深層強化学習を用いたWeb3層コンテナに対する仮想リソースの動的割り当て手法

Dynamic Allocation of Virtual Resources for Web Three-Tier Containers Using Deep Reinforcement Learning

北川 奨真 † 川北 英輝 † 水谷 后宏 ‡,§
Kitagawa Shoma Kawakita Toshiki Mizutani Kimihiro

1. はじめに

近年, 急速に普及している仮想化技術としてコンテナ仮想化技術がある. コンテナ仮想化技術とは, ホストコンピュータ上で仮想的なコンピュータをコンテナという単位で管理し, ホストコンピュータ上に複数のアプリケーション実行環境を仮想的に構築する技術である. この技術により, 同一のハードウェア上で異なるアプリケーションを隔離しながら実行できるため, リソースの効率的な利用が可能となる.

しかしながら, ホストコンピュータの計算リソースがコンテナ間で共有されるため, 特定のコンテナがリソースを過剰に使用すると, 他のコンテナのアプリケーション性能が低下する可能性がある. 例えば, あるコンテナのアプリケーションが高負荷状態になると, ホストコンピュータの CPU やメモリなどのリソースを独占し, 他のコンテナに十分なリソースが割り当てられなくなることがある. これにより, 他のコンテナで稼働しているアプリケーションの性能が劣化し, 全体のシステム効率が低下する可能性がある.

この問題を解決し, 1 台のホストコンピュータ上で多様なアプリケーションを複数のコンテナを用いて安定して運用するためには, CPU やメモリなどの計算リソースを適切に管理することが重要である.

既存研究 [1] では, 2 つのコンテナに強化学習を用いて, リソースの動的割り当てを行っている. しかし, この研究では, 報酬として, 前回からの応答時間の改善の有無を使用しており, 応答時間が大きくなれば +1, 変わらなければ 0, 小さくなれば-1 を取る. そのため, 小さな改善が過大評価されたり, 大きな改善が過小評価されたりするといったように, 改善の程度を報酬に表すことができない. また, Web3 層アーキテクチャに基づく Web3 層コンテナに実装した例はない.

そこで本研究では, 1 台のホストコンピュータ上で Web3 層コンテナが稼働する環境において, 稼働中のアプリケーションの性能劣化を最小限に抑えるために, 稼働中のコンテナの挙動に応じてリソースの制限や開放を自動的かつ適切に制御する深層強化学習を用いたアルゴリズムを提案し, その評価を行った. 本研究のアルゴリズムでは, 応答時間の逆数を報酬値として使用し, 報酬の程度に応じた評価を行うことで, 過大評価や過小評価の問題を解決する. また, Web3 層アーキテクチャに基づくコンテナ環境での動作を実現することで, より現実的なアプリケーション環境に対応することを目指している.

2. 関連技術

本研究では, Web3 層コンテナのリソース割り当てシステムを提案する. このシステムは, Web3 層アーキテクチャに基づき, プレゼンテーション層, アプリケーション層, およびデータ層をそれぞれ異なるコンテナに配置している.

2.1 Web3 層アーキテクチャ

Web3 層アーキテクチャは, 以下のように構成されている.

- **プレゼンテーション層**: ユーザインターフェース（UI）を提供する層である. Web ブラウザと Web サーバで実行され, ユーザと直接やり取りする役割を果たす.
- **アプリケーション層**: ビジネスロジックやアプリケーションの機能を提供する層である. アプリケーションサーバで実行され, ユーザからのリクエストを処理し, 必要なデータを取得して適切な処理を行った後, 結果をプレゼンテーション層に返す.
- **データ層**: アプリケーションで使用されるデータを管理する層である. データベースサーバで実行され, データベース管理システム（DBMS）を使用してデータを保存し, クエリを実行してデータを取得する.

† 近畿大学大学院総合理工学研究科, Graduate School of Science and Engineering, Kindai University
‡ 近畿大学情報学部, Faculty of Informatics (KDIX), Kindai University
§ 近畿大学情報学研究所, Cyber Informatics Research Institute, Kindai University

2.2 Web3 層アーキテクチャの利点

Web3 層アーキテクチャの利点として, スケーラビリティの向上が挙げられる. 各層を独立したコンテナに配置することで, 特定の層の負荷が増加した場合でも, その層だけをスケールアウトすることが可能となる. また, 管理性の向上も期待できる. 各層が独立しているため, メンテナンスやファイルを実際の Web サーバ上に配置して, 利用できる状態にすること (デプロイ) が容易になり, 特定の層に対する変更が他の層に影響を与えることなく行える.

さらに, リソース効率の最適化も可能である. リソース割り当てシステムを用いることで, 各コンテナのリソース利用を動的に調整し, 効率的なリソース利用を実現できる. 信頼性の向上も重要なメリットの一つである. 各層が独立しているため, 特定の層に問題が発生しても他の層に影響を与えにくく, 全体のシステムの信頼性が向上する.

最後に, 柔軟性の向上も期待される. 新しい技術やフレームワークを各層に導入しやすく, システム全体の技術スタックを柔軟に更新できる.

このように, Web3 層アーキテクチャを採用することで, スケーラビリティ, 管理性, リソース効率, 信頼性, 柔軟性の各面で大きなメリットが得られる.

2.3 仮想化技術

仮想化技術は, 物理的なハードウェアリソースを抽象化し, 複数の仮想マシン（Virtual Machines, VMs）として利用可能にする技術である. これにより, 物理ハードウェアの効率的な利用が可能となり, 異なる OS やアプリケーションを同一ハードウェア上で独立して動作させることができる.

2.4 コンテナ型仮想化と Docker

コンテナ型仮想化は, 仮想化技術の一種であり, ホスト OS 上で軽量なコンテナを実行することにより, アプリケーションとその依存関係を分離・管理する手法である. コンテナは, 仮想マシンに比べてリソースのオーバーヘッドが少なく, 高速に起動・停止できるという利点を持つ. この技術により, 開発環境と本番環境の一貫性を保ちながら, アプリケーションのデプロイやスケーリングが簡素化される.

コンテナ型仮想化技術の代表的なプラットフォームが Docker である. Docker を使用することで, アプリケーションのデプロイ, スケーリング, および管理が簡素化される.

Docker の主な特徴として, アプリケーションとその依存関係を含む静的なテンプレートであるイメージ, イメージを基に作成される軽量で高速に起動できるランタイム環境であるコンテナ, 共有可能なイメージのリポジトリである Docker Hub, そして Kubernetes などのツールを用いて複数のコンテナを管理・調整するオーケストレーション機能がある.

2.5 強化学習

強化学習は, エージェントが環境との相互作用を通じて行動を学習し, 報酬を最大化する手法である. 基本的な概念は以下の通りである.

- **エージェント**: 学習を行う主体. エージェントは環境に対して行動を選択する.
- **環境**: エージェントが相互作用する対象. 環境はエージェントの行動に応じて状態を変化させ, 報酬を与える.
- **状態**: 環境の現在の状況を表す情報. エージェントはこの状態を観測して行動を選択する.
- **行動**: エージェントが取る具体的なアクション. 行動によって環境の状態が変化する.
- **報酬**: エージェントの行動の結果として環境から与えられるフィードバック. エージェントは報酬を最大化するように行動を学習する.

状態を観測し, 行動を行うことで新たな状態に遷移する. このとき, 報酬関数の累積総和を最大化するような遷移先を決定することで, エージェントは最適な行動戦略を学習し, 環境との相互作用を通じて効果的に目標達成を実現する.

2.6 深層強化学習

深層強化学習（Deep Reinforcement Learning, DRL）は, 従来の強化学習にディープラーニングを組み合わせたもので, 特に複雑な問題に対して高いパフォーマンスを発揮する. DRL の利点として, ディープニューラルネットワークを用いることで大量のデータから複雑な状態と行動の関係をモデル化できるため, 従来の強化学習では扱えなかった高次元の状態空間にも対応可能となることが挙げられる. また, ディープニューラルネットワークは非線形な関係を学習する能力があるため, より複雑な行動戦略を学習できる. さらに, 大規模なデータセットを利用することでより精度の高いモデルを構築できるため, スケーラブルなシステムの最適化が可能となる.

2.6.1 DDQN

深層強化学習（Deep Reinforcement Learning, DRL）の一つとして, DQN[2], DDQN（Double Deep Q-Network）[3] について説明する.

DQN の状態 s_t において行動 a_t の価値である Q 値の更新式を式 (1) に表す.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right] \quad (1)$$

ここで, $\max_a Q(s_{t+1}, a)$ は次の状態 s_{t+1} におけるすべての行動 a の中で最大の Q 値を示す. この式では, 次の状態での最適な行動の Q 値を現在の Q ネットワークを使って評価しているため, Q 値が過大評価される可能性がある.

DDQN は, この問題を解決するために, 提案された手法である. DDQN の Q 値の更新式を式 (2) に表す.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max Q_{\text{target}}(s_{t+1}, a_{\text{main}}) - Q(s_t, a_t) \right] \quad (2)$$

ここで, $a_{\text{main}} = \arg\max_a Q(s_{t+1}, a)$ である. DDQN では, 次の状態 s_{t+1} での最適な行動 a_{main} を現在の Q ネットワークで選択し, その行動の Q 値をターゲット Q ネットワーク（Q_{target}）で評価する. この分離により, 次の状態での行動選択とその評価が異なるニューラルネットワークで行われ, Q 値の過大評価を減少させることができる.

Q ネットワークとターゲット Q ネットワークについては以下の通りである.

- **Q ネットワーク**: Q ネットワークは, 状態-行動ペアに対する Q 値を近似するために使用されるニューラルネットワークである. エージェントは Q ネットワークを用いて, 現在の状態における各行動の価値を評価する. 学習過程では, Q ネットワークのパラメータが更新され, より正確な行動価値の推定が行われる.
- **ターゲット Q ネットワーク**: ターゲット Q ネットワークは, Q ネットワークとは独立に一定の周期で更新される固定されたニューラルネットワークである. ターゲット Q ネットワークを使用することで, Q 値の評価が安定し, 学習の収束が向上する. これにより, Q 値の過大評価による学習の不安定性を抑えることができる.

ここで, θ は Q ネットワークのパラメータ, θ^- はターゲット Q ネットワークのパラメータである. α は学習率, γ は割引率を表す.

DDQN は, このようにして Q 値の過大評価を抑え, より安定した学習を実現する.

3. 提案手法

本研究では, Docker コンテナの負荷に応じて自動的にリソースを割り当てるシステムを提案する. このシステムは, DDQN（Double Deep Q-Network）を用いてリソースの動的割り当てを行う.

DDQN の実装には, 深層強化学習ライブラリの PFRL[4], 強化学習の学習 API を提供する Gymnasium[5] を使用し, Docker へのリソース動的割り当てには Docker Engine API[6] を使用する. これにより, 各コンテナに対する CPU およびメモリへのリソース割り当て操作がリアルタイムで可能となる.

3.1 強化学習の構成

提案手法の強化学習の構成は以下の通りである.

- **エージェント**: エージェントはリソース制御アルゴリズムとして機能する. DDQN を用いて, 最適なリソース割り当てを学習する.
- **環境**: 環境は Docker コンテナであり, エージェントが操作する対象となる. 各コンテナのリソース状況を観測し, 行動を実行する.
- **状態**: 状態はリソース状況を表し, 具体的には各コンテナの CPU およびメモリの使用率などが含まれる.
- **行動**: 行動は CPU およびメモリリソースの増減を指す. エージェントはこれに基づいて各コンテナのリソースを調整する.
- **報酬**: 報酬は応答時間の逆数とし, 応答時間が短くなるほど報酬が高くなるように設定する. これにより, エージェントは応答時間を最小化する行動を学習する.

状態の例を図 1 に示す. コンテナ数が N の場合, $i(0 \leq i \leq N-1)$ 番目のコンテナの CPU に関する情報は配列の $2i, 2i+2N$ 番目の要素, メモリに関する情報は配列の $2i+1, 2i+2N+1$ 番目の要素が対応する. また, 配列の要素 $X(0 \leq X \leq 2N-1)$ が割り当てリソースに対する使用率, 配列の要素 $Y(2N \leq Y \leq 4N-1)$ がリソース割り当て上限に対する割り当て率を表している.

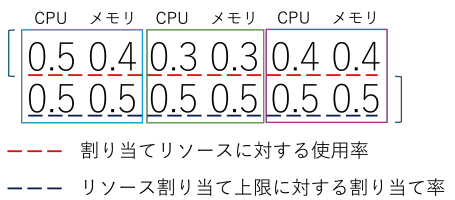

図 1: 状態の例

例えば, 1 つのコンテナに上限 8 GB のメモリを割り当てることができ, 現在 2GB を割り当て, 実際にコンテナが 1GB を使用しているとすると, 次のようになる.

- 配列の 0 ～ 5 番目の要素：使用中の 1GB に対する割り当てられた 2GB の比率

$$\frac{1\text{GB}}{2\text{GB}} = 0.5$$

- 配列の 6 ～ 11 番目の要素：割り当てられた 2GB に対する上限 8GB の比率

$$\frac{2\text{GB}}{8\text{GB}} = 0.25$$

提案手法は, これらの要素を組み合わせることで, 動的に変化する負荷に対して効率的にリソースを割り当てるシステムを実現する. DDQN を用いることで, リソース割り当ての最適化を図り, 全体の応答時間を最小化することを目指す.

4. 実験・評価

4.1 実験環境

実験は, 以下の環境で行った.

図 2 のように, Web3 層アーキテクチャに基づく 3 コンテナを Docker 上に作成した. プレゼンテーション層には Web コンテナとして nginx を使用している. nginx は, 高性能な HTTP サーバおよびリバースプロキシである. アプリケーション層には Flask を使用している. Flask は Python で書かれた軽量な Web フレームワークである. データ層には MySQL を使用している. MySQL は, オープンソースのリレーショナルデータベース管理システム（RDBMS）である.

ユーザからのリクエストがあった場合, これらのコンテナが相互に処理を行う. 具体的には, nginx がリクエストを受け取り, Flask アプリケーションに転送し, Flask がビジネスロジックを処理し, 必要に応じて MySQL データベースにアクセスしてデータを取得または保存する.

4.2 実験に用いた計算機の環境

実験に使用した計算機の環境を表 1, 表 2 に示す. このとき, 環境 (Docker) が動作している計算機が表 1, エージェントが動作している計算機が表 2 である.

項目	サーバ
CPU1	Intel Xeon Silver 4214R
CPU2	Intel Xeon Silver 4214R
RAM	64 GB
OS	Ubuntu 22.04.4 LTS

表 1: 計算機サーバの環境

項目	クライアント
CPU	Intel Core i7-9700K
GPU	GeForce RTX 2070 SUPER
RAM	32 GB
OS	Ubuntu 22.04.4 LTS

表 2: 計算機クライアントの環境

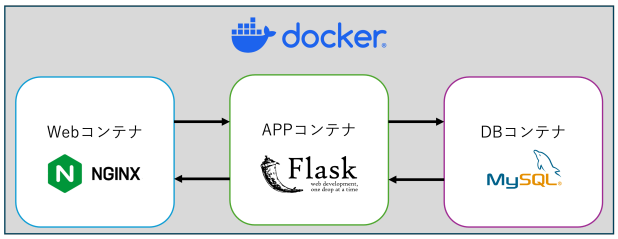

図 2: Web3 層コンテナ

4.3 評価手法

4.1 節の環境に対して, 3. 章の深層強化学習を用いた動的リソース割り当てアルゴリズムを実行した. このとき, 各コンテナの CPU, メモリリソースの割り当て上限, 割り当て下限, 初期割り当てを表 3 の通りに行った. また, 本実験で設定したパラメータを下記に示す.

- **割引率（discount factor）** γ: 将来の報酬をどの程度現在の価値に割り引いて考慮するかを決定するパラメータである. 値が 1 に近いほど, 将来の報酬をより重視することを意味する. 本実験では 0.99 と設定した.
- **リプレイ開始サイズ（replay start size）**: 学習を開始する前に, 経験リプレイバッファに蓄積されるべき遷移の数を指定する. 本実験では 100 と設定した.
- **更新間隔（update interval）**: Q ネットワークの重みを更新する間隔を指定する. 本実験では 1 と設定した.
- **ターゲットネットワーク更新間隔（target update interval）**: ターゲット Q ネットワークの重みを更新する間隔を指定する. 本実験では 100 と設定した.

探索率 (ϵ) は `LinearDecayEpsilonGreedy` を使用して設定した. 探索率 (ϵ) とは, ϵ の確率でランダムに行動, それ以外の確率 $(1-\epsilon)$ で最も期待値の高い行動を選択することを表す.

`LinearDecayEpsilonGreedy` では, この ϵ を学習の進行に伴って線形に減少させることで, 初期段階では探索

項目	CPU	メモリ
割り当て上限	100%	100 MB
割り当て下限	10%	10 MB
初期割り当て	50%	50 MB

表 3: リソース割り当ての設定

を重視し, 学習が進むにつれて利用を重視するように調整する. これにより, 初期のランダムな探索を通じて多様な行動を試し, 後半では学習した知識を基に効率的な行動を選択することができる.

以下の図 3 は, 横軸に学習ステップ数, 縦軸に報酬をとっている. ここで, 本実験では, 行動を選択後, リソースを割り当て, 応答時間から報酬を決定するという一連の手順を 1 ステップとし, 1 エピソードは 100 ステップとしている. また, `LinearDecayEpsilonGreedy` を δ ステップで減少させる.

学習ステップが進むごとに高い報酬を得ていることから, リソースの適切な割り当てにより, 各コンテナの性能向上を達成していることが確認できた.

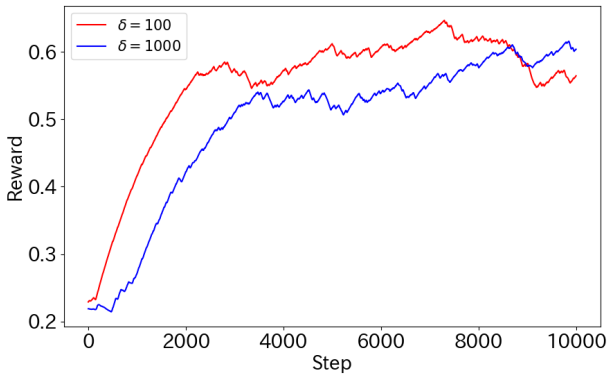

図 3: 報酬の推移

5. 結論・今後の課題

本研究では, 深層強化学習（DDQN）を用いて Web3 層コンテナシステムに対する動的リソース割り当て手法を提案し, 評価を行った. 実験の結果, 提案手法が動的に変化する負荷に対して効率的にリソースを割り当て, システムの応答時間を最適化する能力を有することを確認した. また, 探索率の線形減少を通じて, 初期段階での探索と学習が進むにつれての効率的な行動選択が可能であることも示した.

5.1 今後の課題

今後の課題として, 以下の点が挙げられる.

- **複雑なシステムへの適用**: 本研究では, Web3 層アーキテクチャを対象としたが, より複雑なシステムや多様なアプリケーション環境への適用を検討する必要がある. 特に, 異なるリソース要件を持つアプリケーション間でのリソース競合をどのように最適化するかが重要である.
- **リアルタイム適応**: 提案手法がリアルタイムでのリソース割り当てにどの程度適応できるかを評価するため, より高速な学習アルゴリズムや効率的なリソースモニタリング手法の開発が求められる.
- **多目的最適化**: 本研究では応答時間の最小化を目標としたが, エネルギー消費やコストなど, 他のパフォーマンス指標を同時に最適化する多目的最適化手法の検討が必要である.
- **長期的な評価**: 提案手法の長期的な安定性や効果を評価するために, より長期間にわたる実験や現実環境でのデプロイが必要である.

これらの課題に取り組むことで, 提案手法の実用性と汎用性をさらに高めることが期待される.

謝辞

本研究は JSPS 科研費 JP23K16876 の助成を受けたものである.

参考文献

[1] 川北英輝, 水谷后宏. 実仮想環境下における深層強化学習を用いた効率的なリソース管理手法の提案. Proc. 情報処理学会関西支部支部大会, pp. G–01, 2022.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, et al. Playing atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*, 2013.

[3] H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *Proc. AAAI Conference on Artificial Intelligence*, 2016.

[4] Pfrl. `https://github.com/pfnet/pfrl`.

[5] Gymnasium. `https://gymnasium.farama.org/`.

[6] Docker engine api. `https://docs.docker.com/engine/api/`.