

DRL4HFC: Deep Reinforcement Learning for Container-Based Scheduling in Hybrid Fog/Cloud System

Ameni Kallel¹ ^a, Molka Rekik¹ ^b and Mahdi Khemakhem^{2,1} ^c

¹Data Engineering and Semantics Research Unit, Faculty of Sciences of Sfax, University of Sfax, Sfax, Tunisia

²Department of Computer Science, College of Computer Engineering and Sciences, Prince Sattam Bin Abdulaziz University, AlKharj 11942, Saudi Arabia


Keywords: Hybrid Fog/Cloud Environment, Containerized Microservices Problem, Scheduling Model, Multi-Objective Optimization, Deep Reinforce Learning.


Abstract: The IoT-based applications have a set of complex requirements, such as a reliable network connection and handling data from multiple sources quickly and accurately. Therefore, combining a Fog environment with a Cloud environment can be beneficial for IoT-based applications, as it provides a distributed computing system that can handle large amounts of data in real time. However, the microservice provision to execute such applications with achieving a high Quality of Service (QoS) and low bandwidth communications. Thus, the container-based microservice scheduling problem in a hybrid Fog and Cloud environment is a complex issue that has yet to be fully solved. In this work, we first propose a container-based microservice scheduling model for a hybrid architecture. Our model is a multi-objective scheduler, named DRL4HFC, for Hybrid Fog/Cloud architecture. It is based on two Deep Reinforce Learning (DRL) agents. DRL-based agents learn the inherent properties of the various microservices, nodes, and environments to determine the appropriate placement of each microservice instance required to execute each task within the Business Process (BP). Our proposal aims to reduce the execution time, compute and network resource consumption, and resource occupancy rates of Fog/Cloud nodes. Second, we present a set of experiments in order to evaluate the effectiveness of our algorithm in terms of cost, quality, and time. The experimental results demonstrate that DRL4HFC achieves faster execution times, lower communication costs and better balanced resource loads.


1 INTRODUCTION

As software technology evolved, web application architecture is shifting from monolithic to microservices (Guo et al., 2022). Nowadays, the microservice architecture is being used for developing complex Internet of Things (IoT) applications. Driven by container technology, microservice architecture separates the monolithic application into several microservices that can interact but run independently (Kallel et al., 2022; Kallel et al., 2021). In the computing platform, loosely coupled microservices are distributed, created independently, and maintained. In order to offer low-latency services with IoT devices, the microservice-oriented Fog computing platform is emerging (Bonomi et al., 2014; Dastjerdi and Buyya,

2016; Mahmud et al., 2018). Moreover, Docker¹ and Kubernetes² (Muddinagiri et al., 2019) are gaining more and more attention from academia and industry due to their popularity as tools for container orchestration and application deployment. With the advancement of container and virtualization technologies, Edge and Fog computing technologies are growing thanks to their rapid implementation and low operational costs. Several open source systems, such as KubeEdge (Kim and Kim, 2023) and FogAtlas³, are trying to extend native containerized orchestration capabilities to host applications at Edge/Fog environment. They aimed to provide rapid development and operation of microservice-based applications. Due to the limited resources of the Fog nodes, it is often not possible to deploy all containers of a Business Process (BP) on a single Fog environment. Combining

^a  <https://orcid.org/0000-0002-7354-1276>

^b  <https://orcid.org/0000-0002-8639-4922>

^c  <https://orcid.org/0000-0001-5603-1947>

¹<https://docs.docker.com/engine/>

²<https://kubernetes.io/>

³<https://fogatlas.fbk.eu/>

a Fog environment with a Cloud one can be beneficial for IoT-based applications, as this provides a distributed computing system that is able to handle large amounts of data in real-time (Taneja and Davy, 2017; Bittencourt et al., 2018). Consequently, end devices, Fog nodes, and Cloud servers are the three computing tiers for deploying microservices-based IoT applications (Kallel et al., 2021; Sabireen and Neelannarayanan, 2021; Hu et al., 2017). Therefore, one should consider the communication between the different containers that need to be distributed on multiple nodes and resources. The distribution and management of containerized IoT applications deployed in a hybrid (i.e., Fog/Cloud) federation system is a critical issue that may have an impact on system performance. However, such systems have yet to be proven to support the deployment of containerized IoT applications across widely distributed resources coupled by heterogeneous network connectivity. In the IoT-Fog network, a task scheduling method was proposed to allocate resources to IoT tasks, which optimally selects the best nodes to execute the tasks (Liu et al., 2023; Wadhwa and Aron, 2023). There are, in the literature, several VM-based and container-based solutions (Brogi et al., 2018; Funika et al., 2023; Guo et al., 2022; Han et al., 2021; Lv et al., 2022; Wang et al., 2020) that focus on the microservice scheduling problem. However, some research studies ignored the workflow between the set of microservice container instances, and others considered only optimization of execution time, compute resource usage, or network resource consumption. In this paper, we propose a multi-objective mathematical model for Hybrid Fog/Cloud architecture. In addition, we suggest a Binary Quadratic Program (BQP) based on the proposed model to efficiently reduce the execution time, compute and network resource consumption, and resource occupancy rates of Fog/Cloud nodes. Furthermore, we present an algorithm for BQP based on the proposed model, named DRL4HFC. Furthermore, it is based on the deep learning techniques to reduce system imprecision by addressing its behavior and/or estimations, hence it helps businesses and organisations in developing trust between humans and complicated deep learning models (Yang et al., 2022). The DRL4HFC is a container-based microservice scheduler for hybrid (i.e., Fog/Cloud) federation system, which comports two Deep Reinforce Learning (DRL) based agents (Sutton and Barto, 2018; Li, 2017; François-Lavet et al., 2018). Indeed, the first agent is based on a Q-Learning technique (Deep QLearning or DQN), while the second one is a policy gradient-based agent (REINFORCE). The DRL4HFC may learn the inherent parameters of the set of BP's

microservices, nodes, and environments to ensure the efficient distribution of microservice instances for a given BP into a hybrid federation. In conclusion, the following are the contributions of this work: (i) We propose a container-based microservice scheduling model for a hybrid Fog/Cloud architecture and then a BQP-based model to effectively reduce the compute and network resource execution time, and resource occupancy rates of Fog/Cloud nodes. (ii) We develop an algorithm for BQP based on the proposed model, named DRL4HFC, in a Python environment. (iii) We implement two DRL-based agents, such as DQN and REINFORCE, and train them as scheduling agents in the DRL4HFC algorithm. (iv) We conduct a set of experiments with five different real-world BP use cases to evaluate the DRL4HFC algorithm performance in terms of cost, quality and time. Moreover, we compare the results obtained with some existing schedulers.

The rest of the paper is organized as follows. In Section 2, we discuss some similar existing work. In Section 3, we formulate the scheduling problem mathematically. In Section 4, we define the proposed DRL4HFC algorithm. Section 5 presents the experimentation and our scheduler performance evaluation. Section 6 concludes the paper by outlining our future plans.

2 RELATED WORK

The containerized deployment technique is a virtualization technique that provides a low overhead and securely segregated execution environment for microservices (Tan et al., 2020). Today, the deployment of containerized microservices is gaining attention from researchers. For example, the author in (Funika et al., 2023) has presented a novel approach based on the deep reinforcement learning technique for automating the distribution of heterogeneous resources in a real-world Cloud architecture. In addition, Wang *et al.*, in (Wang et al., 2020), created an elastic scheduling for microservices that combines task scheduling with auto-scaling in the Cloud in order to reduce the cost of virtual machines while still satisfying deadline requirements. In the same context, in order to reduce the total VM usage cost, the authors in (Islam et al., 2021) developed a novel method to allocate executors of the Spark Job to virtual machines. They implemented two DRL-based agents known as DQN and REINFORCE.

However, the IoT-based apps have evolved, and consequently, they are now built on a set of microservices, replacing the old monolithic architectures. This

shift is necessary because of the advantages it offers, such as scalability, flexibility, ease of management, and responsiveness to the dynamic needs of connected devices. Obviously, Fog and Edge computing are currently being adopted for microservice delivery to get better response times and reduce network traffic.

All the aforementioned methods, which have taken into account the deployment of IoT applications in a Cloud environment, are not directly applicable to the execution of containers on Fog servers. This finding is in line with recent studies recommending deploying a distributed scheduler and combining Cloud computing with Edge networking.

The authors in (Lv et al., 2022) proposed a Reward Sharing Deep Q-Learning algorithm (RSDQL) trying to achieve load balancing between Edge nodes while reducing communication costs. The authors in (Guo et al., 2022) proposed a multi-objective optimized microservice composition approach to reduce the service access delay and network resource consumption during the microservice composition process (Valderas et al., 2020; Ma et al., 2020). As shown in Table 1, the two proposals ((Lv et al., 2022) and (Guo et al., 2022)) aim to deploy reliable microservices in Edge computing while taking into account the number of container instances deployed by the microservice. However, these studies have focused only on the optimization of network resource usage and ignored the bandwidth variations of communication links (i.e., inter- and intra-environment communication) and the management of compute resources (i.e., the capacity of nodes, etc.).

Edge-Cloud systems, Han *et al.* (Han et al., 2021) developed a Kubernetes-oriented scheduler (KaiS). This scheduler is a multi-agent decentralized dispatcher based on deep reinforcement learning. The authors' primary goal was to find the best way for Edge access points to handle computing requests coming from numerous Edge nodes. Only the load-balancing IT resources in a hybrid Edge-Cloud environment are optimized by KaiS scheduler. An innovative cost model for deploying apps on a Fog infrastructure has been proposed by (Brogi et al., 2018). The deployment strategy is based on a simulation prototype called FogTorch that helps make a decision about deployment in Fog environment while considering the actual resource usage and cost needs. In fact, it selects the appropriate virtual machine for deploying the components of the IoT application in a hybrid Fog and Cloud computing environment. In this work, microservice containerization was not taken into account; only computing resources and inter-environment communication costs were taken into account (see Table 1). Therefore, as shown in table 1, our approach serves as a microservice scheduler that manages the workflows between the multiple container instances in a hybrid Fog and Cloud environment. Using a meticulous optimization model, the proposed scheduler is based on an algorithm that takes into consideration the minimization of several costs (computational, inter/intra-communication, and load-balancing resource utilization in hybrid environments Fog/Cloud).

3 SYSTEM MODEL AND PROBLEM FORMULATION

As depicted in Figure 1, we consider a hybrid environment composed by Fog and Cloud systems. Therefore, the scheduling problem consists in assigning a set of microservice instances to a set of nodes distributed in heterogeneous Fog and Cloud environments. The selection of nodes should take into account the optimization (i.e., minimization) of: (i) computation cost of each node, (ii) inter- and intra-environment(s) communication costs by simulating bandwidth variations of communication links, and (iii) resource load balancing. In this section, we propose a BQP-based model to optimally solve such a scheduling problem. The proposed BQP is defined by its parameters, decision variables, constraints, and objective function.

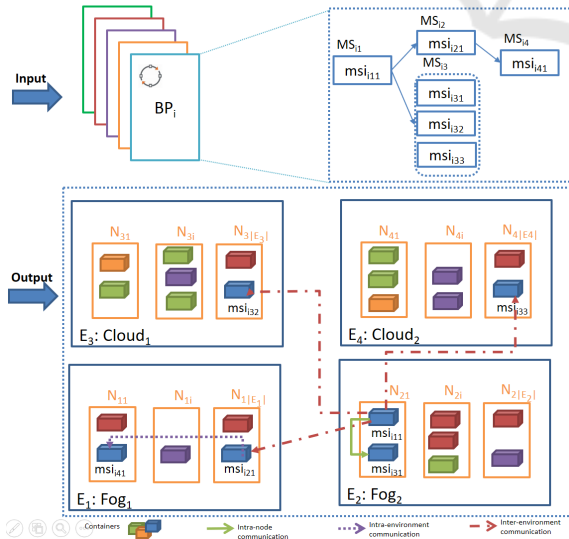


Figure 1: DRL4HFC Framework.

Other researchers were focused on optimizing the use of computing resources and their costs. In fact, for

Table 1: Existing IoT-aware scheduler within distributed environment.

Features	Related Work						Our approach
	(Lv et al., 2022)	(Han et al., 2021)	(Brogi et al., 2018)	(Islam et al., 2021)	(Guo et al., 2022)	(Funika et al., 2023)	
Dependable microservice orchestration	•		•		•	•	•
Computing resources of Fog/Edge	•	•	•		•		•
Computation cost				•		•	•
Communication cost	•		•		•	•	•
Balancing resource load	•	•					•
Multi-environments		•			•		•
Intra/Inter communication			(a)				•
Distributed container in hybrid Fog/Cloud env.		•	(b)		•		•
Microservice instances	•	•		(c)	•	•	•
Deep learning	•	•	•	•		•	•
Resources management	•	•	•	•		•	•

* Full consideration, ^(a) Only the inter-communication, ^(b) Distributed VMs, ^(c) Executor number of a job.

3.1 Parameters

As illustrated in Figure 1, we consider the parameters of both the infrastructure and the BP contexts. The infrastructure context specifies the available network and computing resources to be used to execute microservices within the BP. This context should be considered by the scheduler as a component(s) of the multiobjective optimization function. The BP context refers to the microservices to be executed in the containers. As the same, the properties of such context shall be taken into account by the scheduler. In the following, we will detail all relevant parameters.

3.1.1 Infrastructure Context

We identify a set of key notations to formalize the infrastructure context, as follows:

- ▷ $E = \{E_1, E_2, \dots, E_{|E|}\}$: set of $|E| = \phi$ Fog/Cloud environments.
- ▷ $\forall E_i \in E, E_i = \{N_{i1}, N_{i2}, \dots, N_{i|E_i|}\}$: set of $|E_i| = \omega_i$ nodes (VMs and/or PMs) deployed in a Fog/Cloud environment $E_i \in E$.
- ▷ $\forall E_i \in E, \forall N_{ij} \in E_i, core_{ij} \in \mathbb{N}^*$: number of CPU/vCPU cores of the node N_{ij} .
- ▷ $\forall E_i \in E, \forall N_{ij} \in E_i, mips_{ij} \in \mathbb{N}^*$: CPU/vCPU core speed of the node N_{ij} expressed in million instructions per second (MIPS).
- ▷ $\forall E_i \in E, \forall N_{ij} \in E_i, mem_{ij} \in \mathbb{N}^*$: memory (RAM) capacity of the node N_{ij} expressed in Gigabytes (Gb).
- ▷ $\forall E_i \in E, \forall N_{ij} \in E_i, intraNbw_{ij} \in \mathbb{N}^*$: maximum bandwidth offered by the node N_{ij} to each deployed container for intra-node communications.
- ▷ $\forall E_i \in E, intraEbw_i \in \mathbb{N}^*$: bandwidth capacity between two nodes deployed in the same environment E_i .
- ▷ $\forall E_i \in E, \forall E_j \in E, j \neq i, interEbw_{ij} \in \mathbb{N}^*$: bandwidth capacity between each node deployed in E_i and each node deployed in E_j .

▷ $\forall E_i \in E, \forall N_{ij} \in E_i, Stypes_{ij} \subset \Pi = \{\pi_1, \pi_2, \dots, \pi_{|\Pi|}\}$: set of micro-service types that can be performed by the node N_{ij} (i.e., task types), where Π represents the set of all micro-services types that depends on the considered scenario usage. For example, a smart home scenario requires a surveillance camera that may detect the motion whilst in other scenarios, one can need the storage and processing tasks, etc. Thus, the type of task shall be specified to make the right execution decision.

▷ $\forall E_i \in E, coreThreshold_i, memThreshold_i$: a predefined values specified by the provider of E_i in order to ensure that the occupancy rates of each node, deployed in the E_i , in terms of CPU and memory (respectively), cannot exceed these values.

3.1.2 BP Context

In the following, we identify the key notations to model the BP context:

- ▷ $BP = \{BP_1, BP_2, \dots, BP_{|BP|}\}$: set of $|BP| = \theta$ Business Processes.
- ▷ $\forall BP_p \in BP, BP_p = \{MS_{p1}, MS_{p2}, \dots, MS_{p|BP_p|}\}$: set of $|BP_p| = \alpha_p$ micro-services needed to execute the business process BP_p .
- ▷ $\forall BP_p \in BP, \forall MS_{pq} \in BP_p, MS_{pq} = \{msi_{pq1}, msi_{pq2}, \dots, msi_{pq|MS_{pq}|}\}$: set of $|MS_{pq}| = \beta_{pq}$ instances of the micro-service MS_{pq} needed by the business process BP_p .
- ▷ $\forall BP_p \in BP, \forall MS_{pq} \in BP_p, typeM_{pq} \in \Pi = \{\pi_1, \pi_2, \dots, \pi_{|\Pi|}\}$: the set of the micro-service instance types MS_{pq} .
- ▷ $\forall BP_p \in BP, \forall MS_{pq} \in BP_p, miM_{pq} \in \mathbb{N}^*$: the number of instructions to be executed by any instance msi_{pqr} of the micro-service MS_{pq} expressed in Millions of Instructions (MI).
- ▷ $\forall BP_p \in BP, \forall MS_{pq} \in BP_p, cpuM_{pq} \in \mathbb{N}^*$: number of CPU/vCPU cores needed by any instance msi_{pqr} of the micro-service MS_{pq} .

$\triangleright \forall BP_p \in BP, \forall MS_{pq} \in BP_p, memM_{pq} \in \mathbb{N}^*$: memory (RAM) capacity needed by any instance msi_{pqr} of the micro-service MS_{pq} .

$\triangleright \forall BP_p \in BP, \forall MS_{pq}, MS_{pr} \in BP_p, r \neq q, dataSize_{pqr} \in \mathbb{N}$: transferred data size from any instance msi_{pqs} of micro-service MS_{pq} to any instance msi_{prt} of micro-service MS_{pr} . $dataSize_{pqr} > 0$ if the micro-service MS_{pr} gets the output of micro-service MS_{pq} as an input, otherwise $dataSize_{pqr} = 0$.
 $\triangleright \forall BP_p \in BP, \forall MS_{pq}, MS_{pr} \in BP_p, r \neq q, bw_{pqr} \in \mathbb{N}$: minimum bandwidth capacity required to transfer data from any instance of the micro-service MS_{pq} to any instance of the micro-service MS_{pr} . If $dataSize_{pqr} > 0$ then $bw_{pqr} > 0$, otherwise $bw_{pqr} = 0$.

3.2 Decision Variables

We specify the following set of decision variables in our BQP model: $\forall BP_p \in BP, \forall MS_{pq} \in BP_p, \forall msi_{pqr} \in MS_{pq}, \forall E_i \in E, \forall N_{ij} \in E_i, X_{pqr}^{ij} \in \{0, 1\}$: a binary decision variable that is equal to 1 if instance msi_{pqr} of micro-service MS_{pq} of business process BP_p is deployed into a container of node N_{ij} of Cloud/Fog environment E_i , 0 otherwise.

3.3 Objective Function

Deploying a BP in a Fog and Cloud federation consists in selecting a suitable Fog or Cloud node, which may be a physical or virtual machine for launching the container associated with each instance of the BP's microservice. Indeed, this same node may be allocated to run other instances of the BP's microservices. Thus, our objective function is a quadratic equation. It includes: (i) the sum of intra- and inter-environment communication costs, (ii) the sum of computational costs of all resources, and (iii) the sum of variance of all resource occupancy.

3.3.1 Communication Cost

The communication cost among micro-services is related to two key factors:

- The size of data transmitted in a request between two different micro-service instances, and
- The capacity of bandwidth between the nodes where the two micro-service instances are allocated.

$$CommCost_{pqrst} = dataSize_{pqr} \times \left[\sum_{E_i \in E} \sum_{N_{ij} \in E_i} \frac{X_{pqs}^{ij} \times X_{prt}^{ij}}{intraNbw_{ij}} + \sum_{E_i \in E} \sum_{E_j \in E_i} \sum_{N_{ik} \in E_i} \sum_{N_{jl} \in E_j} \frac{X_{pqs}^{ik} \times X_{prt}^{jl}}{interEbw_{ij}} \right], \quad (1)$$

$$\forall BP_p \in BP, \forall MS_{pq}, MS_{pr} \in BP_p, r \neq q, \forall msi_{pqs} \in MS_{pq}, \forall msi_{prt} \in MS_{pr}$$

Expression (1) calculates the total communication cost $CommCost_{pqrst}$ between different micro-services of a given business process. It is composed by three summation terms, which are:

- The first summation term calculates the **intra-node** communication cost (i.e., communication cost between two micro-service instances deployed in the same node).
- The second summation term computes the cost of **intra-environment** communication (i.e., communication cost between two micro-service instances deployed in two different nodes but in the same environment).
- The third summation term determines the cost of **inter-environment** communication (i.e., communication cost between two micro-service instances deployed in two different nodes but in two different environments).

Thus, the total communication cost C_{comm} between all instances of all micro-services of a given business process can be calculated by equation (2) as follows:

$$C_{comm} = \sum_{BP_p \in BP} \sum_{MS_{pq} \in BP_p} \sum_{r \neq q} \sum_{msi_{pqs} \in MS_{pq}} \sum_{msi_{prt} \in MS_{pr}} CommCost_{pqrst} \quad (2)$$

The normalized communication cost $NormalizedC_{comm}$ can be defined by expression (3) as follows:

$$NormalizedC_{comm} = \frac{C_{comm}}{MaxC_{comm}} \quad (3)$$

where the maximum communication cost $MaxC_{comm}$ can be defined by expression (4) as follows:

$$MaxC_{comm} = \sum_{BP_p \in BP} \sum_{MS_{pq} \in BP_p} \sum_{r \neq q} \sum_{msi_{pqs} \in MS_{pq}} \sum_{msi_{prt} \in MS_{pr}} dataSize_{pqr} \times \left[\sum_{E_i \in E} \sum_{N_{ij} \in E_i} \frac{1}{intraNbw_{ij}} + \sum_{E_i \in E} \sum_{E_j \in E_i} \sum_{N_{ik} \in E_i} \sum_{N_{jl} \in E_j} \frac{1}{interEbw_{ij}} \right]$$

3.3.2 Computation Cost

$$CompCost_{pq} = \sum_{E_i \in E} \sum_{N_{ij} \in E_i} \frac{\sum_{msi_{pqr} \in MS_{pq}} miM_{pq} \times X_{pqr}^{ij}}{mips_{ij} \times core_{ij}}, \quad (4)$$

$$\forall BP_p \in BP, \forall MS_{pq} \in BP_p$$

The Expression (4) computes the total computation cost $CompCost_{pq}$ of each micro-service of a given business process. The total computation cost C_{comp} of all micro-services of a given business process can be calculated by expression (5) as follows:

$$C_{comp} = \sum_{BP_p \in BP} \sum_{MS_{pq} \in BP_p} CompCost_{pq} \quad (5)$$

The normalized computation cost $NormalizedC_{comp}$ can be defined by expression (6) as follows:

$$NormalizedC_{comp} = \frac{C_{comp}}{MaxC_{comp}} \quad (6)$$

where the maximum computation cost $MaxC_{comp}$ is calculated as:

$$MaxC_{comp} = \sum_{BP_p \in BP} \sum_{MS_{pq} \in BP_p} \sum_{E_i \in E} \sum_{N_{ij} \in E_i} \frac{\sum_{msi_{pqr} \in MS_{pq}} miM_{pq}}{mips_{ij} \times core_{ij}}$$

3.3.3 Variance of Resource Occupancy

In order to avoid the degradation of services execution performance, the load balancing has to be considered. We characterize the load balancing by calculating the variance of resource occupancy rates of Fog/Cloud nodes:

▷ $\forall E_i \in E, \forall N_{ij} \in E_i, occ_{Resource_{ij}} \in \mathbb{R}^+$ represents the occupancy of the resource (CPU/vCPU cores or memory) on node N_{ij} . It is calculated as follows using the general expression (7):

$$occ_{Resource_{ij}} = \sum_{BP_p \in BP} \sum_{MS_{pq} \in BP_p} \frac{\sum_{msi_{pqr} \in MS_{pq}} resourceM_{pq} \times X_{pqr}^{ij}}{resource_{ij}} \quad (7)$$

▷ $av_{Resource} \in \mathbb{R}^+$ represents the average occupancy of the resource (CPU/vCPU cores or memory) on all nodes. It is calculated as follows using the general expression (8):

$$av_{Resource} = \sum_{E_i \in E} \sum_{N_{ij} \in E_i} \frac{occ_{Resource_{ij}}}{\omega_i} \quad (8)$$

▷ $var_{Resource} \in \mathbb{R}^+$ represents the variance of the resource (CPU/vCPU cores (var_{Core}) or memory (var_{Mem})) and is calculated as follows using the general expression (9):

$$var_{Resource} = \sum_{E_i \in E} \sum_{N_{ij} \in E_i} \frac{(occ_{Resource_{ij}} - av_{Resource})^2}{\omega_i} \quad (9)$$

The total occupancy variance V_{occ} of all nodes of all clusters of all environments can be identified by expression (10) as follows:

$$V_{occ} = \gamma \times var_{Core} + (1 - \gamma) \times var_{Mem} \quad (10)$$

With $\gamma \in \{0, 1\}$ is a weighting coefficient representing the relative contribution of CPU variance (var_{Core}) to memory variance (var_{Mem}) in the calculation of total occupancy variance (V_{occ}). Its value, between 0 and 1, determines the weight attributed to each component in the overall variability of node occupancy in an HFC environment.

3.3.4 Objective Function

The global objective function aims to minimize the multiple aggregated costs, such as communication cost, computation cost, and resource occupancy variance. We adopt the weighted sum method since it is the most frequently used multi-objective optimization technique (Marler and Arora, 2010). In fact, the method regroups all objective functions into only

one normalized and aggregated function by summing them with the use of weighting factors, as shown by expression (11):

$$\begin{aligned} \text{Min } \mathbf{Z} = & \lambda_1 \times NormalizedC_{comm} + \\ & \lambda_2 \times NormalizedC_{comp} + \lambda_3 \times V_{occ} \end{aligned} \quad (11)$$

The above expression represents the global objective function where λ_1, λ_2 and λ_3 present the weighting factors aggregating the three dependent normalized sub-objective functions $NormalizedC_{comm}$ (time), $NormalizedC_{comp}$ (time) and V_{occ} (rate) calculated by expressions (3), (6), and (10), respectively. Such function is subject to a set of constraints detailed as follows.

3.4 Constraints

We categorize the set of constraints into: capacity constraints, resource occupancy constraints and placement constraints. These constraints are defined as follows.

3.4.1 Capacity Constraints

▷ Constraints (12) and (13) ensure that the required capacities, in terms of CPU and memory, respectively, by each micro-service instance should not exceed the infrastructure resource capabilities.

$$\sum_{BP_p \in BP} \sum_{MS_{pq} \in BP_p} \left(\sum_{msi_{pqr} \in MS_{pq}} cpuM_{pq} \times X_{pqr}^{ij} \right) \leq core_{ij}, \quad \forall E_i \in E, \forall N_{ij} \in E_i \quad (12)$$

$$\sum_{BP_p \in BP} \sum_{MS_{pq} \in BP_p} \left(\sum_{msi_{pqr} \in MS_{pq}} memM_{pq} \times X_{pqr}^{ij} \right) \leq mem_{ij}, \quad \forall E_i \in E, \forall N_{ij} \in E_i \quad (13)$$

▷ Constraints (14) ensure that the bandwidth occupied to transfer data between two micro-service instances, deployed in the **same node**, should not exceed the bandwidth capacity offered by that node.

$$\begin{aligned} \sum_{BP_p \in BP} \sum_{MS_{pq} \in BP_p} \sum_{MS_{pr} \in BP_p} \sum_{msi_{pqs} \in MS_{pq}} \sum_{msi_{prt} \in MS_{pr}} \sum_{r \neq q} bw_{pqr} \times \\ X_{pqs}^{ij} \times X_{prt}^{ik} \leq intraNbw_{ij}, \forall E_i \in E, \forall N_{ij} \in E_i \end{aligned} \quad (14)$$

▷ Constraints (15) ensure that the bandwidth occupied to transfer data between two micro-service instances, deployed in two **different nodes** in the **same environment**, should not exceed the bandwidth capacity offered between these nodes.

$$\begin{aligned} \sum_{BP_p \in BP} \sum_{MS_{pq} \in BP_p} \sum_{MS_{pr} \in BP_p} \sum_{msi_{pqs} \in MS_{pq}} \sum_{msi_{prt} \in MS_{pr}} \sum_{r \neq q} bw_{pqr} \times \\ X_{pqs}^{ij} \times X_{prt}^{ik} \leq intraEbw_i, \forall E_i \in E, \forall N_{ij}, N_{ik} \in E_i, k \neq j \end{aligned} \quad (15)$$

▷ Constraints (16) ensure that the bandwidth occupied to transfer data between two micro-service instances, deployed in two **different nodes** in two **different environments**, should not exceed the bandwidth capacity offered between these environments.

$$\sum_{BP_p \in BP} \sum_{MS_{pq} \in BP_p} \sum_{\substack{MS_{pr} \in BP_p \\ r \neq q}} \sum_{msi_{pqr} \in MS_{pq}} \sum_{msi_{prt} \in MS_{pr}} bw_{pqr} \times X_{pqst}^{ik} \times X_{prt}^{jl} \leq interEbw_{ij}, \forall E_i \in E, \forall E_j \in E, j \neq i, \forall N_{ik} \in E_i, \forall N_{jl} \in E_j \quad (16)$$

3.4.2 Resource Occupancy Constraints

▷ Constraints (17) and (18) ensure that occupancy rates of each nodes, of given environment, in terms of CPU and memory don't exceed the predefined values *coreThreshold* and *memThreshold*, respectively, specified by the environments providers.

$$occCore_{ij} \leq coreThreshold_i, \forall E_i \in E, \forall N_{ij} \in E_i \quad (17)$$

$$occMem_{ij} \leq memThreshold_i, \forall E_i \in E, \forall N_{ij} \in E_i \quad (18)$$

3.4.3 Placement Constraints

▷ Constraints (19) ensure that each instance of a micro-service of a given business process should be assigned to only one node of a given environment.

$$\sum_{E_i \in E} \sum_{N_{ij} \in E_i} X_{pqr}^{ij} = 1, \forall BP_p \in BP, \forall MS_{pq} \in BP_p, \forall msi_{pqr} \in MS_{pq} \quad (19)$$

▷ Constraints (20) ensure that $\forall BP_p \in BP, \forall MS_{pq} \in BP_p$, each micro-service instance $msi_{pqr} \in MS_{pq}$ having a type $typeM_{pq}$ must be deployed in a node that supports this type.

$$\text{If } X_{pqr}^{ij} = 1, \text{ then } typeM_{pq} \in Stypes_{ij}, \quad (20)$$

$$\forall E_i \in E, \forall N_{ij} \in E_i, \forall BP_p \in BP, \forall MS_{pq} \in BP_p, \forall msi_{pqr} \in MS_{pq}$$

4 DRL4HFC ALGORITHM DESIGN

After proposing the container-based microservice scheduling model for a hybrid architecture, we will define in this section our algorithm (see Algorithm 1), which is based on two DRL-based agents to implement it. The first agent adopts the Q-Deep Learning (QDN) technique while the second one is a policy gradient based technique, known as REINFORCE. The authors, in (Islam et al., 2021), explained the role and the features of each techniques. In fact, the DRL agent will receive an instant **reward** each time it conducts an **action** (i.e., each time it places a microservice instance in a container on a specific node). It

should be noted that reward value is initialized by 0 (see line 4 of Algorithm 1 and first term of equation (21)) and it is updated according to the following cases:

- Each time one constraint is verified, the algorithm assigns a small positive value (see line 10 of Algorithm 1 and second term of equation (21))
- One of the nine above-listed constraints is unverified, the algorithm assigns a negative value (see line 13 of Algorithm 1 and third term of equation (21))
- Each time the episode is successfully terminated, the algorithm calculates an episodic value (see line 30 of Algorithm 1 and fourth term of equation (21)).

After each microservice instance/container placement, the next **state** will be modified according to the previous state (see line 20 of Algorithm 1). It is worth mentioning that the DRL agent should finally be able to learn the capacity, resource occupancy, and placement constraints of each microservice within the BP in order to complete one episode and get the episodic reward. In this work, we consider: (i) the state space that contains all nodes' states and the next microservice's state, (ii) the action that is proposed by the DRL-agent (see line 6 of Algorithm 1). It can take only a value in $[0 \dots \theta]$, where $\theta = \sum_{E_i \in E} \omega_i$, indicating the index of the selected node that is able to deploy the container, and (iii) the instant reward *instreward* of each step, is calculated as follows:

$$instreward = \begin{cases} 0 & \text{initially} \\ instreward + 1 & \text{if constraint is verified} \\ instreward - const_{total} * msi_{total} & \text{if constraint is not verified} \\ instreward + r_{fixed} * cost_{reward} & \text{if the current instance is the last one to place} \end{cases} \quad (21)$$

where, (i) *const_{total}* is the total number of constraints, in our case we have nine constraints (see sections 3.4.1, 3.4.2, and 3.4.3), (ii) *msi_{total}* = $\sum_{BP_p \in BP} \sum_{MS_{pq} \in BP_p} \beta_{pq}$ is the total number of microservices instances of all business processes, (iii) *r_{fixed}* is the fixed episodic reward set to a very high value (i.e., 1000), and (vi) *cost_{reward}* = $1 - Z$ (see line 29 of Algorithm 1). In order to make our algorithm flexible and customizable, it accepts, as inputs, the agent's name and the fixed episodic reward *r_{fixed}* (see Algorithm 1).

5 EVALUATION

In this section, first we specify the experimental settings (i.e., the infrastructure, the BP, and some exist-

```

Input : episodeno = 0
1 for iteration from 0 to N do
2   if iteration == 0 then
3     Initialize state;
4     instreward = 0;
5   end
6   DRL agent proposes an action;
7   for i from 0 to consttotal do
8     Verify = "Consti is validated for the current MS instance";
9     if Verify then
10      instreward += 1;
11    end
12    else
13      instreward - = consttotal × msitotal;
14      episodeno += 1;
15      Initialize state;
16      break;
17    end
18  end
19  if Verify then
20    Update state;
21    lastmsi == "currentMSinstanceisthelastonetoplace";
22    if lastmsi then
23      Calculate the total computation cost Ccomp (eq. 5);
24      Calculate the total communication cost Ccomm (eq. 2);
25      Calculate the total variance Vocc (eq. 10);
26      Calculate the normalized computation cost
27      NormalizedCcomp (eq. 6);
28      Calculate the normalized communication cost
29      NormalizedCcomm (eq. 3);
30      Calculate the global objective function Z (eq. 11);
31      costreward = 1 - Z;
32      instreward += rfixed × costreward;
33      episodeno += 1;
34      Initialize state;
35    end
36  end
37 end

```

Algorithm 1: DRL4HFC Algorithm.

ing schedulers parameters). Secondly, we evaluated our scheduler versus current existing scheduling algorithms such as:

▷ Default Kubernetes scheduler⁴: is the Kubernetes Scheduler's default method for evenly distributing pods among nodes. The default schedule is split into two parts: (i) Predicates which apply the scheduling policy to all nodes in the current cluster and filter out any node that doesn't fit the requirements, and (ii) Priorities: After scoring each node, choose the one with the highest priority and attach it to the pod.

▷ RSDQL (Lv et al., 2022) is a microservice scheduler to balance the load between nodes while reducing communication costs.

▷ DRL-Based Scheduler (Islam et al., 2021): is a job scheduler to minimize both the cost of using the nodes and the average job completion time for the jobs.

5.1 Experimental Settings

▷ **Infrastructure Parameters:** Our scheduler is configured on two environments containing four nodes. Each node capacity is defined by the compute, network resources. Let us assume that each node has random values of CPU/vCPU core number, CPU/vCPU core speed, memory capacity and communication link

⁴<https://github.com/kubernetes-sigs/kube-scheduler-simulator>

bandwidth capacity. Note that such values should be in specific ranges as illustrated in Table 2.

▷ **BP Parameters:** To evaluate our proposal, we consider five real-world use cases:

- **BookInfo BP (BIBP)**⁵ (Joseph and Chandrasekaran, 2020): is a microservices-based system that manages book information and reviews. It comprises 4 microservices, including product-page, details, reviews, and ratings, which collaborate to provide a comprehensive book information platform. Additionally, it involves 2 communications between these microservices to ensure smooth operation.
- **Hotel Reservation BP (HRBP)**⁶: is a system created to facilitate the reservation of hotel accommodations, comprising 8 microservices and featuring a network of 16 communications. These microservices collectively manage functions such as reservations, availability, pricing, and customer management, ensuring a seamless hotel booking experience.
- **Improved Hotel Reservation BP (I-HRBP)**: is an enhanced version of the original HRBP. In this improved iteration, we have increased the number of communications between microservices to 30 in order to enhance data flow and further optimize the hotel reservation process. It offers improved efficiency and responsiveness in managing hotel bookings.
- **Interconnected Enterprise Management System BP (IEMS BP)** used by ZettaSpark Company⁷: is a comprehensive system designed to manage various aspects of enterprise operations. Additionally, as depicted in Figure 2, it encompasses 16 microservices and features a network of 30 communications, covering user management, product catalog, order processing, inventory management, and more. All these microservices are interconnected to streamline business processes effectively.
- **Improved Interconnected Enterprise Management System BP (I-IEMS BP)**: is an advanced version of the IEMS BP. In this improved variant, we have increased the number of communications to 50, facilitating a more robust data exchange and optimizing overall enterprise management. It offers advanced capabilities for efficient business operations.

⁵<https://istio.io/latest/docs/examples/bookinfo>

⁶<https://github.com/delimitrou/DeathStarBench/tree/master/hotelReservation>

⁷<https://zettaspark.io/>

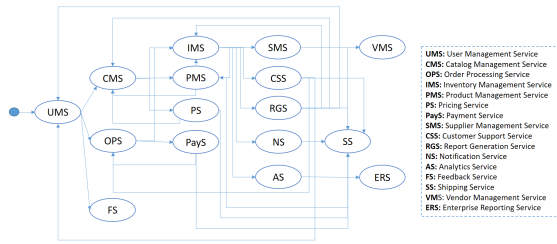


Figure 2: IEMS BP.

The left part of Table 3 presents the parameters of each BP, including the number of CPU/vCPU cores, the number of instructions, and the memory capacity needed by any instance of the BP’s micro-service.

▷ **DRL-Based Agents Parameters:** We use the parameters mentioned in the right part of Table 3 to configure the DQN and REINFORCE agents. It should be noted that these parameters are used by both our DRL4HFC scheduler and the DRL-Based Scheduler.

Table 2: Infrastructure parameter settings.

Parameter	Value
$core_{i,j}$	{16, 32, 64, 128, 256, 384, 512}
$mips_{i,j}$	[25000 – 2500000]
$mem_{i,j}$ (GB)	{16, 32, 64, 128, 256, 384, 512, 768, 1024}
$intraNbw_{i,j}$ (MB/s)	[1000 – 10000]
$intraEbw_{i,j}$ (MB/s)	[100 – 1000]
$interEbw_{i,j}$ (MB/s)	[10 – 1000]
$coreThreshold_i = memThreshold_i$	{0.85, 0.9, 0.95}

Table 3: BP and DRL-based agents parameter settings.

BP		DRL-based agents	
Parameter	Value	Parameter	Value
$cpuM_{pg}$	{1,2,4,8}	Batch Size	64
miM_{pg}	[1000-1000000]	Epsilon	0.001
$memM_{pg}$ (GB)	{1,2,4,8}	Learning Rate	0.001
$dataSize_{pg}$ (MB)	[1-20]	Optimization Priority	1
bw_{pg} (MB/s)	[1-10]	Number Training Iteration	2000
MS_{pg}	{1,2,3}		

5.2 Performance Evaluation

To evaluate the performance of our DRL-based microservices scheduler for Fog/Cloud architecture, we developed a simulation environment in Python language. It is worth mentioning that by evaluating performance, we mean assessing the quality, cost, and time required/taken by our solution. Let us start by the **execution time** metric. It is determined by dividing the number of MI required by a microservice instance by the number of MIPS offered by the node (Rekik et al., 2016). The **cost of traffic communication** metric is the cost of data transfer while considering the different types of communication links. Regarding the third metric, it focuses on the **load balancing rate** that calculates the variance of resource occupancy rates of all Fog/Cloud nodes. In fact, our

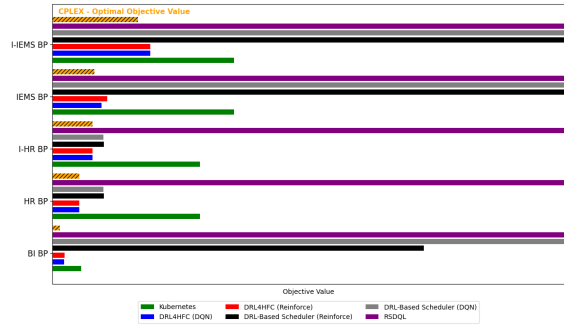


Figure 3: Cost & Quality Efficiency Evaluation.

approach aims at improving the QoS. Thus, we will evaluate in the following the **cost & quality efficiency** through comparing the value obtained by our previously proposed objective function with existing schedulers as well as the **time efficiency** during the model execution.

▷ **Cost & Quality Efficiency Evaluation:** On the one hand, let us start with the total cost metric. As shown in Figure 3, the DRL4HFC algorithm, regardless of the agent adopted, outperforms all others and achieves the lowest cost for all BP use cases. In more detail, the DRL4HFC algorithm and kubernetes as a startup using the BP with the smallest number of microservices achieved the lowest total cost, but as the number of microservices increases, the DRL4HFC algorithm outperforms even kubernetes with a more consistent difference because it learns the parameters inherent in the RDL agent: it learns the inherent parameters of different types of nodes, networks, and BP’s microservices when selecting the best placement strategy into an appropriate container for a given microservice. On the other hand, we focus on the total communication cost metric. The obtained results demonstrate that our DRL algorithm provides always the minimum communication cost (see Figure 4). These two figures show that (i) our algorithm is the better scheduler than the other ones in terms of Cost and Quality Efficiency Evaluation and (ii) the

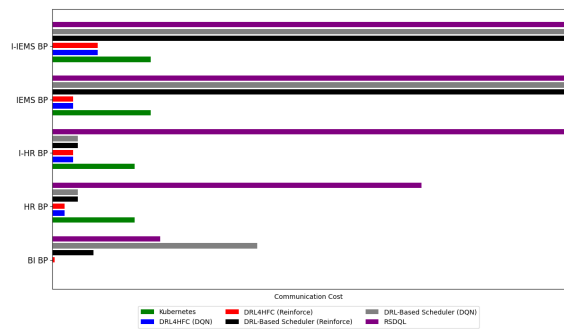


Figure 4: Communication Cost Evaluation.

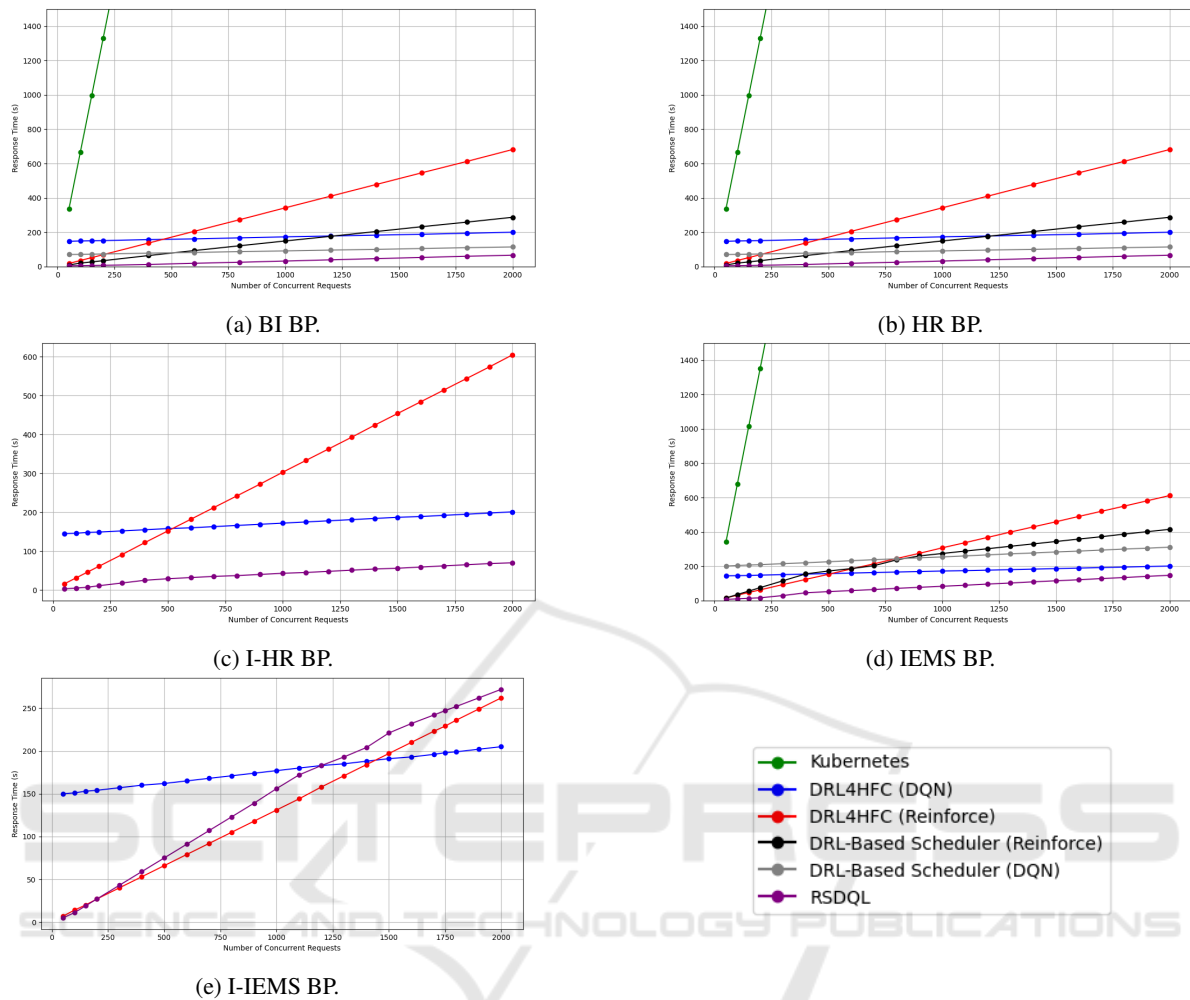


Figure 5: Time Efficiency Evaluation.

results of the two agents DQN and Reinforce, after 20,000 iterations, converged towards the same values. This means that our algorithm has converged to an optimal or near-optimal solution for the problem we are trying to solve. That finding is confirmed by comparing it with the optimal value obtained solving the BQP-based model using the high-performance optimization solver CPLEX⁸. The optimal value is shown as an orange-hatched bar in Figure 3. This figure demonstrates that these two agents constantly provide an optimal or nearly optimal solution, regardless of the specific use case. Therefore, our DRL agent is the first algorithm that seeks to minimize simultaneously the computation and communication times and the variance of the resources' occupancy rates.

▷ **Time Efficiency Evaluation:** In this section, we monitor the response time of all microservice in-

stances running within a BP and then compare our obtained results (i.e., response time) with those of other scheduling algorithms (see Figure 5). It should be noted that since only our algorithm, regardless of the adopted agent, and the RSDQL algorithm take into consideration the communication cost of a BP, we have presented only these three algorithms in Figure 5b and Figure 5d to calculate the response time for I-HR BP and I-IEMS BP. In fact, the other algorithms, including Kubernetes, offer the same values for the response time of IEMS BP and IEMS BP as well as HR BP and IHR BP because they do not account for the number of communications between microservices. As shown in Figure 5, Kubernetes consistently provides a poor value in terms of response time, regardless of the BP adopted, due to the absence of deep learning agent adoption, which helps minimize response time. As illustrated in Figure 5, the DRL4HFC based on the DQN agent and

⁸<https://www.ibm.com/products/ilog-cplex-optimization-studio/cplex-optimizer>

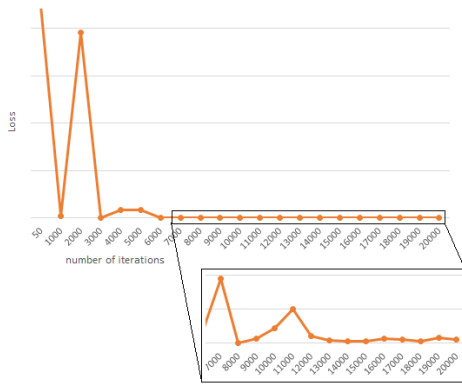


Figure 6: Loss Evolution during Deep Learning Model Training.

the RSDQL technique consistently offer the best response time, regardless of the number of competing requests and the BP adopted. Additionally, we attempted to increase the number of microservices, and dependencies between microservices to evaluate the effectiveness of our proposal. We observed that RSDQL provides better response time values in most BPs, but it remains our top-performing algorithm, especially when we increase the number of microservice instances to 16, and dependency links between microservices to 50 (see Figure 5d). This demonstrates the effectiveness of our proposal, which enables us to achieve more efficient results in terms of response time compared to other algorithms.

Additionally, the graph illustrated in Figure 6, with the x-axis representing the number of iterations and the y-axis representing the loss value, significantly illustrates the process of training a deep learning model. This type of graph is essential for several important reasons. Firstly, it highlights the evolution of our model's performance as it undergoes training. By plotting the loss as a function of the number of iterations, we visualize how the model progressively adjusts to minimize the error between its predictions and the actual values in the training data. This ongoing reduction in loss is a positive indicator, demonstrating that the model is effectively learning from the data. In summary, this loss-versus-iteration graph is a crucial tool for monitoring and evaluating the training of a deep learning model. Figure 6 shows that the loss converges towards a minimum value, and this convergence suggests that our model has achieved stable performance.

6 CONCLUSION

It is a difficult challenge to schedule microservices for IoT applications within a hybrid Fog and Cloud environment. Indeed, it must take into account the heterogeneity of nodes and the diverse requirements of the microservices in the BP. Traditional schedulers have not focused on multi-objective optimization or learning from the states of both assigned Fog/Cloud node(s) and deployed microservices. In response to this, we have proposed a scheduling model for container-based microservice execution in a hybrid architecture. The proposed scheduler, named DRL4HFC, employs two DRL-based agents. The first agent, DQN, is a Q-Learning-based agent, while the second agent, REINFORCE, is a policy gradient-based agent. The DRL4HFC algorithm aims to provide an appropriate solution, minimizing execution time, computing and network resource consumption, and resource occupancy rates of Fog/Cloud nodes. Simultaneously, it takes into account the heterogeneity of environments, nodes, communication links, microservices, and their dependencies. The effectiveness of our proposed approach is validated through a series of simulation experiments aimed at improving the total cost of microservices execution in a hybrid environment for business processes. These experiments were conducted using two real-world business processes. In the future, we plan to investigate the development of a generic model for a microservice-based scheduling system that can be adopted in large-scale Fog/Cloud real-world scenarios to further train the agents. Additionally, we intend to explore whether the DRL agents can recognize new contextual changes within Fog/Cloud environments to obtain an accurate model for an optimal or near-optimal container-based scheduler in a hybrid environment.

REFERENCES

- Bittencourt, L., Immich, R., Sakellariou, R., Fonseca, N., Madeira, E., Curado, M., Villas, L., DaSilva, L., Lee, C., and Rana, O. (2018). The internet of things, fog and cloud continuum: Integration and challenges. *Internet of Things*, 3:134–155.
- Bonomi, F., Milito, R., Natarajan, P., and Zhu, J. (2014). Fog computing: A platform for internet of things and analytics. *Big data and internet of things: A roadmap for smart environments*, pages 169–186.
- Brogi, A., Forti, S., and Ibrahim, A. (2018). Deploying fog applications: How much does it cost by the way? *small*, 1(2):20.
- Dastjerdi, A. V. and Buyya, R. (2016). Fog computing:

- Helping the internet of things realize its potential. *Computer*, 49(8):112–116.
- François-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G., Pineau, J., et al. (2018). An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, 11(3-4):219–354.
- Funika, W., Koperek, P., and Kitowski, J. (2023). Automated cloud resources provisioning with the use of the proximal policy optimization. *The Journal of Supercomputing*, 79(6):6674–6704.
- Guo, F., Tang, B., and Tang, M. (2022). Joint optimization of delay and cost for microservice composition in mobile edge computing. *World Wide Web*, 25(5):2019–2047.
- Han, Y., Shen, S., Wang, X., Wang, S., and Leung, V. C. (2021). Tailored learning-based scheduling for kubernetes-oriented edge-cloud system. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, pages 1–10. IEEE.
- Hu, P., Dhelim, S., Ning, H., and Qiu, T. (2017). Survey on fog computing: architecture, key technologies, applications and open issues. *Journal of network and computer applications*, 98:27–42.
- Islam, M. T., Karunasekera, S., and Buyya, R. (2021). Performance and cost-efficient spark job scheduling based on deep reinforcement learning in cloud computing environments. *IEEE Transactions on Parallel and Distributed Systems*, 33(7):1695–1710.
- Joseph, C. T. and Chandrasekaran, K. (2020). Intma: Dynamic interaction-aware resource allocation for containerized microservices in cloud environments. *Journal of Systems Architecture*, 111:101785.
- Kallel, A., Rekik, M., and Khemakhem, M. (2021). Iot-fog-cloud based architecture for smart systems: Prototypes of autism and covid-19 monitoring systems. *Software: Practice and Experience*, 51(1):91–116.
- Kallel, A., Rekik, M., and Khemakhem, M. (2022). Hybrid-based framework for covid-19 prediction via federated machine learning models. *The Journal of Supercomputing*, 78(5):7078–7105.
- Kim, S.-H. and Kim, T. (2023). Local scheduling in kubeedge-based edge computing environment. *Sensors*, 23(3):1522.
- Li, Y. (2017). Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*.
- Liu, Q., Kosarirad, H., Meisami, S., Alnowibet, K. A., and Hoshyar, A. N. (2023). An optimal scheduling method in iot-fog-cloud network using combination of aquila optimizer and african vultures optimization. *Processes*, 11(4):1162.
- Lv, W., Wang, Q., Yang, P., Ding, Y., Yi, B., Wang, Z., and Lin, C. (2022). Microservice deployment in edge computing based on deep q learning. *IEEE Transactions on Parallel and Distributed Systems*, 33(11):2968–2978.
- Ma, W., Wang, R., Wang, W., Wu, Y., Deng, S., and Huang, H. (2020). Micro-service composition deployment and scheduling strategy based on evolutionary multi-objective optimization. *Systems Engineering and Electronics*, 42(1):90–100.
- Mahmud, R., Kotagiri, R., and Buyya, R. (2018). Fog computing: A taxonomy, survey and future directions. *Internet of Everything: Algorithms, Methodologies, Technologies and Perspectives*, pages 103–130.
- Marler, R. T. and Arora, J. S. (2010). The weighted sum method for multi-objective optimization: new insights. *Structural and multidisciplinary optimization*, 41:853–862.
- Muddinagiri, R., Ambavane, S., and Bayas, S. (2019). Self-hosted kubernetes: deploying docker containers locally with minikube. In *2019 international conference on innovative trends and advances in engineering and technology (ICITAET)*, pages 239–243. IEEE.
- Rekik, M., Boukadi, K., Assy, N., Gaaloul, W., and Ben-Abdallah, H. (2016). A linear program for optimal configurable business processes deployment into cloud federation. In *2016 IEEE international conference on services computing (SCC)*, pages 34–41. IEEE.
- Sabireen, H. and Neelanarayanan, V. (2021). A review on fog computing: architecture, fog with iot, algorithms and research challenges. *Ict Express*, 7(2):162–176.
- Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.
- Tan, B., Ma, H., and Mei, Y. (2020). A nsga-ii-based approach for multi-objective micro-service allocation in container-based clouds. In *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, pages 282–289. IEEE.
- Taneja, M. and Davy, A. (2017). Resource aware placement of iot application modules in fog-cloud computing paradigm. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 1222–1228. IEEE.
- Valderas, P., Torres, V., and Pelechano, V. (2020). A microservice composition approach based on the choreography of bpmn fragments. *Information and Software Technology*, 127:106370.
- Wadhwa, H. and Aron, R. (2023). Optimized task scheduling and preemption for distributed resource management in fog-assisted iot environment. *The Journal of Supercomputing*, 79(2):2212–2250.
- Wang, S., Ding, Z., and Jiang, C. (2020). Elastic scheduling for microservice applications in clouds. *IEEE Transactions on Parallel and Distributed Systems*, 32(1):98–115.
- Yang, Z., Liu, N., Hu, X. B., and Jin, F. (2022). Tutorial on deep learning interpretation: A data perspective. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pages 5156–5159.