Vol.2019-IOT-44 No.23
2019/3/7

ネットワークの状況を考慮したコンテナ型コンテンツ配信基盤の検討

中西 建登[1,a)]　大坐畠 智[1,b)]　山本 嶺[1,c)]　加藤 聰彦[1,d)]

概要：近年，インターネット全体におけるトラフィック量は増加傾向にあり，配信されるコンテンツの品質が向上するとともにコンテンツ毎に利用されるトラフィックは今後も増加していくことが考えられる．現在の IP ベースで構成されたインターネットはサーバクライアントモデルを用いており，クライアントからコンテンツを取得する際，地理的にもネットワーク的にも遠いインターネットを何度か往復した上でコンテンツを取得する必要がある．本稿では利用者により近い箇所にコンテンツやキャッシュを配置するコンテナ技術を用いたコンテンツ配信基盤の検討を行う．コンテナという単一のインターフェースを用いることで，コンテンツそのものや，認証認可，キャッシュなどのコンテンツ配信に必要なコンテナをエッジ上に配置を行う．これにより，クライアントとの往復において通過する経路を短縮し，トラフィックの抑制を実現する．

キーワード：エッジコンピューティング，コンテナ，Kubernetes，サービスメッシュ

1. 背景

現在のインターネットではアプリケーションにアクセスを行うことや動画などのコンテンツを取得する場合，クライアントがサーバの場所に紐付けられた IP アドレスやドメインなどにアクセスを行い，インターネット上に構築されたサーバに接続しその返答を受け取る必要がある．ユーザ側の視点に立って考えた場合，アプリケーションやコンテンツのネットワーク的配置に関しては特に考慮することがなく，適切に返答を受け取ることが可能であれば問題や不満は発生せず，ユーザは出来るだけ高速に返答を受け取ることを期待する．この考えを取り入れた上でネットワークのアーキテクチャを再構築したものとして，CCN (Contents Centric Network) や ICN (Information Centric Network) と呼ばれる，リクエストされたコンテンツをベースとしたネットワーク [1] や，IP ネットワーク上で高速にコンテンツを配信するための CDN (Contents Delivery Network)[2] が提案されている．

CCN や ICN は，現在採用されている IP ベースのルーティングは行うことなく，リクエスト対象であるコンテンツに対してルーティングを行うネットワークアーキテクチャである．CCN ではコンテンツ名を階層構造として扱い，名前解決及びルーティングをネットワーク機器上で行う．しかし，CCN のネットワークは IP ベースのネットワークとは互換性がないため，IP ベースのネットワークが普及した現在は導入するには多くの課題が存在する．

CDN は，サーバを単一点のみに置かず，複数箇所に配置することでバックボーンへのトラヒックを削減するよう設計されている．特に配信するサーバをインターネットの相互接続点 (Internet Exchange) に配置することで，ネットワーク上近い位置に配置することが可能となる [3]．相互接続点に配置するには多くの費用が発生するため，配信するサーバを共有し，サービスとして提供する業者が存在し，CDN 事業者と呼ばれる．配信するサーバは，オリジナルのコンテンツを持つオリジンサーバよりもクライアントに近く，もっともクライアントに近いサーバであることから，エッジサーバと呼ばれる [4]．

「コンテンツがどこから配信されても問題無い」という性質に注目し，近い位置からサービスの提供を行う取り組みとして IP Anycast[5] が利用されている．同じサービスを提供する複数のサーバに対して同じ IP アドレスを割り振ることで，ルーティングによって地理的に近いサーバへアクセスすることが出来るため，ネットワーク遅延を考慮した上で最適なサーバに対して自動的にアクセスが行われ

[1] 電気通信大学 The University of Electro-Communications
[a)] nakanishi@uec.ac.jp
[b)] ohzahata@uec.ac.jp
[c)] ryo-yamamoto@uec.ac.jp
[d)] kato@is.uec.ac.jp

る．ネットワークのルーティングを用いて付近のサーバにアクセスしているため，ネットワーク障害が行った場合でもクライアント側が意識せずに接続先サーバが自動的に切り替わり，この点を考慮したサーバ構築を行う必要がある．具体的には，ユーザによってレスポンスが変わらず状態を持たないステートレスなアプリケーションであれば全てのサーバにおいて差異がないような構築を行ったり，ステートレスでない場合でもバックエンドのデータベースを共通化させたり，他のサーバに移動した場合においてもクライアント側には不整合が発生しないよう設計と構築を行う必要があった．

この問題点を改善するために，コンテナ技術の導入が検討されている [6]．コンテナ実装の中で現在多く利用されている Docker[7] は，コンテナ内で動作する OS イメージが不変であるという性質を持つ．通常サーバにアプリケーションを展開する際には，手作業で同じ作業を繰り返す必要がある．このような操作を複数回行った場合，打ち間違いや権限の設定変更など，いくつかの要因で複数台のサーバ間に設定の差異が発生する可能性が高い．だが，Docker を利用した場合は 1 度ビルドしたイメージは後に変更を行うことが出来ないため，複数台のサーバ上に展開した場合も同じアプリケーションが動作することが担保される．この性質を利用することで，IP Anycast においても存在する「複数台のサーバに全く同じ状態でアプリケーションを展開する」という操作を，差異無く行う事が可能となる．また，これらのコンテナをどの地点に配置するかを自動的に判断し，コンテナを展開しアプリケーション提供を行うための基盤をコンテナオーケストレーターと呼ぶ．その上でコンテナを用いて，機能をコンテナ毎に分割したマイクロサービスを展開するためにはいくつかの諸機能が必要である．これらの機能をアプリケーションコンテナに実装するのではなくサイドカーコンテナとして実行させ，マイクロサービスを機能させるための考え方の 1 つにサービスメッシュがある．実装もいくつか存在しており，コンテナオーケストレーターによってコンテナが移動された場合にどのホストにコンテナがあるのかを判断するサービスディスカバリなどが実装されている場合が多い．

インターネットにおける通信遅延の理由には，光ファイバーを通過する場合に発生する光速の限界や，ルータやスイッチなどのネットワーク機器が処理に必要とするスループットの限界などから発生することが挙げられる．特に前者のケーブル伝送に必要な時間においては技術の向上では解決することが困難であるとされている．

これを解決するために，エッジクラウドコンピューティングが提案されている [8] [9]．エッジクラウドコンピューティングとは，インターネットにおけるトラヒック量の増加や通信の遅延増加を対策するため，携帯基地局などに計算リソースを配置し，モバイル端末と基地局のみで通信を完結させることである．この技術を導入することにより，ユーザに近い部分でアプリケーションを動作させることが可能となるため，近年ではこれらのエッジに対しての利用検討が進んでいる．L. Ma らは OpenFace[10] を用いたアプリケーションを実装し，エッジコンピューティングに Docker の導入を行っている [11]．cri-u (Checkpoint/Restore In Userspace) と呼ばれるコンテナのマイグレーション技術を用いて，ユーザごとに算出結果のことなる OpenFace の結果をユーザの移動によって移動させている．Docker においてコンテナがレイヤー構造で表現されていることに着目し，Docker で当時利用されていた aufs (AnotherUnionFS) における書き込み層をメモリ毎マッピングし移動する手法を提案しているが，ユーザの位置からコンテナの移動先を判定する手法については触れられていない．

本稿では，インターネットにおける遅延を解消するために提案するエッジコンピューティングにおいて起こりえるアプリケーション展開手法の確立及び移動するユーザに対しても問題なくアクセスが行えるようにコンテナオーケストレーターを導入し，既存のコンテナオーケストレーターでは研究が進んでいないネットワーク状況を考慮したコンテナオーケストレーターの実装を行い，実装した環境上での遅延がどの程度発生するのかを評価を行う．

2. 関連技術

2.1 コンテナ管理基盤

コンテナを用いたアプリケーションを運用する場合には，MySQL などのミドルウェアをアプリケーションと同時に実行させ、複数のコンテナを協調動作させる必要が存在する．そのために，Docker のエコシステムの一つとして複数の Docker コンテナを管理する Docker コンテナ管理基盤プロダクトの開発が進められている．

Docker 社が公開している Docker Swarm[12] や，Google 社が開発した Kubernetes[13] などが例として挙げられる．これらはコンテナオーケストレーターなどと呼ばれ，複数のホストサーバの中からサーバの計算機資源の残り状況などを考慮した上で必要数のコンテナを起動する機能や，それらのコンテナに対してアクセスを行うためのロードバランサーアプリケーションを自動的に起動するような機能を持つ．特に Kubernetes は多くの利用者から支持されており，様々なクラウドサービスにおいて，クラウド事業者が Kubernetes の構築，管理，運用を代行し，利用者がホストサーバを意識することなく Kubernetes を利用できるサービスが提供されている．

2.2 Kubernetes におけるコンポーネント

Kubernetes は Google 社が開発した Go 言語によって記述されており、マイクロサービスアーキテクチャのようにいくつかのコンポーネントから成り立っている。本節では

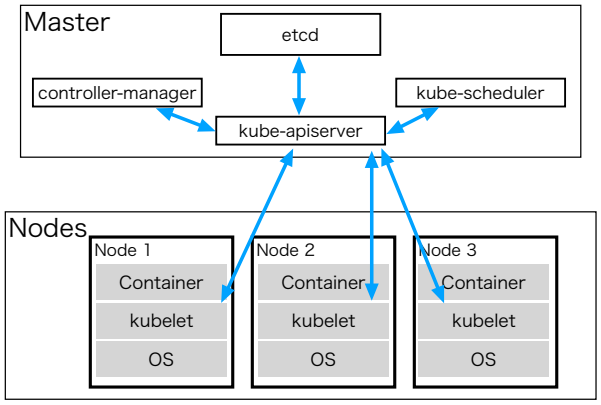

図 1 Kubernetes のコンポーネント概要

本文中にて言及するコンポーネント及び、用語を記述する．概要図を**図 1** に示す．

2.2.1 kube-apiserver

Kubernetes における API を司るコンポーネントの一つ。主に Kubernetes におけるリソースの情報を管理しており、Kubernetes 内外からのアクセスを受けとるように実装されている。Kubernetes のマスターノードという記載をする際には、この kube-apiserver と etcd が動作しているサーバを主に指す。

作業者が Pod や Deployment、Service などのリソースを変更する作業を行うときにアクセスしているのが kube-apiserver である。kube-apiserver は Kubernetes が管理しているリソース情報を全て管理しており、例えば NodePort においてホストのポートが利用出来ない場合などは kube-apiserver がその旨を検知し、Kubernetes に誤った情報を登録せずに作業者に対してエラーを応答する。

2.2.2 etcd

Kubernetes のコアにおけるリソースを保存する永続化層であり、様々な情報を保存するミドルウェアである。分散型データベースとして実装されており、複数台の etcd が強調動作するように動作させる。主に kube-apiserver と共に Kubernetes のマスターノードにおいて動作するが、kube-apiserver と別のサーバで動作させ可用性を向上させることも可能である。他のコンポーネントと異なり Kubernetes が開発したわけではなく、OSS コミュニティによって実装されたアプリケーションである。

etcd が Kubernetes コアにおける数少ない永続化層であり、etcd クラスタは非常に安定した部分に配置されることが要求され．kube-apiserver は etcd に保存されたリソース情報にアクセスして展開可能か判断するため，kube-apiserver が一時的に動作できない状況に陥っても、etcd クラスタが起動していれば復旧は比較的容易である。

2.2.3 kube-controller-manager

Kubernetes におけるリソース管理を司るコンポーネントの一つ。etcd における Deployment などの Pod 以外のリソース情報を定期的に確認し、Deployment などを解釈して Pod リソースを作成する。

Deployment などでレプリカ数を規定するとその分の Pod が作成されるのはこのコンポーネントが動作していることに起因する。

2.2.4 kube-sheduler

Kubernetes におけるスケジューリングを司るコンポーネントの一つ。etcd における Pod リソース情報を定期的に確認し、Kubernetes に参加しているホストのリソース情報などを確認した上でどのノードにどの Pod を展開するべきかのスケジューリングを行う。

2.2.5 kubelet

Kubernetes におけるホスト管理を司るコンポーネントの一つ。Kubernetes に参加しているノードという記載をする場合には、この kubelet が動作しているサーバを主に指す。kubelet はリソース情報を確認した上で、自分のホストに割り当てられたコンテナがあった場合には自ホストで同時に動いている Docker Engine などのコンテナ管理用 API に対してコンテナ作成要求などを発行する。

kubelet は自分の所属するホストの情報を動的に送信する動作を行っており、ホストの情報が変わった場合や、自分のホスト上で動いているコンテナなどに変化が起きた場合には自動で回復を行ったり、他のホストにコンテナ起動の要求を行ったりなどの動作を行う。

2.3 サービスメッシュ

Kubernetes などのコンテナオーケストレーターを用いてマイクロサービスアーキテクチャを構築し運用する場合，いくつか考慮の必要がある点が存在する．具体的には以下の事柄が考えられる．

- ネットワークは何らかの問題で動作しなくなる可能性があるため，タイムアウトを適切に設定する (Timeout)
- タイムアウトなどの理由でリクエストに失敗した場合，適切な回数リトライを行う (Backoff Retry)
- 各コンポーネントごとに認証認可を行い，適切なリクエストであるかどうかを確認する (Authentication / Authorization)
- もし異常なリクエストがコンポーネントに送られてきた場合，リクエスト制限を発行し適切にリクエストを削除する (Rate-Limit)
- コンテナの移動などの理由でアプリケーションが移動した場合，適切にルーティングを行う (Service Discovery)

特にアプリケーションが起動したときや移動したときなどに，適切にルーティングを行いルーティングの変更をお

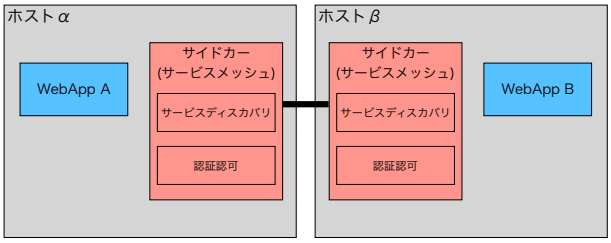

図 2 サイドカーとしてサービスメッシュ用コンテナが起動する

こなうことをサービスディスカバリと呼び，設定次第ではKubernetesにおけるノードのどのノードにアクセスを行っても，適切にサービスが動作しているコンテナにルーティングを行うことが可能である．

著名な実装として，Lyft 社が開発した Envoy をコアに採用した Istio や，Buoyant 社が開発した Linkerd などが挙げられる．

実装としてはメインサービスのコンテナと同時に起動するサイドカーコンテナとして起動する手法が採られている．概要図を**図 2** に示す．これらのサイドカーコンテナがHTTP 通信の終端を行っており，適切なルーティングを行う際にも別のサイドカーコンテナに転送を行うため，それぞれのホストが接続し合いメッシュ状に接続を行う．

2.4 L2 を廃する DC ネットワーク

パブリッククラウドサービスベンダーを始めとして多くのサーバを扱う事業者の場合，大量の物理サーバを管理する必要が存在する．大量の物理サーバを動作させる場合には大量の電力が必要となったり多くの排熱が発生するため生活空間とは分離することが多く，サーバを運用する専用施設をデータセンターと呼ぶ．データセンターにおいては多くのサーバを管理するため特殊なネットワーク構造が取られることが多く，これをデータセンターネットワークと呼ぶ．データセンターネットワークにおいては大手アプリケーション事業者を中心に多くの研究が進んでおり，いくつかの論文も発表されている．

C. Clos が提案 [14] したネットワークは Clos トポロジーと言われ，多くの事業者において拡張したものが採用されている [15] [16]．Facebook を提供する Facebook 社においても同様に拡張を行ったネットワークアーキテクチャを構築しており，N. Farrington らによって Fabric architecture として公開されている [17]．トポロジを 3 次元に展開し，ラック、Pod，Plane のどの次元においても拡張することを可能としたアーキテクチャである。

さらにこのデータセンターネットワークアーキテクチャを発展させたものとして，動的ルーティングプロトコルである BGP を用いて Layer2 を廃したデータセンターネットワークアーキテクチャが LINE 社 [18] より提案されている．計算機リソースやネットワークスイッチとしては Fabric アーキテクチャと同様であるものの，それらの接続プロトコルとして BGP を用いることを提案している．

LINE 社の提案では物理サーバ 1 台ごとに BGP を用いたデーモンを起動しネットワークの疎通性を確保する．物理サーバと ToR との接続のためにサブネットマスク /31 の経路のみを広報する．このサブネットマスクは IP アドレスが 2 つだけであり，サーバ側の IP アドレスと ToR 側の IP アドレスの 2 つだけが存在する Layer2 空間である．ToR は各サーバと /31 の BGP 接続を多く結び，サーバへの経路情報を得る．ToR はそれらのサーバ接続への経路をまとめ，1 つの経路として更に上位のルータに広報を行う．この動作を Spine スイッチに接続するまで行い，全ての経路接続を完了する．このアーキテクチャを採用することにより，BGP という汎用的なプロトコルを用いてベンダーに縛られることなく大幅な拡張性を確保することが可能となる．

3. 提案手法

3.1 概要

本稿では，エッジクラウドコンピューティングにおいてKubenetes を用いたサービスメッシュを導入することに対しての検討，実装，評価を行う．

エッジクラウドコンピューティングにおける現状の課題として，アプリケーションやミドルウェアの展開についての手法が十分に検討されていないことが挙げられる．もし多くの基地局にエッジノードとして計算機資源を配置した場合，既存のアプリケーションを展開するためのクラスタ群とはノード数が大きくことなり，ノード 1 台 1 台にログインするなどしてアプリケーションやサーバを展開することは現実的ではない．これらのアプリケーションやミドルウェアを展開する場合に Kubernetes を採用することを考慮するが，その場合にも課題が存在する．

Kubernetes は Cloud Native Computing Foundation[19] と呼ばれる財団によってホスティングされており，開発に当たって様々な支援を受けている．Google 社が開発した経緯としても多くのノードが運用されているデータセンター上で開発されたクラウドを含めて分散的に運用されるよう実装されており，Kubernetes の動作としてもエッジクラウド上で動作させる事に関してはそれほど考慮されていない．特に，コンテナの配置状況においてはサーバのリソースや高速に起動することが可能であるかどうかなどを基準に起動しているが，複数のネットワークに跨がって起動させた場合にそのコンテナがネットワーク的にどの位置に配置されるのかなどは優先度設定などには規定されていない．そこで，Kubernetes の既存スケジューラを参考にエッジクラウドコンピューティングに適した Kubernetes 向けスケジューラの開発を行った．具体的には BGP のもつ AS Path の機能に着目し，AS Path の長さがネットワー

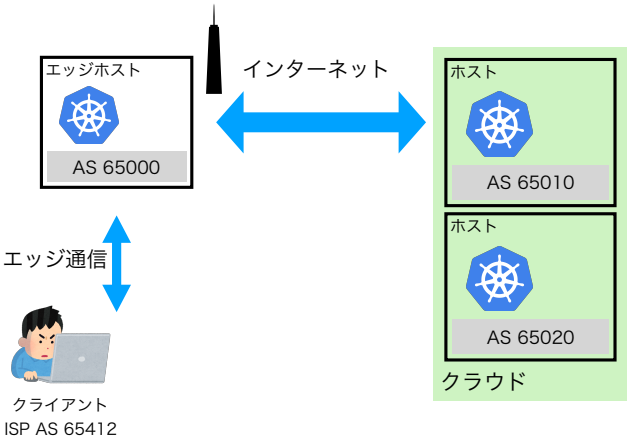

図 3 提案方式の概要図

クにおける距離に相当し，ユーザの用いている ISP により近い箇所に配置された Kubernetes のノードにアプリケーションを展開する事で，ネットワークの状態を考慮した展開を実現する．

本提案において構築したシステムの概要図を**図 3** に示す．エッジホスト，ホストはサーバ群を示しており，一つの Kubernetes クラスタを構築している．これにより，クラウド上に動作しているコンテナをエッジに移動したり，その逆を行うことが可能となる．AS 65000 はエッジホストとなっており，携帯基地局などユーザの用いている ISP に非常に近い箇所に設置されたホストであることを想定している．そのため，クライアントである AS 65412 と AS 65000 のみで通信が完結する場合は，エッジでの通信で完結するため非常に高速な通信を低遅延で行うことが可能となる．AS65010 と AS65020 はクラウド上で展開されたサーバであり，クライアントからアクセスを行う場合はインターネット上に発生する遅延を受けた上で通信を行う．

3.2 スケジューラの実装

Kubernetes におけるスケジューラは，kube-scheduler と呼ばれるコンポーネントとして実装されており，GitHub 上にてホスティングされている [20]．

kube-scheduler におけるホスト割り当て作業は，Pod を展開するための条件である predicates と優先度付けである priorities の 2 点から評価を行う．今回実装した kube-scheduler における動作アルゴリズムの疑似コードをソースコード 1 に示す．

本論文ではスケジューラの実装において，ユーザにより近い Kubernetes に参加しているノードへのデプロイを行い，エッジコンピューティングにおいてユーザにより近いノードへのデプロイによって通信量の削減やユーザ体験の向上を目指す．そのため，Clos トポロジをベースとしたデータセンターネットワークを摸倣したトポロジを構築

ソースコード 1 実装したアルゴリズムの擬似コード

```
if クライアントからのアクセス == 発生
  クライアントISPのAS番号を取得 (*)

if クライアントISPのAS番号 != ""
  Kubernetesの各ノードとのAS Pathをそれぞれ算出 (*)
  AS Pathから最も近いノードを判定 (*)
  AS Pathが近いノードの優先度を上昇させる (*)
  優先度の変化が発生した場合はコンテナの移動命令を生成

if コンテナの移動命令 == 発生
  AS Pathが近いノードでコンテナを起動
  キャッシュコンテナの名前解決結果をエッジノードに切替
  クラウド上のキャッシュコンテナを停止

(*) 新たに実装
```

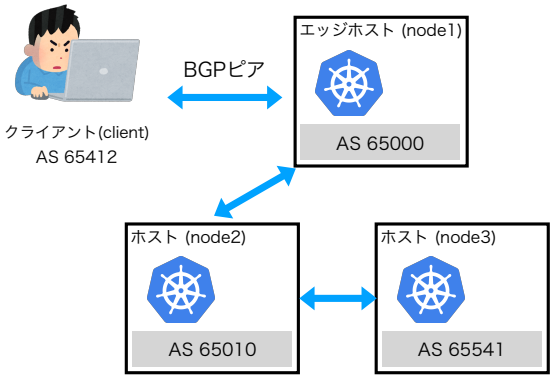

図 4 BGP ピアの状況

し，その上で AS Path を考慮した Kubernetes スケジューラの実装を行った．具体的には kube-scheduler における優先度において AS Path の計算を行い，AS Path が短ければ優先度を高く付けるような実装を行った．

3.3 BGP サーバの実装

LINE 社 [18] の提案する手法により，ホスト 1 台につき 1 台の AS が割り振られており，BGP デーモンが動作している状況を実装する．実装には GoBGP[21] を用いて実装を行った．構成は**図 4** に示す．自らの IP アドレスを router-id とし，それぞれの IP アドレスについて広報を行った．今回は実際の AS を用いることができないため，プライベートな AS 番号を割り当てた上で設定を行った．また，それぞれで/32 の経路を広報し，AS Path の算出を行えるように設定を行った．クライアントからのアクセスが発生

表 1　実験環境

役割	型番/バージョン
ホストサーバ	Dell PowrtEdge R440
ホスト CPU	Intel (R) Xeon (R) Silver 4114
ホストメモリ	64GB
ホスト OS	Ubuntu 18.04.1 LTS (Bionic Beaver)
Kubernetes バージョン	v1.13.2
Kubernetes 構築手法	kubeadm v1.13.2
GoBGP	v2.0.0
Varnish	v6.0.2 LTS

表 2　ゲストマシン

役割	割当
CPU	4 vCPU
メモリ	4GB
OS	Ubuntu 18.04.1 LTS (Bionic Beaver)

しコンテナがエッジに移動することが適切であると判断された場合，クライアントの ISP である AS 65412 との AS Path をそれぞれ返答する HTTP サーバを起動しておき，Kubernetes はそのサーバに問い合わせることで各ノードとクライアントとの距離を知り，エッジ判定に利用する．

4. 評価実験

エッジクラウドコンピューティングの状況を再現し，その上でエッジ，クラウドにおけるネットワークを考慮する．

4.1 実験環境

実験環境を**表 1** に示す．実験に必要なゲストサーバについて**表 2** に示す．ゲストサーバは 4 台存在しており，うち 3 台が Kubernetes のクラスタであり，もう 1 台をクライアントとして実験を行った．Kubernetes 3 台のうち 2 台をマスターノードとし，マスターノードを含む 3 台全台に Pod が展開可能な状態とした．

4.2 キャッシュ検知による移動

クラウド上に展開されたアプリケーションにおいて，動画などのコンテンツを配信する状況は多く存在する．アプリケーションが HTTP リクエストに対して動画を返答する場合，アプリケーション上で動画をメモリ上に保持した上で HTTP レスポンスを発行する必要があるため非常に負荷が高い動作となる．これを回避するため，アプリケーションの前段に HTTP リクエストとレスポンスをキャッシュしておくキャッシュサーバを配置する手法が採られる．HTTP のキャッシュサーバについての実装は多く存在しているが，著名なものとして Varnish が挙げられる．クラウド上にアプリケーションを展開した際にアプリケーションの前段に Varnish などのサービスを配置しておき，キャッシュが可能なコンテンツに関しては Varnish にキャッシュする手法が多く採られる．しかしこの場合，インターネットを経由して動画コンテンツを配信するため，クラウド上の転送量を抑えることが可能でも，インターネット全体における転送量が非常に多くなってしまう課題が存在する．

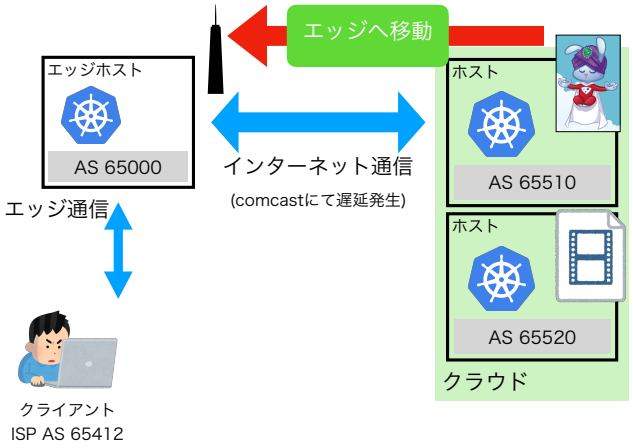

図 5　本実験の概要図

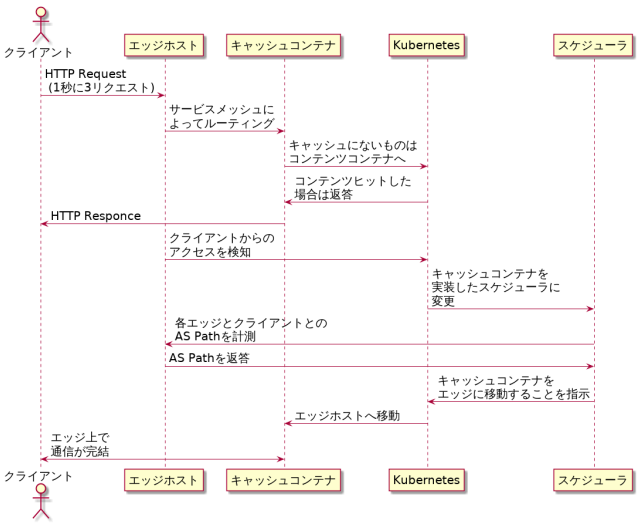

図 6　提案手法のシーケンス図

本節ではこの構成を取った場合に，キャッシュサーバである Varnish がリクエストに応じてエッジに移動させ，エッジ上で再度キャッシュを取得する状況を想定し実験を行う．ユーザからのアクセスに基づいてエッジに移動を行い，キャッシュを発見した後の返答を高速に行うことを想定する．概要図を**図 5** に示し，動作のシーケンス図を**図 6** に示す．

エッジノードと AS 65510 との間には tc コマンドと iptables コマンドをラップすることのできる comcast [22] v1.0.1 を用いて，エッジノードとクラウド上のサーバ間にレイテンシを設定した．コンテンツは AS 65520 に設置されており，クライアントから最も遠いノードに設置されている．これは Kubernetes の Pod として設置されており，Web サーバである Nginx コンテナ内に動画ファイルを配置し，配信されている．comcast の設定によりインターネッ

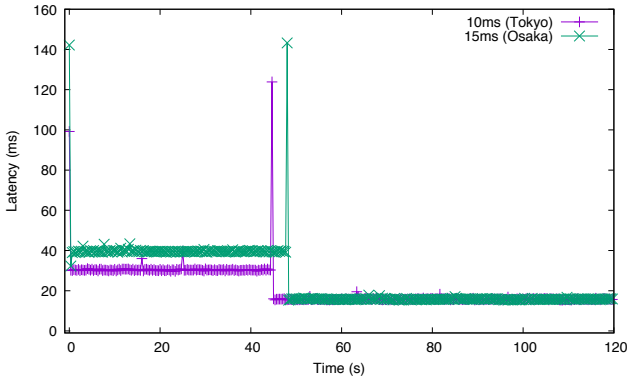

図 7 実験結果

トを模した回線を通過する必要があるため，クライアントから直接取得する場合遅延を含めた通信が必要となる．

実験開始時，Varnish コンテナは AS 65510 若しくは AS 65520 に配置されている．Varnish は HTTP におけるキャッシュサーバであり，HTTP のレスポンスを保存した上で同様のリクエストであればキャッシュをおこない，オリジンサーバにリクエストを飛ばすことなく返答を行うことが可能となる．

クライアントが HTTP アクセスを開始し，クライアントの経路を検知した上でクライアントまでの AS Path を計測し，AS Path の近い地点に Varnish コンテナを移動させる実験を行った．シミュレーションとして，HTTP アクセスを開始してから 10 秒後に Varnish コンテナが AS Path に従いエッジノードに移動を行い，アクセスをエッジで終端させるシナリオを考える．

対象コンテンツは 1.7MB の MP4 動画である．vegeta は都度 GET リクエストを送信し，クライアントに 1.7MB の動画ファイルがダウンロードされるまでのレスポンス時間を計測する．

4.3 クライアントとアクセス先のクラウドへの距離が変動する場合

エッジコンピューティングにおいて，地域に対するクラウド展開は非常に重要な値である．どの程度の地域であればエッジコンピューティングと本手法を導入した際に大きな成果を発揮するのか把握するため，comcast を用いて発生させる遅延を変化させ，どの程度レスポンス時間に変化が見られるのか実験を行う．なお，地域におけるレイテンシの参考値として，Amazon 社の提供する AWS 上の各リージョンにおける死活監視用の IP アドレス [23] に向けて，筆者の所属先である東京にある電気通信大学から ping コマンドを用いてパケットを送信し，概算値を計測したものである．

結果を**図 7** に示す．10ms のグラフにおいて，クラウド上に配置されていた時は約 30ms の遅延が発生しており，通信経路の遅延のみでなくアプリケーションの応答にかかっている時間も存在すると考察できるが，エッジに移動した後は 15ms 程度に遅延が抑えられており，クラウド上に配置されていたときと比較して本計測においても効果があると考察される．想定地点が東京であり，地理的にかなり近い部分に配置されている場合にあっても，キャッシュコンテナを基地局などのエッジに配置することで効果が発揮されると考えられる．

10ms のグラフと比較して，15ms のグラフにおいて初回リクエストに対するレスポンスが若干遅延していると考えられるが，これは 15ms に遅延を増やした際に発生したものと考えられる．また，クラウド上にある場合は 40ms かかっていたレスポンスも，エッジに移動後は 15ms ほどに抑えられており，エッジを移動したことによる遅延を減少が成功していると考えられる．また，レスポンス時感が 15ms という値は 10ms の遅延を発生させたグラフにおけるエッジ移動後の遅延時間とほぼ同一であり，このことからエッジに移動することでクラウドへの遅延時間に左右されずエッジに移動し終わることで高速化が行えたと考えられる．

開始直後のリクエスト，及び実験開始から 50 秒前後にレスポンスが遅いリクエストが存在しており，それ以降はそのリクエストと比較して高速に返答が行われている．実験開始から 10 秒経過した時点でキャッシュコンテナの移動の指示が出されているため，コンテナの起動においては 30 秒ほど要しており，コンテナが起動次第自動的にエッジに配置されたコンテナにキャッシュされていると考えられる．50 秒前後に発生した山はコンテナがエッジに移動した直後のリクエストであり，この 1 回のみインターネットを通過しクラウド上に存在する動画ファイルを取得しているものと考えられる．

二つ目の山が発生する時間，つまりキャッシュコンテナのエッジへの移動が終了し終えた時間において考察を行う．キャッシュがエッジに移動したのちのレスポンス遅延についても考察を行う．全てのグラフに共通して当初のレスポンスにかかった時間は最初の 1 リクエストとエッジに移動した後のリクエストの両方においてほぼ同一の値を示しており，どちらもクラウド上に配置された Nginx コンテナにアクセスしたと考えられる．エッジ移動後のレスポンスを高速化する手法として，エッジに 2 段キャッシュを配置するということも考えられる．このようなコンテナ配置を実現することで，コンテナの移動などで一時的に片側のキャッシュコンテナが利用できない状況にあったとしても，もう片側のコンテナに配置することが考えられる．このような配置においても Kubernetes の Deployment とサービスメッシュを組み合わせることで実現は容易であると考えられる．

5. まとめ

本稿では，計算機リソースをユーザの近くにある携帯基地局などに配置し，インターネットにおける遅延の抑制やトラヒック量の抑制に注目されるエッジコンピューティングにおいて，コンテナオーケストレーターである Kubernetes を用いたサービスメッシュ環境を構築し，またその Kubernetes を拡張することでネットワーク状況を考慮したコンテナ配置を行えるように実装と評価を行った．

その結果，Kubernetes がキャッシュコンテナを移動することで HTTP におけるレスポンスにかかる時間が抑制され，エッジノードへの移動が有用であると考えられる．

今後の予定としては，いくつかの仮定を現実世界上でも適用出来るよう実装を進めていく．Kubernetes 内では他の AS とは接続が行えないプライベートな AS 番号を割り振り，AS Path の計算を行う際には NAT のようなもので AS 番号のグローバルとプライベートを変換した上で AS Path の算出をおこない，よりユーザに近いところにコンテナを配置すべきであると考えられる．また，現在のコンテナ移動の動作は Kubernetes 上の機能を用いて行われているものであるが，これに対してもより良い移動手法がないのか検討を行い，実装を行っていく予定である．

参考文献

[1] Ahlgren, B., Dannewitz, C., Imbrenda, C., Kutscher, D. and Ohlman, B.: A survey of information-centric networking, *IEEE Communications Magazine*, Vol. 50, No. 7, pp. 26–36 (online), DOI: 10.1109/MCOM.2012.6231276 (2012).

[2] Pathan, A. and Buyya, R.A taxonomy and survey of content delivery networks,, http://www.gridbus.org/reports/CDN-Taxonomy.pdf, Technical Report, GRIDS-TR-2007-4, Grid Computing and Distributed Systems Laboratory, The University of Melbourne, Australia. (2007).

[3] Böttger, T., Cuadrado, F., Tyson, G., Castro, I. and Uhlig, S.: Open Connect Everywhere: A Glimpse at the Internet Ecosystem Through the Lens of the Netflix CDN, *SIGCOMM Comput. Commun. Rev.*, Vol. 48, No. 1, pp. 28–34 (online), DOI: 10.1145/3211852.3211857 (2018).

[4] Su, A.-J., Choffnes, D. R., Kuzmanovic, A. and Bustamante, F. E.: Drafting Behind Akamai: Inferring Network Conditions Based on CDN Redirections, *IEEE/ACM Trans. Netw.*, Vol. 17, No. 6, pp. 1752–1765 (online), DOI: 10.1109/TNET.2009.2022157 (2009).

[5] BCP 126 - Operation of Anycast Services, `https://tools.ietf.org/html/bcp126`. <2019 年 2 月 4 日 参照 >.

[6] Pahl, C., Helmer, S., Miori, L., Sanin, J. and Lee, B.: A Container-Based Edge Cloud PaaS Architecture Based on Raspberry Pi Clusters, *2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, pp. 117–124 (online), DOI: 10.1109/W-FiCloud.2016.36 (2016).

[7] Boettiger, C.: An Introduction to Docker for Reproducible Research, *SIGOPS Oper. Syst. Rev.*, Vol. 49, No. 1, pp. 71–79 (online), DOI: 10.1145/2723872.2723882 (2015).

[8] Yu, W., Liang, F., He, X., Hatcher, W. G., Lu, C., Lin, J. and Yang, X.: A Survey on the Edge Computing for the Internet of Things, *IEEE Access*, Vol. 6, pp. 6900–6919 (online), DOI: 10.1109/ACCESS.2017.2778504 (2018).

[9] Abbas, N., Zhang, Y., Taherkordi, A. and Skeie, T.: Mobile Edge Computing: A Survey, *IEEE Internet of Things Journal*, Vol. 5, No. 1, pp. 450–465 (online), DOI: 10.1109/JIOT.2017.2750180 (2018).

[10] Baltrusaitis, T., Zadeh, A., Lim, Y. C. and Morency, L.: OpenFace 2.0: Facial Behavior Analysis Toolkit, *2018 13th IEEE International Conference on Automatic Face Gesture Recognition (FG 2018)*, pp. 59–66 (online), DOI: 10.1109/FG.2018.00019 (2018).

[11] Ma, L., Yi, S. and Li, Q.: Efficient Service Handoff Across Edge Servers via Docker Container Migration, *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, SEC '17, New York, NY, USA, ACM, pp. 11:1–11:13 (online), DOI: 10.1145/3132211.3134460 (2017).

[12] Swarm mode overview., `https://docs.docker.com/engine/swarm/`. <2019 年 2 月 4 日 参照 >.

[13] Burns, B., Grant, B., Oppenheimer, D., Brewer, E. and Wilkes, J.: Borg, omega, and kubernetes, *ACM Queue*, Vol. 14, No. 1, p. 10 (2016).

[14] Clos, C.: A study of non-blocking switching networks, *The Bell System Technical Journal*, Vol. 32, No. 2, pp. 406–424 (online), DOI: 10.1002/j.1538-7305.1953.tb01433.x (1953).

[15] Xia, W., Zhao, P., Wen, Y. and Xie, H.: A Survey on Data Center Networking (DCN): Infrastructure and Operations, *IEEE Communications Surveys Tutorials*, Vol. 19, No. 1, pp. 640–656 (online), DOI: 10.1109/COMST.2016.2626784 (2017).

[16] Greenberg, A., Hamilton, J. R., Jain, N., Kandula, S., Kim, C., Lahiri, P., Maltz, D. A., Patel, P. and Sengupta, S.: VL2: A Scalable and Flexible Data Center Network, *SIGCOMM Comput. Commun. Rev.*, Vol. 39, No. 4, pp. 51–62 (online), DOI: 10.1145/1594977.1592576 (2009).

[17] Farrington, N. and Andreyev, A.: Facebook's data center network architecture, *2013 Optical Interconnects Conference*, pp. 49–50 (online), DOI: 10.1109/OIC.2013.6552917 (2013).

[18] LINE のネットワークをゼロから再設計した話 :: JANOG43, `https://www.janog.gr.jp/meeting/janog43/program/line`. <2019 年 2 月 4 日 参照 >.

[19] Cloud Native Computing Foundation, `https://www.cncf.io/`. <2019 年 2 月 4 日参照 >.

[20] kubernetes/kubernetes, Production-Grade Container Scheduling and Management., url-https://github.com/kubernetes/kubernetes/. <2019 年 2 月 4 日 参照 >.

[21] osrg/gobgp: BGP implemented in the Go Programming Language., `https://github.com/osrg/gobgp`. <2019 年 2 月 4 日 参照 >.

[22] tylertreat/comcast: Simulating shitty network connections so you can build better systems., `https://github.com/tylertreat/comcast`. <2019 年 2 月 4 日参照 >.

[23] EC2 Reachability Test., `http://ec2-reachability.amazonaws.com/`. <2019 年 2 月 4 日 参照 >.