AHAB: Data-Driven Virtual Cluster Hunting

Johannes Zerwas*, Patrick Kalmbach*, Carlo Fuerst†, Arne Ludwig†, Andreas Blenk*,
Wolfgang Kellerer*, Stefan Schmid‡
*Technical University of Munich, Germany †Technical University of Berlin, Germany ‡University of Vienna, Austria

***Abstract*—Virtual clusters are an important concept to provide isolation and predictable performance for multi-tenant applications in shared data centers. The problem of how to embed virtual clusters in a resource efficient manner has received much attention over the last years. However, existing virtual cluster embedding algorithms typically optimize the embedding of a *single* request. We demonstrate that this can lead to fragmentation and suboptimal data center resource utilization over time. We propose an alternative in two stages: First, we describe a novel embedding algorithm, called TETRIS, which, in an effort to avoid resource fragmentation over time, takes into account the specific node-to-link resource ratios of the individual requests. While TETRIS can be suboptimal when embedding only one request, we find that it performs much better than the state-of-the-art algorithms over time. Second, we allow the algorithm to strategically *reject* individual requests, even if there are sufficient resources: our proposed algorithm, AHAB, hence selects ("hunts") useful requests over time. An important property of AHAB is that it is *data-driven*: it uses information about previous requests and embeddings. We report on extensive simulations, which demonstrate the optimization potential of TETRIS (+4%) and AHAB (+13%), compared to existing solutions such as KRAKEN and OKTOPUS. Furthermore, AHAB illustrates how data-driven algorithms can replace man-made heuristics.**

***Index Terms*—Network Virtualization, Embedding, Admission Control**

I. INTRODUCTION

Today's data analysis frameworks and cloud applications generate large amounts of traffic; hence, their overall performance depends on the network. Indeed, it has been shown that cloud applications suffer from resource interference on the network, to the extent that the application execution times may become unpredictable [1]. To overcome this, several systems have been introduced that provide isolation among different customers and ensure network conditions as required by data center applications [2]–[6].

A common resource reservation abstraction provided to the tenant is the virtual cluster (VC) [2]. A VC connects a number of virtual machines (VM) to a virtual switch at a guaranteed bandwidth. The problem of how to embed virtual clusters has already received much attention [2], [7]–[10]. Proposed systems typically optimize the embedding of a single request: Minimizing the physical resource footprint of a single VC is often stated as the goal of the algorithms [7]–[9]. However, VC embedding is usually applied in an online environment where requests arrive over time. Focusing only on a single VC while neglecting the impact on future embeddings may fragment the reserved physical resources. This can in turn harm the resource utilization over time. Instead of looking only at single VCs, we propose to leverage information about the embedded request characteristics. Indeed, recent analysis of data center traces show that request characteristics can be estimated with sufficient quality to make scheduling decisions [11]; an invaluable source to optimize data center resource utilization. By integrating information about VCs into the embedding decision, this work makes two steps to surpass the drawbacks of existing VC embedding algorithms.

First, we present a novel embedding algorithm, TETRIS, which aims to reduce fragmentation over time by accounting for the ratio of requested node and link resources (which can differ from request to request), compared to the available resources in the substrate.

Second, we extend our study to admission control algorithms: we allow algorithms to strategically reject requests even though the substrate would provide enough resources to host the current to-be-embedded VC. In particular, this paper proposes AHAB[1] — a data-driven approach to admission control. The key idea of the data-driven paradigm is to base the decisions on observations from collected data instead of relying on manually designed strategies and it has recently drawn attention in networking research [12].

AHAB exploits knowledge about the characteristics of VCs. More specifically, it uses distributions of the VC attributes (VMs, bandwidth) to generate requests and evaluate the impact of the new VC on the feasibility of future embeddings. Concretely, AHAB answers the question whether the current VC will negatively affect the data center utilization in the future. To do so, AHAB performs several small simulations and compares their outcomes for two cases: one where the new VC is accepted and one where it is not. As it relies on a data-driven concept only, AHAB is independent from embedding algorithms, i.e., any VC embedding algorithm can be used in combination with AHAB. Hence, it can easily extend existing cluster management systems.

Simulations show that TETRIS outperforms state-of-the-art single-request embedding algorithms, enabling providers to host more VCs and hence use their infrastructure more efficiently. Moreover, the evaluation demonstrates that data-driven admission control can greatly improve the resource utilization in data centers by integrating knowledge about the distributions of the requests' attributes into the admission decision. Even when facing mismatched distributions, AHAB provides higher cluster utilizations than existing algorithms.

[1]The name AHAB refers to Moby Dick's captain Ahab, hunting sea monsters like KRAKEN or OKTOPUS (the systems upon which AHAB improves).

ISBN 978-3-903176-08-9

II. Revisiting Virtual Cluster Embeddings

This section describes our considered scenario and VC abstraction. We also revisit the VC embedding problem, and list two state-of-the-art algorithms to optimize the embedding of a single VC.

A. Virtual Cluster Abstraction: State-of-the-art

Virtual clusters [2] are the most prominent abstraction for batch-processing applications. Using VC abstraction, tenants can specify their networking demands, which introduces predictable performance guarantees. A VC request consists of the number of VMs and the bandwidth that should be reserved for each VM. If the provider embeds the request, it creates the number of equally-sized VMs and allocates them on the hosts of the substrate network. Additionally, the provider creates bandwidth reservations on the physical links such that every VM can use the requested bandwidth. Hence, the tenant is provided with the illusion of a dedicated network.

Besides this basic abstraction, extended versions have been proposed [6], [7], [10], [13]. However, existing algorithms do not specifically account for the fact that different requests can have different ratios of node and link resources: virtual cluster specifications are likely to come with different requirements [14], e.g., some requests have high requirements for computational resources but do not transfer much data while others are more network-intensive and require less computational resources.

B. Scenario Description

Table I summarizes the mathematical names and conventions in notation that are used throughout this study.

Substrate. The considered substrate networks (physical cluster) $\mathcal{C}$ hosting VCs employ a tree-like topology, e.g. Fat-Tree [15], a common data center architecture today. A set of pods is connected via core switches. Each pod consists of several racks which are connected to the aggregation switch of the pod. The racks are constituted by several hosts (or servers) that are interconnected by the top of rack (ToR) switch. The capacities of the links of the aggregation levels equal the accumulated bandwidths of the corresponding child nodes. The computational size of a physical server is measured in integer-valued compute units (CU). Similarly, the capacity of the physical hosts' links are normalized to denote the bandwidth in integer-valued bandwidth units (BU).

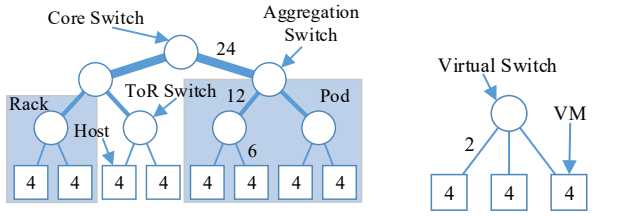
According to the physical cluster modeling in [2], [9], we approximate the Fat-Tree by a simple tree. The Fat-Tree depicted in Fig. 1a consists of two pods, containing two racks each; there are two hosts per rack. A host has a capacity of 4 CUs and the hosts' link capacities are 6 BUs. The links on aggregation and core level have capacities of 12 BUs and 24 BUs respectively.

Virtual Cluster. The VC abstraction should reflect the described observations from Sec. II-A. Customers should be able to specify their computation and communication requirements separately. Concretely, a VC is the triple $\mathcal{R} = (N, S, B)$, where N is the number of VMs, S is the computational requirement (size) of a VM and B is the bandwidth of a virtual link. All VMs are of the same computational size S, and are connected to a virtual switch at bandwidth B. For instance, the VC in Fig. 1b requests 3 VMs with a size of 4 CUs and a bandwidth of 2 BUs between the VMs and the virtual switch.

Online Cluster Arrival Process. Requests arrive in an online fashion and the provider must decide if a new request is embedded or rejected. In order to embed a VC, the provider has to fulfill all its specifications.

TABLE I
Notation and Abbreviations.

Symbol	Description
	Substrate
$\mathcal{C}$	Substrate network with a tree-like topology
CU	Compute unit: Abstract unit to measure computation requirements or capacity
BU	Bandwidth unit: Abstract unit to measure bandwidth requirements or capacity
C^h	Available computing capacity on host h [CU]
B^h	Available bandwidth on up-link of host h [BU]
FreeCapacity($\mathcal{C}$)	Number of free CUs in the substrate $\mathcal{C}$
TotalCapacity($\mathcal{C}$)	Total number of CUs in the substrate $\mathcal{C}$
Hosts($\mathcal{C}$)	Single hosts of $\mathcal{C}$ in groups of 1 sorted by avail. CUs.
Racks($\mathcal{C}$)	Hosts of $\mathcal{C}$ grouped by their racks
Pods($\mathcal{C}$)	Hosts of $\mathcal{C}$ grouped by their pods
Root($\mathcal{C}$)	Hosts of $\mathcal{C}$ in one large group
	Virtual Cluster Request
N	Number of VMs that a request has
S	Size of the VMs of a request [CU]
B	Bandwidth requirement per VM of a request [BU]
$\mathcal{R} = (N, S, B)$	Virtual cluster request with N VMs of size S interconnected with bandwidth B
VMs($\mathcal{R}$)	Virtual machines of request $\mathcal{R}$
host(vm)	Host which is assigned to the VM vm or $NULL$ if no host is assigned
$\rho(h, \mathcal{R})$	$= \frac{C^h - S}{B^h - B}$, ratio of available resources on host h after allocating one VM of request $\mathcal{R}$



(a) Fat-Tree with two pods, two racks per pod and two hosts per rack. (b) VC with $N = 3$, $S = 4$ and $B = 2$.

Fig. 1. Examples for Fat-Tree and VC.

C. Existing VC Embedding Algorithms

In this study, we focus on two prominent VC embedding algorithms: Oktopus and Kraken.

Oktopus. Ballani et al. [2] proposed a first algorithm (henceforth called Oktopus) to embed VCs in Fat-Tree datacenter topologies. Its heuristic approach aims at minimizing

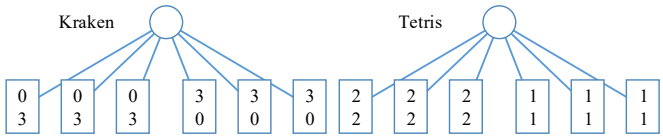


Fig. 2. Embedding behavior of KRAKEN and TETRIS. Six hosts (6 CUs, 6 BUs) are connected to a switch. Requested VCs are $\mathcal{R}^1 = (9, 1, 2)$, $\mathcal{R}^2 = (9, 2, 1)$. The upper number in a host represents mapped VMs of $\mathcal{R}^1$ and the lower number those of $\mathcal{R}^2$.

the allocation costs, but it does not always achieve optimal results. OKTOPUS iterates through the levels of the tree and searches for the first group of hosts (single host, hosts of a rack, hosts of a pod, all hosts) where the VC is feasible.

KRAKEN. An optimal solution to the single request embedding problem has been presented in [9], as part of the KRAKEN system. In contrast to OKTOPUS, KRAKEN returns the embedding with the minimal allocation cost for the given request and cluster state. To do so, KRAKEN does not return the first feasible solution, but checks all feasible solutions and returns the optimal solution for the VC. To maintain linear complexity w.r.t. the number of physical hosts, it uses the center of gravity concept, which corresponds to the location of the abstract virtual switch. It further allows to modify the size of the VC online.

Both algorithms (as well as algorithms lying between the two, like Proteus [7]) focus on single VCs and serve as comparables to the embedding algorithm presented in this study, which sacrifices quality of single embeddings to obtain better overall results. To the best of our knowledge, the challenge of fragmentation over time and the systematic study of the benefits of admission control has not been considered in the literature before.

III. TETRIS: ON THE POTENTIAL OF NON-GREEDY VC EMBEDDING

TETRIS is a VC embedding algorithm that, in an effort to perform better over time and in the long run, accounts for the specific resource ratios (and hence potential undesired resource fragmentations). As we will see, despite its simplicity, TETRIS already outperforms state-of-the-art VC embedding algorithms, which do not account for such fragmentation over time.

A. Key Idea: Sacrificing Footprint for Fragmentation

The main idea is to utilize the different resource dimensions of single hosts in a more balanced fashion, in order to avoid fragmentation (and hence poor resource utilization) over time. OKTOPUS and KRAKEN find embeddings which are dense and use low amount of bandwidth. The problem of such dense embeddings is that requests with a resource ratio $S/B \neq 1$ are collocated which wastes physical resources. Fig. 2 gives an example. For KRAKEN, $\mathcal{R}^1$ is embedded on the right three hosts. Thus, there are 3 CUs left on the host but no capacity on the up-link, which renders it unlikely that those free CUs are used in the future. On the other hand, $\mathcal{R}^2$ only uses half of the link capacity of the left three hosts. Using TETRIS, the VMs of both VCs are distributed over the hosts more evenly such that no host has capacity left for only one resource.

The ratio $\rho(h, \mathcal{R}) = (C^h - S)/(B^h - B)$ serves as a score to determine the placement of the VMs, i.e., TETRIS prefers hosts with ratio. C^h and B^h are the currently available resources on host h. As C^h is in the nominator, hosts that have much compute resources available, but only little bandwidth, are more likely chosen as location for the next VM than hosts with few compute resources available and much bandwidth.

B. Algorithm Details

Algorithm 1 shows the procedure of TETRIS. After checking the general feasibility of the request (l.1f), the algorithm iterates over the levels of the tree topology starting at the host level (similar to OKTOPUS and KRAKEN).

Trying Hosts. TETRIS iterates over every single host and tries to place all requested VMs on the same host (ll.4-9). At this stage, the resource ratio does not matter since $hostGroup$ has only one element (l.5). If the request fits on a single host, no bandwidth reservation is needed and TETRIS returns.

Trying Racks/Pods/Cluster. If the request does not fit on a single host, TETRIS iterates over the racks of the cluster (sorted by the fraction of available compute capacity). But instead of collocating as many VMs on a single host as possible (like OKTOPUS and KRAKEN), TETRIS distributes the VMs over several hosts depending on the ratio of the residual resources per host. For each VM of the request, TETRIS chooses the feasible host with the highest ratio $\rho(h, \mathcal{R})$ as location from all hosts of the current rack (l.5). Then TETRIS reserves resources for the VM on the host and the hosts' link (l.9). Bandwidth reservations on higher layers (aggregation, core) cannot be performed because the location of the virtual switch is not known yet. If any of the VM allocations fails, TETRIS resets the previously allocated VMs and proceeds with the hosts of the next rack (ll.6-8) — no host of the current rack will be used.

When all VMs are placed, TETRIS determines the virtual switch's location and performs the final bandwidth reservations (l.10f). The previous steps (l.4-9) do not guarantee the feasibility of the bandwidth reservations on the aggregation and core layer and the reservations may fail, e.g., if the Fat-Tree is oversubscribed. If this is the case, TETRIS removes the embeddings of the VMs and starts over using the hosts of the next rack (l.14f).

The same procedure is applied for the pod and root levels, if the algorithm has not found any feasible embedding after having evaluated all racks. If TETRIS does not find any feasible solution, the VC is rejected. TETRIS' complexity is linear in the number of topology host like OKTOPUS and KRAKEN.

IV. AHAB: THE CASE FOR DATA-DRIVEN ADMISSION CONTROL

We now take the idea of thinking strategically and being less greedy in how a single request is embedded one step further and initiate the study of algorithms which can even

Algorithm 1 Virtual Cluster Embedding: TETRIS

Input: Substrate $\mathcal{C}$, VC $\mathcal{R} = (N, S, B)$
Output: Embedding success

1: **if** FreeCapacity($\mathcal{C}$) $< N \cdot S$ **then**
2: **return** False
3: **for** $hostGroup \in$ {Hosts($\mathcal{C}$), Racks($\mathcal{C}$), Pods($\mathcal{C}$), Root($\mathcal{C}$)} **do**
4: **for** $vm \in$ VMs($\mathcal{R}$) **do**
5: host(vm)$\leftarrow \arg\max \rho(h, \mathcal{R}),\ h \in hostGroup$: vm is feasible on h
6: **if** host(vm)$== NULL$ **then**
7: Reset host(vm) $\forall vm \in$ VMs($\mathcal{R}$)
8: Continue with next $hostGroup$
9: Reserve S, B on host(vm)
10: $success \leftarrow$reserveBandwidth($\mathcal{R}$)
11: **if** $success$ **then**
12: **return** True
13: **else**
14: Reset $\mathcal{C}$ and host(vm) $\forall vm \in$ VMs($\mathcal{R}$)
15: Continue with next $hostGroup$
16: **return** False

Algorithm 2 Admission Control: AHAB

Input: Substrate $\mathcal{C}$, VC $\mathcal{R} = (N, S, B)$, Embedding algorithm A, $numSeq$, $numVCs$, Distributions for N, S, B
Output: Decision: Accept=True, Reject=False

1: **if** A cannot find a feasible solution **then**
2: **return** False
3: **if** FreeCapacity($\mathcal{C}$)$>$ 0.5 TotalCapacity($\mathcal{C}$) **then**
4: **return** True
5: $avgAccept$ = RunSequences($\mathcal{C}$, $\mathcal{R}$, A, $numSeq$, $numVCs$, true)
6: $avgReject$ = RunSequences($\mathcal{C}$, $\mathcal{R}$, A, $numSeq$, $numVCs$, false)
7: **return** $avgReject < avgAccept$

Algorithm 3 RunSequences

Input: Substrate $\mathcal{C}$, VC $\mathcal{R} = (N, S, B)$, Embedding algorithm A, $numSeq$, $numVCs$, $embedVC$
Output: Average reward per sequence

1: $rewards = \{\}$
2: **for** $i = 1$ **to** $numSeq$ **do**
3: $\mathcal{C}' \leftarrow$ Copy $\mathcal{C}$
4: **if** $embedVC$ **then**
5: A.embed($\mathcal{C}'$, $\mathcal{R}$)
6: $rewards[i] = 0$
7: **for** $j = 1$ **to** $numVCs$ **do**
8: $\mathcal{R}' \leftarrow$ Generate new VC
9: **if** A.embed($\mathcal{C}'$, $\mathcal{R}'$) $==$ True **then**
10: $rewards[i] \mathrel{+}= S^{\mathcal{R}'} \cdot N^{\mathcal{R}'}$
11: **return** Average($rewards$)

reject individual requests entirely, although there are sufficient resources available.

A. Key Idea: Admission Control and Leveraging Data

The admission control algorithm AHAB (Algorithm 2) shall "hunt" for the best VCs to embed. It can be configured with many single-request embedding algorithms, including TETRIS. Similar to DeepMind's AlphaGo [16] and other Monte Carlo Tree Searches [17], AHAB performs a lookahead search to make its decision. The idea is to get the impact of the embedding of a new VC on future arrivals. To do so, AHAB uses knowledge about the distributions of the VCs' attributes N, S, B to generate potential sequences of requests and tries to allocate these along with the actually arrived VC. The data collected with these small simulations is then the basis for the decision. The knowledge can be easily obtained from past requests and as a first step, we expect perfect knowledge about the distributions of N, S, B, which is an acceptable assumption as recent work has shown [11].

B. Algorithm Details

AHAB starts with checking the feasibility of the request. If the cluster is only lightly loaded, AHAB accepts the request (l.3f). This step reduces computational efforts as the probability of acceptance is high in this situation. If current load is $> 50\%$, AHAB performs the lookahead search. Given the current substrate state $\mathcal{C}$ and the new VC $\mathcal{R}$, AHAB generates a number of sequences ($numSeq$) of length $numVCs$ containing possible future requests and embeds them using the embedding algorithm A, e.g. KRAKEN (Algorithm 3). One half contains $\mathcal{R}$ while the other half does not (l.5f). Each allocated VC gives a reward of $N \cdot S$. AHAB uses the accumulated reward of the single sequences as a performance indicator and determines the mean values for the sequences with and without $\mathcal{R}$ (Algorithm 3 - ll.9-11). The comparison of these two values gives the decision of acceptance (Algorithm 2 - l.7).

The complexity of AHAB depends on the complexity of the embedding algorithm and the total number of VC allocations ($= numVCs \cdot numSequences$) in one call, which are a tunable parameters.

V. EVALUATION

In order to analyze TETRIS and AHAB in different settings, we evaluate the results obtained using event-based simulations. Besides elaborating the differences in performance (Sec. V-B & V-C), we also look at the attributes of VCs that are accepted and try to understand why AHAB outperforms the other algorithms (Sec. V-D & V-E). Furthermore, we evaluate the impact of sequence length and number of sequences (Sec. V-F). This section closes with investigating the robustness of AHAB against errors in the distributions used to generate requests and varying host capacities (Sec. V-G & V-H).

A. Setup

Substrate. The physical cluster $\mathcal{C}$ is a three-layer Fat-Tree with construction number $k = 12$ 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 cluster. The links between the ToR switches and the

aggregation switches and the links between the aggregation switches and the core are not oversubscribed.

Virtual Cluster Requests. The VCs arrive according to a Poisson process with an arrival rate λ and have exponentially distributed durations, such that they induce system load levels of 78.5% ($\lambda = 4$), 234% ($\lambda = 12$) and 390% ($\lambda = 20$). Based on the analyses of traces from Microsoft [11] and Google [18], the number of requested VMs N is exponentially distributed with mean 20 in the interval $[3, 60]$. B and S both follow a discrete distribution with $P(1) = 0.45, P(2) = 0.3, P(4) = 0.2, P(8) = 0.05$. All outcomes are sampled independently. We run every setup 30 with 1000 arriving VCs. To avoid artifacts related to the initially empty data center, we start evaluating our metrics after 100 requests.

Metrics. Various works [7], [10], [13] have used the acceptance ratio of an embedding algorithm in order to measure its performance. This metric, however, is biased towards algorithms that accept a large number of small requests instead of few bigger ones. Therefore, the first objective of this analysis is the maximization of the used CUs in the substrate. One sample is the average of this fraction over a whole run.

As second objective the minimization of the footprint $F(VC)$ of the embedded VCs is evaluated. The footprint of a VC is the amount of bandwidth of that VC that is reserved on the physical links (see Fuerst et al. [9]). For instance, the VC in Fig. 1 occupies one host per VM. The optimal embedding fills up one rack and uses one host of another rack. Bandwidth reservations are made on 5 physical links and $F(VC) = 10$.

Baseline Algorithms. The evaluation compares TETRIS with OKTOPUS and KRAKEN, all without admission control. Besides setups without admission control, the benchmark of AHAB also takes an observation-based strawman algorithm (STRAWMAN) into consideration, which simply rejects all VCs with $B > 4$ and uses KRAKEN to embed the VCs. This approach is based on the observations from TETRIS.

B. Is it worth playing TETRIS?

Fig. 3 compares the embedding algorithms for different arrival rates with admission control (AHAB with OKTOPUS, KRAKEN, TETRIS and STRAWMAN) and without admission control (OKTOPUS, TETRIS, KRAKEN). It shows the mean values over 30 runs with 95% confidence intervals. Unless otherwise stated, AHAB runs 20 sequences of 15 VCs.

Fig. 3a shows the cluster utilization w.r.t. CUs, the main objective of TETRIS and AHAB. For $\lambda = 4$, the cluster is not overloaded and all algorithms achieve similar values around 0.6. However, higher system loads, e.g., for $\lambda = 12$, allow to be more selective and make differences in the algorithms' performances visible. Considering TETRIS first, we observe that it outperforms OKTOPUS and KRAKEN for $\lambda \geq 12$ and achieves a mean CU usage of 0.83.

Yet, Fig. 3b suggests that this improvement does come at a certain cost. It shows the mean footprint of a single VC, i.e., the average number of physical link resources that are reserved for one VC. Generally, the mean values decrease with increasing arrival rates. For small arrival rates, the footprints obtained by OKTOPUS, KRAKEN and TETRIS are in a similar range (≈ 75 BUs), but for arrival rates around 12 the values for TETRIS are larger. This emphasizes the approach of TETRIS to sacrifice the footprint of the VCs to improve the utilization of the substrate. However, we observe that the gap between the average footprints of TETRIS and OKTOPUS and KRAKEN decreases further as the arrival rate increases towards 20.

The reason for this is highlighted by Fig. 3c, which shows the average number of concurrently embedded VCs. For all algorithms, this number significantly rises from 50 to ≈ 90 when the system transitions into overload and then only slightly increases further for OKTOPUS and KRAKEN. For TETRIS, it continues to grow with the arrival rate to values around 110 even though, the fraction of used CUs does not increase that much for arrival rates around 20. This implies that the average number of CUs per embedded VC decreases, i.e., TETRIS allocates more aggressively only small requests for high arrival rates while OKTOPUS and KRAKEN behave more moderately. A more detailed analysis follows later.

STRAWMAN pushes the performance of KRAKEN up and achieves cluster utilization values slightly worse than those of TETRIS (Fig. 3a). The VC footprints are smaller since requests with large bandwidth requirements are rejected; otherwise the minimal footprint for a VC is obtained. STRAWMAN trades off resource efficient embeddings with cluster utilization.

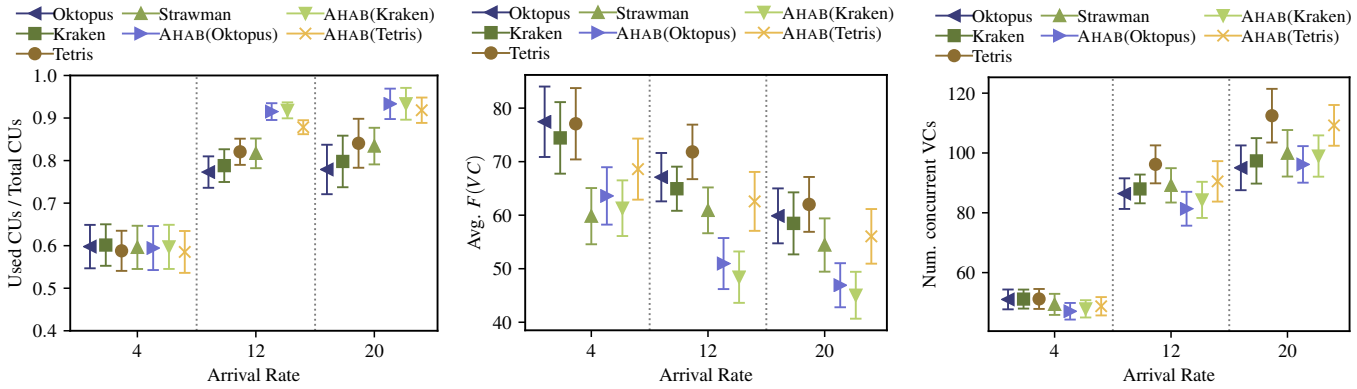
C. How useful is knowledge?

Adding admission control significantly improves the performance in terms of mean fractions of used CUs (Fig. 3a), when the system is overloaded ($\lambda \geq 12$). OKTOPUS and KRAKEN in combination with AHAB perform similar and the utilization exceeds 90%. A detailed explanation why both outreach TETRIS follows in Sec. V-F. Considering the average footprint of a single VC, AHAB(OKTOPUS) and AHAB(KRAKEN) show the best results with an average < 50 BUs for $\lambda \geq 12$. The other algorithms obtain mean values > 55 BUs. For $\lambda = 4$, the STRAWMAN dominates. AHAB(TETRIS) again results in increased footprints compared to AHAB(OKTOPUS) and AHAB(KRAKEN), e.g., for $\lambda = 12$, the mean value is ≈ 62 BUs. Fig. 3c suggests that AHAB also accepts more smaller requests as the number of concurrent VCs continues to rise with the arrival rate; however, it increases less than TETRIS without admission control.

To summarize, TETRIS improves the utilization of the substrate but is outperformed by solutions that incorporate admission control based on knowledge of the request generation process, i.e., the distributions of the VC attributes N, S, B. TETRIS is a credible alternative in case no knowledge is available or inaccessible.

D. Why is AHAB better?

In order to shed light on the reason behind AHAB's good performance, we look at the acceptance patterns of the algorithms. Fig. 4 visualizes the acceptance ratio of different request sizes for KRAKEN, TETRIS and AHAB(KRAKEN). The values are obtained from all requests of all runs.



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

Fig. 3. Performance comparison between embedding algorithms without admission control, with strawman admission control (only KRAKEN) and with AHABagainst the different requests' arrival rates. The subfigures show results for the fraction of used CUs, the average weighted footprint of a VC and the number of concurrently allocated VCs. The figures show the mean values with the 95% confidence intervals.

KRAKEN The color of a pixel corresponds to the acceptance ratio derived from the requests with that size. For instance, the upper left pixel of the block $B = 1$ in Fig. 4a means that KRAKEN accepts 80% of the requests with VM size $S = 1$, bandwidth requirement $B = 1$ and $3 \leq N < 9$. Somehow intuitive, higher acceptance ratios show up for smaller requests and the values decrease for larger requests. Especially for $B = 8$ or $S = 8$, KRAKEN is not able to embed many requests, as these occupy a whole host link or host. Still, KRAKEN allocates some of these requests. A higher number of requested VMs also decreases the acceptance ratio. For small VM sizes ($S \leq 2$), this effect is moderated by the requested bandwidth: For $B \leq 2$, the acceptance ratio is ≥ 0.4 for all bins of Num. VMs. For $B = 4$, the acceptance drops for requests with more than 33 VMs and for $B = 8$, the acceptance already drops to 0.2 for requests with 10 VMs.

TETRIS Fig. 4b shows the same representation for TETRIS. It supports the observations from Sec. V-B. Generally, TETRIS obtains higher acceptances ratios for VCs with small VM sizes and lower number of VMs. For instance, the acceptance ratio is ≥ 0.6 for requests with $B = 1$ and $S = 1$ regardless of N, while KRAKEN achieves these ratios only for requests with less than 27 VMs. But KRAKEN allocates more requests that occupy whole hosts or host links ($S = 8$ and $B = 8$). In particular for $B = 8$, the acceptance ratio of TETRIS and is less or equal to that of KRAKEN for almost all cases. This observation is the basis of the strawman admission control algorithm that was introduced before. Additionally for $S = 8$, TETRIS allocates only $\leq 40\%$ of the requests.

AHAB The behavior of TETRIS might not be optimal, as TETRIS does not perform best among the algorithms. Indeed, AHAB(KRAKEN) selects different VCs as Fig. 4c illustrates. For small bandwidths ($B = 1$), AHAB admits and embeds at least as many requests as KRAKEN without admission control. For $B > 1$, we observe that it embeds less requests with small VMs ($S = 1$). In this case, the acceptance ratio drops below 20% for requests with more than 15 VMs. But for VCs with larger VMs, AHAB obtains an acceptance ratio that is $5 - 10$ percentage points higher compared to KRAKEN in many cases.

In conclusion, TETRIS and AHAB increase the utilization of the substrate network but employ different acceptance patterns to do so.

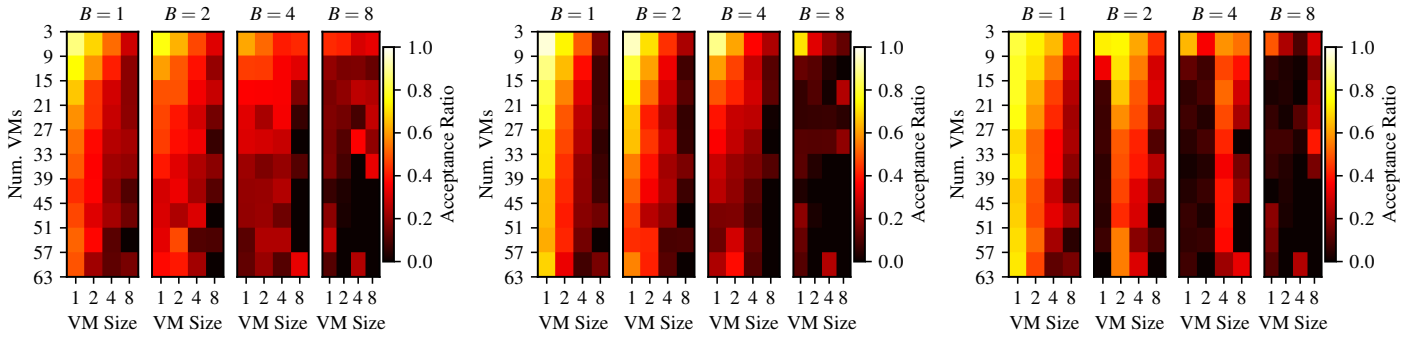
E. Which requests are valuable?

To understand why AHAB's acceptance pattern performs better than that of TETRIS, we look at the acceptance ratio from a different point of view: the value of a VC to the cluster utilization. Fig. 5 shows the acceptance ratio grouped by the resource ratio $\tilde{\rho} = \frac{S}{B}$ of a request and compares the values for KRAKEN and TETRIS without admission control and AHAB(KRAKEN). The previous observation is only weakly affected by the number of VMs in a request, which allows to reduce the dimensionality of the representation.

Small ratios mean that the allocation increases the target metric (used CUs) only little while occupying many network resources. Regardless the difference in absolute values, we note that KRAKEN and TETRIS have higher acceptance for VCs with $\tilde{\rho} \leq 1$ and reject VCs with $\tilde{\rho} > 1$ more likely. This is somehow counterintuitive as the benefit is low, while the probability for high allocation costs is high and further reflects that no explicit admission control is performed. In contrast to this, AHAB(KRAKEN) picks VCs with $\tilde{\rho} > 1$ as indicated by the acceptance ratios. For $\tilde{\rho} < 1$, the average acceptance ratio is 0.23, while it is 0.45 for VCs with $\tilde{\rho} > 1$. Thus, AHAB(KRAKEN) admits more valuable requests and thereby compensates the drawbacks of OKTOPUS and KRAKEN in comparison to TETRIS.

F. Optimization Opportunities: Can we save data?

The parameters that AHAB has used up to now ($numVCs = 15$, $numSeq = 20$), obtain the best results. However, it is generally desirable to minimize the computational overhead of AHAB. Therefore, we analyze the impact of the number of requests per sequence ($numVCs$) on AHAB's performance and also evaluate how sensitive the results are against the number of sequences ($numSeq$) that AHAB runs.



(a) KRAKEN (b) TETRIS (c) AHAB(KRAKEN)

Fig. 4. Comparison of acceptance ratio separated by VC specification. Each pixel of the heatmaps shows the value for the corresponding group of VCs. Subfigures allow to compare KRAKEN, TETRIS and AHAB(KRAKEN) and illustrate the differences in selection behavior. Arrival rate is 20. Note that Num. VMs is grouped into bins of size 6.

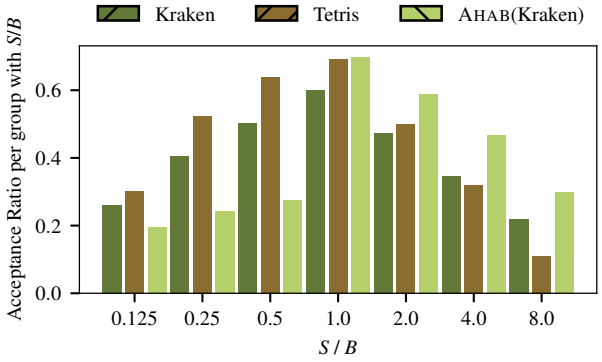
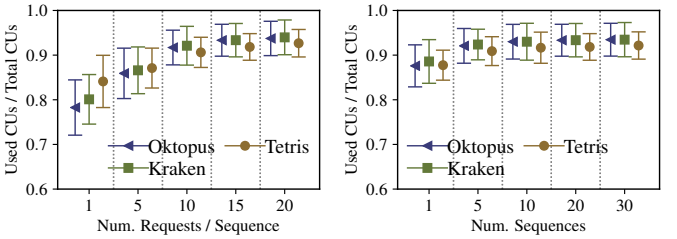


Fig. 5. Acceptance ratio per group of VCs with same resource ratio. Comparison between KRAKEN, TETRIS and AHAB(KRAKEN). Arrival rate is 20.



(a) Num. requests / sequence (b) Num. sequences ($numVCs = 15$) ($numSequences = 20$)

Fig. 6. Fraction of used CUs obtained by AHAB against the two parameters of AHAB. Comparison of results for OKTOPUS, KRAKEN and TETRIS. The sequence length positively affects the metric. 20 sequences containing 15 requests are enough for a high substrate utilizations ($> 90\%$).

Furthermore, we assess if the embedding algorithm affects the performance of AHAB.

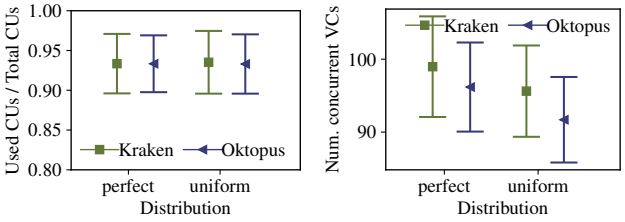
Fig. 6a visualizes how the fraction of used CUs changes with the number of requests per sequence. It shows the performance of AHAB admission control for all three embedding algorithms (OKTOPUS, KRAKEN and TETRIS). The number of sequences is fixed to 20. First, we observe that the utilization is positively affected by AHAB's sequence length. It grows from 0.8 for $numVCs = 1$ to 0.93 for $numVCs = 20$ and KRAKEN. The benefit of adding more requests vanishes as the sequences become longer. Additionally, the differences between the three embedding algorithms do not vary significantly for $numVCs > 5$. However, for small sequence lengths, the inherent performance of the embedding algorithm dominates: TETRIS is better than OKTOPUS and KRAKEN. The difference diminishes with increasing $numVCs$ and the break even is around 5 requests per sequence where AHAB produces the same utilization for all three embedding algorithms. For longer sequences the allocation with OKTOPUS or KRAKEN leads to higher substrate utilization. The implicit selection that TETRIS performs, limits the improvement, but the use of admission control still raises the fraction of used CUs by 0.08.

Fig. 6b shows in a similar way how the number of sequences affects the performance of AHAB. The sequence length is fixed to $numVCs = 15$. Except for the steps from 1 to 5 and from 5 to 10 sequences, we observe only very little change with increasing number of sequences. Thus, $10 - 20$ sequences are sufficient to obtain good results with AHAB. More sequences do not increase the quality of the decisions. This conclusion is not affected by the embedding algorithm.

In summary, looking more steps into the future improves the performance of AHAB at the cost of computation time. However, very long sequences do not further increase the substrate utilization, which allows to find good trade-offs. TETRIS performs an implicit selection of requests and its interference with AHAB leads to worse results compared to KRAKEN and OKTOPUS. The performance is not sensitive to the number of sequences which suffices to be in the range of several 10's.

G. What is the estimation error AHAB can cope with?

Sec. IV we assumed perfect knowledge about the generation process of VCs. This section relaxes the preceding assumption and evaluates how AHAB behaves, when it uses different distributions for generating the requests. The modified distributions have the same support as the original ones, but have a uniform shape.



(a) CU utilization (b) Num. concurrent VCs

Fig. 7. Comparison of AHAB's performance between different distributions for request generation through 95% confidence intervals. In both figures, the left group shows the results for the distributions as described in Sec. V-A. The right group uses uniform distributions with the same boundaries.

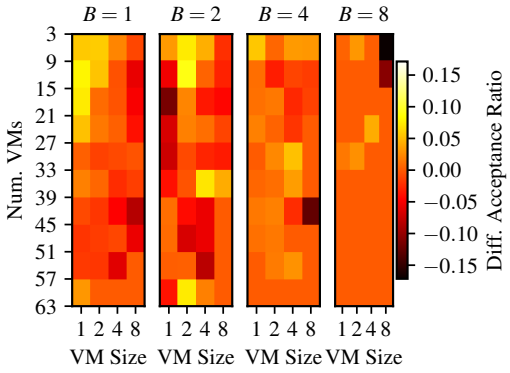


Fig. 8. Difference of acceptance ratio between AHAB(KRAKEN) with perfectly matched distributions and with mismatched/uniform distributions for request generation. Values are grouped by VC specification. Positive values show higher acceptance with matched distributions.

Fig. 7 compares how the substrate utilization and the number of concurrent VCs are affected by this change. The left group shows the results for perfect distribution estimation while the right ones contain the results obtained with the uniform distributions. AHAB runs 20 sequences with 15 requests each. Considering the fraction of used CUs, we observe that using a uniform distribution has no impact on AHAB.

However, Fig. 7b emphasizes that less VCs are embedded concurrently. This implies that larger requests are admitted by AHAB. An explanation for this is that using a uniform distribution instead of a geometric one results in a higher mean values of N, S, B. The mean number of VMs per request increases from 20 to 31.5 and the means for the bandwidth and VM size rise from 2.25 to 3.75. The higher mean value leads to an overestimation of the rewards obtained from future requests, when AHAB calculates the score for a sequence. As a consequence, it is less likely that the mean score of the sequences with accepted request is larger than the mean score of the sequences without the request. This is especially the case, when the arriving request is small and leads to more rejected small requests and a slightly higher number of accepted larger requests.

Fig. 8 underlines this. It shows the difference in acceptance ratio of the runs with perfect distribution estimation by AHAB and the runs with the uniform distribution used for request generation. In particular for $B = 1$, we observe that there are several groups of small requests with higher acceptance ratio when AHAB has access to perfectly fitted distributions (light, positive values). Furthermore, several dark bins (negative values) indicate higher acceptance of larger requests, when uniform distributions are used, e.g., $B = 2$ and $S = 4$.

In conclusion, AHAB's performance seems to be robust against small deviations in the request generation process. But the acceptance pattern changes. Larger deviations are unlikely given today's estimation methodologies but would require a more extensive analysis of AHAB's behavior.

H. How should we design the cluster?

Finally, this section evaluates how the results are affected by the host capacities. Fig. 9 illustrates the CU utilization of the different algorithms for varying computation ($\hat{C}$) and network ($\hat{B}$) capacities of the hosts. The arrival rate is $\lambda = 12$ for all setups with $\hat{C} = 8$ and $\lambda = 20$ for all setups with $\hat{C} = 16$ to keep the offered load similar. In both cases, the system is overloaded and $> 15\%$ of the requests are rejected. The leftmost group of confidence intervals shows the results for $\hat{C} = 8, \hat{B} = 8$ which are already evaluated in Sec. V-B. We recall the significant dominance of AHAB(KRAKEN). When the capacity of the hosts' up-link is doubled ($\hat{C} = 8, \hat{B} = 16$), a first observation is that the utilization increases for all algorithms. However, OKTOPUS, KRAKEN and TETRIS close the gap to AHAB. The performance difference is only ≤ 0.05 compared to ≈ 0.1 in the previous case. Moreover, OKTOPUS, KRAKEN and TETRIS perform now similar as AHAB(KRAKEN) with $\hat{B} = 8$ but at the cost of doubling the physical link capacity. This observation implies that the high utilization of the host link limits the embedding of VCs and leads to fragmented computation resources. With the increased up-link capacity, the resource ratio of a host is now $\frac{\hat{C}}{\hat{B}} = 0.5 < 1$, which is similar to the ratio of the requests that are preferably picked by KRAKEN and TETRIS (see Sec. V-D). Furthermore, a single VM can no longer block an entire host's link. This increases the probability of multiple allocated VMs at one host and reduces the fragmentation of computational resources. A second point is that TETRIS performs worse than OKTOPUS and KRAKEN. Thus, with sufficient network resources available, the benefit of mapping communication intensive with computation intensive requests vanishes.

Doubling $\hat{C}$ while keeping $\hat{B} = 8$, leads in total to lower utilization for all algorithms. In particular, the gap between OKTOPUS and KRAKEN grows as OKTOPUS embeds less efficiently and wastes more resources on the hosts' up-links.

The results for the case $\hat{C} = 16$, $\hat{B} = 16$ show that again the bandwidth is the limiting factor and one main reason why AHAB performs better than the algorithms without admission control. For this case, the utilization obtained with TETRIS and AHAB is almost the same and also OKTOPUS and KRAKEN close the gap to AHAB. Thus, the advantage of AHAB diminishes when the maximum size of the VMs decreases in comparison to the available CUs on a host and the trade-off between performance gain and computational overhead has to be done more carefully. However, further

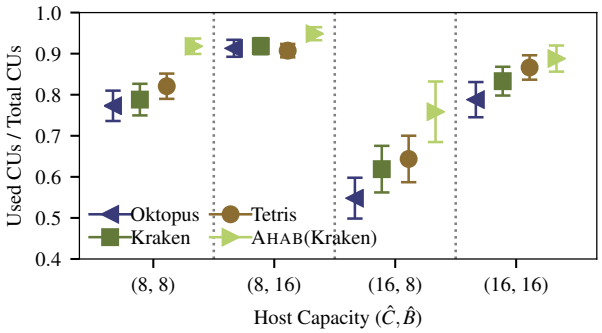


Fig. 9. Performance comparison between embedding algorithms using 95% confidence intervals of CU utilization against host capacities. Note, that the arrival rate for $\hat{C} = 8$ is 12, while it is 20 for $\hat{C} = 16$.

evaluations are necessary to analyze this more in detail.

VI. Conclusions

Virtual clusters are one of the most prominent abstractions that guarantee network performance and isolation in batch-processing and cloud computing. Their efficient embedding on the physical topology is crucial for the economical operation of such systems. This work presented TETRIS, a new VC embedding algorithm that sacrifices the embedding efficiency of a single request in order to maximize the reward in the long run. TETRIS tries to balance the utilization of resources along different dimensions by mapping together computation and communication intensive requests. The evaluations show that this approach beats algorithms such as OKTOPUS or KRAKEN.

As a second step to increase the performance of cluster embedding over time, this work proposed AHAB, a data-driven approach to admission control for VC embedding. AHAB is based on the idea of looking into the future and evaluating the benefit of the current embedding using knowledge about the distributions of the requests' attributes. AHAB shows better performance than algorithms without or with only very simple admission control. This improvement comes at the cost of higher computational complexity which however, can be controlled by AHAB's parametrization. Furthermore, the evaluation shows that the performance difference is impacted by the size of the substrate network.

In future research, it is interesting to look into the possibility provided by machine learning to reduce the online computational effort of AHAB by learning from experience as in [19], [20]. Additionally, within the prediction sequences that AHAB runs, no admission control is applied. The use of more sophisticated policies, as provided by deep learning, can enhance the decision quality of AHAB. Furthermore, we believe that cluster planning that integrates algorithm behaviors, application specifications and demands is another interesting angle for future investigation.

Acknowledgment

This work is part of a project that has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation program (grant agreement No 647158 - FlexNets).

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] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *Proc. ACM SIGCOMM 2011*, Toronto, Ontario, Canada, 2011, pp. 242–253.

[3] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "SecondNet: A Data Center Network Virtualization Architecture with Bandwidth Guarantees," in *Proc. CoNEXT 2010*, Philadelphia, USA, 2010, pp. 15:1–15:12.

[4] K. C. Webb, A. Roy, K. Yocum, and A. C. Snoeren, "Blender: Upgrading tenant-based data center networking," in *2014 ACM/IEEE ANCS*, Los Angeles, CA, USA, 2014, pp. 65–75.

[5] H. Rodrigues, J. R. Santos, Y. Turner, P. Soares, and D. Guedes, "Gatekeeper: Supporting bandwidth guarantees for multi-tenant datacenter networks," in *Proc. 3rd Conference on I/O Virtualization*, Portland, OR, USA, 2011, pp. 1–8.

[6] D. Li, J. Zhu, J. Wu, J. Guan, and Y. Zhang, "Guaranteeing Heterogeneous Bandwidth Demand in Multitenant Data Center Networks," *IEEE/ACM Trans. Netw.*, vol. 23, no. 5, pp. 1648–1660, 2015.

[7] D. Xie, N. Ding, Y. C. Hu, and R. Kompella, "The Only Constant is Change: Incorporating Time-varying Network Reservations in Data Centers," in *Proc. ACM SIGCOMM 2012*, vol. 42, Helsinki, Finland, 2012, pp. 199–210.

[8] M. Rost, C. Fuerst, and S. Schmid, "Beyond the stars: Revisiting virtual cluster embeddings," *ACM SIGCOMM CCR*, vol. 45, no. 3, pp. 12–18, 2015.

[9] C. Fuerst, S. Schmid, L. Suresh, and P. Costa, "Kraken: Online and Elastic Resource Reservations for Cloud Datacenters," *IEEE/ACM Trans. Netw.*, vol. PP, no. 99, pp. 1–14, 2017.

[10] R. Yu, G. Xue, X. Zhang, and D. Li, "Survivable and bandwidth-guaranteed embedding of virtual clusters in cloud data centers," in *Proc. IEEE INFOCOM 2017*, Atlanta, GA, USA, 2017, pp. 1–9.

[11] 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 '17*, Shanghai, China, 2017, pp. 153–167.

[12] J. Jiang, V. Sekar, I. Stoica, and H. Zhang, "Unleashing the potential of data-driven networking," in *Proc. COMSNET 2017*, Bengaluru, India, 2017, pp. 1–8.

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

[14] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center." in *Proc. 8th NSDI*, vol. 11, Boston, MA, USA, 2011, pp. 295–308.

[15] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. ACM SIGCOMM 2008*, vol. 38, Seattle, WA, USA, 2008, pp. 63–74.

[16] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[17] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A Survey of Monte Carlo Tree Search Methods," vol. 4, no. 1, pp. 1–49, 2012.

[18] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Towards understanding heterogeneous clouds at scale: Google trace analysis," *Intel Sci. Technol. Cent. Cloud Comput. Tech Rep*, vol. 84, 2012.

[19] A. Blenk, P. Kalmbach, P. van der Smagt, and W. Kellerer, "Boost online virtual network embedding: Using neural networks for admission control," in *Proc. 12th CNSM*, Montreal, Canada, 2016, pp. 10–18.

[20] A. Blenk, P. Kalmbach, W. Kellerer, and S. Schmid, "O'zapft is: Tap Your Network Algorithm's Big Data!" in *Proc. ACM Big-DAMA*, Los Angeles, CA, USA, 2017, pp. 19–24.