Vol.2020-OS-150 No.4
2020/7/30

ハンドオーバー可能なコンテナ仮想化ベースの Cloudlet

北出 紘章[1]　阿部 洋丈[2]　加藤 和彦[2]

概要：エッジコンピューティングは端末に近いネットワークエッジ上でデータ処理を行うコンピューティングパラダイムであり，IoT とともに発展を続けている．これに関する代表的な先行研究として Cloudlet が挙げられ，これは VM 仮想化を利用したエッジサーバである．ところが VM 仮想化はコンテナ仮想化に比べてオーバーヘッドが大きいため，クラウドデータセンターよりも計算資源が限られているエッジサーバではコンテナ仮想化を用いるほうが効率が良い．そこで本稿ではコンテナ仮想化を利用した Cloudlet を提案するが，ここで問題となるのは VM の場合にライブマイグレーションで対応していた端末の移動に伴うエッジサーバの切替えをコンテナでどのように実現するかという問題である．提案アーキテクチャでは移動体通信における Mobile IP や Session and Service Continuity を参考としたフォワーディングを利用する手法をとる．さらに，コンテナ実行環境のデファクトスタンダードとして知られている Kubernetes を活用した Cloudlet の実装についても説明する．

Handover Feasible Container-based Cloudlet

1. はじめに

IoT (Internet of Things) デバイスやモバイル端末はますます普及している．そのデバイスで動くアプリケーションも高度になっており，機械学習，コグニティブコンピューティング，リアルタイム動画処理など，負荷の高い処理をこなすことも要求されるようになっている．そのような演算を小型の端末上で実行するには性能が不足していることやバッテリー消費が激しいことがあり，実際にはクラウドデータセンターに備えられている高性能なサーバに処理をオフロードすることが多い．

ところが，ユーザの端末とクラウドデータセンターの間の距離は大きく，オフロードに必要な通信のために大きなネットワーク遅延が発生する．これは IoT・モバイルアプリケーションの応答速度を低下させユーザ体験を損ねる原因となる．また IoT デバイスの増加によってネットワークトラフィックが増大すると，近い将来に端末とデータセンター間のコアネットワークのキャパシティが不足することが懸念されている．そこで，端末に近い位置に演算能力を設置するエッジコンピューティングが注目を集めている．オフロードが必要な端末に対し 1 ホップ以内で到達できるネットワークエッジにサーバを設置することで，ネットワーク遅延を大幅に削減することが可能になる．このようなサーバをエッジサーバと呼ぶ．

エッジコンピューティングを利用する端末は移動する場合があり，例えばスマートフォン，タブレット，ロボット，コネクテッドカーなどが挙げられる．エッジサーバにバックエンドサービスを展開するときに仮想環境をプロビジョニングする必要があるが，これを端末の移動によってエッジサーバが切り替わるたびに行うとダウンタイムが断続的に発生してしまう．また，サービスがステートフルアプリケーションである場合にはエッジサーバが切り替わった際に処理を継続する仕組みが必要である．このように，エッジコンピューティングでは端末の移動を考慮する必要がある．

エッジサーバの基盤ソフトウェア技術として代表的な先行研究に Cloudlet [1] がある．これはモバイルコンピューティングにおいて端末の演算能力を補完するためのオフロード環境を低ネットワーク遅延で提供することが可能なエッジサーバである．アプリケーションがエッジサーバに特有の設計に依存することを回避するため，クラウド IaaS と同様の仮想化技術を用いることで一般的なクラウドデータセンターと同じように Cloudlet を利用することが可能になっている．オフロードされるサービスは仮想マシン

[1] 筑波大学大学院 システム情報工学研究科
[2] 筑波大学 システム情報系

(VM) により仮想化され，それを Cloudlet の仮想マシンモニタ上で稼働させる．

しかし VM はゲスト OS によるオーバーヘッドが大きく，起動時間やメモリ消費が問題となる．代わりにコンテナ仮想化を使用することでこの問題を解決できる．[2] はコンテナ仮想化を利用して Cloudlet を改良した研究のうちの一つである．このアーキテクチャでは，端末の移動によって Cloudlet の切り替えが発生した場合，プロセスのチェックポイント／リストア技術 [3] を利用したコンテナライブマイグレーションによってホストの移行を実現している．ところが，この方法には以下のような問題が挙げられる．まず 1) ヘテロジニアスな OS・ソフトウェア環境に対応していない．次に 2) コンテナを一時的にフリーズさせる必要があるため，その間は他のクライアントが通信できなくなる．さらに 3) コンテナのメモリページの転送をホスト間で行う必要があるため，ライブマイグレーション中はメモリサイズに比してダウンタイムやスループットの低下が発生する．

そこで本研究では，コンテナを仮想環境として利用する Cloudlet アーキテクチャとともに，端末の移動にともなう Cloudlet のハンドオーバー処理を安定かつ高速に行う方法を提案する．それにはライブマイグレーションを用いずに，Mobile IP [4] や Session and Service Continuity における手法に類似したフォワーディングによる方法をとる．これにより，ハンドオーバーにライブマイグレーションを使用することによって起こる問題が以下のように解決する．本手法は 1) TCP 通信が成立していれば，ヘテロジニアスな OS 環境であっても対応する．また 2) 元のコンテナを停止する必要がないため，通信中の他のクライアントに影響を及ぼさない．なお 3) 本手法ではダウンタイムは発生しないが，実際の性能については今後実験する予定である．

提案アーキテクチャの実装には一般的なコンテナ実行環境が利用可能であるが，今回は Kubernetes [5] と Docker [6] を利用する．Kubernetes はコンテナオーケストレーションシステムのデファクトスタンダードであり，クラウド開発において Docker とともに広く利用されている，これにより，オリジナルの Cloudlet と同じく，本実装においてもエッジサーバの設計に依存せずに標準的な技術で Cloudlet の機能にアクセスが可能となっている．

本稿は以下のように続く．2 章では本研究に関連する研究を紹介する．3 章では提案するアーキテクチャを説明する．4 章では実装に関する内容を述べる．5 章で提案アーキテクチャの性能を議論する．最後に 6 章で結論を述べる．

2. 関連研究

本研究に関連する先行研究や技術について以下の通りに説明する．

2.1 Cloudlet

モバイルアプリケーションが高負荷かつインタラクティブな処理を要求するようになったなか，性能の限られた端末においてもリッチなモバイルコンピューティングを可能にするために Cloudlet [1] が提案された．Cloudlet は端末から 1 ホップで到達可能なネットワークエッジに設置される小型のデータセンターである．Cloudlet はクラウドコンピューティングを拡張し，「端末 – Cloudlet – クラウドデータセンター」という 3 層のコンピュータアーキテクチャを構成するため，オフロードの目的としては低ネットワーク遅延の Cloudlet を利用し，中央集中型のデータ処理を必要とする場合はクラウドデータセンターを利用する，というような使い分けができる．

“Datacenter in a box” とみなされる Cloudlet は，小規模ながらも標準的なクラウドデータセンターと同等の機能を備え，さらにクライアントアプリケーションの Cloudlet の設計への依存ができるだけ抑えられており，一般的なクラウドコンピューティングと同様に利用できることが要求される．そこで Cloudlet の基盤ソフトウェアとして，OpenStack をベースにしたソフトウェアプラットフォーム (OpenStack++) が開発されている [7]．OpenStack はクラウド IaaS を構築・運用するためのソフトウェアであり，Amazon Web Service (AWS) や Google Cloud Platform (GCP) のようなクラウド環境を自身で用意したハードウェア上に作ることができる．

そのため Cloudlet でホストされるサービスも，標準的なクラウドプラットフォームにおける場合と同じく，仮想マシンモニタ上で稼働する VM として展開される．サービスの VM イメージは端末が持っており，オフロード時にそれを Cloudlet に転送するという形式になっている [*1]．

2.2 コンテナと Cloudlet

VM はゲスト OS を持つため，その分のオーバーヘッドが生じる．まず，アプリケーションの起動前にゲスト OS の起動が必要で，これには時間を要する．エッジコンピューティングはアプリケーションの高速化のためのものだが，その利用準備に長時間かかっては意味がない．また，複数の VM がホストされていると，その分に比例してホストのメモリを使用する．エッジサーバはクラウドデータセンターよりもリソースが限られているため，効率の面で問題となる．

コンテナ仮想化では，ホストから CPU，メモリ，名前空間，ファイルシステム，ネットワーク等のリソースを分離する一方で，ホスト OS のカーネルはコンテナ同士で共有する．コンテナは VM と比べてゲスト OS が不要のため，より高速に起動でき，メモリやイメージサイズも小さいの

[*1] 実際には Dynamic VM synthesis という手法により高速化される．

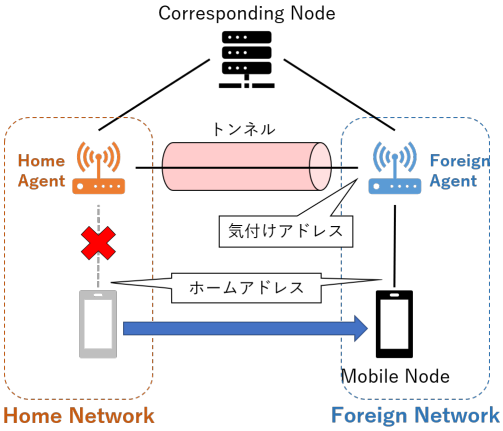

図 1　Mobile IP

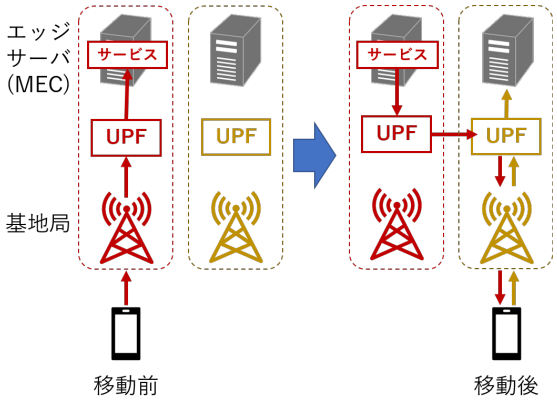

図 2　Session and Service Continuity mode 3

でアプリケーションの集約効率も上がる．そのため，コンテナ仮想化はエッジサーバの運用に適しているといえる．Cloudlet においても，VM の代わりにコンテナによる仮想化を利用したものが研究されている．

コンテナ仮想化ベースの Cloudlet に関する先行研究のうち，Linux Containers (LXC) を仮想環境として採用している [2] は，端末の移動によるサーバの切り替えをサポートしている．これにはコンテナのライブマイグレーションを用いることにより，端末が移動前後で同一のコンテナインスタンスと通信することが可能になる．コンテナライブマイグレーションはプロセスのチェックポイント／リストア技術によって実現する [3]．[2] で使用されているチェックポイント／リストア実装は CRIU (Checkpoint/Restore In Userspace) [8] というシステムソフトウェアである．

2.3　Mobile IP

Mobile IP [4] は IP サービスの継続性を保証し，アクセスネットワーク間のシームレスな端末移動を実現する技術である．これにより，端末が参加するサブネットを変更しても，同じ IP アドレス宛でその端末にパケットを送信することが可能になる．

図 1 は Mobile IP のアーキテクチャを示す．ホームアドレスは，Mobile IP においてユーザ端末 (Mobile Node, MN) に割り当てられる IP アドレスである．これが不変であるために，次のような仕組みが設計されている．いま，MN とサーバ (Corresponding Node, CN) のセッション中に，MN が元のネットワーク (Home Network) から移動先のネットワーク (Foreign Network) に切り替わったとする．Home Network と Foreign Network にはそれぞれ Home Agent (HA) と Foreign Agent (FA) というルータが存在する．MN の移動後に Foreign Network 上で IP アドレス (気付けアドレス) を生成し，FA を通じて HA に登録を行うことで，HA でホームアドレスと気付けアドレスの対応を生成する．さて，CN が MN へパケットを送信するとき，相変わらずホームアドレス宛に送るため，Home Network にルーティングされる．HA はそれを気付けアドレス宛にルーティングし，そのために HA と FA 間にトンネルを作る．そして FA は MN にそれをルーティングする．

2.4　Session and Service Continuity

第 5 世代移動体通信技術 (5G) の要素の一つに Mobile Edge Computing [9] がある．これは移動体通信におけるアクセスネットワークの基地局にエッジサーバ (MEC サーバ) によって実現するエッジコンピューティングアーキテクチャである．そのなかで，モバイル端末の移動によって発生する基地局のハンドオーバーに対処するために，Session and Service Continuity (SSC) という接続方式が定められている．SSC には 3 つのモードがあるが，SSC mode 3 は図 2 のようにアクセスネットワークが変更されたときに新しい MEC サーバに接続しつつ元の MEC サーバの接続を残したままにする方式である [10]．そのため，MEC サーバで処理が進行中の場合にハンドオーバーが発生しても，その処理の結果を移動先のアクセスネットワークで受信することができる．

2.5　Kubernetes

Kubernetes [5] はクラスタ構成に対応したコンテナオーケストレーションツールである．これは Google の社内インフラストラクチャである Borg [11] をもとに設計されており，スケーラブルで高信頼であることと GCP をはじめとする主要クラウドベンダーでサポートされていることから，コンテナオーケストレーションツールのデファクトスタンダードとなっている．

Kubernetes では Pod と呼ばれるアプリケーションの論理的なホストがノード上で動作する．これがデプロイの最小単位となる *2．Pod には 1 つ以上のコンテナと 1 つの

*2　プロダクション環境においては Pod の冗長構成やローリングアップデート構成を使う．

仮想 NIC が含まれる．Pod 同士は Kubernetes クラスタ内部の仮想ネットワークで接続されている．クラスタ外部から Pod へのアクセスを確立するには，ClusterIP Service により仮想ネットワークにおける Pod の IP アドレスを割当て，NodePort Service によりノードの特定のポートをその IP アドレスに転送する [*3]．

3. 提案アーキテクチャ

本研究で提案するコンテナ仮想化ベースの Cloudlet アーキテクチャは図 3 のようになる．オフロードされる処理を行うアプリケーションサービスはコンテナ化されており，コンテナ実行環境上で動作している．コンテナの仮想 NIC はホストの内部でアクセス可能な仮想ネットワークで接続されている．そして Cloudlet には 1 つの Cloudlet Controller コンポーネントがあり，さらにアプリケーションサービスごとに 1 つの Session Service コンポーネントがある．Cloudlet Controller は，コンテナと Session Service の起動と管理を行い，また 3.2.2 節で定義するセッションの開始に必要なセッション ID を配布する．Session Service はクライアントアプリケーションとそのコンテナの通信を中継するものであり，ハンドオーバーのための機能も有している．アプリケーションサービスと Session Service は 1 対 1 で対応付けられている．

3.1 Cloudlet Controller

アプリケーション開発者は，Cloudlet にオフロードしたい処理を行うためのアプリケーションサービスをコンテナ化し，「イメージ」としてビルドする．イメージには，コンテナ内でサービスを開始するための起動スクリプト，コンパイル済みの実行ファイル，および各種ファイルが含まれる．イメージは Cloudlet からアクセス可能なリポジトリに保存しておき，イメージ名をつけて識別できるようにする．

クライアントがオフロード用のコンテナを要求する場合，Cloudlet Controller に接続し，起動したいコンテナのイメージ名を含むリクエストメッセージを送信する．これにより Cloudlet Controller はリポジトリからイメージをロードし，コンテナ実行環境上にコンテナを起動する．同時に，後述の Session Service も起動する．成功すると，コンテナにアクセスするためのポート番号を含むレスポンスメッセージが返される．

3.2 Session Service

コンテナは Cloudlet 内部の仮想ネットワークに接続しており，外部ノードからコンテナへのアクセスを確立するには Cloudlet ホストのポートからコンテナのアドレスにフォワーディングする必要がある．Session Service は，通常はホストのポートからコンテナの仮想 NIC へのポートフォワードを行うためのコンポーネントとして機能する．コンテナ起動の成功時に Cloudlet Controller から返されたポート番号は Session Service がリッスンしている．一方で，フォワーディングの接続先を自分の Cloudlet のコンテナだけでなく別の Cloudlet の Session Service に指定することもでき，この機能を使ってハンドオーバーに対応している．

Session Service を介したクライアントと接続先ホスト間の通信は TCP 上で行われる．そしてその通信は本アーキテクチャにおける後述の「セッション」によって規定される．この仕組みによって，ハンドオーバー前後でのアプリケーションサービスとのコネクションの状態を維持することができる．クライアントがセッションを開始する時に送信するヘッダには以下の情報を含む．ただし接続先ホストのアドレスがデフォルト値である場合は，Session Service に対応するコンテナのアドレスを接続先に指定することを表す．

- セッション ID
- フォワーディングの接続先ホストの IP アドレス (デフォルト値は 0.0.0.0)

3.2.1 ハンドオーバー

端末の移動によって接続先の Cloudlet のハンドオーバーが発生したとき，以前接続していた Cloudlet のコンテナとの通信を継続したい場合，図 4 に示すような通信経路となる．クライアントが移動先の Cloudlet C_{dst} へのアクセスを確立すると，アプリケーションサービスに対応する Session Service S_{dst} に接続する．このときの接続先を移動前の Cloudlet C_{src} に指定する．すると，S_{dst} は指定した接続先に接続するために C_{src} – C_{dst} 間でトンネルを作り，C_{src} 内のアプリケーションサービスに対応する Session Service S_{src} とのフォワーディングを確立する．これによって，S_{src} は S_{dst} とアプリケーションサービスのコンテナとのフォワーディングを確立する．以上により，コンテナ – S_{src} – S_{dst} – クライアントというような通信経路が作られる．

3.2.2 セッション

クライアントが Session Service に接続するときは，Cloudlet Controller から発行されたセッション ID を使う．クライアントが新しいセッション ID で Session Service に接続するとセッションが開始され，Session Service は指定された接続先へ接続し，両者のコネクション間でフォワーディングを行う．ここでクライアントが Session Service との接続をクローズしても，一定時間はセッションが存続しており，その間は接続先へのコネクションは保持される．クライアントが過去に使用したセッション ID で Session Service に接続するとそのセッションが再開され，保持し

[*3] プロダクション環境においてはロードバランサ機能を使用する．

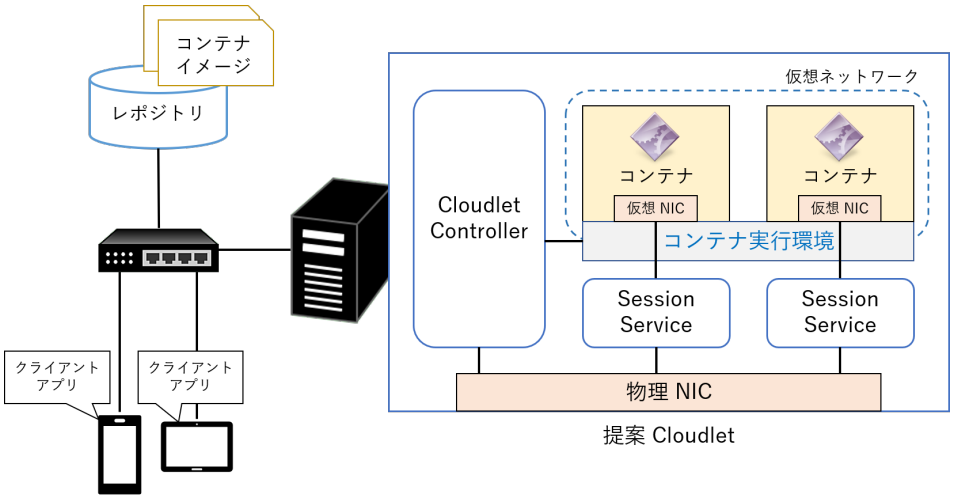

図 3　提案アーキテクチャ

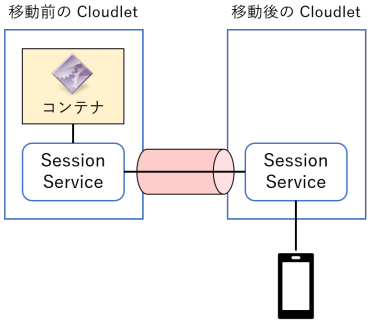

図 4　ハンドオーバー時の通信経路

ておいたコネクションを使ってフォワーディングを再開する．セッションを終了するには，クライアントのクローズ後に一定時間経過するか，特定のメッセージを Session Service に送信する．

ハンドオーバーが行われる前に，端末が移動前の Cloudlet との接続を明示的にクローズしたか (つまり FIN パケットを送信したか) 否かによって，移動前 Cloudlet での Session Service の処理が次のように分かれる．

- 明示的にクローズした場合，移動前の Cloudlet でセッションが存続しているので，それを再開する．
- そうでない場合，移動前の Cloudlet のセッションでは端末とのコネクションが残ったままになっている．そこで，セッションの最中に同一のセッション ID で新しいクライアントからの接続を受理した場合，古いクライアントとのコネクションをクローズして新しいコネクションに交換する．

4. 実装

オリジナルの Cloudlet が OpenStack をベースとしたソフトウェアプラットフォームを利用することで標準的なクラウドコンピューティングの技術に準拠しているのと同様に，本研究においても Kubernetes を利用して実装することでこの要件を満たしている．Kubernetes を利用した Cloudlet はクラスタ構成にすることができるが，今回の実装では簡単のため単一ノードのクラスタとした．

Kubernetes は実際のコンテナ実行環境として Docker [6] を採用している．3.1 節で要求しているイメージのビルドは Docker を使って実施することができる．またイメージの保存場所は Docker Hub [*4] を利用するか，プライベートレジストリを用意する．

提案アーキテクチャにおける Cloudlet Controller によるコンテナの起動・管理は専ら Kubernetes API サーバ (kube-apiserver) を通じて行われる．Kubernetes ではコンテナは Pod 内で動くので，Cloudlet Controller がオフロード用のサービスアプリケーションの起動要求を受けると，Kubernetes API を操作してクラスタに Pod をデプロイする．一方で Session Service は Kuberntes の NodePort Service を拡張した機能を持つため，Cloudlet を利用するクライアントにとってはこれに取って代わられるものとして扱うことができる．

5. 評価

本研究で提案するフォワーディングによるハンドオーバー手法の評価を，従来のコンテナライブマイグレーションによるものと比較して議論する．

5.1 汎用性

コンテナのライブマイグレーションに使用するプロセスのチェックポイント／リストア技術は OS カーネルやシステムコールに依存している．異種 OS 間のコンテナライブマイグレーションを実現する研究も行われている [12] が，実際のソフトウェアで複数 OS に対応しているものは確認されていない．また，ホスト間で CPU アーキテクチャ

[*4] https://hub.docker.com/

表 1　ハンドオーバーのためホスト間の条件

ホスト間の環境	従来手法 (LM)	提案手法 (FW)
TCP 通信	必要	必要
同一 OS	必要	不要
同一 CPU アーキテクチャ	場合により必要	不要

が異なる場合，コンテナ内のプロセスで使われている命令コードのうち移行先ホストの CPU に対応していないものが存在すると実行に失敗する．一方で，フォワーディングであれば OS や CPU の差異は問題にならない．

両ホストでコンテナ仮想化ベースの Cloudlet が動作することを前提条件として，ハンドオーバーの手法としてライブマイグレーション (LM) とフォワーディング (FW) を使用した場合にそれぞれ必要な条件を比較すると表 1 のようになる．

5.2　移行元のサービスの継続性

コンテナライブマイグレーションのためにプロセスのチェックポイントを作成する際，そのプロセスを一時停止する必要がある．ここで，ハンドオーバーを行うクライアントの他に別のクライアントがサービスを使用中である場合，そのクライアントとコンテナの通信が一時的に切断されることになる．したがって，一つのサービスを複数のクライアントが同時に利用するシステムにはライブマイグレーションによる手法は適さない．

一方でフォワーディングによる手法であれば，元のコンテナを一時停止する必要はなく，他のクライアントに影響を与えることもないので，複数クライアントに同時に接続されるサービスであっても問題ない．

5.3　ハンドオーバー性能

コンテナのライブマイグレーションの際には次の手順のように Post-copy 型のページ転送を行うことでサービス開始までの遅延時間を削減できる [3].

(LM-1)まず，対象プロセスを一時停止してチェックポイントをとり，プロセスの再開に必要な最低限の情報だけを移行先ホストに転送してリストアする．

(LM-2)次に，再開したプロセスでページフォルトが発生した順に，移行元ホストのプロセスのページを移行先ホストに転送してゆく．

この方法によるハンドオーバーでは (LM-1) にかかる時間がダウンタイムとなり，また (LM-2) が行われている間はスループットが低下すると考えられる．

一方で，提案アーキテクチャにおけるハンドオーバー処理は次の手順のようになる．

(FW-1)まず，Session Service を開始し，図 4 に示すような通信経路を確立する．

(FW-2)次に，この経路で通信を行う．

この手法ではサービスのダウンタイムは発生しない．しかしクライアント側からすれば (FW-1) にかかる時間はサービス再開までの遅延時間と見なされる．また (FW-2) が行われている間はスループットが低下すると考えられる．

今後，これら 2 つの手法によるハンドオーバー処理にともなうサービス開始までの遅延時間とスループットの低下をそれぞれ測定する実験を行う予定である．

6.　結論

本研究で，コンテナ仮想化による Cloudlet の改良アーキテクチャを提案した．このアーキテクチャは端末の移動にともなう Cloudlet のハンドオーバー処理をフォワーディングにより行うことが可能である．また，Kubernetes を利用することで本アーキテクチャを実装した．

今後の研究計画として，5.3 節で説明したハンドオーバー性能を測定する実験を行う．また，セキュリティに関する問題等の解決を含む，提案アーキテクチャの改良を重ねる．

参考文献

[1] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing*, Vol. 8, No. 4, pp. 14–23, 2009.

[2] Y. Qiu, C. Lung, S. Ajila, and P. Srivastava. Lxc container migration in cloudlets under multipath tcp. In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 2, pp. 31–36, 2017.

[3] Andrey Mirkin, Alexey Kuznetsov, and Kir Kolyshkin. Containers checkpointing and live migration. In *Proceedings of the Linux Symposium*, Vol. 2, pp. 85–90, 2008.

[4] 阪田史郎. Mobile ip. 電子情報通信学会「知識ベース」, 第 4 群, 第 5 編. 電子情報通信学会, 2010.

[5] Cloud Native Computing Foundation. Kubernetes. `https://kubernetes.io/` (2020.06.30).

[6] Docker, Inc. Docker. `https://www.docker.com/` (2020.06.30).

[7] Kiryong Ha and Mahadev Satyanarayanan. Openstack++ for cloudlet deployment. *School of Computer Science Carnegie Mellon University Pittsburgh*, 2015.

[8] Criu. `https://criu.org/Main_Page` (2020.06.30).

[9] Sami Kekki, Walter Featherstone, Yonggang Fang, Pekka Kuure, Alice Li, Anurag Ranjan, Debashish Purkayastha, Feng Jiangping, Danny Frydman, Gianluca Verin, et al. Mec in 5g networks. *ETSI white paper*, Vol. 28, pp. 1–28, 2018.

[10] 音洋行, 滝田亘. 5g 時代の社会インフラに向けたコアネットワーク. NTT DOCOMO テクニカルジャーナル 25 周年記念号, pp. 22–30. NTT ドコモ, 2018.

[11] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at google with borg. In *Proceedings of the Tenth European Conference on Computer Systems*, EuroSys '15, New York, NY, USA, 2015. Association for Computing Machinery.

[12] 雄平高川, 克弥松原. 異種 os 上のコンテナ型仮想化環境間でのライブマイグレーション実現方式の検討. 日本ソフトウェア科学会大会論文集, Vol. 34, pp. 173–178, sep 2017.