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

An effective resource management method using deep reinforcement learning on virtual environment

川北 英輝 † 水谷 后宏 ‡,§
Toshiki Kawakita Kimihiro Mizutani

1. はじめに

近年，急速に利用が進んで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. 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，特定のユーザが使用するコンテナがホストコンピュータのリソースを占有するようなことがあると，一部のコンテナのアプリケーションが高負荷になったり，適切な計算リソースが割り当てられない場合，コンテナのアプリケーションの性能劣化が発生する可能性がある. こうしたことを防ぎ，1 台のホストコンピュータ上で多種多様なアプリケーションを複数のコンテナを用いて運用するためには，CPU やメモリ等の計算リソースの適切な管理が重要である. しかし，既存研究はシミュレーションを用いた検証が多くなされており，学習制御を Docker と組み合わせて実装に至った例は少ない [1]，[2].

そこで，本研究では，1 台のホストコンピュータ上で複数の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できるだけ発生させないようにするため，稼働中のコンテナの挙動に応じて，利用するリソースの制限や開放を自動的に適切に制御するアルゴリズムを実装していく事を目指す.

2. 関連技術

2.1 仮想化技術

仮想化とは，図 1 のように仮想化ソフトウェアによってサーバなどのハードウェアリソース (CPU，メモリ，ディスクなど) を抽象化することで，物理的な制限にとらわれずに，1 台の物理サーバの上に複数の仮想サーバを統合・分割できるようにする技術である. 仮想化による主なメリットとしては以下が挙げられる.

- 物理サーバ台数の削減によりコストが削減できる.
- 物理サーバを集約して仮想化するため，省エネ・省スペースとなる.
- ハードウェアリソースを効率的に配分できるため，リソースの利用効率が向上する.
- 処理能力の変化や構成変更などの要求に素早く対応できる.
- ソフトウェア上で一元管理するため，運用管理コストが削減できる.
- 同一サーバ上で複数の OS が稼動可能であるため，ソフトウェアの依存関係問題を最小化することができる.

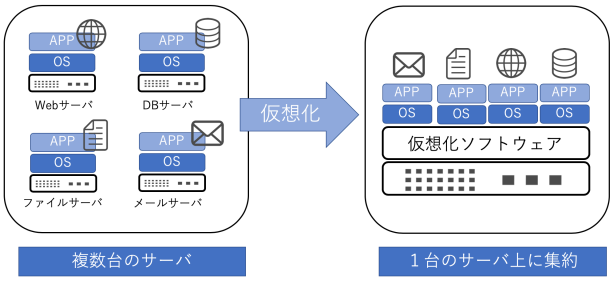

図 1: 仮想化の仕組み

2.2 コンテナ型仮想化

コンテナ型仮想化とは，仮想化の手法の 1 つであり，ホスト OS 上でアプリケーションの実行環境をコンテナと呼ばれる仮想的な環境にパッケージ化し，コンテナエンジンと呼ばれるコンテナの管理や動作環境を提供する仮想化ソフトウェア上で動作させる技術のことである. 各コンテナはホスト OS のカーネルを共有しホスト OS 上の 1 プロセスとして動作するため，コンテナ内で新たに OS を立ち上げる必要がない. そのため，オーバヘッドが少なくアプリケーションを高速，軽量に起動できる. 代表的なコンテナエンジンの例として, Docker[3] などが存在する.

2.3 Docker

Docker とは，コンテナエンジンの代表的な実装例であり，開発者やシステム管理者がコンテナ仮想化を用いて

† 近畿大学大学院総合理工学研究科, Graduate School of Science and Engineering Research, 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

アプリケーションを構築，実行，共有するためのプラットフォームの総称である．図 2 に示すように，Docker が動作している環境であれば，Docker イメージと呼ばれるコンテナを作成するための依存関係や情報を含むパッケージを元に，コンテナを作成するだけでアプリケーションを動作させることができるため，環境構築が容易である．また，Docker イメージを配布することで環境を手軽に共有できるといった特徴がある．

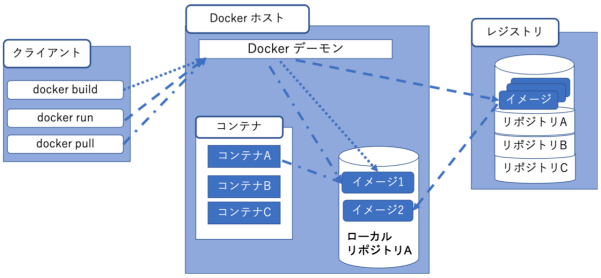

図 2: Docker のアーキテクチャ

2.4 強化学習を用いたリソース制御

強化学習 [4] は，制御対象である環境と制御主体であるエージェントの主に二つの要素から成り立っており，エージェントが環境に対して行動を起こし，環境がその行動によって更新された状態と報酬をエージェントにフィードバックする．この一連のサイクルを回して学習が進み，得られる報酬の合計が最大化されるように，エージェントの行動を最適化していく．深層強化学習 [5]，[6] とは，強化学習に深層学習を組み合わせたものであり，ある状態における，ある行動の価値を意味する行動価値関数を Deep Neural Network (DNN) で近似することで，連続値や，より高次元な状態を扱うことを可能にしたものである．

また，これまで仮想環境下でのリソース割り当てを自動化する試みは多々行われているが，シミュレーションやエミュレーションを用いたものが多く，実装を通した検証を行なっている例は少ない [1]，[2]．

3. 提案手法

サーバの計算リソースを Docker コンテナの負荷に応じて，自動的に適切に割り当てる手法として強化学習を採用した．また，リソース量が将来的に増加していくことから，深層学習を応用した DoubleDQN(DDQN) という手法を適応することとした，DDQN の実装には Chainer RL[7]，リソースの割り当て部分に OpenAI Gym[8] を用いた．強化学習では，エージェント (リソース制御アルゴリズム) が行動 (リソースの割り当て) を行い，報酬 (負荷の改善) をより高く得られる状態 (割り当て状態) に遷移することを目指す．以下に，状態・行動・報酬の具体的な定義を示す．

- 状態：各 Docker コンテナに割り当てられたリソース状況の配列を表す．
  配列の index 番号は CPU コア番号と対応し，割り当てられている場合は 1，そうで無い場合は 0 となっている．また，最後の要素には割り当てられたメモリ上限を 0 ～ 1 で正規化した値となっている．
  図 3 に示した例は，あるコンテナが CPU コア 0 番，メモリ上限に割り当て可能な最大値の約半分が割り当てられている事を表す．

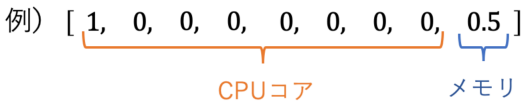

図 3: リソース状況の配列

- 行動：各 Docker コンテナに割り当てられているリソース構成の更新を行う．
  各 Docker コンテナに対して使用を許可する CPU コアの指定や，各 Docker コンテナに割り当てるメモリ最大使用量と，ディスクへのスワップを許可するメモリ容量の制限を行う．

- 報酬：行動の前後で平均処理時間の差を測定し，報酬を決定する．
  WEB サーバの性能測定ツールである Apache Bench を用いて，Docker コンテナにリクエストを送り，1 リクエストの平均処理時間が行動の前後で短ければ +1，長ければ −1，変化がなければ 0 とした．

提案手法のアーキテクチャを図 4 に示す．具体的には，状態 s_t において $[1, 0, 0, 0.5]$ は，コンテナに 1 つ目の CPU とメモリサイズ制限の約半分のメモリサイズが割り当てられていることを示す．行動 a_t で，2 つ目の CPU を新たに割り当てることで，コンテナのリソースの構成が変更され，状態 s_{t+1} に遷移し $[1, 1, 0, 0.5]$ となる．このとき，エージェントは報酬 r_t を獲得し，状態 s_t での行動 a_t を評価する関数である $Q(s_t, a_t)$ を更新する．Q 学習に基づく更新式を以下に示す [4]．α は，更新をどれだけ急激に行うかを制御するパラメータで，γ は，将来もらえる報酬をどれくらい現在の価値として考慮に入れるかを表すパラメータである．

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

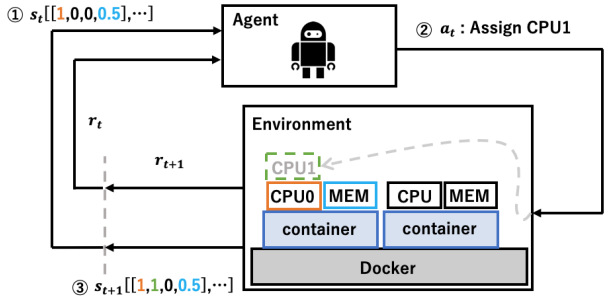

図 4: 提案手法のアーキテクチャ

4. 実験・評価

4.1 実験環境

本研究では，コンテナ型の仮想環境を作成・配布・実行するためのプラットフォームとして，Docker を用いた．研究室内のサーバ上に 2 つの Docker コンテナを稼働させ，各 Docker コンテナ内で Python の Web アプリケーションフレームワークである Flask を用いて，web アプリケーションを作成し稼働させ，各 Docker コンテナに対して負荷をかけた際に，適切な計算リソースを割り当てることができるかを試行した．なお，稼働直後は，各 Docker コンテナに対して CPU を 1 コアずつ，および 5GB のメモリを割り当てておき，片方の Docker コンテナに対してのみ CPU 使用率を 80%に指定し負荷がコンテナ毎で異なる環境を構築した．

4.2 提案手法の評価

4.1 小節で述べた実験環境に対して，3 章で述べた深層強化学習を用いたリソース制御のアルゴリズムを実行した．また，その際に深層強化学習のパラメータである割引率 (γ) と探索率 (ϵ) を変化させて実験を行った．割引率 (γ) とは，報酬の割引率を表し，過去の結果をどのくらい重要視するかを表す．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事を表す．

以下の図 5，図 6 は，横軸に学習ステップ数，縦軸に累積報酬をとっている．図 5 は，ϵ(探索率) の値を固定し，γ(割引率) の値を変化させた場合の累積報酬を，図 6 は，γ(割引率) を固定し，ϵ(探索率) の値を変化させた場合の累積報酬を示しており，それぞれ学習を 1000 ステップ行った．実験結果が示すように，学習ステップが進むにつれ累積報酬が増加していることから，リソースの適切な割り当てによるコンテナの性能向上を達成している事が確認できた．

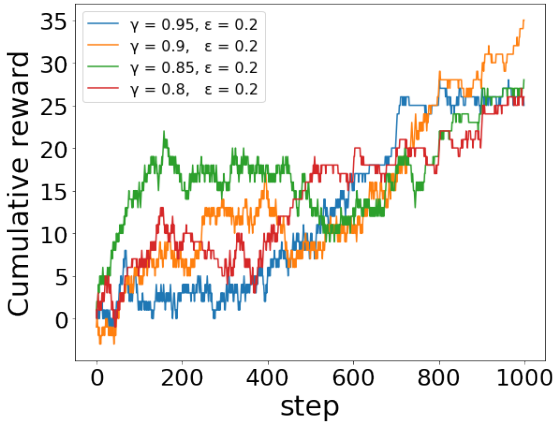

図 5: ϵ を固定し，γ を変化させた場合の累積報酬

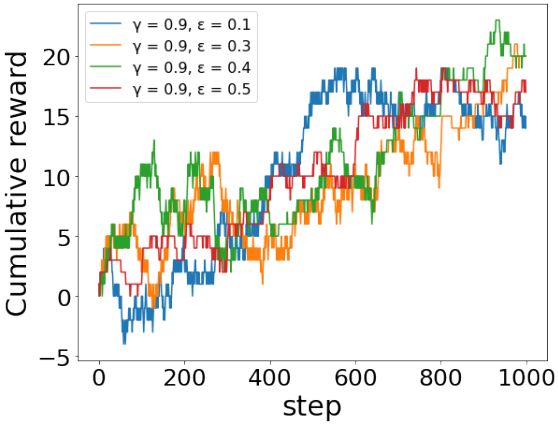

図 6: γ を固定し，ϵ を変化させた場合の累積報酬

5. 結論と今後の課題

本研究では，学習制御を Docker と組み合わせて実際に Docker から各コンテナが使用しているリソースの利用状況を取得し，その利用状況に応じて各コンテナにリソースを適切に割り当て，コンテナの動作性能を改善するようなアルゴリズムを実装する事を提案した．実験結果より，提案手法の有効性を示すことができた．今後の課題として，制御するコンテナ数やリソースを増やす，などが挙げられる．

参考文献

[1] H. Mao, et al. ”Resource Management with Deep Reinforcement Learning” in Proc. The 15th ACM Workshop on Hot Topics in Networks, pp. 50-56, 2016.

[2] Y. Xu et al, ”A Survey on Resource Allocation for 5G Heterogeneous Networks: Current Research,

Future Trends, and Challenges," in IEEE Communications Surveys & Tutorials, vol. 23, no. 2, pp. 668-695, 2021.

[3] Docker, https://www.docker.com/

[4] R.S. Sutton and A.G. Barto, Reinforcement learning: An introduction, vol.1, MIT press Cambridge, 1998.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, et al., "Humanlevel control through deep reinforcement learning," Nature, vol.518, no.7540, p.529, 2015.

[6] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," Thirtieth AAAI confer- ence on artificial intelligence, pp.2094–2100, 2016.

[7] Chainer, https://tutorials.chainer.org/ja/

[8] OpenAI Gym, https://gym.openai.com/