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

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という課題が存在する．本稿の提案手法はアプリケーション内の接続を把握し，負荷試験から各マイクロサービスの応答時間，CPU・メモリ使用率のメトリクスを取得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 クラスタにデプロイして実験を行う．基礎実験では，cart に対して商品を入れて消す動作を行う HTTP リクエストを 1 回，2 回，5 回，10 回，15 回にわけて送信した．その際の order の応答時間は通常の HTTP リクエストを送信したときと比べ 15 回の際には，1395ms 増加した．

1. はじめに

背景

インターネット上で物の売買を行う Web サイトとして EC サイトが存在する．近年，新型コロナウイルスの影響から，在宅時間が増えたことにより，需要が増加している [1]．マイクロサービスアーキテクチャは Amazon(Amazon.co.jp) や Netflix(Netflix.com) のような巨大な Web システムを持つ企業から生まれた手法である [2]．このアーキテクチャはソフトウェア・システムを小規模なサービスのパッケージとして実現することを目的としたクラウドネイティブなアーキテクチャである．各マイクロサービスは異なるプラットフォームや技術スタックを採用することができ，独立したデプロイが可能である．マイクロサービス間の通信は軽量なメカニズム，例えば Finagle を通じて通信を行う．サービスメッシュは管理可能なサービス間通信プラットフォームとして機能し，アプリケーション実行時の操作を標準化することができる [3].

マイクロサービスアーキテクチャは動的な環境であり，サービスの依存関係が複雑である [4]．さらに，アプリケーションの動作はマイクロサービス間のトラフィックの流れに依存するため，実行中のプログラムを制御するためにトラフィックの制御が必須となる．その中で，サービスメッシュというソフトウェアが出現した．これは，サービスの実装に変更を加えることなく，マイクロサービス上に専用のインフラストラクチャ層を導入することができ，動的な環境で実行中のプログラムを制御することを容易にする有望なアプローチである [5].

Google の発表では Web サイトの直帰率は応答時間が 1s から 3s に増加した際に，32 %増加するとされている[*1]．ここでいう直帰とは，ユーザが Web サイトを訪れた際に着地した 1 ページだけを見てそのままサイトから離脱する行動のことである．EC サイトにおいて直帰率が高いということは，ユーザが Web サイトに訪れて何も購入せずにブラウザを終了するということになる[*2]．また，Amazon は 0.1 秒の遅延ごとに売り上げが 1 %減少すると推定している [6]．そのため，EC サイトにおいて，応答時間が増加している場合，直帰率が増加し，Web サイトの売り上げが減少する．

マイクロサービスアーキテクチャで構築するアプリケーションは，単一のマイクロサービスがアプリケーション全体の応答時間を増加させる原因となる可能性がある [7]．応答時間増加の根本的な原因となるものはロックやディスクへの読み書きを例とするソフトウェアリソースや，CPU やメモリを例とするハードウェアリソースが挙げられる．そ

[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://www.linkedin.com/pulse/how-can-we-use-personalization-lower-e-commerce-bounce-navneet-singh-

のため，使用環境やアクセス状況に応じた設定を行わなければアプリケーションの応答時間が増加する．

課題

本稿における課題はアプリケーションをマイクロサービスアーキテクチャで構築する際に，マイクロサービス間に依存関係がある点から応答時間増加の原因を特定することが困難であるという点である [8]．例えば Sock Shop では，カートにある商品を管理する cart，注文を行う order というマイクロサービスがある．この 2 つには依存関係があり，order で注文を行う場合には cart にアクセスを行い，カートの中身を参照する必要がある．この際 cart の応答時間が増加する場合， order の応答時間も増加する．

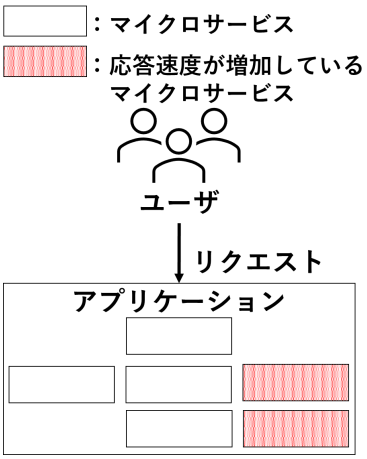

図 1　課題の図

上の図 1 は課題を示しており，ユーザが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対して HTTP リクエストを送った際の図である．図のようにマイクロサービス間の依存関係が分からない際にはどのマイクロサービスが原因で応答時間が増加しているかを判明することができない．

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で構築された Web サービスに対して，マイクロサービス間でリソースを予測可能に共有し全体のリソース利用率を向上させる高精度できめ細やかなリソース管理フレームワークである FIRM を提案している研究がある [9]．クラスタレベルの CPU 使用率を最大 33 %向上させ，タイムアウトしたユーザリクエスト数を最大 8 倍削減している．原因となるマイクロサービスを特定する際に SVM を用いている．その中で，モデル作成のための学習データの準備やモデルの構築の学習時間が増大することが課題である．

QoS 違反につながる空間的および時間的パターンを学習するためにクラウドシステムが収集する大量の追跡データを活用するオンラインクラウドパフォーマンスデバックシステムである Seer を提案している研究がある [10]．Seer は，91 %の確率で QoS 違反を正しく回避し，84 %のケースで最初から QoS 違反を回避した．この提案では根本原因を特定するために大量のデータをディープラーニングを適応する手法である．そのため，根本原因の特定に使用するメトリクスを取得するためにソースコードをインストルメンテーションする必要がある．そのためマイクロサービスの更新が頻繁に行われるとパフォーマンス低下につながる．

サービス指向アーキテクチャにおける以上の根本原因を見つけるために必要な時間，ドメイン知識，人的労力を削減できるアルゴリズムである MonitorRank を提案している研究がある [11]．ここでは，ベースラインおよび現在の最先端の方法と比較して，根本原因を見つける平均精度を 26 %から 51 %向上していることが示されている．しかし，この提案では外部要因から根本原因と特定を行っている．そのため，フロントエンドに影響を与えないバックエンドの異常が起きている際に根本原因となるマイクロサービスを特定できない．

3. 提案

提案方式

本稿では，マイクロサービスで構築するアプリケーションにおいて，応答時間の増加の原因となるマイクロサービスを特定する．本稿で提案するソフトウェアの流れを以下の図 2 に示す．

図 2　提案ソフトウェアの流れ

上記の図のように，応答時間増加の原因を特定するにあたり，図 2 の①のように各マイクロサービスのアクセスログを用いて依存関係を把握する．対象のアプリケーションの全ての動作を行うテストシナリオを用いて負荷試験を行う．これにより各マイクロサービスがそれぞれの機能を果

たすことができ，参照しているマイクロサービスすべてにアクセスを行わせることが可能である．下記の図 3 は各マイクロサービスの依存関係を把握を示す．各マイクロサービスの Pod 内に挿入されたサービスメッシュに保存されているアクセスログには，対象の Pod がどこにアクセスしているかを確認することができる．そのため，アクセスログから対象の Pod のアクセスログからアプリケーションの依存関係を把握する．

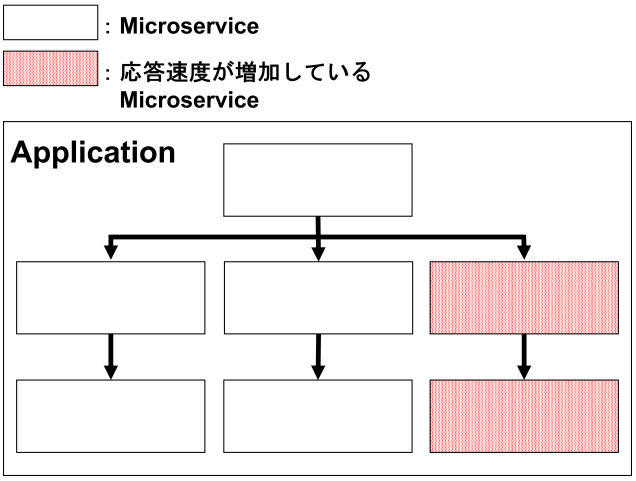

図 3　依存関係の把握

次に図 2 の②のように各マイクロサービスの最大応答時間から候補を選定する．以下の図 4 は候補の選定についてまとめたものである．

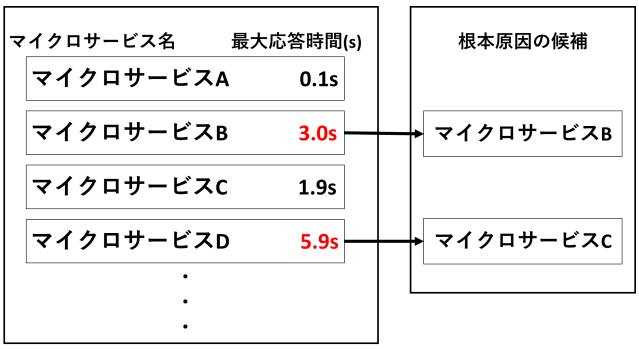

図 4　候補の選定

図 4 のようにマイクロサービスの最大応答時間を使用して候補を選定する．直帰率の関係から最大応答時間の閾値を 3s とし，Locust の負荷試験で最大応答時間が閾値を超えるマイクロサービスを原因の候補とする．

最後に，依存関係のあるマイクロサービスがある場合，図 2 の③のように根本原因となるマイクロサービスを特定する．図 5 は特定を行う際の例である．

例えば，order というマイクロサービスが cart というマイクロサービスを参照している場合，cart の処理時間が増加していると order が cart からのレスポンスを受け取るまでの時間も増加する．そのため，CPU・メモリ使用率を比較し，特定を行う．

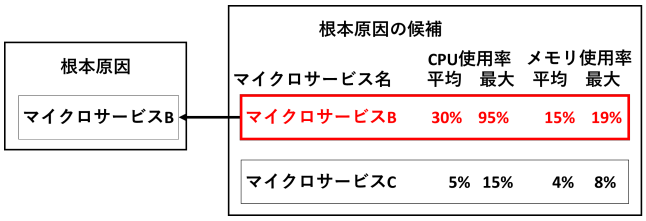

図 5　特定の例

ユースケース・シナリオ

本ユースケースシナリオでは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システム管理者が運用していることを想定する．図 6 にユースケースの概要を示す．

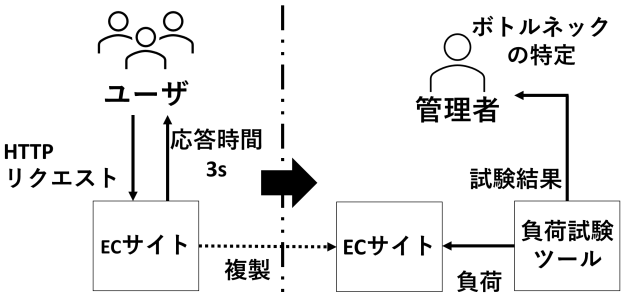

図 6　ユースケースの概要図

運用しているアプリケーションの応答時間が 3s を超えている場合，応答時間と直帰率の関係から売り上げが落ちてしまう．そのため，開発者が負荷試験を行い，アプリケーションの応答時間が増加している原因となっているマイクロサービスを特定する．

4.　実装

提案手法を基にしたソフトウェアを新たに作成する．今回，ソフトウェアを作成するにあたってプログラミング言語は Python を使用する．作成するソフトウェアは，マイクロサービスの依存関係を特定し，最大応答時間のメトリクスから候補を選定，CPU・メモリ使用率から応答時間増加の原因を特定するものである．

図 7 はソフトウェアの概要を示している．アクセスログを取得し，それを用いて Pod の接続関係を把握する．負荷試験ツールを使用し，対象のアプリケーションに対して負荷試験を行う．その際の負荷試験の結果から，応答時間や Pod の CPU・メモリ利用率のメトリクスを取得する．取得した応答時間から応答時間増加の原因の候補となるマイクロサービスを選定する．その後，候補・接続関係からの CPU・メモリ使用率から応答時間増加の原因を特定しユー

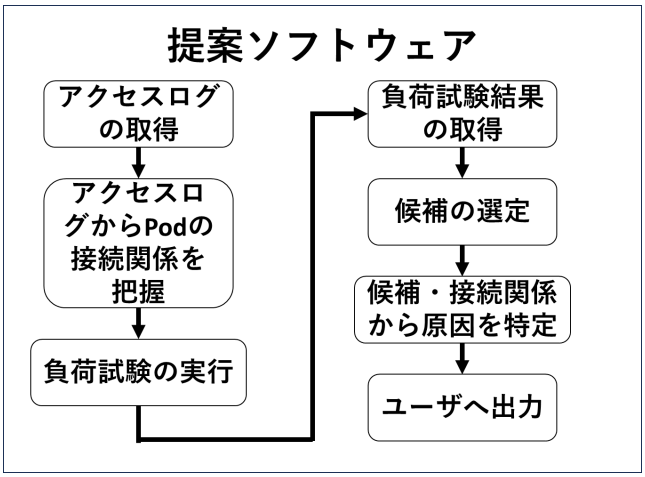

図 7 ソフトウェアの概要図

ザへ出力する．

まず，各マイクロサービスの Pod に挿入した Envoy プロキシからそれぞれのマイクロサービスのアクセスログを取得する．プログラム 1 が実際のログである．

プログラム 1 cart のログの例

```
[2023-06-28T17:44:34.046Z] "GET
/carts/649c71783ea47b0001bf6ba6/items
HTTP/1.1" 200 - via_upstream - "-" 0 113
574 574 "-" "Java/1.8.0_111-internal"
"3d8fdfa0-ac83-977b-a2a7-723b3bb2beb3"
"carts" "10.42.1.11:80"
outbound|80||carts.sock-shop.svc.cluster.local
10.42.0.13:36410 10.43.101.47:80
10.42.0.13:40240 - default
.cluster.local default
```

このログは order の各マイクロサービスへのアクセスログの一部である．このログでは cart に対して GET リクエストを送っており，order が商品を注文する際に cart からカートの内容を参照するための動作を示している．このように，ログから各マイクロサービスの行動を確認することができ，依存関係を把握することができる．

次にマイクロサービスの最大応答時間から応答時間増加の原因の候補を選定する．今回の候補の選定の際に使用する閾値は 3s にする．これは直帰率と応答時間の関係からこの閾値を設定する．

最後に CPU・メモリ利用率から応答時間増加の原因を特定する．閾値は特に設定をせず，リソース利用率の平均と最大値に差があるものを選択する．

5. 実験

本稿の課題となるマイクロサービスアーキテクチャで構築された EC サイトにおいて，各マイクロサービスの依存関係による応答時間の増加が起こるかどうかを確かめる．今回は cart と order における最大応答時間を確認し，負荷試験中の各マイクロサービスの CPU・メモリ使用率から依存関係があるかどうかを確認する．

実験環境

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いて，あるマイクロサービスによって他のマイクロサービスの応答時間が増加するという課題を基礎実験を行う．サーバとしてハイパーバイザの ESXi から作成される仮想マシン (以下VM) を用いる．以下に課題の基礎実験に用いるVMの構成情報を示す．

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

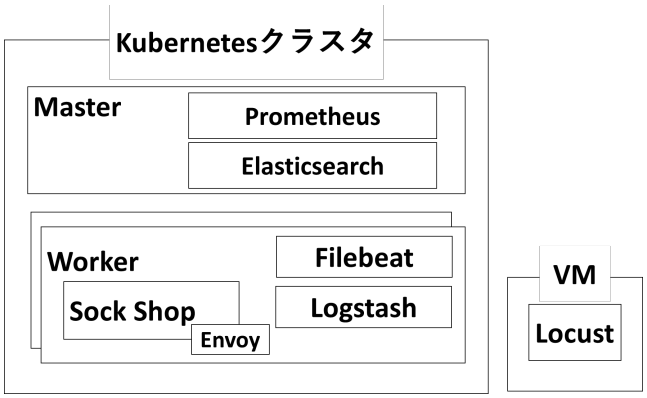

図 8 実験環境

上記の図 8 は本稿での実験環境を表している．本稿の実験環境では 1 台のマスターノードと 2 台のワーカーノード，合計で 3 台の VM で K3s クラスタを作成し使用した．K3s 上に Weaveworks が公開しているマイクロサービスアーキテクチャのデモアプリケーションである Sock Shop をデプロイする．また，Sock Shop をデプロイする際に各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

イドカーコンテナの Envoy コンテナを挿入する．また，各ノードに Filibeat と Logstash をデプロイする．これにより各 Pod のアクセスログを Elasticsearch に格納する．また，マイクロサービスのコンテナのリソース利用率を監視・格納のために Prometheus をデプロイした．

課題はあるマイクロサービスの応答時間が増加している際，他のマイクロサービスの応答時間も増加することである．order は cart を参照するため，cart に大量の HTTP リクエストを送信する負荷をかけ，order の応答時間が増加するかどうかを確かめる．そのためのテストシナリオを Python で作成する．内容は，ユーザ登録を行いカートの中身の商品を入れて消す動作を行った後に商品を注文するという動作である．cart の負荷に対する order の応答時間を確認するためにカートへの商品の出し入れの往復回数を 1 回，2 回，5 回，10 回，15 回に分けて実験を行う．

実験結果と分析

あるマイクロサービスの応答時間が増加すると，他のマイクロサービスの応答時間も増加するという課題を実現するために，負荷試験を行った．その際の各マイクロサービスの最大応答時間の結果について図 9 に示す．縦軸は負荷試験中の最大応答時間，横軸はカートへの出し入れの往復回数を示している．徐々に cart に対する HTTP リクエストの量を増加させていくことで cart の最大応答時間は上昇している．また，order には負荷をかけていないものの，cart への負荷が増加するにつれて order の最大応答時間も増加している．原因は，order が cart の内容を参照する場合，order の HTTP リクエストに対する cart の応答時間がカートへの出し入れの往復回数に伴って増加するためである．カートへの出し入れの往復回数が 15 回の時は 1 回の時より最大応答時間が 1395ms 増加している．

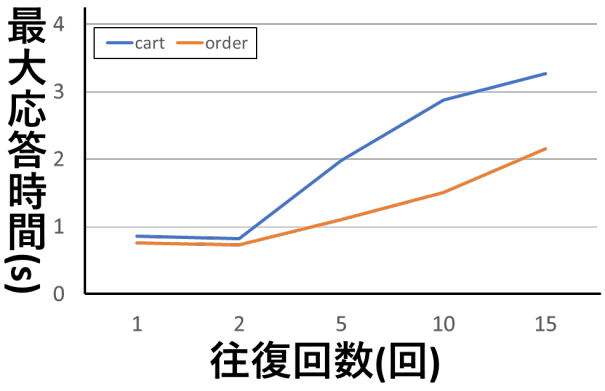

図 9　各マイクロサービスの最大応答時間

図 10,11 は負荷試験中の cart，order，そして cart のデータベースである cart_db というマイクロサービスの最大 CPU・メモリ使用率を示している．cart への負荷が大きくなるほど CPU 使用率は上昇しており，カートへの出し入れの往復回数が 15 回の時の cart_db の CPU 使用率は 84.1%であり，カートへの出し入れの往復回数が 1 回の時の 25.7%と比べて 58.4%上昇している．cart の場合，CPU 使用率は上昇傾向にあったもの，カートへの出し入れの往復回数が 5 回以上になると一定の数値を保っている．order には負荷をかけていないため，CPU 使用率の変化は見られなかった．また，メモリに関しても変化はなかった．

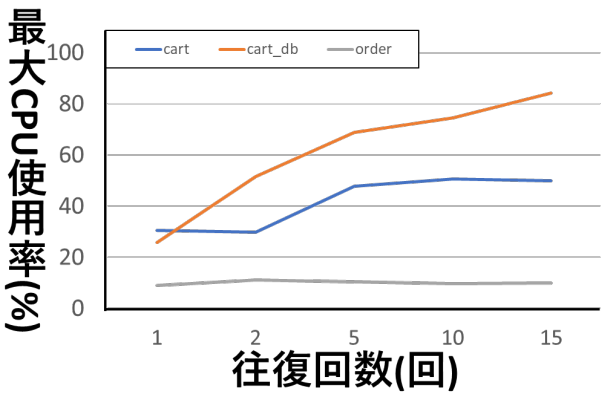

図 10　試験中の最大 CPU 使用率

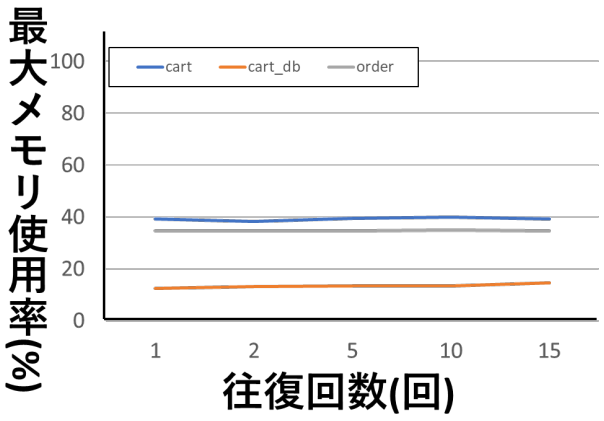

図 11　試験中の最大メモリ使用率

6. 議論

本稿では応答時間増加の原因の特定を提案とした．基礎実験では cart の応答時間が order の応答時間に影響するという結果がでた．提案では CPU・メモリ使用率で原因の特定を行おうと考えたが，データベースへの負荷がかかる場合ではディスク I/O の処理時間も含める必要がある．また，原因の候補を選定する際の閾値を 3s とした．しかし，実験を行う環境は全て同じ LAN 内であるため，外部から接続される際の経路が含まれていない．そのため，外部から内部への接続経路の往復分の接続時間も必要がある．

7. おわりに

本稿の課題は，マイクロサービス間の依存関係により応答時間増加の原因の特定が困難になることである．マイクロサービスの応答時間の増加により，他のマイクロサービスの応答時間も増加することに対する基礎実験は行った．提案では各マイクロサービスの応答時間から候補を選定し，CPU・メモリ使用率から原因を特定を行う．基礎実験の結果では，cart に対して大量の HTTP レスポンスを送信することで order の応答時間も増加した．また，最も多い HTTP リクエストを送信するの order の最大応答時間は最も少ない時に比べ 1395s 増加していた．また，カートのデータベースへの負荷が増加していた. そのため次回のレポートでは，原因の特定に使用する要素としてデータベースのディスク I/O の処理時間も候補に加えて評価を行う．

参考文献

[1] Saito, F., Yamashita, H., Sasaki, H. and Goto, M.: Machine Learning for Demand Prediction of Seasonal Second-hand Fashion Items Based on Prior and Fine-tuning Prediction Models, *2021 IEEE 12th International Workshop on Computational Intelligence and Applications (IWCIA)*, IEEE, pp. 1–6 (2021).

[2] Hagiwara, K., Ono, K. and Hatano, K.: Extracting Experiences Using Dependency Parsing on Japanese e-Commerce Websites, *2014 IIAI 3rd International Conference on Advanced Applied Informatics*, IEEE, pp. 813–818 (2014).

[3] Balalaie, A., Heydarnoori, A. and Jamshidi, P.: Microservices architecture enables devops: Migration to a cloud-native architecture, *Ieee Software*, Vol. 33, No. 3, pp. 42–52 (2016).

[4] Jamshidi, P., Pahl, C., Mendonça, N. C., Lewis, J. and Tilkov, S.: Microservices: The journey so far and challenges ahead, *IEEE Software*, Vol. 35, No. 3, pp. 24–35 (2018).

[5] Li, W., Lemieux, Y., Gao, J., Zhao, Z. and Han, Y.: Service mesh: Challenges, state of the art, and future research opportunities, *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, IEEE, pp. 122–1225 (2019).

[6] 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).

[7] Inagaki, T., Ueda, Y., Ohara, M., Choochotkaew, S., Amaral, M., Trent, S., Chiba, T. and Zhang, Q.: Detecting Layered Bottlenecks in Microservices, *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*, IEEE, pp. 385–396 (2022).

[8] Gan, Y., Liang, M., Dev, S., Lo, D. and Delimitrou, C.: Sage: practical and scalable ML-driven performance debugging in microservices, *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 135–151 (2021).

[9] Qiu, H., Banerjee, S. S., Jha, S., Kalbarczyk, Z. T. and Iyer, R. K.: {FIRM}: An intelligent fine-grained resource management framework for {SLO-Oriented} microservices, *14th USENIX symposium on operating systems design and implementation (OSDI 20)*, pp. 805–825 (2020).

[10] Gan, Y., Zhang, Y., Hu, K., Cheng, D., He, Y., Pancholi, M. and Delimitrou, C.: Seer: Leveraging big data to navigate the complexity of performance debugging in cloud microservices, *Proceedings of the twenty-fourth international conference on architectural support for programming languages and operating systems*, pp. 19–33 (2019).

[11] Kim, M., Sumbaly, R. and Shah, S.: Root cause detection in a service-oriented architecture, *ACM SIGMETRICS Performance Evaluation Review*, Vol. 41, No. 1, pp. 93–104 (2013).