VERA: Resource Orchestration for Virtualized Services at the Edge

Sharda Tripathi*, Corrado Puligheddu†, Somreeta Pramanik†, Andres Garcia-Saavedra‡, Carla Fabiana Chiasserini†
*Birla Institute of Technology and Science Pilani, †Politecnico di Torino, ‡NEC Laboratories Europe

ICC 2022 - IEEE International Conference on Communications | 978-1-5386-8347-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICC45855.2022.9838935

***Abstract*—The combination of service virtualization and edge computing allows mobile users to enjoy low latency services, while keeping data storage and processing local. However, the network edge has limited resource availability, and when both virtualized user applications and network functions need to be supported concurrently, a natural conflict in resource usage arises. In this paper, we focus on computing and radio resources and develop a framework for resource orchestration at the edge that leverages a model-free reinforcement learning approach and a Pareto analysis, which is proved to make fair and efficient decisions. Through our testbed, we demonstrate the effectiveness of our solution in resource-limited scenarios, and show an improvement of around 60% in the CPU budget violation rate with respect to RL based standard multi-agent framework.**

***Index Terms*—Virtual RAN, virtualized services, resource orchestration, machine learning, experimental testbed**

I. Introduction

Network Function Virtualization (NFV) and edge computing are disrupting the way mobile services can be offered through mobile network infrastructure. Third parties such as vertical industries and over-the-top players can now partner up with mobile operators to reach directly their customers and deliver a plethora of services with reduced latency and bandwidth consumption. Video streaming, gaming, virtual reality, safety services for connected vehicles, and IoT are all services that can benefit from the combination of NFV and edge computing: when implemented through virtual machines or containers in servers co-located with base stations, they can enjoy low latency and jitter, while storing and processing data locally. One such use case considering the livecast and vRAN services is presented in Fig. 1.

The combination of NFV, edge computing, and an efficient radio interface, e.g., O-RAN, is therefore a powerful means to offer mobile services with high quality of experience (QoE). However, some important aspects have been overlooked. For instance, user applications are not the only ones that can be virtualized: network services such as data radio transmission and reception are nowadays virtualized and implemented through Virtual Network Functions (VNFs) as well; and both types of virtual services, user's and network's, may be highly computationally intensive. Besides, the amount of data each service has to process is entangled [1], in the sense that a correlation exists between the amount of data processed/generated by virtual applications at the edge and

This work has been supported by the EC H2020 5GPPP 5GROWTH project (Grant No. 856709)

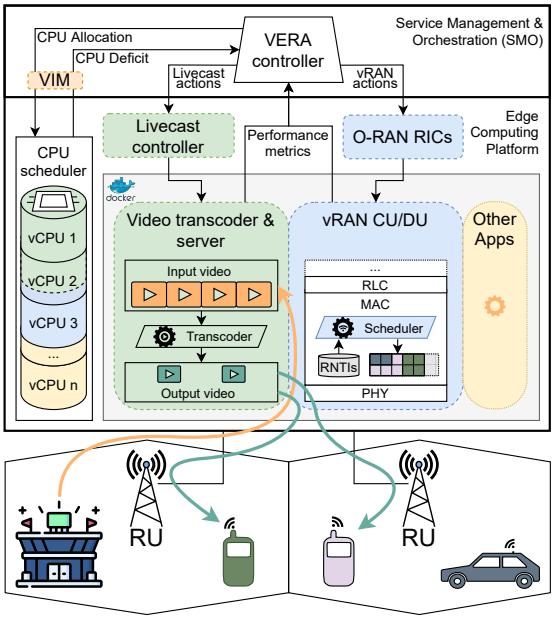


Fig. 1: Virtualized user application and vRAN at the edge: system scenario and reference use case

network services VNFs, and such correlation can be positive or negative depending on the type of involved VNFs. Also, since computational availability at the network edge is limited [2], *user applications and network services have to compete for resources, hence designing automated and efficient resource orchestration mechanisms in the case of resource scarcity is critical.* To withstand these challenges, we design a flexible framework, called VERA (Virtualized Edge for Radio and user Applications), leveraging a model-free reinforcement learning (RL) approach.

Several works have addressed the VNF placement problem at the network edge, which is related but orthogonal to the problem we face. Recent examples include: [3], which minimizes the user-perceived latency and system cost; [4], which optimizes both service placement and traffic routing under different resource constraints; and [5], which uses cooperation among edge nodes for service caching and workload scheduling. Other studies have focused on QoE provisioning to mobile users through edge-assisted solutions. In particular, [6] presents an RL framework for crowdcasting services at the edge meeting bit rate as well as streaming and channel switching latency requirements, while minimizing the

overall computing and bandwidth cost. [7], instead, designs and implements an edge network orchestrator, and a server assignment and frame resolution selection algorithm for best latency-accuracy trade-off in mobile augmented reality. RL-based schemes can also be found in [8], to minimize latency and packet drop rate in 5G systems. Related joint resource allocation problems have been addressed before [1], [9], albeit ignoring the complex relationship between all system parameters and context variables and, therefore, making simplifying assumptions that do not work in practice. Moreover, extending these approaches to multi-service scenario will increase the system complexity manifolds.

We underline that, unlike previous works, we address the allocation of edge resources *constrained to a limited budget across different, competing, virtual services.* To this end, we propose a distributed multi-agent learning approach, wherein not only the actions of individual agents must collectively satisfy the hard capacity constraints of mobile edge platforms, some notion of fairness is also required when enforcing these constraints. Consequently, we introduce a novel Pareto component that guarantees a fair Pareto-efficient solution.

In summary, we provide the following novel contributions:

(i) We propose VERA, an RL framework for an effective, joint allocation of computing and radio resources for user applications and vRAN at the edge;

(ii) To enhance the system flexibility, we leverage distributed learning agents, as well as a novel Pareto analysis for a fair, efficient decision making, in the case of constrained CPU utilization;

(iii) A proof-of-concept is provided by designing a containerised edge and an srsRAN-based testbed implementing the proposed VERA framework;

(iv) We show the excellent performance of the VERA framework in terms of convergence as well as its ability to closely meet the target KPIs of all services in resource-constraint scenarios, and also against an independent multi-agent RL solution that we use as benchmark.

We remark that, to our knowledge, we are the first to address the allocation of a *common pool of edge resources to different, competing, virtualized services* through distributed learning, and to tackle the non-trivial correlations existing among the behaviors of such services in a scalable manner. Moreover, not only VERA can swiftly adapt to time-varying network conditions and application traffic, but it also controls the settings of both user applications and vRAN, selecting at each decision step a fair Pareto-efficient solution.

II. Reference Scenario and Testbed Architecture

The system architecture and reference scenario under study are illustrated in Fig. 1. For the sake of clarity, we focus on one user application and one virtual base station (vBS), implemented within an edge computing platform. As sample use case, we consider a livecast service representing a live video recording of an event occurring, e.g., at a stadium, that is broadcast to mobile users located therein or in the nearby area. The high-quality source video is processed within an edge computing platform through a standard video transcoder.

In addition to the livecast application (as well as, possibly, other user applications running at the edge), the edge computing platform hosts vBS functions, central unit (CU) and/or distributed unit (DU), which are jointly controlled by the VERA controller. As depicted by Fig. 1, the VERA controller is deployed in the Service Management & Orchestration (SMO) platform, and interacts with both O-RAN intelligent controllers (RIC) to configure the vBS functions, the edge service controllers (in this case the livecast controller), and the NFV virtual infrastructure manager (VIM) to configure the CPU schedulers. In this way, VERA's workflows (data collection and decision making) are fully compliant with O-RAN's machine learning procedures. Indeed, VERA continuously monitors the state of the vRAN and the livecast application (hereinafter also referred to as *services*), as well as the overall usage of computing resources in the edge platform. Then, it uses such observations to compute the values of the operating parameters for both livecast and vRAN, which, given the available computing and networking resources, meet both the application and vRAN KPI targets.

The system architecture has been recreated in a smaller scale in our testbed for the development and testing of the VERA framework. The main components are the edge computing platform, and the user equipment (UE), communicating by means of an LTE radio link implemented using the srsRAN suite [10]. The edge platform runs two Docker containers implementing, respectively, the livecast and the vRAN service, which consume the edge resource pool. The radio and livecast services are connected to VERA through a dedicated API, used to dynamically set radio and livecast operating parameters and retrieve performance measurements. VERA also interacts with the edge computing platform operating system and the Docker daemon to monitor and allocate computing resources to the services. The edge platform and the UE are hosted on Linux machines with Intel i7-7700HQ and i7-8550U CPUs, respectively, and with 16 GB of DDR4 memory.

III. The VERA Framework

The VERA framework is designed using a model-free RL approach. It includes distributed learning agents, each corresponding to a service in the edge platform, which simultaneously make decisions for the allocation of radio and computing resources as well as tune service-specific operating parameters with periodicity equal to $N \geq 1$ monitoring slots. These decisions are collectively referred to as a resource allocation policy, which consists of two development stages. Firstly, each RL agent makes decisions based on the shared context representation to obtain a greedy resource allocation policy. Subsequently, greedy policies from all RL agents are collated and further refined in view of the feasibility of the chosen actions to obtain a Pareto-efficient fair resource allocation. The individual stages are elaborated in the sequel. The structure of the VERA framework is shown in Fig. 2.

A. Greedy analysis

We consider a context vector comprising variables pertinent to each service. The context vector is processed through an autoencoder to create a shared context representation that captures the correlation among context variables, as well as reduces the dimensionality of the context vector. Then, each RL agent devises a greedy resource allocation policy by using the same shared context representation and by mapping it to an action vector such that its long-term cumulative reward from the environment is maximized. The elements composing the greedy resource allocation policy are introduced below.

Context space. The resource allocation for the livecast service is governed by the following contextual information: input bit rate (b), input video frame-per-second (FPS) rate (f), and input image resolution (v) of the streaming video. Besides, to accommodate any backlog in video processing, the normalized CPU throttled time of the livecast application (t_v) in the previous monitoring slot is considered. Likewise, resource allocation for the vRAN is based on normalized CPU throttled time (t_r), the 3GPP-compliant Channel Quality Indicator (CQI) (γ) reported from UE to vBS, which is representative of the signal quality, and the traffic from the livecast application sent over the radio link, specified by the network load (l). Thus, the context vector observed in monitoring slot n ($n = 1, \ldots, N$) can be written as $\mathbf{x}^{(n)} \in \mathcal{X}, \mathbf{x}^{(n)} := \{b^{(n)}, f^{(n)}, v^{(n)}, {t_v}^{(n)}, {t_r}^{(n)}, \gamma^{(n)}, l^{(n)}\}$. Further, to extract the correlation between context variables, an autoencoder projects context vector $\mathbf{x}^{(n)} \in \mathcal{X}$ onto its latent representation $\mathbf{y}^{(n)} \in \mathbb{R}^D, \mathbf{y}^{(n)} := \{{\mathbf{y}_1}^{(n)}, \ldots, \mathbf{y}_D^{(n)}\}$ where $D < \dim(\mathcal{X})$. The latent representation $\mathbf{y}^{(n)}$ is shared with each RL agent so that its decision process for a given service is informed of the performance of others accessing the resource pool, thus representing a shared context representation.

Action space. Since services are heterogeneous, we define $\mathcal{A} := \{\mathbf{a}_k\}, \forall k \in (1, \ldots, K)$, comprising action vectors, each having service-specific action variables. In our reference scenario, $K = 2$, and we associate $k = 1, 2$, respectively, to action vectors for livecast and vRAN. Thus, $\mathbf{a}_1$ comprises the CPU allocated to the livecast application (c_v), the video output encoding bitrate (β), and the video output encoding FPS (φ), while $\mathbf{a}_2$ includes the CPU allocated to vRAN (c_r), and the modulation and coding scheme (MCS) value (ω). Since CPU allocation is a capacity-constrained resource, i.e., $c_v + c_r \leq B_c$, where B_c is the total available CPU budget, we replace c_v and c_r with a generic notation $\mathbf{c}_k$ that denotes the capacity-constrained resource allocated to service k. Mathematically,

$$\mathbf{a}_k = \begin{cases} (\beta, \varphi, \mathbf{c}_k), & \text{if } \ k = 1\,, \\ (\omega, \mathbf{c}_k), & \text{if } k = 2\,. \end{cases}$$

Next, we discretize the quantity of capacity-constrained resources that can be allocated, and map each feasible combination of action variables in monitoring slot n into an action index $\mathbf{a}_1^{(n)} := \{1, 2, \ldots, N_\beta \cdot N_\varphi \cdot N_c\}$ and $\mathbf{a}_2^{(n)} := \{1, 2, \ldots, N_\omega \cdot N_c\}$, where N_i is the number of elements in the discretized version of action variable i, ($i \in \{\beta, \varphi, \omega, c\}$).

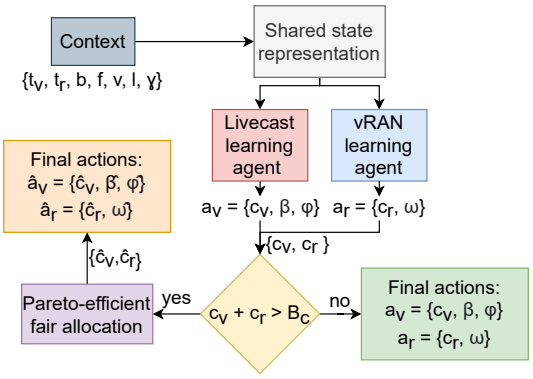


Fig. 2: Structure of the VERA framework

This allows us to limit the action space to a subset of discrete positive values with low cardinality, and facilitates simultaneous selection of several resources with a single action.

Reward. We now define observed KPIs, so as to measure the goodness of our actions. Specifically, for the livecast application, we use the Weighted Video Multimethod Assessment Fusion (WVMAF) (ζ_o) and the buffer state of the client's player (σ_o), which is a good estimator of the user's QoE [11]. The buffer state reports information on the occupancy of the MAC layer buffer, while the WMAF is a custom metric based on the well-known VMAF score, which provides a value between 0 (worst) and 100 (best) *per video frame*. Because VMAF assesses the quality of individual video frames only, it is not helpful to measure the *smoothness* of a video, which is well known to impact the perceived quality. To weigh this in, we multiply the measured VMAF of each frame by the ratio between the output frame rate and the input frame rate, and we refer to this as WVMAF. Concerning the vRAN, we consider the latency measured within the RAN (λ_o) and the packet loss rate (μ_o). Then, for every KPI, we denote their target values by ζ_t, σ_t, λ_t, and μ_t, respectively.

The reward value r is then given by the sum of the components pertaining to each service-specific KPI k in the n-th monitoring slot within the same decision window, as:

$$r(\mathbf{y}^{(n)}, \mathbf{a}_k^{(n)}) = \begin{cases} r_\zeta(\mathbf{y}^{(n)}, \mathbf{a}_k^{(n)}) + r_\sigma(\mathbf{y}^{(n)}, \mathbf{a}_k^{(n)}), & \text{if } \ k = 1\,, \\ r_\lambda(\mathbf{y}^{(n)}, \mathbf{a}_k^{(n)}) + r_\mu(\mathbf{y}^{(n)}, \mathbf{a}_k^{(n)}), & \text{if } k = 2\,. \end{cases}$$

In the above expressions, $r_\zeta(\cdot), r_\sigma(\cdot)$ are the reward components from WVMAF and buffer state (resp.), given by:

$$r_{\text{KPI}}(\mathbf{y}^{(n)}, \mathbf{a}_k^{(n)}) = \begin{cases} 1 - \text{erf}(\text{KPI}_o(\mathbf{y}^{(n)}, \mathbf{a}_k^{(n)}) - \text{KPI}_t), \\ \qquad\qquad\qquad\qquad \text{if KPI is met} \\ \text{erf}(\text{KPI}_o(\mathbf{y}^{(n)}, \mathbf{a}_k^{(n)}) - \text{KPI}_t), \qquad \text{else}\,. \end{cases}$$

The terms $r_\lambda(\cdot)$ and $r_\mu(\cdot)$ are instead the reward components from latency and packet loss rate (resp.), which are given by similar expressions but with $(\text{KPI}_t - \text{KPI}_o(\mathbf{y}^{(n)}, \mathbf{a}_k^{(n)}))$ as an argument of the erf function, since all values of latency and packet loss rate lower than their respective target values are acceptable. Since all values of the erf function lie in $[-1, +1]$, we have: $-2 \leq r(\mathbf{y}^{(n)}, \mathbf{a}_k^{(n)}) \leq 2$. Importantly, meeting the target KPI for each service is not enough, rather, it is essential to keep the observed KPIs as close as possible to their

Algorithm 1: Fair Pareto-efficient CPU allocation

$S = \{\tilde{a}_k\}_k, \mathbf{c}_k \leftarrow \tilde{a}_k(S), S_1 = \{\mathbf{c}_k\}_k, \forall k \in \mathcal{C}$
/* Extract CPU from greedy actions $\tilde{a}_k$ */
if $\sum_{k\in\mathcal{C}} \mathbf{c}_k \leq B_c$ /* Capacity-constraint check */
then
  $S^\star = S1$
  /* Output: Fair Pareto-efficient solution */
else
  $\mathcal{S}_e = \{S_1, S_2, \ldots\}$
  /* Build expanded solution set */
  $\mathcal{S}_s \leftarrow \{S_i/|\mathcal{C}|\}_{\mathcal{S}_e}$
  /* Rescale expanded solution set */
  for $S \in \mathcal{S}_s$ **do**
    $\widehat{\mathcal{S}}_s \leftarrow \{\widehat{a}_k(S)\}$
  /* Define refined actions set wrt $\mathcal{S}_s$ */
  Create $\mathcal{S}_d$
  /* Pareto dominant solution set */
  Choose $S^\star$
  /* Output: Fair Pareto-efficient solution */

target values. Failing that, the system may perform better than required at the cost of extra resource consumption. In light of this, erf is the best choice for estimating individual reward components.

We consider a generic decision window h and, we let $a_k^{(h-1)}$ be the action for the k-th service selected in decision window $(h-1)$ and applied in decision window h. We then define the average reward over h as, $\overline{r}(\mathbf{y}^{(h)}, \mathbf{a}_k^{(h-1)}) := \frac{1}{N}\sum_{n=1}^{N} r(\mathbf{y}^{(n)}, \mathbf{a}_k^{(h-1)})$, where $\mathbf{y}^{(h)}$ is the vector of shared contexts observed in the N monitoring slots in decision window h, while $\mathbf{a}_k^{(h-1)}$ is the action for service k selected and applied in decision window $h-1$ and h, respectively.

Action-value estimation and action selection. At the end of each decision window, actions need to be evaluated and the best one has to be selected. To this end, we compute the mean shared context over the N monitoring slots in h as, $\overline{\mathbf{y}}^{(h)} = \sum_{n=1}^{N} z_n \mathbf{y}^{(n)} / \sum_{n=1}^{N} z_n$, where $z_n > 0$ and $z_N > z_{N-1} > \cdots > z_1$ are the weights assigned so that the latest shared context has the highest weight. We then quantify the goodness of taking an action using action values. For service k, the value of $\mathbf{a}_k^{(h)}$ given policy π_k, which is $q_{\pi_k}(\overline{\mathbf{y}}^{(h)}, \mathbf{a}_k^{(h)})$, is defined as the expected differential return conditioned on $\overline{\mathbf{y}}^{(h)}$ and $\mathbf{a}_k^{(h)}$, following policy π_k, i.e., $q_{\pi_k}(\mathbf{y}, a) = \mathbb{E}_{\pi_k}[\mathbf{G}_k^{(h)} | \overline{\mathbf{y}}^{(h)} = \mathbf{y}, \mathbf{a}_k^{(h)} = a]$. Since the context space $\mathcal{X}$ is real, we use a practical method for action-value estimation using function approximation in an F-dimensional space, yielding the approximated function $\hat{q}_{\pi_k}(\overline{\mathbf{y}}^{(h)}, \mathbf{a}_k^{(h)}, w) = \sum_{f=1}^{F} w_f s_f(\overline{\mathbf{y}}^{(h)}, \mathbf{a}_k^{(h)})$, where $\mathbf{w}$ and $\mathbf{s}(\overline{\mathbf{y}}^{(h)}, \mathbf{a}_k^{(h)})$ denote the F-size weight and feature vectors (resp.), with the latter being generated using tile coding [12].

The estimation of the action values is followed by an ϵ-greedy action selection policy [13], with $\epsilon = 0.5$ and ϵ-decay factor $= 0.999$. This favors higher exploration in the initial phase, while it allows for exploitation of the gained environment knowledge with progression of time, so as to maximize the expected return.

B. Pareto analysis

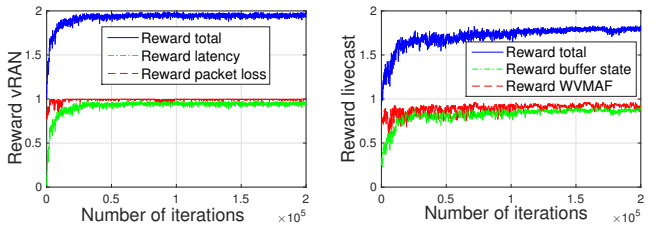
We recall that the CPU allocation for service k is a capacity-constrained resource. Hence, it is essential that the sum of CPU allocated to different services does not exceed the available computing resource budget and that the selected actions can be enacted. To this end, we introduce an algorithm that works on the multi-dimensional actions selected by the ϵ-greedy policy in the RL framework introduced above, and it further refines them so that the resulting actions (i) meet the budget constraint and (ii) entail fair Pareto-efficient resource sharing.

Specifically, given a decision window, we first formulate the fair Pareto-efficient allocation of CPU across the services as a constrained multi-criteria optimization problem. Let $\mathcal{C}$ be the set of services; given a set of coefficients $u_k \geq 0$, $k \in \mathcal{C}$, with $\sum_{k\in\mathcal{C}} u_k = 1$, it is required to find a solution $S^\star = \{\mathbf{c}_k^\star\}$, $k \in \mathcal{C}$, that maximizes $\sum_{k\in\mathcal{C}} \mathbf{u}_k \mathbf{\Gamma}_k^{(n)}(S)$ such that $S \in \mathcal{S}_c$ and $\sum_{k\in\mathcal{C}} \mathbf{c}_k \leq B_c$. Here, $\mathcal{S}_c$ is the set of feasible CPU allocations and $\mathbf{\Gamma}_k^{(n)}(S)$ is the criteria function denoting the reward of service k in monitoring slot n following strategy S.

The optimization problem is solved using Alg. 1, which initially considers the CPU allocation to the services provided by the greedy resource allocation policy in S, and creates the expanded CPU allocation solution set $\mathcal{S}_e$ by considering all possible values for the $\mathbf{c}_k$'s that are greater than those output by the greedy policy and whose sum does not exceed $|\mathcal{C}|$ times the available budget. Such values are then scaled by $|\mathcal{C}|$, to get candidate allocation values that meet the CPU budget. The corresponding action set, $\widehat{\mathcal{S}}_s$, is built starting from such $\mathbf{c}_k$'s and possibly refining the actions so that their components take feasible values. Consequently, $\widehat{\mathcal{S}}_s \leftarrow \{\widehat{a}_k(S)\}$, s.t. $\mathbf{c}_k(S)$ in $\widehat{a}_k(S)$ and other components of $\widehat{a}_k(S)$ fully match, $\forall k \in \mathcal{C}$. Such actions, $\{\widehat{\mathbf{a}}_k(S)\}$, $S \in \widehat{S}_s$, are then used to compute the values of $\mathbf{\Gamma}_k^{(n)}(S)$ to identify the Pareto-dominant solution set through iterative search and update, $\mathcal{S}_d \leftarrow \{S\}$, s.t. $\forall S \in \mathcal{S}_d$, $\forall S' \in \widehat{\mathcal{S}}_s, \mathbf{\Gamma}_i(S) > \mathbf{\Gamma}_i(S'), \mathbf{\Gamma}_j(S) \geq \mathbf{\Gamma}_j(S'), \forall i, j \in \mathcal{C}, i \neq j$. Finally, the Pareto-dominant solution that maximizes the minimum value of criterion function over all the services is chosen as the fair Pareto-efficient solution.

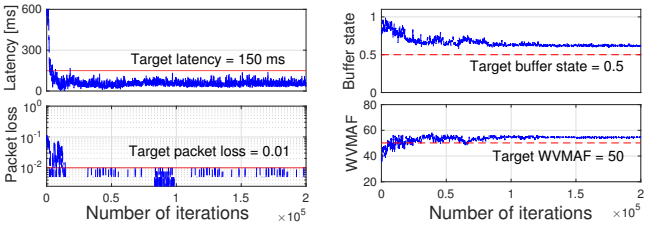
As shown in the Appendix, we prove that: *(i)* Given coefficients u_k's satisfying the aforementioned conditions, solution $S^\star$ is Pareto-efficient (Prop. 1), *(ii)* Alg. 1 converges to a Pareto-efficient solution set at a sub-linear rate (Prop. 2), *(iii)* Alg. 1 converges to a solution that is fair with respect to each of the multiple criteria $\mathbf{\Gamma}_k, k \in \mathcal{C}$ (Prop. 3), thus leading to a fair Pareto-efficient solution.

Over successive iterations, action values are updated using differential semi-gradient SARSA, an on-policy learning algorithm [13]. Hence, best actions corresponding to a given context are identified. To summarize, the main steps in the learning algorithm are: (i) obtain a greedy resource allocation policy for service k through estimation of action values $q_{\pi_k}(y, a)$, (ii) obtain a fair Pareto-efficient resource allocation policy by collating greedy policies of all the services, and (iii) update action-value estimates. Note that, even if the RL agents in (i) are non-greedy, the significance of Pareto block holds;



(a) (b)

Fig. 3: Convergence of reward values: vRAN (a) and livecast (b) services



(a) (b)

Fig. 4: KPI evolution with respect to iterations: (a) vRAN and (b) livecast services

TABLE I: Comparison between VERA and standard IRL

CPU budget	3 vCPUs	2 vCPUs	1 vCPU
IRL [8] violation rate	12.73%	71.54%	100%
VERA violation rate	0%	0%	0%

e.g., CPU orchestration in vRAN is addressed using the actor-critic approach in [9]. Though the maximum CPU capacity constraint is met in the output CPU scheduling policy, multiple solutions are possible owing to the continuous nature of the action set, and hence the need to choose the Pareto-efficient fair solution still remains. Hence, the design of Pareto block is valid irrespective of the learning approach used by the RL agents.

IV. Performance Evaluation

Convergence evaluation. To analyze the performance of the VERA framework, first we discuss its convergence. Fig. 3 depicts the time evolution of reward values for the vRAN and livecast services. Despite the large heterogeneous action set and the diverse context vector, the reward corresponding to each of the KPIs, and hence the total reward, saturates close to the maximum reward value, thereby highlighting the efficient learning capability of the VERA framework. Also, note that the convergence of the livecast service is relatively slower with respect to vRAN owing to its slowly varying dynamics, i.e., since the context variables pertinent to livecast change relatively slowly compared to that of vRAN, it takes longer to explore all the actions, hence slower convergence.

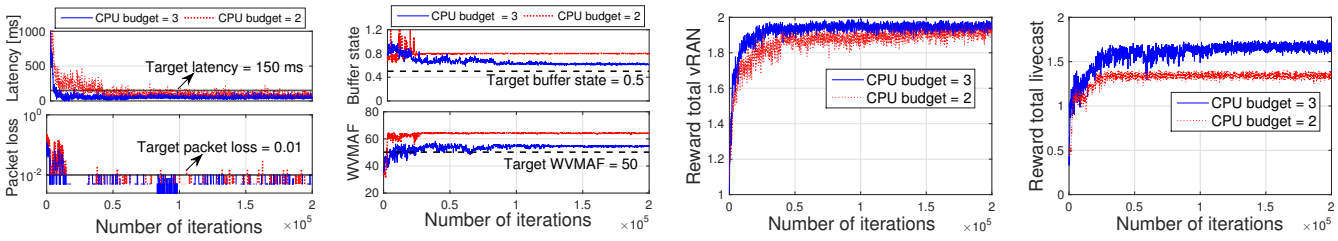
KPI performance. Next, Fig. 4 presents the evolution of KPIs across iterations during the learning process. We observe that KPI satisfaction for vRAN is achieved when its KPIs latency and packet loss do not exceed their respective targets. On the contrary, for the livecast service, the KPIs buffer state and WVMAF should not fall below their target values, while keeping the observed KPIs as close as possible to the target values for both the services. As per the 3GPP 5G specifications and acceptable user's QoE, the target values are set to 150 ms, 0.01, 0.5 s and 50 (resp.) for latency, packet loss, buffer state, and WVMAF. From the plots, we note that barring a few initial iterations during which the algorithm is still learning, the VERA's choice of actions leads to KPI satisfaction for both vRAN and livecast services.

Varying the CPU constraints. We now evaluate the impact that different CPU capacity constraints have on VERA's performance. To this end, we compare two different values of CPU budget (namely, 2 and 3 vCPUs, respectively) in Fig. 5(a) and (b). The difference in performance in the two cases is mild, thus confirming that VERA can successfully meet the target KPI values even in the presence of a reduced CPU budget. Further, for a 3vCPU budget, VERA matches the goal, not just of satisfying, but also of approaching the KPI theresholds better than for 2vCPUs, since a larger set of feasible actions, hence a wider choice, is available. This is confirmed by the reward evolution in Fig. 5(c) and (d), highlighting that the reward is always higher for larger CPU budget.

Comparison with other approaches. A remaining question is the role that the Pareto block has in the VERA decision process. To shed light on this aspect, we compare VERA to an RL-based radio resource management technique [8] which uses differential semi-gradient SARSA. For the sake of fairness, we extend its action set to include discrete CPU values, and use it to make predictions for the livecast and vRAN actions individually using the same context vector. Thus, it acts as a standard independent multi-agent RL (IRL) framework. Tab. I compares the percentage of cases in which a resource allocation decision violates the system's CPU budget when we use VERA and when we use IRL, respectively. While VERA satisfies the system constraint in all cases, IRL incurs in substantial violations, which grow as the constraint becomes tighter. For instance, it is worth noticing that for a budget equal to 2 vCPUs, IRL incurs over 70% of decisions violating the system capacity. We observe from Tab. I that, on average, across varying CPU constraints, VERA improved by around 60% the CPU budget violation rate with respect to IRL. In this respect, VERA's Pareto block plays a key role in providing fair Pareto-efficient allocations.

V. Conclusions

We considered an edge computing platform hosting virtualized user applications and network services (namely, vRAN) competing for the same resources, and developped a distributed learning framework, called VERA, that sets the configuration of both types of services so that the target KPIs can be met in spite of the limited availability of computing resources at the edge. VERA also exploits a Pareto analysis that leads to fair Pareto-efficient decisions. Importantly, as a proof-of-concept, we designed a containerised edge and an srsRAN-based testbed implementing the VERA framework. Through experimental results, we showed the feasibility of



(a) (b) (c) (d)

Fig. 5: Comparison for different CPU budgets: KPI and reward evolution for vRAN (a,c) and livecast (b,d)

our approach and its excellent performance in the presence of capacity-constrained resources where standard multi-agent solution fall into resource capacity violations, e.g., over 70% with a budget of 2 vCPUs in our testbed.

As a future work, we are investigating the scalability of VERA in multi-service, multi-user vRAN scenarios, thereby increasing the scenario heterogeneity and dimensionality, and considering multiple resource constrained variables.

APPENDIX

Proposition 1. *Given a set of coefficients $u_k \geq 0, k \in \mathcal{C}$, such that, $\sum_{k\in\mathcal{C}} u_k = 1$, then the solution $S^\star = \{c^{\star(k)}\}$, $k \in \mathcal{C}$, that maximizes the multi-criteria optimization problem $\sum_{k\in\mathcal{C}} u_k \mathbf{\Gamma}_k^{(n)}(S)$, is Pareto-efficient.*

Proof. For brevity, we provide just a sketch of the proof. In particular, consider that (i) the reward function f is strictly increasing on each component of the set $(\mathbf{\Gamma}_1(S), \ldots, \mathbf{\Gamma}_k(S)), \forall S \in \widehat{\mathcal{S}}_s$, and (ii) the weights u_k are positive, thus any Pareto improved solution S' that dominates $S^\star$ would increase f. Consequently, the result can be easily proved by contradiction. □

Proposition 2. *Alg. 1 converges to a Pareto-efficient solution set at a sub-linear rate.*

Proof. The multi-criteria optimization problem has a set of feasible solutions, $\widehat{\mathcal{S}}_s$, having cardinality $|\widehat{\mathcal{S}}_s|$. Let S be a feasible solution, then $S = \{c_k\}, k \in \mathcal{C}$ s.t. $\sum_{k\in\mathcal{C}} c_k \leq B_c$. The solution update proposed in Alg. 1 resembles an iterative multi-objective search and update algorithm [14]. Thus, based on Sec. 4.3 and Thm. 2 in [14], convergence is ensured.

Further, solutions are feasible only if they adhere to $\sum_{k\in\mathcal{C}} c_k \leq B_c$, which reduces the possible number of solutions, i.e., the iterations over $\widehat{\mathcal{S}}_s$. Thus, the problem complexity is $\mathcal{O}(|\widehat{\mathcal{S}}_s|)$.

Let S_t be the solution at iteration t, and the Pareto-efficient solution $S^\star$ be achieved at $t^\star$. From Alg. 1, at any t, a solution S is added to $\mathcal{S}_d$ only if it dominates the existing solutions. When S dominates another solution S', then by definition there is a Pareto improvement. Thus, over successive iterations, if $f(\mathbf{\Gamma}_1(S), \cdots, \mathbf{\Gamma}_k(S)) = \sum_{k\in\mathcal{C}} u_k \mathbf{\Gamma}_k(S)$, where $K = |\mathcal{C}|$, then, $f^\star(\mathbf{\Gamma}_1(S^\star), \cdots, \mathbf{\Gamma}_k(S^\star)) \geq f_{t+1}(\mathbf{\Gamma}_1(S_{t+1}), \cdots, \mathbf{\Gamma}_k(S_{t+1})) \geq f_t(\mathbf{\Gamma}_1(S_t), \cdots, \mathbf{\Gamma}_k(S_t))$.It follows that $f^\star - f_{(t+1)} \leq f^\star - f_t$, $\frac{f^\star - f_{(t+1)}}{f^\star - f_t} \leq 1$. Since $\frac{f^\star - f_{(t+1)}}{f^\star - f_t} \leq 1$ for $1 \leq t < t^\star \leq |\widehat{\mathcal{S}}_s|$, it implies that $f_{(t+1)}$ is closer to $f^\star$ compared to f_t, thus the algorithm is converging at sub-linear rate. □

Proposition 3. *The solution obtained using Alg. 1 is fair with respect to each of the multiple criteria $\mathbf{\Gamma}_k, k \in \mathcal{C}$.*

Proof. The result can be easily proven by showing that solution S^* is fair with respect to each criterion, by using contradiction. □

REFERENCES

[1] A. Garcia-Saavedra, G. Iosifidis, X. Costa-Perez, and D. J. Leith, "Joint optimization of edge computing architectures and radio access networks," *IEEE JSAC*, vol. 36, no. 11, pp. 2433–2443, 2018.

[2] Nokia, "The edge cloud: An agile foundation to support advanced new services," *White Paper*, 2018.

[3] S. Pasteris, S. Wang, M. Herbster, and T. He, "Service placement with provable guarantees in heterogeneous edge computing systems," in *IEEE INFOCOM*, 2019, pp. 514–522.

[4] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Joint service placement and request routing in multi-cell mobile edge computing networks," in *IEEE INFOCOM*, 2019, pp. 10–18.

[5] X. Ma, A. Zhou, S. Zhang, and S. Wang, "Cooperative service caching and workload scheduling in mobile edge computing," in *IEEE INFOCOM*, 2020, pp. 2076–2085.

[6] F. Wang, C. Zhang, F. wang, J. Liu, Y. Zhu, H. Pang, and L. Sun, "Intelligent edge-assisted crowdcast with deep reinforcement learning for personalized qoe," in *IEEE INFOCOM*, 2019, pp. 910–918.

[7] Q. Liu, S. Huang, J. Opadere, and T. Han, "An edge network orchestrator for mobile augmented reality," in *IEEE INFOCOM*, 2018, pp. 756–764.

[8] S. Tripathi, C. Puligheddu, and C. Chiasserini, "An RL Approach to Radio Resource Management in Heterogeneous Virtual RANs," in *IEEE/IFIP WONS*, 2021.

[9] J. A. Ayala-Romero, A. Garcia-Saavedra, M. Gramaglia, X. Costa-Perez, A. Banchs, and J. J. Alcaraz, "vrain: Deep learning based orchestration for computing and radio resources in vrans," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2020.

[10] I. Gomez-Miguelez, A. Garcia-Saavedra, P. D. Sutton, P. Serrano, C. Cano, and D. J. Leith, "srslte: An open-source platform for lte evolution and experimentation," in *Proceedings of the Tenth ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation, and Characterization*, 2016, pp. 25–32.

[11] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, "A buffer-based approach to rate adaptation: Evidence from a large video streaming service," in *Proceedings of the 2014 ACM conference on SIGCOMM*, 2014, pp. 187–198.

[12] A. A. Sherstov and P. Stone, "Function approximation via tile coding: Automating parameter choice," in *Abstraction, Reformulation and Approximation*, J.-D. Zucker and L. Saitta, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 194–205.

[13] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.

[14] M. Laumanns, L. Thiele, K. Deb, and E. Zitzler, "Combining convergence and diversity in evolutionary multiobjective optimization," *Evolutionary Computation*, vol. 10, no. 3, pp. 263–282, 2002.