Sock Shopにおける内部トラフィックの取得とCPU使用率の比較による応答時間の増加の原因の特定

西嶌 知良[1]　大野 有樹[2]　串田 高幸[1]

概要：マイクロサービスアーキテクチャを用いて EC サイトのアプリケーションを運用する事例がある．その際，アプリケーション内の 1 つのマイクロサービスの応答時間が増加することがあり，他のマイクロサービスの応答時間も増加する．その中で，原因となるマイクロサービスの特定を行うことが困難であることが今回の課題である．本稿の提案手法では KialiAPI を用いてアプリケーション内の各マイクロサービスのトラフィックを取得することで通信経路を把握する．その後，負荷試験から各エンドポイントの応答時間から原因の候補の選定を行い，各マイクロサービスの CPU 使用率のメトリクスを取得し比較することでアプリケーションの応答時間増加の原因となるマイクロサービスを特定する．対象のアプリケーションである Sock Shop と提案手法の機能を持つソフトウェアを K3s クラスタに実装して実験を行う．実験ではアクセス集中によって応答時間を増加させる 8 個のテストシナリオを用いて特定の精度を確かめる評価実験を行い，6 つのシナリオで正確な特定が行われ，75.0%の精度で特定を行うことができた．

1. はじめに

背景

2023 年現在，日本における電子商取引（EC）の市場規模が拡大しており，EC をインターネット上の Web サイトで行っている例として Amazon や，eBay が挙げられる [1,2]．Web サイトにおいて，応答時間が 1 [s] から 3 [s] に増加した際にユーザがサイト上で何もせずにブラウザバックする行動を行う確率である直帰率が 32 %上昇すると Google が発表している[*1]．EC サイトにおける直帰はユーザがなにも購入せずにブラウザバックすることである．また 0.1 [s] の遅延ごとに売り上げが 1 %減少すると Amazon は推測している [3]．これらのことから EC サイトにおいて応答時間は売り上げに直結するものである．応答時間増加は大量のアクセスがサーバの処理能力を超える時に発生し，ユーザ体験の低下につながる[*2]．

Amazon や eBay はマイクロサービスアーキテクチャという手法を用いてアプリケーションを構築している [4,5]．マイクロサービスアーキテクチャは，機能ごとに分割され独立するマイクロサービスが API を介して通信することで大規模なアプリケーションを構築するアーキテクチャアプローチである．[6]．このアーキテクチャで構築されたアプリケーションは俊敏性や運用効率，柔軟性，スケーラビリティを得ることができる [7]．しかし，アプリケーションレベルとインフラストラクチャレベルでの通信を通じてマイクロサービスによる柔軟性が提供されるため，各マイクロサービスにおける関係が複雑になる [7]．マイクロサービス間の関係を管理するためにサービスメッシュが開発された [8]．これはマイクロサービスのプロキシとして機能し，各マイクロサービス間の通信を安全かつ高速であり，信頼性の高いものとした．Istio や Linkerd がサービスメッシュソリューションの例として挙げられる [9]．

マイクロサービスの自動スケーリングでは，パフォーマンスの低いマイクロサービスを短時間で正確にスケーリングすることが困難である．マイクロサービス間の複雑な通信を通してパフォーマンスの低さが他のマイクロサービスに伝播する可能性がある．そのため，同時に多数の異常なマイクロサービスが発生することがある [10]．

課題

マイクロサービスアーキテクチャで構築された EC サイトのアプリケーションにおいて，1 つのマイクロサービスの応答時間が増加すると，そのマイクロサービスを使用するエンドポイントの応答時間が増加してしまう．各エンド

[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://www.thinkwithgoogle.com/intl/en-ca/marketing-strategies/app-and-mobile/mobile-page-speed-new-industry-benchmarks/
[*2] https://patents.google.com/patent/US20100293275A1/en

ポイントの処理を行う際に様々なマイクロサービスを使用する．その中で，原因となるマイクロサービスの特定を行うことが困難であることが課題である．図 1 に課題となるケースを示す．あるアプリケーションがユーザから送信された HTTP リクエストの処理と出力を行う際，通信経路内で応答時間が増加するマイクロサービスが存在することがある．この場合，リクエストの処理に影響を及ぼし，レスポンスに時間がかかり応答時間が増加する可能性がある．

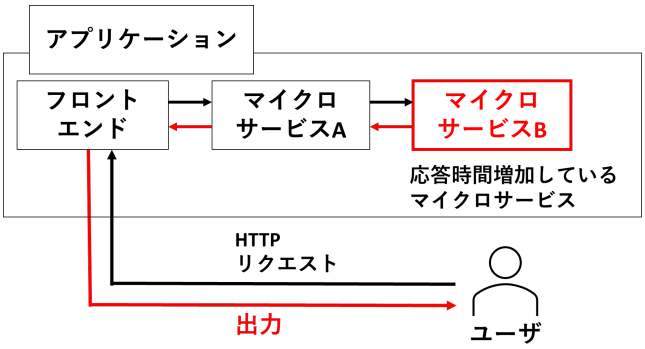

図 1　課題となるケース

1 つのマイクロサービスの応答時間増加が他のマイクロサービスの応答時間に影響をもたらすことを検証する実験を行った．この実験ではあるマイクロサービスの応答時間を増加させる前後の状態で他のマイクロサービスの応答時間に変化があるかを比較した．対象のエンドポイントは**/cart** とし，対象のマイクロサービスは **carts** と **orders** であった．結果では **carts** への処理数を増加させた後の**/order** の応答時間は増加させる前と比べて 1395[ms] 増加した [11]．そのため，1 つのマイクロサービスの応答時間増加が他のマイクロサービスの応答時間に影響をもたらすことが示された．また，**carts** では処理数が増えるほど最大 CPU 使用率が増加したものの，**orders** の最大 CPU 使用率の増加は見られなかった．

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をする. 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.

2. 関連研究

マイクロサービスアーキテクチャベースのアプリケーション向けの新しいパフォーマンステスト及び，分析の提案をしている研究がある [12]．ここでは多目的進化的アルゴリズムである NSGA-II と 3 シグマルールを用いてアプリケーション内のボトルネックの検出を行っている．実験では遅延が発生したリクエストの API を正確に特定している．しかし，3 シグマルールは取得するメトリクスが安定していることが前提条件であるため環境によって使用できない可能性がある．

マイクロサービス環境のアプリケーションにおける異常なサービスを特定し、位置を特定する研究がある [13]．評価実験では，88%の精度と 80%の再現率という結果であった．この研究では SLO メトリクスによって作成された因果関係グラフを横断することでパフォーマンス異常の原因を特定している．SLO 違反したノードの周辺ノードに SLO 違反がない場合，そのノードを候補として選定しており，その候補の中から 1 つのものを原因と出力しているため，異常であるマイクロサービスが複数ある場合の特定が困難である．

クラウドベースのマイクロサービスで構築されたアプリケーションにおいて，運用上の障害を特定するための人的労力とドメインの知識への依存を軽減可能で軽量な障害特定技術を提案した研究がある [14]．ランタイムログから作成されたエラーの時系列のモデルと実際のエラーの因果関係から障害の特定を行う．実験では 88.4%の精度で運用における障害の特定を行った．しかし，ログのみの診断ではエラーメッセージとして出力されたエラーにしか対応できないことや，大規模なログから異常情報のみを特定することが困難であることが課題である．

3. 提案

提案方式

提案方式では，負荷試験，各マイクロサービスの接続関係，応答時間，CPU 使用率を用いて応答時間増加の原因となるマイクロサービスの特定を行う．また，この提案では K3s クラスタ上に Sock Shop が 1 つデプロイされている状況で使用するものとする．提案方式における処理は大きく 3 つに分けられる．

(1) 通信経路の把握
　各マイクロサービスが処理を行う際の通信経路の情報を取得
(2) 候補の選定
　テストシナリオを用いて負荷試験を行い，応答時間が 3[s] を超えているものを候補として選定
(3) 原因の特定
　候補として選定されたマイクロサービスの CPU 使用率から応答時間増加の原因となるマイクロサービスを特定

まず通信経路の把握と候補の選定を行う．その後，2 つの提案から得た情報を用いて原因の特定を行う．ここから，各処理における方法について説明を行う．

通信経路の把握

初めに各マイクロサービス間の通信経路を把握し，接続関係を調べる．この関係を調べることで，あるアクセスに対して処理を行う際にどのマイクロサービスが接続しているかが分かる．例えば，EC サイトのカートに商品を入れるようなアクセスに対して処理を行う際には商品についての情報を持つデータベース，カートの内容を保存するデータベース，商品の情報をデータベースから取得する機能，カートのデータベースに商品の情報を保存する機能を持つマイクロサービスが通信を行い処理を行う．このように 1 つのアクセスに複数のマイクロサービスが関わっている．この時，アクセスの応答時間が売り上げに関わる数値であり，アクセスに関わるマイクロサービスが不明な場合，応答時間増加の原因となるマイクロサービスの特定は困難である．そのため，本提案では初めに各マイクロサービスの通信経路の把握を行う．

候補の選定

次に応答時間増加の原因となるマイクロサービスの候補を選定する．候補の選定方法を図 2 に示す．このユーザは

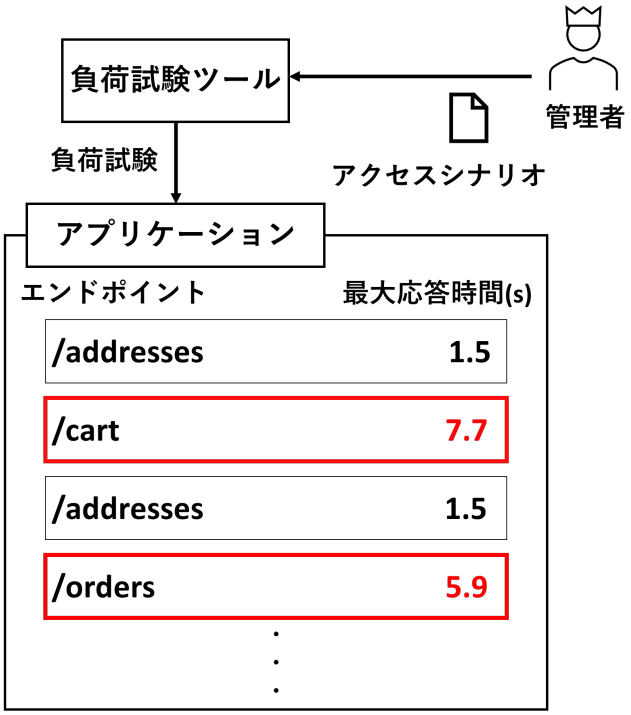

図 2　候補の選定方法

アプリケーションを管理しているシステム管理者である．ユーザの作成したアクセスシナリオと負荷試験ツールを用いてアプリケーションに対して負荷試験を行う．応答時間と直帰率，売上の関係から，この際の HTTP リクエストに対する最大応答時間が 3[s] を超えていた場合，そのマイクロサービスを候補として選定する．図 2 では，赤枠囲まれている部分が候補として選定される．ここでは各マイクロサービスの機能を使用するエンドポイントに対して HTTP リクエストを送信することで，マイクロサービスの応答時間とする．

原因の特定

最後に応答時間増加の原因となるマイクロサービスの特定を行う．候補として選定されたマイクロサービスから原因の特定する．図 3 に特定の方法を示す．まず，候補と

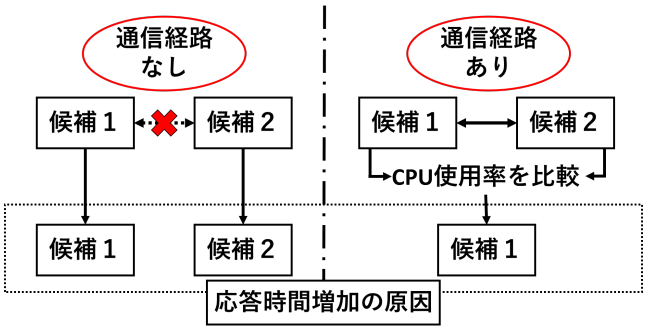

図 3　原因の特定方法

なっているマイクロサービス間の通信経路の有無から 2 通りの方法を使用する．通信経路の有無とはリクエストの処理を行う際にマイクロサービス間に通信が存在するかどうかである．例えば，**/orders** へリクエストを送信する際にはマイクロサービス間の通信経路に **carts** と **order** が存在する．この時，**carts** と **order** 間の通信経路はありとする．また，**/register** へリクエストを送信する場合，処理を行うマイクロサービスには **carts** が存在し，**orders** は存在しない．この時，**carts** と **orders** 間の通信経路はなしとする．候補間に通信経路が無い場合にはそれぞれのマイクロサービス名を原因として出力する．候補間に通信経路が存在する場合，候補の CPU 使用率の比較を行い，マイクロサービス名を結果として出力する．比較の方法は候補となるマイクロサービスの CPU，それぞれの平均使用率と最大使用率を比較し差が大きい方の候補を応答時間増加の原因とする．これは，マイクロサービスの処理数に応じて最大 CPU 使用率が増加しており，負荷試験の間，処理数が増加するとリソースの利用率が上昇しているかどうかを判断するために平均使用率と最大利用率の比較を行う．

ユースケース・シナリオ

本稿のユースケースは管理者がマイクロサービスアーキテクチャで構築された EC サイトのアプリケーションを運用していることを想定する．ユースケースを図 4 に示す．ユーザが EC サイトへ HTTP リクエストを送信し，アプリケーションがそのリクエストの処理を行いレスポンスを送信する．その際アプリケーションからユーザへのレスポ

ンスの応答時間が 3[s] を超える場合ユーザの直帰率は上昇してしまう．対象の管理者とアプリケーションの作成者が異なる場合，アプリケーション内部の通信経路が不明である．そのため，ユーザの HTTP リクエストに対する応答時間が 3[s] を超過している場合，応答時間増加の原因を特定することは困難である．そこで今回作成するソフトウェアを使用することで応答時間増加の原因となるマイクロサービスの特定を行い応答時間改善の手助けを行う．

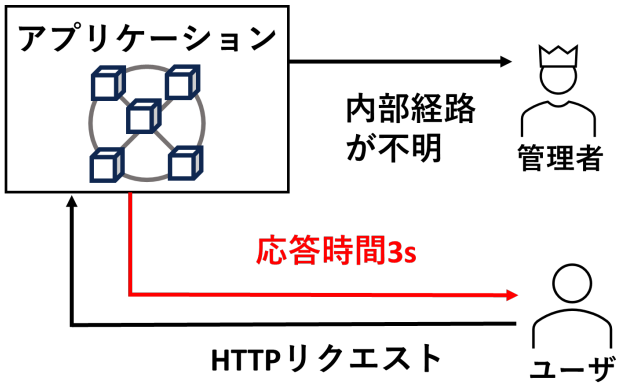

図 4　ユースケースシナリオ

4. 実装

提案手法を基にソフトウェアを実験環境への実装を行う．開発言語は Python を用いてソフトウェアを作成する．作成したソフトウェアは 2 つあり，1 つはマイクロサービスの通信経路を把握するソフトウェアであり，もう 1 つは負荷試験の結果からアプリケーション内で応答時間増加の原因となるマイクロサービスの特定を行うものである．対象とするアプリケーションは K3s クラスタ上にデプロイされた Sock Shop というアプリケーションであり，この中のマイクロサービスを対象とする．特定を行った後，そのマイクロサービス名を出力する．

通信経路を把握するソフトウェア

このソフトウェアでは KialiAPI[*3]を用いて通信経路の把握を行う．まず，対象のアプリケーションのマイクロサービスの Pod に対してサイドカーコンテナである Istio[*4]の Envoy コンテナを挿入する．その後，アプリケーションの持つ URL すべてに対してリクエストを送信するアクセスシナリオと Locust[*5]を用いて負荷試験を行う．負荷試験を実行している際に，KialiAPI を用いて各マイクロサービスのトラフィックを取得する．その後，取得したインバウンド，アウトバウンドトラフィックを relation.csv というファイルに保存することで通信経路の把握を行う．

[*3] https://kiali.io/
[*4] https://istio.io/
[*5] https://locust.io/

応答時間増加の原因を特定するソフトウェア

ここで作成するソフトウェアの流れを図 5 に示す．また，各処理の説明を以下に示す．

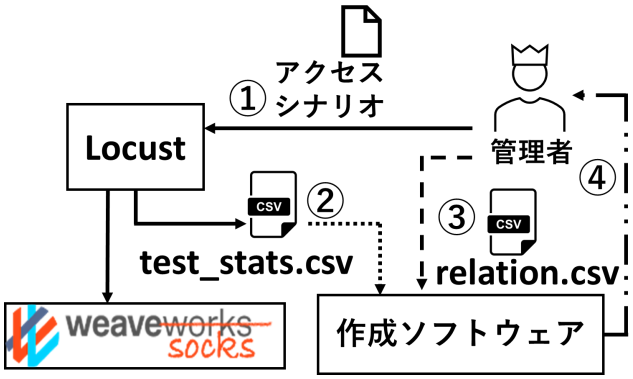

図 5　作成ソフトウェアの処理の流れ

① 作成したアクセスシナリオと Locust を用いて対象のアプリケーションに対して負荷試験を行い，結果を test_stats.csv として保存する．

② test_stats.csv から応答時間が閾値の 3[s] を超過するものを候補として選定する．

③ 選定された候補の中に同じ通信経路を持つマイクロサービスがあるかを relation.csv を用いて取得する．

④ 同じ通信経路を持つマイクロサービスがある場合，それぞれの CPU 使用率の比較を行い，応答時間増加の原因を特定する．同じ通信経路を持つマイクロサービスが無い場合は候補を応答時間増加の原因とし結果として出力する．

5. 実験

評価実験では対象のアプリケーションに対して応答時間を増加させる負荷試験を行い，その時に応答時間増加の原因となっているマイクロサービスを特定できているかどうかの精度を求めて評価を行う．そのため，実験ではアクセスシナリオを作成して実験を行う．作成するアクセスシナリオは温泉の予約・決済を行うことができる EC サイトの実例から，各エンドポイントに対してアクセス集中による応答時間遅延を引き起こすものとする[*6]．また，今回の実験では対象のアプリケーションと同じ LAN 内のサーバから HTTP リクエストを送信する．そのため，外部からリクエストを送信する際に必要な DNS 名前解決が存在しないが，これは通常数十ミリ秒から数百ミリ秒程度で行われ

[*6] https://www.manyo.co.jp/apology20221011/

る[*7]ため閾値は 3[s] とする．表 1 はアクセス集中を行う対象のエンドポイントを示し，対象は 8 種類である．

このテストシナリオを作成する基準は，Sock Shop が公開しているユーザの動きを模倣したテストシナリオ[*8]を参考にしている．ここに記されている各エンドポイントは全てで 9 種類存在する．その中で**/orders** を今回の対象のエンドポイントに選定していない．**/orders** は商品の注文を確定する際にリクエストを送信するエンドポイントであるため，カートの内容が空の時にはリクエストを送信した際にエラーが出てしまう．これを防ぐためには**/cart** というエンドポイントに対して商品をカートに入れるリクエストを送信しなければならない．今回の実験では負荷を与える対象のマイクロサービスは 1 種類であり，**/order** にアクセスを集中させる際に他のエンドポイントにもアクセスが集中し対象のマイクロサービスが 1 種類ではなくなるため今回の実験では 9 種類のエンドポイントのうち 8 種類のエンドポイントを対象とした．

ソフトウェアの出力するマイクロサービス名がアクセスシナリオの対象のマイクロサービスと一致しているかどうか正誤判定を行い，正答率を精度として評価を行う．表 1 が対象のエンドポイントとマイクロサービスである．また，負荷試験を行う際の条件を表 2 に示す．アクセス集中を行うために対象のエンドポイントには他のエンドポイントより 10 倍のリクエストを送信するようにした．

対象エンドポイント	対象マイクロサービス
/cart	carts-db
/detail.html	catalogue
/register	user
/addresses	user
/category.html	catalogue
/cards	payment
/basket.html	carts-db
/	catalogue-db

表 1　対象エンドポイントと対象マイクロサービス

最大ユーザ数 (人)	ユーザ生成人数 (人/秒)	実行時間 (s)
400	5	300

表 2　負荷試験の条件

実験環境

実験環境には 1 つのマスターノードと 2 つのワーカノードで構成された K3s[*9]クラスタを使用する．これらのノードはハイパーバイザーである ESXi を用いて作成される VM を使用する．また，これらの VM とは別に Locust を使用するための VM を使用する．これらの VM の構成要素を以下に示し，構成図を図 6 に示す．

・VM構成情報 (マスター)
　OS Ubuntu-22.04
　vCPU 8 コア
　RAM 10GB
　HDD 75GB
・VM構成情報 (ワーカー 2 台)
　OS Ubuntu-22.04
　vCPU 4 コア
　RAM 8GB
　HDD 75GB
・VM構成情報 (Locust)
　OS Ubuntu-20.04.2
　vCPU 2 コア
　RAM 8GB
　HDD 120GB

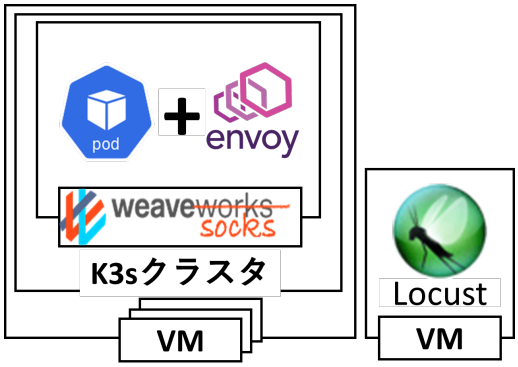

図 6　実験環境の構成図

対象のアプリケーションとしてマイクロサービスで構築された EC サイトのデモアプリケーションである Sock Shop[*10]を K3s 上にデプロイして使用する．また，デプロイされたマイクロサービスの Pod 内にサービスメッシュである Istio のサイドカーコンテナである Envoy コンテナを挿入することで通信経路の把握を行う．

実験結果と分析

負荷試験を行った際の対象のエンドポイントと応答時間が 3 秒を超えたエンドポイントのリクエスト数と最大応答時間 (MRT) を以下に示す．

実験結果を表 4 に示す．8 回の実験のうち，6 回の正解であった．そのため，精度は 75.0%であった．

誤判定の定義として，出力結果が対象のマイクロサービスと異なる場合と，通信経路内に存在するマイクロサービ

[*7] https://x.gd/H4vOK
[*8] https://x.gd/yapZv
[*9] https://k3s.io/
[*10] https://microservices-demo.github.io/

対象	エンドポイント	リクエスト数	MRT(s)
/cart	/cart	24604	10.0
	/orders	2438	5.8
/detail.html	/detail.html	23981	5.7
/register	/register	24192	17.3
	/orders	2398	5.9
/addresses	/addresses	20943	31.4
	/cart	2078	16.9
	/orders	2075	7.0
/category.html	/category.html	24012	3.8
/cards	/cards	23781	7.2
	/orders	2301	3.7
/basket.html	/basket.html	22015	13.6
	/orders	2208	5.2
/	/	20163	6.7
	/cart	2013	3.4
	/orders	2011	4.8

表 3 負荷試験中の出力結果の例

対象エンドポイント	出力結果	正誤
/cart	carts-db	正
/detail.html	catalogue	正
/register	user	正
/addresses	cart,user	誤
/category.html	catalogue	正
/cards	payment	正
/basket.html	carts-db	正
/	carts-db,catalogue-db	誤

表 4 実験の出力結果

ス同士が二つ出力された際に誤判定とする．誤判定したエンドポイントである**/addresses** と/について分析を行う．**/addresses** では購入する際に必要な住所登録を実行するリクエストを何度も送信した．そのため，ユーザの情報が保管する際に使用されるマイクロサービスである user が対象マイクロサービスであった．しかし，出力結果は cart と user であった．cart が出力されたのは，住所を入力した後に注文する際に，ユーザの情報を user を介して user-db から取得するためである．

/は Sock Shop のトップページの情報を取得するエンドポイントである．トップページには商品の画像と値段が表示されている．そのため，このエンドポイントに対してリクエストを集中させることで商品の情報が保管されている **catalogue-db** の処理が多くなる．結果では **carts-db** と **catalogue-db** が出力された．**carts-db** が出力された理由として，トップページではカートの内容を記す箇所があるため，その情報の取得のために cart-db に接続しているためである．誤判定した 2 つの出力結果がどちらも 2 つのマイクロサービス名を出力した理由は今回の手法では 2 つのマイクロサービスの間に経路が確認できなかったためであった．

6. 議論

本稿ではマイクロサービスアーキテクチャで構築されたアプリケーションにおける応答時間増加の原因となるマイクロサービスの特定を行った．しかし，経路の把握の失敗により誤判定してしまうケースが存在した．今回，経路の把握では全エンドポイントに対してリクエストを送信している際に KialiAPI を用いて各マイクロサービスのインバウンド，アウトバウンドのトラフィックを取得し，隣り合うマイクロサービスの確認は可能であった．しかし，今回の実験で誤判定したエンドポイントでは経路の最初と最後を把握する必要がある．そのため，今回の手法のように経路を取得する際に全エンドポイントにリクエストを送信してトラフィックを取得するのではなく，各エンドポイントごとに取得を行い，経路ごとにトラフィックの流れを取得する必要がある．

今回，候補の選定を行う際，閾値には最大応答時間を使用した．この場合，100 回のリクエストのうち 1 度でも 3[s] を超えた場合，応答時間増加の原因として特定してしまう．そのため，対象のアプリケーションの SLA に応じて応答時間の閾値を変更する必要がある．

7. おわりに

複数のエンドポイントの応答時間が閾値を超える際，マイクロサービスの依存関係からそのマイクロサービスが原因となるかを特定することが困難であることが本稿の課題である．そのため，各マイクロサービスの通信経路の把握と CPU 使用率の比較から応答時間増加の原因の特定を行った．実験では 8 つのアクセスシナリオを用いて負荷試験を行い，アクセスシナリオに応じてて各エンドポイントの応答時間を増加させた．結果では 75.0%の精度で特定を行った．各エンドポイントごとにマイクロサービスのつながりを取得することで精度の上昇につながる．

参考文献

[1] OIKAWA, H., OTAKE, K. and NAMATAME, T.: Proposal of Loyal Customer Discriminant Model Based on RFM Concept—Empirical Analysis using Golf EC Site Data—, *development*, Vol. 4, No. 1 (2020).

[2] Mirescu, S. V. and Maiorescu, T.: The premises and the evolution of electronic commerce, *Journal of knowledge management, economics and information technology*, Vol. 1, No. 1, pp. 44–56 (2010).

[3] Teo, Y. M.: Modelling flash crowd performance in peer-to-peer systems: Challenges and opportunities, *2014 5th International Conference on Intelligent Systems, Modelling and Simulation*, IEEE, pp. 3–4 (2014).

[4] Tran, G. K.: Microservice architecture in logistics business (2020).

[5] Guo, X., Peng, X., Wang, H., Li, W., Jiang, H., Ding, D., Xie, T. and Su, L.: Graph-based trace analysis for

microservice architecture understanding and problem diagnosis, *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1387–1397 (2020).

[6] Winchester, G., Parisis, G. and Berthouze, L.: On the Temporal Behaviour of a Large-Scale Microservice Architecture, *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*, IEEE, pp. 1–6 (2023).

[7] Filippone, G., Mehmood, N. Q., Autili, M., Rossi, F. and Tivoli, M.: From monolithic to microservice architecture: an automated approach based on graph clustering and combinatorial optimization, *2023 IEEE 20th International Conference on Software Architecture (ICSA)*, IEEE, pp. 47–57 (2023).

[8] Colman, A., Uzunov, A. V., Vo, Q. B. and Chhetri, M. B.: Agent Controlled Service Meshes for Resilient, Self-Adaptive Microservice Systems, *2023 IEEE International Conference on Software Services Engineering (SSE)*, IEEE, pp. 1–7 (2023).

[9] Ponomarev, K. Y.: Attribute-based access control in service mesh, *2019 Dynamics of Systems, Mechanisms and Machines (Dynamics)*, IEEE, pp. 1–4 (2019).

[10] Xie, S., Wang, J., Li, B., Zhang, Z., Li, D. et al.: PBScaler: A Bottleneck-aware Autoscaling Framework for Microservice-based Applications, *arXiv preprint arXiv:2303.14620* (2023).

[11] 知良西嶌, 高橋風太, 串田高幸: マイクロサービスで動作するアプリケーションの応答時間増加の原因の特定, 技術報告 CDSL-TR-143, Tokyo University of Technology CDSL Technical Report, (online: https://ja.tak-cslab.org/techreport) (Jan.21, 2023).

[12] Pei, Q., Liang, Z., Wang, Z., Cui, L., Long, Z. and Wu, G.: Search-Based Performance Testing and Analysis for Microservice-Based Digital Power Applications, *2023 6th International Conference on Energy, Electrical and Power Engineering (CEEPE)*, IEEE, pp. 1522–1527 (2023).

[13] Lin, J., Chen, P. and Zheng, Z.: Microscope: Pinpoint performance issues with causal graphs in micro-service environments, *Service-Oriented Computing: 16th International Conference, ICSOC 2018, Hangzhou, China, November 12-15, 2018, Proceedings 16*, Springer, pp. 3–20 (2018).

[14] Aggarwal, P., Gupta, A., Mohapatra, P., Nagar, S., Mandal, A., Wang, Q. and Paradkar, A.: Localization of operational faults in cloud applications by mining causal dependencies in logs using golden signals, *International Conference on Service-Oriented Computing*, Springer, pp. 137–149 (2020).

