Kubernetes PodのCPU使用率とアクセス数の変化量に基づく再配置

伊藤 佳城[1,a)]　串田 高幸[1,b)]

概要：K8s(Kubernetes) Pod は新規に作成された時，ノードに配置される．Pod 上で動作するアプリケーションは，ユーザから送られてくる HTTP リクエストを処理する際にノードの CPU を使用する．同一ノード上に複数の Pod がある場合，それぞれの Pod はリクエストを処理するために，ノードの CPU を使用する．その結果，応答時間が遅くなり，SLA(Service Level Agreement) に違反する可能性がある．本稿の目的は Pod における応答時間を短縮する．また，Pod のアクセスログから取得したアクセス数を各曜日で 1 時間ごとに分け，それをアクセス傾向とする．提案では，Pod の CPU 使用率とアクセス傾向の変化量を用いて順位付けを行う．評価実験では，研究室の Web サイトのアクセスログからアクセス傾向を算出し，WordPress で構築した 7 つの Web サイトに対して，各曜日のアクセス傾向を用いた負荷テストを行う．その結果，再配置後で 7 つ中 3 つの WordPress で応答時間が伸び，残り 4 つの WordPress の応答時間が削減できた．

1. はじめに

背景

Kubernetes(K8s) Pod は新規に作成された時，ノードに割り当てられる．これをスケジューリングと呼ぶ．その際に kube-scheduler と呼ばれる K8s のコンポーネントの 1 つであるスケジューラが使用される [1]．K8s の kube-scheduler はフィルタリングとスコアリングを用いてノードのスコアを算出することで Pod のスケジューリングを行う [2]．また，ノードは Pod のデプロイ数，Pod に対するアクセスによりノードの CPU 使用率は変化する．なお，本稿における CPU 使用率とは Pod が使用するノードの CPU 使用率のことを指す．これは，Pod で動作するアプリケーションのコンテンツによって，アクセスするユーザ数やリクエスト数が異なるためである．だが，一度ノードに割り当てられた Pod は割り当て後の移動が発生しない．

kube-scheduler のデフォルト設定では, ノード全体に Pod を均等に分散することで，管理するクラスタ内のすべてのノードの負荷分散が行われる．クラスタ内のノードは, 物理サーバか VM であるため CPU, メモリ，ディスクの上限がある．そのため kube-scheduler で負荷分散を行うことで Pod がノード内のリソースを使用する際に制約される可能性を減らすことが可能となる [3]．しかし，kube-scheduler はデプロイ時のノードの CPU 使用率や Pod 数を見て配置を行う．そのため，配置後の Pod の CPU 使用率におけるノードの影響は考慮されていない．

デプロイされる Pod は利用者ごとに動かすアプリケーション, 中身のコンテンツが異なる．例として本稿におけるアプリケーションで使用している WordPress を挙げる．WordPress は，ブログやホームページを構築する際に使用されるオンラインコンテンツ管理プラットフォームである．世界には 7300 万を超える WordPress のサイトが存在する [4]．また，WordPress は全ての Web サイトの 29%で機能している [5]．

同じ WordPress の Pod でも個人のブログや EC サイト，企業の紹介ページがあり，内部のコンテンツは異なっている．また，WordPress にアクセスするユーザ数は異なる．したがって，WordPress にアクセスされる数も異なる．

この時，アクセスにより発生したトラフィックを処理するために WordPress の Pod の CPU 使用率は変化する．その図を図 1 に示す．Pod はデプロイされたノードの CPU を使用するため，同一ノードの Pod の処理により常に変化する．ここで，同一ノードで使用できる CPU には上限があるため，十分な CPU 使用率を使用できない状態が発生する．

[1] 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
[a)] g21210091d@edu.teu.ac.jp
[b)] kushida@acm.org

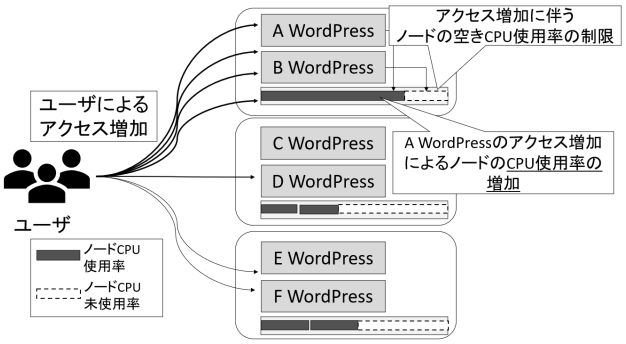

図 1　アクセス数増加によるノードにおける CPU 使用率の偏り

課題

Pod のアプリケーションがリクエストを処理する際にノード上の CPU を使用する．しかし，ノード上の別 Pod もノードの CPU を使用する．そのためお互いにノードの CPU を使用しあう状況になり，リクエストを処理するために必要な CPU を Pod で取り合ってしまう．そうするとノード上の Pod 同士でパフォーマンスを低下させ，将来スケジュールされる Pod も同様にパフォーマンスを低下させることになる [6]．基礎実験では，Pod が必要とするノードの CPU 使用率が使うことができず，応答時間が遅くなることがわかる．

検証実験として K8s クラスタ上に WordPress と MySQL の Pod を構築した．アクセス傾向を分析するため，WordPress で構築されている研究室サイト[*1]のアクセスログから曜日ごとに 1 時間ずつに分けてアクセス数を取得した．下記の図 2 にそのグラフを示す．

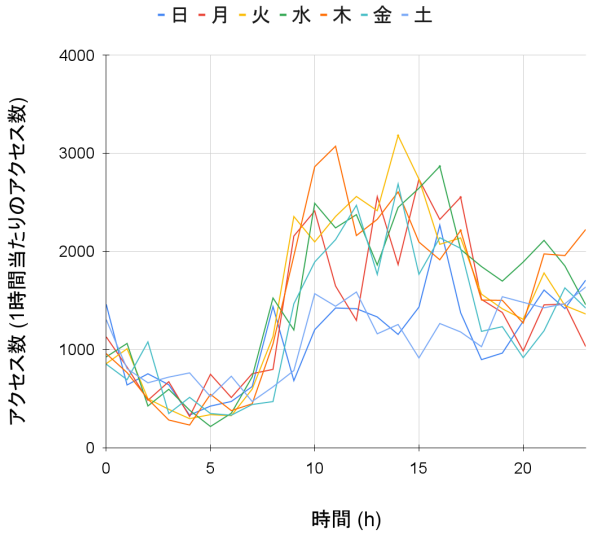

図 2　研究室サイトの WordPress のアクセスログ数

図 2 を見るとアクセス数は異なる．しかし，限時刻と限時刻から 1 時間前のアクセス数との差分を取るとアクセス数が 8 時の時点で全ての曜日が増加していることがわかる．そのことからアクセス数としては異なるがアクセス傾向としては，同じである．十分な負荷を WordPress の Pod に与えるためにアクセスユーザ数を 1000 人としてスケールする．また 1 ユーザ 1 リクエストとしてアクセスすることを前提としてアクセス傾向を適用させる．例として火曜日の 0 時では 2.36%であるため，1000 ユーザの 2.36%に適用させて秒間 24 req/s を送信する．基礎実験の環境を図 3 に示す．

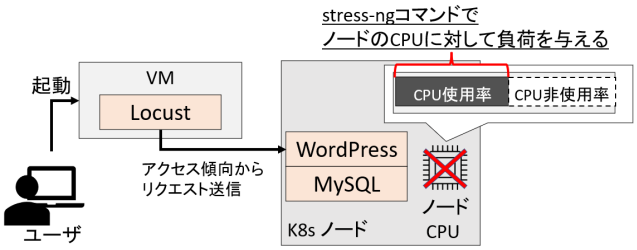

図 3　基礎実験環境

Pod が起動しているノードの CPU 使用率が別プロセスにてノード上の CPU 使用率が使われている状況を再現する．

Pod がデプロイされているノードの CPU に対して stress-ng コマンド[*2]を用いてノードの CPU 使用率を 50%，100%使用する負荷を与える．これによって Pod が使用できる CPU を制限して CPU 使用率とリクエストを処理する上での応答時間の変化を求める．基礎実験の結果を表 1 と図 4,5 に示す．

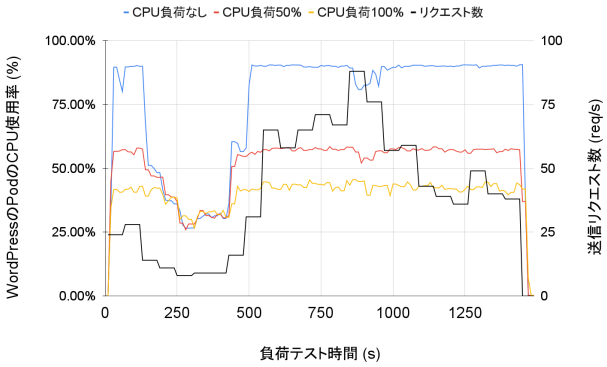

図 4　WordPress の Pod の CPU 使用率と送信リクエスト数

図 4，5 では，WordPress の Pod が使用するノードの CPU 使用率, 応答時間を stress-ng コマンドを用いてノードの CPU に対する負荷別にまとめたグラフとなる．それとは別に負荷テストで用いた送信リクエスト数も含んでいる．この送信リクエストはどちらも同じ req/s で送信している．図 4 では，すべての状況で送信リクエスト数と比例

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

[*2] https://github.com/ColinIanKing/stress-ng

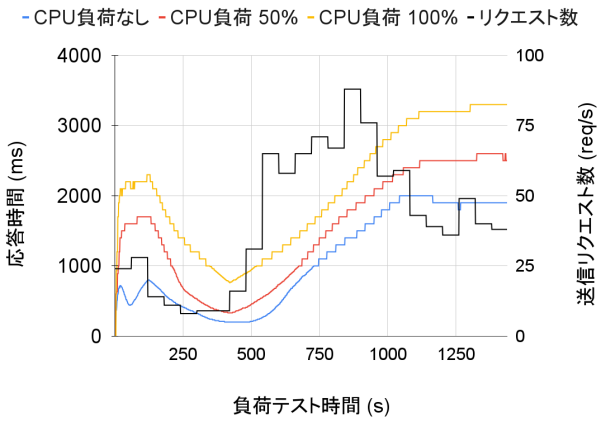

図 5 WordPress の Pod の応答時間と送信リクエスト数

するように WordPress の Pod の CPU 使用率が増減した．しかし，ノードの CPU が stress-ng によって CPU を使用している場合では WordPress の Pod が使用する CPU 使用率は，約 30%以上減少した．図 5 では，応答時間も CPU 使用率と同じくすべての状況で送信リクエスト数と比例して応答時間は増加した．

表 1 WordPress の Pod の応答時間と処理リクエスト数

	応答時間 中央値 (ms)	処理 リクエスト数
ノードの CPU 負荷 0%	1900	27662
ノードの CPU 負荷 50%	2500	19540
ノードの CPU 負荷 100%	3300	15564

表 1 の最終的な応答時間の中央値と処理リクエスト数をみる．ノードが別プロセスで CPU を使用している場合では，応答時間は 500 (ms) 以上の増加が示された．以上のことから Pod がリクエストを処理する際に使用できるノードの CPU 使用率によって応答時間が変化することがわかる．応答時間の増加が発生するとクラウドベンダーとクライアント間の SLA(Service Level Agreement) に違反する場合がある [7]．例えば応答時間が 3000 (ms) を超えると Web サイトにアクセスした 53%が Web サイトから離れることが報告されている [8]．このことから本稿では，応答時間が 3000 (ms) を超えた場合は SLA 違反として扱う．

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・課題を述べる．2 章では，関連研究の説明を行う．3 章では，本稿の提案について述べる．4 章では，実装とその実験環境について述べる．5 章で評価と分析を行う．6 章で今後の課題や議論・考察を行う．最後に 7 章で, 最終的なまとめとする．

2. 関連研究

本稿の対象としている課題を関連研究ではどのように解決しているかについて述べる．Shengbo らは，K8s の GPU のスケジューリングを行うことで負荷分散と GPU リソースの最大化の改善を提案した [9]．この手法ではコンテナがデプロイされる際に GPU の割り当てを行っている．しかし，実際のコンテナが使用する GPU 量は見ておらずコンテナ内のアプリケーションが不明である．

Luis らは，クラウドデータセンターにおける CPU, メモリを共有する仮想マシン間で干渉の影響を削減するスケジューラを提案した [10]．この研究では KVM を対象としているため K8s 上の Pod は対象外となっている．

Rui らは，Virtual Network Function(VNF) を適切な場所に割り当てることで仮想ネットワークの信頼性を向上させるスケジューラを提案した [11]．この提案では K8s に対して VNF をスケジュールするために既存モデルを適応させるプラグインを実装している．

Lukasz らは，ノード間のネットワーク帯域幅をスケジュールにより節約し，アプリケーションの応答時間の削減を提案した [12]．エッジクラスタで遅延の影響を受けやすいワークロードを対象としているがスケジュールが決定した後のクラスタ状態，ネットワーク帯域幅の変動は考慮されていない．

Ghofrane らは，ユーザーがデプロイするジョブの種類に応じてノードの CPU や GPU のリソースを利用する効率的に共有するスケジューラを提案した [13]．このスケジューラには，さまざまなポッドの実行に関する履歴情報が必要である．そのため実行情報を集めるまで Pod のデプロイに時間がかかる．

Yuqi らは，K8s クラスタでのアプリケーションの実行中のジョブの進行状況を監視することで完了時間とメイクスパンを削減する [14]．

K8s に関するスケジューラでは, 既存の K8s のスケジューラに機能を追加するか別途のスケジューラを作成している．しかし，どれもアプリケーションのデプロイ前にスケジューラを実行させるため，アプリケーションの CPU, メモリの実測値を見ての再配置は行われていない．

3. 提案方式

本提案における前提条件を述べる．クラスタ数では，マスターノードは 1 台，ワーカーノードは 4 台で実験を想定している．そうすることでクラスタ全体は奇数となる．また Pod 及びノードのスケールは行わない．Pod の初期配置は kube-scheduler が決定し，その設定値は変更しないものとする．

提案方式

本稿における目的は，K8s クラスタ上の Pod や CPU の使用量の確保と Pod のアクセスによる CPU 変動量を考慮した再配置を行うことである．再配置では，Pod の CPU 使用率の中で最も取得できた回数が多い使用率を基準値とする．それにより Pod が常に使用する CPU 使用率を定義する．基準値から Pod の CPU 使用率の差分を変動量とし，基準値から今後増加する CPU 使用率を予測する．そうすることにより, Pod の CPU 使用率が高いもの同士を同一ノード上に配置することを回避し，応答時間を SLA 違反とならないようにする．

提案における条件として Pod に対するアクセスログが最低でも 1 週間分あるとする．アクセスログから 1 時間当たりのユーザからのアクセス数を 1 時間当たりのアクセス傾向とする．そこから Pod に対するアクセス傾向を月曜日から日曜日までの曜日ごとに 0 時から 23 時までのアクセス傾向を使用する．再配置におけるタイミングとしては，すべての曜日のアクセス傾向の限時刻が前一時間から増加する傾向である場合とする．理由としてアクセスが増加すると WordPress Pod が使用するノードの CPU 使用率も増加する．そのためこのタイミングで再配置を行うことで Pod が使用するノードの使用率を確保する．

提案として CPU 使用率の過去の推移からの CPU の変動量予測を用いた再配置を行う．下記にその手順を示す．

(1) Pod の CPU 使用率から基準値を算出
(2) 基準値からの CPU 使用率の変化量を算出
(3) 変化量をもとに各 Pod を順位づけ
(4) 順位ごとに各ノードの CPU 上限を超えないように再配置

まず初めに Pod が使用した CPU 使用率から基準値を算出する．Pod の CPU 使用率をもとに 0-100%の階級別に出現頻度を求める．その中で最も出現回数が多いものを Pod の CPU 使用率の基準値とする．データ数:T, Pod の CPU 使用率:P_c, Pod の CPU 使用率の基準値:P_s とすると式 (1) のように求められる．

$$P_s = \max(\frac{P_c}{T}) \tag{1}$$

次に基準値 P_s からの CPU 使用率の変化量の算出をする．基準値 P_s から Pod の CPU 使用率を除算し Pod の CPU 使用率の変化量を算出する．その後 Pod の CPU 使用率をもとに 0-100%の階級別に出現頻度を求め，最も出現回数が多いものを Pod の CPU 使用率の変化量とする．変化量:P_r とすると式 (2) ように求められる．

$$P_r = \max(\frac{P_s - P_c}{T}) \tag{2}$$

変化量をもとに各 Pod を順位づけをする．変化量が大きいものから順位をつける．K8s クラスタ内の Pod が 6 個であれば 1~6 までの順位をつける．Pod 数を P_N とすると Pod につける順位の範囲は式 (3) となる．

$$1 \leq x \leq P_N \tag{3}$$

変化量をもとに各 Pod を順位付けを行う．順位付けにおける例を図 6 に示す．

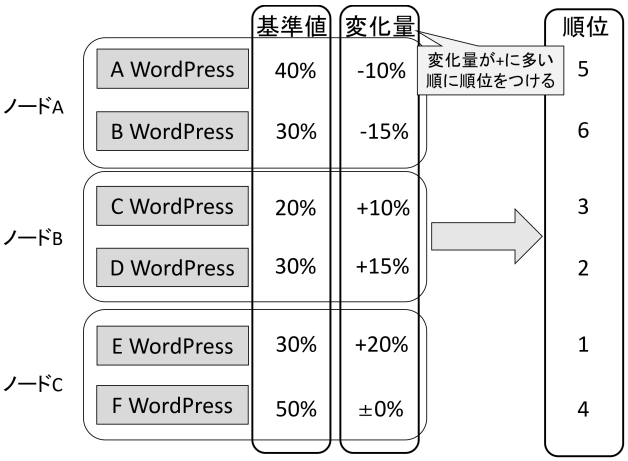

図 6 変化量をもとに順位付け

図 6 では，変化量を降順に順位付けをする．例では E WordPress の変化量が+20%であるため順位は 1 となる．また B WordPress は変化量が-15%となるため順位が 6 となる．

最後に順位をもとに各ノードの CPU 量を超えないように再配置を行う．再配置における例を図 7 に示す．

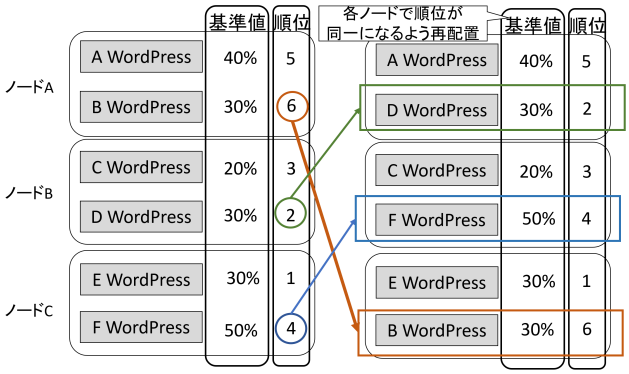

図 7 順位付け後の再配置

図 7 では，WordPress の Pod が 6 つデプロイされている．図 7 場合 1 から 6 までの順位がついておりその合計数は，21 となる．それをノード数の 3 で割ると順位の平均は 7 となり 1 ノードにおける順位合計が 7 となるように再配置を行う．

再配置を行う上で配置の移動を行う Pod は, 順位の低いものとする．理由として順位の高い Pod は変動量が多いものであるため，再配置に伴う移動によるユーザからのアクセス拒否を避ける．

ユースケース・シナリオ

K8s クラスタがクラウドベンダーからマルチテナントで提供されており，それを使用するユースケースを想定する．

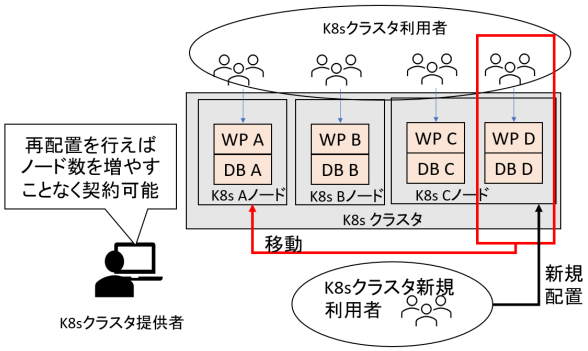

図 8　ユースケース図

K8s クラスタを利用する側では，個人のブログや企業用のサイトや商品を販売する EC サイトを WordPress にて構築する．利用するユーザは Pod に対してのアクセスを，K8s クラスタの提供側は 1 つのサーバに配置できる Pod 数を可能な限り多くする．これにより，サーバに対する契約数の増加が見込まれる．

4.　実装と実験方法

実装

本稿の実装は，K8s クラスタとは別のアクセスサーバにて実装される．その構成図を図 9 に示す．

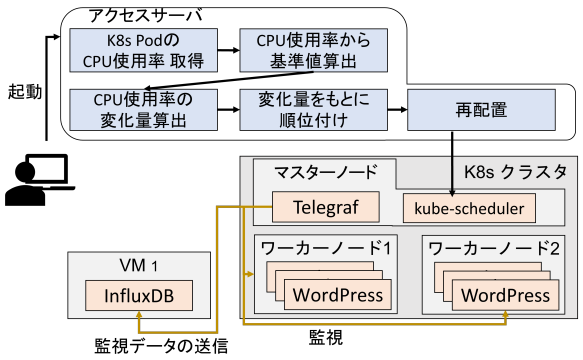

図 9　ソフトウェア構成図

実装は Python3 にて行う．K8s クラスタの CPU 使用率を取得するため，K8s クラスタのマスターノード上に Telegraf を配置し，ノードの監視をする．Telegraf で監視したデータは，別 VM に構築した InfluxDB に保存する．

実験環境

実装環境は Ubuntu22.04 の VM5 台で構築した K8s クラスタと別 VM でアクセスサーバを 1 台を使用する．下記に VM の基本構成，実験環境を図 10 に示す．

- vCPU x 4
- RAM 4GB
- HDD 40GB
- RKE2 v1.22.9

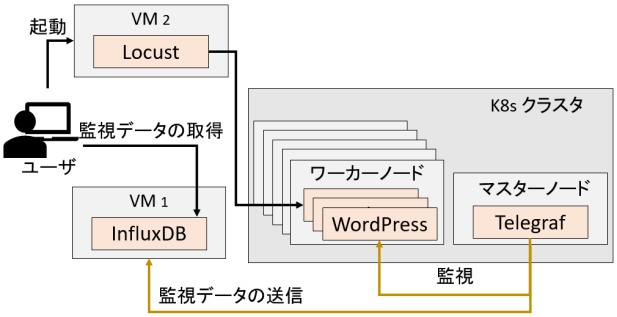

図 10　実験環境図

負荷テストツールは，Locust を使用する．Locust は Docker-Compose で起動し，Worker を 5 台に設定して負荷をかける．また負荷テストツールは上記の構成と同じ VM を別途構築し，その VM 上で起動する．

K8s クラスタ上の Pod の CPU 使用率の取得には，Telegraf を使用し，格納先として InfluxDB を使用する．Telegrf は，K8s クラスタのマスターノードに配置され，InfluxDB は K8s クラスタ外に設置された別の VM に構築する．

5.　評価と分析

評価実験では，WordPress Pod を 7 つ作成する．理由として WordPress の Pod に基礎実験で用いた月曜日から日曜日の各曜日ごとのアクセス傾向を各 Pod に適応させて実験を行うためである．各 WordPress の Pod は曜日ごとの傾向を適応させた Mon, Tue, Wed, Thu, Fri, Sat, Sun まであり，Mon は月曜日にアクセス傾向に対応している．WordPress Pod 1 つに対して対応する Namespece を作成し，1 つの Namespace に対して 1 つの WordPress Pod がデプロイされる．K8s の kube-scheduler で配置された状態と提案する再配置手法を用いた場合の応答時間の変化と Pod が使用するノードの CPU 使用率を比較する．また課題で定義した SLA 違反である 3000 (ms) を超える確率の比較も行う．

表 2 に配置前後のノードにおける Pod 数を示す．再配

表 2　再配置前後のノードに対する Pod 数

配置先	ワーカーノード 1	ワーカーノード 2	ワーカーノード 3	ワーカーノード 4
再配置前 Pod 数	0	2	2	3
再配置後 Pod 数	1	2	2	2

置前では，ワーカーノードが 4 つあるのに対してワーカ

ノード 1 には，配置されない結果となった．再配置後では，ワーカーノードに対して均等に Pod 数を配置する結果となった．再配置前後による WordPress Pod のノードの CPU 使用率と応答時間を配置対象の WordPress Pod と配置対象外の WordPress Pod で比較する．その結果を図 11, 12, 13, 14, 15, 16, 17, 18 に示す．

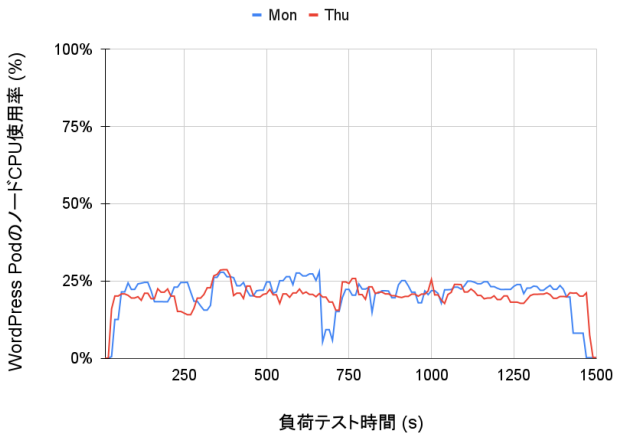

図 11 再配置対象の WordPress Pod のノードの CPU 使用率 (再配置前)

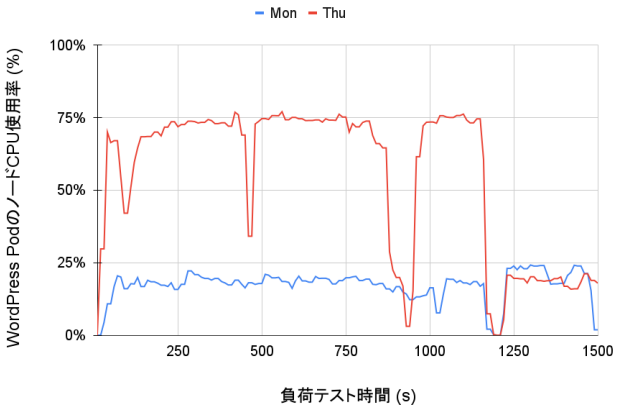

図 12 再配置対象の WordPress Pod のノードの CPU 使用率 (再配置後)

図 11 では，再配置対象となった Mon, Thu の WordPress Pod の CPU 使用率は，25%に推移している．再配置後の図 12 では，Mon の WordPress Pod の CPU 使用率が 20%に推移し，Thu の WordPress Pod は CPU 使用率が 75%にまで跳ね上がる結果になった．Thu に関しては，再配置により同一ノードに配置される Pod がいなくなったため，CPU 使用率が 50%になったと考える．しかし, Mon では同一ノードに置かれる Pod が変更されたためその Pod が使用する CPU 使用率が再配置前の Pod よりも CPU 使用率が高い Pod であると考える．

図 13 では，Tue の WordPress Pod の CPU 使用率は平

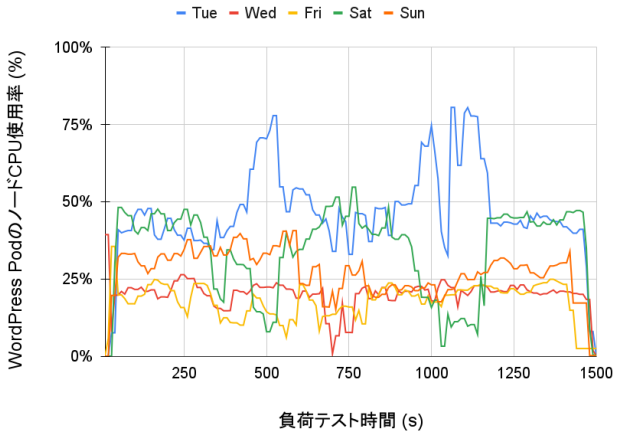

図 13 再配置対象外の WordPress Pod のノードの CPU 使用率 (再配置前)

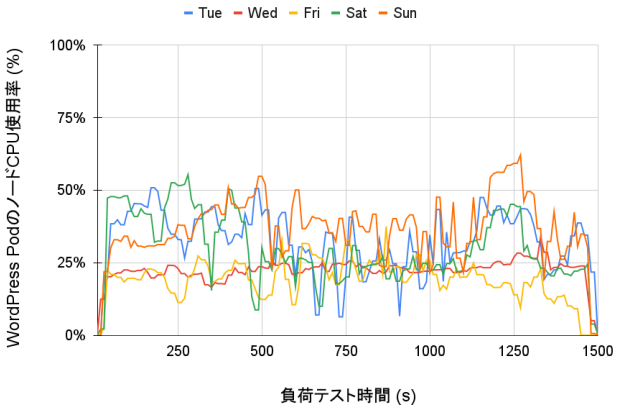

図 14 再配置対象外の WordPress Pod のノードの CPU 使用率 (再配置後)

均 45%，最大で 75%となった．図 14 では，再配置が行われた後の CPU 使用率では，Tue の CPU 使用率が 10%減少した．Sun の CPU 使用率は 10%増加した．これも図 11, 12 と同じく同一ノードに置かれる Pod が変更されたためその Pod が使用する CPU 使用率との競合が行われているためと考える．

図 15 では, Mon の平均応答時間が平均 2400 (ms), Thu の平均応答時間が 7990 (ms) となった．再配置後の図 16 を見ると Mon の平均応答時間が平均 3010 (ms), Thu の平均応答時間が 4540 (ms) となり, Mon の平均応答時間は 610 (ms) 増加，Thu の平均応答時間は 3450 (ms) 減少した．図 11, 12 での CPU 使用率が増加した Thu に関しては，応答時間が減少し，CPU 使用率が減少した Mon は応答時間が増加した．

図 17 と図 18 を比較すると再配置の前後で平均応答時間が減少したのは Sat, Sun である．逆に Fri に関しては，平均応答時間が増加した．Tue, Wed に関しては，変化は 20 (ms) となり前後での変化としては少なく誤差の範囲だと

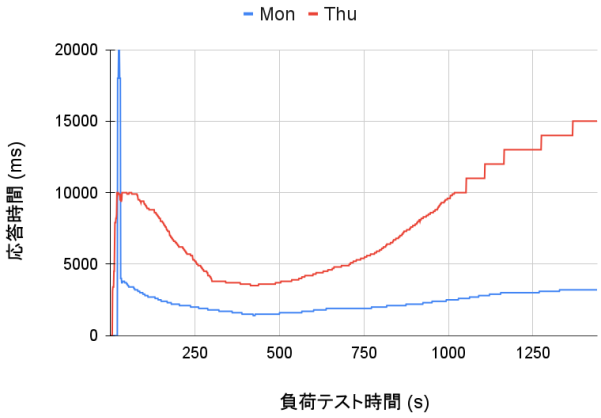

図 15 再配置対象の WordPress Pod の応答時間 (再配置前)

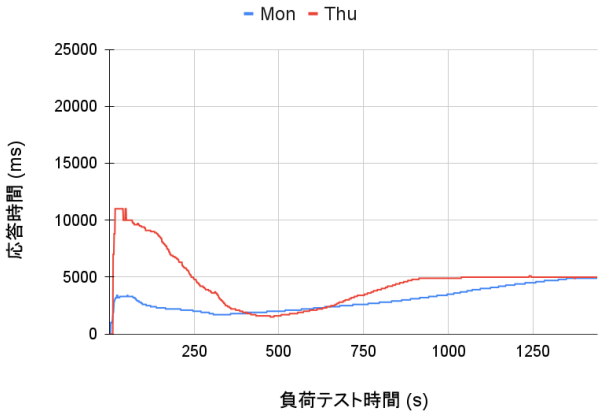

図 16 再配置対象の WordPress Pod の応答時間 (再配置後)

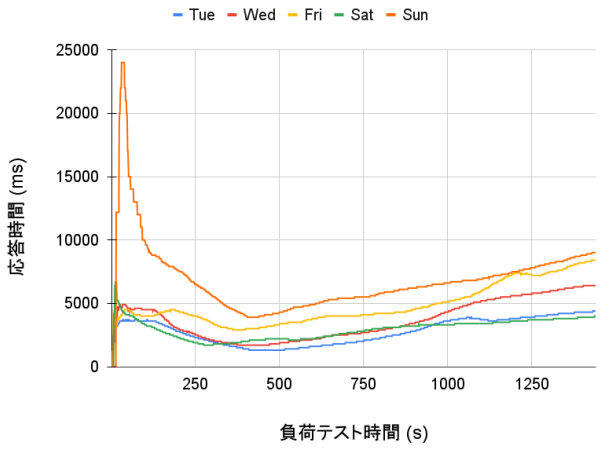

図 17 再配置対象外の WordPress Pod の応答時間 (再配置前)

思われる．

再配置対象の WordPress Pod はの応答時間は，Mon の Pod の再配置後では平均約 25%増加し，ThuPod の再配置後は平均約 43%減少した．

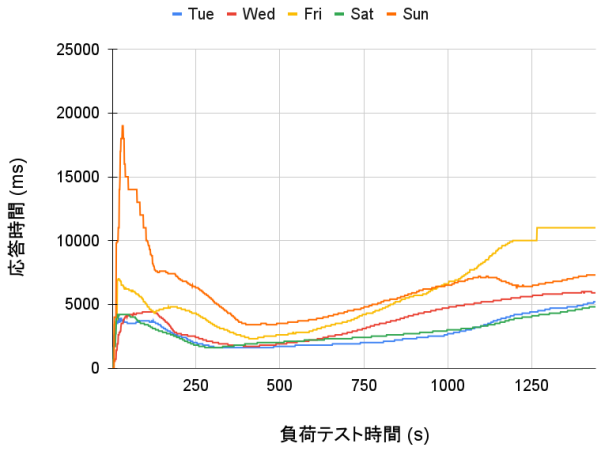

図 18 再配置対象外の WordPress Pod の応答時間 (再配置後)

6. 議論

本稿では，再配置を行うことで Pod が使用する CPU の競合により起こる Pod の応答時間に対しての改善を行った．提案では Pod が使用するノードの CPU 使用率から変動量を算出し，順位づけを行っている．この場合，1 つのノードにおける Pod の配置が配置された Pod の基準値+変動量がノードの上限を超えて配置されてしまう．その際にどの程度上限値を考慮するかが必須となってくる．また，WordPress Pod のノードの CPU 使用率が関係してくるのは，WorsPress Pod に対してのリクエスト数である．現状では WordPress Pod の CPU 使用率の変化量を見ている．そのため WordPress Pod に対するアクセス傾向の増加量を踏まえた再配置を行う．Pod がデプロイされてから再配置までの CPU 使用率とアクセス傾向を照らし合わせる．そうすることにより 1 時間ごとの CPU 使用率に対し，アクセス傾向の増減で Pod に対しての CPU 使用率の変化量が判明する．そして再配置のタイミング時点からアクセス傾向の増加率が最大になるまでの時間とその増加幅を求める．その増加幅と過去のアクセス傾向の増減で変動した CPU 使用率からアクセス傾向の増加分によって増加する CPU 使用率を求める．この増加量を基準に順位づけを行うことで改善をする．

本提案における条件として Pod に対するアクセスログが最低でも 1 週間分あるとした．しかし，各曜日のアクセスログからのアクセス数が次週のアクセス数と必ずしも同じ傾向にならない．そのため各曜日の傾向を出すのではなく 1 日単位で Pod のアクセスログ別のアクセス数を求める．そこから各 1 時間ごとにアクセス数の増減値を求める．0 時の場合，23 時のアクセス数を見てそこから 0 時のアクセス数の増減数を見る．過去のアクセスログからのアクセス数が何時の時点で増減しそれが全体の割合で最も多い点をアクセス数が増加する変化点として再配置を行う．

7. おわりに

課題として Pod がリクエストを処理する際に使用できる CPU 量によって応答時間が変化することを挙げた．本稿における目的は，K8s クラスタ上の Pod CPU の使用量の確保と Pod のアクセスによる CPU 変動量を考慮した配置を行うことである．提案として CPU 使用率の過去の推移からの CPU の変動量予測を用いた再配置を行った．結果として再配置を行うことで応答時間の改善が得られたが配置に伴う別 Pod の応答時間の低下が発生した．

参考文献

[1] Shah, J. and Dubaria, D.: Building Modern Clouds: Using Docker, Kubernetes & Google Cloud Platform, *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 0184–0189 (2019).

[2] El Haj Ahmed, G., Gil-Castiñeira, F. and Costa-Montenegro, E.: KubCG: A dynamic Kubernetes scheduler for heterogeneous clusters, *Software: Practice and Experience*, pp. 213–234 (2021).

[3] Townend, P., Clement, S., Burdett, D., Yang, R., Shaw, J., Slater, B. and Xu, J.: Invited Paper: Improving Data Center Efficiency Through Holistic Scheduling In Kubernetes, *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pp. 156–15610 (2019).

[4] Koskinen, T., Ihantola, P. and Karavirta, V.: Quality of WordPress Plug-Ins: An Overview of Security and User Ratings, *2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Confernece on Social Computing*, pp. 834–837 (2012).

[5] Cabot, J.: WordPress: A Content Management System to Democratize Publishing, *IEEE Software*, pp. 89–92 (2018).

[6] Carrión, C.: Kubernetes Scheduling: Taxonomy, Ongoing Issues and Challenges, *ACM Comput. Surv.*, (online), DOI: 10.1145/3539606 (2022). Just Accepted.

[7] More, D., Mehta, S., Pathak, P., Walase, L. and Abraham, J.: Achieving Energy Efficiency by Optimal Resource Utilisation in Cloud Environment, *2014 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, pp. 1–8 (online), DOI: 10.1109/CCEM.2014.7015479 (2014).

[8] Everman, B. and Zong, Z.: GreenWeb: Hosting High-Load Websites Using Low-Power Servers, *2018 Ninth International Green and Sustainable Computing Conference (IGSC)*, pp. 1–6 (online), DOI: 10.1109/IGCC.2018.8752138 (2018).

[9] Song, S., Deng, L., Gong, J. and Luo, H.: Gaia Scheduler: A Kubernetes-Based Scheduler Framework, *2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*, pp. 252–259 (2018).

[10] Tomás, L., Vázquez, C., Tordsson, J. and Moreno, G.: Reducing Noisy-Neighbor Impact with a Fuzzy Affinity-Aware Scheduler, *2015 International Conference on Cloud and Autonomic Computing*, pp. 33–44 (2015).

[11] Kang, R., Zhu, M., He, F., Sato, T. and Oki, E.: Design of Scheduler Plugins for Reliable Function Allocation in Kubernetes, *2021 17th International Conference on the Design of Reliable Communication Networks (DRCN)*, pp. 1–3 (online), DOI: 10.1109/DRCN51631.2021.9477366 (2021).

[12] Wojciechowski, x., Opasiak, K., Latusek, J., Wereski, M., Morales, V., Kim, T. and Hong, M.: NetMARKS: Network Metrics-AwaRe Kubernetes Scheduler Powered by Service Mesh, *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, pp. 1–9 (online), DOI: 10.1109/INFOCOM42981.2021.9488670 (2021).

[13] 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 (online), DOI: https://doi.org/10.1002/spe.2898 (2021).

[14] Fu, Y., Zhang, S., Terrero, J., Mao, Y., Liu, G., Li, S. and Tao, D.: Progress-based Container Scheduling for Short-lived Applications in a Kubernetes Cluster, *2019 IEEE International Conference on Big Data (Big Data)*, pp. 278–287 (online), DOI: 10.1109/BigData47090.2019.9006427 (2019).