二分探索法を用いたKubernetes Pod数の決定の自動化による応答時間の遅延の低減

富田 啓太[1]　中川 翔太[2]　串田 高幸[1]

概要：ユーザーからチケット販売サイトへのリクエスト数が増加すると応答時間の遅延が発生する．応答時間が 3 秒となるとユーザーの直帰率が上昇する．応答時間の遅延の解決策として Kubernetes では Horizontal Pod Autoscaler(HPA) と呼ばれる Pod のオートスケール機能が存在する． しかし既存 HPA は CPU 使用率を基にしたスケールアウトしか対応しておらず，応答時間を使用したスケールアウトには対応していない．二分探索法を用いた Pod のスケールアウトを自動決定するアルゴリズムを提案する．Pod は WordPress で構築したチケットサイトをデプロイする．Pod の応答時間に基づいてスケールさせる Pod 数を決定し，待ち行列理論を用いて応答時間のシミュレーションを行う．これにより，遅延した応答時間の低減を実現する．

1. はじめに

背景

大規模な EC サイト（例:Alibaba）は，ユーザーからの大量のリクエストにサービスを停止させることなく処理することが求められる [1]．ユーザーへの応答時間の遅延や Web サービスの停止するおそれがあるためである．2018 年 11 月 11 日，Alibaba では 1 秒間に最大約 49 万の注文処理が発生し，データベースの Transactions Per Second が急増した [1]．EC サイトに類似した事例としてチケットサイトがある．チケットサイトの特徴として大量のユーザーからのリクエストが発生しやすい [2]．2008 年の北京オリンピックのチケットサイトでは毎秒 800 万件のリクエスト，20 万件の注文が発生し[*1]，チケット予約システムが停止した [3]．このような Web サービスの応答時間の遅延や停止は，売上にも影響を与える．EC サイトの Amazon は応答時間が 0.1 秒の遅延で売上の 1%売上が減少を確認した [4]．このことから Web サービスの応答時間の遅延や停止は売上の減少に直結する．

これらの課題に対応する Kubernetes の機能として Horizontal Pod Autoscaler(HPA) がある [5]．HPA とは，CPU 使用率に基づいて Pod をオートスケールさせる機能である．HPA を用いることで，予期しないユーザーからのリクエストに対して Pod をスケールさせリクエストを分散させることができる．負荷を分散させることで応答時間の遅延，Web サービスの停止の発生率を下げながらサービスの提供を行える．

課題

本稿で上げる課題はアクセス集中によるアプリケーションである Pod の応答時間の遅延である [6]．図 1 が課題の概要である．

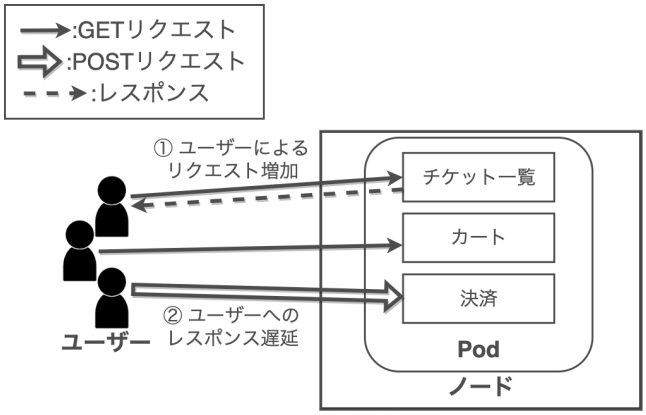

図 1　課題の概要

ユーザーはアプリケーションである Pod の利用者を表しており，Pod に対して HTTP リクエストを送信している．ユーザーからの HTTP GET リクエストや HTTP POST リクエストが集中すると，応答時間の遅延が発生する．応

[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
[*1] https://www.cnet.com/culture/olympics-ticket-system-crashes

答時間の遅延は 3 秒以上となるとユーザーの直帰率は 50% を超える[*2]．そのため HPA を用いて Pod をスケールアウトさせ，リクエストを分散させる必要がある．しかし，HPA は CPU 使用率に基づいたスケールしか行わない．CPU 使用率のみでは，応答時間が改善するのかがわからず不十分である．本稿では，応答時間に基づいた Pod の自動スケールアウトを行い，遅延した応答時間の低減を行う．

次の章では，課題の証明として基礎実験を紹介する．

基礎実験

ユーザーからの HTTP リクエスト数の増加による応答時間の遅延を確認するために基礎実験を行った．応答時間の遅延はユーザーの直帰率が 50% まで上昇する 3 秒以上とする[*3]．

実装

本稿ではユースケースに合わせた実験を行うために，Docker Hub のイメージにある WordPress，MySQL を使用してチケットサイトを作成した．使用したイメージは以下の通りである．

- wordpress:5.6-apache
- mysql:5.7

また，チケットサイト作成には以下の WordPress プラグインを使用した．

- WooCommerce v6.5.1
- WooCommerce Stripe ゲートウェイ v6.4.1
- MTS Simple Booking-C v1.4.1

WooCommerce は WordPress 上にショッピングサイトを構築できるプラグインである．WooCommerce Stripe ゲートウェイは WooCommerce でクレジットカード決済を可能にするプラグインである．本稿の実験ではテストモードであるテストカードを使用して疑似決済を行う．テストカードではテスト用の API キーを呼び出して決済を行う[*4]．MTS Simple Booking-C はチケットサイトの日付予約を行うプラグインである．

基礎実験の環境

基礎実験の実験環境には研究室にある ESXi サーバー上に仮想マシンを作成する．作成した仮想マシンを使った基礎実験の概要を図 2 に示す．

Kubernetes クラスタはアプリケーションである WordPress Pod とデータベースである MySQL Pod をデプロイしたワーカーノードとマスターノードで構成されている．負荷試験ツールは Locust を使用している．Locust を仮想マシンである VM1 にデプロイし，WordPress Pod に対し

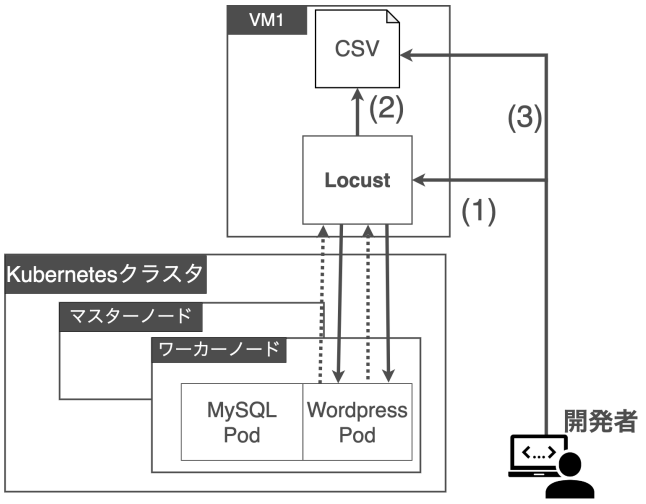

図 2　基礎実験の概要

てリクエストを送信していることを表す．基礎実験の手順は以下の通りである．

(1) 開発者が VM1 で構築した Locust を起動し，負荷試験を開始する．
(2) Locust による負荷試験の結果が，CSV ファイルに出力される．
(3) 開発者は CSV ファイルに記述されている平均応答時間の結果を確認する．

アクセスシナリオ

作成したチケットサイトのアクセスシナリオを設計した．図 3 が想定するアクセスシナリオである．

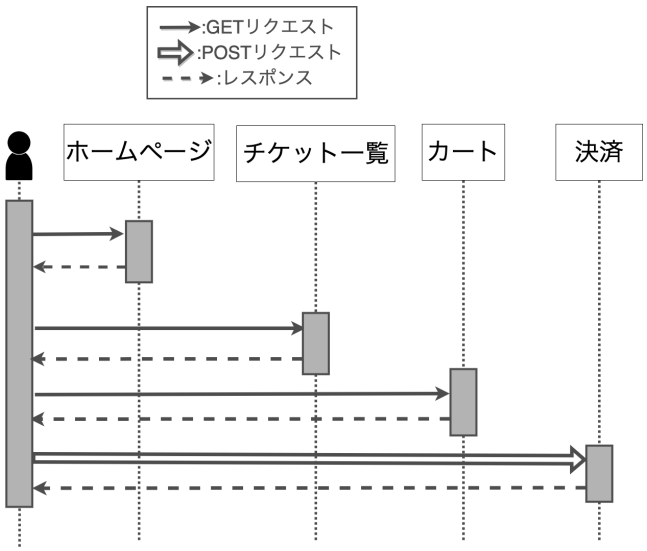

図 3　アクセスのシナリオ図

ユーザーがチケットサイトのホームページ GET リクエストでアクセスしていることを表す．そこからチケット一覧のページ，カートにチケットを入れる動作を GET リクエストでアクセスしていることを表す．決済ページでは

[*2] https://webtan.impress.co.jp/e/2014/07/08/17757
[*3] https://webtan.impress.co.jp/e/2014/07/08/17757
[*4] https://stripe.com/docs/testing

POST リクエストを行い，チケットの決済を行うことを表す．しかし，実際のユーザーはこの手順を 100%リクエストするわけではない．例として，ユーザーがチケットをカートに入れたまま決済しないという場合が考えられる．また応答時間が 3 秒以上でユーザーが離脱をする値である直帰率は 50%を超える．応答時間の遅延によって ユーザーがチケットを購入する前に離脱をしてしまうため，チケットサイトの売り上げの減少につながる．

基礎実験の結果

応答時間の遅延を確認するために Locust を使用して負荷試験を行う．ここでは実験を簡単にするために，図 3 のホームページから決済までの間でのユーザーの離脱は考慮しない．今回使用するトラフィックとして研究室の Web サイト[*5]のアクセスログからユーザー数，アクセス数を取得したデータを使用する．アクセスログは 2021 年 8 月 1 日から 2022 年 7 月 1 日に集計した期間を使用する．アクセスログの総数は 256701 件である．ユースケースに近いリクエストを実現するために，集計したアクセスログを曜日ごとに分け，それぞれ合計したものを使用する．チケットサイトのトラフィックは，ユーザーからのリクエスト数の増加，減少を繰り返す特徴がある [7]．そのため，今回の負荷試験ではリクエスト数の増加と減少が激しい，水曜日のトラフィックを使用する．負荷試験は約 48 分行った．図 4 は基礎実験の結果を表す．

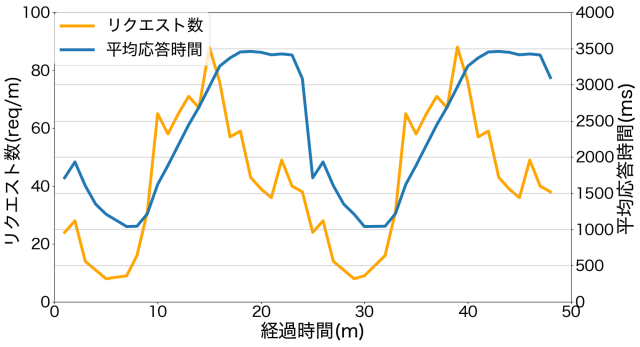

図 4　基礎実験の結果

X 軸は負荷試験の時間を分単位で表し，左の Y 軸は折れ線グラフのリクエスト数 (req/m) を表し，右の Y 軸は折れ線グラフの平均応答時間 (ms) を表す．負荷試験開始から 20 分付近で，平均応答時間が約 3500(ms) を超えている．課題でも述べたように，ユーザーの直帰率が 50%超えるのは約 3000(ms) からである．そのため，ユーザーが離脱する応答時間を超えているため，遅延が発生したといえる．

各章の概要

第 2 章以下の各章の概要について説明する．2 章では本稿に関連する既存研究を紹介する．3 章では本稿の提案手法を説明する．4 章では実装及び実験方法を述べる．5 章では提案手法の評価と分析を説明する．6 章では提案手法の議論をする．7 章では本稿全体をまとめ，貢献を述べる．

2. 関連研究

David Balla らは，負荷に合わせた Pod の CPU 制限のしきい値を自動的に検出しすることで，Pod を自動スケールさせるアルゴリズムを提案している [8]．この研究は 30 秒間の CPU 使用率のメトリクスを収集し，Pod の CPU 制限を計算する適応型オートスケーラ Libra を用いて Pod をスケールさせる．これにより CPU 制限を 15%に設定することで最大 8 つの Pod をスケールさせ 250 リクエスト/s 処理できるようになった．しかし既存手法である HPA の CPU 制限である 90%と比較して，単一の Pod が処理できるリクエスト処理が考慮されていない．Pod の数を増大させることで最大リクエスト処理は増えるが，Pod 数の制限によるノードのリソース改善という観点から改善の余地がある．

Qiang Wu らは，使用していない Kubernetes クラスタのリソースを削減し，ノードのスケールを動的に調整するアルゴリズムを提案している [9]．この研究ではユーザーに対して安定したサービスを提供するために Quality of Service(QoS) を保証しつつノードのスケールを行う．QoS の保証を行うためにノードの CPU 使用率を取得し，CPU 使用率の上限しきい値を決定することで QoS 保証を行っている．スケールさせる理想のノード数の決定は，ノードの CPU 使用率の上限しきい値，ノード CPU 使用率を用いて出力される．実験は Kubernetes ドキュメントの HPA 設定値を用いた Kubernetes クラスタの平均 CPU 使用率との比較を行う. 実験の結果，HPA を用いた Kubernetes クラスタの平均 CPU 使用率は 45.78%に対して，提案システムの Kubernetes クラスタの平均 CPU 使用率は 75.22%であった．CPU 使用率は 28.44%増加しており，提案システムが Kubernetes クラスタの CPU 使用率リソース改善したことを示した．しかし単一のワーカーノードがどこまで CPU を使用するのかの検討がされていない．CPU をどこまで使用するかのしきい値の決定という観点で改善の余地がある．

Tian Ye らは，急激なアクセス増加に対応するためにコンテナの CPU 使用率を監視することでコンテナを自動スケールアウトアルゴリズムを提案している [10]．提案であるアルゴリズムはリソース需要予測モデルを作成し，将来のリソース使用量を予測する．実験の評価ではしきい値は 70%と設定した Kubernetes の HPA を使用し，提案手法と応答時間の比較を行った．また応答時間は 200(ms) でレスポンスすることを想定している．結果，HPA を使用した平均応答時間では 500(ms)，提案手法の平均応答時間は約

[*5] https://ja.tak-cslab.org/

200(ms) であった．このことから応答時間の遅延を解消したことがわかる．しかし，提案手法はコンテナの CPU 使用率しか考慮しておらず，コンテナの数を考慮していない．応答時間 200(ms) を想定した，コンテナの数の決定という観点から改善の余地がある．

3. 提案方式

本稿では応答時間の遅延の低減を行うために，アプリケーションから送信される応答時間に基づいた Pod を自動的にスケールするアルゴリズムを提案する．提案の概要を図 5 に示す．

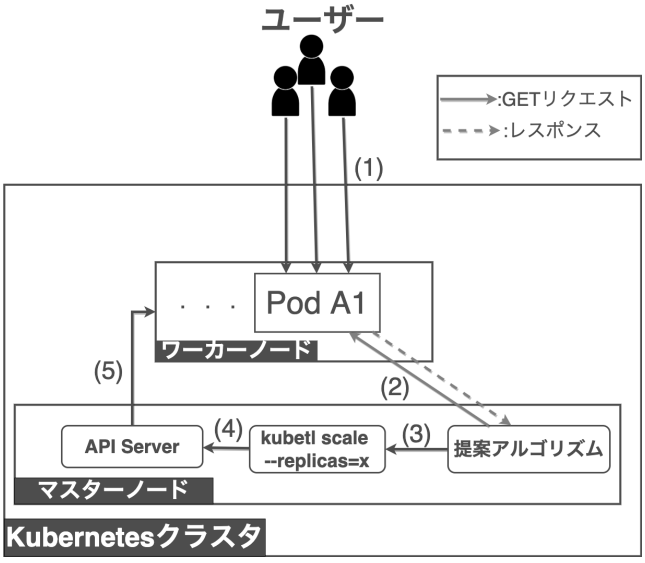

図 5 提案の概要図

ユーザーはアプリケーションにアクセスする利用者を表している．Pod A1 はユーザーが利用するアプリケーションを表している．図 5 の提案アルゴリズムの手順を下記に示す．

(1) ユーザーは，Pod A1 にアクセスするために HTTP リクエストを送信している．
(2) Pod A1 に対して GET リクエストを送信し，応答時間の値を取得する．
(3) Pod のスケール数 x が，提案アルゴリズムの計算結果として出力される．
(4) 出力された x は，Pod をスケールさせるために kubectl scale コマンドで入力する．
(5) kubectl scale コマンドで入力されたスケール数を API Server を経由してスケールされる．

次のセッションでは提案アルゴリズムについて説明をしていく．

3.1 提案アルゴリズム

図 5 の (2) で取得した応答時間の値は式 (1) で Pod をスケールアウトするかを決める．

Algorithm 1 提案アルゴリズム

Input: 現在の Pod の応答時間 now_pod_res
現在デプロイしている Pod 数 pod_x
1 秒あたりのリクエスト処理数 μ
1 秒あたりのリクエスト到着数 λ
Output: Pod のスケール数 $scale\_pod\_x$

```
function SCALE_POD_BINARY(now_pod_res, pod_x, μ, λ)
    left ← 1
    right ← maxpod_x
    while left < right do
        pod_mid = (left + right)//2
        scale_pod_x = pod_x + pod_mid
        S = 1/(scale_pod_x*μ)
        ρ = λ/μ
        W = 1/μ * ρ/(1−ρ)
        scale_pod_T = S + W
        if scale_pod_T < 3000 then return scale_pod_x
        end if
    end while
end function
```

$$3000 < now\_pod\_res \tag{1}$$

now_pod_res は図 5 の (2) で取得した応答時間 (ms) である．now_pod_res が 3000(ms) 以上だった場合に Pod をスケールアウトする数を決定する提案アルゴリズムを行う．3000(ms) はユーザが 50%離脱をする応答時間の値である．Pod のスケール数の探索は二分探索法を用いる [11]. 二分探索法を選んだ理由として，計算速度が速い事が挙げられる．線形探索法は $O(n)$ であるのに対し，二分探索法は，$O(\log n)$ である．このことから二分探索法のほうが高速に探索できる．二分探索法で探索した数で応答時間のシミュレーションを行う．応答時間のシミュレーションには待ち行列理論を用いる [12]．待ち行列理論は，応答時間をもとにした最小限のスケールアウト数の決定が有効であることを示している [13].

提案アルゴリズムの擬似コードを Algorithm1 に示す．入力として，ワーカーノードにデプロイしている Pod の応答時間の値 now_pod_res，ワーカーノードにデプロイしている Pod の数 pod_x，Pod が 1 秒間にリクエストを処理した数 μ，1 秒間にワーカーノードに到着したリクエスト数 λ が用意されている．出力は Pod のスケール数 $scale\_pod\_x$ とする．関数 SCALE_POD_BINARY の引数として now_pod_res， pod_x，μ，λ を用意する．二分探索法を行うために変数 left に 1，変数 right に $maxpod\_x$ を代入する．$maxpod\_x$ は Kubernetes クラスタにデプロイできる最大の Pod 数である．この値は Pod のレプリカ数をデプロイできる最大数まで増やす実験を行う後で設定する．その後，二分探索法を行い，スケールさせる Pod の応答時間を計算し，3000(ms) 未満の場合，$scale\_pod\_x$ を出力する．

図 6 が応答時間の概要図である．Pod はワーカーノードに配置されているアプリケーションを表し，ユーザーは

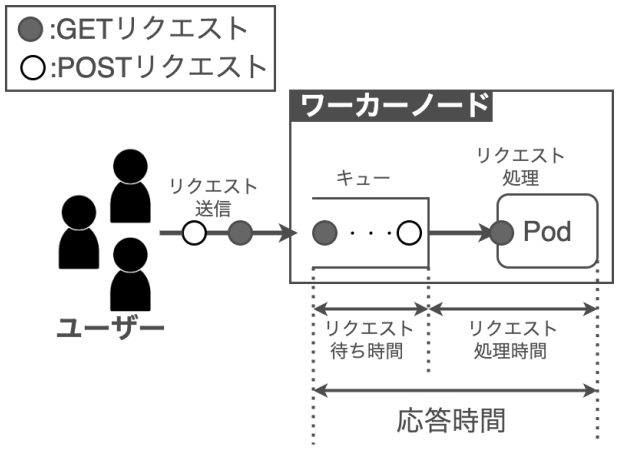

図 6 応答時間の概要図

Pod の利用者を表す．リクエストの処理は，ユーザーへリクエストの結果を返すための動作を表す．キューはユーザーから送信されたリクエストが Pod に到達するまでの列を表している．今回の提案では一つのキューでリクエストの待ち行列を行う．リクエスト処理時間 S は式 (2) で求められる．

$$S = \frac{1}{\mu} \tag{2}$$

μ は 1 秒あたりのリクエスト処理数を表す．図 6 では Pod がリクエストを 1 秒間に処理した数を表す．ρ は Pod の稼働率を表している．Pod の稼働率は，ユーザーからのリクエストをどれほど処理しているのかの指標を表す．ρ は 0 から 1 の値で表され，1 に近づくほど Pod が 1 秒間に多くのリクエストを処理していることがわかる．リクエスト処理の待ち時間 W は式 (3) で求められる．

$$W = \frac{1}{\mu} \cdot \frac{\rho}{1-\rho} \tag{3}$$

ρ は式 (4) で求められる．

$$\rho = \frac{\lambda}{\mu} \tag{4}$$

λ は 1 秒あたりのリクエストの到着数を表している．図 6 ではユーザーからのリクエストがワーカーノードに到達したことを表している．式 (2) と式 (3) で求めた値で応答時間 T を求めることができる．式 (5) で応答時間 T を求められる．

$$T = W + S \tag{5}$$

ユースケース・シナリオ

ユースケースはテーマパークと動物園の複合テーマパークであるチケットサイト[*6]を想定している．ユースケースシナリオを図 7 に示す．

テーマパークでは繁忙期になるとチケットサイトへのリクエストが増加する．通常 1 日 2 万リクエスト数が繁忙期になると 1 時間で 2 万以上のリクエストがチケットサイトに集中する．また，動物園でも限定公開する動物 (例: パンダ) があるとチケットサイトへのアクセスが増加こともを想定している．そのため，急激にリクエスト数が増加してもチケットサイトの応答時間を増加させないことが求められる．

[*6] https://www.cpi.ad.jp/use/ticket/

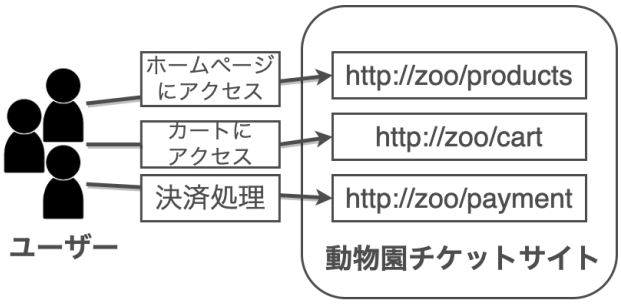

図 7 ユースケースの概要図

4. 実装と実験方法

実装

1 秒間のリクエスト処理時間を取得するために，NGINX Ingress を構築する．メトリクスの取得として Telegraf をマスターノードに構築する．取得したメトリクスを格納する InfluxDB をクラスタ外の VM に作成する．提案アルゴリズムは Python 3.10 で作成する．必要なパッケージは以下の通りである．

- Requests
- Simpy
- Kubernetes Python Client

応答時間の取得には Python の Requests パッケージを用いる．Simpy は応答時間のシミュレーションを行うために使用する．Kubernetes Python Client は，Kubernetes クラスタにデプロイしている Pod 数の取得に使用する．

実験環境

実験環境を図 8 に示す．

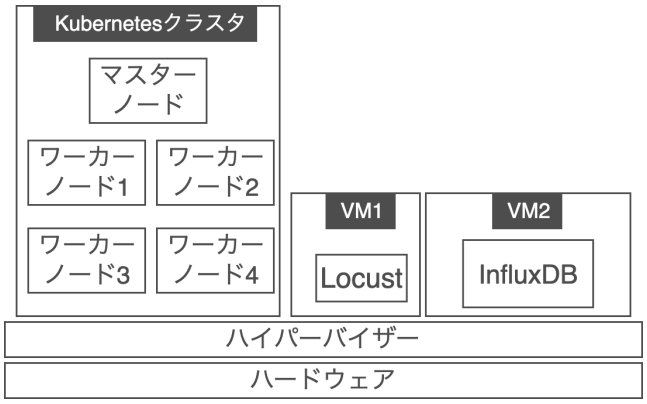

図 8 実験環境の概要図

ハードウェアに ESXi をインストールし，仮想マシンを作成する．仮想マシンにインストールする OS は Ubuntu22.04 である．作成する仮想マシンは，Kubernetes クラスタを構築する VM，Locust 専用の VM1, InfluxDB 専用の VM2 である．Kubernetes クラスタはマスターノード 1 台，ワーカーノード 4 台の計 5 台で構築する．Kubernetes のディストリビューションは Rancher 社が提供している RKE2 を使用する．VM のスペックは表 1 に記載する．

表 1 VM のスペック

VM	vCPU(コア)	RAM(GB)	HDD(GB)
マスターノード	4	6	40
ワーカーノード 1	4	6	40
ワーカーノード 2	4	6	40
ワーカーノード 3	4	6	40
ワーカーノード 4	4	6	40
VM1	2	4	30
VM2	1	2	30

5. 評価と分析

提案手法の比較として HPA を用いる．HPA の設定値である CPU 使用率 (%) は 40，50，60，70，80，90 のそれぞれの値を比較する．評価方法は，以下の項目で行う．

1 リクエスト数が増加した場合の HPA と提案手法でのユーザーからの応答時間の比較
2 リクエスト数が増加した場合の HPA と提案手法でのスケールアウト性能の比較

1 の評価方法である応答時間の計測方法を図 9 に示す．

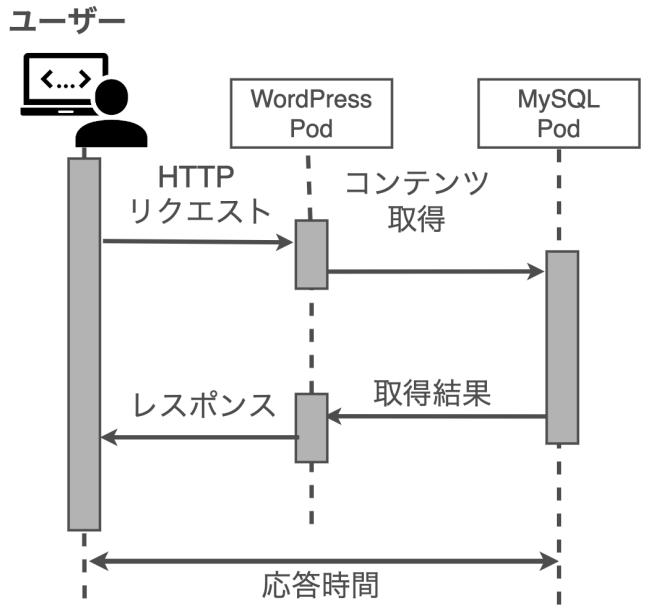

図 9 応答時間の比較

応答時間はユーザーが WordPress Pod に HTTP リクエストを送信し，ユーザーへレスポンス結果が得られるまでの時間を想定している．

2 の評価項目は HPA との Pod 数の比較を行う．Pod の数を最小にすることは，Kubernetes クラスタのメモリ削減につながる．そのため，提案アルゴリズムが最小の Pod で応答時間の遅延の低減できることを示す．

6. 議論

本稿の提案手法では現在の応答時間を基に Pod のスケールアウトの計算を行っている．しかしオリンピックのチケットサイトのような 1 秒間に約 800 万のリクエストに対応するには本稿の提案手法では不十分である [3]． リクエスト数に対して，応答時間の遅延の低減に必要な Pod 数のスケールアウトに時間がかかるためである．したがって，応答時間の遅延が発生する前に Pod をスケールさせることが必要である．これには過去のリクエスト数の変化傾向から応答時間の遅延が発生する前に必要な Pod 数の予測を行う．それにより，事前に必要な Pod のスケールアウト数の決定し，スケールアウト時間の短縮する．

本稿の提案手法は Pod のスケールアウトに焦点を当てているが，Pod のスケールインには焦点を当てていない．応答時間が 3 秒未満の場合のスケールインアルゴリズムを作成する必要がある．これには待ち行列理論から現在ある Pod のうち，応答時間が 3 秒未満の Pod 数を求める．

応答時間の計算で用いたキュー内のリクエストは GET と POST で区別をした．しかしリクエストの応答時間は，想定アクセスシナリオである図 3 のトランザクションごとに違う．そのため，1 つのキュー内で GET と POST で区別するのではなく，トランザクションごとにキューを作成してリクエストを区別する必要がある．これには Chui Ying Hui らが行ったように，ユーザーから要求されるリクエストごとにキューを作成して，リクエストを区別する [14].

7. おわりに

課題は，チケットサイトである Pod へのリクエスト数増加による，応答時間の遅延である．提案は，二分探索法を用いた Pod のスケール数の決定である．提案手法では，二分探索法を用いてスケールさせる Pod 数を探索し，Pod の応答時間をシミュレーションを行った．これにより，応答時間に基づいた Pod のスケールアウトをさせ，3 秒未満の応答時間の維持を実現した．評価では，HPA の CPU 使用率 (%) が 40，50，60，70，80，90 でスケールアウトさせる設定値で応答時間の比較，Pod 数の比較を行う．提案により，遅延した応答時間の低減が期待できる．

参考文献

[1] Li, F.: Cloud-Native Database Systems at Alibaba: Opportunities and Challenges, *Proc. VLDB En-*

dow., Vol. 12, No. 12, p. 2263–2272 (online), DOI: 10.14778/3352063.3352141 (2019).

[2] Zhang, Y., Zhao, J. and Xing, C.: An Extensible Framework for Internet Booking Application Based on Rule Engine, *2009 Sixth Web Information Systems and Applications Conference*, pp. 139–142 (online), DOI: 10.1109/WISA.2009.20 (2009).

[3] Davis, J. A.: *The Olympic Games effect: How sports marketing builds strong brands*, John Wiley & Sons (2012).

[4] Teo, Y. M.: Modelling flash crowd performance in peer-to-peer systems: Challenges and opportunities, *2014 5th International Conference on Intelligent Systems, Modelling and Simulation*, pp. 3–4 (online), DOI: 10.1109/ISMS.2014.165 (2014).

[5] El Haj Ahmed, G., Gil-Castiñeira, F. and Costa-Montenegro, E.: KubCG: A dynamic Kubernetes scheduler for heterogeneous clusters, *Software: Practice and Experience*, Vol. 51, No. 2, pp. 213–234 (2022).

[6] Poggi, N., Carrera, D., Gavaldà, R., Torres, J. and Ayguadé, E.: Characterization of workload and resource consumption for an online travel and booking site, *IEEE International Symposium on Workload Characterization (IISWC'10)*, pp. 1–10 (online), DOI: 10.1109/IISWC.2010.5649408 (2010).

[7] Zhang, Z., Ge, L., Wang, P. and Zhou, X.: Behavior reconstruction models for large-scale network service systems, *Peer-to-Peer Networking and Applications*, Vol. 12, No. 2, pp. 502–513 (2019).

[8] Balla, D., Simon, C. and Maliosz, M.: Adaptive scaling of Kubernetes pods, *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–5 (online), DOI: 10.1109/NOMS47738.2020.9110428 (2020).

[9] Wu, Q., Yu, J., Lu, L., Qian, S. and Xue, G.: Dynamically Adjusting Scale of a Kubernetes Cluster under QoS Guarantee, *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 193–200 (online), DOI: 10.1109/ICPADS47876.2019.00037 (2019).

[10] Ye, T., Guangtao, X., Shiyou, Q. and Minglu, L.: An Auto-Scaling Framework for Containerized Elastic Applications, *2017 3rd International Conference on Big Data Computing and Communications (BIGCOM)*, pp. 422–430 (online), DOI: 10.1109/BIGCOM.2017.40 (2017).

[11] Nowak, R.: Generalized binary search, *2008 46th Annual Allerton Conference on Communication, Control, and Computing*, pp. 568–574 (online), DOI: 10.1109/ALLERTON.2008.4797609 (2008).

[12] 大石進一：待ち行列理論，オーム社 (2003).

[13] Al Qayedi, F., Salah, K. and Zemerly, M. J.: Queuing theory algorithm to find the minimal number of VMs to satisfy SLO response time, *2015 International Conference on Information and Communication Technology Research (ICTRC)*, pp. 64–67 (online), DOI: 10.1109/ICTRC.2015.7156422 (2015).

[14] Hui, C. Y., Kee-Yin Ng, J. and Chung-Sing Lee, V.: On-demand broadcast algorithms with caching on improving response time for real time information dispatch systems, *11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'05)*, pp. 285–288 (online), DOI: 10.1109/RTCSA.2005.66 (2005).