Artificial Intelligence Research and Development
T. Alsinet et al. (Eds.)

doi:10.3233/FAIA240406

Optimizing Energy Consumption of Kubernetes Clusters with Deep Reinforcement Learning

Alejandro ESPINOSA [a,1], Daniel ULIED [a] and Josep ESCRIG [a] and Adriana FERNÁNDEZ-FERNÁNDEZ [a] and Rizkallah TOUMA [a]
[a] *i2CAT Foundation, Barcelona, Spain*
ORCiD ID: Alejandro Espinosa https://orcid.org/0009-0006-6278-2708, Daniel Ulied https://orcid.org/0009-0007-3601-8455, Josep Escrig https://orcid.org/0000-0002-0918-8148 , Adriana Fernández-Fernández https://orcid.org/0000-0003-1616-5582 , Rizkallah Touma https://orcid.org/0000-0003-0543-1275

Abstract. As the usage of the edge-cloud continuum rises, Kubernetes presents itself as a solution that allows easy control and deployment of applications in these highly-distributed and heterogeneous environments. In this context, Artificial Intelligence methods have been proposed to aid in the task allocation process to optimize different aspects of the system, such as application execution time, load balancing or energy consumption. In this paper, we propose a space-time combinational model that uses Deep Reinforcement Learning (DRL) to recommend node allocations for Kubernetes pods with the objective of optimizing the overall energy consumption of the cluster while maintaining pod execution ratio. In particular, our approach uses Proximal Policy Optimization (PPO) with custom Neural Networks to train a DRL agent and includes a custom Kubernetes operator to enforce allocations based on the node recommendations generated by the agent. Using our custom solution, we performed a series of experiments with different workloads and compared the performance with the base Kubernetes scheduler. Our experimental results demonstrate a notable reduction of up to 24% in the energy consumption of the Kubernetes cluster.

Keywords. Artificial Intelligence, Deep Reinforcement Learning, Edge-to-Cloud Continuum, Kubernetes, Dynamic Task Allocation, Energy Optimization

1. Introduction

With the focus of bringing applications closer to the user, the edge-cloud continuum aims to meet security, performance and cost requirements of emerging technologies [1]. In contrast to traditional cloud infrastructures, which largely consist of homogeneous servers, edge-cloud environments are characterised by the heterogeneity of the devices where applications are deployed, ranging from cloud servers to more resource-restricted edge devices, and the large-scale distribution needed to deploy applications [2].

[1]Corresponding Author: Alejandro Espinosa, alejandro.espinosa@i2cat.net

This heterogeneity and distribution pose new challenges on deploying applications in these environments. To address them, virtualization and containerization technologies, such as Kubernetes (K8s), have been widely used to facilitate application deployment as a set of virtual containers, called *pods*, in a standardized manner across different infrastructures and configurations [3]. Furthermore, container-based orchestration frameworks have been proposed to optimize the deployment of pods across the nodes of Kubernetes clusters, a process called dynamic task scheduling or allocation [4, 5].

According to a recent survey of dynamic task allocation methods, Reinforcement Learning (RL) compromises 10% of all surveyed approaches, being tied with swarm methods as the most common technique [6]. Most prominently, RL has been used in edge-cloud orchestration to allocate pods with the objectives of optimizing task execution times [7, 8], balancing workloads across different nodes [9, 10] or improving the overall energy consumption of the deployed applications [11, 12].

In this paper, we explore the usage of Deep Reinforcement Learning (DRL) in order to optimize the energy consumption of Kubernetes clusters. Our approach uses Proximal Policy Optimization (PPO) with custom Neural Networks to train a DRL agent and recommend nodes from the Kubernetes cluster to allocate application pods. Furthermore, our approach also considers already-running pods in the cluster, recommending reallocations to further optimize the energy consumption. Additionally, our model aims to maximize the number of allocated pods in order to maintain the pod execution ratio.

In order to test the viability of our DRL agent, we transform the node recommendations generated by our DRL agent into actionable items by implementing a custom Kubernetes operator that allocates pods to the recommended nodes. We also implemented a benchmarking tool that can generate different workloads as Kubernetes pods for the experiments. We compared our custom operator with the base Kubernetes scheduler and our experimental results show that our approach reduces energy consumption by up to 24% when compared to the Kubernetes scheduler.

The remainder of the paper is organised as follows. Background and related work are reviewed in Section 2. Our approach to the task allocation problem is formalized in Section 3. The custom Kubernetes integration we implemented is explained in Section 4. Our experimental setup and evaluation results are discussed in Section 5. Finally, conclusions and lines for future work are highlighted in Section 6.

2. Background and related Work

Deep Reinforcement Learning (DRL) uses Neural Networks to improve the policy of trained RL models in order to help with the dimensionality of problems and the dependency on environment knowledge [13]. In 2017, Schulman *et al.* proposed Proximal Policy Optimization (PPO), a family of DRL algorithms that use epochs of Stochastic Gradient Ascent in order to perform policy updates. PPO works by using the actor-critic methodology, where two Neural Networks are trained. The *actor* is trained on the current state of the system to output possible actions, while in parallel the *critic* is trained to predict the obtained reward value by using the action of the actor network and the state. The reward is then compared with the real one and the networks are updated [14].

Focusing on optimization of energy consumption, Jayanetti *et al.* propose a PPO-based RL framework with the key novelty of a hierarchical action space to distinguish

between cloud nodes and edge nodes [11]. Furthermore, Buschmann *et al.* implement a PPO-based RL method for energy consumption optimization conducting a comparative study that indicates that PPO outperforms Particle Swarm Optimization (PSO) when considering cases of over 25 tasks [15]. Most recently, Li *et al.* propose an energy-efficient task scheduling scheme based on PPO specific to connected vehicles [16].

Aside from PPO, the most common RL technique to optimize energy consumption has been Deep-Q Learning (DQL) [12, 17]. Other works use less common variants such as off-policy RL [18] or combine RL with custom logic such as heuristics-based RL [19]. For more information on the use of RL in dynamic task scheduling, we refer the reader to the survey published by Shyalika *et al.* [20].

Compared to previous approaches, in this paper we propose an initial modeling to optimize energy consumption specific to Kubernetes clusters. We guarantee the generalization of our approach to different use cases by basing our DRL model on widely-available Kubernetes metrics. The model uses PPO as a starting point as it is one of the most time-efficient DRL methods [21]. Furthermore, we implement a ready-to-use custom Kubernetes operator to allocate pods based on the generated recommendations.

3. Problem modeling and implementation

In our modeling, we address the dynamic task scheduling problem in Kubernetes. As such, in this section we use the corresponding terminology and refer to a Kubernetes cluster where N is the set of nodes in the cluster, with $|N|$ being the total number of nodes. Also, a pod p represents a task to be allocated to a node $n \in N$.

3.1. Reinforcement Learning model

The main objective of our DRL model is to find the optimal node in the Kubernetes cluster to allocate new and running pods in order to minimize the overall energy consumption of the cluster. In our modeling, we solve two sub-problems: (1) new pods that need to be allocated, and (2) running pods in the cluster that might need a reallocation to optimize their energy consumption based on the changing cluster state. We reduced both problems to the first one by considering the running pods as new ones and providing an updated recommendation on the best reallocation for them. The output of the DRL agent is a percentage of confidence for each node $n \in N$, the higher the value, the more the node is recommended to allocate pod p.

State definition

We use a continuous space to represent the observation space, a snapshot of which taken each time a pod p needs to be allocated at a given timestamp t provides a state S_t consisting of:

- C_p, M_p: the pair of values indicating the CPU and RAM requested by the current pod to allocate p, respectively.
- $C_n - C'_n(t), M_n - M'_n(t)\ \forall n \in N$: the list of available CPU and RAM of each node n in the cluster. This is computed by subtracting the consumed CPU and RAM at time t, $C'_n(t)$ and $M'_n(t)$ respectively, from the total CPU and RAM, C_n and M_n respectively, for all nodes. The CPU is represented in percentage of cores used whereas the RAM is represented in GB.

Action space

The action space of our model is a discrete space where values range from 0 to $|N|$, with a total of $|N|+1$ possible actions. Each action represents a possible node allocation, and the additional action represents a fake node that implies that no action will be taken. This fake node is used as a waiting mechanism for scenarios where it is not possible to perform a legal action and the algorithm needs to wait for the cluster to free up resources before allocating pod p. Thus, the action space is defined as $A = \{0, 1, ..., |N|\}$.

Reward

We modeled the reward of the DRL agent with the objective of minimizing the total energy consumption of the cluster while at the same time encouraging the agent to allocate pods to be executed. As such, the reward function r is composed of 5 components.

First, we use a custom metric to optimize the energy in the system by estimating the energy that will be consumed by pod p if allocated to a node n. To do so, we multiply the requested resources of the pod, C_p, M_p, by a power efficiency coefficient, CC_n, MC_n, assigned to CPU and RAM for node n . The power coefficients have a value between 0 and 1 and represent the efficiency of the node's CPU and RAM, given that some hardware consume more energy than others. In this case, we assume that the bigger the node capacity and components, the more it consumes. The values are then normalized between 0 and 1 and multiplied by -1 in order to minimize the metric as seen in Eq. 1.

$$e(p,n) = -1 * (norm(C_p * CC_n) + norm(M_p * MC_n)) \tag{1}$$

Second, we include two components, Load Distribution CPU ldc (Eq. 2) and Load Distribution Memory ldm (Eq. 3), that represent the standard deviation σ of consumed CPU and RAM across the nodes of the Kubernetes cluster, respectively. These components are used to optimize load balancing across the cluster's nodes.

$$ldc_t = -1 * \sigma(\{\frac{C'_n(t)}{C_n} \forall n \in N\}) \tag{2}$$

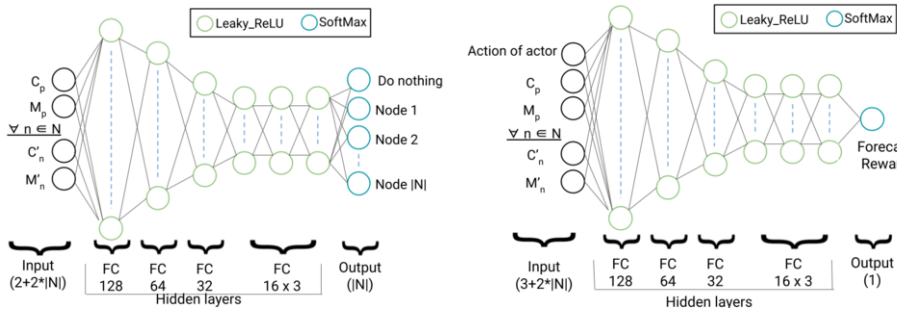
$$ldm_t = -1 * \sigma(\{\frac{M'_n(t)}{M_n} \forall n \in N\}) \tag{3}$$

Third, we include a component representing the cost of allocating a pod (Eq. 4). This component is needed since pods are always allocated from the master node, which can have different types of connections with different worker nodes in the cluster. To calculate this metric, we used the latency $L(m,n)$ measured in seconds between the master node m and the node n where the pod should be allocated. We worked under the assumption that the energy cost associated to shutting down a pod is close to 0, so the energy cost of reallocating a running pod is equivalent to that of allocating a new one.

$$tc(p,n) = M_p * L(m,n) \tag{4}$$

Finally, to ensure that our DRL agent chooses to allocate pods over not performing any action, which would otherwise always be the most energy-efficient solution, an encouraging value eg is given to the reward if the allocation is correctly made to a node n. This value is simply $eg = 1$.

The reward $r_t(p,n)$ for a specific pod-node recommendation p,n at timestamp t is then computed by weighting all components as defined in Eq. 5. The weights aim to



(a) Actor network (b) Critic network

Figure 1. Architecture of the two PPO Neural Networks used to train our DRL agent.

change the objective of the model to focus on reducing total energy consumption w_e, distributing energy consumption across all nodes w_{ldc} and w_{ldr}, reducing cost of pod allocation $w_t c$ or allocating as many pods as possible w_{eg}. The reward has a value of 0 if the agent chooses to do nothing (i.e. allocation to the fake node) and a value of -10 if the action is illegal (e.g. surpassing the maximum available resources of a node).

$$r_t(p,n) = \begin{cases} -10 & \text{if action is illegal} \\ 0 & \text{if action is fake node} \\ e(p,n) * w_e + ldc_t * w_{ldc} + ldm_t * w_{ldr} + & \\ tc(p,n) * w_{tc} + eg * w_{eg} & \text{otherwise} \end{cases} \quad (5)$$

3.2. PPO Neural Network architectures and model training

In our PPO model, both the actor and critic networks have the same hidden layer architecture as depicted in Figure 1. All the activation functions are LeakyReLU, which is used to help with vanishing gradient problems in the training process. We implemented the model using *stable_baselines3* [2] and trained it to focus on reducing the total energy consumption w_e while allocating the maximum number of pods w_{eg}. We performed the training on a custom simulated environment implemented using *gymnasium* [3]. The environment includes a generator that generates a dataset of pod specifications with different requested resources (RAM and CPU) and emulates resource consumption with different quantities from the requested ones to simulate a more realistic diverse behavior.

4. Integration with Kubernetes

In order to validate our trained DRL agent, we integrated it with Kubernetes as depicted in Figure 2. The objective of this integration is to influence the Kubernetes base scheduler so that it schedules pods for execution in the node recommended by the pre-trained DRL model. In order to enforce the desired allocations, we developed a Kubernetes Operator

[2] https://stable-baselines3.readthedocs.io/en/master/. Accessed May 13th, 2024
[3] https://gymnasium.farama.org/index.html. Accessed May 13th, 2024

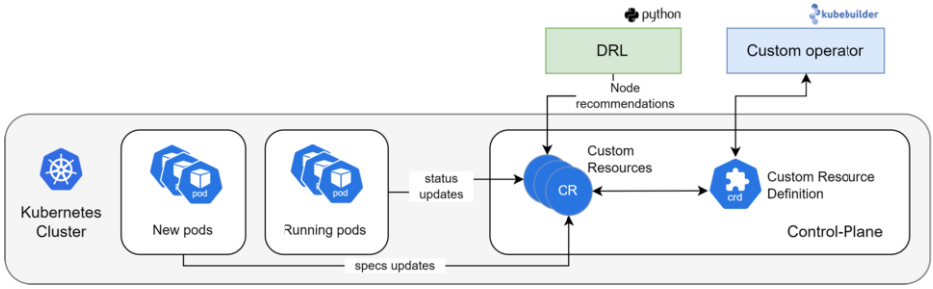


Figure 2. System diagram showing the integration of our DRL agent with the Kubernetes control plane.

```
apiVersion: codeco.codeco/v1
kind: PodPlacement
metadata:
  ...
currentStatus:
  podName: stress-pod-7155
  currentNodeName: spoke-node
  requiredCPU: 2500m
  requiredMemory: 5000Mi
objectiveStatus:
  podName: stress-pod-7155
  objectiveNodeName: worker-node
  timestamp: 2024-04-30T16:16:51Z
```

Figure 3. Example Custom Resource of the *PodPlacement* CRD defined by our custom K8s Operator.

based on the Operator Framework [4] which defines a Custom Resource Definition (CRD), named *PodPlacement*. This CRD specifies both the current status of the pods, including their resource requests (in terms of CPU and RAM) and the node where each pod is currently executed, as well as the objective status with the node indicated by the DRL model as better placement for the pod. An example of a Custom Resource (CR) for this definition with fictitious attribute values is depicted in Figure 3.

The DRL model can indicate at any time the node where a pod, whether new or already running, should be allocated by updating the *objectiveNodeName* in the CR. In parallel, the operator is responsible for both keeping the CR up to date with the current status of the pods as well as for enforcing the pod allocation by constantly monitoring all instances of the *PodPlacement* CR and applying any required allocations or reallocations that appear in the *objectiveStatus*. To monitor the energy consumption of the cluster, the operator uses the Kepler plug-in [5], which measures energy consumption per node and exports the data to a Prometheus service [6].

5. Experimental setup and evaluation results

To perform our experiments, we used a 3-server testbed with the architecture and specifications depicted in Figure 4. In order to mimic an edge-cloud environment, the testbed is conformed of 2 powerful servers, equivalent to cloud servers, and one with more limited resources, simulating an edge device. The *Supermicro 1U* hosts the control plane of the Kubernetes cluster and can also be used to host pods for execution, while the other two servers are used as Kubernetes worker nodes.

[4] https://operatorframework.io/. Accessed May 13th, 2024.
[5] https://github.com/sustainable-computing-io/kepler. Accessed May 13th, 2024.
[6] https://prometheus.io/. Accessed May 13th, 2024.

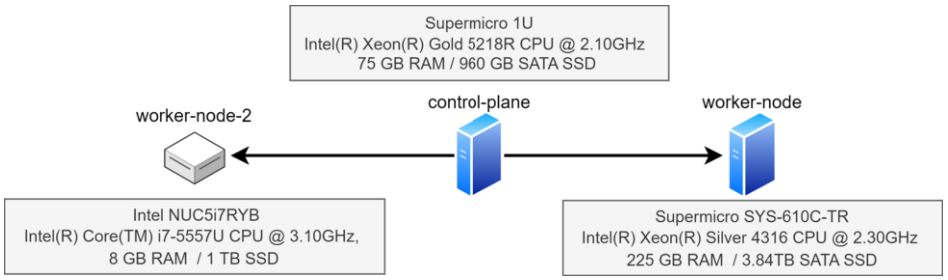


Figure 4. Testbed setup and hardware specifications of the servers used in our experiments.

5.1. Workload benchmarking and pod generator

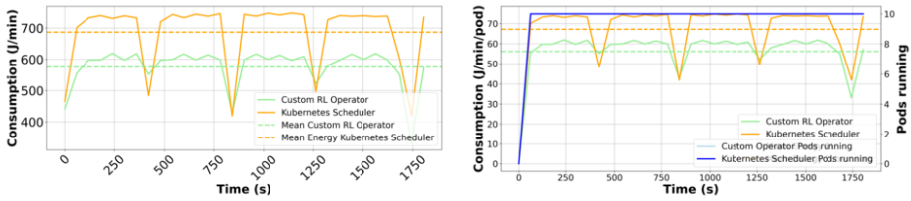
We ran our experiments on simulated edge-cloud workloads by leveraging the stress tool available in Debian OS [7] to develop a benchmarking tool that stresses the Kubernetes cluster in terms of CPU, RAM and I/O. We then used this tool to generate a number of "stresser pods" with different duration, CPU and RAM requirements. Stresser pods ensure that requested resources are always used so that we can have reproducible experiments that stress the hardware components of the cluster. For the reported evaluation, we ran 3 sets of experiments with 10, 25 and 40 stresser pods. To ensure reproducibility, all the stresser pods in all of the experiments were created at timestamp 0 with 2.5 cores of CPU, 5 GB of RAM and 400 seconds of duration. Additionally, we chose to give pods an execution time of 400 seconds to analyze the behaviour of the system throughout the pod lifecycle and to detect any potential patterns in the energy consumption.

5.2. Evaluation metrics

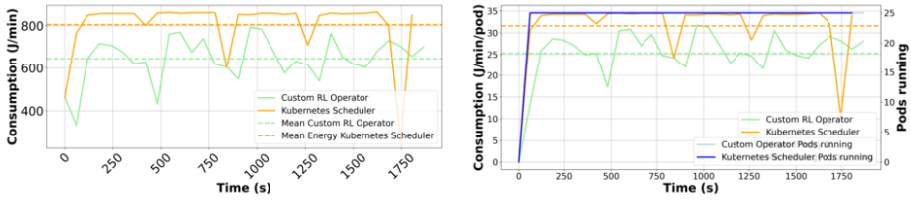
In order to validate our approach, we compared the pod allocations recommended by the DRL agent and enforced by the custom Kubernetes Operator to the pod allocation decisions taken by the standard Kubernetes scheduler. The Kubernetes scheduler allocates new pods to the node with the most available resources at the time, essentially following a greedy algorithm, and does not consider reallocations of running pods at all. In particular, we compared both approaches using the following metrics:

- Total energy consumed in the cluster over the last minute, measured in Joules and obtained from Kepler. Results are reported per time unit instead of aggregated metrics in order to analyze patterns in the behaviour of the cluster.
- Energy consumed per pod running in the cluster over the last minute, also measured in Joules. This metric is considered due to the fact that the Kubernetes scheduler has a simpler logic and allocates pods faster than our custom Kubernetes Operator. Thus, a comparison of the total consumed energy might not always be accurate. This metric is computed by dividing the total consumed energy over the last minute by the number of running pods as obtained from Prometheus.

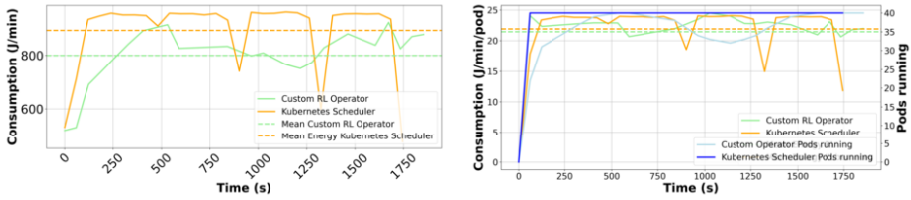
[7] https://manpages.debian.org/buster/stress/stress.1.en.html. Accessed May10th, 2024.



(a) 10 pods



(b) 25 pods



(c) 40 pods

Figure 5. Experimental results for 10, 25 and 40 pods showing total energy consumption (to the left) and energy consumption per pod (to the right) of our DRL-based approach (in green) compared to the base Kubernetes scheduler (in orange).

5.3. Evaluation results

The results obtained from our experiments can be seen in Figure 5, which shows total energy consumed (to the left) and energy consumed per pod and pods running (to the right) for three sets of experiments with 10, 25 and 40 stresser pods. In the case of 10 pods, we can observe that our DRL-powered custom operator is able to save 100 joules per minute, amounting to 19% in the total energy consumption of the experiment as explained in Table 1. A similar improvement of about 120 joules per minute (or 24%) in the consumed energy can be observed with 25 pods. Additionally, in both experiments we observe sharp periodic drops in the energy consumption, which result from the stresser pods restarting every 400 seconds (the duration they were configured to last for). On the other hand, the improvement in total consumed energy in the experiment with 40 pods drops down to around 11.78%. This is due to the fact that the testbed servers start being overloaded with that number of pods, and thus the allocation and execution of pods is slower than in the previous two experiments.

Considering the energy consumption per pod, the results indicate that the improvement in the case of 10 and 25 pods is fairly comparable to the one in the reduction of the total energy consumption. However, when considering 40 stresser pods, the improve-

Table 1. Percentage of improvement in total and per pod energy consumption in our experiments.

Number of pods	Total energy improvement	Energy per pod improvement
10	19.35%	16.45%
25	24.57%	20.05%
40	11.78%	2.08%

ment rate of energy consumption per pod drops down to about 2%. This is due to the fact that the allocation of pods with our custom operator is slower than in the other two experiments due to the clusters saturation. This slower allocation makes our custom operator consume more energy per pod initially, reducing its overall improvement when compared to the Kubernetes scheduler which is able to allocate all pods instantaneously.

Overall, these results indicate that our proposed approach can significantly reduce energy consumption in Kubernetes clusters while not affecting the execution of pods, especially in cases of normal cluster operations. On the other hand, when a cluster's resources start becoming overloaded, the reduction offered by our DRL method is significantly reduced, although a slight improvement is still noticeable. Crucially, our custom Kubernetes operator is able to allocate all pods to the cluster, even in highly-stressed cases such as our 40 pods experiment.

6. Conclusions and future work

In this paper, we proposed a PPO-based DRL agent to optimize energy consumption in Kubernetes clusters and implemented a custom Kubernetes operator to evaluate our agent. Our experiments indicate that our solution manages to reduce the overall energy consumption by between 19% and 24% when the cluster is not saturated, with a lower reduction of 11% in case of saturation. In the future, we plan to extend our energy modeling to include other parameters coming from the network and system layers. Moreover, we will improve our current Kubernetes operator together with the testbed in order to run experiments with larger datasets. We shall also expand our experiments by performing a comparison between PPO and other RL techniques such as A2C or DQN using heterogeneous pods and a more complex baseline than the Kubernetes scheduler. Finally, we shall convert our DRL model into a Multi-Agent Reinforcement Learning (MARL) solution in order to ensure privacy of the used data, especially in multi-cluster environments.

Acknowledgements

This work has received funding from the European Commission Horizon Europe programme under grant agreement number 101092696 (CODECO) and from the Swiss State Secretariat for Education, Research and Innovation (SERI) under contract number 23.00028. It has also received funding from the European Commission Horizon Europe programme under grant agreement number 101070186 (TEADAL).

References

[1] Cao K, Liu Y, Meng G, Sun Q. An Overview on Edge Computing Research. IEEE Access. 2020;8:85714-28. doi:10.1109/ACCESS.2020.2991734.

[2] Maia A, Boutouchent A, Kardjadja Y, Gherari M, Soyak EG, Saqib M, et al. A survey on integrated computing, caching, and communication in the cloud-to-edge continuum. Computer Communications. 2024;219:128-52. doi:10.1016/j.comcom.2024.03.005.

[3] Pahl C, Lee B. Containers and Clusters for Edge Cloud Architectures – A Technology Review. In: 2015 3rd International Conference on Future Internet of Things and Cloud; 2015. p. 379-86. doi:10.1109/FiCloud.2015.35.

[4] Sofia RC, Salomon J, Ferlin-Reiter S, Garcés-Erice L, Urbanetz P, Mueller H, et al. A Framework for Cognitive, Decentralized Container Orchestration. IEEE Access. 2024;12:79978-80008. doi:10.1109/ACCESS.2024.3406861.

[5] Casalicchio E. Container Orchestration: A Survey. In: Puliafito A, Trivedi KS, editors. Container Orchestration: A Survey. Cham: Springer International Publishing; 2019. p. 221-35. doi:10.1007/978-3-319-92378-9_14.

[6] Patsias V, Amanatidis P, Karampatzakis D, Lagkas T, Michalakopoulou K, Nikitas A. Task Allocation Methods and Optimization Techniques in Edge Computing: A Systematic Review of the Literature. Future Internet. 2023 07;15:30. doi:10.3390/fi15080254.

[7] Cho C, Shin S, Jeon H, Yoon S. QoS-Aware Workload Distribution in Hierarchical Edge Clouds: A Reinforcement Learning Approach. IEEE Access. 2020;8:193297-313. doi:10.1109/ACCESS.2020.3033421.

[8] Qi F, Zhuo L, Xin C. Deep Reinforcement Learning Based Task Scheduling in Edge Computing Networks. In: 2020 IEEE/CIC International Conference on Communications in China (ICCC); 2020. p. 835-40. doi:10.1109/ICCC49849.2020.9238937.

[9] Xiong X, Zheng K, Lei L, Hou L. Resource Allocation Based on Deep Reinforcement Learning in IoT Edge Computing. IEEE Journal on Selected Areas in Communications. 2020;38(6):1133-46. doi:10.1109/JSAC.2020.2986615.

[10] Zheng T, Wan J, Zhang J, Jiang C. Deep Reinforcement Learning-Based Workload Scheduling for Edge Computing. Journal of Cloud Computing. 2022 Jan;11(1):3. doi:10.1186/s13677-021-00276-0.

[11] Jayanetti A, Halgamuge S, Buyya R. Deep reinforcement learning for energy and time optimized scheduling of precedence-constrained tasks in edge–cloud computing environments. Future Generation Computer Systems. 2022;137:14-30. doi:10.1016/j.future.2022.06.012.

[12] Sellami B, Hakiri A, Ben Yahia S, Berthou P. Deep Reinforcement Learning for Energy-Efficient Task Scheduling in SDN-based IoT Network. In: 2020 IEEE 19th International Symposium on Network Computing and Applications (NCA); 2020. p. 1-4. doi:10.1109/NCA51143.2020.9306739.

[13] Li Y. Deep Reinforcement Learning: An Overview. CoRR. 2017. arXiv:1701.07274.

[14] Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal Policy Optimization Algorithms. CoRR. 2017. arXiv:1707.06347.

[15] Buschmann P, Shorim MHM, Helm M, Bröring A, Carle G. Task Allocation in Industrial Edge Networks with Particle Swarm Optimization and Deep Reinforcement Learning. In: Proceedings of the 12th International Conference on the Internet of Things. IoT '22. New York, NY, USA: Association for Computing Machinery; 2023. p. 239–247. doi:10.1145/3567445.3571114.

[16] Li P, Xiao Z, Wang X, Huang K, Huang Y, Gao H. EPtask: Deep Reinforcement Learning Based Energy-Efficient and Priority-Aware Task Scheduling for Dynamic Vehicular Edge Computing. IEEE Transactions on Intelligent Vehicles. 2024;9(1):1830-46. doi:10.1109/TIV.2023.3321679.

[17] Qu G, Wu H, Li R, Jiao P. DMRO: A Deep Meta Reinforcement Learning-Based Task Offloading Framework for Edge-Cloud Computing. IEEE Transactions on Network and Service Management. 2021;18(3):3448-59. doi:10.1109/TNSM.2021.3087258.

[18] Wang J, Hu J, Min G, Zhan W, Zomaya AY, Georgalas N. Dependent Task Offloading for Edge Computing based on Deep Reinforcement Learning. IEEE Transactions on Computers. 2022;71(10):2449-61. doi:10.1109/TC.2021.3131040.

[19] Sen T, Shen H. Machine Learning based Timeliness-Guaranteed and Energy-Efficient Task Assignment in Edge Computing Systems. In: 2019 IEEE 3rd International Conference on Fog and Edge Computing (ICFEC); 2019. p. 1-10. doi:10.1109/CFEC.2019.8733153.

[20] Shyalika C, Silva T, Karunananda A. Reinforcement Learning in Dynamic Task Scheduling: A Review. SN Computer Science. 2020 Sep;1(6):306. doi:10.1007/s42979-020-00326-5.

[21] Wang H, Ye Y, Zhang J, Xu B. A comparative study of 13 deep reinforcement learning based energy management methods for a hybrid electric vehicle. Energy. 2023;266:126497. doi:https://doi.org/10.1016/j.energy.2022.126497.