



HAL
open science

FoRLess: A Deep Reinforcement Learning-based approach for FaaS Placement in Fog

Cherif Latreche, Nikos Parlavantzas, Hector A Duran-Limon

► **To cite this version:**

Cherif Latreche, Nikos Parlavantzas, Hector A Duran-Limon. FoRLess: A Deep Reinforcement Learning-based approach for FaaS Placement in Fog. UCC 2024 - 17th IEEE/ACM International Conference on Utility and Cloud Computing, Dec 2024, Sharjah, United Arab Emirates. pp.1-9. hal-04791252

HAL Id: hal-04791252

<https://hal.science/hal-04791252v1>

Submitted on 19 Nov 2024


HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.




Distributed under a Creative Commons Attribution 4.0 International License


FoRLess: A Deep Reinforcement Learning-based approach for FaaS Placement in Fog

Cherif Latreche 
Univ Rennes, INSA Rennes
Inria, CNRS, IRISA
Rennes, France

mohamed-cherif-zouaoui.latreche@inria.fr

Nikos Parlavantzas 
Univ Rennes, INSA Rennes
Inria, CNRS, IRISA
Rennes, France

nikos.parlavantzas@irisa.fr

Hector A. Duran-Limon 
University of Guadalajara
Department of Information Systems, CUCEA
Guadalajara, Mexico
hduran@cucea.udg.mx

Abstract—Function-as-a-Service (FaaS) is a programming model in which developers write event-triggered functions and the FaaS platform automatically manages resource allocation and function execution. FaaS is well-suited for building fog applications deployed at any location on the cloud-edge continuum, delivering both flexibility and resource efficiency. A major limitation of current FaaS platforms is their lack of support for meeting latency and energy consumption requirements. This paper addresses this limitation by exploring the use of Deep Reinforcement Learning, specifically Deep Q-Networks (DQN), to optimize the placement of FaaS functions. The paper proposes FoRLess, a DQN-based scheduler designed to learn the optimal function placement, improving the platform’s ability to satisfy latency and energy consumption requirements. This scheduler is integrated into an open-source FaaS platform. The paper describes an experimental evaluation in the Grid’5000 testbed that demonstrates that our approach achieves reductions in latency and energy consumption of up to 7.15% and 12% respectively, compared to the baseline scheduler.

Index Terms—Function as a Service, Scheduling, Deep Reinforcement Learning, Resource management

I. INTRODUCTION

Cloud computing has fundamentally changed how computing resources are accessed and utilized [1] [2], enabling users to rent processing, storage, and communication resources from a centralized location over the Internet. Cloud computing has recently evolved into the fog computing model that delivers resources closer to users along a cloud-edge continuum [3] [4]. Fog computing supports reduced latency, reduced bandwidth usage, lower costs, and improved privacy, opening up a wide range of new applications, including smart cities, and virtual reality [5].

A promising model for developing fog applications is serverless computing, and in particular the Function-as-a-Service (FaaS) model [6] [7]. In the FaaS model, developers write event-triggered functions and the platform automatically takes care of resource allocation and function execution. Beyond simplifying development, FaaS is well-suited for fog deployments because functions can be flexibly deployed at any location on the cloud-edge continuum, can be rapidly triggered, and consume resources only when needed [8].

Nevertheless, supporting fog applications places stringent requirements on FaaS platforms. First, the platforms should meet latency objectives for functions, particularly important

for latency-sensitive fog applications. Second, the platforms should reduce the energy consumption of the underlying infrastructure, addressing the growing concerns over sustainability. Meeting these requirements is challenging owing to the dynamic nature of FaaS workloads and the difficulty in predicting how resource allocation decisions will impact latency and energy consumption.

Current FaaS platforms fail to effectively meet the latency and energy reduction requirements. Many open-source FaaS platforms are available and can be deployed on diverse physical infrastructures, from cloud data centers to edge devices. Most of these platforms [9] are built on top of the Kubernetes container orchestrator, which offers a standardized method for managing containerized workloads, facilitating portability, and supporting integration with various tools. However, no open-source platform addresses the latency and energy concerns.

A growing number of research solutions seek to optimize FaaS scheduling using diverse decision-making techniques, including heuristics and machine learning [10] [11] [12] [13] [14]. However, most of these solutions focus on optimizing performance and resource efficiency, leaving the energy concern largely unaddressed.

This paper introduces an approach for FaaS scheduling that seeks to jointly satisfy latency and energy reduction requirements. The main novelty of the approach is that it applies Deep Reinforcement Learning (DRL) to automatically adjust resource allocation decisions, improving the platform’s ability to satisfy these requirements and to respond to changing workloads. The approach is implemented by extending Fission, a popular Kubernetes-based open-source FaaS platform, to integrate a custom scheduler, replacing the default Kubernetes scheduler.

Specifically, the paper makes three main contributions:

- an approach for FaaS scheduling that uses DRL, enhancing the platform’s ability to meet latency and energy reduction requirements;
- an implementation of the approach in the Fission open-source FaaS platform;
- an experimental evaluation in the Grid’5000 testbed that clearly demonstrates the benefits of the approach.

The rest of the paper is structured as follows: Sections II and III discuss the background and related work. Section IV

formalizes the problem addressed by our approach. Section V presents the approach of optimizing FaaS function scheduling using DRL. Section VI describes how the approach is implemented in Fission. The experimental evaluation is reported in section VII, followed by an analysis of the system’s limitations in section VIII. Finally, section IX concludes the paper.

II. BACKGROUND

The following section examines the foundational technologies pertinent to our study. Specifically, it covers Kubernetes, serverless platforms, and reinforcement learning.

A. Kubernetes

Kubernetes, a popular container orchestration platform [15], has gained traction for edge computing deployments. Kubernetes distributions tailored for edge environments provide the necessary infrastructure for deploying and managing containerized workloads, including FaaS functions [16]. One such distribution is K3s [17], a lightweight version of Kubernetes designed for resource-constrained environments. K3s simplifies the deployment and management of Kubernetes clusters on edge devices, making it an attractive option for edge computing scenarios.

B. Serverless Platforms

Various tools have been developed to integrate FaaS into fog computing environments, each offering features suited to specific scenarios. Commercial platforms like AWS Lambda [18], Microsoft Azure Functions [19], and Google Cloud Functions [20] provide robust solutions for deploying serverless functions. Backed by major companies, these platforms are known for their scalability and reliability, but they are limited by vendor lock-in, which ties users to specific providers [21].

In contrast, open-source frameworks such as Apache OpenWhisk and Fission provide community-driven alternatives for deploying FaaS in any type of fog environment [22] [23]. Fission, a serverless framework built specifically for Kubernetes [24], is well-suited for organizations already utilizing the container orchestration tool [25]. Fission relies on the Kubernetes’ native scheduler to determine where functions should be deployed across the cluster. It also supports multiple programming languages through customizable environments and facilitates managing serverless functions, handling dynamic workloads, and automating scaling processes [26]. We selected Fission for our work as a representative of Kubernetes-based FaaS platforms.

C. Deep Reinforcement Learning

Reinforcement Learning (RL) is a subset of machine learning where agents learn how to make decisions by interacting with their environment [27] [28]. These interactions take place within a framework called a Markov Decision Process (MDP), which models the decision-making process over time by defining states, actions, probabilities, and rewards [29]. Within this framework, agents develop a policy—a strategy that maps states to actions—aiming to maximize cumulative rewards

over time. The decision-making process involves exploring the environment, observing the outcomes of actions, and adjusting the policy based on the received feedback.

By working within the MDP framework, RL agents can handle complex tasks, like optimizing resource usage by monitoring the system’s status and making decisions that reduce costs.

Recent advancements in RL have been driven by combining RL with deep neural networks, leading to what is known as Deep Reinforcement Learning [30]. A seminal DRL algorithm we use in our work is the Deep Q-Network (DQN), developed by Mnih et al. [31]. DQN helps agents determine the best actions by estimating the value of different options, even in complicated scenarios [32].

III. RELATED WORK

Recent research, particularly in fog and cloud computing, has increasingly focused on FaaS scheduling, exploring techniques ranging from heuristic algorithms to reinforcement learning. This section reviews the existing literature on function scheduling within serverless environments.

Zhang et al. [10] address resource allocation challenges through Maxwell, an approach inspired by Maxwell’s demon in thermodynamics. Maxwell optimizes resource allocation at the per-request level, balancing efficiency and tail latency. By integrating a reinforcement learning predictor and a pipeline mechanism, the authors report a reduction in CPU resource usage as well as in the standard deviation of latency.

Mahmoudi et al. [11] introduce an algorithm for function placement that uses statistical machine learning to enhance both performance and cost-efficiency. This approach, termed ‘smart spread’, strategically selects virtual machines to maintain optimal service quality, thereby reducing the need for premature resource scaling, and outperforming existing methods.

Yu et al. [12] introduce FaaSRank, a function scheduler implemented in Apache OpenWhisk that relies on deep reinforcement learning. FaaSRank reduces the number of inflight function invocations and the average function completion time when compared to baseline schedulers.

Suresh et al. [13] propose ENSURE, a specialized function-level scheduler and resource manager tailored for serverless environments. ENSURE employs heuristic-based approaches to balance operational costs with performance metrics. Implemented within the Apache OpenWhisk framework, ENSURE achieves increased resource efficiency while ensuring that latency remains within acceptable limits.

Mampage et al. [14] propose a heuristic for optimizing function placement and dynamically adjusting resources in serverless environments. Their method focuses on minimizing resource contention and ensuring that user-defined deadlines are met. The method yields substantial gains in terms of resource consumption and meeting function deadlines.

Unlike our work, none of the previous systems addresses the objective of minimizing energy consumption. In the following, we focus solely on solutions that explicitly aim to meet this objective.

Rastegar et al. [33] propose EneX, an energy-aware execution scheduler designed for serverless environments. Their work focuses on optimizing energy consumption while ensuring timely function execution with predefined deadlines. By formulating a linear programming problem and proposing an online scheduler, the authors demonstrate significant improvements in energy efficiency, which could be valuable for serverless service providers. Unlike our work, the EneX scheduler can control the processing frequencies of nodes and can schedule function executions over time in a fine-grained manner. Our scheduler depends on Kubernetes and can only control function placement.

Chiorescu et al. [34] propose a framework for integrating a machine learning model into the Kubernetes scheduler and evaluate the framework in OpenFaaS. Experimental results show a reduction in power consumption with minimal performance loss. The framework relies on a Random Forest classifier, a supervised learning method, which uses static labeled training data and cannot improve scheduling decisions beyond what the data reflects.

Vahabi et al. [35] propose a method for scheduling functions on edge nodes and powering down inactive nodes, with the aim of minimizing overall energy consumption while maintaining the required QoS. The method relies on solving a linear programming model, which is computationally expensive. Moreover, the method is evaluated only through simulation, without implementation in a real FaaS environment.

Similarly, Righetti et al. [36] propose a function scheduling method with the goal of reducing energy consumption. The method is implemented in OpenWhisk and relies on heuristics for consolidating the function load in the smallest number of nodes, allowing the deactivation of inactive nodes. Experiments show that the method reduces energy consumption with respect to the default OpenWhisk scheduler. However, the experiments use estimates of the energy consumption based solely on CPU usage, without considering actual energy measurements.

In summary, while existing research has explored various methods for optimizing performance and resource efficiency in FaaS scheduling, the problem of reducing energy consumption remains largely unaddressed. While some efforts have begun to address this problem [33]–[36], our work is unique in applying a Deep Reinforcement Learning approach. Specifically, FoRLess integrates the dual objective of minimizing energy consumption and latency within a single, cohesive, DRL-based framework, enabling intelligent and responsive scheduling decisions.

IV. PROBLEM FORMULATION

As discussed, placing FaaS functions across multiple nodes to optimize both energy consumption and latency presents a significant challenge. Our goal is to develop a dual-objective scheduling strategy that effectively balances these performance metrics. In this section, we formalize the problem and clearly define these objectives.

We consider a system with a set of functions F placed on a set of nodes N for a period of time T . Each function i can be assigned to any node j . Upon scheduling a function, the energy consumption of the nodes and the latency of executing the function are measured. We introduce the following constraints:

- 1) A decision variable x_{ij} , which must be binary, indicates whether a function is assigned to a specific node:

$$x_{ij} \in \{0, 1\} \quad \forall i \in F, \forall j \in N \quad (1)$$

- 2) A function must be scheduled on one node, ensuring that no function is left unassigned or assigned to multiple nodes:

$$\sum_{j \in N} x_{ij} = 1 \quad \forall i \in F \quad (2)$$

A. Energy Model

Let E_{ij} be the energy consumed by function i when placed on node j for the given duration T and let E_j be total energy consumption of node j such that:

$$E_j = \sum_{i \in F} E_{ij} \cdot x_{ij} \quad \forall j \in N \quad (3)$$

The first objective is to minimize the overall energy consumption of nodes. This is expressed as minimizing the average energy consumption over all nodes:

$$\text{Minimize } \frac{1}{|N|} \sum_{j \in N} \sum_{i \in F} E_{ij} \cdot x_{ij} \quad (4)$$

B. Latency Model

We assume that l_{ij} is the latency for executing function i when placed on node j and SLO_i (Service Level Objective) is the maximum allowable latency for the function.

The second objective is to minimize the average ratio of latency to SLO across all functions, which can be expressed as :

$$\text{Minimize } \frac{1}{|F|} \sum_{i \in F} \sum_{j \in N} \frac{l_{ij} \cdot x_{ij}}{SLO_i} \quad (5)$$

To consider both energy and latency objectives simultaneously, we use a weighted sum approach such that the overall objective becomes:

$$\text{Minimize } \alpha \cdot \frac{1}{|N|} \sum_{j \in N} \sum_{i \in F} E_{ij} \cdot x_{ij} + \beta \cdot \frac{1}{|F|} \sum_{i \in F} \sum_{j \in N} \frac{l_{ij} \cdot x_{ij}}{SLO_i} \quad (6)$$

subject to:

$$x_{ij} \in \{0, 1\} \quad \forall i \in F, \forall j \in N \quad (1)$$

$$\sum_{j \in N} x_{ij} = 1 \quad \forall i \in F \quad (2)$$

This optimization problem, as currently formulated, cannot be solved directly because it requires predicting the energy consumption and latency impact of each possible function placement. The complexity and dynamic nature of these variables make this infeasible. To address this challenge, we propose using Deep Reinforcement Learning. DRL can

learn optimal placement strategies through interaction with the environment, allowing the system to effectively balance energy efficiency and performance.

V. MODEL DESIGN

In this section, we outline our approach to optimizing FaaS function scheduling using DRL. We represent the scheduling process as a Markov Decision Process and define the components of the DRL model—state, action, and reward—as follows:

A. States

In our approach, the state vector reflects the current use of resources in the cluster, which influences decision-making. To this end, we describe our state vector \mathbf{s}_t at time t as follows:

$$\mathbf{s}_t = [\mathbf{N}_t, \mathbf{F}_t, \tau_t] \quad (7)$$

where:

- $\mathbf{N}_t \in \mathbb{R}^{m \times 4}$ is the node information matrix with m nodes and 4 features per node (CPU and memory usage, power consumption and the total number of functions currently executing on the node).
- $\mathbf{F}_t \in \mathbb{R}^{p \times 3}$ is the function descriptor matrix, where each of the p functions is represented by 3 features: CPU and memory requirements, and SLO.
- $\tau_t \in \mathbb{R}$ is a scalar representing the latency of the most recent function scheduled on a node.

B. Action

In our scheduling process, the action a_t at time t involves selecting a node from a set of available nodes to which a function will be placed. This decision-making task is performed based on the system’s current state, \mathbf{s}_t .

The policy function $\pi(\mathbf{s}_t)$ maps the state to an action, which in this context is the selection of a specific node. Formally, this relationship is expressed as:

$$a_t = \pi(\mathbf{s}_t) \quad (8)$$

where a_t represents the chosen node and each action a_t corresponds to a selection from the set of N total available nodes in the system:

$$a_t \in \{1, 2, \dots, |N|\}$$

The policy function is designed to select the node that optimizes the overall system performance by considering the factors described in \mathbf{s}_t . Following node selection, the scheduler will bind the function to the chosen node.

C. Reward

We design the reward function to optimize both energy consumption and latency, aligned with (6). Table I provides information about the components of the reward function, which is defined as:

$$r_t = -(\alpha \cdot R_P + \beta \cdot R_L) \quad (9)$$

such that:

$$R_P = \frac{\sum_{i=1}^N P_i}{N \cdot C_{max}} \quad (10)$$

$$R_L = \begin{cases} \frac{\text{Latency}}{\text{SLO}} & \text{if Latency} < \text{SLO} \\ 1 & \text{otherwise} \end{cases} \quad (11)$$

TABLE I
TERMS AND DESCRIPTIONS OF THE REWARD FUNCTION

Term	Description
r_t	Reward function
α	Weight factor that balances the influence of power consumption on the reward function
P_i	Power consumption of node i
N	Number of nodes
C_{max}	Maximum power consumption among all nodes
β	Weight factor that balance the influence of response time on the reward function
<i>Latency</i>	Time required for a function to complete its execution
<i>SLO</i>	Maximum acceptable latency for a function

The latency of the scheduled function is stored as τ_{t+1} in \mathbf{s}_{t+1} , providing the agent with the most recent system state for the next decision.

The reward function uses a negative sum to penalize higher power consumption and longer response times, incentivizing the agent to minimize these metrics for higher rewards.

We extend the reward function to account for two key constraints in our implementation. First, exceeding the recommended number of concurrently executing workloads on a single node incurs a penalty of -1, as per Kubernetes guidelines [37]. Similarly, if a node lacks sufficient CPU or memory resources to execute a function, the same penalty of -1 is applied. Thus, the final reward function would be defined as follows:

$$\text{Reward} = \begin{cases} r_t & \text{if constraints are satisfied} \\ -1 & \text{otherwise} \end{cases} \quad (12)$$

VI. ARCHITECTURE DESIGN

The system architecture, as illustrated in Fig. 1, comprises Fission deployed on a K3s cluster, where one node functions as the control plane and the other nodes serve as worker nodes.

When a client request arrives, it is first processed by the Fission Router, exposed via a Kubernetes Service. The Router directs the request to the appropriate function, which is defined using a Custom Resource Definition (CRD), specifying function attributes such as CPU and memory requirements. If the function is not already running, the Router triggers the creation of a Kubernetes Deployment through Fission’s newdeploy executor.

The function instances are placed across the worker nodes by a DQN agent, operating on the control plane node and replacing the default Kubernetes scheduler. This DQN agent makes intelligent placement decisions based on real-time metrics collected from Prometheus [38], which monitors the CPU

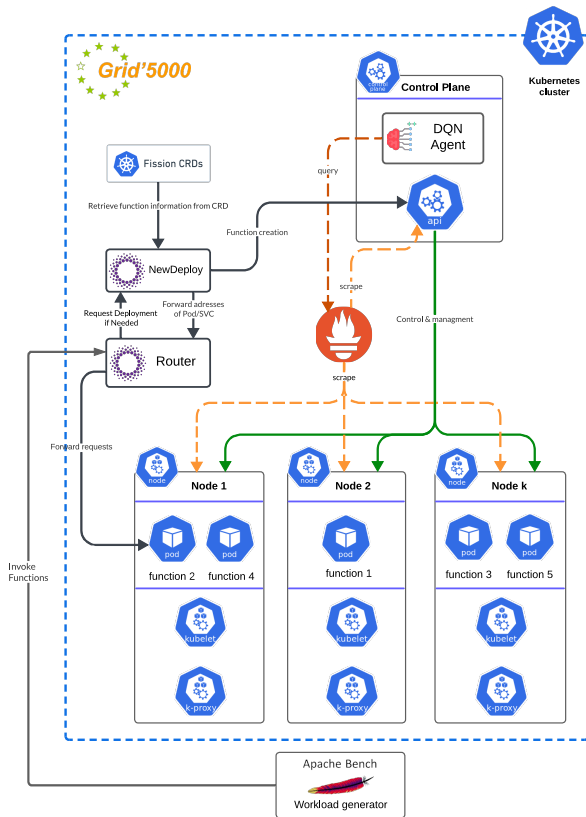


Fig. 1. System Architecture

and memory usage of nodes. The nodes in our testbed are also equipped with high-frequency wattmeters [39], allowing the DQN-based scheduler to use precise power consumption data.

The DRL model that we use has been implemented using a combination of tools. Gymnasium [40] was used to create and simulate the environment in which the agent interacts and learns. Stable Baselines3 [41] was used for leveraging the DQN algorithm. Finally, the Kubernetes Python client [42] enabled management and interaction with the Kubernetes cluster.

VII. PERFORMANCE EVALUATION

This section evaluates the effectiveness of FoRLess in reducing energy consumption and latency and analyzes the overhead introduced by FoRLess. We start by describing the experimental environment, we discuss the results of the training process, and then we compare FoRLess with other scheduling solutions in terms of energy consumption, latency, and incurred overhead.

A. Experimental setup

We use Grid'5000 [43], a large-scale, highly configurable testbed, to emulate a fog computing environment. Specifically, we use five heterogeneous machines configured as a Kubernetes cluster using K3s. One machine acts as the control plane,

while the remaining four machines act as worker nodes. The CPU and memory resources of the nodes are detailed in Table II. Furthermore, we use Apache Benchmark [44] to generate client requests and evaluate how the system handles different load conditions.

B. Training Process

During training, the agent learns to schedule FaaS functions across nodes. The training workload was selected to help the agent generalize its scheduling decisions. The workload is primarily composed of web-serving functions, supporting static and dynamic content delivery, handled by NGINX-based containers [45] with randomized resource demands. To further diversify the workload, additional functions were extracted from the examples available in the Fission GitHub repository [46], providing a wide range of functions tailored for serverless environments.

The training process spans 1,400 episodes. At the beginning of each episode, a new batch of functions is introduced for scheduling. The agent schedules each function iteratively, making decisions based on the current resource usage of the nodes and function requirements. The agent receives feedback in the form of rewards. Progress is monitored by tracking cumulative rewards over the episodes, with the expectation that the agent's policy will improve over time, resulting in higher rewards as training advances.

Fig. 2 illustrates the progression of the average reward over training steps, indicating the agent's learning process.

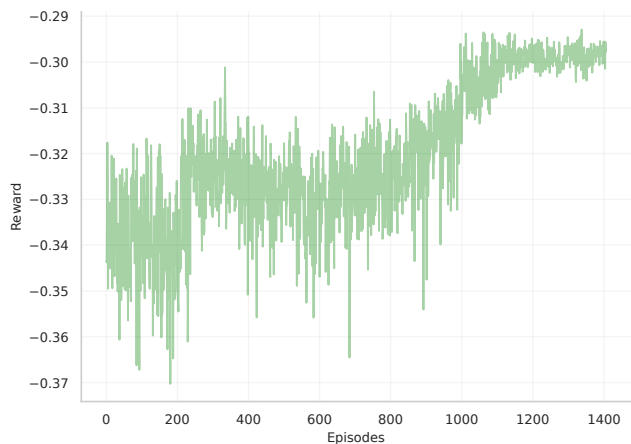


Fig. 2. Average Reward Values during Training

The training starts with the agent performing poorly, with an average reward near -0.36, marked by large fluctuations due to heavy exploration, where the agent tries various strategies. Around episode 250, the reward improves, balancing exploration with exploitation, where the agent uses the strategies it has already learned. By episode 1000, the reward rises to -0.32, with fewer fluctuations as the agent focuses more on applying successful actions. After this point, the improvement continues, albeit at a slower rate. By the end, the reward

TABLE II
COMPUTATIONAL RESOURCES FOR EXPERIMENTS

Node name	CPU	Cores	Memory
Taurus-6 (Control plane)	2 CPUs Intel Xeon E5-2630	6	32GB
Taurus-9 (Worker1)	2 CPUs Intel Xeon E5-2630	6	32GB
Sagittaire-2 (Worker2)	2 CPUs AMD Opteron 250	1	2GB
Nova-1 (Worker3)	2 CPUs Intel Xeon E5-2620 v4	8	64GB
Orion-4 (Worker4)	2 CPUs Intel Xeon E5-2630	6	32GB

stabilizes around -0.29, showing that the agent has refined its policy and consistently applies learned strategies.

The training process for the model is extensive, taking approximately 15 hours to complete. During this period, the control plane node consumes 1.49 kilowatt-hours of energy. When combined with the consumption from the rest of the cluster, the total energy used during training amounts to 6.53 kilowatt-hours.

Fig. 3 specifically illustrates the control plane node’s power consumption during training.

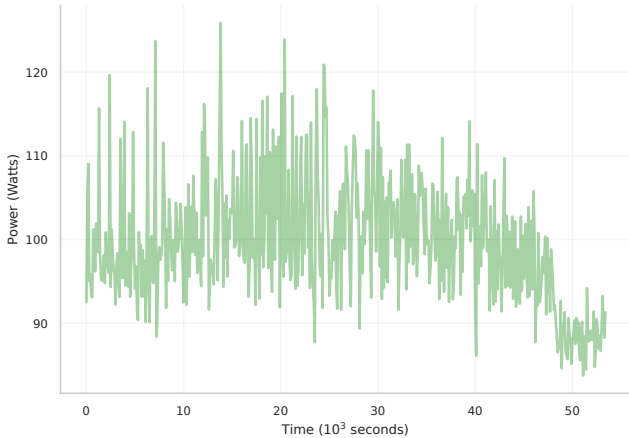


Fig. 3. Power Consumption of Control Plane Node during Training

The graph in Fig. 3 indicates that the power gradually increases throughout the training period, reflecting the resource-intensive nature of managing the training, where the model continuously optimizes itself. As training progresses, the model becomes more efficient in its decision-making, leading to a temporary decrease in power as it converges towards an optimal policy.

Although the training process has an energy cost, it is important to note that training is performed only once. Throughout this phase, we observed that the model converges, reaching stable performance by the end of the process. In the next section, we will focus on the evaluation phase, where the energy efficiency and the performance of the trained model are examined.

C. Performance Comparison

This section compares FoRLess to baseline scheduling algorithms in terms of energy consumption, latency, and overhead during the evaluation phase.

The evaluation workload is based on the Polybench benchmark suite [47], which includes a diverse set of computational functions such as linear algebra routines, and data mining kernels. These functions execute within a brief time frame, typically ranging from a few milliseconds to a few seconds, making them ideal for testing scheduling strategies in a serverless architecture.

All 30 functions from the Polybench suite are used to evaluate the schedulers, providing a wide range of computational tasks. Different input sizes were applied to introduce varying levels of complexity, allowing us to test the schedulers under different conditions. Importantly, these functions differ from those used during the training phase, helping to assess FoRLess’s ability to generalize across workloads.

Deployed on Fission as serverless functions, the minimum latency for each function is measured when executed on its own. A 20% margin is then added to the observed minimum latency to accommodate for variability, forming the basis for the SLOs.

The baseline algorithms against which we test FoRLess are described next:

- 1) Kubernetes Default Scheduler (KDS): Assigns functions to nodes based on resource requirements, affinity, and anti-affinity rules, data locality, inter-pod dependencies, and the current load on the nodes.
- 2) Round Robin (RR): Distributes functions across nodes in a fixed, repeating sequence without considering the node state or capacity.
- 3) Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS): Ranks nodes by comparing their CPU and memory utilization, power consumption, and the number of active functions. Each node’s performance is evaluated based on its distance from an ideal state, which represents the best possible values for all metrics. Functions are then assigned to the nodes closest to this ideal. This approach follows the state-of-the-art scheduling strategy described in [48].

For the evaluation, we incrementally increase the number of deployed functions and observe how each scheduler manages resource allocation.

1) *Energy Consumption*: This subsection analyzes the impact of the solutions on energy consumption. Fig. 4 illustrates the average energy consumption across the worker nodes of the four solutions.

As shown, our DRL-based solution consistently exhibits the lowest energy consumption regardless of how many functions

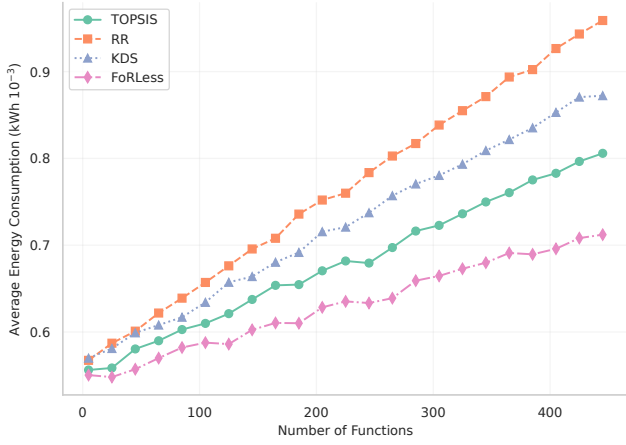


Fig. 4. Average Energy Consumption of Worker Nodes by Scheduler

are deployed in the system, reducing the consumption by 12% compared to KDS. Topsis also shows improvements, with a 5.66% reduction making it the next most efficient solution, while Round Robin underperforms, consuming 5.35% more energy than KDS.

2) *Latency*: This subsection analyzes the impact of the solutions on latency. Fig. 5 illustrates the average normalized latency (i.e., the ratio of latency to SLO) for comparison.

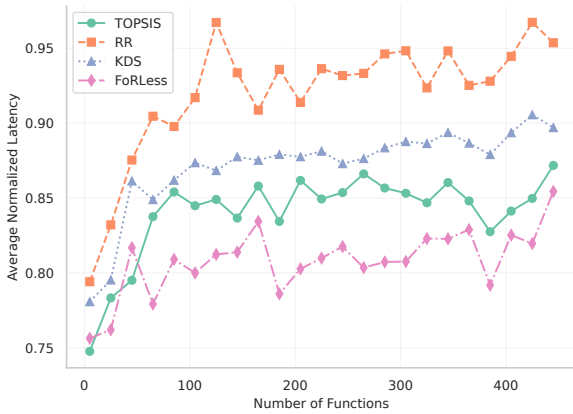


Fig. 5. Average Normalized Latency by Scheduler

Examining the data presented in Fig. 5, we note that FoRLess continues to lead, offering a 7.15% reduction in latency compared to KDS. Topsis comes close to our proposed method in terms of performance and follows it with a 3.37% reduction, while Round Robin falls further behind, showing 5.71% higher latency than KDS.

3) *Overhead*: This subsection analyzes the overhead introduced by the solutions. Specifically, it focuses on the power consumption of the control plane node, where scheduling decisions are made, though it does not execute any functions. This power consumption is shown in Fig. 6.

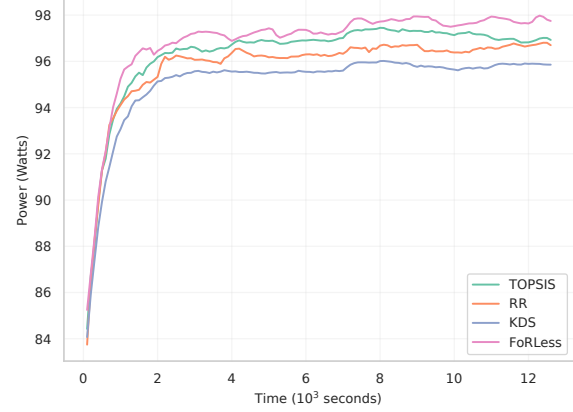


Fig. 6. Control Plane Node Power Consumption by Scheduler

We can observe from Fig. 6 that, at the start, there is an increase in power consumption for all algorithms, indicating an increase in load and activity on the control plane node. Among them, FoRLess stands out with 1.88% higher power consumption than KDS, a result of the computational demands of reinforcement learning, which involves frequent data collection from Prometheus. Topsis also shows an increase in power, consuming 1.24% more than KDS. While it relies on additional data retrieval, it remains less power-hungry than FoRLess. On the other hand, Round Robin exhibits a 0.81% increase in power consumption compared to KDS. Its simplicity translates into lower power consumption, although it still incurs more overhead than KDS, which is optimized for Kubernetes and relies on pre-established rules for scheduling, thus avoiding complex calculations.

To provide a view of how each solution impacts the system-wide energy efficiency, we combine next the energy usage of both the control plane and worker nodes. The total energy consumption during the evaluation phase is shown in Fig. 7.

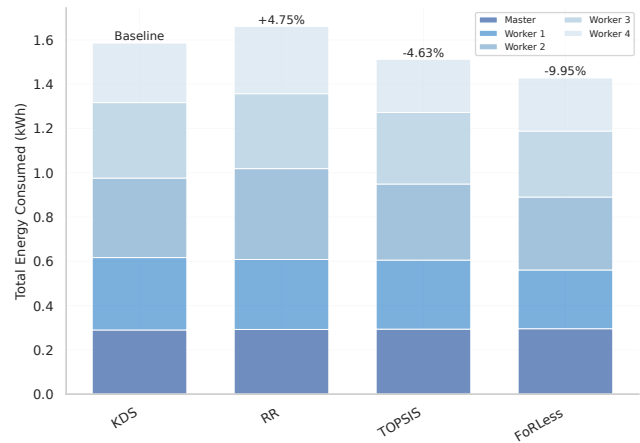


Fig. 7. Total Energy Consumption per Scheduler

In reviewing the results presented in Fig. 7, we observe that the RR solution increases total energy consumption by 4.75% compared to the baseline KDS. In contrast, the TOPSIS solution achieves a moderate reduction of 4.63%. Our FoRLess solution, on the other hand, results in the most significant reduction, decreasing overall energy consumption by 9.95% compared to KDS.

This demonstrates that the energy saved on the worker nodes offsets the additional overhead from the control plane, improving overall energy efficiency. In other words, the increased energy the control plane uses to run the FoRLess agent is a worthwhile trade-off for this experiment, leading to greater energy savings compared to the other scheduling techniques.

VIII. LIMITATIONS AND CHALLENGES

While our proposed solution has shown promising results, certain limitations should be acknowledged, each of which offers opportunities for future improvements.

An important area for improvement relates to the dependence on specific fog environments. Indeed, our current solution relies on a fixed number and type of nodes in the cluster, and any change in that configuration requires retraining the model. To overcome this limitation, we can explore adaptive retraining mechanisms or transfer learning techniques [49] [50], which would allow adjusting to new configurations without complete retraining.

Additionally, as mentioned in section VII-B, our solution incurs significant energy consumption for the control plane node during the training process. While training is only performed once, optimizing training time and resource utilization remains critical for practical deployment. Techniques such as distributed training, or model pruning [51] could be considered to make the training process more efficient.

Finally, while our system performs well for the selected evaluation workload, different from the training workload, its performance with other types of workloads—such as complex function compositions—has not yet been explored. Further investigation is thus needed to ensure the model generalizes well to a broader range of function types.

IX. CONCLUSIONS AND FUTURE DIRECTIONS

In this work, we introduced FoRLess, a Deep Reinforcement Learning approach using DQN to optimize the placement of FaaS functions across fog nodes. Our method aims to reduce both the energy consumption of nodes and the latency of function execution. We validated the effectiveness of FoRLess through experiments on the Grid’5000 testbed, where it was compared against three alternative scheduling techniques: the Default Kubernetes Scheduler, Round Robin, and a state-of-the-art TOPSIS-based solution.

The experimental results demonstrate that FoRLess significantly outperforms these methods, achieving notable improvements in both energy efficiency and latency reduction. Although the solution has some limitations, our work presents an innovative application of DRL to FaaS scheduling under

energy and performance concerns, validated in the context of a practical implementation.

Looking forward, we plan to evaluate our solution on a larger scale with more nodes and to investigate the generalizability of our approach to different types of workloads. Additionally, we plan to explore the impact of interference among FaaS functions, aiming to further enhance the performance and efficiency of our scheduling approach.

ACKNOWLEDGMENT

Experiments presented in this paper were carried out using the Grid’5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see <https://www.grid5000.fr>).

REFERENCES

- [1] N. Venkateswaran, K. Vidhya, M. Ayyannan, S. M. Chavan, K. Sekar, and S. Boopathi, “A study on smart energy management framework using cloud computing,” in *5G, Artificial Intelligence, and Next Generation Internet of Things: Digital Innovation for Green and Sustainable Economies*, pp. 189–212, IGI Global, 2023.
- [2] M. Yenugula, S. Sahoo, and S. Goswami, “Cloud computing for sustainable development: An analysis of environmental, economic and social benefits,” *Journal of future sustainability*, vol. 4, no. 1, pp. 59–66, 2024.
- [3] R. Mahmud, R. Kotagiri, and R. Buyya, “Fog computing: A taxonomy, survey and future directions,” *Internet of everything: algorithms, methodologies, technologies and perspectives*, pp. 103–130, 2018.
- [4] S. H. and N. V., “A review on fog computing: Architecture, fog with iot, algorithms and research challenges,” *ICT Express*, vol. 7, no. 2, pp. 162–176, 2021.
- [5] J. C. Guevara, R. da S. Torres, and N. L. da Fonseca, “On the classification of fog computing applications: A machine learning perspective,” *Journal of Network and Computer Applications*, vol. 159, p. 102596, 2020.
- [6] H. B. Hassan, S. A. Barakat, and Q. I. Sarhan, “Survey on serverless computing,” *Journal of Cloud Computing*, vol. 10, pp. 1–29, 2021.
- [7] H. Shafiei, A. Khonsari, and P. Mousavi, “Serverless computing: A survey of opportunities,” *Challenges and Applications*, vol. 10, 2019.
- [8] M. S. Aslanpour, A. N. Toosi, C. Cicconetti, B. Javadi, P. Sbarski, D. Taibi, M. Assuncao, S. S. Gill, R. Gaire, and S. Dustdar, “Serverless edge computing: Vision and challenges,” in *Proceedings of the 2021 Australasian Computer Science Week Multiconference, ACSW ’21*, (New York, NY, USA), Association for Computing Machinery, 2021.
- [9] V. Yussupov, J. Soldani, U. Breitenbücher, A. Brogi, and F. Leymann, “Faasten your decisions: A classification framework and technology review of function-as-a-service platforms,” *Journal of Systems and Software*, vol. 175, p. 110906, 2021.
- [10] H. Zhang, W. Huang, L. Zhao, and K. Li, “Maxwell’s demon in tail-tolerant, resource-efficient serverless computing,” in *2022 IEEE 28th International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 762–769, IEEE, 2023.
- [11] N. Mahmoudi, C. Lin, H. Khazaei, and M. Litoiu, “Optimizing serverless computing: Introducing an adaptive function placement algorithm,” in *Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering*, pp. 203–213, 2019.
- [12] H. Yu, A. A. Irissappane, H. Wang, and W. J. Lloyd, “Faasrank: Learning to schedule functions in serverless platforms,” in *2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, pp. 31–40, IEEE, 2021.
- [13] A. Suresh, G. Somashekar, A. Varadarajan, V. R. Kakarla, H. Upadhyay, and A. Gandhi, “Ensure: Efficient scheduling and autonomous resource management in serverless environments,” in *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, pp. 1–10, IEEE, 2020.
- [14] A. Mampage, S. Karunasekera, and R. Buyya, “Deadline-aware dynamic resource management in serverless computing environments,” in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pp. 483–492, IEEE, 2021.

- [15] “Kubernetes.” <https://kubernetes.io/>. Accessed: August 2024.
- [16] V. Kjorveziroski and S. Filiposka, “Kubernetes distributions for the edge: serverless performance evaluation,” *The Journal of Supercomputing*, vol. 78, no. 11, pp. 13728–13755, 2022.
- [17] “K3s.” <https://k3s.io/>. Accessed: August 2024.
- [18] “Aws lambda.” <https://aws.amazon.com/fr/pm/lambda/>. Accessed: August 2024.
- [19] “Azure functions.” <https://azure.microsoft.com/fr-fr/products/functions>. Accessed: August 2024.
- [20] “Cloud functions.” <https://cloud.google.com/functions>. Accessed: August 2024.
- [21] A. Mampage, S. Karunasekera, and R. Buyya, “A holistic view on resource management in serverless computing environments: Taxonomy and future directions,” *ACM Computing Surveys (CSUR)*, vol. 54, no. 11s, pp. 1–36, 2022.
- [22] “Apache openwhisk.” <https://openwhisk.apache.org/>. Accessed: August 2024.
- [23] “Fission.” <https://fission.io/>. Accessed: August 2024.
- [24] K. Govindarajan and A. De Tienne, “Resource management in serverless computing—review, research challenges, and prospects,” in *2023 12th International Conference on Advanced Computing (ICoAC)*, pp. 1–5, IEEE, 2023.
- [25] M. Tari, M. Ghobaei-Arani, J. Pouramini, and M. Ghorbian, “Auto-scaling mechanisms in serverless computing: A comprehensive review,” *Computer Science Review*, vol. 53, p. 100650, 2024.
- [26] B. Fonyódi, N. Pataki, and Á. Révész, “Evaluation of scalability in the fission serverless framework,” in *Annales Mathematicae et Informaticae*, vol. 58, pp. 20–29, 2023.
- [27] L. P. Kaelbling, M. L. Littman, and A. W. Moore, “Reinforcement learning: A survey,” *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.
- [28] A. G. Barto, “Reinforcement learning: An introduction by richards’ sutton,” *SIAM Rev*, vol. 6, no. 2, p. 423, 2021.
- [29] M. L. Puterman, “Markov decision processes,” *Handbooks in operations research and management science*, vol. 2, pp. 331–434, 1990.
- [30] Y. Li, “Deep reinforcement learning: An overview,” *arXiv preprint arXiv:1701.07274*, 2017.
- [31] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” *arXiv preprint arXiv:1312.5602*, 2013.
- [32] J. Fan, Z. Wang, Y. Xie, and Z. Yang, “A theoretical analysis of deep q-learning,” in *Learning for dynamics and control*, pp. 486–489, PMLR, 2020.
- [33] S. H. Rastegar, H. Shafiei, and A. Khonsari, “Enex: An energy-aware execution scheduler for serverless computing,” *IEEE Transactions on Industrial Informatics*, 2023.
- [34] R. Chiorescu and K. Djemame, “Scheduling energy-aware multi-function serverlessworkloads in openfaas,” in *Proceedings of the International Workshop on Quality of Service-Aware Serverless Computing (QServ ’23), held at the 16th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2023)*, ACM, 2023.
- [35] S. Vahabi, F. Righetti, C. Vallati, and N. Tonello, “Energy-efficient resource management for real-time applications in faas edge computing platforms,” in *Proceedings of the IEEE/ACM 16th International Conference on Utility and Cloud Computing*, pp. 1–6, 2023.
- [36] F. Righetti, N. Tonello, N. Barsanti, and C. Vallati, “Energy-efficient orchestration strategies for function-as-a-service platforms,” in *2024 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, pp. 290–295, IEEE, 2024.
- [37] “Kubernetes best practices.” <https://kubernetes.io/docs/setup/best-practices/cluster-large/>. Accessed: August 2024.
- [38] “Prometheus.” <https://prometheus.io/>. Accessed: August 2024.
- [39] “Energy consumption monitoring tutorial.” https://www.grid5000.fr/w/Energy_consumption_monitoring_tutorial. Accessed: August 2024.
- [40] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. De Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, *et al.*, “Gymnasium: A standard interface for reinforcement learning environments,” *arXiv preprint arXiv:2407.17032*, 2024.
- [41] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, “Stable baselines.” <https://github.com/hill-a/stable-baselines>, 2018.
- [42] “Kubernetes python client.” <https://github.com/kubernetes-client/python>. Accessed: August 2024.
- [43] “Grid5000.” <https://www.grid5000.fr/>. Accessed: August 2024.
- [44] “Apache benchmark.” <https://httpd.apache.org/docs/2.4/programs/ab.html>. Accessed: August 2024.
- [45] “Nginx image.” https://hub.docker.com/_/nginx. Accessed: August 2024.
- [46] “Fission examples.” <https://github.com/fission/examples>. Accessed: July 2024.
- [47] M. Á. Abella-González, P. Carollo-Fernández, L.-N. Pouchet, F. Rastello, and G. Rodríguez, “Polybench/python: benchmarking python environments with polyhedral optimizations,” in *Proceedings of the 30th ACM SIGPLAN International Conference on Compiler Construction*, pp. 59–70, 2021.
- [48] T. Menouer, “Kcss: Kubernetes container scheduling strategy,” *The Journal of Supercomputing*, vol. 77, no. 5, pp. 4267–4293, 2021.
- [49] B. Huang, F. Feng, C. Lu, S. Magliacane, and K. Zhang, “Adarl: What, where, and how to adapt in transfer reinforcement learning,” *arXiv preprint arXiv:2107.02729*, 2021.
- [50] M. E. Taylor and P. Stone, “Transfer learning for reinforcement learning domains: A survey,” *Journal of Machine Learning Research*, vol. 10, no. 7, 2009.
- [51] W. Su, Z. Li, M. Xu, J. Kang, D. Niyato, and S. Xie, “Compressing deep reinforcement learning networks with a dynamic structured pruning method for autonomous driving,” *IEEE Transactions on Vehicular Technology*, 2024.