WoTに基づくエッジマイクロサービスを用いた論理センサアーキテクチャに関する一検討

宮越 一稀[1] 寺西 裕一[2,1] 川上 朋也[3,1] 義久 智樹[1] 下條 真司[1]

概要：本稿では，エッジコンピューティング環境においてIoTアプリケーションがセンサデータと処理結果を同様に扱うことを可能とする， Web of Thing (WoT) に基づく新たなセンサデータ処理アーキテクチャ「WoTベース論理センサアーキテクチャ (WoT-based Logical Sensor Architecture: WLSA) 」 を提案する．WLSAは，処理結果を複数のデータフロー間で再利用することにより，IoTアプリケーションに必要な計算資源とネットワーク資源を削減する．また，アプリケーション開発者が処理結果を共有することを意識することなく，データフローの部分集合を実行プロセスの結果を再利用するプロセスで自動的に置き換えるデータフロー変換 (Data Flow Transformation: DFT) アルゴリズムも合わせて提案する．著者らは,WLSAに基づくプロトタイプシステムをNode-REDデータフロー処理フレームワークを用いて実装した．また，エッジコンピューティング環境のもと，本プロトタイプシステムを用いてオブジェクト検出アプリケーションを実行し，その有効性を評価した．評価の結果，WLSAによって，少ない計算資源使用量のもと実行時間を低減可能（4% CPU使用率，100 ms以下の応答時間で20のアプリケーションを収容可能）となることを確認した．

A Preliminary Study of a Logical Sensor Architecture Using WoT-based Edge Microservices

Kazuki Miyagoshi[1] Yuuichi Teranishi[2,1] Tomoya Kawakami[3,1] Tomoki Yoshihisa[1] Shinji Shimojo[1]

1. はじめに

インターネットを介してデータを取得できるIoTセンサデバイス数の急速な増大している．これにともない，IoTデバイスからデータを収集して分析することで，家電機器や設備等の制御の効率性や機能を拡大し，制御の自動化等を行うIoTサービスを実現する技術の研究開発が活発化している．こうしたサービスでは，連続的かつリアルタイムにデータ処理を行うストリームデータ処理が必要となる．代表的なIoTサービスの例としては，センサやカメラで観測した室内の温度，湿度，人の数や位置に応じた空調機器の自動制御があげられる．本稿ではこのような制御アプリケーションを典型的なユースケースとして想定する．特に，カメラからの映像データを処理することにより空調機能を改善するアプリケーションの実現を目指す．

異なる機能を組み合わせたストリーム処理アプリケーションでは，プロトコルやデータ定義の違いなどにより異なる機能を相互接続できない，システムの孤立化 (サイロ化) の問題がある．この問題に対処するために，センサやセンサデータを標準化されたWeb技術で提供するWeb of Things (WoT) [1] 呼ばれる概念が提案されている．また，ストリームデータ処理アプリケーションを実装するため，クラウド内のマイクロサービスを接続することによりデータフローを定義するサービスメッシュと呼ばれるシステムの研究開発も多く行われている．開発の容易さと既存の資源の使いやすさのため，サービスメッシュでは，異なるマイクロサービスが接続に，しばしばWoTと同様にWeb技術 (例:HTTP/HTTPSによるREST API) が用られる．サービス開発者がWebマイクロサービス間の接続

[1] 大阪大学
Osaka University, Japan
[2] 情報通信研究機構
NICT, Japan
[3] 福井大学
University of Fukui, Japan

関係や処理ロジックによってサービス全体を記述可能とする WebThings Gateway [2] や Node-RED [3] などの GUI ベースのプログラミング環境も数多く開発されている．

一方，ETSI [4,5] によって標準化が進むエッジコンピューティング環境においては，実行の効率化や応答時間の短縮のために，クラウド上のマイクロサービス等をエッジコンピュータ上にオフロードし，実行することが想定されている．しかし，エッジコンピュータは一般に省電力デバイスで実装されるため，計算リソースや通信リソースが十分でなく，処理要求が増加すると実行時間や転送時間が長くなる等のリソース制限の問題がある．サービスメッシュを実現するネットワーク機能である ISTIO [6] には，単一のマイクロサービスを複数の IoT サービス間で共有・分散する機能が備わっている．こうした分散機能を用いて，エッジコンピュータ上のマイクロサービスを複数の IoT サービス間で共有することにより，IoT サービス毎にプロセスを起動する必要がなくなり，リソース使用量を削減できる．しかし，この方法では，入力となるデータはサービス毎に異なるため，各サービスの処理は独立して実行する必要がある．このため，IoT サービス数が多くなると，データ処理に使用する計算リソース量やデータ転送に使用する通信リソース量は増加する．

本研究では，上記問題を解決するため，IoT サービス間でデータの処理結果を再利用することにより，データ処理に必要なリソースを削減できる「WoT ベースの論理センサアーキテクチャ (WoT-based Logical Sensor Architecture: WLSA)」を提案する．WLSA は，WoT ベースのマイクロサービスのための WoT 論理センサの概念を提案し，複数の IoT サービスにおける処理結果を再利用する．本稿の貢献は以下の通りである．

- WoT ベースのエッジマイクロサービスの新しい論理センサ展開アーキテクチャ WLSA を新たに提案し，その概要を示す．
- 提案アーキテクチャに基づくシステム設計とプロトタイプ実装方法を示す．
- プロトタイプ実装を用いた評価によりアーキテクチャの有効性を実証する．

2. 想定環境

図 1 に本稿で想定するエッジマイクロサービスシステムモデルを示す．本稿では， WoT ベースのマイクロサービスアーキテクチャを想定する．想定される環境は以下の通りである．

- センサは WoT によって管理される．各センサには URI があり，HTTP/HTTPS によるセンサへの参照アクセスやアクチュエーションを可能とする．
- マイクロサービスもセンサと同様に URI を持ち，HTTP/HTTPS のリクエストとレスポンスによりデータ処理のサービスを提供する．

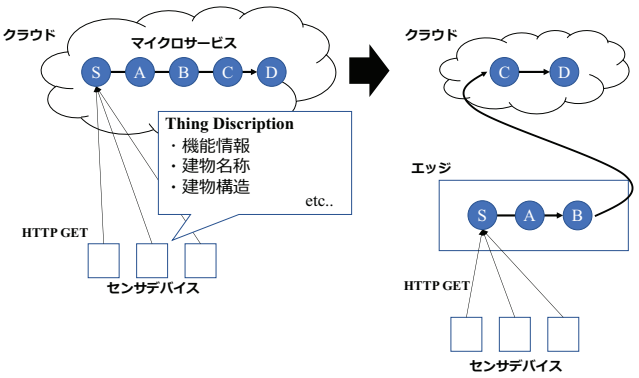

図 1 想定環境

WoT とは， IoT におけるサイロ化を回避し，IoT 運用における Web 技術を利用するために W3C が提案している概念である．WoT の基本的な考え方は，モノにリンク可能で発見可能な URI を Web 上に与え，標準的なデータモデルと API を定義して相互運用可能にすることで，分散化されたモノのインターネット (Internet of Things: IoT) を作ることである．WoT では IoT のアプリケーション層を統一し，HTTP，Javascript，REST などの既存の Web 技術を用いて複数の IoT プロトコルを連携させることが目的とされている．WoT では，各センサに Thing Description（TD）の形式でプロパティが与えられる．本稿では，TD にはセンサの機能情報だけでなく，位置や建物などのメタ情報も付加されていることを想定する．エッジコンピューティングにおいては，これらメタ情報やデータポリシ，ネットワークパス，実行効率を考慮することで，センサが入力されるマイクロサービスの配置の仕方を決定するものとする．アプリケーションからは，処理要求をデータフローとして定義するものとし，処理実行システムでは，クラウド上にデプロイされたマイクロサービス，あるいはエッジコンピュータ上にオフロードされたマイクロサービスを，定義されたデータフローに基づいて実行する．

本稿で扱うデータフローの各データプロセスはステートレスクラス (stateless class) [7] に属するものとする．ステートレスクラスの処理は状態を保存せず，リエントラント (reeentrant)，すなわち，任意の実行コンテキストで再利用可能である．また本稿では，データフローはパイプラインまたはパイプラインの組み合わせにより表現されるものとする．複数のパイプラインの入出力を publish/subscribe 等の多対多のデータ配信の仕組みにより接続することで，複雑な論理構造も表現可能である．一般に，複数のパイプラインを組合せた構造は有向非循環グラフ (Directed Acyclic Graph: DAG) として表される．ステートレスクラスのパイプラインデータフローモデルは，広範囲の応用に適用でき，Spark [8] ，Beam [9] などの OSS プラットフォームにおけるストリームデータ処理で採用されている．

上記の環境において，本稿では，WoT ベースのエッジマイクロサービスを実行するプラットフォームの要件を次のように定義する．

- データ処理にかかる時間 (latency) の最小化
  エッジコンピューティングで想定される IoT サービスでは，実世界の動的な状況の変化に関する情報ができるだけリアルタイムに得られるよう，迅速に処理結果が出力できる必要がある．計測等の処理結果を一定間隔で定期要求するアプリケーションでは，アプリケーションが次の定期要求を発行するまでに応答を返すことが求められる．
- 収容アプリケーション数の最大化
  本稿では，複数のアプリケーションがエッジネットワーク上で動作すると仮定する．エッジコンピューティングリソースには制限があるが，プラットフォームには処理性能を維持しながら可能な限り多くのアプリケーションを収容することが求められる．

3. WLSA

3.1 WLSA の概要

本稿では， WoT ベースマイクロサービスの出力を仮想 WoT センサ (論理センサ) として扱い，それを再利用可能にする WLSA アーキテクチャを提案する．

WLSA では，アプリケーションはデータフローとして定義され，センサによって生成されたデータは，複数のデータプロセスによって処理されることを想定する．また本稿では WLSA において，データ入力は，センサから周期的かつ連続的に生成されるストリームデータであると仮定する．WLSA の基本的な考え方は以下の通りである．

- 本プラットフォームでは，センサデータの処理結果は WoT 論理センサとして提供される．論理センサは一つの WoT のインスタンスとして実行され，アプリケーションは，独立した WoT センサにアクセスしているかのように，HTTP/HTTPS を介して論理センサにアクセスすることができる．
- アプリケーション開発者は，データフローを定義する際，複数のプロセスから構成されるパイプラインを定義の単位とする．データフローが実行されると，パイプライン上の全プロセスの処理結果を WoT 論理センサとする．
- アプリケーションによって指定されたパイプライン上のプロセス集合の部分集合が既に WoT 論理センサとして存在する場合，アプリケーションはパイプラインのサブセットを既存の WoT 論理センサに置き換える．したがって，アプリケーション開発者は，パイプラインの再利用について意識することなく既存の WoT 論理センサを再利用可能である．

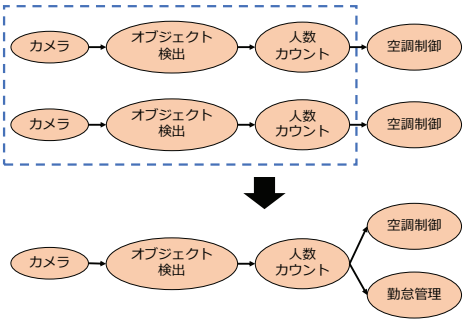

図 2　シナリオ例：オブジェクト検出

3.2 シナリオ例：オブジェクト検出

以下に， WoT 論理センサの典型的なユースケースシナリオを示す．この例では，カメラで撮影した画像からオブジェクトを検出し，その検出結果を用いて画角内に存在する人の人数を得て，人数に応じて空調機器を制御するデータフロー処理アプリケーションを想定する．この場合の処理は「カメラ → オブジェクト検出 → 人数カウント → 空調制御」というパイプラインで表現することができる．

WoT 論理センサが用いられる例として，例えば，パイプライン「カメラ → オブジェクト検出 → 人数カウント → 勤怠管理」で表されるアプリケーションが起動した場合を考える．この場合，上記の空調制御アプリケーションが既に実行されているとすると，この勤怠管理アプリケーションはパイプラインに共通部分が存在するため，その共通部分を再利用する (図 2)．

エッジコンピューティング環境においては，モバイルユーザがそれぞれ，自身が居る周辺のセンサを活用する好みのアプリケーションを利用している状況下においては，このような冗長性（共通部分のあるデータフローの実行）はしばしば生じると考えられる．パイプライン上のマイクロサービスが CPU 負荷の高い計算（例えば，上記オブジェクト検出処理）を必要とする場合，単一のエッジコンピュータ上で多数のアプリケーションを収容することが困難となる．WoT 論理センサを使用することにより，マイクロサービスのプロセスを複数のデータフローアプリケーション間で共有でき，必要な計算リソースを削減可能となる．

3.3 WLSA でのデータフロー変換

WLSA では，プラットフォームが，データフローの一部が既にエッジネットワークで実行されているか否かを判別できるよう，各エッジネットワーク内にリポジトリを配置する．リポジトリでは，エッジネットワーク内の実行中の WoT 論理センサが管理される．プラットフォームはパイプラインに対応する URI を指定してリポジトリを参照することで実行中の WoT 論理センサを発見する．

本稿では，WLSA において，論理センサを提供するためのデータフロー変換 (Data Flow Transforming : DFT) ア

Algorithm 1: DFT アルゴリズム

```
1  upon receiving ⟨D⟩
2  k ← search LCP of D from repository;
3  if k = φ then
4      S ← φ;
5      foreach d_i ∈ D do
6          S ← S ∪ {d_i};
7          p ← start HTTP process to get result of d_i;
8          register S, p to the repository;
9      D′ ← D;
10 else
11     p ← corresponding process for k;
12     D′ ← {p} ∪ {D − k};
13 start D′;
```

ルゴリズムを提案する．アルゴリズム 1 は，DFT アルゴリズムの概要を示している．パイプラインデータフロー D を以下の通り表記する．

$$D = \{d_0, d_1, \ldots, d_{|D|-1}\}$$

ここで，d_i はマイクロサービスの i 番目の URI を表す．D は，データをパイプラインとして適用するマイクロサービスの順序集合である．上記モデルでは条件分岐等の構造があるパイプラインは扱っていない．分岐等がある場合は，データフローを複数のパイプラインに分割する必要がある．

アルゴリズム 1 は，アプリケーションからデータフロー D を実行する要求を受信したプラットフォームの動作を示している．まず，D と共通のプレフィックス が最も長い (Longest Common Prefix: LCP) キーを持つエントリをリポジトリから取得する（2 行目）．D と共通のプレフィックスを持つエントリがない場合，D の全ての部分集合をキーとするエントリを登録する．アルゴリズム上では，S がキーに対応し（6 行目），キーへの GET 要求に対して，対応する結果を応答する HTTP プロセス p を開始する（7 行目）．次に，S，p のペアエントリをリポジトリに登録する（8 行目）．

D と共通のプレフィックスを持つエントリがある場合，対応するプロセス p は D に再利用される．すなわち，共通のプレフィックス (k) は D から削除され，p（12 行目）に接続される．いずれの場合も D' が変換後のデータフローとなり，上記 DFT 処理ののち，D' が開始される（13 行目）．

図 3 に DFT の動作例を示す．この例では，新規アプリケーションが定期的にセンサからデータを取得し，プロセス A，B がデータフローパイプラインとして適用される．この例では，C がデータフローの出力に対応する．DFT では，プロセス A と B の出力がメモリに格納され，これ

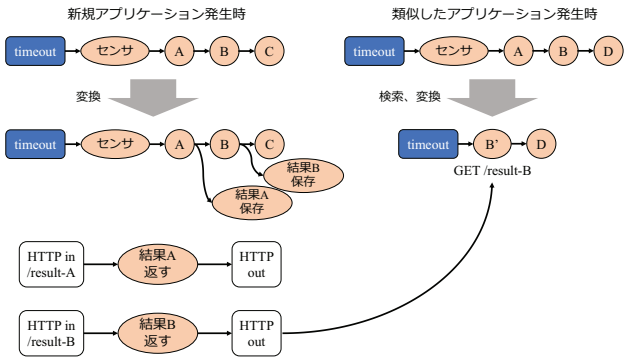

図 3　DFT の動作例

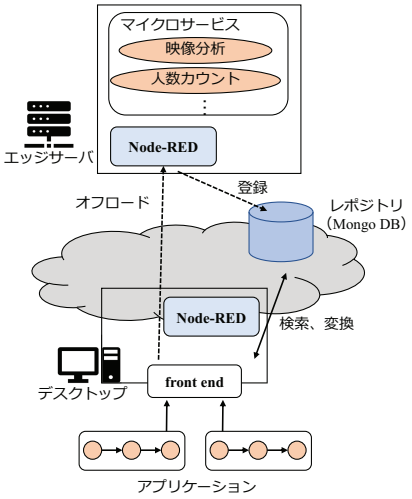

図 4　実装したシステムの構成

らの出力に対応する HTTP プロセスが生成される．その後，共通部分を持つアプリケーションのデータフローパイプライン（この場合は A，B，D）が到着すると，B より前のパイプライン部分は B' に置き換えられる．B' は，新規アプリケーションで B の結果を返す HTTP プロセスに対応する．

上記アルゴリズムにより，プラットフォームは，同じデータ処理結果を生成する既に実行中のプロセスを自動的に再利用する．DFT には，パイプラインの各サブセットに対して新しい HTTP プロセスを生成するオーバーヘッドがある．しかし，このプロセスは，新たにパイプラインを複製する場合と比べると，計算リソースの利用量は小さく抑えられる．

WLSA では，処理結果が複数アプリケーション間で共有されるため，実用システムへ適用する上ではデータのアクセス権について考慮が必要である．処理後のデータへのアクセス権がある場合のみデータフローパイプラインの接続を許可するといった制御を行う必要があるが，本稿の検討スコープ外とする．

3.4　Node-RED を使用した実装

本稿では Node-RED [3] データフロー処理フレームワー

表 1　評価のパラメータ

Parameters	Setup (Value)
エッジサーバ	Mac-mini (Intel Core i6, 32GB メモリ)
クライアント	Macbook Pro (Intel Core i3, 8GB メモリ)
データ生成間隔	3 (秒)
データサイズ	27K bytes
ネットワーク帯域	2.4 Gbps WiFi
オブジェクト検知	YOLOv3
試行回数	10
アプリケーション数	1 から 20 まで変化

クを用いて提案アーキテクチャのプロトタイプを実装する．リポジトリの実装には MongoDB [10] を適用した．プロトタイプシステムの構成を図 4 に示す．

アプリケーションは，クライアントのコンピュータ（デスクトップ）にデータフローを実行するよう要求する．クライアントのコンピュータには，データフロー要求を受け付けるフロントエンドがある．

データフローは，クライアント上の Node-RED によって定義及び起動される．Node-RED 上の新規アプリケーションが起動すると，前節で説明した MongoDB をリポジトリとして使用する DFT アルゴリズムが実行される．

プロトタイプ実装では，変換されたデータフローはエッジサーバへオフロードされ，エッジサーバ上の Node-RED プロセスとして実行される．リポジトリにデータフローが存在しない場合，エッジサーバ上の Node-RED はマイクロサービスを新たに起動する．マイクロサービスは，エッジコンピューティングの動作に従って，クラウドからエッジサーバにオフロードされる．

4. 評価

4.1 設定

提案した WLSA の有効性を示すために，前の節で述べたプロトタイプ実装を用いて評価を行った．

エッジサーバとしては小型 PC サーバ相当 (Mac mini: intel Core i 6，32 GB メモリ)，クライアントコンピュータにはラップトップパソコン相当 (Macbook Pro: intel Core i 4，8 GB メモリ) を使用した．

評価のために通信遅延，CPU 使用量，メモリ使用量を測定した．典型的なアプリケーションとして，「カメラ → オブジェクト検出 → 人数カウント」のデータフローを使用した．「カメラ」，「物体検出」，「人数カウント」プロセスはマイクロサービスとして実装した．カメラは WoT イメージセンサとして実装した．評価では，カメラはダミーとして動作し，JPEG 画像を生成する．JPEG 画像としては，部屋内に人が写っている画像を用いた．オブジェクト検出処理には，YOLOv 3 [11] を用いた．評価のパラメータを表 1 にまとめる．

同じ評価設定で WLSA を以下の 2 つの手法と比較した．

- エッジでプロセスを共有 (Share Process：SP)

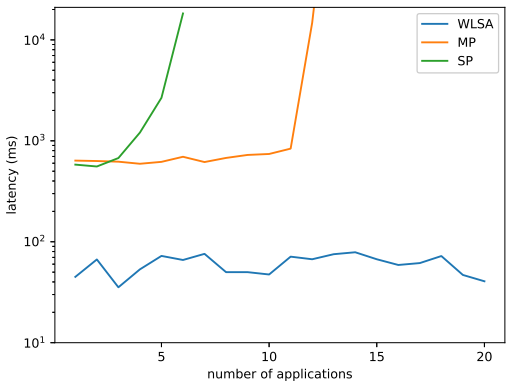

図 5　処理遅延

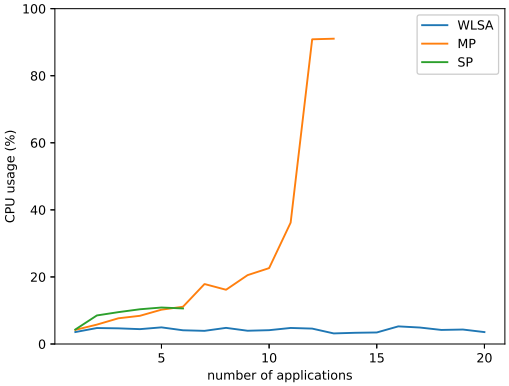

図 6　CPU 使用率

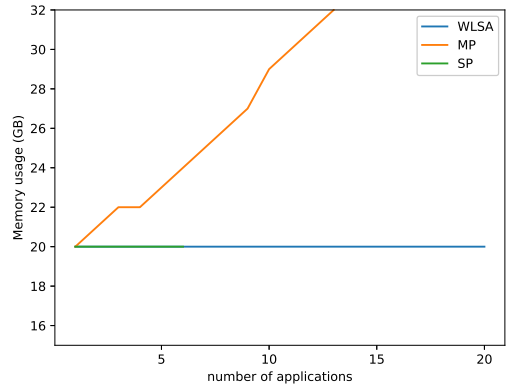

図 7　メモリ使用量

SP は最も単純な手法であり，1 つのオブジェクト検出マイクロサービスと 1 つの人数カウントマイクロサービスのみがエッジサーバ上で実行される．これらのプロセスは，全てのアプリケーションで共有される．実行されるデータフローの数は，SP 内のアプリケーションの数が増えるにつれて増加する．

- エッジで複数のプロセスを実行 (Multiple Process：MP)

MP は，サービス毎に専用のリソースが割り当てられる仮想化環境，いわゆる「スライス」による動作に対

応する．各アプリケーションは，独自のエッジコンピューティングリソース（CPU コア）を割り当てる．MP では，各アプリケーションに対して 1 対 1 対応で Node-RED プロセスが実行される．

4.2 結果

図 5 は，各手法の処理遅延を示す．y 軸は対数目盛で示されている．処理遅延は，10 回のテスト結果の平均である．SP と MP の場合，処理遅延はアプリケーション数の増加とともに増加した．SP の場合はアプリケーション数が 6, MP の場合はアプリケーション数が 13 になると，処理遅延は非常に大きくなった．このように MP のアプリケーション数に限界が生じたのは、マイクロサービスがネットワークの輻輳により待ち時間が大きくなったためと考えられる．WLSA の場合，処理遅延は 20 個のアプリケーションを実行した場合も 100 ms 以下であった．これは，WLSA では，アプリケーションからの要求に対して格納された処理結果を応答するだけでよいためである．したがって，オブジェクト検出の計算負荷はアプリケーション数の増加によらずほぼ横ばいとなった．この結果から，WLSA におけるより多くの HTTP プロセスが起動するオーバヘッドは，この評価環境では十分に小さいことが分かった．

SP と MP で遅延が増加した理由をより詳細に調べるため，アプリケーションの数を変更してエッジサーバの CPU 使用率とメモリ使用率を測定した．図 6 と図 7 に，それぞれ CPU 使用率とメモリ使用量を示す．

SP では，アプリケーション数の増加に伴い，CPU 使用率は 15%に増加したが，メモリ使用量は変化しなかった．これは，プロセスが占有する CPU コアが 1 つだけであるためである．アプリケーション数の上限は，オブジェクト検出プロセスによって CPU 負荷が上限に達し，前回のオブジェクト検出処理要求が終了する前に，ネットワーク経由の要求が到着したために発生しているためと考えられる．

MP では，アプリケーション数の増加に伴い， CPU 使用率は 90%以上に増加し，メモリ使用量は 32 G (最大メモリ使用量) に達した．これは，アプリケーション数の増加に伴い，占有 CPU コアが増加したためである．エッジサーバは 12 個の CPU コアを持つため，実行されるアプリケーションの最大数は 12 となっている．図 6 では，アプリケーション数が 12 より大きいと CPU 使用率 90%程度からほぼ横ばいとなることが分かった．この環境では CPU およびメモリがボトルネックになっていたと考えられる．

これらの方法と比較して，WLSA はアプリケーション数に対する規模拡張性が高いことが分かる．CPU 使用率は約 4%程度であり，メモリ使用量はアプリケーションの数が増えても増加しなかった．これは，アプリケーション数が増加しても，処理結果を再利用するため，新たに処理のためのマイクロサービスが生成されないことによる．

5. 関連研究

IoT/WoT サービスを再利用可能とするサービス発見に関する手法がいくつか提案されている．Mayer ら [12] は，Web 対応の Smart Things のためのセマンティック発見サービス DiscoWoT を提案している．Kallab ら [13] は，WoT を含む RESTful サービスを動的に構成するスケーラブルな検索を可能とするサービス発見手法を提案した．こうしたサービス発見技術は WLSA におけるリポジトリとして利用可能である．

Mrissa ら [14] は，WoT のセマンティックな発見と機能呼び出しをサポートするモデルとして，Avatar と呼ばれるアーキテクチャに基づいてオブジェクトの機能を記述し，機能の推論を可能とするモデルを示している．Avatar は，プロトコルやセマンティックなアノテーションに基づいて，機能を Web サービスとして公開する．Avatar アーキテクチャは WoT の動的呼び出し機能を提供するが，WLSA の重要な機能である処理結果の再利用や共有は行っていない．

クラウドとエッジコンピューティングネットワークにおけるデータフロープラットフォームを扱う既存研究も盛んに行われている．FRED [15] は，複数の IoT アプリケーションのためクラウド上に Node-RED を提供するプラットフォームである．FRED はクラウド上のサーバインスタンス上に複数の Node-RED 実行環境を提供しており，前節の性能評価における比較方法 「MP」 に対応するアーキテクチャとなっている．

Teranishi ら [16] は，近接検索が可能な構造化オーバレイネットワークを使用して，エッジコンピューティング環境でオフロードと自動スケーリング機能を提供するデータフロー処理プラットフォームを提案している．Ishihara ら [17] は階層エッジコンピューティングネットワークにおけるデータフロー処理コンポーネントの効果的な展開方法を提案している．しかし，既存の研究のいずれも，WLSA のような論理センサは扱えない．

一方，論理センサを提供するクラウドベースのサービスはいくつか存在する．Xively [18] (Pachube という名称でも知られる) は，センサを Web に接続することを可能にする初期のセンサオンラインサービスの 1 つである．Xively は，センサデータとセンサアクセス API を管理するための独自のウェブサイトを提供している．ThingSpeak [19] は，LAN 経由またはインターネット経由の HTTP により，デバイスやモノからデータを取得・保存するオープン API を提供している．論理センサに関する既存研究として，Misra ら [20] は，複数のセンサオーナと異種センサノードの存在下で高品質の Sensors-as-a-Service (Se-aaS) を提供するために，センサクラウドにおける仮想センサの動的マッピング法を提案した．Aiko ら [21] は，ソーシャルネットワークサービスから類似性を持つデータを検索可能なソーシャ

ルセンサアーキテクチャを提案した．クラウドベースの仮想センサプラットフォームの中には，すでに実用化されたものもある．SensorCorpus [22] は，いくつかの分析モジュールをリアルタイム処理に組み合わせて論理センサを提供する「センサデータストリーム処理」機能を提供する．しかし，上記であげた論理センサに関する既存の研究またはサービスでは，いずれもアプリケーション開発者は論理センサを明示的に定義および配置する必要がある．このような定義はアプリケーションが動的に配置および実行されるエッジコンピューティング環境では困難となる．これらの既存のアプローチとは異なり，WLSA では，アプリケーション開発者は動的な配置・実行環境を意識せずともデータフロー定義の再利用が可能である．

6. おわりに

本稿では，WoT アプリケーションの増加にともなう処理遅延の増大の課題に対し，処理結果を仮想 WoT センサとして再利用する WLSA を提案し，その有効性を示すことを目的とした．WLSA を用いることでデータ処理に必要なリソース使用量を低減することができ，リソースの上限が比較的小さいエッジコンピューティング環境においてアプリケーションの収容数を増大させ，処理時間を短縮できることをプロトタイプ実装を用いた評価により示した．

IoT におけるリアルタイムアプリケーションではセンサデータ処理の制限時間が必要となる場合がある．WLSA では処理遅延の短縮を達成できたが，センサデータ処理の完了までの制限時間を考慮していない．このような処理の制限時間を考慮した WLSA の拡張は今後の課題である．また，エッジとクラウドコンピューティングリソース (CPU，メモリ) の使用状況や容量を考慮して，データフローを構成するマイクロサービスをエッジ・クラウドに動的に分散させる実行効率を向上させるといった拡張も今後の課題である．さらに，3.3 節でも述べた通り，データフロー中の処理結果の再利用に際してはアクセス制御が必要となる．WLSA はエッジコンピューティングを対象とするため，zero trust [23] の考え方も視野に入れたセキュリティの実現方法を検討したい．より一般的な論理センサの定義の仕方，より進んだ効率的な実装方法，実際のユースケース・アプリケーションへの適用も今後の課題としてあげられる．

謝辞 本研究の一部は JSPS 科研費 JP20H00584，JP18K11316 の助成を受けたものです．

参考文献

[1] D. Guinard, V. M. Trifa, and E. Wilde, “Architecting a mashable open world wide web of things,” ETH Zurich, Tech. Rep., Feb. 2010.
[2] “Webthings gateway.” [Online]. Available: https://iot.mozilla.org/gateway/
[3] M. Blackstock and R. Lea, “Toward a Distributed Data Flow Platform for the Web of Things (Distributed Node-RED),” in *Proceedings of the 5th International Workshop on Web of Things*. Association for Computing Machinery, 2014, p. 34–39.
[4] H. Tanaka, M. Yoshida, K. Mori, and N. Takahashi, “Multi-access edge computing: A survey,” *Journal of Information Processing*, vol. 26, pp. 87–97, 2018.
[5] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, “Mobile edge computing—A key technology towards 5G,” *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.
[6] R. Sharma and A. Singh, “Istio gateway,” in *Getting Started with Istio Service Mesh*. Springer, 2020, pp. 169–192.
[7] “Migration to Object Technology,” 1995.
[8] M. Zaharia *et al.*, “Apache Spark: A Unified Engine for Big Data Processing,” *Commun. ACM*, vol. 59, no. 11, p. 56–65, 2016.
[9] “Apache beam.” [Online]. Available: https://beam.apache.org/
[10] K. Banker, *MongoDB in Action*. USA: Manning Publications Co., 2011.
[11] J. Redmon and A. Farhadi, “Yolov3: An incremental improvement,” *arXiv*, vol. 1804, 2018.
[12] S. Mayer and D. Guinard, “An extensible discovery service for smart things,” in *Proceedings of the Second International Workshop on Web of Things*, 2011, pp. 1–6.
[13] L. Kallab, R. Chbeir, and M. Mrissa, “Automatic K-Resources Discovery for Hybrid Web Connected Environments,” in *2019 IEEE International Conference on Web Services (ICWS)*. IEEE, 2019, pp. 146–153.
[14] M. Mrissa, L. Médini, and J.-P. Jamont, “Semantic discovery and invocation of functionalities for the web of things,” in *2014 IEEE 23rd International WETICE Conference*. IEEE, 2014, pp. 281–286.
[15] M. Blackstock and R. Lea, “Fred: A hosted data flow platform for the iot,” in *Proceedings of the 1st International Workshop on Mashups of Things and APIs*, 2016, pp. 1–5.
[16] Y. Teranishi *et al.*, “Dynamic Data Flow Processing in Edge Computing Environments.”
[17] S. Ishihara, S. Tanita, and T. Akiyama, “A dataflow application deployment strategy for hierarchical networks,” in *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1. IEEE, 2019, pp. 326–335.
[18] “Xively.” [Online]. Available: https://xively.com/
[19] “Thingspeak.” [Online]. Available: https://thingspeak.com/
[20] S. Misra and A. Chakraborty, “QoS-Aware Dispersed Dynamic Mapping of Virtual Sensors in Sensor-Cloud,” *IEEE Transactions on Services Computing*, 2019.
[21] Z. Aiko, K. Nakashima, T. Yoshihisa, and T. Hara, “A social sensor visualization system for a platform to generate and share social sensor data,” in *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2. IEEE, 2018, pp. 628–633.
[22] “Sensorcorpus.” [Online]. Available: https://sensorcorpus.com/
[23] E. Gilman and D. Barth, *Zero Trust Networks*. O’Reilly Media, Incorporated, 2017.