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

坪内 佑樹[†1,†2,a),b)]　鶴田 博文[†1,c)]　古川 雅大[†3,d)]

概要：Web サービスのソフトウェア規模は，長年の機能開発により日々増大しており，ソフトウェア開発者によるソフトウェアの変更が難しくなっている．そこで，変更を容易にするために，一枚岩のアプリケーションを分解して分散させるマイクロサービスアーキテクチャが普及している．しかし，マイクロサービス化によりシステムの構成要素数が増大するにつれて，システムの性能を示す時系列データ形式の指標であるメトリックの個数が増大する．そのため，システムの性能に異常が発生したときに，網羅的にメトリックを目視できず，システム管理者がその異常の原因を診断することが難しくなっている．先行手法では，複数の構成要素を横断したメトリック間の因果関係を推定することにより，システム内の異常の伝播経路を推論する．しかし，診断に利用できるメトリックの個数は限定されるため，より原因に近いメトリックが推論結果から除外される可能性がある．本論文では，性能異常の診断に有用なメトリックを網羅的に抽出するために，観測されたすべてのメトリックの次元数を削減する手法である TSifter を提案する．TSifter は，定常性を有するメトリックを除外したのちに，類似の形状をとるメトリックをクラスタリングすることにより，異常の特徴を強く表すメトリックのみを抽出する．本手法により，メトリック数が膨大であっても，その異常の診断に適した有用なメトリックを都度抽出できる．マイクロサービスのテストベッド環境に故障を注入する実験の結果，TSifter は，ベースラインとなる手法に対して，正確性と次元削減率の指標では同等程度の性能を有しながらも，270 倍以上高速に動作することを確認した．

TSifter: Dimention Reduction of Time Series Data for Quick Diagnosis of Performance Issues in Microservices

Yuuki Tsubouchi[†1,†2,a),b)]　Hirofumi Tsuruta[†1,c)]　Masahiro Furukawa[†3,d)]

Abstract: The scale of Web services is growing day by day due to the development of functions over the years, making it difficult for software developers to change the software. Therefore, microservices architecture that decomposes and distributes monolithic applications has become widespread to facilitate software changes. However, as the number of system components increases due to the adoption of the microservices architecture, the number of metrics, which are indicators of system performance in time-series format, increases. This makes it difficult for system administrators to diagnose the cause of the anomalies because of the lack of comprehensive visibility of metrics when the system performance issues occur. In the prior methods, the propagation path of anomalies in the system is inferred by discovering causal structure between metrics across multiple components. However, since the number of metrics used for diagnosis is limited, metrics closer to the cause may be excluded from the inference results. In this paper, we propose TSifter, a method to reduce the dimensionality of all observed metrics to extract useful metrics for the diagnosis of performance issues comprehensively. TSifter extracts only metrics that strongly represent the characteristics of anomalies by clustering metrics based on their shape similarity after excluding stationary metrics. With this method, even if the number of metrics is large, useful metrics suitable for diagnosing anomalies can be extracted each time. Experiments injecting failures into a microservices testbed environment confirmed that TSifter performed more than 270 times faster than the baseline method while having comparable performance in terms of accuracy and dimension reduction rate.

1. はじめに

ソーシャルネットワーク，メディア，電子商取引，IoT（Internet of Things）などを実現する Web サービスは，利用者のサービス利用体験を向上させるために，高い信頼性を維持した上で，多数の機能を追加していく必要がある．ソフトウェア開発者が多数の機能を日々追加することにより，アプリケーションのコード規模が増大するため，変更を加えるときに問題があった場合の影響がアプリケーション全体に及ぶようになる．変更の影響範囲を小さくするために，昨今のクラウド上に展開される Web アプリケーションのアーキテクチャは，巨大な一枚岩のアプリケーションを複数の異なる小さなサービスに分解した上で，疎結合な状態にして各サービスが協調して動作するマイクロサービスアーキテクチャ [1] へと変遷している．

Web サービスの高信頼化のためには，サービスの性能に異常が発生したときに，異常の根本原因を迅速に特定しなければならない．しかし，マイクロサービスは，分散された構成要素を多数のサーバホスト上に展開するため，構成要素間のネットワーク通信関係は従来の構成よりも複雑となる．そのため，システムに異常が発生したときに，構成要素間の異常が伝搬するため，異常の伝搬経路も複雑化する．マイクロサービスにおけるサービス数は数百から数千にまで増大しているケース [2] もあるため，それらのサービスの性質や関係性をシステム管理者が記憶することは難しい．さらに，マイクロサービスでは，個々のサービスが頻繁に更新されることから，異常の発生確率と負荷傾向が変化する頻度を増大させる．加えて，構成要素数の増大により，システムの性能やリソース消費量を定量的に示す時系列データ形式の指標であるメトリックの個数が増大する [3]．このような依存関係の複雑性，ソフトウェアの動的な変更，および，メトリック数の増大により，システム管理者にとってシステムを認知するための負荷が増大するため，異常の原因を診断するために時間を要するようになる．したがって，マイクロサービス環境にて，システム管理者が異常の原因を迅速に特定できる手法が必要となる．

先行手法では，まず，メトリック以外のデータソースである，各構成要素が出力するテキストログを用いたログベースのアプローチ [4] と，各構成要素間の呼び出し関係や応答速度を追跡する実行経路トレースベースのアプローチ [5–9] がある．しかし，すべての異常の動作が記録されるわけではなく，計測のためにアプリケーションのソースコードを修正する手間があることから，情報の網羅性と適用性に課題がある．次に，複数の構成要素を横断したメトリック間の因果関係を推定することにより，システム内の異常の伝播経路を自動で推論するメトリックベースのアプローチ [10–16] がある．しかし，このアプローチは，診断に利用するメトリックをシステム管理者が一つあるいは複数個に指定しなければならないため，より原因に近いメトリックが探索結果から除外される可能性がある．そのため，診断に有用な情報を迅速に得るためには，できる限り多くの関連するメトリックを高速に解析する必要がある．

本論文では，マイクロサービスにおいて，異常の伝播経路を自動で推論するための基盤を提供することを目的に，異常発生時に観測されたすべてのメトリックから診断に有用なメトリックを高速に抽出可能な次元削減手法である TSifter を提案する．TSifter は，まず，各メトリックに対して，時系列データの定常性を検定し，定常性を有するメトリックを事前に除去することにより，異常発生前後で傾向が変化したメトリックを抽出する．次に，時系列グラフとして類似の形状をとるメトリックをクラスタリングし，各クラスタから代表メトリックを選出することにより，同等の傾向をもつメトリックを集約する．本手法により，異常発生前後で傾向が変化したメトリックは，そうでないものと比較し，原因を示すメトリックである蓋然性が高いため，原因の診断に有用となる．さらに，クラスタリング処理はメトリック数が増加するにつれて，計算量が増加するため，メトリックの事前除去により，クラスタリング処理を高速化できる．また，事前除去とクラスタリングの 2 つの異なる次元削減法を適用することにより，単一手法と比較して，高い削減性能を得られる．

TSifter を評価するために，コンテナ型仮想化環境を管理するための Kubernetes [17] クラスタ上に構築したマイクロサービスのテストベッド環境に故障を注入する実験を行った．実験の結果，TSifter は，ベースラインとなる手法（以降，ベースライン手法とする）に対して，原因であるメトリックが除外されていないことを示す正確性，次元削減率，およびメトリック数と CPU コア数に対するそれぞれの実行時間のスケーラビリティでは，同等程度の性能を有しながらも，270 倍以上高速に動作することを確認した．

本論文を次のように構成する．2 章では，関連研究を述べる．3 章では，提案するメトリックの次元削減手法を説明する．4 章では，提案手法の有効性を確認するための実験を示し，実験結果を考察する．5 章では，本論文をまとめ，今後の展望を述べる．

2. 分散システムにおける原因診断手法

クラウド上の分散システムにおける一般的なシステム障

†1 さくらインターネット株式会社 さくらインターネット研究所 SAKURA internet Research Center, SAKURA internet Inc., Ofukatyo, Kitaku, Osaka 530-0011 Japan
†2 京都大学情報学研究科, Graduate School of Infomatics, Kyoto University, Kyoto 606–8501, Japan
†3 株式会社はてな Hatena Co., Ltd.
a) y-tsubouchi@sakura.ad.jp
b) y-tsubouchi@net.ist.i.kyoto-u.ac.jp
c) hi-tsuruta@sakura.ad.jp
d) masayoshi@hatena.ne.jp

害対応手順は，利用者へ悪影響のある性能異常を検知したのちに，システムの観察と診断により根本原因を特定し，原因を除去することにより異常から回復させる，という流れになる．システム管理者は各種メトリックをデータベースに収集し，サービスの主要メトリックを表示するダッシュボードを作成しておき，性能異常を検知するとダッシュボードを利用して，システムの状況を観察する [18]．一般的に選択される主要メトリックとして，CPU 使用時間，メモリ使用量，ディスク I/O，ネットワーク I/O などの物理資源の利用を示すメトリックと，ソフトウェアレベルの処理時間やスループット，エラー数などのメトリックがある．しかし，これらの主要メトリック以外に，例えば，ロック，ネットワーク接続上限数，OS のファイルディスクリプタ数などの論理資源の使用を示すメトリックがある．論理資源が枯渇すると，性能異常が発生するため，主要メトリック以外のメトリックが診断のために有用となるケースがあるといえる．より原因に近い有用なメトリックがダッシュボードにない場合，メトリックを網羅的に調査する必要がある．

マイクロサービス構成をとると，システム全体のメトリック数が増大するため，システム管理者が異常を診断することがより困難となる．そこで，性能異常が発生したときの原因診断手法として，メトリック以外のデータソースを利用するアプローチとメトリックを自動解析するアプローチがこれまでに提案されている．

各構成要素がデバッグのために出力するテキストログを用いたログベースのアプローチでは，ログを分析して，システムに発生した問題を特定する [4]．テキスト情報であるログは数値情報であるメトリックと比較し，情報量が多いため，デバッグのために有用な情報を提供する．しかし，すべての異常の動作がログに記録されているわけではないため，実際にはログ以外の複数のデータソースから問題を診断しなければならない．

構成要素間の実行パスを追跡する実行経路トレースベースのアプローチは，実行経路に沿った応答時間の偏差を分析することにより，問題領域や潜在的なボトルネックを特定するために有用な情報を可視化する [5–9]．マイクロサービスにこれらのアプローチを適用すると，サービス間でやりとりされるアプリケーションレベルのリクエスト発行経路やアプリケーションのソースコード内の処理経路と各経路の実行時間を把握できる．そのため，アプリケーションのソースコード内の問題箇所を特定しやすい．しかし，各サービスのアプリケーションコードに計測のための処理を追加することになるため，アプリケーション開発者の作業量が大きい．

メトリックベースの自動解析アプローチは，複数の構成要素を横断したメトリック間の因果関係を推定することにより，システム内の異常の伝播経路を自動で推論する [10–16]．MicroRCA [12] 以外の手法は，メトリックをノードとして因果関係グラフを構築した上で，システム内の最前段の構成要素を起点に因果の伝播経路を探索する．Qiu らの研究 [11]，Microscope [15]，および，CauseInfer [16] は，事前に選択した単一または複数の種類のメトリックとサービス間の呼び出し関係や複数のシステム階層の従属関係を示すシステムトポロジ情報を事前情報として利用する．AutoMAP [10]，MS-Rank [13]，および，CloudRanger [14] は，システムトポロジー情報を必要とせず，複数種類のメトリック情報のみを用いて，因果関係グラフを構築する．MicroRCA [12] は，前述の関連研究が利用者に近い最前段のサービスのメトリックを起点として因果を追跡するため，最前段のサービスへの因果伝搬の影響が小さいサービスが原因であるときに精度が低い前提があることに着目し，マイクロサービス内のサービス間の応答速度の情報を利用して，精度を向上させている．これらのメトリックベースのアプローチは，メトリックの収集のためにアプリケーションコードに修正を加える必要がないため，システムへの適用性が高い．しかし，いずれの手法もシステム管理者が単一または複数の種類のメトリックを事前に選択しておく必要がある．

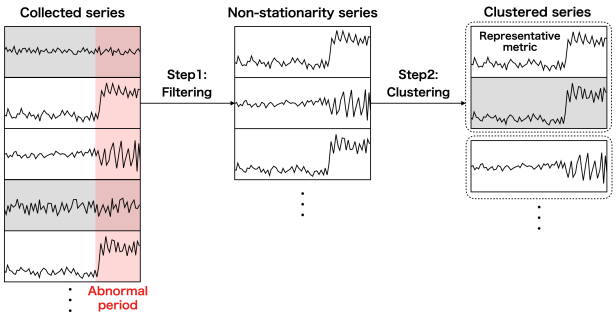


図 1: TSifter の概要図

Sieve [19] は，迅速な原因診断を目的とせず，オートスケールの指標などに恒常的に利用するための代表メトリックを抽出するために，メトリックの次元削減手法を提案している．Sieve は，事前の負荷テストにより，観測されたメトリックを構成要素ごとに時系列クラスタリングにより次元削減した上で，クラスタ内の代表となるメトリックを選択する．しかし，Sieve を異常発生時の迅速な原因診断に応用する場合，事前の負荷テストで顕在化できなかった異常のケースには対応できない．

3. 性能異常の原因診断に向いたメトリックの次元削減手法

情報の網羅性と実システムへの適用性の高さに着目すると，メトリックベースのアプローチを採用した上で，論理資源の枯渇などの潜在的な異常原因に対応する必要がある．そのために，異常の発生前後に観測された全てのメトリックを対象とした上で，システム管理者の迅速な診断のために有用なメトリックのみを抽出し，かつ高速に動作す

る原因診断システムを実現することが望ましい．本論文では，そのような原因診断システムに組み込むことを目指した基盤として，原因となるメトリックを除去させない正確性と時系列データの高い次元削減率を両立した上で，高速性をもつメトリック次元削減手法 TSifter を提案する．

3.1 TSifter の概要

図 1 は TSifter の概要図である．TSifter は，異なるアルゴリズムを適用してメトリックの次元削減率を高めるために，2 段階のステップでメトリックを次元削減する．1 段階目は，原因となるメトリックを除去させない範囲で，次元削減率を高めるために，異常発生前後で 収集した全てのメトリックの定常性を検定し，定常性を有するメトリックを除外する．2 段階目は，1 段階目で残った非定常のメトリックに対して，データの形状の類似性に着目した距離尺度を用いてクラスタリングを適用する．その後，さらなる次元削減率向上のために，各クラスタからそのクラスタを代表するメトリックを選定する．クラスタリングはメトリック数の増加に対して，非類似度を表す距離の計算回数が増えるため計算量が増加する．TSifter は，クラスタリングを実行する前に定常性の検定によりメトリックを除外することで，クラスタリング対象のメトリック数を減らすことができ，クラスタリングを高速に実行できる．以下，次元削減の各ステップについて詳細に示す．

3.2 ステップ 1: 定常性の検定による事前除去

異常に関連するメトリックは，異常発生後にメトリックの値が増加，減少，不安定化するなど，異常発生前後でデータの傾向が変化するはずである．そのため，異常に関連するメトリックは，データの平均および分散が時間によらず一定，かつ自己共分散が時間差のみに依存する性質である定常性を満たさず，非定常性を示す．この洞察に基づき，TSifter では，異常に関連しないメトリックを除去するために，定常性の検定を用いてメトリックが定常性を有するかどうかを判別し，定常性を有するメトリックを除去する．

TSifter は，収集した全メトリックに対して，定常性の検定を行い，除外するかどうかを決定する．メトリックの定常性の検定には，広く用いられている検定手法の一つである Augmented Dickey-Fuller（ADF）検定 [20] を採用する．ADF 検定は，あるメトリックから算出した p 値が有意水準を下回り帰無仮説が棄却された場合，対立仮説である「データが定常性」が採択され，メトリックは定常性を有すると判定できる．TSifter は，ADF 検定により定常性を有すると判定されたメトリックを全て除去する．

3.3 ステップ 2: 形状の類似性に基づく階層的クラスタリング

マイクロサービスにおける各サービスで収集するメトリックの中には，メトリック同士が互いに強く相関しており，時間の推移に対して同様の傾向で変動するものが存在する．例えば，単位時間あたりのネットワーク送信バイト数とネットワーク送信パケット数は，値の単位は異なるが，時間に対して概ね同様の変動傾向を示す．このようなメトリック群は，異常の診断において冗長な情報であり，メトリック群の中からどれか一つのみを抽出すれば十分である．この考えに基づき，TSifter ではクラスタリングにより同様の変動傾向をもつメトリックを集約し，その中から代表メトリックを一つ抽出して他のメトリックを除外する．代表メトリックは，クラスタ全体の特徴をよく反映するために，クラスタ内のメトリックで，それ以外のメトリックとの距離の総和が最小になるメトリックである medoid を選出する．また，TSifter は，マイクロサービスにおけるサービス単位でメトリックを集約してクラスタリングを実行する．

異常の診断のための有用な情報をシステム管理者に提供するためには，同様の変動傾向をもつ冗長なメトリックをなるべく多く集約，除外することでメトリックの削減数を高める必要がある．クラスタリングにおいて，どのようなデータを類似性が高いと判断し，同一クラスタに集約するかは，データ間の非類似度を表す距離の定義に依存する．時系列データであるメトリックの変動傾向の類似性を捉えるためには，メトリックの時系列変化の形状に着目することが有効である．時系列変化の形状の類似性を考慮した距離の定義は，メトリック値の軸方向へのスケールと時間軸方向へのシフトに対する不変性を満たす必要がある．TSifter では，メトリック値の軸方向へのスケールに対する不変性を満たすために，メトリックを平均 0，分散 1 に標準化する．時間軸方向へのシフトに対する不変性を満たすために，標準化したメトリック間の距離を shape-based distance（SBD）[21] に基づき算出する．SBD は，二つの時系列データ $\vec{x}$ と $\vec{y}$ が与えられたとき，$\vec{x}$ を $\vec{y}$ に対してスライドさせながら規格化した相互相関 NCC_w が最大となる位置 w をとり，以下の式に基づき算出される．

$$SBD(\vec{x}, \vec{y}) = 1 - max_w(NCC_w(\vec{x}, \vec{y})) \quad (1)$$

SBD は 0 から 2 の値をとり，0 はデータ間の形状が完全に一致することを意味している．

TSifter は，マイクロサービスの性能異常時の原因診断に活用することを想定しているため，クラスタリングの処理には高速性が要求される．メトリックをクラスタリングする際に，いくつのクラスタに分けるのが適当であるかは事前に決定されておらず，その時々のメトリックの変動傾向によって変わりうる．このようにクラスタ数が事前に決定されていない場合には，最適なクラスタ数を決定するための処理が必要である．k-means 法 [22] を始めとした非階層的クラスタリングは，最適なクラスタ数を決定するためにクラスタ数を変化させて繰り返しクラスタリングを実行

表 1: テストベッドにおけるハードウェアとソフトウェア構成

項目	仕様	
Worker ノード (サービス用途)	CPU	Intel Xeon CPU 2.20GHz
	vCPU	2 core
	Memory	4 GiB
	Type	e2-medium
	OS	Container-Optimized OS 77
Worker ノード (管理用途)	CPU	Intel Xeon CPU 2.20GHz
	vCPU	2 core
	Memory	8 GiB
	Type	e2-standard-2
	OS	Container-Optimized OS 77
解析用サーバ	CPU	Intel Xeon CPU 3.10GHz
	vCPU	8 core
	Memory	32 GiB
	Type	c2-standard-8
	OS	Debian 11
	Python	v3.8.2
Kubernetes	Version	1.16.13-gke.1
Prometheus	Version	2.20.0

し，情報量基準などに基づき最適なクラスタ数を決定する．一方，階層的クラスタリングは，1 回のクラスタリング実行後に，最適なクラスタ数を決定できるため，非階層的クラスタリングに比べてクラスタ数を高速に決定できる．以上の理由から，TSifter では階層的クラスタリングを採用する．

クラスタリングの高速性を満たすためには，クラスタ数の決定に加えて，クラスタリングの処理自体を高速に実行する必要がある．TSifter は，類似性が高いメトリックのペアが一つでも見つかれば積極的に集約するために，階層的クラスタリングの一種である最短距離法 [23] を用いる．最短距離法は，個々のメトリックがそれぞれ一つのクラスタを形成している状態から開始し，メトリック間のうち距離が最も短いメトリックが属しているクラスタを再帰的に併合する．一般に階層的クラスタリングは，クラスタリング対象のデータ数の増加に対して計算量が大幅に増加する問題があり，最短距離法では全てのメトリック間の距離を計算する必要があるため，計算量はデータ数 n に対して $O(n^2)$ である．TSifter ではクラスタリングをマイクロサービスにおけるサービス単位で実行しており，マイクロサービスが大規模化する場合，サービス内の構成要素が増大するより，サービス数が増えることが想定されるため，クラスタリング対象のメトリック数の増加は起きにくく，データ数に対する計算量の増大は大きな問題とならない．また，TSifter で採用した距離尺度である SBD は，式 (1) における相互相関の計算を離散フーリエ変換の高速演算法である高速フーリエ変換 [24] を用いて高速に計算できるため，この利点が最短距離法の高速性にも大きく寄与する．

4. 実験と評価

本章では，実験用に構築したマイクロサービス環境にて，手動で故障を注入する実験を行った結果を用いて，TSifter の次元削減の正確性，次元削減率，および高速性を評価する．

4.1 実験の環境と設定

テストベッド 実験のためのテストベッド環境は表 1 の通りである．本実験では，Kubernetes クラスタの自動管理サービスである GKE（Google Kubernetes Engine）[*1] 上に構築したクラスタ上に，マイクロサービスのベンチマークアプリケーションである Sock Shop[*2] を構築した．Kubernetes クラスタは 5 台の Worker ノードから構成されており，5 台の内 4 台をマイクロサービスを配置するためのサービス用途，残り 1 台をデータ収集と負荷生成のための管理用とした．

ベンチマークアプリケーション Sock Shop は靴下を販売する電子商取引 Web サイトを模したデモアプリケーションである．Sock Shop は，front-end サービス，catalogue サービス，carts サービス，user サービス，payment サービス，shipping サービスの計 7 個のマイクロサービスから構成されている．各マイクロサービスのコンテナの複製数は 1 に設定した．

データ収集 メトリックのデータ収集のために，システムの構成要素が公開するメトリックを収集・保存・取得するためのツールである Prometheus[*3] を使用した．Prometheus により，各コンテナから CPU，メモリ，ディスク，およびネットワークの物理資源とミドルウェアの性能や論理資源に関するコンテナレベルのメトリックを収集した．また，各マイクロサービスから平均応答時間と時間あたりの処理リクエスト数などのサービスレベルのメトリックを収集した．Prometheus のメトリック収集のインターバルを 5 秒に設定した．

負荷生成 Sock Shop アプリケーションに対して，擬似的な負荷を生成するために，Locust[*4] を利用した．本物の利用者がサイトのトップページからカタログ一覧を取得し，その中から商品を選び，注文するまでの典型的なフローを模倣するようなシナリオを記述し，シナリオを読み込ませて負荷を生成した．

故障注入 Locust による負荷を生成している間に，実際のマイクロサービスにおける性能異常を模倣するために，関連研究 [11,12,15] の実験で共通して採用されている故障を注入した．特定のコンテナ内の CPU 負荷が 100%となる故障を注入するために，stress-ng[*5] を利用した．また，特定のコンテナと通信するネットワーク遅延が増大する故障を注入するために，tc（Traffic Control）[*6] を利用した．

ベースライン手法 TSifter を評価するためのベースライン手法として，2 章で挙げた関連研究の中で唯一メトリッ

[*1] Google Kubernetes Engine: `https://cloud.google.com/kubernetes-engine`
[*2] Sock Shop: `https://microservices-demo.github.io/`
[*3] Prometheus: `https://prometheus.io/`
[*4] Locust: `https://locust.io/`
[*5] stress-ng: `https://kernel.ubuntu.com/~cking/stress-ng/`
[*6] tc: `https://linux.die.net/man/8/tc`

クの次元削減手法を提案している Sieve [19] を選択した．Sieve は，変動の少ないメトリックを除去するために，分散値の小さいものを除去したのちに，時系列クラスタリング手法 k-Shape [21] を用いて，構成要素ごとにその構成要素の特徴を最もよく表す代表メトリックを抽出している．Sieve の著者らは，特定の性能異常に対するシステムの特徴を抽出するよりは，事前に対象システムに対して開発者による負荷テストを実施することにより，恒常的に利用可能なシステムの特徴を抽出することを目的としている．本研究の目的とは異なるが，本研究の目的に対して有効な手法でもある可能性を評価する．

TSifter とベースライン手法の実装 両手法ともに Python を用いて実装した．TSifter において，ADF 検定における有意水準は 0.05，最短距離法におけるクラスタ併合の距離の閾値は 0.01 を用いた．CPU のマルチコアによる高速化のために，両手法ともに，全体の実行時間への寄与度が高いタスクを，互いに依存のない小さなタスクに分割し，マルチプロセスで処理するように実装した．

評価指標 TSifter の要件に，メトリックのクラスタリング後に原因メトリックが削減されないことがある．その上で，どの程度次元削減できているかを示す次元削減率と，高速性の要件から次元削減処理の実行時間の評価が必要となる．そのためにまず，各故障注入時のメトリックを次元削減した結果，正解となるメトリック（以降，正解メトリックとする）が次元削減後に残留しているかを TSifter とベースライン手法の両者で正誤を確認する．次に，次元削減率を評価するために，各故障ケースにおけるメトリックの次元削減率をベースライン手法と比較する．最後に，高速性を評価するために，特定の故障ケースにおいて，CPU コア数の増加と，メトリック数の増大に対して，次元削減処理の実行時間の変化を確認する．実行時間の計測値として，TSifter では 5 回試行した結果の平均値を採用した．ベースライン手法では，実行時間の都合により，1 回試行した結果の値を採用した．

著者らが管理する GitHub リポジトリ[*7]内に，テストベッド環境の構築と実験に利用した各種設定ファイルとソースコード，およびデータセットを公開している．

4.2 実験の結果

表 2 に示す各故障ケースに対して，1 メトリックあたり 360 個のデータ点に対して，TSifter とベースライン手法を適用することにより，マイクロサービスごとに複数の代表メトリックを選出した．故障を注入したサービスの代表メトリックが，表 2 に示す各故障ケースに対する代表メトリック候補のいずれかに該当することを以って，性能異常の特徴を表すメトリックを正しく抽出できたこととする．

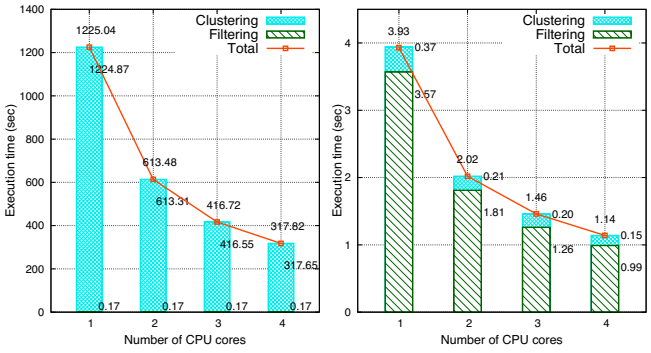
[*7] https://github.com/yuuki/microservices-demo

表 2: 各故障ケースに対する代表メトリックの正誤

	正解メトリックの候補	user		shipping	
		ベースライン	TSifter	ベースライン	TSifter
CPU 過負荷	cpu_usage_seconds_total	×	×	×	×
	cpu_user_seconds_total	×	×	×	✓
	cpu_cfs_periods_total	✓	×	×	×
	cpu_cfs_throttled_periods_total	×	✓	×	×
	cpu_cfs_throttled_seconds_total	×	×	×	×
ネットワーク混雑	network_transmit_packets_total	×	×	×	×
	network_transmit_bytes_total	×	×	✓	×
	network_receive_packets_total	×	×	×	×
	network_receive_bytes_total	✓	✓	×	✓

表 3: 故障の種類ごとのメトリックの次元削減率

	user		shipping	
	ベースライン	TSifter	ベースライン	TSifter
CPU 過負荷	1545/324/79 94.8%	1545/201/122 92.1%	1541/328/98 93.6%	1541/156/89 94.2%
ネットワーク混雑	1596/344/110 93.1%	1596/248/128 91.9%	1543/335/63 95.9%	1543/262/128 91.7%



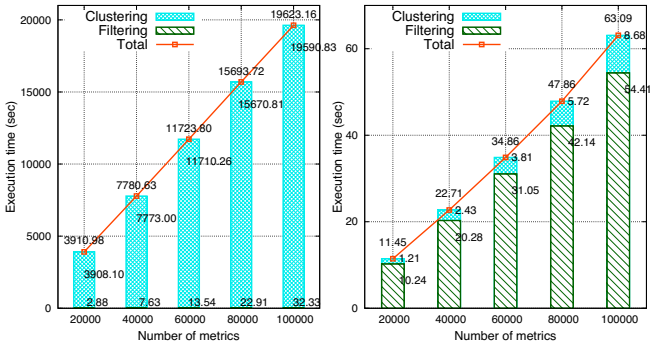
(a) ベースライン **(b)** TSifter

図 2: CPU コア数に対する実行時間の変化

表 2 より，TSifter は全てのケースに対して正しく代表メトリックを抽出した一方で，ベースライン手法は shipping サービスの CPU 過負荷のケースのみ正解でないメトリックを代表として抽出した．

表 3 に，user と shipping サービスへの各故障注入ケースに対するメトリックの次元削減数と削減率を示す．/で区切られた数値は左から元のメトリック数，事前除去後のメトリック数，クラスタリング後の代表メトリック数となる．いずれの手法，いずれのケースにおいても次元削減率は 90%を超えており，削減後のメトリック数は 1/10 以下となった．また，TSifter とベースライン手法を比較すると，shipping サービスの CPU 過負荷のケースを除いて，ベースライン手法のほうが高い次元削減率を示した．

ハードウェアのリソース量を増加させたときに，実行時間をどの程度短縮できるかを確認するために，TSifter とベースライン手法のそれぞれについて，CPU コア数に対する実行時間の変化を確認した．表 1 より，実験環境のコア数は論理コア数 8 であるが，物理コア数は 4 であるため，CPU コア数を変化させる実験では最大コア数を 4 とした．メトリックの個数を 1545 個，メトリックあたりのデータ点数を 360 個で固定した上で，利用する CPU コア数を 1



(a) ベースライン (b) TSifter

図 3: メトリック数増に対する実行時間の変化

から 4 まで増加させたときの実験結果を図 2 に示す．図 2 より，TSifter とベースライン手法はそれぞれコア数に反比例して実行時間が変化していた．また，TSifter の実行時間は，いずれのコア数においてもベースライン手法の 270 倍以上となった．

メトリック数を増加させたときの TSifter とベースライン手法の実行時間の変化を図 3 に示す．CPU コア数を 8，データ点数を 360 に固定した上で，横軸のメトリック数を 1,000 から 100,000 まで増加させた．メトリック数を増加させたデータを作成するために，既存の 7 サービスのメトリックを複製し，サービス数を擬似的に増加させた．実験の結果，両手法にて，メトリック数に対して実行時間が線形に増加した．TSifter は，メトリック数 100,000 において 63.09 秒で処理を完了できた．また，TSifter の実行時間は，いずれのデータ点数においてもベースライン手法の 315 倍以上となった．

4.3 実験の考察

本実験の範囲内において，ベースライン手法は誤ったメトリックを代表として選択した一方で，TSifter は全てのケースにおいて正しく代表メトリックを選択できたことから，TSifter のほうが性能異常の特徴を表したメトリックを削減させないと言える．しかし，本実験では，故障を注入したサービスの種類や故障ケースも限定的であったため，その他の故障ケースやサービスに対して TSifter が有利であるとは言えず，現時点では同等程度の正確性であると評価する．より広範囲の故障ケースやシステムに対する TSifter の正確性の評価は今後の課題とする．次に，次元削減率に関する実験の結果，ベースライン手法のほうが次元削減率が高い結果となったが，いずれもメトリック数を 1/10 以下に削減できたため，TSifter との大きな差はなく，同程度の次元削減率といえる．さらに，実行時間に関する実験の結果，TSifter とベースライン手法のいずれもメトリックの個数と実行時間は線形に増加する一方で，CPU コア数に対して実行時間が反比例した．したがって，両手法ともに，メトリックのデータ量が増加しても同割合の計算リソースを追加することにより，実行時間を維持できる．その一方で，ベースライン手法と比較し，TSifter は最低でも 270 倍以上高速に動作したため，TSifter のほうが高速性の要件を満たすと言える．しかし，ベースライン手法は，TSifter と比較し，低速だが次元削減率が高いことを踏まえると，高速性を公平に評価するには，両手法の次元削減率を揃えた上で，実行時間を比較する必要がある．両手法のアルゴリズムの性質上，次元削減率を揃える実験を行うことは難しいため，より公平な高速性の評価は今後の課題とする．

4.2 節より，ベースライン手法ではクラスタリングに長い実行時間を要している．ベースライン手法では，クラスタ数を変化させながら繰り返し k-Shape を実行し，最適なシルエットスコア [25] に基づき，クラスタ数を決定している．すなわち，クラスタ数を決定するために複数回クラスタリングを実行する必要がある．一方，TSifter における階層的クラスタリングでは，クラスタリング実行後に距離の閾値を用いてクラスタ数を決定できるため，クラスタリングの実行回数は 1 回のみである．このクラスタリングの実行回数の違いが，TSifter とベースライン手法の実行時間の差の主な原因となっている．

TSifter の要件である高速性に対して，実行時間がどの程度であれば十分であるかについて考察する．人間が手動で性能異常から回復させるとすると，原因の診断を秒単位で完了させる必要はない．また，サービスの利用者へシステム障害情報を告知する際に，分単位の事象を報告することがほとんどである．これらを踏まえると，性能異常の検知後，数分以内程度に原因の診断を終えられることが理想である．TSifter は，10 万メトリックのデータに対しても 1 分程度の時間で実行可能であることから，十分に高速であると言える．

TSifter は最短距離法におけるクラスタ併合の距離の閾値をパラメータとして持つ．閾値を大きくすることで，次元削減率は高まるが，原因となるメトリックが削減される可能性が高くなるため，適切な値の設定が必要である．この閾値を統計的かつ自動的に決定することは，今後の課題とする．

TSifter を原因診断システムに組み込むことを想定すると，メトリックの次元削減以外に，大量の時系列データを短時間に取得する必要がある．時系列データの問い合わせ処理を含めた実行時間の評価は，今後の課題とする．

5. まとめと今後の展望

本論文では，マイクロサービスにて性能異常が発生したときの原因をシステム管理者が迅速に診断することを目的に，大量のメトリックから診断に有用なメトリックを抽出するための次元削減手法 TSifter を提案した．マイクロサービスのテストベッド環境にて TSifter を評価した結果，

TSifter は，ベースライン手法に対して，正確性，次元削減率，およびメトリック数と CPU コア数に対するそれぞれの実行時間のスケーラビリティでは，同等程度の性能をもち，最低でも 270 倍以上の高速性をもつことがわかった．TSifter は，10 万メトリックのデータに対しても 1 分程度の時間で実行可能であり，計算リソースの追加投入によりさらに高速化可能である．

今後は，システム管理者が利用可能なシステムを実現するために，TSifter を性能異常の原因診断システムへ組み込むことを検討する．具体的には，2 章に示したメトリックベースの自動解析アプローチと同様に，各構成要素において，抽出された代表メトリック同士の因果関係を判定することにより，性能異常がシステム内をどのように伝搬したかを追跡可能とするといったシステムがありえる．また，マイクロサービス以外にネットワーク層のシステムなどの異なる階層のシステムへの応用を検討する．

参考文献

[1] Newman, S., *Building Microservices: Designing Fine-Grained Systems*, "O'Reilly Media, Inc." 2015.

[2] Taichi Nakashima, SRE Practices in Mercari Microservices, https://speakerdeck.com/tcnksm/sre-practices-in-mercari-microservices.

[3] Introducing Atlas: Netflix's Primary Telemetry Platform, http://techblog.netflix.com/2014/12/introducing-atlas-netflixs-primary.html.

[4] Lin, Q., Zhang, H., Lou, J.-G., Zhang, Y. and Chen, X., Log Clustering based Problem Identification for Online Service Systems, *IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, pp. 102–111 2016.

[5] Kaldor, J., Mace, J., Bejda, M., Gao, E., Kuropatwa, W., O'Neill, J., Ong, K. W., Schaller, B., Shan, P., Viscomi, B. et al., Canopy: An End-to-End Performance Tracing and Analysis System, *the 26th Symposium on Operating Systems Principles (SOSP)*, pp. 34–50 2017.

[6] Sigelman, B. H., Barroso, L. A., Burrows, M., Stephenson, P., Plakal, M., Beaver, D., Jaspan, S. and Shanbhag, C., Dapper, a Large-Scale Distributed Systems Tracing Infrastructure, Technical report, Google 2010.

[7] Fonseca, R., Porter, G., Katz, R. H. and Shenker, S., X-Trace: A Pervasive Network Tracing Framework, *USENIX Conference on Networked Systems Design & Implementation (NSDI)*, pp. 20–20 2007.

[8] Barham, P., Donnelly, A., Isaacs, R. and Mortier, R., Using Magpie for Request Extraction and Workload Modelling, *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Vol. 4, pp. 18–18 2004.

[9] Chen, M. Y., Kiciman, E., Fratkin, E., Fox, A. and Brewer, E., Pinpoint: Problem Determination in Large, Dynamic Internet Services, *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 595–604 2002.

[10] Ma, M., Xu, J., Wang, Y., Chen, P., Zhang, Z. and Wang, P., AutoMAP: Diagnose Your Microservice-based Web Applications Automatically, *The Web Conference (WWW)*, pp. 246–258 2020.

[11] Qiu, J., Du, Q., Yin, K., Zhang, S.-L. and Qian, C., A Causality Mining and Knowledge Graph Based Method of Root Cause Diagnosis for Performance Anomaly in Cloud Applications, *Applied Sciences*, Vol. 10, No. 6, p. 2166 2020.

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

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

[14] Wang, P., Xu, J., Ma, M., Lin, W., Pan, D., Wang, Y. and Chen, P., CloudRanger: Root Cause Identification for Cloud Native Systems, *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pp. 492–502 2018.

[15] Lin, J., Chen, P. and Zheng, Z., Microscope: Pinpoint Performance Issues with Causal Graphs in Micro-Service Environments, *International Conference on Service-Oriented Computing (ICSOC)*, pp. 3–20 2018.

[16] Chen, P., Qi, Y., Zheng, P. and Hou, D., CauseInfer: Automatic and Distributed Performance Diagnosis with Hierarchical Causality Graph in Large Distributed Systems, *IEEE Conference on Computer Communications (INFOCOM)*, pp. 1887–1895 2014.

[17] Hightower, K., Burns, B. and Beda, J., *Kubernetes: Up and Running: Dive into the Future of Infrastructure*, "O'Reilly Media, Inc." 2017.

[18] Beyer, B., Jones, C., Petoff, J. and Murphy, N. R., *Site Reliability Engineering: How Google Runs Production Systems*, "O'Reilly Media, Inc." 2016.

[19] Thalheim, J., Rodrigues, A., Akkus, I. E., Bhatotia, P., Chen, R., Viswanath, B., Jiao, L. and Fetzer, C., Sieve: Actionable Insights from Monitored Metrics in Distributed Systems, *the ACM/IFIP/USENIX Middleware Conference (Middleware)*, pp. 14–27 2017.

[20] Dickey, D. and Fuller, W., The Likelihood Ratio Statistics For Autoregressive Time Series With a Unit Root, *Econometrica*, Vol. 49, No. 4, pp. 1057–1072 1981.

[21] Paparrizos, J. and Gravano, L., k-Shape: Efficient and Accurate Clustering of Time Series, *The ACM Special Interest Group on Management of Data (SIGMOD)*, pp. 1855–1870 2015.

[22] MacQueen, J., Some Methods for Classification and Analysis of Multivariate Observations, *The Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pp. 281–297 1967.

[23] Murtagh, F., A Survey of Recent Advances in Hierarchical Clustering Algorithms, *The Computer Journal*, Vol. 26, No. 4, pp. 354–359 1983.

[24] Cooley, J. and Tukey, J., The Likelihood Ratio Statistics For Autoregressive Time Series With a Unit Root, *Mathematics of Computation*, Vol. 19, No. 90, pp. 297–301 1965.

[25] Rousseeuw, P., Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis, *Journal of Computational and Applied Mathematics*, Vol. 20, pp. 53–65 1987.