Journal of Network and Computer Applications 218 (2023) 103705

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

DQN-based intelligent controller for multiple edge domains

Alejandro Llorens-Carrodeguas [a,*], Cristina Cervelló-Pastor [a,*], Francisco Valera [b]

[a] Department of Network Engineering, Universitat Politècnica de Catalunya (UPC), 08860, Castelldefels, Spain
[b] Telematic Engineering Department, Universidad Carlos III de Madrid (UC3M), 28911, Leganés, Spain

ARTICLE INFO

Keywords:
Edge computing
Deep reinforcement learning
Resilience
Single-board computer
State of charge
VNF allocation

ABSTRACT

Advanced technologies like network function virtualization (NFV) and multi-access edge computing (MEC) have been used to build flexible, highly programmable, and autonomously manageable infrastructures close to the end-users, at the edge of the network. In this vein, the use of single-board computers (SBCs) in commodity clusters has gained attention to deploy virtual network functions (VNFs) due to their low cost, low energy consumption, and easy programmability. This paper deals with the problem of deploying VNFs in a multi-cluster system formed by this kind of node which is characterized by limited computational and battery capacities. Additionally, existing platforms to orchestrate and manage VNFs do not consider energy levels during their placement decisions, and therefore, they are not optimized for energy-constrained environments. In this regard, this study proposes an intelligent controller as a global allocation mechanism based on deep reinforcement learning (DRL), specifically on deep Q-network (DQN). The conceived mechanism optimizes energy consumption in SBCs by selecting the most suitable nodes across several clusters to deploy event requests in terms of nodes' resources and events' demands. A comparison with available allocation algorithms revealed that our solution required 28% fewer resource costs and reduced 35% the energy consumption in the clusters' computing nodes while maintaining high levels of acceptance ratio.

1. Introduction

5G and beyond networks are envisioned to be a game-changer due to their unprecedented capabilities in terms of latency, reliability, and the number of connected devices (Mahmood et al., 2021). These characteristics will support a new era of services and applications, such as the Internet of things (IoT), cooperative sensing, autonomous vehicles, and smart factories. To accommodate the latency requirements of these services, the convergence between the network function virtualization (NFV) and multi-access edge computing (MEC) is crucial for the next-generation networks to place the processing for the requested services near the end-users (Bonomi et al., 2012). Regarding the scalability and availability demands, the used edge nodes can interoperate in commodity clusters to enable failure recovery and accumulate processing capacity.

A device required by edge nodes is the single-board computer (SBC), which has become a mainstream option for IoT environments (Álvarez et al., 2021), although its use has been extended to other sectors. This use-cases extension is mainly due to hardware improvements in the last years (Upton and Halfacree, 2016; Anon, 2022b), being used either as standalone devices or in a cluster. One of the primary characteristics of an SBC is its small size, which enables a higher density of devices and lowers cost when covering huge areas (Upton and Halfacree, 2016), namely, on-boarding autonomous vehicles to provide communication services during natural disasters (Tipantuña et al., 2019; Nogales et al., 2020) and delivering green edge computing at the edge of the network (Bourhnane et al., 2021). However, the computational resource constraints of these devices must always be considered when deploying services.

An increasingly common strategy for efficiently managing deployed clusters is coupling network paradigms like NFV and MEC with containerization technologies (Slamnik-Kriještorac et al., 2020). This deployment allows lightweight virtualization by packaging only required code dependencies for the service execution, accelerating and simplifying the service instantiation (Abu-Lebdeh et al., 2017). However, deploying and managing constituent virtual network functions (VNFs) of services in a commodity cluster can be a laborious and error-prone task. Therefore, a platform capable of controlling the lifecycle of containers associated with service requests is necessary. In this vein, Kubernetes (Anon, 2022a) has become the most prominent framework to perform these functions. However, existing schedulers in this platform and others do not consider energy measurements of the participant nodes in the allocation decisions, which represents a crucial metric

* Corresponding authors.
E-mail addresses: alejandro.llorens.carrodeguas@upc.edu (A. Llorens-Carrodeguas), cristina.cervello@upc.edu (C. Cervelló-Pastor), fvalera@it.uc3m.es (F. Valera).

https://doi.org/10.1016/j.jnca.2023.103705
Received 13 February 2023; Received in revised form 19 June 2023; Accepted 19 July 2023
Available online 23 July 2023

when deploying services in battery-powered SBCs. This scenario can result in under-utilizing the available energy resources or attempting deployments destined to fail due to insufficient resources. Therefore, the necessity for a mechanism to provide energy measurements to the scheduling process.

Additionally, when considering multiple clusters' nodes to deploy service requests across them, local scheduling mechanisms may be insufficient since they need to cooperate among them (e.g., exchange nodes' information during the allocation process) to deploy incoming requests that require nodes from different clusters. This scenario implies communication challenges and extra load to the clusters' nodes in charge of performing the scheduling decisions.

This article tackles the problem of deploying constituent VNFs of service requests in an energy-constrained environment formed by several clusters while guaranteeing cost-effective resource utilization. Different from our previous work in Llorens-Carrodeguas et al. (2021), this paper proposes an intelligent controller as a global allocation mechanism to deploy service requests across existing nodes in the multi-cluster scenario. Meanwhile, local incoming requests are managed by the local scheduling process running in the master node of each cluster. In addition, a communication mechanism and a machine learning algorithm are introduced in the proposed solution to guarantee the exchange of network information and the selection of the most suitable nodes, respectively. The major contributions of this paper can be summarized as follows:

- An intelligent controller as a global allocation strategy is presented to deploy events across multiple domains of SBC clusters. The controller relies on a data distribution service (DDS) mechanism to exchange nodes' status, service requests, and allocation decisions.
- A deep reinforcement learning (DRL) algorithm is used to select the most suitable nodes where constituent VNFs of services can be deployed by considering resource utilization, battery consumption, and service requirements. This algorithm is based on the deep Q-network (DQN) method.
- A real-world testbed to evaluate the proposed approach is built using several energy-constrained SBCs in a multi-cluster edge system deployment.

The remainder of this paper is structured as follows. Section 2 describes the motivation for this work and related works. In Section 3, we present the problem statement and modeling, as well as the notation and system model used in the proposed approach. We introduce the proposed intelligent controller and explain the functions of its blocks in Section 4. Finally, we discuss the obtained results in Section 5.

2. Related work

This section provides a literature review of existing research works related to event allocation in edge/cloud environments. We have focused our attention on papers that consider energy requirements in their solutions. Additionally, we also analyze existing studies that use multi-cluster edge/cloud deployments.

2.1. Energy efficient allocation mechanisms

Gazori et al. (2020) propose a DRL approach to address the task scheduling problem in fog-based IoT applications. As the primary function, its scheduler decides whether to process the task in a fog node or send it to the cloud data center. The authors include an energy consumption model in the scheduler's proposal, thus guaranteeing the selection of the most appropriate virtual machine (VM) in terms of power consumption. Likewise, the researchers of Ding et al. (2020a) propose a Q-learning algorithm to schedule tasks energy efficiently. Their approach aims to minimize the task response time and maximize the CPU utilization of a node. The authors reduce the energy consumption of the whole cloud system by improving its resource utilization.

In Varasteh et al. (2021), propose a framework to solve the problem of the power-aware and delay-constrained joint VNFs' placement and routing (PD-VPR). In the first phase of the proposed solution, a centrality-based ranking method maps the VNFs to physical nodes. Meanwhile, the delay budget between consecutive VNFs is split in a second stage. Then, the shortest path through the selected nodes is found through the Lagrange relaxation-based aggregated cost (LARAC) algorithm (Litvinchev and Ozuna, 2013).

Jayanetti et al. (2022) present a reinforcement learning (RL) model for energy and time-optimized scheduling of tasks in edge-cloud environments. Its design integrates energy and deadline in the reward model to train the agent. Thus, it can establish a trade-off between conflicting objectives, such as energy optimization and time minimization in workload executions. The authors also introduce a hybrid DRL model comprising multiple actor networks and one critic network. The researchers used the Cloudsim simulation toolkit for evaluation purposes to test their approach. The evaluated dataset represents a synthetic workflow structure created through the Pegasus workflow framework (Pegasus, 2022).

The authors of Wahab et al. (2019) address the VNF readjustment and consolidation problem by modeling it as an integer linear programming (ILP) problem that considers a trade-off between the minimization of latency, hardware utilization, service level objective (SLO) violation cost, and VNF readjustment cost. To boost the feasibility of its model in large-scale networks, Wahab et al. propose an optimized k-medoids clustering approach that proactively divides the substrate network into several disjoint clusters. Then, each cluster aims to optimize parameters such as CPU, energy, and delay. Simulation results show that the proposed solution reduces the readjustment time while decreasing resource utilization compared to baseline solutions (e.g., K-means).

Pei et al. (2019) focus their attention on the VNF placement problem in software-defined networking (SDN) and NFV environments. The authors model this problem as binary integer programming (BIP). They propose a double deep Q network-based VNF placement algorithm (DDQN-VNPA) to solve it by considering the VNF placement cost, the instances running cost, and service rejection penalties. Their approach determines the optimal solution from a prohibitively large solution space in the first stage. Then, the VNFs are placed or released according to a threshold-based policy. The researchers evaluate its proposal with trace-driven simulations on real network topology. Despite achieving good results in comparison with baseline algorithms in terms of service rejection ratio, end-to-end delay, and VNF running time, Pei et al. do not analyze the energy consumption since this parameter was encapsulated in the instance running cost.

In Mu et al. (2021), propose a method that considers energy consumption and performance interference when allocating VNFs. The authors formulate the problem as a bin-packing one that is NP-complete. Thus, they implement two solutions according to the homogeneity of the servers. If all the servers are of the same type, the researchers propose a first-fit heuristic (FFH) algorithm to solve the problem with a lower bound. For a more general case, Mu et al. introduce the deep deterministic automatic placement (DDAP), which is based on DRL. The results show that DDAP achieves lower energy consumption and running time cost with respect to state-of-the-art methods such as FFH and ant colony system (ACS).

The authors of Qi et al. (2019a) introduce the accessible scope concept as a constraint to narrow the searching space by dividing into small groups those servers that can be used to allocate a specific request. Thus, the solution space is reduced, and the time efficiency of the VNF allocation is improved while minimizing the server energy consumption. After considering the accessible scope, Qi et al. apply the multi-stage graph algorithm (Bari et al., 2016) and greedy algorithm (Cohen et al., 2015) to verify the influence of the accessible scope on the acceptance ratio, energy consumption, and bandwidth usage.

The obtained results reveal how the proposed approach significantly reduces the runtime in large-scale network scenarios concerning those in which the accessible scope is not used. However, the results do not analyze the energy consumption metric despite the fact that minimizing this value is one of the primary objectives.

Santos et al. (2021), formulate a system model for dynamic service function chaining (SFC) placement regarding computing resources, availability levels, and energy consumption. The authors propose two policy-based RL algorithms to solve the SFC problem: proximal policy optimization (PPO) and advantage actor–critic (A2C). These algorithms aim to minimize energy consumption while taking into account availability levels. The simulations validate that the proposed algorithms can be used for SFC deployments while guaranteeing better results than a greedy approach in terms of energy consumption and acceptance rate. Following similar objectives, the authors of Solozabal et al. (2019) utilize an RL method to implement a VNF placement policy to address the VNF forward graph embedding (VNF-FGE) problem. Solozabal et al. seek to optimize the VNF-FGE by applying neural combinatorial optimization (NCO) (Bello et al., 2016) with defined constraints such as latency, bandwidth, resource utilization, and power consumption. For validating purposes, the researchers compare the feasibility of the proposed approach by comparing its solutions with the Gecode solver (Schulte et al., 2022) and the FFH algorithm. Similar to previous works, we miss an analysis related to energy consumption in the obtained results.

Khemili et al. (2022), propose a placement algorithm that aims to maximize the exploitation of MEC's resources in a balanced manner. To this end, they apply formal concept analysis (FCA) (Fkih and Omri, 2016) in the first stage to place VNFs by considering their sequencing order in the service request. In the second phase, the researchers use Fuzzy-FCA to consolidate VNF groups into the least number of virtual machines. To evaluate the proposed algorithm, this study compares its performance with the MultiSwarm algorithm (Xia et al., 2018) in terms of packaging efficiency, server and network energy consumption, latency, and resource cost. The implementation and evaluation of the algorithms use the Edge-CloudSim tool, and the results show that the proposed mechanisms minimize the number of active virtual machines used in the VNF allocation, thus reducing energy consumption with regard to the baseline strategy.

Table 1 summarizes the revised literature concerning energy-efficient allocation mechanisms.

2.2. Multi-cluster edge/cloud deployments

The authors of Zhanikeev (2015) propose an architecture called cloud visitation platform (CVP) to enable federation between cloud and fog nodes. In this architecture, the participant providers must register in the federated manager to indicate the nodes' information belonging to the federated system. The proposed solution includes modules to provide hardware awareness for VMs and container-based applications, thus allowing to VMs to sense their local environment and adapt accordingly. Additionally, the CVP provides interfaces to perform load balancing and queuing in several requests. However, Zhanikeev et al. do not perform any evaluation regarding the feasibility of the proposal in a real scenario.

In Smith et al. (2022), propose a solution called FaaS functions and data orchestrator (FaDO) to enable data-aware functions scheduling across multi-serverless compute clusters, which are geographically distributed. FaDO performs its functions by distributing the invocation of the function to the most suitable compute clusters according to the storage configurations. To this end, it uses header-based HTTP reverse proxy with three load-balancing algorithms. Thereby, cluster inter-operation is limited since user requests are sent to a selected cluster node without the possibility of being distributed across all the participant clusters.

The authors of Bruschi et al. (2019) develop a multi-cluster overlay (MCO) network paradigm based on a tunnel-less SDN solution to guarantee scalability in virtual tenant networks across the 5G distributed infrastructure. Its proposal supports software instance migrations among geo-distributed computing resources. The proposed mechanism is evaluated through simulation and emulation environments by using the Matlab software and the OpenVolcano platform (Bruschi et al., 2016), respectively.

Javed et al. (2020), present a new IoT edge-cloud federation (IoTEF) architecture for multi-cluster IoT applications by modifying its previous work (Javed et al., 2018). The proposal introduces a modular design composed of four layers that simplify the deployment and monitoring of IoT applications. In addition, it offers a federated management interface to orchestrate several clusters, an exactly-once data delivery mechanism, and fault tolerance at edge and cloud levels. The authors address the fault-tolerance problem through the Apache Kafka publish/subscribe mechanism as a data replication solution and Kubernetes as a management orchestrator. The proposed solution is evaluated by analyzing its performance in a smart building use case. The results show that the IoTEF architecture minimizes latency, saves network bandwidth, and tolerates hardware and network connectivity failures in a proper manner. Nevertheless, this solution has some limitations regarding the communication mechanism since the Kafka implementation used does not support transactions among multiple clusters. Despite the authors proposing a workaround to solve this issue, it implies several hops between other clusters before reaching the final destination.

2.3. Motivation

Despite the plethora of works addressing energy-efficient scheduling algorithms, most use classic energy models in which maximum and idle values are considered, with the objective of minimizing total consumption by reducing resource utilization. Additionally, these works lack the flexibility to be adapted to 5G and beyond networks' use cases since energy values used in their evaluations are arbitrarily selected. Furthermore, the revised literature neglects the evaluation of energy-efficient algorithms in resource-constrained devices such as SBCs, thus limiting their applicability in real use cases. Regarding multi-cluster deployments, existing researches present limitations according to the communication mechanism used in the cluster federations since the proposed strategies require several hops among nodes to communicate distant endpoints. In addition, they do not consider inter-cluster operations to process or allocate requests across existing nodes in the multi-cluster scenario.

To fill the identified gaps in the literature, this paper proposes an intelligent controller as a global allocation mechanism to deploy VNFs across multiple domains of SBC clusters. It considers resource utilization and battery levels of the nodes to select suitable nodes that accommodate the service requirements. The communication between the participant clusters and the intelligent controller is achieved through a DDS mechanism that is integrated with the mentioned elements. In contrast to most of the papers, the proposed solution is evaluated in a real testbed that uses leading technologies.

3. Problem statement and system model

In this section, we formally present the problems we address as well as the notation and system model of use.

3.1. NFV system description

To capture the environment characteristics mentioned above and formally define the problems we investigate, we consider the reference architecture depicted in Fig. 1. Under this architecture, a global SDN controller is deployed to manage a region compounded by several cluster nodes. In this way, a centralized approach is considered to

Table 1
Revised energy-efficient allocation mechanisms.

Ref.	Problem	Algorithm	Objectives	Evaluation	Shortcomings
Gazori et al. (2020)	Task scheduling	DQN	Minimize cost and time	SimPy	It is not clear how the energy consumption model is integrated into the proposed solution
Ding et al. (2020a)	Task scheduling	Q-learning	Minimize response time and maximize CPU utilization	CloudSim	A linear CPU-power consumption model is used. The energy consumption is reduced by improving resource utilization
Varasteh et al. (2021)	VNF placement and routing	LARAC	Minimize used hosts, network devices and energy	Gurobi solver	There is a strong dependency between two defined subproblems. A classic energy model is used where maximum and idle values are defined
Jayanetti et al. (2022)	Task scheduling	DRL	Minimize time and energy	CloudSim and Pegasus	Trade-off between energy optimization and time minimization is achieved by integrating energy and deadline in the reward function
Wahab et al. (2019)	VNF placement	K-medoids and ILP	Minimize latency, SLO violation cost, resource utilization, and VNF readjustment cost	MatLab	The solution requires the elimination of cost functions to boots its feasibility in large-scale networks
Pei et al. (2019)	VNF placement	DQN	Minimize VNF placement and running cost, and rejected SFC requests penalties	Tensorflow	The energy consumption is not analyzed since it was encapsulated in the instance running cost
Mu et al. (2021)	VNF placement	FFH and DQN	Minimize energy	Python-based simulations	The energy consumption is reduced by minimizing the resource consumption since they are linearly related according to the used model
Qi et al. (2019a)	VNF placement	Accessible scope	Minimize VNF time allocation and energy	C-based simulations	The energy consumption is not analyzed despite being one of the primary paper objectives
Santos et al. (2021)	SFC placement	PPO and A2C	Minimize energy	OpenAI	The proposed algorithms obtain better results than a greedy approach in terms of energy consumption and acceptance rate
Solozabal et al. (2019)	VNF-FGE	NCO	Minimize energy	Python-based simulations	An analysis related to energy consumption in the obtained results is missed
Khemili et al. (2022)	VNF placement	Fuzzy-FCA	Maximize used MEC's resources	Edge-CloudSim	The energy consumption is reduced since the number of active VMs in the allocation process is minimized. In this regard, we missed a study of the acceptance service rate

consolidate the system information (e.g., nodes' status and resources) in just one entity (i.e., the intelligent controller), thus having a global view of the whole system and reducing the number of messages that a distributed architecture could generate. Thus, we denote K as a set of clusters ($k \in K$). Each cluster k is formed by a set of physical nodes (N) where virtual network functions (f) can be scheduled. The cluster nodes are considered energy-constrained devices, representing edge nodes with a fixed amount of computational resources.

The virtual functions (f) are placed in a set of deployable units within computing nodes (P). Each virtual node $p \in P$ is identified with an ID. We include a global parameter $p_i^{k_n}$ to indicate that virtual node p_i has been placed in physical node n of cluster k, where $n \in N$ and $k \in K$.

Each physical node (n) in cluster k has resource and power capacity. The former contains the computing resources (e.g., CPU and memory). The latter implies the power source that maintains the device working. Thus, we denote the resource capacity of each node by $C_{k_n} = (C_{k_n}^{CPU}, C_{k_n}^{Mem})$ representing the available resources in terms of CPU and memory. Moreover, we indicate the available energy capacity of each node as E_{k_n}, which is expressed as the state of charge (SOC). Additionally, each node has a total output bandwidth represented as W_{k_n}.

Similar to our previous work (Llorens-Carrodeguas et al., 2021), the system can process two types of events: tasks and services. The former is a set of instructions that require a predefined and fixed value of execution time, such as log rotation associated with a particular running service and backing up a service database before its deletion. Although, we will focus on the latter because it represents the most complex case since several virtual functions form the services. More specifically, this kind of event is comprised of a sequence of VNFs, $F = \{f_1, f_2, \ldots, f_{|F|}\}$.

In this regard, each service instance $f \in F$ has a resource demand in terms of CPU and memory denoted by $D_f = (D_f^{CPU}, D_f^{Mem})$.

We denote S_R as a set of service requests arriving at the controller node. In this vein, each request $s_r \in S_R$ must be directed through the VNFs compounding the request by considering its requirements. The service-related s_r is denoted as follows:

$$s_r = \{f_1, f_2, \ldots, f_{|s_r|}\}, f_i \in F, i = 1, 2, \ldots, |s_r|.$$

Each service request s_r has specific QoS demands, such as the bandwidth requirement W_r and deadline d_r for processing the given request. Additionally, each VNF f in the request has a running time parameter (t_r) to denote the time that must pass before considering it complete. When $t_r > 0$, the event runs during the specified time. Otherwise, the event will be executed during the system's lifetime when this parameter is zero or not specified.

Finally, for each request $s_r \in S_R$, we use a binary variable $x^i_{s_r,k_n}$ to indicate the placement decision of each VNF f_i belonging to the requested service. More specifically, $x^i_{s_r,k_n} = 1$ when VNF f_i is successfully placed on node $n \in N$ of cluster $k \in K$; otherwise, $x^i_{s_r,k_n} = 0$. Table 2 provides a list of notations related to the system model.

3.2. Problem modeling

To face real-time network variations due to receiving requests with an unknown arrival time, we use the concept of time slot τ. The controller verifies the nodes' status at each time slot τ. Additionally, it can receive service requests, make deployment decisions and update the network's states. In this regard, we define $C_{n,\tau}$ as the available capacity of node n at time slot τ. Likewise, $W_{n,\tau}$ represents the available bandwidth of node n at time slot τ.

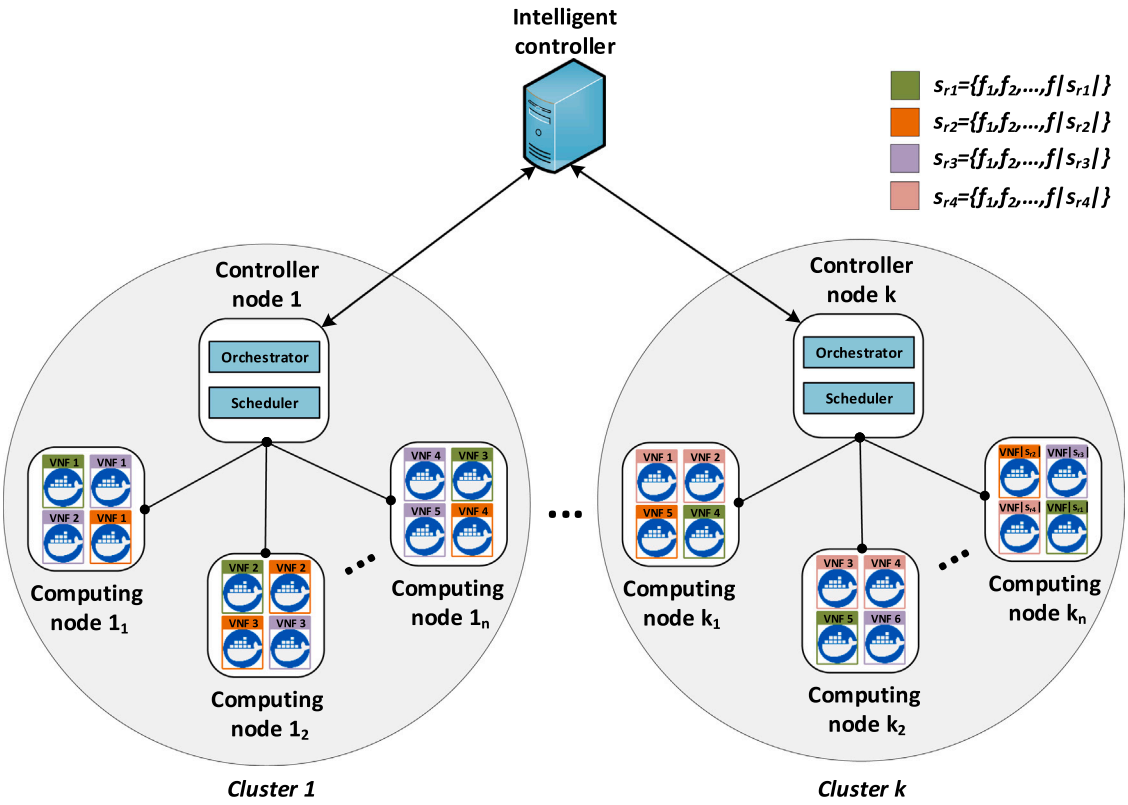


Fig. 1. Reference NFV system formed by several clusters of edge nodes which an intelligent controller manages.

Additionally, we define $S_{R,\tau} \subset S_R$ to cope with the possibility of receiving several requests at time slot τ. Thus, simultaneous service requests will be treated as they arrive by considering the event's ranking in the priority queue. The event's ranking is calculated based on two factors: $delay(f)$ and $wait_{queue}(f)$. The former represents the time that the virtual function's execution f can be delayed without missing its deadline (see (1)). The latter represents the waiting time of the VNF f in the queue (see (2)). In this equation, t_a denotes the arrival time of the service request.

$$delay(f) = f_{d_r} - t_{now} - f_{t_r} \tag{1}$$

$$wait_{queue}(f) = t_{now} - f_{t_a} \tag{2}$$

Considering the previous definitions, we calculate the ranking score for the virtual function (f_{rank}) as follows:

$$f_{rank} = \beta_1 \cdot delay(f) - (1-\beta_1) \cdot wait_{queue}(f) \tag{3}$$

where β_1 is an adjustable positive weight with values between 0 and 1.

To represent whether request $s_r \in S_R$ is still in service at time slot τ, we use the binary variable $a_{s_r,\tau}$ as follows:

$$a_{s_r,\tau} = \begin{cases} 1 & , t_s \le \tau < (t_s + t_r) \\ 0 & , \text{otherwise} \end{cases} \tag{4}$$

where t_s represents the starting time when the selected nodes start processing the service request.

Since we consider multiple instance deployment of VNFs in the same node, we must know the number of VNFs $f \in F$ placed on node $n \in N$ of cluster $k \in K$ at time slot τ. Thus, we utilize the variable $\eta^f_{k_n,\tau}$ to reflect this value and it is calculated as follows:

$$\eta^f_{k_n,\tau} = \sum_{\forall s_r \in S_R} \sum_{1 \le i \le |s_r|}^{f} x^i_{s_r,k_n} \cdot a_{s_r,\tau} \tag{5}$$

Similarly, we indicate when any VNF instance is placed, at time slot τ, on node $n \in N$ of cluster $k \in K$ through the binary variable $\nu_{k_n,\tau}$ as follows:

$$\nu_{k_n,\tau} = \begin{cases} 1 & , \sum_{\forall f \in F} \eta^f_{k_n,\tau} > 0 \\ 0 & , \sum_{\forall f \in F} \eta^f_{k_n,\tau} = 0 \end{cases} \tag{6}$$

After the previous specifications, we formally present our model defined as $\langle S, \mathcal{A}, \mathcal{R}, \gamma \rangle$, where S represents the set of discrete states, $\mathcal{A}$ is the set of discrete actions, $\mathcal{R}$ is the reward function, and $\gamma \in [0,1]$ is a discount factor for future rewards. Thus, the core elements used in the presented model are defined as follows.

State definition: the state $s \in S$ at time t (s^t) is represented as a vector consisting of the remaining resources and bandwidth of each node across all the clusters, and the requirements of the current VNF to be placed. Thus, state s^t is defined as follows:

$$s^t = (C^t, E^t, W^t, R^t),$$

where C^t defines the remaining resources of each node per cluster, therefore $C^t = (C^t_{1_1}, \dots, C^t_{1_{|n|}}, \dots, C^t_{|k|_{|n|}})$. In addition, the remaining SOC of each node per cluster is described as $E^t = (E^t_{1_1}, \dots, E^t_{1_{|n|}}, \dots, E^t_{|k|_{|n|}})$. The remaining bandwidth of each node per cluster is represented by $W^t = (W^t_{1_1}, \dots, W^t_{1_{|n|}}, \dots, W^t_{|k|_{|n|}})$. Finally, the requirements of the VNF to be scheduled are defined as $R^t = (D_i, W_{r_i}, t_{r_i}, d_{r_i}, PF_{r_i})$, where D_i is the resource demand on a node by the VNF, W_{r_i} represents the bandwidth demand, the running time of the service request is established by t_{r_i}, d_{r_i} defines the deadline for processing the given request, and PF_{r_i} is the amount of VNFs in $s_r \in S_R$ waiting to be deployed.

Action definition: we denote action $a \in \mathcal{A}$ as a binary vector. More specifically, each position in the vector corresponds to a possible action. When the first position is 1, it indicates that no node will be selected (i.e., $a^t = (1,0,0,0,\dots,0)$ do nothing). The value of 1 in one of the remaining positions infers the selected node (i.e., $a^t = (0,0,1,0,\dots,0)$).

Table 2
System model notation.

Notation	Description
K	Set of clusters
N	Set of physical nodes where events can be placed
P	Set of deployable units in the computing nodes
S_R	Set of network service requests arriving at controller
F	Sequence of VNFs compounding a network service request
k	Each cluster formed by a set of physical nodes (N)
n	Each physical node where virtual nodes are created
p	Each virtual node created on the physical node to run the events
$p_i^{k_n}$	Indicates the virtual node p_i is placed in node n of cluster k
s_r	Each network service request formed by a sequence of VNFs
f_i	Each VNF compounding a network service
C_{k_n}	Available resource capacity of node $n \in N$ of cluster $k \in K$ in terms of CPU and memory
E_{k_n}	Available energy capacity of node $n \in N$ of cluster $k \in K$ in terms of SOC
D_f	Resource demand of service instance $f \in F$ in terms of CPU and memory
W_{k_n}	Total output bandwidth of node $n \in N$ of cluster $k \in K$
W_r	Bandwidth requirement of service request $s_r \in S_R$
d_r	Deadline for processing a given request
t_r	Running time of a given request before considering it complete
t_a	Arrival time of a given request to controller
t_s	Starting time when the selected nodes process a given request
$x^i_{s_r,k_n}$	1 if the VNF f_i is successfully deployed on node $n \in N$ of cluster $k \in K$, 0 otherwise
$a_{s_r,\tau}$	1 if service request $s_r \in S_R$ is active in time slot $[t_s, t_s + t_r]$, 0 otherwise
$\eta^f_{k_n,\tau}$	Number of VNF instances $f \in F$ that are placed on node $n \in N$ of cluster $k \in K$ in time slot $[t_s, t_s + t_r]$
$v_{k_n,\tau}$	1 if any VNF instance is placed on node $n \in N$ of cluster $k \in K$ in time slot $[t_s, t_s + t_r]$, 0 otherwise
$z^{s_r}_{k_n,\tau}$	1 if any VNF of request $s_r \in S_R$ is placed on node $n \in N$ of cluster $k \in K$ in time slot $[t_s, t_s + t_r]$, 0 otherwise
y_{s_r}	1 if request $s_r \in S_R$ is deployed, 0 otherwise

Reward function: we define the reward function as a weighted sum of objectives that we want to achieve them jointly. The mathematical representation of this function is explained in the following subsection.

State transition: the state transition is defined as (s^t, a^t, r^t, s^{t+1}), where s^t is the current network state, a^t is the action taken (i.e., do nothing or place VNF) and s^{t+1} is the new network state after receiving the reward r^t.

3.3. Problem formulation

This section presents the mathematical formulation of the service deployment problem considering multiple clusters of nodes where virtual functions can be deployed.

In this regard, we consider that multiple VNFs belonging to different service requests can be placed at the same node in time slot τ while it has available resources. Such consideration is known as VNF consolidation (Panda et al., 2016; Qi et al., 2019b; Zhang et al., 2021), and it is represented as follow:

$$\sum_{\forall f \in F} \eta^f_{k_n,\tau} \cdot D_f \leq C_{k_n} \tag{7}$$

Additionally, we use $z^{s_r}_{k_n,\tau}$ to indicate that any VNFs of request $s_r \in S_R$ are placed in node $n \in N$ of cluster $k \in K$ at time slot τ. Then:

$$z^{s_r}_{k_n,\tau} = \begin{cases} 1 & , \sum_{i=1}^{|s_r|} x^i_{s_r,k_n} \cdot a_{s_r,\tau} > 0 \\ 0 & , \sum_{i=1}^{|s_r|} x^i_{s_r,k_n} \cdot a_{s_r,\tau} = 0 \end{cases} \tag{8}$$

Thus, we establish that the bandwidth demand of all requests passing through node $n \in N$ of cluster $k \in K$ cannot exceed its total output bandwidth, represented as follows:

$$\sum_{\forall s_r \in S_R} W_r \cdot z^{s_r}_{k_n,\tau} \leq W_{k_n} \tag{9}$$

Similar to our previous work (Llorens-Carrodeguas et al., 2021), we consider that surpassing the deadline of the service request would not lead to a service rejection since maintaining a required QoS is a desirable metric but not mandatory (Gazori et al., 2020). In this vein, we use a binary variable y_{s_r} to indicate whether s_r is deployed or not. Then:

$$y_{s_r} = \begin{cases} 1 & , \sum_{i=1}^{|s_r|} \sum_{n \in N} x^i_{s_r,k_n} = |s_r| \\ 0 & , \sum_{i=1}^{|s_r|} \sum_{n \in N} x^i_{s_r,k_n} < |s_r| \end{cases} \tag{10}$$

According to the previous considerations, we want to achieve several objectives. First, we want to minimize the resource cost of used nodes ($\mathbb{RC}$), which can be expressed as follows:

$$\mathbb{RC} = \sum_{k \in K} \sum_{n \in N} v_{k_n,\tau} \cdot (\beta_C C_{k_n} + \beta_W W_{k_n}), \tag{11}$$

s.t. (4), (5), (6), (7),

where β_C and β_W are the node resource and bandwidth unit costs, respectively.

Our second objective is to maximize the system's lifetime ($\mathbb{LT}$) by selecting nodes with appropriate levels of SOC to save as much energy as possible. This value can be obtained as follows,

$$\mathbb{LT} = \sum_{k \in K} \sum_{n \in N} E_{k_n}, \tag{12}$$

s.t. (1), (2), (4), (5), (6), (7), (8), (9).

Likewise, the third objective of the proposed model is to maximize the number of deployed services ($\mathbb{DS}$) to benefit the customer's requests, which can be expressed as follows:

$$\mathbb{DS} = \sum_{s_r \in S_R} y_{s_r}, \tag{13}$$

s.t. (1), (2), (4), (5), (6), (7), (8), (9).

After expressing the model's objectives, we define the reward function as a weighted sum of the resource costs, the lifetime of the system, and the number of deployed service requests as expressed in expression (14).

$$r^t(s^t, a^t) = \begin{cases} \zeta \cdot \mathbb{LT} + \xi \cdot \mathbb{DS} - \phi \cdot \mathbb{RC} & , s_{r_i} \text{ or } f_i \text{ is deployed} \\ 0 & , s_{r_i} \text{ or } f_i \text{ is rejected} \end{cases} \tag{14}$$

The adjustable positive weights $\zeta, \xi, \phi \in [0, 1]$ allow a trade-off between the different deployment decisions. Please notice that the terms to be maximized (i.e., $\mathbb{LT}$ and $\mathbb{DS}$) are expressed as positive terms, while the one to be minimized (i.e., $\mathbb{RC}$) is negative. In a nutshell, the reward function aims to increase the lifetime of the system and the number of deployed service requests while reducing the resource cost of used nodes.

4. DQN-based intelligent controller solution

In this section, we propose our deep Q-network (DQN)-based intelligent controller (DQNIC) to deploy virtual functions of a service request among several clusters. After receiving a multi-deployed service request from one of the assigned clusters, the proposed solution selects the best nodes among all the clusters by considering remaining battery estimations and CPU usage. Please notice that in Fig. 2, each cluster has a scheduler in charge of deploying local service requests

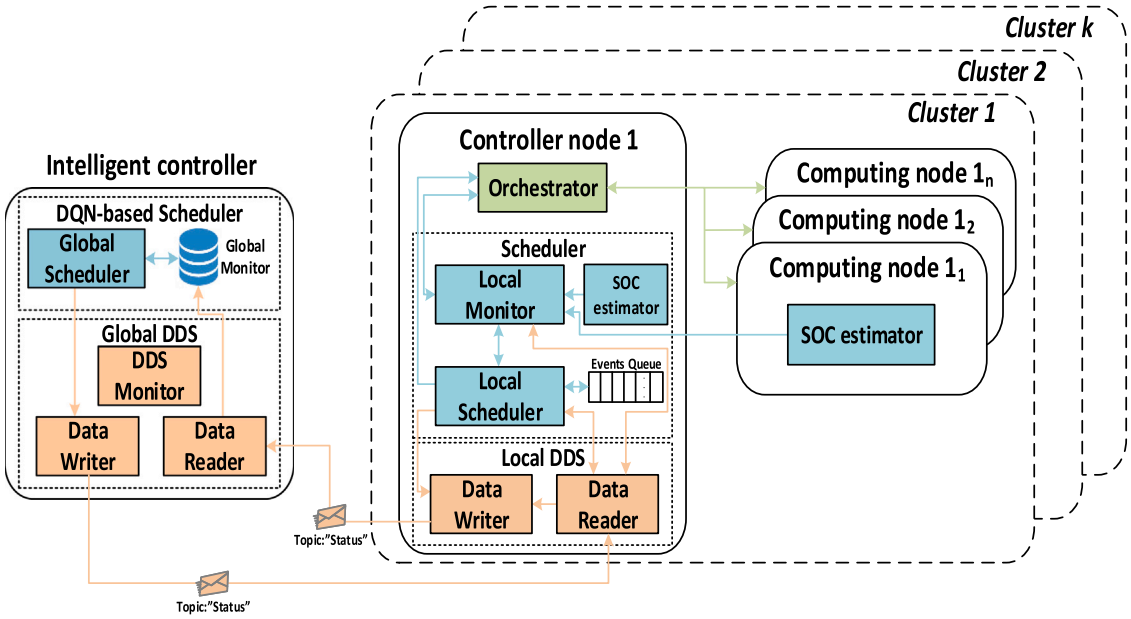


Fig. 2. Intelligent controller solution design and the relationships among its constituent modules.

by using the SOC and capacity-based scheduler (SOCCS) presented in Llorens-Carrodeguas et al. (2021).

Since the proposed solution needs the local schedulers' available information (e.g., CPU, memory, and SOC) to deploy services by considering a multi-cluster placement, we incorporate a mechanism in the local schedulers to guarantee their communication with the intelligent controller. Likewise, this mechanism is also proposed for the intelligent controller, and it is based on DDS.

Thus, our proposal is formed by five main elements: the global scheduler, the global monitor, the data writer, the data reader, and the DDS monitor. These modules have been grouped into two categories according to their functions: DQN-based scheduler and global DDS.

Fig. 2 depicts the relationships among the constituent modules of the proposed solution. More specifically, the orange line represents the DDS communication between global and local entities as well as their relation with the schedulers' blocks. Additionally, the blue line reflects the connection between the schedulers' elements and how the local schedulers communicate with the orchestrator. Finally, the green line shows the interaction between the orchestrator and the cluster nodes. The functions of DQNIC's blocks and local DDS are explained in the following subsections.

4.1. Local DDS

The local DDS implementation is composed of two blocks: *data reader* and *data writer*. The former reads the information sent by the DQNIC regarding the selected computing node where the controller must deploy the current VNF. The controller node of a cluster will place a VNF in its domain when it detects its name in the identifier field of the "Status" topic. This topic represents a data stream composed of seven fields where the identifier one is used to announce the kind of information that the *local data writer* will send. Thus, this element is responsible for publishing data regarding the nodes and service status. More specifically, it sends information about service requests, service and VNF deployments, and service rejections. Additionally, the *local data writer* periodically shares the remaining battery and CPU utilization of the cluster's nodes.

4.2. Global DDS

In contrast to the local DDS, the global implementation is formed by three blocks: *DDS monitor*, *data writer*, and *data reader*. The monitor automatically discovers compatible controller nodes according to the configured security profiles. Thus, we guarantee that the DQNIC only receives information from registered entities during the discovery phase (Llorens-Carrodeguas et al., 2019). Moreover, the *DDS monitor* tracks the status of the controller node in each cluster by either detecting a new participant or a failure one.

In the case of the *global data writer*, it is triggered by the *global scheduler* to send, through the "Status" topic, the selected node where the analyzed VNF must be placed. By indicating the controller node that manages the selected compute in the identifier field, it guarantees the reception of its decision only by the required master.

In correspondence with the information sent by the *local data writer*, the *global data reader* collects the nodes' status in terms of CPU and memory usage and SOC estimation. Additionally, it receives multi-deployment service requests and several placement states. Apart from gathering the mentioned information, the *global data reader* is responsible for storing it on the corresponding global monitor's buffer for further use during the DQN training.

4.3. Global monitor

This block represents a collection of storing data structures used by the different constituent elements of DQNIC. More specifically, the *global data reader* stores the received information related to the nodes' status and service requests in this element. Additionally, the *global scheduler* reads the stored data from the *global monitor* to create the input data forwarded through the DQN. Moreover, this block is used to save the state transitions (i.e., s^t, a^t, r^t, s^{t+1}), which are utilized afterward to train the DQN.

4.4. Global scheduler

This element represents the most crucial component in DQNIC. It determines the best node where a particular VNF can be placed according to predictions that consider the system's current state. Through these predictions, the *global scheduler* chooses what it considers the best action. This element improves its decisions by considering past

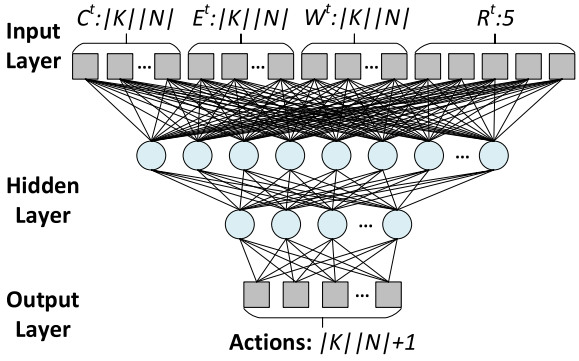


Fig. 3. Neural network design to estimate the Q-value function.

experiences and obtaining rewards from the system after each action. This behavior is due to the reinforcement learning algorithm running on this element.

A well-known reinforcement learning solution is Q-learning which selects actions according to the Q-values stored in its dimensional Q-table. This solution's main weakness is its lack of generality and scalability despite its powerful and straightforward capabilities (Zeng et al., 2019). Therefore, this solution is inapplicable in large-scale networks.

To overcome this limitation, the DQN introduces a neural network to estimate the Q-value function. The architecture of this network is formed by an input layer, an output layer, and several hidden layers, as is shown in Fig. 3. The input layer represents the state vector, and the output layer is the actions' probability distribution.

By considering a DQN model to estimate the possible actions that DQNIC can take, we implement the primary process of *global scheduler* as shown in Algorithm 1.

This algorithm runs simultaneously with the communication process managed by the global DDS. Thus, we avoid delays in the algorithm's execution due to the performance of other modules. The main function of Algorithm 1 is to select the best node to place a VNF request by considering an input state. To this aim, we create a DQN network according to the number of inputs (e.g., node's status and VNF request) and outputs (e.g., set of clusters' nodes in the system) in the system's model (line 1). Additionally, this step includes the possibility of loading a pre-trained model if existing. Later on, we initialize the DQN model's parameters with random weights (lines 2 and 3). Please notice that we use two DQN networks to make our training more stable since the values of $Q^t(s^t, a^t)$ and $Q^t(s^{t+1}, a^t)$ provided by the Bellman equation (Ding et al., 2020b) have only one step between them. Thus, it is difficult for a neural network to distinguish between them, which can alter the estimation values of nearby states after updating the network's parameters. Therefore, we introduce the so-called target Q-network to back-propagate its predicted Q-values and train the main Q-network with fixed weights to stabilize the computation of $maxQ(s^{t+1}, a)$ term in the Bellman equation. Following the algorithm's execution, we define the set of possible actions to take, which includes the existing nodes of the system plus the possibility of doing nothing (line 4). Moreover, this step comprises the initialization of the RL agent and the replay buffer where the system's transitions will be stored.

In line 5, we define the running condition of this algorithm by indicating the number of training steps we want to consider. During this process, the algorithm creates the input state for each interaction with the environment by considering the stored information in *global monitor* (line 6). To make the DQN network easier to train, we use the input data normalization method to format the input data into a small range (i.e., $[0, 1]$). For the computing resource inputs (e.g., CPU and SOC) in state s^t, we divide the remaining capacities of each node by a maximum $C_{max} = max(C_{k_n})$, $E_{max} = max(E_{k_n})$, $\forall k \in K, n \in N$.

Algorithm 1: DQNIC training process.

Input: Nodes' status and VNF request (see expression (15))
Output: Selected node to deploy the VNF request

1 Create DQN model according to the number of inputs and outputs or load a pre-trained model
2 Initialize action-value function Q with random weights Θ
3 Initialize target action-value function $\hat{Q}$ with weights $\Theta^- = \Theta$
4 Initialize action space ($|K||N|+1$), RL-agent and replay buffer (D) with a determined size
5 **while** *trainSteps* < *exploreSteps* **do**
6 Create input state (nodes' status and VNF request) using the stored information in *global monitor*
7 **if** *RL-agent's strategy is exploration* **then**
8 RL-agent selects a random action a^t from action space with probability ϵ
9 **else**
10 RL-agent selects action $a^t = max_a Q(s^t, a; \Theta)$
11 Trigger *global data writer* to indicate VNF's deployment to the selected node in action a^t
12 Wait for acknowledgment of VNF's deployment from selected node
13 Calculate reward r^t using expression (14)
14 Get next state (s^{t+1}) and store transition (s^t, a^t, r^t, s^{t+1}) in D
15 **if** *envSteps* > *observedSteps* **then**
16 Sample random mini-batch of transitions (s^j, a^j, r^j, s^{j+1}) from D
17 **forall** *Transitions in mini-batch* **do**
18 **if** *Episode terminates at step* $j+1$ **then**
19 Set $y^j = r^j$
20 **else**
21 Set $y^j = r^j + \gamma * max_{a'} \hat{Q}(s^{j+1}, a'; \Theta^-)$
22 Perform gradient decent step on $(y^j - Q(s^j, a^j; \Theta))^2$ with respect to the network parameters Θ
23 Every X steps reset $\hat{Q} = Q$

Similarly, we compress the bandwidth-related inputs and the rest of VNF's requirements. Thus, the normalized input state is:

$$\begin{aligned}
&(\frac{C^t_{1_1}}{C_{max}}, \dots, \frac{C^t_{1_{|n|}}}{C_{max}}, \dots, \frac{C^t_{|k|_{|n|}}}{C_{max}},\\
&\frac{E^t_{1_1}}{E_{max}}, \dots, \frac{E^t_{1_{|n|}}}{E_{max}}, \dots, \frac{E^t_{|k|_{|n|}}}{E_{max}},\\
&\frac{W^t_{1_1}}{W_{max}}, \dots, \frac{W^t_{1_{|n|}}}{W_{max}}, \dots, \frac{W^t_{|k|_{|n|}}}{W_{max}},\\
&\frac{D_i}{C_{max}}, \frac{W_{r_i}}{W_{max}}, \frac{t_{r_i}}{t_{r_{max}}}, \frac{d_{r_i}}{d_{r_{max}}}, \frac{PF_{r_i}}{PF_{r_{max}}})
\end{aligned} \tag{15}$$

After creating the input state, the RL agent selects an action according to its strategy. In line 7, we verify whether the agent is in exploration mode. If it is the case, it selects a random action with probability ϵ from the action space (line 8). Otherwise, the agent forwards the input state through the main neural network to obtain the Q-values for all possible actions and choose the best one (line 10). At this point, the algorithm triggers the *global data writer* to notify the deployment of the current VNF in the selected node (line 11). Then, the *global scheduler* waits for the acknowledgment of the VNF's deployment from the selected node (line 12). In line 13, we calculate the reward for the action taken using expression (14). Thus, we evaluate how good the algorithm's decision was. By obtaining the next state s^{t+1}, we complete the current state transition (s^t, a^t, r^t, s^{t+1}) and store it in the replay buffer (line 14). In line 15, the algorithm verifies if there are enough experiences in the replay buffer to perform a training step.

When sufficient state transitions exist, Algorithm 1 executes the training phase. This stage starts by sampling a random mini-batch of

transitions from the replay buffer (line 16). Then, for each element in the mini-batch (line 17), the algorithm checks whether the transition corresponds to the last step in the episode (line 18). If it is the case, the target state–action value for that step is similar to the step's obtained reward since there is no next state from which to gather the reward (line 19). For the other transitions (line 20), the next state–action value is calculated by using the target neural network and applying the discount factor γ for future rewards (line 21). The next step in the training process is to calculate the mean squared error loss between the next state–action value and the obtained Q-value using the main network (line 22). Additionally, this step includes updating the main neural network parameters by applying a gradient descent algorithm to minimize the loss. Finally, for every determined number of steps, the algorithm updates the target DQN network's weights with the ones in the main network to include previously learned experiences (line 23).

5. Evaluation and results

This section aims to evaluate the proposed solution by comparing it with different approaches, thus demonstrating its feasibility for deploying events in a multi-cluster environment.

5.1. Evaluation environment

To evaluate the performance of the proposed solution, we have built a testbed formed by three clusters of four SBC nodes. One of the clusters was deployed in the University Carlos III of Madrid (UC3M) while the remaining clusters were placed at the Universitat Politècnica de Catalunya (UPC). Fig. 4 depicts the described testbed. We have deployed Kubernetes 20.04 as the management framework for virtualized services. They run as Docker containers within pods and are placed into the devices. Thus, the deployed events utilize their available capacity according to predefined requirements. The intelligent controller was implemented using Java 1.8.0. In addition, we used the Deep Java Library 0.16.0 to implement the proposed DQN algorithm. As we mentioned before, the local scheduler of each cluster runs the SOCCS algorithm proposed in Llorens-Carrodeguas et al. (2021).

Testing equipment: the used SBC nodes were Raspberry Pi 4 Model B (Anon, 2022b) with 8 GB of RAM and an ARM64 processor with 4 cores. These nodes have one Gigabit Ethernet port, therefore the total output bandwidth (W_{k_n}) is 1 Gbps. The intelligent controller was run on an Ubuntu system (Intel i9-7900X CPU @ 3.30 GHz with 20 cores and 64 GB of RAM). We used the UM24C module (Anon, 2022c) to measure the power consumption of each node. It connects to Raspberry Pi devices via Bluetooth. The energy sources for the Pi devices are batteries with a capacity of 10,000 mAh.

Service requests: in our evaluation scenarios, the services to be scheduled arrive one at a time following a Poisson distribution. We explored different event arrival rates that range from 5 to 15 events per time unit. The main parameters used for creating the services were selected randomly from the list of values shown in Table 3 following a uniform distribution. The evaluation parameters were defined considering typical workloads derived from the literature.

Hyperparameters: the hyperparameters of the implemented DQNIC solution have been tuned for efficiency and stability. The parameters of the DQN model were initialized with the learning rate $\alpha = 0.01$ and the reward discount factor $\gamma = 0.9$. The former controls how quickly the weights of the neural network are updated in response to estimated error, meanwhile the latter represents how important future rewards are to the current state. The reward function's positive weights ζ, ξ, ϕ were set to 0.2, 0.5, and 0.3, respectively. The parameters related to the epsilon decay schedule (i.e., initial, decay, and final epsilon values) were 0.5, 0.002, and 0.01. We used a replay buffer with a storage capacity of 200 experiences and a batch size of 64 samples. The DQN networks, namely the online and target networks, were synchronized every 50 epochs. These networks were composed of three layers. The input layer with $3|K||N| + 5$ neurons where $|K|$ and $|N|$ were the number of clusters and nodes, respectively. We used one hidden layer with a number of neurons equal to the mean of the input and output layers (Heaton, 2008) and ReLU as the activation function. Finally, the output layer corresponded to the number of actions ($|K||N| + 1$). The weights of the online network were updated after finishing every epoch.

Table 3
Evaluation parameter ranges based on testbed.

Parameter	Values
Number of VNFs in a service	5–10
Processing capacity per node (MIPS)	500–3,000
CPU capacity per node (milli-CPU)	4,000
Memory capacity per node (Ki)	7,998,464
Required processing rate per event (MIPS)	100–500
Required CPU per event (milli-CPU)	150–250
Required memory per event (Ki)	200–500
Running time per event (S)	50–100
Deadline for processing an event (S)	50–100

Compared approaches: we compared the proposed solution with two algorithms, least loaded scheduler (LLS) and global SOC and capacity-based scheduler (GSOCCS).

- LLS — This mechanism allocates the events to the node with the highest available capacity. In this manner, the node with the least CPU usage is chosen. In this manner, it guarantees a balanced use of computational resources
- GSOCCS — This algorithm represents a global version of the solution presented in Llorens-Carrodeguas et al. (2021). In specific, it utilizes the information sent by the master of each cluster to calculate the nodes' score before selecting the best one to deploy a determined request. This score allows the algorithm to choose the node with the maximum SOC and minimum CPU usage

Please notice that we have not compared the proposed approach with other literature's algorithms because either no available implementation code was found or there was not enough data and details of the algorithms to reproduce them. In this regard, it would not be a fair comparison since we could not guarantee the original performance of these algorithms if we use them in our evaluation environment.

5.2. Training phase results

Before utilizing the proposed algorithm to deploy event requests in a multi-cluster environment, we must first train the agent using Algorithm 1. To illustrate the training process, the training model's loss and accuracy are depicted in Fig. 5.

The former represents the mean square error between the label and prediction values obtained by the target and online networks, respectively. It can be observed that the loss between both networks gradually decreases and converges to 0 while increasing the number of epochs, see Fig. 5(a). In contrast, Fig. 5(b) illustrates the behavior of the model's accuracy, which slowly increases with the number of epochs. This metric indicates the number of correct predictions to the total number of predictions.

Another crucial indicator of the training model is the reward obtained after taking every action since it represents how well the environment performed with the decision taken. Fig. 6 depicts the behavior of this metric while training the model.

More specifically, Fig. 6(a) shows the average reward on each epoch which gradually increases with the number of training periods. This performance evidences the quality of the trained model since it guarantees an increasing number of deployed events despite the depletion of available resources in the clusters' nodes.

The aforementioned description is better understood in Fig. 6(b). It illustrates the values of the reward function terms during the experiment. The light blue line represents the lifetime of the systems,

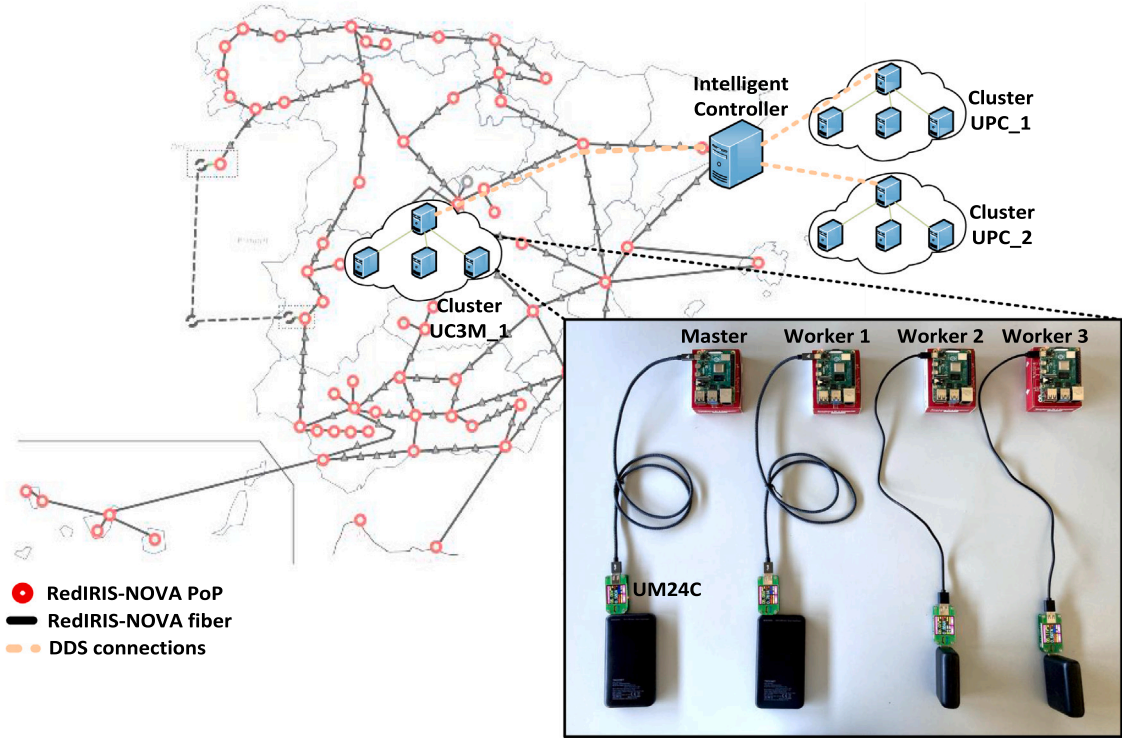
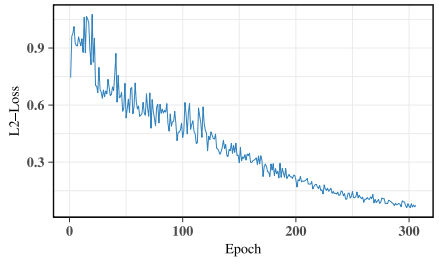
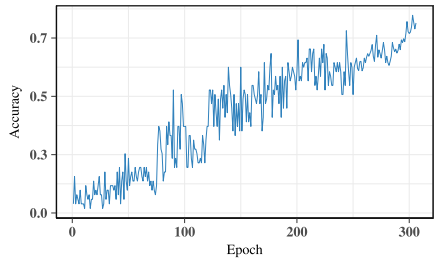


Fig. 4. Deployed testbed formed by three distributed SBC clusters and the DQN-based intelligent controller.

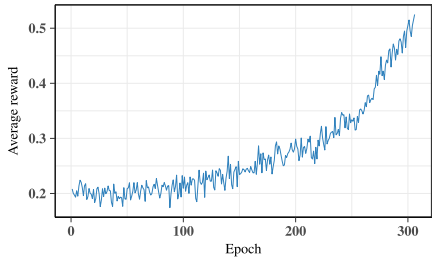


(a) Model's loss

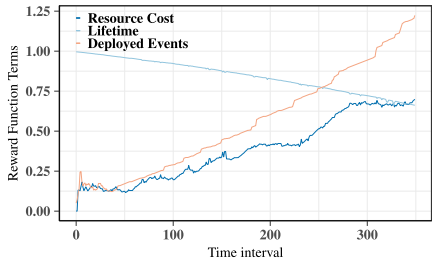


(b) Model's accuracy

Fig. 5. Training metrics of the model.



(a) Model's average reward



(b) Reward function terms

Fig. 6. Reward function behavior during the training phase.

which decreases due to battery depletion. The inclusion of this term in the reward function guarantees that the actions taken consider the remaining battery's node. Additionally, the dark blue line indicates the cost of the used computational resources. This value slightly increases since the model aims to re-utilize nodes that have already deployed events while having available resources, thus minimizing the cost. The primary term of the reward function is represented by the orange line and indicates the proportion of the deployed events against the requested ones. Its increasing behavior indicates the high acceptance ratio of the system while using the trained model.

During the training phase of the presented algorithm, we must guarantee the exploration of the environment by applying random actions with a determined probability. To this aim, we implemented an Epsilon-greedy policy with our agent. Fig. 7 depicts the calculated epsilon values during each environment step. The used epsilon's hyperparameters were indicated in Section 5.1. The showed behavior guaranteed random actions with higher probability at the beginning

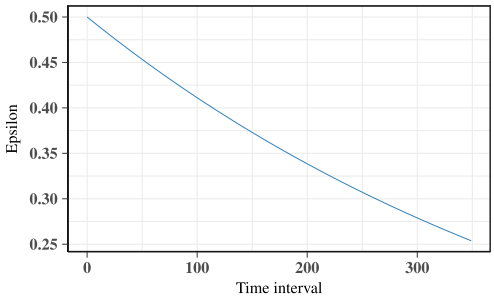


Fig. 7. Obtained values through the Epsilon-greedy policy.

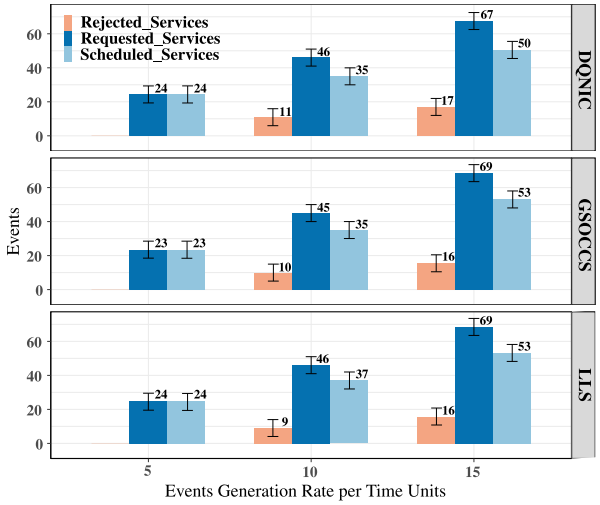


Fig. 8. Number of requested, scheduled, and rejected services for all the scheduling algorithms.

of the model training. Meanwhile, it was most likely to calculate the Q_{max} to make a decision in the rest of the experiment.

5.3. Comparison among scheduling approaches

This section evaluates the proposed algorithm in comparison with the approaches mentioned in Section 5.1. These algorithms were analyzed through the following metrics: scheduled and rejected events, acceptance ratio, resources cost, and battery consumption. We ran several experiments for each generation rate to ensure the reliability of the results. They show a confidence interval of 95% for all the analyzed methods.

5.3.1. Average number of scheduled and rejected events

Fig. 8 depicts the obtained results regarding requested, scheduled, and rejected services. We can notice that the analyzed algorithms achieved similar results for the explored generation rates. All the algorithms deployed the requested services without rejection for the lowest generation rate (i.e., 5 events per time unit). GSOCCS and LLS rejected fewer services than the proposed solution for 10 events per time unit (i.e., one and two services, respectively). For the remaining generation rate, DQNIC increased by around one service the rejected events with respect to the compared approaches. Overall, the results revealed that the difference between the three algorithms was not significantly high.

Fig. 9 illustrates a deeper insight into the constituent network functions. As we mentioned in Section 3.1, the generated services were constituted by a set of VNFs. By analyzing the rejected VNFs, we appreciate that this metric increased with the events generation rate for all the analyzed algorithms. Expressly, none VNFs were rejected for the

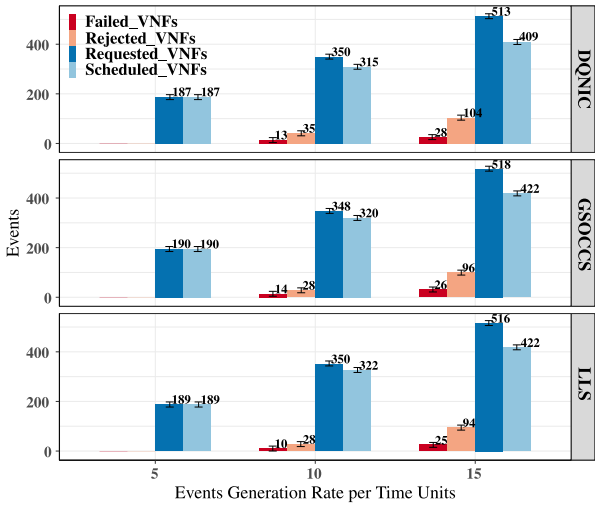


Fig. 9. Number of requested, scheduled, rejected, and failed VNFs for all the scheduling algorithms.

lowest generation rate. Meanwhile, this metric gradually increased for the other rates. Similar to the rejected services, our proposal deployed fewer VNFs than the compared approaches, but with a slight difference (i.e., around eight VNFs).

This figure also includes the number of failed VNFs (red bar) as a metric to indicate the number of wrong decisions each algorithm takes. More specifically, it represents the non-deployed VNFs due to insufficient resources in the selected node, thus leading to the rejection of the service and its constituent VNFs. By considering this metric, we indirectly evaluate the decision quality of the proposed algorithm since the number of failed VNFs is low for the evaluated generation rates.

5.3.2. Average acceptance ratio

We define the acceptance ratio as a quality metric of the algorithms that denotes the proportion between the number of scheduled and requested events. Fig. 10 depicts the average value of this metric for each generation rate. The results are in correspondence with the ones obtained in the above subsections where the proposed algorithm, i.e., DQNIC, had a lower acceptance ratio (i.e., 77% and 74% for 10 and 15 events per time unit, respectively) than the compared algorithms due to reject a small number of services. More specifically, the proposed solution decreased the acceptance ratio by around 3% and 5% with regard to GSOCCS and LLS, respectively, for 10 and 15 events per time unit. The three analyzed approaches have an acceptance ratio of 100% when deploying services with a generation rate of 5 events per time unit.

5.3.3. Average resource cost

The resource cost is a crucial criterion to be considered in cloud and edge environments since it directly impacts the CAPEX and OPEX. We defined it as the sum of the proportion between the used resources and their maximum values, multiplied by a unitary cost. Fig. 11 shows the average result of this metric for the analyzed allocation algorithms.

When analyzing the 5 events per time unit generation rate, the conceived proposal decreased by around 23% and 28% the used resources with respect to GSOCCS and LLS, respectively. Similarly, DQNIC outperformed the compared approaches with a reduction of 14% and 23% for a generation rate of 10 events per time unit. In the case of the highest analyzed generation rate, the proposed algorithm reduced the used resources by 8% and 22% in comparison with GSOCCS and LLS, respectively. These results evidenced that the proposed solution makes cost-effective use of the node resources compared to the studied algorithms since it aims to re-utilize as much as possible the same nodes to deploy new event requests.

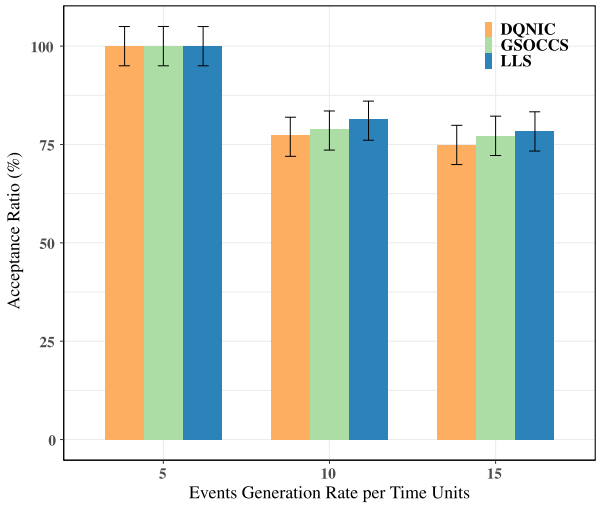


Fig. 10. Events acceptance ratio for each scheduling algorithm.

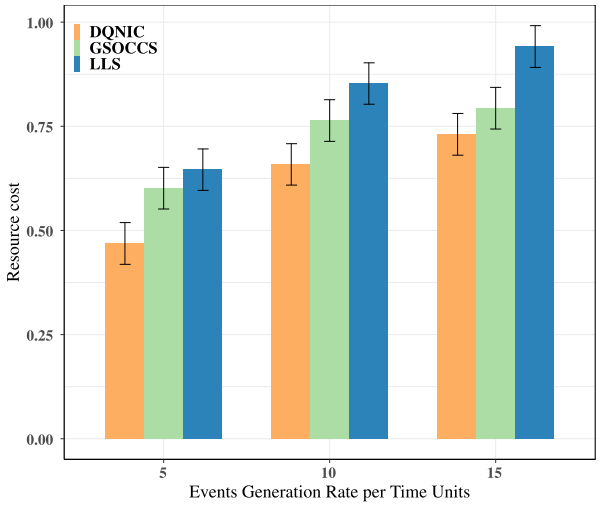


Fig. 11. Resource cost for each scheduling algorithm.

5.3.4. Average battery consumption

The battery consumption was calculated by considering the difference between each experiment's initial and final values of SOC. Fig. 12 illustrates the average battery consumption for each node in the three SBC clusters used when applying the studied allocation algorithms. To improve the readability of the figure, we grouped the nodes by considering their cluster's function. Thus, the dark colors represent the master of each cluster, while the light ones describe the worker nodes.

A general observation indicates that the battery consumption is higher while increasing the generation rates. This behavior was expected since the greater the event arrival rate, the higher the number of requested events. Thus, the clusters' operation time increases. By comparing the three schedulers, LLS had the highest battery consumption for all the generation rates since its balanced use of resources did not consider the nodes' SOC status, as it did GSOCCS. These algorithms achieved less worker consumption imbalance than our proposal because they balanced the event requests among all the clusters' workers. Nevertheless, DQNIC guaranteed the lowest battery consumption in the overall system since it aimed to reduce this metric by re-utilizing nodes to deploy events instead of using a different one.

A deep analysis of this figure revealed a similar battery consumption in the controller nodes of each cluster for the studied approaches. These nodes generally consumed less battery than the worker ones since they were only in charge of deploying the events according to the intelligent controller's decisions. Looking at the highest generation rate, DQNIC saved up to 20% and 26% of the worker's average consumption in cluster 1 (light green bars) with regard to GSOCCS and LLS, respectively. Similarly, the proposed approach reduced the worker's average consumption in cluster 2 (light blue bars) by around 24% and 34% when comparing the same algorithms. Regarding the cluster 3 worker's average consumption (orange bars), the proposed allocation mechanism diminished these values up to 30% and 32% with respect to GSOCCS and LLS, respectively.

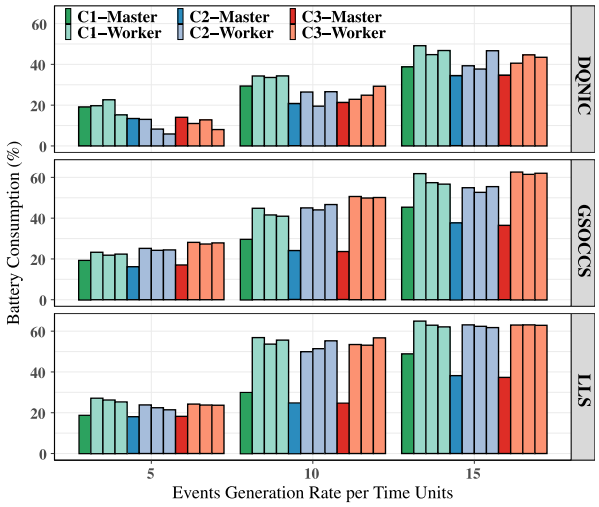


Fig. 12. Battery consumption for each cluster's nodes while running different algorithms.

Table 4
Numeric results of the evaluated metrics in the studied algorithms for the highest event generation rate.

Evaluated features	DQNIC	GSOCCS	LLS
Rejected services	17	16	16
Requested services	67	69	69
Scheduled services	50	53	53
Failed VNFs	28	26	25
Rejected VNFs	104	96	94
Requested VNFs	513	518	516
Scheduled VNFs	409	422	422
Acceptance ratio (%)	75	77	78
Resource cost	0.74	0.79	0.94

Table 5
Numeric results of nodes' battery consumption in the studied algorithms for the highest event generation rate.

Cluster node	DQNIC	GSOCCS	LLS
C1-Master	39	45	49
C1-Worker	47	59	64
C2-Master	35	38	38
C2-Worker	42	55	64
C3-Master	35	37	37
C3-Worker	42	60	62

To summarize the performance of the studied algorithms, Table 4 and 5 show the results of the evaluated metrics for the highest generation rate (i.e., 15 events per time unit). These results evidence that our proposed approach outperformed the others since it reduced the resource cost and the battery consumption while achieving high values of acceptance ratio.

6. Conclusion

This paper proposed DQNIC as a global allocation mechanism capable of deploying service requests in a multi-domain edge environment

by considering clusters' node status and service demands. The conceived approach implemented a DDS communication mechanism to exchange information with the clusters' controller nodes. Additionally, a DRL algorithm was integrated into the proposed solution to select the most suitable node where an event request must be deployed. The algorithm considered the nodes' status (i.e., CPU and memory utilization and battery consumption) and the event's demands to make its decisions.

To evaluate the feasibility of the presented approach, we built a testbed formed by twelve SBC nodes which were grouped into three clusters geographically distributed. In the first stage, we performed the training process of the proposed DRL algorithm to select the best set of hyperparameters that guarantee the expected behavior. Our proposal was compared with two baseline algorithms in the second phase. The results showcased a slight reduction in the number of deployed events (i.e., services and VNFs) when running the suggested solution with regard to the compared algorithms. Therefore, DQNIC had fewer values of acceptance ratio than the other algorithms, although the differences were insignificant.

In contrast, higher distinctions were noticed when analyzing the resource cost and battery consumption. More specifically, the proposed allocation mechanism reduced the resource cost by 23% and 28% with regard to GSOCCS and LLS, respectively, for the lowest generation rate. Similarly, it decreased this metric for the highest generation rate with values up to 8% and 22% when comparing it with the same approaches. In terms of battery consumption, the differences were even more noticeable. Concretely, DQNIC reduced this metric in the clusters' worker nodes up to values between 20% and 35% when comparing it with GSOCCS and LLS for the highest generation rate, thus increasing the lifetime of the overall system's nodes and, therefore, their resilience.

In terms of future work, we intend to provide our solution with a mechanism to migrate events and functionalities to other nodes. This strategy would improve the fault tolerance of the clusters since demanding events in critical nodes can be reassigned to available ones in other clusters before going down due to battery depletion. Additionally, an important parameter to consider in further research is latency because the end-to-end delay of service requests is crucial for future networks. In this sense, migration mechanisms must consider the latency among nodes to select the best target node without violating the service's end-to-end delay.

CRediT authorship contribution statement

Alejandro Llorens-Carrodeguas: Conceptualization, Formal analysis, Methodology, Data curation, Validation, Visualization, Software, Writing – original draft. **Cristina Cervelló-Pastor:** Conceptualization, Formal analysis, Validation, Methodology, Supervision, Writing – review & editing, Visualization, Funding acquisition, Project administration. **Francisco Valera:** Conceptualization, Formal analysis, Validation, Methodology, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

This work has been supported in part (50%) by the Agencia Estatal de Investigación of Ministerio de Ciencia e Innovación of Spain under projects PID2019-108713RB-C51 & PID2019-108713RB-C52 MCIN/AEI/10.13039/501100011033; and in part (50%) by AI@EDGE H2020-ICT-52-2020 under grant agreement No. 10101592.

References

Abu-Lebdeh, Mohammad, Naboulsi, Diala, Glitho, Roch, Tchouati, Constant Wette, 2017. On the placement of VNF managers in large-scale and distributed NFV systems. IEEE Trans. Netw. Serv. Manag. 14 (4), 875–889.

Álvarez, José Luis, Mozo, Juan Daniel, Durán, Eladio, 2021. Analysis of single board architectures integrating sensors technologies. Sensors 21 (18), 6303.

Anon, 2022a. Kubernetes. https://www.kubernetes.io/. (Available online accessed 28 November 2022).

Anon, 2022b. Raspberry Pi 4. https://www.raspberrypi.org/products/raspberry-pi-4-model-b/. (Available online accessed 28 November 2022).

Anon, 2022c. UM24C. https://www.mediafire.com/folder/0jt6xx2cyn7jt. (Available online accessed 28 November 2022).

Bari, Faizul, Chowdhury, Shihabur Rahman, Ahmed, Reaz, Boutaba, Raouf, Duarte, Otto Carlos Muniz Bandeira, 2016. Orchestrating virtualized network functions. IEEE Trans. Netw. Serv. Manag. 13 (4), 725–739.

Bello, Irwan, Pham, Hieu, Le, Quoc V., Norouzi, Mohammad, Bengio, Samy, 2016. Neural combinatorial optimization with reinforcement learning. arXiv preprint arXiv:1611.09940.

Bonomi, Flavio, Milito, Rodolfo, Zhu, Jiang, Addepalli, Sateesh, 2012. Fog computing and its role in the internet of things. In: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing. MCC '12, Association for Computing Machinery, New York, NY, USA, ISBN: 9781450315197, pp. 13–16. http://dx.doi.org/10.1145/2342509.2342513.

Bourhnane, Safae, Abid, Mohamed Riduan, Zine-dine, Khalid, Elkamoun, Najib, Benhaddou, Driss, 2021. Cluster of single-board computers at the edge for smart grids applications. Appl. Sci. 11 (22), 10981.

Bruschi, Roberto, Davoli, Franco, Lago, Paolo, Pajo, Jane Frances, 2019. A multi-clustering approach to scale distributed tenant networks for mobile edge computing. IEEE J. Sel. Areas Commun. 37 (3), 499–514.

Bruschi, Roberto, Lago, Paolo, Lamanna, Guerino, Lombardo, Chiara, Mangialardi, Sergio, 2016. Openvolcano: An open-source software platform for fog computing. In: 2016 28th International Teletraffic Congress, Vol. 2. ITC 28, IEEE, pp. 22–27.

Cohen, Rami, Lewin-Eytan, Liane, Naor, Joseph Seffi, Raz, Danny, 2015. Near optimal placement of virtual network functions. In: 2015 IEEE Conference on Computer Communications. INFOCOM, IEEE, pp. 1346–1354.

Ding, Ding, Fan, Xiaocong, Zhao, Yihuan, Kang, Kaixuan, Yin, Qian, Zeng, Jing, 2020a. Q-learning based dynamic task scheduling for energy-efficient cloud computing. Future Gener. Comput. Syst. 108, 361–371.

Ding, Zihan, Huang, Yanhua, Yuan, Hang, Dong, Hao, 2020b. Introduction to reinforcement learning. In: Deep Reinforcement Learning. Springer, pp. 47–123.

Fkih, Fethi, Omri, Mohamed Nazih, 2016. IRAFCA: an O (n) information retrieval algorithm based on formal concept analysis. Knowl. Inf. Syst. 48 (2), 465–491.

Gazori, Pegah, Rahbari, Dadmehr, Nickray, Mohsen, 2020. Saving time and cost on the scheduling of fog-based IoT applications using deep reinforcement learning approach. Future Gener. Comput. Syst. 110, 1098–1115.

Heaton, Jeff, 2008. Introduction to Neural Networks with Java. Heaton Research, Inc.

Javed, Asad, Heljanko, Keijo, Buda, Andrea, Främling, Kary, 2018. CEFIoT: A fault-tolerant IoT architecture for edge and cloud. In: 2018 IEEE 4th World Forum on Internet of Things. WF-IoT, IEEE, pp. 813–818.

Javed, Asad, Robert, Jérémy, Heljanko, Keijo, Främling, Kary, 2020. IoTEF: A federated edge-cloud architecture for fault-tolerant IoT applications. J. Grid Comput. 18 (1), 57–80.

Jayanetti, Amanda, Halgamuge, Saman, Buyya, Rajkumar, 2022. Deep reinforcement learning for energy and time optimized scheduling of precedence-constrained tasks in edge–cloud computing environments. Future Gener. Comput. Syst. 137, 14–30.

Khemili, Wided, Hajlaoui, Jalel Eddine, Omri, Mohamed Nazih, 2022. Energy aware fuzzy approach for VNF placement and consolidation in cloud data centers. J. Netw. Syst. Manage. 30 (3), 1–29.

Litvinchev, Igor, Ozuna, Edith Lucero, 2013. Lagrangian heuristic for the facility location problem. IFAC Proc. Vol. (ISSN: 1474-6670) 46 (24), 107–113. http://dx.doi.org/10.3182/20130911-3-BR-3021.00022, URL https://www.sciencedirect.com/science/article/pii/S1474667016321735.

Llorens-Carrodeguas, Alejandro, Cervelló-Pastor, Cristina, Leyva-Pupo, Irian, 2019. A data distribution service in a hierarchical sdn architecture: Implementation and evaluation. In: 2019 28th International Conference on Computer Communication and Networks. ICCCN, IEEE, pp. 1–9.

Llorens-Carrodeguas, Alejandro, G. Sagkriotis, Stefanos, Cervelló-Pastor, Cristina, P. Pezaros, Dimitrios, 2021. An energy-friendly scheduler for edge computing systems. Sensors 21 (21), 7151.

Mahmood, Aamir, Beltramelli, Luca, Abedin, Sarder Fakhrul, Zeb, Shah, Mowla, Nishat I., Hassan, Syed Ali, Sisinni, Emiliano, Gidlund, Mikael, 2021. Industrial IoT in 5G-and-beyond networks: Vision, architecture, and design trends. IEEE Trans. Ind. Inform. 18 (6), 4122–4137.

Mu, Yanyan, Wang, Lei, Zhao, Jin, 2021. Energy-efficient and interference-aware vnf placement with deep reinforcement learning. In: 2021 IFIP Networking Conference. IFIP Networking, IEEE, pp. 1–9.

Nogales, Borja, Vidal, Iván, Sanchez-Aguero, Victor, Valera, Francisco, Gonzalez, Luis, Azcorra, Arturo, 2020. OSM PoC 10 automated deployment of an IP telephony service on UAVs using OSM. (Available online accessed 28 November 2022).

Panda, Aurojit, Han, Sangjin, Jang, Keon, Walls, Melvin, Ratnasamy, Sylvia, Shenker, Scott, 2016. {NetBricks}: Taking the V out of {NFV}. In: 12th USENIX Symposium on Operating Systems Design and Implementation. OSDI 16, pp. 203–216.

Pegasus, 2022. Pegasus: Makes the work flow. https://pegasus.isi.edu/. (Available online accessed 28 November 2022).

Pei, Jianing, Hong, Peilin, Pan, Miao, Liu, Jiangqing, Zhou, Jingsong, 2019. Optimal VNF placement via deep reinforcement learning in SDN/NFV-enabled networks. IEEE J. Sel. Areas Commun. 38 (2), 263–278.

Qi, Dandan, Shen, Subin, Wang, Guanghui, 2019a. Towards an efficient VNF placement in network function virtualization. Comput. Commun. 138, 81–89.

Qi, Dandan, Shen, Subin, Wang, Guanghui, 2019b. Virtualized network function consolidation based on multiple status characteristics. IEEE Access 7, 59665–59679.

Santos, Guto Leoni, Lynn, Theo, Kelner, Judith, Endo, Patricia Takako, 2021. Availability-aware and energy-aware dynamic SFC placement using reinforcement learning. J. Supercomput. 77 (11), 12711–12740.

Schulte, Christian, Lagerkvist, Mikael, Tack, Guido, 2022. Gecode. https://www.gecode.org. (Available online accessed 28 November 2022).

Slamnik-Kriještorac, Nina, Kremo, Haris, Ruffini, Marco, Marquez-Barja, Johann M., 2020. Sharing distributed and heterogeneous resources toward end-to-end 5G networks: a comprehensive survey and a taxonomy. IEEE Commun. Surv. Tutor. 22 (3), 1592–1628.

Smith, Christopher Peter, Jindal, Anshul, Chadha, Mohak, Gerndt, Michael, Benedict, Shajulin, 2022. Fado: Faas functions and data orchestrator for multiple serverless edge-cloud clusters. In: 2022 IEEE 6th International Conference on Fog and Edge Computing. ICFEC, IEEE, pp. 17–25.

Solozabal, Ruben, Ceberio, Josu, Sanchoyerto, Aitor, Zabala, Luis, Blanco, Bego, Liberal, Fidel, 2019. Virtual network function placement optimization with deep reinforcement learning. IEEE J. Sel. Areas Commun. 38 (2), 292–303.

Tipantuña, Christian, Hesselbach, Xavier, Sánchez-Aguero, Victor, Valera, Francisco, Vidal, Ivan, Nogales, Borja, 2019. An NFV-based energy scheduling algorithm for a 5G enabled fleet of programmable unmanned aerial vehicles. Wirel. Commun. Mob. Comput. 2019.

Upton, Eben, Halfacree, Gareth, 2016. Raspberry Pi User Guide. John Wiley & Sons.

Varasteh, Amir, Madiwalar, Basavaraj, Van Bemten, Amaury, Kellerer, Wolfgang, Mas-Machuca, Carmen, 2021. Holu: Power-aware and delay-constrained VNF placement and chaining. IEEE Trans. Netw. Serv. Manag. 18 (2), 1524–1539.

Wahab, Omar Abdel, Kara, Nadjia, Edstrom, Claes, Lemieux, Yves, 2019. MAPLE: A machine learning approach for efficient placement and adjustment of virtual network functions. J. Netw. Comput. Appl. 142, 37–50.

Xia, Xuewen, Gui, Ling, Zhan, Zhi-Hui, 2018. A multi-swarm particle swarm optimization algorithm based on dynamical topology and purposeful detecting. Appl. Soft Comput. 67, 126–140.

Zeng, Deze, Gu, Lin, Pan, Shengli, Cai, Jingjing, Guo, Song, 2019. Resource management at the network edge: A deep reinforcement learning approach. IEEE Netw. 33 (3), 26–33.

Zhang, Qixia, Liu, Fangming, Zeng, Chaobing, 2021. Online adaptive interference-aware VNF deployment and migration for 5G network slice. IEEE/ACM Trans. Netw. 29 (5), 2115–2128.

Zhanikeev, Marat, 2015. A cloud visitation platform to facilitate cloud federation and fog computing. Computer 48 (05), 80–83.

Alejandro Llorens-Carrodeguas received his BSc. degree in Telecommunication and Electronic Engineering from the Technological University of Havana José Antonio Echeverría (Havana, Cuba). He also received his Ph.D. in Network Engineering from the Universitat Politècnica de Catalunya (UPC). Experienced as an operation and management system engineer, his main research interests include NFV, SDN, data distribution service, 5G networking, OpenStack, and Kubernetes.

Cristina Cervelló-Pastor received her MSc. and Ph.D. degrees in Telecommunication Engineering, both from the Barcelona School of Telecommunications Engineering (ETSETB), Universitat Politècnica de Catalunya (UPC), Barcelona, Spain. She is an Associate Professor and has been the Head of the Network Engineering Department at UPC from 2012 to 2020. From 2021 she is the Director of the Castelldefels School of Telecommunications and Aerospace Engineering (EETAC). She is currently the leader of the research group on Broadband Network Modeling and Evaluation (BAMPLA). She has been responsible and actively participated in diverse national and European competitive projects (IA@EDGE, 5GCity, 5G, NOVI, FEDERICA, ATDMA, A@DAN, Euro-NGI, Euro-FGI, EURO-NF, etc.) and private funding R\&D projects. In parallel, she has published more than 100 papers in national and international journals and conferences and she has been supervising several theses in the field of optimization, management, optimal resource allocation, topology discovery, AI, and routing in SDN/NFV and 5G.

Francisco Valera received the degree in telecommunication engineering from the Technical University of Madrid (UPM), in 1998, and the Ph.D. degree in telecommunications from the University Carlos III de Madrid (UC3M), in 2002. He is currently a Tenured Associate Professor and deputy Director of the Telematics Engineering Department with UC3M. He has been involved in several national and international research projects and contracts related with experimental facilities, protocol design, inter-domain routing, protocol engineering, next generation networks and multimedia systems, serving there as PI, work package leader and also as coordinator. Some of the recent research projects in which he has participated, are: TRUE5G, 5GCity, H2020 FISHY, H2020 ZORRO, H2020 Labyrinth, IST Trilogy, IST LEONE, etc. He has published over 100 articles in the field of advanced communications in magazines and conferences.