エッジクラウド環境におけるネットワーク特性を考慮したコンテナスケジューラの検討

2020SE094 内田諭志
指導教員：宮澤元

1　はじめに

エッジデバイスから生成される大量のデータを低遅延かつ効率的に処理するため，エッジクラウドが注目されている．エッジクラウドは，クラウドコンピューティングの仮想化技術をエッジコンピューティングに応用した技術である．エッジクラウド環境ではデータセンタのサーバに加え，ネットワークのエッジに設置されたサーバやデバイスを利用してアプリケーションを実行できる．

しかし，エッジクラウド環境におけるコンテナスケジューラには考慮すべき問題点がある．一般に，エッジクラウド環境に接続された計算ノードは，クラウド環境の計算ノードと比べて計算リソースの変動が大きい．また，ネットワークエッジの計算ノードは比較的狭帯域で高遅延のネットワークを介してクラウドノードに接続されるという特徴を持つ．従来のコンテナスケジューラは，こういったクラスタ内でのネットワーク特性の違いを考慮していない．これはサービスが複数のマイクロサービスから構成され，各マイクロサービスがそれぞれ異なる計算ノードに配置されうる場合には特に問題となる．したがって，これらの点を考慮したスケジューラを開発する必要がある．

本稿では，サービス間の通信量と計算ノード間の通信遅延を考慮したスケジューリング手法について検討する．研究課題は，コンテナをノードに配置する際のスコアリングアルゴリズムの評価と通信量や通信遅延のメトリクスの取得方法の検討と実装の 2 点である．

2　関連研究

エッジクラウド環境におけるコンテナのスケジューリングについて様々な研究がされている．Marchese らは通信時点でのクラスターネットワークの状態だけでなく，マイクロサービス間の通信の相互作用も考慮するネットワーク対応スケジューラープラグインを実装することで，デフォルトの kube-scheduler を拡張した [1]．これに対し Miyazawa はクライアントと計算ノード間の通信遅延を考慮したコンテナスケジューリングシステムを提案している [2]．また，Murakami らは地理的に近いサーバに対して優先的にリクエストを送るように制御することでリクエストのスループットを改善する手法を提案している [3]．一方，エッジクラウド環境における利用者の状況や，クラスタ内および利用者とクラスタ間のネットワーク特性の違いを総合的に考慮するような研究はあまり行われていない．

3　ネットワーク特性を考慮したコンテナスケジューラ

本節では，提案するコンテナスケジューラのアーキテクチャと，コンテナの配置先を決定するためのスコアリングアルゴリズムについて述べる．

3.1　本スケジューリングシステムのアーキテクチャ

本システムは各ノード間の通信遅延とサービス間の通信量を定期的に計測し，それらの値を元にコンテナのスケジューリングを行う．本システムのアーキテクチャを**図 1** に示す．このアーキテクチャにおいて，すべての要素は Kubernetes クラスター内で実行される．カスタムスケジューラは提案するネットワーク対応スコアリングプラグインを実装することでデフォルトの Kube-scheduler を拡張する．カスタムスケジューラとデスケジューラはどちらも，Prometheus から通信量と通信遅延のメトリクスを取得する．Envoy プロキシは，Istio のコントロールプレーンによって各アプリケーションポッドに導入され，Prometheus へメトリクスを送る．

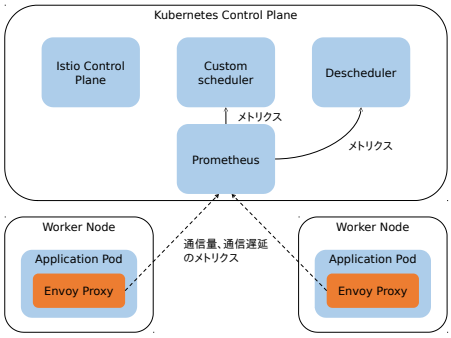

図 1　提案スケジューリングシステムのアーキテクチャ

3.2　スコアリングアルゴリズム

カスタムスケジューラに組み込むスコアリングアルゴリズムについて述べる．配置対象コンテナが，マイクロサービスとして共にサービスを構成する他のコンテナと通信している状況を想定する．マイクロサービス間の通信量が多いほど通信に時間がかかると考えられるので，通信によるサービス処理時間への影響を最小化するために，コンテナ間の通信量が大きいほど低遅延で通信できるようにしたい．そこで，コンテナ間通信量とノード間通信遅延の積を通信相手ごとに足し上げた値をメトリックとして利用する．配置対象コンテナを，配置先候補の各ノードに配置し

た場合に他のノードとの間で発生する通信量を，あらかじめ取得したコンテナ間の通信量に基づき，それぞれ計算する．この通信量とノード間通信遅延との積を，各候補ノードごとに合計したものを各候補ノードのスコアとする．すべてのノードについてスコアを計算し，最もスコアの値が低いノードを Pod の配置先として決定する．

4 実装

Kubernetes クラスタ内の各ノード同士の通信遅延を計測するシステムを Go 言語で実装した．他ノードとの通信遅延を ping を用いて計測して Prometheus 形式でエクスポートするプログラムを Pod として各ノードで動作させる．この通信遅延値と，Kubernetes に導入した Istio が測定する Pod 間通信量を，Promtheus API を用いて取得するプログラムも作成した．

5 実験

3.2 節のスコアリングアルゴリズムの効果を確認するための実験を行った．ノード間の通信遅延をさまざまに設定し，それぞれの状況で Pod の配置パターンを変化させ，サンプルアプリケーションのサービス応答時間を測定する．その後，いくつかの Pod について，提案したスコアリングアルゴリズムに基づいて再配置を仮想的に行う．再配置前後でサービス応答時間の変化を実験結果から確認することでスコアリングアルゴリズムが適切に働くかどうかを確認する．

マスタとして sv1，ワーカとして sv2，sv3 の計 3 台の PC が 1000Base-T LAN で接続された Kubernetes クラスタを構成し，Istio のサンプルアプリケーションである bookinfo を動作させた．bookinfo は prodcutpage, ratings など 6 個のマイクロサービスからなる．事前に bookinfo を構成するサービス間の通信量を測定したところ，productpage とは他のマイクロサービスのうち 4 つが，ratings とは 2 つが通信を行っていることがわかった．

実験では，ノード間の通信遅延が異なる 2 通りのノード構成を tc コマンドを用いてエミュレーションし，それぞれの構成に対して，bookinfo のマイクロサービス配置を 4 通りに変化させた．それぞれの場合で，Apatch Bench コマンドを用いてクラスタ外から bookinfo に対して 1000 リクエスト (100 接続 ×10) を送信した際のリクエストごとの平均応答時間を計測した．実験結果を表 1 に示す．クラウドのノード構成では計算ノードが相互に遅延なく通信し，エッジでは計算ノード相互の通信に片道 30 msec. ± 5 msec. の遅延を設定している．

この結果をもとに，3.2 節のスコアリングアルゴリズムを用いていずれかの Pod を再配置した際に，測定したどの Pod 配置の結果に近くなるか調べることで，スコアリングアルゴリズムが適切に働くかどうかがわかる．検討するケースとして以下の 2 つを考える。

- ratings が sv3, 他のサービスが sv2 に配置されており、ratings を再配置する
- productpage が sv2，他のサービスが sv3 に配置されており、productpage を再配置する

表 1 各マイクロサービス配置におけるリクエストごとの平均サービス応答時間 (msec.)

マイクロサービス配置	クラウド	エッジ
all on 02	9.504	7.607
all on 03	9.637	9.483
rating on 03	8.415	9.798
productpage on 02	6.715	6.914

3.2 節のアルゴリズムを適用すると，前者のケースはエッジのノード構成で ratings を sv2 に再配置することになる．実験結果からはこの再配置でサービス応答時間が短縮されるので、スコアリングアルゴリズムが適切に働くと言える．一方，後者のケースはクラウドのノード構成で productpage を sv3 に再配置することになる。実験結果からはこの再配置でサービス応答時間が増加してしまうことになる．以上より，このスコアリングアルゴリズムは全てのケースで有効であるとは限らないことがわかる．

6 終わりに

本稿では，エッジクラウド環境においてクラスタ内の通信遅延を考慮したコンテナスケジューリングシステムについて述べた．ノード間通信遅延やサービス間通信量のメトリクスの取得方法を検討し，通信遅延取得プログラムを実装した．また，実験を行い，Pod 配置時のノードのスコアリングアルゴリズムの効果を確認したところ，検討したスコアリングアルゴリズムが全ての場合に有効とは限らないことがわかった．今後は，クライアントとクラスタ間の通信特性も反映してスコアリングアルゴリズムを改善し，スケジューラの実装を行う．

参考文献

[1] A. Marchese and O. Tomarchio, “Network-Aware Container Placement in Cloud-Edge Kubernetes Clusters,‘ ’ in *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, May 2022, pp. 859865. doi: 10.1109/CCGrid54584.2022.00102.

[2] H.Miyazawa, “A Latency-aware Container Scheduling in Edge Cloud Computing Environment,‘ ’ in *Proceedings of the 24th International Conference on Internet Computing and Internet of Things (ICOMP ’ 23)*, 2023.

[3] K.Murakami, “エッジコンピューティングにおける拠点間の通信遅延を考慮したリクエスト分散制御‘ ’，奈良先端科学技術大学院，2022.