データ遅延に着目したエッジコンピューティングアーキテクチャの提案と評価

2015SE029 伊藤 亮太 2015SE037 河合 直也 2015SE075 高橋 哲也

指導教員 青山 幹雄

1. 研究背景と目的

IoT が社会に浸透しつつある. しかし, 常に最新のデータを必要とするシステムでは, データ発生順で送信すると重要なデータの処理に遅延が発生するケースがある. そのため, 重要なイベントには優先的にクラウドへ配信する方法が必要である. 本研究ではエッジコンピューティングアーキテクチャと NoSQL データベースを用いた方法を提案する.

2. 研究課題

本研究課題を以下に示す.

(1) センサ/デバイス層からエッジ層へ配信されたメッセージに優先度を付与し, それに従う順位付け方法の提案.

(2) 優先順位に従って高優先イベントをクラウド層へ優先的に配信するアーキテクチャの設計方法の提案.

(3) 提案アーキテクチャが高負荷時においてもリアルタイム配信が可能かプロトタイプを用いて検証.

3. 関連研究

3.1. エッジコンピューティングアーキテクチャ

エッジコンピューティングアーキテクチャは, センサ/デバイス層, エッジ層, クラウド層の 3 層から構成される[4].

3.2. Publish/Subscribe アーキテクチャ

Publish/Subscribe アーキテクチャとは, パブリッシャ, サブスクライバ, ブローカから構成される非同期メッセージ配信アーキテクチャである. 空間と時間の点から分離が可能であり, システムの拡張が可能である[3].

3.3. Redis

Redis とは, オープンソフトウェアの Key Value 型データベースである[3]. KVS が持つ特徴以外の主な特徴として以下の 2 点があげられる.

(1) Publish/Subscribe アーキテクチャ

Publish/Subscribe アーキテクチャに基づいた非同期メッセージ配信が可能である.

(2) リアルタイムランキングの実装

ランキングを識別する Key に, Value(Member)と Score(順位を判断するための値)を付与可能である.

4. アプローチ

多種多様のセンサから発生するイベントには優先的にクラウドへ配信するべきイベントが存在すると考え, イベントのデータを基に生成されるメッセージの優先度に着目した. 生成されるメッセージの優先度は Redis の Score 付与機能を用いることでランキング表現が可能であると考えた. 適切な優先度の指標に基づき生成されるメッセージに優先度を付与し, Redis を用いて重要なメッセージを優先的に配信可能なエッジコンピューティングアーキテクチャの設計方法を提案する.

5. 提案アーキテクチャ

5.1. アーキテクチャの設計方法

提案アーキテクチャの構成を図 1 に示す.

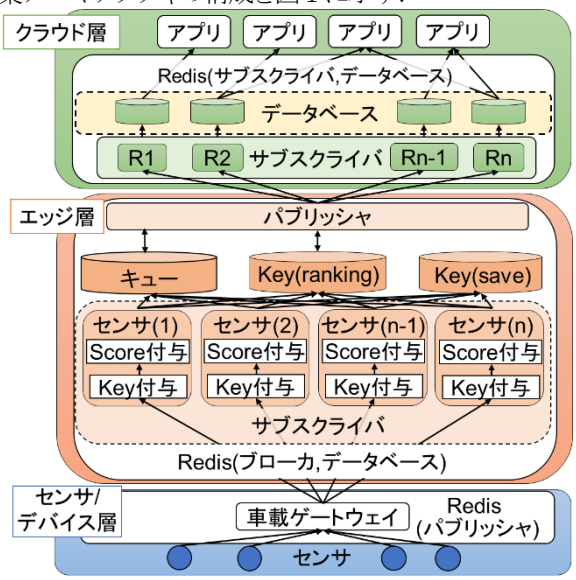

図 1 提案アーキテクチャの構成

(1) センサ/デバイス層

1) センサ

対象の検知及び相互間の距離を測定し, 測定したデータを車載ゲートウェイへ送信する.

2) Redis(パブリッシャ)

送信されたデータを基にメッセージを生成し, Redis Broker へメッセージをパブリッシュする.

(2) エッジ層

1) Redis(サブスクライバ)

センサ/デバイス層からセンサ毎にメッセージを受信する.

2) Redis(優先順位決め)

パブリッシュされたメッセージを Member に格納し，Key を付与する．センサ ID を参照して対応する値(1~n)を Score に格納し，付与する．優先度の高いメッセージには Score の値を低く設定し，それぞれのメッセージを R1~Rn とする．

3) Redis(データベース)
優先度が付与されたメッセージを格納する．

4) Redis(キュー)
格納通知メッセージをエンキュー，デキューする．

5) Redis(パブリッシャ)
データベースからソートされた先頭のメッセージを取得し，クラウド層へ優先度毎に配信する．

(3) クラウド層

1) Redis(サブスクライバ)
順位分けされたイベントをそれぞれ受信するサブスクライバを用意する．

2) Redis(データベース)
Redis Broker から配信されたメッセージを格納する．

3) アプリケーション
優先的に配信されたメッセージを利用するクラウド上のアプリケーションである．

5.2. 振る舞い定義

Redis Broker でのメッセージ受信からメッセージ配信の流れを図 2 に示す．

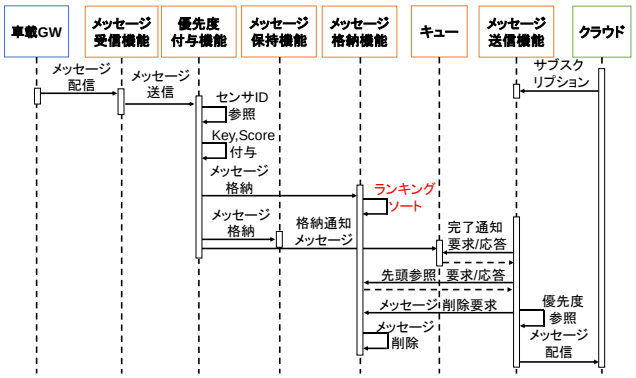

図 2 Redis Broker のシーケンス図

(1) クラウドは，Redis Broker のメッセージ送信機能へサブスクリプションを配信する．

(2) メッセージ受信機能は，優先度付与機能へ車載ゲートウェイから受信したメッセージを配信する．

(3) 優先度付与機能は，メッセージ受信機能から配信されたメッセージに付与されているデータを解析し，センサ ID を取得する．

(4) 優先度付与機能はセンサ ID 取得後，メッセージ受信機能から配信されたメッセージへ，Key と Score を付与する．

(5) 優先度付与機能は，メッセージ保持機能とメッセージ格納機能へ Key と Score を付与したメッセージを配信する．

(6) 優先度付与機能はメッセージを格納後，格納通知メッセージをエンキューする．

(7) メッセージ格納機能は，配信されたメッセージを受信し格納，メッセージを Score の値の昇順ソートする．

(8) メッセージ送信機能は，キューに格納通知メッセージを要求する．

(9) メッセージ送信機能は，格納通知メッセージをデキューする．

(10) メッセージ送信機能はキューから格納通知メッセージを取得後，メッセージ格納機能にソートされた先頭のメッセージを要求する．

(11) メッセージ格納機能は，メッセージ送信機能が要求するメッセージを配信する．

(12) メッセージ送信機能は，配信されたメッセージを受信し，メッセージ格納機能へ取得したメッセージの削除要求をする．

(13) メッセージ送信機能は，削除要求されたメッセージを削除する．

(14) メッセージ送信機能は，(12)で受信したサブスクライブ条件を参照する．

(15) メッセージ送信機能は，優先度毎にメッセージをクラウドへ配信する．

5.3. 優先度定義

本研究では，イベントを Rn(n は自然数)とする．n の値が大きいほど優先度が高いイベントとする．扱うセンサは前方距離測定センサ，ブレーキセンサ，ステアリングセンサ，アクセルセンサとし，発生するイベントの優先度を表 1 に示す．

表 1 イベントへの優先度定義

センサ名	優先度	理由
前方距離測定センサ	R4	衝突の危険性を測定する必要があるため
ブレーキセンサ	R3	衝突回避の最も有効な手段は減速することであるため
ステアリングセンサ	R2	衝突回避の有効な手段は進路を変更することであるため
アクセルセンサ	R1	むやみなスロットル制御は危険に繋がる可能性があるため

Redis では，同一 Key に Score を付与した Member を Score の値でソートが可能である．Key にはランキング名を設定する．付与する Score の値は整数を用い，下限を 1，上限を 100 とする．付与する値はセンサ ID を参照し付与する．Score の値は，同一センサでの Score の競合を防ぐために値は一定の範囲を保持する．Member にはそれぞれの発生時刻，測定値を設定する．

Key，Score，Member の関係を図 3 に示す．

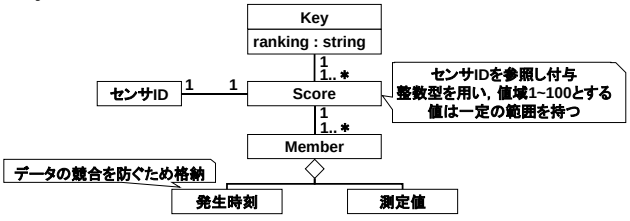

図 3 Key，Score，Member の関係

5.4. 優先度付与構造

Redis は Key に対して Member を Score の値でソートが可能である．優先度付与構造を図 4 に示す．

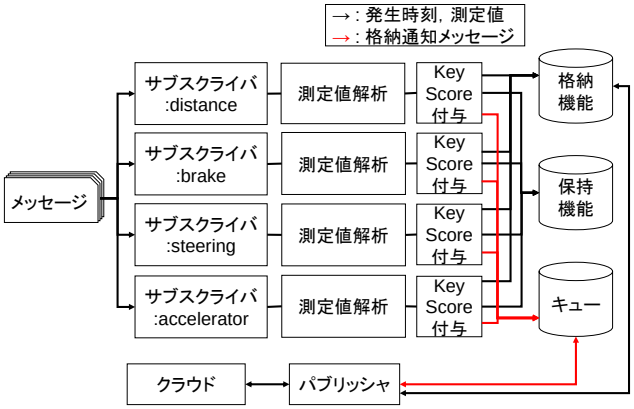

図 4　優先度付与構造

6. プロトタイプの実装

6.1. 前提条件

提案アーキテクチャの設計においての前提条件を示す．

(1) 自車のみ走行中で，直進のみをする．
(2) 前方に障害物が存在し，衝突の危険性がある．
(3) イベントを発生させるセンサは前方距離測定センサ，ブレーキセンサ，ステアリングセンサ，アクセルセンサの 4 つとする．
(4) 測定値は整数とし，乱数を用いて決定する．
(5) イベントへの優先度は決定済みである．
(6) メッセージ配信には Redis の Publish/Subscribe アーキテクチャを利用する．

6.2. 実装環境

プロトタイプの実装環境を表 2 に示す．

表 2　実装環境

階層名	センサ/デバイス層	エッジ層	クラウド層
ハードウェア	Raspberry Pi 3 Model B	Raspberry Pi 3 Model B	ESPRIMO WD2/W
OS	Raspbian 8.0	Raspbian 8.0	Ubuntu 16.04
プロセッサ	ARMv7 Processor rev4	ARMv7 Processor rev4	Intel Core i7-6000
メモリ	1GB	1GB	16GB
ストレージ	32GB micro SD カード	32GB micro SD カード	1TB HDD
実装言語	Python3.4.2	Python 3.5.3	Python3.6.5
データベース	Redis 4.0.10	Redis 4.0.10	Redis 4.0.10

6.3. プロトタイプの構成

プロトタイプを提案アーキテクチャに基づいて実装する．プロトイプの構成を図 5 に示す．

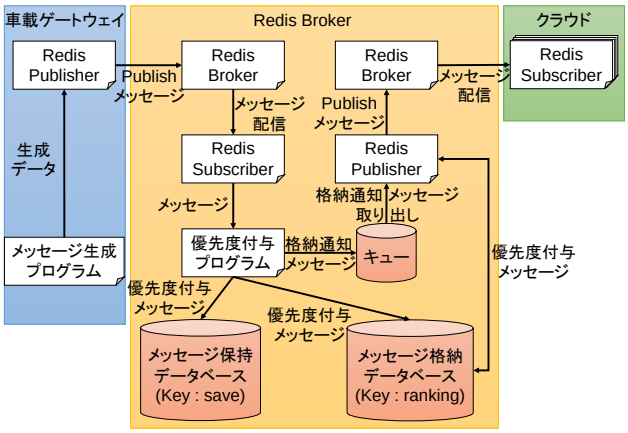

図 5　プロトタイプの構成

6.4. プロトタイプの実装と実行

プロトタイプでは Python を実装言語として用い，通信方法として Redis の Publish/Subscribe アーキテクチャを利用し，Redis Broker 内では受信層と配信層を分けて負荷分散をする．

プロトタイプを実行する上で，プロトタイプの構成に基づきメッセージ生成プログラムで生成されるメッセージの配信間隔を，高負荷時を想定した 1msec と 2msec，通常負荷時を想定した 10msec と 20msec の 4 パターンで配信する．生成されたメッセージの Redis Broker 内での処理時間を計測し比較検証をする．

7. プロトタイプの適用と実行結果

7.1. プロトタイプの適用事例

プロトタイプを自動ブレーキシステムに適用し，以下の 4 項目の検証をする．

(1) メッセージの送受信を確認
(2) メッセージへの優先度付与の確認
(3) メッセージをデータベースに格納し，昇順ソートされているかの確認
(4) 優先度の高いメッセージ取得から送信までのリアルタイム性の確認

7.2. プロトタイプのシナリオ

本研究におけるプロトタイプの実行シナリオを図 6 に示す．

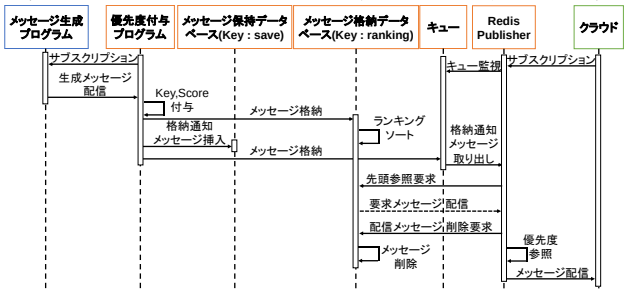

図 6　プロトタイプの振る舞い

プロトタイプは車載システムからメッセージ配信間隔を 4 パターンで Redis Broker へ配信する．また，Redis Broker は配信されたメッセージの受信と優先度付与をし，優先度が高いメッセージを優先的にクラウドへ配信する．

7.3. 実行結果

プロトタイプを実行し，4 パターンの間隔で各センサ 500 件ずつメッセージを配信した時の優先度毎での Redis Broker 内の処理時間を計測し比較した．それぞれ 3 回計測し，1 回目を a，2 回目を b，3 回目を c とした．

1msec の Redis Broker 内処理時間のグラフを図 7 に示す．

高優先度メッセージ(以降 R4)は低遅延で配信された．

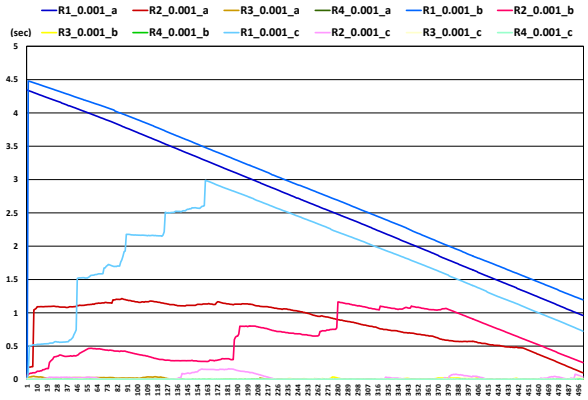

図 7　1msec 間隔メッセージ配信

R4 を 4 パターンの間隔で配信し，それぞれの 1 回目，2 回目，3 回目の Redis Broker 内の処理時間を比較したグラフを図 8 に示す．

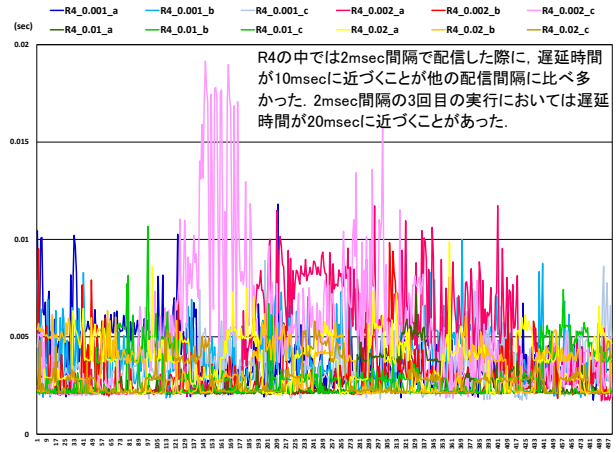

図 8　高優先度メッセージ配信

Redis Broker 内での平均遅延時間を図 9 に示す．

R4 の遅延時間は高負荷時でも通常負荷時から増大しないことがわかった．

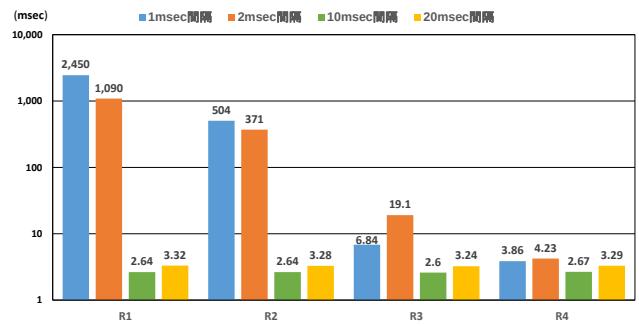

図 9　平均遅延時間

8. 提案アーキテクチャの評価

(1) メッセージへの優先度付与

センサ毎生成されたメッセージに優先度付与をすることで，センサ ID が異なるデータに優先順位を示すことが可能になった．これにより，クラウドが要求するデータを優先的に配信することが可能となった．

(2) メッセージのリアルタイム配信及びメッセージの共有

センサ/デバイス層で発生するイベント間隔は発生データ間隔高負荷時を想定した 1msec と 2msec，発生データ間隔通常負荷時を想定した 10msec と 20msec とし，処理時間の比較をした結果，図 9 で示すように優先度の高いメッセージは高負荷時でも通常負荷時から遅延時間が増大せず，リアルタイム性の保証が確認できた．

9. 提案アーキテクチャの考察

センサ/デバイス層，エッジ層，クラウド層の各層でメッセージ共有が可能となった．また，優先度の高いメッセージはイベント発生間隔が高負荷時でも通常負荷時から遅延時間が増大しなかった．

以上より，重要なイベントを優先的にクラウドへ配信する方法として有用である．

10. 今後の課題

(1) 異常値の制御

メッセージ処理時間の計測値から異常値が確認されたため，システム要求を満たさない場合がある．

(2) サブスクライバの数の増減に対応

Redis Broker へメッセージ配信をする際，センサ ID とサブスクライバが 1 対 1 であるため，多種類のセンサに対応できるように 1 つのサブスクライバで対応するセンサ数を改善する必要がある．

11. まとめ

本稿では，オープンソースソフトウェアの NoSQL データベースである Redis を用いてクラウド層へメッセージ配信をするアーキテクチャを提案した．提案アーキテクチャでは，優先度付与をする Redis Broker を設置することで，優先度の高いメッセージを優先的に配信し共有が可能となった．さらに，優先度の高いメッセージはリアルタイム配信が可能となった．

以上より，重要なイベントを優先的にクラウドへ配信する方法として有用である．

参考文献

[1] J. L. Carlson, Redis 入門 インメモリ KVS による高速データ管理, KADOKAWA, 2013.

[2] 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, No. 12, Mar. 2018, pp. 1-8.

[3] 井ノ口 仁也, 他, MQTT を用いた車載エッジコンピューティングアーキテクチャの提案と評価, 情報処理学会第 79 回全国大会, Mar. 2017, pp. 229-230.

[4] 緒方 研一, 他, データベース徹底攻略, 技術評論社, 2014.

[5] W. Shi and S. Dustdar, The Promise of Edge Computing, IEEE Computer, Vol. 49, No. 5, May 2016, pp. 78-81.