レスポンスタイムの維持のための異なるファイル名とハッシュ値を用いたプログラムのコピー

野木 空良[1]　飯島 貴政[2]　串田 高幸[1]

概要：マイクロサービスアーキテクチャは，アプリケーションを機能単位で分割して実行, 管理する. マイクロサービスに対しリクエストが急増した際，サービス管理者はマイクロサービスのスケールアウトを行う．しかし，マイクロサービスにリクエストが急増した際，スケールアウトによって複製されたマイクロサービスは，起動中に処理を実行しない為，その間のレスポンスは遅延する．またマイクロサービスのスケールアウトを行ってから次のスケールアウトまでインターバルが発生する．インターバル中にリクエスト数が増加し続けるとレスポンスは遅延する．本研究ではプログラムのファイル名とハッシュ値をそれぞれのマイクロサービスで比較し，同一ファイル以外のプログラムのコピーを行うソフトウェア「DocCP」を提案する. これにより, スケールアウトのインターバル中に，別のマイクロサービスがリクエストの応答を肩代わりし，レスポンスタイムの維持が可能となる. 評価では「DocCP」適用後のマイクロサービスと，オートスケール機能を用いたマイクロサービスのレスポンスタイムを比較し, 分析・評価を行う.

1. はじめに

背景

マイクロサービスでは，アプリケーション機能を分割している．一単位の機能に対して，ユーザーのアクセス数が増加すると負荷が上昇し，SLO(Service Level Objective)に違反する場合がある [1]．SLO の違反発生は，マイクロサービスの核となるアプリケーションの CPU やメモリの過剰な使用により，リクエストの処理に時間がかかるのが原因である [2]．SLO 違反の対策として，単体のアプリケーションを複製し，使用リソースの分散を行うスケールアウトという機能がある [3]．マイクロサービスのスケールアウト方法を図 1 に示す．マイクロサービスでは，個々の機能をコンテナに格納し，独立したマイクロサービスとして実装しているため，「機能 A」の負荷が上昇した際にコンテナ単位でのスケールアウトが可能である．サービス管理者はマイクロサービス単位でのスケールアウトにより，リクエスト数の分散を図る．Amazon ECS ではクラスター及び，マイクロサービスのメモリと CPU 使用率をモニタリングし，リソースが不足した際に，クラスターを自動的にスケールアウトする機能がある [4]．マイクロサービス内部のメトリクスを取得し，CPU やメモリの使用率が減少し，スケールアウトをする必要がなくなった場合に自動でスケールを戻す．スケールアウトの機能により，時間の経過によって動的に変化する負荷を予測し，SLO 違反の対策を行っている．

図 1　マイクロサービスのスケールアウト

課題

マイクロサービスにリクエスト数が急増した際, マイクロサービス単位でスケールアウトを行う. スケールアウトにより, マイクロサービスの CPU やメモリの消費を分散

[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

できる. しかし, スケールアウトはマイクロサービスの起動に時間がかかり, リクエストの分散が間に合わない．その為，スケール時にかかる時間を考慮した閾値を設定する必要がある [5]. 例えば, リクエスト数の分散を間に合わせる為，閾値の値を CPU 全体の 10 %に達した時にスケールアウトを行うように設定する．しかし閾値を 10 %に設定したことにより，リクエストが急増し続けた場合でも急増の最中にスケーリングを行ってしまう．その結果，次のスケールアウト実行までにインターバル (AWS の場合デフォルトで 300 秒) が発生し，その間のレスポンスタイムの維持は不可能となる [6]．スケールのタイミングが遅れた場合, リクエスト分散が間に合わなくなり, レスポンスの遅延が生じる [7]．本研究では，スケールアウトのインターバルによるレスポンスの遅延を再現する為に基礎実験を行った．HTTP リクエスト 1 つにつき 2 の 6 乗の計算を返す Pod に負荷実験を行った際のレスポンスタイムのグラフを図 2 に示す．横軸は実験開始時間 (秒)，縦軸は Pod のレスポンスタイム [ms] とリクエスト毎秒を表している．

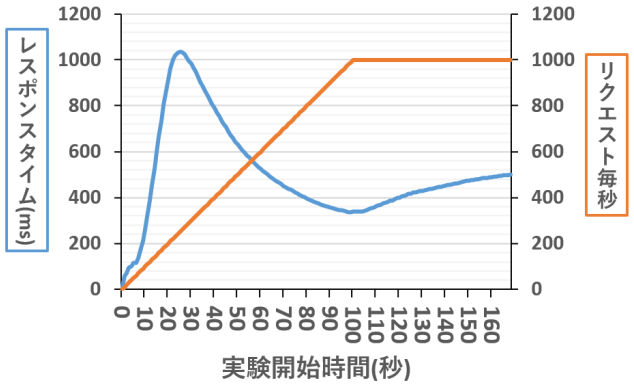

図 2 Pod のレスポンスタイム

図 2 では，HTTP リクエストを送信する Locust[*1]を用いて，毎秒 10 リクエスト上昇させ 1000 リクエスト以降は一定のリクエストを送信する実験を行った．Pod の CPU の上限は 500[mcore] とし，CPU 使用率が 10 %を超えるたびに，スケールアウトを行い最大 100 個の Pod が起動する．Pod のレスポンスタイムは 30 秒まで上昇し続け，最終的に約 1000[ms] まで上昇した．その後，100 秒までレスポンスタイムは約 400[ms] まで減少し，以降のレスポンスタイムは 500[ms] まで上昇した．実験の結果，リクエスト数が増加し Pod のスケールアウトの閾値を 10 %に設定した場合でもレスポンスタイムは急上昇することが分かる．スケールアウトによるレスポンスタイムの遅延を図 3 に示す．

図 3 は Web チケット販売サービスの例である．Web チケット販売サービスではユーザー数が不定期に増加する [8]．例えば，ハロウィンやクリスマスといった行事に向けて，テーマパークのチケットを購入するユーザーが増

*1 https://docs.locust.io/en/stable/

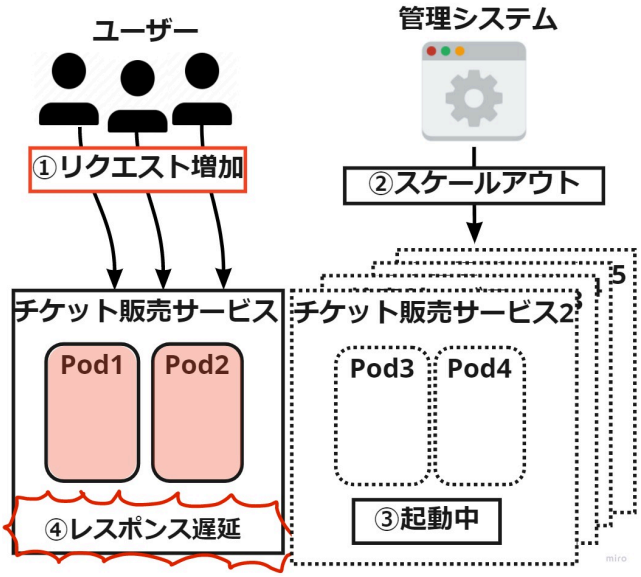

図 3 スケールアウトによるレスポンスの遅延

加する．その際に，チケット販売サービスのリクエスト数が急増し，管理システムはマイクロサービス単位のスケールアウトを行う．しかし，スケールアウトによって追加されたチケット販売サービスは，起動している最中のリクエストの処理を受け付けない為，その間はレスポンスの遅延が生じる．レスポンスの遅延は，web サイトの直帰率の原因により，アクセスするユーザー数の減少につながる [9]. よって，スケールアウトによるサービス起動中のレスポンスタイム維持が必要である．以下にマイクロサービスに対してリクエストが急増した際の，スケールアウトによるレスポンスの遅延が発生するまでの流れを示す.

(1) ユーザー数の増加により，チケット販売サービスのリクエスト数が急増
(2) マイクロサービスの管理システムはリクエスト数増加に伴い，スケールアウトを行う
(3) 追加したチケット検索サービス n 個 (n=2，3，4...) の起動
(4) マイクロサービスの起動中または，スケールアウトのインターバルによりレスポンスが遅延

各章の概要

2 章では，本研究と関連した既存研究について述べる．次に，本研究の提案を 3 章で述べる．4 章では，実装と実装環境について述べる．5 章では，提案内容の分析方法とどのように評価するかを述べる．6 章では，本研究の提案をもとに議論する．最後に，本論文の取り組みと貢献について簡潔に述べる．

2. 関連研究

マイクロサービスのオートスケールによる時間を短縮する既存の取り組みがある．オートスケールを行う為のメト

リクスの閾値を決める研究では，サーバーの状態を監視し，スケールの閾値を動的に変化させることで，CPU 使用率やメモリ使用率の分散を可能にしている [10]．メトリクスの閾値の決定は，マイクロサービスの状態に依存する為，全てのマイクロサービスに適応させることはできない．また，取得するメトリクスを決定する必要がある為，取得するべきメトリクスもマイクロサービスによって変化する．

マイクロサービスに，オートノミックコンピューティングをアプリケーションに適用する研究では，サーバーの環境や状況の変化を記憶し，自己修復を行うオートノミック機能を搭載し，オートスケールのタイミングを動的に変更している [11]．オートスケールはサーバー台数を追加する為，スケールを行うほどコストが増加する．また，サーバー台数が増加すれば，ハードウェア上の CPU やメモリが圧迫する為，ハードウェアを複数企業で運用するマルチテナントの場合，CPU 使用率やメモリ使用率は削減する必要がある．

さらに，サーバーを意識せずにシステムを運用が可能となるサーバーレスについての研究では，サーバーの起動におけるセットアップの時間を課題としている [12]．ホットスタンバイにより，事前にサーバーを用意しておくことで，起動の際の，セットアップの時間を削減している．これにより，オートスケールによる起動時間の遅延を改善し，分散タイムラグの解決をした．しかし，オートスケールは，サーバーの運用する台数や使用する環境に依存する．例えば，サーバー台数が 1 台の時は，急激なリクエスト数増加に対して，素早くスケールアウトする必要がある．さらに，サーバーを新しく追加する為，コストの増加を考慮する必要がある．そのため，サービスにリクエスト数が急増した際の，レスポンスの遅延を改善するには十分でない．

3. 提案

本稿では，マイクロサービス起動時にサービス管理者が実行ファイルを別機能のマイクロサービスにコピーし，1 つのマイクロサービスにリクエスト数が急増した際に，リクエスト数を分散するソフトウェア「DocCP」を提案する．また本提案の流れを以下に示す．

(1) 各マイクロサービスで予めプログラムのハッシュ値を算出したリストを「DocCP」に送信
(2) 比較元リストと比較先リストのハッシュ値を比較
(3) ハッシュ値が同一のプログラムが存在していた場合，リストから削除をする
(4) 比較元に存在し，比較先に存在しないプログラムを取得し各マイクロサービスにコピーをする

機能の異なるマイクロサービスでプログラムをコピーする手法を図 4 に示す．図 4 は，Web でチケットを販売しているサービスの例である．まず，予めマイクロサービス内でリスト化したプログラムのハッシュ値を「DocCP」が取得する．チケット検索サービスのプログラムをチケット購入サービスとチケット決済サービスにコピーする場合，チケット検索サービス側のハッシュ値リストを 2 つのマイクロサービスと比較する．同一ハッシュ値のプログラムが存在した場合，リストから削除しコピーをするプログラムの削減を図る．比較後のハッシュ値リストから，必要なプログラムをチケット検索サービスから取得し，チケット購入サービスとチケット決済サービスにコピーを行う．

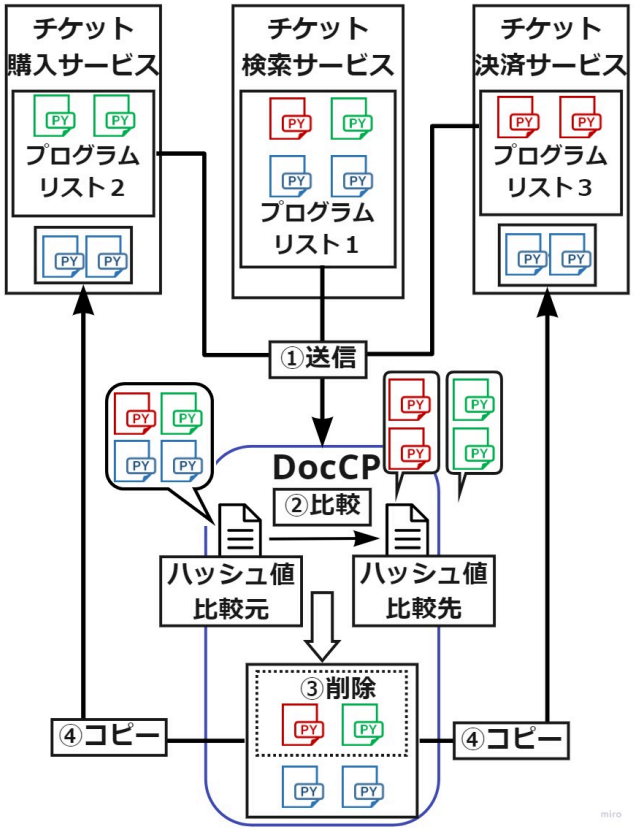

図 4 ソフトウェア「DocCP」によるプログラムコピー手法

チケット検索サービスとチケット購入サービスのハッシュ値をリスト化したファイルの比較手法を図 5 に示す．

チケット検索サービスでは，Web に検索内容を表示する「web.py」と，ユーザーのリクエストからチケット情報を取得する「search.py」が実装されている．また，チケット購入サービスでは，購入結果を Web に表示する「web.py」と，ユーザーのリクエストからチケットの購入手続きを行う「buy.py」が実装されている．2 つのマイクロサービスで実行されている Python ファイルのハッシュ値を取得し．リスト化したファイルを，チケット検索サービスでは「Search-List.csv」，チケット購入サービスでは「Buy-List.csv」とする．本稿では，「web.py」のハッシュ値を「α」，「search.py」のハッシュ値を「β」，「buy.py」のハッシュ値を「γ」と置く．リストの作成方法はファイル名を Key，ハッシュ値を Value とした連想配列で行う．まず「Search-List.csv」の「web.py」を「Buy-List.csv」の

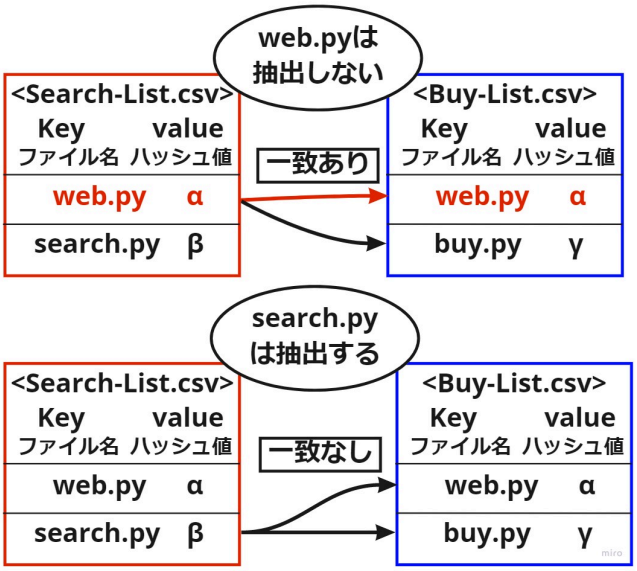

図 5 リスト化したハッシュ値の比較例

全ての Key Value と比較する．「web.py」のハッシュ値が，「Buy-List.csv」内のファイルと一致した場合，「web.py」は除外する（$\alpha = \alpha$，$\alpha \neq \gamma$）．また，「search.py」のようにハッシュ値が「Buy-List.csv」内の全てのファイルと一致しなかった場合，「search.py」は抽出する（$\beta \neq \alpha$，$\beta \neq \gamma$）．「Search-List.csv」と「Buy-List.csv」の全てのファイルで比較を行い，抽出するファイルを選定する．ハッシュ値の一致したファイルの除外により，既にお互いのマイクロサービスで存在している Python ファイルのダウンロードをせず，オーバーヘッドの減少が可能となる．

3.1 ユースケースシナリオ

本研究のユースケースは，web チケット販売のマイクロサービスである．ユースケースシナリオの概要を図 6 に示す．

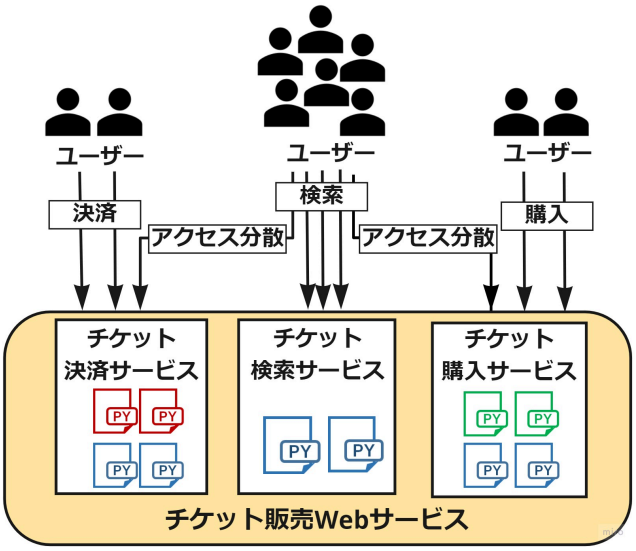

図 6 ユースケースシナリオ

例えば，チケット検索サービスにリクエストが集中した際に，検索のリクエストをチケット購入サービスとチケット決済サービスに分散させることで，レスポンスタイムの維持を図る．また，web チケット販売ではマイクロサービスの利用上昇率が 1 つのマイクロサービスの設定した CPU 上限設定値を超えることで，レスポンスが遅延する．ここでスケールアウトによる起動中の時間を，「DocCP」によりプログラムのコピーを行い，アクセスの集中していないマイクロサービスにリクエスト数を分散させることで，レスポンスタイムの維持が可能となる．

4. 実装と実験環境

4.1 実装

プログラム共有ソフトウェア (DocCP)

本提案「DocCP」のソフトウェアアーキテクチャを図 7 に示す．

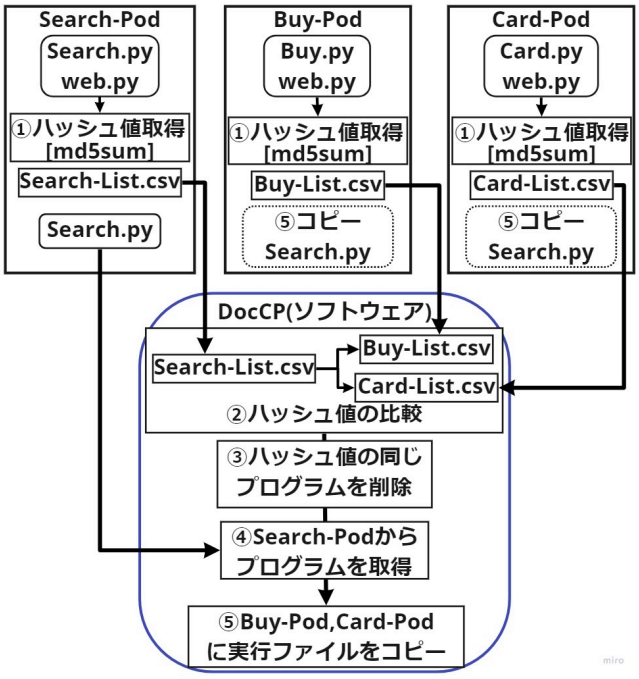

図 7 ソフトウェアアーキテクチャ

また，「DocCP」による実行ファイルコピーの流れを以下に示す．

(1) 各 Pod の Python ファイルを md5sum[*2]を用いてハッシュ値を取得し csv ファイル化する
(2) 「DocCP」は csv ファイルを取得し，ハッシュ値が一致しているかの比較を行う
(3) ハッシュ値が一致しているリストが存在した場合，削除を行う
(4) 比較後のリストを基に，Search-Pod から必要な Python ファイルを取得する

[*2] https://datatracker.ietf.org/doc/html/rfc1321.html

(5) 「DocCP」は Buy-Pod，Card-Pod に Python ファイルをコピーする

「DocCP」は Python を用いて開発を行う．「DocCP」を実行した際にハッシュ値の比較からマイクロサービスに Python ファイルをコピーするまで自動で行う．まず，「DocCP」の実行時に予めハッシュ値を取得したファイル「Search-List.csv」「Buy-List.csv」「Card-List.csv」を「DocCP」が取得する．ハッシュ値の取得は，md5 コマンドを用いる．次に，転送した「Search-List.csv」と「Buy-List.csv」と「Card-List.csv」のハッシュ値を比較し，除外と抽出を行う．比較後のリストを基に kubernetes のコマンドである「kubectl cp」コマンドを実行し，Buy-Pod と Card-Pod に存在しない Python ファイル (search.py) を「DocCP」が取得する．取得した Python ファイル (search.py) を Buy-Pod と Card-Pod にコピーを行うまでを「DocCP」により自動で行う．コピー方法は，「Search-List.csv」や「Buy-List.csv」の取得方法と同一のコマンド「kubectl cp」を用いて行う．

4.2 実験環境

本研究で用いたシステムアーキテクチャを図 7 に示す．Ubuntu 18.04.2 のノード上に Docker 20.10.7 をインストールし，Docker 環境の使用を可能にする．また，microk8s をインストールすることで Kubernetes 環境の使用を可能にする．プログラムコピーを行うソフトウェアの「DocCP」は Python 言語を用いて作成する．本実験は，サービス内のコンテナを Kubernetes の Pod を用いて開発する．評価分析の際は，負荷試験ツールの Locust を用いて，HTTP リクエストを送信する．

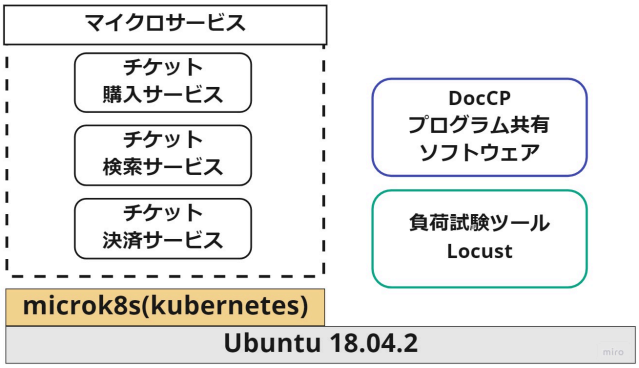

図 8 システムアーキテクチャ

5. 評価と分析

本実験では，Locust がリクエストを送信してからレスポンスが返ってくるまでの時間をレスポンスタイムと定義し，実験を行う．評価では，プログラムコピー後にリクエストを分散させたマイクロサービスとプログラムコピー前のマイクロサービスのレスポンスタイムを比較する．マイクロサービスは，論文掲載サービスである「Doktor」を用いる．また，コピーを行うマイクロサービスの数は 3 つとする．リクエスト数の分散方法として，Kubernetes のコンポーネントである Service のロードバランサーを用いる．CPU の上限設定値が予め設定された「Doktor」を用いて，CPU が上限設定の値に達した場合，Service のラベルにプログラムコピー後のマイクロサービスを追加することで，リクエスト数の分散を行う．取得したレスポンスタイムをもとに，提案手法により，対象のマイクロサービスのレスポンスタイムの維持を実現できたか評価する．

6. 議論

本提案では，機能の異なるマイクロサービス上のプログラム (Python ファイル) のコピーを行うが，実行させる為に，依存ライブラリをインストールする必要がある．各 Python ファイルで使用している import 文を列挙し，インストールするべきモジュールをマイクロサービスどうしで比較，抽出する．これによりインストールするべきモジュールの把握が可能となる．

また，本提案はプログラムのコピーを行うが処理を肩代わりするには，リクエストの振り分けを行う必要がある．リクエストの振り分け方法として，Kubernetes の API を用いる．ロードバランサーの yaml ファイルにあるラベルを分散先のマイクロサービスに変更することでリクエストの振り分けが可能になると考える．分散のアルゴリズムとして，動的分散方式を用いて，CPU 使用率が一番低いマイクロサービスに対して，リクエストを送信することで，処理の肩代わりを行うことが出来ると考える．

プログラムコピーの際に，同じファイル名で違うハッシュ値のプログラムを比較する場合，ハッシュ値が違うためプログラムのコピーを行う．その際に，同一のディレクトリにコピーしてしまうとファイル名が一致していることからオーバーライドが発生し，プログラムの上書きを行ってしまう可能性がある．そのためプログラムをコピーする際にコピー用のディレクトリを新しく用意する必要がある．コピーを行う先を別々のディレクトリにすることでオーバーライドを発生させずに，プログラムのコピーを行うことが出来ると考える．

7. おわりに

本研究では，マイクロサービスにおけるリクエスト数が急増した際のレスポンスの遅延を課題とした．既存の手法として，サーバー台数を増やす，スケールアウト機能があるが，マイクロサービスの起動時間を考量する必要があり，その間のレスポンスタイムは維持することはできない．そこで，異なるマイクロサービス同士のプログラムコピーを行い，実行の代替をし，リクエスト数の分散を行うソフト

ウェアの「DocCP」を提案した．本提案の手法を用いることで，スケールアウト時間を考慮せず，マイクロサービスに対するリクエスト数の分散を行い，レスポンスタイムの維持が可能となる．今後の作業として，本提案の手法と，スケールアウト機能を使った手法のレスポンスタイムを比較し，評価分析を行う．

参考文献

[1] Avritzer, A., Ferme, V., Janes, A., Russo, B., van Hoorn, A., Schulz, H., Menasché, D. and Rufino, V.: Scalability assessment of microservice architecture deployment configurations: A domain-based approach leveraging operational profiles and load tests, *Journal of Systems and Software*, Vol. 165, p. 110564 (2020).

[2] Saurabh, H. Q. S. S. B., Kalbarczyk, J. Z. T. and Iyer, R. K.: FIRM: An Intelligent Fine-Grained Resource Management Framework for SLO-Oriented Microservices, *ile*, Vol. 400, p. 800.

[3] Filho, R. R., de Sá, M. P., Porter, B. and Costa, F. M.: Towards emergent microservices for client-tailored design, *Proceedings of the 19th Workshop on Adaptive and Reflexive Middleware*, pp. 1–6 (2018).

[4] Jung, M., Mallering, S., Dalbhanjan, P., Chapman, P. and Kassen, C.: Microservices on AWS, *Amazon Web Services, Inc., New York, NY, USA, Tech. Rep* (2016).

[5] Bunch, C., Arora, V., Chohan, N., Krintz, C., Hegde, S. and Srivastava, A.: A pluggable autoscaling service for open cloud PaaS systems, *2012 IEEE Fifth International Conference on Utility and Cloud Computing*, IEEE, pp. 191–194 (2012).

[6] Podolskiy, V., Jindal, A. and Gerndt, M.: Iaas reactive autoscaling performance challenges, *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, IEEE, pp. 954–957 (2018).

[7] Kanagala, K. and Sekaran, K. C.: An approach for dynamic scaling of resources in enterprise cloud, *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, Vol. 2, IEEE, pp. 345–348 (2013).

[8] 貴志爲重，良史高本，直樹宇都宮：多数サーバ環境におけるサービスレベル向上技術 (2)，第 66 回全国大会講演論文集，Vol. 2004, No. 1, pp. 5–6 (2004).

[9] Dolma, Y., Kalani, R., Agrawal, A. and Basu, S.: Improving Bounce Rate Prediction for Rare Queries by Leveraging Landing Page Signals, *Companion Proceedings of the Web Conference 2021*, pp. 1–6 (2021).

[10] Rossi, F., Cardellini, V. and Presti, F. L.: Self-adaptive Threshold-based Policy for Microservices Elasticity, *2020 28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, IEEE, pp. 1–8 (2020).

[11] Florio, L. and Di Nitto, E.: Gru: An approach to introduce decentralized autonomic behavior in microservices architectures, *2016 IEEE International Conference on Autonomic Computing (ICAC)*, IEEE, pp. 357–362 (2016).

[12] Gunasekaran, J. R., Thinakaran, P., Nachiappan, N. C., Kannan, R. S., Kandemir, M. T. and Das, C. R.: Characterizing Bottlenecks in Scheduling Microservices on Serverless Platforms, *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, IEEE, pp. 1197–1198 (2020).