マイクロサービスで動作するOnline Boutiqueへの負荷試験と応答時間のグラフ作成の自動化

西嶌 知良[1]　河竹 純一[2]　串田 高幸[1]

概要：ソフトウェアを開発する技法としてマイクロサービスというものがある．このようなアプリケーションを運用する場合，1 つのサービスの応答時間が遅い場合，アプリケーションの応答時間が遅くなってしまう．そのためアプリケーションのパフォーマンスの低下などのアプリケーションの劣化を防ぐには，アプリケーションのボトルネックとなっているマイクロサービスを発見し修正する必要がある．本論文ではボトルネック発見のために各マイクロサービスに負荷試験を行い，それぞれの処理能力を表すグラフを自動作成する．それによるシステム管理者のボトルネックの発見を容易にすることを目標にする．今回作成したコマンドの実行時間は 25 分 06 秒であった．

1. はじめに

背景

マイクロサービスは，小さな独立した複数のサービスでソフトウェアを構成する，ソフトウェア開発に対するアーキテクチャ的，組織的アプローチである[*1]．アプリケーション開発を行う上でこのアーキテクチャを使用することで，開発サイクルの短縮などの俊敏性の向上や，それぞれのサービスが独立しているためデプロイが容易であることがメリットとして挙げられる．

マイクロサービスで動作するアプリケーションとして Google-CloudPlatform がデモアプリケーションとして公開している Online Boutique が存在する．[*2]．このアプリケーションは 11 個のマイクロサービスからなる Web ベースの EC サイトのアプリケーションである．ユーザは商品の選択や，カートへの追加，購入などの動作を行うことができる．Online Boutique のアーキテクチャ図を以下の図 1 に示す．

マイクロサービスを用いたアプリケーションを運用する上で，ボトルネックとなるマイクロサービスはアプリケーションのパフォーマンスの劣化の原因となる [1]．ここでいうボトルネックによるパフォーマンスの劣化は，必ずしもエラーや障害につながるものではなく，リソースの飽和、

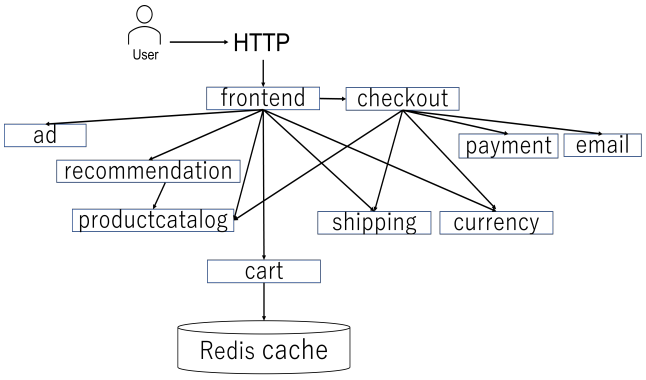

図 1　Online Boutique のアーキテクチャ図

リソースの競合、またはマイクロサービスアプリケーションの誤設定によって生じるパフォーマンスの低下のことである [2]．

マイクロサービスを用いたアプリケーションのボトルネックを発見することで，そのマイクロサービスの修正を行い，アプリケーションの性能を向上させることができるようになる [3]．

アプリケーションが通常の負荷状態や，ピーク時の負荷状態でどのように動作するのかを確認するためには，負荷試験を行う必要がある[*3]．マイクロサービスで動作するアプリケーションに対しては，それぞれのマイクロサービスに対して負荷試験を行うことで，マイクロサービスごとの処理能力を測ることができる．処理能力は，リクエスト量に対する応答時間で測る．これによって，処理能力がアプリケーションの中で他のマイクロサービスよりも劣ったマ

[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 バイオ・情報メディア研究科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] https://aws.amazon.com/jp/microservices/
[*2] https://github.com/GoogleCloudPlatform/microservices-demo
[*3] https://www.microfocus.com/en-us/what-is/load-testing

イクロサービスを発見することができる．

負荷試験を行うためのアプリケーションの一つに Locust がある．これは，オープンソースのアプリケーションであり，一般的なほかの負荷試験を行うアプリケーションと比べ開発が簡単である [4]．ユーザの動作を定義するシナリオは，Python のコードによって開発され，Locust によって負荷テストに含まれる．また，負荷をかけるアプリケーションに対して，何千もの仮想ユーザーロードを作成して負荷試験を行うことができる．

課題

本稿ではマイクロサービスによるアプリケーションのボトルネックによるパフォーマンスの劣化を課題とする．課題を図 2 に示す．

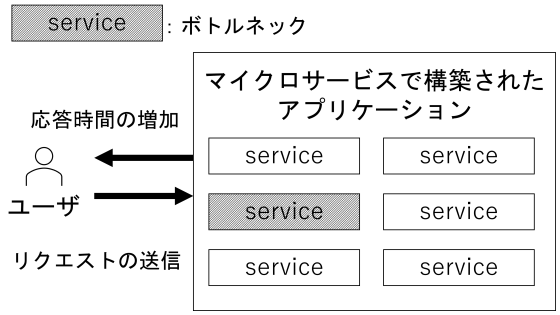

図 2 課題の概要図

この図では，アプリケーションを構築する一つマイクロサービスがボトルネックとなってしまうことでアプリケーションのパフォーマンスが劣化し，ユーザへの応答時間が増加することを示している．ユーザへの応答時間が 3 秒を超えると，ユーザの直帰率は 50%を超える[*4]．直帰率とは，ユーザが web サイトに訪れた際に何もせずにページから離れていく確率であり，最適な直帰率は 26%〜40%である[*5]．そのため，直帰率を最適なものに保つためには，システム管理者はアプリケーション中のボトルネックを発見し修正する必要がある．

各章の概要

第 2 章では, 関連研究について述べる. 第 3 章では, 本稿の提案方式の説明とユースケースの説明する. 第 4 章では, 提案方式を用いた実際の実装方法と実験方法について述べる. 第 5 章では, 評価方法と分析手法について述べる. 第 6 章では, 本稿の議論についてを述べる. 第 7 章では, 本稿のまとめを述べる.

[*4] https://webtan.impress.co.jp/e/2014/07/08/17757
[*5] https://blog.hubspot.com/marketing/what-is-bounce-rate-fix

2. 関連研究

Anshul Jindal らは，実装された機能や作業量による各マイクロサービスのリソース使用料の変化量を特定するという課題に取り組んでいる [1]．この研究ではマイクロサービスの個々の能力を特定するために各マイクロサービスをサンドボックス化し，それぞれのサービスに対して負荷試験を行う．その後，各マイクロサービスのパフォーマンスをモデリングし，分析を行う．最後に分析されたデータをグラフ・JSON 形式でユーザに提示し課題を解決する．この手法ではユーザにデータを提示するまでの時間が考慮されていない．

Jayathilaka らは，ソフトウェアにパフォーマンス異常が起きた際の根本的な原因を特定することを課題としている．この論文でいう根本的な原因とは，パフォーマンス異常の原因が作業負荷の増加，もしくは異常に遅いソフトウェアのコンポーネントが存在しているかどうかのどちらかである．[5]．この研究では Platform as a service(PaaS) サービスの Web アプリケーションを対象にしたリアルタイムな監視と診断を行うフレームワークである Roots を提案しているアプリケーションのパフォーマンスに異常が発生した際，根本的な原因が作業負荷ではなかった際に，すべてのコンポーネントへサービスコールを実行する．各コンポーネントがその実行にかかった時間を計測する．その結果を用いて，相対的重要度，時間に伴う相対的重要度の変化，分位数，テールエンドの値によって 4 つの候補を全コンポーネントの中から選択する．その後，過半数の手法で選ばれた候補をボトルネックとして選定するものである．この研究では動作中にボトルネックを検知するものであるため，本稿とは立ち位置が異なる．

Gunasekaran らは，マイクロサービスごとのコンテナへのリソースの割り当てが過剰になることによって発生する SLO 違反の増加を課題としている [6]．コールドスタートによる影響やステージごとのアプリケーションにおけるデータリクエスト時間とデータの活用，コンテナをスケールアウトの効果の調査を行う．これによって，ボトルネックとなるマイクロサービスの特徴づけを行う．

3. 提案

提案方式

本稿では，マイクロサービスで動作するアプリケーションのボトルネックを発見するためのグラフを作成する．ここで作成するグラフは各サービスごとの処理能力を見比べることができるように，サービスの最大応答時間とその際の秒間リクエスト量の関係を表したものとする．提案の概要図を図 3 に示す．

図 3 の流れについて具体的に説明する．

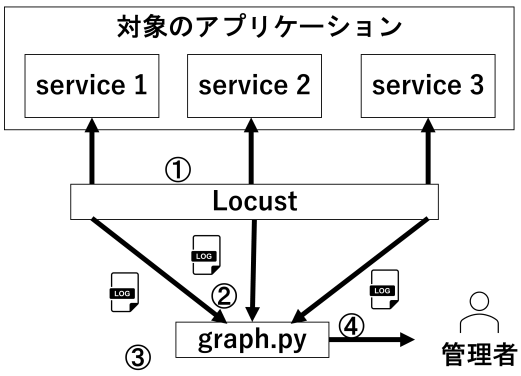

図 3 提案の流れ

① 各マイクロサービスの処理能力を測るために，負荷試験をそれぞれに対して行う．今回の提案では，Locust を用いて負荷試験を行う．テストシナリオは，それぞれのサービスに同じ量のリクエストを送ることができるようなものとする．

② 負荷試験終了後，各サービスごとの負荷試験のログを Locust を用いて取得する．

③ ②で取得したログからグラフを作成する．しかし，取得したログには今回使用しないデータも存在するため必要データだけを抽出する．その後，システム管理者がサービスごとの処理能力を比べることができるようにするために，最大応答時間と秒間リクエスト量，経過時間に関するグラフを作成する．

④ 作成したグラフをシステム管理者へ出力し，動作を終了する．

図 3 の動作を行うことで，システム管理者には負荷試験の結果による各サービスの処理能力に関するグラフが出力される．そのグラフから，各サービスの処理能力を比較することができるため，課題であるボトルネックの発見をすることができる．

ユースケース・シナリオ

本ユースケースシナリオでは，システム管理者が Kubernetes クラスタ上でマイクロサービスで動作するアプリケーションを運用することを想定している．ユースケースシナリオを図 4 に示す．

図 4 のユースケースの流れについて具体的に説明する．

① システム管理者が対処のアプリケーションの各サービスの処理能力を計測するために，対象のアプリケーションの情報とテストシナリオを負荷試験ツールに伝達する．

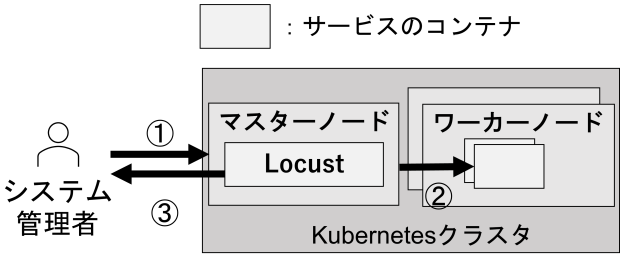

図 4 ユースケース図

② システム管理者から得た情報を基に，対象のアプリケーションの各サービスに対して負荷試験を行う．

③ 負荷試験から得た結果のログをシステム管理者に出力する．

マイクロサービスで動作するアプリケーションの処理能力はサービスの処理能力に影響を受けるため，アプリケーションのボトルネックとなっているサービスを発見し修正することが求められる．

4. 実装

本稿での実装はシェルスクリプトによって作成したコマンドで実装される．図 5 に実装図を示す．

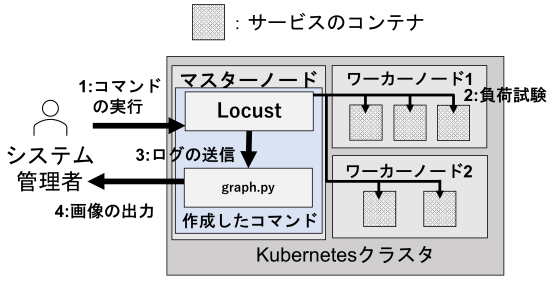

図 5 実装の概要図

図 5 は実装するコマンドの動作を示している．今回実装したコマンドは，Locust を用いて負荷試験をクラスタ上にあるアプリケーションの各サービスに対して行い，負荷試験の結果のログを出力する．その後，graph.py を用いてグラフを作成後，ユーザに出力するという内容である．

Locust で負荷試験を行う際のシナリオは，それぞれのサービスに同じ量のリクエストを送るために GET メソッドのみのリクエストを送信するようにした．また負荷試験において，ユーザの最大数は 1000 人，ユーザの増加量は毎秒 50 人，負荷試験にかける時間 300 秒とした今回，負荷試験を 300 秒行った理由は，サービスは CPU 使用率があるタイミングで高騰した後，安定した値に戻るためである [7].．負荷試験の際の結果のログは，Locust のオプションによってサービスごとにディレクトリに保存する．負荷試験の対象のサービスはユーザと直接やり取りを行う

frontend，カートの中身を閲覧する cart，言語の選択を行う currency，商品の詳細を閲覧する product，決済を行う checkout の 5 つである．また，Locust のコマンドラインのオプションによって負荷試験のログを取得する．

取得したログからグラフの作成を行うにおいては matplotlib と pandas を用いて python で graph.py というプログラムを作成し，グラフの作成を行った．図 6 に graph.py について示す．

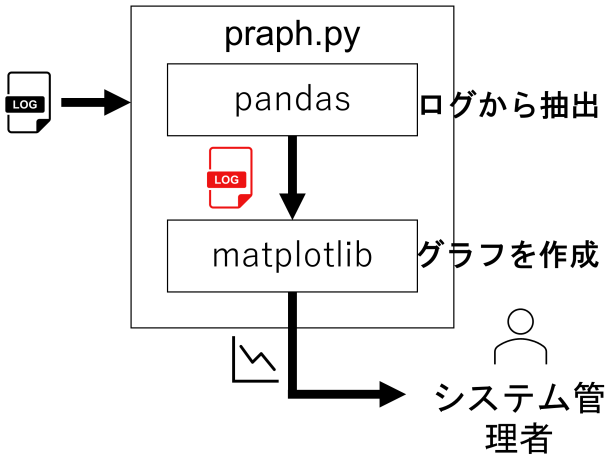

図 6　グラフの作成

まず，Locust から得たログを pandas によって必要なデータだけを抽出する．各サービスの処理能力に関するグラフを作成するために，抽出するデータは最大応答時間と，秒間リクエスト量，経過時間である．このデータをこの抽出したデータを用いて matplotlib によってグラフ化を行う．図 7 は作成されるグラフの例である．

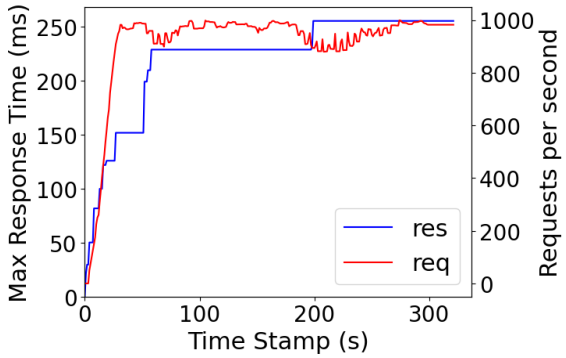

図 7　作成するグラフの例

このグラフは Online Boutique のフロントエンドのサービスである frontend に対して行ったものである．横軸は経過時間を表しており，左の縦軸と青の折れ線グラフが最大応答時間，右の縦軸と赤の折れ線グラフが秒間のリクエスト量を表すようなグラフを作成する．

図 7 のようなグラフを各サービスごとに作成し，それらを見ることでシステム管理者がサービスごとの処理能力を比べることができるようにする．

5.　実験と分析

実験環境

実験環境は K3S のクラスタを Ubuntu22.04 の仮想マシン 3 台で構築して使用した．各仮想マシンの性能は全て同じである．仮想マシンの性能の詳細を以下に示す．

OS:Ubuntu-22.04
vCPU:2 コア
RAM:4GB
HDD:30GB

実行環境の構成図を以下の図 8 に示す．

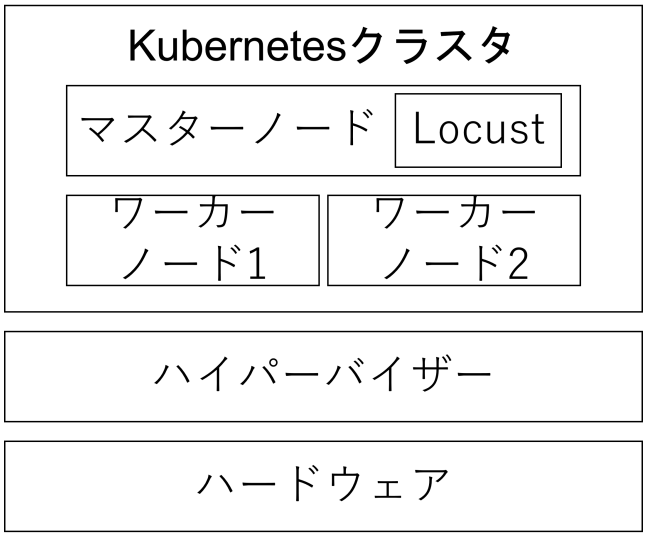

図 8　実行環境の構成図

それぞれのサービスに割り当てられているリソースを以下の表 1 に示す．

サービス	CPU(上限値)	RAM(上限値)
frontend	200m	128Mi
product	200m	128Mi
cart	200m	128Mi
checkout	300m	128Mi
currency	200m	128Mi

表 1　Caption

実験結果と分析

図 9, 10, 11, 12 にコマンドによって出力された frontend 以外のサービスである product，cart，checkout，currency のグラフを示す．

コマンド実行から上記のグラフを作成するまでにかかった時間は 25 分 06 秒であった．上記のグラフから，checkout

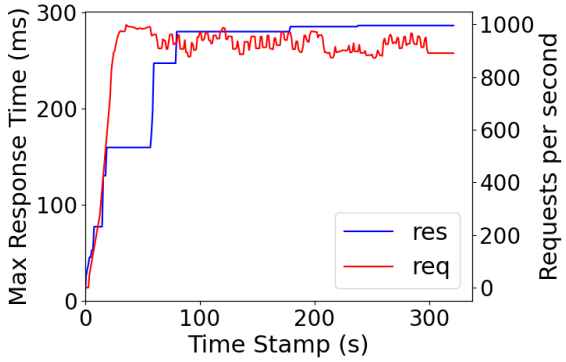

図 9 product の処理能力

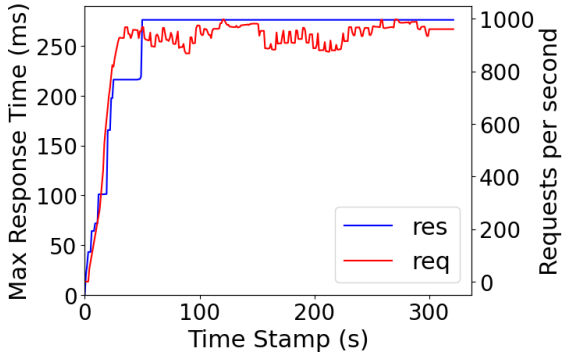

図 10 cart の処理能力

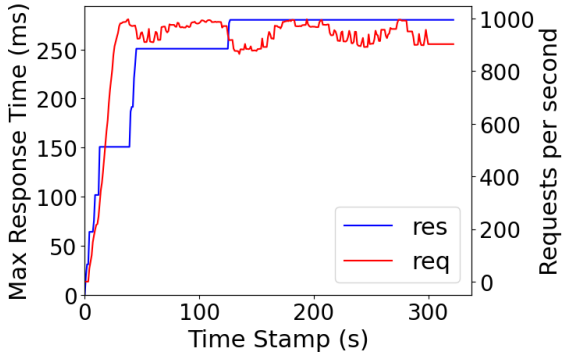

図 11 checkout の処理能力

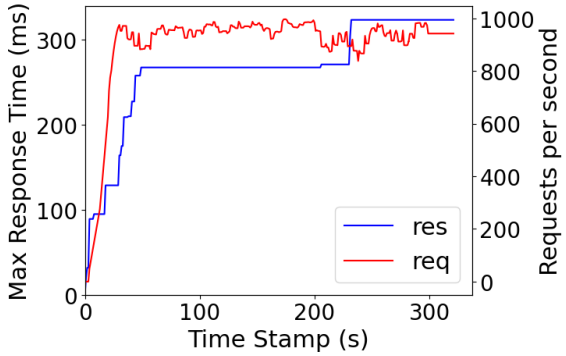

図 12 currency の処理能力

は序盤から応答速度が 250ms であり，ほかのサービスと比べ長くなっている．そのため，アプリケーションの中で checkout が最も処理能力が低いサービスであると考える．

6. 議論

今回はデモアプリケーションである Online Boutique を用いて実験を行った．負荷試験を行った対象は Locust で接続できる 5 つであったが，このアプリケーションに存在するマイクロサービスは 11 個である．そのため，エンドポイント数とマイクロサービス数の一致関係がなく，エンドポイントにできなかったマイクロサービスについて負荷試験を行えていない．そのため，最初にフロントエンドをエンドポイントにした後，その下にあるサービスを探知することで，今後エンドポイント数とマイクロサービス数が一致するように，アプリケーションのマイクロサービス数を探知するサービスを生成できれば良いと考える．

また，今回コマンドの実行に要した時間が 25 分 06 秒であった．対象の 5 つの各サービスに対して 300 秒，つまり 5 分の負荷試験を行った．コマンドの実行にかかった時間は負荷試験 5 つ分の時間とほとんど同じであり，グラフ作成のに時間はかからなかった．そのため，負荷試験を図 2 のように順番に行うのではく，並列にして動作させることによって，負荷試験時間の短縮を行うことができると考える．

7. おわりに

本研究の課題では，マイクロサービスで動作するアプリケーションのボトルネックによるパフォーマンスの劣化であった．提案手法は各サービスの処理能力のグラフを作成することであったが．今回の手法では，ボトルネック発見の手助けはできるものの，根本的な原因の解決はユーザが行わなければならないものとなった．

参考文献

[1] Jindal, A., Podolskiy, V. and Gerndt, M.: Performance modeling for cloud microservice applications, *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, pp. 25–32 (2019).

[2] Somashekar, G., Dutt, A., Vaddavalli, R., Varanasi, S. B. and Gandhi, A.: B-MEG: Bottlenecked-Microservices Extraction Using Graph Neural Networks (2022).

[3] De Camargo, A., Salvadori, I., Mello, R. d. S. and Siqueira, F.: An architecture to automate performance tests on microservices, *Proceedings of the 18th International Conference on Information Integration and Web-based Applications and Services*, pp. 422–429 (2016).

[4] Kemer, E. and Samli, R.: Performance comparison of scalable rest application programming interfaces in different platforms, *Computer Standards & Interfaces*, Vol. 66, p. 103355 (2019).

[5] Jayathilaka, H., Krintz, C. and Wolski, R.: Performance monitoring and root cause analysis for cloud-hosted web applications, *Proceedings of the 26th International Conference on World Wide Web*, pp. 469–478 (2017).

[6] Gunasekaran, J. R., Thinakaran, P., Nachiappan, N. C., Kannan, R. S., Kandemir, M. T. and Das, C. R.: Charac-

terizing bottlenecks in scheduling microservices on serverless platforms, *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, IEEE, pp. 1197–1198 (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 (2020).

