Optimal Flow Admission Control in Edge Computing via Safe Reinforcement Learning

A. Fox$^\diamond$, F. De Pellegrini$^\diamond$, F. Faticanti*, E. Altman†, and F. Bronzino *

Abstract—With the uptake of intelligent data-driven applications, edge computing infrastructures necessitate a new generation of admission control algorithms to maximize system performance under limited and highly heterogeneous resources. In this paper, we study how to optimally select information flows which belong to different classes and dispatch them to multiple edge servers where applications perform flow analytic tasks. The optimal policy is obtained via the theory of constrained Markov decision processes (CMDP) to take into account the demand of each edge application for specific classes of flows, the constraints on computing capacity of edge servers and the constraints on access network capacity.

We develop DRCPO, a specialized primal-dual Safe Reinforcement Learning (SRL) method which solves the resulting optimal admission control problem by reward decomposition. DRCPO operates optimal decentralized control and mitigates effectively state-space explosion while preserving optimality. Compared to existing Deep Reinforcement Learning (DRL) solutions, extensive results show that it achieves 15% higher reward on a wide variety of environments, while requiring on average only 50% learning episodes to converge. Finally, we further improve the system performance by matching DRCPO with load-balancing in order to dispatch optimally information flows to the available edge servers.

Index Terms—Edge computing, Admission Control, Constrained Markov Theory, Safe Reinforcement Learning.

I. INTRODUCTION

Edge computing techniques have emerged in recent years as a powerful solution to locally process a variety of information flows. Facing the need of serving exponentially growing service demands, infrastructure and service providers have responded by deploying their resources, from processing to storage, at the network edge. Processing information as close as possible to its source significantly reduces the amount of data to transfer to remote cloud locations, thus decreasing latency and overhead during remote service access [22], [12]. Enabled by edge clouds, new classes of data intensive AI-based applications [12], [38], [26], [30], [4] are now widespread. Unfortunately, while edge clouds offer an on premise computing solution, they are easily overwhelmed when demand exceeds available resources.

In fact, in contrast to the previous data center driven cloud model, edge clouds are often co-located with the existing network equipment and deploy limited computational resources. Thus, they can host a limited number of applications at any point in time. This generates the need of carefully designing solutions to orchestrate the operations of deployed applications. For instance, existing edge-based solutions often aim to efficiently configure available computing resources [14], [13], [15], [40] or attempt to manipulate how data flows are transported to reduce the transmission overhead [26]. This is indeed a major concern especially in smart-city environments [17]. Yet, as the number of applications and, more significantly, the number of

$^\diamond$ LIA, Avignon university, Avignon, France; *ENS, Lyon, France; †INRIA, Sophia Antipolis, France.

information flows increase, the need for a new generation of admission control algorithms becomes apparent.

Admission control is essential for managing resources efficiently, preventing under-utilization and degradation of service quality. It is widely used across various communication and computing systems, including mobile networks [29], [27], web services [6], optical networks [31], and cloud computing [18], [32]. However, the performance of AI-based edge applications depends not just on networking or compute metrics but also on the information content, posing new challenges for admission control algorithms. When deployed at the edge, admission control algorithms must select information flows processed on edge servers to maximize the information extracted by deployed applications. Flow arrivals and departures affect application operations, especially when information flow sources are mobile nodes entering or leaving an area. Edge service virtualization allows replicating multiple instances of applications and deploying them on several servers simultaneously. Replication enhances robustness but requires precise performance considerations. The results obtained in this work highlight the need to orchestrate flow admission by considering the actual installation of compute modules on edge servers and the required access bandwidth.

Modern edge applications can be commonly characterized by five features: applications process flows generated by a large number of sources of different nature; these flows can enter or leave the architecture over time due to various events; the edge infrastructure deploys a set of applications to process the flows on edge servers which are equipped with a given amount of resources (e.g., compute and memory); finally, the distributed nature of both sources and edge servers imposes the implementation of a control plane mapping flows to compute infrastructure.

Earlier models for admission control in edge-computing systems have not yet addressed all of these challenges. Hence, in this paper we develop new theoretical foundations for the edge admission problem. We extend models originally developed for admission control in loss systems, which established the paradigmatic concept of trunk-reservation [25]. In those early models, a finite service pool is made available to a finite set of service classes and each class is associated a certain reward for the admission of one of its customers. Markovian single-queue models for trunk-reservation have been studied in depth [8], [7], [25], [24]. While some multi-server admission control techniques have been studied for cloud computing, the focus is primarily on virtual machine placement relative to pricing [18] or overbooking [32]. Once applications are placed onto edge servers, the framework considered in this work provides an optimal decentralised flow admission control logic. This necessitates several novel contributions:

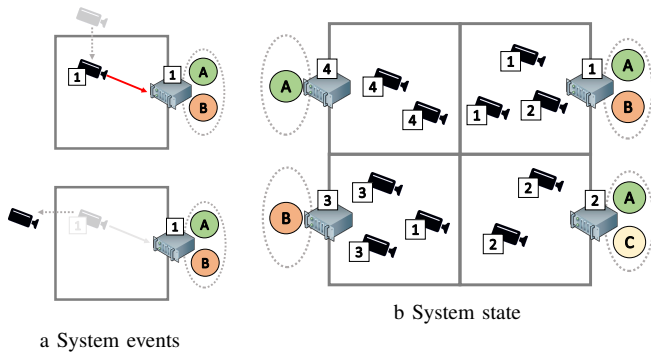*System model (Sec. II).* We develop a novel constrained Markov

Figure 1: (a) *Camera arrival and departure:* a camera arrives in area, it transmits its flow towards a tagged server (boxed index), then departs; (b) *System state:* using notation in Tab. I, $M = 4$; $\mathcal{D}^1 = \{A,B\}; \mathcal{D}^2 = \{A,C\}; \mathcal{D}^3 = \{B\}; \mathcal{D}^4 = \{A\}$; $X^1 = (2,1,0,0); X^2 = (0,2,0,0), X^3 = (1,0,2,0); X^4 = (0,0,0,2)$; $j = 1, i = 1; Y^1 = 3, Y^2 = 2, Y^3 = 3, Y^4 = 2$.

Symbol	Meaning
M	number of classes
ζ_j	arrival rate of class j flows
μ_j	mean duration of class j flows
u_j^i	prob. of routing flows of class j to server i
$\mathcal{D}^i$	set of applications installed on server i;
d^i	number of applications installed on server i
ϕ_d	servers on which $d \in \mathcal{D}$ is installed
$\mathcal{S}$	state space
S	state $S = (X, J, I)$, $X = (X^1, \ldots, X^M)$
$X^i = (X_j^i)$	flows of class j active on server i
Y^i	total occupation $Y^i = \sum_{j=1}^M X_j^i$ of server i
$\mathcal{A} = \{0, 1\}$	action space
ψ^i	computational capacity of server i
θ^i	access capacity of server i
$\chi_j(d)$	coverage requirement of app. d for class j

Table I: Main model notation.

decision model to capture the dynamic admission control and load balancing of information flows originating from multiple sources. It accounts for heterogeneous capacity constraints for both access network and edge servers. It also includes applications' replication on multiple servers and their preferences on the classes of information flows they process.

Solution concept (Sec. III). Using constrained Markov decision theory, we have derived the structural properties of the optimal decentralized admission control policy, showing it requires at most one randomized action per server. The result is not obvious since servers' states are reward-coupled.

A new learning algorithm (Sec. IV). We introduce new tools to optimize mobile information admission control policy rooted in SRL. DRCPO is a novel actor-critic scheme that leverages the structure of the optimal solution to implement the optimal flow admission policy effectively. It is tailored for cases where the same application may be installed on several edge servers simultaneously.

Load balancing (Sec. V). Finally, a two-stages joint optimization procedure increases further the system performance by jointly optimizing routing and admission control.

Our numerical results (Sec. VI) demonstrate that, by leveraging the properties of the underlying Markovian model, not only it is possible to learn the optimal admission policy with no approximation, but this can be attained with a significant reduction in complexity with respect to state of the art techniques, which are typically oblivious to the structure of the optimal policy and value function.

II. SYSTEM MODEL

We introduce a semi-Markov model general enough to cover the main characteristics of the edge flow admission control problem just outlined. It features the point process of arrivals and departures of flows belonging to a certain class, the coverage requirements of applications installed on edge servers (described by their utility function), the routing of flows to different servers and, finally, a policy to admit flows to edge servers. We now precise its mathematical definition.

Flows belong to class index $j \in \{1, \ldots, M\}$. They are generated according to a Poisson process of intensity ζ_j. A flow of class j remains active for an exponential time of mean

$1/\mu_j$ seconds, after which it leaves the system. The flow arrival processes of different classes are independent and independent of the servers' occupancy. Edge applications consist of different modules installed on some designated servers; we say that an application is installed on a server if that application has a module deployed there. In Figure 1b, we have represented a use case for video analytics. There, each class corresponds to video stream sources situated in one of $M = 4$ areas. Each area hosts a designated edge server. An application is installed on multiple servers, for instance application A in Fig. 1 resides on server 1, 2 and 4. We consider the case of M edge servers: the general case is a straightforward extension.

Let u_j^i denote the probability that a flow of class j is routed towards server i.[1] The aggregated arrival rate at server i is $Z^i = \sum_j u_j^i \zeta_j$, and the total arrival rate $Z = \sum_j \zeta_j$. Let denote $\alpha_j^i = u_j^i \zeta_j / Z^i$ the probability that an arrival is of class j and it is routed to server i. Once routed to server i, a flow is either accepted or rejected for service depending on the system state. If accepted at server i, it can feed the modules of applications installed on that server. We further assume perfect information, i.e., different servers are aware of the state of other servers. The decision-making process regarding accepting or rejecting an incoming flow also depends on the number of flows from the same class already processed by the same application across the entire system. The computational capacity of server i allows it to process at most ψ^i concurrent flows simultaneously.

Our semi-Markov decision process extends the models presented in [8], [7]. The continuous process is sampled at each arrival time t of an information flow. This results into the discrete-time MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P)$ [19]. We define $\mathcal{S}$ the system state, $\mathcal{A}$ the action set and P the MDP probability kernel. Whenever possible, *uppercase* notation, e.g., S, refers to a random process, and *lowercase* notation, e.g., s, to its realization. $\mathcal{D}^i$ is the set of applications installed on server i. Variable $\chi_j(d) \in \{0,1\}^M$ indicates whether application d is interested in flows of class j ($\chi_j(d) = 1$) or not ($\chi_j(d) = 0$).

System state. The state is a triple $S(t) = (X(t), J(t), I(t))$:
i. $X(t)$ is the matrix representing the system occupation at time t, where $X_j^i(t)$ denotes the number of flows of class j being routed to server i at time t and being processed by applications in $\mathcal{D}^i$. $Y^i(t) := \sum_j X_j^i(t)$ is the corresponding server i total occupancy.

---

[1]Throughout the paper we use subscript indexes to denote the class of the flow and superscript indexes to denote destination servers.

ii. $J(t)$ represents the class of the incoming flow;
iii. $I(t)$ is the destination server for the incoming flow.

Action set. The admission of an incoming flow for processing at a certain server is represented by action $A(S(t)) \in \mathcal{A}(S(t)) \subset \{0,1\}$. Here $A(S(t)) = 0$ signifies *reject* and $A(S(t)) = 1$ denotes *accept*. If i is the destination server and $Y^i(t) = \psi^i$, then $\mathcal{A}(S(t)) = \{0\}$, as server i has no available capacity to host additional flows.

Probability kernel. Policy $\pi : \mathcal{S} \to \mathcal{A}$ associates to state $S(t)$ a probability distribution over action set $\mathcal{A}(S(t))$. Let $p(s'|s,a) = \mathbb{P}(S(t+1) = s'|S(t) = s, A(t) = a)$ denote the transition probabilities

$$p(s'|s,a) = p((x',j',i')|(x,j,i),a)$$
$$= \alpha_{j'}^{i'} \, p(x'^i_j|x^i_j + a) \prod_{\substack{k,m=1 \\ (k,m)\neq(i,j)}}^{M} p(x'^k_m|x^k_m) \quad (1)$$

where $p(x'^k_m|x^k_m) = \mathbb{P}\left(X^k_m(t+1) = x'^k_m|X^k_m(t) = x^k_m\right)$. Let denote $\widehat{p}(u; x^i_j)$ the probability of the event that u flows of class j being routed to server i leave in between two arrivals, given that x^i_j flows are active on server i: it holds

$$\widehat{p}(u; x^i_j) = Z \int_0^\infty \binom{x^i_j}{u} e^{-\mu_j t(x^i_j - u)}(1 - e^{-\mu_j t})^u e^{-Zt} dt$$

for $0 \le u \le x^i_j$ and it is zero otherwise. Hence, the state transition probabilities at server i are derived as

$$p(x'^i_j|x^i_j, a) = \widehat{p}(x^i_j - x'^i_j + a^i_j; x^i_j) \quad (2)$$

with $a^i_j = a$ if and only if $I(t) = i, J(t) = j$ and $a^i_j = 0$ otherwise. Clearly this probability is nonzero only when $x'^i_j \le x^i_j + a^i_j$.

Rewards. Let R_{t+1} be the reward attained after the action at time t, following the traditional notation in [33]. In particular, $r(s,a) = \mathbb{E}[R_{t+1}|S(t) = s, A(t) = a]$. By admitting a flow of class j to server i, the instantaneous reward for applications binding to the tagged flow is expressed as

$$r(s,a) = a \cdot \sum_{d \in \mathcal{D}_i} r_d(x)\chi_j(d) \quad (3)$$

where $r_d(\cdot)$ is the marginal gain attained by binding a new flow to application d. Later, we define $w_{j,d}$ as the total amount of flows of class j currently being processed by application d in the system, and ϕ_d as the set of servers on which application d has been installed. Specifically, $w_{j,d} = \sum_{i \in \phi_d} x^i_j$. The immediate reward considered will only depend on this quantity: $r_d(x) = r_d(w_{j,d})$. Additionally, we assume that the immediate reward for application d is a non-increasing function of $w_{j,d}$. Finally, we define $w_j = (x^1_j, \ldots, x^M_j)$ as the vector describing the number of flows of class j active across all servers.

Policy. The admission policy π is stationary, i.e., a probability distribution over the state-action space set $\pi : \mathcal{S} \to \mathcal{A}$. In the unconstrained setting, the objective function to maximize for the admission control problem is the expected discounted reward $G_t := \sum_{t=0}^\infty \gamma^k R_{t+1+k}$ starting from initial state s_0. We define the value function

$$v_\pi(s) = \mathbb{E}_\pi\left[\sum_{t=0}^\infty \gamma^t r(S(t), \pi(S(t)))|S(0) = s\right]$$

For every stationary deterministic policy, the resulting Markov chain is regular, meaning it has no transient states and a single recurrent non-cyclic class [8]. Next, we introduce the CMDP formulation to account for the physical constraints of the system considered, particularly the constraint on access capacity.

III. THE CMDP MODEL

In CMDP theory [2], the discounted reward is taken w.r.t. the initial state distribution $\beta : \mathcal{S} \to \Delta$:

$$J_\pi(\beta) = \mathbb{E}_{s \sim \beta}[v_\pi(s)] \quad (4)$$

The access network to server i has capacity θ^i. Thus, the aggregated long-term throughput demanded by the admitted flows should not exceed such value. We define $c^i : S \times A \to \mathbb{R}$ the instantaneous cost related to the access bandwidth constraint:

$$c^i(s,a) = a \cdot \widehat{c}^i(y^i) \quad (5)$$

where $\widehat{c}^i$ is an increasing function of y^i.
The vector $K_\pi(\beta) = (K^1_\pi(\beta), \ldots, K^M_\pi(\beta))$ represents the discounted cumulative constraint, where

$$K^i_\pi(\beta) = \mathbb{E}_{\pi, s \sim \beta}\left[\sum_{t=0}^\infty \gamma^t c^i\left(X^i(t), \pi(S(t))\right)|S(0) = s\right] \quad (6)$$

For a fixed access capacity vector $\theta = (\theta^1, \ldots, \theta^M)$, and a feasible initial state distribution β, we seek an optimal policy solving the edge flow admission control (EFAC) problem

$$\underset{\pi \in \Pi}{\text{maximize:}} \ J_\pi(\beta) \quad \text{(EFAC)}$$
$$\text{subj. to:} \ K_\pi(\beta) \le \theta \quad (7)$$

We denote as $J^*(\beta)$ the corresponding optimal value.

The following structural result will be the basis of the SRL algorithm presented in the next section. The proof is provided in the technical report [10].

Theorem 1. *If the EFAC problem is feasible, then*
i. There exists an optimal stationary policy π which is randomized in at most M states;
ii. Such policy is a deterministic stationary policy if the constraint is not active;
iii. When at least one constraint is active, within the optimal stationary policy outlined in i., each state where the optimal policy is randomized corresponds to a distinct destination server.

From the computational standpoint, an optimal solution of EFAC can be determined by solving a suitable dual linear program CMDP [3], which depends explicitly on the initial distribution β.

The learning approach utilized in the following section is grounded in the lagrangian formulation, which simplifies problem EFAC to a non-constrained inf-sup problem [2].

$$\inf_{\lambda \ge 0} \sup_\pi L(\lambda, \pi) = \inf_{\lambda \ge 0} \sup_\pi [J(\pi, \beta) - \lambda(K(\pi, \beta) - \theta)] \quad (8)$$

Hence, the penalized Q-function for a given policy becomes

$$Q^\lambda_\pi(s,a) = \mathbb{E}_\pi\left[\sum_{t=0}^\infty \gamma^t r^\lambda(S(t), A(t)) \mid S(0) = s, A(0) = a\right]$$

where, for a fixed multiplier λ, $r^\lambda(s,a) = r(s,a) - \lambda c(s,a)$ is the penalized reward. The penalized value function writes $v^\lambda_\pi(s) = Q^\lambda_\pi(s, \pi(s))$.

Remark. We note that (6) *can be considered also in the form of an average constraint [36], [1]. To this respect, it is worth observing that our solution works also for the average reward form of EFAC. However, for the sake of comparison with state of the art methods, the discounted form is the most popular formulation in safe reinforcement learning [36].*

IV. LEARNING THE OPTIMAL ADMISSION POLICY

Algorithm 1 Decomposed Reward Constrained Policy Optimization

---
1: Initialize $\lambda(0)$, initial policy $\pi(0)$
2: **for** $k = 0, 1, \dots$ **do**
3:     Initialize $S(0) = \big(x^i(0), j(0), i(0)\big) \sim \beta$
4:     **for** $t = 0, \dots, T$ **do**
5:         $A(t) \sim \pi(k)(S(t))$
6:         $S(t+1), R(t+1) \sim \texttt{actor}(S(t), A(t))$
7:         Critic update for each component according to [39]
8:         Actor update: ϵ-greedy policy
9:     **end for**
10:     λ update according to (10) and (11)
11: **end for**

In situations where transition probabilities (2) are unknown, we can resort to RL algorithms to determine an optimal policy for EFAC. We design a model-free safe reinforcement learning (SRL) algorithm to account for both the instantaneous reward (3) and cost (5). This section begins with a brief introduction to reward decomposition in reinforcement learning, followed by an explanation of the motivation for the chosen type of safe reinforcement learning (Lagrangian relaxation). The next paragraph describes the introduced algorithm, which is built on these concepts. The section concludes with a brief discussion of the algorithm's convergence properties and a remark on its adaptation to the multi-constraint setting studied.

Reward and cost decomposition. For the system at hand, the full state space $\mathcal{S}$ has cardinality $\Omega(M^{\psi+2})$ where $\psi = \max\{\psi^i\}$. A direct tabular RL approach is not viable, as typical in resources allocation problems [23]. Reward decomposition has been introduced in [28] to decompose a RL agent into multiple sub-agents, where their collective valuations determine the global action. In previous works, the method has been applied in the conventional unconstrained setting only, see [34], [37], [16]. By breaking down the reward function into components within the original setting, the policy improvement step is simplified by considering a separable state-action value function [16]. To best of the author's knowledge, the algorithm proposed is the first one to consider the idea of reward and cost decomposition in the context of safe reinforcement learning.

Lagrangian relaxation methods for safe reinforcement learning. Our method is rooted in the template SRL actor-critic algorithm for the single constraint case proposed by Borkar in 2005 [5]. This algorithm prescribes a primal-dual learning procedure to solve the CMDP linear program [2] using a three-timescale framework. The two fast timescales learn the optimal policy using an actor-critic approach for a fixed Lagrange multiplier λ. The optimal value of λ is determined via gradient ascent performed at the slowest timescale.

Compared to other methods like interior point methods [20] or trust-region methods [1], this approach considers an estimate

of the penalized Q-function, along with the usual Q-learning update rule, which more naturally handles the decomposition operation of both the reward and cost functions.

Decomposed Reward Constrained Policy Optimization. This paragraph describes the RL algorithm derived from Line 11 by incorporating the decomposed actor-critic component and the Lagrange multiplier update. This algorithm is referred to as Decomposed Reward Constrained Policy Optimization (DRCPO).

In our scenario, a natural option is to identify a component for each pair (i, d) representing a destination edge server i and an application d installed therein. Moreover, fixed component (i, d), we can observe how a reduced representation of the state, $\tilde{s} = (w_{k,d}, y^i, k, m)$, is sufficient to compute the immediate penalized reward for each component. This observation can be useful in reducing the amount of estimates to compute for each component, as it aggregates several different states of the system. For each of the $\sum_i d^i$ components, we will consider a reduced state space of cardinality $\Omega(M\psi^2)$.

In general, given an arrival of class k routed to server m, if the flow is admitted, the reward is non-zero only for the components (m, d), where the application d interested in information flows of class k ($\chi_k(d) = 1$). This is given by

$$r_d^{i;\lambda}(s, a) := r_d(w_{k,d}, a) - \lambda^i c^i(y^i, a) \tag{9}$$

with $r_d(\cdot, a)$ and $c^i(\cdot, a)$ having the properties described in (3) and (5), respectively. For all other components the penalized reward is null.

Finally, we can define, for each component, the corresponding Q-function, following the usual definition:

$$Q_d^{i;\lambda}(s, a) = \mathbb{E}_\pi\left[\sum_{t=0}^\infty \gamma^t r_d^{i;\lambda}(S(t), A(t)) | S(0) = s, A(0) = a\right]$$

In the resulting actor-critic scheme, the critic will feature the aggregated Q-function, with updates being performed for all components Q_d^i at each step, using the traditional learning rule of Q-learning [39].

Regarding the action selection, we observe that, due to the reduced size of the action space, the actor can utilise a simple ϵ-greedy exploration strategy [35]. In doing so, convergence to the optimal solution still occurs, but in the set of ϵ-greedy policies: the resulting policy may be sub-optimal, in sight of Theorem 1. Conversely, this approach greatly simplifies the exploration process by considering just deterministic policies, while significantly reduces the policy search space. As seen in the numerical experiments in Sec. VI, for large values of M, the loss in performance becomes negligible. Also, alternative, policy gradient methods which can handle more complex action spaces are possible, but are left as future work.

Finally, the Lagrange multiplier update [5] is the gradient descent step

$$\lambda_{k+1} = \lambda_k - \eta_t \nabla_\lambda L(\lambda, \pi_\tau) \tag{10}$$

for suitable values of the learning rate η_t (line 10), where

$$\nabla_\lambda L(\lambda, \pi) = -\left(\mathbb{E}_s\left[K(\pi, \beta)\right] - \alpha\right) \tag{11}$$

Line 11 outlines the structure of the proposed scheme for the episodic form[2].

Convergence of DRCPO. In this paragraph, for the sake of readability, we will omit the symbol λ used to denote the Lagrange multiplier. In [16] the following results regarding the components of the Q-function obtained following the decomposition approach have been proved:

Proposition 1. *Denote $Q_d^i(s,a)(t)$ the update of the (i,d)-th component after t learning update. Under the usual conditions for the convergence of Q-learning [39], $Q_d^i(s,a)(t)$ converges almost surely to the optimal component $Q_d^{i,*}(s,a)$, for every component (d,i) and for every pair s,a. Moreover, it holds*

$$Q(s,a)(t) = \sum_{i=1}^{M} \sum_{d \in \mathcal{D}^i} Q_d^i(s,a)(t)$$

converges a.s. to the optimal Q-function $Q^*(s,a)$ so that

$$Q^*(s,a) = \sum_{i=1}^{M} \sum_{d \in \mathcal{D}^i} Q_d^{i,*}(s,a)$$

for every pair (s,a).

The convergence of DRCPO to the optimal solution is guaranteed, as stated in the following

Proposition 2. *Under standard assumptions on learning rate of stochastic approximation, DRCPO converges to an optimal solution of EFAC w.p.1.*

Proof. The proof is provided in the technical report [10]. □

Actually, the template described in Line 11 provides some flexibility in the implementation of DRCPO. For scenarios where the system involves large values of ψ^i, for instance, the critic component can be replaced by a neural network to estimate the value function component-wise. Of course, at the cost of losing the guarantees of convergence to an optimal policy.

Remark. Remarkably, the literature on SRL does not provide efficient methods to solve EFAC under multiple constraints [21]. However, Theorem 1 shows that, while $\mathcal{M}$ has coupled rewards, constraints are actually independent. This, in combination with the reward decomposition described in the previous section, let us perform the parallel of multiple single-constraint learning updates. It's worth noting that this reduction permits to optimize the lagrangian vector component-wise in a single timescale.

V. LOAD BALANCING

Up to this point, we have solved EFAC while assuming a given static routing control $\{u_j^i\}$. We now seek to optimize the routing control for the sake of load balancing. The objective is hence to maximize the reward of the system w.r.t. to joint admission control and routing:

$$J_u(\beta) = \mathbb{E}_{s \sim \beta}[v_u(s)] \tag{12}$$

where $v_u(s)$ is the value function of the load balancing policy, defined as

$$v_u(s) := \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r\left((X(t), J(t), u(J(t))), A(t)\right) \mid S(0) = s \right] \tag{13}$$

2The average reward formulation can be derived as described in [33]

and $u(J(t))$ represents the server towards which the incoming flow of class $J(t)$ arriving at time t has been routed to.

However, the analysis of the full Markov system, i.e., the SMDP where the action space encompasses both routing and admission appears extremely challenging, because the actions taken at each state are mutually dependent. Its analysis goes beyond the scope of the current work.

The optimization algorithm we propose alternates between two steps: the first one computes the new admission control policy given a fixed load balancing policy, according to the results of Sec. IV. The second step optimizes the load balancing policy for a given admission policy with the following update step

$$u_j^i(t+1) = \Pi\left[u_j^i(t) + \alpha(t)\hat{g}_j^i(t)\right] \tag{14}$$

where α is a standard step-size sequence and $\Pi[a] = \max(0, \min(a,1))$ is a projection into $[0,1]$. The gradient of the total return, denoted as $\hat{g}$, is approximated in the symmetric unbiased form, according to the SPSA algorithm [11]

$$\hat{g}_j^i(t) = \frac{\hat{R}_{\tilde{\pi}}\left(u_j^i(t) + c(t)\Delta_j^i(t)\right) - \hat{R}_{\tilde{\pi}}\left(u_j^i(t) + c(t)\Delta_j^i(t)\right)}{2c(t)\Delta_j^i(t)}$$

where c is the term of a standard stepsize sequence. $\{\Delta\}$ is a vector part of a sequence of perturbations of i.i.d. components $\{\Delta_j^i, i = 1, \ldots, M, j = 1, \ldots, M\}$ with zero mean and where $\mathbb{E}\left[|(\Delta_j^i(t))^{-2}|\right]$ is uniformly bounded. Since we cannot ensure appropriate conditions on the objective function, namely unimodality or convexity, sequence (14) is guaranteed to converge w.p.1 to a local maximum [11], thanks to some weaker regularity properties proved in the technical report [10].

The iterative procedure can continue until the convergence condition with respect to routing probabilities $\{u_j^i\}$ is attained . This procedure does not necessarily converge to the optimal value of the objective function (12), however, in the numerical section we have compared its performances to some heuristic methods and have observed its superiority. Details about the heuristic policies and the pseudocode of the adaptive method can be found in the technical report [10].

VI. NUMERICAL RESULTS

Numerical experiments are divided into three main groups.

The first reported one compares the performance of our learning algorithm against the state-of-the-art general-purpose algorithm, namely RCPO [36]. RCPO follows the same template outlined in Line 11: it uses two Neural Networks (NNs) to approximate both the value function (critic) and the policy (actor). In Deep Reinforcement Learning (DRL), NNs act as an interpolator, greatly reducing the number of represented states [23]. RCPO has been implemented to incorporate the full system state $S(t) = (X(t), J(t), I(t))$ as input for the neural networks. In the second experiment, we investigate how the reward varies as the number of applications installed per server increases. The last experiment performs a comparative analysis of the load balancing policies proposed in Sec. V. The system parameters for all experiments were randomly sampled from predefined sets: they are provided in the technical report [10]. Each column in the table represents the sets considered to generate the corresponding data. The code is available in [9].

Learning the optimal admission policy. The results depicted in Figure 2 illustrate a comparison of the performance among
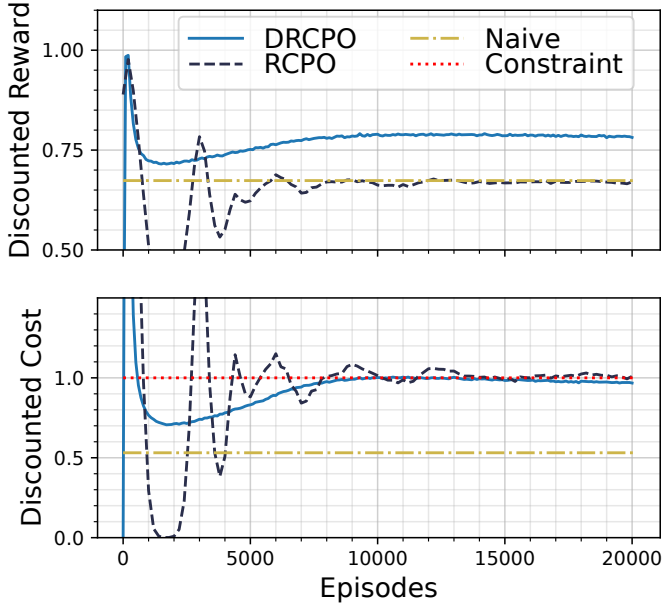
Figure 2: Learning dynamics for the a) discounted reward and b) discounted cost function.
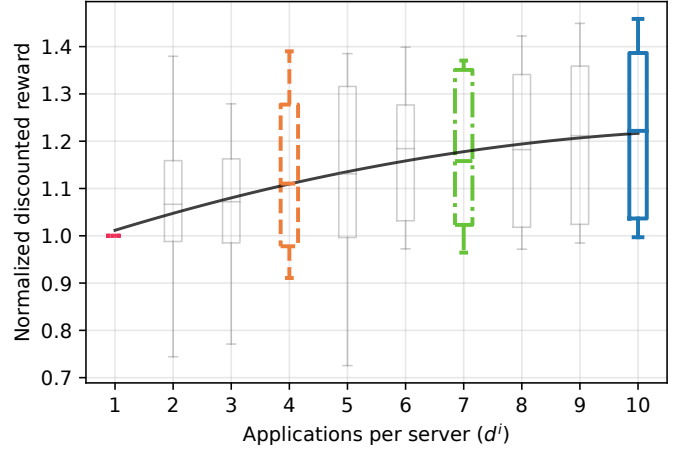

Figure 3: Optimal reward distribution at the increase of the number of applications per server.
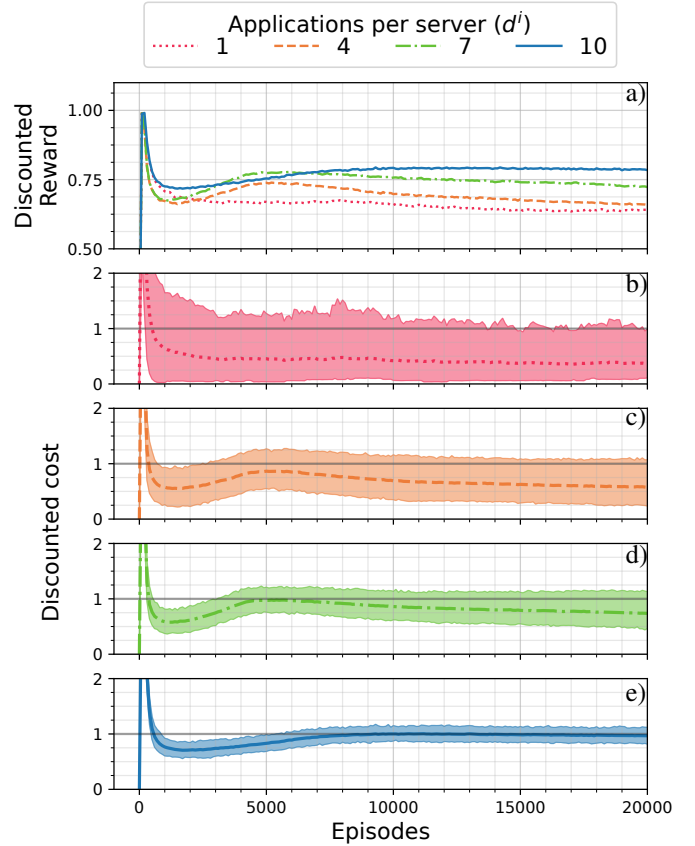

Figure 4: The joint reward and Learning dynamics for various values of d^i; in (a), (b), (c), (d), and (e) the discounted cost dynamics. Dashed line: median. Upper and lower borders of the shaded regions: server with highest and lowest associated cost, respectively.

various admission control algorithms. The load balancing policy is uniform. Without loss of generality, the system hosts only one application per flow class, potentially installed across multiple servers. In this first test, the scenario features 10 applications per server, ensuring precisely one application per flow class on each server. The experiments encompassed a total of $20,000$ episodes, with policy evaluations conducted every 100 episodes. Due to the heterogeneity of the tested system environments, the reward of each sample is normalized w.r.t. to the unconstrained optimal reward, while the cost is normalized w.r.t. the value of the constraint. The performance of DRCPO is compared with RCPO and also with a naive baseline policy. The baseline policy admits flows only when the server's total occupation is below a specific fixed threshold. The threshold value has been optimized to ensure feasibility while maximizing the reward.

The findings from Figure 2 demonstrate that in the conducted experiments, DRCPO consistently outperforms RCPO in terms of the reward, while also demonstrating better compliance to access bandwidth constraints. Specifically, the results reveal that DRCPO achieves convergence to the optimal solution in fewer than 10^4 episodes on average, whereas RCPO requires about twice the number of episodes.

Furthermore, when comparing the two SRL approaches with the previously described naive baseline, it becomes apparent that the solution provided by DRCPO yields a reward approximately 15% higher, while the performance of RCPO are comparable to those of the naive policy based on the server's total occupation.

Impact of Application Installation. We conduct 20 distinct experiments and analyze a system consisting of 10 servers, flows classes, and applications. We observe the trend followed by the optimal discounted reward as the number k of servers hosting each application increases. It's worth noting that in each experiment, exactly k applications are installed on each server, and each application is installed on exactly k servers.

Although the experiments span all possible values of d^i, Figures 3 and 4 specifically highlight the results for $d^i \in$

$\{1, 4, 7, 10\}$. To facilitate comparison, the same color is maintained for each value throughout the figures in the section. Notably, the linestyle chosen for the case with $d^i = 10$ in Figures 3 and 4 matches the line corresponding to DRCPO in Figure 2, as they represent the same data. Again, in order to compare different results, the values appearing in Figure 3 are normalized with respect to the optimal discounted reward which is obtained, in each experiment, assuming applications are installed on just one server. The plot displays median

data values, and a box plot represents the data distribution. Furthermore, a quadratic regression line illustrates the overall trend of the median data. The highest value obtained in each experiment and for each number of applications per server is recorded, ensuring that at least 50% of constraints are respected and any additional violations remain below 5%. This criterion accommodates the tendency of discounted costs to closely approach constraints while occasionally surpassing them.

From Figure 3, we first observe that installing edge applications on all servers appears to be the configuration with best performance across most of the examined experiments. In particular, with just one application per server ($d^i = 1, \forall\ i$), certain servers end up receiving a disproportionately large volume of flows. This is the case when they host applications interested in flows classes with very high arrival rates (or very low departure rates). As a result, feasibility constraints on the access bandwidth require them to admit only a small fraction of flows. On the other hand, as the number of applications per server increases, load balancing attenuates the presence of such hot-spots. The increase of the long term reward eventually levels off: it reaches a plateau towards the highest possible value, as it shows the black regression line which depicts the trend of the median rewards. In these experiments the mean increase in reward is around 20% passing from $d^i = 1$ to $d^i = 10$.

Finally, Figure 4 provides further insight into the learning dynamics for reward and cost, respectively, for different number of applications per server. The top plot reports on the learning dynamics for the discounted reward, averaged and normalized across all the experiments: it is clear that for higher values of d^i the discounted reward is higher and the convergence to an optimal solution is faster. The subsequent plots in Figure 4b/c/d/e represent the learning dynamics of the cost function as the number of applications per server increases, namely for $d^i \in \{1, 4, 7, 10\}$, respectively. In these plots, the upper (lower) boundary of the colored area denotes the dynamics of the cost of the server with highest (lowest) associated cost, which may change as the number of episodes increase. The line in the middle denotes the average cost across all the servers. It is apparent that the difference between the highest cost and the lowest is substantially higher in the case with $d^i = 1$, for the same reason previously described. On the contrary, this difference decreases at the increase of d^i. In the extreme case $d^i = 10$ all servers consistently maintain costs proximal to their respective constraints. A higher number of applications per server apparently grants more efficient utilization of available resources, and consequently it increases the discounted rewards across most of the sample data.

Another reason behind the poor performances in the case with lower values of d^i is that, as previously mentioned, the implementation of DRCPO presented here exclusively adopts deterministic policies for practical reasons, while, as indicated in Theorem 1, the optimal policy is stochastic in one state per destination server. The deterministic nature of the sought policy has a particularly adverse effect on the performance of DRCPO, especially for lower values of d^i, as observed in Figure 4. This is likely because the state with the optimal stochastic policy is more frequently visited in these cases.

Developing an SRL algorithm that incorporates stochastic policies to address this specific issue, which is notably problematic only in scenarios with low values of d^i, would have been more challenging, slower, and ultimately of limited practical utility for more realistic scenarios involving multiple applications per server. The search for an efficient method to derive a policy that is stochastic in a single state is left for future work.

Comparing different load balancing policies. The last set of numerical experiments of Figure 5 compares different load balancing policies in a scenario where each server may have different parameters. These experiments examine an increasing number of servers, i.e., $M \in \{3, 5, 7, 10\}$. Once the arrival rates ζ_j from class j are defined, the flow arrival rate per class to a designated server is determined based on the routing policy.

For ease of comparison, the values on the y-axis of Figure 5 are normalized relative to the naive load balancing, which routes flows to all servers uniformly at random, regardless of whether they host applications interested in such flows. Consequently, all displayed values exceed unity.

Figure 5a illustrates the normalized reward per server concerning the increasing average ratio between servers' capacity ψ^i and access capacity θ^i. Notably, for lower values of the ratio ψ/θ, the occupation-based load balancing policy exhibits the best performance. In this scenario, the reward shows a non-increasing trend concerning a server's occupation: with low ψ/θ values, the optimal admission policy tends to admit flows more frequently. Consequently, the occupation-based load balancing policy prioritizes routing flows to underloaded servers, attaining the highest cumulative reward. However, as the ratio per server ψ/θ increases, the performance gap in favour of the adaptive load balancing widens. Finally, both the uniform and origin-based policies yield comparable results across the experiments.

Figure 5b depicts the normalized reward as the number of servers increases. Once again, the adaptive load balancing consistently outperforms all other methods. Furthermore, as the value of M increases, the advantage over the naive load balancing widens.

In these experiments, applications installed on each server are interested, on average, in half of the possible flow classes. Consequently, with increasing M, the naive uniform load balancing becomes increasingly inefficient, leading to a significant fraction of flows being discarded. However, it's worth noting that while the adaptive load balancing demonstrates clearly superior performance, this comes at the cost of a larger number of policy evaluation steps.

VII. Conclusions

Pushed by the surge of edge analytics, the integration of flows from diverse classes poses a significant challenge to existing edge computing architectures. In response, we have introduced a decomposed, constrained Markovian framework for the decentralized admission control of varied information flows. The objective is maximizing the utility of edge applications while accounting for constraints on access network bandwidth and compute capacity. Within this framework, we adopt safe reinforcement learning as the solution concept to derive an optimal policy, even in presence of unknown and highly heterogeneous system parameters. Leveraging the structure of the underlying Markovian model, our proposed solution outperforms state-of-the-art approximated deep reinforcement learning approaches, reducing significantly the number of required learning episodes
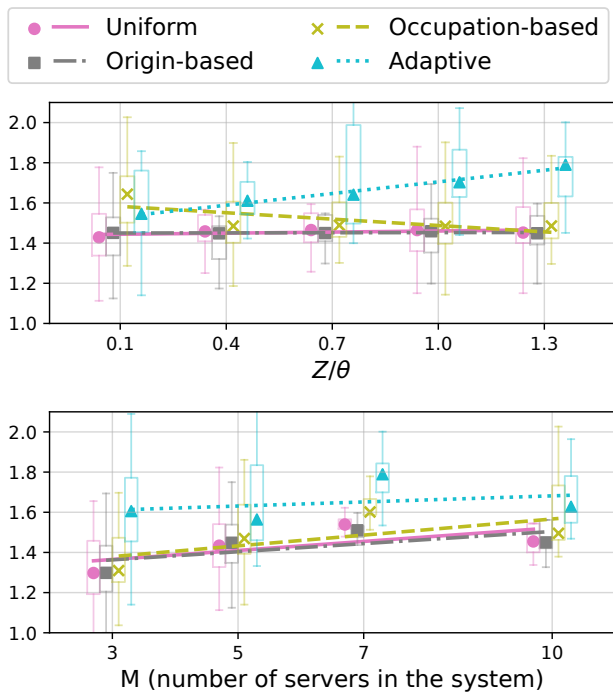
Figure 5: Performance of different load balancing policies; values on the y-axis normalized w.r.t. naive uniform load balancing: (a)increasing ratio ψ/θ per server and (b)increasing number of classes M.

for convergence. Moreover, our novel reward decomposition method, DRCPO, attains an optimal admission policy.

This work marks an initial step in the field of admission control for edge analytics, opening several directions for future investigation. A particularly challenging one involves developing a comprehensive Markovian model for joint admission control and routing. Furthermore, addressing network constraints beyond edge access capacity requires considering also the core network topology and the requirements of application modules deployed beyond edge servers. Additionally, one could introduce specific application performance metrics into the model. This would permit to obtain specialized admission policies for flow analytic tasks such as video analytics or anomaly detection. In this regard, model extensions to incorporate per-flow information content are left as part of future work.

REFERENCES

[1] J. Achiam et al. Constrained policy optimization. In *Proc. of International Conference on Machine Learning (ICML)*, 2017.
[2] E. Altman. *Constrained Markov Decision Processes*. Chapman and Hall, 1999.
[3] E. Altman and A. Schwartz. Adaptive control of constrained Markov chains. *IEEE Transactions on Automatic Control*, 36(4), 1991.
[4] N. Borgioli, L. Thi Xuan Phan, F. Aromolo, et al. Real-time packet-based intrusion detection on edge devices. In *Proc. of Cyber-Physical Systems and Internet of Things Week*. 2023.
[5] V. Borkar. An actor-critic algorithm for constrained Markov decision processes. *Elsevier Systems & Control Letters*, 54(3), 2005.
[6] L. Cherkasova and P. Phaal. Session-based admission control: A mechanism for peak load management of commercial web sites. *IEEE Trans. Comput.*, 51(6), 2002.
[7] X. Fan-Orzechowski and E. A. Feinberg. Optimality of randomized trunk reservation for a problem with a single constraint. *Advances in Applied Probability*, 38(1), 2006.
[8] E. A. Feinberg and M. I. Reiman. Optimality of randomized trunk reservation. *Probability in the Engineering and Informational Sciences*, 8(4), 1994.

[9] A. Fox, F. De Pellegrini, F. Faticanti, E. Altman, and F. Bronzino. https://github.com/andrea-fox/decomposedrcpo, 2024.
[10] A. Fox, F. De Pellegrini, F. Faticanti, et al. Optimal flow admission control in edge computing via safe reinforcement learning. *arXiv preprint https://arxiv.org/abs/2404.05564*, 2024.
[11] M. Fu and S. Hill. Optimization of discrete event systems via simultaneous perturbation stochastic approximation. *IIE Transactions*, 29(3), 1997.
[12] M. Hu, Z. Luo, A. Pasdar, et al. Edge-based video analytics: A survey. *arXiv preprint arXiv:2303.14329*, 2023.
[13] J. Huang, B. Lv, Y. Wu, et al. Dynamic admission control and resource allocation for mobile edge computing enabled small cell network. *IEEE Transactions on Vehicular Technology*, 71(2), 2022.
[14] C. Hung, G. Ananthanarayanan, and P. e. a. Bodik. Videoedge: Processing camera streams using hierarchical clusters. In *Proc. of the IEEE/ACM Symposium on Edge Computing (SEC)*, 2018.
[15] J. Jiang, G. Ananthanarayanan, P. Bodik, et al. Chameleon: scalable adaptation of video analytics. In *Proc. of ACM SICOMM*, 2018.
[16] Z. Juozapaitis, A. Koul, A. Fern, et al. Explainable reinforcement learning via reward decomposition. In *Proc. of IJCAI*, 2019.
[17] L. U. Khan, I. Yaqoob, N. H. Tran, et al. Edge-computing-enabled smart cities: A comprehensive survey. *IEEE Internet of Things Journal*, 7:10200–10232, 2019.
[18] K. Konstanteli, T. Cucinotta, K. Psychas, et al. Elastic admission control for federated cloud services. *IEEE Transactions on Cloud Computing*, 2(3), 2014.
[19] S. A. Lippman. Applying a new device in the optimization of exponential queuing systems. *Operations Research*, 23(4), 1975.
[20] Y. Liu, J. Ding, and X. Liu. Ipo: Interior-point policy optimization under constraints. In *Proc. of AAAI*, 2019.
[21] Y. Liu, A. Halev, and X. Liu. Policy learning with constraints in model-free reinforcement learning: A survey. In *Proc. of IJCAI*, 2021.
[22] Q. Luo, S. Hu, C. Li, et al. Resource scheduling in edge computing: A survey. *IEEE Communications Surveys & Tutorials*, 23(4), 2021.
[23] H. Mao, M. Alizadeh, I. Menache, et al. Resource management with deep reinforcement learning. In *Proc. of ACM HotNets*, New York, NY, USA, 2016.
[24] A. Massaro, F. De Pellegrini, and L. Maggi. Optimal trunk-reservation by policy learning. In *Proc. of IEEE INFOCOM*, 2019.
[25] B. L. Miller. A queueing reward system with several customer classes. *Management science*, 16(3), 1969.
[26] C. Pakha, A. Chowdhery, and J. Jiang. Reinventing video streaming for distributed vision analytics. In *Proc. of USENIX HotCloud*, 2018.
[27] M. Raeis, A. Tizghadam, and A. Leon-Garcia. Reinforcement learning-based admission control in delay-sensitive service systems. In *Proc. of IEEE GLOBECOM*, 2020.
[28] S. J. Russell and A. Zimdars. Q-decomposition for reinforcement learning agents. In *Proc. of ICML*, 2003.
[29] S. Senouci, A. Beylot, and G. Pujolle. Call admission control in cellular networks: a reinforcement learning solution. *International journal of network management*, 14(2), 2004.
[30] M. Seufert, K. Dietz, N. Wehner, et al. Marina: Realizing ml-driven real-time network traffic monitoring at terabit scale. *IEEE Transactions on Network and Service Management*, 2024.
[31] C. Sue, Y. Hsu, and P. Ho. Dynamic preemption call admission control scheme based on Markov decision process in traffic groomed optical networks. *Journal of Optical Communications and Networking*, 3(4), 2011.
[32] M. Sultan, L. Marshall, B. Li, et al. Kerveros: Efficient and scalable cloud admission control. In *Proc. of USENIX OSDI*, 2023.
[33] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018.
[34] R. S. Sutton, J. Modayil, M. Delp, et al. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *Proc. of AAMAS*, 2011.
[35] C. Szepesvari. *Algorithms for reinforcement learning*. Number 9 in Synthesis lectures on artificial intelligence and machine learning. Morgan & Claypool, 2010.
[36] C. Tessler, D. Mankowitz, and S. Mannor. Reward constrained policy optimization. In *Proc. of ICLR*, 2019.
[37] H. van Seijen, M. Fatemi, J. Romoff, et al. Hybrid reward architecture for reinforcement learning. In *Proc. of NIPS*, 2017.
[38] J. Wang, Z. Feng, Z. Chen, et al. Bandwidth-efficient live video analytics for drones via edge computing. In *Proc of IEEE/ACM SEC*, 2018.
[39] C. J. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3/4), 1992.
[40] W. Zhang, S. Li, L. Liu, et al. Hetero-edge: Orchestration of real-time vision applications on heterogeneous edge clouds. In *Proc. of IEEE INFOCOM*, 2019.