Sock Shopのサービス絞り込みとPod数とリクエスト数のピアソン相関係数に基づくグラフ表示変更による利便性向上

増田 和範[1]　大野 有樹[2]　串田 高幸[1]

概要：EC サイトのデモアプリケーションである Sock Shop の CPU 量を対象にグラフ作成を行った場合，サービスにまとめることで 14 個のグラフが作成される．グラフ利用者はグラフ同士を比較し分析する必要がある．課題は Sock Shop から作成されたグラフを選択する際の組み合わせ数である．既存グラフツールの場合，表示順がアルファベット順なため，エラーに関与するサービスが表示順に考慮されない．提案手法はリクエストの処理に関与するサービスとリクエストを紐付ける．負荷試験にてエラーが発生したリクエストと紐付けられたサービスに絞り込む．絞り込まれたサービス内の Pod 数と秒間リクエスト数のピアソン相関係数を優先度としてその値が降順になるようにグラフの表示順を変更する．実験は Locust を用いて Sock Shop のサイトへ負荷試験を実施し，負荷試験におけるサービスごとの CPU 使用量のグラフを作成した．評価はリクエストの処理に関与するサービスに絞り込みを行った場合と提案ソフトウェアによる絞り込みによるグラフの組み合わせ数の削減率を求める．表示順の評価はエラーが発生したリクエストと関係のあるサービスの優先度が上位になるか検証した．提案前は 11 個のグラフが表示され，その中から 2 個を選択する組み合わせ数は 55 通りの組み合わせ数である．PSSP 使用時は 3 個のグラフが表示され，その中から 2 個を選択する組み合わせは 3 通りである．この結果から PSSP の使用によって，選択数の削減率が 94.5%となった．PSSP によって「orders」,「carts」,「users」の順にグラフが表示された．エラーが発生した商品決済である「orders」と応答時間が増加したカート操作である「carts」の順位となり．エラーや応答時間の増加が発生していない，ユーザ情報の処理に関与する「users」の優先度が 3 位になった．

1. はじめに

背景

オンラインショッピング環境を模倣したオープンソースアプリケーションである Sock Shop を用いて EC サイトへの負荷試験を実施する．この試験を通じて，システムの応答速度，一定時間内に処理できるリクエストの数，サーバ内の CPU 使用量などの複数の指標を分析する. これらの指標から，システムへのリクエスト数が増加した際のシステムの挙動を分析することができる [1].

マイクロサービスアーキテクチャでは，各サービスが独立した Pod として動作し，それぞれが CPU・メモリを消費する. これらの Pod ごとの CPU 使用量を視覚的に把握するために，グラフを代表する可視化を行う. これにより，各サービスの CPU・メモリ使用状況を明確にし，複数のデータの比較・大量データの流れを視覚的に分析することができる [2]. そして CPU 使用量と EC サイトのリクエスト数は比例関係にあり，この数値を分析することは，システムのパフォーマンスと顧客体験の向上に直結する重要な要素である [3].

マイクロサービスアーキテクチャでは，個々のサービスが特定の機能を担い，それぞれ独自の Pod が稼働している. リクエストは特定のサービスを通して内部の Pod によって処理される．各サービスのパフォーマンス分析し，問題を発見した場合，個別のサービスごとに対処が可能になる. そのため，Pod をサービスごとに分類し監視・分析を行うことは重要である [4].

Sock Shop のように，機能として分割したサービスはそれぞれデータベースがある．Sock Shop の場合，サービス名「orders」,「users」,「catalogue」,「carts」の 4 個のサービスがそれぞれ固有のデータベースと接続されている．そのため，これらのサービスに問題が発生した際は，サービスだけではなく，それに付随するデータベースの分析が必要である [5].

多数あるグラフを人に説明する際には，必要な情報だけに焦点を当て，不必要な情報を排除する必要がある．これ

[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

は，視覚的な情報の過多が理解を妨げ，誤解を生む可能性があるためである．したがって，目的に合致した，かつ，必要十分なグラフのみを選択し，他人に伝えることが求められる [6].

既存のグラフ作成ツール，例えば Grafana などが存在するが，これらのツールの利用にはユーザー自身による適切なフィルタリングの設定が必要である. しかし，フィルタリングはユーザ自身の主観にもとづくため，人によって選択するグラフが異なる可能性がある [7].

課題

課題は Sock Shop の 14 個のサービスからグラフ作成ツールによって作成され，ユーザに表示されるグラフを選択する際の作業数である．グラフの使用用途としてグラフ同士を比較する方法があげられる [8]．そのため，選択作業数は Sock Shop における，異なる n 個のサービスをもとに作成されたグラフ画像から 2 個を選ぶ場合の、組み合わせ数とする．選択作業数を式 (1) によって算出する．

$$\text{選択作業数} = {}_nC_2 \tag{1}$$

n　表示グラフ数

x　選択数

図 1 は課題が発生するグラフ選択までの流れを示す．サービス内の Pod ごとに作成される CSV files からそれぞれのサービスにまとめてグラフを作成する．図 1 では A から N までの 14 個のサービスが存在する．この場合，グラフは 14 個作成されグラフ作成ツールの使用者に表示される．ユーザーは表示されたグラフの中から，必要と不必要なグラフを分ける作業が必要である．

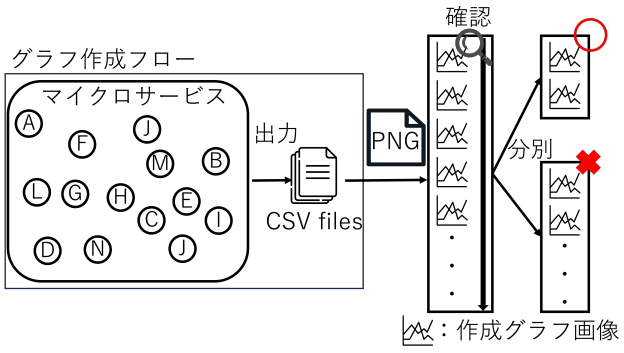

図 1　グラフ選択までの流れ

式 (1) を用いた場合の組み合わせ数を表 1 に示す．この表が示すように，選択の候補となる n の値を小さくすると，組み合わせ数が減少する．よって，候補となるグラフ数の削減，つまりグラフの表示数を削減させることで，組み合わせ数を減少させることができる．

表 1　サービスの組み合わせ

n	選択数
1	算出不可
2	1
3	3
4	6
5	10
6	15
7	21
8	28
9	36
10	45
11	55
12	66
13	78
14	91

これらの課題は対象アプリケーションに対する負荷試験において発生する．負荷試験では，アプリケーションが処理可能なリクエスト数を検証する．CPU 使用量が不足すると，送信したリクエストが正常に返されず，500 番台のステータスコードが返される．そのため，システム管理者はサービス毎の CPU 使用量のグラフを確認し，エラー関連のサービスのグラフを対象として CPU 使用量とリクエスト数の関係性を調べる必要がある．

マイクロサービスアーキテクチャでは，1 つのリクエスト処理に複数のサービスが関与するため，リクエストとサービスの 1 対 1 対応は成立しない．これにより，エラー発生時に 1 つサービスに絞り込むことができない．図 2 は，Sock Shop の「/orders」エンドポイントへのリクエスト処理のシステム内サービス関係を示す．図 2 が示すように処理のために合計 11 個のサービスが関与する．以下の表に処理に関与するサービスの一覧とその役割を示す[*1].

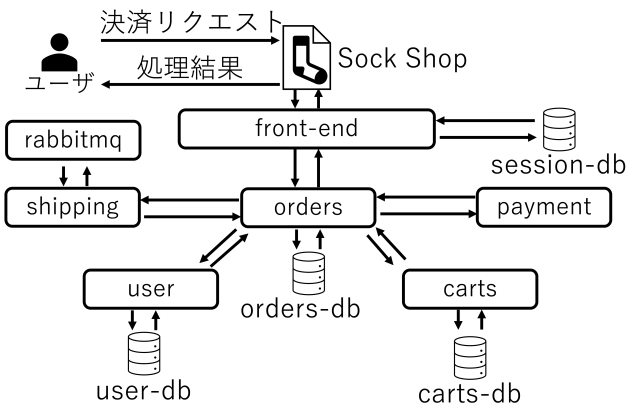

図 2　リクエスト送信処理経路 1

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

サービス名	役割
carts	ショッピングカート機能を提供
carts-db	ショッピングカートのデータベースサービス
front-end	ユーザーインターフェースのサービス
orders	注文処理サービス
orders-db	注文データベースサービス
payment	支払い処理サービス
rabbitmq	メッセージブローカーサービス
session-db	セッションデータベースサービス
shipping	配送サービス
user	ユーザーサービス
user-db	ユーザーデータベースサービス

表 2　Sock Shop のサービスとその役割

送信されるリクエストで指定したエンドポイント名「orders」とシステム内に同名のサービスの「orders」がある．しかしこのエンドポイントへのリクエストによってエラーが発生した場合，orders サービスのみがエラーの原因と考えられない．他のサービスとの依存関係によりエラーが発生した可能性がある [9]．そのため，このリクエストのエンドポイント名とサービス名だけを考慮してエラー原因とすることができず，他のサービスの状況を比較，分析する必要がある．/orders へのリクエストが送信された際に，このリクエストを処理するために，11 個のサービスが関与することを考慮すると，それらのサービスが使用する CPU 使用量を 1 つまたは複数の組み合わせで分析する必要があり，選択肢が増加する．これにより，「選択肢のパラドクス」が発生する [10]．選択肢のパラドクスは，選択肢が多い場合，最良の選択をすることがより複雑になり，多くの選択肢がある場合よりも，限られた選択肢がある場合の方が，ユーザの意思決定が迅速になる．

また，表示されるグラフの順序も重要である [11]．人は表示された対象から，上から順に確認する．そのため，アルファベット順やランダムな表示順よりも WEB ページの検索結果の表示のように，検索結果に関連のあるサイトを順に表示することで，ユーザの選択作業を助ける．そのため，表示数の絞り込みと，表示順の変更はグラフ生成ツールの利便性に影響を及ぼす．

図 3 は課題が発生する際の負荷試験の状況を示す．

負荷試験ツールから出力される CSV file1 には負荷試験の結果のリクエスト数，失敗したリクエスト，エラーコードが保存される．

CSV files2 は Pod ごとに使用する CPU 使用をそれぞれ別の CSV 形式に記述したファイルである．

各章の概要

2 章はマイクロサービスにリソースメトリクス監視を行い，その結果をグラフ化している研究について述べる．3 章は本稿で提案するグラフ画像の表示順を変更する方法について述べる．4 章は提案を実装した概要と各コンポーネントの説明について述べる．5 章は基礎実験と評価実験の環境と目的そしてその結果と考察について述べる．6 章の議論では実験結果と分析から今後の展望や新たな課題について述べる．7 章のおわりにでは，全体のまとめについて述べる．

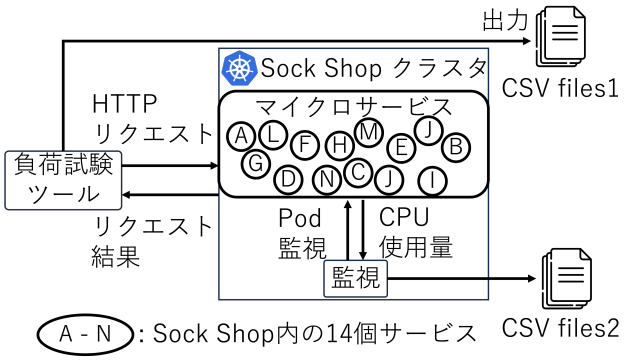

図 3　課題発生の前提状況

2. 関連研究

リソース監視による高性能コンピューティングクラスタの使用パターンの理解の研究がある [12]. 実行時のアプリケーションのログメッセージと実際のリソース消費量の両方を監視する．課題は監視対象の最小単位がノードで行わることである．Kuberntes によるマイクロサービスの監視はより細かい粒度である Pod の監視が必要である．

OpenStack 環境のパフォーマンスや可用性を監視する研究がある [13]．マイクロサービスを採用した監視システムを提案する．提案ソフトウェアにより監視結果の表示までの流れを簡素化する．課題は監視結果を一律に表示するため，マイクロサービス内のサービスごとに確認する必要がある．状況において，マイクロサービス内のサービスに優劣を決める必要がある．

マイクロサービスのリアルタイムのリソース監視を行う研究がある [14]．既存のツールを利用してクラスターベースの異なるエッジコンピューティングシステムを提案する．課題は既存ツールである Grafana を用いてグラフ化を行っているため，グラフをレポートとして添付する場合，グラフ内のフォントサイズと配色を設定する必要がある．

3. 提案

提案方式

本稿で提案手法の全体の流れを図 4 に示す. アルファベット順に並べられた全サービスごを対象としたそれぞれのグラフを表示するのではなく，図 4 の流れのように提案ソフトウェアによってユーザに表示されるグラフの数を絞り込む．そして，サービスごとに表示優先度を割り当て，並び替えを行う．これにより，グラフ生成ツールの利便性を向

上させる．この提案はリクエストに対してエラーが発生する処理に関与したサービスのグラフを分析することを想定する．エラー発生の原因には，リクエストの増加に対して，Pod 数の増加が十分でないことがあげられる．この状況に対処するために，リクエスト数と Pod 数の相関関係が存在するサービスのグラフを分析するユーザを対象とする．

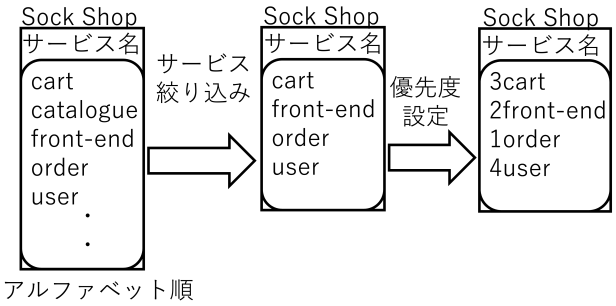

図 4　提案手法の概要

図 5 はカート処理と決済処理のリクエストを送信した際の処理関与サービスの関係を示す．カート処理では「front-end」,「session-db」,「user」,「user-db」,「carts」,「carts-db」を使用する．決済処理では図 5 で示す全てのサービスを使用する．このように，リクエストごとに処理に関与するサービスは異なる．

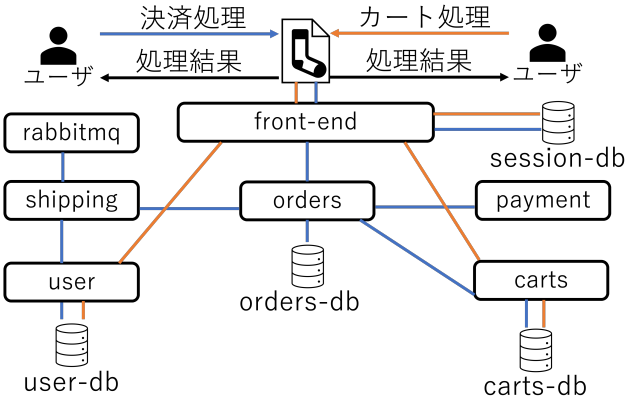

図 5　リクエストごとの処理関与サービスの関係

負荷試験の実施前に負荷試験で送るリクエストを Sock Shop に送信する．Istio を使用することで，リクエストが送信された際の Sock Shop 内のサービス間の接続状況を取得することができる．これにより，リクエストとその処理に関与するサービスを紐付ける．負荷試験によってエラーが発生したリクエストを Locust から出力されるファイルから取得する．そのリクエストに関連のあるサービスを事前に紐づけを行ったサービスから選択する．リクエストに対してエラーが発生したサービスに絞り込むことで，表示グラフ数の削減を行う．エラーに無関係なグラフを非表示にすることで，ユーザがグラフを選択する作業を削減できる．

上記の提案には問題がある．Pod 数の増加のみを考慮してしまう場合，すべてのリクエストが通過する「front-end」サービスの Pod 数が一番増加することで，優先度が最上位になってしまう．一方で，図 6 が示すように，front-end に問題がある状況を示す．このように「front-end」サービスに異常がある場合，送信されるリクエストが「front-end」で処理できなくなるため，エラーが発生する．

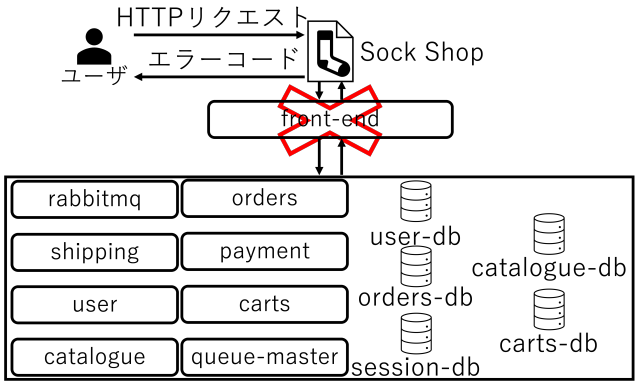

図 6　front-end サービスの必要有無の判断

一方で，負荷試験中に 1 度もエラーが発生していないリクエストが 1 種類でも存在する場合，他のリクエストのエラー原因は front-end 以外になる．なぜなら，front-end を通るリクエストにエラーが発生しないためである．そのため，負荷試験によって送信するリクエストの種類数とエラーが発生したリクエストの種類数を比較する．エラーが発生したリクエストの種類数が少ない場合，特定のリクエストは処理が正常に行われたことを意味する．これにより，front-end をグラフ表示サービスの候補から除外する．

マイクロサービス内の各サービスに対して優先度を付与し，これにもとづいてサービスごとに作成されるグラフの表示順を並び替える．CPU 使用量と EC サイトのリクエスト数は比例関係にあり，この数値を分析することは，システムのパフォーマンスと顧客体験の向上に直結する重要な要素である．この関係を数値化するために，負荷試験におけるサービスごとの Pod 数の推移と 1 秒間あたりに送信されるリクエスト数推移に対してピアソン相関係数を算出しその値を優先度として使用する．

送信されるリクエスト数が増加することで特定のサービスの負荷が増大すると，Kubernetes のスケーリング機能により，該当サービス内の Pod 数を増加させる．これにより負荷分散を行い，そして使用可能 CPU 量を増加させる．本稿の提案では，サービスごとにスケーリングされる Pod 数と秒間リクエスト数からピアソン相関係数を算出し，求めた係数を優先度として各サービスに与えグラフ表示順を変更する．

ピアソンの相関係数は式 (2) で算出される[*2]．サービスごとにこの式を用いて相関係数を算出する．負荷試験の時間（分）を使用する．同時刻の秒間あたりのリクエスト数とサービス内の Pod 数を代入することで，相関係数を求めることが可能である．

$$\rho = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \qquad (2)$$

ρ　ピアソンの相関係数

n　データ点の個数

x_i と y_i　各時刻の値

$\bar{x}$　x_i のサービス内の Pod 数

$\bar{y}$　y_i の秒間あたりのリクエスト数

本提案のサービスの絞り込みと優先度の作成の手順を図 7 に示す．

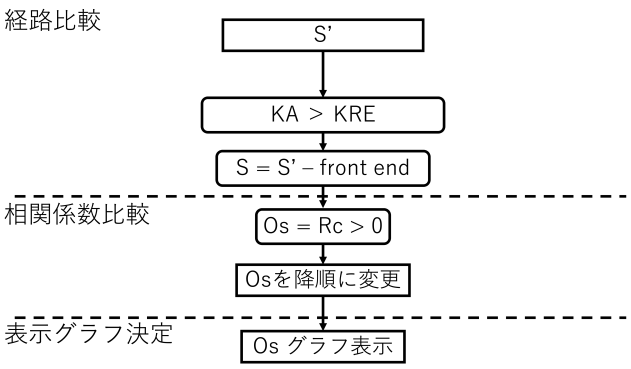

図 7　サービスの絞り込みと優先度の作成

負荷試験後，成功したリクエストとエラーが発生したリクエストに分類を行う．エラーが発生したリクエストと紐付けられたサービスを S’ とする．エラーが発生したリクエストの種類数 (KRE) とし，この値と負荷試験によって送信されたリクエストの種類数 (KA) の比較を行う．KA ＞ KRE の関係が成り立つ際，S’ から front-end を除外したものを S とする．S の各 Pod 数と RPS からピアソン相関係数を算出し，0 以上のサービスを Os とする．Os を相関係数が降順になるように並び替えを行う．並び替えられたサービスのグラフを表示する．

ユースケース・シナリオ

本稿の想定するユースケースを図 8 に示す．

本稿では，Kubernetes 環境で稼働しているマイクロサービスアーキテクチャのシステムである Sock Shop に対して，送信可能なリクエスト数を検証するシステム管理者を

[*2] https://x.gd/WftJy

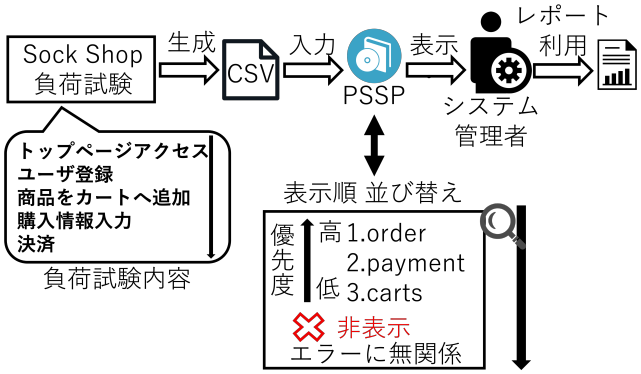

図 8　ユースケース

想定している．サービスごとの使用可能 CPU 量を増加させることで，処理可能リクエスト数を増加させることができる [3]．リクエスト数の増加に伴い，サービスの CPU 使用量が増加する場合，システム管理者はサービスに新たに CPU 使用可能量を割り当てることでリクエスト数を増加させることができる．

負荷テストによって生成されるサービスごとの CPU 使用量のデータからグラフ作成ツールによってグラフが生成される．これらのグラフの中からエラーが発生したリクエストの処理に関与するサービスのグラフを探す必要がある．

提案ソフトウェアは，エラーに関するサービスのみをグラフ表示することで，ユーザの選択作業数を減らす．また，ピアソンの相関係数を表示順の優先度としてサービスに紐付ける．この優先度に基づいて，グラフ画像を並び替えて表示する．これにより，ユーザーは無作為にグラフを探すことなく，リクエスト増加に相関関係のあるサービスの CPU 使用量のグラフの確認を表示順にもとづいて行うことができる．

4. 実装

実装はプログラミング言語の Python を用いて本稿のソフトウェアを実装した．以降，提案ソフトウェア名を PSSP とする．

データ取得

データ取得は既存のメトリクス監視アプリケーションを使用せず，独自で作成した．以下の監視項目を実行するソフトウェアを実装しデータの取得を行う．

- pod ごとの CPU 使用量
- サービス内の Pod 数

これらのデータは CSV 形式のファイルで保存される．kubectl コマンドを使用し，1 秒ごとに上記のデータをそれぞれ異なるファイルに記述する．

データ整形

データ取得で得られた CSV ファイルには，クラスタ内の全 Pod の CPU 使用量が記録されている．このデータをより扱いやすくするため，Python の Pandas ライブラリを活用して，サービスおよび Pod ごとにデータを整形し，新たな CSV ファイルとして分割する．これにより，後続の処理であるグラフ作成が容易になる．

グラフ作成

整形された CSV ファイルから，サービスおよび Pod の CPU 使用量を可視化するためのグラフを作成する．本ソフトウェアでは，折れ線グラフ，ヒストグラム，箱ひげ図の 3 種類のグラフを生成する．これにより，ユーザーは各サービスのパフォーマンスメトリクスを直感的に理解することが可能になる．

サービス絞り込み

事前に各リクエストと処理に関与するサービスを紐づけし，式を使用し，エラーに関係するサービスに絞り込む必要がある．Sock Shop の front-end はユーザに WEB ページを表示する機能がある．そのため，リクエスト数が増加すると CPU 使用量が増加する．それに伴い，Pod がスケーリングされてしまう．そのため，Pod 数とリクエスト数の相関係数を求めると，front-end が最上位になる．しかし，front-end ではエラーが発生しない．その理由は特定のリクエストのみエラーになっていることが挙げられる．

優先度作成

ユーザーケースで述べた通り，サービス内の Pod 数と秒間リクエスト数の間には相関が存在する．ピアソン相関係数を Pandas の corr メソッドを用いて計算し，各サービスの優先度を算出する．マイクロサービスアーキテクチャはサービスに付随してデータベースのサービスが存在する．このサービスはアプリケーションの特性上スケーリングすることができないため，本提案では相関係数を求めず，その付属先のサービスとセットにする．しかし，付属先のサービスの優先度をデータベースよりも上位にする．相関係数はサービス名とともに辞書形式で保存され，グラフ表示の際の並び順に反映される．

グラフ表示

最終的に，Python の WEB アプリケーションフレームワークを用いて WEB ページの表示を行う．並びえられたグラフをユーザーに表示する．このアプリケーションでは，サービスごとに異なるグラフを優先度に基づいた降順で表示する機能を有している．これにより，ユーザーは最も注意を払うべきサービスから順にグラフを確認することができ，より効率的なキャパシティプランニングが実現される．

以上の実装により，提案されたソフトウェアは，Kubernetes 環境下でのマイクロサービスの運用管理を効率化し，システム管理者の課題を軽減する．本研究で開発されたソフトウェアは，実際の環境での適用を通じて，その効果をさらに検証する予定である．

5. 評価実験

実験はユースケース (図 8) の状況を再現して行う．Sock Shop に対して負荷試験を行い，負荷試験中の Pod ごとの CPU 使用量を監視し，取得した CPU 使用量をもとにサービスごとにグラフ作成を自動で行う．負荷試験において，エラーが発生したリクエストに関連のあるサービスの組み合わせ数と PSSP を使用した場合の選択数ごとの組み合わせ数を比較し削減率 (%) を用いて評価する．削減率は式 3 を用いて算出する．

$$\text{削減率} = 100 - (\frac{P}{B} \times 100) \tag{3}$$

P　PSSP 使用時のサービス数

B　提案前のサービス数

優先度の評価はエラーに関係のあるサービスが上位になるかを検証する．

日本航空株式会社が運営する EC サイトではセール時に 1 時間あたり 3 万人のアクセスが発生する[*3]．1 時間あたりに 3 万人のアクセスが発生するため，1 秒あたりのサイトアクセス人数は 8.3 人である．そのため，負荷試験でのユーザの生成割合を 8.3 (人/秒) に設定した．最大ユーザ数を 300(人) 以上に設定した際，1 秒あたりのリクエスト数 (以下: RPS) が増加しないため，最大ユーザ数を 300(人) とし，株式会社スリーシェイクのテスト時間を参考にテスト時間を 1800 秒とした[*4]．表 3 が示す条件で 10 回の負荷試験を実施した．

表 3　ユーザ数と生成割合の関係

最大ユーザ数 (人)	ユーザの生成割合 (人/秒)	テスト時間 (s)
300	8.3	1800

実験環境

図 9 は実験環境を示す．負荷試験用に Locust クラスタを作成，K3s クラスタ作成のためにそれぞれ 3 台の VM を用意した．K3s クラスタは Sock Shop を稼働させるために使用した．PSSP は Locust クラスタの Master VM に導入した．負荷試験が実行されると，PSSP は K3s クラスタの

[*3] https://xtech.nikkei.com/atcl/nxt/column/18/00001/07852/
[*4] https://x.gd/O17c0

CPU 使用量とサービスごとの Pod 数を監視する．負荷試験終了後，本提案によってサービスが絞り込まれ，また優先度順に並び替えられ表示される．

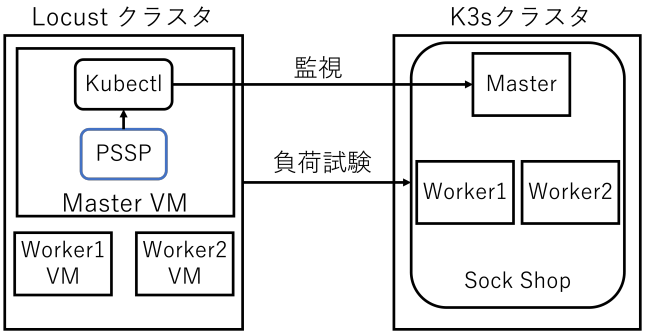

図 9　実験環境

以下の流れで Sock Shop に対して表 4 の順にリクエストを送信する．

処理内容	送信内容
ホームページへのアクセス	self.client.get(”/”)
ユーザ登録	client.post(”/register”)
商品詳細ページへのアクセス	self.client.get(”/category.html”)
カートの初期化	self.client.delete(”/cart”)
商品をカートへ追加	self.client.post(”/cart”)
カート内の商品を確認	self.client.get(”/basket.html”)
住所情報の送信	self.client.post(”/addresses”)
カード情報の送信	self.client.post(”/cards”)
決済	self.client.post(”/orders”)

表 4　Sock Shop のサービスとその役割

実験結果と分析

実験終了後，「/orders」のエンドポイントへのリクエストでエラーが発生した．表 5 が示すように，PSSP によりエラーが発生したリクエストの処理に関与するサービスに絞り込み，それぞれのサービスの相関係数を算出した．この結果，「/orders」のエンドポイントへのリクエストの処理に関与するサービス 11 個 (図 2) の内，「orders」，「carts」，「users」の 3 つのサービスに絞り込まれ，それぞれの相関係数を比較し，優先度を算出した．

表 5　PSSP を用いて絞り込まれたサービスの優先度

サービス名	相関係数	アルファベット順	表示優先度
orders	0.81	6	1
carts	0.72	1	2
users	0.31	13	3
他サービス	0	アルファベット順	表示なし

サービスの種類ごとに表示されるグラフがまとめられる場合，提案前は表示されるグラフ数が 11 個である．そして PSSP を使用した場合，3 個のグラフが表示される．選択作業数は表示されるグラフから 2 個を選択する組み合わせ数としている．提案前と PSSP 使用時のサービスの組み合わせを表 6 に示す．この場合の選択作業数は提案前の場合，55 通りであり，PSSP 使用時は 3 通りである．この結果から PSSP の使用によって，それぞれの選択数における削減率は 94.5%である．

表 6　提案前と PSSP 使用時の選択作業の比較

選択数	提案前	**PSSP** 使用時	削減率 (%)
2	55	3	94.5

評価実験によりエラーが発生した商品決済のリクエストでの処理を行う「orders」と応答時間が増加が発生したカート操作のリクエストの処理を行う「carts」の順で優先度が上位になったが，ユーザ情報処理を行う「users」が 3 位になってしまった．「users」は商品決済の処理において関与し，他のカート操作，クレジット情報の登録，配達場所の登録といった複数の処理に関与する．これにより，リクエスト数が増加することで，「users」の負荷が増加することで Pod 数が増加する．これにより，優先度の 3 が付与される．「users」を表示候補から削除することで，エラーと応答時間の増加に関与していないサービスのみをグラフ表示することができる．

6.　議論

負荷試験で送信するリクエストとその処理に関与するサービスを事前に紐づけしているが，これらを負荷試験中に自動で実行することで，ユーザの手間をより省くことができる．Kiali の API を使用することでサービスのつながりを取得することができる．しかし，Kiali の API のみではリクエストと処理関与サービスを紐付けることができない．Kiali の API を使用することで過去 60 秒のサービス間のつながりを取得できる．しかし，60 秒間に複数のリクエストが発生した場合，リクエストごとにサービスを紐付けることができない．そのため，各リクエストの送信間隔を 70 秒にする．これにより，Kiali で取得するリクエストごとに紐付けるサービスのつながりが他のリクエストとの干渉が発生しないため，自動で負荷試験で送信するリクエストと処理関与サービスを紐付けられる．

今回はリクエスト数の上限を確認するシステム管理者を対象としたが，負荷試験の目的は他にも考えられる．例えば，爆発的にリクエストが発生した場合のシステムの応答時間と Pod 数の関係性をグラフで分析したい場合，今回の提案手法では対処が不可能である．そのため，事前にユーザによるグラフ使用方法の入力から，その値と相関関係のある値に関するグラフを作成，表示する必要がある．

本提案は Pod 数の増加がリクエスト数の増加に対応できず，エラーが発生した際のグラフ分析を対象としてい

る．エラーが発生したリクエストと応答時間が増加したリクエストの優先度が上位になったが，これらに該当しない「users」サービスの優先度が次いで高い結果となった．この「users」サービスを削除するために，サービスの絞り込み方法を修正する．本提案ではリクエストと処理に関与するサービスを紐づけ，リクエストがエラーになった場合，その紐づけられたサービスを対象に優先度を算出した．この場合，関与するすべてのサービスが対象となる問題が発生する．この問題の対処として，エラーが発生したリクエストと応答時間が増加したリクエストに関与するサービスから 1 度もエラーが発生していないリクエストに関与するサービスを削除する．これにより，本実験で優先度が 3 になった「users」を削除することが可能になる．

折れ線グラフは 1 枚のグラフ画像に複数の Pod をプロットしている場合，それぞれの Pod の比較を行いやすい．しかし Pod の数が増加してしまい，1 つのグラフ画像に 10, 20 個の Pod がプロットされてしまうと，視認性が低下してしまう．その際は 1 つのグラフにプロットしてもユーザの視認性が低下しない閾値を設定し，似たグラフをまとめて表示数を減らして，視認性低下の問題を解決する必要がある．

7. おわりに

本提案はユーザに表示するグラフ数をリクエストのエラーに関与しているサービスのみに絞り込むことを目的とする．Sock Shop 内の各サービスに対して表示優先度付けとリクエストのエラーに関係するサービスの絞り込みを行う．それによりユーザは全サービスを対象に探すのではなく，表示の順に従いグラフの選択作業を行うことが可能である．これにより，グラフ生成ツールの利便性が向上する．課題は Sock Shop のサービスから作成されるグラフ数である．実験は，Locust を用いて負荷試験を実施した．評価は，提案ソフトウェアを使用した場合と，課題で述べた状況の場合のグラフ表示数を比較した．提案ソフトウェアでは 3 個のサービスに絞り込むことができ，提案前の 9 個とサービスによりグラフ数と比較して 66.7%削減できた．リクエストと処理関与サービスの紐づけを自動化することで，手作業が減少しグラフ生成ツールの利便性が向上する．

参考文献

[1] Kraemer, E., Paixão, G., Guedes, D., Meira, W. and Almeida, V.: Minimizing the Impact of Orphan Requests in E-Commerce Services, *SIGMETRICS Perform. Eval. Rev.*, Vol. 28, No. 2, p. 36–42 (online), DOI: 10.1145/362883.362911 (2000).

[2] Sonnad, S.: Describing data: statistical and graphical methods., *Radiology* (2002).

[3] 坂本一俊, 伊藤佳城, 串田高幸：マイクロサービスの CPU 使用可能量に基づく最小二乗法を用いた処理リクエスト数の推定，技術報告，東京工科大学コンピュータサイエンス学部, 東京工科大学大学院コンピュータサイエンス専攻 (2023).

[4] Sampaio, A. R., Kadiyala, H., Hu, B., Steinbacher, J., Erwin, T., Rosa, N., Beschastnikh, I. and Rubin, J.: Supporting Microservice Evolution, *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 539–543 (online), DOI: 10.1109/ICSME.2017.63 (2017).

[5] Munonye, K. and Martinek, P.: Evaluation of Data Storage Patterns in Microservices Archicture, *2020 IEEE 15th International Conference of System of Systems Engineering (SoSE)*, pp. 373–380 (online), DOI: 10.1109/SoSE50414.2020.9130516 (2020).

[6] Raghunath, B. R.: Virtual Machine Migration Triggering using Application Workload Prediction, *Procedia Computer Science* (2015).

[7] Krahn, R.: TEEMon: A continuous performance monitoring framework for TEEs, *Proceedings of the 21st International Middleware Conference* (2020).

[8] Andrews, K., Wohlfahrt, M. and Wurzinger, G.: Visual Graph Comparison, *2009 13th International Conference Information Visualisation*, pp. 62–67 (online), DOI: 10.1109/IV.2009.108 (2009).

[9] Sakai, M., Takahashi, K. and Kondoh, S.: Method of Constructing Petri Net Service Model Using Distributed Trace Data of Microservices, *2021 22nd Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pp. 214–217 (online), DOI: 10.23919/APNOMS52696.2021.9562589 (2021).

[10] Kida, T., Moreno, K. K. and Smith, J. F.: Investment Decision Making: Do Experienced Decision Makers Fall Prey to the Paradox of Choice?, *Journal of Behavioral Finance*, Vol. 11, pp. 21 – 30 (online), DOI: 10.1080/15427561003590001 (2010).

[11] Al-Samarraie, H. and Al-Hatem, A. I.: The effect of web search result display on users' perceptual experience and information seeking performance, *The Reference Librarian*, Vol. 59, No. 1, pp. 10–18 (2018).

[12] Pi, A., Chen, W., Zhou, X. and Ji, M.: Profiling Distributed Systems in Lightweight Virtualized Environments with Logs and Resource Metrics, *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '18, New York, NY, USA, Association for Computing Machinery, p. 168–179 (online), DOI: 10.1145/3208040.3208044 (2018).

[13] Yang, M. and Huang, M.: An Microservices-Based Openstack Monitoring Tool, *2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS)*, pp. 706–709 (online), DOI: 10.1109/ICSESS47205.2019.9040740 (2019).

[14] Chan, Y.-W., Fathoni, H., Yen, H.-Y. and Yang, C.-T.: Implementation of a Cluster-Based Heterogeneous Edge Computing System for Resource Monitoring and Performance Evaluation, *IEEE Access*, Vol. 10, pp. 38458–38471 (online), DOI: 10.1109/ACCESS.2022.3166154 (2022).

