CPU 使用量に基づく最小二乗法を用いたマイクロサービスの処理リクエスト数の上限の推定

坂本 一俊[1]　伊藤 佳城[2]　串田 高幸[1]

概要：マイクロサービスとは，単一のアプリケーションを小さなサービスの集合体としてユーザにサービスを提供するアプローチである．マイクロサービスの一例として，ファッション EC サイトがある．マイクロサービスは，Kubernetes や Docker を用いて実装される．クラスタの CPU のコア数を上限まで使用した際に，リクエストが処理されて応答が返ってくるまでの応答時間が増加する．負荷試験を行い，マイクロサービスの WEB サイトにサービスを提供する HTTP サーバーを公開する機能を持つ frontend を 1 としたときの，サービスごとに CPU の使用コア数の比を求める．ユーザがマイクロサービスにアクセスする際に，frontend を通過するため基準とした．負荷試験の結果から frontend の CPU の使用コア数の値を最小 2 乗法を用いてリクエスト数から近似式を求める．近似式とサービスごとの比をもとにリクエスト数からマイクロサービス全体の CPU の使用コア数を求める式を求める．求められた式にマイクロサービスに割り当て可能な CPU のコア数の上限からリクエスト数の上限を推定する．評価する方法として推定した値で，実際にリクエストを送信した際に，マイクロサービス全体の CPU の使用コア数を計測する実験を行う．実験を行った結果，誤差 1%でリクエスト数の推定を行うことができた．

1. はじめに

背景

日本国内では，アパレル Electronic Commerce(EC) は EC 業界の中で市場規模は大きく経済産業省によると 2021 年に 2 兆 4279 億円にもなる[*1]．EC サイトにおいてコンバージョンレートとカート放棄率という統計データがある．コンバージョンレートとは，全体サイト訪問者のうち購入者の割合である．カート放棄率とは，EC サイトで購入するためにカートを作成したが，購入せず放棄した割合である．2022 年 9 月に 960 のアパレル EC サイトを対象に調査をした Littledata によるとコンバージョンレートは，1.4%であった[*2]．2021 年のアパレル EC サイトを対象に調査をした SaleCycle によると，カート放棄率は 83.98%であった[*3]．上記の 2 つの調査の割合から，全体のサイト訪問者を 100%としたとき 8.75%がカートを作成し，1.4%が購入している．

アパレル EC サイトである ZOZOTOWN では，アプリケーションをサービスごとに分けるマイクロサービスを用いている[*4]．アパレル EC サイトの例として，GoogleCloudPlatform の microservices-demo がある[*5] [1]．このアプリケーションは，11 個のマイクロサービスアプリケーションで構成されており，ユーザが商品を閲覧し，カートに追加して購入できる WEB ベースの EC アプリケーションである．このマイクロサービスは，コンテナ技術を用いている．コンテナ技術を用いることで，異なる種類のハードウェアでアプリケーションをシームレスに展開，移行，および管理することができる [2,3]．コンテナは，コンテナ管理ソフトウェアを用いて管理することができる．

コンテナ管理ソフトウェアのひとつに Kubernetes(K8s) がある [3]．これは，コンテナ化されたアプリケーションの展開，スケーリングおよび管理を自動化するためのオープ

[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
[2] 東京工科大学大学院コンピュータサイエンス専攻
〒 192–0982 東京都八王子市片倉町 1404-1

[*1] 令和 3 年度 電子商取引に関する市場調査 報告書 経済産業省 商務情報政策局 情報経済課 https://www.meti.go.jp/press/2022/08/20220812005/20220812005-h.pdf (2022 年 11 月 17 日)

[*2] What is the average conversion rate? https://www.littledata.io/average/ecommerce-conversion-rate-(all-devices) (2022/11/17)

[*3] What is Cart Abandonment? GRAHAM CHARLTON https://www.salecycle.com/blog/strategies/what-is-cart-abandonment/#Abandonment_Surveys (2022/11/17)

[*4] 「ZOZO 開発組織の 2021 年の振り返りと現状」より https://qiita.com/sonots/items/629b8d5785c04ae9c953(2022/11/17)

[*5] GoogleCloudPlatform/microservices-demo https://github.com/GoogleCloudPlatform/microservices-demo (2022/11/21)

ンソースソフトウェアである [4]．Pod とは K8s 内で作成・管理できる最小のデプロイ可能なユニットである．Pod は 1 つまたは複数のコンテナのグループでストレージやネットワークの共有リソースを持つ．

課題

リクエストが来た際に，各マイクロサービスがネットワークで通信を行いタスクを実行することでレスポンスを返している．サービスごとレスポンスを返すためにタスクを実行する．例えば，microservices-demo では，ユーザがカートに商品を追加した場合，frontend がリクエストを受け取り，cartservice にリクエストを送る．cartservice が，ユーザがカートに追加した商品を Redis に保存し，frontend に追加したレスポンスを返す．frontend が cartservice からレスポンスを受け取り，ユーザにカートに追加したレスポンスを返す．K8s 上にデプロイされたマイクロサービスアプリケーションはリクエスト数が上昇すると，クラスタ内の CPU の使用コア数が増加する．マイクロサービスアプリケーションがクラスタ内の CPU の使用可能な上限のコア数を使用した際に，毎秒処理できるリクエスト数が上限に達し，応答時間が上昇する．

基礎実験

この基礎実験では，実際に課題があるか計測した．実験を行った環境は，GoogleCloudPlatform の microservices-demo を対象に, Locust を用いてリクエストを送り, Telegraf で Pod ごとに CPU の使用コア数を取得した．マイクロサービスの microservices-demo は，サービスごとに 1 つの Pod に分割されてデプロイされている．すべてのサービスの Pod 数が 1 の時に，Pod が使える CPU の limits を 200m に設定し，設定リクエスト数を 500,150 にした．実際に処理されたリクエスト数と応答時間の平均値を以下の表 1 に表す．基礎実験の結果，設定リクエスト数が 500 の時，リクエストした数と 300 (req/s) の差がある．また応答時間が 1461 (ms) 上昇している．

表 1 基礎実験の結果

設定リクエスト数 (req/s)	リクエスト数 (req/s)	平均応答時間 (ms)
150	150	8.89
500	200	1470

各章の概要

第 2 章では，本論文の関連研究を説明する．第 3 章では，本研究の課題を解決するための提案手法を説明する．第 4 章では，章の提案手法を実現するための実装を説明する．第 5 章では，第 3 章の提案手法の実験環境と実験の結果の分析を説明する. 第 6 章では，提案，実験，評価が本研究の課題を解決しているかを議論する．

2. 関連研究

関連研究として Anshul Jindal，Vladimir Podolskiy, Michael Gerndt らが行った Performance Modeling for Cloud Microservice Applications がある [5]． SLO を違反せずに処理できるリクエストの最大レートをマイクロサービスキャパシティー (MSC) として定義し，マイクロサービスごとに MSC を求めるため負荷テストを行った．取得したパフォーマンスデータから適切な回帰モデルを適合させマイクロサービスのキャパシティーを推定している．ただしこの研究では，実験で行っているアプリケーションが hello world を返すシンプルなものとなっている．また MSC をもとめるまでに時間がかかる．Han, Jungsu and Hong, Yujin and Kim, Jongwon らが行った Refining Microservices Placement Employing Workload Profiling Over Multiple Kubernetes Clusters がある [6]．これはマイクロサービスを経験的プロファイリングから得られるアプリケーションのワークロードの特性から，ヒューリスティックアルゴリズムを用いて複数の K8s クラスタからどのクラスタに配置をおこなうか決定する研究である．この研究では，単一のリクエスト数から得られたワークロードを用いて配置を行っている．リクエスト数の変化により，ワークロードの特性が変化することを考慮していない．

3. 提案

提案方式

Pod 数の上限を求める手法として，4 つの工程を踏んで求める．

1 つ目に，5 分間毎秒 50,100,150 回リクエストを送る負荷試験を行う．負荷試験を行った結果から，サービスごとに CPU の使用コア数の平均値を求める．5 分間行う理由は，Refining Microservices Placement Employing Workload Profiling Over Multiple Kubernetes Clusters の研究によると負荷試験で一定のリクエストを送った時に，CPU の使用量の結果が, 収束すると書かれていたためである [7].

2 つ目に，求められた平均値を用いて，マイクロサービスの WEB サイトにサービスを提供する HTTP サーバーを公開する機能を持つ frontend を 1 としたときの他のサービスの比を式 1 のように求める．M は frontend を除いたマイクロサービスのサービスの数を表している．frontend と他のサービスには相対的に比があると考えたからである．

$$1 : p_1 : p_2 ... p_{M-1} \tag{1}$$

3 つ目に，1 つ目に行った負荷試験のデータから，マイクロサービスの frontend の CPU の使用コア数を算出する近似式を最小 2 乗法を用いて求める．求めた式を式 2 のように定義する．n は毎秒リクエスト数，c は CPU の使用コ

ア数とする．

$$c = an + b \tag{2}$$

4 つ目に，デプロイされたマイクロサービスに割り当て可能な CPU のコア数の上限 $C_{\max}$ から，リクエスト数の上限 $n_{\max}$ を求める．1 つ目の比と 2 つ目の近似式を用いて式 3 を作成し，C を求める．C はマイクロサービスの全てのサービスの CPU の使用コア数である．

$$C = \sum_{i=1}^{M} p_i c \tag{3}$$

式 3 が $C_{\max}$ であるときの，リクエスト数の上限を算出する式を式 4 に表す．CPU のコア数の上限 $C_{\max}$ から比の合計で割り，式 2 の b,a を用いてリクエスト数の上限を算出する．

$$n_{\max} = \frac{\left(\frac{C_{\max}}{\sum_{i=1}^{M} p_i} - b \right)}{a} \tag{4}$$

基礎実験

提案においてマイクロサービスのサービスごとに CPU の使用コア数の比の特徴が，Pod 数とリクエスト数によって変化があるか実験を行った．実験を行った環境は，Google-CloudPlatform の microservices-demo を対象に，Locust を用いてリクエストを送り，Telegraf で Pod ごとの CPU の使用コア数を取得した．毎秒 50 リクエストですべてのサービスの Pod 数を 1，2，3 個としたときの比を図 1 に表す．

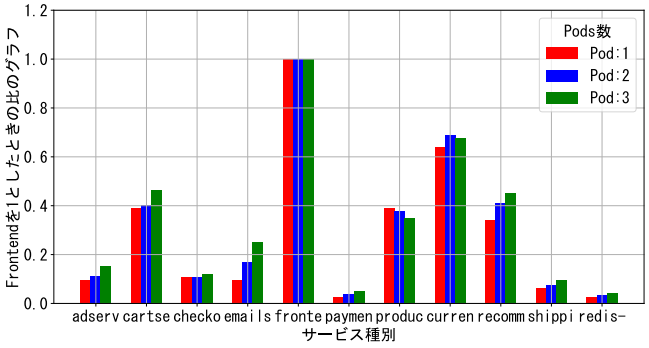

図 1 Pod 数が変化した時の frontend を 1 としたときのサービスごとの比

次に，全てのサービスの Pod 数が 2 個でリクエスト数を 50，100，150 と変化させたときの frontend を 1 としたときの CPU の使用コア数の比を図 2 に表す．

以上 2 つの実験から負荷試験を行った時に，リクエスト数が 50 と 150 の時では，比の誤差が，平均で 0.041 あった．Pod 数が 1，2，3 のときでは，比の誤差が平均 0.051 あった．以上により，リクエスト数と Pod 数では，サービスごとの CPU の使用コア数の比に大きく影響しないことが分かった．

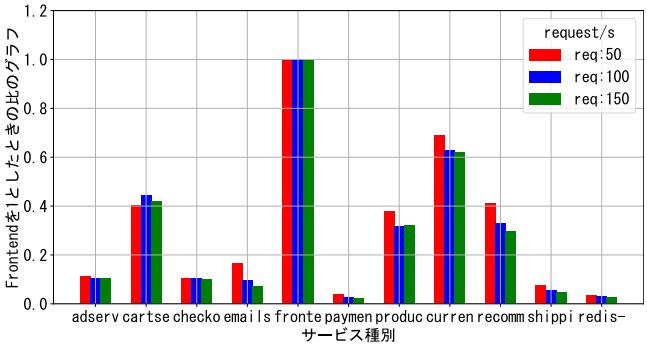

図 2 それぞれのリクエスト数の時の frontend を 1 としたときのサービスごとの CPU の使用コア数の比

ユースケース・シナリオ

マイクロサービスアプリケーションを運用している開発者がいる．開発者が負荷試験を行うことで提案を用いて，アプリケーションがマイクロサービスの CPU の使用可能なコア数からリクエストを処理できる上限を求める．

4. 実装

実装

提案の実装をした図を以下の図 3 に示す．Locust でリクエストを送り，負荷試験を行う．負荷試験中の CPU の使用コア数を Telegraf を用いて監視する．監視した CPU の使用コア数を InfluxDB に送信する．ソフトウェアが Locust のリクエストの統計データをもとに，InfluxDB から CPU の使用コア数のデータをクエリする．ソフトウェアがクエリしたデータと開発者から CPU の使用コア数の上限の値をもとに，計算した結果を開発者に出力する．

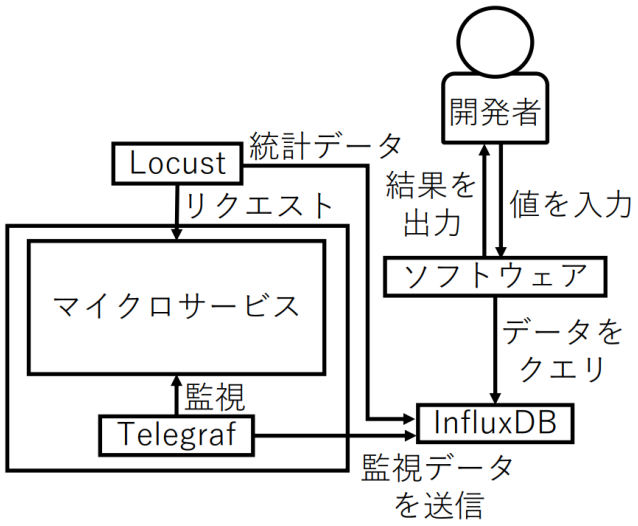

図 3 実装図

ソフトウェアは，Pyhton を用いて作成した．ソフトウェアのシーケンス図を以下の図 4 に示す．開発者が，Locust を用いて負荷試験を行った時のリクエストの統計データのファイルパスと，マイクロサービスをデプロイするクラス

タの使用可能な CPU のコア数を入力する．ソフトウェアが Locust のリクエスト統計データから負荷試験の開始時間と終了時間を取得する．その時間から，ソフトウェアが InfluxDB にマイクロサービスの Pod ごとの CPU の使用コア数を取得する．ソフトウェアが取得した Pod ごとの CPU の使用コア数をサービスごとに変換し，frontend を 1 としたときのサービスごとの比を算出する．ソフトウェアが負荷試験の結果から frontend の CPU の使用コア数とリクエスト数から最小 2 乗法を用いて近似式を算出する．ソフトウェアが算出した比と近似式，開発者が入力した使用可能な CPU のコア数からリクエスト数の上限を開発者に出力をする．

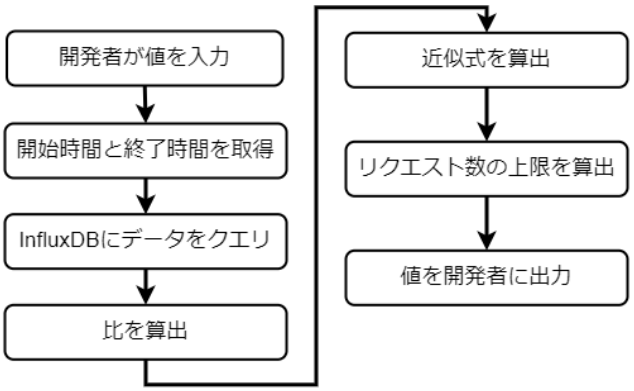

図 4　シーケンス図

5. 実験と分析

実験環境

EC サイトの実装を以下のように示す．K8s でデプロイされているアプリケーションを，GoogleCloudPlatform の microservices-demo とした[*6]．これは 11 個のサービスに分かれている．ユーザは商品を閲覧し，カートに追加して購入できる WEB ベースの EC サイトのアプリケーションである．この実験で行ったアーキテクチャを図 5 に示す．Locust が frontend にリクエストを送る．このマイクロサービスのメトリックスを取得・監視するため，マスターノードに Telegraf をインストールする．Telegraf で得られたメトリックスを InfluxDB に送信する．実験した時の Pod 数は 2 つで行った．実験環境の構成図を図 6 に示す．2 台の物理マシンがある．1 台目は，マシンの CPU は AMD Ryzen 9 5950X で 16 コアである．2 台目の CPU は，AMD Ryzen 7 3800X で 8 コアである．それぞれの物理マシンにハイパーバイザーの VMware ESXi がある．仮想マシンは合計 7 つある．うち 5 台で K8s クラスタの作成，1 つに Locust を導入し，もう 1 台に InfluxDB をおいた．K8s クラスタのディストリビューションは RKE2 を用いた．このクラスタにマイクロサービスをデプロイして実験を行った．このクラスタのマスターノードに，Telegraf を導入した．Telegraf は，クラスタ内のメトリックスを InfluxDB に送信する．この 5 つの仮想マシンの CPU とメモリは，4vCPU,4GB, 4vCPU,6GB, 4vCPU,7GB, 4vCPU,6GB, 4vCPU,6GB である．VM1 の Locust は，マイクロサービスにリクエストを送るものである．このマシンの CPU は 4vCPU, メモリは 6GB である．VM2 の InfluxDB は，Telegraf から送られてきた CPU やメモリのメトリックスを保存している．この仮想マシンの CPU は 5vCPU, メモリは 6GB である．

[*6] GoogleCloudPlatform/microservices-demo https://github.com/GoogleCloudPlatform/microservices-demo(2022/11/24)

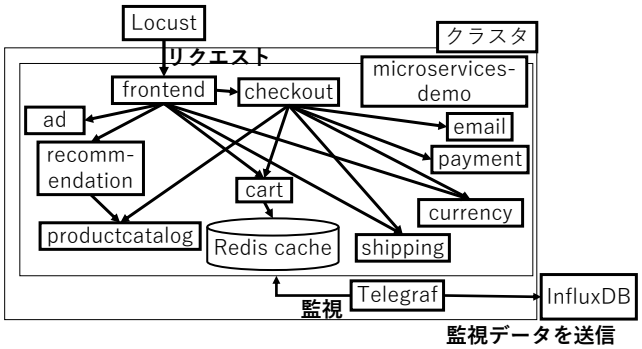

図 5　実験を行った際のアーキテクチャ

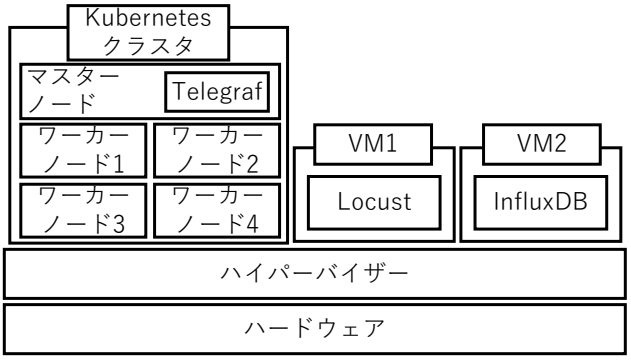

図 6　実験環境の構成図

Locust で負荷試験を行う際のリクエストのシナリオを記述する Locustfile.py は，第 1 章の背景で記述したコンバージョンレートとカート放棄率から作成した．また GoogleCloudPlatform の microservices-demo のソースコードがある GitHub 上に，Locust で負荷試験を行うテンプレートの locustfile.py を用いて作成した[*7]．

実験結果と分析

実験した結果について以下に示す．初めに，毎秒 50,

[*7] microservices-demo/src/loadgenerator/locustfile.pyhttps://github.com/GoogleCloudPlatform/microservices-demo/blob/main/src/loadgenerator/locustfile.py (2022/11/24)

100，150 回リクエスト送る負荷試験を行った時の frontend の CPU の使用コア数と，最小 2 乗法で求められた，近似式を図 7 に示す．次に，負荷試験を行った時のサービスご

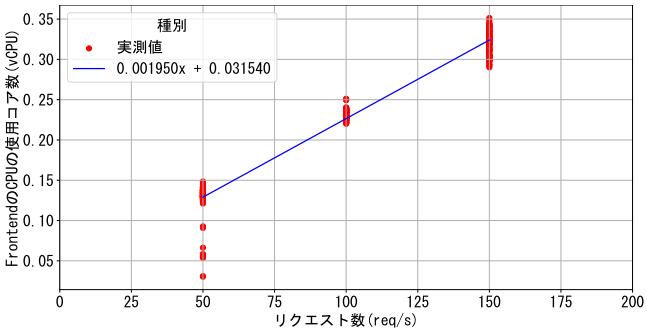

図 7 毎秒 50,100,150 回リクエストを送った時の CPU の使用コア数と最小 2 乗法で算出した近似式

とに比を図 8 に示す．

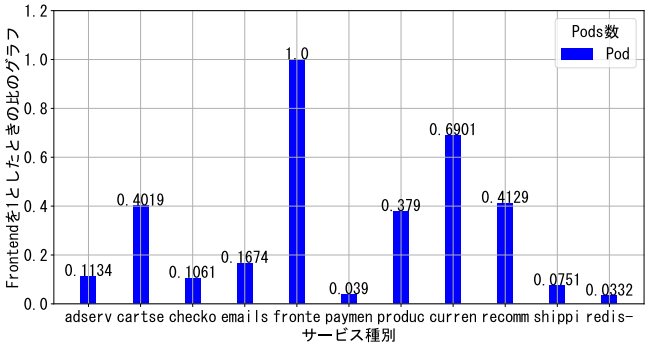

図 8 サービスごとの比

これにより提案方式を用いてリクエスト数の上限は，式 5 となる．

$$n_{\max} = \frac{\frac{C_{\max}}{3.418119} - 0.031540}{0.001950} \tag{5}$$

クラスタの使用可能な CPU のコア数を 0.83vCPU としたときに式 5 からリクエスト数の上限は，108.375899 となる．この値から Locust で毎秒 108 リクエスト行った際のマイクロサービス全体の CPU の使用コア数の平均値と，処理しているリクエスト数と全てのリクエストの w2qb v 平均応答時間を表 2 に表す．この実験結果から，毎秒 108 リクエストをマイクロサービスに送った時のマイクロサービスの全体の CPU の使用コア数の平均は，0.841 (vCPU) となった．また Locust のリクエストの統計データから，毎秒リクエスト数は，117 (req/s) と設定したリクエスト数と 9 異なった．

CPU の使用コア数の上限を 0.83 (vCPU) としたとき，本提案よりリクエスト数を算出し，108 (req/s) となった．実際に 108 (req/s) 送った際に，CPU の使用コア数は 0.84 (vCPU) と 0.01 しか変わらないため，CPU の使用コア数からリクエスト数を算出した結果 CPU の使用コア数と誤差 1%で算出することができた．

表 2 毎秒 108 リクエストを送った時の結果

全体の CPU の使用コア数の平均 (vCPU)	リクエスト数 (req/s)	平均応答時間 (ms)
0.841481	117.73	22.82

6. 議論

この実験では，GoogleCloudPlatform の microservices-demo のアプリケーションをもちいて実験を行った．しかし，提案方式の比が他のマイクロサービスに適合するかわからない．そのため，他のマイクロサービスにおいても同様のことがいえるのか，実験する必要がある．

7. おわりに

マイクロサービスがクラスタ内の CPU の使用可能上限のコア数を使用した際に，毎秒処理できるリクエスト数が上限に達し，応答時間が上昇する．リクエスト数の上限を求める手法として，負荷試験を行い，サービスごとの CPU の使用コア数の平均値の比と，最小 2 乗法を用いて算出した近似式から CPU の使用可能な上限をもとに，リクエスト数の上限を算出する．本提案を用いてリクエスト数の上限を算出した結果から，誤差 1%でリクエスト数の推定することができた．

参考文献

[1] Park, J., Choi, B., Lee, C. and Han, D.: GRAF: A Graph Neural Network Based Proactive Resource Allocation Framework for SLO-Oriented Microservices, *Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '21, New York, NY, USA, Association for Computing Machinery, p. 154–167 (online), DOI: 10.1145/3485983.3494866 (2021).

[2] Ranjan, R.: The Cloud Interoperability Challenge, *IEEE Cloud Computing*, Vol. 1, No. 2, pp. 20–24 (2014).

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

[4] Burns, B., Grant, B., Oppenheimer, D., Brewer, E. and Wilkes, J.: Borg, Omega, and Kubernetes, *Commun. ACM*, Vol. 59, No. 5, p. 50–57 (online), available from ⟨https://doi.org/10.1145/2890784⟩ (2016).

[5] Jindal, A., Podolskiy, V. and Gerndt, M.: Performance Modeling for Cloud Microservice Applications, *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, ICPE '19, New York, NY, USA, Association for Computing Machinery, p. 25–32 (online), DOI: 10.1145/3297663.3310309 (2019).

[6] Han, J., Hong, Y. and Kim, J.: Refining Microservices Placement Employing Workload Profiling Over Multiple Kubernetes Clusters, *IEEE Access*, Vol. 8, pp. 192543–192556 (online), DOI: 10.1109/ACCESS.2020.3033019 (2020).

[7] Han, J., Hong, Y. and Kim, J.: Refining Microservices Placement Employing Workload Profiling Over Multiple Kubernetes Clusters, *IEEE Access*, Vol. 8, pp. 192543–192556 (online), DOI: 10.1109/ACCESS.2020.3033019

(2020).

