「マルチメディア，分散，協調とモバイル
(DICOMO2021)シンポジウム」 令和3年6月-7月

マイクロサービスにおけるコンポーネントの依存関係を考慮した障害原因特定手法の提案

土手 貴裕[1] 近堂 徹[2] 前田 香織[1]
今村 光良[3] 日野 悠平[3] 高野 知佐[1]

概要： IT システムの拡張性や耐障害性などを向上させるための新たな設計手法として，マイクロサービスが注目されている．しかし，マイクロサービスで構築されたシステムは，ネットワーク上に分散配置されるコンポーネント数が増加する上，コンポーネントの接続関係も複雑化するため，障害発生時に障害の原因となるコンポーネントの特定が困難である．本論文では，コンポーネント間の呼び出しに依存関係があることと障害原因コンポーネントの処理がボトルネックとなることに着目し，これらを組み合わせて効率的に障害の原因の特定を行う手法を提案する．また，本手法で必要なパラメータであるメトリック（システムの状態を表す時系列データ）の収集間隔とシステムの定常状態を表すしきい値の生成に使う学習データ数を変化させたときの障害原因の特定に対する影響ついて，実験環境から収集したメトリックを用いて評価した結果を報告する．

A Proposal of How to Identify the Cause of Failure in Microservices Considering Component Dependency

TAKAHIRO DOTE[1] TOHRU KONDO[2] KAORI MAEDA[1]
MITSUYOSHI IMAMURA[3] YUHEI HINO[3] CHISA TAKANO[1]

1. はじめに

近年，IT システムの新たな設計手法としてマイクロサービス[1]が注目を集めている．IT システムを複数の独立したコンポーネント（IT システムを構成する小さなサービス）に分割し，ネットワークを介してそれらを疎結合な関係で動作させることで，コンポーネント単位での負荷状況に応じたスケーリングや高頻度なリリースなどが可能となる．しかしマイクロサービスで構築された IT システムは，従来の一枚岩（モノリシック）な設計と比べてネットワーク上に分散配置されるコンポーネント数が増加し，それらの接続関係も複雑化する．構成が複雑になると，障害発生時に管理者の経験に基づいて膨大な量のログや時系列データ形式のメトリック（コンポーネントの応答時間，リソース使用率など）からどのコンポーネントが障害原因であるかの切り分け作業を行うことになるため，障害原因の究明に時間を要するという課題がある．中には原因究明に数日を要した事例[2]もあるため，マイクロサービスにおいて迅速かつ正確に障害原因を特定できる手法が求められる．

そこで本研究では，マイクロサービスにおける障害発生時の迅速かつ高精度な障害原因の特定を目的として，マイクロサービスの構成をモデル化し，コンポーネント間の依存関係とメトリックの変化をもとに，効率的に障害原因を特定する手法について提案する．

2. 関連研究

マイクロサービスにおける障害原因の特定手法として，これまでに多くの提案がされている[3]-[7].

Zhou らは，正常時と障害注入時のトレースログを学習することで，マイクロサービスの潜在的なエラーの有無とその原因箇所を予測する手法を提案している[3]．トレースログには，ユーザのリクエストを処理する際のコンポーネント間の呼び出し関係やリクエスト処理時間など，リクエストの処理全体を追跡した情報が含まれる．そのため，ネットワーク的に構成が複雑なマイクロサービスにおいて，トレースログはどのコンポーネントが障害原因であるかの切り分けに有効である．しかしトレースログには CPU 使用率やメモリ使用量といったリソースに関する情報は含まないため，リソース系の障害（CPU Hog，メモリリークなど）に対する特定精度が低下する課題が挙げられている．

また L. Wu らは，最もユーザに近いコンポーネントへの影響が小さいコンポーネントが障害原因であるパターンにも対応することを目指した，各コンポーネントの応答時間とリソース使用率の振る舞いの相関を用いた手法を提案している[4]．しかし，メトリック値ではなく相関値を用いているため，応答時間とリソース使用率の相関が弱い障害に対する特定精度が低下する結果が得られている．

他にも，複数種類のメトリックからコンポーネント間の

1 広島市立大学大学院情報科学研究科
2 広島大学情報メディア教育研究センター
3 野村アセットマネジメント株式会社

依存関係を有効非巡回グラフ DAG（Directed Acyclic Graph）でモデル化し，重みとして扱うメトリックをコンポーネントの特性に応じて動的に変更する手法[5]や，HHMM（Hierarchical Hidden Markov Models）と相関分析により応答時間以外のメトリックと応答時間の関係をモデル化した手法[6]，収集する全メトリックのうち最も障害原因と関係のあるメトリックを定常性検定とクラスタリングにより抽出する手法[7]などが提案されている．これらはメトリックを用いてコンポーネント間の依存関係を推定し障害原因を特定するアプローチであるが，メトリックから「どのコンポーネントが API を介した呼び出し関係にあるか」という依存関係を捉えることは難しいため，誤った依存関係を用いた特定を行ってしまう可能性がある．

以上の関連研究を鑑みると，障害原因の特定においては以下の 3 点を考慮する必要がある．

- 通信障害とリソース障害両方に対する特定には，サービスメトリック（各コンポーネントの応答時間，受信リクエスト数など）やリソースメトリック（CPU，メモリなど）といった複数のメトリックを用いること．
- 複数のメトリック間の相関値だけではなく，個々のメトリック値の変化に着目した特定を行うこと．
- コンポーネント間の依存関係を考慮するために，それらの API 呼び出しの関係性を用いた特定を行うこと．

3. 提案手法

本研究で対象とするマイクロサービス環境と提案手法について述べる．

3.1 本研究が対象とするマイクロサービスシステム

本研究で対象とするのは一般的なマイクロサービスであるが，提案手法の有効性を示すために，まずは実環境で提案システムが動作することの確認が必要である．動作確認のために今回構築したマイクロサービスシステムおよび現在実装中の提案システムの構成を図 1 に示す．

本研究では複数の VM（Virtual Machine）で構成される Kubernetes[*1] クラスタ上で構築されるマイクロサービスを想定し，マイクロサービスのデモアプリケーションとして今回は Sock Shop[*2] を利用する．また提案システムには各種のメトリックの収集をするためにマイクロサービスを監視するフレームワークが必要である．今回はサービスメトリックとマイクロサービス間の呼び出し関係（トレースログ）の収集にサービスメッシュである Istio[*3] を,リソースメトリックの収集に Prometheus[*4] と cAdvisor[*5] を用いた．Istio

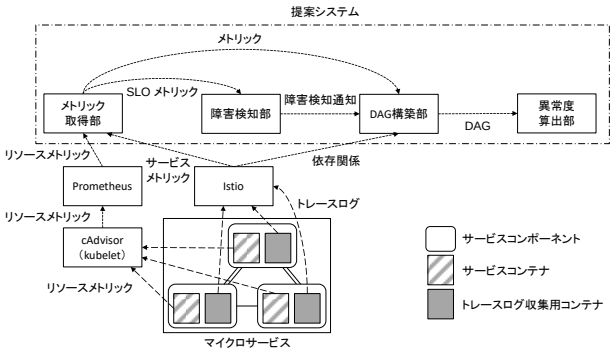


図 1: 構築したマイクロサービス環境および提案システムの概要

表 1: 今回構築した環境のハードウェア・ソフトウェア構成

項目	仕様
Master ノード 1 台 （クラスタ管理用）・ Worker ノード 7 台 （サービス提供用）	vCPU: 2 core memory: 4GB OS: CentOS 7.8.2003
Worker ノード 1 台 （メトリック・ログ収集用）	vCPU: 4 core memory: 8GB OS: CentOS 7.8.2003
Kubernetes	Version 1.19.2
Prometheus	Version 2.21.0
Istio	Version 1.7.4

による監視では，トレースログ収集用のサイドカーコンテナを各コンポーネント内に配置し，各コンポーネントに流入するリクエストを一旦サイドカーコンテナに経由させ，得られたトレースログを Istio に集約する．これにより，マイクロサービスで構築されたシステムに変更を加えることなくコンポーネント間の API 呼び出しにかかるサービスメトリック及び依存関係の情報を収集することができる．一方で Prometheus による監視では，Kubernetes のコンポーネント（kubelet）と統合した状態で各コンテナのリソースメトリックを取得する cAdvisor に対して取得メトリックをスクレイピングし，Prometheus に集約する．なお，今回利用したハードウェアとソフトウェアの構成については表 1 の通りである．

提案システムは Prometheus と Istio からメトリックを取得する「メトリック取得部」，SLO（Service Level Objective）メトリックに基づいて障害を検知する「障害検知部」，取得したメトリックと Istio から得られる依存関係を用いて DAG を構築する「DAG 構築部」，構築した DAG から異常度（障害原因であるか判断するための指標）を算出する「異常度算出部」の 4 つのパートから構成される．

[*1] Kubernetes: https://kubernetes.io/ja/
[*2] Sock Shop: https://microservices-demo.github.io/
[*3] Istio: https://istio.io/
[*4] Prometheus: https://prometheus.io/
[*5] cAdvisor: https://github.com/google/cadvisor

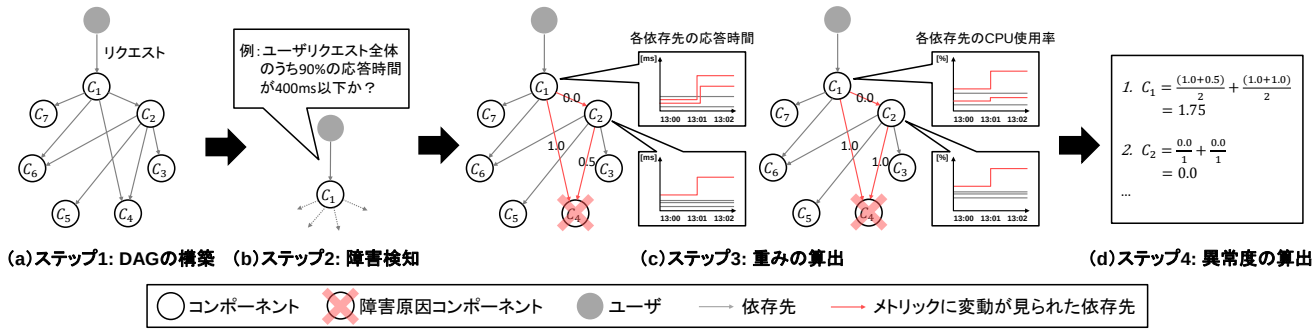


図 2: 提案手法による障害原因特定の流れ

3.2 障害原因の特定手法

3.2.1 提案手法

2 章で述べた考慮するべき点を踏まえて提案する障害原因の特定手法について述べる．提案手法は 4 つのステップで構成される．図 2 に提案手法の概要を示す．まず各コンポーネント間の依存関係を，トレースログを用いて DAG でモデル化する（a）．次に SLO に基づいて障害検知した後（b），メトリックごとに定常状態からの外れ具合を示す乖離度を計算し[0, 1]で正規化する．正規化した値で DAG のエッジへ重み付けを行い（c），最後に DAG の重みを用いて異常度をコンポーネントごとに算出し，降順で上位にあるものを障害原因とする（d）．以下に各ステップの詳細を述べる．

(a)ステップ 1: DAG の構築

最初のステップでは事前準備として，コンポーネント間の依存関係を DAG でモデル化する．依存関係はコンポーネント間の呼び出し関係から構築できるが，コンポーネント数が膨大で複雑だと手動で構築することは難しい．そのため，提案手法ではマイクロサービスに対して典型的なフローを模した負荷を生成し，サービスメッシュで収集したトレースログを用いて構築する．本手法で構築する DAG では，コンポーネントiからコンポーネントjへ向かう有効エッジは「コンポーネントiはコンポーネントjに依存している」ことを意味する．

(b)ステップ 2: 障害検知

次は，実運用中のステップとして，メトリックの収集間隔 $t_{interval}$ に基づいたメトリックの収集と監視を行う．対象とするマイクロサービスの最前段のコンポーネントのメトリックを用いて SLO（例：○○%のユーザリクエストの応答時間が××ミリ秒以下であるか？）に基づいた障害検知を行う．

(c)ステップ 3: エッジの重みの算出

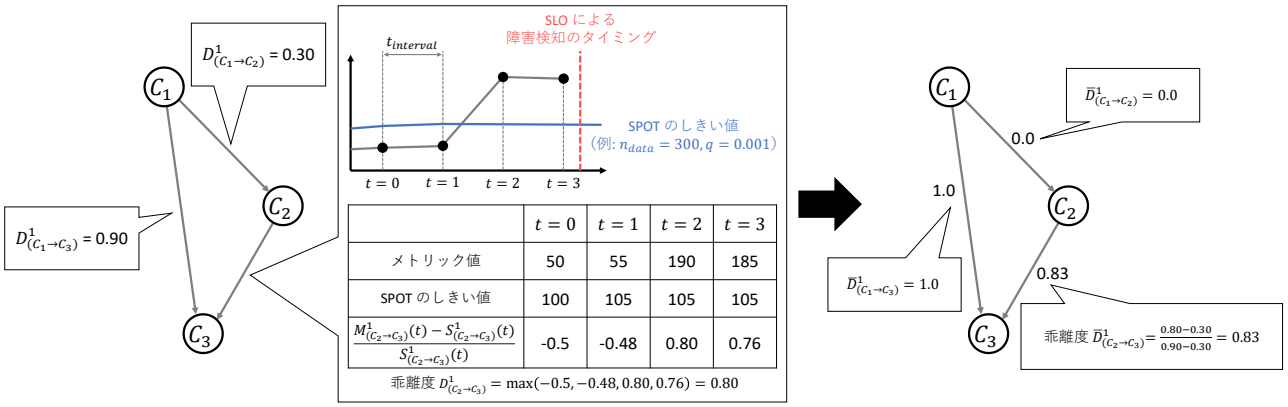
このステップでは障害検知後，事前に構築した DAG のエッジへ重み付けを行う．障害原因となるコンポーネントでは，障害発生時にそのコンポーネントにおける処理がボトルネックとなる，すなわちメトリックに定常状態と比較して大きな変化が見られると考えられる．そのため各エッジの重み付けでは，収集する各メトリックの定常状態とどれだけ異なるか（例えば，定常状態で CPU 使用率 10%から 80%になる）を表す「乖離度」をコンポーネントごとに算出する．このとき，CPU 使用率，応答時間などの異なる種類のメトリックから算出された重みを統合して乖離度を計算するため，各メトリックの分布（最大，最小）を合わせる正規化も行う．

定常状態を表すしきい値の算出には，過去のメトリック値の極値分布に基づいてしきい値を生成する SPOT[8]を用いる．SPOT では上限と下限の 2 種類のしきい値を生成できるが，提案手法ではボトルネックとなっているエッジ，すなわち高負荷となり値が増加しているメトリックに着目して重み付けを行うため，上限のしきい値のみを用いる．また，いずれの時刻においても SPOT のしきい値より小さい値を取るメトリックの場合，そのコンポーネントに向かうエッジについては異常がないものとし，重み付けは行わない．なお，提案手法において SPOT に必要なパラメータとして学習データ数 n_{data} とデータの振る舞いに異常が現れる確率 q の 2 つがあり，データを学習する前（すなわち障害検知する前）までに設定する必要がある．

メトリックにはエッジの特性（コンポーネント呼び出し時における応答時間，エラーレートなど）を表すものと，コンポーネントの特性（CPU 使用率，メモリ使用量など）を表すものがある．以下，それぞれにおけるエッジの重み付け方法を説明する．

(i)エッジ特性からエッジの重みを算出

DAG において，コンポーネントの集合を $V = 1, 2, \ldots, |V|$（$|V|$：コンポーネント数），エッジの集合を E としたとき，コンポーネント $i \in V$ からコンポーネント $j \in V$ へ向かう有向エッジを $(i \to j) \in E$ と表す．メトリック $m = 1, 2, \ldots, |M|$（$|M|$:メトリック数）に対し，時刻 $t = 0, 1, \ldots, T$ のエッジ $(i \to j)$ のメトリック値を $M^m_{(i \to j)}(t)$，SPOT のしきい値を $S^m_{(i \to j)}(t)$ とおくと，メトリックの観測区間[0, T]にお



	$t=0$	$t=1$	$t=2$	$t=3$
メトリック値	50	55	190	185
SPOT のしきい値	100	105	105	105
$\frac{M^1_{(C_2 \to C_3)}(t) - S^1_{(C_2 \to C_3)}(t)}{S^1_{(C_2 \to C_3)}(t)}$	-0.5	-0.48	0.80	0.76

図 3: エッジ特性（メトリック $m = 1$）からエッジの重みを算出する例

けるエッジ $(i \to j)$ の乖離度 $D^m_{(i \to j)}$ は式(1)で計算される．

$$D^m_{(i \to j)} = \max_{t \in [0,T]} \frac{M^m_{(i \to j)}(t) - S^m_{(i \to j)}(t)}{S^m_{(i \to j)}(t)} \quad (1)$$

また，エッジの集合 E において，$D^m_{(i \to j)}$ の最大値を $D^m_{max} = \max_{(i \to j) \in E} D^m_{(i \to j)}$，最小値を $D^m_{min} = \min_{(i \to j) \in E} D^m_{(i \to j)}$ とする．メトリック m におけるコンポーネント j の入エッジの集合を ∂j_m としたとき，入エッジの数は $|\partial j_m|$ で表される．ただし，エッジ $(i \to j)$ の乖離度 $D^m_{(i \to j)}$ が負の場合，そのエッジのメトリックは常に SPOT のしきい値よりも小さく障害に関与しないと考え，∂j_m はエッジ $(i \to j)$ を含まないものとする．[0, 1]で正規化されたメトリック m，エッジ $(i \to j)$ の乖離度 $\bar{D}^m_{(i \to j)}$ は式(2)で表される．

$$\bar{D}^m_{(i \to j)} = \begin{cases} 1 & (\sum_{j \in V} |\partial j_m| == 1) \\ \frac{D^m_{(i \to j)} - D^m_{min}}{D^m_{max} - D^m_{min}} & (otherwise) \end{cases} \quad (2)$$

式(2)の 1 行目は，DAG の中で重みがついているエッジ $(i \to j)$ が 1 本だけであることを意味しており，障害へ関与しているのはこのエッジのみであるとみなし，その乖離度を最大値の 1 としている．本手法におけるエッジ特性を用いたエッジの重み付けでは，式(2)により算出した $\bar{D}^m_{(i \to j)}$ をエッジの重みとする．算出例として，全体のエッジ数が 3 本の DAG においてエッジ全てのメトリック $m = 1$ の値が SPOT のしきい値より高くなった場合の流れを図 3 に示す．

(ii)コンポーネント特性からエッジの重みを算出

コンポーネント特性を表すメトリック m において，コンポーネント $i \in V$ のメトリック値を $M^m_i(t)$，SPOT のしきい値を $S^m_i(t)$ とおくと，メトリックの観測区間$[0, T]$におけるメトリック m，コンポーネント i の乖離度 D^m_i は式(3)により算出される．

$$D^m_i = \max_{t \in [0,T]} \frac{M^m_i(t) - S^m_i(t)}{S^m_i(t)} \quad (3)$$

式(3)は式(1)の添字 $(i \to j)$ が i になっただけであり，計算方法は同じである．式(2)と同様に[0,1]で正規化し，正規化された乖離度を $\bar{D}^m_i$ とする．

乖離度 $\bar{D}^m_i$ はコンポーネント特性を表す値であり，(i)で述べたエッジ特性から得られた重みと統合するためには，乖離度 $\bar{D}^m_i$ をエッジの重み $\bar{D}^m_{(i \to j)}$ に変換する必要がある．本手法では単純に，コンポーネント特性は入エッジに等しく還元されると考え，以下のように算出する．

$$\bar{D}^m_{(i \to j)} = \bar{D}^m_j \quad (4)$$

(d)ステップ 4: 異常度の算出と障害原因の特定

最後のステップは重み付けした DAG を用いて，障害原因であるかどうかを表す異常度をコンポーネントごとに算出し，障害原因を特定するステップである．

コンポーネント j の異常度 α_j は式(5)で表される．

$$\alpha_j = \sum_{m=1}^{|M|} \{ \frac{1}{|\partial j_m|} \sum_{(i \to j) \in \partial i_m} \bar{D}^m_{(i \to j)} \} \quad (5)$$

式(5)の中括弧の中は，各コンポーネントの入エッジの重みの合計を，入エッジの数で按分しており，エッジ 1 本あたりの障害への関与の強さを示す．

システム全体に及ぶような障害に対して，どの程度コンポーネントが障害へ関与しているかを直接測ることは難しい．例えば，あるコンポーネントの応答時間が定常状態より長いとしても，他のコンポーネントで発生した障害の影響による可能性も考えられる．このような障害に対して，本手法ではコンポーネントが周囲のコンポーネントへ与える影響の度合い（ここではエッジ 1 本あたりの障害への関与の強さ）を利用してコンポーネントの異常度を算出することで，各コンポーネントがどの程度障害原因として考え

られるかを推定する．最終的にはコンポーネントごとに算出された α_j を降順に並び替えたときの上位のコンポーネントが障害原因と特定する．

4. 提案手法のパラメータ決定に関する実験

提案手法では代表的なメトリックとしてコンポーネントの応答時間とコンポーネント内のコンテナの CPU 使用率の 2 種類を選択（$|M| = 2$）し，これらについて 3 章で述べた計算を行う．ステップ 2 と 3 で必要なパラメータ，メトリックの収集間隔 $t_{interval}$ と SPOT の学習データ数 n_{data}，メトリックの振る舞いに異常が現れる確率 q の 3 点を事前に決めなければならない．このうち $t_{interval}$ と n_{data} の決定のため，これらの値を変更した際に提案手法による障害特定にどのような影響があるかを実験により調査した．残る qについては学習データの振る舞いによって適する値が変わるが，提案手法では学習データ内での異常は全く無いものと仮定し，小さい値（$q = 0.001$）を設定するものとする．

4.1 実験内容

本研究における評価指標としては，障害やその予兆に対する検知がどれだけ漏れなく行えるかを表す検知精度とその原因箇所の特定がどれだけ正確に行えるかを表す特定精度，およびそれらにかかる時間（検知するまでの時間・特定するまでの時間）の 4 つが挙げられるが，提案手法は原因特定の部分を主としており，また提案システムも現在実装を進めている段階である．そのため，今回は提案手法に則り障害原因を特定したときの特定精度のみについて，今回構築した環境から得られるメトリックを使って確認した．

今回の実験でデモアプリケーションとして利用した Sock Shop のアーキテクチャは図 4 に示す通りで，計 7 個のコンポーネントで構成される．今回構築した環境の Sock Shop に対してユーザからのリクエストを模した負荷を生成しつつ Sock Shop を構成するコンポーネントである User コンポーネントと Shipping コンポーネントに障害を注入したときのメトリックを用いて，提案手法により障害原因を特定した場合の特定精度を机上の計算により検証した．ここで，特定精度は障害原因として推定したコンポーネントの上位 k 番目以内に障害原因が含まれる確率 PR@k ($0 \leq \text{PR@k} \leq 1$) の値で表す．負荷生成には Locust[*6] を利用し，ユーザがサイトのトップページを閲覧してから商品カタログを取得し，商品を選び注文するまでのシナリオを典型的なフローとして負荷生成に適用した．コンテナの CPU 負荷が 100%となる障害の注入に stress-ng[*7]，他コンテナとの通信遅延が増加する障害の注入に tc（Traffic Control）[*8] を使用した．メトリックの収集間隔 $t_{interval}$ については学習データ数 n_{data} を 300 個に固定した上で 1 秒，3 秒，5 秒と変化させたとき，また学習データ数 n_{data} についてはメトリックの収集間隔 $t_{interval}$ を 5 秒に固定した上で 100 個，300 個，500 個と変化させた．今回は実験時間の都合上，いずれのパターンも 3 回検証した値を実験結果として採用した．

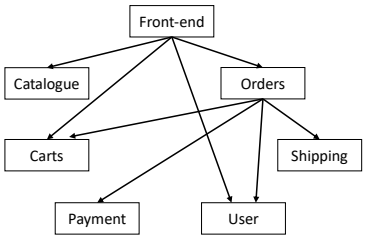


図 4: Sock Shop のアーキテクチャ

4.2 結果と考察

表 2 に実験結果を示す．表 2 の見方として，例えば PR@1 の値が 0.67 である場合，そのパターンにおいて障害原因と推定した上位 1 番目以内に障害原因が含まれる確率が約 67%（今回は 3 回検証した結果であるため 3 回中 2 回）であることを意味する．表 2 より，どのパターンにおいても PR@1 は概ね 1.0，PR@2 は常に 1.0 と安定しているため，今回検証した中ではメトリックの収集間隔および学習データ数はどの値に設定しても特定精度への大きな影響は無いことが確認できた．

一方で表 2 からわかることとして，CPU 負荷に対しては安定して $\text{PR@1} = 1.0$ と高精度で特定できるものの，通信遅延に対しては $\text{PR@1} = 0.67$ となる結果が散見され，特定精度が低下する傾向にあることが見て取れる．このときに使用したメトリックおよび DAG を確認したところ，CPU 負荷においては CPU 使用率の DAG と応答時間の DAG の両方で障害原因コンポーネントへの有向エッジのみに大きい重みが付いていたが，対して通信遅延においては応答時間の DAG では障害原因コンポーネントへの有効エッジのみに大きい重みが付いていたものの，CPU 使用率の DAG では他のコンポーネントへの有効エッジに更に大きい重みが付いていた．提案手法では定常状態から最も乖離したメトリック値を使って重みの計算を行うため，定常状態から乖離したメトリック値が外れ値のみである場合にその値が得られた有向エッジのみに大きい重みが付くことになる．これが特定精度の低下する主な原因となっている．

今回は 2 つのコンポーネントに対してのみ実験を行ったため他のコンポーネントについても同様の結果が得られるかの確認は必要だが，今回の実験結果による改善点として外れ値も考慮した重みの計算を行うことが考えられる．そ

[*6] Locust: https://locust.io/
[*7] stress-ng: https://kernel.ubuntu.com/~cking/stress-ng/
[*8] tc: https://linux.die.net/man/8/tc

表 2: 各パラメータを変化させたときの特定精度

障害内容	設定パラメータ	パラメータ値	障害原因コンポーネント特定精度			
			User		Shipping	
			PR@1	PR@2	PR@1	PR@2
CPU 負荷	$t_{interval}$ (n_{data} = 300 固定)	1	1.0	1.0	1.0	1.0
		3	1.0	1.0	1.0	1.0
		5	1.0	1.0	1.0	1.0
	n_{data} ($t_{interval}$ = 5 固定)	100	1.0	1.0	1.0	1.0
		300	1.0	1.0	1.0	1.0
		500	1.0	1.0	1.0	1.0
通信遅延	$t_{interval}$ (n_{data} = 300 固定)	1	0.67	1.0	1.0	1.0
		3	1.0	1.0	1.0	1.0
		5	1.0	1.0	0.67	1.0
	n_{data} ($t_{interval}$ = 5 固定)	100	1.0	1.0	1.0	1.0
		300	0.67	1.0	1.0	1.0
		500	1.0	1.0	1.0	1.0

の方法としては，収集したメトリックからの外れ値の除去や, DAG のエッジの重みを算出する過程で正規化ではなく標準化を行うなどが挙げられるが，具体的な対処方法については今後の課題とする．

5. まとめと今後の展望

マイクロサービスにおけるコンポーネント間の依存関係とメトリックの定常状態からの変化に基づいた障害原因の特定手法を提案した．また提案手法においてあらかじめ決める必要があるパラメータを変更したときの特定精度への影響についても調査した．

現在, 提案手法に基づいたシステムの実装を進めている．今後は他のコンポーネントに対しても本手法が有効であるかの確認とアルゴリズムの改良を行った上で，実装後に障害原因となるコンポーネントの特定に要する時間および精度について定量的評価を行い，既存手法と比較することで提案手法の有効性を検証する予定である．

謝辞 本研究の一部は JSPS 科研費 21K11846, 19K11929, 21H03432, 及び 18K11271 の支援を受けて実施しました．

参考文献

[1] Lewis, J. and Fowler, M.: Microservices a definition of this new architectural term, available from <https://martinfowler.com/articles/microservices.html> (accessed 2021-05-07).

[2] Zhou, X., Peng, X., Xie, T., et al.: Fault Analysis and Debugging of Microservice Systems: Industrial Survey, Benchmark System, and Empirical Study, *IEEE Transactions on Software Engineering*, vol.47, no.2, pp.243-260 (2021).

[3] Zhou, X., Peng, X., Xie, T., et al.: Latent Error Prediction and Fault Localization for Microservice Applications by Learning from System Trace Logs, Proc. *European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2019)*, Association for Computing Machinery, pp.683–694 (2019).

[4] Wu, L., Tordsson, J., Elmroth, E., and Kao, O.: MicroRCA: Root Cause Localization of Performance Issues in Microservices, Proc. *2020 IEEE/IFIP Network Operations and Management Symposium (NOMS 2020)*, pp. 1-9 (2020).

[5] Ma, M., Lin, W., Pan, D., and Wang, P.: Self-Adaptive Root Cause Diagnosis for Large-Scale Microservice Architecture, *IEEE Transactions on Services Computing* (2020).

[6] Samir, A., and Pahl, C.: DLA: Detecting and Localizing Anomalies in Containerized Microservice Architectures Using Markov Models, Proc. *2019 7th International Conference on Future Internet of Things and Cloud (FiCloud)*, pp.205-213 (2019).

[7] 坪内佑樹, 鶴田博文, 古川雅大：TSifter: マイクロサービスにおける性能異常の迅速な診断に向いた時系列データの次元削減手法, インターネットと運用技術シンポジウム論文集, Vol. 2020, pp.9–16 (2020).

[8] Siffer, A., Fouque, P.-A., Termier, A. and Largouet, C.: Anomaly detection in streams with extreme value theory, Proc. *23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp.1067-1075 (2017).