実行配置管理システムにおけるノードの処理能力を考慮した実行地点の最適化の検討

鎌田 幸希[1] 稲村 浩[2] 中村 嘉隆[2]

概要：IoT などの応用において発生する大量のデータを効率良く扱うためにクラウドコンピューティングのパラダイムをネットワークの端まで広げたフォグコンピューティングが提唱されている．フォグコンピューティングでは，実行地点の最適配置について議論されている．ネットワーキングの分野では，従来のホスト指向のアーキテクチャではない，CCN(Contents Centric Network) の研究がされており，ネットワーク内にキャッシュされたコンテンツを自然に扱うことによる静的データの最適配置が議論されている．我々は，CCN の考え方を援用し，フォグコンピューティングにおける実行地点選択と資源配置のための実行配置管理システムを提案している．本研究は，機械学習などの目的に特化した処理ユニットの利用を考慮し，フォグネットワーク内の計算能力の不均一性に対処するためにこれまでの実行配置管理システムの候補選出基準にサービス応答時間を加え，クライアントとの経路上の利用可能な計算資源を発見すべく探索手法を拡張した．提案したシステムをシミュレータ上に実装し，処理能力が不均一なフォグネットワークにおいてもサービス実行地点の最適化が行われることを示した．

1. はじめに

2017 年時点で 274 億個ある IoT 機器は 2020 年までには約 400 億個にまで増えるという予想がある [1]．これらの大量の IoT 機器の生成するデータに対してクラウドコンピューティングのような処理集中型のアーキテクチャでは，エッジの持つ処理能力が活用されておらず，データ発生地点から遠隔にあるデータセンターまでのレイテンシも無視できない．そこで，クラウドコンピューティングのパラダイムをネットワークの端まで広げたフォグコンピューティングが提唱されその研究が活発である [2]．

ネットワーキングの分野では，従来の IP アドレスによるアーキテクチャではなく NDN(Named Data Networking) を始めとするコンテンツ指向型ネットワーク (CCN:Contents Centric Network) の研究 [3] がなされており，位置によらないコンテンツを識別子とすることで，ネットワーク内 (In-Network / インネットワーク) でキャッシュされたコンテンツを自然に扱うことができ，トラフィックや遅延時間の削減が可能であることが示されている．

そこで我々は，フォグコンピューティングにおける実行地点選択と資源配置のための実行配置管理システムを提案した [4]．フォグコンピューティングのユースケースとして，機械学習を用いたサービスを，データ取得と操作対象に近いフォグノードで行うことが考えられる．フォグノードでサービスを実行し，サービス応答時間を短縮することで実時間性能の向上が期待できる．機械学習をフォグノード上で行うには，GPU や TPU (Tensor Processing Unit)[5] のような目的に特化した専用のユニットが搭載されていることが望ましい．しかし，すべてのフォグノードに対して，専用のユニットを搭載するのはコストの問題が生じるため，専用ユニットが搭載されるノードと普通のノードがフォグネットワーク内に混在することが予想される．このような，フォグネットワークの処理能力が不均一になるユースケースにおいて，これまでの実行配置管理システムでは，ネットワーク遅延のみをサービス移送の候補選出基準にしているため，処理能力を持ち合わせていないノードに実行地点を移送してしまい，十分な最適化が行われない可能性がある．本研究は，実行配置管理システムの候補選出基準にサービスの要求処理能力を加えることで，処理能力が不均一なフォグネットワーク上でサービスの要求処理能力に見合ったサービス移送を行いサービス応答時間をより短くすることを目標とする．本システムの有効性を示すにあたり，シミュレータ上に本システムを実装しサービス応答時間の評価を行った．

[1] 公立はこだて未来大学大学院 システム情報科学研究科
[2] 公立はこだて未来大学 システム情報科学部

2. 関連研究

2.1 フォグコンピューティング

フォグコンピューティングでは，実行処理に必要な地点を選択し，移送させることで，実行にかかる遅延時間を抑えている．例えば，Wireless Sensor and Actuator Networking に，センサノードが収集したデータをクラウドに移す前に，単純な処理はフォグノードなどの中間ノードで実行し，クラウドの代わりにアクチュエーションに対して中間ノードが命令をすることで遅延時間を削減することが可能になっている．他にも，コードオフローディングと呼ばれる技術がある [6], [7], [8]．コードオフローディングとは，リソースが制約されたモバイルデバイス上でのモバイルアプリケーションのエネルギー効率と実行速度の向上を目指している技術である．具体的には，モバイルアプリケーションにおいて，リソースの余裕があるクラウドサーバなどのモバイルデバイスよりも計算資源を持っているデバイスに対してコードの実行を依頼することで，モバイルデバイスが持っている，電池や CPU，メモリなどの資源を使って動作するよりも資源を節約できるというものである．このように，フォグコンピューティングでは計算リソースの最適配置について多くの議論がなされているが，コンテンツの最適配置という面では議論がなされていない．

2.1.1 Code Bubbling Offload System

Berg ら [9] は，それまでのコードオフローディング技術では，2 つまたは，3 つのデバイスからなる比較的単純で制限的システムモデルしか考慮されていない研究しか存在しないことに着目した．高度に分散した様々なリソースの複数クラスがコードオフロードを利用する場合に，N 層のアーキテクチャが相互に関係を持つようになると主張している．例えば，スマートウォッチ (第 1 層) はブルートゥースを介してスマートフォン (第 2 層) に接続する．そのスマートフォンは，エッジサーバ (第 4 層) に接続された車 (第 3 層) に Wifi を介して接続する．さらに，4G モバイル通信や固定ネットワークをまたいでクラウドセンターのサーバ (第 5 層) に接続される (図 1 参照)．このような N 層のシステムにおける異機種間デバイスは，エネルギー，計算リソースの点で大きく異なる．この例では，第 1 層から第 5 層までになっているが今後，層の複雑さは，無限に増加する．そこで彼らは，異なる性能特性とコストの関係を持つ高度に分散した異機種リソースを含む N 層環境を対象としたコードオフロードシステム CoBOS(Code Bubbling Offload System) を提案した．この提案の中にはコードバブリングと呼ばれる概念も含まれている．

コードバブリングは，コードをより強力でより遠い層に動的かつ適応的に移動させ，N 層環境で効率的でスケーラブルなコードオフローディングを可能にしている．これにより，N 層環境でのコードオフローディングで 77 %のエネルギー消費と 83 %の実行時間を削減している．

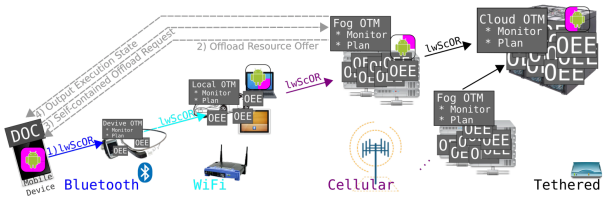

図 1 Overview of an exemplary Mobile Cloud Computing environment, where a mobile device offloads application parts to multiple autonomous tiers based on code bubbling. The CoBOS components are a Device Offload Controller (DOC), Offload Tier Managers (OTMs), and Offload Execution Engines (OEEs). ([9] の図 1 より引用)

この研究では，モバイルアプリケーションの実行時に，アプリケーションのコード部品をより強い層へとオフロードすることで，モバイルアプリケーションにおける実行地点の最適化を図っている．そこで筆者は，これを逆に考え，クラウドで実行されているサービスを，よりユーザーに近い場所で実行するアーキテクチャを考えることで，サービスの実行地点の最適化を行うことができるのではないかと考えた．

2.2 CCN

Jacobson ら [3] は，従来の IP アドレスによるアーキテクチャではない CCN を提案している．CCN の通信では，Interest と Data という 2 種類の CCN メッセージを用いたプロトコルで行われる．メッセージの送受信には，Interest メッセージをルーティングするための FIB(Forwarding Information Base)，コンテンツをキャッシュする CS(Content Store)，Data を要求元に送り返すための PIT(Pending Interest Table) の 3 つの主要なデータ構造が用いられている．これらのデータ構造を用いて CCN では Interest と Data のメッセージのやりとりをしている．彼らは，これにより，IP のシンプルさとスケーラビリティを維持したまま，セキュリティ，配信効率，および中断耐性が大幅に向上したことを証明している．このように，CCN ではコンテンツをユーザーに近い場所に置くことによる静的なコンテンツの配布は可能である．しかし，実行されているシステムを同様に扱うには，単純にキャッシュを増やしてユーザーにより近い場所へ配置するだけでは，同一の内部状態をもとにしたシステムを継続させることができず，動的なコンテンツやサービスの提供は不可能である．

CCN におけるキャッシュ効率についての研究 [10], [11] やどのように効率よく Interest パケットをルーティングするかの研究 [12], [13] が存在する．これらの研究では，静的なコンテンツの扱いや分散したコンテンツをどれだけ効率的に透過なものとして考えられるかという点について議論

が進められており，動的なサービスをどのようにしてネットワーク上で分散して配置していくのかという点については議論がない．

2.2.1 Services over Content-Centric Routing

ホスト指向の通信では，コンテンツやサービスの複製，キャッシングサービス，ロード・バランシング，コンテンツ要求に対するルーティングなどの技術はアプリケーション (CDN や P2P など) の導入によって可能になったが，ネットワークの管理，運用のコストが高くなる．これらの問題に対して，CCN と SCN(Service Centric Network)[14] の 2 つのパラダイムを用いて解決に取り組まれている．

コンテンツとサービスを別々に扱ってしまうとコンテンツとサービスの関係性を示すことができない．実際には，サービスは新しいコンテンツを生成したり，既存のコンテンツに対して様々な機能を実行するサービスがあるように，コンテンツとサービスは深い関係性があるにもかかわらず，コンテンツとサービスの溝が生まれてしまう問題があり，CCN と SCN の差を埋める必要がある．そこで，Shanbhag ら [13] は SoCCeR(Services over Content-Centric Routing) を提案した．SoCCeR は．Interest メッセージのルーティングテーブルの操作のために，CCN の上に新たな蟻コロニー最適化 [15] を利用したサービスルーティング制御層を付け加えた．CCN のコンテンツ要求および検索機能に影響を与えずに，SCN 機能を追加し，サービス要求をより軽い負荷でサービスインスタンスに選択的にルーティングし，ネットワークおよびサービスの状態の変化に対して俊敏に反応できることを示した．

3. 実行配置管理システム

ここまでに挙げた研究には，それぞれ問題点が存在する．CCN では，現在の TCP / IP をすべてコンテンツ指向ネットワークに置き換えるという考えのもと議論が進んでいるため，静的なコンテンツのキャッシュ方式に関する議論が複数存在する．しかし，現実の TCP / IP によって作られる現在のインターネットでは，動的なサービスによる動的なコンテンツが不可欠になっている．例えば，ユーザー認証を行い，そのユーザーに適した情報を表示する Web ページや，ユーザーが Web ページのアップデートをしたときに分散しているキャッシュに更新がされないなどがある．これらの動的なコンテンツが大量に存在する現在のインターネットを CCN に置き換えるには，静的なコンテンツのキャッシュ方式を考えるだけでは，不十分なのではないかと我々は考えた．フォグコンピューティングの研究においても，フォグネットワーク上にどのようにデータを配置していくかの問題については，議論が少ない．そこで，2.1 で紹介したようなフォグコンピューティングにおける計算資源の扱い方を CCN に取り入れることでフォグコンピューティングの研究分野の問題と CCN の研究分野の問題を相互に解決できると考えた．

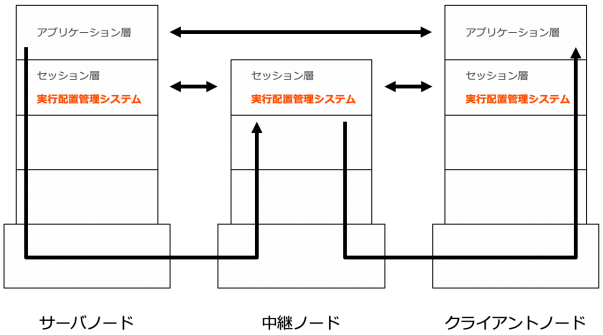

図 2 実行配置管理システム

我々はこれまで実行配置管理システムを提案している [4]．このシステムではクライアントはセンサやユーザーなどと直接のやりとりがあるためネットワークにおける動作位置は固定されているものと想定している．それに対してサービスを提供しているサーバはフォグ/クラウド内での配置に任意性があるため，処理の実行にあたり必要な資源を求めてサービス実行の状態を移送することで実行地点を変更可能であり望ましいと仮定する．

実行地点管理システムの動きを図 2 で示す．本システムは全参加フォグノードのセッション層で動作しているものとする．サービスの実行地点の変更はクライアントやサーバに透過的に行われることが望ましい．従って 7 層モデルで言うところのセッション層で動作するものとする．サービスの移送などの管理動作はトランスポート層以下の資源が操作対象であるため，それらが可視である上位層に位置する必要があり，かつクライアントやサーバが動作しているアプリケーション層よりも下位であるのはセッション層である．サーバノードとクライアントノードとのアプリケーション層での通信を利用して，セッション層の実行配置管理システムで通信処理を中継する．サーバノード，中継ノード，クライアントノードの実行配置管理システム間で通信し，サービスの実行地点選択と資源配置を実現する．

本提案システムは 3 種類のノードで構成される．オリジナルのデータを持ちネットワーク遅延はあるが処理能力の高いクラウドノード，次に中処理能力でエンドデバイスと比較的低レイテンシで通信ができる中継ノードであるフォグノード，スマートフォンのようなサーバに参加するエンドデバイスであるクライアントノードをそれぞれ複数接続しているネットワークを構築している．

クライアント/サーバ型の通信モデルを基本にクラウドノードとフォグノードがサーバの役割をクライアントノードがクライアントの役割をこなす．サーバは，参加しているクライアントの通信状況を監視し，通信状況から自律的に自分がサーバの役割をするべきか，他のフォグノードにサーバの役割を委託するべきか判断する．委託されたフォ

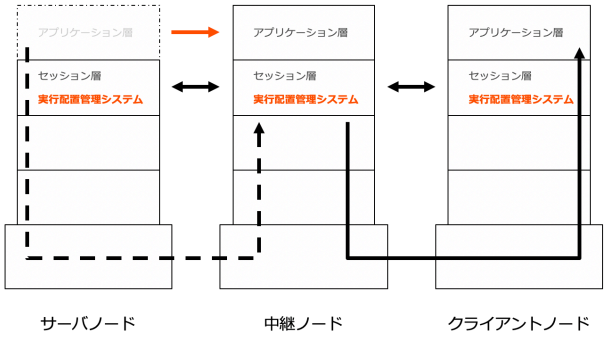

図 3　実行配置管理システムの移送後のイメージ

グノードは，サーバの役割を引き継ぐ．

本システムの動きを簡単に説明する．図 2 で示した構造で，中継ノードにアプリケーション層で実行されているサービスを移送するための資源が余っている場合かつ，中継ノードがサービスを実行してもクライアントへのサービス提供のクオリティが低下しない場合，図 3 のように本システムの上に存在するアプリケーション層のサービスを中継ノードへと移送する．そうすることで破線で書かれたサーバノードのアプリケーション層のサービスが利用していた計算資源が空き，破線で書かれた部分の通信をすることなくサービスの実行ができるためネットワーク資源も効率的に利用できる．さらに，クライアントノードから見ると，サービスの実行地点がクラアントノードに近い場所に移送されるため，レイテンシなどのサービスのクオリティの向上が期待できる．サービス移送を行うために，中継ノードであるフォグノードはすべてサービス実行のための計算資源を十分に持っていると仮定する．このようにして，サービスの実行地点最適化を図る．

3.1　ノードの持つ機能

各フォグノードは，ネットワーク内資源監視機能，候補選出機能とサービス移送機能を持っている．各ノードにおける実行配置管理システムはセッション層で動作することによりサーバとクライアント間の通信メッセージに合わせて管理情報を重畳させ，ネットワーク内資源監視機能を実現する．同時にネットワーク内資源の変化を常時監視し，変化が見られた場合に候補選出機能を実行し，必要と判定されれば選択されたノードに向けてサービス移送機能を実行しサーバの実行地点を決定する．

4.　課題

我々は，このシステムを改善してフォグネットワーク上の処理能力の不均一性から生じる実行地点の最適化の問題の解決を図る．これまでのこのシステムの課題として，ネットワークの遅延時間のみをサービス実行地点の候補選出基準にしており，フォグノードの処理能力は考慮していない点が挙げられる．フォグネットワーク上のノードの処理能力が均一であると仮定したときには，ネットワークの遅延時間のみを考慮したサービス実行地点の移送を行うことで，クライアントがサービスにリクエストを送ってレスポンスが帰ってくるまでのサービスレスポンスタイムが短くなり，クライアントに有益なサービス実行地点の最適化ができる．しかし，フォグネットワーク上のノードの処理能力が不均一と仮定すると，どのノードでサービスを実行するかで，ネットワークの遅延時間だけでなく，サービス処理時間が変動する．したがって，ネットワークの遅延時間だけの候補選出基準では，サービス応答時間の評価ができない．さらにサービス最適化のための探索戦略として遅延時間の経路における単調増加性から 1 ホップの移送を繰り返す単純な方式を採ってきたがこれも見直す．これまでのシステムでは，サーバは隣接するノードの情報を用いて，それら隣接するノードのみを候補として移送を行っていた．

5.　提案システム

我々が提案したシステム [4] に追加した点について説明を行う．フォグネットワークに存在するノードの処理能力の不均一性を扱うために，ノードの処理能力に基づくサービス処理時間を定義し候補ノード選出基準に利用する．さらに，サーバノードに接続しているクライアントノードの到達経路上のノードならびに経路のホップ毎の推定遅延時間を収集し，サービス処理時間の推定に利用する．これにより，提案システムが候補を選出する際にサーバノードが到達可能な全クライアントとの経路上の全てのノードを移送候補として移送候補選出の判断を行うことで，サーバと 2 ホップ以上離れた位置に目的特化ユニットが搭載されたノードが存在した場合にそのノードへ移送を行うことを可能にした．この資源の探索動作は前節で述べたようにクライアントからのリクエストメッセージに重畳させることが可能なため，オーバーヘッドを抑えることが可能である．

5.1　候補ノード選出基準

何らかの性能指標の向上がサーバ移送の基準となるが，ここではクライアントノードから見たサービス応答時間の短縮を取り扱う．サービス応答時間はサービス処理時間とサーバまでの往復遅延時間の和で表現できる．次にサービス処理時間を考える．

サービス処理時間

ノードの処理能力 C や，ノードが実行する要求されたサービスの処理量 L を，CPUC と，目的特化ユニット T に分解して表現するため，それぞれ 2 次元のベクトルで示す．フォグノードの処理能力を (C_C, C_T) で表し，サービス実行に必要な処理量を (L_C, L_T) と定義する．これら処理能力と処理量から，サービス処理時間 T_{est} を以下に定義

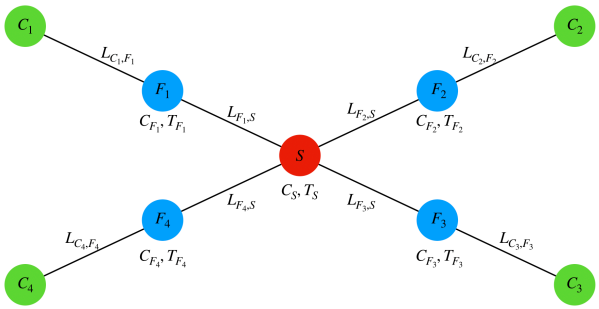

図 4　システム構成例 (S：サーバーノード, F：フォグノード, C：クライアントノード, $L_{A,B}$：A から B までの通信遅延, C_X：X の CPU 処理能力, T_X：X の目的特化ユニットの処理能力)

する．

$$T_{est}(C_C, C_T, L_C, L_T, \alpha) = \begin{cases} \frac{1}{C_C}(L_C + \frac{L_T}{\alpha}) & (C_T = 0) \\ \max(\frac{L_C}{C_C}, \frac{L_T}{C_T}) & (otherwise) \end{cases}$$

ただし，CPU は目的特化ユニットに要求された処理を行うことができるものとし，その比を表現する係数 α は 5 とした．

5.2　メッセージ到達経路上の PCEL 情報収集

各クライアントノードからサーバまでの経路上に存在する全てのノードを移送候補として扱うために，経由したリンク間の遅延と，経由したノードの処理能力の情報を収集する必要がある．これらを経路上の PCEL (available Processing Capacity and Estimated Latency：利用可能な処理能力と通信状況) 情報と呼ぶ．本システムではリクエストメッセージがサーバノードに到達するまでに通過したノードにて，PCEL 情報をリクエストメッセージに追加することでサーバに伝達する．

例えば図 4 のように本システムが利用されている場合，クライアントノード C_1 からのリクエスト・メッセージがそれぞれのフォグノードを経由した際に C_1 からフォグノード F_1 までのリンクの通信遅延である L_{C_1,F_1} と同様にサーバーノード S から F_1 までのリンクの通信遅延である $L_{F_1,S}$ と F_1 の CPU 処理能力である C_{F_1} と F_1 の目的特化ユニットの処理能力である T_{F_1} のパラメータを PCEL 情報としてリクエストメッセージに付加する．この付加された情報を S はリクエストメッセージから取得することができる．同様に S は全ての参加クライアントノードである C_1 から C_4 からのリクエストメッセージに付加されている PCEL 情報から全クライアントとの経路上の情報を取得することができる．

5.3　候補ノード選出アルゴリズム

サーバは，クライアントから受信したリクエストメッセージに付加された PCEL 情報から Algorithm 1 に示すアルゴリズムを用いて，移送候補ノードを選出する．Algorithm 1 は，4 で示したサーバがリクエストメッセージから取得することができる情報を元に，5.1 で示した評価基準から算出されるサービス応答時間を得て，最小になるノードを見つけている．

Algorithm 1 Find Candidate Node

Require: L_{All}:経路上の L のリスト
Require: F_{All}:経路上の F のリスト
Require: L_C:サービス実行時の CPU 処理量
Require: L_T:サービス実行時の目的特化ユニット処理量
Require: α:CPU が目的特化ユニットの処理量をこなすときの比率を表す係数
Ensure: $MinNode$ は　移送候補ノード
  $MinCost \leftarrow \infty$
  for all $node$ in F_{All} **do**
    $Cost \leftarrow (T_{est}(node.C_C, node.C_T, L_C, L_T, \alpha) + \sum L_{All} * 2)$
    if $MinCost > Cost$ **then**
      $MinCost \leftarrow Cost$
      $MinNode \leftarrow node$
    end if
  end for
  return $MinNode$

6.　実験評価

提案システムの有用性を示すために，ネットワークシミュレータを用いた実験評価を行う．前提事項とシミュレーション結果，考察について順に述べる．提案システムの有用性の検証に向けて，ユースケースを用意し，それに適した管理対象のネットワーク内資源と候補選出基準を定義し，シミュレータ上に本システムを実装し動作の確認を行う．まず，不均一な計算資源に対する我々の従来のシステムでの移送によるサービス応答時間の最適化の様子を確認する．次に，サービス実行地点から 2 ホップ以上離れた位置に目的特化ユニットを持つノードが存在する場合には探索されない可能性があることに留意し，拡張された提案手法を用いた最適化の結果を示し有効性を確認する．評価するにあたり，シミュレーションソフトウェアには，既存のネットワークシミュレーションソフトウェアである ns3[16] に CCN の実装の 1 つである NDN の拡張モジュールを追加した ndnSIM[17] を用いる．

提案システムの有用性を示すにあたり以下のようなユースケース，及び，シミュレーションシナリオにおいて実験を行った．

6.1　ユースケース

フォグネットワーク上で機械学習を用いるサービスを想定する．効率良く機械学習の処理を行うには，TPU (Tensor Processing Unit) のような専用の目的特化ユニットがあると良い．しかし，すべてのフォグノードに対して，目的特

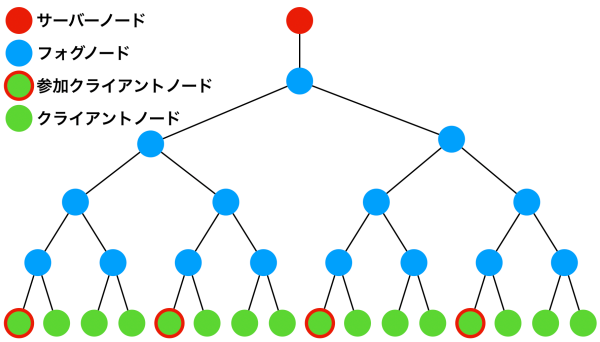

図 5　実験に用いたネットワークのトポロジ

化ユニットを搭載するのはコストの面で不利になる可能性がある．そこで，フォグネットワーク上の一部を専用ユニット搭載ノードで構成することで，コストの問題を解決する．フォグネットワークは通常の CPU のみのノードと目的特化ユニットを搭載したノードが混在しており，ある一点の近傍での処理能力は不均一になる．このような状況において，機械学習のような目的特化ユニットによって加速される処理を，適切な処理能力をもったノードにサービス実行点を移送することで，サービス応答時間を最小に保つことを目標とする．このユースケースにおける実験で，フォグネットワーク上に TPU のような目的特化ユニットを搭載したノードを全体に対して一定の割合でランダムに配置し，ネットワーク全体においてサービス応答時間の平均が想定通りに短縮できること明らかにする．

6.2　シミュレーションシナリオ

実験の目的を達成するため以下のシミュレーションシナリオを用意した．ネットワークに繋がる複数のクライアントは，遠方のクラウド上のサーバと，様々な処理能力をもったフォグノードが利用可能である．クライアントは TPU のような目的特化ユニットの処理能力が活用可能なサービス要求を行う．この実験では，フォグノード全体の約何%配置することでサービス応答時間が最小化できるかを明らかにするため，フォグネットワーク内の専用ユニットを搭載したノードの割合を 0%から 100%まで 20%ごとに変え，配置をランダムに変化させた実験を 100 回試行した．

シミュレーションの際に用いたネットワークトポロジは図 5 のような 2 分木のトポロジを用いた．表 1 にシミュレーションに用いたパラメータを記す．それぞれのノードが持ち合わせるコンテンツキャッシュの容量についても，サービスの実行に必要なデータをすべてキャッシュするには十分な容量を持っていると仮定して行った．

表 1　シミュレーションパラメータ

パラメータ	設定値
Server 数	1
FogNode 数	15
参加 Client 数	4
キャッシュアルゴリズム	LRU
回線容量	10Mbps
伝搬遅延 (Server-Fog)	20ms
伝搬遅延 (その他)	2ms
シミュレーション時間	100s
Server の C_C	100.0
Server の C_T	100.0
専用ユニットの C_C	20.0
専用ユニットの C_T	50.0
普通ユニットの C_C	20.0
普通ユニットの C_T	0
実行サービスの L_C	50.0
実行サービスの L_T	100.0

7.　結果と考察

7.1　従来方式での最適化

まず，不均一な計算資源に対する我々の従来のシステムでの移送によるサービス応答時間の最適化の様子を確認する．従来方式ではサービス実行地点，すなわち現在のサーバの位置から周囲 1 ホップのノードの PCEL 情報を収集しそれに基づいてサービス移送を判断していた．図 6 から図 11 は，それぞれトポロジ中の目的特化ユニット搭載ノードの配置割合を 0%から 100%にし, 移送候補の探索をサーバーの隣接ノードのみにした場合における，全ノードのサービス応答時間の平均の時刻推移を表したグラフである．図 6 にて，全てのノードを CPU のみのノードで構成した場合，最終的にサービス応答時間の平均が約 44ms で安定している．一方，図 11 の全てのノードを目的特化ユニット付きノードで構成した場合は，最終的にサービス応答時間の平均が約 20ms で安定している．図 7 から図 9 では，最終的なサービス応答時間の平均は約 24ms で安定している．最適と最悪値の 2 値のみ存在すると仮定すると，20-60%の間では，80%ほどの確率で最適解であるサービス応答時間の平均が約 20ms に到達しているが，2 ホップ先以上離れたノードに最適解があった場合である 20%の確率で最適化できていないサービス応答時間の平均が約 44ms になってしまっていると見ることができる．図 10 の実験結果を見ると，最終的なサービス応答時間の平均が約 20ms で安定している．80%以上の配置だとおおよそサービス応答時間の最適化ができている．

7.2　提案方式での最適化

従来方式の単純な 1 ホップの探索方法では，サービス実行地点から 2 ホップ以上離れた位置に目的特化ユニットを持つノードが存在する場合には探索されない可能性があることに留意し，サーバから見て到達可能なクライアントとの通信経路上に存在するノード全体の PCEL 情報を用いて探索を行う機能を実装し評価した．それが図 12 から図 17 である．1 ホップ先までのみの探索の結果と比べ，経

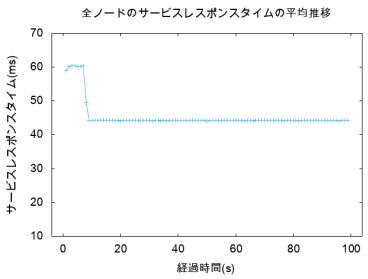

図 6 1 ホップ探索:配置 0%:ワーストケース

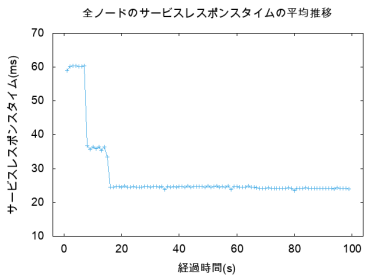

図 7 1 ホップ探索:配置 20%

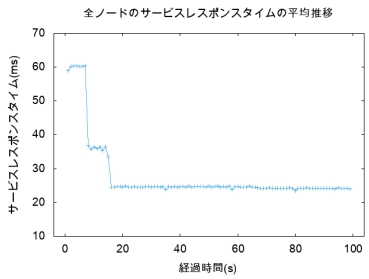

図 8 1 ホップ探索:配置 40%

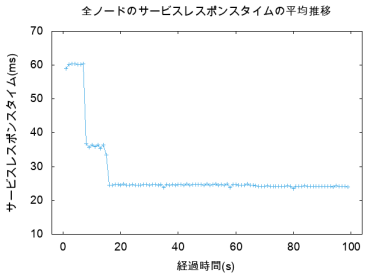

図 9 1 ホップ探索:配置 60%

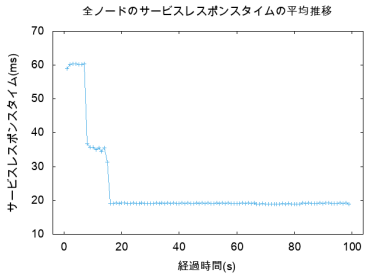

図 10 1 ホップ探索:配置 80%

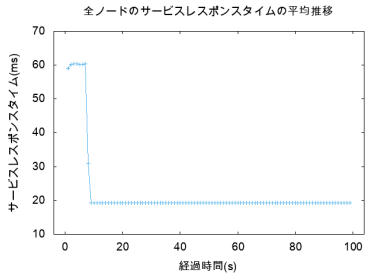

図 11 1 ホップ探索:配置 100%:ベストケース

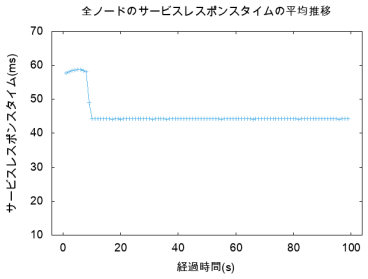

図 12 到達ノード:配置 0%:ワーストケース

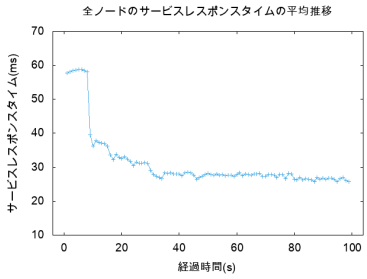

図 13 到達ノード:配置 20%

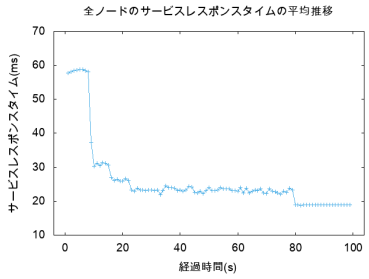

図 14 到達ノード:配置 40%

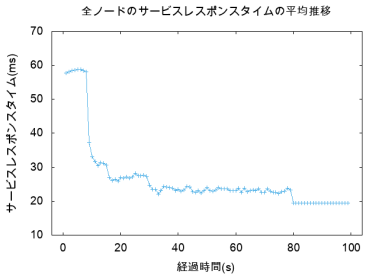

図 15 到達ノード:配置 60%

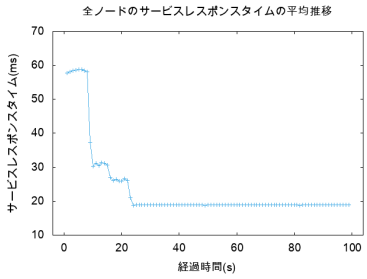

図 16 到達ノード:配置 80%

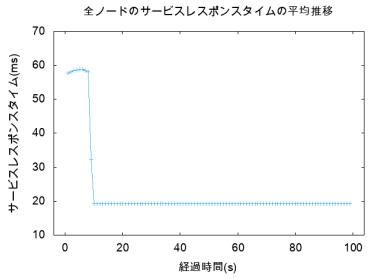

図 17 到達ノード:配置 100%:ベストケース

路上に目的特化ノードが存在した場合には，確実に発見できるため，図 6 から図 11 と比べ，最終的に安定したサービス応答時間の平均が全体的に低くなっている．最終的なサービス応答時間は最大で 40 から 60%の割合で目的特化ユニットを持つノードを配置した場合に約 17%向上した．

8. おわりに

フォグコンピューティングの研究では，動的サービスの実行地点の最適化が議論されており，CCN の研究では，コンテンツの配置の最適化について議論されている．そこで我々は，CCN とフォグコンピューティングを組み合わせ，自律的なサービスの実行地点選択と資源配置のためのフォグコンピューティングとインネットワークキャッシュの融合の実現を本研究の目的においた．目的の実現のために，実行処理とデータ配置の最適化を実現可能なアーキテクチャを設計し有効性を確認するため，実行処理の配置を自律的に最適化する実行配置管理システムを提案した．実行配置管理システムは，フォグノードのセッション層にネットワーク内資源監視機能，候補選出機能，サービス移送機能を実装することで，アプリケーション層で動くサーバやクライアントを意識することなく，フォグネットワークにおいて自律的にサービスの実行地点選択と資源配置を実現するものである．本研究は，機械学習などの目的に特化した処理ユニットの利用を考慮し，フォグネットワーク内の

計算能力の不均一性に対処するためにこれまでの実行配置管理システムの候補選出基準にサービス応答時間を加え，クライアントとの経路上の利用可能な計算資源を発見すべく探索手法を拡張した．提案したシステムをシミュレーション上に実装し，処理能力が不均一なフォグネットワークの場合にサービス実行地点の最適化が行われることを示した．

今後，実験の際にシミュレータとして用いた ndnSIM に内部的に利用されている NDN のプロトタイプ実装 NDN Platform[18] を用いるなどして，IP ネットワーク上にオーバーレイした CCN ネットワークを構築し，その CCN ネットワーク上で動作する本システムを設計・実装し，検証を進めていく．

参考文献

[1] Ministry of Internal Affairs and Communications, Japan: 平成 30 年版情報通信白書，Japanese Government (2018).

[2] Bonomi, F., Milito, R., Zhu, J. and Addepalli, S.: Fog Computing and Its Role in the Internet of Things, *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, New York, NY, USA, ACM, pp. 13–16 (online), DOI: 10.1145/2342509.2342513 (2012).

[3] Jacobson, V., Smetters, D. K., Thornton, J. D., Plass, M. F., Briggs, N. H. and Braynard, R. L.: Networking Named Content, *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '09, New York, NY, USA, ACM, pp. 1–12 (online), DOI: 10.1145/1658939.1658941 (2009).

[4] 鎌田幸希，稲村　浩，中村嘉隆：フォグコンピューティングにおける実行配置管理システムの提案，技術報告 1，公立はこだて未来大学システム情報科学部, 公立はこだて未来大学システム情報科学部, 公立はこだて未来大学システム情報科学部 (2018).

[5] Jouppi, N. P., Borchers, A., Boyle, R., Cantin, P.-l., Chao, C., Clark, C., Coriell, J., Daley, M., Dau, M., Dean, J., Gelb, B., Young, C., Ghaemmaghami, T. V., Gottipati, R., Gulland, W., Hagmann, R., Ho, C. R., Hogberg, D., Hu, J., Hundt, R., Hurt, D., Ibarz, J., Patil, N., Jaffey, A., Jaworski, A., Kaplan, A., Khaitan, H., Killebrew, D., Koch, A., Kumar, N., Lacy, S., Laudon, J., Law, J., Patterson, D., Le, D., Leary, C., Liu, Z., Lucke, K., Lundin, A., MacKean, G., Maggiore, A., Mahony, M., Miller, K., Nagarajan, R., Agrawal, G., Narayanaswami, R., Ni, R., Nix, K., Norrie, T., Omernick, M., Penukonda, N., Phelps, A., Ross, J., Ross, M., Salek, A., Bajwa, R., Samadiani, E., Severn, C., Sizikov, G., Snelham, M., Souter, J., Steinberg, D., Swing, A., Tan, M., Thorson, G., Tian, B., Bates, S., Toma, H., Tuttle, E., Vasudevan, V., Walter, R., Wang, W., Wilcox, E., Yoon, D. H., Bhatia, S. and Boden, N.: In-Datacenter Performance Analysis of a Tensor Processing Unit, *Proceedings of the 44th Annual International Symposium on Computer Architecture - ISCA '17*, Toronto, ON, Canada, ACM Press, pp. 1–12 (online), DOI: 10.1145/3079856.3080246 (2017).

[6] Cuervo, E., Balasubramanian, A., Cho, D.-k., Wolman, A., Saroiu, S., Chandra, R. and Bahl, P.: MAUI: Making Smartphones Last Longer with Code Offload, ACM Press, p. 49 (online), DOI: 10.1145/1814433.1814441 (2010).

[7] Chun, B.-G., Ihm, S., Maniatis, P., Naik, M. and Patti, A.: CloneCloud: Elastic Execution between Mobile Device and Cloud, ACM Press, p. 301 (online), DOI: 10.1145/1966445.1966473 (2011).

[8] Kosta, S., Aucinas, A., Hui, P., Mortier, R. and Zhang, X.: ThinkAir: Dynamic Resource Allocation and Parallel Execution in the Cloud for Mobile Code Offloading, *2012 Proceedings IEEE INFOCOM*, pp. 945–953 (online), DOI: 10.1109/INFCOM.2012.6195845 (2012).

[9] Berg, F., Dürr, F. and Rothermel, K.: Increasing the Efficiency of Code Offloading in N-Tier Environments with Code Bubbling, *Proceedings of the 13th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, MOBIQUITOUS 2016, New York, NY, USA, ACM, pp. 170–179 (online), DOI: 10.1145/2994374.2994375 (2016).

[10] 山本 真由，重安哲也：履歴に基づいたコンテンツ要求予測による先行キャッシュ配信手法の提案，技術報告 9，県立広島大学経営情報学科, 県立広島大学経営情報学科 (1 月 2018).

[11] 神本 崇史，佐藤 和也，重野 寛：Information Centric Networking における人気度の収集と通知を用いたキャッシング手法，情報処理学会論文誌，Vol. 58, No. 2, pp. 333–342 (2 月 2017).

[12] 國安 哲郎，重安哲也：適応的なリクエストコンテンツ制御を導入した WSN 向けコンテンツ指向型データ収集手法，情報処理学会論文誌，Vol. 59, No. 2, pp. 404–414 (2 月 2018).

[13] Shanbhag, S., Schwan, N., Rimac, I. and Varvello, M.: SoCCeR: Services over Content-Centric Routing, *Proceedings of the ACM SIGCOMM Workshop on Information-Centric Networking - ICN '11*, Toronto, Ontario, Canada, ACM Press, p. 62 (online), DOI: 10.1145/2018584.2018600 (2011).

[14] Wolf, T.: Service-Centric End-to-End Abstractions in Next-Generation Networks, *Proceedings of 15th International Conference on Computer Communications and Networks*, Arlington, VA, USA, IEEE, pp. 79–86 (online), DOI: 10.1109/ICCCN.2006.286249 (2006).

[15] Dorigo, M. and Birattari, M.: Ant Colony Optimization, *Encyclopedia of Machine Learning* (Sammut, C. and Webb, G. I., eds.), Springer US, Boston, MA, pp. 36–39 (online), DOI: 10.1007/978-0-387-30164-8_22 (2010).

[16] Henderson, T. R., Lacage, M., Riley, G. F., Dowell, C. and Kopena, J.: Network Simulations with the Ns-3 Simulator, *SIGCOMM demonstration*, Vol. 14, No. 14, p. 527 (2008).

[17] Mastorakis, S., Afanasyev, A. and Zhang, L.: On the Evolution of ndnSIM: An Open-Source Simulator for NDN Experimentation, *ACM Computer Communication Review* (2017).

[18] NDN Project: Libraries / NDN Platform, NDN Project (online), available from ⟨https://named-data.net/codebase/platform/⟩ (accessed 2019-01-28).