修士論文

エッジコンピューティング環境における
ユーザの移動性を考慮した
コンテナの同期先の選択手法の提案

仲野 弘一

2019年3月15日

奈良先端科学技術大学院大学
情報科学研究科

本論文は奈良先端科学技術大学院大学情報科学研究科に
修士 (工学) 授与の要件として提出した修士論文である。

仲野 弘一

審査委員：

藤川 和利 教授	（主指導教員）
笠原 正治 教授	（副指導教員）
新井 イスマイル 准教授	（副指導教員）
市川 昊平 准教授	（副指導教員）

エッジコンピューティング環境における ユーザの移動性を考慮した コンテナの同期先の選択手法の提案*

仲野 弘一

内容梗概

スマートデバイスの普及により、多数のセンサから発生した大量のデータがクラウド上のデータセンタに届くため、デバイスとデータセンタ間の通信量を削減し、ネットワーク全体に流れるデータ量を削減するエッジコンピューティングと呼ばれる技術の研究が盛んに行われている。ユーザはアクセスポイントを経由してエッジサーバ内のアプリケーションと通信を行うため、ユーザの移動により異なるアクセスポイントに接続した場合、異なるエッジサーバへ通信を行う可能性がある。通信するエッジサーバが異なる場合、異なるエッジサーバを経由して移動前に通信していたエッジサーバへアクセスする必要があるため、通信遅延が大きくなる恐れがある。そのため、ユーザの移動に合わせてエッジサーバ内のアプリケーションをユーザの現在アクセスしているエッジサーバへ移行することで、ユーザとの通信遅延を低下させる必要がある。しかし複数のユーザが移動する場合アプリケーションの移行先として同一のエッジサーバを選択する可能性があり、エッジサーバが一時的に高負荷になる恐れがある。そのため、本研究ではエッジサーバ上で実行されている仮想マシンやコンテナを移行させる際に特定のエッジサーバが高負荷にならないように負荷を分散させながらアプリケーションの移行時間の削減を目的とする。提案手法として仮想マシンおよびコンテナの移行時間とエッジサーバの負荷を考慮することで、移行に発生するダウンタイムを抑えな

*奈良先端科学技術大学院大学 情報科学研究科 修士論文, 2019 年 3 月 15 日.

がらエッジサーバの負荷分散手法を提案する。提案手法をシミュレータ上に実装し、評価を行い他手法と比較してほぼ同等または移行時間の削減が可能となった。

キーワード

エッジコンピューティング, 負荷分散, 仮想マシン移行

Proposal of a Selection Method of Synchronization Destination of Container Considering User Mobility in Edge Computing Environment*

Hirokazu Nakano

Abstract

Due to the spread of smart devices, a large amount of data generated from many sensors arrives at the data center on the cloud, so edge computing that reduces the amount of communication between the device and the data center and reduces the amount of data flowing throughout the network Research on called technology is actively conducted. However, when the movement of the device occurs, the communication distance between the edge servers communicating with each other increases, so that communication delay and an increase in communication amount of the entire network due to an increase in communication path between the edge server and the device occur As a result, the user experience may decrease. Therefore, in this research, we aim to not lower the user experience by migrating the virtual machines and containers running on the edge server to the edge server which the user may move. In this paper, we propose a migration method that minimizes traffic and communication delay caused by migration considering user mobility as proposed method. The proposed method is implemented on a simulator and evaluated, and it is possible to reduce the transition time almost the same as other methods.

*Master's Thesis, Graduate School of Information Science, Nara Institute of Science and Technology, March 15, 2019.

Keywords:

Edge Computing, Load Balancing, Virtual Machine Migration

目 次

図 目 次

表 目 次

1. 序論

Internet of Things (IoT) の普及によりセンサやモバイルデバイスなどから生成されるデータが年々増大している。シスコシステムズ合同会社の調査によれば 2021 年までに全世界の IP トラフィックは 3.3ZB まで達すると予測している [1]。特にワイヤレスデバイスやモバイルデバイスからのトラフィックの割合は 2021 までに 63%を越えると予想されている。今後 IoT を用いたサービスが増加するにつれて、この割合は増加を続けると予想される。現在主流であるクラウドコンピューティングではユーザは地理的に離れたデータセンタに対してサービスを要求し、応答を得る。しかし、ユーザとデータセンタ間の遅延は数百ミリ秒に及ぶ場合があり、自動運転の制御に要求されるような数ミリ秒内の遅延に抑えることができない。そこでユーザ側に近いネットワークの端 (エッジ) に小規模なサーバやゲートウェイを配置し、データの処理を行う仮想マシンやコンテナを動作させることで低遅延のサービスを提供するエッジコンピューティングと呼ばれるコンピューティングパラダイムが注目されている [2][3]。これによりユーザからのデータをクラウドのデータセンタではなく、ユーザに近いエッジサーバ上でデータ処理を行うことで低遅延なサービスを提供可能となる。

エッジコンピューティングの重要な課題の一つにユーザの動きに合わせたエッジサーバ上のアプリケーションのシームレスな移行が存在する [4]。エッジコンピューティングでは各エッジサーバに複数のアクセスポイントが接続されているケースが多い。ユーザがアクセスポイントの通信可能範囲から移動する場合、次にユーザが通信するアクセスポイントに接続されているエッジサーバは異なる可能性がある。エッジサーバが異なる場合、ユーザは移動前にアクセスしていたエッジサーバのアプリケーションへ通信する必要があるため、ユーザはサービスの要求から結果を受け取るまでの遅延が大きくなる可能性がある。そのため、ユーザの移動に合わせてエッジサーバ内のアプリケーションをある時点で通信しているエッジサーバへ移行することで、アプリケーションとの通信遅延を短縮することが重要となる。しかし複数のユーザが移動する場合、アプリケーションの最適な移行先として同一のエッジサーバを選択する可能性があり、エッジサーバが一時的に高負荷になる恐れがある。そのため、移動するユーザへの影響を抑えながら

ユーザの移動先のアクセスポイントに接続されているエッジサーバが高負荷にならないようにエッジサーバ内のアプリケーションを移行する必要がある。

本研究ではエッジサーバ上で実行されている仮想マシンやコンテナを移行させる際に、特定のエッジサーバが高負荷にならないように負荷を分散させつつ移行時間の削減を目的とする。提案手法として移行先を決定するための移行コストの計算にエッジサーバの負荷と移行時間を導入することで、移行によるユーザの通信遅延を最小限に抑えながらエッジサーバの負荷分散を行う。

本論文の構成について説明する。2 章では研究背景として現在主流のコンピューティングパラダイムであるクラウドコンピューティングについて説明した後、エッジコンピューティングおよびエッジコンピューティングに適したサービスについて説明する。3 章ではエッジコンピューティングにおける仮想マシンおよびコンテナの移行の関連研究について説明する。4 章ではユーザの移動を考慮したコンテナ移行の提案を行う。5 章では提案手法をシミュレータ上で動作させ、評価実験および実験結果について考察を行う。6 章で本研究についてまとめる。

2. 研究背景

本章ではクラウドコンピューティングとエッジコンピューティングについて説明する。

2.1 クラウドコンピューティング

クラウドコンピューティングとは、遠隔地に存在するソフトウェアやハードウェア、ストレージ、メモリなどといった計算資源に利用者がネットワーク経由でどこからでもアクセスし利用可能とするコンピューティングモデルである。図 1 にクラウドコンピューティングの概要図を示す。クラウドコンピューティングはクラウド事業社が利用者に提供するサービスの階層に応じて SaaS (Software as a Service), PaaS (Platform as a Service), IaaS (Infrastructure as a Service) に分類できる。クラウドコンピューティングの利点として利用者によるハードウェア管理が不要で，必要なときに必要なだけリソースを追加してスケールアウトできることなどが挙げられる。クラウドコンピューティングの問題点として計算リソースがデータセンタ内に集中しているため、大量のデータが短時間で集中して送信されるとネットワーク帯域の圧迫や通信遅延の増加などサービス品質の低下がある。

2.2 エッジコンピューティング

エッジコンピューティングとは、可能な限りデータ処理を行うアプリケーションなどをネットワークの端（エッジ）に存在する小規模サーバ (エッジサーバ) に配置し，クラウドと連携しながらサービスを提供するコンピューティングモデルである。図 2 にエッジコンピューティングの概要図を示す。似たような概念として、Bonomi らが提案したフォグコンピューティング (Fog Computing)[5] や European Telecommunications Standards Institute (ETSI) が研究開発しているモバイルエッジコンピューティング (Mobile Edge Computing) などが存在する。エッジサーバ上でデバイスから生成されたデータを処理し即座に結果を返すことで低遅延なサービスの提供やネットワーク全体を流れるデータ量の削減が可能となる。またクラ

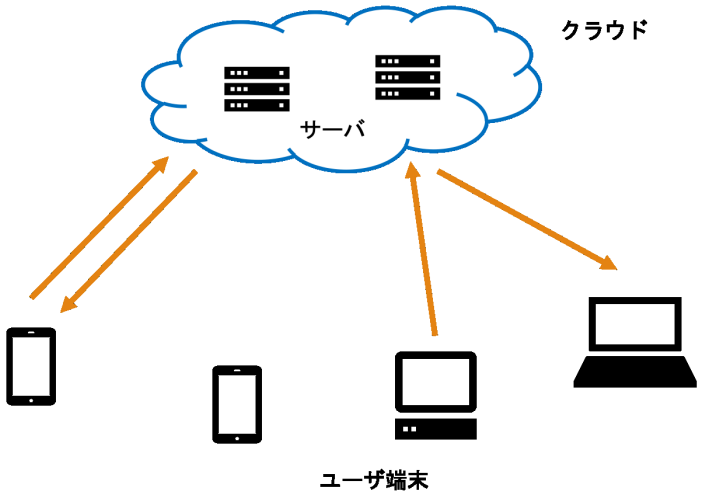

図 1　クラウドコンピューティングの概要図

ウド上のサーバは各エッジサーバから加工処理されたデータを受け取り、データに基づいてエッジサーバの制御を行う。Amazon Web Service (AWS) や Microsoft Azure、Google のような大手のクラウドベンダは自社のクラウドインフラ基盤と専用のソフトウェアをインストールしたデバイスを経由して接続することでセンサからのデータをクラウドへアップロードし、制御を行うサービスを提供している [6][7][8]。欠点としては、エッジサーバはクラウドコンピューティングにおけるデータセンタのような高性能な計算機を配置することが困難なため、リソース管理がより重要になったり、ユーザの移動に合わせたサービスの提供など様々な課題が存在する。

Dolui、Chao、Vaquero らはフォグ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などのエッジ付近で計算を行う同様のアーキテクチャについて調査している [9][10][11]。表 1 に上記 2 つの特徴をまとめる。

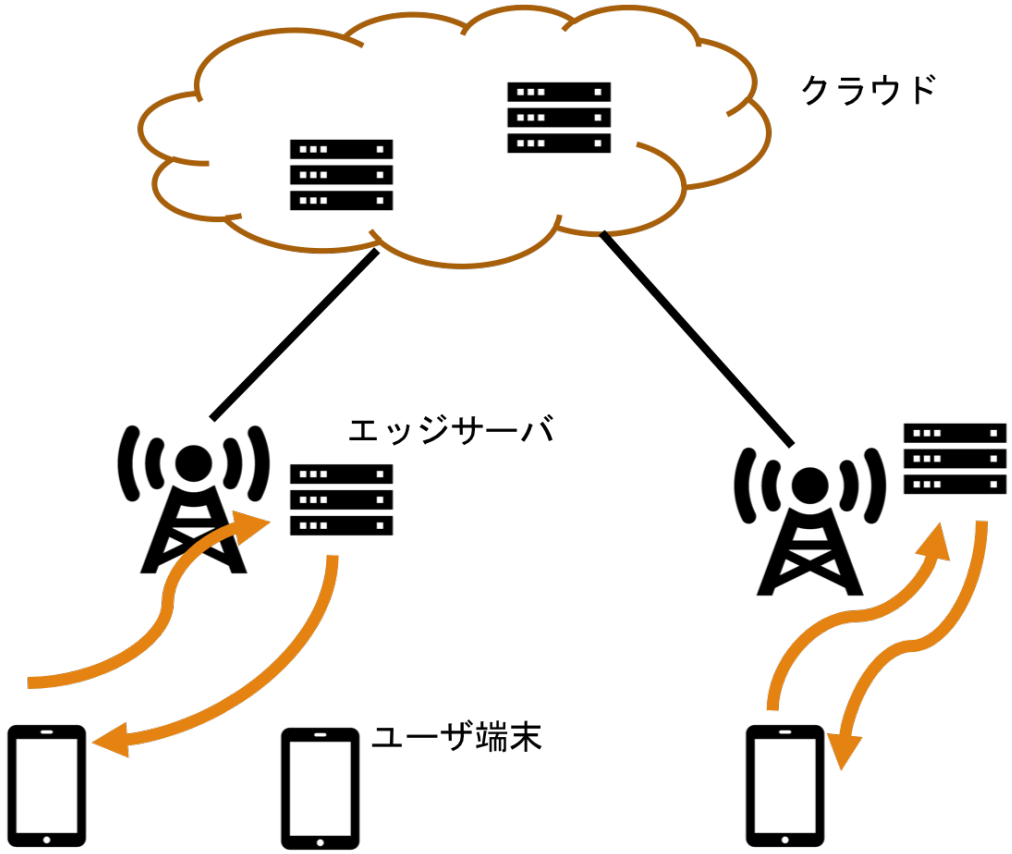

図 2　エッジコンピューティングの概要図

表 1 エッジコンピューティングアーキテクチャの説明

	Fog Computing	Mobile Edge Computing
ノードの種類	ルータやスイッチ等	基地局内のサーバ
ノードの位置	エッジ部分からクラウドまで	無線ネットワーク
通信遅延	小	小
ホップ数	1 または複数	1

フォグコンピューティングとモバイルエッジコンピューティングの大きな違いとしてエッジサーバに該当するノードの部分がフォグコンピューティングの場合ルータやスイッチなどのネットワーク機器が担当している点が挙げられる。またルータやスイッチ内部に配置したアプリケーションにセンサデータの処理などをさせるため、末端ノードでは処理能力が足りず処理機能が高いノードまで複数ホップ移動しないといけない可能性がある。また、モバイルエッジコンピューティングは携帯電話事業者を対象にしているが、フォグコンピューティングでは一般のネットワークを対象としている。

2.3 エッジコンピューティングを活用した IoT サービス例

スマートデバイスの普及により、様々な IoT サービスが登場すると予測される。ここではエッジコンピューティングの特徴を利用したサービスを複数説明する [12]。

- コネクテッドカーを用いた自動運転
  Information and Communication Technology (ICT) 機能を備えた車両 (コネクテッドカー) を用いたサービスとして、車両に設置されたセンサや路上に設置されたセンサなどから得られるデータを処理し、事故や渋滞の回避などの情報を即座に車両に伝えることが可能となる。また大量のコネクテッドカーから生成されるデータの処理をクラウドではなくエッジサーバで実行することでネットワーク使用量も削減することが可能となる。

- ウェアラブルセンサを用いたスマートヘルスケア
  身体に心電図や心拍数などのデータを取得可能なセンサを装着し、健康情報をモニタリングすることで身体の異変を事前に検知したり、異常時はセンサから自動的に緊急通報することも可能である。従来のクラウドコンピューティング環境ではデータの取得から検知までの遅延が緊急患者にとっては非常に重要になってくる。その為、ユーザの近くでデータを収集し異常を瞬時に判断する状況でエッジコンピューティングは有効に作用する。

- リアルタイム画像認識
  セキュリティの向上を目的として人物・物体認識など監視カメラで撮影した映像をリアルタイム処理する必要のある状況が多く存在する。しかし監視カメラ自体にはリアルタイムな動画解析処理を行えるほどのリソースを保持していない。その為、現状では監視カメラから収集した動画データを解析する為にはクラウドで処理をする必要がある。しかし、常時動画データをクラウドに送信することはネットワークの帯域やクラウドのリソース資源の制限などのため困難である。監視カメラで取得した動画データを監視カメラの近くのエッジサーバに送信し処理することで、帯域や遅延に関する課題を解決可能である。

2.4 IoT サービスの課題

上記の IoT サービス例から様々な IoT サービスの課題として以下の要求事項を満たす必要があると予想される。

- デバイスやセンサから発生する大量のデータの効率的な処理
  デバイスやセンサから発生するデータは多種多様であり、動画データや単純な文字列のデータなどを収集しデータの分析を行う必要があるため、データの種類に応じて処理の優先度を変更する必要がある。

- 突然の高負荷に対処可能な負荷分散
  エッジサーバでは固定されたデバイスから送信されるデータだけでなく、移

動デバイスから送信されるデータも処理する必要がある。複数の移動デバイスが特定のエッジサーバに対してデータの送信やサービスの要求をすることによりエッジサーバの処理限界を超えるのを防ぐため、他のエッジサーバやクラウドと連携し負荷を分散させる必要がある。

- ネットワークに対する負荷の軽減
  デバイスから送信されるデータは一つ一つが少なくともデバイスが大量に存在すると膨大なトラフィック量になる。そのトラフィックが直接クラウドへ送信されるとバックボーンネットワークに負荷をかけるだけでなく、クラウドのデータセンタも高負荷になる。エッジサーバがデータの前処理を行うことで不要なデータをネットワークへ流さないようにすることが重要である。

- ユーザの移動に合わせた低遅延なサービスの提供
  移動するユーザのデバイスにサービスを提供する場合、あるエッジサーバとのみ通信を行なっているとエッジサーバとデバイスの距離が離れていくにつれ、データがネットワーク内で経由するホップ数が増えるため通信遅延が増加する。常にデバイスに近い場所でエッジサーバと通信することで、移動していても低遅延なサービスを提供可能となる。

本論文では 2.4 節に示した要求事項のち、(1) 突然の高負荷に対応可能な負荷分散と (2) ユーザの移動に合わせた低遅延なサービスの提供を目的とする。

3. 関連技術

本章ではエッジコンピューティング環境でサービスを提供する為に必要な技術について説明し、またその技術を用いた関連研究について説明する。

3.1 仮想マシン

仮想マシンとはコンピュータの動作をエミュレーションする技術であり、1 台のコンピュータ上で複数の仮想コンピュータを動作させることができる。仮想マシンを用いることで、一台のマシンの上で複数の Operating System (OS) を同時稼働させることが可能になる。仮想マシンは仮想化システムにより物理リソースと隔離されているため、安全性が高い。また、仮想マシンのある時点の環境をバックアップすることで環境構築の効率化や新技術の検証など様々な用途に利用される。仮想マシンのような OS 仮想化技術にはホスト型とハイパーバイザ型の 2 種類に大きく分けられる。ホスト型とは物理リソース上で実行している OS の上に OS 仮想化ソフトウェアのミドルウェアを動作させ、その上に仮想マシンを配置するものである。ハイパーバイザ型とは、物理リソースの上にハイパーバイザと呼ばれる仮想マシンを管理する専用の OS を動作させ、その上に仮想マシンを配置するものである。以下にホスト型とハイパーバイザ型の図を示す。ハイパーバイザ型はホスト型仮想化と異なりミドルウェアを介して操作しないため、ホスト型より性能低下が少ない。現在のクラウドコンピューティング上で動作しているサービスは基本的に仮想マシンの上で動作していることが多い。ホスト型の一例として Oracle VM VirtualBox[1]や VMware ESXi[2]などが存在する。ハイパーバイザ型では Kernel-based Virtual Machine (KVM)[3]や Xen[4]などが挙げられる。

[1] https://www.oracle.com/technetwork/server-storage/virtualbox/overview/index.html

[2] https://www.vmware.com/products/esxi-and-esx.html

[3] https://www.linux-kvm.org/page/Main_Page

[4] https://www.xenproject.org/

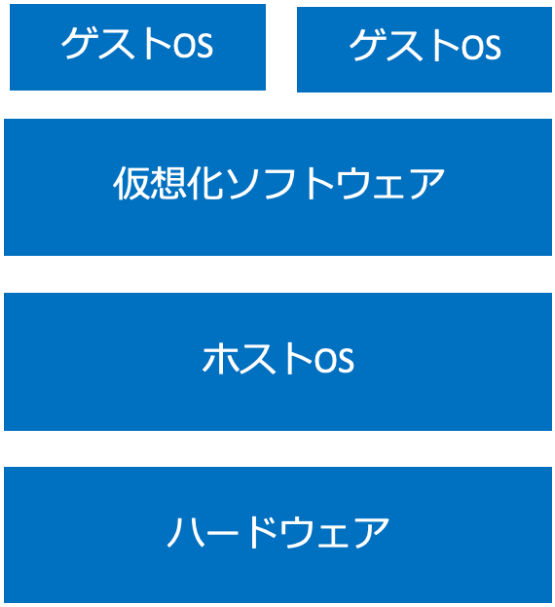

図 3　ホスト型仮想化

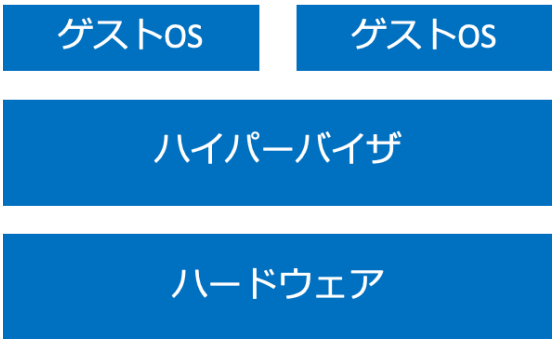

図 4　ハイパーバイザ型仮想化

3.2 コンテナ

上記の OS 仮想化では仮想マシンのハードウェア処理ををソフトウェアでエミュレーションしていたため、入出力にオーバーヘッドが存在した。近年 OS 仮想化と同様の隔離された環境を構築することが可能なコンテナが注目されている。コンテナは Linux Kernel の複数の機能を利用し構成されることが多い為、Linux コンテナとも呼ばれる。以下にコンテナの図を図示する。コンテナはハードウェアのエミュレーションを行わないため、OS の仮想化のようにオーバーヘッドがなく物理リソースの性能をほぼ有効活用することが可能である。コンテナの代表的なプラットフォームとして Docker[5]や Liux Containers (LXC)[6]があげられる。

図 5 コンテナ

3.3 マイグレーション

あるコンピュータ上に存在する仮想マシンは他のコンピュータに移行することができる。このことを仮想マシンのマイグレーションと呼ぶ。データセンタではサーバ上の仮想マシンはサーバの利用状況などから利用率の低いサーバの仮想マシンを他のサーバ内にマイグレーションし、電源をオフにすることでデータセンタ運用における電力削減が行われている。あるサーバのメンテナンスが必要な場合、メンテナンスが行われるサーバから仮想マシンを他のサーバにマイグレーショ

[5]https://www.docker.com/
[6]https://linuxcontainers.org/

ンしメンテナンスを行った後に仮想マシンを元のサーバに再びマイグレーションするといった柔軟な運用が可能となる。またマイグレーションには仮想マシンを停止することなく他のマシンに移行することが可能なライブマイグレーションと呼ばれる手法が存在する。ライブマイグレーションには大きく分けて Pre-Copy 方式と Post-Copy 方式, 両方を組み合わせた Hyblid 方式の 3 種類が存在する。ここでは代表的な Pre-Copy 型と Post-Copy 型について説明する。

- Pre-Copy 型
    1. マイグレーション対象の仮想マシンのメモリデータを転送する
    2. メモリの転送中にメモリの内容が書き換わった場合、その内容も転送を行う
    3. メモリの書き換わるサイズが十分少なくなるまで転送を繰り返す。その後仮想マシンを停止し、残りのメモリを全て転送する
    4. 転送先で仮想マシンを起動させる
- Post-Copy 型
    1. マイグレーション対象の仮想マシンの CPU レジスタおよび仮想マシン実行に必要最低限のデータを転送
    2. 転送先で仮想マシンを起動させる
    3. メモリアクセスによって発生したページフォールトを検知すると転送元のメモリページを取得
    4. メモリを転送

3.4 エッジコンピューティング環境での仮想マシンの移行に対する要求事項

2.4 節で述べた要求事項から仮想マシンの移行により改善が可能なものおよび発生する課題について述べる。

- デバイスやセンサから発生する大量のデータの効率的な処理
  処理するデータに最適なアプリケーションが格納された仮想マシンをデータ処理効率が最大になるように最適なエッジサーバへ移行させることで資源を有効活用することが可能となる。発生する課題として移行途中に仮想マシンが受け取る予定だったデータの対応や移行後のネットワークの経路制御などが必要となる。

- 突然の高負荷に対処可能な負荷分散
  ユーザ毎に仮想マシンを割り当てることでエッジサーバの負荷が上昇してきた場合、利用頻度の少ない仮想マシンを資源に余裕のあるエッジサーバへ再配置し直すことでエッジサーバの高負荷や消費電力を抑えることが可能となる。しかし再配置後の通信遅延や

- ネットワークに対する負荷の軽減

- ユーザの移動に合わせた低遅延なサービスの提供
  エッジコンピューティング環境ではユーザ毎に最も遅延の少なくなるようにエッジサーバに仮想マシンを配置することで通信遅延によるサービスへの影響を抑えることが可能となるが、移行によるダウンタイムの発生や仮想マシンを移行することによって発生するネットワークへの影響を最小限にする必要がある。

3.5 エッジコンピューティング環境におけるコンテナ

エッジコンピューティング環境のようなリソースが限られたような環境では効率良くサービスの運用やリソースの管理を行う必要がある。コンテナは仮想マシンと比較してホストマシンのリソースを効率良く利用可能なことや、コンテナの起動が高速であるなど様々な面でメリットが存在するため、エッジコンピューティングに適していると考えられる。エッジコンピューティング環境で小規模エッジサーバとして利用されると予想される Raspberry Pi の様なハードウェアに制約がある小型デバイスに Docker コンテナを展開することで安価かつ省電力でエッジコンピューティングプラットフォームが構築可能となる [13]。

4. 関連研究

エッジコンピューティングにおける仮想マシン (Virtual Machine, VM) のマイグレーションに関する研究が多く存在する。しかし VM は OS 全体を仮想化しているため転送サイズが大きく、ネットワーク全体に大きな負荷をかける恐れがある。図 6 のようなデータセンタ内で行われるような仮想マシンのライブマイグレーションではなく図 7 のようなエッジサーバ間での VM の移行は、高性能なサーバや広帯域なネットワークを持たないため、VM のサイズが大きくなるとエッジサーバ上で提供しているサービスにも影響を与える。VM より軽量な仮想化技術であるコンテナを使ったサービスの移行を行うことでネットワークに対する影響を抑えながら、サービスのダウンタイムを最小限にすることが可能になる。以下にエッジコンピューティングにおける仮想マシンおよびコンテナの移行に対する既存研究を紹介する。

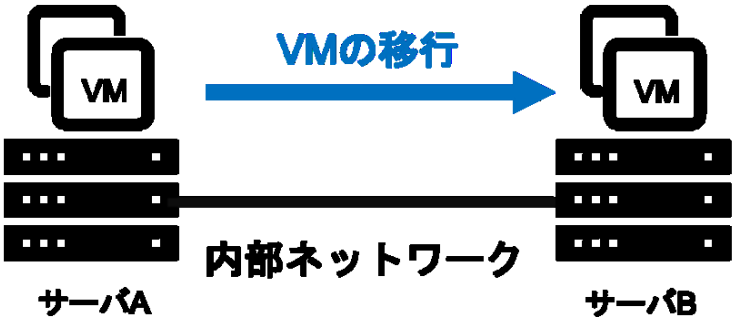

図 6　データセンタ内のマイグレーション例

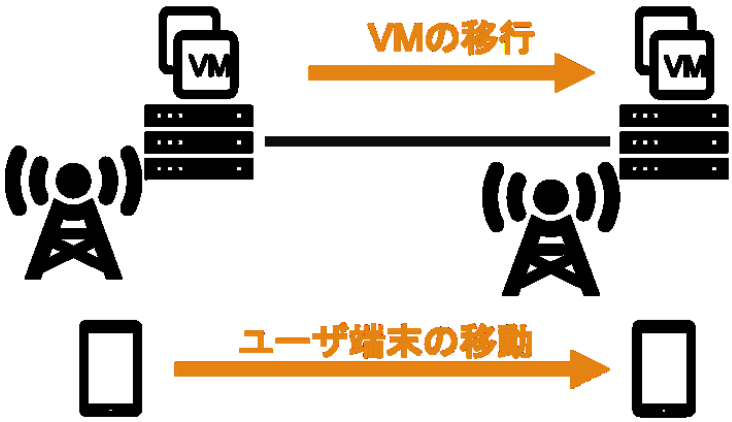

図 7 エッジコンピューティング環境のマイグレーション例

4.1 コンテナの移行に関する研究

コンテナの特徴である高速な展開やリソースの有効活用など仮想マシンではなくコンテナを用いたエッジコンピューティングの研究が盛んに行われている。

Machen らは、VM およびコンテナを利用した OS 仮想化である LXC をカプセル化した 3 層から成る、階層型の移行フレームワークを提案した [14]。1 層目はベースレイヤと呼ばれるアプリケーションが入っていないゲスト OS やカーネルなどを含むレイヤ、2 層目はアプリケーションレイヤと呼ばれるアプリケーションがインストールされたレイヤ、最上層の 3 層目にはアプリケーションの実行状態のみが保存されたインスタンスレイヤの 3 層に分離する手法を提案している。3 層に仮想環境を分離し、移行先に必要なレイヤのみを rsync を用いて同期することでアプリケーションの移行時間を削減している。LXC と KVM を用いて提案移行フレームワークを実装し、実験を行った結果コンテナの移行時間は仮想マシンと比較して最大 8 倍の高速化に成功した。

Ma らはコンテナプラットホームである Docker[15] のファイルシステムを用いたコンテナの移行を提案した [16]。コンテナの移行が行われる前に事前にコンテナイメージおよびコンテナのプロセスイメージを移行先に転送し、実際に移行が行われるタイミングでコンテナのプロセスイメージとコンテナイメージの差分のみを送信することでコンテナの移行時間を削減している。実験の結果同じエッ

ジコンピューティングのプラットフォーム上での仮想マシンのハンドオフ時間を56%〜80%削減した。

Farris らはコンテキストベースのステートレスなアプリケーションにおける移行時間を削減している [17]。各エッジサーバ内に Docker コンテナの永続的なストレージであるボリュームにコンテキスト情報を格納し、Docker アプリケーションはボリュームに格納されたコンテキスト情報からサービスを提供する。コンテキスト情報は常に変更するため、ユーザの移行先のエッジサーバ内のボリュームと、移行元のエッジサーバ内のボリュームの定期的な同期を行うことでアプリケーションの移行で発生する Docker コンテナのダウンタイムの短縮を行っている。しかし、ネットワークの帯域幅を考慮していないため、一つのサービスに複数のボリュームが存在する場合、通信量がボリュームの数に比例して増加する可能性がある。

上記の研究はコンテナの移行により発生するダウンタイムを通信量を削減することを目的としているが、エッジコンピューティング環境におけるユーザの移動やエッジサーバの高負荷時などにおけるコンテナの移行タイミングや移行先エッジサーバの選択手法は考慮されていない。

4.2 仮想マシンの移行に関する研究

Adlen らは一次元ランダムウォークにおけるデバイスと移行元のデータセンタ間の距離からサービスの移行を決定するポリシーをマルコフ決定過程を用いて設計した [18]。デバイスがあるデータセンタの通信範囲から別のデータセンタの通信範囲に移動した後に移行を行うかをポリシーによって決定を行う。シミュレーションの結果移行を行わないポリシーと比較して、常に最大の利得を得ることが可能であることを示した。

Wang らは最適な仮想マシンの移行の決定によって全体の総コストの最小化を目的とする移行ポリシーを提案した [19]。Adlen の提案に加えて閾値を追加し、デバイスの状態がある閾値の組み合わせの範囲内に存在する場合常に移行を行うポリシーを設計した。また仮想マシンの移行に最適な閾値を求めるために計算量を抑える反復アルゴリズムを設計し、シミュレーションを行った。シミュレーショ

ンの結果「常に移行を行う」または「決して移行を行わない」ポリシーと比較しコストが最小になることを示した。

また、Wang らは上記のモデルを二次元移動モデルに拡張し、送信コストと移行コストを合わせた総コストの最小化を目的とするユーザとサービスの距離を状態としたマルコフ決定過程として定式化した [20]。また距離に基づくマルコフ決定過程の構造的性質を用いた最適ポリシーの計算アルゴリズムを使用することで計算量を $O(N^3)$ から $O(N^2)$ へと削減した。シミュレーションの結果「常に移行を行う」、「決して移行を行わない」および近視眼的なポリシーと比較し、合計コストを 9%から 54%の範囲で削減できたことを示した。

上記の研究は仮想マシンの移行コストとユーザとの通信コストの総和を最小化することを目的としている。しかしエッジサーバ自身の負荷は考慮していないため、仮想マシンの移行により高負荷なエッジサーバの発生を抑制することが困難である。

4.3 関連研究のまとめと問題点

エッジコンピューティングにおける仮想マシンおよびコンテナのマイグレーションの問題点をまとめる。コンテナの移行における既存研究では仮想マシンのマイグレーションを高速に行うために移行データの削減を行なっている。しかし、データを削減するための処理によるエッジサーバやネットワークへの負荷などは考慮していない。またエッジコンピューティング環境における移行の最適化のためにマルコフ決定過程を用いて移行コストの最適化を行なっているが、マルコフ決定過程が保持する状態が多くなるにつれて状態空間が増加するため、膨大なデバイスが接続されるエッジコンピューティング環境では最適解を出すために必要な計算リソースはエッジサーバへの負荷を考慮すると少ない方が好ましい。また状態空間をユーザとサービスの格納されているエッジサーバとの距離とし、コスト関数が距離にのみ依存すると仮定しているが、実際にはユーザのサービスの要求により発生するエッジサーバへの負荷を考慮する必要がある。また関連研究ではユーザのエッジサーバへのアクセスの局所的な集中によりエッジサーバが極端に過負荷になるようなケースを考慮していない。

本研究ではユーザに対する移行による遅延を考慮したエッジサーバの負荷分散を目的とする。エッジサーバ自身の負荷を移行コストに導入することで、コンテナの移行によるエッジサーバへの影響を最小限に抑えつつ、ユーザへの通信遅延が極力発生しないようにする。

5. 提案手法

本章ではユーザの移動を考慮したコンテナの移行手法について提案する。提案手法は以下のデータを取得可能であると仮定する。前提としてユーザは常に任意のアクセスポイント (Access Point, AP) に入っているものとする。

- デバイスの位置情報
- エッジサーバの負荷情報
- 仮想マシンおよびコンテナの特性情報

そのため、ユーザが AP と通信可能なエリアにいない場合は本研究では考慮しない。前章の関連研究ではエッジサーバの負荷に対する考慮はされていないため、使用可能なリソースが少ないエッジサーバが最適な移行先となる恐れがある。そのため、移行先を決定する際に移行先候補となるエッジサーバの CPU リソースを超えず、かつ移行コストが最小になるようなエッジサーバを移行先とする。また、移行先の決定のタイミングで移行が完了していない異なるコンテナが存在する場合、要求リソースを事前に移行候補のエッジサーバに加算しておくことで、移行した後にエッジサーバのリソースが枯渇することを防ぐ。

5.1 移行コストの定義

移行によって発生するサービスへの影響を最小限にするために、以下の 2 種類が移行コストとして考えられる。

- コンテナの移行に必要な時間
- モバイルデバイスと移行先エッジサーバの通信遅延

コンテナの移行に必要な時間とエッジサーバとの通信遅延を比較した場合、移行によるダウンタイムは通信遅延に比べて極めて小さいため、コンテナの移行に必要な時間を最小にすることが重要だと考える。そのため、本提案手法では移行時間が最も最小になるようにコンテナを選択する。コンテナを移行することによっ

て発生する移行コストを以下の式によって求める。エッジコンピューティング環境では少ない遅延が求められるが、コンテナや仮想マシンの移行はデータ量が多く、複数の移行が発生した

$$C_{\text{mig}} = S_{\text{conainer}}/B_{\text{src\_dest}}$$

- C_{mig}：コンテナを移行する際に発生する移行コスト
- $S_{\text{container}}$：コンテナのサイズ (メモリ使用量)
- $B_{\text{src\_dest}}$：移行元エッジサーバ、移行先エッジサーバ間の帯域幅

移行コストおよびエッジサーバの負荷を考慮したエッジサーバ選択アルゴリズムを Algorithm1 に示す。

Algorithm 1 移行コストとエッジサーバの負荷を考慮したエッジサーバ選択手法

Input: 全エッジサーバリスト E, 全コンテナリスト C, C

Output: 移行先エッジサーバ $e \in E$

移行先エッジサーバ $= null$
for each $e_i \in E$ **do**
  INITIALIZE エッジサーバ e_i の CPU 使用率 $= 0$
  for each $c_j \in C$ **do**
    if コンテナ c_j がエッジサーバ e_i に存在、または移行途中のコンテナ c の移行先エッジサーバが e_i **then**
      エッジサーバ e_i の CPU 使用率を更新
    end if
  end for
end for
移行候補のエッジサーバのリストを移行コストの昇順にソート
for each $e_i \in E$ **do**
  if $e_i \leq th_{over}$ **then**
    移行先エッジサーバ $= e_i$
  end if
end for

6. 実験および考察

6.1 実験環境

本論文では、提案手法をシミュレータに実装を行い、動作させることにより、評価実験を行う。シミュレータとしてはMyiFogSim[21] を用いた。MyiFogSim はフォグコンピューティング環境のシミュレータである iFogSim[22] を拡張して作成され、特に iFogSim では実装されていなかったユーザ移動モデルや仮想マシン・コンテナの移行機能が実装されている。

6.2 事前実験

本研究においてコンテナに負荷を与えた時の移行への影響を調査するため、コンテナプラットフォームである Docker とプロセスの実行状態をファイルとして出力する CRIU[7]を用いて調査を行った。実験に用いた環境とソフトウェアを表 2 に示す。コンテナプラットホームとして Docker を、負荷ツールとして stress-ng を用いた。また仮想マシン上で Docker のコンテナを一台起動し、その内部で負荷ツールを実行させながら CRIU を用いてプロセスイメージの出力を行った。CRIU とは Checkpoint/Restore In Userspace の略称であり、Linux 上で動作しているアプリケーション (プロセス) の状態をファイルに保存 (Checkpoint) し、保存されたファイル情報からプロセスを保存時の状態からの再開が可能となるツールである。このツールを用いて実際にコンテナのプロセスイメージのチェックポイントを作成する。

[7] https://criu.org/Main_Page

表 2 負荷実験に用いた環境

使用ソフトウェア	バージョン
CPU	Intel Core i7-380 2.8GHz
Docker	18.09.1
仮想マシンおよびコンテナイメージ OS	Ubuntu 16.04
CRIU	2.6
stress-ng	0.05.23

作成した Docker コンテナに対して以下の種類の負荷を与えた。各負荷ごとに 10 回計測したのち、平均をとった。

- CPU 使用率の変化
  CPU 使用率に負荷を与える割合を 10%から 100%まで 10%ずつ増加
- メモリ使用量の変化
  メモリ使用量を 128MB, 256MB, 512MB, 1024MB と 4 段階に変化

6.2.1 実験結果

図 8 および図 9 よりコンテナの CPU 負荷に依存したプロセスダンプの時間およびプロセスファイルサイズの大きな変化は見られなかった。またコンテナのメモリ使用量が増加するとプロセスファイルサイズとプロセスダンプに必要な時間が線形に増加していくことが確認できた。図 10 より移行に必要なデータサイズはコンテナのメモリ使用量にほぼ比例することが確認できた。

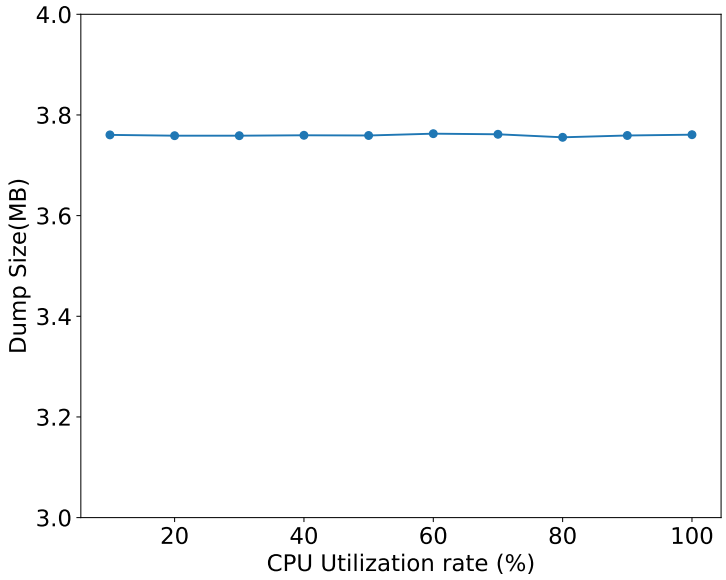

図 8　コンテナの CPU 使用率とプロセスダンプサイズ

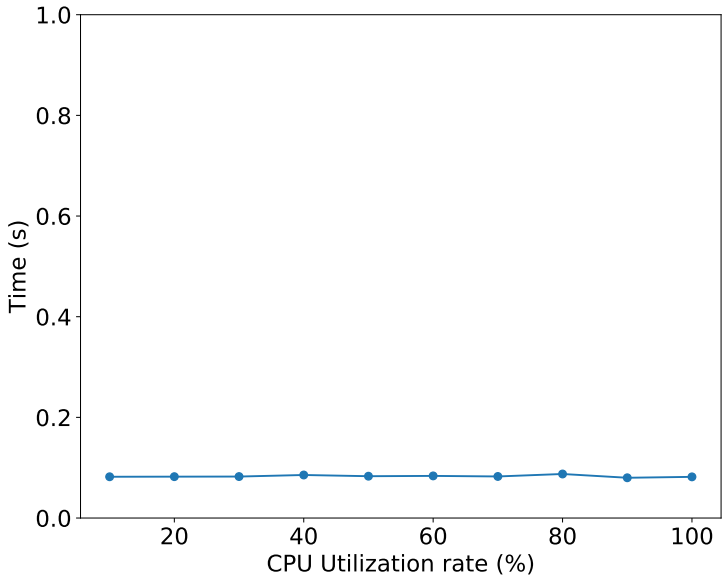

図 9　コンテナの CPU 使用率とプロセスダンプ完了時間

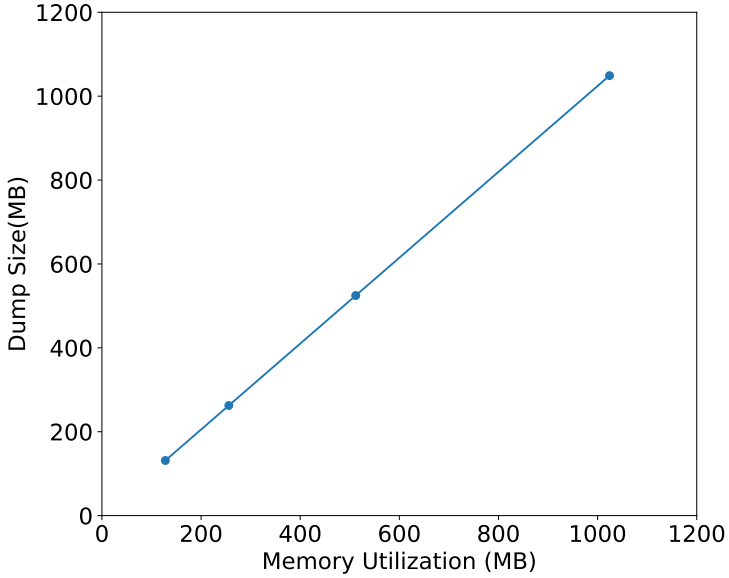

図 10　コンテナのメモリ使用量とプロセスダンプサイズ

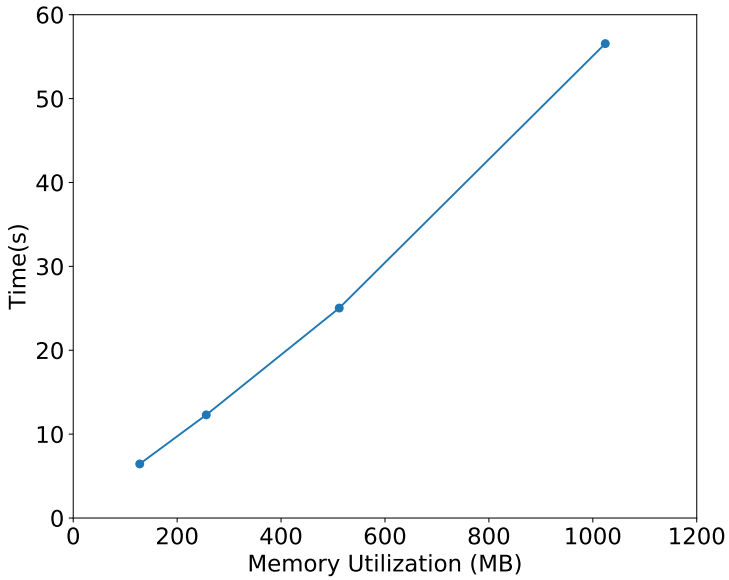

図 11　コンテナのメモリ使用量に対するとプロセスダンプ完了時間

また、事前実験より一つのコンテナの移行に必要な総データサイズを事前実験から以下のように定義する。

- S_{total}：コンテナの総データサイズ
- $S_{\text{container}}$：コンテナのイメージサイズ
- $S_{\text{process\_img}}$：コンテナのプロセスデータ (メモリ使用量) サイズ

$$S_{\text{total}} = S_{\text{container}} + S_{\text{process\_img}}$$

本研究では Alpine Linx や Ubuntu などの OS イメージ上でアプリケーションを実行すると仮定する。一般的な Alpine Linux や Ubuntu の OS イメージはそれぞれ 2MB、188MB となっており、コンテナの総データサイズはその上で実行するアプリケーションイメージ (Nginx で 109MB) を考慮し、50MB〜500MB の範囲で存在すると仮定する。

6.3 ユーザの移動モデル

ユーザの移動は図 12 に表されるようなある特定のエリアにユーザが集合するような場合を考える。位置情報ゲームのイベントより大量のユーザがある特定の場所に移動すると、ユーザがアクセスするエッジサーバやアクセスポイントだけでなく近傍のエッジサーバにも負荷が集中する恐れがある。研究の目的であるユーザがある特定のエリアに集まることによってアクセスポイントに接続されている cloudlet と呼ばれるエッジサーバへの負荷を分散できているかを確認する。ユーザの移動シナリオを以下のように設定する。ユーザの初期座標はシミュレータ範囲内に一様に分布しているものする。

- 移動シナリオ：あるランダムに指定された位置に向かってユーザが一斉に移動し続け、目的地に到達してもユーザは方向を変えず移動し続けるシナリオ

6.4 Myifogsim の設定パラメータ

Myifogsim には SmartThig と呼ばれるモバイル端末や ServerCloudlet と呼ばれるエッジサーバなどがオブジェクトとして準備されている。このオブジェクトのパラメータを任意で変更することで様々なエッジコンピューティング環境のシミュレーションを行うことが可能となる。ここでは主要なパラメータの設定項目について説明する。

6.5 実験環境

実験パラメータを表 3 に、アクセスポイント、エッジサーバ、モバイル端末およびコンテナの特性を表 5 から表 7 にまとめる。アクセスポイントの通信可能範囲および通信可能範囲は [19] を参考に設定した。シミュレータ場でのエッジサーバおよびアクセスポイントの配置を図 13 に示す。

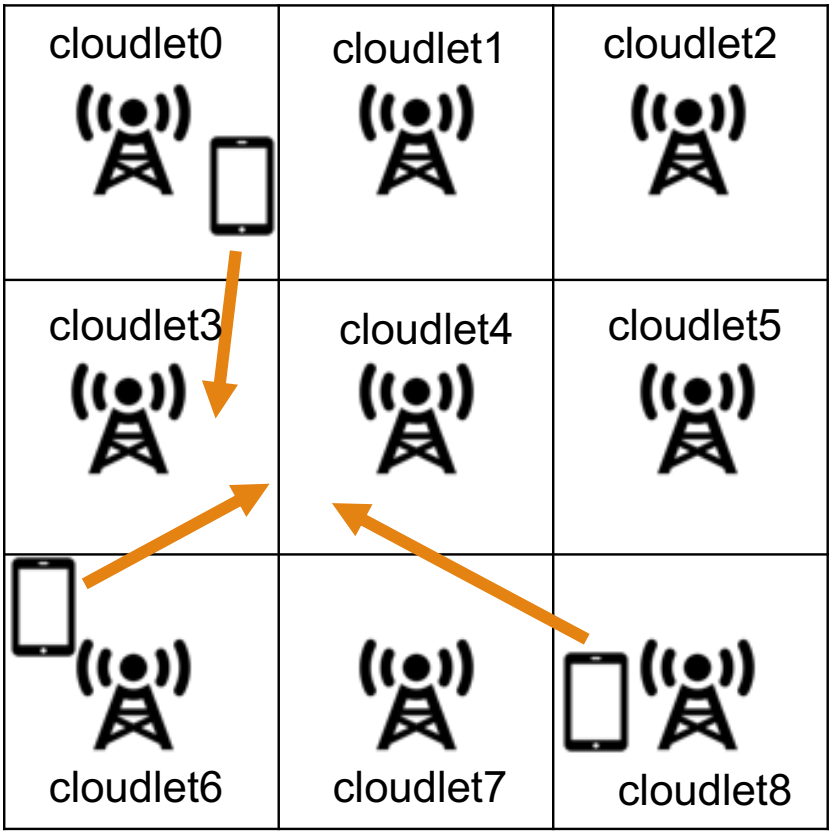

図 12　ユーザの移動モデル概要

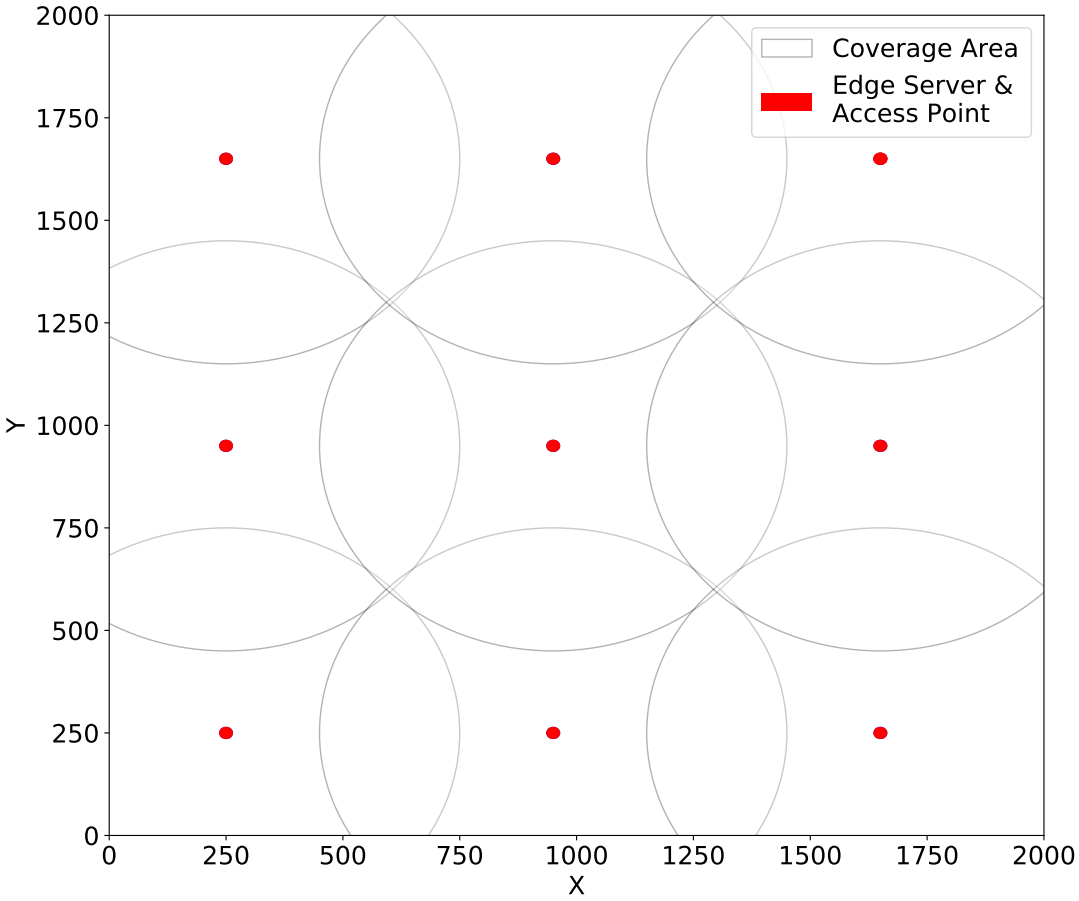

図 13　エッジサーバとアクセスポイントの配置図

表 3 実験パラメータ

シミュレーション時間	3600 秒
シミュレータ範囲	9 平方 km
モバイルデバイス数	10,20,30,40,50,60
アクセスポイント	700m 間隔で配置
アクセスポイントの通信可能範囲	半径 500m
アクセスポイント数	9
エッジサーバ数	9
モバイルデバイス移動速度	1m/s, 2m/s, 3m/s の中からランダム
移行可能と判断するエッジサーバの CPU 負荷の閾値	90%

表 4 アクセスポイントのパラメータ

	100GB
RAM	25GB
CPU	2800MIPS
モバイル端末との遅延	4ms
ネットワークトポロジ	アクセスポイントに最も近い 1 台のエッジサーバのみ接続

表 5 エッジサーバのパラメータ

ストレージ	100GB
RAM	25GB
CPU	2800MIPS
エッジサーバ間の帯域幅	1Gbps〜10Gps の範囲でランダム
エッジサーバ間の遅延	5ms〜10ms の中でランダム
エッジサーバ間のネットワークトポロジ	フルメッシュ

表 6 モバイル端末の設定

ストレージサイズ	40GB
RAM サイズ	1GB
CPU	1000MIPS
帯域幅 (上り)	23Mbps
帯域幅 (下り)	123Mbps

6.6 提案手法と他の移行手法との比較

提案手法の有効性を検証するため、以下の手法と比較した。

- 移行を行わない手法 (no-migration)
- エッジサーバの CPU 利用率が低いエッジサーバを選択する手法 (minimum-CPU)
- ユーザとの通信遅延が最小になるようにエッジサーバを選択する手法 (lowest-latency)

表 7 コンテナのパラメータ

CPU	80〜300MIPS の範囲でランダム
コンテナのサイズ (使用メモリ量)	50〜500MB の範囲でランダム

移行を行わない手法はユーザの通信遅延が提案手法と比較してどれほど低く抑えられているかの比較に利用する。エッジサーバの CPU 利用率が低いエッジサーバを選択する手法は提案手法と比較して移行によるエッジサーバの負荷分散が移行時間および遅延に影響を与えるかの比較に利用する。ユーザとの通信遅延が最小になるようにエッジサーバを選択する手法は提案手法と比較して移行時間が削減されているかの比較に利用する。

またエッジコンピューティング環境を再現するため、以下の 2 種類のネットワーク環境を構築し実験を行った。

- 帯域幅が 1Gbps から 10Gbps の範囲で均一に分布 (一様分布)

- 上記の分布に 500Mbps から 1Gbps までの帯域幅のネットワークを局所的に作成 (局所的)

帯域幅が一様に分布しているのはサービスを提供するエッジサーバが小規模なデータセンタ内で運用されているものとし、通信環境が安定しているものと仮定する。帯域幅が局所的に狭くなっているものは小規模なデータセンタではなく屋外やそれに近い環境で配置しているエッジサーバが存在し、通信環境が無線 LAN のように広帯域でないものと仮定する。

計測には以下の値を取得し、比較を行う。

- 通信遅延 (平均、最大および最小)

- 移行時間 (平均、最大および最小)

通信遅延および移行時間の評価により移行によるサービスの品質を評価することができる。通信遅延が低いほどユーザはサービスを高速に受け取ることができるため、サービスの品質が向上すると考えられる。また移行時間が短いほどサービスのダウンタイムを削減することが可能となりサービスの可用性が向上すると考えられる。

6.7 通信遅延の評価

移行を行った際の平均遅延および最大、最小遅延について評価する。平均通信遅延の結果を図 14 および図 15 に、最大通信遅延の結果を図 16 および図 19 に示す。移行を行わない場合と比較して提案手法、比較手法は常に移行しない場合より遅延が低いことが確認できる。またデバイス数が 60 の時に大きく遅延が発生しているのは特定のアクセスポイントにユーザが集中してアクセスしたためアクセスポイントの性能が低下したためだと思われる。

最小通信遅延の結果を図 18 および図 19 に示す。一様分布および局所的な環境共に大きな変化は見られなかった。これは各ユーザに対して適切なエッジサーバと通信を行っていたためだと予想される。

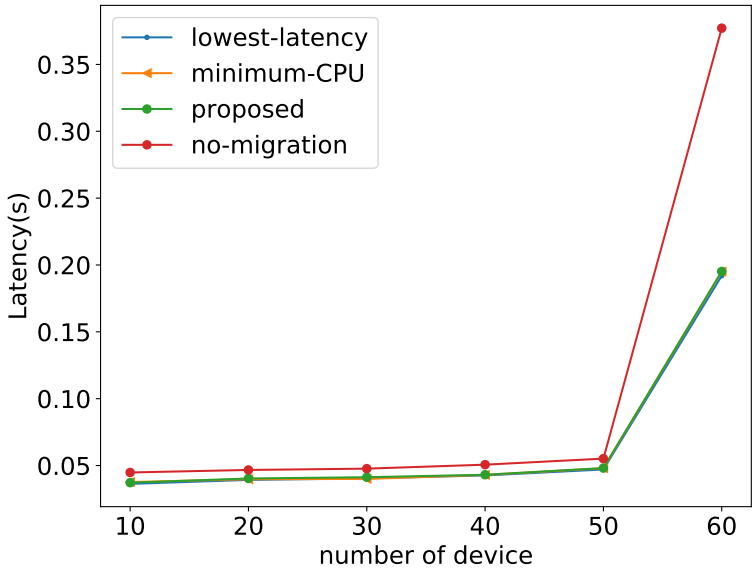

図 14 平均通信遅延 (一様分布)

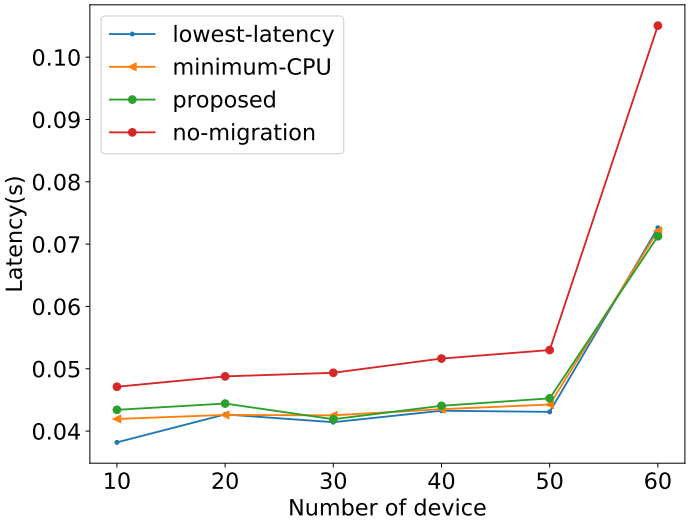

図 15　平均通信遅延 (局所的)

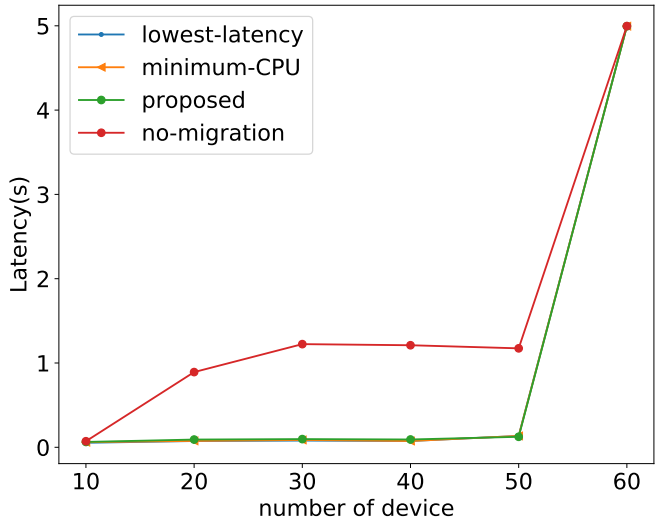

図 16　最大通信遅延 (一様分布)

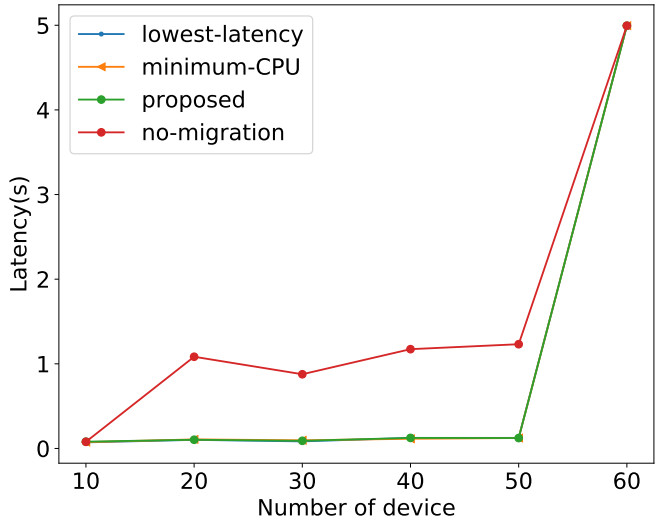

図 17　最大通信遅延 (局所的)

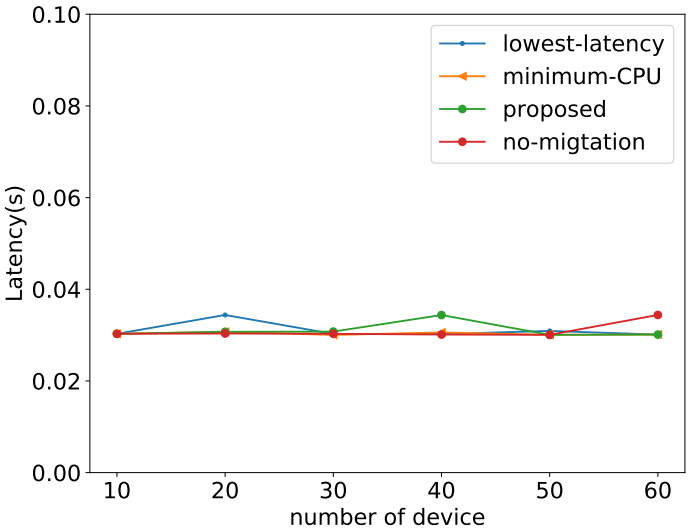

図 18　最小通信遅延 (一様分布)

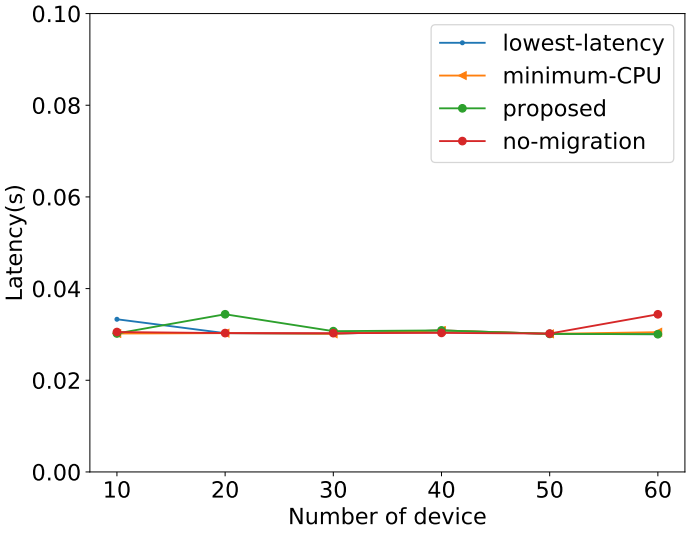

図 19　最小通信遅延 (局所的)

6.8　移行時間の評価

移行時間について評価する。各環境における平均移行時間を図 20 および図 21 に示す。一様分布な環境では提案手法は他手法と比較するとデバイス数が 60 の場合以外はほぼ同等かそれより短い時間で移行が可能と言うことが分かった。局所的な環境では提案手法がデバイス数が 30 および 60 の場合に移行時間が他の手法よりも増加していることが分かった。理由として原因としてエッジサーバを選択するタイミングで最適なエッジサーバを選択できなかったためと考えられる。

最大移行時間の結果を図 22 および図 23 に示す。一様分布な環境では提案手法は他の手法と比較して同等か移行時間の削減が見られた。局所的な環境においても各手法ともほぼ同様の結果となった。理由として帯域幅が狭いエッジサーバにコンテナが存在すると、移行するにあたり移行元の帯域幅がボトルネックになるため移行を行うには避けることができないと考えられる。

最小移行時間の結果を図 24 および図 25 に示す。一様に分布した帯域幅の環境ではほぼ同等または他手法と比較して移行時間を短縮することができた。これは

移行コストとして最も帯域幅の広いネットワークで接続されているエッジサーバを優先的に選択したためだと思われる。局所的に帯域幅が狭い環境では他手法と比較して大きく変化は見られなかった。しかし局所的な環境では他手法と比較して同等または移行時間の短縮が可能となった。理由として帯域幅の大きいエッジサーバを可能な限り選択することで、移行時間の短縮に繋がったのだと考えられる。

最も遅延が最小になるように選択する手法 (lowest-latency) では移行後のエッジサーバの CPU 利用率が 100%を超えなければ移行先候補として決定されるため、常にエッジサーバが高負荷になり続ける

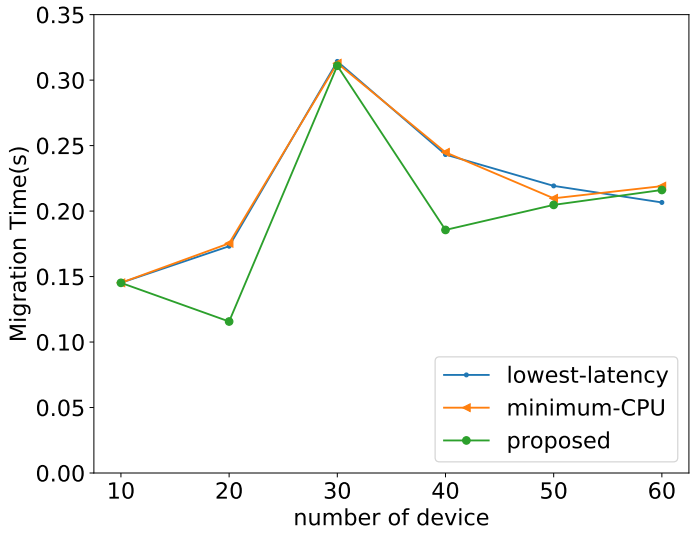

図 20　平均移行時間 (一様分布)

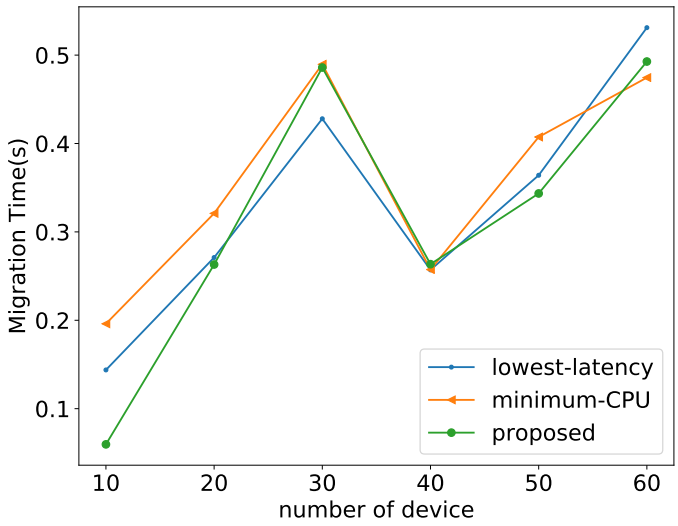

図 21　平均移行時間 (局所的)

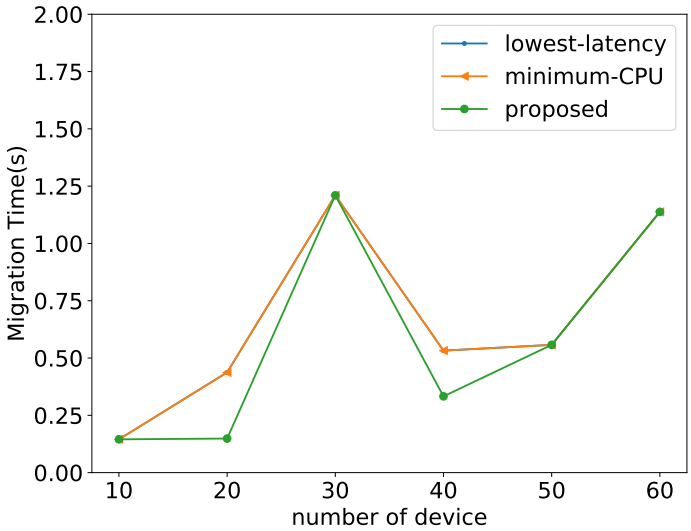

図 22　最大移行時間 (一様分布)

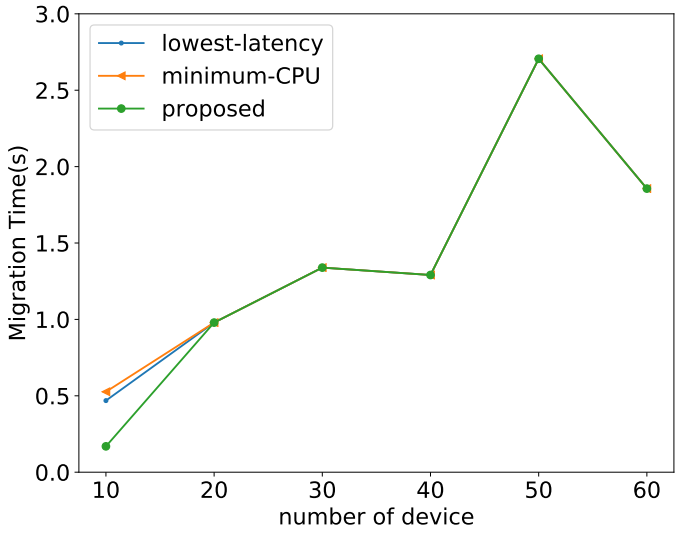

図 23　最大移行時間 (局所的)

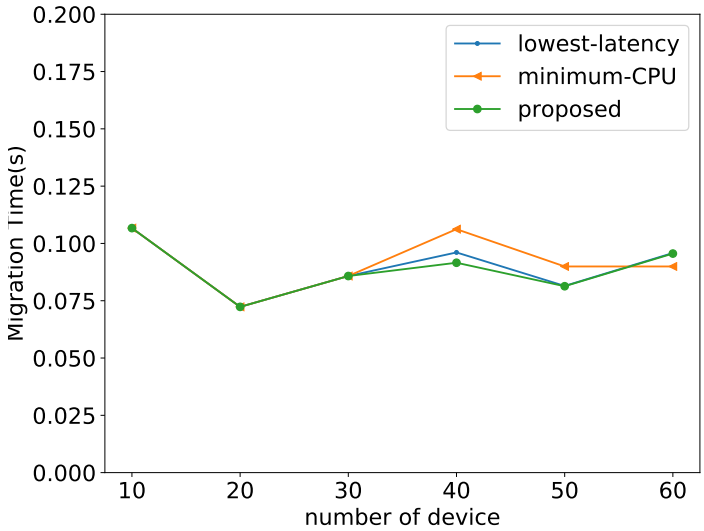

図 24　最小移行時間 (一様分布)

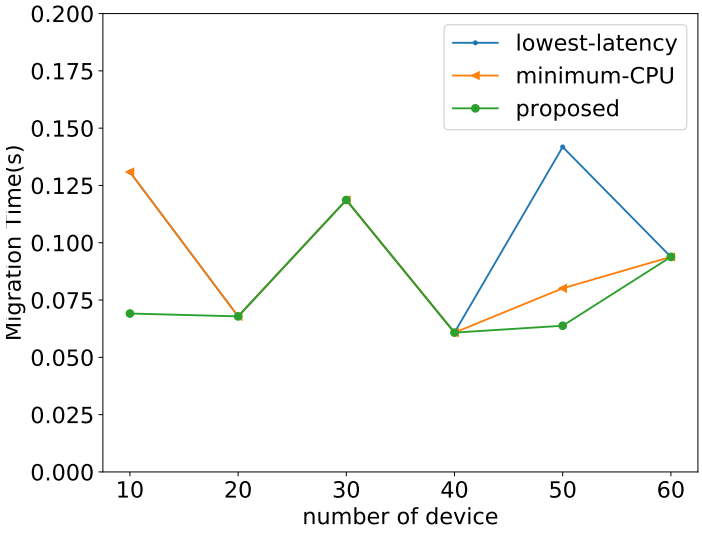

図 25　最小移行時間 (局所的)

6.9　考察

実験結果から、エッジサーバの選択に高負荷にならないように閾値を与えたため、エッジサーバが極端に高負荷にならない代わりに平均移行時間が他手法と比較して増加するケースが見られた。提案手法は移行後のエッジサーバが高負荷にならないように閾値を設けて負荷の分散を行ったため、他の手法と比較して平均移行時間が増加する場合があった。また移行アルゴリズムに移行が終了していないコンテナの要求リソースを移行先エッジサーバの CPU 利用率に加算したことで、ある時点でエッジサーバが高負荷でなくともアルゴリズムが高負荷であると判断したため、最適な移行を行うことができなかったと予想される。

6.10 今後の課題

今後の課題としてシミュレーションの規模を拡大することが挙げられる。本論文では最大60台までのデバイスしかシミュレートできておらず、またアプリケーションの特性を考慮したデバイスの移動モデルやデバイスとエッジサーバ間のトラフィックモデルをシミュレータに実装する必要がある。またデバイスから受け取ったタスクの処理遅延やコンテナの移行による消費電力の削減などを考慮をする必要がある。現状はツールによる移行のみが実験用として提供されているため、仮想マシンのように自動でコンテナを最適なエッジサーバへ自動で移行するようなプラットフォームの作成が重要である。また、1コンテナ1ユーザのモデルではなく複数人のアクセスに対してどのように移行のポリシーを設計すればいいのかの課題が残る。実際のエッジコンピューティング環境は様々な機器特性を持ったエッジサーバが存在することがある。そのため、局所的なデータの処理遅延や伝送遅延などに対処可能なアーキテクチャや移行アルゴリズムを開発する必要がある。またフォグコンピューティングのようなルータやスイッチなどのネットワーク機器にフォグコンピューティング用アプリケーションを導入した際のネットワークへの負荷や処理遅延なども今後の検討課題となる。

7. 結論

膨大な数のモバイルデバイスやスマートセンサがクラウドにデータを送信し続けている。しかし従来の中央集約されたクラウドコンピューティングではデータの処理に対する遅延などに対処することが困難になってきている。そのため、ネットワークのエッジでデータを処理するエッジコンピューティングに注目し、コンテナの移行時間を可能な限り最小限に抑え、かつサーバへの負荷を可能な限り分散させる手法を提案した。シミュレータに提案手法を実装し、その結果他手法と比較し移行時間を同等または削減することが可能となった。今後の課題としてエッジサーバに対するより現実的なネットワークモデルでの実験、クラウドや他のユーザのデバイスなどと連携した負荷分散手法およびタスク処理高速化の検討が必要と考えられる。

謝辞

主指導教員であり、適切な研究指導をしていただくなど様々な側面から研究のサポートをしてくださいました本学情報基盤システム学研究室の藤川和利教授に心から感謝致します。副指導教員であり、研究の方向性について的確な助言をくださいました本学大規模システム管理研究室の笠原 正治教授に心から感謝致します。副指導教員であり，研究指導、論文執筆指導、そして多くの研究発表の機会を与えてくださいました本学情報基盤システム学研究室の新井イスマイル准教授に心から感謝致します。研究方針や論文執筆に関して数々のご助言をくださいました本学情報基盤システム学研究室の垣内正年助教に心から感謝致します。研究活動や論文執筆に関してご指導をいただきました本学情報基盤システム学研究室の油谷曉助教に心から感謝致します。研究に関して的確なご指摘、ご指導を戴きました、本学ソフトウェア設計学研究室の市川昊平准教授に心から感謝致します。さまざまな面から研究活動を支援してくださいました本学総合情報基盤センターの辻元理恵女史、中野彩子女史に心から感謝致します。さまざまな面から研究活動を支え、共に技術的にも切磋琢磨しあった本学情報基盤システム学研究室の同期をはじめとした友人の皆様に心より感謝いたします。最後に私の意思を尊重し、経済面や生活面において多大な支援を頂き最後まで見守ってくださいました家族に心から感謝致します。

参考文献

[1] シスコシステムズ合同会社. Cisco Visual Networking Index：予測と方法論、2016 〜 2021 年ホワイト ペーパー - Cisco. https://www.cisco.com/c/ja_jp/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html, 2018. Accessed 2018-11-11.

[2] Mahadev Satyanarayanan. The Emergence of Edge Computing. *Computer*, Vol. 50, No. 1, pp. 30–39, Jan 2017.

[3] Tuyen X. Tran, Abolfazl Hajisami, Parul Pandey, and Dario Pompili. Collaborative Mobile Edge Computing in 5G Networks: New Paradigms, Scenarios, and Challenges. *IEEE Communications Magazine*, Vol. 55, No. 4, pp. 54–61, April 2017.

[4] Wei Bao, Dong Yuan, Zhengjie Yang, Shen Wang, Wei Li, Bing Bing Zhou, and Albert Y. Zomaya. Follow Me Fog: Toward Seamless Handover Timing Schemes in a Fog Computing Environment. *IEEE Communications Magazine*, Vol. 55, No. 11, pp. 72–78, Nov 2017.

[5] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog Computing and Its Role in the Internet of Things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pp. 13–16, 2012.

[6] AWS IoT Greengrass - Amazon Web Services. https://aws.amazon.com/greengrass/, 2019. Accessed 2019-11-11.

[7] IoT Edge — Microsoft Azure. https://azure.microsoft.com/en-us/services/iot-edge/, 2019. Accessed 2019-01-25.

[8] Cloud IoT Edge - Extending Google Cloud's AI & ML — IoT Edge — Google

Cloud. https://cloud.google.com/iot-edge/, 2019. Accessed 2019-01-25.

[9] Koustabh Dolui and Soumya Kanti Datta. Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing. In *2017 Global Internet of Things Summit (GIoTS)*, pp. 1–6, June 2017.

[10] Chao Li, Yushu Xue, Jing Wang, Weigong Zhang, and Tao Li. Edge-Oriented Computing Paradigms: A Survey on Architecture Design and System Management. *ACM Comput. Surv.*, Vol. 51, No. 2, pp. 39:1–39:34, April 2018.

[11] Luis M. Vaquero and Luis Rodero-Merino. Finding Your Way in the Fog: Towards a Comprehensive Definition of Fog Computing. *SIGCOMM Comput. Commun. Rev.*, Vol. 44, No. 5, pp. 27–32, October 2014.

[12] 総務省. 平成 27 年版 情報通信白書 (PDF 版). http://www.soumu.go.jp/johotsusintokei/whitepaper/ja/h27/pdf/index.html, 2018. Accessed 2018-11-11.

[13] Claus Pahl, Sven Helmer, Lorenzo Miori, Julian Sanin, and Brian Lee. A Container-Based Edge Cloud PaaS Architecture Based on Raspberry Pi Clusters. In *2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, pp. 117–124, Aug 2016.

[14] Andrew Machen, Shiqiang Wang, Kin K. Leung, Bong Jun Ko, and Theodoros Salonidis. Live Service Migration in Mobile Edge Clouds. *IEEE Wireless Communications*, Vol. 25, No. 1, pp. 140–147, February 2018.

[15] Enterprise Container Platform — Docker. https://www.docker.com/, 2018. Accessed 2018-11-11.

[16] Lele Ma, Shanhe Yi, and Qun Li. Efficient service handoff across edge servers via docker container migration. In *Proceedings of the Second ACM/IEEE*

Symposium on Edge Computing, SEC '17, pp. 11:1–11:13, New York, NY, USA, 2017.

[17] Ivan Farris, Tarik Taleb, Hannu Flinck, and Antonio Iera. Providing ultra-short latency to user-centric 5G applications at the mobile network edge. *Transactions on Emerging Telecommunications Technologies*, Vol. 29, No. 4, p. e3169, 2018.

[18] Adlen Ksentini, Tarik Taleb, and Min Chen. A Markov Decision Process-based Service Migration Procedure for Follow Me Cloud. In *2014 IEEE International Conference on Communications (ICC)*, pp. 1350–1354, 2014.

[19] Shiqiang Wang, Rahul Urgaonkar, Ting He, Murtaza Zafer, Kevin Chan, and Kin K. Leung. Mobility-Induced Service Migration in Mobile Micro-clouds. In *2014 IEEE Military Communications Conference*, pp. 835–840, Oct 2014.

[20] Shiqiang Wang, Rahul Urgaonkar, Murtaza Zafer, Ting He, Kevin Chan, and Kin K. Leung. Dynamic Service Migration in Mobile Edge-Clouds. In *2015 IFIP Networking Conference (IFIP Networking)*, pp. 1–9, May 2015.

[21] Márcio Moraes Lopes, Wilson A. Higashino, Miriam A.M. Capretz, and Luiz Fernando Bittencourt. MyiFogSim: A Simulator for Virtual Machine Migration in Fog Computing. In *Companion Proceedings of the10th International Conference on Utility and Cloud Computing*, UCC '17 Companion, pp. 47–52, 2017.

[22] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K. Ghosh, and Rajkumar Buyya. iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments. *Software: Practice and Experience*, Vol. 47, No. 9, pp. 1275–1296, 2017.

発表リスト

1. 仲野 弘一, 垣内 正年, 新井 イスマイル, 藤川 和利. エッジコンピューティング環境におけるユーザの移動性を考慮したコンテナの同期先の選択手法の提案, 信学技報,vol.118,no.360,IA2018-53,pp.85-88,Dec,2018.