Enabling Efficient Placement of Virtual Infrastructures in the Cloud

Ioana Giurgiu, Claris Castillo, Asser Tantawi, Malgorzata Steinder

Enabling efficient placement of virtual infrastructures in the Cloud

Ioana Giurgiu[1], Claris Castillo[2], Asser Tantawi[2], and Malgorzata Steinder[2]

[1] Systems Group, Dept. of Computer Science, ETH Zurich
{igiurgiu}@inf.ethz.ch
[2] IBM T.J. Watson Research Center
{claris,tantawi,steinder}@us.ibm.com

Abstract. In the IaaS model, users have the opportunity to run their applications by creating virtualized infrastructures, from virtual machines, networks and storage volumes. However, they are still not able to optimize these infrastructures to their workloads, in order to receive guarantees of resource requirements or availability constraints. In this paper we address the problem of efficiently placing such infrastructures in large scale data centers, while considering compute and network demands, as well as availability requirements. Unlike previous techniques that focus on the networking or the compute resources allocation in a piecemeal fashion, we consider all these factors in one single solution. Our approach makes the problem tractable, while enabling the load balancing of resources. We show the effectiveness and efficiency of our approach with a rich set of workloads over extensive simulations.

Keywords: Network-aware virtual machine placement, Cloud, Performance

1 Introduction

In enterprise data centers, infrastructure architects tailor hardware and software configuration to optimize for their workloads. To run a production application, the administrator provisions physical machines, storage, networks, middleware, and application code such that the application is resilient to hardware failures and performance bottlenecks. The Cloud changes the infrastructure provisioning model. A typical IaaS offers virtualized building blocks, such as virtual machines, storage volumes, and networks, which users of the Cloud connect together to create virtualized infrastructures for their workloads. Very little control is given to a user with respect to the layout of these virtualized building blocks on the physical infrastructure. As a result, it is impossible for the user to build a virtualized infrastructure that guarantees, for example, high communication bandwidth between virtual machines, proximity to storage, or spreading of multiple virtual machines across different racks for availability reasons. As a matter of fact, the only support for workload optimization available in today's Cloud is via pre-built virtual infrastructures which are tuned to specific workloads. For instance, Amazon EC2 offers high performance computing (HPC) instances [1].

We believe that this cookie-cutter approach hinders further adoption and development of the technology. Instead, Cloud users should be able to design

and deploy virtual infrastructures that optimize for their workload. We refer to these virtual infrastructures as Virtual Network Infrastructures (*VNI*). More specifically, a VNI is represented as a set of heterogeneous virtual machines with constraints governing the performance of these virtual machines as a whole in order to satisfy application requirements. In this work we address one crucial problem in enabling this vision. We are concerned with developing placement techniques that allow the Cloud to efficiently and effectively allocate resources that satisfy VNI constraints and Cloud level goals. We consider VNIs consisting of virtual machines with compute and network demand, as well as availability requirements.

A VNI can be represented as an attributed graph. As such, the VNI placement problem is equivalent to the problem of graph monomorphism and therefore is NP-hard [2]. The complexity of the problem arises from its combinatorial nature, thus efficiency is a major challenge. Others in the community have tackled less constrained versions of this problem [3–5]. The proposed approaches however, address the placement problem in a piecemeal fashion: they either focus on the aspects pertaining to network performance leaving aspects of the allocation of compute resources as a secondary objective or vice versa, or suffer from high complexity. Our approach is unique in that it tackles the problem in a comprehensive manner by factoring in network, compute and availability performance aspects into one single solution.

We have developed a novel placement framework that makes the problem tractable and is generic enough to support increasing complexity. The core of the framework is the introduction of a novel resource abstraction, called a *cold spot* from here on. A cold spot consists of a collection of compute nodes that exhibit high availability of compute resources and network connectivity. Cold spots identify subsets of resources where VNIs should be best deployed. They help reduce and guide the search space for the optimization problem. Our placement framework consists of four steps: (a) identifying cold spots, (b) clustering virtual machines to reduce overall communication traffic and reduce placement complexity, (c) identifying candidate cold spots whose features are similar to those of the VNI in order to increase the chances of deployment, and (d) performing the actual placement by using efficient graph-based search algorithms that optimize for load balancing.

The main contributions of this work can be summarized as follows:

- We develop a placement framework that breaks down the placement problem into four tractable subproblems.
- We introduce a novel resource abstraction called *cold spot* that effectively reduces the search space and improves performance.
- We present experimental results that show the efficiency and effectiveness of our approach in large data centers.

The remainder of this paper is organized as follows. We formulate the problem in Section 2. Section 3 overviews all four stages of our technique in detail, while Section 4 presents our experimental results. Finally, in Section 5 we discuss open questions and future work. Section 6 provides an overview of the current state of the art and positions our work. And, we conclude in Section 7.

2 Problem formulation

We consider a data center which consists of a collection of physical machines (PM) that are inter-connected by a network consisting of a set of links (LK). Every PM can host one or more virtual machines (VM). A VNI comprises a set of networked and constrained VMs and is the deployable unit within the data center. The placement problem consists of mapping VMs in a given VNI to PMs in the data center. Next, we describe the physical infrastructure and VNI characterization in more detail. Table 1 summarizes the most common terms used throughout the paper.

Let $\mathcal{PM} = \{PM_i | i = 1, 2, ..., n_{PM}\}$ denote the set of physical machines in the data center. Each PM has a set of resources $\mathcal{R} = \{r_m | m = 1, 2, ..., n_R\}$, such as *CPU*, *memory*, and *disk storage*. The total capacity of resource r_m on PM_i is denoted by $rc_{i,m}$, whereas $ru_{i,m}$ represents its usage on the same PM, $ru_{i,m} \leq rc_{i,m}$. We define the amount of resource available for r_m on PM_i as $ra_{i,m} = rc_{i,m} - ru_{i,m}$. We assume that a PM is connected to a switch through an edge link, and that switches are interconnected through core links.

The network is modeled as a graph, where PMs and switches are vertices, while links are edges. We denote the set of links as $\mathcal{LK} = \{LK_k | k = 1, 2, ..., n_{LK}\}$. Each link is characterized by a communication bandwidth. The total bandwidth capacity of LK_k is denoted by bc_k, whereas bu_k represents its usage, $bu_k \leq bc_k$. The amount of available bandwidth is defined as $ba_k = bc_k - bu_k$. We characterize data center $\mathbf{D}$ by the tuple $(\mathcal{PM}, \mathcal{LK})$.

A VNI $\mathbf{P}$ is characterized by the tuple $(\mathcal{VM}, \Lambda, S)$. The set $\mathcal{VM} = \{VM_j | j = 1, 2, ..., n_{VM}\}$ represents the collection of virtual machines which constitute the VNI. VM_j is characterized by a set of resource demands $rd_{j,m}$, one per resource type in $\mathcal{R}$. These resource demands are considered when placing a specific VM onto a PM, in order to make sure that there are enough available resources on the PM to satisfy the VM demands. The communication bandwidth demand between

Term	Description	Term	Description
r_m	Resource (e.g., CPU)	ba_k	Bandwidth available in LK_k, i.e., $b_{ck} - b_{uk}$
$rc_{i,m}$	Total capacity of resource r_m on PM_i	bu_k	Bandwidth usage of LK_k
$ru_{i,m}$	Usage of resource r_m on PM_i	bc_k	Total bandwidth capacity of LK_k
$ra_{i,m}$	Availability of resource r_m on PM_i, i.e., $rc_{i,m} - ru_{i,m}$	$rd_{j,m}$	Resource demand of VM_j on resource r_m
$\lambda_{i,j}$	Network demand between VM_i and VM_j	$n_{PM_{CS}}$	Cardinality of the set of PMs belonging to cold spot CS
$l_{i,j}$	Locality constraint between VM_i and VM_j (∞, $-\infty$)	$path(i, j)$	Set of links on path between PM_i and PM_j
n_{VM}	Cardinality of the set of VMs belonging to a VNI	n_{PM}	Cardinality of the set of PMs in data center

Table 1: Common terminology

VM_i and VM_j is denoted by $\lambda_{i,j} \geq 0$, where $1 \leq i, j \leq n_{VM}$ and $\lambda_{i,i} = 0$. We assume that the matrix $\Lambda = [\lambda_{i,j}]$ is symmetrical with zero diagonal. In other words, bandwidth requirements among VMs in a given VNI may be modeled as an undirected graph, where the vertices are VMs and the edges are pairwise communication demands.

Availability constraints are characterized as follows. We consider a data center which is partitioned into a hierarchy of availability zones, where PMs in the same zone have similar availability characteristics. As an example, a hierarchy of availability zones may be induced by the containment hierarchy of PMs, blade centers, racks, cluster and data centers. In such case, one may model this hierarchy as a tree, where the leaves are the PMs and the intermediate node represents a zone of availability. Thus, we associate an availability level, $V_l, l = 0, \cdots, L$, for a node at level l in the tree, where $l = 0$ represents the leaves, i.e., PMs, and $l = L$ represents the root of the tree with height L, i..e, highest level switch. We assume that $V_0 \leq V_1 \cdots \leq V_L$, since two PMs in distant availability zones have higher chances of having one of them available. Using this tree model, two PMs PM_i and PM_j with the lowest common ancestor at level l have $v_{i,j} = V_l$ (Clearly, $v_{i,i} = V_0$). For convenience we define $g_i(l), i = 1, \cdots, n_{PM}$ and $l = 0, \cdots, L$ as the set of PMs such that for $PM_j \in g_i(l)$ we have $v_{i,j} = V_l$. Following these observations, availability constraints can be directly mapped into locality constraints. More specifically, to represent location constraints between VMs, we define the matrix $S = [s^l_{i,j}]$, where $s^l_{i,j}$ represents the type of location constraint between VM_i and VM_j, where $1 \leq i, j \leq n_{VM}$ and $i \neq j$ and l refers to the availability zone level required by the constraint. In this paper we assume two distinct types, namely $s^l_{i,j} \in \{+\infty, -\infty\}$, corresponding to *colocation* and *anti-colocation*, respectively. To illustrate, an *anti-colocation* constraint at the PM-level ($l = 0$) between VM_i and VM_j indicates that VMs must be placed on different PMs and is associated with an infinitely large communication cost between them. Alternatively, a *colocation* constraint means that the VMs must be placed on the same PMs and is associated with a small communication cost.

We denote by $\pi(\mathbf{P}, \mathbf{D})$ a particular placement of VNI $\mathbf{P}$ in data center $\mathbf{D}$. For brevity we will write it as $\pi(\mathbf{P})$. $\pi(\mathbf{P})$ is a vector of length n_{VM}, where $\pi_j(\mathbf{P})$ is the PM onto which VM_j is placed. The placement process maps every VM in $\mathcal{VM}(\mathbf{P})$ to a particular PM in $\mathcal{PM}$, such that (a) the VM's resource demands are satisfied by the PM, (b) the bandwidth constraints between any two communicating VMs are met by the links of the data center connectivity network, (c) the pairwise location constraints are satisfied.

Placement goals We consider two classes of objectives in our placement technique: *system behavior objectives* (1-2) and *performance objectives* (3-5):

1. *Efficient and scalable placement* – We need to place VNIs in such a way that (a) the performance of the application is maximized (e.g., fewer hops between communicating VMs reduces network delay), and (b) the placement time scales with the increasing size of the data center.
2. *High VNI acceptance rate* – The placement algorithm must maximize the number of VNIs for which constraints are satisfied.

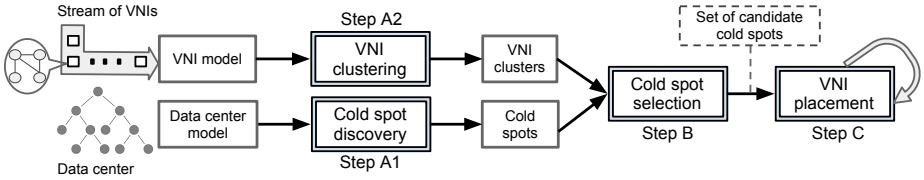


Fig. 1: VNI placement process.

3. *Load balancing* – For all placed VNIs, we seek to balance allocation of all resources across the data center.
4. *Resource constraints* – We assume that resources are not over-committed. Hence, VNI resource demands must be met by the corresponding resources available on the data center. Formally, $\forall PM_i \in \mathcal{PM},\ \ \forall r_m \in \mathcal{R}$,

$$ru_{i,m} \equiv \left[\sum_{\mathbf{p}} \sum_{VM_j \in \mathcal{VM}(\mathbf{p})} rd_{j,m}\ \ \mathbf{I}_{\pi_j(\mathbf{p}),PM_i} \right] \ \leq\ rc_{i,m},$$

where $\mathbf{p}$ runs over all placed VNI and $\mathbf{I}$ is the indicator function. Furthermore, $\forall LK_k \in \mathcal{LK}$,

$$bu_k \equiv \left[\sum_{\mathbf{p}} \sum_{VM_i, VM_j \in \mathcal{VM}(\mathbf{p})} \lambda_{i,j}\ \ \mathbf{I}_{LK_k \in path(\pi_i(\mathbf{p}),\pi_\mathbf{j}(\mathbf{p}))} \right] \ \leq\ bc_k,$$

where $path(PM_i, PM_j)$ represents the set of links along the path between PM_i and PM_j. For simplicity, we assume that the traffic demand between two PMs is routed through a single path in the network.
5. *Hard location constraints* – The colocation and anti-colocation constraints must be satisfied for all placed VNIs. As we explain later, softening constraints can be easily achieved. That is, $\forall \mathbf{p},\ \ \forall VM_i, VM_j \in \mathcal{VM}(\mathbf{p})$,

$$s^l_{i,j} = +\infty \ \ \Rightarrow\ \ VMj \in g_i(l), s^l_{i,j} = -\infty \ \ \Rightarrow\ \ VMj \notin g_i(l).$$

3 Placement Algorithms

Our approach to meet the constraints and performance objectives presented earlier is to divide the placement problem into four steps as shown in Figure 1. In this section, we discuss these steps in detail.

3.1 Cold spot discovery

One key component of our placement technique is the concept of *cold spot*. A cold spot is a resource construct consisting of a collection of physical computing nodes that exhibit high availability of compute resources and ample network connectivity to other PMs. In principle, any property of interest can be considered when constructing cold spots. This step is concerned with discovering such cold spots in the system and is invoked periodically and asynchronously relative to the placement request. This observation is important since performing any analysis on the data center graph is expected to be computationally intensive.

The intuition behind this stage is twofold. First, it reduces the search space when placing a VNI within the data center, which results in an overall lower placement time and hence better scalability. Second, cold spots improve resource

utilization by reducing resource fragmentation. We demonstrate these benefits in Section 4. This step takes as input the data center model, its state (which includes resource allocation), as well as a dynamic parameter, called *threshold* and produces a set of cold spots. This stage is further divided into two main steps: ranking the PMs and generating a set of cold spots.

Ranking of physical compute nodes The first step ranks all the PMs in the data center based on their resources availability. We define the availability of a particular PM_i by a measure, RA_{PM_i}, which is based on the compute resources and the network bandwidth of all outgoing links, namely,

$$RA_{PM_i} = \sum_{r_m \in \mathcal{R}} w_m \, ra_{i,m} \; \cdot \sum_{LK_k \in links(PM_i)} ba_k,$$

where w_m is a weight which can be adjusted in order to tailor the PMs ranking relative to a specific type of resource and $links(PM_i)$ represents the set of links connected to PM_i (via NICs).

We want PMs with ample network connectivity to other PMs in the system to be ranked higher, as they have a higher potential to satisfy the needs of VNIs. To this end, we propose a heuristic that has a long-sighted view of the PMs network connectivity. That is, to compute the rank of a given PM the heuristic first identifies the PM's neighborhood as the set of PMs that are K hops away, and then computes its network connectivity to these PMs as a function of network bandwidth. K is a parameter which could be set in relation to the data center diameter or VNI size. This step generates a list of all PMs in decreasing order of their availability, computed with the heuristic NRA_{PM_i} defined as,

$$NRA_{PM_i} = \frac{\sum_{PM_j \in neighbors(PM_i,K)} \frac{RA_{PM_i}+RA_{PM_j}}{2} \, min_{LK_k \in path(PM_i,PM_j)}(ba_k)}{\mid neighbors(PM_i, K) \mid}$$

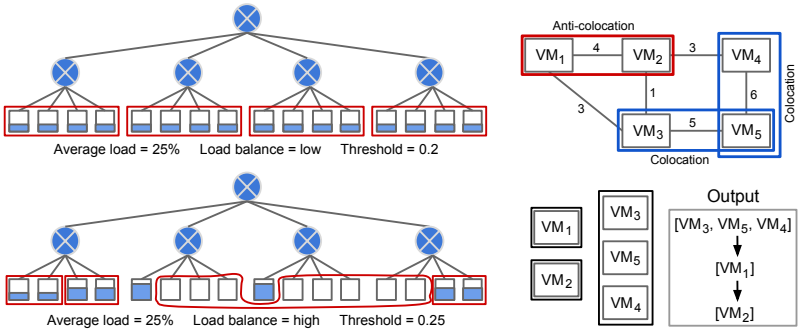
where $neighbors(PM_i, K)$ is the set of PMs that are at most K hops away from PM_i. Notice that the heuristic accounts for the minimum available bandwidth on the path between PM_i and a neighbor PM_j to characterize the network component, as well as for both PM_i's and PM_j's compute resources availability.

Cold spots generation Cold spots are constructed by continuously adding PMs to already existing collections based on several heuristics. To keep track of the non-added PMs we maintain an updated list. The first entry in the list, which corresponds to the PM with the highest rank, becomes the root of the new cold spot. In order to decide whether a particular not-yet-added PM_i should be added to the new cold spot, we define the measure $Potential_{PM_i}$ as follows. Let CS denote the currently identified cold spot. We define the potential of PM_i as the weighted sum,

$$Potential_{PM_i} = (1 - w)R_i + w\sqrt{\frac{H_i^2 + B_i^2}{2}}$$

which includes three terms: R_i, H_i, and B_i. They are defined as,

$$R_i = \sum_{r_m \in \mathcal{R}} w_m \, ru_{i,m},$$



(a) Cold spot discovery example (b) VNI clustering example

Fig. 2: Cold spot discovery and VNI clustering examples.

$$H_i = \frac{\sum_{PM_j \in CS} \frac{hops(PM_i,PM_j)-1}{hops(PM_i,PM_j)+1}}{n_{PM_{CS}}}, B_i = \frac{\sum_{PM_j \in CS} 1 - \frac{hops(PM_i,PM_j)}{\sum_{LK_k \in path(PM_i,PM_j)} \frac{1}{1-bu_k}}}{n_{PM_{CS}}}$$

where R_i is a measure of resource utilization of PM_i, H_i captures the distance, expressed in number of hops ($hops(PM_i, PM_j)$), between PM_i and all PMs that have been already added to the cold spot CS, and B_i the bandwidth utilization between all the links connecting PM_i to all PMs in the cold spot. Both H_i, and B_i terms are expressions of the network connectivity aspect. As such, the weight w provides better controllability of the algorithm over the characteristics of the cold spot. The complexity of the algorithm is $O((n_{PM} + n_{PM_{CS}} n_{VM}) log_{n_{VM}})$. It is important to note that our network metrics can be easily modified to accommodate for network technologies wherein the number of hops is not a metric of relevance, e.g., flat networks, by only incorporating link utilization.

PMs that have lower potential values are more desirable, thus a PM_i is included as part of CS if $Potential_{PM_i} \leq threshold$. The *threshold* is a parameter that greatly influences the features of the resulting cold spots. Fig. 2(a) provides an example of how cold spots are discovered for a data center consisting of 16 PMs, depending on the variance of the resources load on PMs and the threshold value. If the load variance on the PMs is low, then the prevailing factor for adding new PMs to a cold spot is their distance to PMs already found in the cold spot. In the opposite scenario, the cold spot discovery step groups PMs with similar characteristics of their compute loads and neighborhood qualities, even beyond the first-level switch. The algorithm is driven by the threshold value. The lower it is, the more selective the filtering (i.e., adding PMs to a cold spot) is, and vice-versa. Thus, with a threshold of 0.2 and a low load variance, the algorithm groups together PMs under the same first-level switch.

3.2 VNI clustering

The clustering step groups the highly communicating VMs of a VNI in order to reduce traffic, while at the same time satisfying the location constraints. The purpose of performing the clustering is to guide and simplify the placement, by establishing the order in which the VMs should be considered by the placement algorithm to improve network utilization–while respecting locality constraints.

Our proposed algorithm is based on stochastic flow injection [6]. Due to space constraints we omit the description of the algorithm and refer the reader

to [6]. We extended this technique to also consider locality constraints between pairs of VMs as follows. Since anti-colocation and colocation constraints translate to either placing VMs separately on different PMs or placing them together, we add logical links between location-constrained VMs with ∞ or $-\infty$ weights, as expressions of communication demands. The algorithm complexity is $O(n_{LK_{CS}} {n_{VM}}^2)$ where LK_{CS} corresponds to the number of links in the cold spot. Notice that other clustering techniques such as K-mean clustering could have been used and extended to incorporate locality constraints.

To illustrate consider a VNI composed of 5 VMs as shown in Fig. 2(b). Assume that VM_3 has the highest average compute demand followed by VM_5, VM_1, VM_4 and VM_2. The communication links between VMs have demands expressed in Mbps (e.g., 3Mbps between VM_1 and VM_3). Additionally, we include two colocation and one anti-colocation constraints. By applying the algorithm, we generate 3 clusters of VMs, satisfying all location constraints. As expected, VM_1 and VM_2 need to be placed on different PMs, while all remaining VMs must be colocated – with these being hard requirements. The last step orders the VMs within each cluster based on their average compute demands, followed by a sorting between clusters. We can easily see that the first cluster considered in the placement step contains VM_3, VM_4 and VM_5, since VM_3 is the most demanding VM. Similarly, VM_1 precedes VM_2 at placement.

3.3 Cold spot selection

This step compares specific VNI features against the properties of the available cold spots and selects those cold spots that have an increased chance of allocating resources to match the VNI demands. To do this we introduce a metric $Score_{CS}$ which is used to rank all cold spots. Let us consider a VNI $\mathbf{P}$, then $Score_{CS}$ is defined as,

$$Score_{CS} = (n_{PM_{CS}} - sparsity_{\mathbf{P}}) * Avg_{CS,P} * Dev_{CS,P}.$$

We describe in detail each of the three components that make up $Score_{CS}$. $Sparsity_{\boldsymbol{P}}$ provides a lower-bound in the number of PMs needed for placing $\mathbf{P}$ if all the resources in the cold spot were fully available. We omit the algorithm to compute this metric due to space constraints. Instead, we explain by example. Suppose we have a VNI $\mathbf{P}$ is composed of VM_1, VM_2, and VM_3. Also, suppose that there is an anti-colocation constraint between VM_2 and VM_3. This means that we need at least 2 PMs to place the VNI, since VM_2 and VM_3 need to be placed on different machines. A similar approach is followed for colocation constraints.

$Avg_{CS,P}$ is the average remaining availability over all n_R resources and the links. To define it, consider cold spot CS and the set of physical machines in CS as $\mathcal{PM}_{CS} = \{PM_i | i = 1, 2, ..., n_{PM_{CS}}\}$. Further, let $\mathcal{LK}_{CS} = \{LK_k | k = 1, 2, ..., n_{LK_{CS}}\}$ be the set of links in CS. Let the VNI under consideration include the set of VMs, $\mathcal{VM}_P = \{VM_j | j = 1, 2, ..., n_{VM}\}$. We define

$$rr_m = \frac{1}{n_{PM_{CS}}} \left[\sum_{i=1,2,...,n_{PM_{CS}}} ra_{i,m} \; - \sum_{j=1,2,...,n_{VM}} rd_{j,m} \right]$$

as the average remaining availability of resource $r_m, m = 1, 2, ..., n_R$, in CS after satisfying the resource demand of VNI, $\mathbf{P}$.

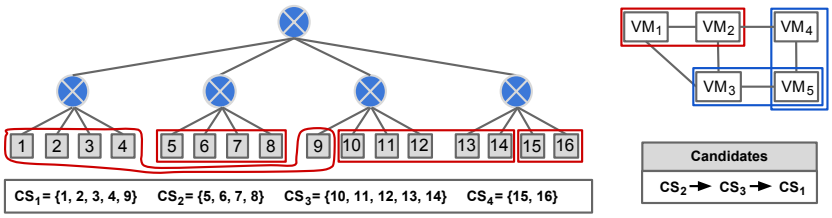


Fig. 3: Cold spot selection example for clustered VNI and a data center with 16 PMs.

Further, we define

$$br = \frac{1}{n_{LK_{CS}}} \left[\sum_{k=1,2,\ldots,n_{LK_{CS}}} ba_k \; - \sum_{i,j=1,2,\ldots,n_{VM};\, i>j} \lambda_{i,j} \right]$$

as the average remaining bandwidth over all links in the CS after satisfying the bandwidth demand of VNI, $\mathbf{P}$.

Thus, we can now define

$$Avg_{CS,P} = \frac{\sum_{m=1,2,\ldots,n_R} rr_m \; + \; br}{n_R + 1}.$$

$Dev_{CS,P}$ is the absolute deviation in remaining resource availability, given by

$$Dev_{CS,P} = \sum_{m=1,2,\ldots,n_R} | \, rr_m - Avg_{CS,P} \, | \; + \; | \, br - Avg_{CS,P} \, | \, .$$

In a nutshell, the first term of the equation evaluates whether the size of the cold spot is bigger than the sparsity of the VNI. The second term computes for each resource the difference between the average aggregate cold spot availability and the average aggregate VNI demand. Finally, the last term computes the overall variance we would obtain if the VNI was placed in the given cold spot – the smaller the variance, the better. The score needs to be positive for the cold spot to be considered a candidate and we always choose the cold spot with the minimum score value. The intuition behind the $Score_{CS}$ metric is that the cold spots whose features are most similar to those of the VNI should be ranked higher, and therefore tried first for placement. The algorithm complexity is $O(n_{PM_{CS}} \log n_{PM_{CS}})$ for each cold spot found.

Consider the scenario from Fig. 3, with the same VNI as in Fig. 2(b) and four cold spots, each having 2, 4, 5 and 5 PMs, respectively. Assume that the VNI to be placed has the sparsity value 3, given by the location constraints and the fact that the PMs capacity allows neither VM_1 or VM_2 to be placed together with the cluster VM_3, VM_4, and VM_5. Thus, the first step of the algorithm already eliminates CS_4, by comparing its size with the sparsity metric, and builds the candidates set with the remaining cold spots. Consider, in the second step, that by computing $Avg_{CS,P}$ and $Dev_{CS,P}$, all candidates obtain similar values and the metric differentiating them is the size against sparsity. Given the way we score the candidates, the cold spot with the smallest size (CS_2) is ranked first in the placement step, while the largest ones are last.

3.4 VNI placement

The final step performs the actual VNI placement within the current cold spot, selected from the candidate list. We employ a breadth-first search algorithm, which attempts to retrieve the optimal path from the search tree based on

Algorithm 1 VNI placement algorithm

Params: VNI $\mathbf{P} = (\mathcal{VM}, \lambda, S)$, $\mathcal{VM} = \{\texttt{VM}_1, ..., \texttt{VM}_{n_{VM}}\}$, $\texttt{CS} = \{\mathcal{PM}_{CS}, \mathcal{LK}_{CS}\}$, $\{\mathcal{PM}_{CS} = \{\texttt{PM}_1, ..., \texttt{PM}_{n_{PM_{CS}}}\}$, `path`

1: *Initialize* `pending`, `placed`, `path`, `V`, *and* `S` *to* $\emptyset$
2: *For each* `VM` $\in$ `P`, *add* `VM` *to* `pending`
3: **while** `pending` <> $\emptyset$ **do**
4: $\quad$ $\texttt{VM}_{current} \leftarrow$ `pending[0]`
5: $\quad$ **if** `placed` == $\emptyset$ **then**
6: $\quad\quad$ *For each* `PM` $\in$ `CS`, *add* $(\texttt{VM}_{current}$`, PM, h)` *to* `S`
7: $\quad\quad$ *Add* $(\texttt{VM}_{current}$`, PM)` $= min_{k \in S}\{\texttt{h}\}$ *to* `path`
8: $\quad$ **else**
9: $\quad\quad$ `V` $\leftarrow$ `getPMsForConstraints(`$\texttt{VM}_{current}$`, path)`
10: $\quad\quad$ *For each* `PM` $\in$ `V`, *add* $(\texttt{VM}_{current}$`, PM, h)` *to* `S`
11: $\quad\quad$ *Add* $(\texttt{VM}_{current}$`, PM)` $= min_{k \in S}\{\texttt{h}\}$ *to* `path`
12: $\quad$ **end if**
13: $\quad$ *Remove* $\texttt{VM}_{current}$ *from* `pending`
14: $\quad$ *Add* $\texttt{VM}_{current}$ *to* `placed`
15: **end while**

heuristics. We describe the optimization goal later in this section. The search tree is an expression of the optimization problem of finding the optimal placement for a VNI. Its root is the starting search point (i.e., no VM placed yet), the inner nodes correspond to partial placements and leaf nodes to complete placements. The search tree is dynamically constructed at runtime by iteratively creating successor nodes linked to the current node. This is achieved by considering the possible placements for VMs sequentially, depending on how VMs are ordered as a result of the clustering step. A heuristic function, estimating the cost of the paths from the root to the current node, is used. At each step during traversal, the node with the lowest heuristic value is chosen. In what follows, we discuss our heuristic used in the search algorithm given in Algorithm 1.

Our cost heuristic is an expression of the *resource fragmentation* in the cold spot caused by the partial placement decisions, from the root to the current node in the search tree. Since the algorithm always advances on the path with minimum cost, i.e., minimizing resource fragmentation, our heuristic is effectively seeking at balancing the cold spot resources utilization. Thus, for cold spot CS, we introduce the cold spot fragmentation measure denoted by h_{CS}, which includes contributions due to (1) *network fragmentation*, expressed as the number of *isolated regions*, and (2) *resource imbalance*, expressed as the deviation of utilized CPU, disk storage, and memory resources within the cold spot. We define an *isolated region* as a set of PMs that share a link whose utilization is higher than 90% when communicating to the rest of the network starting from the first level switch. To illustrate, all the PMs contained in a bladecenter whose link connecting to the rack-level switch is 92% utilized would comprise an isolated region. Let $N_{isolatedregions}$ be the number of isolated regions in CS. In order to compute its value, we implemented a recursive algorithm that we

omit due to space limitation. As described earlier, the cold spot CS consists of the set of physical machines $\mathcal{PM}_{CS} = \{PM_i | i = 1, 2, ..., n_{PM_{CS}}\}$. We define, $ra_m = \frac{1}{n_{PM_{CS}}} \sum_{i=1,2,...,n_{PM_{CS}}} ra_{i,m}$, as the average availability of resource $r_m, m = 1, 2, ..., n_R$, in CS. Further, we define

$$Avg_{CS} = \frac{1}{n_R} \sum_{m=1,2,...,n_R} ra_m \text{ and } Dev_{CS} = \sum_{m=1,2,...,n_R} \mid ra_m - Avg_{CS} \mid,$$

as the average availability over all n_R resources in CS and the absolute deviation in resource availability, respectively.

We denote the cold spot fragmentation measure as

$$h_{CS} = \frac{N_{isolatedregions} * Dev_{CS}/n_R}{Avail_{CS}},$$

where $Avail_{CS}$ denotes the overall availability of CS and is given by

$$Avail_{CS} = \frac{\sum_{i,j=1,2,...,n_{PM_{CS}};\, i>j} \frac{(ra_{PM_i} + ra_{PM_j})}{2} * min_{k \in path(PM_i, PMj)}\, ba_k}{n_{PM_{CS}}\,(n_{PM_{CS}} - 1)},$$

where

$$ra_{PM_i} = \frac{1}{n_R} \sum_{m=1,2,...,n_R} ra_{i,m}.$$

The algorithm complexity is $O(n_{PM_{CS}} + {n_{VM}}^2)$. In order to speedup the VNI placement, we consider a simple, but effective variant based on beam search. Instead of accounting for all valid PMs for the current VM (i.e., by satisfying the location constraints relative to VMs already placed), only a reduced number of PMs are processed. Given the previously placed VMs, for the current VM we consider those PMs that are closest, in number of hops, to all the PMs already allocated. Only if, by computing the heuristic, none of the closest PMs have the necessary resources for the current VM, we expand the search by including the PMs that have not been considered in the first step. In most cases, the solution found by applying beam search will be suboptimal, but significantly faster.

4 Evaluation

In this section we present simulation results to demonstrate the performance of our VNI placement technique. We use the method of batch means to estimate the performance parameters we consider (and which we discuss shortly), with each batch consisting of 15 runs. For every run, the following methodology is used. We start with an empty data center and sequentially place VNIs until its average compute load – as mean over CPU, memory, and disk storage utilization– reaches 25%, 50% and 75%. Next, we remove random placed VNIs and add new ones with an exponential distribution, such that the average compute load remains stable around the respective targeted values. Each experiment is run such that 50% of the initially placed VNIs are replaced by new VNIs and we collect the performance metrics periodically. Our simulator is written in Java and the experiments were performed on a ThinkPad T520, with 4GB RAM and Intel Core i3-2350m processor, running Ubuntu 11.04.

We consider three types of performance metrics which capture the perspective of the *VNI*, *user* and the *system*. The average path length per placed VNI represents the VNI metric and captures the distance in number of hops between

VMs belonging to the same VNI instance. We consider three user metrics: (1) placement time, which represents the time it takes to solve the placement problem, (2) number of attempts, which captures the average number of attempts or cold spots considered until successfully placing a VNI, and (3) drop rate, which is a ratio of the number of rejected VNIs over the total number of offered VNIs. Finally, the system metrics include the (1) average network utilization, (2) average network congestion, defined as the mean over most congested links per placed VNI, (3) network variance, and (4) compute resources variance.

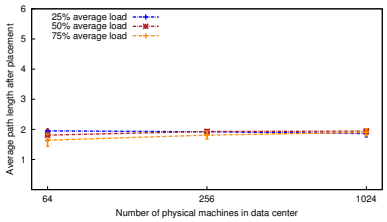
Tree networks are widely adopted in data centers due to their cost-effectiveness and simplicity. Therefore, we consider a data center consisting of a three level tree structure. Following a bottom-top order it can be described as follows: the first level consists of PMs, the second level consists of bladecenters – with each bladecenter containing 16 PMs, the third level consists of racks – with each rack containing 4 bladecenters. We vary the number of racks to produce data centers of different capacities, where by capacity we refer to the size of the data center in number of PMs. We consider three data center sizes: 64, 256, and 1024 PMs. Each PM has 32 cores, 64GB RAM, and 4TB storage capacities, while each network link has 1Gbps capacity. Additionally, between any two PMs there exists a unique path in the data center. Following, the data center diameter (i.e., maximum number of hops between any two PMs) is 4 for 64 PMs, 6 for 256 PMs, and 8 for 1024 PMs. Note that our technique is applicable to any network topology.

We consider a rich set of workloads. We first evaluate our technique against a generic VNI mix in Section 4.1 and show the impact that each placement stage has on the performance of our approach. Second, we consider a more realistic workload mix consisting of cache, hadoop, and three-tiered like VNIs as described in Section 4.3. Finally, we report on the impact that the threshold value has on the cold spot discovery step (Section 4.4) and compare our approach to a technique proposed for virtual network embedding [7] in Section 4.5.

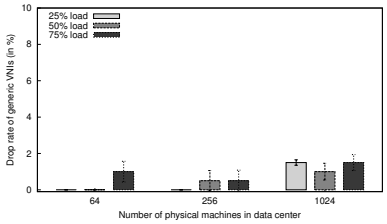
4.1 Generic VNI mix

We consider three types of VNIs: small, large, and extra-large consisting of small, large, and extra-large VM instances, respectively. The resource demands of the VMs follow the specifications of Amazon EC2 instances [8]. That is, their respective resource demands are: (1 core, 1.7GB memory, 160GB storage), (4 cores, 7.5GB memory, 850 GB storage), and (8 cores, 15GB memory, 1690 GB storage). A generic mix is composed of 60% small VNIs, 25% large VNIs, and 15% extra-large VNIs. The number of VMs per VNI is between 2 and 10 following a uniform distribution. For every pair of VMs, we create network demand and locality constraints with probability 0.5 and 0.1, respectively, with the ratio of colocation to anti-colocation constraints being 0.5. The network demands between small, large, and extra-large pairs of VMs are 5, 20, and 50Mbps, respectively.

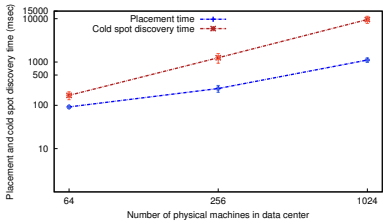
Results are shown in Fig. 4. As it can be observed, the average path length for placed VNIs remains stable at a value of 2 hops and is independent of the data center diameter, thus demonstrating the scalable nature of our approach. This is due to the fact that cold spots enable keeping network traffic under the first-level switch, hence reducing network traffic across higher-level switches. We



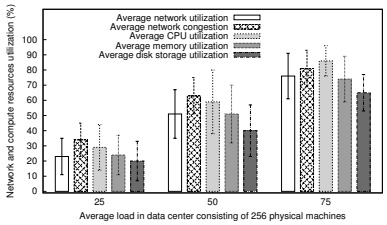
(a) Average path length per VNI



(b) Average drop rate



(c) Placement and discovery times



(d) Network and compute utilizations

Fig. 4: Results for the generic VNI workload with various data center sizes.

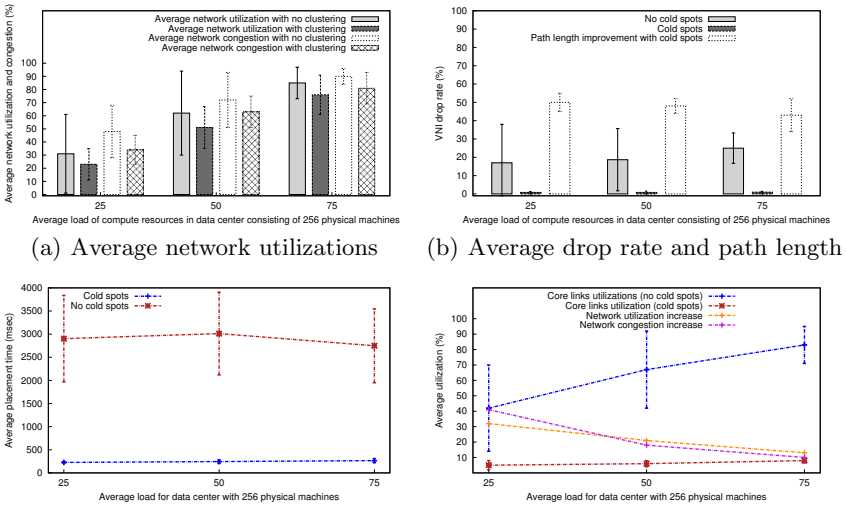
also note that the average number of attempts to place a VNI varies between 1 and 2, which demonstrates that our selection techniques is effective at ranking cold spots as a function of how their features compare to the offered VNI.

Fig. 4(b) depicts a low drop rate of less than 2%. More specifically, for the 64-PM, 256-PM, and 1025-PM data center, (62, 118, and 184), (239, 468, and 723), and (977, 1858, and 2820) VNIs are offered in total, respectively for the 25%, 50%, and 75% loads. As expected, we observe in Fig. 4(c) that the placement time increases linearly with the size of the data center, going from 90 ms for 64 PMs to 1100 ms for 1024 PMs. Similarly, the cold spot discovery time increases as the data center becomes larger, from ≈170 ms for 64 PMs, to ≈1255 ms for 256 PMs, and to ≈9590 ms for 1024 PMs. However, recall that this time is amortized since the cold spot discovery step is executed asynchronously to VNI placement calls.

Fig. 4(d) considers the 256-PM data center and shows the average compute and network utilizations, as well as their corresponding variances, as measures of resource load balancing. Note that Amazon EC2 instances are CPU intensive, therefore the CPU load for all three loads (i.e., 25%, 50%, 75%) is slightly higher than memory and disk storage. To characterize the data center network, we measure the average utilization and the average congestion. The network utilization has similar values as the compute one and its deviation is less than 16%. As expected, the network congestion is higher than the utilization, since for every placed VNI it accounts only for the most congested link on the path between the corresponding PMs. We observe that the maximum network congestion VNIs experience is 81% corresponding to a load of 75%.

4.2 Breaking down the placement technique

In this section we investigate the impact that each individual placement step has on the overall performance. To do this we repeat the experiments with each individual step disabled or modified as described below.



(a) Average network utilizations (b) Average drop rate and path length

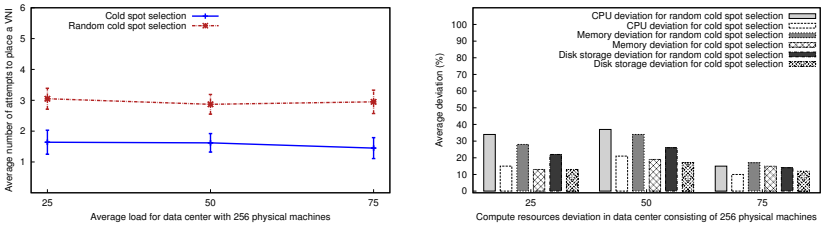
(c) Average placement time (d) Average network performance

Fig. 5: Network performance with vs. without VNI clustering (a) and cold spot vs. data center level (b-d) in a 256-PM data center.

VNI clustering. First, we repeat the experiment for the 256 PMs data center with the VNI clustering step disabled. As it can be observed in Fig. 5(a), without clustering the network utilization and congestion increases by 10–25% and 10–30%, respectively. Furthermore, the variance in the links utilization is higher by up to 60% for lower data center loads, while the congestion variance is higher by up to 45%. We conclude that the clustering step is effective at ensuring that highly-communicating VMs are placed in close proximity, which, as a result, improves network utilization and load balancing.

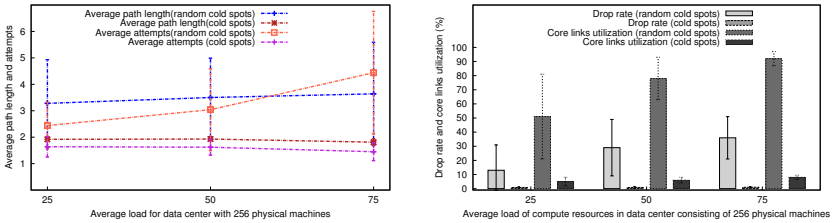
Cold spot discovery. Next, we compare our previous results for the generic VNI workload mix with those obtained when disabling the cold spot discovery step. That is, for every incoming VNI, the placement considers all the PMs in the data center for placement. We plot the results in Fig. 5(b)– 5(d). We observe that without cold spots the average path length for placed VNIs increases by a factor of 2. That is, VMs belonging to the same VNI are placed further apart from each other, thus impacting the traffic in the core links. In fact, we observe the core links utilization increases up to 10x factor. As a consequence, the drop rate increases from less than 2% to up to 25%. This is due to the fact that as core links become congested, the network becomes fragmented and it is more difficult to find a feasible placement for incoming VNIs. A secondary effect is observed in the increased average network utilization and congestion by up to 40% for lower loads of the data center. Finally, given that to place a VNI the algorithm considers all the PMs in the data center, the placement time increases by 10–12x factor as shown in Fig. 5(c).

Random cold spot selection. Further, we are interested in assessing our cold spot selection technique. To do this we consider a selection algorithm wherein cold spots are selected in a random fashion. Fig. 6 shows the average placement attempts and variance of compute resources. As observed, randomly selecting



(a) Average attempts per VNI (b) Average compute variance

Fig. 6: Random cold spot selection vs. our default algorithm in a 256-PM data center.



(a) Path length and attempts per VNI (b) Drop rate and core links utilization

Fig. 7: Comparison between random cold spots and our default algorithm on 256 PMs.

cold spots increases the number of attempts required for successful placements by a factor of 1.5–2x. In addition, the compute resources variance is higher by 15% to 55%, with the more significant impact for lower data center loads. Similarly, the network utilization and congestion are also increased by up to 40% and 15%, respectively. We conclude that the cold spot selection step intelligently chooses the best candidate cold spots for each VNI, to achieve better load balancing and VNI performance in the data center.

Random cold spot discovery. Finally, we evaluate how our cold spot discovery technique influences the performance of our placement technique (Fig. 7). We consider a cold spot discovery algorithm wherein PMs are randomly added to cold spots, as opposed to being added based on their rankings. In this algorithm, the size of the randomly generated cold spots corresponds to the average observed in our previous experiments which is 16. This step is invoked every 20 new incoming VNIs. As expected, the average path length of the placed VNIs increases to 3–4 hops and in some cases even reaches the data center diameter. We also observe an increase in the average number of attempts to place VNIs. Given the random locality in the data center of the VMs within one cold spot, many VNI placements impose demands on the core links. As such, we notice an increase to up to 90% utilization, as opposed to utilizations under 10% with our technique. An important effect of the core links congestion is the increased drop rate, to up to 36% of the total number of offered VNIs.

4.3 Placing cache, hadoop, and three-tiered VNIs

The second part of the evaluation considers realistic workloads, composed of cache, hadoop, and three-tiered VNIs, and measures the effectiveness of our placement technique for the performance metrics considered in the previous section. Fig. 8 shows the topologies corresponding to these specific VNIs.

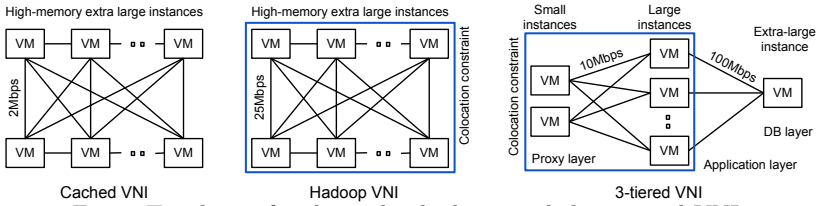
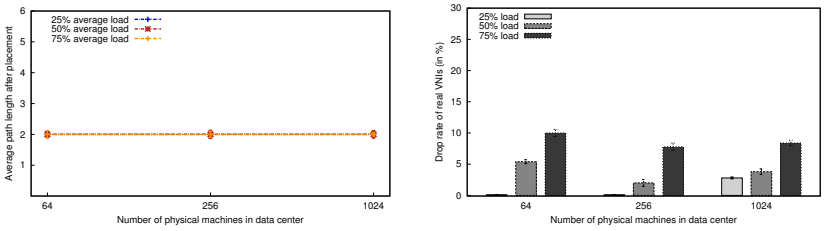
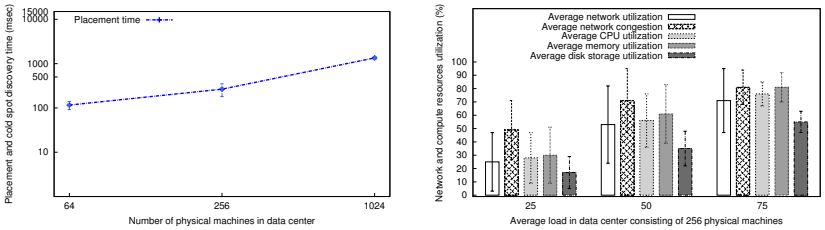


Fig. 8: Topologies for the cache, hadoop, and three-tiered VNIs.



(a) Average path length per VNI (b) Average drop rate

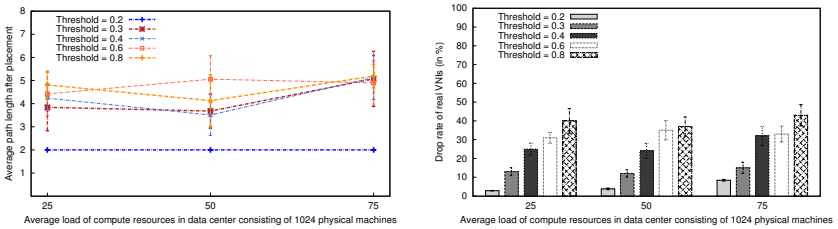


(c) Average placement time (d) Network and compute utilizations

Fig. 9: Results for mix VNI workload with various data center sizes.

Note that cache and hadoop VNIs have identical topologies, where all VMs communicate in a full mesh model and their number varies between 6 and 12. The compute demands match the specifications of Amazon EC2's high-memory extra large instances (6.5 cores, 17.1GB memory, 420GB storage). The network demands are 2Mbps and 25Mbps for the cache and hadoop VNIs, respectively. Whereas, the cache VNI has no location constraints, the VMs of hadoop VNIs need to be placed on PMs located under the same first-level switch (bladecenter). The three-tiered VNIs contain a proxy layer with 2 small EC2-like instances, an application layer, consisting of 5 to 10 large EC2-like instances, and the database layer with 1 extra-large EC2-like instance. The network connectivity between layers is full mesh, with 10Mbps demand for proxies and 100 Mbps for the database instance. The location constraints apply to the VMs belonging to the application and proxy layers, such that they need to be placed on PMs located under the same first-level switch.

Fig. 9 reports the results obtained when placing a mix of realistic VNIs, where 50% are three-tiered, 25% are cache and the remaining 25% are hadoop. We observe that the average path length per placed VNI is 2 and remains independent of the data center diameter. It is also noticeable that the network constrained nature of the cache and hadoop VNIs results in higher drop rate of up to 8% when the average load in the data center is 75% and longer placement time by



(a) Average path length per VNI (b) Average drop rate

Fig. 10: Results with different threshold values in a 1024-PM data center.

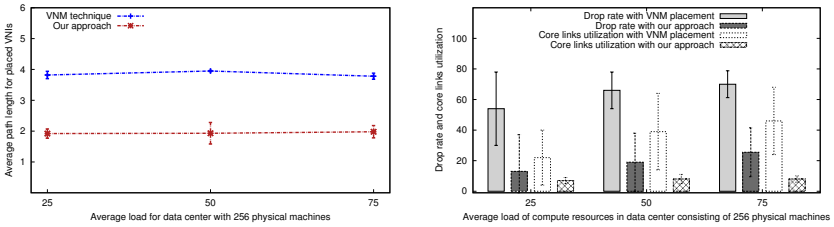
at most 20% as compared to placing generic VNIs. This results in an increase in the number of attempts to place a VNI to an average between 2 and 3.2. In Fig. 9(d) we note that the average memory load is higher than both CPU and disk storage. This effect is due to the higher memory footprint of both cache and hadoop VNIs. We also observe that network congestion increases by up to 30% for lower loads as compared to when generic VNIs are used due to the higher network demand of the VNIs.

4.4 Cold spot discovery threshold

Next, we investigate how the threshold used in the cold spot discovery step influences the properties of the cold spots and the effectiveness of our technique. To do this we run our simulation with the mix of realistic workloads on a 1024 PMs size data center for various threshold values. Selective results are shown in Fig. 10. We notice an increase in average path length per VNI placed as the threshold increases. Over extensive experiments, we found that a value of 0.2 results in cold spots that are balanced and PMs are closely located. With other values, one can easily generate cold spots that are either too small in size, and thus not suitable for the offered VNIs, or too large, and thus making the placement process less effective. As a result, the drop rate increases with higher thresholds, reaching 43% in some cases. This follows intuition since VMs are located further from each other, which means core links quickly become congested. We conclude that choosing different thresholds can impact the performance of our technique. In Section 5 we discuss this aspect further.

4.5 Comparison to VNM

Finally, we want to compare our approach with previous techniques.The closest work to ours is a virtual network mapping (VNM) technique previously proposed in [7]. In [7], the authors recognize the need to consider both physical node and links optimizations together throughout the placement process. Additionally, the algorithm controls the network allocation (routing) and therefore has more flexibility at finding a feasible placement. The algorithm in [7] finds a cluster of physical nodes that are lightly loaded without considering the network connectivity of the physical nodes and solves the routing problem of connecting physical nodes based on the topology of the virtual network. To match the authors' assumption that one physical node can only allocate resources for one virtual node, we apply our placement technique on generic VNIs whose VMs and links require at least 50% of a PM's compute resources and bandwidth capacity. We note that this is an unrealistic assumption since in practice, VM to PM den-



(a) Average path length per VNI (b) Average drop rate

Fig. 11: Comparison between our approach and the VNM technique proposed in [7].

sities as high as 100 VMs per PMs are the norm in cloud environments. However, it allows us to perform a more fair comparison between both techniques.

Our results for a 256-PM data center are presented in Fig. 11. As expected, satisfying the compute constraint of 50% demand imposes additional difficulty on our algorithm, which results in an increased drop rate to up to 25%. However, since the VNM technique does not consider the ample network connectivity of physical nodes, the resources chosen do not properly match the nature of VNIs. Consequently, the drop rate increases to up to 70% and the core links utilization increases as the average load of the data center increases. Similarly, the average path length per placed VNI is higher by a 2x factor. We conclude that considering locality and neighborhood quality in the cold spot discovery are primary factors for resource allocation in data centers.

5 Discussion

In this section, we briefly address some additional considerations towards a more complete VNI placement framework to be considered as future work.

VNI-aware dynamic cold spot discovery: We have shown that constructing cold spots based on network and compute resource availability suffices to achieve good placement performance for workloads with compute, network, and availability constraints. As workloads become more complex, the process of cold spot discovery needs to be extended to address workloads characteristics, e.g., proximity to specific storage devices. We favor an online approach wherein via learning mechanisms the cold spot discovery process is tuned to identify cold spots whose properties are aligned with the incoming workload.

Integration into a real system: Our technique is model driven, therefore to adopt it in a real environment one requires to build a model of the data center and the workloads. This is a simple software engineering task. In fact, significant part of the placement technique presented in this paper has been deployed in a data center environment.

Optimizations: First, we foresee being able to merge and split cold spots in order to produce cold spots with specific characteristics for different workloads. Second, when VNIs are destroyed, we do not keep track of their placement scheme. This could be used to improve the placement efficiency of future VNIs. Third, the placement algorithm could be improved by using A* algorithms so that partial placement decisions are based on estimations of the final resource fragmentation. This enhancement has the potential of pruning paths in the search space even further and achieve better placement outcomes.

6 Related work

Following we present a comparative analysis of similar research problems, as well as simpler versions of the VNI placement problem.

Virtual network placement. The VNI placement problem shares similarities with the Virtual Network Mapping (VNM) problem which plays a central role in building a virtual network (VN). During the mapping process each virtual node (link, respectively) is assigned to a node (path, respectively) of the physical network such that a set of resource constraints is satisfied. In the VNI problem however, we are concerned with a broader and finer-grained set of constraints in addition to network and compute resources. Several efficient heuristics have been developed to solve the VNM problem in the past [9–12]. Some of these restrict the search space by assuming the node place is given in advance and only solve the link embedding problem [12]. Others [11] rely on probabilistic metaheuristics such as simulated annealing and reduce complexity by type-casting the virtual routers and physical nodes. In [10] the authors reduce complexity by decomposing the network substrate and the virtual networks into known topologies and assume that a substrate node can only host one virtual node. Others such as [9] only focuses on the network aspects of the problem.

Topology-aware task mapping. The problem of placing task graphs in parallel computers is also similar to our problem. Tasks nodes must be placed on processors nodes while respecting the network communication constraints and the resource constraints of the processors. In the task graph placement problem however, compute resources are specified coarsely. Therefore, existing solutions focus on maximizing communication throughput. Typically, task-mapping algorithms consist of two stages: partitioning and mapping. In the partitioning stage, a *clustering* process groups together task nodes with high communication requirements. In our technique we have adopted and extended an existing clustering technique used in [6] for task graphs to cluster virtual machines. In [13] the authors present a topology-aware placement technique that considers the topology of the processor network when making placement decisions. Note that both works focus on the network aspects of the problem.

Network-aware virtual machine placement. In [5] the authors formulate the network aware virtual machine placement problem as an optimization problem and prove its hardness. The heuristic proposed assumes a homogeneous *slot* resource model, thus each physical machine can allocate a fix number of virtual machines and is only concerned with network and compute resources. Furthermore, its complexity is $O(n^2)$ where n is the number of physical machines. Our technique, in contrast, addresses the exploding combinatorial nature of the problem by identifying *cold spots* in advance and thus achieving better performance. In [4] a novel cloud network platform is proposed which extends the provisioning model of the Cloud to include a rich set of network services. In CloudNaas the placement of virtual machines and network demand is fully decoupled, i.e., a bin-packing technique is used to decide on the allocation of compute resources, while a separate technique is used to handle the allocation of network demand. This approach leads to resource fragmentation since compute and network resources can be unevenly utilized. We advocate for a more coupled approach that

considers the management of both network and compute resource into one single integrated solution. Finally, in [3] the authors propose using a placement engine based on an optimization solver to orchestrate multiple resources. This solver can take up to 6.1 seconds to load balance 1000VMs on a 15 PMs Cloud when only compute resources are considered. This is another evidence of the scalability challenge faced in provisioning resources in the Cloud. Note that we target solving the initial placement problem in sub-seconds on much larger systems.

7 Conclusions

We have considered the problem of placing virtual infrastructures with compute, network, and availability constraints in the Cloud. Unlike previous approaches, that address the placement problem from either the network or computer resources perspective, our approach factors in both in one integrated solution. We have developed a novel placement framework which makes the problem tractable and is generic enough to support increasing complexity. The center of our technique lies in the introduction of *cold spots*, defined as resource constructs that reduce the combinatorial complexity of the problem, while enabling the load balancing of resources. We have shown the effectiveness and efficiency of our approach with a rich set of workloads over extensive simulations.

References

1. Amazon: HPC Applications. `http://aws.amazon.com/hpc-applications/` (2012)
2. Bengoetxea, E.: Inexact graph matching using estimation distribution algorithms, Ecole Nationale Supérieure des Télécommunications (Paris), (2002)
3. Liu, C., Loo, B.T., Mao, Y.: Declarative automated cloud resource orchestration. In: Proceedings of SOCC'11, ACM (2011) 1–8
4. Benson, T., Akella, A., Shaikh, A., Sahu, S.: CloudNaaS: a cloud networking platform for enterprise applications. In: Proceedings of SOCC'11. (2011) 1–13
5. Meng, X., Pappas, V., Zhang, L.: Improving the scalability of data center networks with traffic-aware virtual machine placement. In: Proceedings of the 29th IEEE Conference on Computer Communications (INFOCOM'10), IEEE (2010) 1–9
6. Taura, K., Chien, A.: A heuristic algorithm for mapping communicating tasks on heterogeneous resources. In: Proceedings of HCW'00. (2000) 102–115
7. Zhu, Y., Ammar, M.: Algorithms for assigning substrate network resources to virtual network components. In: Proceedings of INFOCOM'06. (2006) 1–12
8. Amazon: EC2 instances. `http://aws.amazon.com/ec2/instance-types/` (2012)
9. Yu, M., Yi, Y., Rexford, J., Chiang, M.: Rethinking virtual network embedding: substrate support for path splitting and migration. SIGCOMM Computing Communications Review (2008) 17–29
10. Zhu, X., Santos, C., Beyer, D., Ward, J., Singhal, S.: Automated application component placement in data centers using mathematical programming. International Journal of Network Management **18** (2008) 467–483
11. Ricci, R., Alfeld, C., Lepreau, J.: A solver for the network testbed mapping problem. SIGCOMM Computing Communications Review **33** (2003) 65–81
12. Szeto, W., Iraqi, Y., Boutaba, R.: A multi-commodity flow based approach to virtual network resource allocation. In: Proceedings of GLOBECOM'03. (2003)
13. Agarwal, T., Sharma, A., Laxmikant, A., Kale, L.: Topology-aware task mapping for reducing communication contention on large parallel machines. In: Proceedings of IPDPS'06. (2006)