RESEARCH ARTICLE

INTERNATIONAL RESEARCH JOURNAL OF MULTIDISCIPLINARY TECHNOVATION

Asian Research Association

DOI: 10.54392/irjmt24614

A Deep Reinforcement Learning Framework for Optimized Container Scheduling and Load Balancing

Sreedar Bhukya [a,*], P. Suraj Goud [a], K. Yuvraj [a], Goud K. Harin [a]

[a] Department of Computer Science Engineering, Sreenidhi Institute of Science and Technology, Telangana-501301, India.

* Corresponding Author Email: sreedharb@gmail.com

DOI: https://doi.org/10.54392/irjmt24614

Received: 02-03-2024; Revised: 12-11-2024; Accepted: 19-11-2024; Published: 23-11-2024

Abstract: Unlike VMs, containerization is a modern method for packaging and deploying software in distributed environments like the cloud. Containers are widely used due to their efficient software packaging and deployment. Efficient management of containers is crucial in dynamic cloud environments with heterogeneous infrastructure. Deep learning techniques are being applied to optimize resource utilization in cloud environments, including mapping containers to suitable nodes for energy conservation. However, the existing works on container scheduling have limitations like inability to cope with dynamic runtime scenarios. To overcome this problem, the aim of this paper is to design and implement a framework using deep reinforcement learning techniques to improve container scheduling and load balancing. The proposed algorithm, Reinforcement Learning based Container Scheduling (RLbCS), uses an action-reward iterative approach to optimize container scheduling. Experimental results showed that RLbCS outperformed existing methods, achieving a 92% success rate in placing containers and optimizing resource utilization. The proposed method can be integrated with cloud-based systems to automatically schedule containers for resource optimization and load balancing.

Keywords: Container Services, Container Scheduling, Load Balancing, Cloud Computing, Deep Reinforcement Learning

1. Introduction

Unlike Virtual Machines (VMs) in virtualization technology used in cloud computing, containerization technology is the modern way of packaging software and deploying it in distributed environments like the cloud. The concept of containers is widely used in cloud computing environments due to perceived benefits linked to packaging and deploying software efficiently. Due to the unprecedented usage of containers, it became indispensable to manage them efficiently. In other words, container scheduling and load-balancing strategies have assumed significance in distributed environments. There are many existing contributions to the scheduling of containers.

For good QoS, resource allocation must be done efficiently. For the best job scheduling in cloud settings, this research suggests a Clipped Double Deep Q-learning method that outperforms conventional algorithms and reinforcement learning techniques [1]. Complicated resource interferences frequently make it difficult for cloud providers to place long-running applications (LRAs). Using deep reinforcement learning (RL) to deploy LRAs appropriately, Metis is a revolutionary scheduler that improves throughput by as much as 61% over conventional schedulers while providing quantitative criteria [2]. Improved flexibility to variations in workload, the research in suggests Reinforcement Learning (RL) methods for both forms of elasticity [3]. Elastic Docker Swarm incorporates RL versions such as model-based and Q-learning techniques. Unlike conventional threshold-based heuristics, model-based reinforcement learning is adaptable in prototypes and simulations. The Container-as-a-Service (CaaS) for Internet of Things workloads that is energy-efficient is proposed in this study. Resource allocation in simulated situations is done using a Stackelberg game, demonstrating enhanced service quality [4]. The existing heuristic methods need to be revised in highly dynamic cloud infrastructures where heterogeneity prevails. Scheduling containers in a distributed environment is complex. From the literature, it is clear that there are numerous issues related to container scheduling and load balancing. Existing heuristic approaches have limitations in handling dynamic runtime situations. Because heuristic methods are designed to have specific procedures, and they cannot learn from situations. Learning-based approaches have been found to be more efficient in dealing with the runtime environment. In this paper, we propose a methodology based on reinforcement learning

that is suitable for evaluating runtime situations and making appropriate decisions. The benefits of containerization include the ability to deploy applications in a "write once, run anywhere" fashion, along with other advantages such as ease of development and deployment. Our contributions to the scheduling of containers are as follows.

1. We proposed a framework for improving container scheduling and load balancing using an actor-critic approach on top of the deep reinforcement learning technique.
2. An algorithm known as Reinforcement learning-based Container Scheduling (RLbCS) exploits the runtime environment with an action-reward iterative approach for optimal container scheduling.
3. Experimental results revealed that RLbCS is efficient in resource optimization and convergence of the scheduling process. RLbCS outperformed other existing methods regarding the percentage of placed containers, with 92%.

The rest of the paper is structured as follows: Section 2 reviews the literature on prior container scheduling methods. Section 3 provides the required preliminary details. Section 4 presents our framework, underlying mechanisms, and algorithm. Section 5 provides empirical observations. Section 6 discusses the importance of our method and describes its limitations. Section 7 concludes our work.

2. Related Work

This section reviews prior works on container scheduling methods. Swarup *et al*. [1] stated that although handling more data presents issues, cloud computing is essential for data analysis, storage, and the Internet of Things. For good QoS, resource allocation must be done efficiently. For the best job scheduling in cloud settings, this research suggests a Clipped Double Deep Q-learning method that outperforms conventional algorithms and reinforcement learning techniques. Wang *et al*. [2] focused on complicated resource interferences that frequently make it difficult for cloud providers to place long-running applications (LRAs). Using deep reinforcement learning (RL) to deploy LRAs appropriately, Metis is a revolutionary scheduler that improves throughput by 61% over conventional schedulers while providing quantitative criteria. Rossi *et al*. [3] improved flexibility to variations in workload; this research suggests Reinforcement Learning (RL) methods for both forms of elasticity. Elastic Docker Swarm incorporates RL versions such as model-based and Q-learning techniques. Unlike conventional threshold-based heuristics, model-based reinforcement learning is adaptable in prototypes and simulations. Subsequent research endeavors to expand RL policies to encompass multi-component applications under fog computing settings, investigating decentralized control patterns to facilitate effective microservice implementation. Singh *et al*. [4] required practical edge computing; the growth of IoT infrastructure puts pressure on cloud data centers. The Container-as-a-Service (CaaS) for Internet of Things workloads that is energy-efficient is proposed in this study. Resource allocation in simulated situations uses a Stackelberg game, demonstrating enhanced service quality. Ahmad *et al*. [5] observed that relying heavily on containers requires careful scheduling. This review categorizes scheduling approaches, and their advantages, disadvantages, and prospects for growth are examined. Swarup *et al*. [6] used for many different activities, such as ultra-low latency IoT demands and Infrastructure as a Service. The technique reduces service latency by 10.85% and 31.02% and energy usage by 1.5% and 6.2%, respectively, outperforming FCFS, Random, and Q-learning schedulers. Future aims involve investigating dynamic approaches for optimization and improving rescheduling procedures.

Deng *et al*. [7] found that because SLO requirements are stringent, it is imperative to install Long-Running Applications (LRA) in cloud containers as efficiently as possible. To optimize for goals like avoiding SLO violations, this research offers a deep reinforcement learning (DRL) model that improves deployment performance by 56.2% compared to manual approaches. Wang *et al*. [8] opined that for Internet of Things applications, edge/fog computing is essential; however, server overload might cause problems. We present DRLIS, a Deep Reinforcement Learning-based scheduling system that balances server loads and maximizes heterogeneous IoT app response times in FogBus2. Mattia *et al*. [9] investigated decentralized scheduling and load balancing in Edge and Fog Computing, which are made possible by the P2PFaaS platform. Docker containers are used, providing mobility and versatility. The framework supports many algorithms, including Reinforcement Learning, and offers scheduling, discovery, and learning services. Future iterations will include energy-aware scheduling techniques and improved service management. Wang *et al*. [10] provided a scheduling technique for container cloud systems using CNN-BiGRU-Attention for load prediction. The results indicate a 21.7% increase in scheduling efficiency and a 37.4% improvement in forecast accuracy. Additional verification in actual industrial settings is required.

Li *et al*. [11] discussed effective resource scheduling for space-based systems and suggested a hierarchical approach based on deep Q network and ant colony optimization. According to the results, the work completion rate and resource balance are improved. Future research aims to enhance algorithm appropriateness and generality for in-orbit computing. Luo *et al*. [12] presented the dynamic, flexible job shop scheduling issue (DFJSP) with new job insertions using

a deep Q-network (DQN). The DQN works better than other dispatching rules and Q-learning agents in some production circumstances since it was trained using double DQN and soft target weight update. Future research should investigate policy-based RL techniques and incorporate more uncertainty. Oleghe *et al*. [13] examined the location of containers in edge computing while going over the frameworks and scheduling model techniques. It draws attention to the requirement of heuristic-based algorithms and additional models considering dispersed edge jobs. Zhang *et al*. [14] proposed a neural network and reinforcement learning-based edge computing workflow scheduling technique. Separating scheduling choices from workflow parsing maximizes reaction time and resource use. Compared to alternative approaches, the methodology exhibits an advantage in computing time and scheduling performance. Subsequent research endeavors to tackle the problem of computing time and implement the technique in cluster settings, suggesting a bi-phase scheduling algorithm and regulations for load balancing. Ma *et al*. [15] explained a container migration-based CMDM technique for load balancing in the Internet of Things. Comparing it with GA and SABFD, the load balancing efficiency increases by 7.3% and 12.5%, respectively. To implement edge computing in the power IoT, further work will focus on improving the MACS algorithm and job scheduling.

Ramos *et al*. [16] considered load balance as a hard constraint in the Container Loading Problem (CLP), according to rules by employing diagrams relevant to each vehicle. Stability and load balancing are maintained without sacrificing container capacity utilization thanks to a multi-population BRKGA algorithm with a novel fitness function. Future research aims to investigate mathematical programming techniques and integrate CLP with the vehicle routing problem. Baburao *et al*. [17] focused on efficiency and effectiveness while introducing dynamic matrix factorization for coevolving time series missing value prediction. The results show that performance is still enhanced even with large missing ratios. Taluder *et al*. [18] found that the DTLB model for SDN settings suggests a mechanism for process selection, migration, and assignment utilizing Docker containers to achieve effective load balancing with better energy usage and reaction time. Kaur *et al*. [19] improved by two orders of magnitude when an ITO layer is inserted between printed silver and ZnO. The decreased band offsets or carrier buildup improves TFT performance. Aruna *et al*. [20] observed that, for effective load balancing in Docker containers, the ACO-LWC algorithm is suggested. It outperforms current methods in terms of error rates and response times.

Patra *et al*. [21] suggested using Simulated Annealing in conjunction with Grey Wolf Optimization (GWO) to handle load balancing in cloud containers. Unlike other algorithms, GWO-SA ensures that over 97% of tasks are completed before the deadline. Rabiu *et al*. [22] examined difficulties with auto-scaling and load balancing for Cloud-based Container Microservices (CBCM) to enhance user Quality of Service (QoS). Implementing hybrid features will be part of future efforts. Smimite and Afdel [23] examined how workload consolidation and virtualization may optimize energy in data centers. For QoS, a hybrid strategy that combines the FFD and ACO algorithms for container placement is suggested. Li *et al*. [24] enhanced the ant colony algorithm for optimal container migration in its container migration strategy for load balancing in edge computing. Simulation demonstrates improved load balancing and lower migration costs. Xie and Govardhan [25] are used for job scheduling and load balancing with Micro Service—deep Learning and IoT benefit from cloud and edge computing advances. A technique for flexible Deep Learning deployment using Kubernetes and Istio is presented in this article.

Zhao *et al*. [26] improved containerized cloud services' load balance and application performance. A novel model that uses heuristic algorithms and statistical techniques is put forth to maximize both variables. Quantifying input/output needs, streamlining network traffic, and integrating with GPFS for broader use are potential prospects. Hanafy *et al*. [27], the proposed algorithm load balances the same load across fewer hosts with less power and inevitably outperforms other competitive algorithms. Patra *et al*. [28] Optimized performance requires load balancing, which divides work across servers. Load balancing in containerized clouds is suggested using a randomized Balls into Bins technique. Tasks are represented as balls and servers as bins using a linked graph to minimize network overhead. Thongthavorn and Rattanatamrong [29] provided a middleware architecture that uses parallel TCP connections to enable concurrent multi-container migration in WANs. A feedback controller calculates the ideal number of parallel migrations based on network bandwidth. The scheduler balances migration requests for the best overall and individual migration times. The overall migration time over sequential migration in static and dynamic networks was shown to be improved by 32.7% and 43.9%, respectively. Naik *et al*. [30] provided effective resource management and simple deployment. This paper compares the Honeybee Mating Algorithm (HBMA) with GWO, GA, and PSO to provide load balancing in containerized clouds. Both load variance and makespan measurements were excellent for HBMA. The literature shows a need to leverage container scheduling with more optimized approaches.

3. Preliminaries

This section presents preliminaries about reinforcement learning and its related aspects. We examine the typical context for reinforcement learning, in which an agent engages with an environment E across several distinct time intervals.

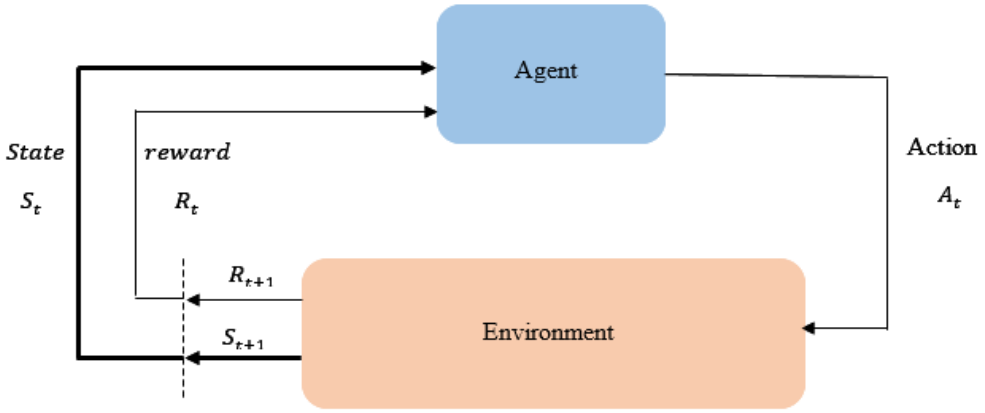

Figure 1. Illustrates generic approach of reinforcement learning

The agent gets a state s_t at each time step t and based on its policy π, which maps states s_t to actions a_t chooses an actiona_t from a collection of potential actions A. As per its policy π, a mapping from states s_tto actionsa_t A. The agent is rewarded with a scalar reward r_t and the next state s_(t+1)in return. The procedure continues until the agent enters a condition known as terminal, at which point it resumes. The outcome $R_t = \sum_{k=0}^{\infty} \Upsilon^k r_{t+k}$ is the entire return that has accumulated since time step t, discounted by a factor $\gamma \in (0,1]$ maximizing the expected return from each state s_t the agent's objective. Figure 1 shows a generic approach toreinforcementlearning.

The worth of the action $Q^{\pi}(s,a) = \mathbb{E}[R_t|s_t = s,a]$ is the anticipated profit for choosing a course of action a in state s and adhering to policy π. The ideal function for values $Q^*(s,a) = max_{\pi} Q^{\pi}(s,a)$ provides the highest possible action value for a state and the highest action that any policy can achieve. In the same way, policy π defines the value of states s as $V^{\pi}(s) = \mathbb{E}[R_t|s_t = s]$ this is just the anticipated benefit of adhering to state policy π. Value-based model-free reinforcement learning techniques use a function approximator, such as a neural network, to represent the action value function. Let Q(s, a;θ) be an action-value function approximation with θ parameters. Numerous reinforcement learning techniques can produce the changes to θ. Q-learning is a method that attempts to directly approximate the optimum action-value function. $Q*(s,a) \approx Q(s,a;\theta)$. In one-step Q-learning, a series of loss functions is iteratively minimized, with the ith loss function defined as the action value function Q(s,a;θ) parameters θ being learned.

$$L_i(\theta_i) = \mathbb{E}\left(r + \Upsilon \max_{a} Q(s^{'}, a^{'}; \theta_{i-1}) - Q(s,a;\theta_i)\right)^2 \quad (1)$$

where $s^{'}$ is the state encountered after state s. Because it changes the action value Q(s,a) toward the onestep return, we refer to the aforementioned technique as one-step Q-learning $r + \gamma\, max_{a^{'}}\, Q(s^{'}, a^{'}; \theta)$. The value of the state action pair s,a that produced the reward is the sole thing that is directly impacted when one uses one-step procedures to acquire a reward r. The changed value Q(s,a) only indirectly impacts the values of the other state-action pairs. Due to the numerous changes needed to propagate a reward to the pertinent previous states and actions, this may cause the learning process to lag.

Using nstep returns is one method of expediting the propagation of incentives. The n-step Q-learning updates Q(s,a) in the direction of the n-step return, which is defined as $r_t + \gamma r_{t+1} + \cdots + \gamma^{n-1} r_{t+n-1} + max_a \gamma^n Q(s_{t+n}, a)$. As a result, the values of n previous state-action pairings are directly impacted by a single reward (r). This possibly greatly increases the efficiency of spreading rewards to pertinent state-action pairings. Model-free policies are directly parameterized by policy-based approaches, as opposed to value-based approaches $\pi(a|s;\theta)$ and execute, usually approximatively, gradient ascent on E[Rt] to update the parameters θ. The REINFORCE family of algorithms is one instance of such a technique. The policy parameters θ are updated using Standard REINFORCE in the direction $\nabla_{\theta}\, log\, \pi(a_t|s_t;\theta) R_t$, which is a fair assessment of $\nabla_{\theta} \mathbb{E}[R_t]$. By deducting a learning function of the state, this estimate's variance may be decreased without compromising its objectivity $b_t(s_t)$, called a baseline from the return. The gradient that appears as a result is $\nabla_{\theta}\, log\, \pi(a_t|s_t;\theta)(R_t - b_t(s_t))$.

Typically, a baseline of the value function is a learned approximation $b_t(s_t) \approx V^{\pi}(s_t)$ resulting in a significantly reduced policy gradient variance estimate. When the baseline is an approximatively value function, the quantity$R_t - b_t$can be interpreted as an estimate of the benefit of actiona_t in state s_t, when used to scale the policy gradient or $A(a_t, s_t) = Q(a_t, s_t) - V(s_t)$, since R_t is a calculation that $Q^{\pi}(a_t, s_t)$ and bt is a calculation of$V^{\pi}(s_t)$.According to Degris *et al*. [31], this method may be thought of as an actor-critic architecture in which the policy π is the actor and the baseline b_t is the critic.

4. Proposed Framework

Considering a runtime dynamic environment with diversified container workloads and resources, we are developing a deep learning framework to efficiently schedule containers, leading to resource optimization and load balancing. This research considers the problem of resource optimization and load balancing. We proposed a Deep Reinforcement Learning (DRL) based framework, shown in Figure 2, for container scheduling. The multi-threaded asynchronous versions of advantage actor-critic, one-step Q-learning, one-step Sarsa, and n-step Q-learning are used. Finding RL algorithms capable of consistently training deep neural network rules with low resource requirements was the goal in constructing these approaches. With actor-critic being an on-policy policy search approach and Q-learning being an off-policy value-based method, the underlying RL methods are very different, but we leverage two major principles to make all four algorithms feasible given our design aim. Initially, we employ asynchronous actor-learners like the Gorila framework [32] however, we employ several CPU threads on a single system in place of separate machines and a parameter server. We can utilize the style updates as in [33] for training and avoid the communication expenses of delivering gradients and parameters by keeping the learners on a single computer.

Secondly, concurrently operating numerous actor learners are probably investigating distinct regions of the surroundings. To further optimize this variety, distinct exploration rules might be used explicitly in each actor-learner. The aggregate changes to the parameters produced by several actor-learners applying online updates in parallel are probably less correlated in time than a single agent applying online updates since they execute various exploration procedures in separate threads. Therefore, we don't use experience replay in the DQN training method to perform the stabilizing function; instead, we rely on parallel actors using various exploration rules.

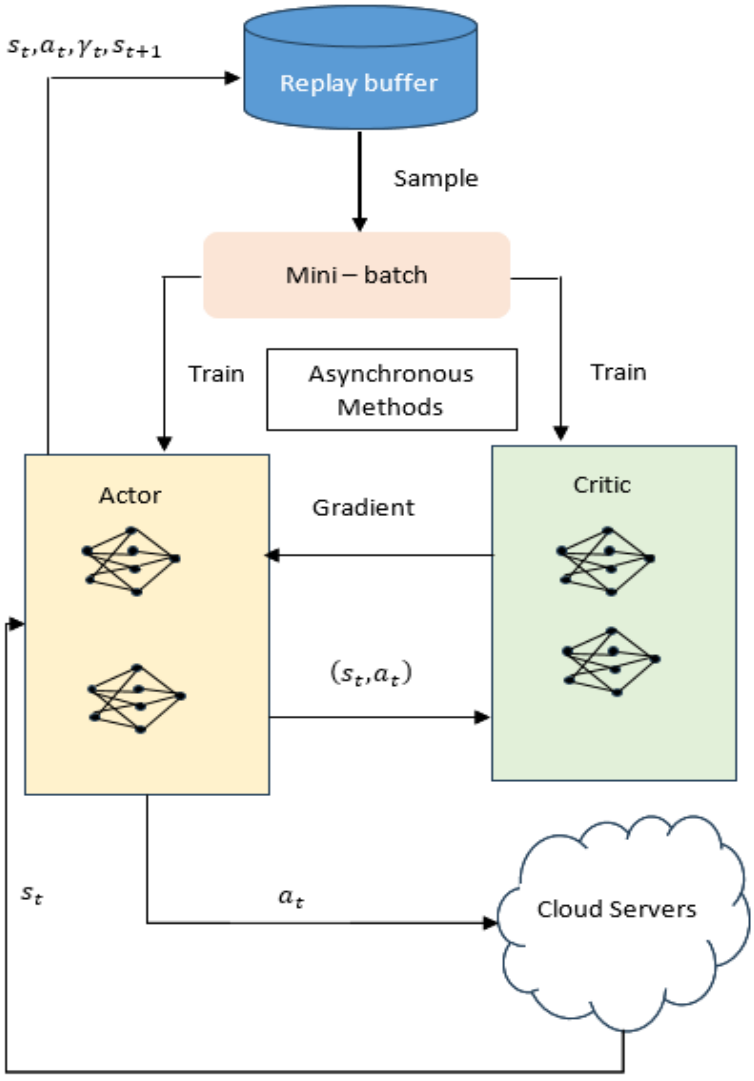

Figure 2. Proposed DRL-based scheduling framework

Using many concurrent actor-learners has several practical benefits beyond stabilizing learning. Initially, we get a training time reduction proportional to the number of concurrent actor-learners. Second, we can now stably train neural networks using on-policy reinforcement learning techniques like Sarsa and actor-critic, as we no longer depend on experience replay to stabilize learning. We now review our variations on advantage actor-critic, n-step Q-learning, one-step Sarsa, and one-step Q-learning. Algorithm 1 displays the pseudocode for our version of Q-learning, which we refer to as Asynchronous one-step Q-learning. Every thread engages with a separate instance of the environment and calculates the gradient of the Q-learning loss at every iteration. As the DQN training approach suggests, we compute the Q-learning loss using a shared and slowly evolving target network. Similarly to mini-batches, we aggregate gradients across several steps before application.

This lessens the possibility that different actor learners may overwrite one another's updates. There is also some opportunity to trade data efficiency for computational efficiency when updates are accumulated across several stages. Lastly, we discovered that varying the exploration policy for each thread enhances robustness. Incorporating diversity into exploration also leads to better exploration, improving performance. We experiment with ϵ-greedy exploration, where each thread periodically samples ρ from a distribution. There are various methods to make the exploration strategies different.

Except using a different goal value for Q(s,a), the asynchronous one-step Sarsa method is identical to the asynchronous one-step Q-learning technique provided in method 1. The one-step Sarsa's goal value is $r + \gamma Q(s', a'; \theta^-)$ where a is the action performed in state (a'). To stabilize learning, we once more employ a target network and updates gathered over several time steps. The approach functions in the forward perspective, directly computing n step returns, which sets it apart from the more popular backward view of eligibility traces. We discovered that while training neural networks utilizing momentum-based techniques and backpropagation across time, employing the forward perspective is more straightforward. To compute an individual update, the algorithm initially chooses actions based on its exploration policy for a maximum of t_{max} until a final condition is attained, in phases. The agent has received up to, as a consequence of this process, processt_{max} incentives from the surroundings since the last update. Next, for every state-action pair encountered since the last update, the technique computes gradients for n-step Q-learning updates. For a total of up to t_{max} updates, each n-step update utilizes the longest feasible n-step return, producing a one-step update for the previous state, a two-step update for the next-to-last state, and so on. One gradient step is applied to the cumulative updates. Our method, named asynchronous advantage actor-critic (A3C), keeps track of a policy$\pi(a_t|s_t;\theta)$as well as a value function estimate $V(s_t;\theta_v)$. Our version of actor-critic, like our version of n-step Q-learning, is forward-looking and updates the policy and the vvalue function with the same mixture of n-step returns. Following each update, the value function and the policy are changed $t_{\max}$ activities or upon reaching a final condition. An interpretation of the algorithm's update would be$\nabla_{\theta'} log\ \pi(a_t|s_t;\theta')A(s_t,a_t;\theta,\theta_v)$ where $A(s_t,a_t;\theta,\theta_v)$a rough approximation of the advantage function provided by $\sum_{i=0}^{k-1}\gamma^i r_{t+i} + \gamma^k V(s_{t+k};\theta_v) - V(s_t;\theta_v)$, as $t_{\max}$ sets an upper restriction on k, which might change from one state to the next. Algorithm S3 Supplementary provides the pseudocode for the algorithm. Similar to value-based approaches, we enhance training stability by utilizing cumulative updates and parallel actor learners. It should be noted that in actuality, we always share some of the parameters, even if the parameters θ of the policy and θ_v of the value function's parameters are depicted as independent polarity. Typically, we utilize a convolutional neural network with shared non-output layers that has one linear output for the value function$V(s_t;\theta_v)$, and one softmax output for the policy$\pi(a_t|s_t;\theta)$.

Furthermore, we discovered that preventing early convergence to inferior deterministic policies, including the policy's entropy (π) in the goal function enhanced exploration. When Williams and Peng first presented this strategy, they discovered it was handy for jobs requiring hierarchical behavior. Concerning the policy parameters, the gradient of the entire objective function, which includes the entropy regularization component, has the following form $\nabla_{\theta'} log\ \pi(a_t|s_t;\theta')(R_t - V(s_t;\theta_v)) + \beta\nabla_{\theta'}H(\pi(s_t;\theta'))$, H represents the entropy. The hyperparameter β controls the intensity of the entropy regularization term. We examined three distinct optimization algorithms within our asynchronous framework: SGD with momentum, RMSProp [34] without shared statistics, and RMSProp with shared statistics. The typical non-centered RMSProp update provided by

$$g = \alpha g + (1-\alpha)\Delta\theta^2\ and\ \theta \leftarrow \theta - \eta\frac{\Delta\theta}{\sqrt{g+\epsilon}} \quad (2)$$

Where each procedure is carried out piecemeal. A comparison of the two approaches using a subset of Atari 2600 games revealed that the variation of RMSProp where statistics g is shared across threads is significantly more reliable. We proposed an algorithm (Algorithm 1) known as Reinforcement Learning Container Scheduling (RLbCS), which exploits the runtime environment with an action-reward iterative approach for optimal scheduling of containers.

Algorithm 1. Reinforcement Learning based Container Scheduling (RLbCS)

Algorithm: Reinforcement Learning based Container Scheduling (RLbCS)
Input: Set of containers D
Output: Optimal scheduling of containers

1. t←0 (step counter of thread)
2. $\theta' \leftarrow \theta$ (network weights)
3. $d\theta \leftarrow 0$ (network gradient)
4. s←obtainInitialState()
5. while $T \leq T_{max}$
6. using ϵ-greedy policy, take action considering $Q(s, a; \theta)$
7. obtain new state s'
8. obtain reward r
9. gradient accumulation
10. s←s'
11. T←T+1
12. $t \leftarrow t$-1
13. If $T \ mod \ I_{target}$ is zero Then
14. Update θ'
15. End If
16. If s is terminal or $t \ mod \ I_{asncu}$ is zero Then
17. Use $d\theta$ and update θ
18. $d\theta \leftarrow 0$
19. End If
20. End While

The algorithm aims to achieve optimal scheduling of containers, represented as the set D. It utilizes a step counter t0 for thread control, network weights, and a network gradient to adjust these weights. The algorithm operates iteratively, starting with an initial state obtained through the obtain Initial State () function. It employs a -greedy policy to select actions, which balances exploration and exploitation in the learning process. At each step, the algorithm takes an action, observes the new state, and receives a reward, which provides feedback on the quality of the action taken. The network gradient is accumulated to update the network weights, which are adjusted to optimize the scheduling policy.

The process continues until either the network gradient is zero, indicating a potential convergence to an optimal policy, or a terminal state is reached. When the gradient is zero, the network weights are updated using the accumulated gradient. If the state is terminal or the gradient is zero, the accumulated gradient is used to update the network weights and the gradient is reset to zero for the next iteration. The algorithm's structure suggests a temporal difference learning approach, where the agent learns from the difference between the estimated value of the current state and the actual value received after taking an action. This method is commonly used in reinforcement learning to improve the policy by reducing the difference between the estimated and actual values. In summary, the RLbCS algorithm is a reinforcement learning method designed to schedule containers optimally by learning from the outcomes of its actions. It uses a -greedy policy for action selection and temporal difference learning to update its policy. The algorithm's process is controlled by a step counter and involves iterative updates to network weights based on gradient accumulation.

Each thread is associated with its environment in a multi-threading context, and the gradient linked to a loss in Q-learning is considered in an asynchronous approach. It is an iterative process based on DQN methodology. Gradients accumulate over many steps in mini-batches. This procedure ensures that there is no chance of overwriting updates of each other in the presence of a multi-threading-based learning process. It also ensures that there is a trade-off between efficiency and computational overhead. Each thread has its exploration polity to leverage robustness in the execution process. This also provides diversity in the exploration capability of threads, leading to better exploration and efficiency. We found that ϵ -greedy policy-based exploration is better than another kind of exploration. Section 4 presents our empirical results.

5. Experimental Results

Experiments are done with a prototype application with different scenarios for which real-time data traces are used to evaluate the performance and convergence dynamics of the proposed model and existing models such as DDQN [35], A2C, PPO [36], and DQN [37].

A2C is also an actor-critic-based model similar to the proposed one but lacks a parallel approach. Evaluation is made under two scenarios: static (container placed in a server and remains in the same server) and dynamic (container placed in a server may be moved to another server later). Table 1 shows the results regarding mean episode reward against iterations in the static scenario. Experiments are made with traces involving 100 physical machines of the same capacity. In a static scenario, 250 containers are used for scheduling each episode, while 500 containers are used in a dynamic scenario.

As the number of iterations increased, there was a difference between mean episode rewards, and each method showed a different mean episode reward. Figure 3 shows the mean episode reward against iterations exhibited by each model in the static scenario.

Convergence is reflected in each method used in an empirical study in the static scenario. As the number of iterations increases, there is a gradual but relatively less increase in the mean episode reward. A2C and the proposed models showed better performance over other existing methods.

As the number of iterations increased, there was a difference between mean episode rewards, and each method showed a different mean episode reward.

Table 1. Mean episode reward comparison in a static scenario

Iterations	Mean Episode Reward				
	Proposed	A2C	DQN	DDQN	PPO
0	-4725	-5500	-4125	-4700	-5000
10	-3100	-3100	-3900	-3700	-3750
20	-3150	-3150	-3750	-3500	-3900
30	-3050	-3050	-3500	-3500	-3800
40	-3050	-3050	-3400	-3500	-3900
50	-3020	-3100	-3300	-3550	-3700

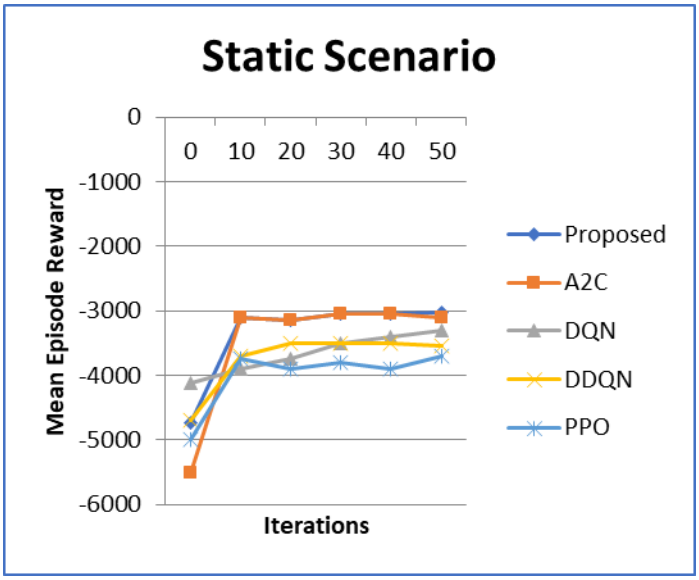

Figure 3. Mean episode reward comparison for all models in a static scenario

Table 2. Mean episode reward comparison in the dynamic scenario

Iterations	Mean Episode Reward				
	Proposed	A2C	DQN	DDQN	PPO
0	-25500	-26000	-26000	-26000	-21000
20	-15300	-15500	-20500	-22000	-19500
40	-15200	-15500	-19500	-20500	-19100
60			-19000	-20000	-18500
80			-18100	-18300	-18050
100			-18050	-18050	-17900

Table 3. Resource wastage comparison for all methods in a static scenario

Scheduling Models	Resource Wastage
Proposed	0.13
A2C	0.14
DQN	1.32
DDQN	1.35
PPO	0.16

Table 4. Resource wastage comparison for all methods in the dynamic scenario

Scheduling Models	Resource Wastage
Proposed	0.16
A2C	0.17
DQN	1.34
DDQN	1.36
PPO	0.19

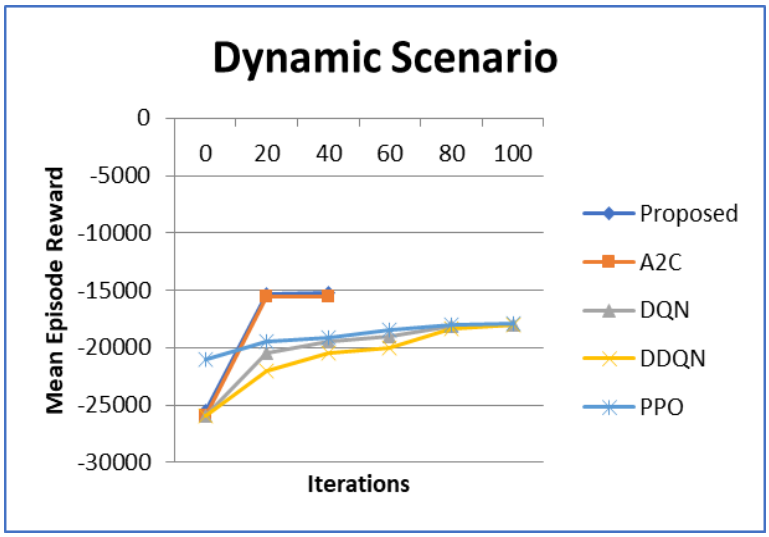

Figure 4. Mean episode reward comparison for all models in a dynamic scenario

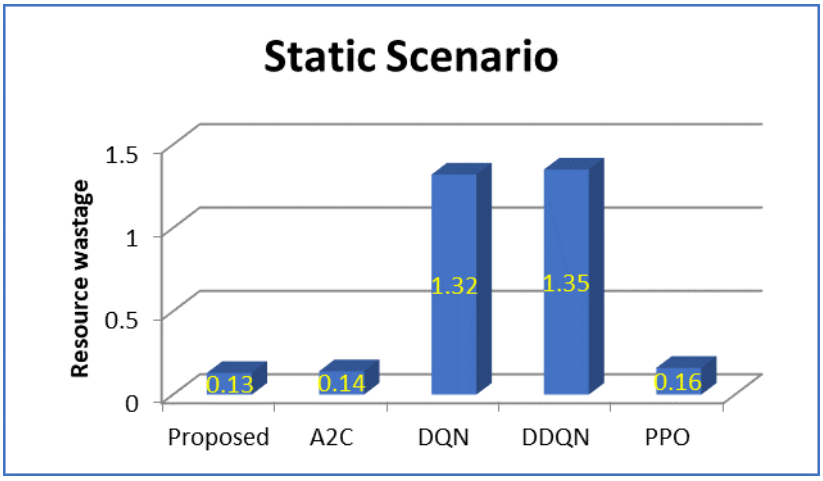

Figure 5. Resource wastage comparison visualization for all methods in a static scenario

Table 5. Performance in terms of the percentage of placed requests in a static scenario

Scheduling Models	% Placed requests
Proposed	0.88
A2C	0.87
DQN	0.35
DDQN	0.30
PPO	0.85

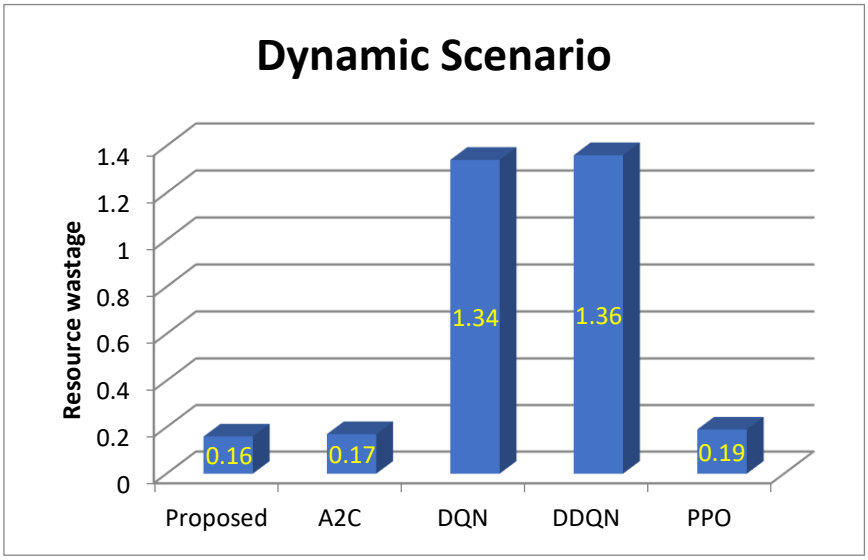

Figure 6. Resource wastage comparison visualization for all methods in the dynamic scenario

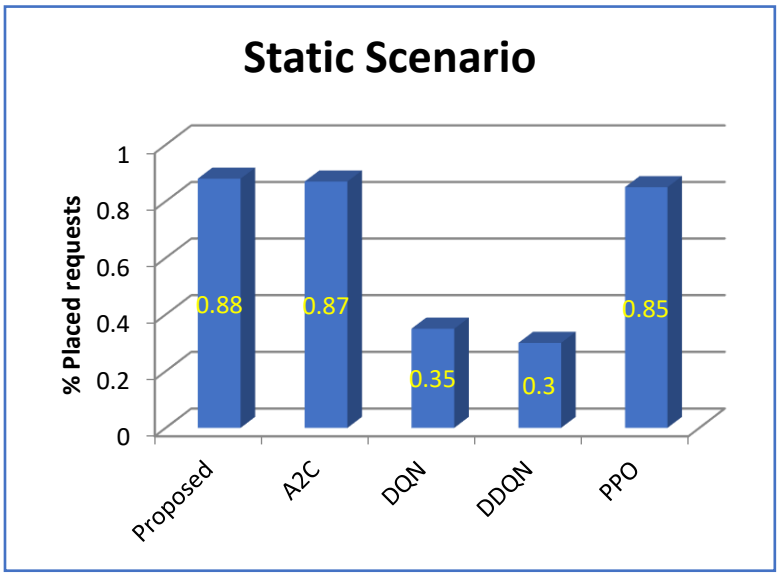

Figure 7. Performance in terms of percentage of placed requests in static scenario visualized

Table 6. Performance in terms of the percentage of placed requests in a dynamic scenario

Scheduling Models	**% Placed requests**
Proposed	0.92
A2C	0.91
DQN	0.41
DDQN	0.36
PPO	0.89

Table 7. Performance of the proposed method with AS and without AS approaches in terms of the number of requests placed in an episode

Scheduling Models	**Placed requests per episode**
Proposed-Without AS	10
Proposed-With AS	174

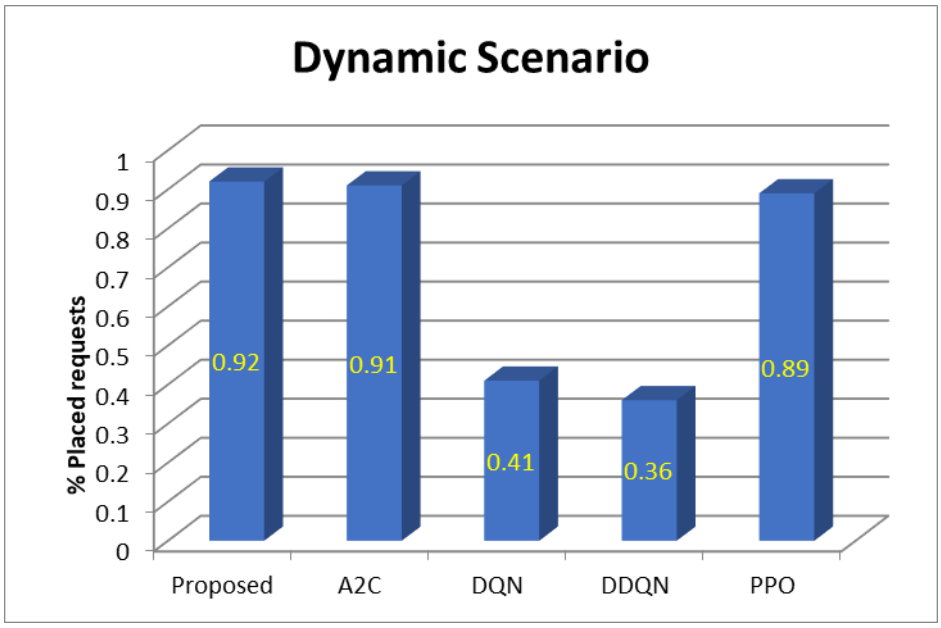

Figure 8. Performance in terms of percentage of placed requests in dynamic scenario visualized

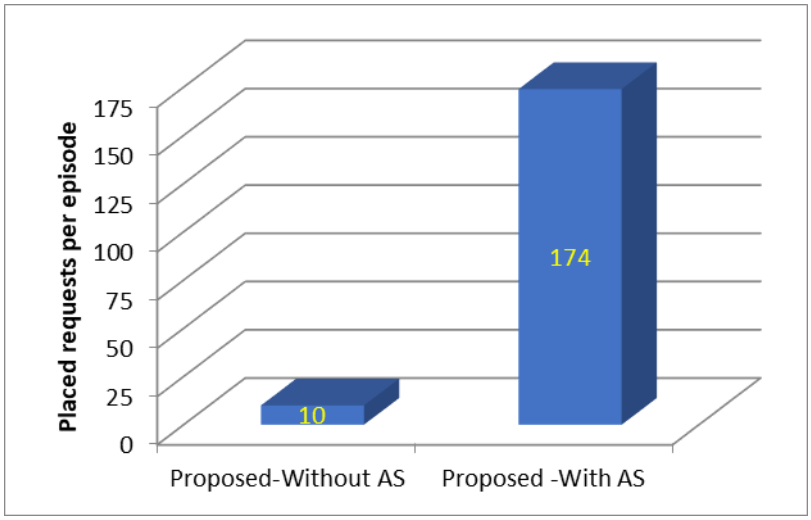

Figure 9. Visualization of performance of the proposed method with AS and without AS approaches in terms of the number of requests placed in an episode visualized

Figure 4 shows each model's mean episode reward against iterations in a dynamic scenario.

Each method used in an empirical study in a dynamic scenario reflects convergence. As the number of iterations increases, the mean episode reward gradually but relatively less increases. A2C and the proposed models showed better performance over other existing methods. However, the proposed method showed the highest convergence at iteration 40. Table 3 shows a comparison of resource wastage for all methods.

In a static scenario, resource wastage exhibited by DDQN is highest, at 1.35. DQN shows 1.32, less than DDQN but higher than all other methods. PPO is much better than DQN and DDQN, with 1.32 resource wastage. A2C and the proposed model, based on the actor-critic approach, showed the highest performance in reducing wastage. However, the proposed method outperforms its A2C counterpart with the least resource wastage of 0.13. Table 4 shows resource wastage comparison for all methods in a dynamic scenario.

Resource wastage is relatively low for A2C in dynamic scenarios, and the proposed method is based on both the actor and critic approach. Figure 5 shows a graphical analysis of the processes' resource wastage dynamics.

In a dynamic scenario, DDQN's resource wastage is highest at 1.36. DQN shows 1.34, less than DDQN but higher than all other methods. PPO is much better than DQN and DDQN, with 1.19 resource

wastage. Based on the actor-critic approach, A2C and the proposed model showed the highest performance in reducing wastage. However, the proposed method outperforms its A2C counterpart with the least resource wastage of 0.16. Table 5 shows the performance of methods regarding the percentage of placed requests.

Knowing the percentage of container placement requests completed by different methods is essential. Figure 7 visualizes the methods' performance in this regard in a static scenario.

The percentage of placed container requests is used to measure the performance of scheduling methods. A higher percentage indicates better performance. The DDQN method exhibited the most minor performance in a static scenario with 30%. DQN showed better performance than DDQN with 35%. The PPO method outperformed both DQN and DDQN models with 85% of placed container requests. The actor-critic-based models, such as A2C and the proposed ones, outperformed all other methods. However, the proposed model is only slightly better than its A2C counterpart. Table 6 provides the performance regarding the percentage of placed requests in a dynamic scenario.

It is important to know the percentage of container placement requests successfully completed by different methods. Figure 8 visualizes the methods' performance in a dynamic scenario.

The percentage of placed container requests is used to measure the performance of scheduling methods. A higher percentage indicates better performance. DDQN exhibits the lowest performance at 36% in dynamic scenarios, while DQN performs better at 41%. PPO outperforms DQN and DDQN models, with 89% of container requests placed. Actor-critic-based models outperform all other methods, such as A2C and the proposed ones. The proposed model is slightly better than its A2C counterpart. Table 7 shows the performance of the proposed method with AS and without AS approaches in terms of the number of requests placed in an episode.

The results show that the asynchronous approach impacts container scheduling, as visualized in Figure 9.

The results show that the asynchronous approach significantly impacts container scheduling. When the proposed method is used with AS, optimal scheduling allows for processing 174 container requests per episode. Only ten requests were successfully processed per episode when the proposed method is used without AS. This demonstrates that the asynchronous approach has a noticeable impact on scheduling performance.

6. Discussion

The actor-critic-based framework proposed is found to be superior to existing DRL-based methods for container scheduling. Our approach has several features that contribute to its performance. These include the actor-critic-based approach, asynchronous methodology with multiple threads, and a parallel approach. As each thread can have its copy of the environment, there is potential for higher scalability. The workloads are evaluated using both static and dynamic scenarios. The proposed framework is evaluated regarding its ability to converge and performance in scheduling containers in static and dynamic contexts. It is simulated based on the real traces collected from cloud infrastructure. The proposed model has been found to perform better than existing methods.

6.1 Limitations

The proposed system is based on real cloud traces used for both static and dynamic scenarios. It could be tested in a real live cloud environment in the future. The workloads are limited, and the framework's performance cannot be generalized unless there are diversified workloads. Therefore, there is a threat to validity. Moreover, the proposed system is limited to a specific cloud environment and does not work for a federated cloud scenario.

7. Conclusion and Future Work

The paper introduces a framework for improving container scheduling and load balancing using deep reinforcement learning. Our approach is based on an actor-critic method that enables enhanced learning. In addition, our methodology is asynchronous and supports multi-threading. Each thread is associated with its environment, and the gradient linked to the loss in Q-learning is considered asynchronous. It is an iterative process based on the DQN methodology. Gradients accumulate over many steps in mini-batches. This procedure ensures that updates from each other are not overwritten in a multi-threading-based learning process. It also ensures a balance between efficiency and computational overhead. The algorithm, called Reinforcement Learning based Container Scheduling (RLbCS), utilizes the runtime environment with an action-reward iterative approach for optimal scheduling of containers. The experimental results showed that RLbCS is effective in resource optimization and the convergence of the scheduling process. RLbCS outperformed other existing methods, with 92% of placed containers. In the future, we plan to enhance our framework by extending the reward function to consider latency in a federated cloud environment. Another direction for future research is to leverage reinforcement learning with an improved agent critic methodology.

Optimizing the underlying deep learning method can further extend the proposed framework.

References

[1] S. Swarup, E.M. Shakshuki, A. Yasar, Task scheduling in cloud using deep reinforcement learning. Procedia Computer Science, 184, (2021) 42-51. https://doi.org/10.1016/j.procs.2021.03.016

[2] L. Wang, Q. Weng, W. Wang, C. Chen, B. Li, (2020) Metis: Learning to schedule long-running applications in shared container clusters at scale. In SC20: International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, USA. https://doi.org/10.1109/SC41405.2020.00072

[3] F. Rossi, M. Nardelli, V. Cardellini, (2019) Horizontal and vertical scaling of container-based applications using reinforcement learning. In 2019 IEEE 12th International Conference on Cloud Computing (CLOUD), IEEE, Italy. https://doi.org/10.1109/CLOUD.2019.00061

[4] A. Singh, G.S. Aujla, R.S. Bali, Container-based load balancing for energy efficiency in software-defined edge computing environment. Sustainable Computing: Informatics and Systems, 30, (2021) 100463. https://doi.org/10.1016/j.suscom.2020.100463

[5] I. Ahmad, M.G. AlFailakawi, A. AlMutawa, L. Alsalman, Container scheduling techniques: A survey and assessment. Journal of King Saud University-Computer and Information Sciences, 34(7), (2022) 3934-3947. https://doi.org/10.1016/j.jksuci.2021.03.002

[6] S. Swarup, E.M. Shakshuki, A.Yasar, Energy efficient task scheduling in fog environment using deep reinforcement learning approach. Procedia Computer Science, 191, (2021) 65-75. https://doi.org/10.1016/j.procs.2021.07.012

[7] L. Deng, Z. Wang, H. Sun, B. Li, X. Yang, A deep reinforcement learning-based optimization method for long-running applications container deployment. International Journal of Computers Communications & Control, 18(4), (2023) 1-17. https://doi.org/10.15837/ijccc.2023.4.5013

[8] Z. Wang, M. Goudarzi, M. Gong, R. Buyya, Deep reinforcement learning-based scheduling for optimizing system load and response time in edge and fog computing environments. Future Generation Computer Systems, 152, (2024) 55-69. https://doi.org/10.1016/j.future.2023.10.012

[9] G.P. Mattia, R. Beraldi, P2PFaaS: A framework for FaaS peer-to-peer scheduling and load balancing in Fog and Edge computing. SoftwareX, 21, (2023) 101290. https://doi.org/10.1016/j.softx.2022.101290

[10] L. Wang, S. Guo, P. Zhang, H. Yue, Y. Li, C. Wang, Z. Cao D. Cui, An Efficient Load Prediction-Driven Scheduling Strategy Model in Container Cloud. International Journal of Intelligent Systems, 2023(1), (2023) 5959223. https://doi.org/10.1155/2023/5959223

[11] Y. Li, X. Guo, Z. Meng, J. Qin, X. Li, X. Ma, J. Yang, (A Hierarchical Resource Scheduling Method for Satellite Control System Based on Deep Reinforcement Learning. Electronics, 12(19), (2023) 3991. https://doi.org/10.3390/electronics12193991

[12] S. Luo, Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning. Applied Soft Computing, 91, (2020) 106208. https://doi.org/10.1016/j.asoc.2020.106208

[13] O. Oleghe, Container placement and migration in edge computing: Concept and scheduling models. IEEE Access, 9, (2021) 68028-68043. https://doi.org/10.1109/ACCESS.2021.3077550

[14] J. Zhang, T. Wang, L. Cheng, Time-Sensitive and Resource-Aware Concurrent Workflow Scheduling for Edge Computing Platforms Based on Deep Reinforcement Learning. Applied Sciences, 13(19), (2023) 10689. https://doi.org/10.3390/app131910689

[15] Z. Ma, S. Shao, S. Guo, Z. Wang, F. Qi, A. Xiong, Container migration mechanism for load balancing in edge network under power Internet of Things. IEEE Access, 8, (2020) 118405-118416. https://doi.org/10.1109/ACCESS.2020.3004615

[16] A.G. Ramos, E. Silva, J.F. Oliveira, A new load balance methodology for container loading problem in road transportation. European Journal of Operational Research, 266(3), (2018) 1140-1152. https://doi.org/10.1016/j.ejor.2017.10.050

[17] D. Baburao, T. Pavankumar, C.S.R. Prabhu, (2019) Survey on service migration, load optimization and load balancing in fog computing environment. IEEE 5th International Conference for Convergence in Technology (I2CT), IEEE, India. https://doi.org/10.1109/I2CT45611.2019.9033579

[18] A. Talukder, S.F. Abedin, M.S. Munir, C.S. Hong, (2018) Dual threshold load balancing in SDN environment using process migration. In 2018 International Conference on Information Networking (ICOIN), IEEE, Thailand. https://doi.org/10.1109/ICOIN.2018.8343226

[19] K. Kaur, S. Garg, G. Kaddoum, F. Gagnon, D.N.K. Jayakody, (2019) Enlob: Energy and load balancing-driven container placement strategy for data centers. IEEE Globecom Workshops (GC Wkshps), IEEE, USA. https://doi.org/10.1109/GCWkshps45667.2019.9024592

[20] K. Aruna, G. Pradeep, Ant Colony Optimization-based Light Weight Container (ACO-LWC) Algorithm for Efficient Load Balancing. Intelligent Automation & Soft Computing, 34(1), (2022) 1-15. https://doi.org/10.32604/iasc.2022.024317

[21] M.K. Patra, S. Misra, B. Sahoo, A.K. Turuk, GWO-based simulated annealing approach for load balancing in cloud for hosting container as a service. Applied Sciences, 12(21), (2022) 11115. https://doi.org/10.3390/app122111115

[22] S. Rabiu, C.H. Yong, S.M.S. Mohamad, A cloud-based container micro services: a review on load-balancing and auto-scaling issues. International Journal of Data Science, 3(2), (2022) 80-92. https://doi.org/10.18517/ijods.3.2.80-92.2022

[23] O. Smimite, K. Afdel, Hybrid solution for container placement and load balancing based on aco and bin packing. International Journal of Advanced Computer Science and Applications, 11(11), (2020) 1-10. https://doi.org/10.14569/IJACSA.2020.0111174

[24] K. Li, C. Chang, K. Yun, J. Zhang, (2021) Research on container migration mechanism of power edge computing on load balancing. IEEE 6th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA), IEEE, China. https://doi.org/10.1109/ICCCBDA51879.2021.9442546

[25] X.I.E. Xiaojing, S.S. Govardhan, (2020) A service mesh-based load balancing and task scheduling system for deep learning applications. 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID), IEEE, Australia. https://doi.org/10.1109/CCGrid49817.2020.00009

[26] D. Zhao, M. Mohamed, H. Ludwig, Locality-aware scheduling for containers in cloud computing. IEEE Transactions on cloud computing, 8(2), (2018) 635-646. https://doi.org/10.1109/TCC.2018.2794344

[27] W.A. Hanafy, A.E. Mohamed, S.A. Salem, (2017) A load balancing with power optimization algorithm for container-based infrastructure management. 12th International Conference on Computer Engineering and Systems (ICCES), IEEE, Egypt. https://doi.org/10.1109/ICCES.2017.8275296

[28] M.K. Patra, D. Patel, B. Sahoo, A.K. Turuk, (2020) A randomized algorithm for load balancing in containerized cloud. In 2020 10th International conference on cloud computing, data science & engineering (confluence), IEEE, India. https://doi.org/10.1109/Confluence47617.2020.9058147

[29] W. Thongthavorn, P. Rattanatamrong, (2019) Multi-container application migration with load balanced and adaptive parallel TCP. International Conference on High Performance Computing & Simulation (HPCS), IEEE, Ireland. https://doi.org/10.1109/HPCS48598.2019.9188218

[30] K.D. Naik, R.R. Sahoo, S.K. Kuana, A Bio inspired Approach for Load Balancing in Container as a Service Cloud Computing Model. International Research Journal on Advanced Science Hub, 5(05), (2023) 426-433. https://doi.org/10.47392/irjash.2023.S058

[31] T. Degris, P.M. Pilarski, R.S. Sutton, (2012) Model-free reinforcement learning with continuous action in practice. In 2012 American control conference (ACC), IEEE, Canada. https://doi.org/10.1109/ACC.2012.6315022

[32] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. De Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen, S. Legg, V. Mnih, K. Kavukcuoglu, D. Silver, (2015) Massively parallel methods for deep reinforcement learning. arXiv. https://doi.org/10.48550/arXiv.1507.04296

[33] B. Recht, C. Re, S. Wright, F. Niu, Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. Advances in neural information processing systems, 24, (2011) 693–701.

[34] T. Tieleman, Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning, 4(2), (2012) 26.

[35] Q. Zhang, M. Lin, L.T. Yang, Z. Chen, S.U. Khan, P. Li, A double deep Q-learning model for energy-efficient edge scheduling. IEEE Transactions on Services Computing, 12(5), (2018) 739-749. https://doi.org/10.1109/TSC.2018.2867482

[36] W. Funika, P. Koperek, J. Kitowski, (2020) Automatic management of cloud applications with use of proximal policy optimization. In International Conference on Computational Science, Springer International Publishing.

[37] Z. Wang, C. Gwon, T. Oates, A. Iezzi, (2017) Automated cloud provisioning on aws using deep reinforcement learning. arXiv. https://doi.org/10.48550/arXiv.1709.04305

Authors Contribution Statement

Dr. Sreedhar Bhukya: Conceptualization, Methodology, Formal analysis, Data Curation, Validation, Writing - Original Draft. P. Suraj Goud: Conceptualization, Methodology, Formal analysis, Data Curation. K. Yuvraj Goud: Validation, Writing -Original Draft, Review & Editing. K. Harin: Formal analysis, Writing Original Draft, Review & Editing. All Authors read and approved the final version of the manuscript.

Has this article screened for similarity?

Yes

Funding

The authors declare that no funds, grants or any other support were received during the preparation of this manuscript.

Competing Interests

The authors declare that there are no conflicts of interest regarding the publication of this manuscript.

Data Availability

The data supporting the findings of this study can be obtained from the corresponding author upon reasonable request.

About the License