PERX: Energy-aware O-RAN Service Orchestration with Pairwise Performance Profiling

Arshad Javeed, Viktoria Fodor, György Dán
Division of Network and Systems Engineering, KTH Royal Institute of Technology, Sweden
{ajaveed,vfodor,gyuri}@kth.se

***Abstract*—Motivated by the potential of machine-learning-based (ML) algorithms for radio access network (RAN) control and management, we consider the problem of energy-aware O-RAN service orchestration subject to ML inference time constraints. While ML applications enable complex operations in RAN control, guaranteeing service level agreements to close RAN operations in real time is a key requirement to facilitating their wider adoption. In this paper, we focus on orchestrating ML/AI workloads as near-real-time applications in O-RAN Cloud (O-Cloud). We propose *PERX*, an energy-efficient and performance-aware O-RAN orchestrator that predicts the performance of diverse sets of colocated ML/AL applications by learning a pairwise characterization of application inference times via hierarchical Bayesian learning. We formulate a latency-constrained integer optimization problem for application orchestration and propose an iterative procedure to solve the problem. In line with industry standards, we adopt Kubernetes as the orchestration framework to develop a latency-aware O-Cloud orchestrator. Experimental results reveal up to 50 % increase in profit with guaranteed service level agreements, compared to state of the art benchmarks.**

***Index Terms*—O-RAN, Service orchestration, Performance profiling**

I. Introduction

New generations of mobile networks bring the promise of serving a large variety of networked applications, all with different, often challenging performance requirements [4], [14]. Meeting the requirements of these applications efficiently requires intelligent and flexible radio access network resource management and control [16].

The key approach to achieve this flexibility is the deployment of open radio access networks (O-RANs) [13], controlled remotely by virtualized network functions, implemented in a cloud or edge computing environment, forming a so-called O-Cloud. Applications implemented in the O-Cloud can include real-time (RT) and near-real-time (Near-RT) applications with inference loops less than 10 ms and 10 ms to 1 s, respectively, and provide intelligent RAN control capabilities, such as network slicing, pricing, traffic classification, and anomaly detection, among others [13]. With the advancement of artificial intelligence, many of these applications are expected to be data-driven, using machine learning (ML) models [15].

The O-Cloud approach allows network operators (tenants) to share computing resources, and it also allows the dynamic allocation and de-allocation of application instances, potentially increasing compute resource utilization and decreasing energy consumption. Applications deployed on the same server, however, contend for the same computing and memory resources and affect the inference time of each other [9]. Therefore, placing applications so that their inference deadline is met while the resources are highly utilized is a challenge.

Existing cloud orchestration frameworks such as Kubernetes and OpenShift are ineffective in meeting the latency-constrained service requirements in O-Cloud [18], necessitating an orchestration framework that orchestrates the application instances considering their latency requirements as well as the effect of resource contention [3], [8].

In this work, we propose *PERX*, a performance-aware and energy-efficient, low-complexity service orchestrator. The key component of *PERX* is the effective characterization of resource contention among different types of applications to aid in scaling and placing application instances. Our main contributions are as follows:

- Based on extensive experiments, we propose a tractable model of application inference times under pairwise resource contention. Based on this, we construct an inference time model for an arbitrary set of applications.
- By leveraging a hierarchical Bayesian learning framework, we estimate the parameters of the inference time of ML models from limited datasets.
- We propose an energy-aware utility function for application placement and formulate an integer optimization problem for application orchestration under latency constraints. We find a near-optimal solution through an iterative approach that scales well with the system size.
- We perform extensive experiments by emulating ML/AI workloads on Kubernetes clusters. Our results show that *PERX* outperforms state-of-the-art latency-aware orchestration frameworks in terms of operator revenue and observes reduced SLA violations.

The rest of the paper is organized as follows. In Section II, we introduce the system model. In Section III, we propose an application inference time model. We formulate the latency-constrained application orchestration problem in Section IV and propose an iterative solution approach in Section V. We show numerical results in Section VI. In Section VII, we review the related work, and we conclude the paper in Section VIII.

II. System Model

We consider a set $\mathcal{S}$ of servers dedicated to execute data-driven O-RAN applications from an application catalog $\mathcal{A}$.

The available memory of server $s \in \mathcal{S}$ is C_s. We denote by C_a the memory requirement of an instance of the application $a \in \mathcal{A}$. Network tenants may request services by specifying the application type, $a \in \mathcal{A}$, the expected task arrival rate λ_a^r, the number n_a^r of application instances to be deployed, the 95^{th} percentile of the inference time τ^r, and the price ρ^r they are willing to pay. A tenant request can thus be characterized by the tuple $r = (a, n_a^r, \rho^r, \tau^r, \lambda_a^r)$, and we denote the set of tenant requests by $\mathcal{R}$. Aligned with industry standards (Kubernetes, OpenShift), we make the reasonable assumption that the orchestrator uses load-balancing for the application instances deployed for the same tenant request, i.e., each instance receives tasks at rate λ_a^r/n_a^r.

Note that tenant requests specify the inference time requirement. This should be chosen by the tenant such that for the task arrival process of the request it leads to an acceptable response time (i.e., waiting plus inference time). This is a calculation left to the tenant, without the involvement of the service provider. Importantly, this service abstraction relieves the tenant from specifying the task arrival process as part of the request.

The orchestrator decides on which server the requested instances are placed and may block some of the requests to avoid the violation of the inference time requirements. We denote the decision whether a tenant request r is to be served by $z^r \in \{0, 1\}$. We model the placement decisions of application instances on servers by the allocation $\mathbf{x} = (x_{a,s}^r)$, where $x_{a,s}^r$ denotes the number of instances of application a placed on server s in response to serving request r. Clearly, $\sum_s x_{a,s}^r = n_a^r$ if $z^r = 1$ and is 0 otherwise. We define the binary variable $w_{a,s}^r = \mathbb{1}_{\{x_{a,s}^r > 0\}}$, indicating whether at least one instance of application a is placed on server s in response to tenant request r, and the set $A_s = \{a | w_{a,s}^r = 1\}$ to be the set of applications placed on server s. Furthermore, we denote the activation state of server s by $w_s = \mathbb{1}_{\{\sum_{a,r} w_{a,s}^r > 0\}}$, i.e., whether at least one application instance is placed on the server.

III. Inference Time Characterization

In order to be able to find an application placement that satisfies inference time constraints, we need to develop a simple but accurate inference time model that enables counterfactual reasoning, i.e., prediction of the inference time of an application instance in a given placement, without having to perform the placement. Going beyond previous work, which proposed to characterize the inference time based on the number of colocated applications on a server [8], in what follows, we provide a novel characterization that, as we show later, allows significant savings in terms of energy consumption and latency constraint violation. The proposed characterization consists of two steps, pairwise latency profiling and application contention modeling, as discussed in the following.

A. Pairwise Latency Profiling

Tasks in O-RAN could be performed by a variety of ML-models. To capture the potential diversity of future models, we consider applications implemented using classical models (e.g., random forest) and deep neural networks (feed-forward, CNN, and LSTM architectures). Table I shows the describes the architectures of the models considered.

We aim to derive a simple model of the inference time (i.e., time spent on computation) of a model, as a function of the workload of the server on which it is deployed. For the measurements, we deploy the ML models on a single worker node in a K8S cluster. The ML models are implemented using TensorFlow and deployed as docker containers. Task arrivals to the applications follow a Poisson process emulating Machine-type Communications (MTC) and Ultra Reliable and Low Latency Communications (URLLC) traffic [12], and we measure the inference times of the tasks to determine the 95^{th} percentile of the application inference time.

Fig. 1 shows the CDF of the inference times of the models when executed without other workloads. We note that although the model sizes are similar in terms of the number of parameters, the average inference times differ by an order of magnitude. While this may be partly explained by the differences in the operations involved, e.g., the use of convolutions and sequential processing, this also implies that each model makes use of the hardware resources, such as the memory bus, CPU data caches, and instruction caches, to a different extent. As a consequence, to get a good estimate of the inference time of an application, it is important to consider what other applications it is contending with for shared hardware resources.

Impact of colocated applications: To assess the impact of colocated applications, we deploy a single instance of model a together with $n_{a'} \in \mathbf{N} = \{4, 8, 16, 32\}$ instances of model a'. The task arrival intensity to each application is $\lambda_a = \lambda_{a'} = 4/\text{sec}$. Fig. 2 shows the 95^{th} percentile of the measured inference times for each of the four models as a function of the number of models it is colocated with. The figure shows that different applications affect the inference times to a different extent. For instance, even though the inference time of the $LSTM$ (and hence the offered load), is higher than that due to the FF neural network, it is the FF neural network that affects the inference times of the models the most, followed by the $LSTM$. Thus, the type of colocated application instances also plays a vital role in determining the impact of resource contention.

Impact of task arrival rate: A key factor that may affect the inference time of an application is the arrival rate of the tasks to contending applications, since this affects the operating load of the applications. To confirm this, we place

Application	Architecture	Model size (No. of parameters)
FF	$[200, 500, 100, 10]$	151,810
LSTM	$[100, 100, 10, 1]$	161,821
CNN	$[256, 32 \times 3, 16 \times 3, 100, 10]$	139,390
RF	Max. depth: 10 Num. trees: 100	$\sim$ 53,870

TABLE I: ML models used for profiling.

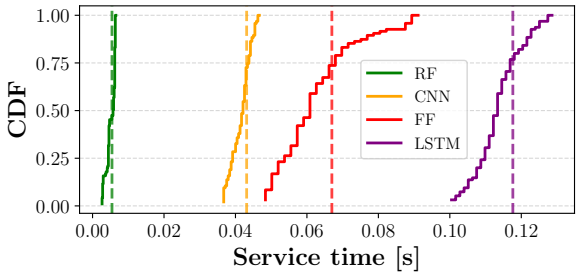
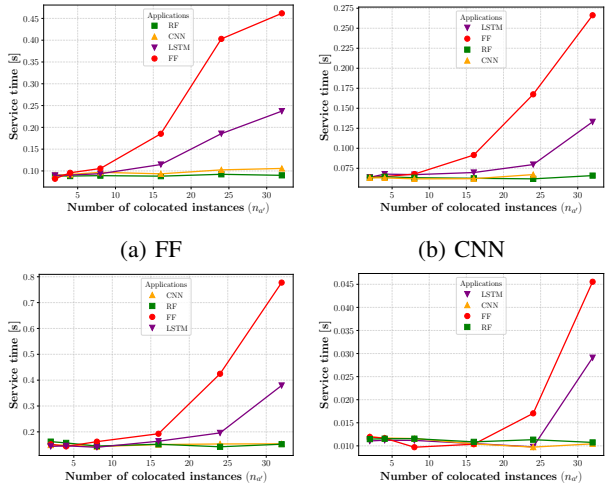


Fig. 1: Inference time of ML models when executed alone. Dashed lines show the averages.



(a) FF (b) CNN

(c) LSTM (d) RF

Fig. 2: 95^{th} percentile of inference time of a model vs. number of contending applications $n_{a'}$.

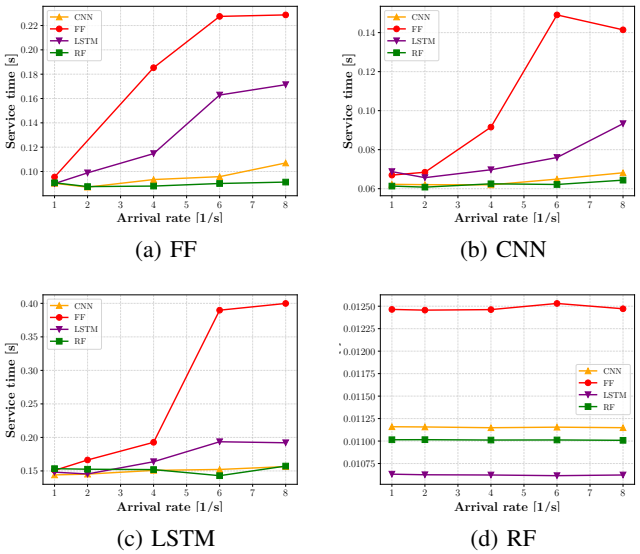
an application a and measure its inference time as a function of the arrival intensity $\lambda_{a'} \in \Lambda = \{1, 2, 4, \dots 8\}/s$ of the contending application a'. Fig. 3 shows the 95^{th} percentile of the inference time of application a as a function of the arrival intensity $\lambda_{a'}$ of the contending application for $n_{a'} = 16$.

The figure shows that for the applications FF, $LSTM$, and CNN, the task arrival rate to the contending application has a significant impact on the inference time, while the inference time of RF seems to be unaffected.

B. Application Contention Model

The above results show that colocation affects the inference time of an application depending on the types of colocated applications along with the number of instances and their task arrival rates. Capturing these for all possible combinations of application placements would make the model difficult to parametrize and thus intractable. Instead, we propose to build an inference time model based on pairwise models of application contention. For simplicity, we omit the subscript s whenever it is possible.

Pairwise contention model: The proposed model is based on a pairwise characterization of the inference time



(a) FF (b) CNN

(c) LSTM (d) RF

Fig. 3: 95^{th} percentile of the inference times of applications as a function of the arrival intensity $\lambda_{a'}$ of the contending application a', for $n_{a'} = 16$.

$t^s_{a,a'}(n_{a',s}, \overline{\lambda}_{a',s})$ of application a as a function of the number of contending application instances $n_{a',s}$ and their average arrival rates $\overline{\lambda}_{a',s} = \sum_r \lambda^r_{a'} x^r_{a',s} / n^r_{a'}$, i.e., considering the average task arrival intensity to an instance of the contending application a' on server s. Motivated by the shapes of the curves in Fig. 1 and Fig. 3, we propose to use a sigmoid function as an approximation. Our choice is also motivated by the fact that a sigmoid is easy to parametrize. The resulting pairwise approximation of the 95^{th} percentile of the inference time is given by

$$\hat{t}^s_{a,a'}(\overline{\lambda}_{a'}, n_{a',s}; \theta^{a,a'}) = \frac{\theta_1^{a,a'}(n_{a',s})}{1 + e^{-\theta_2^{a,a'}(n_{a',s})(\overline{\lambda}_{a',s} - \theta_3^{a,a'}(n_{a',s}))}} \tag{1}$$

$$\theta_i^{a,a'}(n_{a',s}; \alpha_i^{a,a'}, \beta_i^{a,a'}) = \alpha_i^{a,a'} + n_{a',s}\beta_i^{a,a'}\ , \tag{2}$$

where $\theta^{a,a'} = (\theta_1^{a,a'}, \theta_2^{a,a'}, \theta_3^{a,a'})$ are the parameters of the sigmoid function computed as an affine function of the number of instances $n_{a',s}$ of the contending application. Equations (1) and (2) allow a convenient parameterization of a family of sigmoid functions for each application pair (a, a'), yet the simplicity allows the parameterization to be incorporated as a constraint in an optimization problem.

Composite contention model: Given these pairwise contention models, we estimate the 95^{th} percentile inference time of application $a \in A_s$ on server s under an application placement $\mathbf{x}$ via an additive composite contention model

$$\begin{aligned}\hat{t}^s_a(\mathbf{x}) &= \bar{t}^s_a + \sum_{a' \in \mathcal{A}_s} \left[\hat{t}^s_{a,a'}(n_{a',s}, \overline{\lambda}_{a'}) - \bar{t}^s_a\right] \\ &= \bar{t}^s_a + \sum_{a' \in \mathcal{A}_s} \tilde{t}^s_{a,a'}(n_{a',s}, \overline{\lambda}_{a',s}),\end{aligned} \tag{3}$$

Variable	Description
z^r	Binary variable indicating if request r is served
$x^r_{a,s}$	Number of instances of application a hosted on s assigned to request r
$w^r_{a,s}$	Auxiliary variable indicating if request r has an instance on server s
$w_{a,s}$	Auxiliary variable indicating if application a is hosted on server s
w_s	Auxiliary variable indicating if server s hosts at least one application instance
$n_{a,s}$	Total number of instances of application a hosted on server s
E^f_s	Fixed energy consumption of running server s
$E^d_{a,s}$	Energy consumption per inference task processed by application a on server s

TABLE II: Frequently used notation.

where $\bar{t}^s_a$ is the 95^{th} percentile inference time of application a in isolation, and $\tilde{t}^s_{a,a'} = t^s_{a,a'} - \bar{t}^s_a$ is the increase of the inference time of application a when contending against a'. While this composite model is admittedly simple, it is relatively easy to integrate it an optimization problem, as we discuss next.

IV. Problem Formulation

We use the proposed inference time model for formulating the problem of maximizing the revenue of the operator from serving tenant requests.

A. Operator Profit

We express the operator profit as the difference of its revenue from serving tenant requests and the cost of the energy consumed for serving the requests. We adopt a widely used model of the power consumption, formulated as the sum of the fixed power consumption E^f_s if server s is active and the dynamic energy consumption, which depends on the task arrival rates of application instances placed on server s, i.e.,

$$E(\mathbf{w},\mathbf{x}) = \sum_{s\in\mathcal{S}} E^f_s w_s + \Sigma_{s\in\mathcal{S}}\Sigma_{a\in\mathcal{A}_s} E^d_{a,s}\overline{\lambda}_{a,s}(\mathbf{x}) \ , \quad (4)$$

where $E^d_{a,s}$ is the energy consumption per inference request for application a running on server s, which is multiplied by the average task arrival intensity $\overline{\lambda}_{a,s}(\mathbf{x}) = \Sigma_{r\in\mathcal{R}} x^r_{a,s}\lambda_a/n^r_a$. The profit of the operator is then expressed as

$$U(\mathbf{w},\mathbf{x},\mathbf{z}) = \Sigma_{r\in\mathcal{R}}\rho^r z^r - \sigma_E E(\mathbf{w},\mathbf{x}), \quad (5)$$

where σ_E is the unit cost of power consumption for the duration of the tenant requests.

B. Optimization Problem

Using the notation defined in Section II and the inference time characterization defined in Section III, we can formulate the profit maximization problem as,

$$\max_{\mathbf{w},\mathbf{x},\mathbf{z}} U(\mathbf{w},\mathbf{x},\mathbf{z}) \quad (6)$$

subject to:

$$\sum_{s\in\mathcal{S}} x^r_{a,s} \geq n^a_r - M(1-z^r), \forall r\in\mathcal{R} \quad (7)$$

$$\sum_{s\in\mathcal{S}} x^r_{a,s} \leq n^a_r + Mz^r, \forall r\in\mathcal{R} \quad (8)$$

$$\sum_{r\in\mathcal{R}}\sum_{a\in\mathcal{A}} C_a x^r_{a,s} \leq C_s, \forall s\in\mathcal{S} \quad (9)$$

$$t^s_a(\mathbf{x}) \leq \tau^r + M(1-w^r_{a,s}), \forall s\in\mathcal{S}, r\in\mathcal{R} \quad (10)$$

$$x^r_{a,s} \geq 1 - M(1-w^r_{a,s}), \forall r\in\mathcal{R}, a\in\mathcal{A}, s\in\mathcal{S} \quad (11)$$

$$x^r_{a,s} \leq Mw^r_{a,s}, \forall r\in\mathcal{R}, a\in\mathcal{A}, s\in\mathcal{S} \quad (12)$$

$$\sum_{r\in\mathcal{R}} x^r_{a,s} \geq 1 - M(1-w_{a,s}), \forall a\in\mathcal{A}, s\in\mathcal{S} \quad (13)$$

$$\sum_{r\in\mathcal{R}} x^r_{a,s} \leq Mw_{a,s}, \forall a\in\mathcal{A}, s\in\mathcal{S} \quad (14)$$

$$\sum_{r\in\mathcal{R}}\sum_{a\in\mathcal{A}} x^r_{a,s} \geq 1 - M(1-w_s), \forall s\in\mathcal{S} \quad (15)$$

$$\sum_{r\in\mathcal{R}}\sum_{a\in\mathcal{A}} x^r_{a,s} \leq Mw_s, \forall s\in\mathcal{S} \ . \quad (16)$$

Constraints (7) and (8) employ the big-M linearization to ensure that n^r_a application instances are provisioned for request r. (9) enforces that the memory requirement of the applications is met and that the memory consumption of provisioned application instances on the server s does not exceed the system capacity, (10) enforces the latency constraint on each server, using the delay model $t^s_a(\mathbf{x})$ as defined in (3). Constraints (11) - (16) introduce auxiliary variables, (11) and (12) set the variable $w^r_{a,s}$ to 1 if request r for application a has at least one instance on server s, (13) and (14) ensure that $w_{a,s} = 1$ if server s hosts at least one instance of application a. Finally, constraints (15) & (16) set w_s to one if server s is active. We refer to the above problem as the energy-aware O-RAN service orchestration with pairwise performance profiling (*PERX*) problem.

V. Solution Methodology

We now address the methodological challenges faced in estimating the parameters of the inference time model and in solving *PERX*.

A. Parameter Estimation

For efficient parametrization, we leverage the observation that (1) and (2) form a hierarchical model, which allows us to use a hierarchical Bayesian approach for estimating the model parameters for each application pair (a,a').

For this, we model the observed inference time as $t^s_{a,a'} = \hat{t}^s_{a,a'}(\lambda_{a'}, n_{a'}; \theta^{a,a'}) + \epsilon_1$, with $\epsilon_1 \sim \mathcal{N}(0,\sigma^2_1)$, i.e., observations are subject to Gaussian noise. Furthermore, we represent the parameters of the sigmoid as $\Theta^{a,a'}_i = \theta^{a,a'}_i(n_{a'}; \alpha^{a,a'}_i, \beta^{a,a'}_i) + \epsilon_2$, with $\epsilon_2 \sim \mathcal{N}(0,\sigma^2_2)$ accounting for modeling errors. We can then formulate the hierarchical model

$$t^s_{a,a'}|\{\Theta^{a,a'}_i\}^3_{i=1} \sim \mathcal{N}(\hat{t}^s_{a,a'}(\lambda_{a'}, n_{a'}; \{\Theta^{a,a'}_i\}^3_{i=1}), \sigma^2_1) \quad (17)$$

$$\Theta^{a,a'}_i|\alpha^{a,a'}_i, \beta^{a,a'}_i \sim \mathcal{N}(\theta^{a,a'}_i(n_{a'}; \alpha^{a,a'}_i, \beta^{a,a'}_i), \sigma^2_2)$$

$$\alpha^{a,a'}_i \sim \mathcal{U}(\gamma,\delta) \ , \beta^{a,a'}_i \sim \mathcal{U}(\gamma,\delta), \ \gamma < \delta.$$

Given a dataset $\mathcal{T}^s_{a,a'}$, containing the measured inference times of tasks of application a when contending against $n_{a'}$ instances of application a' as part of the pair-wise experiments, we initialize the parameters $\{\sigma_1, \sigma_2, \gamma, \delta\}$ based on the dataset

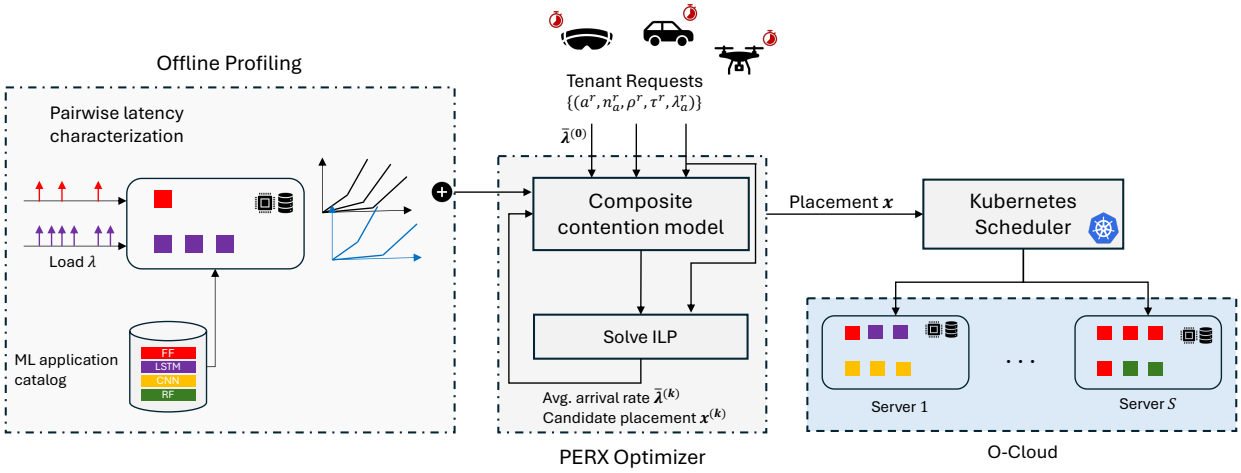


Fig. 4: *PERX* architecture and integration with cloud orchestrators. *PERX* performs a pairwise characterization of application inference time and iteratively solves the ILP yielding the placements.

as $\sigma_1 = 1$, $\sigma_2 = 1$, $\gamma = 0$, $\delta = 2.5$. We then estimate the pairwise parameters by maximizing the posterior distribution,

$$\alpha^{*\ a,a'}, \beta^{*\ a,a'} = \arg \max_{\alpha^{a,a'}, \beta^{a,a'}} f(\alpha^{a,a'}, \beta^{a,a'} | \mathcal{T}^s_{a,a'}) \tag{18}$$
$$\propto \arg \max_{\alpha^{a,a'}, \beta^{a,a'}} f(\mathcal{T}^s_{a,a'} | \Theta^{a,a'})$$
$$f(\Theta^{a,a'} | \alpha^{a,a'}, \beta^{a,a'}) f(\alpha^{a,a'}, \beta^{a,a'}) \ ,$$

This way we obtain $|\mathcal{A}| \times |\mathcal{A}|$ pair-wise inference models that characterize the inference time of application a when contending against $n_{a'}$ instances of a' as a function of the arrival rate $\lambda_{a'}$. Our results in Section VI show that the inference time estimates based on pairwise profiling are accurate for our purpose.

B. *Solving* PERX

Observe that the delay constraint (10) in *PERX* is nonlinear due to that in (3) the average arrival intensity to application a' is a function of the decision variable $\mathbf{x}$, $\overline{\lambda}_{a',s} = \sum_r \lambda^r_{a'} x^r_{a',s} / n^r_{a'}$, which in turn is multiplied by the number of instances of application a' on server s, $n_{a',s} = \sum_{r'} x^{r'}_{a',s}$. The constraint thus contains the product of two decision variables.

Instead of using an off-the-shelf nonlinear problem solver, we propose an iterative approach that decouples the computation of $\overline{\lambda}_{a',s}$ from the choice of the placement in *PERX*. Initially, we set the arrival intensity per instance of the application by considering the average arrival intensities of all the tenants, i.e., $\overline{\lambda}^{(0)}_{a',s} = \frac{1}{\sum_r \mathbb{1}_{\{a^r = a'\}}} \sum_r \frac{\lambda^r_a}{n^r_a} \mathbb{1}_{\{a^r = a'\}}, \ \forall s \in \mathcal{S}$ to obtain a latency estimate, which we use to obtain a placement by solving *PERX* (via branch-and-bound). We then iteratively recompute $\overline{\lambda}^{(k)}_{a',s}$ based on the most recent placement $\mathbf{x}^{(k)}$. We iterate the process until convergence, i.e., until the successive deployments yield the same result, $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)}$. Fig. 4 illustrates the proposed solution methodology for PERX, including the pairwise profiling-based inference time estimation and its ability to integrate with existing cloud orchestrators such as Kubernetes.

VI. Numerical Results

A. *Evaluation Methodology*

We perform extensive experiments to evaluate *PERX*. We consider deployments on $S = |\mathcal{S}| \in \{2, 5, 10, 20\}$ servers using the application catalog shown in Table III. The applications are packaged as docker containers for deployment, with a memory requirement of $C_a = 750$MB approximately. For the deployment, we use Dell R7515 PowerEdge servers with $C_s = 32$GB of RAM and 32 physical cores.

For the evaluation, we consider three scenarios. In the first two scenarios, `scenario-NRT` and `scenario-AS`, the number of tenant requests is $R = 100S/3$, i.e., the number of requests is scaled with the number of servers. Each tenant may request $n^r_a \in \{3, \ldots, 15\}$ instances of application $a \in \mathcal{A} = \{FF, LSTM, CNN, RF\}, A = |\mathcal{A}| = 4$, with an arrival intensity $\lambda^r_a \in [20, 100]$ (tasks / s). Task arrivals for a tenant request follow a Poisson process with intensity λ^r_a, as in [12]. In `scenario-NRT`, the tenant latency requirements follow the near-RT specification, $\tau^r \in [300, 500]$ (ms). In `scenario-AS`, latency requirements are application-type specific, as shown in Table III. The latency requirements in the `scenario-AS` are tighter for the CNN and the RF applications, aligned with their low inference times. In the third scenario, `scenario-AS50`, the number of tenant requests is 50, the other parameters are as in `scenario-AS`. The parameters are chosen uniformly at random for each tenant request within the specified intervals. The monetary value of requests and the energy consumption per task execution are shown in Table III, and we use $\sigma_E = 0.032$ \$ / kW for the unit power cost. The fixed power consumption for the servers is set to $E^f_s = 100$W. The results shown are the averages of 10 experiments.

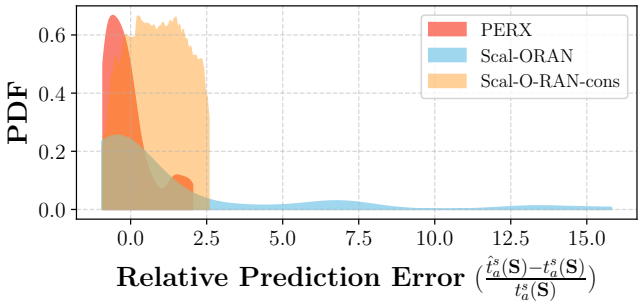
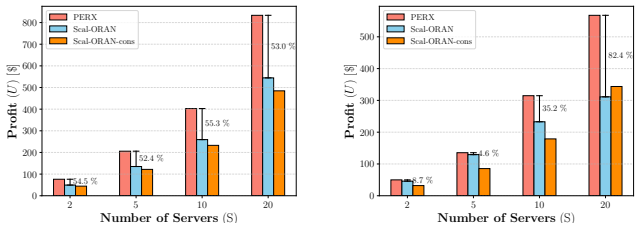
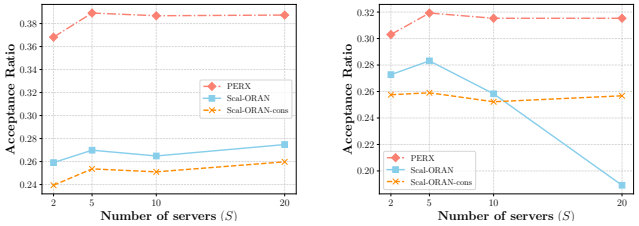


Fig. 5: Distribution of prediction errors of application inference times in a mixed deployment.



(a) `scenario-NRT` (b) `scenario-AS`

Fig. 6: Operator profit U vs. number of servers S.



(a) `scenario-NRT` (b) `scenario-AS`

Fig. 7: Acceptance ratio vs. number of servers S.

We consider two baselines for comparison. The first baseline is Scal-ORAN [8], which colocates application instances by ensuring that the average inference time of all applications hosted on the server meets the delay requirements and uses a piecewise linear approximation of the inference time, based on the total number of colocated instances. The second baseline, Scal-ORAN-cons, is a modified version of Scal-ORAN that assumes application-agnostic contention and estimates the inference time of application a as $t_{a,a'}(\mathbf{x}) = t_{a,a}(\sum_{a \in A_s} n_a, \lambda_a)$. We use Scal-ORAN-cons to show the detrimental impact of overestimating the inference time. We use the branch-and-bound implementation of Gurobi for computing the placements in each iteration of PERX, as well as for the solution of Scal-ORAN and of Scal-ORAN-cons.

B. Inference Time Estimation Performance

We start with evaluating the effectiveness of (3). To parametrize the pairwise models, we collect a data set of 16 data points for each pair of applications. We then create 60 random placements on a Kubernetes cluster and use (3) to predict the inference times of the applications. Fig. 5 shows the probability mass function of the relative prediction error. We observe that the proposed pairwise profiling approach combined with (3) results in a relative prediction error centered around 0. On the contrary, Scal-ORAN [8] and Scal-ORAN-cons tend to overestimate the inference times, potentially leading to under-utilization of resources.

C. Operator Revenue

Fig. 6 shows the operator's revenue as a function of the number of servers. Fig. 6a shows that *PERX* consistently achieves around 50% higher revenue than the baselines in the case of `scenario-RT`. We attribute this to the higher accuracy of the proposed inference time estimation, which takes into account the types and task arrival rates of contending applications. We note that compared to Scal-ORAN, Scal-ORAN-cons has a tendency to overestimate the inference time, which leads to conservative application placement, consequently yielding a lower revenue. The problem appears to be exacerbated in the case of `scenario-AS`, Fig. 6b, due to the fact that the latency constraints for the RF application are much more stringent compared to other applications (Table III), and overestimating the inference time for the RF application results in accommodating significantly fewer instances of it.

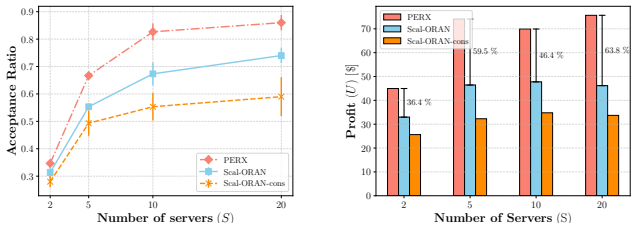
Application	Payment ($ / hr)	Energy Consumption (J / request)	Latency Requirement (ms)	
			AS, AS50	NRT
FF	6	22	[250, 400]	[300, 500]
LSTM	5	16	[400, 500]	
CNN	4	8.77	[130, 250]	
RF	3	5	[15, 40]	

TABLE III: Tenant request parameters.

This hypothesis is confirmed by Fig. 7. The figure shows that the higher revenue of *PERX* is owing to the fact that it is able to accept about 60% more tenant requests for both near-real-time latency constraints (Fig. 7a) and application-specific latency requirements (Fig. 7b). Furthermore, we observe that Scal-ORAN fails to meet the demand in case of `scenario-AS` (Fig. 7b). Given the dissimilar application requirements (`scenario-AS`, Table III), averaging the service times leads to overestimation errors; for instance, placing the FF application together with RF overestimates the service times for RF applications, inhibiting their colocation.

The decrease in acceptance ratio under `scenario-AS` for Scal-ORAN (Fig. 7b) may seem unexpected, and it is due to that branch-and-bound fails to converge for $S \geq 10$ (within 10 minutes) due to the large number of tenant requests. To confirm this, we show results for `scenario-AS50`, where the number of tenant requests is limited to $R = 50$. Fig. 8a shows a steady increase in the acceptance ratio, but *PERX* can still admit more tenant requests. As a result, *PERX* achieves consistently higher revenue (Fig. 8b).

Fig. 9 shows the number of tenant requests served per appli-



(a) Acceptance ratio (b) Operator Profit.

Fig. 8: Acceptance ratio and operator profit (U) vs. number of servers S for `scenario-AS50`.

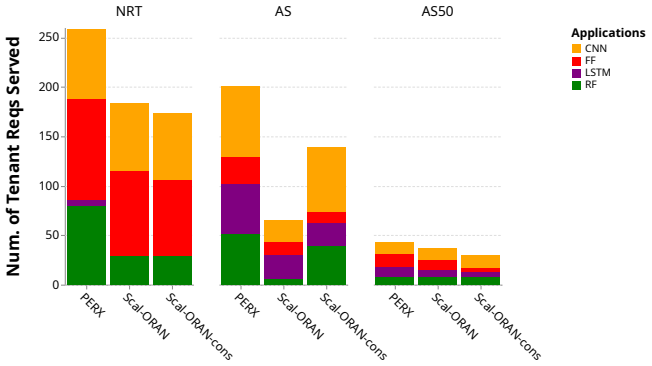


Fig. 9: Average number of tenant requests served per application type on $S = 20$ servers under the three scenarios.

cation type in the three scenarios, using the three algorithms. The figure shows that Scal-ORAN serves significantly less tenant requests for CNN and RF applications. Furthermore, we observe that *PERX* can not only accept more tenant requests (for instance, FF application requests), but it can also serve application requests that other baselines cannot, for instance, $LSTM$ application requests. The fact that *PERX* can accept more tenant requests for FF applications is especially interesting, considering that the FF application is the one that contends most with other applications (Figs 2a - 2d).

D. Task Latency Requirement Violation Probability

Fig. 10 shows the percentage of task latency requirement violations for the four applications for $S = 5$ servers, based on an actual deployment for `scenario-AS`, measured over 5 minutes. Note that a value below 5% means that the SLA is not violated, as the SLA specifies the 95^{th} percentile requirement. We observe that as a consequence of averaging the inference times, Scal-ORAN underestimates the inference time for the $LSTM$ application due to averaging, which results in more latency constraint violations for the $LSTM$ applications. Being conservative, Scal-ORAN-cons incurs the least latency violations for all applications (but also lowest profit). Although *PERX* results in a marginally higher latency constraint violation rate for CNN and $LSTM$, the mean latency constraint violations are below 5%, which implies that the SLA is met.

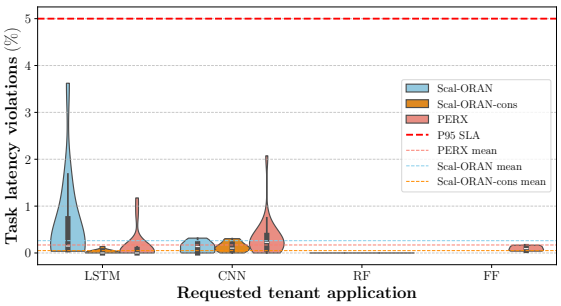


Fig. 10: Percentage of task latency violations for the four application types, $S = 5$ servers, `scenario-AS`. The dashed lines mark the mean task latency violation.

VII. Related Work

Service orchestration in O-RAN is receiving increasing attention in the literature [6], [17]. Authors in [11] focus on latency-constrained service orchestration and DU placement, considering deployment constraints imposed by the infrastructure. Similarly, [1], [10] formulate an optimal placement problem to minimize energy consumption. Authors in [3] propose an O-RAN service orchestrator for ML and AI workloads. Authors in [8] proposed a data-driven application inference time model to aid in proactive application placement, but the model leads to overestimation and underestimation of application inference times, potentially leading to lost revenue. Compared to these works, we propose a novel approach based on pairwise inference time profiling and show its benefits in application placement subject to meeting SLAs.

Designing latency-aware orchestration policies has recently been considered in the cloud computing literature [2]. Authors in [7] profile the application inference time on a pairwise basis, but do not consider colocated applications with multiple application instances. Authors in [7] consider the impact of colocating multiple application pairs, their approach does, however, not scale to complex deployments and does not support placement, only request scheduling. Authors in [5] train a binary classifier to determine whether an application instance can meet the latency requirements, given the current resource utilization, such as CPU and RAM, but do not consider placement, which is the main focus of our work.

VIII. Conclusion

In this paper, we proposed *PERX*, an energy-aware O-RAN service orchestrator that places O-RAN application instances on the O-Cloud server such that the inference time limits are honored and the profit of the O-Cloud operator is maximized. The challenge of the application placement is that the application inference times are unknown due to the complex effects of resource contention. The key component of *PERX* is, therefore, a low complexity solution for the estimation of the inference times that takes into account the colocated mix of application instances and the arrival intensity of tasks. Our

extensive evaluation shows that PERX outperforms state-of-the-art baselines by up to 50 % in terms of operator revenue while observing reduced SLA violations, demonstrating the importance of accurate inference time characterization.

PERX leaves life cycle management of the application instances to the underlying orchestrator, it could thus be integrated with the schedulers of existing orchestration frameworks such as Kubernetes and OpenShift. Interesting directions of future work include extending PERX to modeling performance when using hardware accelerators, such as GPUs and TPUs, and to services consisting of chains of ML models.

ACKNOWLEDGEMENT

This work was supported in part by Vinnova Center for Trustworthy Edge Computing Systems and Applications (TECoSA) and in part by the Swedish Research Council under Grant 2020-03860.

REFERENCES

[1] J. Baranda, J. Mangues-Bafalluy, Engin Zeydan, L. Vettori, R. Martínez, Xi Li, A. Garcia-Saavedra, C.F. Chiasserini, C. Casetti, K. Tomakh, O. Kolodiazhnyi, and C. J. Bernardos. On the integration of ai/ml-based scaling operations in the 5Growth platform. In *Proc. of IEEE NFV-SDN*, pages 105–109, 2020.

[2] Carmen Carrión. Kubernetes scheduling: Taxonomy, ongoing issues and challenges. *ACM Comput. Surv.*, 55(7), 2022.

[3] Salvatore D'Oro, Leonardo Bonati, Michele Polese, and Tommaso Melodia. OrchestRAN: Network automation through orchestrated intelligence in the Open RAN. In *Proc. of IEEE Conference on Computer Communications (INFOCOM)*, pages 270–279, 2022.

[4] Hongzhi Guo, Jiajia Liu, and Lei Zhao. Big data acquisition under failures in fiwi enhanced smart grid. *IEEE Transactions on Emerging Topics in Computing*, 7(3):420–432, 2019.

[5] Suyi Li, Wei Wang, Jun Yang, Guangzhen Chen, and Daohe Lu. Golgi: Performance-aware, resource-efficient function scheduling for serverless computing. In *Proc. of ACM Symposium on Cloud Computing (SoCC)*, page 32–47, 2023.

[6] Tania Lorido-Botran, Jose Miguel-Alonso, and Jose A Lozano. A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments. *Journal of Grid Computing*, 12:559–592, 2014.

[7] Ashraf Mahgoub, Edgardo Barsallo Yi, Karthick Shankar, Sameh Elnikety, Somali Chaterji, and Saurabh Bagchi. ORION and the Three Rights: Sizing, Bundling, and Prewarming for Serverless DAGs. In *Proc. of USENIX OSDI*, pages 303–320, 2022.

[8] Stefano Maxenti, Salvatore D'Oro, Leonardo Bonati, Michele Polese, Antonio Capone, and Tommaso Melodia. ScalO-RAN: Energy-aware Network Intelligence Scaling in Open RAN. In *Proc. of IEEE Conference on Computer Communications (INFOCOM)*, pages 891–900, 2024.

[9] Víctor Medel, Rafael Tolosana-Calasanz, José Ángel Bañares, Unai Arronategui, and Omer F. Rana. Characterising resource management performance in Kubernetes. *Computers & Electrical Engineering*, 68:286–297, 2018.

[10] Fernando Zanferrari Morais, Gabriel Matheus F. de Almeida, Leizer Pinto, Kleber Vieira Cardoso, Luis M. Contreras, Rodrigo da Rosa Righi, and Cristiano Bonato Both. PlaceRAN: Optimal placement of virtualized network functions in beyond 5G radio access networks. *IEEE Transactions on Mobile Computing*, 22(9):5434–5448, 2023.

[11] Turgay Pamuklu, Shahram Mollahasani, and Melike Erol-Kantarci. Energy-Efficient and Delay-Guaranteed Joint Resource Allocation and DU Selection in O-RAN . In *Proc. of IEEE 5G World Forum (5GWF)*, pages 99–104, 2021.

[12] Michele Polese, Leonardo Bonati, Salvatore D'Oro, Stefano Basagni, and Tommaso Melodia. ColO-RAN: Developing machine learning-based xapps for Open RAN closed-loop control on programmable experimental platforms. *IEEE Trans. on Mobile Computing*, 22(10):5787–5800, 2023.

[13] Michele Polese, Leonardo Bonati, Salvatore D'Oro, Stefano Basagni, and Tommaso Melodia. Understanding O-RAN: Architecture, Interfaces, Algorithms, Security, and Research Challenges. *IEEE Communications Surveys & Tutorials*, 25(2):1376–1411, 2023.

[14] Wen Sun, Jiajia Liu, and Haibin Zhang. When smart wearables meet intelligent vehicles: Challenges and future directions. *IEEE Wireless Communications*, 24(3):58–65, 2017.

[15] Feridun Tütüncüoglu, Ayoub Ben-Ameur, György Dán, Andrea Araldo, and Tijani Chahed. Dynamic Time-of-Use Pricing for Serverless Edge Computing with Generalized Hidden Parameter Markov Decision Processes. In *Proc. of IEEE Intl. Conf. on Distributed Computing Systems (ICDCS)*, pages 668–679, Jul. 2024.

[16] Feridun Tütüncüoglu and György Dán. Optimal Service Caching and Pricing in Edge Computing: A Bayesian Gaussian Process Bandit Approach. *IEEE Transactions on Mobile Computing*, 23(01):705–718, Jan. 2024.

[17] Luis M. Vaquero, Luis Rodero-Merino, and Rajkumar Buyya. Dynamically scaling applications in the cloud. *SIGCOMM Comput. Commun. Rev.*, 41(1):45–52, 2011.

[18] Denis Weerasiri, Moshe Chai Barukh, Boualem Benatallah, Quan Z. Sheng, and Rajiv Ranjan. A taxonomy and survey of cloud resource orchestration techniques. *ACM Comput. Surv.*, 50(2), 2017.