スケーラブルなエッジ指向 IoT アーキテクチャの提案と評価

濱野 真伍[†1] 青山 幹雄[†2]

概要：Publish/Subscribe とエッジを統合したスケーラブルな IoT アーキテクチャな未確立である．本稿ではスケーラビリティとリアルタイム性の確保のために，エッジを階層化するアブストラクションと，2 つのスケールアウトするフェデレーションの 2 つの方法を用いたアーキテクチャ提案する．さらに，デバイスからクラウドへのメッセージングには非同期配信の MQTT と同期配信の REST を併用し，クラウドにおけるメッセージの完全性を確保する．提案アーキテクチャのプロトライプを作成し，スケールアウト前後を比較するための 6 つのシナリオを適用し，CPU 使用率とデータの遅延を計測し，提案アーキテクチャの有効性を示す．

キーワード：IoT，エッジコンピューティング，MQTT，スケーラビリティ，ソフトウェアアーキテクチャ，Publish/Subscribe アーキテクチャ

A Scalable Edge-Oriented IoT Architecture and its Evaluation

SHINGO HAMANO[†1] MIKIO AOYAMA[†2]

1. はじめに

IoT(Internet of Things)システムは大量の多種多様なデバイスがインターネットに接続され，タイムクリティカルなデータを活用する．したがって，IoT アーキテクチャではスケーラビリティとリアルタイム性が求められる．多数のデータを非同期に収集するために，Publish/Subscribe(以下 Pub/Sub)アーキテクチャモデルに基づく MQTT[7]が適用されているが，ブローカに負荷が集中するためスケーラブルな拡張は行われていない．一方，リアルタイム性向上などのためにクラウドコンピューティングをネットワークのエッジへ拡張するエッジコンピューティング(以下エッジ)が提案されている．しかし，Pub/Sub とエッジを統合した，スケーラブルなアーキテクチャは未確立である．

本稿では，メッセージ配信に非同期配信と同期配信[10]を統合したエッジを階層化するアブストラクションと，スケールアウトを行うフェデレーションによって，スケーラビリティとリアルタイム性を保証と，デバイスのモビリティをサポートするアーキテクチャを提案する．提案アーキテクチャのプロトタイプを作成し，アブストラクションやフェデレーションを実装した 6 つのシナリオを実行する．それぞれのシナリオで CPU 使用率とメッセージ遅延を計測し比較することで提案アーキテクチャを評価する．また，既存の MQTT やエッジコンピューティングを用いたシステムと比較し，提案アーキテクチャの有用性を示す．

2. 研究課題

本研究では以下の 3 点を研究課題(RQ)とする．

(1) **RQ1:** スケーラビリティとリアルタイム性を保証する，エッジ指向 IoT アーキテクチャの提案．

(2) **RQ2:** クラウドにおいて，デバイスからのメッセージ受信の完全性を保証するアーキテクチャの提案

(3) **RQ3:** (1),(2)のアーキテクチャにおける，デバイスからクラウド及びエッジ間のメッセージング．

3. 関連研究

3.1 エッジ指向 IoT アーキテクチャ

IoT におけるエッジ[3,11]はクラウドをネットワークの端へ拡張し，デバイスにより近い位置でデータを処理することで，データ発生源とクラウド間の遅延，全データの一括処理による非効率性などの課題を解決する．

3.2 MQTT

MQTT[8]は ISO/IEC20922 で標準化されたトピックベースの Pub/Sub アーキテクチャ[12]を取るメッセージプロトコルである．本研究では，メッセージ受信者 Subscriber，メッセージ送信者を Publisher，メッセージ配信要求を Subscription，Publisher がブローカへのメッセージ配信を Publish，Subscriber がブローカからのメッセージ受信を Subscribe と定義する．

3.3 QEST ブローカ

QEST ブローカ[5]は MQTT ブローカと REST サーバを統合した非同期配信と同期配信が併用可能なメッセージングブローカである．Radis を用い，各トピックの最新の値のみを保存する．デバイスは Publish を利用しデータを送信する．1 つの Radis に対して複数の QEST ブローカを並列に動作し，スケーラビリティを確保する．

3.4 GaaS(Gateway as a Service)

GaaS(Gateway as a Service)[13]は IoT のためのクラウドコンピューティングフレームワークである．ゲートウェイに REST サーバとデータベースを組み込み，データのフィルタリングなどを行う．メッセージ配信には同期配信のみを用いる．Morabito[9]らは GaaS のフレームワーク上で Web サーバ，データベース，オーケストレイタを仮想化した．エンドユーザは REST API を用いて，Docker イメージを受信する．

3.5 Pulga

Pulga[8]は組み込みデバイスのための軽量な MQTT ブローカ

†1 南山大学大学院 理工学研究科 ソフトウェア工学専攻
Graduate Program of Software Engineering, Nanzan University
†2 南山大学 理工学部 ソフトウェア工学科
Dep. of Software Engineering, Nanzan University

である．デバイスとクライアントが直接接続され非同期配信でのみメッセージの配信を行う．はクライアントの接続を管理することができる．Pulga は MQTT のトピックの階層構造を用いることはできない．

4. アプローチ:アーキテクチャコンセプト

4.1 前提条件

本アーキテクチャは以下のように前提条件をおく．

(1) エッジはアーキテクチャの適用範囲内に点在する．
(2) エッジのネットワークの適用範囲をエリアネットワーク[2]とする．デバイスはエリアネットワーク内に存在する場合，そのエリアネットワークを定義するエッジに接続される．
(3) エッジとクラウドが扱うトピックを事前に定義可能とする．
(4) デバイスからクラウドへのメッセージ送信のみ対象とする．

また，デバイスの前提条件を以下のように定める．

(1) ユニークな ID が割り当てられ，識別可能とする．
(2) 計測データごとに事前にトピックが定義されている．
(3) 送信メッセージ量は MQTT の上限である最大 1KB とする．
(4) 移動しないデバイスと，移動可能なデバイスが混在する．

4.2 アーキテクチャコンセプト

図 1 に提案アーキテクチャのコンセプトを示す．提案アーキテクチャでは，RQ1 を解決するために，エッジ指向 IoT アーキテクチャに(1)アブストラクションと(2)フェデレーションによるスケールアウトの統合を提案する．また，RQ2 を解決するために非同期配信(MQTT)に同期配信(REST)を統合するアーキテクチャを提案する．

4.2.1 デバイスからクラウドのメッセージの完全性の保証

デバイスからクラウドに MQTT を介して大量のメッセージを配信する場合，デバイスはメッセージを保存せず，クラウドからデバイスへの受信確認ができないため，クラウドにおけるメッセージの完全性が保証されない場合がある．そこで，エッジに非同期配信のためのブローカと，(KVS)Key-Value Store を統合する．これによって，デバイスが生成したメッセージはエッジをにおいてクラウドに配信されると同時に，エッジの KVS に保存される．エッジはクラウドに配信するメッセージの全てを一定期間保存することでデバイスからのメッセージの受信の完全性を保証する．クラウドは一定期間ごとに，Subscription を送信した配信層に REST を用いてメッセージの問い合わせを行い，KVS に保存されているメッセージを受信する．これによりクラウドは，非同期配信と同期配信で受信したメッセージの比較によって，非同期配信で受信できなかったメッセージを受信することができる．これによって，クラウドはエッジの KVS に保存されている一定期間のメッセージの受信の完全性を保証する

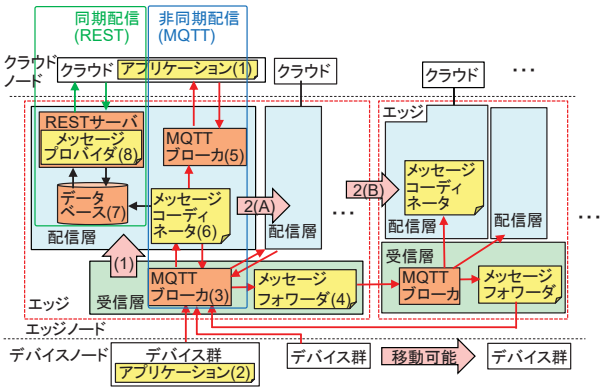

図 1 アーキテクチャモデル
Figure1 Architecture model

4.2.2 アブストラクション

エッジの機能にはデバイスからのメッセージの受信，エッジ間のメッセージの送受信，クラウドへのメッセージの配信がある．エッジに接続されるデバイスはエリアネットワーク内のデバイスに限定されるため，メッセージ受信の負荷も限定される．しかし，エッジに接続されるクラウドの数には制限が無いため，大量にメッセージを配信する場合，メッセージごとに配信先を参照することで負荷が増加し，スケーラビリティが確保できない．そこで図 1 の(1)に示すようにエッジを，負荷が限定されるデバイスからのメッセージ受信とエッジ間のメッセージの送受信を行う受信層と，クラウドへのメッセージ配信とメッセージの保存を行う配信層の 2 層に分離し抽象化する．1 つの配信層に対して，複数の受信層を接続することで，エリアネットワークを変更せず，配信層の負荷を分散する．以下に受信層，配信層の概要を示す．

(1) 受信層

エリアネットワークのデバイスからメッセージを受信する．他のエッジとメッセージの送受信を行う．受信したメッセージを配信層に転送する．クラウドと直接メッセージの送受信を行わない．

(2) 配信層

受信層からメッセージを受信する．メッセージを保存する．クラウドへメッセージを配信する．デバイスと直接メッセージの送受信を行わない．他の配信層へメッセージの送受信を行わない．

4.2.3 フェデレーション

エッジでのスケーラビリティを確保するために，本提案ではエッジ内のスケールアウトとエッジ間のスケールアウトの 2 つのスケールアウトを導入する．

(1) エッジ内のスケールアウト

配信層では KVS へのメッセージの保存とクラウドへのメッセージにより多くのコンピュータリソースを使用する．そのため，スケーラビリティを確保しながら大量のメッセージを処理するために，図 1 の 2(A)に示すように配信層のスケールアウトによってスケーラビリティを確保する．配信層とデバイスは分離されているため，デバイスへの影響はない．

(2) エッジ間のスケールアウト

デバイスが異なるエリアネットワーク間を移動する場合にも，KVS におけるメッセージの受信の完全性を確保するために，デバイスが生成したメッセージを対象のエッジに転送する必要がある．そこで，図 1 の 2(B)に示すようにエッジ間でメッセージの送受信を行う場合，負荷が制限され，デバイスからのメッセージを受信する受信層間でメッセージを転送する．クラウドと受信層は分離されているため，クラウドへの影響はない．

5. 提案アーキテクチャ

5.1 アーキテクチャモデル

図 1 にアーキテクチャのモデルを示す. 提案アーキテクチャでは受信層と配信層をまとめてエッジとする. 受信層, 配信層はハードウェアとしての実体を持つ. それぞれのコンポーネントとアプリケーションの詳細を以下に示す.

(1) クラウドアプリケーション

配信層からメッセージを非同期配信と同期配信を併用して受信する. 非同期配信でメッセージを Subscribe する場合はトピックを指定した Subscription を MQTT ブローカ(配信層)に送信する. 同期配信でメッセージを受信する場合は, トピックを必須項目とし, 計測値や計測日時などの範囲をオプションとして指定した GET メソッドを REST サーバに送信する.

(2) デバイスアプリケーション

事前に定義されたトピックと, 発生日時やセンサなどの計測した値を JSON にエンコードしたメッセージをエリアネットワーク内の MQTT ブローカ(受信層)に Publish する.

(3) MQTT ブローカ(受信層)

デバイスと他の受信層のメッセージフォワーダが Publish したメッセージを受信する. 受信したメッセージは, 同じエッジのメッセージコーディネータからの Subscription に従って配信する. メッセージコーディネータから Subscription されたトピック以外のメッセージを受信した場合は, メッセージフォワーダへ転送する.

(4) メッセージフォワーダ

MQTT ブローカ(受信層)からのメッセージを, そのメッセージのトピックを管理する配信層と同じエッジの MQTT ブローカ(受信層)に Publish する. Publish する際, メッセージの構造はデバイスアプリケーションで生成されたままとする.

(5) MQTT ブローカ(配信層)

クラウドからの Subscription を受信し, メッセージコーディネータが Publish したメッセージを, クラウドアプリケーションからの Subscription に従って, 非同期で配信する.

(6) メッセージコーディネータ

事前に定義されたトピックの Subscription を接続される MQTT ブローカ(受信層)に送信する. MQTT ブローカ(受信層)からメッセージを Subscribe した場合, MQTT ブローカ(配信層)にメッセージを Publish すると同時に, キーをトピック, バリューをコンテンツとし, データベースに格納する.

(7) データベース

Key-Value 型の NoSQL データベースを使用. メッセージコーディネータからのデータを格納する. キーをトピック, バリューをコンテンツとすることで様々な構造のデータに対応. コンテンツ内は階層構造のキーバリュー形式を持つ

(8) REST サーバ・メッセージプロバイダ

クラウドアプリケーションからの GET リクエストに従ったクエリを生成しデータベースを検索する. データベースからの結果は JSON 形式でクラウドアプリケーションに返却する. 受信する GET メソッドではトピックを必須項目とし, オプションによって値やデータの発生時刻によるフィルタリングが可能.

提案アーキテクチャにおいて, クラウドはエッジに Subscription やリクエストを送信し, デバイスもエッジに Publish するため, 互いに直接通信することがない. そのため, クラウドはデバイスのアドレスなどの情報を管理する必要がない.

5.2 アブストラクション

エッジ指向 IoT アーキテクチャに非同期配信のための MQTT ブローカと同期配信のための REST サーバを導入する場合, エッジが処理するメッセージ量の増加によって非同期配信に遅延が生じる場合がある. エッジの数を増やすスケールアウトを行う場合, それぞれのエッジが管理するトピックの再定義などによりエッジをブローカを一旦停止する必要があり, その間にデバイスが Publish したメッセージ受信することができないため, データベースへの登録とクラウドへの配信ができず, クラウドにおけるメッセージの完全性が保証できない. そこで, 図 1 中の(1)に示すようにエッジを配信層と受信層の 2 層に階層化し,「デバイスからのメッセージ受信」及び「エッジ間のメッセージの送受信」を行う受信層と,「クラウドへのメッセージ配信」及び「メッセージの保存」を行う配信層の 2 層に階層化する. それぞれの層の詳細を以下に示す.

(1) 受信層

エリアネットワークを定義し, そのエリア内のデバイスが Publish したメッセージを受信する. 配信層からの Subscription を受信する. デバイスからメッセージを受信した場合, 受信層からの Subscription に従って非同期配信でメッセージを受信層に配信する. 同じエッジの配信層から Subscription されていないトピックのメッセージを受信した場合, そのメッセージのトピックを Subscribe している配信層と同じエッジに存在する配信層にメッセージを Publish する. 他のエッジが Publish したメッセージを受信した場合にも同様にして配信層にメッセージを配信する. これによりデバイスとエッジ間やエッジ同士でのメッセージの送受信に関してクラウドや配信層への影響をなくすことができ, デバイスの位置や状態に関係なくクラウドはメッセージを受信することができる.

(2) 配信層

非同期配信と同期配信を併用してクラウドへメッセージを配信する. 非同期配信を用いる場合, クラウドから Subscription を受信し, その Subscription と同じトピックを指定した Subscription を受信層に送信する. 受信層から Subscribe したメッセージを配信層内のブローカを経由してクラウドに配信する. メッセージの完全性を保証するために, 受信層から Subscribe したメッセージを非同期配信でクラウドへ配信すると同時に, トピックをキー, コンテンツをバリューとしたキーバリュー形式でデータベースに保存する. クラウドからの GET リクエストに従ってデータベースを検索し, その結果返却する.

次に, 階層化されていないエッジをアブストラクションにより階層化する場合の手順を示す.

(1) 受信層の設定
既存エッジを受信層とし，クラウドアプリケーションへのメッセージの配信を停止する．これにより，エリアネットワークの範囲は変更されず，デバイスはアブストラクションによる接続先の変更は無い．

(2) 配信層の設置
新たに配信層を設置し，クラウドアプリケーションからの Subscription を受信する MQTT ブローカ，メッセージコーディネータ，データベース，REST サーバを導入する．

(3) メッセージコーディネータの設定
クラウドアプリケーションは配信層に Subscription を送信し，それに従ってメッセージコーディネータは受信層に Subscription を送信する．

図 2 に 1 つのエッジをアブストラクションにより 2 層に階層化されたエッジのメッセージングモデルを示す．

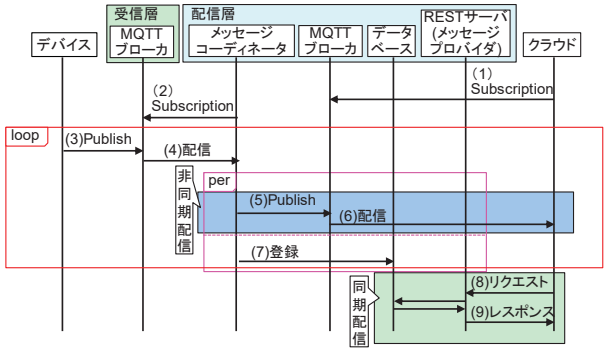

図 2 アブストラクション後メッセージングモデル
Figure2 Messaging Model after Abstraction

(1) クラウドアプリケーションは配信層の MQTT ブローカに事前に定義したトピックの Subscription を送信する．

(2) デバイス起動前に，メッセージコーディネータは事前に定義したトピックの Subscription を受信層の MQTT ブローカに送信する．

(3) デバイスはエリアネットワークネットワークの受信層の MQTT ブローカに計測したデータを Publish する．

(4) 受信層の MQTT ブローカはメッセージコーディネータからの Subscription に従ってメッセージを配信する．

(5) メッセージコーディネータは受信層の MQTT ブローカにメッセージを Publish する．

(6) 配信層の MQTT ブローカはクラウドアプリケーションからの Subscription に従ってメッセージを配信する．

(7) メッセージコーディネータはデータベースにデータを登録する．

(8) クラウドアプリケーションは配信層の REST サーバに対し GET メソッドを用いたリクエストを送信する．

(9) REST サーバはデータベースを検索し，クラウドアプリケーションにレスポンスを送る．

クラウドはメッセージを受信するために(1)，(5)，(6)に示す非同期配信と，(8)，(9)に示す同期配信を併用する．これによりクラウドは非同期配信を用いたリアルタイムなメッセージ受信が可能である一方で，MQTT の Retain 機能で保存されているメッセージ以前のメッセージのような非同期配信では受信できないメッセージを同期配信を用いて受信可能である．

5.3 フェデレーション

5.3.1 エッジ内のフェデレーション

アブストラクションによって階層化したエッジの配信層において，処理するメッセージの数や種類の増加などによりスケーラビリティが確保できない場合，図 1 中の 2(A)に示すように配信層の数を増やす「エッジ内のフェデレーション」を行う．以下の手順において，エッジ内フェデレーション以前に存在する配信層を配信層 A，新たに追加する配信層を配信層 B とする．

(1) アブストラクションによって階層化したエッジに配信層 B を導入する．

(2) 配信層 A におけるメッセージ処理を配信層 B に負荷分散する場合，配信層 A のメッセージコーディネータは一度受信層からの Subscribe を停止する．

(3) 受信層からのメッセージ受信は非同期配信であるため，トピックが階層構造であることを利用し，配信層 A が Subscribe していたメッセージのトピックを，具体化したトピックをそれぞれの配信層に再定義する．

(4) 配信層 A，B は，新たに定義したトピックの Subscription を受信層の MQTT ブローカに送信する．

エッジ内のフェデレーションで配信層の数を増やす場合，アブストラクションによって配信層とデバイスは分離しており，受信層の MQTT ブローカはサービスを停止しないため，デバイスはエッジ内のフェデレーションの影響を受けることなくメッセージを Publish し続けることができる．ただし，クラウドはトピックの再定義によりメッセージの一部が受信できなくなる場合，配信層 B に新たに Subscription を送信する．

5.3.2 エッジ間のフェデレーション

移動するデバイスに対応するために，図 1 の 2(B)に示すように複数のエッジを設置する．この際，クラウドが非同期配信を用いてメッセージを受信し，配信層が同期配信のためにメッセージを保存するために，デバイスが Publish したメッセージをクラウドが Subscription を送信したエッジへ転送する必要がある．

提案アーキテクチャではアブストラクションによって階層化したエッジの受信層がエリアネットワークを定義し，そのエリアネットワーク内のデバイスが Publish したメッセージを全て受信する．受信したメッセージのトピックが，同じエッジの配信層から受信した全ての Subscription のトピックと一致しない場合，そのトピックを管理する配信層が存在するエッジの受信層にメッセージを Publish する．クラウドは受信層，配信層はデバイスの影響を受けないため，デバイスが移動した場合にも配信層は Subscription を送信した受信層からメッセージを受信し続け，保存することができる．また，クラウドは非同期配信で Subscription を送信した配信層からメッセージを Subscribe し続けることができ，同期配信でも同じ配信層にリクエストを送信することでメッセ

ージを受信することができる．これにより，デバイスのモビリティをサポートする．

デバイスが移動する場合のメッセージングモデルを図 3 に示す．図 3 エッジ間フェデレーションによってスケールアウトし，アブストラクションによって階層化されたた 2 つのエッジ A，B が存在し，デバイス a がエッジ A のエリアネットワークからエッジ B のエリアネットワークに移動した場合，クラウドが非同期配信を用いてメッセージを受信するまでのメッセージングを示す．ただし，デバイス a が Publish するメッセージのトピックはエッジ A の配信層で管理されるとする．デバイスがエッジ A に接続されている場合は図 2 に示すメッセージングモデルと同様である．

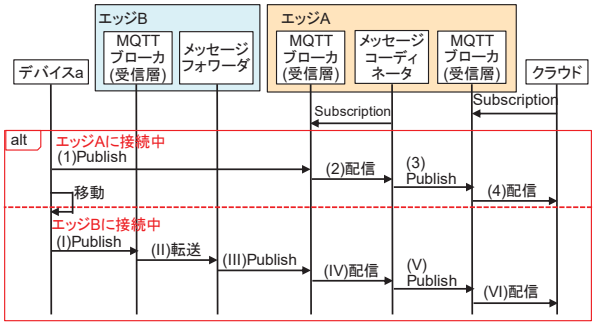

図 3 メッセージングモデル
Figure3 Messaging Model

(I) デバイスはエッジ B の MQTT ブローカ(受信層)にメッセージを Publish する．

(II) エッジ B の MQTT ブローカ(受信層)はメッセージがエッジ B の配信層から受信した Subscription のトピックではないのでメッセージフォワーダにメッセージを転送する．

(III) メッセージフォワーダは転送されたメッセージのトピックを管理するエッジ A の MQTT ブローカ(受信層)にメッセージを Publish する．

(IV) エッジ A の MQTT ブローカ(受信層)はメッセージフォワーダから受信したメッセージをデバイスから受信したメッセージと同様に扱い，メッセージコーディネータに配信する．メッセージコーディネータは同様にクラウドへのメッセージ配信と保存を行う．

6. プロトタイプの実装

提案アーキテクチャのスケーラビリティ，リアルタイム性，クラウドアプリケーションでのメッセージの受信を確認するためにプロトタイプを実装した．

6.1 実装環境

プロトタイプに使用したハードウェアコンポーネントを表 1 に，ソフトウェアコンポーネントを表 2 に示す．ただし，デバイスは Ubuntu OS 上に仮想的に生成するものとする．

表 1 ハードウェアコンポーネント
Table1 Hardware Components

クラウド，デバイス	
OS	Ubuntu 14.04
メモリ	2GB
CPU	Intel® CoreTM2 Duo CPU E7300 @ 2.66Ghz×2

エッジ	
デバイス	Raspberry Pi 2 model B+
OS	Raspbian 7.11
メモリ	512MB
ストレージ	microSD カード 8GB
CPU	ARM1176JZF-S @ 700MHz

表 2 ソフトウェアコンポーネント
Table2 Software Components

コンポーネント名	使用ソフトウェア	バージョン
MQTT ブローカ	Mosquitto	1.4.9
MQTT クライアント	Paho-MQTT	3.1
データベース	MongoDB	2.1.1
REST サーバ	Apache	2.2.22

6.2 アプリケーションの構築

プロトタイプに実装したアプリケーションを表 3 に示す．

表 3 実装アプリケーション
Table3 Imprement Application

アプリケーション名	実装言語	LOC
クラウドアプリケーション	Python 2.7	35
デバイスアプリケーション	Python 2.7	69
メッセージコーディネータ	Python 2.7	58
メッセージプロバイダ	PHP	40
メッセージフォワーダ	Python 2.7	130

```
def on_message(client, userdata, msg):
    #print(msg.topic + ' ' + msg.payload)
    lead=json.loads(msg.payload)
    value=lead['value']
    date=lead['date']
    print(msg.topic)
    #データベースに保存
    db.temp_test.update( { "key" : msg.topic } , { '$set' :
     { "value" : value , "date" : date } } , upsert=True)
    #Publish する
    Publisher.publish(msg.topic,msg.payload)
    #受信時刻 を追加してログ保存
    d = datetime.datetime.today()
    f.write ( msg.topic + ": value=" +str(value)+ "    date="+
    date + " receivetime=" + str(d) + "¥n")
```

図 4 メッセージコーディネータ
Figure4 Message Coordinator

図 4 に実装したメッセージコーディネータのうち，メッセージの受信，データベースへの登録，配信層の MQTT ブローカへ Publish を行う部分を示す．

7. 例題への適用

アブストラクションとフェデレーションによるスケーラビリティとリアルタイム性，クラウドアプリケーションにおけるデータの完全性の確保を確認するために 6 つのシナリオでテストを行った．シナリオの概要を図 5 と表 4 に示し，シナリオ 6 の詳細な構成を図 6 に示す．各シナリオは 3 回ずつ実行し，それぞれのエッジの配信層の CPU 使用率を 0.5 秒ごとに計測し，デバイスでデータが生成されてからクラウドで受信するまでの遅延時間を計測する．クラウドの動作は以下に定義する．

(1) デバイスアプリケーションの起動前に配信層の MQTT ブローカに対して Subscription を送信する．

(2) 各シナリオの実行後に非同期配信と同期配信を用いて受信したメッセージに差異がないことを確認する．

各シナリオにおいてデバイスアプリケーションは以下のように動作するものとする．

(1) デバイスはマシン上で仮想的に 20 個生成される.
(2) 計測するトピックは ”temperature/roomX” と ”humidity/roomX”(X は 0~9 の数字)の 20 通り.
(3) データは乱数を用いて 0.05 秒ごとに計測し, 各トピックの値を 20 個ずつ, 計 400 個計測する.
(4) 計測順序は”temperature/room0”, ”humidity/room0”, ”temperature/room1”, ”humidity/room1”とし room9 まで計測したら, room0 に戻る.

表 4 シナリオの詳細
Table4 Detail of Scenario

シナリオ	受信層数	配信層数	概要
1	-	-	アブストラクション, フェデレーションを共にせず, 1 つのエッジでメッセージの受信, 保存, クラウドへの配信
2	1	1	アブストラクションにより, 受信層と配信層に階層化
3	1	2	アブストラクションとエッジ内フェデレーションにより配信層数を増加.
4	2	1×2	エッジ間フェデレーションにより, エッジを 2 つに分離. それぞれのエッジ間でのメッセージングはない.
5	2	1×2	シナリオ 4 の構成で, エッジ間のメッセージングが発生するようにデバイスを変更.
6	2	2×2	アブストラクションと 2 つのフェデレーションを併用. エッジ間のメッセージングは発生する.

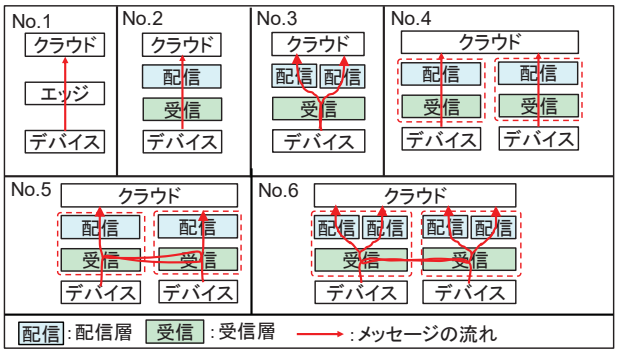

図 5 シナリオの概要
Figure5 Outline of Scinario

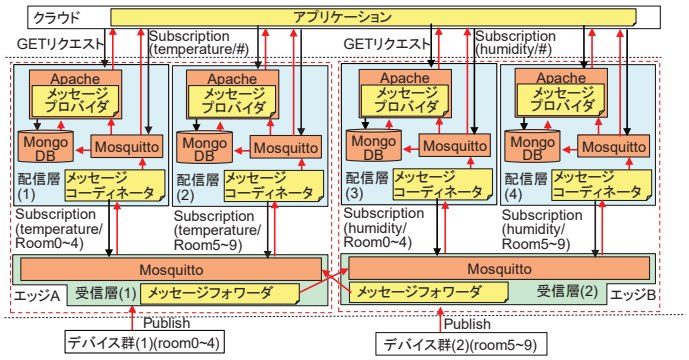

図 6 シナリオ 6
Figure6 Scinario6

8. 提案アーキテクチャの評価

8.1 メッセージの完全性の評価

シナリオ 1~6 において, デバイスで生成したデータ, クラウドアプリケーションが非同期配信で受信したデータ, 同期配信で受信したデータのログを取得した. クラウドアプリケーションがエッジから非同期配信の MQTT で受信したデータを図 7, 同期配信の REST で受信したデータ図 8 に示す.

図 7 及び図 8 とデバイスアプリケーションのログを比較したところ, 差異が無いため, デバイスが動的に変化する場合においても, クラウドは非同期配信で対象のメッセージを漏れなく受信することができ, 同じ配信層から同期配信でもメッセージの受信が可能であることを確認した. 全てのシナリオにおいてデバイスアプリケーションの起動後にクラウドアプリケーションを起動し Subscription を送信した場合にも, Subscription 以降のデータは MQTT を用いて受信し, Subscription 以前のデータはパラメータを指定し REST で受信可能であることを確認したためクラウドにおけるメッセージ受信の完全性を保証する.

```
temperature/room0/N080: value=-7.01  date=2017-05-05 15:54:07.14339
temperature/room1/N080: value=-6.41  date=2017-05-05 15:54:07.16382
temperature/room2/N080: value=-6.19  date=2017-05-05 15:54:07.18421
temperature/room3/N080: value=11.16  date=2017-05-05 15:54:07.20462
temperature/room4/N080: value=-7.32  date=2017-05-05 15:54:07.22502
temperature/room5/N080: value=5.42  date=2017-05-05 15:54:07.24545
temperature/room6/N080: value=20.18  date=2017-05-05 15:54:07.26586
temperature/room7/N080: value=29.64  date=2017-05-05 15:54:07.28626
temperature/room8/N080: value=20.83  date=2017-05-05 15:54:07.30667
temperature/room9/N080: value=3.39  date=2017-05-05 15:54:07.32707
temperature/room0/N081: value=-7.63  date=2017-05-05 15:54:07.34747
temperature/room1/N081: value=9.29  date=2017-05-05 15:54:07.36789
```

図 7 クラウドアプリケーションログ(MQTT)
Figure7 Log of Cloud Application(MQTT)

```
["date"]=>
string(25) "2017-05-05 15:54:07.16382"
["key"]=>
string(22) "temperature/room1/N080"
["value"]=>
float(-6.41)
["date"]=>
string(25) "2017-05-05 15:54:07.18421"
["key"]=>
string(22) "temperature/room2/N080"
["value"]=>
```

図 8 クラウドアプリケーションログ(REST)
Figure8 Log of Cloud Application(REST)

8.2 スケーラビリティの評価

(1) CPU 使用率

シナリオ 1~6 において, 配信層の CPU 使用率を計測し, その結果を図 9 に示す. 図 9 のグラフの縦軸各シナリオの 3 回のテストの平均 CPU 使用率, 横軸はデバイスアプリケーション起動からの経過時間を示す. また, 表 5 にグラフの線形近似式を示す.

400 件のデータをクラウドアプリケーションへの配信とデータベースへの登録が完了するまで CPU 使用率はほぼ 30%以上である. また, 線形近似式より時間とともに増加しているため, エッジでの処理がスタックされている. そのため, アブストラクションののみの実行では十分なスケーラビリティは確保できないことを

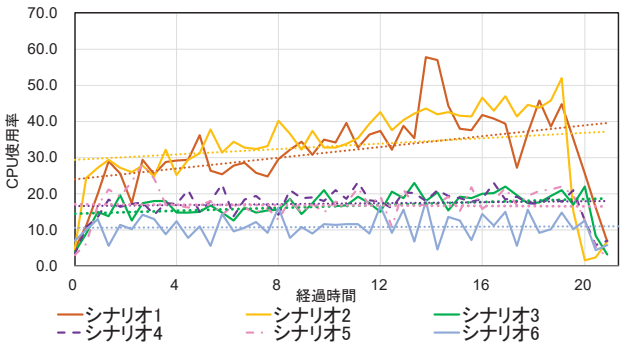

図 9 CPU 使用率の変化
Figure9 CPU Usage

表 5 CPU 使用率の線形近似
Table5 Linear Prediction of CPU Usage

シナリオ	線形近似式
シナリオ 1	y=0.33x+23.65
シナリオ 2	y=0.16x+29.20
シナリオ 3	y=0.09x+14.33
シナリオ 4	y=0.03x+16.47
シナリオ 5	y=-0.01x+17.16
シナリオ 6	y=0.01x+10.56

確認した. しかし, シナリオ 3 では 20%前後を推移し, 線形近似シナリオ 1, 2 でエッジがデバイスからのデータ受信開始から, による傾きがシナリオ 1,2 よりも穏やかになったため, エッジ内のフェデレーションによってそれぞれの配信層の負荷が軽減できていることを確認した. シナリオ 4,5 では 17%前後を推移し, 線形近似による傾きがほぼ 0 に等しいことから, エッジ間のフェデレーションにおいても負荷の軽減が可能であることが確認できた. シナリオ 6 では線形近似による傾きはほぼ 0 であり, CPU 使用率も 10%前後を推移しているため, これ以降においても, メッセージがエッジにスタックされることは無いと考えることができるため, アブストラクションとフェデレーションによって, 十分なスケーラビリティを確保することができたと言える.

(2) メッセージの遅延

シナリオ 2~6 においてデバイスがデータを生成してから, クラウドアプリケーションが非同期配信の MQTT を用いてデータを受信するまでの遅延を計測した. シナリオ 2~4 はメッセージフォワーダを用いない場合, シナリオ 5, 6 ではメッセージフォワーダから他方のエッジにメッセージが転送される場合を計測する. 計測したシナリオの流れを図 10 に, 計測結果を図 11 に示し, グラフでは各シナリオで行った 3 回のテストの平均遅延と最大遅延を示す. グラフの縦軸は 400 件目のデータがデバイスで生成されてから, 非同期配信を用いてクラウドアプリケーションで受信するまでの遅延時間を示し, 横軸はシナリオを示す.

アブストラクションのみを行ったシナリオ 2 では平均約 30 ミリ秒の遅延が生じていたが, エッジ内のフェデレーションを併用したシナリオ 3 では平均遅延, 最大遅延がともに. 10 ミリ秒以上短くなった. また, エッジ間フェデレーションを行ったシナリオ 4 でもシナリオ 3 とほぼ同じ結果が得られたため, メッセージの遅延時間は配信層が処理するメッセージ量の減少のよって短くなることが確認できた. エッジ間のメッセージングを行ったシナリオ 5

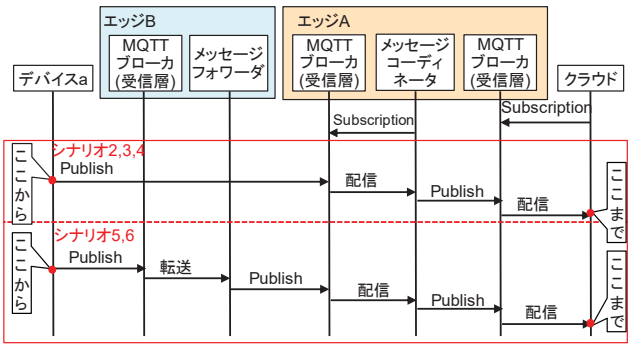

図 10 メッセージ遅延計測シナリオ
Figure10 Scinatio of Message Latency

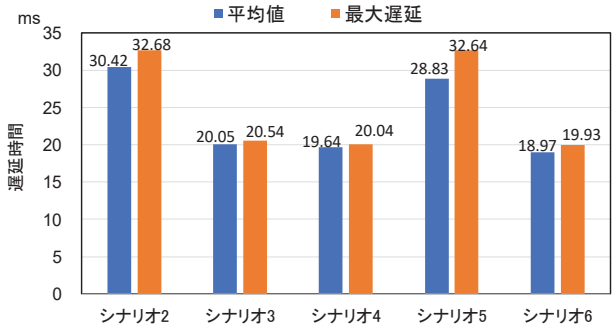

図 11 データの遅延
Figure11 Data Latency

では平均遅延が約 29 ミリ秒となってしまったため, エッジ間のメッセージングによりメッセージの遅延が発生し, クラウドでメッセージを受信するまでの遅延が大きくなってしまった. しかし, アブストラクションと 2 つのフェデレーションを併用したシナリオ 6 ではエッジ間のメッセージングに関わらず全シナリオ中で, 平均, 最大遅延ともに最小の値となり, アブストラクションと 2 つのフェデレーションを併用することで十分なリアルタイム性を確保できると言える.

(3) 提案アーキテクチャの評価

プロトタイプの実行により, アブストラクションと 2 つのフェデレーション を併用することで, スケーラビリティとリアルタイム性を十分に確保したアーキテクチャであることを確認した. よって RQ1 を解決していると言える. また, 提案アーキテクチャにおいて事前に定義したトピックではエッジ間のメッセージング及び, クラウドアプリケーションでのメッセージの完全性を保証することができるため, RQ2 を解決していると言える.

9. 考察:関連研究との比較

提案アーキテクチャと関連研究のシステムの性能を比較する. 以下の表において++は十分にその性能を持っている. +は性能を持っているが他方と比較して十分ではない. ー はその性能を持っていいないことを示す.

(1) QEST ブローカ

表 6 に QEST ブローカとの比較を示す. 表 6 より, スケーラビリティに関して, 提案アーキテクチャと比較して, QEST ブローカは 1 つの内部でしかスケールアウトできないため十分とは言えない. また, 過去データのアクセシビリティから提案アーキテク

表 6 QEST ブローカとの比較
Table6 Comparing with QEST Broker

視点	提案アーキテクチャ	QEST ブローカ
スケーラビリティ	++(アブストラクションとフェデレーションの併用により実現)	+(1 つの QEST ブローカ内部でのみスケールアウト可)
メッセージの完全性の保証	++(非同期配信と同期配信の統合により全データにアクセス可能)	+(各トピックの最新値以外は, 受信することができない)
データの操作性	ー(クラウドからはデータ操作不可)	++(クラウドから Radis のデータを書換え可)
リアルタイム性	++(非同期配信の MQTT とスケールアウトにより実現)	+(Radis からメッセージを受信するため, アクセス時間が増加する.)

チャはネットワーク障害などが発生した時でもクラウドにおいてデータの完全性を保証することができるが，QEST ブローカではトピックごとの最新のコンテンツのみを保存しているため，十分ではない．一方で，QEST ブローカでは PUT メソッドなどを用いてクラウドからでもデータの Publish や，データベースの操作ができる点は優れていると言える．

(2) GaaS

表 7 に GaaS との比較を示す．表 7 より，スケーラビリティに関して，GaaS はその能力や機能がゲートウェイに依存しているため，提案アーキテクチャと比較して十分に確保できない．しかし，コンポーネントの仮装化により，実装環境に依存しない点が利点としてあげられる．また，データ配信に関してクラウドは GaaS にサービスを組み込むことによって様々な形式での受信が可能であるが，非同期配信を用いないためリアルタイムな受信をすることはできない．

表 7 Gaas との比較
Table7 Comparing with GaaS

視点	提案アーキテクチャ	GaaS
スケーラビリティ	++(十分に可能)	+(ゲートウェイの能力に依存)
リアルタイム性	++(非同期配信の MQTT により実現)	－(同期配信を用いてメッセージを受信するため)
コンポーネントの仮想化	－(各コンポーネントはエッジに組み込まれなければならない)	++(Docker の上で仮装化可能)
データのフィルタリング	+(同期配信ではトピックと値の指定が可能)	++(ゲートウェイに様々なサービスを組み込み，データの処理が可能)

(3) Pulga

表 8 に Pulga ブローカとの比較を示す．表 8 より，Pulga はデバイス組み込み型の MQTT ブローカであるため，デバイスのモビリティは十分に確保することができるが，スケーラビリティを確保することはできない．また，内部にデータを保存する領域を持たないため，クラウドなどのクライアントは，通信途中で失われたデータを受信することができず，メッセージの完全性は保証できない．．一方で，Pulga はデバイスに組み込まれるために十分なモビリティがあるが，提案アーキテクチャはエリアネットワーク内という制限がある．

表 8 Pulga の比較
Table8 Comparing with QEST Pulga

視点	提案アーキテクチャ	Pulga
スケーラビリティ	++(十分に可能)	－(デバイスに組み込むブローカであるため不可)
メッセージの完全性	++(非同期配信と同期配信を統合し実現)	－(データは一切保存しないため消失したデータは受信不可)
デバイスのモビリティ	+(エッジ間フェデレーションにより実現)	++(デバイスに組み込むため十分に可能)

関連研究との比較より，提案アーキテクチャは IoT アーキテクチャの求められるスケーラビリティとリアルタイム性を十分に持っていると言える．

10. 今後の課題

今後，下記の 3 点を研究する必要がある．

(1) エッジの構成の動的な変更の方法
スケールアウト時にエッジの設定や構成を手動で変更しているため，接続されるデバイスやクラウドアプリケーションに応じた動的な構成の変更の方法の検討．

(2) エッジ間フェデレーションにおけるメッセージ配信方法
エッジが管理するトピックを事前に定義せず，配信層が管理するトピックが動的に変化する場合に，メッセージフォワーダが対象トピックを管理するエッジを発見する方法の検討．

(3) 非同期配信のコンテンツによるフィルタリング
非同期配信の MQTT を用いた場合にコンテンツベースにフィルタリングができないので，その方法の検討．

11. まとめ

IoT システムのスケーラビリティとリアルタイム性を確保するために，アブストラクションと 2 つのフェデレーションによるスケールアウトを用い，クラウドにおけるメッセージ受信の完全性を保証するために非同期配信と同期配信を統合するアーキテクチャを提案した．提案アーキテクチャのプロトタイプを作成しスケールアウト前後の比較を行うために 6 つのシナリオを実行した．それぞれのシナリオで CPU 使用率とクラウドにおけるメッセージの遅延を計測した結果，提案アーキテクチャは十分なスケーラビリティとリアルタイム性があることが確認できた．また，既存システムと性能を比較し提案アーキテクチャの有用性を示した．

本研究は，エッジを用いた共通アーキテクチャを提案し，そのスケーラビリティやリアルタイム性を十分に示すことができた点から，意義があると考えている．

参考文献

[1] A. Al-Fuqaha, et al., Internet of Things, Proc. of CST ’15, IEEE, Jun. 2015. pp. 2347-2376.
[2] ETSI, Machine-to-Machine Communications (M2M), ETSI TR 102 966 V1.1.1, Feb. 2014.
[3] F. Bonomi, et al., Fog Computing and Its Role in the Internet of Things, Proc. of MCC ’12, ACM, Aug 2012 pp. 13-16.
[4] R. Buyya, et al. (eds.), Internet of Things, Morgan Kaufmann, 2016.
[5] M. Collina, et al., Introducing the QEST Broker: Scaling the IoT by Bridging MQTT and REST, Proc. of PIMRC ’12, IEEE, Sep. 2012, pp. 36-41.
[6] 濱野 真伍 ほか, MQTT と REST を用いたエッジ指向 IoT アーキテクチャの評価，ウィンターワークショップ 2017・イン・飛騨高山，情報処理学会, Feb. 2017, pp. 61-62.
[7] ISO/IEC 20922:2016, Information Technology - Message Queuing Telemetry Transport (MQTT) V. 3.1.1, 2016.
[8] J. Luis, et al., Pulga, A Tiny Open-Source MQTT Broker for Flexible and Secure IoT Deployments, Proc. of CNS ’15, IEEE, Sep. 2015. pp. 690-694.
[9] R. Morabito, et al., Enabling a Lightweight Edge Gateway-as-a-Service for the Internet of Things, Proc. of NOF ’16, IEEE, Nov. 2016, 5 pages.
[10] L. Richardson, et al., RESTful Web Services, O’Reilly, 2007.
[11] W. Shi and S. Dustdar, The Promise of Edge Computing, IEEE Computer, Vol. 49, No. 5, May. 2016, pp. 78-81.
[12] S. Tarkoma, Publish/Subscribe Systems, Wiley, 2012.
[13] Z. Wu, et al., Gateway as a Service: A Cloud Computing Framework for Web of Things, Proc. of ICT ’12, IEEE, Jun. 2012, 6 pages.