アプリケーションでは、図 5 に示したように、「領域」と「道路列」という発行ポイントが含まれ、それらの直後にそれぞれマップマッチングオペレータと TTCLessThan

オペレータが配置されている。これは、領域それぞれに対してマップマッチングオペレータを実行し、道路列それぞれに対して TTCLessThan オペレータを実行することを意味する。それら発行ポイントは、図 6 の例に示すように地図において具体的に明示する。この例では、領域 a~c、および道路列 1~3 である。

発行ポイントを含んだ DM クエリと、発行ポイントを埋め込んだ地図をセットにすることで、図 6 に示すように DM クエリを展開できる。たとえば、発行ポイント「領域」で展開する場合は、地図に埋め込まれている「領域」それぞれに対して DM クエリを発行することになるため、領域フィルタオペレータで車両 N 台を各領域に振り分け、その後それぞれでマップマッチングを実行するように DM クエリを展開できる。道路列の場合も同様である。そのようにすると、マップマッチングオペレータの部分と TTCLessThan の部分が独立した並行なデータフローとなるため、それぞれをコアに割り当てることによって並列処理が可能となる。本評価では、名古屋駅周辺地図に 8 個の領域と 8 本の道路列を埋め込み、発行ポイントとすることにより、8 並列での実行に対応できるようにする。

4.2 評価システムの構成

評価システムのスペックを表 1 に示す。並列ストリーム型 DM は、プライベートクラウド上に構築し、DM ビューアと運転シミュレータ UC-win/Road [21] にギガビット・イーサネットで接続する。並列ストリーム型 DM は、運転シミュレータから N 台分の車両データ（位置・速度等で 54 [byte]/台）を約 100 [ms] 周期で受信し、N : N 衝突回避アプリケーションを動作させ、N 台分の衝突警告・回避の結果を得る（以降、これを 1 タイムステップと呼ぶ）。

表 1 評価システムのスペック

Table 1 The specification of the evaluation system.

部名	項目	詳細
DM ビューア	CPU	Intel Core i5-2540M 2.6 GHz
	メモリ	8 GB
	OS	Windows7 (64 bit)
並列ストリーム型 DM	CPU	Intel Xeon E5-2620 v2 2.1 GHz (6core × 2CPU)
	メモリ	32 GB
	OS	Windows7 (64 bit)
運転シミュレータ	CPU	Intel Xeon E3-1240 v3 3.4 GHz
	メモリ	16 GB
	OS	Windows7 (64 bit)

N : N 衝突回避アプリケーションは、周期 100 [ms] ごとに、すべての車両データが収集されたタイミングで実行される。よって、車両の衝突予測処理 (TTCLessThan オペレータ) において、ある車両の TTC を求める際には、その道路列に存在する他の車両データがすべて収集されていることを前提とする。運転シミュレータは、N : N 衝突回避アプリケーションより、N 台分の衝突警告・回避の結果を受信し、警告表示とブレーキ制御を行う。DM ビューアにも、N 台分の車両データと衝突警告・回避の結果を送信し、各車両の位置・警告状態・ブレーキ制御状態を視覚化する。送受信プロトコルには UDP を用いる。

一般的に無線通信においては、電波干渉、輻輳等により、UDP 通信でパケットロスが発生しやすい。一方、本評価においては、UDP 通信におけるパケットロスがきわめて少ない状況を前提とする。その理由としては、3.1 節に述べたように、車両とエッジクラウドの無線通信に、次世代移动通信システム 5G の使用を想定していることがあげられる。5G は、無線通信容量を従来の 1,000 倍にまで高めるために、周波数の利用効率を向上させる NOMA、高密度に配置された高周波数帯のセルを運用するファントムセル等の技術を組み合わせて実現されるため [19], [22], パケットロスの原因となる電波干渉、輻輳等が大幅に低減されることが期待できる。また、5G PPP では、自動車業界から 5G に対する技術要件として、自動運転・安全運転支援を実現するために、パケット到達率を 99.999[%] に設定している [23]。よって、本稿では、パケットロスの影響を除いて評価を行う。

4.3 評価項目

評価項目は以下の 4 点である。第 1 の評価項目は、N : N 衝突回避アプリケーションのリアルタイム動作である。運転シミュレータ上で実際に運転をした場合に、警告表示とブレーキ制御がねらいどおりに実現されるかを評価する。

第 2 の評価項目は遅延時間である。ここでいう遅延時間とは、N 台分の車両データが並列ストリーム型 DM に入力

されてから、N : N 衝突回避アプリケーションがブレーキ量を出力するまでの時間を指す。車両台数 N を最大 1,000 台まで変化させ、性能要件を満たしうるかを評価する。また、単純に、車両データの ID 順に、その車両データの処理をコアに割り当てて並列化を行った場合と比較する (以降、この比較対象を車両 ID 順と呼ぶ)。車両 ID 順での並列化とは、地理的なデータ非依存性を考慮して各発行ポイント内の車両データを各コアに割り当てる本手法に対比して、道路列へのマップマッチング情報等に代表される地理的な情報を考慮せずに車両データを各コアに割り当てる場合を示す。すなわち、車両の位置にかかわらず、車両 ID 順でコアに車両データを割り振り、それぞれのコアにおいて各車両の衝突回避処理を行う。その際に、衝突回避に必要となる近隣の車両データが自コアにすべてそろっているとは限らないため、コア間のデータ共有を行いながら、近隣の車両データをグルーピングした後に、衝突回避処理を並列実行する。

第 3 の評価項目は台数効果である。台数効果とは、並列処理で用いられる指標の 1 つで、n 個の実行主体 (プロセッサ・コア等) を用いた場合、1 個の実行主体のみを用いた場合に比べて何倍高速化されるかを表す。並列ストリーム型 DM と本アプリケーションにおいて、マルチコア実行で使用するコア数を増加させた場合に、遅延時間をどの程度短縮できるかを評価する。

第 4 の評価項目は記述量である。本評価では、アプリケーション開発者の負担の指標として記述量に着目し、記述量の増減がアプリケーション開発者の負担増減と定義する。本手法では、アプリケーション開発者が、発行ポイントとオペレータからなる DM クエリを記述することで、並列実行に適した DM クエリが機械的に生成される。本手法を用いずに、並列処理に適した DM クエリを直接書き下す場合と比較することで、記述量をどの程度削減できるかを評価する。

4.4 評価結果

4.4.1 リアルタイム動作

名古屋駅周辺の地図を用いて、車両台数 N = 1,000 台を走行させたときの通常走行、衝突警告、および衝突回避の様子を以下に示す。

図 7、図 8、および図 9 の左側は、1,000 台のうちの 1 台 (以下、自車と呼ぶ) を UC-win/Road で運転した様子 (以下、UC-win/Road 図と略す) を表し、右側は DM ビューアでの全体地図の俯瞰図 (以下、DM ビューア図と略す) を表す。DM ビューア図の矢印の先は自車の位置を示す。自車は、名古屋駅前の桜通り (図 6 の道路列 1) を東から西に向かって走行する。

図 7 は、通常走行時の様子である。UC-win/Road 図には右上にマークが表示されている。これはクラウドの N : N



図 7 通常走行

Fig. 7 Normal driving.



図 8 衝突警告

Fig. 8 Collision warning.



図 9 衝突回避

Fig. 9 Collision avoidance.

衝突回避アプリケーションから送信されている自転車へのシグナルの状態を表し、通常走行状態・衝突警告状態・衝突回避状態の3種類を表示する。緑色のマークが表示されて

いる場合は、通常走行状態であり、安全に走行できることを示す。

図 8 は、衝突警告時の様子である。時速約 80 [km/h] で

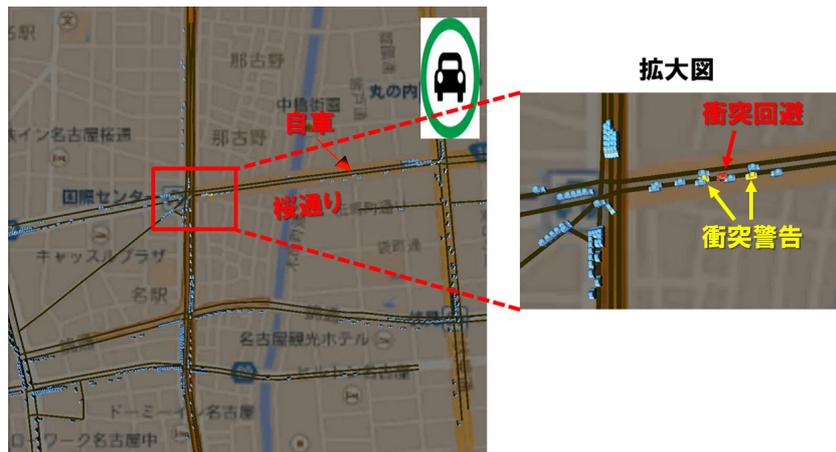


図 10 DM ビューア
Fig. 10 DM viewer.

前方の停車車両に接近する場面を UC-win/Road でシミュレートしている。衝突警告状態の場合は、図 8 のようにエクスクラメーションマークに変化する。すなわち、クラウドの N : N 衝突回避アプリケーションが、前車に対する自車の TTC を計算し、4.5 [s] 未満で危険と判断して警告シグナルを自車にフィードバックしている状態である。

図 9 は、図 8 の続きで、さらに前車に接近して、クラウドからブレーキが制御されている様子を示す。前車との TTC が 2.5 [s] 未満になると、クラウドの N : N 衝突回避アプリケーションがそれを検知し、フルブレーキ制御を行って、衝突を回避する。その間、運転者がアクセルを踏んでいたとしても、クラウドからの制御が優先される。結果として車両は停止し、前車への衝突を防ぐことができる。

図 10 は、ある時点での自車の前方交差点を DM ビューアで拡大している。DM ビューアは、地図上に車両がどのように分布して走行しているか、およびクラウドから各車両にどのようなシグナルが発信されているかを可視化する。青で表示されている車両は通常走行状態に、黄は衝突警告状態に、赤は衝突回避状態にあることを示す。このように、DM ビューアで車両の位置と制御状態を可視化することで、自車だけでなく他車もリアルタイムに制御されていることが分かる。

4.4.2 遅延時間

遅延時間を図 11 に示す。縦軸が遅延時間を表し、横軸が使用コア数を表す。車両台数 N を 400 台、800 台、1,000 台と変化させ、使用コア数に対する遅延時間の変化をみる。遅延時間は、300 タイムステップの平均を示す。実線が本手法の遅延時間を、破線が車両 ID 順で並列化した場合を示す。

本手法と車両 ID 順を比較すると、車両台数と使用コア数のすべての組合せにおいて、本手法の方が遅延時間を低減できていることが確認できる。平均して 11.2 [%] の低減率となっている。これは、本手法が地理的データ非依存性

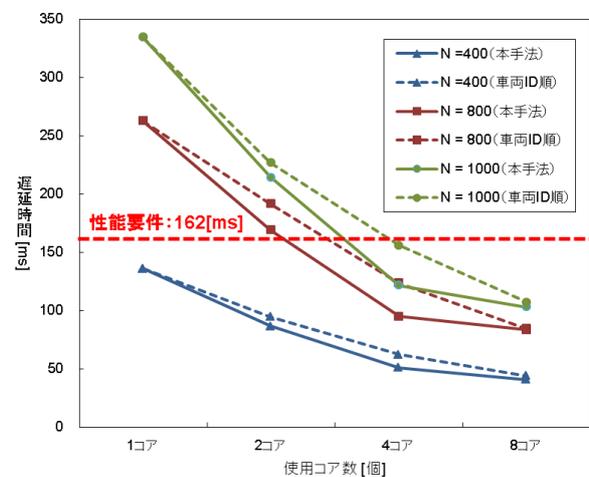


図 11 遅延時間
Fig. 11 Latency.

を用いてデータ分割を行うために、各コアに割り当てられたオペレータが処理すべき車両データに重複がなく、最低限の車両データを入力としてオペレータが駆動するからである。一方で、車両 ID 順の場合は、当該オペレータの計算に必要な車両データが他のオペレータに入力される可能性があり、その場合は、他のオペレータが保持する車両データを当該オペレータも保持する必要があって、オペレータ間で保持する入力データに重複が発生する。このことが、コア間通信の増加、ならびにオペレータの処理負荷を引き起こし、性能劣化を引き起こす。

アプリケーション性能要件の充足性について、本手法の場合に着目すると、N = 400 の場合は、1 コア使用時においてもアプリケーション要件の 162 [ms] を満たす一方で、N = 800、および N = 1,000 では、1 コア使用時においてそれぞれ 262.5 [ms]、334.4 [ms] であり、アプリケーションの性能要件を満たさないことが分かる。しかしながら、使用コア数を増加させることで、8 コア使用時では、遅延時間

を $N = 800$ で 83.5 [ms], $N = 1,000$ で 102.9 [ms] に低減でき、アプリケーション要件を満たしうることが確認できる。

8 コア使用時において車両 ID 順と本手法の差の絶対値が小さくなっている理由は、本手法の場合、地理的なデータ非依存性を考慮して並列化を行うために、地理的に車両密度の偏りが生じたときに、各コアが処理すべき車両データ量にばらつきが生じ、並列処理性能の伸びが観測されにくくなる可能性があるためである。この解決については、4.4.3 項でも考察するように、領域フィルタと道路列フィルタにおいて動的負荷分散の仕組みを設けることで対処可能と考える。一方、車両 ID 順の場合は、各コアで処理する車両数がほぼ一定であるため、絶対的な遅延時間の低減こそ本手法に劣るものの、地理的に車両密度の偏りが出たとしても、実行主体を増加させることで8 コア使用時に並列処理性能の伸びが観測できる。

図 12 に、本手法と車両 ID 順における $N = 1,000$ の遅延時間の内訳を示す。衝突警告部と衝突回避部それぞれが占める遅延時間の割合に着目すると、衝突回避部は、衝突警告部に対して、1 コア実行時においては 5.4% 、本手法による並列化では 10.2% ~ 19.4% 、車両 ID 順による並列化では 9.1% ~ 19.3% を占める。このことから、本アプリケーションにおいては、衝突回避部の遅延時間が、衝突警告部と比較して大きな割合でないことを示している。よって、衝突回避部が逐次実行であっても、衝突警告部を並列化することで遅延時間を短縮できていると考える。

図 13 に、本手法を用いた場合の遅延時間 ($N = 1,000$) のばらつきを箱ひげ図で示す。長方形とその両側に出たひげで、下から最小値、第1四分位点、中央値、第3四分位点、および最大値を表現している。8 コア実行時においては、中央値の 103.5 [ms] に対し、最大値が 106.7 [ms]、最小値が 96.1 [ms] であり、遅延時間が最大値であってもアプリケーション性能要件を充足していることが確認できる。また、この結果より、無線通信の遅延のばらつきが発生した場合でも、8 コア実行時で 162 [ms] - 106.7 [ms] = 55.3 [ms] を、

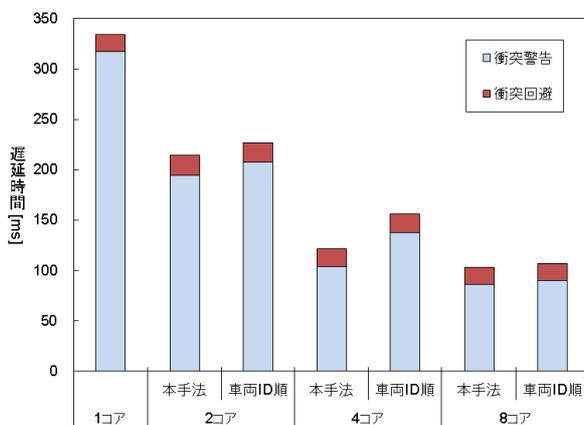


図 12 遅延時間の内訳

Fig. 12 The breakdown of the latency.

本評価で設定した遅延である 1 [ms] に上乗せした 56.3 [ms] まで、無線通信の遅延を許容できることが分かる。

4.4.3 台数効果

図 14 に本手法と車両 ID 順における台数効果を示す。縦軸が台数効果を表し、横軸が使用コア数を表す。車両台数 N を 400 台、 800 台、 $1,000$ 台と変化させたときの、使用コア数に対する台数効果を評価する。すべての項目において、本手法が車両 ID 順を上回る台数効果を得ている。これは、前述したように地理的なデータ非依存性によるデータ分割の効果と考える。本手法では、8 コア使用時において、台数効果が 3.14 倍~ 3.33 倍となっている。本実装では、衝突警告クエリのマップマッチングオペレータ、および TTCLessThan オペレータの部分のみが並列化され、衝突回避部は並列実行されていないため、衝突回避部の実行を並列化することで、さらなる台数効果を得ることが可能と考える。また、並列実行されている部分においては、時間の経過につれて1 コアあたりの処理負荷に偏りが生じる。なぜなら、地図上の DM クエリの発行ポイントは、静的に決定されるため、車両の移動によって、それぞれの発行ポイントにおける DM クエリの処理量変動するから

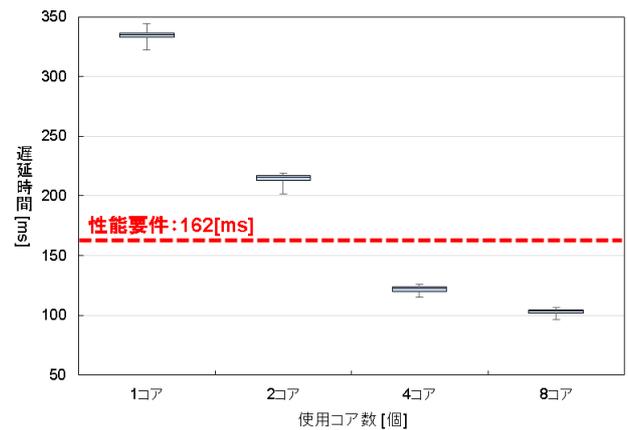


図 13 遅延時間のばらつき

Fig. 13 The distribution of the latency.

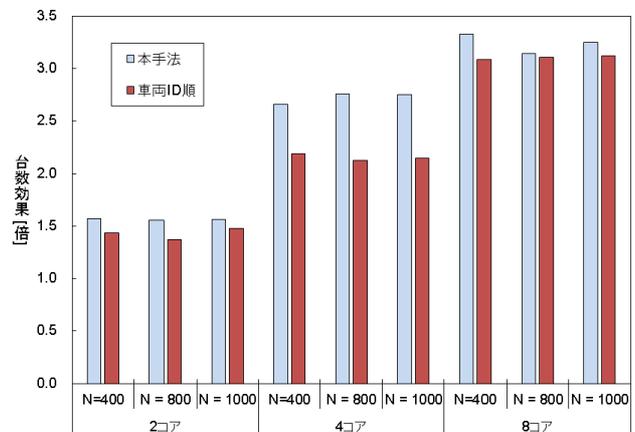


図 14 台数効果

Fig. 14 Scalability.

表 2 記述量の削減率

Table 2 The reduction rate of the description.

	記述ブロック数	記述リンク数
本手法	4	5
書き下し	19	34
削減率	78[%]	88[%]

である。そのため、負荷のかかるコアとそうでないコアが生まれ、1コアあたりの処理負荷に偏りが生じる結果となる。この点に関しては、図6における領域フィルタと道路列フィルタにおいて、動的負荷分散の仕組みを設けることで、台数効果の一層の向上が見込めると考える。

4.4.4 記述量

本手法を用いた場合の記述量の削減率を表2に示す。アプリケーション開発者は、並列ストリーム型DMが提供する発行ポイントとオペレータを、互いにリンクで接続することでDMクエリを記述する。本手法を、8並列実行に対応するN:N衝突回避アプリケーションの衝突警告部に適用した場合、アプリケーション開発者は、発行ポイントとオペレータ（以降、まとめてブロックと呼ぶ）を4個と、それらを接続するリンクを5本記述する必要がある。一方、本手法を用いずにアプリケーション開発者が並列化に適したDMクエリを直接書き下す場合は、ブロックを19個と、リンクを34本記述する必要がある。本手法を用いることで、記述ブロック数を78[%]削減でき、かつ記述リンク数を88[%]削減できている。4.3節に定義したように、記述量の増減がアプリケーション開発者の負担増減であるならば、本手法によりアプリケーション開発者の負担を軽減できることが分かる。

4.5 評価のまとめ

これらの評価結果をまとめると以下ようになる。N:N衝突回避アプリケーションのリアルタイム動作に関しては、N=1,000で、衝突警告の表示と衝突回避のブレーキ制御を運転シミュレータ上でリアルタイムに行うことができることを確認した。遅延時間に関しては、本手法と車両ID順を比較すると、すべての車両台数と使用コア数の組合せにおいて、本手法の方が遅延時間を低減できることを確認した。また、本手法において、N=1,000のとき、1コア使用時では334.4[ms]のところを、8コア使用時に102.9[ms]で処理可能であり、コア数を増加させることでアプリケーション要件である162[ms]を満たしうることを確認した。台数効果に関しては、8コア使用時に最大3.33倍の台数効果を確認した。記述量に関しては、本手法を用いることで8コア使用時に約8割の削減ができ、アプリケーション開発者の負担を軽減できることを確認した。以上より、並列ストリーム型DMが、車両台数が増加したとしても実行主体を増加させることで低遅延を維持できることを確認した。

5. 関連研究

DMの実装形態としては、蓄積型DM (PG-LDM, NT-LDM)が存在する[24]。蓄積型DMに本手法を適用し並列化することも可能である。しかしながら、先に述べたように、蓄積型DMは、車両台数増加による遅延時間の増加が顕著であることが文献[9]で報告されている。すなわち、N:N衝突回避アプリケーションで用いているマップマッチング処理において、遅延時間が車両台数に比例して増加し、車両50台のときに平均遅延時間が約30[ms]であって、ストリーム型DMと比較して約17倍の遅延が発生する。仮に1,000台を扱う場合は、マップマッチングのみで約600[ms]の遅延が発生することとなり、それに加えて、TTCLessThanと衝突回避の処理が加われば、N:N衝突回避アプリケーション(N=1,000)の1コアにおける全処理(マップマッチング+TTCLessThan+衝突回避)の平均遅延時間である344.4[ms]と比較して、著しい遅延が発生しうることが分かる。したがって、蓄積型DMを本手法で並列化して遅延時間を低減するよりも、ストリーム型DMに本手法を適用したほうが、本手法が目指す遅延時間低減の効果を一層発揮できると考える。

交通社会ダイナミックマップ[25]は、都市レベルの交通流の最適化、運転者への交通サービスの個別最適化等を目的とした基盤システムであり、レーンレベルの詳細地図を管理するDBMSと車両等の動的データを管理するDSMSから構成される。本研究のように多台数の車両データの低遅延処理手法を追求するというよりもむしろ、詳細地図の表現方法、物体のない領域の管理方法等に特徴を有する。

SPATIOWL[26]は、リアルタイムに位置情報を収集し解析するアプリケーション基盤であり、Hadoop[27]とその分散ファイルシステムであるHDFSをベースとして構築されている。本研究のような車両群のミリ秒単位の制御に用いるというよりも、たとえば、タクシーの人の乗り降りの分析、運輸業の集荷・配送のタイミング配信等の用途に用いられる。

多数の並列ストリーム処理基盤が以下のように開発されている。Storm[15]は、オープンソースで公開されているリアルタイムな並列ストリーム処理基盤である。入力データソースであるSpoutとデータ処理を行うBoltを組み合わせて、ストリームを処理するためのデータフローモデルを表現する。その際に、Spout、あるいはBoltごとに実行の並列度を指定することができ、さらにその並列度を実行時に動的に変更することも可能である。Spark streaming[16]は、入力ストリームデータを、RDDと呼ばれる数秒単位のバッチデータに区切り、それらのバッチデータをインメモリで処理することによりリアルタイムな並列ストリーム処理を実現する。InfoSphere Streams[17]は、ITSサービスのための、スケーラブルかつリアルタイムな並列ストリー

ム処理基盤である。開発者は、アプリケーションクエリを、SPADE と呼ばれる言語で、オペレータを接続したデータフローとして記述し、各オペレータを複数の実行主体に分散して配置することにより並列実行する。これらに代表される従来の並列ストリーム処理基盤と並列ストリーム型 DM の最大の相違点は、地理的なデータ非依存性に着目した並列処理向けのクエリ展開機能を有するか否かにある。従来の並列ストリーム処理基盤では、地理的なデータ非依存性を用いて並列処理に適した DM クエリを記述しようとすると、アプリケーション開発者が自ら、並列ストリーム型 DM が展開した後の DM クエリを書き下す必要があり、DM クエリの発行ポイントが多数であればあるほど、あるいは発行ポイントの変更があればあるほど、その労力が増加する。一方で、並列ストリーム型 DM は、アプリケーション開発者が、DM クエリと発行ポイントを埋め込んだ地図を作成することによって、並列実行に適した DM クエリを機械的に生成できる。これにより、アプリケーション開発者の労力を大幅に低減でき、かつ地理的なデータ非依存性を考慮せずに処理を並列化した場合よりも並列処理性能が得られる点が並列ストリーム型 DM の最大の特徴である。

6. まとめと今後の展開

本稿では、クラウド型自動運転を実現するための DM の重要な要件の 1 つに車両台数増加時の低遅延維持をあげ、DM クエリの地理的なデータ非依存性を利用して、並列処理向けのクエリを機械的に生成可能な並列ストリーム型 DM を提案した。車両台数の増加に対して処理量の増加が顕著となる N : N 衝突回避アプリケーションを取り上げ、評価したところ、並列ストリーム型 DM を用いると、車両台数増加時に低遅延を維持できることを確認した。

今後の展開としては、実際のエッジクラウド上に並列ストリーム型 DM と N : N 衝突回避アプリケーションを実装し、エッジクラウドを用いない場合と比較して遅延時間の違い等を評価することがあげられる。また、安全性と乗り心地を考慮したクラウド型自動運転における連続的な車両制御手法、およびクラウドからの速度制御による渋滞フリー走行等のアプリケーション開発を目指す。

参考文献

[1] 内閣府政策統括官：SIP（戦略的イノベーション創造プログラム）自動走行システム研究開発計画（2015）。

[2] Guicco, E.: How google's self-driving car works, IEEE Spectrum Online (online), available from <http://spectrum.ieee.org/automaton/robotics/artificial-intelligence/how-google-self-driving-car-works> (accessed 2017-10-06).

[3] Ericsson: Connected vehicle cloud, Ericsson (online), available from <https://www.ericsson.com/ourportfolio/industries-solutions/connected-vehicle> (accessed 2017-

10-06).

[4] Cisco: Passenger vehicles, Cisco (online), available from <http://www.cisco.com/web/strategy/transportation/passenger.html> (accessed 2017-10-06).

[5] Dresner, K. and Stone, P.: A multiagent approach to autonomous intersection management, *Journal of Artificial Intelligence Research*, Vol.31, No.1, pp.591-656 (2008).

[6] Tachet, R., Santi, P., Sobolevsky, S., Reyes-Castro, L., Frazzoli, E., Helbing, D. and Ratti, C.: Revisiting Street Intersections Using Slot-Based Systems, *PLoS ONE*, Vol.11, No.3, pp.1-9 (2016).

[7] ETSI: Intelligent transport systems (ITS); Vehicular communications; Basic set of applications; Local dynamic map (LDM), EN 302 895 (2014).

[8] 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, 情報処理学会研究報告, Vol.2013-ITS-53, pp.1-8 (2013).

[9] 山口晃広, 島田秀輝, 伊藤信一, 石原直樹, 本田晋也, 佐藤健哉, 高田広章: ストリーム LDM: データストリーム管理システムによる LDM の高速化, 第 11 回 ITS シンポジウム 2012, pp.127-130 (2012).

[10] 佐藤健哉, 山田真大, 島田秀輝, 金 榮柱, 本田晋也, 高田広章: Local Dynamic Map (LDM) の設計と評価, 問題点とその対策, 第 10 回 ITS シンポジウム 2011, pp.85-90 (2011).

[11] 鈴木有也, 牧戸知史, 佐藤健哉, 高田広章: LRTS: LDM リアルタイムシミュレータの開発と評価, 組込みシステムシンポジウム 2014, pp.75-83 (2014).

[12] Welford, A.: *Reaction Times*, pp.371-381, Academic Press (1980).

[13] ETSI: Intelligent transport systems (ITS); Communications architecture, EN 302 665 (2010).

[14] ETSI: Mobile Edge Computing - A key technology towards 5G, ETSI white paper No.11 (2015).

[15] Apache Storm Project: Apache Storm, Apache Storm Project (online), available from <http://storm.apache.org/> (accessed 2017-10-06).

[16] Apache Spark Project: Apache Spark Streaming, Apache Spark Project (online), available from <http://spark.apache.org/streaming/> (accessed 2017-10-06).

[17] Biem, A., Bouillet, E., Feng, H., Ranganathan, A., Riabov, A., Verscheure, O., Koutsopoulos, H. and Moran, C.: IBM infosphere streams for scalable, real-time, intelligent transportation services, *Proc. 2010 ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD '10)*, pp.1093-1104 (2010).

[18] 今井悟史, 宇式一雅, 藤野信次, 町田 守, 森谷正義, 蔡 晟尉, 間瀬公太, 丹羽栄二: 車車間通信サービスにおける CSMA/CA 通信品質の解析, 情報処理学会論文誌, Vol.51, No.3, pp.914-929 (2010).

[19] NTT ドコモ: ドコモ 5G ホワイトペーパー (2014).

[20] Nikaein, N. and Krco, S.: Latency for Real-Time Machine-to-Machine Communication in LTE-Based System Architecture, *European Wireless 2011*, pp.263-268 (2011).

[21] フォーラムエイト: UC-win/Road Ver.9 Driving Sim, フォーラムエイト (オンライン), 入手先 (<http://www.forum8.co.jp/product/ucwin/road/ucwin-road-1.htm>) (参照 2017-10-06).

[22] NTT ドコモ: 5G 無線アクセス技術, NTT ドコモテクニカルジャーナル, Vol.23, No.4, pp.18-29 (2016).

[23] 5G PPP: 5G Automotive vision, 5G PPP white paper on automotive vertical sector (2015).

- [24] SAFESPOT: SAFESPOT core architecture - LDM API and usage reference (2010).
- [25] 渡辺陽介, 高木建太郎, 手嶋茂晴, 二宮芳樹, 佐藤健哉, 高田広章: 協調型運転支援のための交通社会ダイナミックマップの提案, DEIM Forum 2015 (2015).
- [26] 玉井恭平, 脇谷 哲, 尾林俊文: 位置情報サービスのためのアプリケーション基盤, 雑誌 FUJITSU, Vol.64, No.1, pp.32-37 (2013).
- [27] Apache Hadoop Project: Apache Hadoop, Apache Hadoop Project (online), available from (<http://hadoop.apache.org/>) (accessed 2017-10-06).



鈴木 有也 (正会員)

2004年筑波大学大学院工学研究科電子・情報工学専攻博士課程修了。博士(工学)。同年同大学大学院システム情報工学研究科産学官連携研究員。2005年(株)豊田中央研究所入社。現在に至る。自動車システムにおけるソフトウェアの高信頼化技術, 自動車制御用ソフトウェアプラットフォーム, ITSプラットフォーム等の研究に従事。ACM会員。



佐々木 健吾 (正会員)

2010年東京大学大学院学際情報学府総合分析情報学コース修士課程修了。同年(株)豊田中央研究所入社。現在に至る。車車間通信, 非接触給電, Software-Defined Network等の研究に従事。IEEE会員。



佐藤 健哉 (正会員)

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



高田 広章 (正会員)

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