7J-02

IoTシステムのためのスケーラブルエッジアーキテクチャの提案と評価

濱野 真伍† 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† 南山大学 理工学部 ソフトウェア工学科‡

1. 研究の背景と課題

IoT(Internet of Things)[1]システムでは大量メッセージを処理するためにゲートウェイとして，エッジコンピューティング(以下エッジ)が導入されている．本稿ではエッジに Publish/Subscribe(以下 Pub/Sub)アーキテクチャを統合し，階層化による機能分散とスケールアウトを可能にするエッジアーキテクチャを提案する．プロトタイプを実装し，その有効性を評価する．

2. 関連研究

(1) エッジコンピューティング

エッジ[4]はクラウドをネットワークのエッジへ拡張し，データ処理を行うアーキテクチャである．

(2) Publish/Subscribe アーキテクチャ

ブローカを介して非同期でメッセージを配信するアーキテクチャである[5]．IoT 向けの実装にトピックベースでメッセージを配信する MQTT[3]がある．

(3) QEST ブローカ[2]

IoT のゲートウェイに MQTT ブローカと REST サーバを統合したメッセージングブローカである．

3. 前提条件

本稿では以下の 3 点を前提条件とする．

(1) クラウド，エッジ，デバイスの 3 層アーキテクチャ．

(2) デバイスからクラウドへのメッセージ配信を対象．

(3) Pub/Sub のプロトコルには MQTT を用いる．

4. アプローチ

エッジの機能にはデバイスからのメッセージを受信する受信層と，クラウドへのメッセージを配信する配信層がある．配信層ではメッセージごとに配信先を決定するため高負荷となる．本稿では以下の 2 点に着目しスケーラブルなエッジアーキテクチャを提案する．

(1) 階層化: エッジを受信層と配信層の 2 層に機能分散する．

(2) エッジ内スケールアウト: 配信層をスケールアウトし，クラウドへのメッセージ配信処理を分散する．

5. 提案アーキテクチャ

5.1. アーキテクチャモデル

図 1 に提案アーキテクチャの構造を示す．提案アーキテクチャではエッジにおけるメッセージ配信の完全性を保証するために，メッセージ配信に非同期配信と同期配信を併用する．エッジ内スケールアウトに

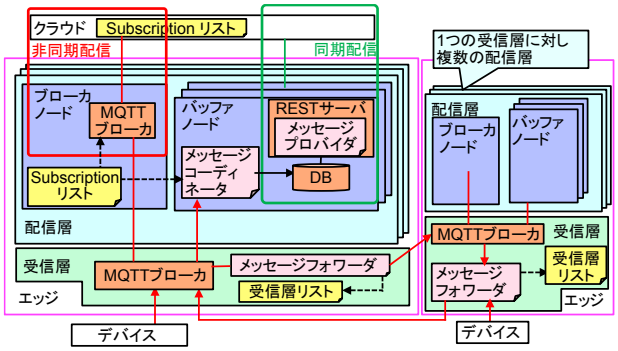

図 1 アーキテクチャの構造

よってエッジは 1 つの受信層と複数の配信層からなる．階層化により機能分散した受信層，配信層の機能を以下に示す．

(1) 受信層: 次の 3 つの機能を配置する．1)デバイスからのメッセージ受信．2)他のエッジにメッセージをフィルタリングして送信．3)配信層へメッセージをフィルタリングして配信．

(2) 配信層: 1 つのブローカノードと複数のバッファノードからなる．各ノードの詳細を以下に示す．1)ブローカノード: クラウドへメッセージをフィルタリングし非同期で配信する．2)バッファノード: メッセージをドキュメント指向データベースへ一定期間保存しクラウドへ同期で配信する．

配信層ではブローカノードとバッファノードが独立して受信層に対する Subscriber となることで，同期配信と非同期配信を分離し，非同期配信におけるスケーラビリティを向上する．

5.2. メッセージ配信モデル

図 2 にメッセージ配信モデルを示す．図 2 ではクラウドはエッジ A からメッセージを受信する場合を示す．

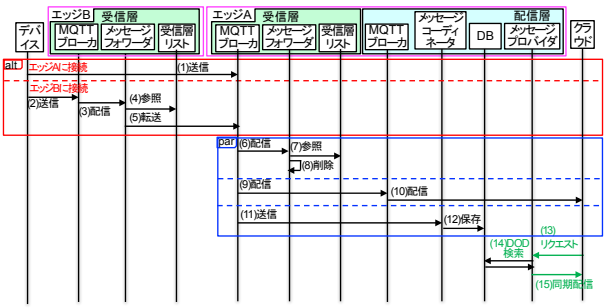

図 2 メッセージ配信モデル

メッセージフォワーダは同じ受信層の MQTT ブローカが受信するメッセージを全て受信し，受信層リストを参照して他のエッジに転送，もしくは転送先が存

A Scalable Edge Architecture for IoT Systems and its Evaluation
† Shingo Hamano, Graduate School of Science and Engineering, Nanzan University.
‡ Mikio Aoyama, Department of Software Engineering, Nanzan University.

在しない場合は削除する．これにより配信層はデバイスがどのエッジにメッセージを送信する場合でも，クラウドに配信するメッセージを 1 つの受信層から受信できる．さらに受信層でメッセージの一部をフィルタリングし配信層に配信することで，配信層でのフィルタリング処理を削減することができる．

6. プロトタイプの実装

提案アーキテクチャのプロトタイプを表 1 に示す環境上で実装した．主たるコンポーネントは図 1 に示すメッセージのフォワーダとコーディネータである．エッジのマシンには Raspberry Pi model B+を用いた．

表 1 エッジのソフトウェアコンポーネント

コンポーネント名	ソフトウェア/バージョン
MQTT プロトコル	3.1.1
MQTT クライアント	Apache Paho-MQTT / 1.3.1
MQTT ブローカ	Apache Mosquitto / 1.4.14
DBMS(ドキュメント指向 DB)	MongoDB / 2.4.14
受信層リスト管理システム(KVS)	Redis / 3.2.6
アプリケーション実装言語	Python / 2.7.13

7. 例題への適用

提案アーキテクチャのスケーラビリティを確認するために図 3 に示す 3 つのシナリオをそれぞれ 3 回実行した．デバイスは 2 分間に 20 ミリ秒間隔で計 6,000 個のメッセージをエッジに送信する．デバイスのメッセージ送信先は乱数を用いて決定する．

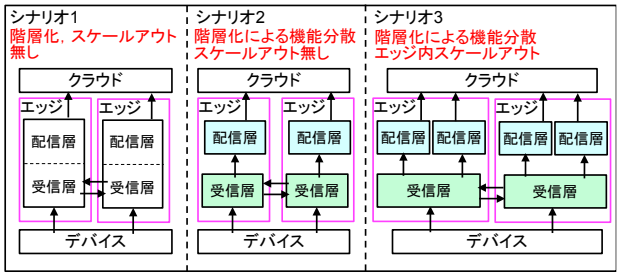

図 3 実行シナリオの概要

8. 評価

(1) CPU 使用率による評価

図 4 に各シナリオの配信層における OS を除く CPU 使用率を示す．図 4 内の吹出しは 1 件目のメッセージを受信を起点とし，10 秒後から 110 秒後までのシナリオ毎の CPU 使用率平均値を示す．シナリオ 1 と 2 の CPU 使用率の差から，階層化によって配信層から他のエッジへのメッセージ転送処理を分離したことで，メッセージ配信処理の負荷の軽減を確認した．さらに，シナリオ 3 ではシナリオ 2 と比較し，CPU 使用率が約 2 分の 1 になっている．これは，エッジ間スケールアウトにより各配信層が処理するメッセージの数が約 2 分の 1 になったためである．

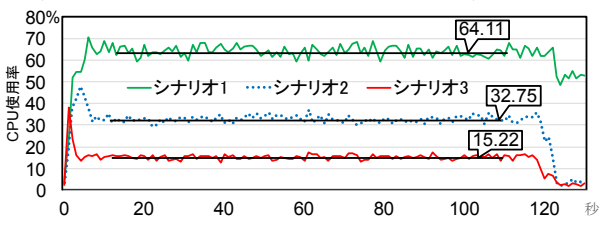

図 4 CPU 使用率

(2) ロードアベレージによる評価

図 5 に各シナリオの配信層における式 1 で定義したロードアベレージを示す．図 5 で実線は全配信層の平均値を示し，破線は最高値を示す．

$$\left(\sum^{\text{過去1分間分}} \frac{\text{実行中のプロセス数}+\text{待ちプロセス数}}{\text{CPUの最大処理プロセス数}}\right)/\text{1分間の計測回数} \quad (1)$$

シナリオ 1 では時間経過と共に増加しているが，シナリオ 2，3 では時間経過による増大しないことから，階層化によってスケーラビリティの実現が可能であることを確認した．また，シナリオ 3 では最高値が 1.0 を超えることはない．これは，配信層で処理するメッセージの負荷分散により，各配信層でメッセージ配信と保存で実行するプロセス数が削減できたためである．

9. 考察(先行研究との比較)

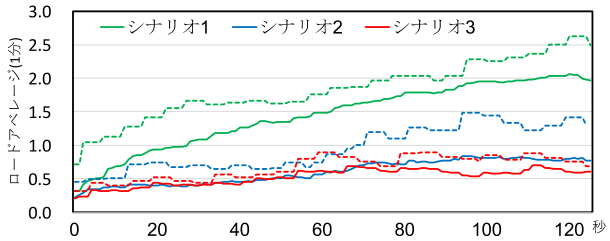

図 5 ロードアベレージ

提案アーキテクチャと先行研究の QEST ブローカ[2]の特性を比較した結果を表 2 に示す．提案アーキテクチャでは，処理メッセージ数の増大に応じてエッジでスケールアウトできるので高いスケーラビリティを実現していると言える．

表 2 関連研究との比較

視点	提案アーキテクチャ	QEST ブローカ
拡張性	＋＋	＋
配信処理の分散性	＋＋	－
メッセージ数の耐久性	＋	－

10. まとめ

階層化とエッジ内スケールアウトによるスケーラブルな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 CPU 使用率とロードアベレージの測定によって提案アーキテクチャの有効性を示した．

参考文献

[1] R. Buyya, et al. (eds.), Internet of Things, Morgan Kaufmann, 2016.
[2] 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.
[3] ISO/IEC 20922:2016, Information Technology - Message Queuing Telemetry Transport (MQTT) V. 3.1.1, 2016.
[4] W. Shi, et al., The Promise of Edge Computing, IEEE Computer, Vol. 49, No. 5, May 2016, pp. 78-81.
[5] S. Tarkoma, Publish/Subscribe Systems, Wiley, 2012.