コンテナ型仮想化におけるスケーリング方式の ヘテロジニアス IoT システムへの応用

1205080　小川 友暉　【 コンピュータ構成学研究室 】

Autoscaling of Container-Based Virtualization towards Heterogeneous IoT Systems

1205080　Tomoki OGAWA　【 Advanced Computer Engineering Lab. 】

1　はじめに

近年, IoT の普及に伴いフォグコンピューティングに注目が集まっている. 従来のクラウドコンピューティングによるサーバでの集中処理方式は, IoT システムで発生する大量のデータやネットワークの輻輳によって現実的では無く, またリアルタイム処理などの厳しいレイテンシ要件を満たすことができない. このため, エッジ側での処理の分担が求められている. しかしながら, エッジデバイスはサーバに比べて性能が極端に劣り, リソース制約も厳しい. また設置場所などの温度変化などによって性能にばらつきが発生し処理効率が変動する. そのため, ヘテロジニアスな環境でも, エッジ側で処理効率を高く維持する機構が必要である [1].

本研究ではこのような機構としてコンテナ型仮想化を用いた Kubernetes[2] でのスケーリング方式を提案し, ヘテロジニアス IoT システム全体の可用性を向上させることを目標とした.

2　コンテナ型仮想化と Kubernetes

Docker などのコンテナ型は VM 型と比べてハードウェアリソースの消費が少なく, またコンテナがホスト OS のプロセスとして扱われるため, 起動・停止も高速であるという特徴がある. そのため, IoT などのリソース制約の厳しい環境での有効性が高い. Kubernetes はクラスタ環境においてコンテナのオーケストレーションを行う OSS である. Docker ではコンテナの作成やイメージの作成・管理・実行などの機能が提供されているが, ネットワークルーティング, 複数コンテナの連携, 複数ホストで横断的に管理する機能を備えていない. そのため, Kubernetes を用いることで複数ホストにまたがる環境を構築することが容易になる.

Kubernetes では, 図 1 のようにマスタとノードに分かれ, マスタが API などのサービスを提供し, ノードを管理する. ここで管理されるノードは基本的にホスト OS に対応している. コンテナを 1 つ以上でグルーピングした Pod という単位で扱うことによって, 連携しているコンテナ内で IP アドレスやストレージの共有などを行なう [2].

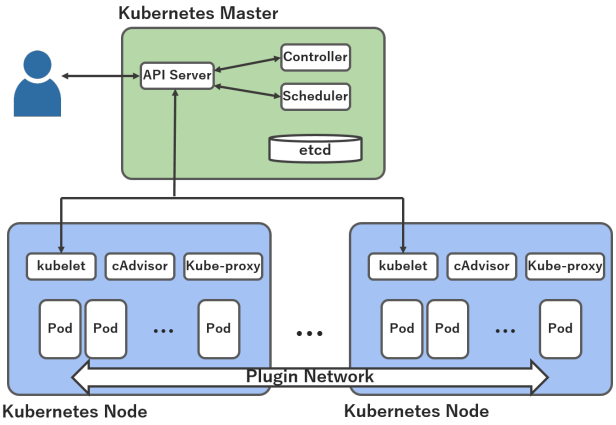

図 1　kubernetes のアーキテクチャ

3　ヘテロジニアス性による課題

IoT では家や都市, ビルや工場などに様々な場所にデバイスが設置されヘテロジニアス性が高い. 結果として仕様上のハードウェア性能のみならず, 温度の変化によっても CPU の性能に変動が起こる. また, 他にもアーキテクチャや OS, 互換性などのヘテロジニアス性がある [3].

Kubernetes はクラウドサーバなどのホモジニアスな環境における使用を想定しているため, IoT のようなヘテロジニアスな環境向けに最適化されていない. また, 各ホスト間でのスケーリングは水平スケーリングのみであり Pod の垂直スケーリングの機能は備わっていない. 水平スケーリングはクラウドなどのホモジニアスな環境では実行ホストを変更しても処理能力に変化が生じないため有効であるが, IoT のようなヘテロジニアス性の高い環境では同じ Pod を生成しても性能の低いマシンでは有効に働かない. そのため, 温度変化といったノードの性能変化が生じると, 必要なパフォーマンスを発揮することができない. そのため, このヘテロジニアス性を吸収し, アプリケーションを効率的に実行できるシステムが必要である.

4　動的なスケーリング方式

Pod の負荷が大きくなり，ノードへの負荷が大きくなった場合，マスタによって Pod をどのノードに割り振るかが決定される．しかしながら，Kubernetes のスケジューラはラウンドロビン形式であり，ノードの絶対性能が考慮された負荷分散が行なわれない．そこで，モニタリング機能によって各ノードの CPU 負荷を収集し，より要求にあった性能を持つノードへ Pod を再配置できるように再生成を行なう仕組みを導入する．

Kubernetes ではアプリケーションをデプロイする際に Manifest ファイルによって Pod を生成する．この Manifest ファイルに Pod が配置される際に必要なリソース量を Request として示し，限界値として Limit を記述することで，Pod が配置される際に必要なリソースを保証する．リソース量を設定することによって，ノードの性能に対する Pod の負荷が過剰にならないように設定することができる．しかしながら，Pod が配置されているにも関わらず，温度変化などによる CPU の性能変化を受けて必要な性能を実質的に満たさなくなった場合，リミット値が固定の場合，負荷が大きくなったり温度上昇によってクロック周波数が低下したりなどの問題が生じる．そこで，Manifest ファイルの CPU のリクエストとリミット値を向上させることによって，使用できる CPU 使用量を向上させる．図 2 にシステムの概要を示す．

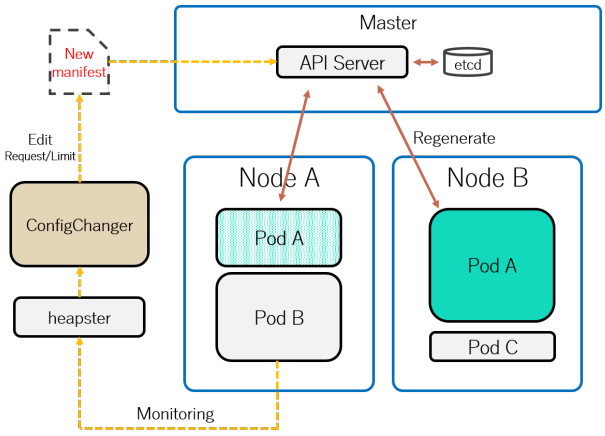

図 2　システムの概要図

ConfigChanger は一定間隔で各ノードから負荷を収集し，その結果を元に，Request と Limit を変更し Pod の再生成を行なう．生成後，Kubernetes が Pod のリソース量にフィットするノードに配備を行なうため，要求性能にあったノードに配置されることが保証される．

しかしながら，配置 Pod の CPU 使用量がノード単体の CPU 量を超えたり，全体の CPU 使用量を超えたりしてしまうと，Pod が要求するリソースを満たせず生成できなくなるため，負荷が増大した Pod のリミット値の向上と同時に，CPU 使用量の低いノードで動いている Pod の manifest のリミット値とリクエスト値を低下させる．そうすることで，タスクが不飽和状態にあるノードでの CPU 使用量を下げることによって，過負荷状態を回避する．

5　評価

提案した動的構成法について評価行う．実験環境では，Raspberry pi 2 を 5 台を用いる．表 1 に主要な性能と環境を示す．

表 1　使用機器の諸性能

CPU	ARM Cortex-A53
メモリ	1 GByte
ネットワーク I/F	CAT 5e 1000Mbps
電源	DC 5V
OS	HypriotOS

評価では，各 Raspberry pi の負荷を変化させることによってヘテロジニアスな環境を模擬させる．CPU 使用量が高い状態になると Pod のリソースの Request と Limit が書き換わり Pod 再生成後，マスタによって Pod の再配置先として選ばれるノードが，要件を満たしたノードに再配置されて可用性が向上することを評価する．

6　おわりに

本研究では，ヘテロジニアス IoT システムにおける可用性および効率性の向上を行うスケーリング方式について示した．各 Pod のリソースの Request と Limit を負荷に応じて動的に変化させることで，リソース要求にあったノードに，再生成した Pod を配置可能にした．また結果として単一ノードに負荷が集中することを避けることが可能となった．今後の検討事項として，メモリなどの CPU 以外のリソースにおいても動的に変化させ，各 Pod が使用するハードウェアにあったノードへの配置などを行う必要がある．

参考文献

[1] Bruzual Balzan, Daniel, “Distributed Computing Framework Based on Software Containers for Heterogeneous Embedded Devices,” Service Design and Engineering, Aalto University, October 2017.

[2] Google. Inc, “Kuberntes”, https://kubernetes.io, December 2017.

[3] Asad Javed, “Container-based IoT Sensor Node on Raspberry Pi and the Kubernetes Cluster Framework,” ICT Innovation, Aalt University, June 2016.

[4] 叶 如絵, 田胡 和哉, “コンテナ型仮想化環境向き負荷予測システム「Tetris」の開発,” 第 78 回全国大会講演論文集, vol.1, pp. 11–12. May 2016.