一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報

エッジ・クラウド連携のための IoT サービスファンクションチェイニングの性能評価

関根 響† 金井 謙治† 金光 永煥‡† 甲藤 二郎† 中里 秀則†

† 早稲田大学 〒169-8555 東京都新宿区大久保 3-4-1
‡ 東京工科大学 〒192-0982 東京都八王子市片倉町 1404-1
E-mail: †{h_sekine, kanai, katto}@katto.comm.waseda.ac.jp

あらまし 筆者らは，自動で柔軟なネットワーク内資源の有効活用の実現に向けて，IoT サービスをサービスファンクション(マイクロサービス)として分割定義し，ネットワーク上のエッジ・クラウドに分散配置しチェイニングする仕組みを研究開発している．本稿では，この仕組みを，IoT 指向ファンクションオーケストレーションと定義し，Apache Kafka，Docker，Kubernetes を用いてプロトタイプを実装し，IoT サービスを分散配備アルゴリズムに従い最適配備し，Pub/Sub プロトコルでチェイニングさせた際のアプリケーション応答時間を評価し，その性能を検証する．

キーワード IoT，サービスファンクションチェイニング，ファンクションオーケストレーション

Performance Evaluations of IoT Service Function Chaining for Edge-Cloud Collaboration

Hibiki SEKINE† Kenji KANAI† Hidehiro KANEMITSU‡† Jiro KATTO† and Hidenori NAKAZATO†

† Waseda University 3-4-1, Okubo, Shinjuku-ku, Tokyo, 169-8555, Japan
‡ Tokyo University of Technology 1404-1, Katakuramachi, Hachioji City, Tokyo, 192-0982, Japan
E-mail: †{h_sekine, kanai, katto}@katto.comm.waseda.ac.jp

Abstract To realize automatic and dynamic efficient network resource utilization, authors research and develop the IoT service function chaining. In this scheme, IoT service is divided into multiple micro service functions, and these functions are allocated to computing resources that are virtualized in the networks. In this paper, we develop a prototype implementation of this scheme, called IoT centric function orchestration, by using Docker, Kubernetes, and Apache Kafka. We optimize deployment of IoT service functions optimally according to the distributed deployment algorithm, and evaluate its application response time based on pub/sub protocol-based IoT service function chaining.

Keywords IoT, Service function chaining, Service function orchestration

1. はじめに

現在，スマートシティやスマートホームといった様々な分野において，Internet of Things(IoT)が広く普及しており，多くの研究開発が行われている[1]. IoT は，デバイス，通信，処理といった非常に多くの技術の集合体と言える．IoT サービスにおけるサービス要求は多様化しており，一方で，ネットワーク上に存在する通信資源や計算資源も非常に流動的になっている．このような流動的に変化する環境下において，常に高い IoT サービスの品質要求を満足するためには，自動的にかつ柔軟にこれら流動的に変動する資源を管理・運用する基盤が必要と言える．

これに対して，近年，ネットワーク仮想化技術の発展に伴い，通信資源や計算資源の動的スケーリングや動的割当が容易に可能となっている．また，同様にコンテナ技術の発展に伴い，サービスをファンクションレベルに分割し，コンテナを利用してファンクションを実現し，ネットワーク内に分散配備するネットワークサービスの仮想化も広く研究されている．

このような背景を受け，筆者らも，これまで総務省の「IoT 機器増大に対応した有無線最適制御型電波有効利用基盤技術の研究開発」の「課題ア：有無線ネットワーク仮想化の自動制御技術」として，IoT 指向ファンクションオーケストレーション技術の研究開発を

進めてきている．IoT 指向ファンクションオーケストレーション技術は，エッジやクラウドに配備されている資源の変動を監視し，その資源を有効利用するために，IoT サービスファンクション（マイクロサービス）の最適な選択や配置，実行スケジュールの制御を担う．これまで筆者らは，この最適配備を実現するアルゴリズムについて，実装を意識した単純な線形計画法によるもの[2]から，サービスファンクションのクラスタリングを考慮し，ワークフローとして取り扱うより複雑なアルゴリズム[3]まで提案してきた．また，サービスファンクションチェインのプロトコルとして，Pub/Sub メッセージングプロトコルを利用してエッジ・クラウド間で効率的にデータやファンクション共有を実現する仕組み[4]を提案してきた．

本稿では，これらの筆者らの先行研究をプラットフォームとして統合することを目的に，コンテナ操作を自動化するオープンソースプラットフォームとして知られる Kubernetes を利用して，エッジ・クラウド連携基盤の実現のため，IoT 指向ファンクションオーケストレーションのプロトタイプ実装およびその性能について実機検証を行う．

2. 関連研究

近年，IoT の発展と共に，重要なパラダイムとしてエッジ/フォグコンピューティングの研究が進められている．エッジ/フォグコンピューティングは物理的にエンドユーザーの近くにコンピューティングリソースを設置する．このことから，低遅延かつ局所性を用いたデータ処理が可能になっている．これらの研究が行われる中で，エッジ/フォグコンピューティングを用いた多くのシステムやアプリケーションが提案されている．[5]では，EdgeMesh と呼ばれる，ネットワーク内に存在するエッジデバイス(エッジコンピューティング)間で計算タスクを共有するシステムが提案されている．また，IoT プラットフォームにおいては，エッジ/フォグコンピューティングはクラウドコンピューティングとの連携が重要である．[6]では，クラウドコンピューティング，モバイルエッジコンピューティング，エンドデバイスからなる 3 層を統合的に利用するための EdgeFlow と呼ばれるシステムが提案されている．また，Osmotic Computing というパラダイムは，アプリケーションをマイクロサービスに分解し，エッジおよびクラウドのリソースを活用するためにマイクロサービスの動的な調整を実行することを目的としている[7]．

そして仮想化技術の研究も数多く進められている．ネットワーク仮想化技術としては，Network Function Virtualization(NFV)[8]に関する研究が発展している．NFV では，ネットワーク機能を汎用サーバ上で実行可能になる．Service Function Chaining(SFC)[9]という技術によって，NFV におけるサービス機能である Virtualized Network Function(VNF)を適切な順序で連結しサービスを提供する．またハードウェアの仮想化として，多くの研究者がコンテナベースの仮想化を試みている．コンテナ型仮想化はハイパーバイザ型と比べて，オーバヘッドが小さい．コンテナ型仮想化としては特に Docker[10]が主流となっていて，Docker を始めとするコンテナを利用したアーキテクチャが提案されている[11,12,13]．この Docker コンテナを管理するための有名なオープンソースコンテナオーケストレーションソフトウェアとしては Kubernetes[14]があり，これらを組み合わせた実用化が進められている．

3. IoT 指向ネットワークファンクションオーケストレーション

我々が研究開発している IoT 指向ネットワークファンクションオーケストレーションのイメージ図を図 1 に示す．まず前提として，IoT ネットワークファンクションは，ネットワーク機能よりもより上位のサービスレイヤを対象としており，IoT サービスを機能単位に分割し，それらを Docker に代表されるコンテナイメージによって実装する．このコンテナ化された IoT サービスを IoT サービスファンクションと定義し，ネットワーク内の計算資源を持つノードで分散配備する．IoT 指向ファンクションコントローラーは，IoT サービスが持つサービス要求を満足するように，最適なノードへ分散配備するための最適配備プランの導出や多数の IoT サービスを共存させるため，各ノードで IoT サービスファンクションの共有や実行順のスケジューリング管理も担う．また，その他の機能として，ネットワーク内の資源情報を収集するとともに，検証環境において，各 IoT サービスファンクションの性能検証も行うこととする．

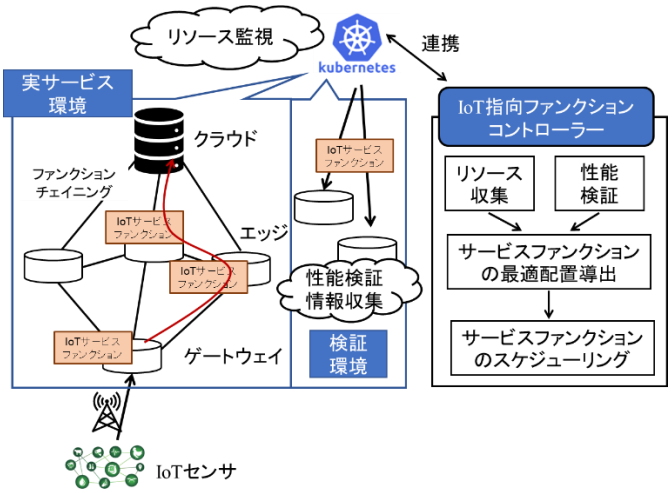

図 1. IoT 指向ファンクションオーケストレーションの概要図．

なお,サービスファンクションの最適配置やスケジューリングアルゴリズムについては，これまで非常に多くの提案がなされているため，色々なアルゴリズムを利用可能とするために,共通のインターフェース（API）を定義し，サービス要求に応じて選択できるような仕組みを想定している．実際の IoT サービスファンクションのネットワーク内へのデプロイや資源監視は，Kubernetes を利用することを想定しており，IoT 指向ファンクションコントローラーは，Kubernetes と連携することで，柔軟な資源利用を可能とする．

4. 評価実験

本稿では，前述した IoT 指向ファンクションオーケストレーションの各機能について Kubernetes, Docker, Apache Kafka を用いてプロトタイプ実装した．なお，サービスファンクションの最適配備手法については，まずは，単純なアルゴリズムとして，筆者らの先行研究である[2]を適用することとする．本アルゴリズムの詳細は省略するが，通信時間と計算時間の線形和でアプリケーションの応答時間を表現し，この値を最小化するように最適化問題を解くものである．このアルゴリズムを適用した際のアプリケーションの応答時間について，理論値と IoT 指向ファンクションオーケストレーションによる実機評価値とを比較し，その性能検証を実施する．また，その際，Kubernetes オリジナルのデプロイアルゴリズムとも比較する．

4.1. 実験機器構成

実験機器構成として，早稲田大学の研究室内に IoT デバイスとして Raspberry Pi 3 B+を 3 台，処理ノードとしてデスクトップ PC を 3 台(CPU: Intel Core i7-7700 3.60Ghz×2，Intel Core i7-9700K)用意し，処理ノードをそれぞれ Node 1，Node 2，Node 3 と定義する．また，Pub/Sub メッセージングモデルで通信するために，PC 1 台に Apache Kafka Broker をインストールし，また，さくらクラウド(CPU: Intel Xeon E5-2650 v3 2.30Ghz)には Kubernetes のマスターをインストールした．Web カメラは 3 台，それぞれの IoT デバイスに 1 台ずつ接続し,画像を取得する処理を行う(画像解像度は 640×480 とする)．なお IoT デバイスは Kafka ブローカーに Wi-Fi(2.4GHz の IEEE 802.11n)を経由して接続しているものとする．

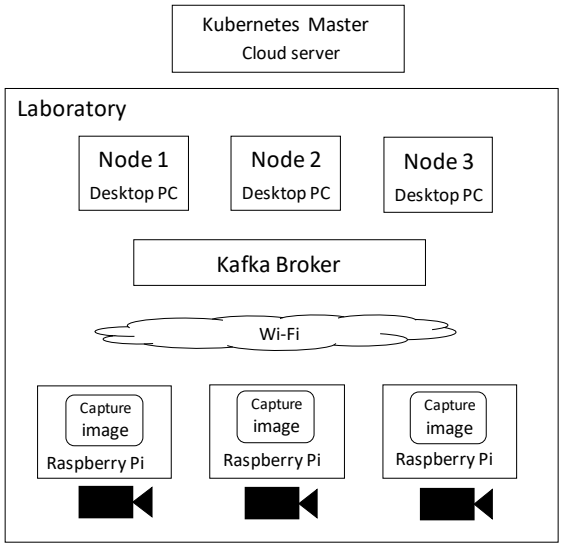

図 2　実験機器構成図.

4.2. 実験概要

本実験では，YOLO(darknet with NNPACK[13])による物体検出処理を，サービスファンクションとして Docker を用いてコンテナ実装を行った．

IoT アプリケーションは 3 つ用意し，それぞれのアプリケーションは，YOLO 処理を行うサービスファンクションを複数チェイニングすることで成り立つと仮定する．それぞれのアプリケーションは，IoT デバイスから画像を取得し，サービスファンクションで処理を行った後，ある 1 つのノード(Node3)で画像を受信する．

それぞれのアプリケーションに必要なサービスファンクション数は全て共通で 3 とする.またこのとき，それぞれのサービスファンクションには実行のために必要な CPU リソース量が与えられているものとする．3 つのアプリケーションを App1，App2，App3 とし，それぞれのチェイニングの順番を A→B→C とする．本実験で設定したそれぞれのサービスファンクションの CPU リソース量を以下の表 1 に示す．なお，各ノードが持つ CPU リソース量は共通で 7000millcores とする．

表 1　CPU リソース量 (cores)

	A	B	C
App 1	1000mill	2000mill	3000mill
App 2	3000mill	1000mill	2000mill
App 3	2000mill	2000mill	2000mill

それぞれのノードにおけるサービスファンクションの処理時間を,コンテナに割り当てる CPU リソース量を変化させ，事前に測定した．以下の表 2 にそれぞれのノードにおける YOLO 処理サービスファンクションの処理時間を示す．

表 2　各ノードにおける SF の処理時間

	1000 m	2000 m	3000 m
Node 1	0.32 s	0.25 s	0.25 s
Node 2	0.32 s	0.25 s	0.25 s
Node 3	0.25 s	0.20 s	0.20 s

上記の処理時間のデータを元に，[2]の最適配置アルトリズムを用いて，それぞれのサービスファンションを割り当てるのに最適なノードを導出する．

最適配置ノードが決定したら，その情報を利用して YAML ファイルを定義し，Kubernetes によって Docker コンテナをノードにデプロイする．デプロイした後，Docker の機能を用い，それぞれのコンテナに対応する CPU 量を割り当てる．

以上の実験環境下において，Kubernetes に用意されているデフォルトのスケジューラによってデプロイされた場合のそれぞれのアプリケーションの応答時間と，最適配置にデプロイした場合の応答時間を比較する．なお，応答時間とは，画像取得開始からデータを最後に Node3 で受信するまでの時間とする(今回は画像取得の時間はかからないものとする)．

4.3. 応答時間評価

今回はネットワークトポロジーとしてブローカーを仲介する Pub/Sub モデルと Point to Point (P2P)モデルの 2 種類に関して評価を行った．

4.3.1. Pub/Sub モデル

まずはサービスファンクション間のデータのやり取りにブローカーを仲介する Pub/Sub モデルにおいて評価を行った．それぞれの処理ノードと Kafka ブローカーの間のスループットは全て 100Mbps に設定する．

以下の表 3 に Kubernetes のデフォルトスケジューラによるサービスファンクションの配置ノード，表 4 に最適配置アルゴリズムにおけるサービスファンクションの配置ノードを示す．

表 3 デフォルトスケジューラによる SF 配置ノード

	A	B	C
App 1	Node 1	Node 2	Node 3
App 2	Node 1	Node 2	Node 3
App 3	Node 1	Node 2	Node 3

表 4 最適配置アルゴリズムにおける SF 配置ノード

	A	B	C
App 1	Node 3	Node 3	Node 1
App 2	Node 1	Node 2	Node 3
App 3	Node 3	Node 1	Node 1

図 3 にデフォルトスケジューラによる配置の場合および最適配置におけるそれぞれのアプリケーションの応答時間を示す．また，最適配置アルゴリズムによって算出されたそれぞれのアプリケーションの応答時間も示す．

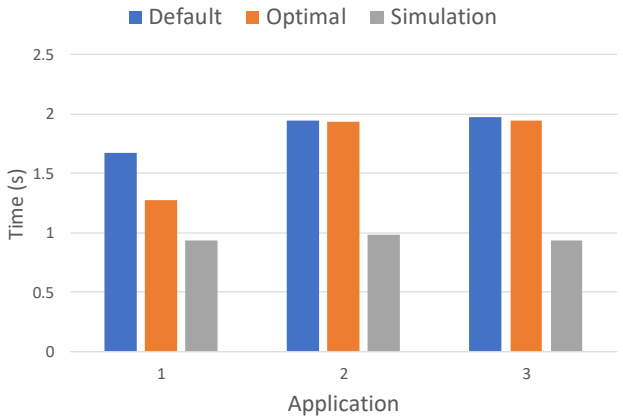

図 3 アプリケーション応答時間(Pub/Sub モデル)

図 3 より，デフォルト配置と比較して最適配置の応答時間が小さいことがわかる．しかし，App2 と App3 は最適配置アルゴリズムで算出された応答時間と比べて大きくなってしまっている．これは処理を行う上で CPU リソース等の競合が発生してしまい，事前に測定した処理時間を上回ってしまったことが原因であると考えられる．

4.3.2. Point to Point モデル

次に，各ノードがお互いに直接通信を行うことでチェイニングを行う Point to Point モデルにおいて評価を行った．各ノード間のスループットは 500Mbps とし，同一ノードにおける通信には時間はかからないものとする．

以下の表 5 に Kubernetes のデフォルトスケジューラによるサービスファンクションの配置ノード，表 6 に最適配置アルゴリズムにおけるサービスファンクションの配置ノードを示す．

表 5 デフォルトスケジューラによる SF 配置ノード

	A	B	C
App 1	Node 3	Node 1	Node 2
App 2	Node 3	Node 1	Node 2
App 3	Node 3	Node 1	Node 2

表 6 最適配置アルゴリズムにおける SF 配置ノード

	A	B	C
App 1	Node 3	Node 1	Node 1
App 2	Node 3	Node 3	Node 3
App 3	Node 2	Node 2	Node 2

図 4 にデフォルトスケジューラによる配置の場合および最適配置におけるそれぞれのアプリケーションの応答時間を示す．また，最適配置アルゴリズムによって算出されたそれぞれのアプリケーションの応答時間も示す．

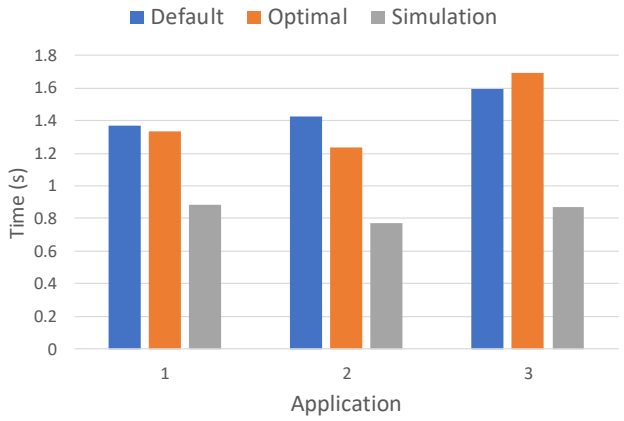

図 4 アプリケーション応答時間
(Point to Point モデル)

図 4 を見ると，App 1 と App 2 に関しては最適配置の応答時間がデフォルト配置の応答時間より小さくなっている．しかし App 3 において最適配置の応答時間がデフォルト配置の応答時間を上回ってしまっている．これも Pub/Sub モデルで述べた原因と同様で，処理時間が想定より大きくなってしまったことが原因である．

5. まとめ

本稿では，ネットワーク内において自動的に柔軟な資源の有効利用を実現する基盤構築に向けて，IoT サービスファンクションチェイニングおよび IoT 指向ネットワークファンクションオーケストレーションについて紹介した．また，Docker や Kubernetes，Apache Kafka を用いてその仕組みをプロトタイプ実装し，Pub/Sub モデルおよび Point to Point モデルのサービスファンクションチェイニングに関して性能評価を行った．

今後は，筆者らの先行研究である[3]をはじめ，最適配備アルゴリズムについて実装を進め，必要に応じて実機環境への適応を行っていく予定である．また，より現実的なアプリケーションや，より大規模なネットワークトポロジーでの評価を行っていく予定である．

謝　辞

本研究成果は，総務省委託研究 研究課題 VI「IoT 機器増大に対応した有無線最適制御型電波有効利用基盤技術の研究開発」技術課題ア「有無線ネットワーク仮想化の自動制御技術」の支援を受けている．

文　献

[1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, et al., “Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications”, IEEE Communications Surveys $ Tutorials, vol.17, no.4, 2015.

[2] 関根響，金井謙治，金光永煥，甲藤二郎，中里秀則，”IoT デバイス仮想化のためのサービスファンクション最適割り当て手法の検討”, CS 研究会，Dec. 2019.

[3] H. Kanemitsu, K. Kanai, J. Katto, H. Nakazato, “A Function Clustering Algorithm for Resource Utilization in Service Function Chaining”, IEEE 12th International Conference on Cloud Computing, July. 2019.

[4] K. Ogawa, K. Kanai, J. Katto, et al., “IoT Device Virtualization for Efficient Resource Utilization in Smart City IoT Platform”, IEEE Percom 202, Mar. 2019.

[5] Y. Sahni, J. Cao, L. Yang, “Data-Aware Task Allocation for Achieving Low Latecy in Collaborative Edge Computing”, IEEE Internet of Things Journal, vol.6, no.2, April. 2019.

[6] P. Wang, C. Yao, Z. Zheng, et al., “Joint Task Assignment, Transmission, and Computing Resource Allocation in Multilayer Mobile Edge Computing Systems”, IEEE Internet of Things Journal, vol.6, no.2, April. 2019.

[7] M. Villari, M. Fazio, S. Dustdar, et al., “Osmotic Computing: A New Paradigm for Edge/Cloud Integration”, IEEE Cloud Computing, vol.3, Dec. 2016.

[8] ETSI GS NFV 001: “Network Functions Virtualization (NFV); Use Cases”, [online]: http://www.etsi.org/deliver/etsi_gs/NFV/001_099/001/01.01.01_60/gs_NFV001v010101p.pdf

[9] A. M. Medhat, T. Taleb, A. Elmangoush, et al., “Service Function Chaining in Next Generation Networks: State of the Art and Research Challenges”, IEEE Communications Magazine, vol.55, no.2, Feb. 2017.

[10] Docker: Empowering App Development for Developers [online]: https://www.docker.com/

[11] K. Kaur, T. Dhand, N. Kumar, and S. Zeadally, “Container-as-a-Service at the Edge: Trade-off between Energy Efficiency and Service Availability at Fog Nano Data Centers”, IEEE Wireless Communications, vol.24, no.3, June. 2017.

[12] M. Amaral, J. Polo, D. Carrera, et al., “Performance of Microservices Architectures Using Containers”, IEEE 14th International Symposium on Network Computing and Applications, Sep. 2015.

[13] J. Rufino, M. Alam, J. Ferreira, et al., “Orchestration of containerized microservices for IIoT using Docker”, IEEE International Conference on Industrial Technology, Mar. 2017.

[14] Kubernetes: Production-Grade Container Orchestration [online]: https://kubernetes.io/

[15] Darknet with NNPACK [online]: https://github.com/digitalbrain79/darknet-nnpack