1N-03

二重エッジアーキテクチャ設計方法の提案と MQTT-Bridge を用いたプロトタイプによる評価

宇野 聡将† 青山 幹雄†

南山大学 大学院 理工学研究科 ソフトウェア工学専攻†

1. 研究の背景と課題

コネクテッドカーにおいて大量データを処理するために，エッジコンピューティング（エッジ）が注目されている[1]. 本稿では 2 層のエッジにデータ配信処理を分散する二重エッジアーキテクチャを提案する. エッジ間でブリッジによるメッセージ共有を行い，スケールアウト可能な設計方法を提案する. プロトタイプを実装し，設計方法の有用性を示す.

1. 関連研究

1.1 エッジコンピューティング

ネットワークのエッジにクラウドの機能の一部を配置して分散処理するアーキテクチャである[5].

1.2 Publish/Subscribe(Pub/Sub)アーキテクチャ

メッセージの非同期通信アーキテクチャである. 国際標準として MQTT があり[3]，ブローカ間でメッセージを共有するブリッジ機能がある.

2. アプローチ

エッジに Pub/Sub を適用し，ブローカをエッジに配置する. ブローカは 1 つのメッセージ受信に対し，複数のサブスクライバに同一のメッセージを配信する場合がある. そのためクライアントの増加によりメッセージ配信処理のスケーラビリティが問題となる. そこで，以下を満たすアーキテクチャ設計方法を提案する.

(1) MQTT のブリッジを用いた低遅延でデータを共有可能とするアーキテクチャ

(2) エッジ上のブローカの稼働数を増加し，スケーラビリティを確保とするアーキテクチャ

3. アーキテクチャ設計方法論

3.1 二重エッジアーキテクチャ

二重エッジアーキテクチャを図 1 に示す. 提案アーキテクチャはコネクテッドカーにおいて，自動車から発生する大量データ処理を目的とする. 車外にのみエッジを設ける従来アーキテクチャ[1]に対し，車載と車外に二重エッジを設け，処理データを適切に制御し，車載での低遅延処理を実現する.

エッジ間のデータ配信方法として MQTT ブリッジ機能[3]を用いる. 車載エッジ上のブローカをローカルブローカ，ブリッジを介した車外エッジ上のブローカをリモートブローカとする. 大量データの配信処理の負荷増大に対し，ローカルブローカがボトルネックとなる問題に対し，リモートブローカに配信処理を分散することでブローカの負荷軽減が可能となる.

A Design Methodology of Double Edge Architecture and Its Evaluation with a Prototype Using MQTT-Bridge

† Toshimasa Uno, Mikio Aoyama, Graduate School of Science and Engineering, Nanzan University.

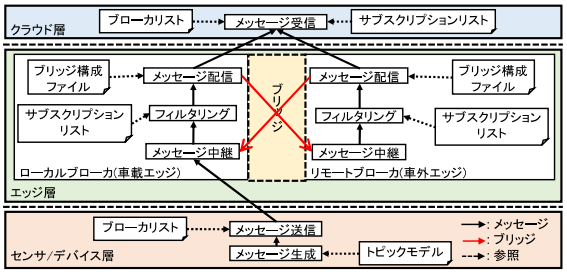

図 1 二重エッジアーキテクチャ

3.2 ブリッジを用いたメッセージング方法

本稿でのブリッジについて以下に定義する. また，ブリッジを有効化したブローカの挙動を図 2 に示す.

(1)2 つのブローカ間で，一方が受信したメッセージを他方に送信して共有可能とする通信である.

(2)ブリッジ構成ファイルでトピックを指定することで，共有するメッセージを限定可能とする.

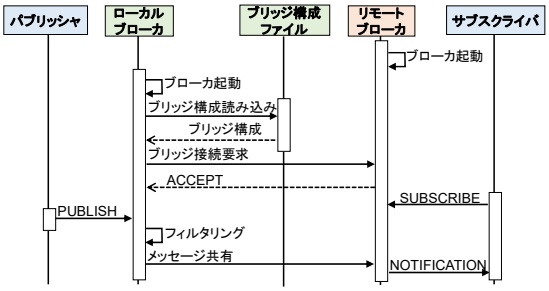

図 2 ブリッジの振る舞い

ブリッジによるメッセージングはブリッジ構成ファイルで指定したトピックに基づく. トピック構造の例を図 3 に示す. トピックはメッセージに含まれたデータに基づいて決定するリソース部とメッセージを一意に識別可能とする ID 部で構成される. またトピックは階層構造をとることから，リソース部を上位層から下位層に詳細化するよう決定する. ブリッジでメッセージ配信を分散する際に，エッジ毎に異なる上位のトピックをワイルドカード#を用いることで下位のトピックを抽象化して指定可能である. これにより，複雑なトピック指定を必要としない効率的にメッセージを分散可能となる.

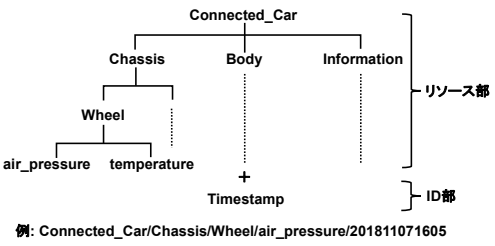

図 3 トピック構造

4. プロトタイプの実装と適用事例

4.1 プロトタイプの実装

プロトタイプの実行環境および実装に用いたソフトウェア情報を表 1, 表 2 に示す.

表 1 実行環境

システム名	デバイス	ローカルブローカ	リモートブローカ	クラウド
ハードウェア	Raspberry Pi 3 B+	Raspberry Pi B+	Raspberry Pi 3 B+	MacBook Pro
OS	Debian 9.6	Debian 9.6	Debian 9.6	OS X El Capitain
プロセッサ	Cortex-A53	ARM1176JZF-S	Cortex-A53	Intel Core i5

表 2 ソフトウェア情報

ソフトウェア	実装範囲	バージョン
Mosquitto	ブローカ	1.4.10
Paho-MQTT client	クライアント	1.4.0

4.2 適用事例

コネクテッドカーのエッジ連携に提案アーキテクチャを適用する. プロトタイプの構成を図 4 に示す. ローカルブローカはデバイスが生成したメッセージをあらかじめ定めた割合(メッセージ分散率: 0%, 25%, 50%, 75%)に基づいてリモートブローカに分散する. 全てのメッセージを 2 つのブローカによって重複なくクラウドに配信する. このときのローカルブローカの CPU 使用率を測定することでブリッジによって得られるスケーラビリティを検証する. また遅延時間を測定するための Raspberry Pi を稼働する. 同一ノード上でパブリッシャ, サブスクライバを実行し, ローカルブローカにタイムスタンプを含んだメッセージを送信, 受信することで遅延時間を測定する.

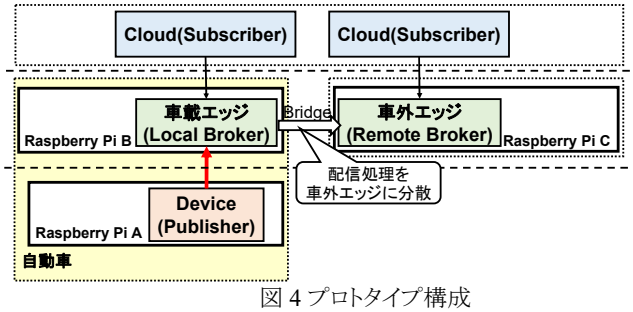

図 4 プロトタイプ構成

5. プロトタイプの評価

図 4 に示したプロトタイプを 4 パターンのメッセージ分散率で, それぞれ3回実行した. プロトタイプを 2.5 分間実行し, デバイスから 5 ミリ秒間隔でローカルブローカにメッセージを送信した. その間に 1 秒間隔でメッセージ送受信の遅延時間と CPU 使用率を測定し, その平均値を算出した(図 5, 図 6).

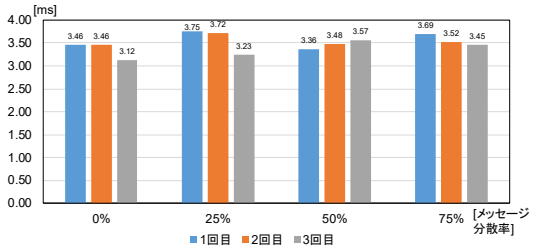

図 5 遅延時間

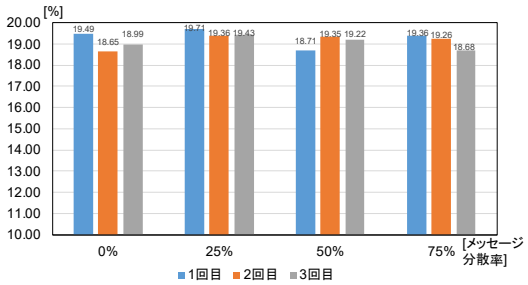

図 6 ローカルブローカの CPU 使用率

遅延時間の平均値が最小となったのはメッセージ分散率が 0%のときで 3.12 ミリ秒であり, 最大でもメッセージ分散率が 25%のときに 3.75 ミリ秒である. その差は 0.63 ミリ秒と大きな変化はない. CPU 使用率も同様に最小値が 18.65%, 最大値が 19.71%とエッジにかかる負荷に差はない. メッセージ分散率の増加は遅延時間と CPU 使用率への影響は少なかった. 以上から, 本稿で実装したプロトタイプでは遅延時間および CPU 使用率とメッセージ分散率に相関関係は見られなかった. しかしブリッジのプロセスの増加はブローカへの大きな負荷ならないことを確認した.

6. 考察

本稿で提案した二重エッジアーキテクチャと先行研究の分散連携プロキシ[4]と比較する. 先行研究はブローカの実装に依存しない連携を可能としたが, 遅延時間の増加が問題となった. 本稿で提案したアーキテクチャはエッジ間でのブローカ連携を遅延時間が増加することなく, またブローカへ負荷をかけずにメッセージ配信処理の分散を実現した. コネクテッドカーのシステムにはリアルタイム性が求められることから, 提案アーキテクチャが有用であると言える.

まとめ

二層のエッジにメッセージ配信処理を分散して負荷を軽減する二重エッジアーキテクチャを提案した. プロトタイプを実装し, ローカルブローカにおけるメッセージ送受信の遅延時間と CPU 使用率の測定によりスケーラビリティを評価することで提案アーキテクチャの有用性を示した.

参考文献

[1] AECC, General Principle and Vision, V. 2.0.0, Apr. 2018, https://aecc.org/wp-content/uploads/2018/02/AECC_White_Paper.pdf.
[2] Eclipse Mosquitto, https://mosquitto.org.
[3] ISO/IEC 20922:2016, Information Technology – Message Queuing Telemetry Transport(MQTT) V.3.1.1, 2016.
[4] 坂野 遼平 他, スケーラブルな MQTT ブローカの実現に向けた分散連携プロキシの提案, DICOMO2016 論文集, 情報処理学会, Jul. 2016, pp. 541-547.
[5] W. Shi, et al., The Promise of Edge Computing, IEEE Computer, Vol. 49, No. 5, May 2016, pp. 78-81.
[6] 濱野 真伍 他, IoT システムのためのエッジアーキテクチャ設計方法論の提案と評価, 第 198 回ソフトウェア工学研究会, 情報処理学会, Mar. 2018, pp. 1-8.