「マルチメディア，分散，協調とモバイル
(DICOMO2021)シンポジウム」 令和3年6月-7月

ユーザの要求を反映するデータフロー処理基盤の提案

多々納啓人[1] 前田香織[1] 近堂徹[2] 高野知佐[1]

概要：IoT デバイスから生成される大量のストリームデータをデータソースに近いところから複数ステップで目的別に処理を行うデータフローでは，生成したデータを IoT デバイスに物理的に近いエッジで一部の処理を行うことで，負荷の分散や処理遅延の短縮，クラウドを使う運用コストの削減やリアルタイム性の高いフィードバック制御が見込める．これまでデータフローを実行できるスマートシティ向けの IoT プラットフォームの研究が行われているが，様々なデータソースを対象とした相互運用性を担保するため制御コストのオーバーヘッドが大きいという課題がある．本研究では，データフロー技術を使った特定のデータ処理や分散したデータソース上でのデータ処理ニーズを満たすために，データフローの展開にユーザ要求の反映を行いつつ，IoT サービス連携やデータ収集に必要な制御情報量やコンポーネント間のデータ転送量を減らすことで，データフロー処理全体の性能劣化を抑えることが可能なデータフロー処理基盤を提案する．この処理基盤の特徴のひとつである，データ処理頻度に応じてデータフローの停止と再展開を行う自動スリープ制御の検証を行った．検証より，データフローの制御コストに対する有効性を示した．

A proposal for dataflow platform considering user requirements

YOSHIHITO TATANO[1] KAORI MAEDA[1] TOHRU KONDO[2]
CHISA TAKANO[1]

1. はじめに

IoT 技術の進展に伴い，大きな施設やインフラを構成する装置・設備から得られる大量の情報を，ネットワークを通してクラウドに集めて一括処理したり，分析したりすることが一般的となってきている．情報の種類や量が増えるにつれて，その処理方法は目的別にセンサーや監視装置など情報発生源に近いところ（エッジ）から段階的にデータ処理され，最終的にクラウドで分析をするような処理の流れに変わってきている[1]．各段階でデータを処理する一連の流れをデータフローと呼ぶ．エッジでのデータ処理機能を有することで，すべてのデータをクラウドと送受信するのではなく，現場に近い場所でほぼリアルタイムでデータ処理をする．その後，長期稼働監視や施設，インフラ全体管理の効率化などに関する分析に必要なデータはクラウドに向かって段階的に処理されるデータフローとなる．これによりクラウドでの一括処理に比べて，通信遅延を低減した処理やクラウドへの通信量の削減が可能となる．

クラウド事業者のサービスとして，Google Cloud Platform DataFlow(GCP)[2]や Azure IoT Edge[3], AWS Greengrass[4], EdeX[5]といったデータ処理や分析用のサービスがある．これらのサービスでは，処理の優先度やデータ処理を行うリソース選択などの要望や要求を記述したコンテンツ情報を反映したデータフローの実現やデータセンタ上のリソースをオートスケールする．しかし，データセンタ上で一括処理するので，データ量が多いと使用するストレージや帯域は大きくなるため，運用コストが増加してしまう．データセンタまでの転送遅延もありリアルタイム性を必要とするアプリケーションには向かない．複数事業者のクラウドサービスを組み合わせてデータフローを設定することも可能だが，個々の事業者によって取得できるリソース管理情報や方法，スケジューリング要件が異なり，拡張性や相互運用性を満たすことは困難である．

この問題を解決するために，大規模複数拠点からデータ収集するスマートシティ向けの IoT プラットフォームの研究[6][7]が行われている．これらの研究では標準化したデータ構造やデータ共有のためのゲートウェイ，仮想化したデバイスを設けることにより，複数拠点間で異なる IoT サービスを連携したデータフロー基盤として相互運用性を満たすことが可能である．しかし，相互運用性を担保するためのオーバヘッドが生じ，データフロー全体の処理性能が低下するという課題がある．

本稿では，これらの課題を解決するため，広域に分散した IaaS 環境を利用してデータフローを設定し，複数拠点間をまたがるデータ量の削減を考慮したデータフロー処理基盤である“Flexible Dataflow Architecture”(以降，FDA)を提案する．

本論文の構成は次の通りである．2 章において関連する技術や研究について紹介した後，3 章において提案手法について示す．4 章において，実装方法と自動スリープ制御の実装にあたって必要なスリープ時間に関する実験の結果を示す．5 章において，まとめと今後を述べる．

1 広島市立大学大学院情報科学研究科
2 広島大学情報メディア教育研究センター

2. 関連研究

2.1 オーバーレイ技術を応用したデータフロー技術

データフローにおいて複数のデータ処理コンポーネント組み合わせてセンサーデータを転送するための技術が研究されている[8][9].

文献[8]では，P2P 構造化オーバレイをコンポーネントの検索に活用し，ネットワーク間の地理的距離とコンポーネント配備先のリソース状況を考慮した通信を実現する．分散型 MQTT Broker PIQT[10]を用いてオーバレイネットワーク上で収集したリソース情報をもとに条件付きマルチキャストを拡張することでリソース状況を考慮し適切なコンポーネント選択が可能なエニーキャスト機構を提案している.

文献[9]では，エッジやデータセンターに配置された分散型計算資源上でプロセスコンポーネントを複製し，データフローの構造を動的に変化させる P2P ベースのデータストリームルーティングアルゴリズム“Locality-Aware Stream Routing(LASR)”を用いることで IoT アプリケーションに起因するエッジやクラウドのネットワーク，計算資源の過負荷を回避している.

しかし,これらのオーバーレイネットワークを利用した制御の場合，情報収集や制御，配置にコンポーネントの検索や更新が必要となりコストが掛かるため，広域環境への適用が課題となっている.

2.2 大規模複数拠点でデータ収集をするデータフロー基盤

複数のアプリケーション・プラットフォームが混在するスマートシティのような大規模な環境を想定し，クラウドやエッジ，フォグコンピューティング上で大量のアプリケーションやデータの相互運用性を考慮した IoT プラットフォームが提案されている[6][7].

文献[6]の FogFlow では，IoT スマートシティ・プラットフォーム向けのフレームワークを標準化したプロトコルである“Next Generation Service Interface–based (NGSI) ”[11]を定義して，データ処理時に NGSI でのやりとりが可能なデータに変換する．これによりエッジやクラウドの相互運用性の課題を解決し，サービス間でのコンテキストデータの共有・再利用が可能なインターフェースの提案を行っている．そのため，FogFlow では，NGSI を用いている環境であれば，異なる IoT プラットフォーム間でも処理を行うことが可能である.

文献[7][12]の Fed4IoT では，既存の IoT プラットフォームを利用して，Fed4IoT で仮想化させた IoT デバイスやデータ共有のための Broker を用いる．共有するデータとして，生のセンサデータや中間処理結果だけでなく，それぞれの IoT プラットフォームで採用しているコンテンツ情報も共有する．利用者は共通の API を用いてデバイスへの接続方法を知る必要がなく，データ取得や処理を行い，異なるアプリケーション間でも IoT デバイスの共有を行うことが可能である.

FogFlow や Fed4IoT は複数データ収集拠点や異なるサービス連携を対象としたデータフローの設定ができ，オンデマンドにデータフローの変更や制御を行うことができるため相互運用性や拡張性がある．その一方で，アプリケーション間での相互運用性を確保するための共通化 API に必要な収集データ量や制御情報が多くなる．特に経由するコンポーネントが増えることでデータ転送スループットが低下し，データフロー全体の処理性能が低下するという課題がある.

3. データフロー処理基盤 FDA の設計

3.1 設計方針

前述のように，さまざまな機器が扱うデータ量は爆発的に増えており，そのデータ処理に求められるスピードも高くなっていることからエッジ処理が可能なデータフローが必要である．また，収集するデータの種類やその処理も多様化しており，異なる IoT サービスを連携したデータフロー基盤が求められている．そのとき，多種多様なデータ収集や異なる IoT サービスの連携に必要な相互運用性が必要であるが，それによるオーバーヘッドのためのデータフロー処理性能の低下を小さくすることが求められる．本稿では，IoT サービスの連携やデータフローの利用に必要な制御情報量やコンポーネント間の接続と維持に必要なデータ量を制御コストと呼ぶ.

提案する FDA（Flexible Dataflow Architecture）では，以下のような方針でデータフローを実行する基盤を開発する.

1) 複数拠点から異なる種類のデータ収集が可能である．ただし，データフロー実行時に拠点の変更やデータの種類の変更はないものとする.
2) 異なる IoT サービスを連携したデータフロー基盤とするが，データフロー実行時に連携するサービスの種類や数が変更することがないものとする.
3) 地理的に分散した計算リソースを利用して IaaS 環境を構築し,データフローを実行する.これにより IaaS 環境の API で取得可能なリソース情報やデータをもとにオンデマンドにデータフローの展開や負荷分散を行う.
4) 制御コストを小さくし，データフロー処理全体の性能劣化を抑えて安定したデータフローの実行を実現する.
5) ユーザの要望や要求として，以下の項目をデータフローに反映可能である.
   - データフロー開始や終了時間の指定
   - 使用するネットワーク/リソースの指定・制限
   - データ量による制限
   - 処理方法の指定

(リアルタイム処理/バッチ処理)

- データフローの優先度

FDA の設計方針と既存研究との相違点を表 3 に示す．FDA ではデータフローの制御コスト削減を目的に送信するデータはセンサーデータのみとしている．そのため FogFlow や Fed4IoT に比べて，データを利用したアプリケーションや IoT プラットフォーム間での相互運用性は低下する．しかし，異なる IoT プラットフォーム間での連携やデータの種類に変動がない環境では有効であると言える．

表 3：FDA と関連技術の比較

項目	FDA	FogFlow [7]	Fed4IoT [8]
相互運用性への対応	IaaS 環境を利用した仮想化	共通のプロトコル NGSI を定義	IoT プラットフォームを利用した仮想化
アプリケーション間共有	あり	なし	あり
処理トリガー	トピックベース	コンテキストベース	コンテキストベース
プログラミングモデル	コンテンツベース	コンテンツベース	コンテンツベース
モビリティサポート	なし	あり	なし
制御コストの削減	自動スリープ制御	考慮なし	考慮なし

3.2 システム構成

FDA のシステム構成と FDA を用いるデータフロー処理全体をそれぞれ図 1 と図 2 に示す．FDA は FDA コントローラ，OpenStack[13]が導入されたリソースノード群，各リソースノード内で稼働する FDA エージェントで構成される．これが図 2 のように複数のリソース上で連携して稼働し，IaaS 環境を構築する．

FDA コントローラのリソースデータ取得・管理部では各リソースノードから収集したリソース使用状況などの情報収集し，リソース情報 DB にそれを格納する．データフロールール適用部ではデータフローの作成時にデータフローの定義を記述するコンテンツ情報（以降，データフロールール）である JSON 形式のファイルを基に有向非巡回グラフ(Directed Acyclic Graph，以降 DAG)を作成し，さらにデータフローの構成情報を作成し，FDA エージェントに送る．このとき，使用するデータ処理用に使うコンテナ（以降，処理コンポーネント）のイメージとそこで動作するアプリケーションに関する情報はリソース情報 DB で管理する．次に，リソース情報 DB から入力されたリソース選択情報をもとにデータフローに割り当てるリソースノードを決定し，OpenStack の API を利用して，該当するリソースノードの OpenStack 上の管理用コンポーネントを用いて，処理コンポーネントの起動/停止などの制御を行う．

FDA エージェントのデータフロールール構成部はデータフロー構成情報を受け取り，その情報を基に起動された処理コンポーネントや Broker へトピック制御を行う．データフロー実行制御部では処理コンポーネントのデータの到達状況や実行状況を監視し，処理コンポーネントの停止が必要な場合はその情報を FDA コントローラのデータフロールール適用部に通知する．

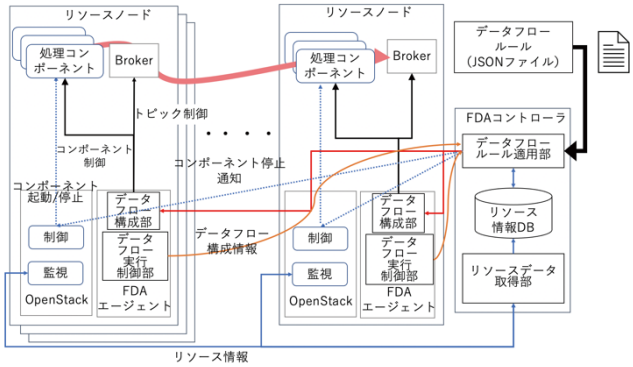

図 1　FDA のシステム構成

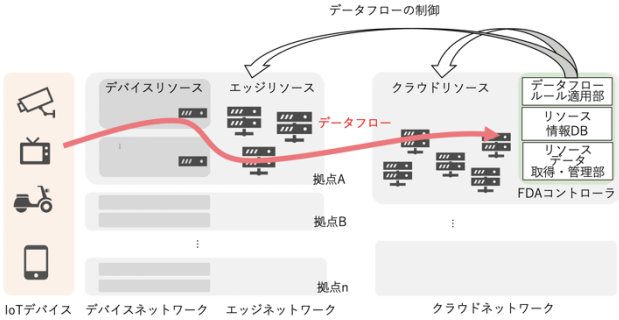

図 2　FDA を用いるデータフロー処理基盤全体の構成

3.3 自動スリープ制御

図 3 に自動スリープ制御の概要図を示す．自動スリープ制御ではデータフローの実行時にコンポーネントの状態をデータの到達間隔により制御する．自動スリープ制御によりデータが到達していないコンポーネント間の接続を解除して，最終的にコンポーネントを停止することでセッション維持に必要なデータ量の削減および計算資源の有効利用を行う．データフローを OpenStack 上の各サーバに展開する際に掛かるコンポーネント間の接続処理時間とコンポーネントの起動時間を展開コストとして，コントローラで記録する．加えて，各コンポーネントでデータが到達する間隔を記録する．各コンポーネントは展開コストを自動スリープ制御の閾値としてコントローラから受け取り，データ間隔が閾値を超えた場合に接続関係を解除して，次のデータ到達まで待機する．さらに閾値を超過した場合はコンポーネントを停止する．データが再到達した場合，各 Broker からコンポーネントへ再起動のシグナルを送信することでデータフローを再開する．

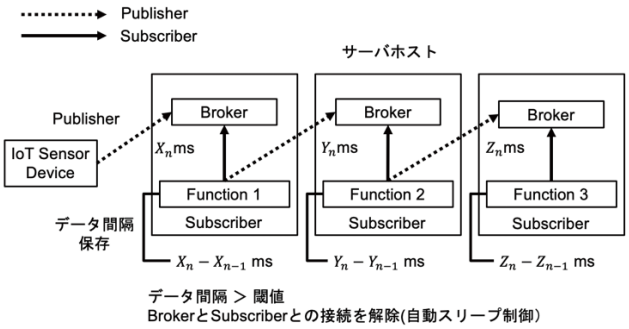

図 3　自動スリープ制御の概要

3.4 処理コンポーネント間の通信方式

展開したコンポーネント間でのデータ送信や処理結果の配信には，Publish/Subscribe 型メッセージングモデルである Apache Kafka[14] (以降，Kafka)を用いる．このメッセージングモデルは，Topic と呼ばれる識別子単位で接続を管理することができ，Topic を管理する Broker を中継して，Publisher が Subscriber にデータを送信する．Topic の命令規則は Fed4IoT と似た命令規則を用いており，データフローコンポーネントの識別子とセンサーデバイスの識別子で構成される．図 4 に Publish/Subscribe 型メッセージングモデルを用いたデータフローの構成を示す．Topic の命令規則は“Device.Function1.Function2.Function3”の構成になっており，Device がセンサーデバイスの識別子，Function がデータフローコンポーネントの識別子である．データフローの処理回数分，Function 部分を追加する．データフローの処理ごとに Topic 内の Function を削除して，接続用の Topic を作成して中継用の Broker へ接続を繰り返すことで，データフローの連続処理のためのデータ送信関係を実現している．

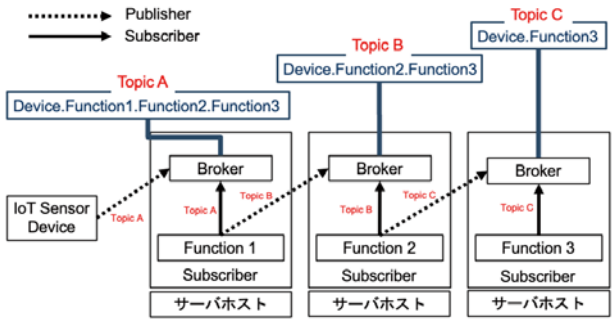

図 4　データフローの概要図

4. 実装に向けた自動スリープ機能に関する検証実験

3.1 節の設計方針(4)を実現するために提案した自動スリープ制御の有効性の検証する．小規模な IoT データ収集環境を構築し，実トラフィックを発生させて実験を行った．その詳細について記載する．

4.1 データ収集の実験環境

実験環境では図 5 のようにセンサーデータとして温度，湿度のデータを取得し，一部機能を実装したサーバホスト（リソースノード相当）にデータを送信する．

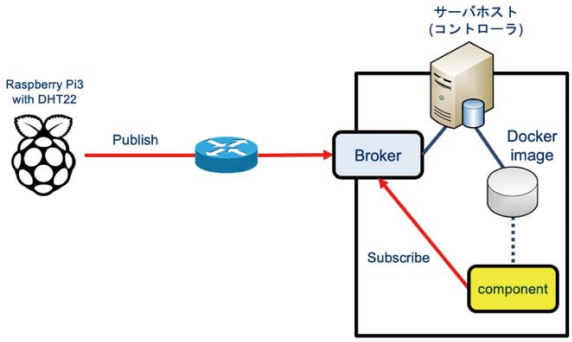

図 5　実装環境の概要

センサーデータを収集する IoT デバイスとして，Raspberry Pi3 model B（CPU: Cortex-A 53 1.2GHz）を使用し，その OS には Ubuntu20.04 を採用した. センサーには AM2302 DHT22 を使用した．

使用したサーバホストの機器を表 2 に示す．IoT デバイスとのデータ送信は Apache Kafka version 2.13-2.7.0 を利用した. コンポーネントに使用するコンテナは Docker を利用し，OS はホストと同様のものを使用する．コンテナリソースとして Docker の設定による CPU 配分の重みを 512，メモリサイズを 1GB，ストレージ 10GB のイメージを使用する．

表 2：実装環境で使用したサーバの諸元

項目	サーバ
CPU	Intel(R) Core(TM) i7-9700 CPU @ 3.00GHz
RAM	16GB
Storage	1TB
Kernel	5.8.0-45-generic
Distribution	Ubuntu 20.04.2 LTS
Docker	20.10.5
Python	Python 3.8.5

4.2 実験環境における構成要素の動作概要

(1)　IoT デバイスの処理

IoT デバイスでは取得したデータをデータ送信できるように Kafka に接続するためのパラメータ（Broker と Topic, port），リソースノードの情報やオプションを設定ファイルから指定する．Broker へ接続後は定期的にセンサーデータを Broker へ送信する．オプションにより，任意の時間を指定してデータ送信が可能である．

(2)　コンポーネントの処理

コンポーネント起動後は設定ファイルより処理アプリ

ケーションを起動して，IoT デバイスとのデータ送受信とデータ処理を開始する．コンポーネントは，データの到達間隔が設定された閾値以上になった場合，自動スリープ制御により Broker への接続解除とスリープ状態への移行が行われる．センサーデバイスからデータの送信が再開した場合，ホストから再接続の要求を送り，到達したデータの初めから受信を再開する．コンポーネントでは受信データがエラーでないかのチェックを正規表現と比較して，欠損値の除外を行う．

なお，本検証では閾値超過時にコンポーネントの停止を行わず，コンポーネント間の制御のみを行っている．

(3) サーバホストの処理

サーバホスト上のコンテナイメージをコントローラからのコンテナ立ち上げの命令とコンポーネント間の接続を行うための Topic を受け取ると，コンポーネントを立ち上げる．サーバホストではコンテナの起動時間とアプリケーションの起動時間を自動スリープ制御の閾値として使用する展開コストをコントローラに送信する．IoT デバイスからセンサーデータが送信されず，自動スリープ制御がコンポーネントで実行されたことを Broker の接続状態から認識し，次のデータが送信するまで Broker に送信されるデータを監視する．データが再度，送信された場合はコンポーネントに対して Broker への再接続のシグナルを送信する．再接続が完了するまでのデータ損失がないように Broker 内でデータのバッファリングを行う．

4.3 Publisher/Subscriber 間のデータ量の測定

スリープ時間の見積もりをするために Kafka を用いて Publisher と Subscriber 間のデータ量を測定した．図 5 の IoT デバイス（Raspberry Pi）が Publisher，コンポーネントが Subscriber として Broker を介して接続した時のデータ量である．以下の３つのシナリオで検証を行った．測定は 1 時間のデータ量であり，測定回数 3 回の平均である．接続する Topic は 1 つであり，サブスクライバーからのデータ取得間隔は 500msec と 100msec で，それぞれ検証を行なった．３つのシナリオは以下の通りである．測定した結果を図 6 に示す．

(1) Subscriber と Broker 間のみの接続

(2) Subscriber が Publisher のデータを受信
Publisher からセンサーデータの送信はしない．

(3) Subscriber が Publisher のデータを受信
Publisher からセンサーデータの送信を行う．
センサーデータの送信間隔は 1 秒間隔で，Subscriber からと Publisher からのデータのサイズはそれぞれ平均 160 バイトと Publisher から平均 51 バイトである．

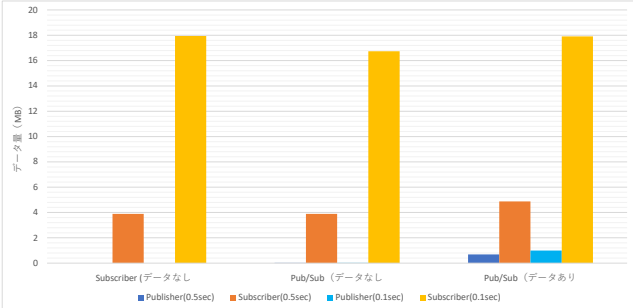

図 6　Publisher と Subscriber 間のデータ量

図 6 の結果より，データ送信がなくとも接続の確立を維持するのに，1 時間あたり Subscriber には 3.8M バイトのデータ量がある．Subscriber からのデータ更新間隔が 100msec の場合は 1 時間あたり 17.9M バイト必要となる．複数のコンポーネントを経由するデータフローでは，Publisher と Subscriber の接続数がコンポーネントの数やデータフローの数によって増加し，その間のデータ量も増加する．また，ストリームデータのようなデータ転送間隔が短いデータの場合は，Subscriber からのデータ取得間隔を短くする設定が必要になり，接続時のデータ量が増加することがわかる．

4.4 自動スリープ制御の有効性検証のための実験

図 5 の環境で自動スリープ制御の有無で転送されるデータ量を測定する実験を行う．(3)のシナリオで，データ送信間隔およびデータ停止間隔を 1~600 秒のランダムな時間間隔の場合のデータ量の測定をする．このとき，自動スリープ制御におけるスリープ判断までの閾値をコンポーネントの起動時間（3.8 秒）と Subscriber の接続時間（1.2 秒）およびセンサーのデータ取得に掛かる最大時間（5 秒）の合計 10.0 秒とする．Subscriber からのデータ取得間隔は 500msec であり，6 時間と 12 時間でそれぞれ測定を行なった．測定した結果を図 7 に示す．

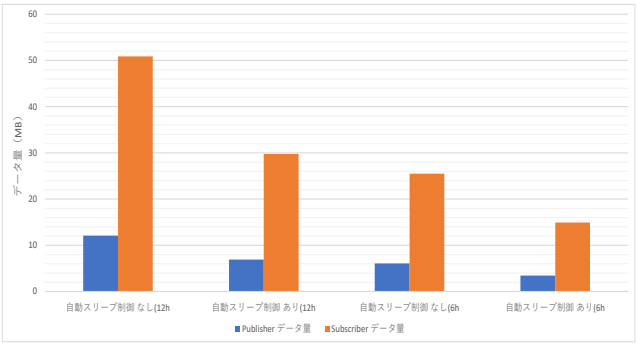

図 7　自動スリープ制御の検証結果

Subscriber では，自動スリープ制御のありとなしの場合でそれぞれ，データ量は 29.7M バイトと 50.8M バイトで自動スリープ制御によりデータ量の削減ができていることがわかる．今回の測定では，Subscriber の更新間隔は 500msec 間隔であったが，4.3 節で述べたように，更新間隔が短い場合はさらにデータ量を削減することが可能である．今回の

実験では閾値を処理に必要な時間の合計により算出した．しかし，Subscriber の再接続やデータ更新間隔を考慮した閾値の算出を行うことで，より効果的にデータフロー展開時の制御コストを抑えることができると考えている．

5. まとめと今後

本研究では，利用者の要望や要求を反映したデータフローの展開とデータフロー処理基盤の制御コストの削減を実現するため，コンテンツ情報を利用したデータフローの定義と OpenStack によるデータフロー処理基盤の設計について述べた．また，コンポーネント間の制御コストを抑えることを目的としたコンポーネントの自動スリープ制御を提案した．さらに実験環境を用いて，コンポーネント間でやり取りされるデータの測定を行い，自動スリープ制御の有効性を考察した．

現在, 本稿に示した設計方針に基づきデータフロー基盤の実装を進めている．今後，より効果的な自動スリープ制御を実現するための適切な閾値設定の評価を行う．さらに基盤としての動作を確認するために，具体的なアプリケーションを用いたデータフロー制御の有効性を明らかにしていく．

謝辞

本研究の一部は JSPS 科研 18K11266，19K11929，21H03432，及び 18K11271 の支援を受けて実施しました．

参考文献

[1] Matteo, N. Stefan, N. Schahram, D. Massio, V. and Raijv, R.: Osmotic Flow: Osmotic Computing + IoT Workflow, *IEEE Cloud Computing* Vol 4, Issue 2, pp.68-75(2017).

[2] Google Cloud: Dataflow (online), available from <https://cloud.google.com/dataflow/docs?hl=ja> (accessed 2020-06-26).

[3] Microsoft Azure: Azure IoT Edge (online), available from <https://azure.microsoft.com/ja-jp/services/iot-edge/> (accessed 2021-04-20).

[4] AWS: AWS IoT Greengrass (online), available from <https://aws.amazon.com/jp/greengrass/> (accessed 2021-04-20).

[5] EdgeX Foundry: EdgeX Foundry Home (online), available from <https://www.edgexfoundry.org> (accessed 2021-04-20).

[6] Bin, C. Gurkan, S. Flavio, C. Ernoo, K. Terasawa, K. and Kitazawa, A.: FogFlow: Easy Programming of IoT Services Over Cloud and Edges for Smart Cities, *IEEE Internet of Things Journal* Vol 5, Issue 2, pp696-707(2018).

[7] K, Ogawa. K, Kanai. K, Nakamura. et al.: IoT Device Virtualization for Efficient Resource Utilization in Smart City IoT Platform, *IEEE Percom Work in Progress*, (2019).

[8] 安田，石原，秋山：分散型 MQTT Broker を活用したコンポーネント選択手法の比較評価，Internet and Operation Technology Symposium (2019).

[9] Teranishi, Y. Kimata, T. Yamanaka, H. Kawai, E. and Hirai, H.: Dynamic Data Flow Processing in Edge Computing Environments, *IEEE 41st Annual Computer Software and Applications Conference* (2017).

[10] PIQT distributed pub/sub broker: PIQT (online), available from <http://www.piqt.org> (accessed 202103-10).

[11] Oma Open Mobile Alliance: NGSI Context Management Approved Version 1.0 – 29 May 2012 (online), available from <http://www.openmobilealliance.org/release/NGSI/V1_0-20120529-A/OMA-TS-NGSI_Context_Management-V1_0-20120529-A.pdf> (accessed 2021-04-20).

[12] 金井，吉田，金光，中里，横谷，向井，中村，上杉：Things as a Service を実現する Fed4IoT プラットフォームの研究開発, *IEICE Technical Report 信学技報*, vol. 118, no. 371, NS2018-165, pp. 41-46(2019)

[13] OpenStack: Welcome to OpenStack Documentation (online), available from <https://docs.openstack.org/victoria/> (accessed 2022-09-17).

[14] Kafka: Apache Kafka (online), available from <https://kafka.apache.org> (accessed 2020-11-09).