負荷テストを用いた Kubernetes Pod におけるメモリの実測値との差を縮小するための Requests の再設定

森井 佑誠[1]　串田 高幸[1]

概要：Kubernetes のマニフェストファイルにおいて, メモリの Requests は Kubernetes ユーザによって経験的に決められている. 実際のメモリ使用量がメモリの Requests より上回りかつ合計が Node のメモリ容量を超える場合, Pod の動作に必要なメモリが確保できない. また, 実際のメモリ使用量がメモリの Requests より下回りかつ Pod の Requests の合計が Node のメモリ容量を超える場合, Node にデプロイできるメモリ容量があるにも関わらず, Requests の合計が Node のメモリ容量を超える Pod をデプロイできない. 本研究では, デプロイしたアプリケーションに負荷を与え, 得られたメモリ使用量から Requests を再設定するかを決定し, Requests と実測値との差を小さくする. 本研究の評価として 1 秒に 1 リクエストを 20 分間実行し, そのときのメモリ使用量と再設定する値の差を評価とする. その結果, 最高で 11.7%, 最低で 0.0%, 平均で 4.9% の差で値の算出ができた.

1. はじめに

背景

オープンソースのコンテナオーケストレーションシステムである Kubernetes では, YAML 形式のマニフェストファイルを使い設定を記述する [1][2]. マニフェストファイルにおいて, Pod の CPU およびメモリの Requests (要求), Limits (制限) を決めることができる. Requests を指定すると, Requests の値からどの Node に Pod を配置するか決定される. 全ての Pod において Requests の合計は個々の Node の容量以下である必要がある. Limits を指定すると, Pod が Limits の値を超えて CPU やメモリを使用することができない. 特に, メモリについて Limits の値が超える場合, OOM (Out Of Memory) Killer によって Pod のプロセスは異常終了する [3]. 実際の運用において, メモリの Requests と Limits の値は経験的に決められている.

課題

マニフェストファイルで設定されているメモリの Requests と実際の運用における実際のメモリの使用量に差が生じる場合, 以下の 2 つの問題が発生する.

課題 1: メモリ使用量が Requests より上回りかつ合計が Node のメモリ容量を超える場合

Pod の合計メモリ使用量が Node のメモリ容量を超え, Pod の動作に必要なメモリが確保できない問題が発生する. これは, Pod のメモリ使用量が Requests より多く使用されてしまうにも関わらず Node にデプロイされることが原因で発生する. この問題により, Pod が必要とするメモリが使用できず, Pod の動作に影響を与える. 動作の影響の例として, Pod の処理が遅くなるというものがあげられる. この場合の Pod 構成の例を以下の図 1 に示す.

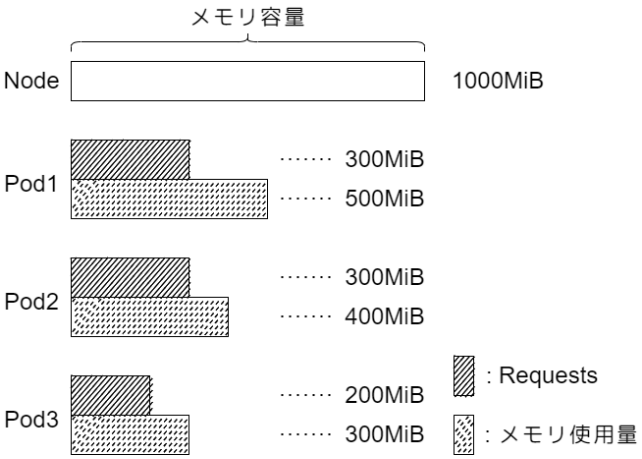

図 1 メモリ使用量が Requests より上回りかつ合計が Node のメモリ容量を超える場合

図 1 では, 1 つの Node に対し 3 つの Pod が存在する. 1000MiB の Node, Requests が 300MiB メモリ使用量が 500MiB の Pod1, Requests が 300MiB メモリ使用量が 400MiB の Pod2, Requests が 200MiB メモリ使用量

[1] 東京工科大学コンピュータサイエンス学部
〒 192–0982 東京都八王子市片倉町 1404-1

が 300MiB の Pod3 である．このとき，これらの Pod の Requests の合計値は 300 + 300 + 200 = 800[MiB] であり，Node のメモリ容量未満であるが，メモリ使用量の合計は 500 + 400 + 300 = 1200[MiB] であり，Node のメモリ容量を超えている．そのため，これらの Pod はデプロイ可能であるが，Node のメモリ容量を超えているため，Pod の動作に必要なメモリが確保できない．

課題 2: メモリ使用量が Requests より下回りかつ Pod の Requests の合計が Node のメモリ容量を超える場合

Requests の合計が Node のメモリ容量を超える場合，Node にデプロイできるメモリ空き容量があるにも関わらず，Requests の合計が Node のメモリ容量を超える Pod をデプロイできない問題が発生する．Kubernetes のスケジューラは Node に Pod をデプロイするとき，デプロイされている Pod と新たにデプロイされる Pod の Requests の合計値が Node の容量を超えていないことを確認し，Node にデプロイするか判断する．メモリ使用量が Requests より下回る Pod が多いと，実際には Node にデプロイできるのにも関わらず，Requests の合計が Node のメモリ容量を超えた Pod のデプロイができない．この場合の Pod 構成例を以下の図 2 に示す．

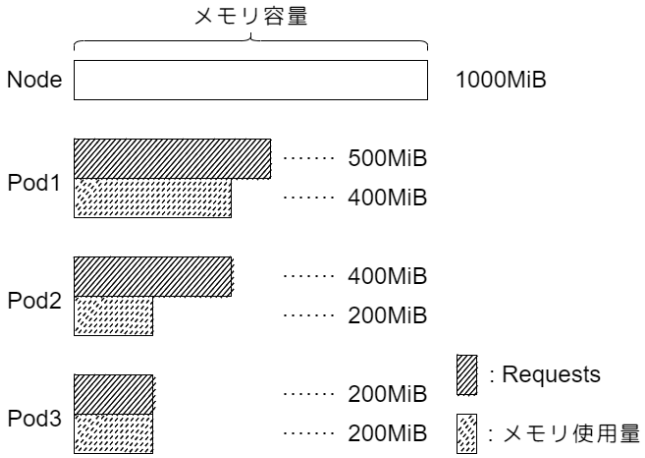

図 2 メモリ使用量が Requests より下回りかつ Pod の Requests の合計が Node のメモリ容量を超える場合

図 2 では，図 1 と同様に 1 つの Node に対し 3 つの Pod が存在する．1000MiB の Node，Requests が 500MiB メモリ使用量が 400MiB の Pod1，Requests が 400MiB メモリ使用量が 200MiB の Pod2，Requests が 200MiB メモリ使用量が 200MiB の Pod3 である．これらの Pod の Requests の合計値は 500 + 400 + 200 = 1100[MiB] である．このとき，Pod1 および Pod2 はデプロイされるが，Pod3 をデプロイする場合，Requests の合計が Node のメモリ容量を超えているため，Pod3 はデプロイできない．しかし，メモリ使用量の合計は 400 + 200 + 200 = 800[MiB] であり，Node のメモリ容量未満である．そのため，実際には Pod3 はデプロイ可能である．

基礎実験

以下の VM 環境で基礎実験を行い，課題の検証を行う．基礎実験では Kubernetes エンジンとして k3s を使用する．

- Master Node
  - CPU: 1vCPU
  - メモリ: 2048MB
  - OS: Ubuntu 20.04
- Worker Node
  - CPU: 1vCPU
  - メモリ: 1024MB
  - OS: Ubuntu 20.04

実験に使用したマニフェストのテンプレートをソースコード 1 に示す．ソースコード 1 を用いて，メモリ使用量が Requests より上回りかつ合計が Node のメモリ容量を超える場合および下回る場合の実験を行った．実験に使用するマニフェストファイルでは，stress コマンドを使用できるコンテナイメージを用いて，実際のメモリ使用量を引数で指定し，メモリを確保する．実験において確認のために使用したコマンドは kubectl get pods -o wide および，kubectl top pods である．kubectl get pods -o wide では，Pod 名，ステータス，デプロイされているノードを確認できる．kubectl top pods ではデプロイされている Pod が CPU およびメモリをどのくらい使用しているかを確認できる．これらを用いて，Pod がデプロイできているかやメモリを確保できているかを確認した．

ソースコード 1 template.yaml

```
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: <Pod 名>
5  spec:
6    containers:
7    - name: <コンテナ名>
8      image: polinux/stress
9      resources:
10       requests:
11         memory: <メモリの Requests>
12     command: ["stress"]
13     args: ["--vm", "1", "--vm-bytes", "<実際のメモリ使用量>", "--vm-hang", "1"]
14   nodeName: <Worker Node>
```

メモリ使用量が Requests より上回りかつ合計が Node のメモリ容量を超える場合

図 1 の例を使い実験を行った結果を図 3 に示す．図 3 では，Pod1 から Pod3 までの全ての Pod がデプロイできていることが確認できる．しかし，Pod1 および Pod2 は要求したメモリ使用量 500MiB，400MiB に対し，それぞれ 157MiB，240MiB しか確保されておらず，メモリが確保できていない．

```
root@ym-master:~# kubectl get pods -o wide
NAME   READY   STATUS    RESTARTS   AGE
pod3   1/1     Running   0          71s
pod2   1/1     Running   0          71s
pod1   1/1     Running   0          71s
root@ym-master:~# kubectl top pods
NAME   CPU(cores)   MEMORY(bytes)
pod1   40m          157Mi
pod2   39m          240Mi
pod3   39m          227Mi
```

図 3 メモリ使用量が Requests より上回りかつ合計が Node のメモリ容量を超える場合

メモリ使用量が Requests より下回りかつ Pod の Requests の合計が Node のメモリ容量を超える場合

図 2 の例を使い実験を行った結果を図 4 に示す. 図 4 では, Pod1 から Pod3 のうち Pod1 および Pod2 のみデプロイされており, Pod3 は OutOfMemory でデプロイに失敗していることが確認できる. また, Pod1 および Pod2 のメモリ使用量は 401MiB, 200MiB であり, Node に Pod3 をデプロイすることができるメモリ容量がある.

```
root@ym-master:~# kubectl get pods -o wide
NAME   READY   STATUS        RESTARTS   AGE
pod3   0/1     OutOfmemory   0          6d2
pod1   1/1     Running       0          6d2
pod2   1/1     Running       0          6d2
root@ym-master:~# kubectl top pods
NAME   CPU(cores)   MEMORY(bytes)
pod1   80m          401Mi
pod2   45m          200Mi
```

図 4 メモリ使用量が Requests より下回りかつ Pod の Requests の合計が Node のメモリ容量を超える場合

各章の概要

本研究は, 次のように構成される. 第 1 章では, 本研究の背景・課題について述べる. 第 2 章では, 本研究の関連研究について述べる. 第 3 章では, 第 1 章で述べた課題を解決するための提案手法, ユースケース・シナリオについて述べる. 第 4 章では, 提案手法を実現するための実装と実験方法について述べる. 第 5 章では, 提案手法における評価手法と分析手法について述べる. 第 6 章では, 本研究における議論について述べる. 第 7 章では, 本研究のまとめについて述べる.

2. 関連研究

Medel らは, Petri Net ベースのパフォーマンスモデルを通した Kubernetes のパフォーマンスを分析した [4]. この研究のモデルでは, Kubernetes からのデータを使って特徴付けを行うことが可能である. 研究の課題として, コンテナのリソース競合によるオーバヘッドの予測, リソース競合の問題を認識できるように Kubernetes スケジューラを改善すること, 異なるリソース管理ポリシーの挙動を調査するために様々な what-if シナリオを着手することの 3 つの課題をあげている. また, この研究では, パフォーマンスの分析を行っているだけであり, Pod の自動設定は行ってはいない.

Ferreira らは, クラウダプロバイダのマネージド Kubernetes サービスのパフォーマンス評価を行った [5]. この研究では, AWS, Azure, GCP, NeCTAR のインスタンスと EKS, AKS, GKE のマネージド Kubernetes クラスタのパフォーマンス評価を行っている. しかし, この研究では, パフォーマンス評価にとどまっており, メモリの Requests の導出は行われていない.

3. 提案方式

提案方式

本研究の提案では, 課題を解決するために負荷を与えることによってメモリを使用する Service に負荷を与えメモリ使用量を一定の値にする. そのときの値をもとにメモリの Requests の値を決め, Requests の再設定を行う. 本研究における提案の一連の流れを以下の図 5 に示す.

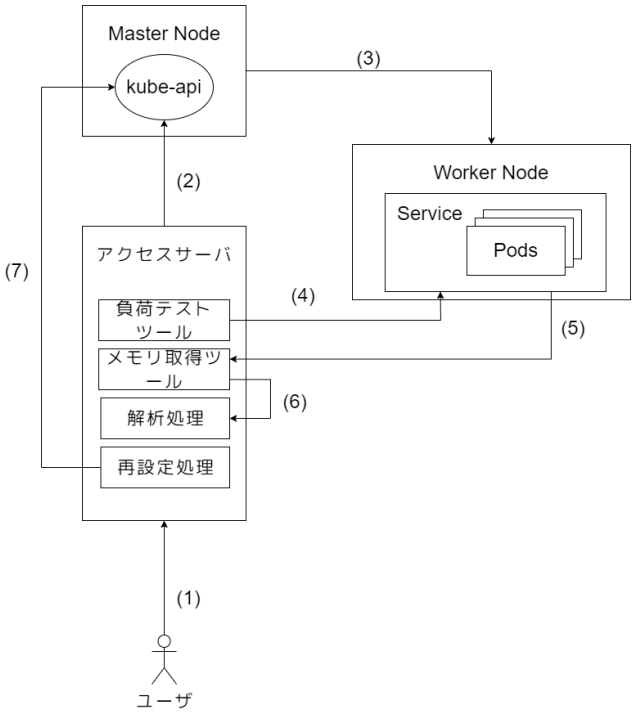

図 5 提案の流れ

(1) ユーザがアクセスサーバに SSH ログイン
(2) アクセスサーバから Master Node 上の kube-api にアクセス
(3) Master Node による Worker Node への Service デプロイ

(4) アクセスサーバ上の負荷テストツール を使用して, Worker Node の Service へ負荷テスト
(5) アクセスサーバからデプロイされた Service の Pods のメモリ使用量をメモリ取得ツールを用いて取得
(6) メモリ取得ツールから解析処理を行う
(7) 解析した結果から再設定処理を行い, 再設定したメモリの Requests で Service デプロイ

図 5 におけるユーザは Kubernetes を管理するユーザである. アクセスサーバは, Master Node 上に存在する kube-api にアクセスを行うサーバである. Master Node は, Kubernetes の Master の役割を果たすサーバであり, Worker Node は Worker の役割を果たすサーバである. 本研究の提案では, デプロイした Service に対し負荷テストを 10 回行う. 取得したメモリ使用量の結果の中央値が Requests の値から 5% の差であるか判断し, 超える場合はその中央値に再設定を行う. Service のデプロイによって作られた Pod を P と表す. 作られた Pod のメモリの Requests を P_{req}, 負荷テストで得られた中央値を R とすると, Pod P に対して以下の式 (1)

$$\left|1 - \frac{R}{P_{req}}\right| \leq 0.05 \tag{1}$$

を満たす場合, P_{req} は再設定を行わず, 以下の式 (2)

$$\left|1 - \frac{R}{P_{req}}\right| > 0.05 \tag{2}$$

を満たす場合, $P_{req} = R$ に再設定を行う. このとき, 再設定を行う場合の条件および式を以下の式 (3) に示す.

$$\begin{cases} 1 - \dfrac{R}{P_{req}} > 0.05 & (R < P_{req}) \\ \dfrac{R}{P_{req}} - 1 > 0.05 & (R \geq P_{req}) \end{cases} \tag{3}$$

また, 条件における P_{req} と R の関係を以下の図 6, 7 に示す.

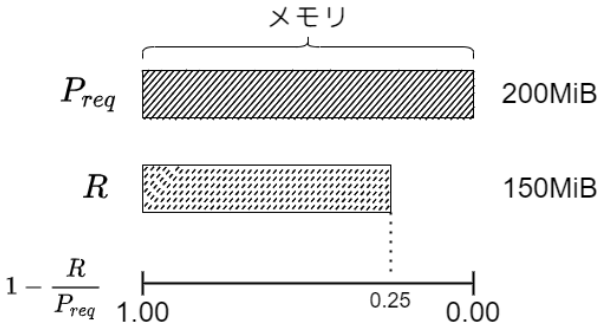

図 6 $R < P_{req}$ における P_{req} と R の関係

図 6 では, $P_{req} = 200$[MiB], $R = 150$[MiB] の場合における関係を示している. $R < P_{req}$ より, 求める式は $1 - \frac{R}{P_{req}}$ であり値は $1 - \frac{150}{200} = 0.25$ となる. このとき, 求めた値は

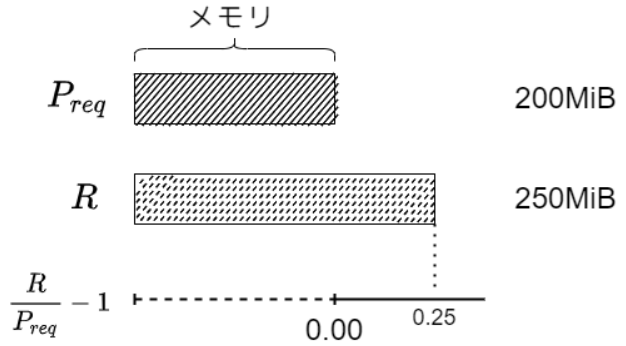

図 7 $R \geq P_{req}$ における P_{req} と R の関係

P_{req} に対して減少した割合を示している.

図 7 では, $P_{req} = 200$[MiB], $R = 250$[MiB] の場合における関係を示している. $R \geq P_{req}$ より, 求める式は $\frac{R}{P_{req}}$ であり値は $\frac{250}{200} - 1 = 0.25$ となる. このとき, 求めた値は P_{req} に対して増加した割合を示している.

ユースケース・シナリオ

ユースケースとして, Kubernetes 上に構築された WordPress を例とする. WordPress ではデータベースとして MySQL を利用する. また, ユースケースのターゲットとして Kubernetes ユーザをターゲットとする. ユースケースの図を以下の図 8 に示す.

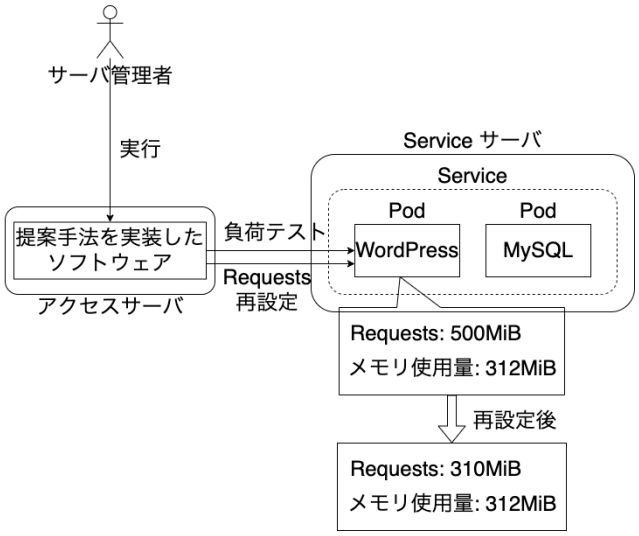

図 8 WordPress のユースケース

図 8 では, あらかじめ WordPress の Pod に 500MiB の Requests が設定されており, 実際の運用におけるメモリ使用量は 312MiB となっている. サーバ管理者が提案手法を実装したソフトウェアを実行することにより, ソフトウェアは WordPress の Pod に対して負荷テストを行う. そのあとに, ソフトウェアは Requests 再設定を行う. 図 8 では, 再設定により Requests の値は 320MiB となり再設定前の Requests とメモリ使用量との差を小さくすることが

できる．このことによってサーバ管理者は今まで手動で設定していたメモリの Requests を自動的に変更することが可能となる．

4. 実装と実験方法

実装

実装の概要を図 9 に示す．実装では，アクセスサーバ上に Service デプロイ，負荷テスト，メモリ取得を行うプログラムを一つのコンポーネントとして実装を行う．Service デプロイでは，Master Node の kube-api にアクセスを行い，Worker Node に Service をデプロイさせる．負荷テストでは，Worker Node にデプロイされた Service に対して負荷を与える．メモリ取得では，Service の Pods のメモリを取得し，提案手法による再設定を行うかを決定する．再設定を行う場合，新たな値で再度 Service デプロイを行う．また，メモリ取得の間隔は Rattihalli らの研究に従い 60 秒に設定する [6]．

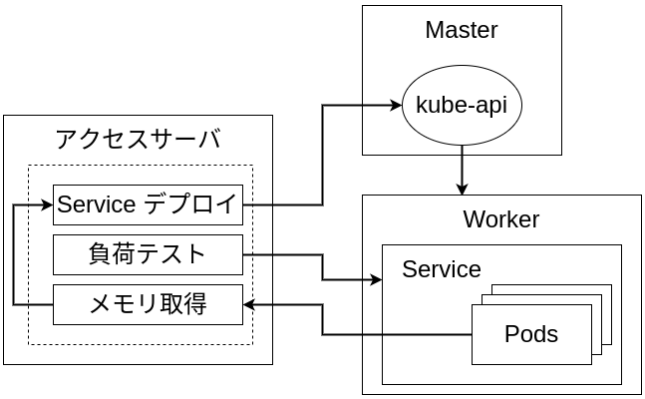

図 9 実装の概要

実験環境

Master Node および Worker Node は基礎実験で使用した VM 上に構築された環境を利用する．また，アクセスサーバも同様に VM 上に構築する．これらは全て各 1 台で構成される．アクセスサーバの構成は以下である:

- CPU: 1vCPU
- メモリ: 1024MB
- OS: Ubuntu 20.04

実装するソフトウェアは Go 言語で実装し，ライブラリとして Kubernetes を利用するための client-go[*1]，負荷テストを行うための vegeta[*2] をライブラリとして利用する．また，Service として WordPress を用いて実験を行う．

5. 評価手法と分析手法

評価手法ではデプロイした Service に対し，1 秒ごとに 1 リクエストの負荷をかける．このとき，再設定した値に対して実際のメモリ使用量との差を評価項目として用いる．

図 10 に 1 秒に 1 リクエストを送った場合のメモリ使用量を示す．図 10 では，20 分間負荷を与えたときのメモリ使用量の遷移を示している．結果より，1 秒に 1 リクエストを送った場合，9 分から 20 分では，80MiB から 120MiB の間でメモリ使用量が遷移している．このとき，負荷テスト終了時点での中央値は 101MiB である．これを Requests の再設定の値とし，図 10 の 20 分におけるメモリ使用量との差を計算する．その結果，最高で 11.7%，最低で 0.0%，平均で 4.9% の差が生じていた．

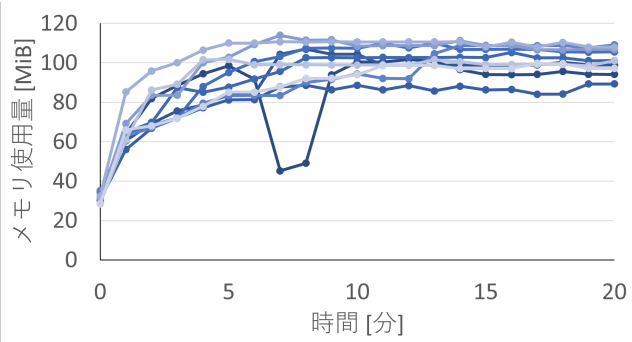

図 10 1 秒に 1 リクエストを送った場合のメモリ使用量

6. 議論

提案手法および評価手法では，一定時間に同じ負荷をかける処理を行っている．実際の運用において，一定時間に一定の回数アクセスがされるとは限らないため，今回の実験では実運用に向いていない．そのため，実際のリクエストに沿った負荷をかけるために，東京工科大学のクラウド・分散研究室[*3]で運用されているサイトのアクセスを用いて，負荷テストシナリオを作成する方法がある．加えて，本実験では負荷テスト中にメモリ使用量が突発的に減少，または増加する時間があった．本研究では，メモリ使用量を同時間で比較し，中央値を用いて値を算出していたため，値の算出に影響をもたらさなかった．しかし，他の時間と比較する場合，突発的に減少，増加したメモリ使用量が値の算出に影響をもたらすため，本実験で使用した中央値といったこれらのメモリ使用量が影響しない値の算出方法が必要となる．

7. おわりに

本研究では，Kubernetes のマニフェストファイルにおいてメモリの Requests と実際のメモリ使用量に差が生じる場合の課題を提示した．メモリ使用量が Requests より上回りかつ合計が Node のメモリ容量を超える場合，Pod の動作に必要なメモリが確保できない問題が発生する．メモ

[*1] https://github.com/kubernetes/client-go
[*2] https://github.com/tsenart/vegeta
[*3] https://ja.tak-cslab.org/

リ使用量が Requests より下回りかつ Pod の Requests の合計が Node のメモリ容量を超える場合, Node にデプロイできるメモリ容量があるにも関わらず, Requests の合計が Node のメモリ容量を超える Pod をデプロイできない問題が発生する. それに対する提案として, 負荷テストを用いたメモリの Requests の再設定を提案した. 負荷テストを 10 回行い, メモリの Requests が得られたメモリ使用量の中央値より 5% の差がある場合, 再設定を行い, 差がない場合は, 再設定を行わない. 評価として 1 秒ごとに 1 リクエストの負荷をかけ, 再設定した値に対して実際のメモリ使用量の差を評価として用いる. その結果, 最高で 11.7%, 最低で 0.0%, 平均で 4.9% の差で値の算出ができた.

参考文献

[1] Bernstein, D.: Containers and Cloud: From LXC to Docker to Kubernetes, *IEEE Cloud Computing*, Vol. 1, No. 3, pp. 81–84 (online), DOI: 10.1109/MCC.2014.51 (2014).

[2] Shah, J. and Dubaria, D.: Building Modern Clouds: Using Docker, Kubernetes & Google Cloud Platform, *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 0184–0189 (online), DOI: 10.1109/CCWC.2019.8666479 (2019).

[3] Larsson, L., Tärneberg, W., Klein, C., Elmroth, E. and Kihl, M.: Impact of etcd deployment on Kubernetes, Istio, and application performance, *Software: Practice and Experience*, Vol. 50, No. 10, pp. 1986–2007 (online), DOI: https://doi.org/10.1002/spe.2885 (2020).

[4] Medel, V., Tolosana-Calasanz, R., Ángel Bañares, J., Arronategui, U. and Rana, O. F.: Characterising resource management performance in Kubernetes, *Computers & Electrical Engineering*, Vol. 68, pp. 286–297 (online), DOI: https://doi.org/10.1016/j.compeleceng.2018.03.041 (2018).

[5] Pereira Ferreira, A. and Sinnott, R.: A Performance Evaluation of Containers Running on Managed Kubernetes Services, *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 199–208 (online), DOI: 10.1109/CloudCom.2019.00038 (2019).

[6] Rattihalli, G., Govindaraju, M., Lu, H. and Tiwari, D.: Exploring Potential for Non-Disruptive Vertical Auto Scaling and Resource Estimation in Kubernetes, *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pp. 33–40 (online), DOI: 10.1109/CLOUD.2019.00018 (2019).