情報処理学会論文誌　デジタルプラクティス　Vol.3　No.3　32–43 (July 2022)

遅延値に基づく車両向けコンテナ型アプリケーションの動的エッジオフローディングの実装と評価

古澤 徹[1,2]　阿部 博[1]　中尾 彰宏[2]

受付日 2021年11月22日, 再受付日 2022年3月3日,
採録日 2022年4月22日

概要：車両アプリケーションをサーバで遠隔実行するオフローディングにより車載機の電力消費削減や開発コストの削減が可能になる．多くの車両アプリケーションはサーバの低遅延応答を必要とするが，車載機とクラウド間のモバイル通信遅延が発生するため確実な低遅延応答を実行できない課題がある．MECサービスの登場により，エッジサーバに対してオフローディングを行うことでオフローディングの安定実行が可能になると期待される．しかし，エッジサーバはクラウドと比べて利用可能リソースが限られることから，エッジサーバへのオフローディングは必要最小限に抑えることが望ましい．本研究では，車両向けコンテナ型アプリケーションのオフローディング先を，車載機と各サーバ間の応答時間に基づき車載機，エッジサーバ，クラウドの中から動的に切り替えるフレームワークを提案する．クラウドの応答時間の平均値が一定の閾値を下回る間はクラウドが，閾値を上回る間はエッジサーバがオフローディング先として動的に選定されることで，要求される低遅延応答を安定実行しつつエッジサーバの利用を最小限に抑えることが可能になる．Kubernetesとコンテナ仮想化技術を用いて本フレームワークを実装する．100ミリ秒以内の応答が求められるアプリケーションの動的オフローディング実験を行い，従来手法と比べてオフローディング処理のエラー率が最大91%削減されることを確認する．

キーワード：コネクティッドカー，エッジコンピューティング，エッジオフローディング，コンテナ，Kubernetes

Implementation and Evaluation of Latency-based Dynamic Edge Offloading of Containerized Applications for Connected Cars

Toru Furusawa[1,2]　Hiroshi Abe[1]　Akihiro Nakao[2]

Received: November 22, 2021, Revised: March 3, 2022,
Accepted: April 22, 2022

Abstract: Offloading, which is the remote execution of in-vehicle applications on a server, is expected to reduce power consumption and development costs of in-vehicle devices. Many in-vehicle applications require low-latency response from the server, but the mobile communication delay between the in-vehicle device and the cloud makes it impossible to perform reliable low-latency response. With the advent of MEC services, it is expected that offloading to edge servers will enable stable execution of offloading. However, offloading to the edge server should be kept to a minimum, since the edge server has limited resources available compared to the cloud. In this paper, we propose a framework that selects the offload target of containerized applications for vehicles among the in-vehicle device, edge servers, and cloud servers. The cloud server is selected as the offload destination while the average response timeof the cloud server is below a certain threshold, and the edge server is selected in other cases. This framework is implemented using Kubernetes and container virtualization technology. Experiments for offloading applications that require response within 100 ms show that our framework reduces the error rate of offload response time by up to 91% compared to the conventional method.

Keywords: connected car, edge computing, edge offloading, container, Kubernetes

1. はじめに

ネットワークに接続された車であるコネクティッドカーの普及が進んでいる．コネクティッドカーはネットワークを介してクラウド上のシステムと連携し，ナビゲーションや情報配信といったインフォテイメントサービスを実現している．今後は，運転支援や自動運転，車両センサ情報収集等の様々なコネクティッドサービスの実現が期待される．

クラウド上のサーバへの車両アプリケーションのコンピュテーションオフローディングは今後実現が期待されるユースケースの１つである [1]．特に，自律自動運転機能を搭載したコネクティッドカー（Connected and Autonomous Vehicle; CAV）のアプリケーションのオフローディングが期待される．CAV はサーバに匹敵する規模のコンピュータを搭載し，推論処理等の複雑な計算処理を行う様々なアプリケーションを常時実行する [2] ため，車両バッテリーを大きく消費する．このような処理を積極的にオフローディングすることで，車両バッテリー消費を節約し走行可能距離を拡大できる．また，オフローディングが中心になることで，車載機の求められる機能がシンプルになり車載機の開発コストが削減されると期待される．

しかし，CAV のアプリケーションの多くは低遅延応答が求められる [3]．特に，サーバからの遠隔運転を行うアプリケーションは常時安定して処理応答時間が往復 100 ミリ秒以内に収まる必要があるとされる [4]．しかし，クラウド上のサーバと接続する場合，無線アクセスネットワーク，モバイルコアネットワーク，そしてインターネットを介してサーバと接続する必要がある．各ネットワークでそれぞれ通信遅延が変動するため，End-to-End で常時 100 ミリ秒以内の処理応答時間を維持することは不可能であることから，クラウド上のサーバでオフローディングを安定的に継続することは困難である．

このような課題に対しエッジコンピューティングの適用が期待される．エッジコンピューティングは，クラウドと端末の中間に位置するエッジサーバで処理を行うことで低遅延処理や中継トラフィック削減を実現する新しいパラダイムである．特に，モバイル通信キャリアの設備内にエッジサーバを配置する Multi-access Edge Computing（MEC）は，安定かつ低遅延通信でサーバ接続が可能になる技術として着目されている [5]．コネクティッドカーはエッジコンピューティングの重要なユースケースの１つとして様々な研究が進められる [6]．

また，近年は Docker や Kubernetes（K8s）をはじめとするコンテナ仮想化技術とオーケストレーションソフトウェアがエッジコンピューティング基盤にも導入されており，車載機アプリケーション基盤としての利用も検討される．たとえば，すでに商用 MEC サービスとしてリリースされる MobiledgeX Edge Cloud[*1] や AWS Wavelength[*2] はいずれもマネージド型 K8s サービスを提供する．DENSO 社は，コネクティッドカーの Electronic Control Unit（ECU）を K8s で制御するプラットフォームを開発する [7]．このような動向を踏まえると，近い将来クラウドに加えてエッジサーバと車載機でも K8s で制御されるコンテナ型アプリケーションが動作するようになると予見される．そして，コンテナの特徴であるアプリケーションの可搬性から，クラウド，エッジサーバ，そして車載機間で動的にコンテナをスケジューリングすることも可能になると考えられる．

しかし，従来の K8s は，クラスタのノードの CPU やメモリを指標としてスケジューリングを行うため，ノードの地理的な場所やノード間の通信遅延はスケジューリングに用いられない．地理的に分散配置されるサーバを用いて端末との低遅延応答を実現するようにコンテナをスケジューリングするには，端末とサーバ間の通信遅延や距離等の情報を用いるスケジューリング手法が新たに必要となる．文献 [8] や文献 [9] では，エッジサーバの位置情報やエッジサーバ間の RTT 値を使用し，指定場所に近いエッジサーバにコンテナを効率よくスケジューリングする手法が提案される．このような手法は，端末とサーバ間の通信遅延が安定する環境においては低遅延応答を実現するスケジューリングが可能となるが，端末とサーバ間の距離や通信遅延が時事刻々と変化する環境においては，低遅延応答を安定して実現するスケジューリングは不可能である．車両のように，高速で移動しモバイル通信回線の品質や各サーバとの距離や通信遅延が大きく変化する環境においては，端末とサーバ間の遅延変動を考慮したスケジューリング手法が新たに必要となる．

このような課題を踏まえ，我々はコンテナ仮想化技術と K8s を用いた車両向けコンテナ型アプリケーションの動的エッジオフローディングフレームワークの開発に取り組んでいる [10]．本フレームワークでは，アプリケーションをクラウド，エッジサーバ，および車載機の中から最適な場所に動的に配置，実行することを可能にする．文献 [10] では，エッジサーバが複数存在する環境で，車両移動に伴い近接するエッジサーバが変化する場合において，車載機と複数のエッジサーバ，および車載機とクラウドサーバ間の遅延値を定期的に測定し，平均遅延値が最も小さくなるサーバに動的にオフローディング先を変更する手法が提案

1　トヨタ自動車株式会社
Toyota Motor Corporation, Chiyoda, Tokyo 100–0004, Japan
2　東京大学
The University of Tokyo, Bunkyo, Tokyo 113-0033, Japan

*1　https://mobiledgex.com/
*2　https://aws.amazon.com/jp/wavelength/

される．

しかし，文献 [10] の手法は，複数のエッジサーバ間でのオフローディング先変更を目的としており，エッジサーバとクラウド間のオフローディング先変更は考慮されていない．1 つのエッジサーバとクラウドが利用可能であり，クラウドへオフローディングしても車載機が要求する低遅延応答品質が確保される環境においても，文献 [10] の手法では基本的にエッジサーバが使用される．エッジサーバはクラウドと比較して利用可能なリソースが限られていることから，エッジサーバへのオフローディングは必要最小限に抑えることが望ましい．

そこで本稿では，1 つのエッジサーバとクラウドが利用可能な環境において，クラウドを優先してオフローディングを行うフレームワークの設計と実装を新たに提案する．具体的には，クラウドサーバの応答時間が事前に定義された閾値を下回る間はオフローディング先としてクラウドサーバを選定し，閾値を上回る間は一時的にエッジサーバにオフローディング先を切り替える．また，ネットワーク接続断等の理由で遠隔サーバへのオフローディングが不可能な場合は一時的にオフローディング先を車載機内部に切り替える．評価にあたり，車載機から LTE 回線を介してクラウドサーバに一定期間アクセスする際の遅延値の時系列データを事前に測定し，測定された遅延値を用いて検証環境にて車載機とクラウドサーバ間の通信遅延を付与する実験を行う．100 ミリ秒以内の応答時間を要求する車両アプリケーションを用いてオフローディング実験を行い，エッジサーバを使用しない手法と比較し，応答時間が 100 ミリ秒を超えるリクエスト数が最大で 91% 削減されることを示す．

以降の構成は以下のとおりである．続く 2 章では関連研究を紹介する．3 章と 4 章では，本研究で提案するエッジオフローディングフレームワークの設計と実装を紹介する．5 章では評価実験の結果と考察を述べる．最後にまとめと今後の課題について述べる．

2. 関連研究

コネクティッドカー向けにエッジコンピューティングを用いる研究は近年盛んに行われており [6]，エッジオフローディングに関する研究は最も活発な研究分野の 1 つである [1]．文献 [11] では，協調型自動運転の実現に向け，車両とエッジサーバ，および車両とクラウド間のネットワーク品質の測定値に基づき，車両が接続する遠隔運転制御アプリケーションの実行場所をエッジサーバとクラウド間で動的に切り替える手法が提案される．アプリケーションの実行場所をエッジサーバとクラウド間で動的に変更する点は本研究と類似するが，エッジサーバとクラウド双方に予めアプリケーションがインストールされ常時稼働する点が異なる．本研究では，オフローディング先のサーバ上のみコンテナが動作し，他のサーバではコンテナが削除されるため，オフローディング先でないサーバのリソースを節約することが可能である．

コンテナと K8s を使ったエッジコンピューティング実装に関する研究は複数なされている．文献 [8] では，エッジサーバが広域に多数分散配置されるエッジコンピューティング基盤において，コンテナと K8s を用いたエッジアプリケーションの効率的なデプロイメントとオーケストレーション手法が提案される．K8s のカスタムスケジューラが用いられ，標準の K8s スケジューラと比較してより高速かつ CPU 計算量の少ないコンテナのスケジューリングが実現される．文献 [9] ではフォグコンピューティング環境向けに，指定場所に近いエッジノードに優先的にコンテナを配置する K8s スケジューラ実装方法が提案される．これらはいずれも，エッジサーバに対して K8s を用いてコンテナをスケジューリングする手法が提案される点で本研究と類似するが，端末はコンテナの実行場所では無いことと，端末とエッジノード間の通信品質の変動は考慮されていないことが本研究とは異なる．

また，文献 [12] では，車両のコンテナ型アプリケーションをエッジにオフローディングするフレームワークとアルゴリズムが提案される．しかし，ネットワークの輻輳や遅延の影響と，コンテナのエッジオフローディングを実現するソフトウェア実装が考慮されていない点が本研究とは異なる．

3. エッジオフローディングフレームワークの設計

本章では，本研究で提案するフレームワークの設計について述べる．フレームワークの概要を**図 1** に表す．フレームワークは，車載機，エッジサーバ，そしてクラウドサーバから構成され，車載機上のクライアントは各サーバ上のアプリケーションにモバイルアクセス回線を介してア

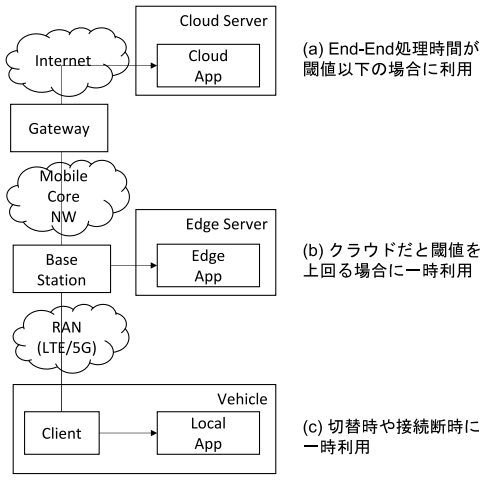


図 1　オフローディングフレームワークの概要
Fig. 1　Offloading framework overview.

クセスする．エッジサーバは，モバイル網の基地局に直接接続しており，車載機クライアントからは安定かつ低遅延で接続可能なサーバを利用する．車載機，エッジサーバ，クラウドサーバはいずれもコンテナ仮想化基盤およびオーケストレーションシステムを持つ．各環境ごとにオフローディング先サーバ上のアプリケーションとの応答時間が測定され，測定値に基づきオフローディング実施判断やオフローディング先切替が実行される．

本フレームワークは 3 つの特徴を持つ．(a) 応答時間の一定期間 $interval_{avg}$ における平均値が事前に定義された閾値（*timeout*）を下回る間は，クラウドサーバが優先的にオフローディング先として選定される．(b) クラウドサーバとの応答時間の平均値が *timeout* を上回り，かつエッジサーバとの応答時間の平均値が *timeout* を下回る場合は，その間に限りエッジサーバがオフローディング先として一時的に選定される．(c) モバイル回線の遅延増加や接続断，オフローディング先サーバの切替動作の影響で応答時間の平均値が *timeout* を下回りオフローディング先サーバが存在しない間は，クライアントアプリケーションはただちに接続先を車載機内で稼働するアプリケーションに切り替える．

以降，フレームワークの構成要素について述べる．

3.1　オフローディングアプリケーション

フレームワークのユーザが用意する，コンテナランタイム上で実行されるオフローディング対象のコンテナ型アプリケーションである．本アプリケーションは，車載機では常時起動しリクエストを待機する一方，エッジサーバとクラウドサーバについては，マルチクラスタスケジューラによって選定されたサーバ上でのみ稼働する．

3.2　クライアント

オフローディングアプリケーションを利用する，車載機に配置されるクライアントである．マルチクラスタスケジューラが指定するオフローディング先サーバ上のアプリケーションに対して一定間隔 $interval_{client}$ で，リクエストを送信しレスポンスを受け取る処理を継続する．

また，以下に述べる「ローカルフォールバック機能」を持つ．

- オフローディング先サーバの Probe Listener に対してヘルスチェック間隔 $interval_{probe}$ でヘルスチェックを行う．
- ヘルスチェックに 2 回連続失敗した場合は一定時間クライアントからのリクエストを車載機上のオフローディングアプリケーションに対して転送する．
- オフローディング先サーバの Probe Listener とのヘルスチェックに再び成功したら，リクエストをオフローディング先サーバに転送する．

3.3　応答時間測定器

車載機と各サーバ間の応答時間を一定間隔で測定する．具体的には，Probe Agent を車載機に，Probe Listener を各サーバに配置し，Probe Agent は一定間隔 $interval_{probe}$ で各 Probe Listener に対してヘルスチェック用リクエストを送信し応答時間を測定する．測定結果をマルチクラスタスケジューラに送信する．

3.4　マルチクラスタスケジューラ

クラウドサーバに配置され，エッジサーバとクラウドサーバ間でのオフローディング先変更の判定および実行を行う．一定間隔 $interval_{avg}$ で以下のオフローディング先サーバの判定処理を繰り返す．

- Probe Agent から送信される各サーバの応答時間を取得し，一定間隔 $interval_{avg}$ での平均値を計算する．
- クラウドサーバの応答時間の平均値が *timeout* を下回る場合はクラウドサーバをオフローディング先候補サーバとして選定する．
- クラウドサーバの応答時間の平均値が *timeout* を上回る場合は，各サーバの応答時間の平均値の最小値を確認し，その値が *timeout* を下回る場合はそのエッジサーバをオフローディング先候補サーバとして選択する．*timeout* を上回る場合はオフローディング先候補サーバは存在しないと判断する．
- オフローディング先候補サーバに，現在オフローディング中ではないサーバが 2 回連続で選択された場合，オフローディング先候補サーバをオフローディング先サーバに選定し，そのサーバ上の K8s クラスタにオフローディングアプリケーションをデプロイする．

さらに，車載機のクライアントに対して，アクセス先をオフローディング先サーバに変更するようリクエストを送信する．

4.　実装

本章では，前章で紹介するフレームワークの各構成要素の実装方法を説明する．実装の概要を**図 2** に表す．

4.1　オフローディングアプリケーション

フレームワークのユーザが用意する任意のアプリケーションが実行可能であるが，クライアントから gRPC リクエストを受け，処理を実行後に gRPC でレスポンスを返す仕様を満たす実装である必要がある．

4.2　クライアント

本実装では，クライアントアプリケーションと Envoy Proxy[*3] の 2 つのコンテナを用いて実装する．クライアン

[*3] https://www.envoyproxy.io/

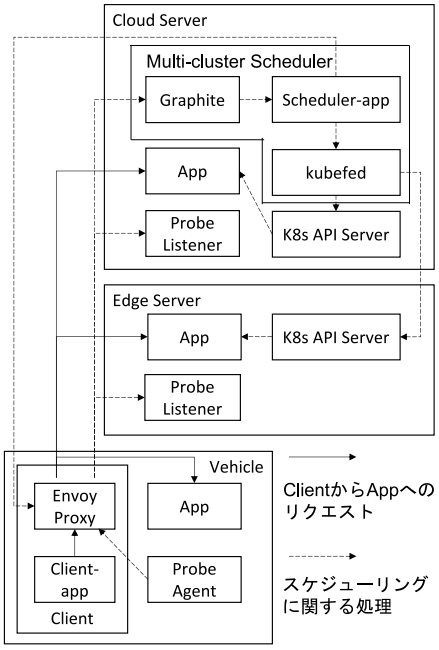


図 2　フレームワークの実装
Fig. 2　Framework implementation.

トアプリケーションは，オフローディングアプリケーションに対して一定間隔 $interval_{client}$ で gRPC リクエストを送信しレスポンスを受け取る．リクエストとレスポンスは Envoy Proxy を経由する構成とし，マルチクラスタスケジューラが指定するオフローディング先サーバへの転送と，ローカルフォールバック機能は Envoy Proxy が行う．具体的には，Envoy の Aggregate Cluster 機能を用いて，すべてのオフローディング先サーバのアプリケーションと車載機上のアプリケーションを clusters に指定する．ただし，車載機上のアプリケーションは最も優先度が低くなるよう clusters の末端として登録する．本設定により，オフローディング先サーバが正常稼働しない場合のみ車載機上のアプリケーションにリクエストが転送されるようになる．

4.3　応答時間測定器

3.3 に記述する機能はすべて Python アプリケーションとして実装する．

4.4　マルチクラスタスケジューラ

本実装では，Graphite[*4]，スケジューリングアプリケーション，そして KubeFed[*5] の 3 つのコンテナを用いて実装する．Graphite はオープンソースのモニタリングツールであり，本実装では送信される Probe Agent から送信される応答時間の集約と，一定間隔 $interval_{avg}$ ごとの応答時間の平均値算出に用いる．スケジューリングアプリケーションは，3.4 に記述する動作を Python アプリケーションとして実装する．KubeFed は，単一の API セットを用いて複数の K8s クラスタへの設定を可能とするオープンソースソフトウェアである．本実装では，オフローディング先サーバ上の K8s クラスタへのアプリケーションのデプロイを KubeFed の API を用いて行う．

[*4] https://graphiteapp.org/
[*5] https://github.com/kubernetes-sigs/kubefed

5.　評価と考察

本章では，通信遅延を擬似的に変動させる環境でエッジオフローディングフレームワークを動作させフレームワークの有効性を検証する．具体的には，車載機クライアントとオフローディングアプリケーションとの応答時間の変化と，アプリケーションの実行場所の変化を測定する．また，応答時間が一定時間を超えるリクエストはエラーとし，エラー発生率を測定する．あらかじめ一定時間車載機とクラウド間の通信遅延を測定し，その測定値を用いて検証環境にて車載機とクラウド間，およびエッジサーバとクラウド間の遅延を擬似的に発生させることで動作検証を行う．

車載機とクラウド間の通信遅延については，LTE 回線を用いて遅延値を測定する予備実験を行い，その測定値を用いて検証環境で遅延値を擬似的に再現する．

車載機とエッジサーバ間の通信遅延については，基地局に接続されるエッジサーバが利用可能な MEC サービスは未だ実現されていないため，現時点では実測することが不可能である．しかし，無線アクセス網（RAN）部分のデータレートに限定すれば 99.999% の信頼性を確保した上で LTE では片道 5 ミリ秒，5G では片道 1 ミリ秒以内の通信を実現するための 3GPP 規格が存在するため，端末とエッジサーバ間の通信も低遅延かつ信頼性の高い通信が実現されやすい [13]．また，端末とクラウド間の End-to-End 通信における遅延要因は，ホップ数と伝送距離が大きくなるほどモバイルコアネットワーク区間とインターネット区間が大部分を占めるとされる [14]．そこで，本研究では，モバイルコアネットワークとインターネットが End-to-End 通信における遅延要因の大部分を占めており，RAN 区間は安定かつ低遅延通信が実現されている環境を仮定し，検証環境における車載機とエッジサーバ間の通信遅延には固定的な通信遅延を付与する．しかし，端末と MEC 上のエッジサーバ間の通信遅延は RAN 区間の遅延が支配的になると推察されるものの，明確な標準規格が無いため，今後商用 MEC サービスが提供されても端末とエッジサーバ間の通信遅延は MEC サービスを提供する通信キャリアの実装に依存すると推測される．AT&T 社が MEC によって遅延値が 5 ミリ秒から 20 ミリ秒に収められると唱えている [15] ことから，本研究ではその上限値 20 ミリ秒を，検証環境における車載機とエッジサーバ間

の通信遅延値として用いる．

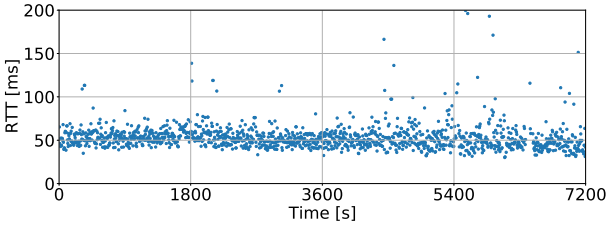
エッジサーバとクラウド間の通信遅延は，車載機とクラウド間の通信遅延と車載機とエッジサーバ間の通信遅延の差分値であるとし，検証環境で遅延値を擬似的に再現する．

5.1　クラウド環境との遅延測定

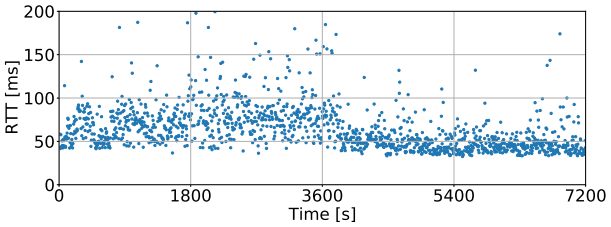
まず，車載機とクラウド間の通信遅延の変動を測定する．本研究では，au LTE 回線のモバイルルータと Softbank LTE 回線のモバイルルータを接続したノート PC を車両に搭載し，ノート PC から AWS EC2（東京リージョン）上に設置するサーバに向けて ping コマンドを 5 秒間隔で実行する際の RTT 値を測定する．静岡県裾野市周辺の幹線道路を約 2 時間走行し測定を行う．

測定結果を**図 3** および**表 1** に示す．au 回線，Softbank 回線ともに RTT の中央値は 60 ミリ秒を下回るが，しばしば RTT が散発的に 200 ミリ秒以上となることが確認できる．具体的には，au 回線では RTT が 100 ミリ秒以上となる割合は 3.8%，75 ミリ秒以上となる割合は 6.5%，Softbank 回線では RTT が 100 ミリ秒以上となる割合は 9.7%，75 ミリ秒以上となる割合は 28.1% となる．

1 章で述べたように，遠隔運転アプリケーションでは 100 ミリ秒以内の応答を継続することが求められるが，通信遅延だけでも 10% 近くのリクエストが 100 ミリ秒を超える．従って，遠隔運転のように安定的に 100 ミリ秒以内の応答が求められるアプリケーションのオフローディング先として，LTE 回線を経由したクラウド上サーバだけを利用することは現実的ではないと言える．



(a) au 回線



(b) Softbank 回線

図 3　車両とクラウド環境間の遅延測定結果

Fig. 3　Latency measurement results between vehicle and cloud environment.

表 1　モバイル回線遅延の平均値と中央値

Table 1　Mean and median of mobile line latency.

回線	平均値	中央値
au	91.5 ミリ秒	51.1 ミリ秒
Softbank	73.5 ミリ秒	57.1 ミリ秒

5.2　スケジューリング実験

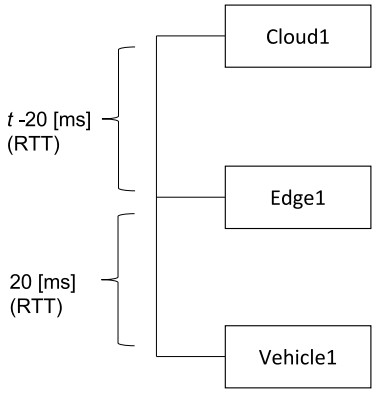
次に，前節の遅延測定結果を用いて，AWS EC2 を用いる環境で擬似的に通信遅延を発生させることでフレームワーク動作の評価を行う．

5.2.1　実験構成と実験手法

図 4 に示すように，車載機，エッジサーバ，およびクラウドサーバを想定する仮想マシンを AWS EC2 インスタンスを用いてそれぞれ構築する．EC2 インスタンスタイプは，Vehicle1 には t3.large（vCPU 数 2，メモリ 8 GiB）を，Edge1 と Cloud1 には m6i.4xlarge(vCPU 数 16，メモリ 64 GiB）をそれぞれ割り当てる．なお，Edge1 は通信キャリアの MEC サービスが提供する仮想マシンを用いることを想定している．NTT ドコモが現在提供する MEC サービス[*6] では，1 インスタンスあたり最大で vCPU 数 31，メモリ 126,976 MB の性能を持つ仮想マシンを利用することが可能であることから，m6i.4xlarge インスタンスタイプは MEC サービスの仮想マシンとしては一般的な性能を持つといえる．

いずれの仮想マシンも OS は ubuntu 18.04.5 server を用いる．

なお，同時に使用するコンテナ数が多い場合はエッジサーバとクラウドサーバではいずれも複数の仮想マシンを用いて Kubernetes クラスタを構成する手法が一般に用いられる．しかし，本研究では同時に使用可能なコンテナ数は評価の対象としていないことから，Edge1 と Cloud1 は



*t*は車載機とクラウド間の
往復遅延測定値を割り当て

図 4　実験構成

Fig. 4　Experiment environment.

[*6] https://developer.dev-portal.d-oic.com/portal/price/index.html

いずれも 1 つの仮想マシンを用いる．

各仮想マシン間の通信には，tc コマンドを用いて以下のとおり遅延を挿入する．車載機とエッジサーバ間の通信遅延は，往復 20 ミリ秒の固定的な遅延を付加する tc コマンドを実験開始前に一度実行する．車載機とクラウドサーバ間の通信遅延は，図 3 に示す au，Softbank のそれぞれの回線で 5 秒間隔で計測された往復遅延の時系列データを用いて，実験開始時刻から 5 秒ごとに計測された往復遅延時間を tc コマンドを用いて反映する．エッジサーバとクラウドサーバ間の通信遅延は，車載機とクラウドサーバ間の往復遅延値から 20 ミリ秒をひいた値を同様に tc コマンドを用いて反映する．

実験に用いる共通パラメータ値を**表 2** に示す．au 回線構成と Softbank 回線構成それぞれについて，エッジサーバの有無，ローカルフォールバック機能の有無，*timeout* 値の設定（100 ミリ秒または 75 ミリ秒）を変化させ，計 8 パターンずつ実験を行う．

実験用オフローディングアプリケーションには，クライアントからのリクエストを受信したら以下の処理を行うコンテナを用いる．

- ランダムな 1024 バイトのバイト列を生成する
- 生成されたバイト列の各々のバイトに対し，ランダム値との XOR を計算し上書きする処理を 200 回繰り返し実行する
- リクエストに対する応答を返す

また，このコンテナを実行する仮想マシンが Vehicle1，Edge1，Cloud1 のいずれであっても，コンテナを実行するポッドのリソースの要求値と制限値として以下の値を割り当てる．

- CPU: 300 m
- memory: 128 MiB

なお，この処理は単に CPU への負荷を与えることを目的とするものであり，特定の用途は想定されていない．

本実験構成にて，クライアントがオフローディングアプリケーションにリクエストを送信してから完了通知を受信するまでの応答時間と各処理の実行場所を測定する．また，*timeout* 以内に応答を受信しないリクエストはエラーとし，リクエストのエラー率をあわせて測定する．

表 2　実験パラメータ値
Table 2　Experiment parameters.

種類	値
$timeout$	100 ミリ秒または 75 ミリ秒
$interval_{client}$	1 秒
$interval_{probe}$	1 秒
$interval_{avg}$	10 秒

5.3　スケジューリング実験結果

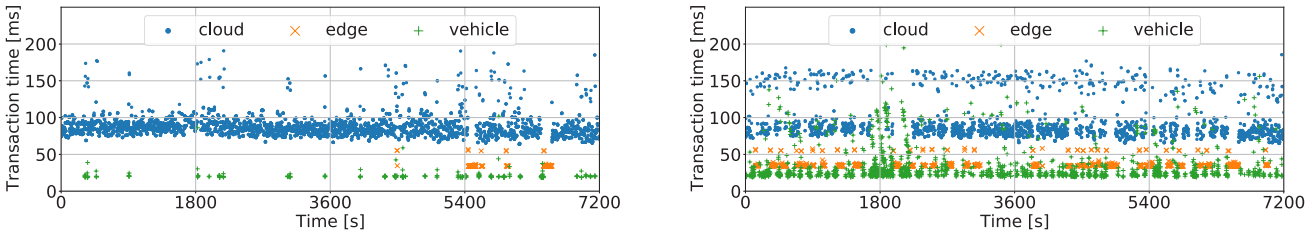
応答時間の時系列変化について，au 回線構成における実験結果を**図 5** に，Softbank 回線構成における実験結果を**図 6** に示す．また，実験パターンごとのアプリケーション実行場所とリクエストのエラー率，応答時間の平均値および期待値について，au 回線構成における実験結果を**表 3** に，Softbank 回線構成における実験結果を**表 4** に示す．

パターン au-(g)，au-(h)，sb-(g)，sb-(h) は，クライアントが接続先を切り替えることなくクラウドサーバのアプリケーションにリクエストを送信し続けるパターンであり，本研究におけるベースラインに位置づけられる．表 3 と表 4 が示すように，パターン au-(g) ではリクエストのエラー率は 10.8% にとどまるものの，パターン au-(h) では 51.9%，パターン sb-(g) では 37.6%，パターン sb-(h) では 73.9% と高いエラー率を示している．遠隔運転のように 100 ミリ秒以下の低遅延応答が継続して要求されるアプリケーションは，LTE 回線を経由してクラウドサーバにアクセスするだけではオフローディング処理を安定的に継続することは難しいことが確認できる．

次に，フォールバック機能だけ有とする実験パターンの結果を確認する．リクエストのエラー率は，パターン au-(e) では 5.4%，パターン au-(f) では 21.3%，パターン sb-(e) では 7.6%，パターン sb-(f) では 7.9% となっている．パターン sb-(f) については，ベースライン構成のパターン sb-(h) と比較すると最大で 90% 近くエラー率が減少している．このことからも，フォールバック機能により，モバイル通信遅延が急増する場合にもエラー発生を抑えて処理を継続できていることが確認できる．一方，パターン au-(f) では依然高いエラー率を示している．これは，フォールバック切替が発生する前のエラー数が多いことに起因する．本提案手法では，クライアントのリクエストが 2 回連続してタイムアウトするとフォールバック切替が発生する手法を採用している．すなわち，応答時間が *timeout* を上回るほど通信遅延が高まっても，少なくともフォールバック切替前の 2 回のリクエストはエラーとなる．図 3 から確認されるように，車両とクラウド環境間の通信遅延は離散的に発生していることから，フォールバック切替の発生頻度が大きくなるため，必然的にフォールバック切替前のエラー数も多くなる．

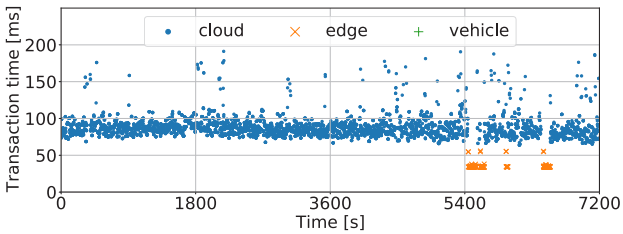
エラー率を減少させるには，ヘルスチェック間隔 $interval_{probe}$ を，アプリケーションへのリクエスト送信間隔 $interval_{client}$ よりも小さくすることが有効である．しかしながら，$interval_{probe}$ を減少させるほどヘルスチェックによるシステム負荷が高まることから，許容できる負荷とエラー率を鑑みてこれら閾値を設定する必要がある．

続いて，エッジサーバだけ有とする実験パターンのエラー率を確認する．パターン au-(c) では 4.6%，パターン

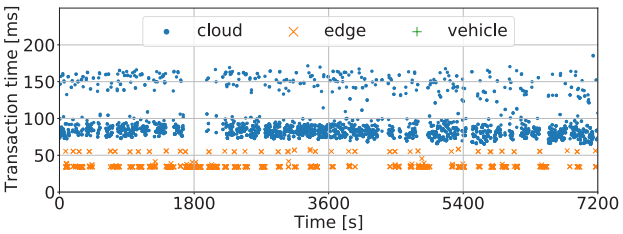


(a) エッジ有/フォールバック有/ *timeout* 100 ミリ秒

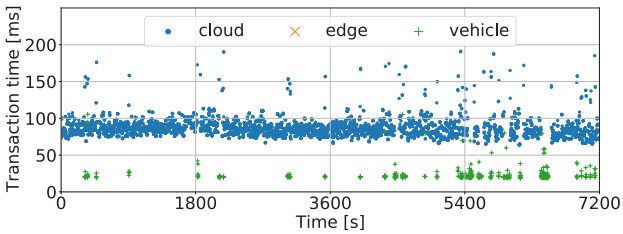
(b) エッジ有/フォールバック有/ *timeout* 75 ミリ秒



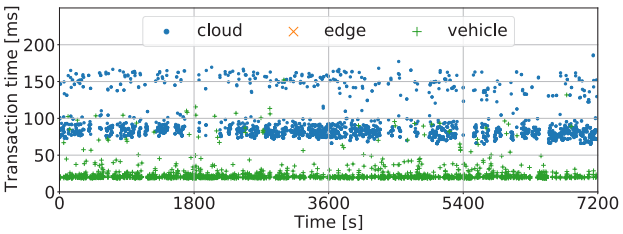
(c) エッジ有/フォールバック無/ *timeout* 100 ミリ秒



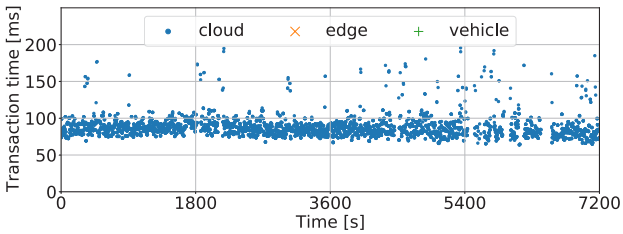
(d) エッジ有/フォールバック無/ *timeout* 75 ミリ秒



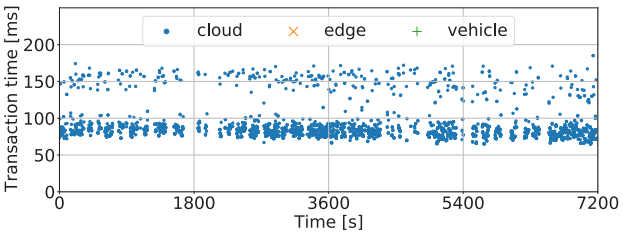
(e) エッジ無/フォールバック有/ *timeout* 100 ミリ秒



(f) エッジ無/フォールバック有/ *timeout* 75 ミリ秒



(g) エッジ無/フォールバック無/ *timeout* 100 ミリ秒



(h) エッジ無/フォールバック無/ *timeout* 75 ミリ秒

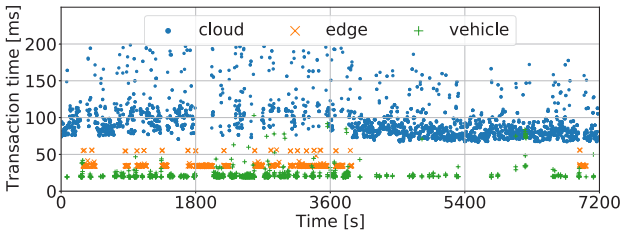
図 5　応答時間の変化（au 回線）

Fig. 5　Changes in response time (au line).

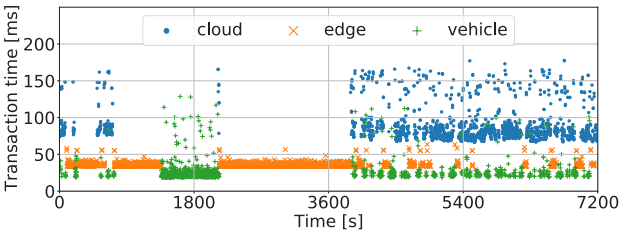
au-(d) では 34.1%，パターン sb-(c) では 21.9%，パターン sb-(d) では 23.0% であり，ベースライン構成と比較するといずれもエラー率が改善されているものの，フォールバック機能だけ有とするパターンよりもエラー率が高い結果となっている．これは，クラウドサーバからエッジサーバへクライアントの接続先切替が完了するまでに要する時間が，フォールバック機能と比較して大きいためである．本提案手法では，エッジサーバへの切替は Probe Agent と Probe Listener によるヘルスチェック結果を用いてマルチクラスタスケジューラが各 K8s クラスタと車載機の Envoy Proxy に対してリクエストを行うことで実行される．マルチクラスタスケジューラの実行間隔 $interval_{avg}$ はヘルスチェック間隔 $interval_{probe}$ よりも必然的に大きく設定されることに加え，マルチクラスタスケジューラのオフローディング先サーバ切替は少なくとも $2interval_{avg}$ の時間を要する．本実験では $interval_{avg}$ を 10 秒としているため，エッジサーバへの切替には少なくとも 20 秒を要する．

$interval_{probe}$ と $interval_{avg}$ の閾値をともに小さくすることでエッジサーバへの切替時間を短縮することは可能である．しかし，$interval_{probe}$ を小さくすると応答時間測定による負荷が，$interval_{avg}$ を小さくすると不要な切替が多発する可能性がそれぞれ高まるデメリットがあるため，システム全体の規模や負荷をふまえてこれらパラメータを調整する必要がある．

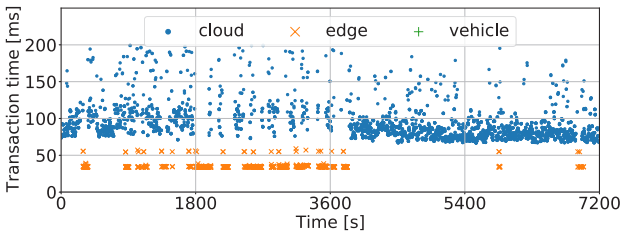
次に，提案手法，すなわちフォールバック機能とエッジサーバの両方を有とする実験パターンでのエラー率を確認



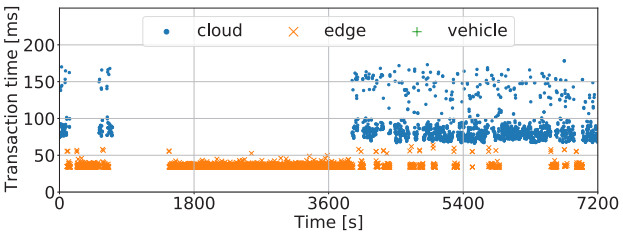
(a) エッジ有/フォールバック有/ *timeout* 100 ミリ秒



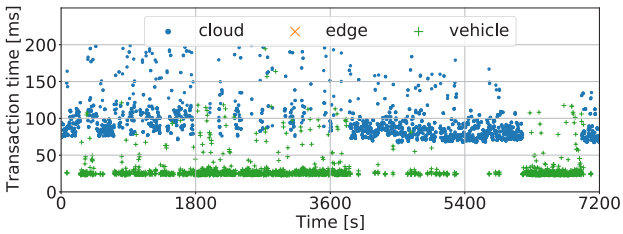
(b) エッジ有/フォールバック有/ *timeout* 75 ミリ秒



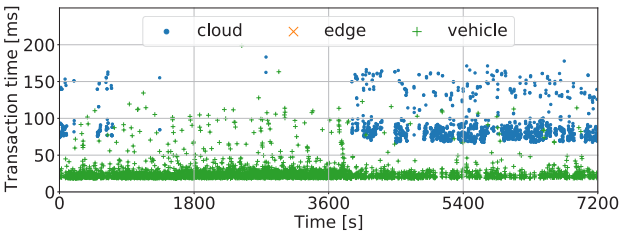
(c) エッジ有/フォールバック無/ *timeout* 100 ミリ秒



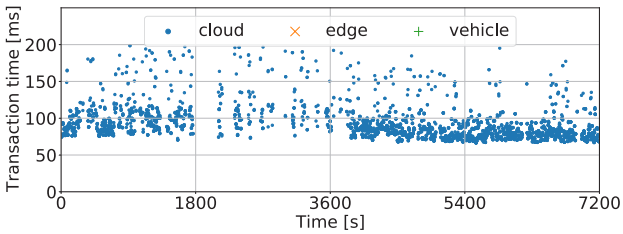
(d) エッジ有/フォールバック無/ *timeout* 75 ミリ秒



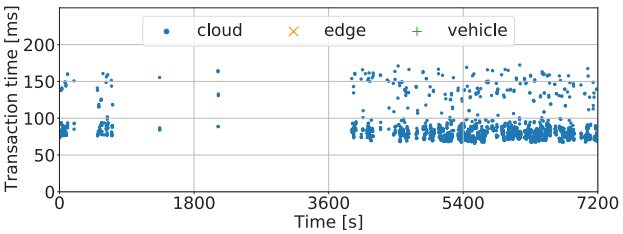
(e) エッジ無/フォールバック有/ *timeout* 100 ミリ秒



(f) エッジ無/フォールバック有/ *timeout* 75 ミリ秒



(g) エッジ無/フォールバック無/ *timeout* 100 ミリ秒



(h) エッジ無/フォールバック無/ *timeout* 75 ミリ秒

図 6　応答時間の変化（Softbank 回線）

Fig. 6　Changes in response time (Softbank line).

表 3　アプリケーション実行場所の割合と応答時間の期待値および中央値（au 回線）

Table 3　The percentage of application execution locations and the expected and median response times (au line).

実験パターン				実行場所の割合				応答時間	
パターン	エッジ	フォールバック	*timeout*	Cloud	Edge	Vehicle	Error	期待値	中央値
au-(a)	有	有	100 ミリ秒	91.8%	2.2%	4.6%	1.4%	85.4 ミリ秒	84.7 ミリ秒
au-(b)	有	有	75 ミリ秒	51.7%	12.5%	18.4%	17.4%	64.6 ミリ秒	74.4 ミリ秒
au-(c)	有	無	100 ミリ秒	91.9%	3.5%	0.0%	4.6%	86.7 ミリ秒	84.8 ミリ秒
au-(d)	有	無	75 ミリ秒	52.5%	13.4%	0.0%	34.1%	78.5 ミリ秒	80.2 ミリ秒
au-(e)	無	有	100 ミリ秒	89.6%	0.0%	5.0%	5.4%	84.6 ミリ秒	84.4 ミリ秒
au-(f)	無	有	75 ミリ秒	46.7%	0.0%	32.0%	21.3%	62.6 ミリ秒	75.5 ミリ秒
au-(g)	無	無	100 ミリ秒	89.2%	0.0%	0.0%	10.8%	95.0 ミリ秒	86.2 ミリ秒
au-(h)	無	無	75 ミリ秒	48.1%	0.0%	0.0%	51.9%	62.6 ミリ秒	75.5 ミリ秒

表 4　アプリケーション実行場所の割合と応答時間の期待値および中央値（Softbank 回線）
Table 4　The percentage of application execution locations and the expected and median response times (Softbank line).

実験パターン				実行場所の割合				応答時間	
パターン	エッジ	フォールバック	*timeout*	Cloud	Edge	Vehicle	Error	期待値	中央値
sb-(a)	有	有	100 ミリ秒	64.7%	13.8%	14.8%	6.7%	74.3 ミリ秒	77.8 ミリ秒
sb-(b)	有	有	75 ミリ秒	29.8%	44.0%	19.3%	6.9%	50.5 ミリ秒	35.0 ミリ秒
sb-(c)	有	無	100 ミリ秒	65.6%	12.5%	0.0%	21.9%	84.3 ミリ秒	81.8 ミリ秒
sb-(d)	有	無	75 ミリ秒	31.1%	45.9%	0.0%	23.0%	56.7 ミリ秒	36.7 ミリ秒
sb-(e)	無	有	100 ミリ秒	52.7%	0.0%	39.7%	7.6%	65.8 ミリ秒	73.9 ミリ秒
sb-(f)	無	有	75 ミリ秒	26.3%	0.0%	65.8%	7.9%	43.5 ミリ秒	24.3 ミリ秒
sb-(g)	無	無	100 ミリ秒	62.4%	0.0%	0.0%	37.6%	93.1 ミリ秒	85.4 ミリ秒
sb-(h)	無	無	75 ミリ秒	26.1%	0.0%	0.0%	73.9%	89.0 ミリ秒	79.9 ミリ秒

する．パターン au-(a) では 1.4%，パターン au-(b) では 17.4%，パターン sb-(a) では 6.7%，パターン sb-(b) では 6.9% となる．特にパターン sb-(b) について着目すると，ベースライン構成のパターン sb-(h)，エッジだけ有とするパターン sb-(d)，フォールバック機能だけ有とするパターン sb-(f) と比較し，それぞれ 91%，70%，13% ずつエラー率が減少している．

さらに，実行場所がクラウドサーバの割合と実行場所がエッジサーバの割合の和をオフローディング実行割合と定義し，オフローディング実行割合を確認する．パターン sb-(h) のオフローディング実行割合は 26.1%，パターン sb-(d) のオフローディング実行割合は 31.1%+45.9%=77.0%，パターン sb-(f) のオフローディング実行割合は 26.3% であるのに対し，パターン sb-(b) のオフローディング実行割合は 29.8%+44.0%=73.8% と高い割合を示す．これらの結果から，本提案手法はエラー率を最も低減しつつ，70% 以上の高いオフローディングを実現できることが確認できる．また，sb-(h) の結果が示すように，クラウドだけだと 73.9% ものリクエストがエラーとなるほど通信遅延が大きい環境においても，sb-(b) の結果が示すように，提案手法を用いることでエッジサーバの実行割合を 44.0% に抑えられていることも確認される．

フォールバック機能だけ有のパターンよりもエラー率が改善している理由は，本実験では車載機とエッジサーバ間の通信遅延は往復 20 ミリ秒に固定されており，エッジサーバへのオフローディング発生中は応答遅延が *timeout* を超えることが無いことからフォールバック切替が発生せず，オフローディング先切替の発生数が最も少なくなるためである．エッジサーバだけ有とする構成ではオフローディング先切替に時間を要しエラーが多く発生する問題があるが，本構成では，マルチクラスタスケジューラがエッジサーバへの切替を準備する間にもフォールバック機能が補完的に車載機にオフローディングを行うことでエラー率の低減に成功している．

なお，本提案手法において *timeout* を 75 ミリ秒とする場合，モバイル回線遅延の中央値は au 回線より Softbank 回線のほうが高いにも関わらず，パターン sb-(b) のエラー率が 6.9%，パターン au-(b) のエラー率が 17.4% となり，au 回線のほうが高いエラー率が発生している．これは，実験時の au 回線のほうが通信遅延のばらつきが大きく，頻繁に *timeout* を上回る事象が生じていることに起因する．フォールバック機能だけ有とするパターンと同様に，フォールバック切替が発生するには少なくとも 2 回のエラーが発生するため，*timeout* を上回る通信遅延が頻繁に発生するほど，エラー率は高くなる．エラー率を減少させるにはヘルスチェック間隔を小さくすることが有効である．

以上に述べるように，提案手法を用いることでエラー率を減少させつつエッジの実行割合を低く抑えることに成功している．しかし，au-(a) の結果が示すように，最もエラー率が低くなる場合でもエラー率が 1.4% を示している．従って，提案手法は少なくとも数 % のエラー率が許容できるアプリケーションのみ適用可能であるといえる．本研究で想定するサーバからの遠隔運転アプリケーションは，往復 100 ミリ秒以内の応答が期待されるものの，現時点では許容されるエラー率については明確に定義されていない [4]．よって，遠隔運転アプリケーションを提案手法に用いる場合は，アプリケーションをエラー率が一定値以上許容される機能と許容されない機能に分類し，各機能をコンテナ型マイクロサービスに分割し，許容される機能のマイクロサービスのみ，提案手法の動的エッジオフローディング対象とすることが有効である．

6.　おわりに

本研究では，車両向けコンテナ型アプリケーションのオフローディング先を車載機，エッジサーバ，クラウドの中から選定し動的に切り替えるフレームワークを提案した．本フレームワークでは，車載機と各サーバ間の応答時間が測定され，応答時間の平均値に基づきオフローディング先サーバが切り替えられる．クラウドの応答時間の平均値が一定の閾値を下回る場合はクラウドが，閾値を上回る場合はエッジサーバが，そしてネットワークの接続断や切替時

の影響でクラウドとエッジサーバどちらもオフローディング先サーバとして使用できない場合には車載機が，オフローディング先サーバとして選定される．Kubernetes とコンテナ仮想化技術を用いて本フレームワークを実装し，複数モバイル通信キャリアの LTE 回線の通信遅延を車載機とクラウド間に付与する環境にて本フレームワークを動作させる実験を行った．100 ミリ秒以内の応答時間を要求する車両アプリケーションのオフローディングを行い，従来手法と比較してエラー率が最大で 91% 減少しつつ，オフローディング実行割合が 70% 以上となることを示した．

今後の課題と展望を以下に述べる．本フレームワークは現状，オフローディング先の切り替えが発生するたびにコンテナが初期化される．そのため，切替前のコンテナが起動後に生成するファイルやメモリ情報を切替先にも移行するには別の仕組みが求められる課題がある．今後は，共有ストレージやサーバ間での個別送信を用いた切替先へのファイルマイグレーション機能や，CRIU[*7] を用いるプロセスのメモリ情報のマイグレーション機能をフレームワークに組み込むことを検討している．

また，本稿では車載機からエッジサーバへのアクセスには固定的な遅延値を割り当てる実験をおこなっている．しかし，現実にはエッジサーバへのアクセスにおいても遅延は変動する．特に，今後エッジサーバは多数地理的に分散配置されていくと見込まれるが，通信キャリアによってエッジサーバの展開地域やアクセス方法なども異なる．今後は，複数の通信キャリアが提供するエッジサーバとの通信遅延値を測定し評価を行う予定である．

参考文献

[1] Lin, L., Liao, X., Jin, H. and Li, P.: Computation Offloading Toward Edge Computing, *Proc. of the IEEE*, Vol.107, No.8, pp.1584–1607 (online), DOI: 10.1109/jproc.2019.2922285 (2019).

[2] Kopelias, P., Demiridi, E., Vogiatzis, K., Skabardonis, A. and Zafiropoulou, V.: Connected & autonomous vehicles–Environmental impacts–A review, *Science of the total environment*, Vol.712, p.135237 (2020).

[3] Wang, Y., Liu, S., Wu, X. and Shi, W.: CAVBench: A benchmark suite for connected and autonomous vehicles, *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, IEEE, pp.30–42 (2018).

[4] Kang, L., Zhao, W., Qi, B. and Banerjee, S.: Augmenting Self-Driving with Remote Control: Challenges and Directions, *Proc. of the 19th International Workshop on Mobile Computing Systems & Applications*, HotMobile '18, New York, NY, USA, Association for Computing Machinery, p.19–24 (online), DOI: 10.1145/3177102.3177104 (2018).

[5] Shi, W., Pallis, G. and Xu, Z.: Edge Computing [Scanning the Issue], *Proc. of the IEEE*, Vol.107, No.8, pp.1474–1481 (online), DOI: 10.1109/JPROC.2019.2928287 (2019).

[6] Liu, S., Liu, L., Tang, J., Yu, B., Wang, Y. and Shi, W.: Edge Computing for Autonomous Driving: Opportunities and Challenges, *Proc. of the IEEE*, Vol.107, No.8, pp.1697–1716 (2019).

[7] Koizumi, S., Kai, Y. J. and Gupta, A.: Deploy Anywhere: An End to End Kubernetes based Mobility Service Framework, *Denso Technical Review*, Vol.25 (2020).

[8] Ogbuachi, M. C., Reale, A., Suskovics, P. and Kovács, B.: Context-Aware kubernetes scheduler for edge-native applications on 5 g, *Journal of Communications Software and Systems*, Vol.16, No.1, pp.85–94 (online), DOI: 10.24138/jcomss.v16i1.1027 (2020).

[9] Santos, J., Wauters, T., Volckaert, B. and De Turck, F.: Towards network-aware resource provisioning in Kubernetes for fog computing applications, *2019 IEEE Conference on Network Softwarization (NetSoft)*, IEEE, pp.351–359 (2019).

[10] 古澤　徹，阿部　博，中尾彰宏：遅延値に基づき車両向けコンテナ型アプリケーションの動的エッジオフローディングを実現するフレームワークの開発，ネットワークシステム研究会，電子情報通信学会，Vol.2021, No.3 (2021).

[11] Sasaki, K., Makido, S. and Nakao, A.: Vehicle Control System for Cooperative Driving Coordinated Multi -Layered Edge Servers, *2018 IEEE 7th International Conference on Cloud Networking (CloudNet)*, pp.1–7 (online), DOI: 10.1109/CloudNet.2018.8549396 (2018).

[12] Tang, J., Yu, R., Liu, S. and Gaudiot, J.-L.: A Container Based Edge Offloading Framework for Autonomous Driving, *IEEE Access*, Vol.8, No.Vm, pp.33713–33726 (2020).

[13] Parvez, I., Rahmati, A., Guvenc, I., Sarwat, A. I. and Dai, H.: A Survey on Low Latency Towards 5G: RAN, Core Network and Caching Solutions, *IEEE Communications Surveys Tutorials*, Vol.20, No.4, pp.3098–3130 (online), DOI: 10.1109/COMST.2018.2841349 (2018).

[14] Al-Saadeh, O., Wikstrom, G., Sachs, J., Thibault, I. and Lister, D.: End-to-End Latency and Reliability Performance of 5G in London, *2018 IEEE Global Communications Conference (GLOBECOM)*, pp.1–6 (online), DOI: 10.1109/GLOCOM.2018.8647379 (2018).

[15] AT&T Labs & AT&T Foundry: AT&T Edge Cloud (AEC) White Paper, 〈https://about.att.com/content/dam/innovationdocs/Edge_Compute_White_PaperFINAL2.pdf〉.

[*7] https://criu.org/

古澤 徹（正会員）

トヨタ自動車株式会社 コネクティッドカンパニー コネクティッド先行開発部 Info-Tech E2E コンピューティンググループシニアリサーチャー．東京大学大学院学際情報学府博士課程在籍．

阿部 博（正会員）

トヨタ自動車株式会社 コネクティッドカンパニー コネクティッド先行開発部 Info-Tech E2E コンピューティンググループ主幹/シニアリサーチャー/博士（情報科学）.

中尾 彰宏（正会員）

1991 年 3 月東京大学理学部卒業．1994 年 3 月東京大学工学系研究科修士修了．同年 4 月日本アイ・ビー・エム株式会社入社（2005 年 退 社）．2000 年 3 月 Princeton University, Computer Science, M.S. 修 了．2005 年 3 月 Princeton University, Computer Science, Ph.D. 修了．同年 4 月東京大学大学院情報学環助教授．2007 年 4 月同大学大学院情報学環准教授．2014 年 2 月同大学大学院情報学環教授．2016 年 4 月同大学大学院情報学環学際情報学専攻長．2019 年 4 月同大学総長補佐．2019 年 10 月同大学大学院情報学環副学環長．2020 年 4 月同大学総長特任補佐．2021 年 4 月同大学大学院工学系研究科教授．現在，東京大学大学院工学系研究科システム創成学専攻教授・東京大学総長特任補佐.