Vol.2020-DPS-182 No.54
Vol.2020-CSEC-88 No.54
2020/3/13

アプリケーション処理連接基盤における ネットワーク・計算資源管理機構

林 和輝[1,a)]　渡邊 大記[1,b)]　佐藤 友範[1,c)]　近藤 賢郎[1,d)]　寺岡 文男[2,e)]

概要：第 5 世代移動通信方式 (5G) の導入や Edge/Fog/Cloud Computing の登場により，ヘテロジニアスなネットワーク・計算資源環境における効率的なサービス展開が求められる．筆者らはアプリケーション処理 (AF) の連接によってアプリケーション (Chained-AF Application) を構築する AFC (Application Function Chaining) を提案している．本稿は AFC における MANO (Management and Network Orchestration) の機構を提案することを目的とする．AFC 全体のアーキテクチャを再設計することで，プラットフォームのマルチドメイン化に加え，認証・認可・アカウンティングの仕組みを実現した．

Management and Network Orchestration Mechanism for Application Function Chaining Infrastructure

1. はじめに

現在，次世代の無線通信システムである第 5 世代移動通信方式 (5G) の導入が進められている．5G コアネットワークは eMBB (enhanced Mobile Broadband)，URLLC (Ultra-Reliable and Low Latency Communications)，mMTC (massive Machine Type Communications) という 3 つの通信サービスを要件としている．eMBB であればより高精細な VR (Virtual Reality) や AR (Augmented Reality)，URLLC であれば自動運転支援や遠隔医療，mMTC であれば大量のセンサを用いた IoT (Internet of Things) といったサービスが普及すると考えられる．一方で，ネットワーク上でデータを分散処理する Edge/Fog/Cloud Computing の技術により，ヘテロジニアスなネットワーク資源・計算資源が存在する環境において効率的なサービス展開が可能となる．ヘテロジニアス環境ではネットワークの特性等を考慮したファンクション配置が非常に重要であり，低遅延を要求する処理はエッジサーバで実行し，高負荷が予想される複雑な処理はクラウドサーバで実行するといった配置が考えられる．SFC (Service Function Chaining) [1] は，NF (Network Function) の連接によってネットワークサービスを提供する技術である．サービスのセマンティクスを理解できるのはあくまでも通信事業者であり，利用者の位置に応じたファンクションの配置はできない．

AFC (Application Function Chaining) [2] は，通信と計算の融合に向けたアプリケーション処理連接基盤である．図 **1** に従来の AFC [3] におけるアーキテクチャを示す．通信事業者の判断で通信路上のパケットに適切な処理を適用していく SFC とは異なり，AFC ではアプリケーション開発者が主体となってアプリケーション処理 (AF) を連接 (Chaining) することで 1 つのアプリケーション (Chained-AF) を作成する．Chained-AF User の視点では，AFC のプラットフォーム全体が Gate Node を端点とする巨大な計算機のように見える．

従来の AFC は AF の連接によって Chained-AF を作成する仕組みに焦点を当てており，以下の課題が存在した．

- AFC における MANO (Management and Network Orchestration) の機構が定義されていない．
- AFC プラットフォームが複数のドメイン (事業者) によって管理されることを想定していない．

[1] 慶應義塾大学大学院理工学研究科
Graduate School of Science and Technology, Keio University
[2] 慶應義塾大学理工学部
Faculty of Science and Technology, Keio University
[a)] gordon@inl.ics.keio.ac.jp
[b)] nelio@inl.ics.keio.ac.jp
[c)] glue@inl.ics.keio.ac.jp
[d)] latte@itc.keio.ac.jp
[e)] tera@keio.jp

Vol.2020-DPS-182 No.54
Vol.2020-CSEC-88 No.54
2020/3/13

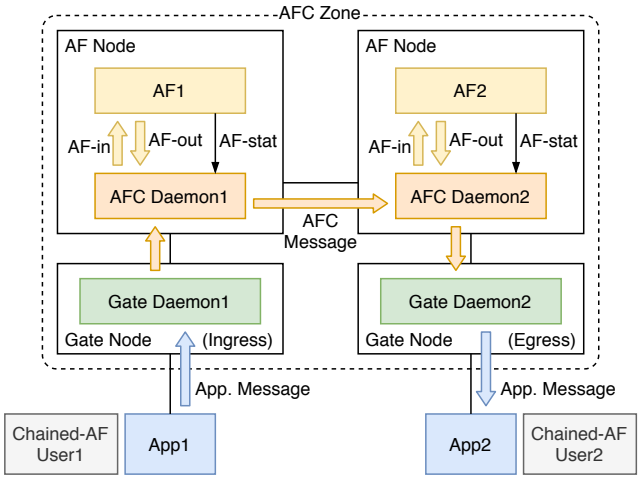


図 1: 従来の AFC におけるアーキテクチャ.

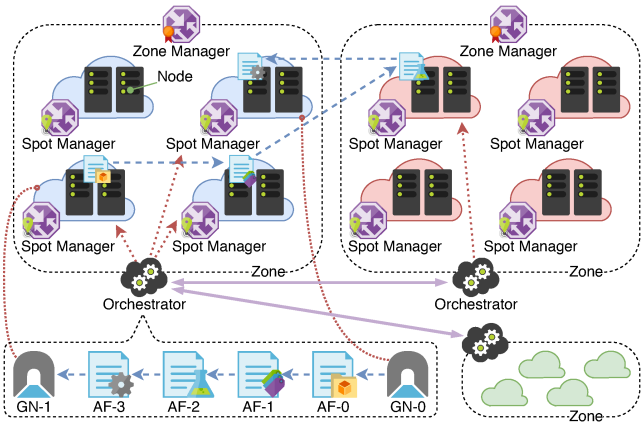


図 2: AFC MANO の概要.

本稿では AFC の MANO 機構 (AFC-MANO) を含む AFC 全体の新アーキテクチャを提案する．図 **2** に AFC-MANO の概要を示す．AFC は地理的な集合である Spot と，管理ドメインである Zone によって構成される．Zone Manager は Spot Manager 経由で各 Node の資源情報を収集する．Orchestrator はその資源情報をもとに，Chained-AF 内の各 AF の最適な Spot 配置を計算する．新アーキテクチャでは Zone が複数存在するマルチドメイン環境も想定している．また，Orchestrator による AF の配置手法も提案する．AF の Spot 配置問題を数理最適化モデルとして定義し，最適化手法とヒューリスティック手法の 2 種類を実装・評価した．

2. 関連研究

2.1 ETSI NFV MANO

専用ハードウェアで提供されていたネットワーク機能をソフトウェアで実現して汎用ハードウェア上で動作させるという考えを NFV (Network Functions Virtualization) と呼ぶ．また，NFV 環境を構築するために要求される管理機能や調整機能のことを MANO (Management and Network Orchestration) と呼ぶ．図 **3** に ETSI (European Telecommunications Standards Institute) が提案する NFV-MANO のアーキテクチャ [4] を示す．NFVO (NFV Orchestrator)，VNFM (VNF Manager)，VIM (Virtualized Infrastructure Manager) の 3 つによって MANO 機構が構成されている．ETSI は NFVO 同士の連携機構を標準化 [5] することによって NFV-MANO をマルチドメイン環境に対応させた．DASMO [6] では ETSI の NFV-MANO に ISM (In-Slice Management) を統合したアーキテクチャを提案している．ネットワークスライスの内部を管理するインタフェースを導入し，サードパーティによるスライス管理を可能にした．MEC-NFV [7] では VAF (Virtual Application Function) という概念を導入することによって，ETSI の NFV と MEC (Multi-access Edge Computing/Mobile Edge Computing) の統合を図っている．アプリケーション指向かつ QoS (Quality of Service) ベースの MEC サービスを実現するが，MEC における AF は Control-Plane に属しており，AFC のようにパケットのデータを処理することはない．ETSI の NFV-MANO やそれを拡張した手法では，NFVI (NFV Infrastructure) によって仮想化された資源を VIM で管理する．MANO が管理できるのは既に仮想化された資源のみであり，ヘテロジニアスな分散環境においてネットワーク特性を十分に考慮したファンクション配置は難しい．AFC における管理ドメインは地理的に分散した複数のデータセンタを跨いでおり，NFV-MANO を適用することはできない．

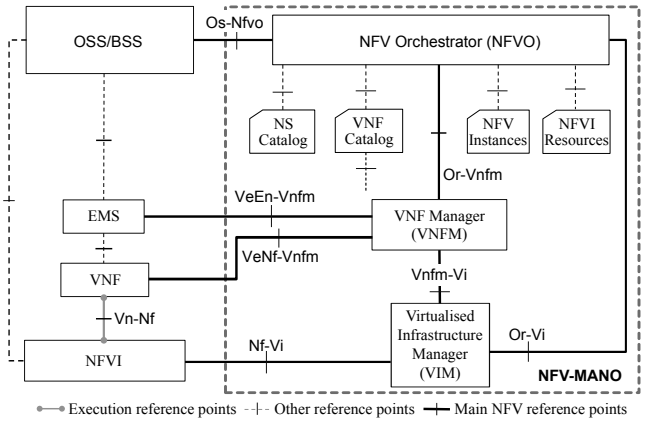


図 3: NFV-MANO アーキテクチャ [4].

2.2 Edge/Fog コンピューティング

FogTorchII [8] は，複合的なアプリケーションをヘテロジニアスな Fog 環境にデプロイする際の手法を提案している．QoS，インフラストラクチャ，アプリケーションをモデル化し，2 段階の探索によって適切なデプロイ場所の候補を複数出力する．Fog at Edge [9] は Edge コンピューティングプラットフォームである．フローベースプログラミン

グツール Node-RED [10] を分散環境向けに拡張した DNR (Distributed Node-RED) を提案している．FogTorch や Fog at Edge でデプロイされるアプリケーションは，AFC における Chained-AF と同じく Data-Plane に属している．Edge/Fog 環境をプラットフォームとして提供する際，データの入力位置やパラメータとしてユーザの位置を考慮できる一方で，AAA (Authentication, Authorization, and Accounting) の機構は定義されていない．

2.3 NFV における VNF の配置問題

NFV における VNF の配置問題は大きく 2 種類に分類できる．1 つは，独自のアルゴリズムを用いる手法である．動的計画法を用いて SFC をプロビジョニングする手法 [11] では，プロバイダの収益を最大化する VNF 配置を計算する．分割統治法とメモ化を用いた再帰的なアルゴリズムにより，多項式時間での実行を可能とした．遺伝的アルゴリズムを用いて VNF を管理する手法 [12] は，計算資源とネットワーク資源の両方を考慮している．DFS (Depth-First Search) でランダムに選択した後，GA (Genetic Algorithms) の交叉と突然変異を模擬したアルゴリズムによって NF の配置を決定する．もう 1 つは，問題を適切にモデル化し，ソルバーによって解を求める手法である．遅延を考慮して VNF を配置する手法 [13] では，リソース消費を最小化しつつ一定の End-to-end 遅延を満たし，SLA (Service Level Agreement) を侵害しない VNF 配置を決定する手法を提案している．数理最適化モデルに基づいて VNF を配置する手法 [14] では，NF 配置とチェイニングの問題を定式化し，ILP (Integer Linear Programming) によって解を算出している．巨大なインフラストラクチャを用いた場合の計算時間増大に対応するため，ヒューリスティックな手法も提案している．

AFC の MANO は遅延や帯域に限らず AF や Chained-AF のポリシまで考慮する必要がある．独自のアルゴリズムに基づいて計算し，多項式時間で最適解を求めるのは難しい．AF の配置問題の解決にソルバーを用いることで，多様なアプリケーション要求に応えられるようにした．

3. AFC の新アーキテクチャ

アプリケーション開発者は，クラウドコンピューティングサービスを利用することで，インフラストラクチャの構築・管理コストを削減することができる．クラウドコンピューティングサービスは単一の管理ドメインによる中央集中型のシステムであり，利用者は国ごとに数個あるデータセンタ単位でしかデプロイ場所を選択できない．AFC では，ISP (Internet Services Provider) のようなネットワーク事業者が PoP (Point of Presence) に小規模な計算資源を持ち，アプリケーション開発者から透過的に扱える時代を想定している．従来はネットワーク事業者向けであった MEC のエッジ計算資源をサードパーティへ開放する動きにより，日本であれば都道府県レベルで小型のデータセンタが存在するようになると考えられる．このような小型のデータセンタを，AFC における Spot と定義する．複数の Spot を利用したアプリケーション作成するには，Spot を統括する管理ドメインが必要である．AFC ではその管理ドメインを Zone と定義する．ISP のようなネットワーク事業者は Zone と契約し，自身の計算資源の一部を Spot として提供する．データセンタを管理・運営する事業者が Zone と契約し，計算資源を Spot として提供することによって収益を得ても良い．同様にして，AF や Chained-AF の開発者も Zone と契約し，使用状況に応じて Zone から収益を得ることになる．一方で利用者は Chained-AF の使用状況に応じて Zone から課金される．優れた AF を所有する Zone は，他 Zone の Chained-AF の一部として AF を提供することもある．このように，中央集中型のクラウド事業者に対応する形で分散処理型の AFC Zone 事業者が存在するため，マルチドメインの仕組みが必要である．AFC は 5G の高機能な通信サービスを想定した Edge/Fog/Cloud プラットフォームと言える．

3.1 アーキテクチャの全体像

図 **4** で示すように，AFC の新しいアーキテクチャは Data-Plane，Control-Plane，MANO-Plane の 3 プレーンに AAA 基盤を加えた構造になっている．ETSI の NFV-MANO は資源情報の管理から VNF のライフサイクル管理までを一括して扱う．一方 AFC では Spot が地理的に分散しており，各 Node の計算資源情報が AF の配置計算にとって非常に重要であることから，MANO-Plane という資源情報収集に関わる専用のプレーンを設けている．代わりに Control-Plain が AF や Chained-AF のライフサイクル管理を担う．Network Monitor や Node Monitor は Control-Plane と Data-Plane に関係なくネットワーク資源や計算資源の情報を収集する．AF や Chained-AF のライフサイクルに関する情報が Monitor に通知されることはなく，処理の遅延や帯域のオーバーヘッドが少ない．AF の配置先が決定した後の処理は Orchestrator から指示された AFC Controller が担う．また，アプリケーション開発者・利用者を考慮するため，パケットのデータを処理する Data-Plane をアーキテクチャ上に定義している．Data-Plane には Zone または Spot ごとの集約機構が存在しない．AF 間のデータ転送は AFC Daemon が担い，Zone を跨いだ場合も AFC Daemon 間の通信は IP アドレスとポート番号の組によって識別される．IP ネットワークのルーティングに基づいたデータ転送となるため，AFC によるオーバーヘッドが少ない．

新アーキテクチャでは OSS/BSS (Operation Support System/Business Support System)，Orchestrator，AFC Controller のそれぞれが Zone 間で通信することにより，複数の管理ドメイン (Zone) を考慮した．認証・認可・アカウンティングの問題は，Zone 間の AAA Server 同士が通信することによって解決している．

3.2 AFC の構成要素

3.2.1 Node

Node は物理または仮想マシンであり，計算資源情報を管理する最小の単位である．**Node Monitor** は Node の計算資源情報 (CPU 使用状況，RAM 使用状況) を取得し，Spot Manager の Computing Resource Manager に送信する．**Node AFC Controller** は Zone の AFC Controller からの要求に応じて AFC Daemon を生成する．**AFC Daemon** は Node で Chained-AF ごとに 1 つ生成され，AF の起動や AF 間のデータ転送を担う．**AF (Application Function)** は データを入力して特定の演算をし，結果を出力する関数のようなものである．

3.2.2 Spot

Spot は Node の集合であり，ISP やクラウド事業者の一拠点またはデータセンタを想定している．**Network Monitor** は Spot 間のネットワーク状態 (帯域, 遅延) を測定し，Spot Manager の Network Resource Manager に送信する．**Spot Manager** は Network Resource Manager は Network Monitor から収集したネットワーク資源情報を，Computing Resource Manager は Node Monitor から収集した計算資源情報を集約し，Zone Manager に通知する．

3.2.3 Zone

Zone は Spot の集合であり，同じポリシを共有する Spot 群の管理ドメインと言える．**Zone Manager** は Spot Manager の Network Resource Manager，Computing Resource Manager から資源情報を収集して保存する．資源情報は Spot Manager からの通知によって定期的に更新され，Orchestrator からの要求に応じて送信する．**Orchestrator** は Zone Manager から取得した資源情報をもとに AF の Spot 配置計算をする．Spot 内の Node 配置を決定する役割も担う．Zone 間を跨ぐ Chained-AF を扱う場合には，他 Zone の Orchestrator にも配置計算を依頼する．AF の配置先が決まると，AFC Controller に Chained-AF の設置を要求する．**AF Catalog** は AF Developer が作成した AF を登録するためのデータベースである．AF の登録・削除や AF の情報取得といった要求を受けた場合，その正当性を検証する．**Chain Catalog** は Chained-AF Developer が作成した Chained-AF を登録するためのデータベースである．Chained-AF の登録・削除や Chained-AF の情報取得といった要求を受けた場合，その正当性を検証する．**AFC Controller** は Orchestrator の要求に基づいて Node AFC Controller に指示を出す．**AAA Server** は Zone 内の認証・認可・アカウンティングを担う．複数の Zone に跨る認証・認可が必要な場合は，AAA Server 自身が他 Zone の AAA Server に問い合わせる．**OSS/BSS** は AFC のプラットフォームを管理・運営していくためのシステム全般を表す．AFC Zone Operator，AF Developer，Chained-AF Developer，Chained-AF User のそれぞれにダッシュボードを提供する．**AFC Zone Operator** は Zone の管理者である．**AF Developer** は AF の開発者であり，AFC で規定された構文に基づいてプログラムを作成する．開発した AF は AAA Server での AF Developer の認証・認可を経て AF Catalog に登録することができる．**Chained-AF Developer** は Chained-AF の開発者であり，AF の組み合わせによって Chained-AF を作成する．このとき，他 Zone の AF Catalog に含まれる AF を使用しても良い．開発した Chained-AF は AAA Server での Chained-AF Developer の認証・認可を経て Chain Catalog に登録することができる．**Chained-AF User** は Chained-AF の利用者であり，Chained-AF の使用状況に基づいて料金を支払う．

4. Orchestrator による最適配置手法

4.1 AFC プラットフォームのモデル化

モデルで使用する記号の定義を表 **1** に示す．Spot の計算資源としては仮想 CPU (vCPU) のコア数とメモリ (RAM) 容量を想定し，AF に割り当て可能な最大容量と割り当て済みの容量を定義した．Spot ごとに vCPU と RAM の単価を設定できる．ネットワーク資源としては Spot 間のリンクのみを考慮し，リンクには AFC 専用のプライベートネットワークを利用する場合と，パブリックネットワークを利用する場合の 2 種類を想定している．Chained-AF によって提供されるサービスによっては，一定の通信速度やネットワーク遅延を保証したい場合もあれば，ベストエフォートのネットワーク品質で十分な場合もあるからである．AFC 専用線においては QoS の保証が可能であるという前提のもと，最大帯域や予約済みの帯域，ネットワーク遅延を定義した．AF Developer は AF が動作するのに必要な vCPU や RAM，AF の単価を設定することができる．AF の冗長度は Chained-AF 内の各 AF に対して，冗長なインスタンスを何個生成するかを表す値である．GN を設置する Spot は Chained-AF User の位置情報によって決まる．Chain とは AF 同士，または AF と GN 間の接続関係のことであり，1 つの Chained-AF には 1 つ以上の Chain が含まれる．$A^N_{m,n} \in \{0,1\}$，$Q^{band}_{m,n}$，$A^D_{r,m,n} \in \{0,1\}$ における m，n とは AF または GN を表している．要素

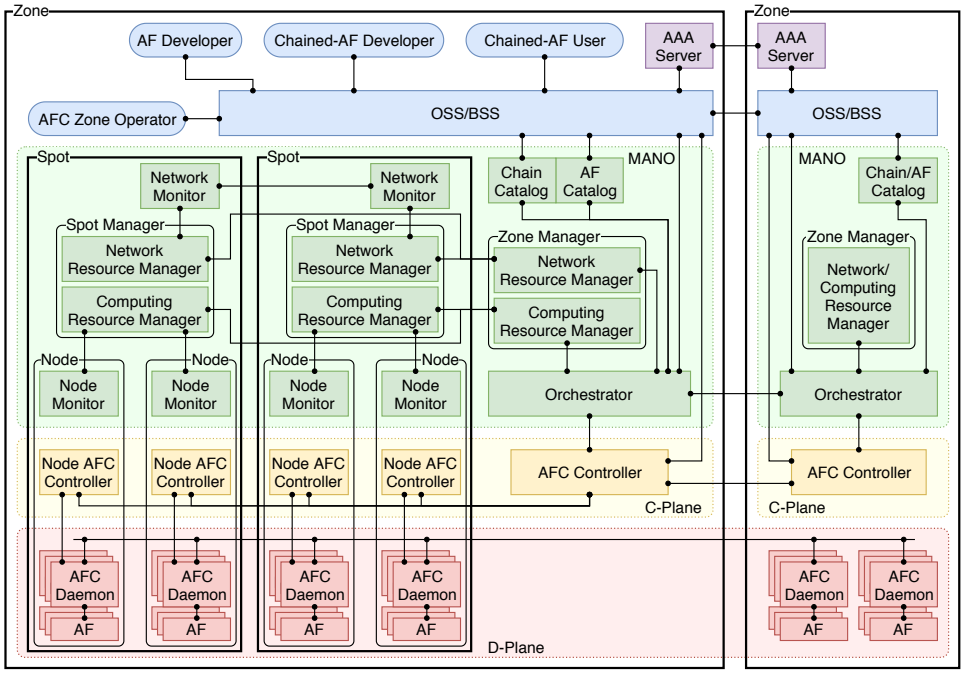


図 4: AFC の新アーキテクチャ.

表 1: モデルで使用する記号の定義.

記号	説明
$i \in S$	Spot
C_i / C_i^{used}	Spot i の vCPU 最大数 / 使用数
P_i^{cpu}	Spot i の vCPU 単価
M_i / M_i^{used}	Spot i の RAM 最大量 / 使用量
P_i^{ram}	Spot i の RAM 単価
$(i,j) \in L$	Spot i と Spot j の間のリンク
$B_{i,j}$ / $B_{i,j}^{used}$	リンク (i,j) の最大帯域 / 予約帯域
$D_{i,j}$	リンク (i,j) の遅延 (Private)
$P_{i,j}^{band}$	リンク (i,j) の専用線単価
$D_{i,j}^{pub}$	リンク (i,j) の遅延 (Public)
$m \in F$	AF (Application Function)
F_m^{cpu} / F_m^{ram}	AF m の vCPU / RAM 要求量
P_m^{af}	AF m の単価
F_m^{dup}	AF m の冗長度
$n \in G$	GN (Gate Node)
$A_{n,i}^G \in \{0,1\}$	Spot i への GN n の設置の有無
$N = F \cup G$	AF と GN の和集合
$A_{m,n}^N \in \{0,1\}$	AF と GN (m/n) 間の Chain の有無
$Q_{m,n}^{band}$	m と n の間の Chain の帯域要求
$r \in D^{req}$	Chain の遅延要求
$A_{r,m,n}^D \in \{0,1\}$	r における m と n の組の有無
Q_r^{delay}	r における NW 遅延の和
$x_{l,m,i} \in \{0,1\}$	Spot i への AF m の配置の有無
$l \in \{1,2,...,\prod F^{dup}\}$	冗長度 F_m^{dup} に関する変数

が 0 または 1 の行列によって Chained-AF の形状や，各 Chain に対する帯域・遅延の要求を設定できる．

4.2 AF の Spot 配置問題

AFC-MANO における AF の Spot 配置問題は，GN を設置する Spot $A_{n,i}^G \in \{0,1\}$ を入力とし，AF を配置する Spot $x_{l,m,i} \in \{0,1\}$ を出力するモデルで表される．GN n が Spot i に設置されるとき $A_{n,i}^G = 1$ となり，最終的に AF m は $x_{l,m,i} = 1$ となるような Spot i に配置される．

目的関数を式 (1) に示す．第 1 項は，AF 間のネットワーク遅延 $x_{l,m,i} \cdot x_{l,n,j} \cdot A_{m,n}^N \cdot A_{r,m,n}^D \cdot D_{i,j}$ を全ての AF, Spot, r, l に関して合計したものを表している．第 2 項は，AF と GN の間のネットワーク遅延 $x_{l,m,i} \cdot A_{n-|F|,j}^G \cdot A_{m,n}^N \cdot A_{r,m,n}^D \cdot D_{i,j}$ を全ての AF，GN，Spot，r，l に関して合計したものを表している．求める AF の Spot 配置 $x_{l,m,i} \in \{0,1\}$ には添字 l が含まれており，$x_{l,m,i} = x_{k,m,i}$ のとき $x_{l,m,i}$ と $x_{k,m,i}$ が表すインスタンスは同一のものである．

$$\sum_{\substack{m,n \in N \\ :m<n<|F|}} \sum_{\substack{i,j \in S \\ r \in Q, l}} x_{l,m,i} \cdot x_{l,n,j} \cdot A_{m,n}^N \cdot A_{r,m,n}^D \cdot D_{i,j} + \sum_{\substack{m,n \in N \\ :m<|F| \leq n}} \sum_{\substack{i,j \in S \\ r \in Q, l}} x_{l,m,i} \cdot A_{n-|F|,j}^G \cdot A_{m,n}^N \cdot A_{r,m,n}^D \cdot D_{i,j} \tag{1}$$

制約条件を式 (2)〜(5) に示す．式 (2)，(3) は Chained-AF 内の AF の vCPU または RAM 要求量の合計が，各 Spot において許容量を超えないための条件である．式 (4) は Chained-AF 内の Chain の帯域要求の合計が，各 Spot 間リンクの許容量を超えないための条件である．左辺の第 1 項は，AF 同士の間の帯域要求 $x_{l,m,i} \cdot x_{l,n,j} \cdot A_{m,n}^N \cdot Q_{m,n}^{band}$ を全ての AF に関して合計したものを表している．第 2 項は，AF と GN の間の帯域要求 $x_{l,m,i} \cdot A_{n-|F|,j}^G \cdot A_{m,n}^N \cdot Q_{m,n}^{band}$

を全ての AF，GN に関して合計したものを表している．右辺は Spot i と Spot j の間のリンクにおいて，最大帯域 $B_{i,j}$ から予約帯域 $B_{i,j}^{used}$ を引いた値である．式 (5) は Chained-AF 内の各 AF が冗長度を満たすための条件である．

$$\sum_{m\in F,l} x_{l,m,i}\cdot F_m^{cpu}\cdot\frac{F_m^{dup}}{\prod F^{dup}} \le C_i - C_i^{used} \quad \forall i\in S \tag{2}$$

$$\sum_{m\in F,l} x_{l,m,i}\cdot F_m^{ram}\cdot\frac{F_m^{dup}}{\prod F^{dup}} \le M_i - M_i^{used} \quad \forall i\in S \tag{3}$$

$$\sum_{\substack{m,n\in N\\ :m<n<|F|}} x_{l,m,i}\cdot x_{l,n,j}\cdot A_{m,n}^{N}\cdot Q_{m,n}^{band} + \sum_{\substack{m,n\in N\\ :m<|F|\le n}} x_{l,m,i}\cdot A_{n-|F|,j}^{G}\cdot A_{m,n}^{N}\cdot Q_{m,n}^{band} \le B_{i,j} - B_{i,j}^{used} \quad \forall(i,j)\in L,\ \forall l \tag{4}$$

$$\sum_{i\in S} x_{l,m,i} = 1 \quad \forall m\in F,\ \forall l \tag{5}$$

これらの目的関数と制約条件をもとに，Orchestrator は AF の Spot 配置を計算する．ただし，式 (4) で示したように，帯域に関する要求は Chained-AF 内の各 Chain ごとに設定することができる．Spot 間のリンクにパブリックネットワークが混在している場合は，該当するリンクの空き帯域を 0 に設定すれば，自ずと帯域要求のある Chain が AFC 専用プライベートネットワークに割り当てられることになる．目的関数や制約条件の式を変更することなく，同一のモデルで表現することが可能である．

5. 実装

5.1 資源情報の収集機構

図 4 で示した AFC の新アーキテクチャのうち，以下 3 種類のモジュールを実装した．MANO-Plane において資源情報は Pub/Sub の形式で収集される．

5.1.1 Network/Node Monitor

Network Monitor と Node Monitor は単一のプログラムとして実装している．Spot Manager に接続を要求した後，コマンド入力によって Spot Manager へネットワーク資源情報もしくは計算資源情報を出版 (Publish) する．

5.1.2 Spot Manager

Network Resource Manager と Computing Resource Manager は単一のプログラムとして実装している．起動時に指定した IP アドレス・ポート番号で Network/Node Monitor や Zone Manager の接続要求を待つ．Spot Manager は Broker の役割を担っており，Network/Node Monitor から Publish された資源情報を Spot ごとに集約し，Zone Manager へ Notify する．資源情報は KVS (Key-Value Store) である Redis [15] に保存される．

Algorithm 1 ヒューリスティック手法の疑似コード

Ensure: GN の Spot 位置 $x_{l,m,i}\in\{0,1\}$
Require: AF の Spot 配置 $A_{n,i}^{G}\in\{0,1\}$
  $upper \leftarrow MAX$ ▷ 予め遅延の最大値を算出
  $lower \leftarrow 0$
  while $upper - lower < k$ **do**
    $delay \leftarrow (upper + lower)/2$
    $s \leftarrow solveModel(delay)$ ▷ 目的関数を定数として計算
    if s is ok **then**
      $result \leftarrow s$ ▷ 解が求められた場合は保存
      $upper \leftarrow delay$ ▷ 制約条件を厳しく
    else
      $lower \leftarrow delay$ ▷ 制約条件を緩く
    end if
  end while
  if $result = \emptyset$ **then**
    return error
  else
    return $result$
  end if

5.1.3 Zone Manager

Network Resource Manager と Computing Resource Manager 単一のプログラムとして実装している．指定した設定ファイルに記述された Spot Manager に対して接続を要求する．接続の確立後，コマンド入力によって Spot Manager へネットワーク資源情報もしくは計算資源情報を購読 (Subscribe) する．Subscribe していた情報が Network/Node Monitor から Publish されると，最終的に Zone Manager まで通知される．

5.2 Orchestrator の計算処理

5.2.1 最適化手法

最適化手法とは，式 (1) 〜 (5) で定義した目的関数・制約条件を利用し，ソルバーによって解を求める手法である．モデリング言語として Python で数理最適化モデルを記述可能なライブラリ Pyomo [16] を使用し，ソルバーとして IBM ILOG CPLEX Optimizer 12.9.0 [17] を使用した．

5.2.2 ヒューリスティック手法

ヒューリスティック手法とは，ソルバーによる計算に独自のアルゴリズムを組み合わせた手法である．アルゴリズム 1 に，ヒューリスティック手法の擬似コードを示す．制約条件次第で計算時間が増大するのを防ぐため，ヒューリスティック手法では目的関数を定数とし，制約条件を満たす解を 1 つ求めた段階で計算を終了させた．二分探索によってネットワーク遅延に関する制約条件を狭めながら繰り返しソルバーを実行することで，最適解に近い値を求めている．ここで，ネットワーク遅延の最大値 MAX は Chained-AF の形状と Spot 間遅延の最大値から算出する．

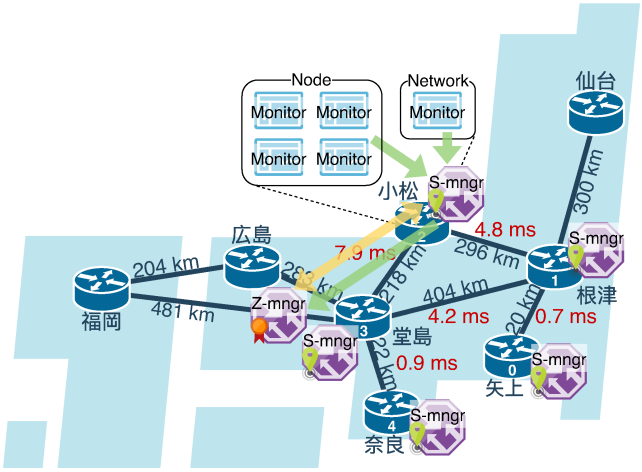
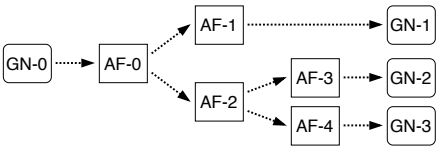
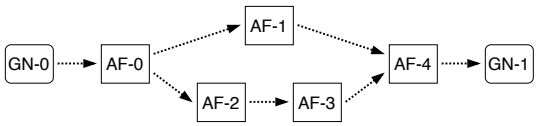


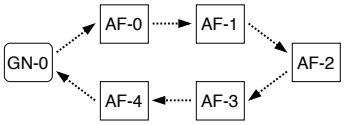
図 5: WIDE Cloud を用いた実験環境.



(a) 分岐 (Branch) 型 Chained-AF の例.



(b) 分岐・合流 (Merge) 型 Chained-AF の例.



(c) 循環 (Cycle) 型 Chained-AF の例.

図 6: 様々な形状の Chained-AF.

6. 評価

6.1 資源情報の収集機構

図 5 に実験環境を示す．WIDE プロジェクト [18] が運営する大学間 VM クラウドインフラストラクチャ「WIDE Cloud」上の 5 拠点 (矢上，根津，小松，堂島，奈良) に 1 つずつ VM (Virtual Machine) を配置し，AFC における Spot と見立てた．Spot 間のリンクは全て AFC 専用プライベートネットワークで構築されているものとする．各拠点の VM は QEMU の仮想 CPU を 2 コア，RAM を 2 GB 搭載した Ubuntu 18.04 LTS である．各 Spot には Spot Manager と Network Monitor を 1 つずつ，Node Monitor を 4 つを設置し，中央の Spot (堂島拠点) にのみ Zone Manager を 1 つ設置した．

6.1.1 Subscribe Request & Subscribe Response

Spot Manager へ Subscribe Request が送信されてから，全ての Subscribe Response が返ってくるまでの時間を計測した．図 **5** における黄色の矢印に相当する．Zone Manager の Network Resource Manager が最初の Subscribe Request を送信してから，最後の Subscribe Response を受信するまでにかかった時間は 21.2 ms であった．Subscribe Request, Subscribe Response のメッセージ送受信は，Spot ごとに並列処理されていることが確認できた．

6.1.2 Publish Request & Publish Notify

Node Monitor から Publish Request が送信されてから，全ての Subscribe Notify が受信されるまでの時間を計測した．図 5 における緑色の矢印に相当する．Node Monitor が最初の Publish Request を送信してから，Zone Manager の Computing Resource Manager が最後の Subscribe Notify を受信するまでにかかった時間は 8.9 ms であった．Publish Request，Publish Response，Publish Notify のメッセージ送受信は，Node ごとに並列処理されていることが確認できた．

表 2: Orchestrator の評価環境.

HV (Hypervisor)	
OS	VMware(R) ESXi(TM) 6.7.0
CPU	Intel(R) Xeon(TM) Gold 6248 2.50GHz × 40
RAM	400GB
VM (Virtual Machine)	
OS	Ubuntu Server 18.04 LTS
vCPU	Intel(R) Xeon(TM) Gold 6248 2.50GHz × 32
RAM	16GB

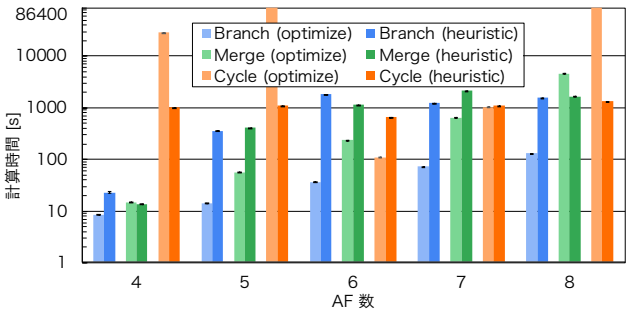


図 7: AF 数に対する Orchestrator の計算時間.

6.2 Orchestrator による AF の Spot 配置計算

Orchestrator の計算処理に関して，AF の配置結果や計算時間を評価した．ネットワークトポロジとして AT&T が公開している PoP (Point of Presence) レベルのネットワーク [19] の一部を使用し，アメリカ全土に分散配置されている 18 の PoP を AFC における Spot とみなした．評価環境として，表 **2** に示す HV (Hypervisor) 上の VM を使用した．

図 **7** に結果を示す．AF 数ごと，図 **6** に示す 3 種類の形状の Chained-AF を設置する場合に関して，最適化手法とヒューリスティック手法による計算時間を表している．最適化手法に関しては，計算時間が Chained-AF の形状に

表 3: End-to-end 遅延の比 [heuristic]/[optimize].

AF 数	4	5	6	7	8
Branch 型	1.5	1.0	1.1	0.8	1.1
Merge 型	1.0	1.0	1.0	1.1	1.5

大きく依存していることがわかる．特に Cycle 型の計算時間が増大しているのは，制約条件が極端に緩くなるからだと考えられる．一方，ヒューリスティック手法に関しては，AF 数の増加による計算時間への影響が少ない．例えば，Merge 型の Chained-AF に関する計算で AF 数が 6 から 7 へ増加するとき，計算時間は減少している．全体的に計算時間の大幅な増加が起こりにくいのは，ヒューリスティック手法においてソルバーによる計算を繰り返す中で，一定時間のタイムアウトを設定しているためだと考えられる．

表 3 に，Orchestrator による AF の Spot 配置結果に基づいて計算した End-to-end ネットワーク遅延の比の値を示す．一部計算が終わらなかった Cycle 型以外について，ヒューリスティック手法で計算した場合のネットワーク遅延の値を，最適化手法で計算した場合の値で割っている．ヒューリスティック手法の方が，最適化手法に比べて概ね 1.0 〜 1.5 倍程度の遅延となっている．比の値が 0.8 となっている箇所があるのは，最適化手法においてソルバーが局所最適解に陥っているためだと考えられる．

7. おわりに

従来 AFC は AF の連鎖によってアプリケーションを作成する仕組み自体に焦点を当てていた．複数の管理ドメインを考慮しておらず，ネットワーク資源や計算資源の管理機構も定義されていなかった．本稿では AFC における MANO 機構を含んだ AFC 全体のアーキテクチャを再定義するとともに，AF の Spot 配置計算手法を提案した．AFC-MANO では地理的に分散したヘテロジニアス環境において，ネットワーク資源と計算資源の管理機構を実現した．AFC の新しいアーキテクチャは Data-Plane，Control-Plane，MANO-Plane の 3 つのプレーン構造に加え，認証・認可・アカウンティングを担う AAA 基盤によって構成されており，マルチドメインを考慮したスケーラブルな設計となっている．評価結果から，MANO-Plane における一部機構の実装が正常に動作することを確認した．また，AF の Spot 配置手法に関して，30 分程度の計算時間で解を算出できることを確認した．End-to-end のネットワーク遅延に関しては，最適化手法に比べてヒューリスティック手法の方が 1.0 〜 1.5 倍程度であった．

参考文献

[1] Halpern, J. and Pignataro, C.: Service Function Chaining (SFC) Architecture (2015). RFC 7665.
[2] 渡邊 大記, 近藤 賢郎, 寺岡文男：通信と計算の融合に向けたアプリケーション処理連接基盤，信学技報，Vol. 118, No. 326, pp. 9–16 (2018).
[3] 佐藤 友範, 渡邊 大記, 林 和輝, 近藤 賢郎, 寺岡文男：5G コアネットワーク向けアプリケーション処理連接基盤，研究報告マルチメディア通信と分散処理 (DPS)，Vol. 2019-DPS-180, No. 18, pp. 1–8 (2019).
[4] ETSI: Network Functions Virtualisation (NFV); Management and orchestration, *ETSI GS NFV-MAN 001*, Vol. V1.1.1 (2014).
[5] ETSI: Network Functions Virtualisation (NFV) Release 3; Management and Orchestration; Report on architecture options to support multiple administrative domains, *ETSI GS NFV-IFA 028*, Vol. V3.1.1 (2018).
[6] Kukliński, S. and Tomaszewski, L.: DASMO: A Scalable Approach to Network Slices Management and Orchestration, *Proceedings of 2018 IEEE/IFIP Network Operations and Management Symposium (NOMS 2018)*, pp. 1–6 (2018).
[7] Sciancalepore, V., Giust, F., Samdanis, K. and Yousaf, Z.: A double-tier MEC-NFV Architecture: Design and Optimisation, *Proceedings of 2016 IEEE Conference on Standards for Communications and Networking (CSCN)*, pp. 1–6 (2016).
[8] Brogi, A., Forti, S. and Ibrahim, A.: How to Best Deploy Your Fog Applications, Probably, *Proceedings of 2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, pp. 105–114 (2017).
[9] Giang, N. K., Lea, R., Blackstock, M. and Leung, V. C. M.: Fog at the Edge: Experiences Building an Edge Computing Platform, *Proceedings of 2018 IEEE International Conference on Edge Computing (EDGE)*, pp. 9–16 (2018).
[10] OpenJS Foundation: Node-RED. https://nodered.org.
[11] Ghribi, C., Mechtri, M., Soualah, O. and Zeghlache, D.: SFC Provisioning over NFV Enabled Clouds, *Proceedings of 2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pp. 423–430 (2017).
[12] Rankothge, W., Ma, J., Le, F., Russo, A. and Lobo, J.: Towards making network function virtualization a cloud computing service, *Proceedings of 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 89–97 (2015).
[13] Alleg, A., Ahmed, T., Mosbah, M., Riggio, R. and Boutaba, R.: Delay-aware VNF Placement and Chaining based on a Flexible Resource Allocation Approach, *Proceedings of 2017 13th International Conference on Network and Service Management (CNSM)*, pp. 1–7 (2017).
[14] Luizelli, M. C., Bays, L. R., Buriol, L. S., Barcellos, M. P. and Gaspary, L. P.: Piecing Together the NFV Provisioning Puzzle: Efficient Placement and Chaining of Virtual Network Functions, *Proceedings of 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 98–106 (2015).
[15] Redis Labs: Redis. https://redis.io.
[16] COIN-OR: Pyomo. http://www.pyomo.org.
[17] IBM: ILOG CPLEX Optimization Studio. https://www.ibm.com/products/ilog-cplex-optimization-studio.
[18] WIDE Project: WIDE. http://www.wide.ad.jp.
[19] Ciavattone, L., Morton, A. and Ramachandran, G.: Standardized active measurements on a tier 1 IP backbone, *IEEE Communications Magazine*, Vol. 41, No. 6, pp. 90–97 (2003).