「マルチメディア，分散，協調とモバイル
(DICOMO2019)シンポジウム」 令和元年7月

ノード間通信の優位性評価を目的とした Fog コンピューティングテストベッドの構築に関する一検討

菊地 俊介[†1,a)] 佐々木 健[†2] 大木 裕介[†2] 松本 亮介[†1]

概要：IoT においてクラウドのリッチなコンピューティングリソースを利用する際には，現場とクラウド間の通信遅延が大きな問題となる．この問題を解決するためのエッジコンピューティングの研究開発が盛んである．筆者らは，エッジコンピューティングの実際上の導入・普及プロセスは，個々の現場にそれぞれの裁量の範囲での小さいシステムが導入され，それが徐々に有機的に結合されていく形態ではないかと考え，エッジコンピューティングにおけるノード間での相互接続性を重視した Fog コンピューティングに注目している．今回，相互接続性を考慮した Fog コンピューティングテストベッドを作成し，Fog ノード間接続の対クラウド接続に対する優位性を確認するための基礎的な性能計測を実施した．本稿では，Fog コンピューティングテストベッドのアーキテクチャ紹介と併せてその結果を報告する．

キーワード：Fog コンピューティング, テストベッド, クラウド, エッジ, 使い分け, RTT

A Study on Building a Fog Computing Testbed for Assessing the Superiority of Inter-Node Communication

Shunsuke Kikuchi[†1,a)] Ken Sasaki[†2] Yusuke Ohgi[†2] Ryosuke Matsumoto[†1]

1. はじめに

長年に渡りユビキタス，アンビエント，IoT，サイバーフィジカルシステム [1] 等と名前を変えながらも，我々の生活環境（=現場）の中にコンピュータのサポートが入ることでより便利で快適な生活を実現しようとする，次世代コンピューティング環境の研究開発が続けられてきた．

一方で，ネットワーク経由でコンピューティングリソースを利用するクラウドコンピューティングは登場以来飛躍的な発展を続けており，電力調達の優位性等から都市部から離れた場所に超大規模なデータセンターを設立し，そこからもたらされるリッチなコンピューティングリソースを利用して高度なサービス・アプリを実現する形態が一般化している．

しかし，クラウドによるリッチなコンピューティングリソースを現場に導入しようとすると，遠方にあるデータセンターとの通信遅延（レイテンシ），通信帯域，またその通信に伴う費用などが必ずしも現場で求められる要件に適合しないという問題が想定されるようになってきた．このことから，クラウドコンピューティングの形態をベースとしつつも現場（あるいは現場近傍）（=エッジ）にも限定された形のコンピューティングリソースを配置し，クラウドと現場のコンピューティングリソースを組み合わせることでそれらの問題を解決しようとする「エッジコンピューティング」が提唱され，研究開発されてきている [2].

一方筆者らは，エッジコンピューティングの次世代コンピューティング環境への適用においては，現場デバイス-エッジ-クラウドというエッジコンピューティングの縦連携だけではなく，現場におけるデバイスやシステム間の横連携もなければ高度でかつスムーズなサービス・アプリは実現できないのではないかと考え，エッジでのノード間の横連携機能を備えたエッジコンピューティングの拡張，すなわち「Fog コンピューティング」の実現に向けた実際的な検討を進めている [3][4][5]．今回筆者らは，Fog コンピューティングの実現に向けて，実際に使えるサービス・アプリ

†1 現在，さくらインターネット株式会社 さくらインターネット研究所
Presently with SAKURA Internet Research Center, SAKURA Internet Inc.
†2 現在，フリーランス
a) s-kikuchi@sakura.ad.jp

を作り出すためにはどのような課題があるのかを抽出することを目的に「Fog コンピューティングテストベッド」を設計・実装した．

本稿では，作成した Fog コンピューティングテストベッドの概要を紹介するとともに，実装したサンプルとなるアプリケーション，それを用いて実施した現場-クラウド間と現場のノード間での通信遅延の比較，それらを通じて得た Fog コンピューティングの実現に向けた課題等について報告する．

本稿は以下の構成を取る．まず第 2 節で関連研究を参照しながらエッジコンピューティングを整理し，第 3 節でエッジコンピューティングに対する問題意識と筆者らの Fog コンピューティングについての考え方について述べ，そこから，第 4 節にて今回作成した Fog コンピューティングテストベッドの設計意図を説明，続いて第 5 節で Fog コンピューティングテストベッドの実装方式とその上で稼働させたアプリケーションを紹介，第 6 節で実施した通信遅延計測とその結果について報告し，第 7 節では Fog コンピューティング実現上の課題について説明し，最後，第 8 節でまとめる．

2. 関連研究

エッジコンピューティングは，検討の母体となる組織・団体のバックグラウンドやターゲット領域の差異などから形態が多岐に渡っており，議論に際しては具体的にどの形態を対象としているかに注意が必要である．ここでは [6] を元に大別するが，a) ネットワークキャリア（特に携帯キャリア）が推進しネットワークの内部（携帯基地局）にコンピューティングリソースを設置してサービス提供あるいは利用を目指すもの（MEC[7]），b) 産業領域で工場のシステムの高度化などを想定しているもの（OpenFog[8], IIC[9] など），c) クラウドベンダーによるクラウドから現場への伸長を目指して検討されるもの（これにはプロプライエタリなものだけでなくオープン規格によるものも含む）（AWS Greengrass[10], Azure IoT Edge[11], Cloudlet[12] など）などがある．それぞれクラウドの存在を前提とした上で現場にコンピューティングリソースを提供するノード「エッジノード」を設置し，現場のコンピューティング環境の高度化のために両者を補完的に使うという点は共通であるが，想定するエッジノードの規模や性能，実行させるアプリケーション，そしてエッジノードを誰が所有・管理するのか等が異なっている．MEC はネットワークキャリアによりエッジノードを物理的に広い範囲に設置し，統一的なポリシーの下で管理，利用することを想定している．IIC は主にはエンタープライズを対象として工場や建物などにエッジノードを設置・管理・利用することを想定，クラウド主導型ははっきりした範囲の想定はないが，比較的小規模な範囲でのエッジノード設置も可能な構成である．

3. エッジコンピューティング普及における問題意識

筆者らは，前述のエッジコンピューティングの複数の形態のうちのクラウド主導型のエッジコンピューティングを前提として，次世代コンピューティング環境，つまり生活環境 (現場) の改善にコンピューティングを利用する際に，エッジコンピューティングは実際の利用・普及に際してどのような実装形態をたどり，そこにはどのような課題があるか，を調査・検討・解決することを目的に研究開発を推進している．

エッジコンピューティングの研究においては，エッジノードにジョブを割り当てる方法や動的なメッセージルーティングの方法などについての検討がよくなされている．これらは，エッジノードが単一の管理ポリシー下にありそこに複数のジョブを動的に割り当ててコンピューティングリソースを共有するモデルを前提としている．これは前述のエッジコンピューティングの形態のうちキャリア主導モデルであれば妥当な想定である．しかしながら一般的な企業や組織が IoT やサイバーフィジカルシステムの観点から現場に何らかのサービス・アプリを導入することを想定した時，費用面などから，本来目的とするジョブを実行するためのリソースを超える余剰リソースを持ったエッジノードを現場に導入することは考えづらく，また展開するエッジ領域も自身の管理下にある限られた範囲に限定されるものと考えられる．

次世代コンピューティング環境を実現するためには，一つの空間に様々なサービス・アプリが多層的に導入されていることが望ましいし，空間的にも広い範囲をカバーしている必要がある．ある特定のセンサしか使えず特定のデータしか取得できない，あるいは，この部屋ではこの制御はできるが違う部屋ではできない，という状態は望ましくない．しかしながら前述のように現場へのサービス・アプリの導入は局所的に進むことが想定されるため，目的とする環境を実現するためには，管理している企業・組織が異なる個々の現場のサービス・アプリをシームレスに連携させていくことで最終的に多層的かつ広範囲の環境を形成していくというのが現実的に想定される道筋ではなかろうか．

つまり個々の現場サービス・アプリが互いに連携可能であることが重要であり，これはすなわち，API やデータ形式（データモデル）における相互接続性がある，ということが重要になると考えられる．

また，サービス・アプリ間の連携方法を考えた時に現状で一般的なのは，クラウドに設置したサーバにおいて API を公開しそこを通じて互いに通信・データ交換を実施するという方法である．しかし，現場のサービス・アプリ間での連携を想定する場合，そもそものエッジノードの導入の理由が現場での処理能力の高度化でありすなわちクラウド

までの往復の通信遅延やデータ量の多さなどの問題の解決のためであったことを考えると，クラウドを経由した連携では目的を満たせないと考えられる．したがって現場のサービス・アプリ間の連携では，クラウドではなくエッジ内での相互接続が求められると考えられる．

これらの点から筆者らは，エッジコンピューティングにおいてエッジノード上でのジョブの間の相互接続を可能にすることが必要であり，エッジコンピューティングに相互接続性機能を追加した拡張を「Fog コンピューティング」と定義している[*1]．そして Fog コンピューティングの実現，言い換えれば個々のエッジサービス・アプリの相互接続の実現における，実装形態や課題の調査・検討・解決をより具体的な問題意識と設定している．

これに加えて現実的なエッジコンピューティングの普及を想定した際の問題意識として，クラウドと比較してエッジノードの不安定さを考慮して何らかの高可用性対応が必要と考えられることも問題意識の一つである．

これらをまとめると以下の様になる．

(1) 単一管理ポリシー配下でのエッジノード展開ではなく局所的なサービス・アプリ，エッジノードの集合になることを想定し，異なるサービス・アプリ間での連携を可能にする

(2) サービス・アプリ間連携をクラウドではなくエッジノードで実施可能にする

(3) エッジノードの不安定さを考慮し，エッジ実行ジョブの可用性を高める

4. Fog コンピューティングテストベッドの設計意図

前節で述べた問題意識をもとに，筆者らは以下のような特徴を組み込んだ Fog コンピューティングテストベッドを設計した．なお以下の各項目は，前節のそれぞれの問題意識に対応している．

4.1 NGSI および FIWARE データモデルの採用

Fog コンピューティングテストベッドではサービス・アプリ間連携のための API とデータモデルに，欧州を中心に開発されスマートシティやスマートビルディング等のシステムにおいて使用が始まっている NGSI - Next Generation Service Interfaces[13] と，FIWARE データモデル [14] を採用した．NGSI はデータ交換を担うコンテキストブローカーをハブとした Pub/Sub 型通信モデルで，仕様はオープンであり，OSS の実装も存在している．FIWARE[15] は NGSI の標準化を推進する標準化組織であり，同時にそれを実装したソフトウェア群の名称でもある．NGSI/FIWARE を採用することで，同じ方式に準拠しているサービス・アプリ間や FIWARE の既存ソフトウェア資産との相互接続が可能になる．既存のソフトウェア資産とは例えば，センサデータの一般公開（マーケットプレイス）機能コンポーネントや，データのグラフ表示や地図上での表示を可能にするコンポーネントなどである．

スマートシティ・スマートビルディングの検討は現在，特に自治体等が保有する公共データ等の公開などに留まっているが [16][17]，本来は現場の状態把握や制御を目指すものであり現場の高度化を目指す次世代コンピューティングと本質的には目指すものは同じであるため，その規格等を採用することには合理性があると考えている．

NGSI 及び FIWARE データモデルは，仕様を理解しそれに従ってメッセージングを使えるようになるまでに若干の学習コストが必要であるが，ベースは REST 形式の通信プロトコルであり理解は容易である．本テストベッド用に独自の通信仕様を定義することも可能であるが結局学習コストが必要であることを考えると，汎用性の高い方式を採用することがより良いと考えた．

4.2 データプレーンのメッセージングでクラウド/エッジ折返しを併用する

NGSI ではコンポーネント間のデータ交換（メッセージング）はコンテキストブローカーと呼ばれるハブとなるコンポーネントを経由することで実施される．各コンポーネント間が直接通信するのではない．Fog コンピューティングテストベッドではコンテキストブローカーをクラウド上およびエッジノードの双方に配備し，サービス・アプリがそれぞれを併用可能な構成にしている．これにより，低遅延でのサービス・アプリ間の通信も可能であり，同時に，クラウド上に設置されたリッチなコンピューティングリソース（計算能力や永続化ストレージ）の利用も可能になる．

4.3 ノード管理のためのコントロールプレーンの導入

Fog ノード（Fog コンピューティングテストベッドにおけるエッジノード）は現場に設置されるノードであり，小型・軽量であることを想定としている．このため所謂 PC のようなキーボードや画面などを持たず，その管理はリモート接続によるもの（=ヘッドレス運用）となる．また現場に多数設置され，かつ個々のノードは長期の安定可動を求められない想定である．このため，Fog ノード (群) のリモートでの管理のための仕組み（=コントロールプレーン）が必要である．

Fog コンピューティングテストベッドのコントロールプレーンには大きく 2 つの機能を持たせる必要があると考えている．1 点目はノードへのジョブの配備機能，もう 1 点

[*1] なお Fog コンピューティングの定義も使用者によって曖昧で定まっていないが，このエッジにおける横連携は OpenFog Consortium における Fog コンピューティングの定義 [8] にほぼ準拠する．

はノード自身の状態把握・管理機能である．

ジョブ配備機能については，前項にて現実的なエッジコンピューティングでは複数ジョブの動的配備は考えにくいと述べたが，単一ジョブを利用するだけの想定であっても，ジョブ（を構成するプログラム）のアップデートを容易に実施するためのジョブデプロイ機能は必要であろう．またサービス・アプリ間の通信のためのデータプレーンを実現するためのコンテキストブローカーコンポーネント自身も一つのジョブであるため，その配備ができる必要がある．

また後者については，コントロールプレーンは各 Fog ノード上に存在しアドレスやその稼働状態を把握し，動作不調等がある場合には再起動や他の Fog ノードへの機能振替等の管理操作を実施する．

これら各ポイントを纏めたシステム構成図（アーキテクチャ図）を図 1 に示す．

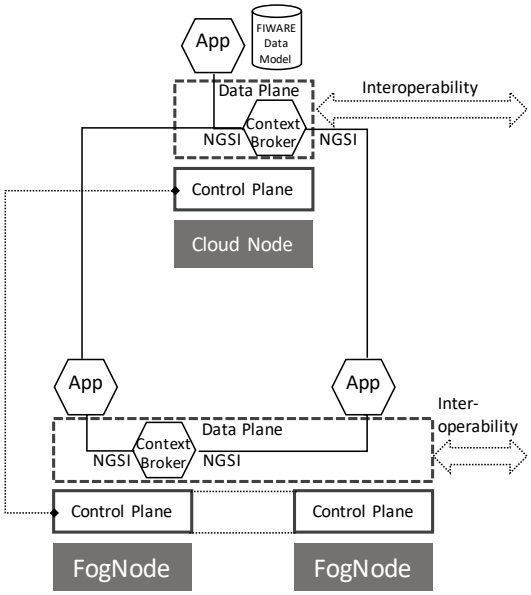

図 1 テストベッドシステム構成

なお，Fog コンピューティングとしては本来セキュリティ要件の考慮も重要であるが，これについては今後の検討予定としている．

5. テストベッドの実装方式

本節では，作成した Fog コンピューティングテストベッドの具体的な実装について説明する．

Fog コンピューティングテストベッドは，前節までで説明したようにデータプレーンとコントロールプレーン，さらにそれらを稼働させる物理的なノード，の 3 パートによって構成される．それぞれについて説明する．

5.1 ノード構成

- クラウド側

  クラウドノード VM1 台（さくらのクラウド上）1 コア 2GB メモリ

  SSH Proxy ノード VM1 台（さくらのクラウド上）1 コア 2GB メモリ

- エッジ側

  Fog ノード Raspberry Pi 3B+ 4 台，入力デバイスとして赤外線人感検出 (PIR) センサー，出力デバイスとして LED アレイを搭載．

  Fog ノード (補) Intel Compute Stick BOXSTCK1A8LFC 1 台 Atom コア 1GB メモリ

SSH Proxy ノードは実装上必要なノードで，FW 内に設置する Fog ノードにクラウド（グローバルアドレス空間）からアクセス可能にするために設定するの SSH トンネルエントリーポイントを提供する．Fog ノード（補）は，データプレーンを構成するコンテキストブローカーが ARM プロセッサでは動作しないため，Intel x86 プロセッサを搭載したノードとして用意したものである．

5.2 データプレーン

Fog コンピューティングテストベッドでは，データプレーンとして実際にメッセージ交換を担うコンテキストブローカーと，サービス・アプリを実現するプログラムは，同じユーザランドのジョブとして扱われる．

最終的に実現したい現場改善のためのサービス・アプリは，クラウド上で実行される「クラウドプログラム」と，エッジの Fog ノード上で実行される「Fog プログラム」の組み合わせにより構成される．両者を合わせて「Fog アプリケーション」と呼ぶ．なお，各プログラムは Docker コンテナとして構成する．

Fog アプリケーションを構成するクラウドプログラムはクラウドノードに，Fog プログラムは Fog ノードにそれぞれ配備される．Fog アプリケーションのプログラムとは独立に，クラウドノード，Fog ノード（のうちの 1 台）にはコンテキストブローカーが配備・実行されている．Fog アプリケーションの両プログラムはそれぞれ何らかのイベント発生（センサの検出など）を契機に，他の Fog プログラムやクラウドプログラムとコンテキストブローカーを経由してメッセージを交換し，目的の処理を実行できる．

Fog アプリケーションのクラウド・Fog ノード上の各プログラム間の通信は，NGSI 準拠の Pub/Sub 型通信モデルにより実施する．

各プログラムは起動時に自分自身をコンテキストブローカーに登録する．これはコンテキストブローカーに対する Subscription 設定の登録として実行される．この Subscription 登録を実施しないと，当該プログラムは他のプログラムからのメッセージを受信することができない．

Fog アプリケーションのプログラムモデルについて図 2 に示す．

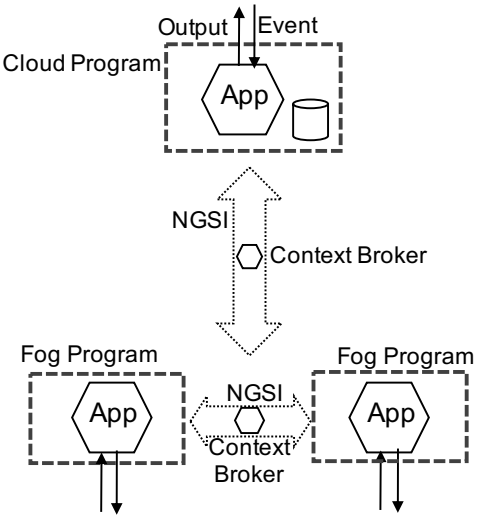

図 2 Fog アプリケーションプログラムモデル

本テストベッドの特徴として，コンテキストブローカーをクラウド側とエッジ側に 2 台配置することで，プログラム間のメッセージング経路をクラウド側とエッジ側の 2 本構成していることが挙げられる．これにより Fog アプリケーションは，低レイテンシでの Fog ノード間通信と，クラウド側でのリッチなコンピューティングリソースの使用や外部のシステムとの連携を自由に使い分けることが可能になっている．

なお，コンテキストブローカーのソフトウェアとしては OSS の Orion[18], [19] を使用している．

5.3 コントロールプレーン

コントロールプレーンは 2 つの機能（ジョブ配備機能とノード管理機能）を持つ必要があるが，本テストベッドではこれらは Kubernates（実際にはミニセットの k3s[20]）により実現している．Fog ノード群から任意に選択した 1 台を k3s マスターノードとして実行させる．k3s マスターノードは自身を含めた Fog ノード群を kubernates Nodes として管理下に置く．クラウド側コントロールプレーンから k3s マスターノードに対して Fog プログラムのコンテナイメージの配備を指示することで，k3s マスターノードが各 Nodes にコンテナイメージを配備する．併せて，k3s マスターノードは各 Nodes の生死状態を把握し，ノード障害が発生した場合には，復帰したときに Fog プログラムを再登録する．

クラウド側コントロープレーンは Kubernates ではなく単一の管理プログラムで，クラウドノード自体にクラウドプログラムを配備する．

コントロールプレーン構成について図 3 に示す．

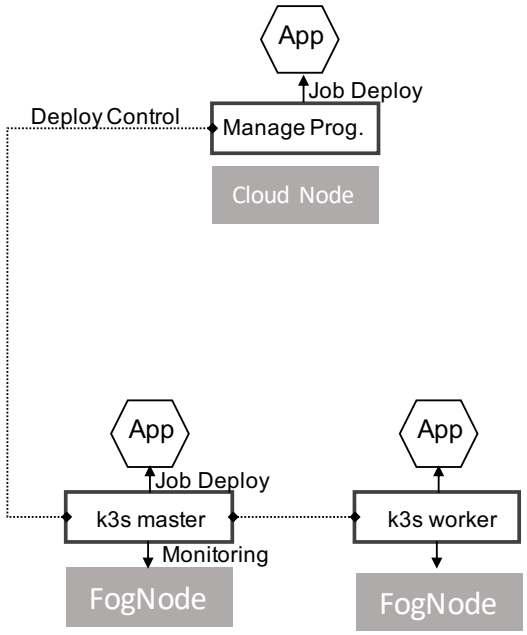

図 3 コントロールプレーンの構成

現在 k3s マスターノード自体の選定・設定は静的だが，クラウド側コントロールプレーンからの設定自動化に対応させる予定である．

5.4 アプリケーション

本検討にあたっては，Fog コンピューティングテストベッド上で稼働させるサンプルアプリケーションとして，「光る通路案内」を作成した．人感センサー及び LED が搭載された Fog ノード群を居室廊下等に設置し，人が来たことを検出した場合には LED を光らせることで他の通行者に知らせる，というものである．ある Fog ノード上でのイベント (人物検出) が他の Fog ノード上に高速に伝搬していく形態を持ったアプリケーションを想定してこのようなサンプルを作成した．現状は Fog ノード間での連携だけであるが，データをクラウド側に送信・蓄積し機械学習等により分析させる等で，通行者通知だけでなく進行・停止指示や迂回路提示などの交通案内動作をさせることも想定している．

サンプルアプリケーションの動作について図 4 に示す．また Fog ノードの実際の様子を図 5 に示す．

6. テストベッドアーキの効果計測

本テストベッドでは，エッジ側での高速なノード間連携とリッチなクラウドリソースの利用やクラウド経由での外部連携の実現を想定して，データプレーンを 2 つ構成している．まずは，この 2 つのデータプレーンの有効性を検証する必要がある．このため，2 つのデータプレーンについて，基礎的な通信遅延とデータ量に関する計測を実施した．

- Fog ノード上でデータ参照リクエストをコンテキスト

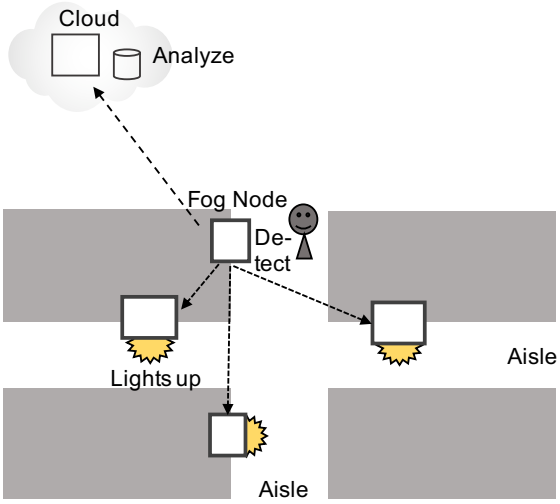

図 4 Fog アプリケーションの構成（光る通路案内）

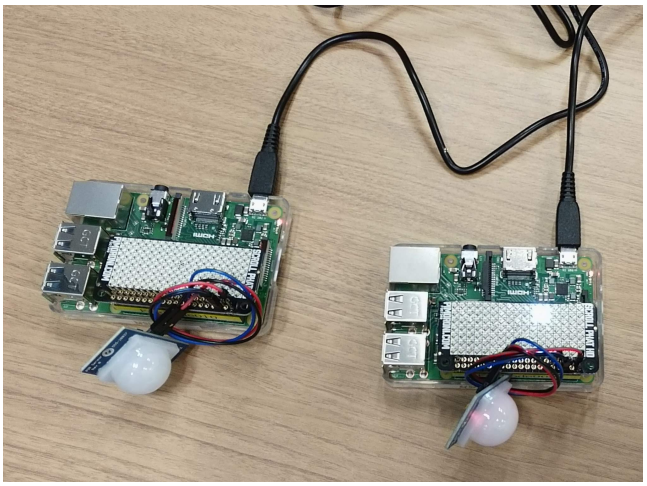

図 5 実際の Fog ノードの写真

ブローカーに送信し，それが折り返ってくるまでの時間を計測

- 送信するコンテキストブローカーをエッジ側，クラウド側と切り替えてそれぞれの経過時間を計測
- Fog ノードおよびエッジ側コンテキストブローカーの設置場所はさくらインターネット東京オフィス内
- クラウド側コンテキストブローカーの設置場所はさくらのクラウド (石狩) データセンタ
- コンテキストブローカーでの処理遅延を推定するため，各ケースにおいて処理内容の異なるデータ参照リクエストと，それらと別に ping による応答遅延時間も計測

計測結果について表 1 に示す．計測内容における RTT(nop) はコンテキストブローカーにおいて実際のデータベースアクセスを伴わないバージョン取得コマンド，RTT(op) はデータベースアクセスを伴う subscription 登録内容取得コマンドに対応している．

クラウドとの間では RTT 20[msec] と，エッジ折返しの場合（13.97 [msec]）のおよそ 1.4 倍の応答遅延が掛かっていることが計測された．したがって，クラウド折返しではなくエッジ内折返しを利用したほうが高速にノード間連携が実施できることが確認できた．

しかしながらクラウド折返しとエッジ内折返しの差は僅差とも言える．これは，エッジ内折返しにおいて Fog ノードが接続している無線 LAN での折返しの遅延が大きいことや，ローカルコンテキストブローカーの処理時間が大きいこと，また逆に，クラウドまでの通信遅延が小さいこと（実際上弊社内に閉じた経路のため有利）などが考えられる．

もう一点，エッジコンテキストブローカーとの通信においては，計測時にジッタが大きかったことも特徴として挙げておく．ゆらぎの原因はネットワークに起因するものと，ローカルコンテキストブローカー自体の双方に存在していると見られる．Fog ノードからローカルコンテキストブローカーまでの ping 計測の結果について参考までに記載しておく．`(rtt min/avg/max/mdev = 3.335/6.324/9.465/2.201 ms)`

7. Fog コンピューティングテストベッドの今後

本節では，本 Fog コンピューティングテストベッドの設計及び実装を通じて見えてきた課題と今後の展開について説明する．

本テストベッドの設計と実装の過程で明らかになった課題としてまず挙げられるのは，エッジ環境側が FW 下のプライベート空間にあることで，直接接続やシームレスな制御・管理ができないということである．具体的には以下のような点が明らかになった．

1) データプレーンにおいてクラウドからエッジへは直接接続できない．現在の実装では SSH トンネルを構築して接続可能にしているが，ここはセキュリティ観点から検討が必要であろう．2) データプレーンを構成するための Subscription 設定がクラウド側とエッジ側で 2 重管理になっている．クラウド側のコンテキストブローカーに投入する Subscription 設定は，SSH トンネルの外側アドレス（グローバルアドレス）を指定するのに対し，エッジ側のコンテキストブローカーに投入する Subscription 設定は，LAN 内アドレスを指定するため，設定内容を共有できない．3) クラウドとエッジ間でコントロールプレーンが途切れている．エッジ側の Fog ノード群の管理のために Kubernetes(k3s) を用いているが，そのマスターノードも Fog ノード上にあり，クラウド側からマスターノードに対する指示は独自プロトコルに依っている．

表 1 Fog ノードからのエッジ内折返し，クラウド折返しの計測結果 (単位はいずれも [msec])

ターゲット	RTT(nop)	RTT(op) (Δ)	ping (δ)	推定処理時間 ($\Delta - \delta$)
エッジコンテキストブローカー	8.90	13.97	6.32	7.65
クラウドコンテキストブローカー (石狩)	20.01	20.26	19.71	0.55

また別の課題として設定の自動化が不十分な点も挙げられる．設定自動化はテストベッドとしての使い勝手に直結し，また可用性の面からも重要である．また最終的にはFog コンピューティングの普及に大きく影響するので十分な検討と対応が必要と考えている．a) データプレーンの実体であるコンテキストブローカーの配備が現状手動である．b) エッジ側コントロールプレーンの k3s マスターノードの設定は現在静的構成である．

特にエッジ側の自動化は，上述の FW 環境下での接続に関する問題と関係するため，今後早急に検討を進めたいと考えている．

使い分けは，求められるレイテンシ，必要帯域，通信コストや，将来的に想定される動的なプログラムデプロイ等を考慮して，何らかの判定基準により自律的に決定されることが望ましい．現時点では，そもそもどのようなアプリケーション想定でどの程度の遅延が求められる/許容されるのか，またデータ通信量がどの程度になるか，などが明らかになっておらず判定基準を議論できる段階に至っていない．この点から，メッセージングの基礎的なデータを収集すべく実際に計測を実施した．この内容については第 6 節で説明した．

8. まとめ

本研究に置いて筆者らは，サイバーフィジカルコンピューティングなど次世代コンピューティング環境の実現に向けて，エッジコンピューティングを現場環境に投入することを前提に，システム・アプリ間の連携，局所的な小さいシステム間の相互接続を重視し，それに対応した Fog コンピューティングテストベッドを設計・開発した．

筆者らが開発した Fog コンピューティングテストベッドでは，現場エッジの Fog ノード，クラウド上でクラウドノードを組み合わせた Fog アプリケーションを稼働させることができる．またデータプレーンとしてクラウド上に配備したコンテキストブローカー，Fog ノードに配備したコンテキストブローカーによる２経路のメッセージ交換が可能である．また，それらアプリケーションを配備・稼働させるため，およびノード管理ための，コントロールプレーンを構成・(部分的に) 実装し，その必要要素を抽出した．

今回は，テストベッドのアプリとして人感センサー反応を契機にした光る道案内サービスを試作し，人検出からデバイスの表示器が光るまでの応答速度について，クラウド経由，エッジ内経由での差異について評価した．その結果として，現場デバイスを連携させたシステムにおける設計指標を得ることができた．

また，テストベッドの設計・実装を通じ，FW 下にあるエッジ環境をクラウド側とシームレスに接続させる点やそれを考慮した上での設定自動化方法についての課題点を得た．

今後はこのテストベッド上で様々なアプリ・サービスを実装・実行させ相互接続性についての知見を深めながら，同時に上述の課題解決方法についての検討・実装，さらなる評価を進めていく予定である．現場に溶け込んだ次世代コンピューティング環境におけるより一般的な機能要件・性能要件の抽出とその解決手段の開発を進める予定である．

参考文献

[1] 岩野和生，高島洋典：サイバーフィジカルシステムと IoT (モノのインターネット) 実世界と情報を結びつける，情報管理， Vol. 57, No. 11, pp. 826–834 (2015).

[2] Satyanarayanan, M.: The emergence of edge computing, *Computer*, Vol. 50, No. 1, pp. 30–39 (2017).

[3] さくらインターネット：さくらインターネット、フォグコンピューティングを推進する「Open-Fog Consortium」に加入, https://www.sakura.ad.jp/information/pressreleases/2016/04/13/90130/.

[4] 菊地俊介：Fog コンピューティングのご紹介，https://research.sakura.ad.jp/2017/12/08/intro-fog/.

[5] さくらインターネット：さくらインターネット、「Fog World Congress 2018」で実施するフォグコンピューティングの日中台相互接続デモに「さくらのクラウド」を提供, https://www.sakura.ad.jp/information/pressreleases/2016/04/13/90130/.

[6] 飯田勝吉：エッジコンピューティング研究開発の現状と今後の課題 (インターネットアーキテクチャ)，電子情報通信学会技術研究報告= IEICE technical report: 信学技報， Vol. 117, No. 187, pp. 25–30 (2017).

[7] ETSI: Multi-access Edge Computing (MEC), https://www.etsi.org/technologies/multi-access-edge-computing.

[8] OFC: OpenFog Consortium, https://www.openfogconsortium.org/what-we-do/#definition-of-fog-computing.

[9] IIC: Industrial Internet Consortium, https://www.iiconsortium.org/.

[10] AWS: AWS IoT Greengrass, https://aws.amazon.com/jp/greengrass/.

[11] Microsoft: Azure IoT Edge, https://azure.microsoft.com/ja-jp/services/iot-edge/.

[12] Satyanarayanan, M., Bahl, P., Caceres, R. and Davies, N.: The case for vm-based cloudlets in mobile computing, *IEEE pervasive Computing*, No. 4, pp. 14–23 (2009).

[13] Jose Manuel Cantera Fonseca, Fermin Galan Marquez, T. J.: FIWARE-NGSI v2 Specification, http://telefonicaid.github.io/fiware-orion/

api/v2/stable/.
[14] FIWARE: FIWARE DATA MODELS, https://www.fiware.org/developers/data-models/.
[15] FIWARE: FIWARE, https://www.fiware.org/.
[16] 国土交通省：スマートシティの実現に向けて【中間とりまとめ】, http://www.mlit.go.jp/common/001249774.pdf.
[17] 総務省：総務省におけるスマートシティの展開について, https://www.kantei.go.jp/jp/singi/tiiki/kokusentoc/supercity/dai2/shiryou2.pdf.
[18] telefonicaid: An implementation of the Publish/Subscribe Context Broker GE, providing NGSI interfaces., https://github.com/telefonicaid/fiware-orion.
[19] Telefonica: Welcome to Orion Context Broker., https://fiware-orion.readthedocs.io/en/master/.
[20] LABS, R.: Lightweight Kubernetes, https://k3s.io/.