メモリとCPU使用量と応答時間に基づくKubernetes上のPod数の決定

坂本 一俊[1]　伊藤 佳城[2]　串田 高幸[1]

概要：Kubernetes(K8s) は，ReplicaSet で Pod 数を設定することができる．WordPress Pod は，ユーザから送られてくるリクエストを処理する際に CPU とメモリを使用する．リクエスト数に対して Pod 数が少ないとリクエストからレスポンスまでの応答時間が増加する．リクエスト数に対して Pod 数が多いとメモリの使用量が増加し，ノードのメモリの使用可能の上限に近くなると，ノードのメモリ不足による Out of Memory Killer が発生する．本提案では，多目的最適化アルゴリズム NSGA-Ⅱを用いて Pod 数を求める．NSGA-Ⅱを使うため目的関数を作成した．基礎実験では，Pod 数とリクエスト数から CPU とメモリの使用量・応答時間を計測し，最小二乗法を用いて重回帰式を求め回帰分析を行った．回帰分析の結果，Pod 数とリクエスト数から CPU とメモリの使用量を求める重回帰式の決定係数が 0.975，0.997 となった．このことから CPU とメモリの使用量を求める重回帰式とデータの当てはまり度合いが高い．CPU の使用量を求める際に，t 検定より毎秒リクエスト数から CPU の使用量を求めることができる．評価実験では，負荷をかけて Pod 数別で CPU とメモリの使用量を計測し，提案で算出した Pod 数がほかの Pod 数の CPU とメモリの使用量を比較し評価する．本研究の提案により Pod 数を決定することで応答時間の増加を抑制する．

1. はじめに

背景

WEB サイトを運用する際に，WordPress を Kubernetes(K8s) クラスター上にデプロイする [1][2]．図 1 のようにユーザは，K8s の WEB サイトにアクセスする．K8s

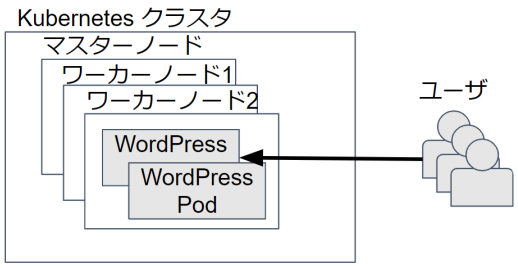

図 1　ユーザが WordPress にアクセスする様子

とは Google が開発した OSS のコンテナオーケストレーションシステムである [3]．

Pod は K8s がコンテナを実行管理するための最小単位で，1 つ以上のアプリケーションコンテナを内包しているものである．

K8s では，ワークロードリソースを設定することができる．その中に，ReplicaSet がある．同じ機能をもつ Pod が指定した値になるように自動的に起動する．ReplicaSet を指定する目的は，K8s がどのような時でも安定した Pod 数を維持するためである．例えばノードや Pod に障害が発生した場合に，指定された数を満たすように別のノードで Pod を起動する．そのため Pod のアプリケーションコンテナにアクセスできない障害が発生した際にアクセスできない時間を低減することができる．図 2 のように，開発者は Pod の設定ファイルの.spec.replicas フィールドに 2 と設定すること．K8s クラスタに Pod を 2 つ配置することができる．Pod はノードの CPU とメモリを使用する．Pod のコ

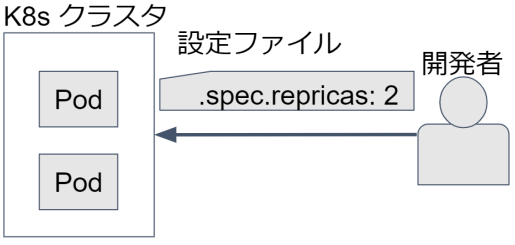

図 2　Pod の配置方法

[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

ンテナがリクエストを処理する際に，ノードの CPU とメモリを使用する．リクエスト数に対して Pod 数が少ないとき，リクエストが処理できないことでタイムアウトが発生する．タイムアウトが発生することでアプリケーションからレスポンスが来ない．リクエスト数に対して Pod 数が多いと Pod のメモリの使用量が増加する．Pod のメモリの使用量の増加によりノードのメモリがひっ迫することで K8s が自動的に Pod の削除や，メモリ不足でシステムが停止するのを避ける．そのためノードによる Out of Memory Killer が発生し，メモリ確保のためにプロセスを強制的に停止する．それらが発生するとノードが使用することができなくなる．使用できなくなったノードにある Pod は使用できるノードに移動する．そのノードにリクエストが集中する．そのノードでも同じことが発生することでクラスタ全体のノードが落ちることがある．アプリケーションがダウンすることでサービスの利用が不可能になる．

基礎実験

基礎実験 1 では，課題において Pod 数が変化するとメモリの使用量に変化があるのか調べた．基礎実験 2 では，提案方式に用いる Pod 数とリクエスト数から CPU とメモリの使用量と応答時間を求める重回帰式を算出する．またその重回帰式に有意性があるのか重回帰分析を行った．

基礎実験環境

基礎実験の実験環境について図 3 に示す．実験環境はハイパーバイザーの上に 5 つの仮想マシンを作成した．それらのうち 3 つの仮想マシンを K3s を用いて K8s クラスタを作成した．それらには，その Pod では WordPress が実装されている．この WordPress はクラウド・分散システム研究室の WEB サイト[*1]と同じコンテンツが複製されて実装されている．残り 2 つの仮想マシンでは，1 つが MariaDB を実装し，もう 1 つはコンテンツを保存するための NFS サーバを建てた．それぞれの仮想マシンの CPU コア数とメモリ，ディスクの容量について表 1 に示す．K8s クラスタは，2 コア 8GB である．負荷テストツールとして Locust を用いた．Locust は，別の仮想マシンに実装している．Locust を用いて WordPress Pod にリクエストを送信した．マスターノードには Telegraf を導入し，クラスタそれぞれのノードに配置されている Pod の CPU 使用量やメモリの使用量を取得する．取得したデータを NFS サーバに実装した InfluxDB に送信し，時系列データとして保存している．

基礎実験 1

K8s クラスタ上で WordPress Pod 数が 10，20，30 個の時，メモリの使用量がどのように変化するか調べた．調べた結果を表 2 に示す．Pod 数が 10 個から 30 個の増加によりメモリの使用量が 492MB 増加することが分かった．これにより Pod 数が増加すると，メモリの使用量が増加することが分かった．

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

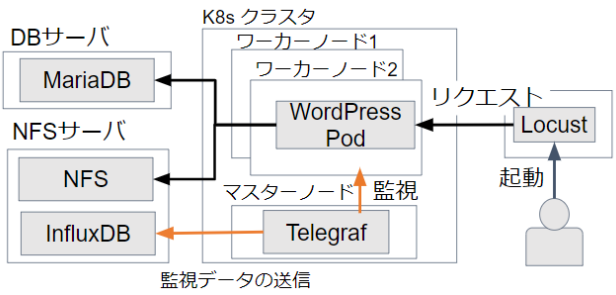

図 3 実装環境

表 1 ノードのリソース

	CPU 使用量 (vCPU)	メモリ (GB)	ディスク (GB)
マスターノード	2	8	40
ワーカーノード 1	2	8	40
ワーカーノード 2	2	8	40
DB サーバ	2	3	30
NFS サーバ	5	8	50

表 2 メモリの使用量の変化

Pod 数 (個)	WordPress Pod のメモリ使用量 (MB)
10	719
20	983
30	1201

課題

課題として，毎秒リクエスト数に対して WordPress Pod 数が少ないとリクエストからレスポンスまでの応答時間が増加する．図 4 のようにユーザかリクエストが来た際に，リクエストの処理が追い付かずレスポンスが返ってこないことを示している．なぜならリクエストを処理する Pod の数が少ないためである．毎秒リクエスト数に対して WordPress Pod 数が多いとメモリの使用量が増加し，ノードのメモリの使用可能の上限に近くなると，K8s が自動的に Pod の削除やメモリ不足による Out of Memory Killer が発生する．

各章の概要

このテクニカルレポートは，次のように構成される．1 章では本稿の背景と課題と基礎実験について述べる．第 2 章では関連研究について述べる．第 3 章では提案するシステムについて述べる．第 4 章では提案したシステムの実装と実験方法について述べる．第 5 章では実験の評価と分析の手法についてを述べる．第 6 章では提案したシステムの議論を述べる．第 7 章では本研究のまとめを行う．

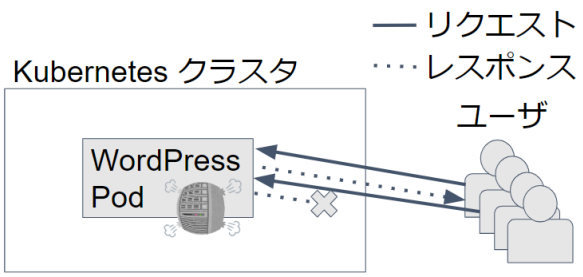

図 4　課題

2. 関連研究

Pod 数に関する関連研究として K8s のコンテナ自動スケーリング技術に基づくリソース予測モデルに関する研究がある [4]．これは，スケーリングする際に，応答時間の遅延の問題を解決できる自動スケーリング最適化を行っている．経験的モーダル分解と ARIMA モデルの組み合わせを使用して，Pod の負荷を予測し，予測結果に応じて Pod 数を事前に調整する．これにより Pod 数を負荷のピークに合わせて既存のスケーリング技術よりも早くスケーリングを行った．しかしこの研究では，メモリの使用量を考慮せずスケーリングを行っている．

3. 提案方式

基礎実験 2

基礎実験 2 では，提案方式において毎秒リクエスト数と Pod 数からメモリの使用量と CPU の使用量，応答時間を求める重回帰式を作成するため基礎実験を行う．基礎実験環境は，基礎実験と同様のものである．この実験では，ユーザ数は，100 人，150 人，200 人，250 人，300 人と 5 分毎に増やしていき，1 ユーザが 1 秒に 1 回リクエストを行う．ユーザ数の値は指数関数的にユーザ数 2 から送った際にユーザ数を 3000 人で送った際に実際に送られていた毎秒リクエスト数が 350 回で，Pod が配置されているすべてのノードの CPU 使用率が 100%に達していたためである．Pod 数は 1 個，2 個，3 個，4 個，5 個，6 個，7 個，8 個に変化させ，それぞれ 5 回ずつ実験を行った．ユーザがリクエストを送る場所は，実際の研究室の WEB サイトの 133 日分のログを用いた．1 ユーザごとにどこのページにリクエストしたか調べ，ユーザがリクエストを送る際に用いた．ユーザが送る順番のリストは 7582 種類ある．ただし別のユーザが同じリクエストの内容を行っていた場合は，違うものとしてリストに加えている．ユーザはリクエストを送る順番のリストの中からランダムで 1 つ選ぶ．

Pod 数が変化した時にそれぞれのユーザ数で毎秒リクエスト数の平均値の変化を図 5 に表す．この図から Pod 数によって毎秒リクエスト数の最大値が変化していることが分かる．Pod 数が 1 個，2 個の時リクエスト数が重なっているところがある．これは，Pod が配置されたノードが増加し CPU の使用量の上限が変化したため毎秒リクエストの上限が変化したからである．ユーザ数が 300 人で Pod 数が 4 個の時大きく減少している．その理由としてワーカノード 1 の CPU の使用量が 2 (vCPU) で他のノードが約 1 (vCPU) しか使っていなかった．なぜなら K8s の NodePort を使用して Pod にリクエストを均等に分散していたためワーカノード 1 にリクエストの 2 分の 1 のアクセスが集中しワーカノード 1 の CPU の使用量が 2 (vCPU) になったためだと考察した．Pod 数が変化した時にそれ

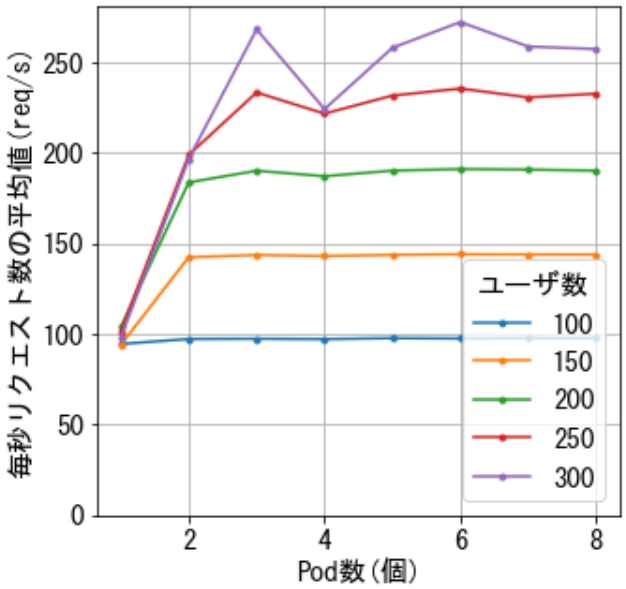

図 5　ユーザ数に対する Pod 数ごとの毎秒リクエスト数の変化

ぞれのユーザ数で，単位時間当たりの全ての WordPoress Pod の CPU 使用量の合計の平均値を図 6 に表す．リクエスト数によって CPU の使用量が変化していることを示している．Pod 数が 1，2 の時にユーザ数が異なるにもかかわらず同じ CPU 使用量がある．その理由は，使用可能な CPU のコア数の上限に達したからである．よってこのことから CPU 使用量の上限によってリクエスト数の上限が変化する．CPU コアが 2 コアの時 100req/s であり，4 個の時 200req/s であることが分かる．ユーザ数が 100 の時，Pod 数が増えても CPU の使用量が変化していない．このことから CPU の使用量は，ユーザのリクエスト数に依存することが分かる．

Pod 数が変化した時にそれぞれのユーザ数で，単位時間当たりの全ての WordPoress Pod のメモリの使用量を図 7 に表しており単位は GB である．リクエスト数が増えるとメモリの使用量が増加している．また Pod 数が増加するとメモリの使用量が増加していることを示している．次に Pod 数を変更した際のそれぞれのリクエストを行った

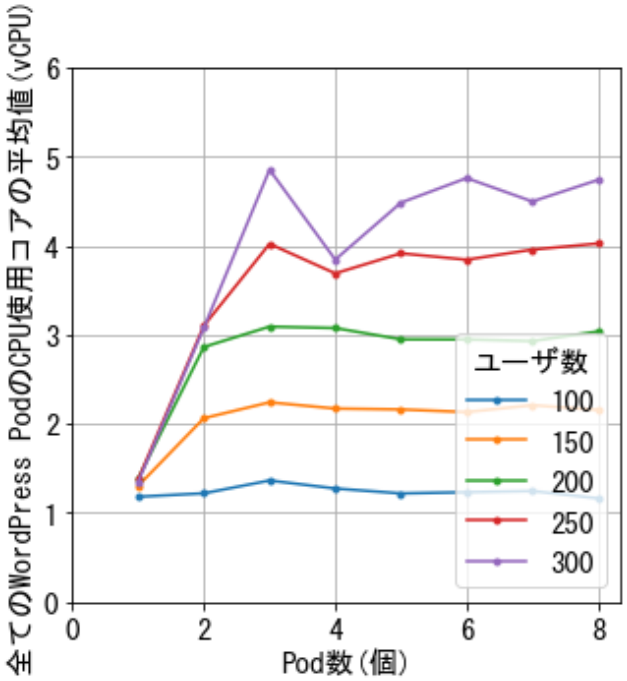

図 6　リクエスト数に対する Pod 数ごとの CPU 使用量の変化

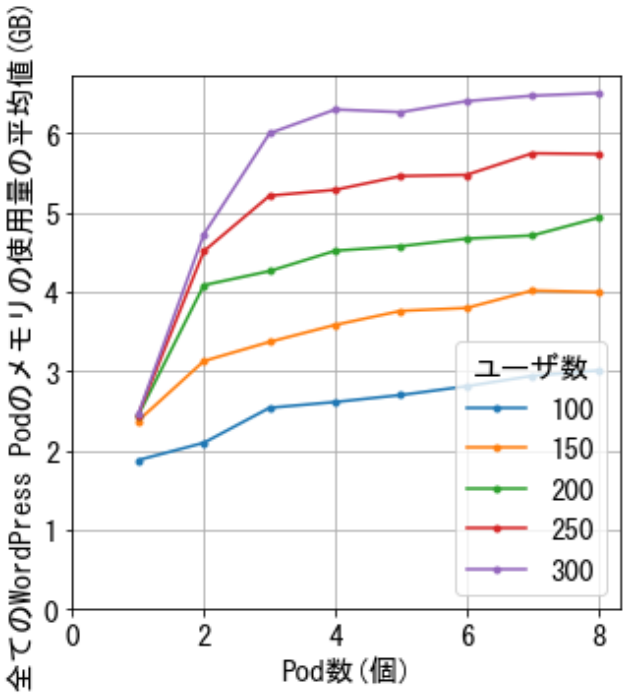

図 7　毎秒リクエスト数に対する Pod 数のメモリ使用量の変化

際にレスポンスが返ってくるまでの応答時間の平均値を図 8 に示す．Pod 数が増加すると応答時間の減少がみられる．Pod 数が 3，4，5 個の時に上下している．これはユーザが送信するリクエストが，ランダムに決まるためリクエスト数が同じでもリクエストの種類や宛先が異なるためである．

これらの結果から最小二乗法を用いて重回帰式を求め，

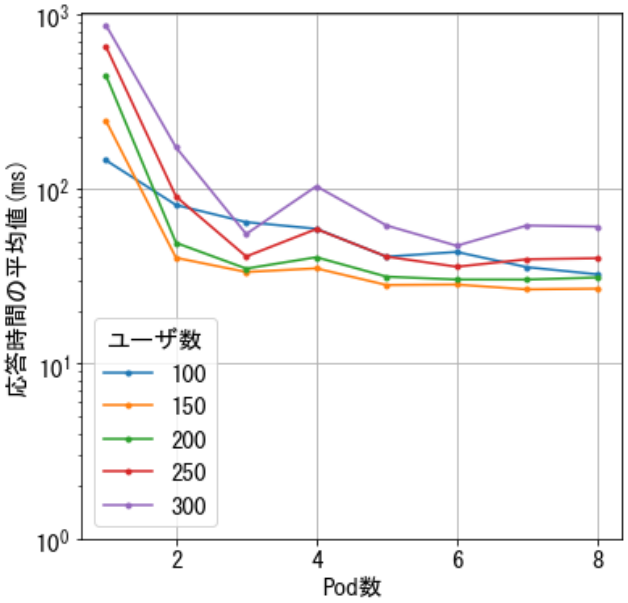

図 8　毎秒リクエスト数に対する Pod 数ごとの応答時間の変化

求められた重回帰式の重回帰分析を行う．説明変数は毎秒リクエスト数と Pod 数として目的変数は，CPU とメモリの使用量の平均値，応答時間の平均値とする．ただし Pod 数が 1 個，2 個の時は毎秒リクエスト数の平均のそれぞれが最大 100，200 回だったため除外している．よって重回帰式を求める際に，使用する説明変数は，Pod 数が 3 から 8 個の時で，毎秒リクエスト数が 100 から 300 までの値である．求められた重回帰式を重回帰分析を行った．目的変数に対する説明変数の影響を比較するために各説明変数のデータの平均をゼロ，分散が 1 になるように標準化を行う．重回帰分析は 3 種類行う．1 つ目に重回帰式とデータとのあてはまり度合を確認した．確認する方法として決定係数を用いる．2 つ目に得られた重回帰式が統計的に意味があるのか確認するため F 検定の p 値を求めた．3 つ目に推定された標準偏回帰係数が統計的に意味がある値か偏回帰係数の有意性の t 検定を行った．1 つ目と 2 つ目の分析の結果を表 3 に示す．決定係数は 0 から 1 の値をとり，1 に近いほど当てはまり度合が非常に高いといえる．結果から，決定係数は，CPU とメモリの使用量データのあてはまり度合が 0.975，0.997 と非常に高いことが分かる．しかし応答時間は 0.296 とあてはまり度合いが高くない．この理由として Pod 数が 3 から 8 にかけて応答時間が上下しており，求められた重回帰式と応答時間のデータで大きな誤差が生まれていると考えられる．F 検定の p 値から全ての説明変数において p 値が 0.05 以下と得られているため重回帰式に有意性があるといえる．3 つ目に推定された標準偏回帰係数が統計的に意味がある値か t 検定を行った．t 検定の結果を表 4 に示す．CPU とメモリの使用量を求める際の Pod 数と毎秒リクエスト数それぞれの偏回帰係数ごとの t 検定の p 値を表している．帰無仮説は，「偏回帰係数

表 3　重回帰分析の結果

	決定係数	F 検定の p 値
CPU	0.975	8.6910^{-23}
メモリ	0.997	4.3810^{-35}
応答時間	0.296	0.00878

が 0 である」とし，対立仮説は，「偏回帰係数が 0 ではない」とする．p 値が有意水準 0.05 以下となれば帰無仮説が棄却され．対立仮説が採択される．結果から，メモリの使用量は，Pod 数と毎秒リクエスト数の偏回帰係数の値は，統計的に意味があるといえる．CPU の使用量を求める際に，Pod 数の偏回帰係数は t 検定の p 値が 0.938 であったため，CPU と Pod 数には，影響されないといえる．

表 4　標準偏回帰係数と有意性

	標準偏回帰係数		t 検定の p 値	
偏回帰係数	CPU	メモリ	CPU	メモリ
Pod 数	0.0023	0.1476	0.938	0.000
毎秒リクエスト数	0.9883	0.9876	0.000	0.000

提案方式

Pod 数と CPU とメモリの使用量・応答時間に相関関係があると考えた．このことから，本提案手法では，Pod 数を算出する手法として多目的最適化アルゴリズム NSGA-Ⅱを用いる．多目的最適化とは，目的関数の引数を変化させ，それらの関数の値を最小 (最大) 化する引数を求めることを指す．CPU とメモリの使用量・応答時間を最小化する Pod 数を算出する．NSGA-Ⅱは，NSGA にエリート主義を導入したアルゴリズムであり，Dep らによって開発されたものである [5]．目的関数として，毎秒リクエスト数と Pod 数の 2 変数を用いて CPU の使用量とメモリの使用量・応答時間を求める関係式を使用する．関係式を作成するために，基礎実験 2 で求める．基礎実験 2 では，Pod 数ごとにリクエスト数を変化させ，CPU とメモリの使用量・応答時間のデータを取得する．それらのデータから最小二乗法を用いて重回帰式を作成する．求められた Pod 数は，最初に Pod をデプロイする際に，使用される．

基礎実験で求められた重回帰式について以下の式を表す．x が Pod 数，n が毎秒リクエスト数で，CPU の使用量 (vCPU) を式 1 とメモリの使用量 (GB) を式 2，応答時間 (ms) を式 3 に表す．基礎実験で得られた Pod 数とリクエスト数の時のそれぞれの CPU とメモリの使用量，応答時間の値をもとに最小二乗法を用いて重回帰式を求めたものである．

$$f_1(x) = 0.002x + 0.017n - 0.350 \quad (1)$$

$$f_2(x) = 0.109x + 0.177n + 0.452 \quad (2)$$

$$f_3(x) = -2.50x + 0.15n + 24.08 \quad (3)$$

次に上記の式の制約について以下に表す．この制約は，実装する際に，CPU とメモリには上限があるためである．C は使用可能な CPU のコア数の上限である．M は使用可能なメモリの使用量の上限である．式 4 では Pod 数であるため x は自然数をとる．式 5 は CPU 使用量の上限を表している．式 6 はメモリの使用量には上限を示している．式 7 は応答時間は 0 未満にはならない．基礎実験で Pod 数とリクエスト数が変化しても最小の応答時間が 0.9 (ms) であった．応答時間が最低で 0.9 (ms) かかると考えられるのでこれよりも小さい値はないと考え最小応答時間は 0.9ms とした．

$$x \in \mathbb{N} \quad (4)$$

$$f_1(x) < C \quad (5)$$

$$f_2(x) < M \quad (6)$$

$$f_3(x) = \{0.9 \mid f_3(x) < 0.9\} \quad (7)$$

求められたこれらの回帰式を 3 つの目的関数として NSGA-Ⅱを用いて Pod 数を算出する．

ユースケース・シナリオ

本研究のユースケースは，図 9 のように開発者が K8s 上で WordPress Pod があり，WEB サイトを運用している．ユーザが WEB サイトにアクセスする．WordPress が使用できる CPU とメモリの使用量に上限がある．制限がある中で CPU とメモリの使用量を抑え，応答時間を小さくする必要がある．

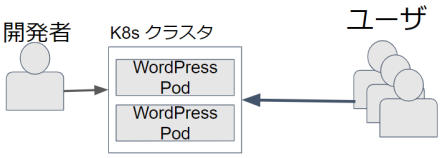

図 9　ユースケース・シナリオの概要図

4. 実装と実験方法

実装

作成したソフトウェアは，Python を用いて作成した．実装の概要を図 10 に示す．開発者が (1) で毎秒リクエスト数の n と CPU の上限の C，メモリの上限 M を入力する．(2) でソフトウェアが Pod 数を算出して開発者に出力する．(3) で開発者が Pod 数を設定ファイル Deployment.yaml で Pod 数を設定する．

次に作成するソフトウェアについて図 11 に示す．毎秒リクエスト数 n 回と CPU 使用コア数の上限 C (vCPU)，メモリの上限 M (GB) が入力されると提案方式を用いて Pod 数の算出を行う．Pod 数の算出を終えると，Pod 数の出力を行う．

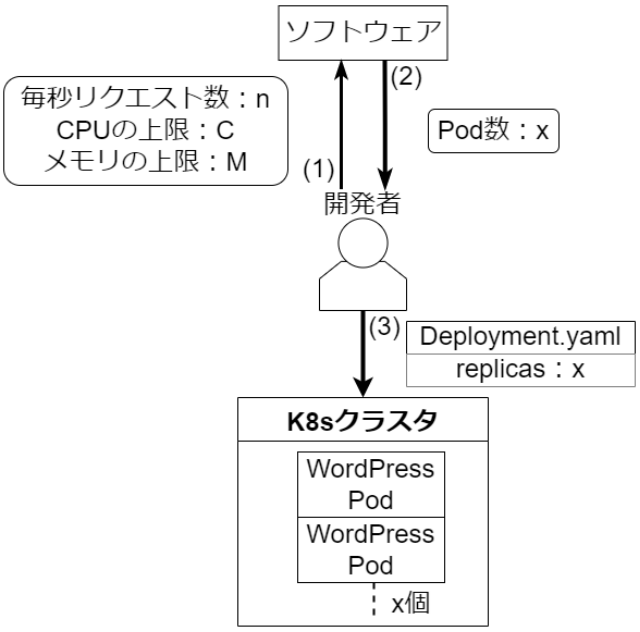

図 10 実装の概要図

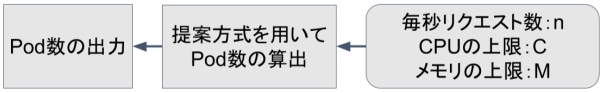

図 11 ソフトウェアの概要図

実験環境

実験環境は Ubuntu20.04 の仮想マシン 3 台で構築した K8s クラスタ 1 と仮想マシン 3 台で K8s クラスタ 2 を作成する．クラスタ 2 では負荷テストツールの Locust の Pod をデプロイしクラスタ 1 に負荷を行う．

5. 評価手法と分析手法

評価手法として，様々なリクエスト数の時，Pod 数を 1 個から Out of Memory Killer が発生するまでの Pod 数を計測し，CPU とメモリの使用量・応答時間を計測する．本提案で算出された Pod 数の時の CPU とメモリの使用量・応答時間の値と，他の Pod 数の時のリソースを比較し評価する．

6. 議論

今回基礎実験 2 で回帰分析を行った結果，応答時間の重回帰式とデータの決定係数が 0.296 と低くなってしまった．基礎実験環境においてリクエストを送った際に，最小二乗法を用いて重回帰式を作成することができないことが分かった．最小二乗法は，外れ値の影響を受けやすい．そのためロバスト推定などを用いて重回帰式への影響を小さくする．基礎実験環境において問題がなかったも検証する．基礎実験においてリクエストを均等に分散する NodePort を使用したため，Pod 数が 4 個の時に，ノードでリクエストに偏りが出てしまい，重回帰式の決定係数が低くなってしまったと考えたため．Pod 数が 4 個の時のデータを除いて最小二乗法を用いて重回帰式を求めた場合の決定係数を求めることで最小二乗法が使えるか判断をする．基礎実験のリクエストを送る順番のリストを作成した．そのリストについて研究室のログから異なるユーザごとにアクセス先について調べて作成した．そのためリストには同じリクエスト先が存在する．そこで同じシナリオの削除し，実際のログからシナリオごとに重み付けを行い，ランダムに選択する．

7. おわりに

課題として，リクエスト数に対して Pod 数が少ないと応答時間が増加する．Pod 数と CPU とメモリの使用量・応答時間に相関関係があると考えた．提案として Pod 数を算出するため，多目的最適化アルゴリズム NSGA-Ⅱを用いる．NSGA-Ⅱを用いるため，基礎実験にて Pod 数とリクエスト数から CPU とメモリの使用量・応答時間を求める重回帰式を作成した．重回帰式は最小二乗法を用いて求めた．求めた結果から Pod 数とリクエスト数から応答時間の重回帰式の決定係数が 0.296 となり，最小二乗法を用いて重回帰式を求めることができないことが分かった．本提案により，WordPress Pod の WEB サイトの応答時間の増加を抑制する．

参考文献

[1] Sabuhi, M., Musilek, P. and Bezemer, C.-P.: Studying the Performance Risks of Upgrading Docker Hub Images: A Case Study of WordPress, *Proceedings of the 2022 ACM/SPEC on International Conference on Performance Engineering*, ICPE '22, New York, NY, USA, Association for Computing Machinery, p. 97–104 (online), DOI: 10.1145/3489525.3511683 (2022).

[2] Zhu, H. and Gehrmann, C.: Kub-Sec, an automatic Kubernetes cluster AppArmor profile generation engine, pp. 129–137 (2022).

[3] Zhong, Z. and Buyya, R.: A Cost-Efficient Container Orchestration Strategy in Kubernetes-Based Cloud Computing Infrastructures with Heterogeneous Resources, *ACM Trans. Internet Technol.*, Vol. 20, No. 2 (online), DOI: 10.1145/3378447 (2020).

[4] Zhao, A., Huang, Q., Huang, Y., Zou, L., Chen, Z. and Song, J.: Research on Resource Prediction Model Based on Kubernetes Container Auto-scaling Technology, *IOP Conference Series: Materials Science and Engineering*, Vol. 569, No. 5, p. 052092 (online), DOI: 10.1088/1757-899x/569/5/052092 (2019).

[5] Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II, *IEEE Transactions on Evolutionary Computation*, Vol. 6, No. 2, pp. 182–197 (online), DOI: 10.1109/4235.996017 (2002).