ISMAEL: Using Machine Learning To Predict Acceptance of Virtual Clusters in Data Centers

Johannes Zerwas, Patrick Kalmbach, Stefan Schmid, and Andreas Blenk

***Abstract*—Existing virtual network admission control algorithms targeting high utilization of data center infrastructure are computationally expensive or provide poor performance. In particular, existing algorithms have in common that they are *oblivious to the past*, i.e., requests are handled in a fire-and-forget manner, not taking into account information from previously solved instances. This can be inefficient and misses out on a basic optimization opportunity: as for any network optimization algorithm that faces repeating problem instances, it may be beneficial to learn from network states and the outcome of acceptance decisions of the past. In this paper, we propose ISMAEL, a Machine Learning framework for predicting the acceptance of Virtual Clusters, one of the most common virtual network abstractions in data centers. ISMAEL can be configured with, and learn from, different existing algorithms by combining fixed-size feature representations for graphs with a Convolutional Neural Network or a Fully Connected Deep Neural Network. We report on extensive simulations, which demonstrate that it is possible to mimic existing, computationally-intensive admission control algorithms with an accuracy of up to 94 %, while significantly reducing runtime.**

***Index Terms*—Data Center Resource Management, Virtual Cluster, Admission Control, Machine Learning.**

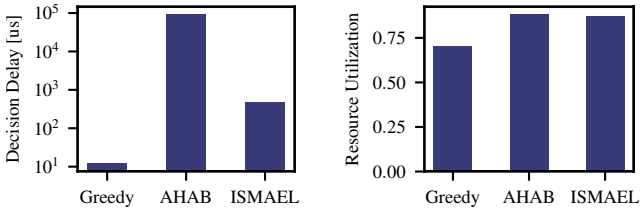
I. INTRODUCTION

The Objective: High Data Center Utilization. The ongoing migration of service and application providers to the cloud promises cost savings through resource sharing and economies of scale. However, it introduces a risk of interference, especially regarding network resources, which can negatively affect application performance [1]. Network virtualization allows to remove this risk of unpredictable cloud application performance by guaranteeing performance isolation: virtual network abstractions provide resource reservations (on both nodes and links). However, a predictable performance can only be guaranteed to individual users of multi-tenant data centers if the admitted requests do not exceed the amount of available physical resources. In other words, a smart admission control algorithm is needed. Indeed, deciding on which virtual network requests to admit and embed is a difficult algorithmic problem. Especially under high demands (the desired mode of operation of expensive infrastructure), admission control is crucial: a smart admission control algorithm can lead to significantly higher utilization than the common "accept-whenever-it-fits" policy. Surprisingly, only little is known today on how to design efficient admission control for the network virtualization scenario.

J. Zerwas, P. Kalmbach and A. Blenk are with the Department of Electrical and Computer Engineering, Technical University of Munich, Germany, e-mail: {johannes.zerwas,patrick.kalmbach,andreas.blenk}@tum.de.

S. Schmid is with University of Vienna, Austria, e-mail: stefan_schmid@univie.ac.at.

Manuscript received November 30, 2018; revised April 08, 2019; accepted June 10, 2019.



(a) Mean Decision Delay (b) Resource Utilization

Fig. 1: The challenge of admission control: existing simulation-based algorithms providing high-quality solutions (high utilization and hence, profit), have a high decision delay. The shown values are averaged over 30k decisions made by the admission control algorithms. Greedy is fast but has low utilization. AHAB [2] has high utilization but suffers from large delay. ISMAEL combines small decision delay with high resource utilization.

The Problem: High Computational Effort. Deciding which virtual networks to accept and embed is computationally challenging: as the embedding problem is generally hard [3], it is already difficult to estimate the resulting resource cost. Also recent approaches, which perform what-if analyses to make predictions using simulations [2], while providing good solutions, still come at a high runtime cost: an open problem in prior work [2] (Fig. 1).

Our Idea: Learning From Algorithm Data. This paper shows that high-quality admission control and low runtime are not contradictory goals in practice. In particular, we argue that Monte-Carlo Tree Search-based approaches such as AHAB [2] can greatly benefit from Machine Learning (ML), speeding up decisions without sacrificing quality. This is motivated by the hypothesis that the behavior of such algorithms is likely to have clear patterns when the scenario does not change much. This facilitates prediction models, making expensive simulations obsolete, and consequently speeds up the admission control process.

Accordingly, we propose an ML approach, ISMAEL, which is motivated by the key observation that while repeatedly solving admission control and embedding problems, network algorithms create a valuable source of knowledge: the problem and solution data generated over time [4]. Having seen enough data and being well-trained, ISMAEL can even make the original admission control algorithm obsolete. Although gathering and training data might increase the computational effort in the beginning, this pays off in the long run: we will show that in the long run computation resources are saved.

The Challenges: Data Representation and Machine-Learning Pipeline Design. Designing an ML-based system such as ISMAEL is non-trivial and involves many challenges,

regarding data gathering, data representation, learning model selection and design, classification algorithm selection etc. The available data for an ML algorithm comprises, e.g., the current substrate (data center) state and the current virtual network request. Generally, an ML algorithm needs to make an acceptance decision for a request based on the current utilization of the substrate's resources and the given request. Just a simple representation for a substrate can already contain several thousands of servers (i.e., features) when considering common sizes of today's data center networks [5], [6]. Hence, expressive representations are needed that make it possible for an ML algorithm to carry out the best decision. Moreover, algorithm selection, i.e., which classifier to choose, and configuration problems are still inherent to ML pipeline designs. Accordingly, human domain and expert knowledge is usually needed to make design choices due to the complex task of fully constructing end-to-end ML-based architectures for enhancing (networking) algorithms.

Contributions in this paper are:

- The design of an ML framework, ISMAEL, to learn the behavior of admission control algorithms, which includes ML pipeline design like feature engineering and algorithm selection. As a case study, we focus on Virtual Clusters (VC) [7], a common virtual network abstraction.
- A comprehensive evaluation and comparison. In particular, we propose different representations of substrate and VC states as inputs for ML classifiers. The representations differ in size and complexity. We then report on extensive simulations and provide guidelines for selecting the best data representation and ML classifier. We also provide an analysis of how the classifier behaves compared to the original algorithm, and report on problems and challenges when combining different ML concepts.
- A proof-of-concept implementation. We configure ISMAEL with a state-of-the-art algorithm [8] which derives a fixed-size deep-learning representation from graphs and report on our insights of this case study.

As a contribution to the community and in order to allow researchers to reproduce our results, we will make our C++/Python-based simulator implementing our embedding and admission control algorithms publicly available with the acceptance of this paper.[1]

Structure of this paper. Section II puts our work into perspective and reviews related work from different areas. Section III introduces VC abstraction and presents our model. Section IV presents our ML-based decision pipeline for speeding up admission control decisions for VCs. Section V reports on the results of our simulations. Section VI draws conclusion and sketches potential future work.

II. Related Work

This section identifies and summarizes four different areas of related work: admission control in general, ML for admission control, ML for combinatorial optimization problems, and how ML helps or even solves completely network optimization problems.

[1] https://github.com/tum-lkn/vc-simulator

A. Call Admission Control

Classicly, Call Admission Control (CAC) has been studied in the context of wireless/cellular or ATM networks. In this scenario, users request "lines" by making calls which must be served by the network. There exist many heuristic algorithms for this problem, and also ML has been applied, e.g., leveraging Supervised Learning (SL) methods [9]–[13] for estimation of performance metrics or Reinforcement Learning (RL) for admission policy optimization [14]–[16]. Bashar *et al.* [12] give a concrete example for an SL approach. They compare the performance of Neural Networks and Bayesian Networks to estimate Quality of Service metrics, such as delay or packet loss, that are used as input to the CAC algorithm. Marbach *et al.* [14] implement an RL approach to solve the CAC problem in a multi-service class network. The works of this category differ from our work as they do not aim at learning admission decisions from an existing algorithm, but support admission control algorithms by predicting input data. Furthermore, they differ in structure and complexity of the requests which are single connections instead of more complex structures such as VCs, and their allocation. The request and the substrate have only a single resource dimension (bandwidth). The multi-resource scenario in our work requires a more thorough approach of data representation and learning problem formulation.

B. Admission Control for Virtual Network Embedding

Admission control has also been studied for virtual networks: Virtual Networks (VNs) are a generalization of the VC abstraction that imposes less structural constraints on the requests. Most of the existing works in this area consider the online scenario where requests arrive over time and propose heuristic [17], [18] or approximate solutions [19], [20]. Nejad *et al.* [21] consider the scenario of batch arrivals. They propose an admission control for chains of virtual network functions that increases the network performance over time by jointly optimizing admission control and embedding using Mixed Integer Linear Programs. In [18], another VN admission control scheme is proposed. The main objective is the number of accepted high priority VN requests. Based on knowledge about potential future arrivals the algorithm decides to drop embedded low priority requests to free resources for future high priority requests. In contrast to our work, both approaches consider increasing the performance of the embedding, they address different scenarios, e.g., the offline problem and requests with different priorities and allow eviction, and do not consider ML techniques to speed up the decision process.

AHAB [2] is an admission control algorithm that is evaluated with VCs and serves as a baseline of our work. AHAB aims at maximizing the utilization of the substrate over time. To do so, it evaluates the impact of an arriving request on future arrivals using a Monte-Carlo Tree Search approach. Upon arrival of a VC, AHAB generates potential future VCs and using simulations, evaluates if acceptance of the arrived VC increases resource utilization. It yields good performance in terms of resource utilization but suffers from high computational effort to make admission decisions.

A first admission control system for VNs that leverages ML techniques is proposed in [22]. Since Virtual Network Embedding (VNE) in general is an NP-hard problem [3], the goal is to reduce the computational effort in cases where the VN is infeasible. The system uses a Recurrent Neural Network to learn from past runs of VN embedding algorithms and to predict if a VN request is feasible on the current state of the substrate. Requests that are predicted to be feasible are forwarded to the embedding algorithm, the others are rejected. However, the admission control here does not aim at maximizing the revenue or resource utilization of the substrate over time.

Beside ML for admission control in the context of VNE, we also see ML for (network) optimization problems as related to our work: the problems might rely on, e.g., deep graph representations. Hence, we also comprehensively summarize work which might potentially integrate similar mechanisms as we do for admission control.

C. Machine Learning for Optimization Problems

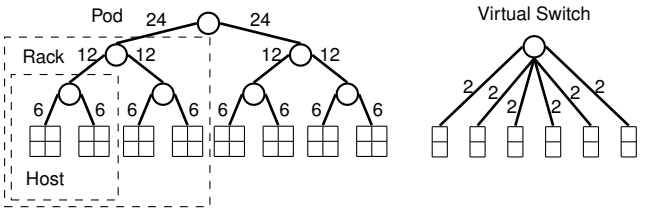
The technological evolution of computing hardware and the ability to make use of large-scale cloud resources enabled numerous breakthroughs in ML, like deep learning. This also advanced the application areas of ML, e.g., to combinatorial optimization problems. Bello *et al.* [23] propose a framework that uses neural networks and RL to solve combinatorial optimization problems like the traveling salesman and the knapsack problem. In particular, they show for the traveling salesman problem how a Recurrent Neural Network can predict a distribution over cities, i.e., finding the most likely route.

Dai *et al.* [24] ask whether it is possible to automate the challenging task of designing good heuristics or approximation algorithms for hard-to-be-solved optimization problems. They argue that in the real-world, similar optimization problems are repeatedly solved creating valuable amount of data which can be exploited by RL. The learned greedy algorithm can solve a wide range of problems: minimum vertex cover, maximum cut, and traveling salesman problem.

Li *et al.* [25] perform combinatorial optimization using Graph Convolutional Network (GCN) and guided tree search. The basic idea is to estimate the likelihood that a node is part of the solution using the GCN which then guides the tree search towards the solution. The authors show that their approach is applicable to different canonical NP-hard problems and outperforms state-of-the-art solvers.

While the previous papers [23]–[25] are examples on how ML can directly solve optimization problems, Hutter *et al.* [26] address the configuration of solvers for mathematical problems with an ML framework. The framework auto-tunes the parameters of solvers to speed-up the execution time when solving exact algorithms. Rachelson *et al.* [27] predict the outcomes of binary variables of a Mixed Integer Program based on past outcomes to reduce the runtime of the solver.

All these examples demonstrate that ML can either solve optimization problems entirely or help speeding up existing algorithms. Nevertheless, there are still many shortcomings: e.g., many solved problems were either small scale or they were not addressing details of real system models, i.e., the problem models were rather abstract. In order to deepen the modeling level when applying the ideas and concepts to networking, expert knowledge is still needed.



(a) A substrate. (b) A virtual cluster request.

Fig. 2: Illustration of substrate and VC. The numbers on the links represent either the available uplink bandwidth of the entity in case of the substrate or the requested bandwidth in case of the VC. The number of the boxes illustrate the computational capacities, four in case of the substrate (a) hosts, and the requested capacity, two for each of the six VMs of the VC (b).

D. Machine Learning for Network Optimization Problems

Applying ML to networked and distributed systems is generally not new. To name a few exemplary studies, Gao and Jamidar [28] deployed a Neural Network that learned from operational data to predict plant performance, which helped improving the energy consumption of Google's data centers.

Winstein *et al.* [29] initiated the study of data-driven design for developing RemyCC, a TCP congestion control algorithm. Recent publications extended this idea. The authors of [30], [31] demonstrated the feasibility of online learning-based and pure data-driven TCP (or more abstract congestion control in general) design. In the context of resource management of cloud networks, DeepRM [32] is a system that learns to schedule data center tasks from experience.

To the best of our knowledge, ISMAEL, is the first proposal to leverage ML on algorithm generated data to improve both the performance and efficiency of admission control algorithms.

III. Model and Preliminaries

This section presents our setting and the necessary preliminaries. We introduce the VC admission control problem.

A. Virtual Cluster and Substrate

We consider the problem of admitting and embedding virtual networks (i.e., VCs) on a given physical network (i.e., the substrate). We define the substrate and virtual networks in turn. The mathematical names and conventions are summarized in Table I for the substrate and in Table II for the VCs.

Substrate. We consider tree-like substrates C hosting VCs, e.g., a Fat-Tree [33], a common data center architecture today (cf. Fig. 2). A set of pods $\mathcal{P}$ is connected via core switches. A set of racks $\mathcal{R}$ is distributed among the pods $\mathcal{P}$, i.e., each pod contains a subset of racks $\mathcal{R}' \in \mathcal{R}$. The racks connect via their Top-of-Rack (ToR) switches to the aggregation switches of the pods. The set of hosts $\mathcal{H}$ is distributed among the racks $\mathcal{R}$, i.e., each rack contains a subset of hosts $\mathcal{H}' \in \mathcal{H}$ which are connected via the ToR.

TABLE I: Notation and abbreviations for substrate.

Symbol	Description
C	Substrate with a tree-like topology
$\mathcal{H}$	Set of hosts of substrate C
$\mathcal{R}$	Set of racks of substrate C
$\mathcal{P}$	Set of pods of substrate C
$\mathcal{E}$	Set of all edges interconnecting the hosts with the ToR switches, the ToR switches with the aggregation switches, and the aggregation switches with the core switches
π	A subset of links constituting a path connecting hosts in the substrate C
CU	Compute unit: Abstract unit to measure computation requirements or capacity
BU	Bandwidth unit: Abstract unit to measure bandwidth requirements or capacity
$\texttt{CU/BU}_{\text{free}}(x)$	Free resources BU/CU of $x \in \mathcal{H}$, $x \in \mathcal{R}$ or $x \in \mathcal{P}$ in the substrate C
$\texttt{CU/BU}_{\text{total}}(x)$	Total resources BU/CU of $x \in \mathcal{H}$, $x \in \mathcal{R}$ or $x \in \mathcal{P}$ in the substrate C
$\texttt{UplinkBU}_{\text{free}}(x)$	Free resources BU on up-link of $x \in \mathcal{H}$, $x \in \mathcal{R}$ or $x \in \mathcal{P}$ in the substrate C. = 0 if $x \in \mathcal{H}$.

TABLE II: Notation and abbreviations for virtual cluster request.

Symbol	Description
VC	A virtual cluster request defined as a tuple, i.e., $\texttt{VC} = (\mathcal{VM}^{\text{VC}}, \mathcal{E}^{\text{VC}}, \texttt{CU}_{\text{VM}}(\texttt{VC}), \texttt{BU}_{\text{VM}}(\texttt{VC}))$.
$\mathcal{VM}^{\text{VC}}$	Set of VMs of a VC request $\texttt{VC} \in \mathcal{VC}$
$\mathcal{E}^{\text{VC}}$	Set of virtual edges interconnecting all virtual machines with the virtual switch of a VC
$\#\texttt{VM}(\texttt{VC})$	Number of VMs that a request has, i.e., $\#\texttt{VM}(\texttt{VC}) = \lvert\mathcal{VM}^{\text{VC}}\rvert$
$\texttt{CU/BU}_{\text{VM}}(\texttt{VC})$	BU/CU demand per VM of a VC request $\texttt{VC} \in \mathcal{VC}$

TABLE III: Notation and abbreviations of metrics and sets used within the VC problem.

Symbol	Description
$\mathcal{VC}$	Set of all VC requests arrived over time, containing both accepted (i.e., also embedded) and rejected requests
$\mathcal{VC}^{\text{Acc}}$	Set of all VC requests accepted and embedded over time
$\mathcal{VC}^{\text{Rej}}$	Set of all rejected VC requests
$\mathcal{VC}^{\text{Fea}}$	Set of all feasible requests, independent of whether a request was accepted or rejected
$\mathcal{VC}^{\text{Inf}}$	Set of all infeasible requests, independent of whether a request was accepted or rejected

We differentiate between two types of abstract resources: bandwidth units $\texttt{BU} \in \mathbb{N}$ and compute units $\texttt{CU} \in \mathbb{N}$. For BU and CU, each network entity x (either host, rack or pod) has a total capacity determined by $\texttt{BU/CU}_{\text{total}}(x)$ and a remaining capacity determined by $\texttt{BU/CU}_{\text{free}}(x)$. The bandwidths of the links of the aggregation levels equal the accumulated bandwidths of the corresponding child nodes: for instance, for each rack $R \in \mathcal{R}$, the total capacity $\texttt{BU}_{\text{total}}(R)$ of the uplink of rack R is equal to the sum of capacities of the uplinks of the hosts $\texttt{BU}_{\text{total}}(h)$, with $h \in R$. $\texttt{UplinkBU}_{\text{free}}(x)$ returns the free resources on the uplink of a network entity $(\mathcal{H}, \mathcal{R}, \mathcal{P})$. For $x \in \mathcal{H}$, it returns 0 as for this entity the resources on the uplink are already described by $\texttt{BU}_{\text{free}}(x)$. Similarly to [7], [34], we approximate the Fat-Tree by a simple tree. The simplified Fat-Tree depicted in Fig. 2(a) consists of two pods, containing two racks each; there are two hosts per rack. A host has a capacity of 4 CUs and the hosts' link capacities are 6 BUs. The links on aggregation and core level have capacities of 12 BUs and 24 BUs respectively.

Virtual Cluster. Using the VC abstraction [7] customers are able to specify their computation and communication requirements separately. A VC is a quadruple $\texttt{VC} = (\mathcal{VM}^{\text{VC}}, \mathcal{E}^{\text{VC}}, \texttt{CU}_{\text{VM}}(\texttt{VC}), \texttt{BU}_{\text{VM}}(\texttt{VC}))$, where $\mathcal{VM}^{\text{VC}}$ and $\mathcal{E}^{\text{VC}}$ describe the set of Virtual Machines (VM) and edges, respectively, and $\texttt{CU/BU}_{\text{VM}}(\texttt{VC})$ yield the demanded capacities. Note that we assume symmetric capacity demands. All VMs are of the same computational size $\texttt{CU}_{\text{VM}}(\texttt{VC})$, and are connected to a virtual switch at bandwidth $\texttt{BU}_{\text{VM}}(\texttt{VC})$. For instance, the VC in Fig. 2(b) requests 6 VMs with a size of 2 CUs and a bandwidth of 2 BUs between the VMs and the virtual switch.

B. Virtual Cluster Embedding Problem

The VC embedding problem describes the situation where an operator has to embed a VC to its infrastructure. According to the preceding definitions, the physical resources of the environment are constrained; similar, the VC request might constitute resource demands. Under these conditions, the operator is faced with the problem to find a valid embedding where all requested VMs are placed on physical machines with enough available computational capacities. Moreover, all VMs need to be interconnected with enough link bandwidth to make a valid embedding of a VC request. To accomplish a valid embedding for a VC, the task is to find a valid mapping for VMs and the interconnecting links:

$$\texttt{map}_{\text{VM}} : \mathcal{VM}^{\text{VC}} \rightarrow \mathcal{H} \tag{1}$$

$$\texttt{map}_{\mathcal{E}^{\text{VC}}} : \mathcal{E}^{\text{VC}} \rightarrow \pi(\mathcal{E}) \tag{2}$$

Moreover, computational and bandwidth resource constraints need to be satisfied, i.e.,

$$\texttt{CU}_{\text{free}}(h) \geq \sum_{v \in \mathcal{VM}^{\text{VC}} \ \wedge \ \texttt{map}_{\text{VM}}(v)=h} \texttt{CU}_{\text{VM}}(\texttt{VC}), \ \forall h \in \mathcal{H} \tag{3}$$

$$\texttt{BU}_{\text{free}}(e) \geq \sum_{e' \in \mathcal{E}^{\text{VC}} \ \wedge \ e \in \texttt{map}_{\mathcal{E}^{\text{VC}}}(e')} \texttt{BU}_{\text{VM}}(\texttt{VC}), \ \forall e \in \mathcal{E}, \tag{4}$$

where a host must provide enough computational resources to embed one up to many (maximal $\#\texttt{VM}(\texttt{VC})$) VMs of a request; simultaneously, all links need to provide enough residual capacities to host the bandwidth of the virtual links. Note that multiple VMs can be hosted on one host; vice versa multiple virtual links can share the same physical edge. The equations (1)-(4) must be fulfilled for a valid embedding; otherwise a request cannot be embedded.

C. Virtual Cluster Embedding with Admission Control

In the online version of the VC embedding problem, operators face requests arriving and leaving the infrastructure over time. In order to define the admission control procedure

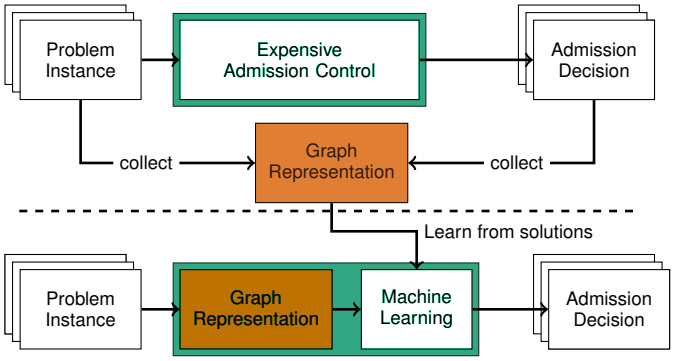


Fig. 3: The proposed ML architecture as presented in [4] is extended by the graph representations and applied to the admission control problem within VCs. The ML components now use graph representations for learning as input: either to make a decision or when learning from the problem solutions.

precisely, we need additional definitions to classify the set of virtual clusters $\mathcal{VC}$ that arrived over time (additionally summarized in Table III).

Greedy-acceptance: A first naive solution is to perform greedy admission control: in a *greedy acceptance strategy*, every request will be embedded if feasible: $\mathcal{VC}^{\text{Acc}} = \mathcal{VC}$ and $\mathcal{VC}^{\text{Rej}} = \emptyset$. Note again that requests that cannot be embedded are classified as $\mathcal{VC}^{\text{Inf}}$ and the accepted and embedded requests are defined by $\mathcal{VC}^{\text{Acc}} \cap \mathcal{VC}^{\text{Fea}}$. However, such a strategy might lead to a sub-optimal substrate utilization over time: for instance, a larger virtual request being embedded with high resource consumption (the VM instances are not colocated in the same racks or pods), might potentially block many subsequently arriving smaller requests. Accordingly, the greedy decision to accept a request with high-footprint results in an overall low substrate utilization over time.

Non-greedy-acceptance: The VC embedding problem with admission control is, thus, defined as the problem to find a strategy that accepts or rejects VC requests in a clever way, e.g., to increase the overall substrate utilization which is given by the sum of CUs of the accepted and feasible VCs:

$$\max_{\mathcal{VC}^{\text{Acc}}} \frac{1}{\text{CU}_{\text{total}}(C)} \sum_{\text{VC} \in \mathcal{VC}^{\text{Acc}} \cap \mathcal{VC}^{\text{Fea}}} \text{CU}_{\text{VM}}(\text{VC}) \cdot \#\text{VM}(\text{VC}). \quad (5)$$

Accordingly, a potentially more sophisticated admission control strategy also rejects requests leading to $|\mathcal{VC}^{\text{Rej}}| \geq 0$. For comprehensiveness, the offline variant of this problem describes a situation with n requests given, out of which potentially a subset of requests has to be chosen to maximize the substrate utilization over time. As a consequence, a non-greedy strategy leads to a set of rejected requests containing actually both feasible and infeasible requests: $\mathcal{VC}^{\text{Rej}} \subseteq (\mathcal{VC}^{\text{Fea}} \cup \mathcal{VC}^{\text{Inf}})$. An optimal admission control strategy will produce a set of accepted requests that (1) only contain feasible requests, i.e., $\mathcal{VC}^{\text{Acc}} \subset \mathcal{VC}^{\text{Fea}}$ which (2) additionally lead, e.g., to the highest possible substrate utilization over time.

IV. ISMAEL: A MACHINE-LEARNING PIPELINE FOR SPEEDING UP ADMISSION CONTROL ALGORITHMS

We now present our solution, ISMAEL: a framework for admission control which achieves both high utilization and low runtime. ISMAEL is based on a ML pipeline that aims

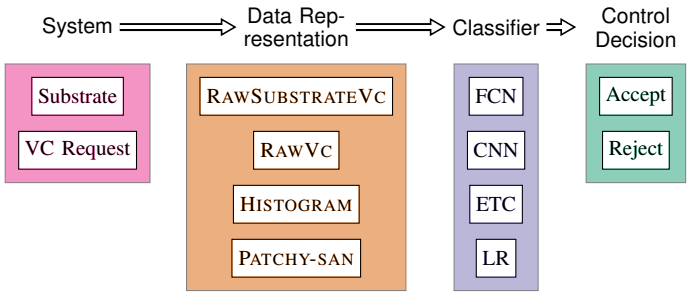


Fig. 4: Overview of ISMAEL's ML-based admission control framework. The system can be split into two components: substrate and VC request. Four ML-readable inputs are available: RAWSUBSTRATEVC, RAWVC, HISTOGRAM, and PATCHY-SAN. Four different classifiers are implemented to work with the different data representations: Fully Connected Deep Neural Network (FCN), Convolutional Neural Network (CNN), Extra Trees Classifier (ETC) and Logistic Regression (LR). ISMAEL makes predictions about the control decisions to accept or reject a VC request.

to speed up admission control algorithms. First, the general approach of ISMAEL is elaborated. Detailed explanations of data representation and classification follow.

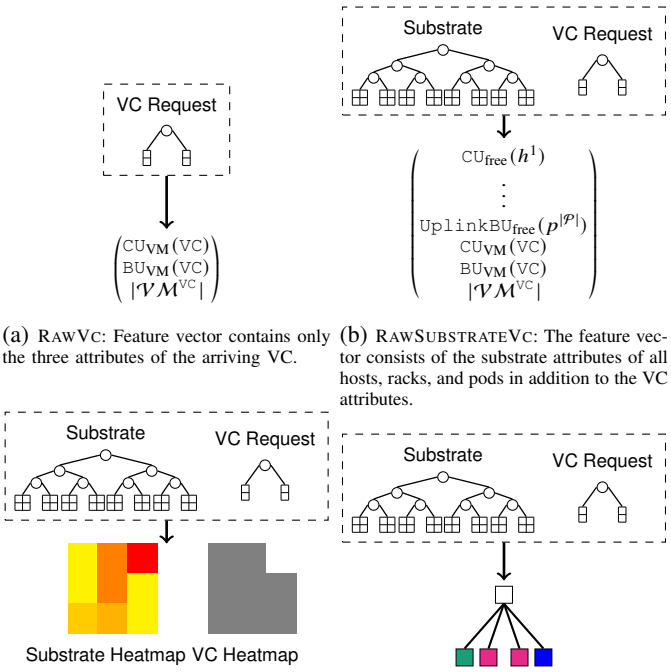
A. Key Idea: Learning from the Past

With each decision of the admission control (whether to admit or not), data is generated and can be collected over time resulting in a large amount of data for further use (upper part of Fig. 3). Examples of such data are the state of the substrate network, which contains the usage of the resources and the currently embedded VCs with their embeddings, the attributes of the arriving VC, the values of the embedding performance metrics that the admission control obtained so far and the decision made by admission control algorithm.

Following the data-driven paradigm and similar to recent work [4], [22], ISMAEL uses collected tuples of substrate state, arriving VC and corresponding decision in order to mimic the strategy of AHAB by training a binary classifier. The specific classification task that ISMAEL tackles, is whether a new VC should be accepted given the current state of the substrate. Eventually, ISMAEL can replace AHAB in an online scenario, offering less computational effort at runtime and faster decision times (lower part of Fig. 3). AHAB is an example here for a computation intensive algorithm. However, ISMAEL can replace other admission control algorithms as well. ISMAEL has a modular structure as Fig. 4 illustrates. Its components are explained in the following.

B. Data Representation Strategy

One of the major challenges for learning the behavior of AHAB is to adopt a meaningful representation of the system's state and the arriving VC. There are a plethora of possible representations that need to be selected and parameterized. Not wisely selecting an efficient representation can result in the curse of dimensionality [35]. In ML research, many representations for graphs have been proposed [36], [37], out of which we select PATCHY-SAN [8] due to its efficiency. Moreover, we develop three other domain specific representations of different complexity that leverage observations from the performance analysis of AHAB [2] (RAWVC, RAWSUBSTRATEVC, HISTOGRAM). Since AHAB is agnostic to requests' lifetimes



(a) RawVc: Feature vector contains only the three attributes of the arriving VC.

(b) RawSubstrateVc: The feature vector consists of the substrate attributes of all hosts, racks, and pods in addition to the VC attributes.

(c) Histogram: The substrate heatmap representation provides a compact view of the substrate. The VC heatmap contains a 1 (white) for the respective VC.

(d) Patchy-san: The visualized receptive field contains the selected rack (leftmost node, green), the two hosts of the rack (middle, pink) and the pod of the rack (right, blue).

Fig. 5: Comparison between data representations: RawVc, RawSubstrateVc, Histogram and Patchy-san.

Algorithm 1 Data Representation: RawSubstrateVc

Input: Substrate C, Arriving VC $\texttt{VC}$

1: Initialize $\mathcal{S} \in \mathbb{R}^{3\cdot(|\mathcal{H}|+|\mathcal{R}|+|\mathcal{P}|)+3}$
2: $idx = 0$
3: **for** $n \in \mathcal{H} \cup \mathcal{R} \cup \mathcal{P}$ **do**
4: $\quad \mathcal{S}\ [idx] = \texttt{CU}_{\text{free}}(n)$, $\mathcal{S}\ [idx+1] = \texttt{BU}_{\text{free}}(n)$, $\mathcal{S}\ [idx+2] = \texttt{UplinkBU}_{\text{free}}(n)$
5: $\quad idx += 3$
6: $\mathcal{S}\ [idx] = \#\texttt{VM}(\texttt{VC})$, $\mathcal{S}\ [idx+1] = \texttt{CU}_{\text{VM}}(\texttt{VC})$, $\mathcal{S}\ [idx+2] = \texttt{BU}_{\text{VM}}(\texttt{VC})$
7: **return** $\mathcal{S}$

and temporal patterns, none of the proposed representations considers the lifetime of the requests or the temporal utilization of physical resources. We leave this to future work.

1) RawVc*:* As the simplest approach, we consider a data representation which captures only the arriving VC. With this representation, we want to answer the question whether VC attributes are enough for an admission control to make good decisions. The RawVc representation (Fig. 5(a)) only consists of the three attributes of the arriving VC $\#\texttt{VM}(\texttt{VC}), \texttt{CU}_{\text{VM}}(\texttt{VC}), \texttt{BU}_{\text{VM}}(\texttt{VC})$. This results in the feature vector

$$\mathcal{S} = [\#\texttt{VM}(\texttt{VC}), \texttt{CU}_{\text{VM}}(\texttt{VC}), \texttt{BU}_{\text{VM}}(\texttt{VC})] \in \mathbb{R}^3 \tag{6}$$

with a fixed size of $|\mathcal{S}| = 3$.

2) RawSubstrateVc*:* A more complex approach to represent the system is to collect all resource utilizations in the substrate C. The feature vector $\mathcal{S}$ consists of three parts that represent the hosts, racks and pods (Fig. 5(b)).

Algorithm 2 Data Representation: Histogram

Input: Substrate C, Arriving VC $\texttt{VC}$, N :=bins for $\#\texttt{VM}$, S :=bins for $\texttt{CU}_{\text{VM}}(\texttt{VC})$, B :=bins for $\texttt{BU}_{\text{VM}}(\texttt{VC})$

1: Initialize $\mathcal{S}_C \in \mathbb{R}^{|N|\times|S|\times|B|}$
2: **for** $vc \in \mathcal{VC}_{emb}(C)$ **do**
3: $\quad$ Get bin i_N, i_S, i_B for $\#\texttt{VM}(vc), \texttt{CU}_{\text{VM}}(vc), \texttt{BU}_{\text{VM}}(vc)$
4: $\quad \mathcal{S}\ [i_N, i_S, i_B] += \frac{1}{|\mathcal{VC}_{emb}(C)|}$
5: Initialize $\mathcal{S}_{VC} \in \mathbb{R}^{|N|\times|S|\times|B|}$
6: Get bin i_N, i_S, i_B for $\#\texttt{VM}(\texttt{VC}), \texttt{CU}_{\text{VM}}(\texttt{VC}), \texttt{BU}_{\text{VM}}(\texttt{VC})$
7: $\mathcal{S}_{VC}\ [i_N, i_S, i_B] += 1$
8: **return** $\mathcal{S} = [\mathcal{S}_C,\ \mathcal{S}_{VC}]$

Algorithm 1 shows the procedure for the RawSubstrateVc representation. After initializing the feature vector $\mathcal{S}$ (line 1), the algorithm iterates over every node in the substrate and collects three values that describe the resource utilization of a node (lines 3-5). The first value is the number of available $\texttt{CU}$s in the entity the node represents ($\texttt{CU}_{\text{free}}(n)$). For a host, this simply means the number of available $\texttt{CU}$s. For racks and pods, the number of available $\texttt{CU}$s are accumulated over the hosts contained in this entity. The representation employs an analogous approach for the second value ($\texttt{BU}_{\text{free}}(n)$), the number of available $\texttt{BU}$s per node, i.e., host, rack or pod. The third value ($\texttt{UplinkBU}_{\text{free}}(n)$) describes the number of available $\texttt{BU}$s on the up-link (the link to the parent node) of the entity. For hosts, this value is already considered by the previous value and therefore $\texttt{UplinkBU}_{\text{free}}(n) = 0, \forall n \in \mathcal{H}$. As this information is not yet present for racks and pods, it is added to the feature vector. Finally, the arriving VC is added with plain numbers of the attributes, i.e., $\#\texttt{VM}(\texttt{VC}), \texttt{CU}_{\text{VM}}(\texttt{VC}), \texttt{BU}_{\text{VM}}(\texttt{VC})$ (line 6). The size of this representation depends only on the size of the substrate and is calculated as

$$|\mathcal{S}| = \left(\frac{k^3}{4} + \frac{k^2}{2} + k\right) \cdot 3 + 3 \tag{7}$$

where k is the Fat-Tree construction number.

3) Histogram*:* The evaluation in [2] points out that Ahab favors requests with certain characteristics - with large $\texttt{CU}_{\text{VM}}(\texttt{VC})$ compared to bandwidth requirement ($\texttt{BU}_{\text{VM}}(\texttt{VC})$) - and identifies a distinct pattern. This motivates the use of the VC attributes $\#\texttt{VM}(\texttt{VC})$, $\texttt{CU}_{\text{VM}}(\texttt{VC})$, $\texttt{BU}_{\text{VM}}(\texttt{VC})$ to categorize the currently embedded VCs and in addition to the arriving request as in case of RawVc. The Histogram representation implements this idea. The result is a tensor that contains two $3D$-histograms: one for the set of embedded VCs and one for the arriving VC as Fig. 5(c) illustrates. Algorithm 2 lists the procedure to create these histograms. It iterates over all allocated VCs and increments the values of the corresponding bins (lines 2-4). Note that the histogram for the arriving request contains only zeros but is one for the bin that corresponds to the attributes of the arriving VC (lines 6f).

Parameters of this representation are the number and size of the bins for each of the VC's attributes. They determine the size of the feature vector as

$$|\mathcal{S}| = 2 \cdot (\text{Total number of bins})\,. \tag{8}$$

Algorithm 3 Data Representation: PATCHY-SAN

Input: Substrate C, Arriving VC VC, num. receptive fields w, size of receptive fields k, stride s

1: Initialize $\mathcal{S} \in \mathbb{R}^{3 \cdot k \cdot w+3}$
2: $N = \{\mathcal{R}, \mathcal{H}, \mathcal{P}\}$ where each subset $\mathcal{R}, \mathcal{H}, \mathcal{P}$ is sorted by $\mathtt{CU}_{\text{free}}(n)$
3: $i = 1,\ j = 1,\ idx = 0$
4: **while** $j < w$ **do**
5: $V \leftarrow$ neighborhood of size k for node $N[i]$ sorted by $\mathtt{CU}_{\text{free}}(n)$
6: **for** $n \in V$ **do**
7: $\mathcal{S}\ [idx] = \mathtt{CU}_{\text{free}}(n), \mathcal{S}\ [idx+1] = \mathtt{BU}_{\text{free}}(n), \mathcal{S}\ [idx+2] = \mathtt{UplinkBU}_{\text{free}}(n)$
8: $idx = idx + 3$
9: $i = i + s,\ j = j + 1$
10: $\mathcal{S}\ [idx] = \#\mathtt{VM}(\mathtt{VC}), \mathcal{S}\ [idx+1] = \mathtt{CU}_{\mathtt{VM}}(\mathtt{VC}), \mathcal{S}\ [idx+2] = \mathtt{BU}_{\mathtt{VM}}(\mathtt{VC})$
11: **return** $\mathcal{S}$

4) PATCHY-SAN*-based:* This representation implements the approach of Niepert *et al.* called PATCHY-SAN [8]. PATCHY-SAN is a powerful framework to create fixed size representations of attributed graphs that are suited for the use with Convolutional Neural Networks (CNN). CNNs exploit spatial locality of features by connecting only a subset of the input to a successive layer. By stacking multiple such layers a CNN first learns local features that become more global with depth. In our scenario, local features can correspond to the resource utilization of racks or pods.

CNNs were originally designed for images and cannot directly be applied to arbitrary graphs [8]. PATCHY-SAN addresses this issue by transforming the graph into a fixed-size structure that can be consumed by a CNN. To fill the requested number of receptive fields w with the corresponding size k, the algorithm samples a sequence of w nodes from the graph using a labeling (and ranking) procedure. For each of the w nodes in the sequence, a neighborhood containing k nodes is created using the same labeling procedure. The features of each of these nodes serve as the values of the different channels c of the pixels of the receptive fields. For graphs with less than k nodes, the receptive fields are padded with zeros to obtain again $k \cdot w \cdot c$ values.

Algorithm 3 sketches how we match the PATCHY-SAN approach to the VC admission control scenario (for more details on PATCHY-SAN see [8]). As for the RAWSUBSTRATEVC representation we consider every host, rack and pod as one node of our graph. PATCHY-SAN selects a sub-set of these nodes and also rearranges the values in the feature vector making the representation more suitable for CNNs and reducing the symmetries that are introduced by the regular structure of the substrate's topology. To cover as many levels as possible from our tree topology, we sort these nodes in a way such that the racks occur first, followed by hosts and pods (line 2). The intuition behind the racks first approach is that constructing a one hop neighborhood from the racks adds hosts and pods to the single receptive fields. Among nodes of the same type, i.e., hosts, racks and pods, they are sorted according to the number of available CUs ($\mathtt{CU}_{\text{free}}(n)$). This ensures that mainly those nodes are added to the feature vector that have a high number of available resources.

As stated before, for every node that is chosen from the sorted list, a neighborhood of size k is constructed starting with the hosts that are immediate neighbors (line 5). For every node in this neighborhood, the amount of available resources extracted and added to the feature vector similar to the procedure for the RAWSUBSTRATEVC representation, i.e., the number of available CUs within the node ($\mathtt{CU}_{\text{free}}(n)$), the available BUs within the node ($\mathtt{BU}_{\text{free}}(n)$) and the available BUs on the uplink ($\mathtt{UplinkBU}_{\text{free}}(n)$) (lines 6-8). The stride parameter s determines how many nodes in N are skipped before creating the next neighborhood.

Fig. 5(d) illustrates an example of one receptive field of size 4 and one resource dimension (channel). The rightmost node (green) is the rack that is selected and the remaining three nodes constitute the neighborhood. The next two pixels (pink) denote the two hosts of the rack while the last pixel (blue) represents the pod of the rack. Increasing the number of receptive fields results in concatenating multiple of these groups of pixels for different racks.

As for the RAWSUBSTRATEVC representation, the arriving request is represented by the three attributes $\#\mathtt{VM}(\mathtt{VC})$, $\mathtt{CU}_{\mathtt{VM}}(\mathtt{VC})$, $\mathtt{BU}_{\mathtt{VM}}(\mathtt{VC})$ (line 10). The size of the modified PATCHY-SAN depends on the parameters k, w:

$$|\mathcal{S}| = 3 \cdot k \cdot w + 3. \tag{9}$$

Although PATCHY-SAN is specifically tailored for use with CNNs, we also apply other classifiers to this representations.

C. Classifiers and Decision

The ISMAEL framework can work with several types of classifiers. We compare four classifiers: a Fully Connected Deep Neural Network (FCN), a CNN, an ExtraTrees Classifier (ETC) [38] and Logistic Regression (LR). All of them can be fed with the different data representations. The outputs of the classifiers are the probabilities of accepting and rejecting the request. The request is rejected if the probability of rejecting p_{reject} is larger than the probability of accepting p_{accept}, i.e.,

$$reject = \begin{cases} 1, & p_{reject} \geq p_{accept} \\ 0, & \text{else.} \end{cases} \tag{10}$$

When using FCN, ETC and LR, all elements of the data representation are directly fed to the classifier. In case of the CNN and PATCHY-SAN or RAWSUBSTRATEVC, the feature vector is split into the part that represents the substrate and the part that represents the arriving VC (Fig. 6). The substrate is encoded by convolutional layers of the network, while the VC part skips these layers and is directly connected to the dense layers of the CNN (cf. [39]).

V. EVALUATION

In the following, we report on our simulation setup (Section V-A) and our main results. We start with an assessment of the accuracy of different classifiers and data representation

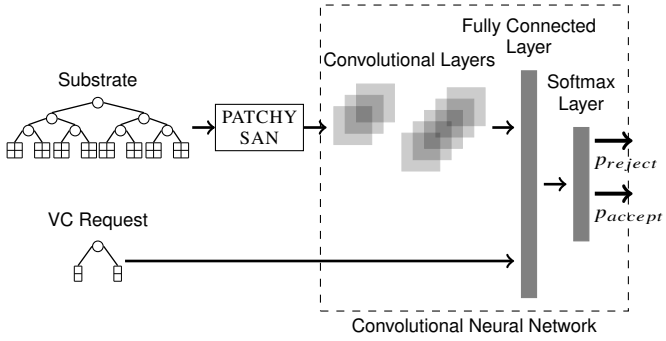


Fig. 6: Example for a complete pipeline realization within ISMAEL: The substrate is put into PATCHY-SAN, which forwards the transformed data to the input of the convolutional neural network. The VC request bypasses the convolutional layers and is directly fed into the fully connected layer of the CNN architecture. The fully connected layer is connected to the softmax output layer providing the two probabilities: one for accept and one for reject.

types (Section V-B). Afterwards, we compare the online performance of ISMAEL with a greedy admission control strategy and AHAB (Section V-C1). Finally, we vary the scenario to evaluate whether ISMAEL's performance generalizes to scenarios it has not been trained for (Section V-C2 and V-C3).

A. Training & Evaluation Setup

Substrate. The substrate C is a three-layer Fat-Tree with parameter $k = 12$, resulting in 432 hosts in total. A host has a compute capacity of 8 CUs and 8 BUs on the connecting link which leads to a total of 3 456 CUs available in the substrate. The links between the ToR switches and the aggregation switches and the links between the aggregation switches and the core are not oversubscribed, i.e., the links from ToR to aggregation switches have 48 BUs and those from aggregation to core switches have 288 BUs.

Virtual Cluster Requests. The VCs arrive according to a Poisson process with an arrival rate λ and discrete arrival times with a fixed spacing of 1 time unit, i.e., we use the Poisson process to determine the number of requests that arrive per time instance. Although multiple requests arrive at the same time, they are handled one-by-one by the admission control. The requests have geometrically distributed durations ($\mu = 15$), such that they induce a system load level of 200% ($\lambda = 4$). This load creates the possibility for the admission control to actually perform a selection of requests that are most valuable and also results in $\approx 50\%$ of the VCs being accepted by AHAB, i.e., does not require balancing the data [4]. If not stated otherwise, the attributes of the VCs are uniformly distributed. #VM(VC) is sampled from the interval $[3, 60]$ while $\text{CU}_{\text{VM}}(\text{VC})$ and $\text{BU}_{\text{VM}}(\text{VC})$ are from $\{1, 2, 4, 8\}$. All outcomes are sampled independently. To collect samples for training the classifiers, we perform 200 runs with 1000 arriving VCs and use AHAB with 20 sequences of 15 requests to perform the admission control. For evaluation, the values are aggregated over 30 runs different from those that provide the samples.

Data Representations. The HISTOGRAM uses the following bin configuration: For $\text{CU}_{\text{VM}}(\text{VC})$ and $\text{BU}_{\text{VM}}(\text{VC})$, there is one bin per potential value, i.e., the bins' boundaries are $\{0.5, 1.5, 2.5, 4.5, 8.5\}$. For #VM(VC), ten bins with the boundaries $\{3, 9, 15, 21, 27, 33, 39, 45, 51, 57, 63\}$ are used to cover the whole range of potential values and provide sufficient granularity. PATCHY-SAN is configured to create $w = 12$ receptive fields of size $k = 4$ such that 48 nodes in total are selected, a decrease in state size by a factor of ≈ 10 compared to RAWSUBSTRATEVC. As we grow the neighborhoods from racks, the selected nodes contain at least 12 racks, which, in empty state, provide enough resources to allocate the largest possible requests from the generation process. The stride is set to $s = 2$. The RAWSUBSTRATEVC and RAWVC representations do not have any parameters.

Classifier. We compare the four classifiers LR, ETC, FCN and CNN. The LR is fitted using the L2-norm for regularization and a regularization strength of $C = 10$. For the ETC [38], 500 trees are generated. The impurity within a node is measured by the GINI index based on at most 20 features that are selected to perform the split calculations.

FCN and CNN are both trained using the ADAM [40] optimizer with its default parametrization ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$). The learning rate is 0.001 and we execute 100 epochs with mini-batch size of 128 over the samples. The FCN consists of two hidden layers with 160 and 40 neurons respectively and rectified linear units (ReLU) as activation functions. The outputs are obtained using a two neuron softmax layer. We experienced that the usage of two output neurons results in a better performance. An intuition is, that the neural network has to learn to maximize the one output and minimize the other, as opposed to maximizing only one output. This can provide additional training signals that lead to a better performance. A detailed evaluation of the network architecture is omitted due to space constraints. To make the training more robust and reduce overfitting, we use a dropout probability of $p_{dropout} = 0.3$. The structure of the CNN is based on [41]. The first convolutional layer has 16 filters of size 8 and applies a stride of 4. The second convolutional layer consists of 32 filters of size 4 and uses stride 2. The third layer is fully connected (256 neurons) and again the output is given by a two neuron softmax layer.

All activation functions are ReLUs and dropout of $p_{dropout} = 0.3$ is applied during training. Note that for the RAWVC representation, the CNN only consists of a fully connected layer due to the skip layer architecture. As this is basically the structure of the FCN classifier, we train only the FCN, ETC and LR with RAWVC.

Metrics. Various works [42], [43] have used the acceptance ratio of VC embeddings in order to measure its performance. This metric, however, can be biased towards algorithms that accept a large number of small requests instead of few big ones. This, however, might not result in an efficient cluster utilization. Therefore, the first objective of this analysis is the maximization of the used CUs in the substrate (CU utilization, cf. Eq. (5)). We calculate this in every time-step of a simulation run. One sample in our evaluation corresponds to one simulation run and is computed by averaging over all samples of a run after removing transient phases.

In order to also investigate the efficiency in terms of network footprint, the minimization of the footprint of the embedded

TABLE IV: Training duration in seconds for different classifiers, data representations and amount of samples. RAWSUBSTRATEVC consumes most time for all classifiers and CNN takes longest for all representations.

	LR		ETC		FCN		CNN	
Num. samples	200k	26k	200k	26k	200k	26k	200k	26k
RAWSUBSTRATEVC	185	25	380	34	766	93	2332	275
HISTOGRAM	2.76	0.2	909	111	227	37	1458	175
PATCHY-SAN	9	1.2	166	16	172	23	471	63

VCs $F(VC)$ is evaluated. The footprint of a VC is the amount of bandwidth that is reserved on the physical links for this VC (see Fuerst et al. [34]). For instance, the VC in Fig. 2 occupies three empty hosts. The optimal embedding fills up one rack and uses one host of another rack. Bandwidth reservations are made on 5 physical links and $F(VC) = 20$.

Baseline Algorithms. The online performance evaluation compares ISMAEL to the greedy admission control (GRD) and AHAB. AHAB runs 20 sequences of 15 requests to collect data for the decision. All VC embeddings are performed using KRAKEN [34] which renders embeddings with minimal footprint given the state of the substrate. KRAKEN builds on the fact that given the location of the virtual switch, the VMs can be allocated greedily. It iterates over all potential locations of the virtual switch and selects that allocation with the minimal footprint. This results in a runtime which is roughly linear with the size of the substrate. For more detailed information and proofs the reader is referred to [34].

B. Classifier Performance and Learning Data Analysis

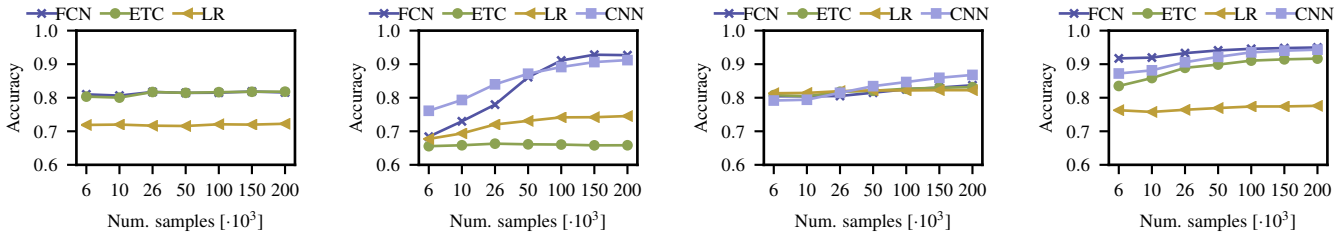
First we want to assess, how the classifiers and data representations perform and how much learning data is required to obtain good prediction qualities. As the dataset is balanced and both mis-classifications, falsely accepting and falsely rejecting a request, can harm the overall objective in a similar way, we focus on the accuracy as main performance indicator.

Generally, the training times for all classifiers and configurations were below one hour as Table IV shows. Most time was consumed when using the RAWSUBSTRATEVC representation. Using less samples reduces the training duration but must be traded-off against the performance of the classifiers and representation that is evaluated in the following. Fig. 7 shows the impact of the number of samples used during the training process on the prediction accuracy on the test set for RAWVC, RAWSUBSTRATEVC, HISTOGRAM and PATCHY-SAN. The accuracy of the classifiers with RAWVC is not affected by the number of samples and is almost constant at ≈ 0.81 for FCN and ETC and 0.72 for LR. For RAWSUBSTRATEVC (Fig. 7(b)), the obtained accuracy values range from 0.65 to 0.92 and we observe an improvement for FCN, CNN and LR. ETC however, stagnates around 0.65. The performance using the HISTOGRAM representation (Fig. 7(c)) is only little affected by the number of samples. Already with 6k samples, an accuracy > 0.65 is achieved by all classifiers, and they all perform similarly. For PATCHY-SAN (Fig. 7(d)), LR achieves $\approx 78\%$ accuracy and is insensitive to the number of samples in the shown range. The other three classifiers benefit from more samples and improve their performance to an accuracy of 0.94 for CNN and FCN with $200k$ samples. ETC achieves slightly lower performance around 0.9. RAWSUBSTRATEVC and PATCHY-SAN obtain similar accuracy with FCN and CNN around 0.94.

Comparing the number of required training samples to obtain such a high accuracy shows that PATCHY-SAN achieves an accuracy > 0.9 using only a fraction of the samples that RAWSUBSTRATEVC requires. More specifically, training the FCN with PATCHY-SAN and $6k$ samples already provides an accuracy > 0.9 while training with RAWSUBSTRATEVC requires $> 100k$ samples to reach this value. Thus, PATCHY-SAN significantly decreases the computational effort for learning compared to the RAWSUBSTRATEVC and additionally, obtains slightly higher classification performance. An explanation for this improvement is found in the sizes of the data representations. With the given substrate configuration, RAWSUBSTRATEVC has a size of 1551 hence has a larger state space than PATCHY-SAN which has a size of 147. Consequently, RAWSUBSTRATEVC requires a larger number of samples to sufficiently cover the state space. Furthermore, in comparison to PATCHY-SAN, RAWSUBSTRATEVC does not sort the nodes and by this does not reduce symmetries of the states. The reduction of symmetry however, would shrink the state space and thereby further reduce the amount of samples required for sufficient coverage.

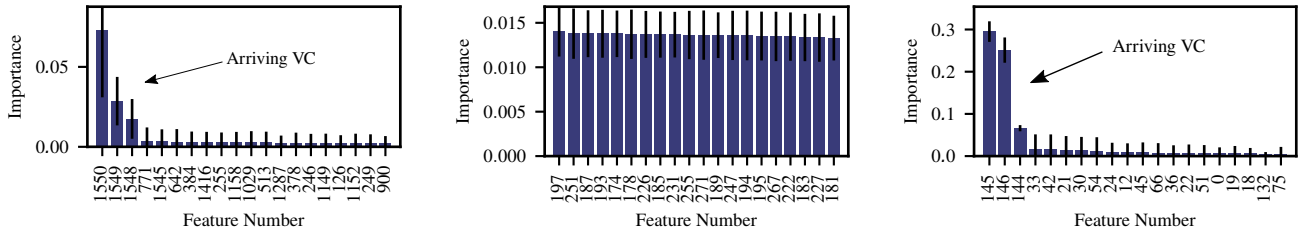
For the impact of the data representation on the performance of a classifier, we observe that FCN and CNN obtain the best performance with RAWSUBSTRATEVC and PATCHY-SAN. ETC performs best with PATCHY-SAN. Only LR performs best when using HISTOGRAM but achieves only an accuracy of about 0.81. The RAWVC representation leads to the worst results for all classifiers except for ETC which is worst with RAWSUBSTRATEVC. This indicates, that information about substrate state is generally leveraged by the classifiers and hence useful for better performance.

To understand why the behavior is different in the case of ETC, we consider the feature importance derived from it. Fig. 8 shows the mean values of obtained feature importances over all trees in the ETC of the 20 most important features for RAWSUBSTRATEVC, HISTOGRAM and PATCHY-SAN. The values on the abscissa represent the position of the feature in the feature vector. The results for HISTOGRAM on one hand and RAWSUBSTRATEVC and PATCHY-SAN on the other hand differ significantly. For the former one, the importance of the features slowly decreases. This illustrates that many features are evenly important. In contrast to this, there is a significant dominance of three features in the case of RAWSUBSTRATEVC and PATCHY-SAN. The feature numbers indicate that these features represent the attributes of the arriving VC request. The worse performance of ETC for RAWSUBSTRATEVC is thus counter-intuitive, since conceptually more information is available in RAWSUBSTRATEVC compared to PATCHY-SAN. The ETC cannot easily use this additional information, since it does not learn a data transformation that could allow it to capture the necessary information, in contrast to the other approaches. The ETC has to separate between accept and reject solely by combining the presented features. Since the size of RAWSUBSTRATEVC is very large, the ETC cannot extract rules that are predictive for the outcome of a specific request. The variance in the substrate features is



(a) RawVc (b) RawSubstrateVc (c) Histogram (d) Patchy-san

Fig. 7: Offline comparison of accuracy for different classifier configurations against the number of samples. Sub-figures compare the data representation types. Training with > 150k samples obtains best results.



(a) RawSubstrateVc (b) Histogram (c) Patchy-san

Fig. 8: Feature importance of ETC. Figures show the top 20 features by their index in the feature vector. Bars indicate mean values and standard deviation over all trees in the ETC. For RawSubstrateVc and Patchy-san, the features with the highest indices, i.e., at the end of the vector, represent the arriving VC (1548 – 1550 for RawSubstrateVc, 144 – 146 for Patchy-san). In both cases, these features show a strong dominance over the substrate related features.

too large, since there are too many possibilities in which a request could be accepted or rejected. The substrate features in RawSubstrateVC thus act as noise variables, which have a detrimental effect on model performance [44].

In summary, the number of training samples has an impact on the performance of the classifiers. The strength of this impact varies depending on the data representation and the classifier. This allows to choose a combination of classifier and data representation that trades off performance of Ismael against the computational effort for training. In the evaluated scenario, the best classification performance is achieved with Patchy-san and FCN. Moreover, the previous analysis demonstrates that adding substrate state information is beneficial to the classifiers performance. Therefore, we omit the analysis of the RawVc representation in the following.

C. Online Performance Analysis

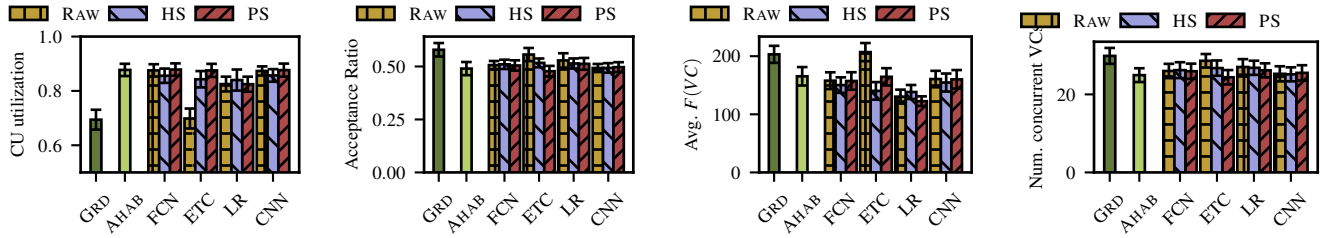
Analysis of the classifiers' performance gives already first insights on the potential of Ismael. As we expect that decisions have an immediate impact on the states the system transitions into and wrong decisions may lead to degradation of the CU utilization over time, we evaluate now the performance of Ismael in an online setting. The analysis focuses on three aspects. First, we compare the performance of Ismael to that of Grd and Ahab using the same scenario as for the training data generation. Later, Ismael is stressed with different scenarios to check how it generalizes to scenarios with larger substrates and different request distributions.

1) Can Ismael *keep up with resource-intensive simulation-based approaches?:* Fig. 9 visualizes the VC embedding performance metrics for Grd, Ahab and Ismael with all classifiers and data representations. The bars indicate the mean value with the corresponding 95% confidence interval.

Fig. 9(a) shows CU utilization. Ahab obtains a mean value of 0.87. For Ismael, the mean CU utilization varies with the used data representation. All combinations of data representation and classifier outperform Grd except RawSubstrateVc in combination with ETC as already observed in the offline analysis. This combination achieves by far the lowest performance at the level of Grd. The difference between other classifiers with RawSubstrateVc, Histogram and Patchy-san and is smaller. For FCN, CNN and LR, RawSubstrateVc and Patchy-san obtain the same substrate utilization. Moreover, Patchy-san reaches the performance of Ahab with FCN, ETC and CNN while Histogram misses 0.02 – 0.05 to reach Ahab. These observations coincide with those from Sec. V-B, i.e., CNN and FCN with Patchy-san have the most accurate adoption of Ahab's behavior.

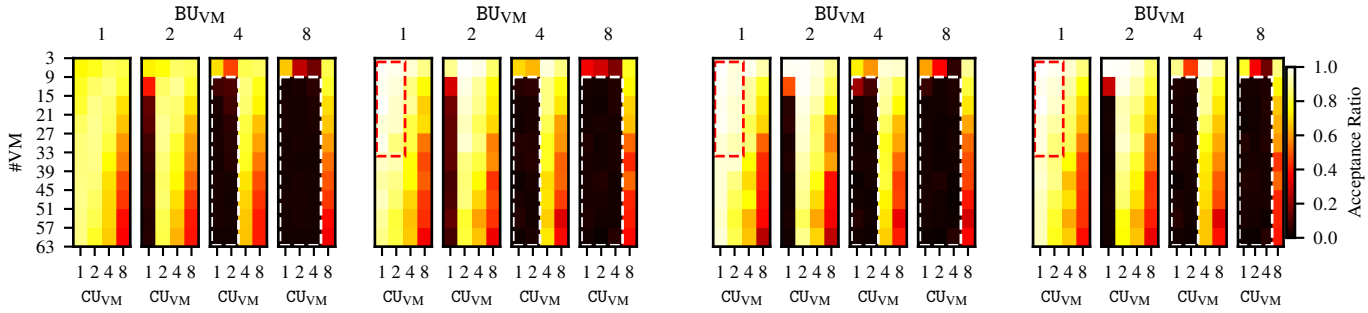
To shed light into why the combinations with LR and ETC with RawSubstrateVc perform worse, we consider the obtained acceptance ratios (Fig. 9(b)) and the number of concurrently embedded requests (Fig. 9(d)). In particular for RawSubstrateVc with ETC and LR, Ismael accepts slightly more VCs than Ahab and attains acceptance ratios around 0.58 compared to 0.5. Also for the number of concurrently embedded VCs, we observe higher that RawSubstrateVc results in higher average values around 29 compared to 25 for Ahab and Ismael with the other data representations. This means that Ismael, favors small requests more than Ahab, as both use Kraken as embedding algorithm and yield embeddings with minimal footprints.

Fig. 10 shows heatmaps of the acceptance ratios separated by the attributes of the arriving VC for Ahab and Ismael



(a) CU utilization (b) Acceptance ratio (c) VC footprint (d) Num. concurrent VCs

Fig. 9: Online performance comparison between GRD, AHAB and ISMAEL for all classifiers. The sub-figures show results for CU utilization, acceptance ratio, the average weighted footprint of a VC and the number of concurrently allocated VCs. The bars show the mean values with the 95% confidence intervals. ISMAEL with neural network and PATCHY-SAN achieves CU utilization close to AHAB.



(a) AHAB (b) RAWSUBSTRATEVC (c) HISTOGRAM (d) PATCHY-SAN

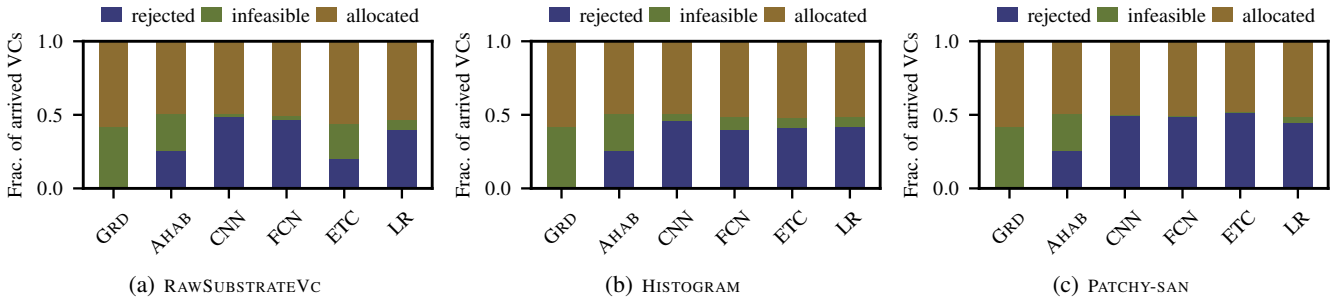
Fig. 10: Comparison of online acceptance ratio separated by VC attributes. The pixel of a heatmaps shows the value for the corresponding group of VCs. Sub-figures compare AHAB to ISMAEL with RAWSUBSTRATEVC, HISTOGRAM and PATCHY-SAN as data representations. Classifier is FCN. In all cases, the major acceptance pattern of AHAB is adopted.

with the different data representations. The results are obtained using FCN. AHAB demonstrates a strong preference for VCs with $\frac{\text{CU}_{\text{VM}}(\text{VC})}{\text{BU}_{\text{VM}}(\text{VC})} > 1$ as almost all VCs where the number of requested CUs is smaller than the number of requested BUs per VM are rejected (Fig. 10(a), white-dashed rectangle). For ISMAEL, all data representations employ a very similar acceptance behavior (Fig. 10(b,c,d)) but show generally higher acceptance ratios for requests with $\text{BU}_{\text{VM}}(\text{VC}) < 2$ and $\text{CU}_{\text{VM}}(\text{VC}) < 2$, for instance in the regions indicated by the red-dashed rectangles. Thus, ISMAEL with FCN filters bad requests and obtains similar performance as AHAB.

To visualize how ISMAEL reduces the computational effort, Fig. 11 shows the average fractions of requests in a simulation run split by the result of classification and embedding. The lower part of a bar (blue) indicates how many requests are rejected by ISMAEL, i.e., where the expensive embedding algorithm is never called ($\mathcal{VC}^{\text{Rej}}$). The middle part (green) shows the requests that ISMAEL accepts but are not feasible ($\mathcal{VC}^{\text{Acc}} \cap \mathcal{VC}^{\text{Inf}}$). These are requests which are misclassified and lead to unnecessary calculation of an embedding. Lastly, the upper part (brown) are the admitted and successfully allocated requests ($\mathcal{VC}^{\text{Acc}} \cap \mathcal{VC}^{\text{Fea}}$). Here, the call of the embedding algorithm is necessary and beneficial. As already seen by the acceptance ratio (Fig. 9(b)), this part makes approximately 50% of the requests. The potential for saving computations compared to GRD and AHAB is limited to the remaining part involving the infeasible and rejected requests.

For all combinations of classifiers and data representations, we observe that ISMAEL rejects more than 15% of the requests (lowest value from Fig. 11(a), RAWSUBSTRATEVC with ETC). Consequently for our settings, simulation runs of 1000 requests, ISMAEL determines in the worst case 850 embeddings while already GRD determines one embedding per arriving request, i.e., 1000 embeddings per run. AHAB determines with the used configuration up to $600k$ embeddings per simulation run.[2] Thus, even with the worst combination of classifier and data representation the online computational effort of admission control can be significantly reduced. Comparing the data representation and classifiers more in detail reveals that choosing ISMAEL's configuration more wisely reduces the computational effort even further. Again RAWSUBSTRATEVC with ETC yields the worst results. But RAWSUBSTRATEVC with FCN reduces the calculated embeddings per run to 500 for 1000 arriving requests. HISTOGRAM reduces this number for combinations by at least 40% and PATCHY-SAN in combination with CNN, FCN or ETC reduces the number of falsely accepted requests to almost 0. In these cases, embeddings are only determined for VCs that are allocated, i.e., the computational effort is reduced to a minimum which pays off the initial training effort in the long run. In addition to reducing the overall computational effort, ISMAEL also provides smaller decision delays which in turn enables a more agile operation of the system. Fig. 12 shows the cumulative distribution function of the decision delay comparing AHAB and ISMAEL with FCN and PATCHY-SAN. ISMAEL provides delays two magnitudes smaller than AHAB. Moreover, for the latter one, the decision delay varies with the runtime of the embedding algorithm while ISMAEL's delay is independent of that. The decision delay with ISMAEL varies only little for $\geq 99\%$ of the arriving requests while we observe strong variations in case of AHAB.

[2]For every request, AHAB runs $2 \cdot 20$ sequences with each 15 arriving requests, i.e., embeddings



(a) RAWSUBSTRATEVC (b) HISTOGRAM (c) PATCHY-SAN

Fig. 11: Actions performed by GRD, AHAB and ISMAEL(CNN,FCN,ETC,LR) on arriving requests. The bars show the average fraction of requests per simulation that are rejected by the classifier (blue), forwarded to the embedding algorithm but infeasible (green) and successfully allocated (brown). CNN and FCN in combination with PATCHY-SAN minimize the number of determined embeddings most.

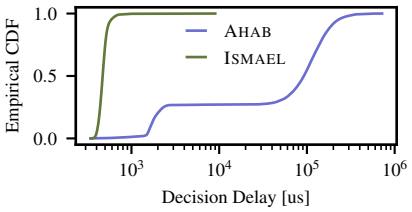


Fig. 12: Empirical CDF of the decision delay of AHAB and ISMAEL(FCN, PATCHY-SAN) on arriving requests. ISMAEL provides more stable and significantly smaller decision delays than AHAB.

The results show that ISMAEL can outperform the VC embedding solutions without admission control, and is able to obtain the performance of AHAB. It reduces the online computational effort and delay of decision making in comparison to AHAB. On the other hand, the ISMAEL requires some effort to train the classifiers. So it might be beneficial to re-use classifiers for multiple scenarios.

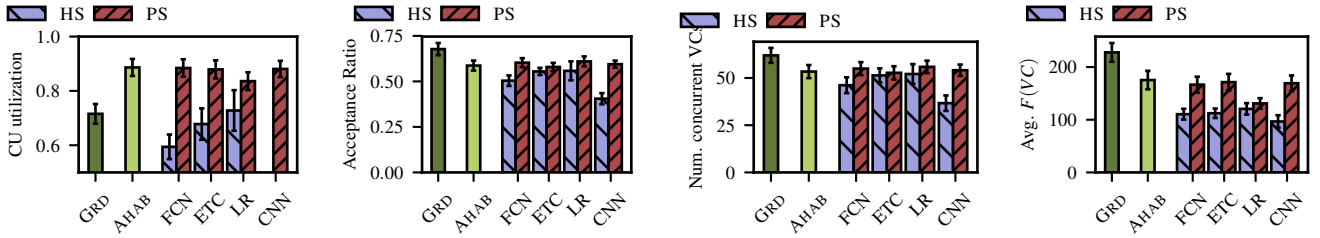
2) Does ISMAEL *generalize to larger substrates?:* Next, we assess the question whether the previously trained and evaluated classifiers perform well in a larger scenario, i.e., whether ISMAEL generalizes to scenarios with substrate sizes it has not been trained for. The data representations HISTOGRAM and PATCHY-SAN provide feature vector sizes that are independent of the size of the substrate while the size of RAWSUBSTRATEVC depends on the size of the substrate. Thus, with the latter one, classifiers cannot be reused to derive decisions on substrates with different sizes. Therefore, we evaluate how classifiers that were trained using HISTOGRAM and PATCHY-SAN on samples from a substrate with 432 hosts perform on a larger substrate with 1024 hosts (Fat-Tree with $k = 16$). The resources per host remain constant (8CUs, 8BUs). Further, we fix the VC generation process but increase the arrival rate to maintain the offered system load at $\approx 200\%$.

Fig. 13 shows mean values and 95% confidence intervals of the performance metrics. In contrast to Fig. 9, there is a significant difference when comparing HISTOGRAM and PATCHY-SAN. PATCHY-SAN obtains for all classifiers but LR a performance similar to that of AHAB while HISTOGRAM only reaches the utilization of GRD when using LR. Additionally with HISTOGRAM, the size of the confidence intervals increases. The HISTOGRAM representation does not cover the actual resource utilization of the substrate. With the larger substrate, in total more requests can be allocated. With HISTOGRAM however, the classifiers mimics the pattern of acceptance ratios for the case of the smaller substrate. Thus, in this case, ISMAEL rejects VCs although a sufficient amount of resources is available. In contrast, PATCHY-SAN encodes the actual substrate resources. Furthermore, PATCHY-SAN always returns the nodes with most remaining resources and thereby pre-filters the state space for the classifier rendering it invariant to varying substrate sizes. It adapts to situations where more resources are available. As a consequence, ISMAEL with PATCHY-SAN uses the additional resources and obtains the same substrate utilization as AHAB. Thus, using PATCHY-SAN, ISMAEL can generalize to scenarios with larger substrates.

3) Does ISMAEL *generalize to other request distributions?:* Another dimension of generalization are the distributions of the attributes of the VCs. Up to now, the evaluation considers uniform distributions within each of the VC attributes. This is not a realistic setting as shown by analyses of traces from Google [45] and Microsoft [46]. Accordingly, we evaluate now whether ISMAEL can use the classifiers trained with the uniform distribution set to infer decisions when the requests are generated using a more skewed distribution set similar to those mentioned in [45] and [46]. The number of requested VMs $\#\texttt{VM}(\texttt{VC})$ is exponentially distributed with mean 20 in the interval $[3, 60]$. $\texttt{CU}_{\texttt{VM}}(\texttt{VC})$ and $\texttt{BU}_{\texttt{VM}}(\texttt{VC})$ both follow a discrete distribution with $P(1) = 0.45, P(2) = 0.3, P(4) = 0.2, P(8) = 0.05$. Consequently, more small requests arrive while the probability for a very large request is low.

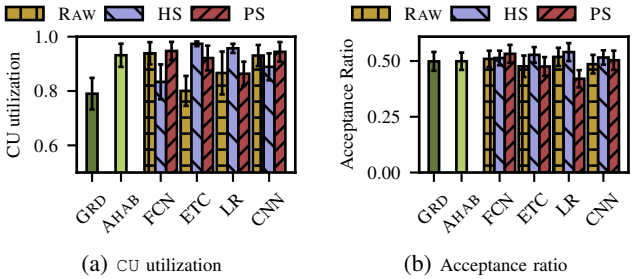
Fig. 14 visualizes the comparison of all classifiers and data representations to GRD and AHAB for acceptance ratio and CU utilization. Regarding the CU utilization, all combinations achieve values at least as good as GRD. RAWSUBSTRATEVC and PATCHY-SAN match the performance of AHAB with FCN and CNN. For HISTOGRAM, the performance with ETC and LR is better than with CNN and FCN, achieving utilizations of 0.96 which is higher than AHAB with 0.94. Analyzing the acceptance patterns in this scenario however, explains why this situation occurs and uncovers a weakness of AHAB (Fig. 15).

The main difference in the behavior of AHAB and ISMAEL is the rejection of requests with $\frac{\texttt{CU}_{\texttt{VM}}(\texttt{VC})}{\texttt{BU}_{\texttt{VM}}(\texttt{VC})} < 1$. Focusing on the requests with $\texttt{BU}_{\texttt{VM}}(\texttt{VC}) = 8$, ISMAEL with HISTOGRAM and ETC rejects all requests with $\texttt{CU}_{\texttt{VM}}(\texttt{VC}) < 8$ and $\#\texttt{VM} > 9$ while AHAB accepts some of them. This group of requests



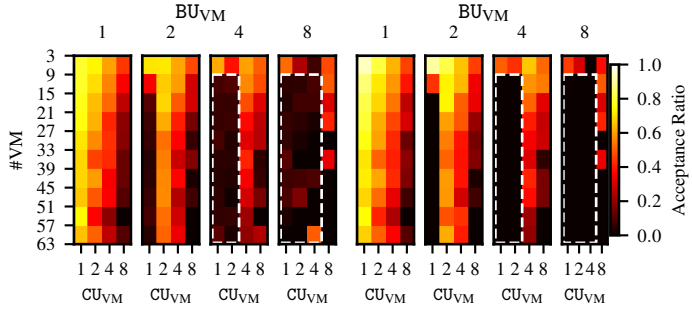
(a) CU utilization (b) Acceptance ratio (c) Num. allocated VCs (d) Avg. allocation costs

Fig. 13: Online performance comparison between GRD, AHAB and ISMAEL for all classifiers and data representations on scenarios with a larger substrate. The sub-figures show results for CU utilization, acceptance ratio, the average weighted footprint of a VC and the number of concurrently allocated VCs. The bars show the mean values with the 95% confidence intervals. PATCHY-SAN enables the classifiers to perform well also in scenarios with larger substrates.



(a) CU utilization (b) Acceptance ratio

Fig. 14: Online performance comparison between GRD, AHAB and ISMAEL for all classifiers and data representations on scenarios with skewed request generation. The sub-figures show results for CU utilization and acceptance ratio. The bars show the mean values with the 95% confidence intervals. ISMAEL can reach and even outperform AHAB also in scenarios with different request distributions.



(a) AHAB (b) HISTOGRAM, ETC

Fig. 15: Comparison of online acceptance ratio separated by VC attributes. The pixel of a heatmaps shows the value for the corresponding group of VCs. Sub-figures compare AHAB to ISMAEL with HISTOGRAM and ETC. HISTOGRAM with ETC rejects large requests more strictly which results in higher acceptance of small requests compared to AHAB.

is generally not beneficial for the substrate utilization as it imposes high costs in terms of occupied bandwidth. However, depending on the specific request sequences that AHAB evaluates they might be accepted in some cases (Fig. 15(a)). Since it is trained on uniform request distributions (cf. Fig. 10), ISMAEL rejects these requests more rigidly (Fig. 15, white-dashed rectangles) and as a consequence, can accept more small requests, e.g., with $\texttt{BU}_{\texttt{VM}}(\texttt{VC}) = 1$, that are more beneficial and fill the cluster more tightly. The result is a higher overall substrate utilization. Similar observations hold for ISMAEL with PATCHY-SAN and FCN.

In conclusion, ISMAEL is able to generalize from uniform request distributions towards skewed ones as they appear in data center traces. Furthermore, the evaluation of this scenario shows that although it significantly improves upon greedy admission control, AHAB still leaves room for further improvement, i.e., increase of the physical resource utilization.

VI. FUTURE WORK

Admission control is an essential component in any scenario where infrastructure is shared and where a predictable performance is required. While optimized admission control decisions are important to ensure a high profit and utilization, the underlying algorithmic problems are hard. Hence, fast solutions are likely to result in suboptimal decisions. Our work opens a new perspective on this seemingly inherent quality-efficiency trade-off, by leveraging the experience from past admission decisions using ML techniques like deep learning.

We believe that this also reveals several interesting directions for future research. While our proof-of-concept already shows promising results, optimizations can be performed along several dimensions. For example, it will be interesting to build on the data representations that ISMAEL provides to integrate Reinforcement Learning and develop more tailored admission policies. More generally, we believe that the ideas underlying ISMAEL are of more general interest, not only for other fundamental admission control problems, but also for other network algorithms solving hard problems repeatedly.

ACKNOWLEDGMENT

This work is part of a project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No 647158 - FlexNets). The authors alone are responsible for the content of the paper.

REFERENCES

[1] J. C. Mogul and L. Popa, "What we talk about when we talk about cloud network performance," *ACM SIGCOMM CCR*, vol. 42, no. 5, pp. 44–48, 2012.

[2] J. Zerwas, P. Kalmbach, C. Fuerst, A. Ludwig, A. Blenk, W. Kellerer, and S. Schmid, "AHAB: Data-Driven Virtual Cluster Hunting," in *Proc. IFIP Networking*, 2018, pp. 1–9.

[3] M. Rost and S. Schmid, "Charting the complexity landscape of virtual network embeddings," in *Proc. IFIP Networking*, 2018, pp. 1–9.

[4] A. Blenk, P. Kalmbach, S. Schmid, and W. Kellerer, "*o'zapft is:* tap your network algorithm's big data!" in *Proc. ACM SIGCOMM Workshop Big-DAMA*, 2017, pp. 19–24.

[5] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg," in *Proc. 10th European Conference on Computer Systems*, 2015, pp. 1–17.

[6] T. Benson, A. Akella, and D. A. Maltz, "Network Traffic Characteristics of Data Centers in the Wild," in *Proc. 10th ACM SIGCOMM IMC*, 2010, pp. 267–280.

[7] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," *ACM SIGCOMM CCR*, vol. 41, pp. 242–253, 2011.

[8] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *Proc. ICML*, 2016, pp. 2014–2023.

[9] S. P. Singh and D. P. Bertsekas, "Reinforcement learning for dynamic channel allocation in cellular telephone systems," in *Advances in NIPS 9*, 1996, pp. 974–980.

[10] P. Guo, M. Zhang, Y. Jiang, and J. Ren, "Policy-based QoS Control Using Call Admission Control and SVM," in *2007 2nd International Conference on Pervasive Computing and Applications*, 2007, pp. 685–688.

[11] R. Zhang and A. J. Bivens, "Comparing the Use of Bayesian Networks and Neural Networks in Response Time Modeling for Service-oriented Systems," in *Proc. Workshop on Service-Oriented Computing Performance: Aspects, Issues, and Approaches*, 2007, pp. 67–74.

[12] A. Bashar, G. Parr, S. McClean, B. Scotney, and D. Nauck, "Machine learning based Call Admission Control approaches: A comparative study," in *Proc. CNSM*, 2010, pp. 431–434.

[13] A. Bashar, G. Parr, S. McClean, B. Scotney and Nauck, D., "Knowledge Discovery Using Bayesian Network Framework for Intelligent Telecommunication Network Management," in *Proc. International Conference on Knowledge Science, Engineering and Management*, 2010, pp. 518–529.

[14] P. Marbach, O. Mihatsch, and J. N. Tsitsiklis, "Call Admission Control and Routing in Integrated Services Networks Using Neuro-Dynamic Programming," *IEEE JSAC*, vol. 18, no. 2, pp. 197–208, 2000.

[15] H. Tong and T. X. Brown, "Adaptive call admission control under quality of service constraints: A reinforcement learning solution," *IEEE JSAC*, vol. 18, no. 2, pp. 209–221, 2000.

[16] A. Pietrabissa, "A Reinforcement Learning Approach to Call Admission and Call Dropping Control in Links with Variable Capacity," *European Journal of Control*, vol. 17, no. 1, pp. 89–103, 2011.

[17] J. Leguay, L. Maggi, M. Draief, S. Paris, and S. Chouvardas, "Admission control with online algorithms in SDN," in *Proc. IEEE/IFIP NOMS*, 2016, pp. 718–721.

[18] S. Shakeri, S. Parsaeefard, and M. Derakhshani, "Proactive admission control and dynamic resource management in SDN-based virtualized networks," in *Proc. 8th International Conference on the Network of the Future (NOF)*, 2017, pp. 46–51.

[19] G. Even, M. Medina, G. Schaffrath, and S. Schmid, "Competitive and deterministic embeddings of virtual networks," in *Proc. ICDCN*, 2012, pp. 106–121.

[20] T. Lukovszki and S. Schmid, "Online Admission Control and Embedding of Service Chains," in *Proc. International Colloquium on Structural Information and Communication Complexity*, 2015, pp. 104–118.

[21] M. A. T. Nejad, S. Parsaeefard, M. A. Maddah-Ali, T. Mahmoodi, and B. H. Khalaj, "vSPACE: VNF Simultaneous Placement, Admission Control and Embedding," *IEEE JSAC*, vol. 36, no. 3, pp. 542–557, 2018.

[22] A. Blenk, P. Kalmbach, P. van der Smagt, and W. Kellerer, "Boost online virtual network embedding: Using neural networks for admission control," in *Proc. CNSM*, 2016, pp. 10–18.

[23] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural Combinatorial Optimization with Reinforcement Learning," 2016. [Online]. Available: http://arxiv.org/abs/1611.09940

[24] H. Dai, E. B. Khalil, Y. Zhang, B. Dilkina, and L. Song, "Learning Combinatorial Optimization Algorithms over Graphs," 2017. [Online]. Available: https://arxiv.org/abs/1704.01665

[25] Z. Li, Q. Chen, and V. Koltun, "Combinatorial Optimization with Graph Convolutional Networks and Guided Tree Search," in *Advances in NIPS 30*, 2018, pp. 1–10.

[26] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Automated Configuration of Mixed Integer Programming Solvers," in *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, Berlin, Heidelberg, 2010, pp. 186–202.

[27] E. Rachelson, A. B. Abbes, and S. Diemer, "Combining Mixed Integer Programming and Supervised Learning for Fast Re-planning," in *Proc. 22nd IEEE International Conference on Tools with Artificial Intelligence*, 2010, pp. 63–70.

[28] J. Gao and R. Jamidar, "Machine learning applications for data center optimization," *Google White Paper*, 2014.

[29] K. Winstein and H. Balakrishnan, "TCP ex machina," in *Proc. ACM SIGCOMM 2013 conference*, 2013, pp. 123–134.

[30] M. Dong, T. Meng, D. Zarchy, E. Arslan, Y. Gilad, B. Godfrey, and M. Schapira, "PCC Vivace: Online-Learning Congestion Control," in *Proc. 15th USENIX Symposium on Networked Systems Design and Implementation*, 2018, pp. 343–356.

[31] F. Y. Yan, J. Ma, G. Hill, D. Raghavan, R. S. Wahby, P. Levis, and K. Winstein, "Pantheon: the training ground for Internet congestion-control research," in *Proc. 15th USENIX NSDI*, 2018, pp. 731–743.

[32] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource Management with Deep Reinforcement Learning," in *Proc. 15th ACM HotNets*, 2016, pp. 50–56.

[33] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM CCR*, vol. 38, pp. 63–74, 2008.

[34] C. Fuerst, S. Schmid, L. Suresh, and P. Costa, "Kraken: Online and Elastic Resource Reservations for Cloud Datacenters," *IEEE/ACM Trans. on Networking*, vol. 26, no. 1, pp. 422–435, 2017.

[35] R. Bellman, *Dynamic programming*. Courier Corporation, 2013.

[36] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation Learning on Graphs: Methods and Applications." [Online]. Available: http://arxiv.org/abs/1709.05584

[37] D. Zhang, J. Yin, X. Zhu, and C. Zhang, "Network Representation Learning: A Survey." [Online]. Available: http://arxiv.org/abs/1801.05852

[38] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely Randomized Trees," *Machine Learning*, vol. 63, pp. 3–42, 2006.

[39] C. Bishop, C. M. Bishop *et al.*, *Neural networks for pattern recognition*. Oxford university press, 1995.

[40] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[41] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," 2016. [Online]. Available: http://arxiv.org/abs/1602.01783

[42] D. Xie, N. Ding, Y. C. Hu, and R. Kompella, "The Only Constant is Change: Incorporating Time-varying Network Reservations in Data Centers," *ACM SIGCOMM CCR*, vol. 42, no. 4, pp. 199–210, 2012.

[43] L. Yu and H. Shen, "Bandwidth Guarantee under Demand Uncertainty in Multi-tenant Clouds," in *Proc. IEEE ICDCS*, 2014, pp. 258–267.

[44] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, ser. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001.

[45] Y. Chen, A. S. Ganapathi, R. Griffith, and R. H. Katz, "Analysis and lessons from a publicly available google cluster trace," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2010-95*, vol. 94, 2010.

[46] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini, "Resource Central: Understanding and Predicting Workloads for Improved Resource Management in Large Cloud Platforms," in *Proc. SOSP*, 2017, pp. 153–167.

Johannes Zerwas received his M.Sc. degree in Electrical Engineering and Information Technology from Technical University of Munich (TUM), Germany, in 2018. He joined the Chair of Communication Networks at the TUM as a research and teaching associate in February 2018. His research is focused on flexible and predictable network virtualization, adaptive topologies, as well as data-driven networking algorithms.

Patrick Kalmbach received his M.Sc. from the TUM, Germany, in 2017. He joined the Chair of Communication Networks at TUM in April 2017, where he is currently working as a researcher and teaching assistant. His research is focused on self-driving networks that learn to utilize flexibility offered by modern networking technologies.

Stefan Schmid is a Full Professor at the University of Vienna, Austria. MSc and PhD at ETH Zurich, Postdoc at TU Munich and University of Paderborn, Senior Research Scientist at T-Labs in Berlin, and Associate Professor at Aalborg University, Denmark. Stefan Schmid received the IEEE Communications Society ITC Early Career Award 2016.

Andreas Blenk received the Dr.-Ing. degree (Ph.D.) from the Technical University of Munich in 2018. He joined the Chair of Communication Networks at the Technical University of Munich in June 2012, where he is currently working as a senior researcher and associate lecturer. His research is focused on self-driving, flexible and predictable virtual and software-defined networks, as well as data-driven network algorithms and designs.