エッジコンピューティングを考慮したコンテナオーケストレーションシステムに関する研究

2016SE047 前田幹英　2016SE078 高橋良輔
2016SE086 内山健太朗
指導教員：宮澤 元

1　はじめに

Internet of Things (IoT) の普及に伴い，エッジコンピューティングが注目されている．エッジコンピューティングとは，ユーザからネットワーク的に近い場所に設置されたエッジサーバと呼ばれる計算ノード上でサービスの処理を実行することにより，クラウドコンピューティング (クラウド) への負荷を分散させたり，通信遅延を削減してリアルタイム性を確保するための技術である．

エッジコンピューティングを効果的に利用するためには，サービスの処理を適切な計算ノードで実行させることが重要である．データセンタ内に設置された均質な計算ノードを利用できるクラウドとは異なり，エッジコンピューティングでは様々な計算ノードがネットワーク上に分散配置されるので，利用者である IoT デバイスからのネットワーク距離やサービスの処理に必要な計算能力を考慮してサービスの処理を実行する計算ノードを決定する必要がある．

クラウドにおいてコンテナ化されたサービスの処理を計算ノードに配置するのに利用されているコンテナオーケストレーションシステムを拡張してエッジコンピューティングで利用できるようにするための研究が行われている．Dupont らは IoT デバイスの位置やエッジサーバの負荷状況に応じてコンテナのマイグレーションを行う Cloud4IoT と呼ばれるコンテナオーケストレーションシステムを提案しているが，適切な計算ノードを決定する具体的なアルゴリズムは示していない [1]．深見と早川は IoT デバイスから送信したデータをクラウドサーバエッジサーバで処理するようなアプリケーションを想定し，最適なコンテナ配置がアプリケーションの通信データ量によって異なることを示したが，比較的処理性能の低い計算ノードだけがエッジサーバとして利用されることを前提としており，計算ノードごとに処理性能が異なるような状況を想定していない [2]．

本稿では，ファイル転送を行う IoT サービスにおけるファイル転送時間と処理時間について述べる．転送ファイルサイズやエッジサーバの性能の違いに応じて，これらの時間がどのように変化するか実験で確かめる．実験結果をもとに，転送ファイルサイズやエッジサーバの性能の違いに応じた最適なコンテナ配置を示すとともに，実際に最適なコンテナ配置を行うようにコンテナオーケストレーションシステムを制御する IoT サービスを作成し，動作を確認する．

2　研究の背景

この節では，コンテナオーケストレーションシステムについて示す．IoT サービスにおけるエッジコンピューティングを考慮した最適なコンテナオーケストレーションシステムに関する先行研究について述べたうえで，エッジコンピューティングにおけるコンテナオーケストレーションシステムの課題について示す．

2.1　コンテナオーケストレーションシステム

仮想化の手法のひとつにコンテナ仮想化がある，これはコンテナと呼ばれる仮想的なユーザ空間を提供する技術であり，1 台の物理サーバ上で複数のコンテナを用いることによって，それぞれが独立したサーバ環境であるかのように利用できる．

しかし，コンテナ仮想化は，コンテナを開発するまでは容易であるという反面，コンテナの数が増えてくると管理と運用に手間がかかってしまうという問題がある．

この問題を解決するためにコンテナオーケストレーションシステムがよく用いられる．コンテナオーケストレーションシステムとは，コンテナの監視と制御を行うシステムである (Kubernetes, Mesosphere, Docker Swarm など) このシステムの主な役割は，コンテナのプロビジョニング，監視，ローリングアップグレード，ロールバックなどがある．

コンテナオーケストレーションシステムの一つに Kubernetes と呼ばれるものがある．Kubernetes とは，コンテナ型の仮想化ソフトウェア (Docker など) のコンテナホストのクラスタを管理するコンテナオーケストレーションシステムである．Kubernetes を用いれば，複数台のホストから構成される実行環境であっても一台の実行環境のように扱うことができる．ハードウェアが，複数存在していてもコンテナを適切に配置して要件通り動かしたりリクエストを複数のコンテナに割り振って実行することで負荷を分散したりすることができる．

2.2　コンテナ配置の最適化

kubernetes の機能を活かしてクラウドサーバ，エッジサーバ，IoT デバイスの 3 つの層で構成された IoT サービスを Kubernetes クラスタで構成することでクラウドとエッジと IoT デバイス間のオフロードを可能にする研究が行われている [1]．その研究では，IoT サービスの機能をコンテナ化してコンテナを最適な配置で実行できる

Cloud4IoT と呼ばれるプラットフォームの研究がされている．Cloud4IoT は，IoT デバイスが移動する場合への対応やゲートウェイへの負荷集中した場合への対処を実現している．この研究のプラットフォームには，Docker と Kubernetes を用いている．

しかし，コンテナを動的に配置させる際にどのコンテナをどこに配置するのかを様々なケースに合わせて自動的に判断させることは容易ではない．もし，誤って処理の大きいクラウドサーバで実行するべき処理をエッジサーバで実行してしまった場合，エッジサーバの処理能力はクラウドサーバに比べて劣るため余計に処理が遅くなってしまう．このような状況を防ぐためにエッジコンピューティングを用いた IoT サービスにおいてコンテナを最適なサーバで実行するための条件について検討を行ってその条件に合うようにコンテナを自動配置するコンテナオーケストレーションシステムを実現しなければならない．

そこで IoT デバイスからサーバに何らかのデータを送信する IoT サービスを想定して，条件ごとに処理にかかる時間がどのように変化するかを調べた先行研究がある [2]．
その実験結果から，データ送信回数やデータのサイズに応じてコンテナを動作させるサーバを変更することで，処理時間が短くなることがわかった．具体的には，二桁の整数のようなとても小さいデータを多数送信する場合には，通信レイテンシが小さいエッジサーバに送信して処理をした方が効率が良く，動画のようなデータのサイズが大きい場合には処理能力が高いクラウドサーバで処理をしたほうが処理にかかる時間が早いということがわかった．

この実験では，処理に用いるデバイスを二種類しか用いていないので，処理能力の違いによる処理時間の変化が単純な結果しか得られていない．そのため，より多くの種類のデバイスを用意して，処理や転送にかかる時間を測定し，処理時間がどのように変化していくかを実験する必要がある．

3　最適なコンテナ配置条件の検討

データサイズの違いだけではなく，エッジサーバの性能の違いを考慮することにより，応答時間を短くするコンテナ配置の最適な場所は存在すると考えた．よって，コンテナを用いて，いくつかの IoT のサービスの処理を実行する上で，コンテナを配置するノードを変更し，エッジサーバの性能を実際に変化させる形で予備実験を行った．その結果から，応答時間が短くなる組み合わせを特定し，エッジサーバの性能の違いも考慮した最適なコンテナ配置を決める．今回は，IoT デバイスと処理能力が近いノード，クラウドと処理能力が近いノード，IoT デバイスよりは処理が速いが，クラウドよりは処理が遅いノードと，三種類のノードを用意し実験を行う．特定した組み合わせをもとに，より良い条件の検討を行う．

4　予備実験

この節では，エッジコンピューティングを用いた IoT サービスにおいて最適なコンテナ配置を行うための条件について検討を行う．

4.1　実験内容

IoT デバイスで生成されたデータを転送することを想定し、IoT デバイスノードからエッジノードを経由してクラウドノードまで転送する。実験では、事前に用意したサイズが異なる二つの動画ファイルを転送する。この時、IoT デバイスノード，エッジノード，クラウドノードのいずれかで動画の圧縮処理を行い，ファイル転送を含む全体の処理にかかった時間を測定する．また，tc コマンドを用いて IoT デバイス，エッジサーバ間に帯域幅 (50Mbps)，レイテ ンシ (5ms) とエッジサーバ，クラウドサーバ間に帯域幅 (50Mbps)，レイテンシ (50ms) の制限をかけることで実際 の IoT サービスにおける通信遅延の差を再現した．

具体的には以下の 4 つの実験を行う．エッジノードとして使用するコンピュータを複数種類用意し、エッジノードに使用するコンピュータをかえて転送時間を測定する．

ファイル転送 (大)　事前に用意した 550.4MB の動画ファイルを転送する

ファイル転送および圧縮 (大)　550.4MB の動画ファイルに圧縮処理をした動 画ファイルを転送する

ファイル転送 (小)　事前に用意した 15.4MB の動画ファイルを転 送する

ファイル転送および圧縮 (小)　15.4MB の動画ファイルに圧縮処理をした動画 ファイルを転送する

4.2　実験環境

擬似的に IoT サービスを構成してそれらを Kubernetes で制御する環境を構築した．実験を行うための動画を送信する通信プログラムを作り Kubernetes で制御して実験を行った．クラウドサーバにデスクトップ PC，エッジサーバにデスクトップ PC，ノート型 PC，Raspberrypi，IoT デバイスに Raspberrypi を用いて実験環境を構築した．Kubernetes のバージョンは全て v1.16.1 で Kubernetes のネットワークは flannel:v0.11.0 である．

4.3　ファイル総転送時間

各ノードでのファイルの圧縮処理と転送時間をまとめたものを表 1，表 2 に示す．

表 1　550.4MB の動画ファイル

圧縮場所エッジ	RaspberryPi	ノート型 PC	デスクトップ型 PC
クラウドサーバ	4514.3 秒	188.0 秒	188.4 秒
エッジサーバ	1428.9 秒	269.4 秒	75.5 秒
IoT デバイス	1220.4 秒	906.6 秒	902.0 秒

表 2 15.4MB の動画ファイル

圧縮場所エッジ	RaspberryPi	ノート型 PC	デスクトップ型 PC
クラウドサーバ	104.2 秒	12.9 秒	15.4 秒
エッジサーバ	170.6 秒	24.9 秒	10.8 秒
IoT デバイス	121.4 秒	103.3 秒	104.8 秒

エッジサーバにクラウドサーバと同程度の性能のコンピュータを用いた際にはエッジサーバで処理を行った方が全体の処理時間が速い．しかし，エッジサーバの性能がクラウドサーバに比べて大きく劣る場合にはクラウドサーバで処理を行った方が全体の処理時間が速い．

4.4 最適なコンテナ配置の方針

処理をするデータが動画のようなファイルサイズが大きい場合には，エッジサーバがクラウドサーバと同程度の性能でないと処理時間が長くなってしまう．したがって我々は，ファイルサイズが大きくエッジサーバとクラウドサーバの性能が同程度の場合はエッジサーバで処理を行い，大きく劣る場合はクラウドサーバで処理を行うようコンテナ配置を行えば最適であると考えた. しかし，先行研究から 2 桁の整数のような小さいデータを多数送信する場合にはレイテンシの少ないエッジサーバに送信して処理をした方が速いことが分かった．これらの結果により,IoT サービスの処理をエッジサーバかクラウドサーバのどちらで行うのかをエッジサーバの性能に応じて変化させるコンテナオーケストレーションシステムを実装しなければならない．

5 最適なコンテナ配置を行う IoT サービスの実装

第 4 章にて得た，実験結果から考察した最適なコンテナ配置の条件をふまえ，様々な条件に対して動的に判断を行い，最適な場所にコンテナ配置をする IoT サービスの実装を行う．

5.1 実装内容

Kubernetes の直接操作を行うことができるアプリケーションを作成することで実装を行った．作成したアプリケーション内にて，IoT コンテナから edge コンテナ間のレイテンシ，動画ファイルサイズ，エッジサーバの性能の判断を行っている．想定した IoT サービスとしては，IoT デバイスとエッジサーバ，クラウドサーバを持つ監視システムで，IoT デバイスは監視カメラとし，撮影した動画をエッジサーバに送り，エッジサーバからクラウドサーバへ送るものとする．また，撮影した動画は三層構造内のどこかで動画の圧縮処理を行う．圧縮処理を行うノードを選択するポリシーは，ファイルサイズが 99MB 以上の際，エッジサーバがデスクトップ PC の時はエッジノードにて圧縮を，ノート型 PC の時はクラウドノードにて処理を，RaspberryPi3 の時は IoT デバイスノードにて圧縮処理を行う．ファイルサイズが 99MB 以下の際は，エッジサーバがデスクトップ PC の時はエッジノードにて圧縮を，ノート型 PC の時はクラウドノードにて処理を，RaspberryPi3 の時はクラウドノードにて圧縮処理を行う．

プログラムは C 言語を用いて作成した．system 関数を用い Kubernetes のシェルコマンドを実行することで，コンテナを最適なノードへ配置している．実現した IoT サービスに，レイテンシを動的に判断し，必要な際に他ノードへエッジサーバをマイグレーションさせる機能がある．これは展開した各ノードのコンテナに，動画を送受信する機能とは別に 10 秒に一度 IoT コンテナから edge コンテナ間のレイテンシを計測しそのレイテンシを master ノード上へ送信する機能を追加させることで実現した．この機能には，C 言語の fork 関数を用いた．プログラムはデータ転送先や，圧縮処理を行うノードを切り替えるため，それぞれデータ転送先や圧縮を行うか決めたプログラムを，各ノード分用意した．これらのプログラムを用いて，ノードと条件に応じたプログラムを，それぞれ起動している．

5.2 実験

本稿で実装したプログラムの実行時間と，動画を転送するのみのプログラムの実行時間をファイルサイズが大きい動画と小さい動画に分けそれぞれ 5 回計測する．また，エッジサーバのノードがデスクトップ PC の時と，ノート PC の時をそれぞれ計測する．

測定を行うにあたり，転送する動画のファイルサイズが大きいものとして，ファイルサイズが 221.7MB で再生時間が 4 分 46 分になるものを用意した．転送する動画ファイルサイズが小さいものとしては，221.7MB の動画のフレームレート，ビットレート，サイズをそれぞれ圧縮した，16.1MB の動画を用意した．また，本測定も第 4 章と同様に，tc コマンドを用い下記の制限をかけて測定を行った．

- エッジサーバ…帯域幅は 50Mbps，レイテンシ 5ms
- クラウド…帯域幅 50Mbps，レイテンシ 50ms

5.3 測定

それぞれ用意した，プログラムと動画ファイルを用いて下記の測定を行った．

エッジサーバ:デスクトップ PC

- 測定 1…ファイルサイズ 221.7MB の動画ファイルをプログラム 1 を用いて転送する
- 測定 2…ファイルサイズ 16.1MB の動画ファイルをプログラム 1 を用いて転送する
- 測定 3…ファイルサイズ 221.7MB の動画ファイルをプログラム 2 を用いて転送する
- 測定 4…ファイルサイズ 16.1MB の動画ファイルをプログラム 2 を用いて転送する

エッジサーバ:ノート PC

- 測定 5…ファイルサイズ 221.7MB の動画ファイルをプログラム 1 を用いて転送する
- 測定 6…ファイルサイズ 16.1MB の動画ファイルをプログラム 1 を用いて転送する

- 測定 7…ファイルサイズ 221.7MB の動画ファイルをプログラム 2 を用いて転送する
- 測定 8…ファイルサイズ 16.1MB の動画ファイルをプログラム 2 を用いて転送する

ファイルサイズが 221.7MB の動画ファイルを，圧縮した後のファイルサイズは 16.1MB で，ファイルサイズが 16.1MB の動画ファイルを圧縮した後のファイルサイズは，3.8MB になる．

5.4 測定結果

測定 1 から測定 8 の結果を表 3 と表 4 に示す．

デスクトップ型 PC

表 3 測定 1〜4

	プログラム 1	プログラム 2
221.7MB	56.3 秒	60.1 秒
16.1MB	12.1 秒	10.2 秒

エッジがデスクトップ PC の場合は，どちらのプログラムでも，エッジにて圧縮処理を行うため，測定結果としてあまり大きな変化は見られなかった．

ノート型 PC

表 4 測定 5〜8

	プログラム 1	プログラム 2
221.7MB	61.6 秒	73.8 秒
16.1MB	12.9 秒	21.8 秒

エッジがノート PC の場合は，実装したプログラムでは圧縮処理をより効率の良いクラウドにて行うため，約 10 秒近く時間を短縮することができた．

6 システムに求められる要件に関する考察

本研究では，実装したプログラムを用いることにより，転送時間を短縮することができた．しかし，今回は想定した IoT サービスとして，三層構造の監視システムで，IoT デバイスは監視カメラとし，また，撮影した動画は三層構造内のどこかで動画の圧縮処理を行うと仮定を行い，ポリシーの条件を考察し実装を行った．そのため，本研究での考察したポリシーは，一般の IoT アプリケーションに対するポリシーとしては，妥当ではない可能性があると考える．そのため，Kubernetes などのコンテナオーケストレーションシステムに対して，様々な IoT アプリケーションではどのような影響があるのか調べる必要がある．そして，それぞれのアプリケーションの性質を考慮させ，その性質を考慮したポリシーをどのように描いていくかといった問題を解決しなければならないと考える．また，エッジを用いた IoT サービスにおいてコンテナオーケストレーションシステムには，下記のような要件を指摘をしている研究もある．

1. フォグのクラスタへの参加と離脱
2. 特定のノードでのアプリケーションのスケジューリング
3. フォグノードからコンテナへのデバイスマッピング

上記の要件を満たす既存のコンテナオーケストレーションシステムは存在しないため，要件を満たすようなコンテナオーケストレーションシステムを開発しなければならない．そのためには，アプリケーションのデータ容量や実行内容などの性質をコンテナオーケストレーションシステムに理解させなければならない．今後，実用的なコンテナオーケストレーションシステムを実装する際には，このようなことを今後考慮すべきであると考えられる．

7 おわりに

本研究では，IoT サービスのエッジサーバに用いるコンピュータを複数用意し，エッジサーバの性能を変化させた際の処理時間の測定をそれぞれ行うことで，IoT サービスを構成するコンテナを最適なサーバで実行するための条件について検討を行った．また，その結果から最適なコンテナ配置の条件をふまえ，様々な条件に応じて動的に判断を行い最適な場所にコンテナ配置をする IoT サービスを実現することができた．

今回実装したプログラムにて，エッジサーバの性能やファイルサイズ，レイテンシについて判断を行い，コンテナを展開することができた．今回作成したプログラムは，C 言語の system 関数を用い，Kubernetes のシェルコマンドを実行することで，最適なノードへコンテナを展開させた．今後の課題としては，Kubernetes へ組み込みを行い，どの IoT サービスにおいてもコンテナをマイグレーション及び最適なノードへコンテナを展開できるようにする必要がある．

参考文献

[1] Dupont, Corentin and Giaffreda, Raffaele and Capra, Luca: “Edge computing in IoT context: Horizontal and vertical Linux container migration”, http://ieeexplore.ieee.org/document/8016218, (2020/10).

[2] 深見 宗世，早川 剛生，“エッジコンピューティングにおけるコンテナ配置の最適化，” 2018 年度南山大学理工学部卒業論文，2019.

[3] Hoque, Saiful and Brito, Mathias Santos De and Willner, Alexander and Keil, Oliver and Magedanz, Thomas, “ Towards Container Orchestration in Fog Computing Infrastructures, ” in it Proceedings of 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC), pp.294299, 2017 .