Sock Shopの1秒間あたりのリクエスト数とサービス内Pod数の相関係数にもとづく表示順変更による利便性の向上

増田 和範[1]　大野 有樹[2]　串田 高幸[1]

概要：マイクロサービス内の Pod の CPU 使用量の数値データをグラフ化する．これにより各 Pod の CPU 使用量パターンを視覚的に理解できる．課題は各 Pod から出力される CPU 使用量を対象にグラフを作成する際，Pod 数に応じてグラフ数が増加することで必要なグラフを選択するまでの作業数が増加することである．提案手法は同時刻のサービス内の Pod 数と Sock Shop への 1 秒間あたりのリクエスト数の相関係数を求める．その値をサービスの表示優先度として用いる．これによりユーザは表示されたグラフを選択作業数を削減できる．実験は，Locust を用いて Sock Shop に HTTP リクエストを送信する負荷試験を実施した．これにより Pod から出力される CPU 使用量から 3 種類のグラフを作成した．1 秒間あたりのリクエスト数と Sock Shop 内の Pod 数，作成されるグラフ数の変化の検証を行った．実験前に生成されるグラフ数は 42 個であったが，負荷がかかり Pod 数が増加した際のグラフ数は 62 個増加した．

1. はじめに

背景

マイクロサービス内の Pod の CPU 使用量の数値データをグラフ化することには多くのメリットがある．一つは，情報を視覚的に表現することにより，データのパターンを理解することが可能になることである．また，複数のデータセットや大量の数値データを画像にできるため，データの比較や解釈を容易にすることができる．これにより，意思決定プロセスを迅速かつ効率的に進めることが可能となる [1].

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．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 [2].

Kubernetes 環境下のマイクロサービスでは，各サービスが独立して実装されている．これにより，各サービスは他のサービスから独立してスケーリングすることが可能となる．つまり，一部のサービスが大きな負荷を受けた場合でも，そのサービスだけを増強することで全体のパフォーマンスを維持することができる [3].

[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

Sock Shop は，マイクロサービスを採用した EC サイトのデモアプリケーションとして実験の対象として使用される．これは，靴下販売サイトを模したサービスである．EC サイトでのビジネストランザクションをそれぞれサービスとして独立して構成されている．このソフトウェアはオープンソースとして公開されているので，試験の対象として用いられる [4, 5].

サービスの監視方法として，リソースメトリクス監視がある．リソースメトリクスは例えとして CPU 使用率，メモリ使用量，ディスク使用率が挙げられる．これらのリソース使用状況を定期的に監視し，異常が発生した場合に早急に対応するための方法である．これにより，サービスのパフォーマンスを最適化し，ダウンタイムを防ぐことが可能となる[*1].

Kubernetes 環境で稼働するマイクロサービスは Pod ごとに前述した種類のリソースメトリクスを取得される．既存の監視，グラフ化ツールとして Prometheus と Grafana がある．これにより，各 Pod のリソース使用状況を個別に把握し，必要に応じてリソースを調整することが可能である [6]．他には，データの取得からグラフ作成までを Python で行うことができる．この言語のライブラリを用いることで Prometheus からリソースメトリクスを取得し，そのデータを元にグラフ化することが可能である．グラフ作成に関する設定 (縦横軸の単位，グラフの種類，フォントサイズ) をプログラムで設定することで，Pod ごとに出

[*1] https://onl.bz/6UWZa3W

力されるデータをすべて自動でグラフ化できる．

課題

マイクロサービスアーキテクチャである Sock Shop へ負荷試験をし，各 Pod の CPU 使用率をグラフ化する際の課題について詳述する．課題が発生想定する際のを図 1 に示す．負荷試験ツールから Sock Shop に対して HTTP リクエストを送信する．HTTP リクエストの内容はページにアクセスする際は GET，値を送信する際は POST を行う．このリクエストが Sock Shop のエンドポイントに送信され Sock Shop は処理を行う．処理の結果を負荷試験ツールに送信する．負荷試験ツールは送ったエンドポイントとリクエスト数，平均応答時間を fileA として出力する．

Sock Shop の監視に Prometheus を用いる．Prometheus は Sock Shop が稼働している Kubernetes クラスタの Kube API Server へアクセスし，クラスタ内の Pod ごとの CPU 使用量に関する値を保存する．保存された値のデータは Primetheus にクエリを送信することで fileB として出力される．

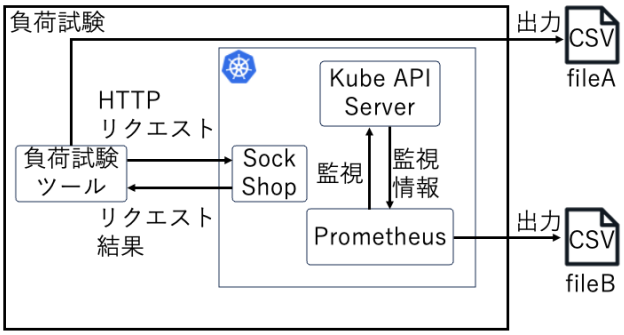

図 1　負荷テスト実施からデータ取得までの流れ

この取得データをグラフ化して分析する流れを図 2 に示す．図 1 にて出力された CSV ファイルをグラフ作成を行うために，ファイル内のデータを整形する．整形された CSV ファイルはプログラムによってグラフ作成される．作成されたグラフを確認する．Prometheus のメトリクスデータからグラフ作成を行う Grafana の場合，グラフ化するサービス名を補完する機能がある．その際，サービス名の候補はアルファベット順に表示される．これによりユーザは 1 つ 1 つのサービスに関する Pod を確認する必要がある．

Sock Shop のアーキテクチャを図 3 に示す．このように Sock Shop は 14 個のサービスによって構成されており，HTTP リクエストのエンドポイントとリクエスト内容 (GET， POST) によって使用するサービスが異なる．サービス名と HTTP リクエストが送信するエンドポイントが異なる．それにより，負荷試験のリクエスト結果からサービスを一意に決めることはその判断を下す人の主観による．

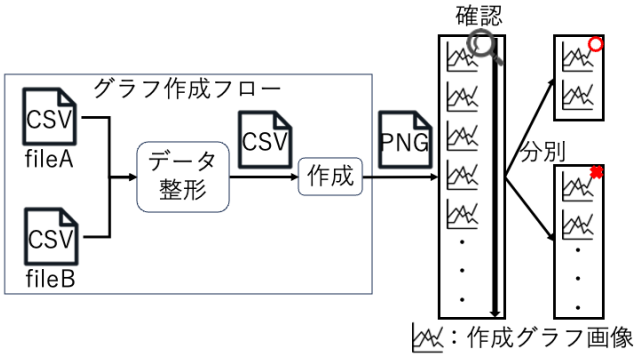

図 2　取得データをグラフ化

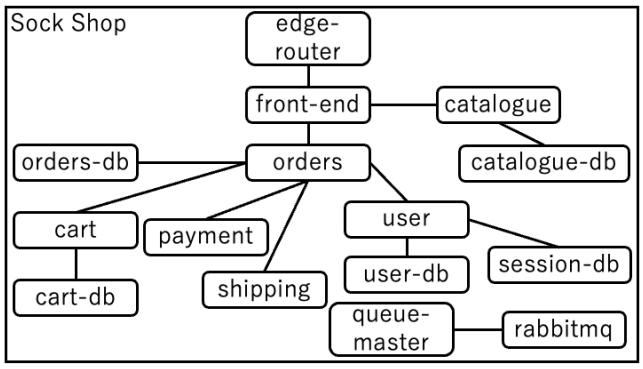

図 3　Sock Shop のアーキテクチャ

検討がつかない場合，システム管理者はアルファベットの降順に一つずつサービス内の Pod の作成グラフを確認する必要がある．しかし，Pod ごとに出力されるグラフ数はマイクロサービス内の Pod 数と作成するグラフによって増加する．

一つ目の原因は，Pod ごとにデータが出力されることである．そして Pod 数が一定ではなく，リクエスト (数，内容) による負荷に応じて増加することである．これにより，Pod の数に応じて，生成されるグラフの数が増加する．

二つ目の原因は，グラフの種類の多さである．円グラフ，棒グラフ，折れ線グラフなど多種多様な形式のグラフの選択肢があり，それぞれが異なる情報を視覚化するために最適化されている．しかし，Pod とグラフの個数の関係が 1 対 1 ではなく，1 対 多の関係性になる．これにより，作成されるグラフの個数がより増加する．

ユーザが多数のグラフの中から実験結果の分析に必要なものだけを選ぶことは，情報を適切に理解し伝えるために重要である．しかし，負荷試験の結果からグラフが自動的に生成され表示される場合，ユーザはそれらの中から最適なものを選ぶ．そのために先に述べた，表示されたグラフ画像を順番に確認する必要がある．この確認の作業は Pod 数と共に増加する．本稿での利便性はこの作業数を意味する．よってグラフ画像の選択作業工程が少ないほど利便性が向上するということである．

ここで選択作業数を定量化する．1 つのサービスについ

て確認する作業を 1 とする．例えば，サービスがアルファベット順に並んでいる (A サービスから F サービス)．この際に必要なサービスが D サービスだとすると，上から順に確認するため，該当サービスにたどり着くには 4 つのサービスを確認する必要がある．この場合，作業数は 4 になる．

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．

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 [7]. 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 [8]．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 [9]．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

本稿の提案の目的はユーザが表示されたグラフを選択する際の作業数を軽減することである．図 4 はマイクロサービス内の各サービスに対して優先度付けを行い，グラフの並び替えの流れを示す．サービス名のアルファベット順に表示されているグラフの画像を本稿の提案によって並び替えを行う．これによりユーザはサービスを手探りで探すのではなく，表示の順に従いグラフの選択作業を行うことが可能になる．

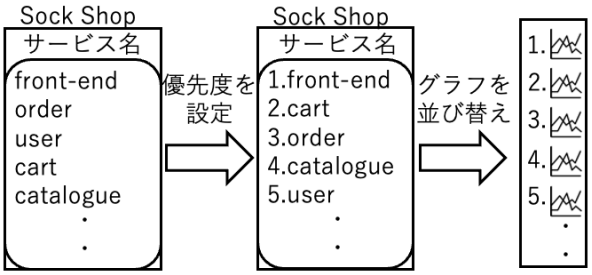

図 4 提案方式の概要

図 5 はレプリカ数と秒間リクエストの関係性を示す．サービスへ送信されるリクエストを処理すると負荷が増大する [10]．負荷を分散させるために Kubernertes ではサービスの Pod を HPA と呼ばれる機能によって増加させる[*2]．本稿の提案ではこのスケーリングされる Pod と秒間リクエスト数の関係性を求めることで表示順をサービスごとに作成する．

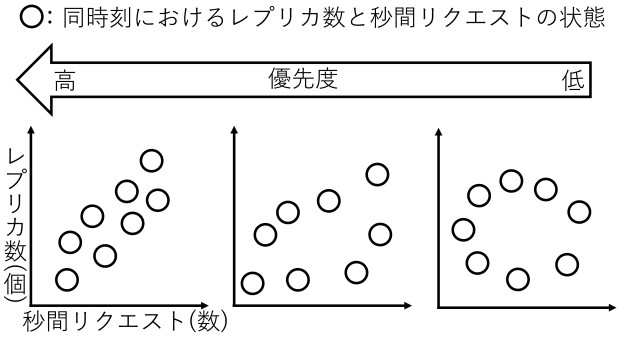

図 5 レプリカ数と秒間リクエストの関係性

図 6 はグラフ表示の順番を決定するまでの流れを示す．サービスごとの秒間リクエスト数と Pod 数を代入値としてピアソン相関係数を求める．相関係数が高いと表示優先度は高い．この相関係数を優先度としてこれに従ってユーザに対してグラフを表示する．WEB サービスへの HTTP リクエスト数と各 Pod の CPU 使用量は比例の関係がある [10]．そして使用可能な CPU 量不足により，応答時間が増加してしまうことがある [11]．これらより，CPU の使用量についてグラフ化を行う．この提案によるソフトウェアを PSSP(Pearson Sorted Service Primary) とする．

図 6 で用いるピアソン相関係数は以下の式によって求める式である．

[*2] https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/

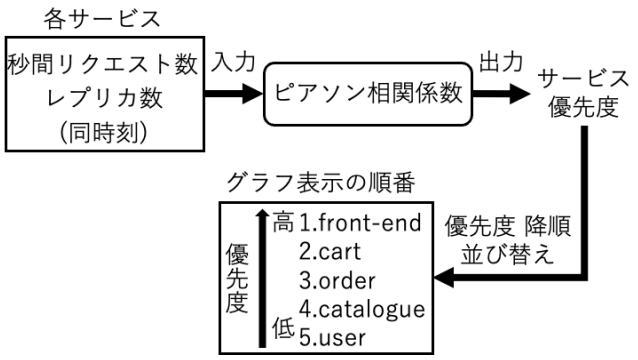

図 6　提案方式の流れ

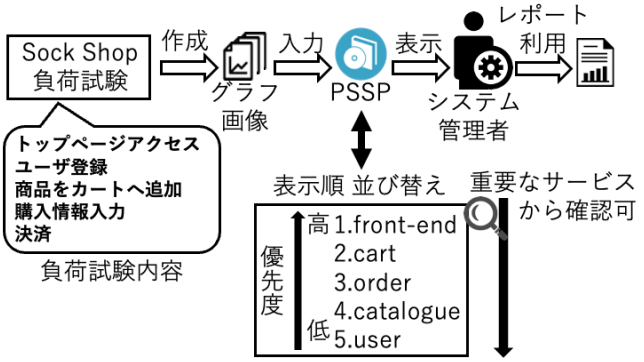

図 7　ユースケースの全体図

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

サービスごとの表示優先度にこの式から算出される相関係数を用いる．この数式はピアソン相関係数を計算するためのものである．この係数は，二つの変数間の線形相関関係の度合いを評価するために使用される．数値は-1 から 1 まで変動し，1 は完全な正の相関，-1 は完全な負の相関，0 は全くの無相関を表す．

具体的には，上記の数式では，x_i と y_i は各時刻における値で，それぞれサービス内の Pod 数と 1 秒間あたりのリクエスト数を表している．$\bar{x}$ と $\bar{y}$ はそれぞれ x_i と y_i の平均値を示す．n はデータ点の個数である．

分子は，各データ点の x の値と平均 $\bar{x}$ との差と，そのデータ点の y の値と平均 $\bar{y}$ との差の積の和を表す．これは「共分散」を求める公式の一部であり，二つの変数がどの程度一緒に変動するかを評価している．

分母は，各データ点の x の値と平均 $\bar{x}$ との差の二乗の和と，各データ点の y の値と平均 $\bar{y}$ との差の二乗の和の平方根を求める．これは，それぞれの変数が自分自身の平均値からどの程度乖離しているか，つまりその変数の「分散」を求め，その平方根を取ることで「標準偏差」を得る操作である．

これにより，ピアソンの積率相関係数 ρ は，x と y の共分散をそれぞれの標準偏差の積で正規化したものとなり，2 つの変数がどの程度一緒に変動するかを，それぞれの変数の変動の大きさで割って評価している．これにより，相関係数は必ず-1 から 1 の間に収まるようになる．しかし，リクエスト数が増加することにより，レプリカ数が減少しないので，0 から-1 の値を取らないとする．

ユースケース・シナリオ

本稿で想定するユースケースを示す図 7 について説明を行う．

Kuberntes 環境で稼働している Sock Shop に対して自分の提案アルゴリズムを用いてキャパシティプランニングを行う人を対象とする．

キャパシティプランニングにより，使用可能な CPU 量を設定することで特定のサービスがリクエストを処理できないまたは応答速度が遅くなることを防ぐことが可能になる [10, 11]．

キャパシティプランニングのアルゴリズム導入による効果を検証する．または導入前に, 負荷試験を行い，Sock Shop に対して HTTP リクエストを送信する．

これによる，CPU 使用量に関するデータは Pod ごとに生成されるため，Pod ごとにグラフを作成しても，グラフ数の個数の増加に比例して，必要なサービスの特定が困難になる．

グラフ化された画像はサービスごとに並び替えられ, ユーザは表示順から画像を確認することで負荷試験によるエラーが発生した HTTP リクエストに関するサービスのグラフを探す．

その際に本稿の PSSP(提案ソフトウェア) により，負荷テストから得られるリクエストの結果と CPU 使用量に関するファイルを読み込み，秒間リクエスト数に対してサービスの Pod 数の関係性から優先度を求め，求めた値をサービスごとに紐付ける．

優先度に従って表示されるグラフ画像によって対象者は無作為に探す必要がなく，必要なサービス内の Pod に関するグラフを確認することが可能になる．

以下に図 7 内の Sock Shop への負荷試験のシナリオについて説明を行う．

(1) Sock Shop のトップページにアクセス
(2) ユーザー情報の登録を行う．
(3) カタログページから商品をカートに追加.
(4) 決済に必要なクレジットカード情報の入力
(5) 上記の情報を元に決済を実行

4. 実装

実装の全体を図 8 に示す．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．

以下に実装したソフトウェアの各コンポーネントについて説明する．

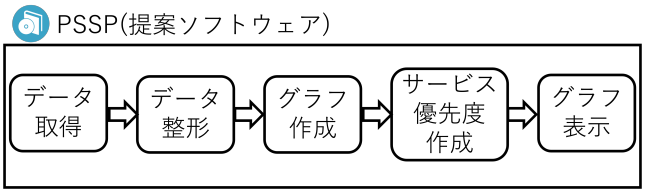

図 8　実装の全体図

データ取得

表 1 が示すように，Prometheus によって Kubernetes クラスタ内にデプロイされている Sock Shop から CPU に関する値を取得し Prometheus 内に保存する．Python のライブラリの Prometheus-api-client を使用し Prometheus にクエリを送信する．これにより，Sock Shop 内の Pod の CPU 量に関する値を取得する．取得した値は時系列データとして CSV ファイル形式 (表 1) で保存する．

表 1　ファイル内のフォーマット

Timestamp	Value	Pod-name

データ整形

データ取得で作成された CSV ファイル内にはすべての Pod の CPU 使用量が入力されている．グラフ化しやすくするために，サービス，Pod ごとにそれぞれ分けて CSV ファイルを作成する．

Python のライブラリの Pandas を使用することで統計処理と時系列データの扱いを容易に行える．

グラフ作成

整形された CSV ファイルを読み込みグラフを作成する．本稿の実装ソフトウェアは表 2 に示す 3 種類のグラフを作成する．折れ線グラフはサービス内の Pod ごとに作成されるグラフを 1 つのグラフ画像にまとめて作成する．他のヒストグラム，箱ひげ図は Pod ごとにグラフ画像を作成する．

表 2　作成するグラフの種類

グラフの種類	説明
折れ線グラフ	CPU 使用量の時系列データをプロットし，その点を直線で結んだもの．時間の経過とともにどのように CPU 使用量が変化するかを可視化可能.
ヒストグラム	CPU 使用量の頻度分布を可視化する．CPU 使用量の中央値や分布などを理解するために有用.
箱ひげ図	CPU 使用量の五数要約（最小値，第一四分位数，中央値，第三四分位数，最大値）を可視化する．データの分布や外れ値を理解するために使用.

サービス優先度作成

サービス内の Pod 数と秒間リクエスト数の間の相関係数を計算する．この相関係数は Pandas の corr メソッドを用いて計算され，値は 0 から 1 までの範囲を取る．相関係数が 1 に近い場合優先度が高くなる．逆に 0 に近い場合は優先度が低くなる．0 の場合，二つの変数間には相関がないことを意味する．算出された値はサービス名と辞書形式で紐づけされる．

グラフ表示

Flask を用いて WEB アプリケーションを開発した．このアプリケーションでは，事前に作成した各サービスのグラフを WEB ページ上に表示する機能が提供されている．表示するグラフはサービスごとに異なり，その表示順序はサービスの優先度に基づいている．具体的には，優先度が高いサービスのグラフから順に降順で表示されるように設計されている．

5.　実験

Sock Shop へ Locust を用いた負荷試験を行う．これにより 1 秒あたりの HTTP リクエスト数と Pod 数の増加の関係を検証した．そしてそれに伴い，PSSP(提案ソフトウェア) を用いた場合，作成されるグラフ数の変化を検証した．

実験環境

課題の検証を行う基礎実験である．図 9 が示す環境のもと実験を行う．使用する負荷試験ツールは Locust を使用する．図 7 が示すシナリオのもと負荷試験を行う．

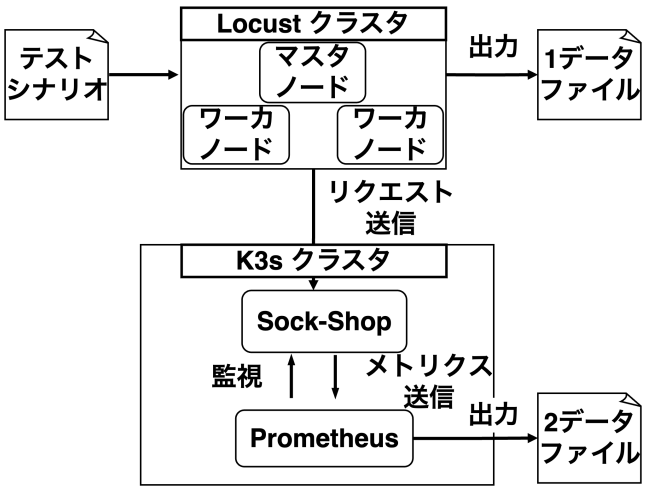

図 9　実験の全体図

テストシナリオを Locust に読み込ませ負荷試験を実行する．Locust で負荷試験を行う際に設定する 3 つの項目 (最大ユーザ数，ユーザの生成割合，テスト時間) の説明を 3 に記載する．これらの条件で K3s のクラスタ内の Sock Shop に HTTP リクエストを送信する負荷試験を行う．

Sock Shop の CPU 量を監視するために Prometheus を使用する．HTTP リクエストの結果と Sock Shop 内の Pod の CPU 使用量，それぞれの CSV ファイルを出力する．このファイルから 3 種類のグラフを作成する．作成されたグラフ数が Sock Shop 内の Pod 数の増加に応じてどれぐらいの数に増加するか検証を行う．

表 3　ユーザ数と生成割合の関係

最大ユーザ数 (s)	ユーザの生成割合 (人/秒)	テスト時間 (s)
300	0.4	1000

実験結果と分析

図 10 は経過時間と秒間リクエスト数を折れ線グラフで表したグラフである．Elasped Time は経過時間，Requests/s は 1 秒間あたりのリクエスト数を示す．

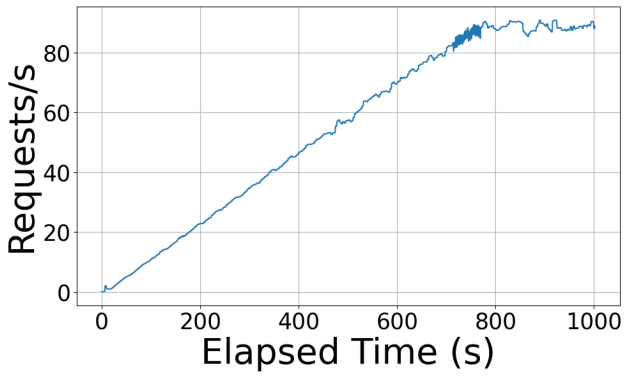

図 10　RPS の時系列グラフ

11 は経過時間ごとの Pod 数と作成されるグラフ数を積み上げ棒グラフで表したグラフである．

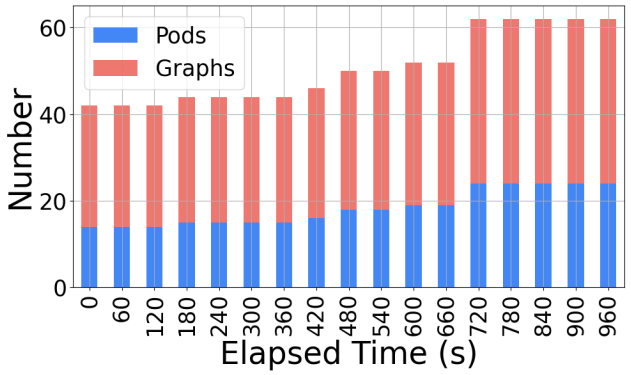

図 11　Pod 数とグラフ数の棒グラフ

これらの結果からサービスへのリクエスト数が増加することでマイクロサービス内の Pod が増加することが確認できる．Pod の増加率は，リクエスト数とリクエスト内容に依存する．具体的には，リクエスト数が多い場合や処理負荷の高いリクエストが増えると，より多くの Pod が必要となり，その結果，Pod の増加率が上昇する．これは，マイクロサービスがスケーラビリティと性能の調整を目的として設計されているためである．

この結果により，Sock Shop へリクエスト数が増加する場合，各サービス内の Pod が増加し，それに応じて作成されるグラフ数が実験前では 42 個のグラフが作成され，Pod 数が最大の時は 62 個と増加することを検証した．

6.　議論

本稿はサービスの優先度付けを行う新たな指標を提案した．具体的には各サービス内の Pod のレプリカ数と Sock Shop への 1 秒間リクエスト数とのピアソン相関係数を求めた．この値をサービスの優先度として用いる．この値が値と優先度が高くなる．その優先度に従ってグラフの表示順を変更することで，ユーザーは各サービスの負荷状況を効率的に把握することが可能である．

課題でも述べたが，マイクロサービスは Pod ごとに CPU 使用量のデータを出力される．サービスでの表示順を変更するだけではなく，より細かな粒度である Pod の情報から表示順の変更をすることが必要である．サービス内の Pod 数と 1 秒間リクエスト数の相関関係を求めるのではなく，Pod 単位の優先度を決めるには別の指標 (CPU 使用率) を Pod 数の代わりに適用することで，Pod ごとの表示する際の優劣を決めることができる可能性がある．

また，エラーが起きたリクエストに対するサービスを優先的に表示することを目的としたがその際にサービス内の Pod 数を用いた場合，Sock Shop のデータベースのサービスでは Pod のスケーリングの設定をしていない．それによりリクエスト数と相関係数が 0 と算出されるので本提案ソフトウェアでは優先度が最低になる．

本提案は HTTP リクエストによって発生したエラーに関するサービスを優先的に表示することを目的としている．しかし，エラーが発生する際，経路にあるサービスの front-end の優先度が最上位になってしまう．解決策として，事前にリクエストが処理されるサービスの経路を把握することでサービスを限定する．そのために Istio と Jaeger を用いた HTTP リクエストの処理経路を確認する．Istio によってマイクロサービス間の通信に関するデータを自動的に収集し、Jaeger に送信する．Sock Shop のエンドツーエンドのリクエストトレーシングを行うことで，リクエストがシステム全体でどのように流れるかを追跡する．事前に Sock Shop のすべてのエンドポイントに HTTP リクエストを送信する．Istio を用いてサービス間の処理の流れを把握する．エンドポイントごとの HTTP リクエストと経路を辞書型で事前に保存することにより，サービスを限定する．

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，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 と 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アンケート調査により求めることで改善できる．

7. おわりに

本提案はユーザに表示するグラフを選択する際の作業数を軽減することを目的としている．マイクロサービス内の各サービスに対して優先度付けを行う．それによりユーザはサービスを手探りで探すのではなく，表示の順に従いグラフの選択作業を行うことで作業数の削減が可能になる．課題は各 Pod から出力される CPU 使用量を対象にグラフを作成する際，Pod 数に応じてグラフ数が増加することで必要なグラフを選択するまでの作業数が増加する．実験は，Locust を用いて負荷試験を実施した．1 秒間あたりのリクエスト数と Sock Shop 内の Pod 数，作成されるグラフ数の変化の検証を行った．実験前に生成されるグラフ数は 42 個であったが，負荷がかかり Pod 数が増加した際のグラフ数は 62 個増加した．相関関係の対象がサービスであるので，より細かな粒度の Pod 内の CPU 使用量を用いて Pod ごとの相関係数を求めることで表示順を変更できる．グラフの表示方法は改善の余地がある．

謝辞 本テクニカルレポートの執筆にあたりご助言を賜りました東京工科大学コンピュータサイエンス学部の平尾真斗さんに感謝申し上げます．

参考文献

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

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

[3] Autili, M.: A Hybrid Approach to Microservices Load Balancing (2019).

[4] Rahman, J. and Lama, P.: Predicting the End-to-End Tail Latency of Containerized Microservices in the Cloud, *2019 IEEE International Conference on Cloud Engineering (IC2E)*, pp. 200–210 (online), DOI: 10.1109/IC2E.2019.00034 (2019).

[5] Aderaldo, C. M., Mendonça, N. C., Pahl, C. and Jamshidi, P.: Benchmark Requirements for Microservices Architecture Research, *2017 IEEE/ACM 1st International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering (ECASE)*, pp. 8–13 (online), DOI: 10.1109/ECASE.2017.4 (2017).

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

[7] 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).

[8] 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).

[9] 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).

[10] 一俊坂本, 佳城伊藤, 高幸串田：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 44，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, 東京工科大学コンピュータサイエンス学部 (2023).

[11] Shobana, G., Geetha, M. and Suganthe, R. C.: Nature inspired preemptive task scheduling for load balancing in cloud datacenter, *International Conference on Information Communication and Embedded Systems (ICICES2014)*, pp. 1–6 (online), DOI: 10.1109/ICICES.2014.7033816 (2014).