Artificial Intelligence 278 (2020) 103196

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Scalable constraint-based virtual data center allocation

Sam Bayless [a,*], Nodir Kodirov [a], Syed M. Iqbal [a], Ivan Beschastnikh [a], Holger H. Hoos [b,a], Alan J. Hu [a]

[a] University of British Columbia, Canada
[b] Universiteit Leiden, the Netherlands

ARTICLE INFO

Article history:
Received 14 December 2017
Received in revised form 19 October 2019
Accepted 25 October 2019
Available online 31 October 2019

Keywords:
Constraint solver
Data center
Virtual data center
Allocation

ABSTRACT

Constraint-based techniques can solve challenging problems arising in highly diverse applications. This paper considers the problem of virtual data center (VDC) allocation, an important, emerging challenge for modern data center operators. To address this problem, we introduce NETSOLVER, a system for VDC allocation that is based on constraint solving. NETSOLVER represents a major improvement over existing approaches: it is sound, complete, and scalable, providing support for end-to-end, multi-path bandwidth guarantees across all the layers of hosting infrastructure, from servers to top-of-rack switches to aggregation switches to access routers. NETSOLVER scales to realistic data center sizes and VDC topologies, typically requiring just seconds to allocate VDCs of 5–15 virtual machines to physical data centers with 1000+ servers, maintaining this efficiency even when the data center is nearly saturated. In many cases, NETSOLVER can allocate 150% – 300% as many total VDCs to the same physical data center as previous methods. Finally, we show how NETSOLVER can be extended with additional optimization constraints, such as VM affinity and hotspot minimization, demonstrating the flexibility of our approach.

The performance and flexibility of NETSOLVER are made possible by our formalization of the VDC allocation problem in terms of multi-commodity flows, and the corresponding efficient handling of network flow problems in the underlying constraint solvers. This shows the importance of supporting flow-based constraints, which are more mature in ILP- vs. SMT-based constraint solving.

1. Introduction

Constraint-based techniques, such as Integer Linear Program (ILP) and SAT modulo theory (SMT) solvers, play a key role in state-of-the-art approaches for solving challenging problems across a wide range of applications (see, e.g., [1–4]). In this work, we demonstrate how virtual data center (VDC) allocation, a prominent and increasingly important problem arising in the operation of modern data centers, can be tackled using a pair of high-performance constraints solvers: Gurobi [5] and MONOSAT [6]. We obtain substantial improvements in performance and functionality over previous VDC allocation techniques. Central to our results is the formalization of VDC allocation in terms of multi-commodity flows, allowing us to exploit the efficient handling of network-commodity flow problems in Gurobi and MONOSAT. We are the first to demonstrate that constraint solvers can be successfully applied to this setting at full data center scales (1000+ servers), while also improving on the state-of-the-art.

* Corresponding author.
E-mail address: sbayless@cs.ubc.ca (S. Bayless).

https://doi.org/10.1016/j.artint.2019.103196

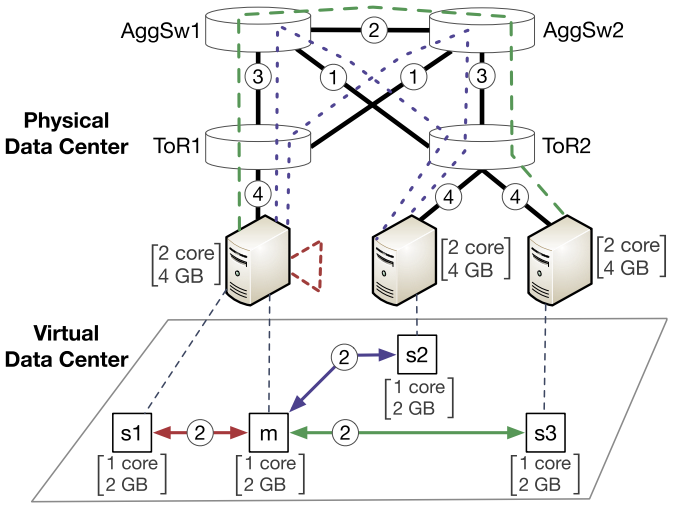


Fig. 1. (Top) Example physical data center topology with three physical servers, two top-of-rack (ToR) switches, and two aggregation switches (AggSw). Circled numbers on links denote available bandwidth in Gbps. **(Bottom)** Example Hadoop VDC with one master (m) and three slave VMs (s1-s3) with a required throughput of 2 Gbps between each slave and the master (shown in circles). Each VM also requires a certain number of CPU cores and RAM. The problem is to find an allocation of the VDC to the physical data center, for example, as illustrated with the dashed lines. Note that s1 and m are mapped to the same physical server, while the virtual link m – s2 is allocated a multi-path route.

A VDC consists of multiple communicating virtual machines (VMs), each with individual server resource requirements (e.g., CPU or RAM), along with a virtual network of pair-wise bandwidth requirements between the VMs. The *VDC allocation problem* is to find a valid allocation of VMs to servers and links in the virtual network to links in the physical network. A valid allocation satisfies the compute, memory, and network bandwidth requirements of each VM across the entire data center infrastructure, including servers, top-of-rack (ToR) switches, and aggregation switches [7,8]. Fig. 1 shows a simple instance of the VDC allocation problem and one solution.

Here, a virtual topology representing a 4-node Hadoop VDC (bottom) needs to be mapped onto a physical data center with three physical servers, two ToR switches, and two aggregation switches (top). The allocation is indicated with dashed lines. For example, the VMs for s1 and m are mapped to the same physical server and the virtual link s2 – m is allocated a multi-path route, in which each sub-path provides 1 Gbps. As we will discuss in Section 2, support for end-to-end and multi-path are two characteristics that distinguish capabilities of our tool, NETSOLVER, from prior tools.

In this work, we introduce NETSOLVER,[1] a constraint-based virtual data center allocation procedure that is scalable, sound, and complete, with support for end-to-end, multi-path bandwidth guarantees across all the layers of the networking infrastructure, from servers to top-of-racks to aggregation switches to access routers. NETSOLVER efficiently allocates VDCs with a dozen or more VMs to full-size physical data centers (with 1000+ servers), typically in seconds per allocation. Across a wide variety of data center topologies, NETSOLVER can allocate 150% – 300% as many total VDCs to the same physical data center as state-of-the-art heuristic methods, such as SecondNet's VDCAlloc algorithm [9].

Furthermore, NETSOLVER offers the flexibility and extensibility characteristics of a constraint-based approach. In real-world applications, DC operators often need to support additional constraints or optimization goals while allocating VMs (such as ensuring that certain VMs are placed together). We demonstrate that NETSOLVER can be easily extended to support additional VDC allocation constraints: VM affinity constraints, minimization of the total number of utilized servers, and constraints to load-balance allocations and avoid hotspots.

This paper extends results previously published in [10] with significant new capabilities (Section 6), a new ILP-based solver back end (Section 4.1), and broadened experimental results (Section 5). Additionally, we have improved the overall runtime performance of NETSOLVER by $\approx 15\%$ through the use of automatic algorithm configuration [11] (Section 5 and Appendix A), as well as upgrading the SMT solver (MONOSAT) from version 1.4 to version 1.6. We use Gurobi 8.1.0 (linux64, Python) for all experiments.

2. Related work

We now survey prior work with respect to features of VDC allocation that are relevant to modern data centers. As can be seen from Table 1, all prior approaches have important limitations relative to the contributions of this paper.

[1] See https://www.cs.ubc.ca/labs/nss/projects/netsolver-aij-2019.

Table 1
A comparison of the features of contemporary sound VDC allocation algorithms and four recent VNE algorithms: GAR-SP/PS (and variant RW-MM), D-ViNE, and ASID, based on linear programming, mixed integer programming, and subgraph isomorphism detection, respectively.

Algorithm	Sound	Complete	Multi-path	Multi-VM	VDC topology	DC topology
SecondNet [9]	✓				All	All
Importance Sampling [12]	✓			✓	All	Tree
Oktopus [13]	✓			✓	Star	All
VDCPlanner [14]	✓			✓	All	All
HVC-ACE [15]	✓		✓	✓	Hose	All
GAR-SP/PS [16]	✓		✓	✓	All	<200 nodes
RW-MM-SP/PS [17]	✓		✓		All	<200 nodes
D-ViNE [18]	✓		✓		All	<200 nodes
ASID [19]	✓				All	<200 nodes
VirtualRack [20]	✓	✓			Hose	All
Z3-AR [21]	✓	✓	✓		All	Tree
NETSOLVER (this paper)	✓	✓	✓	✓	All	All

Soundness Sound VDC allocation tools respect end-to-end bandwidth guarantees, while unsound tools only attempt to minimize data center network traffic without a guarantee that VMs will have sufficient dedicated bandwidth. Examples of unsound approaches to VDC allocation include [22,23], which dynamically identify VM communication patterns through network traffic analysis.

This prior work is in contrast to the approaches discussed in this paper, all of which, including our contribution, NETSOLVER, are sound and assume that VDCs and their communication requirements are explicitly known to the allocator.

Completeness Most VDC allocation tools that respect bandwidth guarantees are *incomplete*: they can fail to find feasible VDC allocations in cases where such allocations exist (even when given unlimited runtime). Oktopus [13], VDCPlanner [14], HVC-ACE [15], and SecondNet [9] are examples of incomplete allocation algorithms. For example, SecondNet's algorithm is greedy in that it maps VMs to servers before checking for available paths, and allocates bandwidth one path at a time; if either of these steps fail, it will fail to allocate the VDC.[2] Similarly, Hadrian [24], Cicada [25], and CloudMirror [8] use incomplete greedy heuristic algorithms that attempt to co-locate VMs of the VDC in the smallest physical DC sub-tree. Hadrian models VDC network allocation as a maximum flow problem, finding the shortest network path with sufficient capacity to carry each VDC's VM-to-VM traffic.[3] Pulsar [26] uses Hadrian's VDC allocation algorithm and extends it to accommodate VM-appliances (such as SSDs and encryption devices).

In contrast with this prior work, the constraint-solver-based approaches described in [21] and NETSOLVER are both complete: they are guaranteed to (eventually) find a feasible allocation if one exists. We will show in our experiments that completeness does not merely represent a theoretical benefit, but can translate into substantial gains in practical allocation capability. NETSOLVER is the first sound and complete VDC allocator that can be applied to any VDC and data center topology without simplifying abstractions.

Multi-Path Allocations Many data centers use multi-path allocations to maximize bandwidth and to provide fault-tolerance and load-balancing [27,28]. Lack of multi-path support in traditional L2/L3-based networks was a primary motive for data center operators to develop networking stacks with multi-path support [29]. There are now multiple efforts underway to eliminate this restriction, which include using architectures specifically designed for multi-path routing, e.g., BCube [30], VL2 [31], and making the data center networking fabric itself multi-path [32].

Despite the increasing importance of multi-path routing, to the best of our knowledge, there is only one previous VDC allocator that supports multi-path communication between VMs: HVC-ACE [15], a sound but incomplete allocator that uses a *hose-model* for VDCs (we describe hose-models below). There are also several incomplete algorithms for virtual network embedding that have support for multi-path allocation for smaller physical networks with 50–150 servers [16,18,17]. NETSOLVER is the *first* sound and complete multi-path tool for VDC allocation.

Multi-VM Allocations Some tools simplify VDC placement by assuming that the VMs in a VDC must all be placed on separate servers. For example, SecondNet [9] uses bipartite graph matching to assign VMs to servers; as a result, it can place only a single VM per server when allocating a given VDC. Similarly, VirtualRack's [20] virtual tree abstraction places each VM into a separate leaf node server. D-ViNE [18] uses mixed-integer programming to perform virtual network embedding, but their encoding does not support allocating multiple virtual nodes per

[2] In fact, SecondNet will try this process several times on different sub-sets of the data center before giving up.

[3] Note that NETSOLVER, our approach, also models VDC allocation using maximum flow, but combines this with additional constraints to achieve completeness.

server. In many cases, it can be advantageous to place multiple VMs on one server, since communication between the co-located VMs is cheap. Multi-VM placement is useful to take advantage of data locality between VMs and can be explicitly requested by a tenant. Conversely, a tenant may want single-VM placement for higher fault tolerance. For example, VMs hosting different database replicas can be assigned to different servers to decrease fate-sharing. Section 6 discusses a VDC used in a commercial setting that requires the multi-VM property as part of the tenant's request. By default, NETSOLVER performs multi-VM placement. However, NETSOLVER also supports anti-affinity constraints (as well as other advanced placement options, discussed in Section 6), which can be used to force some or all of the VMs in a given VDC to be placed on disjoint servers.

Unrestricted Topologies Many VDC allocators simplify the problem, either by abstracting VDC topologies into simpler ones that are easier to allocate, or by restricting the physical data center to simpler topologies. For example, the abstraction-refinement encodings from [21] only apply to tree-topology data centers. Oktopus [13] abstracts VDCs into virtual clusters, which are VMs connected to central virtual switch in a star topology. VirtualRack [20] and HVC-ACE [15] use a less-restricted *hose-model* [33] abstraction for VDCs. A hose-model only allows one to specify aggregate, rather than pairwise, bandwidth requirements for virtual machines – that is, each VM is guaranteed a certain amount of ingress and egress bandwidth into the virtual network as a whole, but is not guaranteed to have any specific amount of bandwidth to any specific VM. Hose-models generalize the star-topology used in Oktopus, but cannot, for example, model virtual networks that include cycles or (non-trivial) trees. NETSOLVER is the first sound and complete VDC allocation approach that supports arbitrary VDC and data center topologies.

As observed by Ballani et al. [24], VDC allocation is closely related to *virtual network embedding* (VNE) [34,35]. The VNE literature, however, has focused on allocating virtual networks onto substrate networks that are representative of medium-sized ISPs, with 50–150 servers and few or no intermediate switches (*e.g.,* recent VNE tools: GAR-SP/PS [16], RW-MM-SP/PS [17], D-ViNE [18], ASID [19], and [36] all fall into this range). In contrast, work on VDC allocation has typically focused on allocating to larger physical networks with topologies representative of typical data centers, often with thousands (or even hundreds of thousands) of servers, along with intermediate switches [9,37]. Therefore, even though the problem definitions in the VNE and VDC literature often overlap, VDC tools have made different trade-offs to focus on scalability. We compare NETSOLVER to several representative VNE approaches in Section 5.4 and confirm that these tools perform poorly on typical VDC instances.

3. The multi-path VDC allocation problem

The problem we consider in this work is defined as follows. We are given the description of a physical network (PN) and a virtual data center (VDC). The PN is specified through a set of servers S, switches N, and a directed (or undirected) graph $(S \cup N, L)$, with capacities $c(u, v)$ for each link in L. The VDC consists of a set of virtual machines VM and a set $R \subseteq VM \times VM \times \mathbb{Z}^+$ of directed (or undirected) bandwidth requirements between those machines. For each server $s \in S$, we have CPU core, RAM, and storage capacity specifications, $cpu(s), ram(s), storage(s)$, and for each virtual machine $v \in VM$, we are given CPU core, RAM, and storage requirements $cpu(v), ram(v), storage(v)$.

The objective in the multi-path VDC allocation problem is to find an assignment $A : VM \mapsto S$ of virtual machines $v \in VM$ to servers $s \in S$ along with an assignment of non-negative bandwidth $B_{u,v}(l)$ to links $l \in L$ for each bandwidth requirement $(u, v, b) \in R$, satisfying the following constraints:

- **Local VM allocation constraints** (**L**) ensure two properties: First, that each virtual machine is assigned to exactly one server:

$$\forall v \in VM : \sum_{s \in S} \big(A(v) = s\big) = 1$$

Secondly, that each server provides sufficient CPU core, RAM, and storage resources to accommodate the requirements of all VMs allocated to it:

$$\forall s \in S : \left(\sum_{v \in V(s)} cpu(v) \leq cpu(s)\right) \wedge \left(\sum_{v \in V(s)} ram(v) \leq ram(s)\right) \wedge \left(\sum_{v \in V(s)} storage(v) \leq storage(s)\right),$$

where $V(s) = \{v \in VM \mid A(v) = s\}$.

Resource requirements are modeled using integer values, and VMs do not share resources.

- **Global bandwidth allocation constraints** (**G**) ensure that sufficient bandwidth is available in the physical network to satisfy all bandwidth requirements between pairs of VMs. We formalize this by requiring that for all $(u, v, b) \in R$, the bandwidth assignments $B_{u,v}(l)$ must form a valid network flow with its source at the node that u is allocated in, $A(u)$, and its sink at the node that v is allocated in, $A(v)$. Further, we require that network flow to be no smaller than the required bandwidth b, and that none of the link capacities l in the physical network is exceeded: $\forall l \in L : \sum_{(u,v,b)\in R} B_{u,v}(l) \leq c(l)$. Bandwidths are represented by integer values; bandwidth between VMs allocated on the same server is unlimited.

It has been previously observed [38,39,16,18] that when allowing path-splitting, the global bandwidth allocation constraints give rise to a multi-commodity flow problem, which is strongly NP-complete even for undirected integral flows [40]. Conversely, any multi-commodity flow problem maps directly into bandwidth constraints above, establishing the NP-hardness of the multi-path VDC allocation problem [18].[4]

4. NetSolver

Previous work on constraint-based virtual machine placement has drawn on techniques from two communities: integer linear programing (ILP), and SAT modulo Theories (SMT). In this section, we will describe how the VDC allocation can be implemented and efficiently solved using either Gurobi, a state-of-the-art Integer Linear Program (ILP) solver [5], or MonoSAT, a SAT modulo theory (SMT) solver [6].

4.1. Encoding multi-path VDC allocation in ILP

ILP solvers are commonly used for solving maximum flow and multi-commodity flow problems, and are widely cited in the literature for that use-case, across a broad range of applications (e.g., [41,42,25]). CPLEX and Gurobi, for example, are able to automatically recognize properly encoded multi-commodity flow problems and handle them internally using special-cased techniques [43,5]. The details of how these solvers handle flow problems are proprietary, but examples of such approaches are discussed in the literature (e.g., [44]).

The local and global constraints **L** and **G** defined in Section 3 are directly expressible as integer linear programming constraints. We describe these fully below:

For each $v \in VM$ and each $s \in S$, we introduce a binary variable $A_{v,s}$ to represent whether v is placed on s, and add constraints to enforce that each virtual machine is placed on exactly one server:

$$\forall v \in VM : \sum_{s \in S} A_{v,s} = 1$$

Then, for each of the three resource constraints (CPU, RAM, Storage) we enforce the resource constraints:

$$\forall s \in S : \left(\sum_{v \in VM} A_{v,s} \cdot cpu(v) \right) \leq cpu(s)$$

$$\forall s \in S : \left(\sum_{v \in VM} A_{v,s} \cdot ram(v) \right) \leq ram(s)$$

$$\forall s \in S : \left(\sum_{v \in VM} A_{v,s} \cdot storage(v) \right) \leq storage(s)$$

Together, the above constraints enforce the local constraints **L**.

Similarly, we directly encode the global constraints **G** as multi-commodity flow constraints: For each $v \in VM$, and each link $l = (a, b) \in L$, we introduce a non-negative integer variable $B_{v,a,b}$, representing the bandwidth on link l on the physical data center originating from VM v. We then assert that for each link, the sum of all bandwidth originating from all VMs does not exceed the capacity of that link.[5]

$$\forall (a, b) \in L : \Big(\sum_{v \in VM} (B_{v,a,b}) \Big) \leq c(a, b)$$

[4] In principle, this multi-commodity flow problem can be solved efficiently via linear programming for real-valued flows, but this approach does not support the local VM allocation constraints. Approaches that express the global constraints as a linear program therefore either require an additional mechanism to perform local server allocation [16] or use mixed integer programming [18].

[5] Note: this constraint assumes that the links of the physical network are directed (e.g., full duplex). If the physical network has undirected links, we can instead enforce that the sum of the bandwidth in both directions on each link $l = (a, b)$ does not exceed $c(l)$.

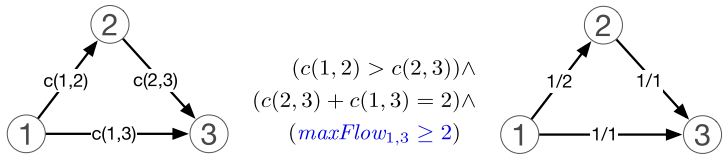


Fig. 2. **(a)** Example symbolic graph, with variable capacities $c(u, v)$ on each edge includes the capacity assigned to each edge, as well as the flow along that edge. **(b)** A formula constraining the graph. **(c)** A solution, assigning a flow and a capacity (f/c) to each edge.

Next, we enforce the network flow constraints on the switches for each originating VM. Since VMs cannot be placed on switches, this simply enforces that the incoming flow equals the outgoing flow for each switch and each source VM:

$$\forall v \in VM, \forall n \in N : \sum_{(n,b)\in L} B_{v,n,b} = \sum_{(a,n)\in L} B_{v,a,n}$$

Finally, we enforce the network flow constraints on each server. Here we add two extra terms that account for the flow entering the network at the server that the source VM is placed on, and the flow exiting the network at servers that destination VMs are placed on:

$$\forall v \in VM, \forall s \in S :$$
$$\sum_{(n,b)\in L} B_{v,n,b} + \sum_{w\in V\setminus v} \left(R(v,w)\cdot A_{v,s}\right) - \sum_{w\in V\setminus v} \left(R(w,v)\cdot A_{v,s}\right) = \sum_{(a,n)\in L} B_{v,a,n}$$

Gurobi has the ability to incrementally re-solve a system of equations after changing the co-efficients. In the above equations, the constants that define the resource requirements and bandwidth requirements of the VDC, and that define the capacities of each physical server and link in the physical network, appear as constants. So long as there is a bound on the number of virtual machines per VDC, the same set of constraints can be re-used for subsequent allocations, after updating each of those constant values appropriately (for example, to subtract used bandwidth from the capacities of the links of the physical data center, or to alter the bandwidth requirements between two VMs). Our implementation makes use of this incremental solving capability when encoding successive VDC allocations; doing so results in a substantial performance improvement.

4.2. Encoding multi-path VDC allocation into SMT

In contrast to ILP solvers, SMT solvers have not traditionally been applied to large multi-commodity flow problems. As a result, techniques for handling network flow problems efficiently in SMT are less mature and require some additional discussion. In this section, we will describe how MonoSAT, a recently introduced SMT solver with support for single-commodity network flow problems, can be used to solve multi-commodity network flow problems in a practically useful way.

MonoSAT is a SAT modulo theory (SMT) solver that extends quantifier-free first-order Boolean logic with highly efficient, built-in support for a wide set of *finite monotonic predicates* [6]. MonoSAT is different from other SMT solvers in that it has built-in predicates for (single-commodity) *s-t* maximum flow. While this does not directly provide support for multi-commodity flows, we will show that by expressing multi-commodity flows as a combination of single-commodity maximum flow predicates, we can use MonoSAT to solve large multi-commodity flow problems – a first for SMT solvers.

By combining this encoding for the global constraints **G** with a pseudo-Boolean encoding of the local constraints **L**, we are able to tackle the full multi-path VDC allocation problem in MonoSAT.

Intuitively, a finite monotonic predicate is a predicate for which increasing the value of its arguments can never change the value of the predicate from true to false, e.g., adding links to a network can only increase the connectedness of the network. MonoSAT supports many common graph constraints, such as reachability, shortest paths, minimum spanning trees, and maximum flows. MonoSAT also supports a subset of the theory of fixed-width bitvectors.

MonoSAT accepts formulas with one or more directed *symbolic graphs*, each of which comprises a fixed set of nodes and symbolic edges (u, v). Each edge has an integer capacity, $c(u, v)$, which may be either a constant or a variable (a fixed-width bitvector). Finally, MonoSAT supports a number of common graph predicates, of which only one is relevant here: $maxFlow_{s,t,G} \geq f$, where G is a directed graph, s and t are nodes in G, and f is a constant integer or a bitvector term. This predicate is True iff the maximum *s-t* flow in G, *under assignment to the edge capacities associated with* G, is greater or equal to f.

As an example, consider the directed graph G shown in Fig. 2a, with variable integer capacities $c(u, v)$, and the formula in Fig. 2b. In this example, MonoSAT finds edge capacities that satisfy the constraints and also produces a flow satisfying the maximum flow predicate in Fig. 2c.

In the remainder of this section, we will first describe how we model integer-value multi-commodity flow in terms of the built-in maximum flow predicates supported by MonoSAT; then we will show how to use these multi-commodity flow constraints to express VDC allocation. More extensive discussion about MonoSAT can be found in [6,45].

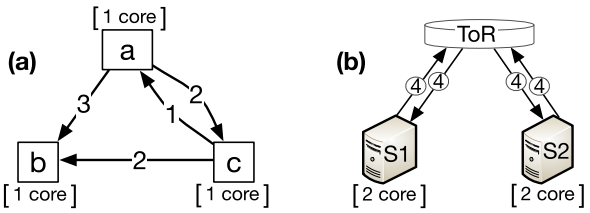


Fig. 3. (a) A VDC with 3 VMs, and 4 directed bandwidth constraints. In this example, each VM requires 1 core, and has no RAM requirements. VM a requires 3 Gbps of outgoing bandwidth to VM b, and 2 Gbps to VM c. VM c also has bandwidth requirements to VM a and b, while VM b requires no outgoing bandwidth. **(b)** A physical datacenter with two servers and one Top-of-Rack (ToR) switch. Each server has 2 cores, and has 4Gbps of bandwidth available to and from the switch.

4.2.1. *Multi-commodity flow in* MonoSAT

While there are many obvious ways to encode multi-commodity flows in SMT solvers, the one we present here is, to the best of our knowledge, the only SMT encoding to scale to multi-commodity flow problems with thousands of nodes. As there are many applications to which SMT solvers are better suited than ILP solvers (and vice-versa), this SMT formulation has many potential applications beyond VDC allocation.

Given a directed graph $G = (V, E)$, an integer capacity $c(u, v)$ for each edge $(u, v) \in E$, and a set of commodity demands K, where a commodity demand $i \in K$ is a tuple (s_i, t_i, d_i), representing an integer flow demand of d_i between source $s_i \in V$ and target $t_i \in V$, the integral multi-commodity flow problem is to find a feasible flow such that each demand d_i is satisfied, while for each edge (u, v) the total flow of all demands (summed) is at most $c(u, v)$:

$$f_i(u, v) \geq 0, \quad \forall (u, v) \in E, i \in K$$

$$\sum_{i \in K} f_i(u, v) \leq c(u, v), \quad \forall (u, v) \in E$$

$$\sum_{v \in V} f_i(u, v) - \sum_{v \in V} f_i(v, u) = \begin{cases} 0, \text{if } u \notin \{s_i, t_i\} \\ d_i, \text{if } u = s_i \\ -d_i, \text{if } u = t_i \end{cases}, \forall (s_i, t_i, d_i) \in K$$

We instantiate symbolic graphs $G_{1..|K|}$ with the same topology as G. We set the capacities of each edge $(u, v)_i \in G_i$ to a new integer variable, $c(u, v)_i$, with constraint $0 \leq c(u, v)_i \leq c(u, v)$. Next, we assert that the capacities in each graph sum to no more than the original edge capacity: $\sum_{i=1}^{|K|} c(u, v)_i \leq c(u, v)$. Together, these constraints partition the original capacity graph into K separate graphs, one for each demand. To complete the encoding, for each commodity demand (s_i, t_i, d_i), we use MonoSAT's built-in maximum flow constraints to assert that the maximum s_i–t_i flow in G_i is at least d_i.

In our formulation, we explicitly enforce only that the maximum s_i–t_i flow in G_i is $\geq d_i$, as opposed to enforcing that the maximum flow is exactly d_i. Notice that a flow that is greater than d_i will necessarily contain a flow that is equal to d_i, and that an exact d_i flow can be easily recovered if necessary (e.g., with one extra application of any standard maximum flow algorithm). Alternatively, an extra, external 'source' node can be added to the graph, with exactly one edge of capacity d_i leading to the original source node from this new, extra 'source' node. This will ensure that the maximum possible s_i–t_i flow is at most d_i.

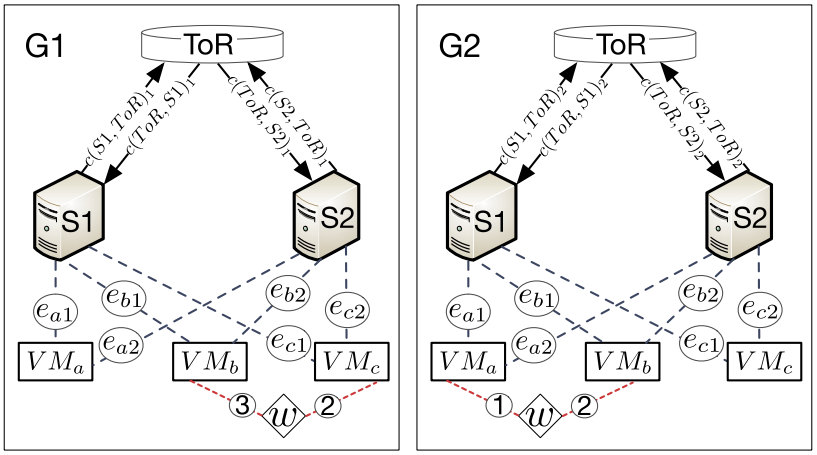
We implement our constraints in this way to improve the performance of the underlying constraint solver. In MonoSAT, it is typically more efficient to enforce one-sided $(>, \geq, \leq, <)$ constraints, rather than two-sided $(=, \neq)$ constraints. This is because all theory predicates in MonoSAT must be monotonic, and so equality needs to be implemented as two (individually monotonic) one-sided comparison atoms.

4.2.2. *Multi-path VDC allocation in MonoSAT*

In this section, we will show how the global and local constraints described in Section 3 can be encoded into MonoSAT, and used to perform VDC allocation. As a running example, we will consider a small VDC and physical data center (Fig. 3). These examples are much smaller than the ones we will consider in Section 5: In a typical application, the physical datacenter might have 1000s of servers, while the VDC might have 10–30 VMs.

The global constraints **G** (Section 3) can be encoded as a multi-commodity flow as described in the previous section, with up to $|VM|^2$ commodity demands (one for each bandwidth tuple $(u, v, bandwidth) \in R$).[6] However, we can greatly improve on this by merging bandwidth constraints that share a common source into a single commodity demand: Given a set of bandwidth constraints $(u, v_i, bandwidth_i) \in R$ with the same source u, we can convert these into a single commodity demand, by adding an extra node $w \notin VM$, along with edges (v_i, w) with capacity $bandwidth_i$. The commodity demands

[6] Note that in our approach, bandwidth values are required to be integers, as we are restricted to finding integer maximum flows. In practice, this is not a limitation, as data centers typically only offer a small number of (integer-valued) bandwidth/CPU choices to clients.



$(e_{a1} \vee e_{a2}) \wedge (e_{b1} \vee e_{b2}) \wedge (e_{c1} \vee e_{c2})$ Each server is assigned to a server (1)

$(\neg e_{a1} \vee \neg e_{a2}) \wedge (\neg e_{b1} \vee \neg e_{b2}) \wedge (\neg e_{c1} \vee \neg e_{c2})$ No VM is assigned to more than 1 server (2)

$(\neg e_{a1} \vee \neg e_{b1} \vee \neg e_{c1}) \wedge (\neg e_{a2} \vee \neg e_{b2} \vee \neg e_{c2})$ Each server has sufficient CPUs for its VMs (3)

$c(S1,ToR)_1 + c(S1,ToR)_2 \leq 4$
$c(ToR,S1)_1 + c(ToR,S1)_2 \leq 4$
$c(S2,ToR)_1 + c(S2,ToR)_2 \leq 4$
$c(ToR,S2)_1 + c(ToR,S2)_2 \leq 4$ Capacities on each physical edge sum to $\leq c$ (4)

$G1.maxflow(VMa, w) \geq 5$ Maximum flow from VMa to w in G1 is ≥ 5 (5)

$G1.maxflow(VMc, w) \geq 3$ Maximum flow from VMc to w in G2 is ≥ 3 (6)

Fig. 4. Two symbolic graphs $G1, G2$, and the corresponding constraints enforcing allocation for the VDC and PN of Fig. 3. Edges e_{vs} control which VMs are placed on which servers, and have the same assignments in the two graphs. Edges marked with integers have constant capacities; edges e have unlimited capacity, and edges c have variable, non-negative integer capacities. In this example, constraints 2 and 3 are simple enough to be expressed with a few clauses, but for more complex examples we would use pseudo-Boolean constraints. Constraint 4 is enforced using bitvector arithmetic, while 5 and 6 use the built-in maximum flow predicates of MonoSAT.

$(u, v_i, bandwidth_i)$ can then be replaced by a single commodity demand $(u, w, \sum_i bandwidth_i)$. As there are at most $|VM|$ distinct sources in R, this reduces the number of demands from $|VM|^2$ in the worst case to $|VM|$ demands.[7]

In our example from Fig. 3, the VDC has 4 directed bandwidth requirements, but only two distinct bandwidth sources (VMa and VMc). So we can safely merge these 4 bandwidth requirements into two multi-commodity flow constraints. In cases where the VDC is undirected, we improve on this further by swapping sources and sinks in communication requirements so as to maximize the number of requirements with common sources. This can be done efficiently even for large networks by finding an approximate minimum-cost vertex cover of R (e.g., using the 2-opt approximation from [46]).

We first construct an undirected graph of communication requirements, with an undirected edge of weight $(u, v) = bandwidth$ for each bandwidth requirement and find an approximate minimum-cost vertex cover. Necessarily, each edge, and hence each communication requirement, will have at least one covering vertex. For each requirement $(u, v, bandwidth)$, if v is a covering vertex and u is not, we replace the requirement with $(v, u, bandwidth)$, swapping u and v. After swapping all uncovered source vertices in this way, we then proceed to merge requirements with common sources as above. For cases where the VDC is directed, we skip this vertex-cover optimization and only merge together connection requirements with the same (directed) source in the input description. Given this set of commodity demands, we construct an undirected (or directed) graph G consisting of the physical network $(S \cup N, L)$, and one node for each virtual machine in VM. If any VDC communication requirements $(u, v_i, bandwidth_i)$ have been merged into combined requirements $(u, w, \sum bandwidth_i)$ as above, we add additional, directed edges (v_i, w) with capacity $bandwidth_i$ to G.

In our running example, we had two multi-commodity flow constraints, so we will construct two graphs (Fig. 4), $G1$ and $G2$. For each $v \in VM$ and each server $s \in S$, we add a directed symbolic edge e_{vs} from v to s with unlimited capacity to G; this edge controls the server to which each VM is allocated. Next, we assert (using a cardinality constraint) that for each VM v, exactly one edge e_{vs} is enabled, so that the VM is allocated to exactly one server: $\forall v \in VM : \sum_s e_{vs} = 1$. Using the multi-commodity flow encoding described above, we assert that the multi-commodity flow in G satisfies $(u, v, bandwidth)$ for each commodity requirement. The above constraints together enforce global constraints **G**; to enforce local constraints **L**, we use pseudo-Boolean constraints (using the efficient SAT encodings described in [47]) to assert: $\sum_v cpu(v) \leq cpu(s) \wedge \sum_v ram(v) \leq ram(s) \wedge \sum_v storage(v) \leq storage(s)$. A satisfying solution to our running example, implementing all of the above constraints, is shown in Fig. 5.

[7] Converting a single-source, multi-destination flow problem into a single-source, single-destination maximum flow problem is a well-known transformation, and safely preserves the maximum possible flow to each destination.

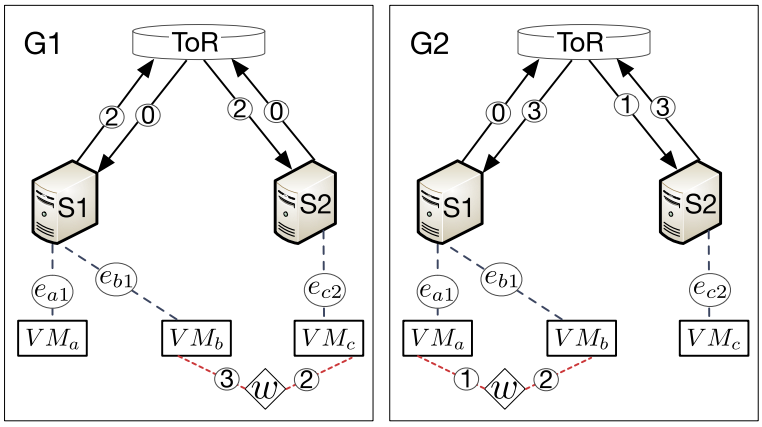


Fig. 5. A satisfying assignment to these constraints (showing only the edges that are assigned to 'true'). Notice that the e_{vs} assignments must be the same in the two graphs. The capacity assignments c are each at least large enough to allow for the required flow between the assigned VMs (but may be larger than required, as is the case for $c(ToR, S2)$), and the individual capacities assigned to each edge across the two graphs sum to at most the bandwidth available on each edge of the data center (4, in this case).

Note that these encodings are novel contributions and critical to NetSolver-SMT's performance; however, they are empirically efficient only because MonoSAT (unlike other SMT solvers) has built-in support for network flow constraints. As we show next, carefully crafted encodings alone, such as the one developed in Z3-AR [21], are not competitive. Instead, fundamental improvements in the constraint solver, such as the ones we use in MonoSAT, are necessary.

4.2.3. Reusing constraints

As with the ILP-based approach, it's important to use incremental solving in the SMT-based approach as well. The preceding discussion assumes that the VDC topology is constant and known in advance. However, in a real data center environment, it is typically the case that one will want to allocate VDCs of differing topologies. We briefly summarize here how we extend the above encoding to support this use case (which we will demonstrate in Section 5).

Many SAT solvers, including MonoSAT, support an 'assumption' mechanism [48] allowing for a formula to be repeatedly solved under multiple, differing restricted portions of the search space (that is, under an *assumed* set of assignments). In order to support allocating VDCs of differing topologies, without needing to re-encode the entire set of constraints in a new solver at each allocation (which would be prohibitively expensive), we initially encode a VDC topology that is the superset of all the VDCs to be allocated. Then, for each individual VDC to allocate, we use the assumption mechanism to temporarily disable portions of that superset VDC topology in the formula, such that only the edges corresponding to the current VDC to be allocated remain enabled in the solvers search space. In this way we can efficiently reuse the same solver to perform each allocation, while supporting VDCs of multiple sizes (as well as supporting the de-allocation of previously allocated VDCs).

5. Evaluation

We now present results from an extensive empirical evaluation demonstrating that our approach offers substantial advantages compared to state-of-the-art methods for VDC allocation. Specifically, we compare the performance of the ILP and SMT versions of NetSolver to that of SecondNet's VDCAlloc [9] — a seminal, sound VDC allocation algorithm with end-to-end bandwidth guarantees — and the Z3-based abstraction-refinement procedure from [21], which resembles our approach in that it makes use of a constraint solver (SMT). In each experiment, the algorithms repeatedly allocate VDCs to the DC until they are unable to make further allocations (or until a 1 CPU hour timeout is reached). This metric, the number of total VDC allocations, was also used in prior work [9,21] and is important in practice, as it captures data center utilization. Except where noted, experiments were run on a server with a 2.40 GHz (10 MB L3 cache) Intel Xeon E5-2407 v2 processor with 8 cores across 2 NUMA nodes and hyperthreading disabled. The server uses Ubuntu 16.04.6 LTS with 96 GB RAM that is uniformly distributed (48 GB each) across both NUMA nodes. All experiments are limited to 80 GB RAM and 1 hour of CPU-time. No experiment actually consumed 80 GB of RAM.

SecondNet's VDCAlloc algorithm ('SecondNet', except where ambiguous) is an incomplete, heuristic-driven algorithm that can find VDC allocations for physical networks with hundreds of thousands of servers. As SecondNet is based on bipartite matching, it fundamentally cannot allocate more than one VM in each VDC to any given server. Furthermore, because it performs allocation in an incomplete, greedy fashion, especially in heavily utilized networks, it can fail to find a feasible allocation. As we will demonstrate, under many realistic circumstances, this happens quite frequently, leading to substantially lower DC utilization than can be achieved with a complete method, such as NetSolver.

The above-mentioned, constraint-solving-based work [21] introduced two approaches for performing single-path VDC allocation with bandwidth guarantees, using the general-purpose SMT solver Z3 [49]. Like almost all SMT solvers, Z3 has no built-in support for network flow predicates. Therefore, in order to use Z3 for VDC allocation, the global bandwidth

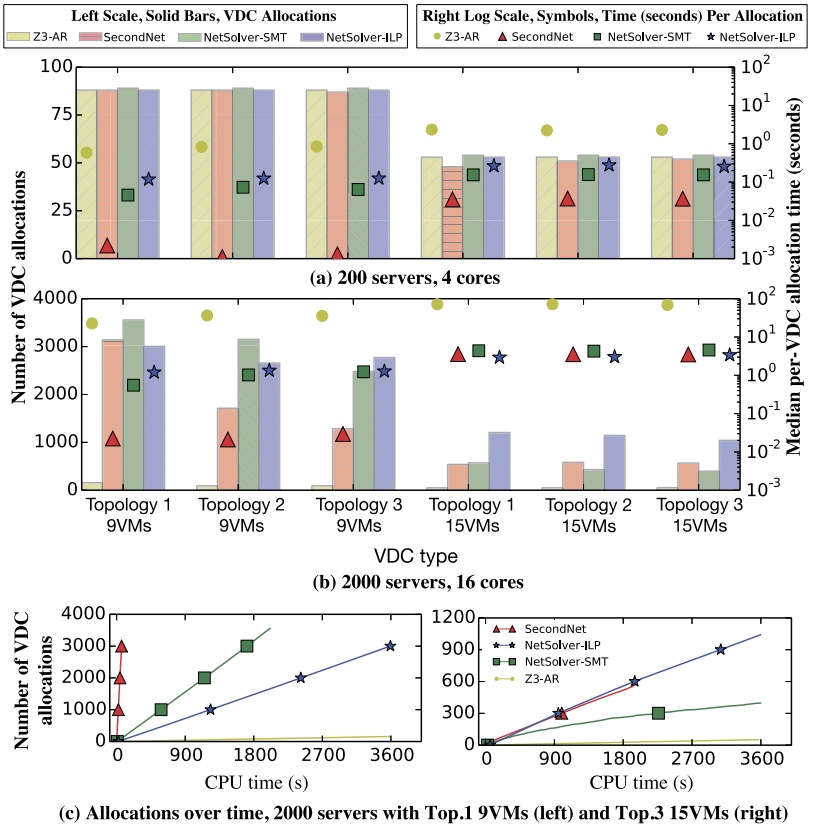


Fig. 6. Total number of consecutive VDCs allocated by different algorithms on various tree topologies from [21]. Above, we report the median running time for allocating individual VDCs; below, we report allocations over time for two selected instances. SecondNet is in some cases an order of magnitude faster than NetSolver, and both are consistently much faster than Z3. In many cases, NetSolver makes substantially more allocations than SecondNet, with NetSolver-ILP outperforming NetSolver-SMT on larger instances. (All figures are in color in the web version of this article.)

and connectivity constraints have to be expressed using a lower-level logical formulation. The first such encoding presented by [21] (which we call Z3-generic) can handle any data center topology but scales extremely poorly [21]. The second approach (which we call Z3-AR) makes use of an optimized abstraction-refinement technique; while substantially more scalable than the generic encoding, it is restricted to data centers with tree topologies. In preliminary experiments (not reported here), we confirmed that Z3-generic performed poorly, often failing to find any allocations within a 1-hour timeout on the benchmarks used in our experiments.

5.1. Comparison on trees from [21]

Our first experiment reproduces and extends an experiment from [21], in which a series of identical VDCs is allocated one-by-one to tree-structured data centers, until the solver is unable to make further allocations (or a timeout of 1 CPU hour is reached). In this experiment, there are 6 VDC instances considered: three consisting of 9 VMs each, and three consisting of 15 VMs each. Each VDC has a unique, randomly generated topology.[8] We obtained the original implementation of Z3-AR from the authors for this experiment, along with a version of SecondNet they implemented with support for the tree-structured data centers considered here. In this experiment, the VDCs always have identical structure; this is a restriction introduced here for compatibility with the solvers from [21]. This restriction makes the experiment less representative of real-world use cases, and it also allows all three of the constraint based approaches (Z3-AR, NetSolver-SMT, and NetSolver-ILP) to avoid substantial costs that would otherwise be incurred to support changing VDC topologies. In our subsequent experiments, below, we will consider cases where VDC topologies are non-constant.

Fig. 6 summarizes our results, showing the total number of consecutive VDCs allocated within 1 CPU hour. In Fig. 6a, we used the 200-server/4-cores-per-server physical data center from [21]. In Fig. 6b, we considered a larger data center with

[8] Note that here and in the remainder of this paper, we allocate individual VDCs one at a time, without looking ahead at the remaining VDCs that have yet to be allocated. This online allocation process can potentially result in a sub-optimal total number of allocations, even though our approach is complete for individual VDC allocations. Our approach is similar in this respect to the previous works [21,9] that we compare to.

2000 16-core servers. Fig. 6c shows the allocations made over time by each approach, for two representative VDCs for the 2000 server, 16-core case.

We note that, although all three solvers perform similarly on the small tree-structured data centers (with SecondNet, being heuristic and incomplete, faster than NETSOLVER, and NETSOLVER faster than Z3-AR), on the larger data center, NETSOLVER-ILP greatly outperforms SecondNet and Z3-AR, often allocating two or even three times as many VDCs on the same infrastructure. In most cases, NETSOLVER-ILP performs better than NETSOLVER-SMT, but both versions of NETSOLVER scale to thousands of servers, with median per-instance allocation times of a few seconds or less per VDC. On instances with smaller VDCs, NETSOLVER-SMT tends to have both faster runtimes and more allocations than NETSOLVER-ILP, while on instances with larger VDCs, NETSOLVER-ILP performs substantially better than NETSOLVER-SMT, sometimes achieving more than double the allocations of NETSOLVER-SMT.

5.2. Comparison on FatTree and BCube from [9]

The second experiment we conducted is a direct comparison against the original SecondNet implementation (which we also used for all comparisons reported later). Note that the implementation of Z3-generic, and both the theory and implementation of Z3-AR, are restricted to tree topologies, so they could not be included in these experiments.

The SecondNet benchmark instances are extremely large – in one case exceeding 100 000 servers – but also extremely easy to allocate: the available bandwidth per link is typically $\geq 50\times$ the requested communication bandwidths in the VDC, so with only 16 cores per server, the bandwidth constraints are mostly irrelevant. For such easy allocations, the fast, incomplete approach that SecondNet uses is the better solution. Accordingly, we scaled the SecondNet instances down to 432–1024 servers, a realistic size for many real-world data centers. For these experiments, we generated sets of 10 VDCs each of several sizes (6, 9, 12 and 15 VMs), following the methodology described in [21]. These VDCs have proportionally greater bandwidth requirements than those originally considered by SecondNet, requiring 5–10% of the smallest link-level capacities. The resulting VDC instances are large enough to be representative of many real-world use cases, while also exhibiting non-trivial bandwidth constraints. For each of these sets of VDCs, we then repeatedly allocated instances (in random order) until the data center is saturated.[9]

Like most SAT-solvers, MONOSAT (and consequently, the SMT version of NETSOLVER) exposes a large number of parameters in the form of command line options that can affect performance, often in non-obvious ways. Previous research [50] has shown that for many important applications, choosing good parameter settings can have a substantial impact on performance. For this experiment, we used SMAC, a prominent, state-of-the-art general-purpose algorithm configurator [11], to automatically search for a good configuration of NETSOLVER, resulting in a 14% decrease in average running time on these instances. We also used this configured version of NETSOLVER for the experiments on commercial instances shown in Fig. 8, where it resulted in a 17% decrease in running time. Our training set for configuration consisted of a generated set of physical (BCube and FatTree) and virtual data centers, each differing in size or topology from the ones used for evaluating NETSOLVER in this section. Configuration was performed over a combined total of 1680 CPU hours. We provide further details on how we used SMAC in Appendix A.

Fig. 7 shows the total number of allocations made by SecondNet and NETSOLVER on two data centers: (a) FatTree topology with 432 servers, and (b) BCube topology with 512 servers. For each data center, we also show the median CPU time required to allocate each VDC, in seconds. Further results can be found in the Appendix B, Table 8.[10]

5.3. Comparison on commercial networks

The above comparisons consider how NETSOLVER compares to existing VDC allocation tools on several artificial (but representative) network topologies from the VDC literature. To address whether there are real-world VDC applications where NETSOLVER performs not only better than existing tools, but is also fast enough to be used in practice, we also considered a deployment of a standard Hadoop virtual cluster, on a set of actual data center topologies (see Fig. 8). We collaborated with the private cloud provider ZeroStack Inc. to devise an optimal virtual Hadoop cluster to run Terasort.[11] Each Hadoop virtual network consists of a single master VM connected to 3–11 slave VMs.[12] We considered 5 different VM sizes, ranging from 1 CPU and 1 GB RAM, to 8 CPUs and 16 GB of RAM; for our experiments, the slave VMs were selected at random from this set, with the master VM also randomized but always at least as large as the largest slave VM. The Hadoop master has tree connectivity with all slaves, with either 1 or 2 Gbps links connecting the master to each slave (as Fig. 1 VDC).

[9] Our intent in this experiment is to simulate an unpredictable online workload, in which allocation requests must be processed in the order they arrive. An alternative that we have not yet explored would be to batch allocation requests, allowing one to re-order them heuristically, at the cost of increased scheduling latency.

[10] In these experiments, all solvers are restricted to a single CPU core. However, as Gurobi supports parallel execution, we also tried running this experiment with Gurobi's multi-threaded support enabled, using up to 8 CPU cores. We found that the results were similar to those for single-threaded execution (and in particular, neither consistently better nor worse), so we report only the latter.

[11] The Sort Benchmark Committee: http://sortbenchmark.org/.

[12] Many industrial VDCs have fewer than 15 VMs; e.g., [51] states that 80% of Bing services use fewer than 10 VMs. NETSOLVER performs well with up to 30 VMs.

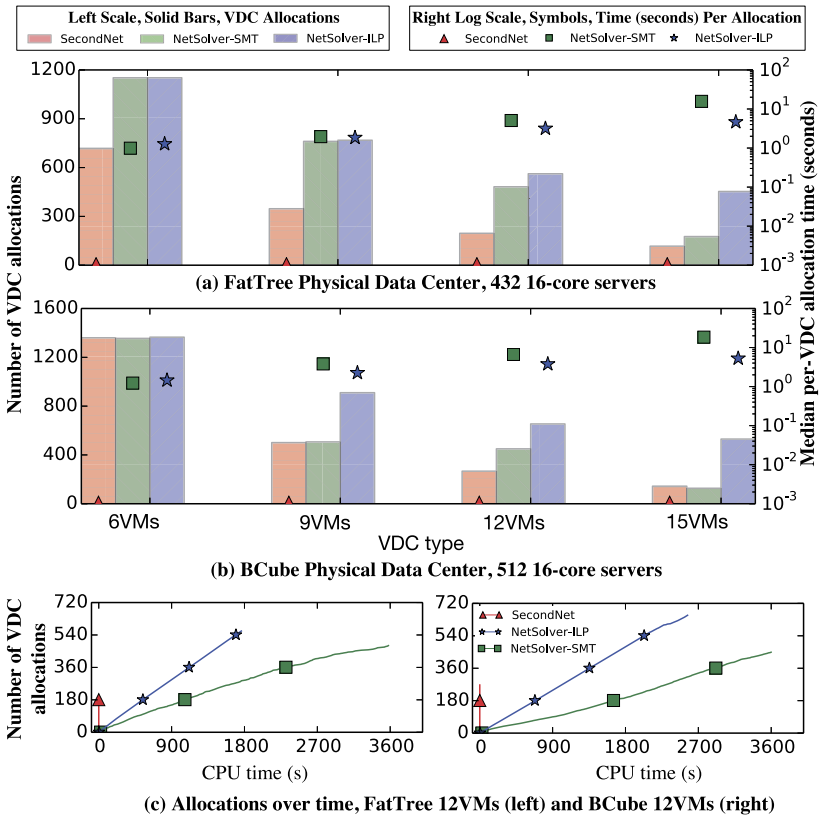


Fig. 7. Total number of consecutive VDCs allocated by different algorithms and time required per allocation on FatTree and BCube topologies from [9]. Above, we report the median running time for allocating individual VDCs; below, we report allocations over time for two selected instances. See Table 8 for further results.

The physical data center topology was provided by another company, which requested to remain anonymous. This company uses a private cloud deployed across four data centers in two geographic availability zones (AZs): *us-west* and *us-middle*. Each data center contains between 280 and 1200 servers, spread across 1 to 4 clusters with 14 and 40 racks. Each server has been scaled down to 8 cores, 16 GB RAM, 20 Gbps network bandwidth (via two 10 Gbps links). The network in each data center has a leaf-spine topology, where all ToR switches connect to two distinct *aggregation switches* over 40 Gbps links each (a total of 2 links with 80 Gbps; one on each aggregation switch) and aggregation switches are interconnected with four 40 Gbps links each. For each cluster, there is a *gateway switch* with a 240 Gbps link connected to each aggregation switch. All data centers use equal-cost multi-path (ECMP) to take advantage of multiple paths.

A VDC is allocated inside one AZ: VMs in one VDC can be split across two clusters in an AZ, but not across two AZs. Fig. 8 summarizes VDC allocation results per AZ; complete results for each AZ can be found in the Appendix, Tables 12–16.[13] More generally, executing NetSolver on distinct physical network units, such as an AZ, improves its scalability. This also works well in practice, as modern data centers are modular by design. For example, one of the largest data center operators in the world, Facebook, designed its Altoona data center with over 100 000 servers using *pods*, with each pod containing fewer than 1000 servers [52].

We applied SecondNet and NetSolver in this setting, consecutively allocating Hadoop VDCs of several sizes, ranging from 4 to 12 VMs, until no further allocations could be made. Note that, in addition to using a realistic physical topology, the CPU/memory, bandwidth values, and the VDCs being allocated are all real-world VDCs derived from real Hadoop jobs. By contrast, previous experiments used artificial VDCs from the Z3-AR paper [21]. Again, we could not run Z3-AR in this setting, as it is restricted to tree-topology data centers.

In Fig. 8, we show the results for the largest of these data centers (results for the smaller DCs were similar). As observed in our previous experiments, although SecondNet was much faster than either version of NetSolver, NetSolver's per-instance allocation time was typically just a few seconds, which is reasonable for long-running applications, such as the Hadoop jobs considered here. Again, NetSolver was able to allocate many more VDCs than SecondNet (here, 1.5–2 times

[13] NetSolver is not limited to allocating to a single AZ and can support multi-AZ allocation, assuming it is aware of the capacity of each AZ, including inter-AZ network bandwidth.

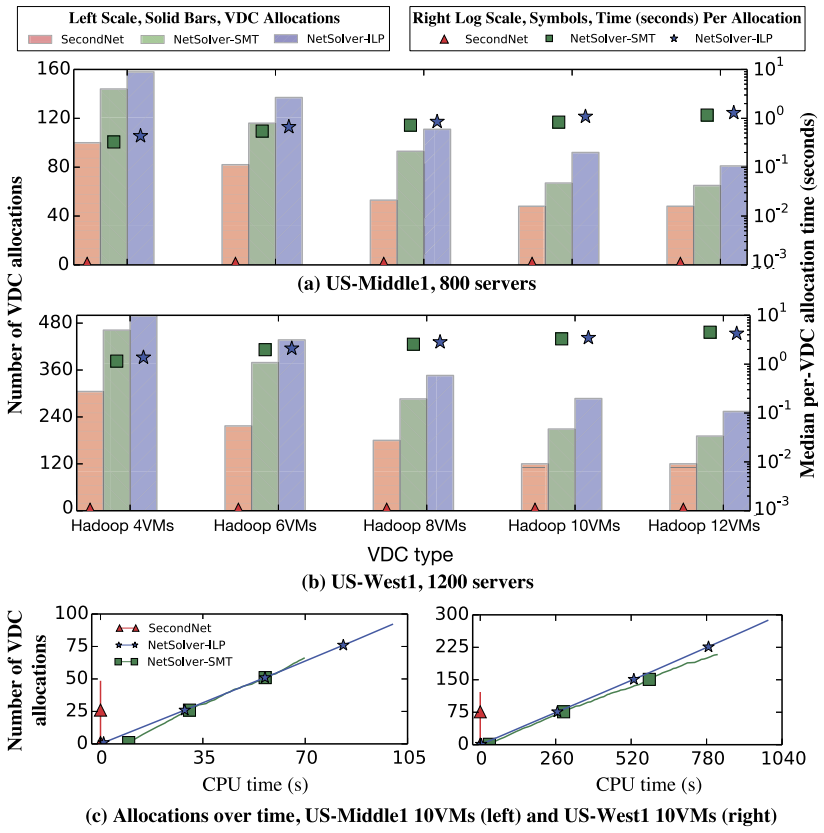


Fig. 8. Total number of consecutive VDCs allocated by different algorithms and time required per allocation on commercial data center topologies. Above, we report median running times for allocating individual VDCs; below, we show allocations over time for two selected instances. See Tables 12–16 for further results.

as many), across a range of DC and VDC sizes, including a commercial DC with more than 1000 servers. Moreover, with increasing virtual network size, NetSolver was able to allocate many more virtual machines, while respecting end-to-end bandwidth constraints. Often NetSolver allocated several times as many VDCs as SecondNet, and in extreme cases, it found hundreds of allocations, while SecondNet was unable to make any allocations (not shown for brevity). Similarly, keeping the virtual network the same size, but doubling the bandwidth requirements of each virtual machine greatly decreased the number of allocations made by SecondNet, while NetSolver showed considerably more robust performance in these more congested settings.

Overall, NetSolver was not only able to find many more allocations than SecondNet in this realistic setting, but NetSolver's median allocation time, 1–30 CPU seconds, shows that it can be practically useful in a real, commercial setting, for data centers and VDCs of this size. This provides strong evidence that NetSolver can find practical use in realistic settings where large or bandwidth-hungry VDCs need to be allocated. It also demonstrates the practical advantage of a (fast) complete algorithm like NetSolver over a much faster but incomplete algorithm like SecondNet: for bandwidth-heavy VDCs, even with arbitrary running time, SecondNet's VDCAlloc was unable to find the majority of the feasible allocations.

The experiments on BCube, FatTree, and the commercial networks reinforce our observations from the earlier experiments with artificial tree topologies: both versions of NetSolver improve greatly on state-of-the-art VDC allocation as compared to SecondNet or Z3. Further, the ILP version of NetSolver generally out-performs the SMT version, consistently finding 10% to 30% more allocations.

5.4. Comparison to virtual network embedding (VNE) approaches

In addition to the VDC allocation tools we considered above, we also compare to several state-of-the-art virtual network embedding tools, as implemented in the VNE testing framework ALEVIN [36]. We provide these comparisons mainly for reference, as the VNE tools we consider here were neither designed nor optimized for allocating to these large and sparely connected networks. As VNE algorithms are technically capable of performing VDC allocation, it is relevant to ask how they perform in this setting. However, it is also important to recognize that these experiments do not reflect how VNE algorithms might compare to VDC algorithms when applied to VNE instances. A more extensive discussion of the VNE literature is beyond the scope of this work, but we refer readers to [35] for a comprehensive survey.

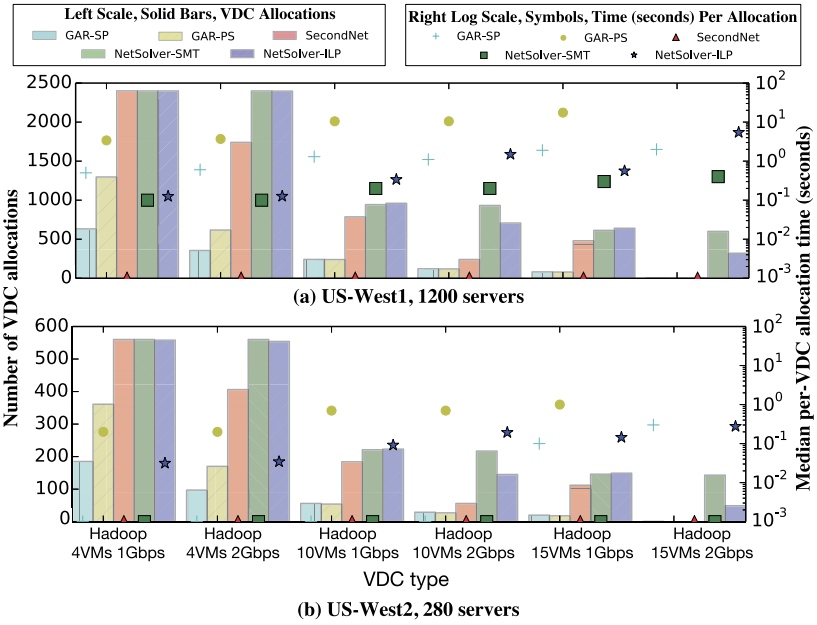


Fig. 9. Virtual Network Embedding (as implemented in the testing framework ALEVIN [36]) applied to VDC allocation. The VNE solvers perform poorly in this setting, achieving a small fraction of the allocations that NETSOLVER-SMT or NETSOLVER-ILP achieves, while also running an order of magnitude slower.

The VNE experimental framework we tested [36] uses a GUI, and so we employed a (significantly faster) 3.4 GHz Intel Core-i7-2600K processor with 32 GB of RAM for these VNE experiments.

In Fig. 9, we show two variants of the 'Greedy Allocation Resources' algorithm from [16]. The PS ('path-splitting') variant supports multi-path allocation, while the SP ('shortest-paths') variant does not. Both of these are greedy, incomplete, linear programming based algorithms, and are appropriate to consider as they are two of the fastest and simplest VNE algorithms from the literature. Unlike SecondNet, both of these algorithms do support allocating multiple VMs per server. We applied these algorithms to two of the largest (1200 servers) and smallest (280 servers) commercial topologies from the previous experiment, on the same VDC instances.[14] In Fig. 9, we can see that these two VNE algorithms perform significantly worse than both SecondNet's VDCAlloc and NETSOLVER, in many cases finding less than a quarter of the allocations of either tool, and, in the case of GAR-PS (the path splitting variant), taking more than 10 times as much time to perform these allocations.

We also tested several variants of three other families of state-of-the-art VNE algorithms from the ALEVIN framework: RW-MM-SP/PS [17], DViNE [18], and ASID [19]. Unfortunately, none of these were able find *any* allocations (within several hundred seconds). This strongly suggests that at least the VNE algorithms we evaluated are not sufficiently scalable for virtual data center allocation. Our findings here are consistent with those reported in [9].

5.5. Allocation robustness

In the above experiments, we showed that across many conditions, NETSOLVER was able to make many (often hundreds) more allocations than SecondNet or Z3-AR. One may wonder whether these additional allocations are the result of NETSOLVER having a better ability to solve challenging allocations quickly (completeness and efficiency), or if NETSOLVER is somehow making "smarter" allocations early on that leave more space for later VDC allocations.

In the experiments where Z3-AR makes many fewer allocations (Fig. 6b), Z3-AR's problem is excessively slow run times, allocating only a handful of VDCs in data centers with room for hundreds or thousands. In those cases, both NETSOLVER and SecondNet can make hundreds of further allocations starting from where Z3-AR was cut off.

The robustness question is more apropos versus SecondNet. We found conclusive evidence that good early allocations cannot be entirely responsible for NETSOLVER's performance, by observing that NETSOLVER can continue to allocate VDCs in cases where SecondNet can no longer make any further allocations. We repeated the experiments from Fig. 7 by first using SecondNet to allocate as many VDCs as it can into the data center. Then, starting from that already partially utilized data center, we used NETSOLVER to allocate further VDCs. The results of this experiment are shown in Fig. 10. Similarly to the earlier experiment, NETSOLVER can still allocate hundreds of additional VDCs starting from SecondNet's final allocation.

[14] Note that due to limitations in the ALEVIN platform, for these experiments, we consider just a single VDC instance of each size, rather than a set of such instances.

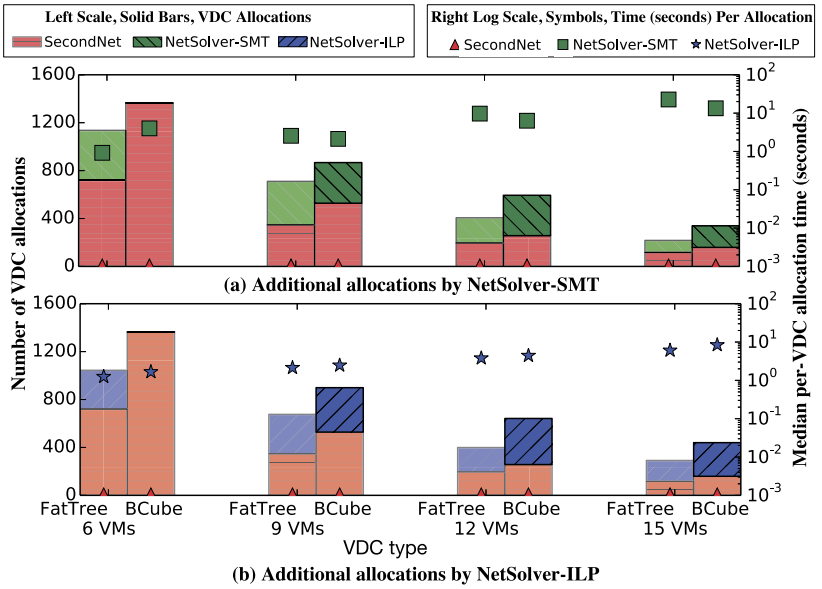


Fig. 10. Additional VDC allocations made by NetSolver-SMT (green) and NetSolver-ILP (blue), after SecondNet (red) has allocated its maximum number of VDCs. These experiments used the same VDCs and physical topologies as in Fig. 7. In many cases, NetSolver allocated hundreds of additional VDCs after SecondNet could not make further allocations.

The most interesting comparison is between NetSolver-ILP and NetSolver-SMT. In this case, both solvers are quite fast, and both solvers are complete in the sense that they will find an allocation if one exists. Therefore, in the cases where both solvers could find no more allocations, the additional allocations for NetSolver-ILP *must* be due to NetSolver-ILP somehow finding "smarter" allocations. In close examinations of the output of some of our experiments, we indeed found this to be the case, with NetSolver-ILP packing early allocations more tightly, thereby consuming less overall bandwidth with the early allocations, whereas NetSolver-SMT makes more spread-out allocations that consume more overall bandwidth.

For example, consider one of the largest examples from Fig. 8: 12 machine VDCs placed in the US-West 1 data center, with 1200 servers. Here, in the first 100 VDCs allocated by NetSolver-ILP, just 130 connections are to a top-of-rack switch, and just 71 connections pass between racks (for example, passing through gateway or aggregation switches).[15] As all VMs are placed on servers, and all servers are contained in racks, anytime that connected VMs in a VDC are placed on different servers, connections to the top-of-rack switch will be required. Similarly, anytime that VMs from a VDC are placed on multiple racks, connections between top-of-rack switches will be required. In contrast to NetSolver-ILP, NetSolver-SMT's first 100 allocations require 603 connections to the top-of-rack switch, and 449 connections between rack switches.

We hypothesize that this is due to the different approaches to incremental solving in ILP and SMT: an ILP solver will typically attempt to re-use a previous solution, whereas an SMT solver's main re-use strategy is to retain learned clauses. Therefore, during the early, highly unconstrained phase of the experiments, NetSolver-ILP will tend to allocate VDCs repeatedly onto the same machines, packing them in more tightly, whereas NetSolver-SMT spreads the allocations more arbitrarily around the data center. This suggests an obvious way to tune NetSolver-SMT heuristically, but more generally, it suggests a direction to explore for improving incremental SMT solving.

6. Extensions

Because NetSolver is built on constraint solving, we can easily extend it to enforce constraints beyond the ones used for the basic VDC allocation problem. Using such additional constraints, we can deal with advanced aspects of VDC allocation that occur in realistic data center management situations. Here, we consider three such extensions: *soft-affinity*, *server usage minimization*, and *no-hotspot* (see Fig. 11). These constraints go beyond what previous VDC allocation algorithms support and represent substantially more challenging allocation objectives than standard VDC allocation. Technically, they are soft constraints, and NetSolver is not guaranteed to find optimal solutions with respect to them, but instead heuristically maximizes the degree to which they are satisfied. We now describe each extension and, as before, evaluate the number of VDC allocations and median per-VDC allocation time for NetSolver with each type of constraint. Since comparable VDC allocation techniques lack these capabilities, we do not compare to existing techniques.

[15] Note that as each placement is for 12 VMs with multiple connections between them, there are many more total connections between VMs than there are VDCs allocated.

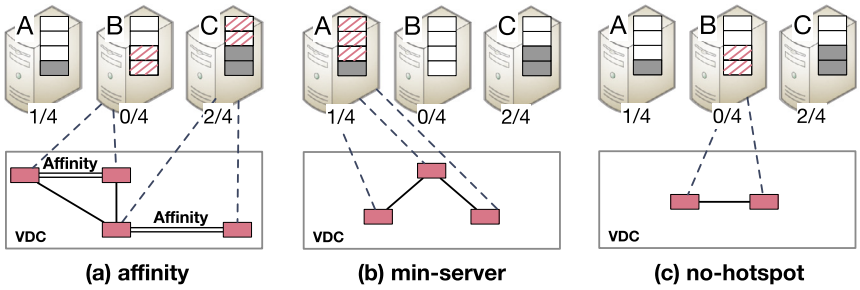


Fig. 11. Extensions to NETSOLVER. Three extensions are considered: affinity constraints between VMs (a); minimizing the number of utilized servers (b); and hot-spot minimization (c), which avoids placing VMs on highly utilized servers. Grey boxes in servers (top) indicate previously allocated VMs. Red boxes in VDC (bottom) indicate VMs to allocate. Red striped boxes indicate placements satisfying each constraint.

6.1. Affinity constraints

The first constraint we consider is an affinity constraint, consisting of a set of virtual machines in the VDC which should, if possible, be allocated to the same server (Fig. 11a). This has been used in practice to substantially improve data locality in the context of cluster scheduling [53,54]. While NETSOLVER provides good support for hard affinity and anti-affinity constraints (the virtual machines in the affinity set *must* or *must-not* be allocated on the same server), in many realistic settings, a soft affinity constraint may be more useful, so we consider the latter instead.

Formally, we consider one or more affinity sets A consisting of virtual machines $\{VM_1, VM_2, \ldots\}$. The goal is to preferentially place VMs in A on the same servers. For each affinity set A, and each server $s \in S$, we add a new Boolean literal (0/1-integer variable), A_s, which is true (1) if one of the VMs from A is assigned to server s, and false (0) if it contains no VMs from the affinity set, e.g.:

$$A_s = \max_{v \in A} A_{v,s}$$

(Recall from Section 4.1 that $A_{v,s}$ is a 0/1-integer variable indicating that VM v has been assigned to server s.) or

$$A_s = \bigvee_{v \in A} e_{v,s}$$

(Recall from Section 4.2.2 that $e_{v,s}$ is a Boolean literal indicating that VM v has been assigned to server s.). We then use MONOSAT[16] and Gurobi's built-in optimization support to minimize the number of servers that are allocated for the affinity set (this number must be at least 1, if the set is non-empty):

$$minimize(\sum_{s \in S} A_s).$$

In the case where there are multiple affinity constraints, we assume that they are provided in order of importance and enforce them lexicographically.

The affinity constraints for the VDCs in our evaluation try to maximize the number of slave VMs that are co-located with the master VM (thus reducing network traffic by keeping the communication local to the server). Such a constraint is useful for our sample Terasort workload, as slaves can take advantage of data locality with the master, i.e., they can locally fetch generated raw data (to be sorted) and complete the Terasort workload faster.

6.2. Server usage minimization

Our second optimization constraint minimizes the total number of utilized servers (Fig. 11b). A server that has no virtual machines allocated to it can be put into a low-power mode to save energy.

As in our experiments in Section 5, we assume a setting in which many VDCs will be allocated in consecutively (and sometimes deallocated). As we are considering repeated VDC allocation, at the beginning of each new allocation, each server may either already have some VMs assigned to it, or not. We want to preferentially allocate VMs to servers that already host VMs (from previous allocations), and minimize the number of VMs allocated to previously unused servers. This soft constraint can be enforced and optimized in much the same way as the (soft) affinity constraint, but has different semantics.

Formally, we introduce, for each server $s \in S$, two Boolean literals: U_s and P_s. The variable U_s is true iff server s is used in the current allocation, i.e., it has at least one VM allocated to it, e.g.,

[16] MONOSAT internally implements optimization as a sequence of increasingly constrained decision problems, forming a binary search over bitvector or Boolean values.

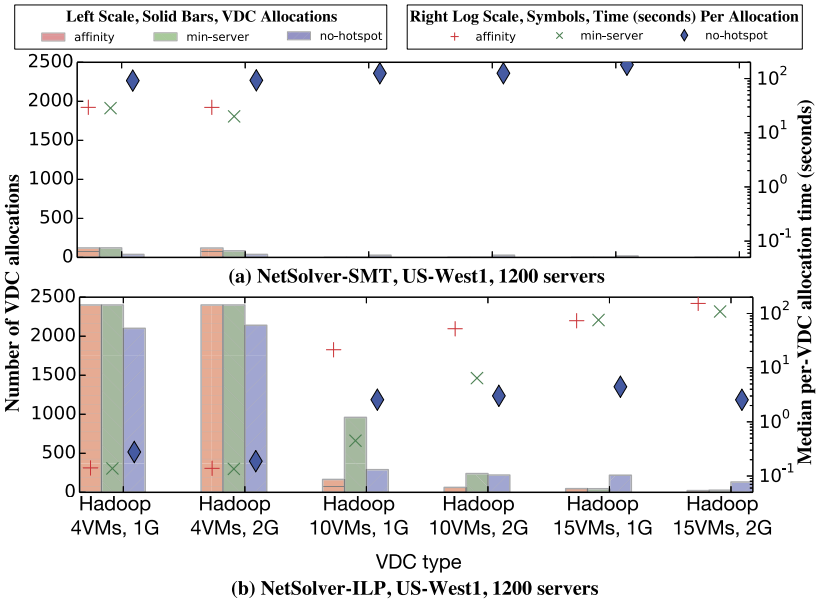


Fig. 12. Allocating Hadoop instances with the additional optimization constraints described in Fig. 11. These optimization constraints are significantly more expensive to solve than the standard NETSOLVER constraints. We can see also that the ILP version of NETSOLVER generally performs substantially better than the SMT version.

$$U_s = \max_{v \in VM} A_{v,s}$$

for ILP, or

$$U_s = \bigvee_{v \in VM} e_{v,s}$$

for SMT. The variable P_s is true iff s already had at least one VM previously allocated to it before the current VDC allocation. Then, we ask the solver to minimize the number of VMs used in this allocation that were not previously used:

$$minimize(\sum_{s \in S}(U_s \wedge \neg P_s))$$

In order to set P_s for each server, we rely on some additional record keeping (outside the constraints). Before solving, we add the constraint $(\neg P_s)$ for each server s that previously had no VMs allocated to it, and the constraint (P_s) for each server s that previously had at least one VM allocated to it.[17]

6.3. Hotspot minimization

The final constraint we consider is balanced VM placement across all servers (Fig. 11c). Such placement avoids *data center hotspots* where some servers consume much higher compute power as compared to others due to unevenly placed VMs. In personal communication, data center operators noted that no-hotspot placement improves overall reliability and reduces server failure rate. This is consistent with previous findings on data center operations [55].

In no-hotspot placement, NETSOLVER avoids, during each VDC allocation, placing VMs on servers that are already heavily utilized. Formally, during each VDC allocation round, we want to minimize the number of utilized cores in the most utilized server (among those servers that received allocations during this round). Note that this is not exactly the opposite of utilized server minimization, since that constraint does not distinguish between highly and slightly utilized servers:

$$minimize\Big(\max_{s \in S}(\sum_{v \in VM} e_{vs} \cdot cpu(v))\Big)$$

[17] One could also consider a slightly simpler and logically equivalent formulation, in which the total number of utilized servers is minimized, rather than minimizing only the servers in the current allocation. Restricting the encoding to only consider the servers in the current allocation allows to avoid introducing an unbounded number of (mostly constant) variables into the solver.

In Fig. 12, we show results for both versions of NETSOLVER extended with these three soft constraints, applied to the largest real-world data center topology (with 1200 servers), on the Hadoop VDC instances. In these experiments, NETSOLVER-ILP performed substantially better than NETSOLVER-SMT. For larger VDCs, NETSOLVER-SMT was unable to make any allocations at all for the server utilization constraints even in cases where NETSOLVER-ILP was able to find dozens or hundreds of allocations.[18] Overall, it is clear that applying any of these three constraints makes the allocation process significantly more expensive for NETSOLVER, and the costs of these extensions grow dramatically for larger VDCs. This is in contrast to our findings for the (unaugmented) VDC allocation problem, for which we found that NETSOLVER scales well to data centers with thousands of servers, and for VDCs with as many as 30 VMs.

These constraints are important to many data center operators, and, to the best of our knowledge, NETSOLVER is the first VDC allocation tool (with end-to-end bandwidth guarantees) to support them.

7. Conclusions

We introduced a new, constraint-based VDC allocation method, NETSOLVER, for multi-path VDC allocation with end-to-end bandwidth guarantees. Our approach differs from previous constraint-based approaches by making use of efficient network flow encodings in the underlying constraint solvers. NETSOLVER scales well to data centers with 1000 or more servers, while substantially improving data center utilization as compared to current methods. Notably, we have demonstrated that in several realistic settings, NETSOLVER allocates *3 times* as many virtual data centers as previous approaches, with a runtime that is fast enough for practical use. We found that in most cases, the ILP NETSOLVER backend outperforms the SMT NETSOLVER backend, often achieving 10% to 20% more allocations while requiring half the time or less per allocation. In some cases, the ILP backend achieved two or more times as many allocations as the SMT backend. We also found that both versions of NETSOLVER greatly outperform the other approaches we compared to.

NETSOLVER overcomes major limitations of current state-of-the-art approaches for VDC allocation with hard bandwidth guarantees: Unlike SecondNet, our approach is complete and, as a result, is able to continue making allocations in bandwidth-constrained networks; unlike the abstraction-refinement techniques from [21], NETSOLVER supports arbitrary data center topologies (as well as being much faster). Our constraint-based approach represents the first complete VDC allocation algorithm supporting multi-path bandwidth allocation for arbitrary network topologies – an important capability in modern data centers.

Finally, as NETSOLVER is built on-top of constraint solvers, it is easily extensible. We have demonstrated that it can handle additional VDC allocation constraints not supported by other approaches, yet relevant in practice, such as maximization of data locality with VM affinities, minimization of the total number of utilized servers, and load balancing with hotspot avoidance. While these constraints make VDC allocation substantially more challenging, our approach can still efficiently allocate VDCs with up to 15 VMs in realistic scenarios.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

We thank the authors of [9] and [21] for making their implementations available to us, and especially Yifei Yuan for his extensive assistance. This research was funded by NSERC discovery grants to Ivan Beschastnikh, Holger H. Hoos and Alan J. Hu. Nodir Kodirov is supported by a UBC four-year doctoral fellowship (4YF).

Appendix A. Automated algorithm configuration

As described in Section 5, we used automatic algorithm configuration to optimize NETSOLVER-SMT's performance for the experiments reported in Figs. 7 and 8. Here we provide details on the configuration process.

NETSOLVER-SMT has a large set of parameters that affect its performance, including 23 Boolean, 1 integer-valued and 2 real-valued parameters. Most of these parameters are exposed by the underlying SAT solver, where they control settings such as the frequency of restarts in the solver. Other settings control the way the problem instance is translated into an SMT formula in NETSOLVER-SMT, for example determining how Pseudo-Boolean constraints are encoded. These settings can significantly impact NETSOLVER-SMT's running time.

To find a good configuration of these settings, we used SMAC, a well-known, state-of-the-art automatic algorithm configuration procedure [11]. Given an algorithm, a space of parameter configurations (induced by exposed parameters and their permissible values), as well as a set of training instances, SMAC searches among the space of configurations for one that performs well across the training instances. For our training instances, we produced a set of 80 BCube and FatTree data

[18] Interestingly, NETSOLVER-SMT using the earlier MonoSAT version 1.4 performed better on this experiment than using the current MonoSAT 1.6. Although MonoSAT 1.6 has better overall performance, the changes evidently hurt this experiment, highlighting the challenge of balancing different optimizations.

center instances (varying the numbers of servers in each), as well as a set of 100 randomly generated VDC instances with 9–15 VMs. None of these instances are otherwise used in the experiments reported in this article. Mirroring our experimental setup described in Section 5, in each invocation of NETSOLVER-SMT by SMAC, NETSOLVER-SMT repeatedly allocates VDCs from that set of 100 (in random order) to one of the 80 data center topologies until the data center is saturated (or until a cut-off time of 10 000 seconds is reached). We then used SMAC to minimize the average runtime (treating timeouts as taking 10 times the cut-off time).

We performed 10 separate, independent runs of SMAC with different random seeds, each with a budget of 7 CPU days (for a total of 1680 CPU hours), resulting in 10 optimized parameter configurations. We then validated each of these configurations on a disjoint set of 80 data centers and a disjoint set of 100 VDCs (generated in the same way as the previously described training instances and not containing any of the instances used in our later experiments), selecting from those 10 configurations the one with the best performance on this validation set. This configuration of NETSOLVER-SMT was then used for the experimental results described in Section 5, Figs. 7 and 8, as well as Tables 8 and 12. We found that, compared to running NETSOLVER-SMT with its default parameters, the configured version was able to solve the instances in Table 8 14% faster on average, and was able to solve the instances in Table 12 17% faster on average.

Appendix B. Additional data on experimental results (Tables 2–16)

Table 2
Experiments on data centers with tree topologies from [21]. This table shows the total number of VDC allocations (#VDC) and the total CPU time in seconds required by each algorithm to allocate those VDCs. Additionally, we show the minimum (Min), Average (Avg), median (Mdn) and Maximum (Max) CPU time in seconds per VDC allocation. Each algorithm was limited to 3600 seconds of CPU time; overtime data points are listed as >3600 s. The full experimental setup is described in Section 5. Results continue on following pages.

	Topology 1, with 9 VMs					
	#VDC	Total (s)	Min (s)	Avg (s)	Mdn (s)	Max (s)
Physical Data Center of 200 servers with 4 cores each						
Z3-AR	88	51.436	0.364	0.584	0.587	0.846
SecondNet	88	0.226	<0.001	0.003	0.002	0.013
NETSOLVER-SMT	89	4.921	0.042	0.055	0.046	0.810
NETSOLVER-ILP	88	12.204	0.097	0.114	0.118	0.123
Physical Data Center of 200 servers with 16 cores each						
Z3-AR	355	215.457	0.359	0.607	0.617	0.874
SecondNet	313	0.764	0.002	0.002	0.003	0.003
NETSOLVER-SMT	356	16.045	0.040	0.045	0.042	0.814
NETSOLVER-ILP	355	44.068	0.095	0.118	0.120	0.126
Physical Data Center of 400 servers with 16 cores each						
Z3-AR	711	881.982	0.678	1.240	1.226	7.787
SecondNet	628	3.213	0.003	0.005	0.005	0.007
NETSOLVER-SMT	712	72.117	0.093	0.101	0.098	1.649
NETSOLVER-ILP	711	176.631	0.192	0.242	0.245	0.256
Physical Data Center of 2000 servers with 16 cores each						
Z3-AR	158	3586.688	19.224	22.701	22.685	23.662
SecondNet	3140	68.015	0.013	0.022	0.022	0.031
NETSOLVER-SMT	3556	2012.584	0.512	0.566	0.551	8.751
NETSOLVER-ILP	3005	3598.621	0.934	1.190	1.196	1.259

Table 3
Continuation of Table 2.

	Topology 2, with 9 VMs					
	#VDC	Total (s)	Min (s)	Avg (s)	Mdn (s)	Max (s)
Physical Data Center of 200 servers with 4 cores each						
Z3-AR	88	72.421	0.458	0.823	0.828	1.217
SecondNet	88	0.093	<0.001	<0.001	<0.001	0.005
NETSOLVER-SMT	89	7.848	0.070	0.088	0.073	1.292
NETSOLVER-ILP	88	12.994	0.100	0.122	0.125	0.132
Physical Data Center of 200 servers with 16 cores each						
Z3-AR	281	3597.849	1.124	12.804	2.413	363.489
SecondNet	171	0.470	<0.001	0.003	0.003	0.003
NETSOLVER-SMT	352	26.839	0.066	0.076	0.072	1.277
NETSOLVER-ILP	354	44.096	0.095	0.118	0.120	0.131

(*continued on next page*)

Table 3 (*continued*)

	Topology 2, with 9 VMs				
	#VDC	Total (s)	Min (s)	Avg (s)	Mdn (s)	Max (s)
Physical Data Center of 400 servers with 16 cores each						
Z3-AR	89	3543.388	2.752	39.813	3.032	401.203
SecondNet	342	1.635	0.003	0.005	0.005	0.007
Netsolver-SMT	705	113.263	0.141	0.161	0.155	2.528
Netsolver-ILP	711	181.286	0.203	0.249	0.251	0.268
Physical Data Center of 2000 servers with 16 cores each						
Z3-AR	92	3561.564	31.964	38.713	36.456	120.569
SecondNet	1712	35.664	0.012	0.021	0.021	0.031
Netsolver-SMT	3150	3598.026	0.770	1.142	1.025	13.597
Netsolver-ILP	2656	3598.639	1.299	1.346	1.336	1.414

Table 4
Continuation of Table 2.

	Topology 3, with 9 VMs				
	#VDC	Total (s)	Min (s)	Avg (s)	Mdn (s)	Max (s)
Physical Data Center of 200 servers with 4 cores each						
Z3-AR	88	75.205	0.491	0.855	0.845	1.212
SecondNet	87	0.174	<0.001	0.002	<0.001	0.010
Netsolver-SMT	89	7.025	0.060	0.079	0.064	1.068
Netsolver-ILP	88	12.978	0.102	0.123	0.125	0.132
Physical Data Center of 200 servers with 16 cores each						
Z3-AR	61	339.435	1.127	5.565	1.278	62.278
SecondNet	129	0.509	0.002	0.004	0.004	0.006
Netsolver-SMT	350	22.921	0.053	0.065	0.061	1.071
Netsolver-ILP	355	46.795	0.101	0.126	0.126	0.134
Physical Data Center of 400 servers with 16 cores each						
Z3-AR	86	3481.516	2.689	40.483	3.023	490.405
SecondNet	257	1.942	0.003	0.008	0.008	0.011
Netsolver-SMT	684	104.888	0.116	0.153	0.144	2.076
Netsolver-ILP	711	183.422	0.206	0.252	0.253	0.264
Physical Data Center of 2000 servers with 16 cores each						
Z3-AR	94	3554.965	31.788	37.819	35.689	96.954
SecondNet	1286	37.396	0.013	0.029	0.029	0.046
Netsolver-SMT	2480	3598.481	0.610	1.451	1.236	11.014
Netsolver-ILP	2772	3599.038	1.219	1.290	1.277	1.410

Table 5
Continuation of Table 2.

	Topology 1, with 15 VMs				
	#VDC	Total (s)	Min (s)	Avg (s)	Mdn (s)	Max (s)
Physical Data Center of 200 servers with 4 cores each						
Z3-AR	53	130.090	1.023	2.455	2.347	4.595
SecondNet	48	1.472	0.006	0.031	0.035	0.053
Netsolver-SMT	54	10.950	0.142	0.203	0.154	2.320
Netsolver-ILP	53	19.274	0.237	0.261	0.263	0.278
Physical Data Center of 200 servers with 16 cores each						
Z3-AR	28	481.402	3.243	17.193	3.754	246.589
SecondNet	56	1.893	0.009	0.034	0.037	0.042
Netsolver-SMT	165	43.675	0.132	0.265	0.214	2.314
Netsolver-ILP	205	62.503	0.223	0.278	0.277	0.453
Physical Data Center of 400 servers with 16 cores each						
Z3-AR	62	2948.420	7.209	47.555	7.768	1562.491
SecondNet	109	13.356	0.016	0.123	0.130	0.155
Netsolver-SMT	312	253.934	0.271	0.814	0.556	5.750
Netsolver-ILP	422	250.751	0.446	0.568	0.565	1.267
Physical Data Center of 2000 servers with 16 cores each						
Z3-AR	50	3569.764	65.582	71.395	71.686	73.675
SecondNet	539	1931.105	0.070	3.583	3.586	4.293
Netsolver-SMT	571	3596.354	1.394	6.298	4.383	49.902
Netsolver-ILP	1205	3597.799	2.831	2.940	2.942	3.049

Table 6
Continuation of Table 2.

	Topology 2, with 15 VMs					
	#VDC	Total (s)	Min (s)	Avg (s)	Mdn (s)	Max (s)
Physical Data Center of 200 servers with 4 cores each						
Z3-AR	53	122.701	0.990	2.315	2.236	3.591
SecondNet	51	1.567	0.006	0.031	0.037	0.049
Netsolver-SMT	54	10.873	0.147	0.201	0.156	2.374
Netsolver-ILP	53	20.023	0.249	0.273	0.276	0.290
Physical Data Center of 200 servers with 16 cores each						
Z3-AR	39	3573.899	3.342	91.638	7.160	1594.074
SecondNet	59	2.092	0.009	0.035	0.038	0.042
Netsolver-SMT	153	66.725	0.141	0.436	0.223	3.026
Netsolver-ILP	210	58.558	0.217	0.253	0.255	0.270
Physical Data Center of 400 servers with 16 cores each						
Z3-AR	27	2886.366	7.371	106.902	7.835	1586.927
SecondNet	117	15.524	0.014	0.133	0.138	0.153
Netsolver-SMT	283	637.132	0.289	2.251	0.693	51.426
Netsolver-ILP	414	242.050	0.520	0.559	0.561	0.613
Physical Data Center of 2000 servers with 16 cores each						
Z3-AR	50	3574.869	65.324	71.497	71.893	75.121
SecondNet	582	2052.790	0.071	3.527	3.550	4.052
Netsolver-SMT	430	3588.706	1.577	8.346	4.279	90.796
Netsolver-ILP	1144	3597.381	2.923	3.096	3.029	3.342

Table 7
Continuation of Table 2.

	Topology 3, with 15 VMs					
	#VDC	Total (s)	Min (s)	Avg (s)	Mdn (s)	Max (s)
Physical Data Center of 200 servers with 4 cores each						
Z3-AR	53	121.831	0.997	2.299	2.286	3.495
SecondNet	52	1.594	0.005	0.031	0.037	0.053
Netsolver-SMT	54	10.857	0.142	0.201	0.154	2.240
Netsolver-ILP	53	18.794	0.219	0.252	0.254	0.268
Physical Data Center of 200 servers with 16 cores each						
Z3-AR	20	3205.922	3.278	160.296	3.532	3128.527
SecondNet	57	1.910	0.007	0.034	0.038	0.039
Netsolver-SMT	132	58.942	0.133	0.447	0.229	2.416
Netsolver-ILP	213	62.049	0.217	0.266	0.268	0.281
Physical Data Center of 400 servers with 16 cores each						
Z3-AR	31	1443.791	8.095	46.574	8.456	748.569
SecondNet	114	13.522	0.014	0.119	0.129	0.134
Netsolver-SMT	264	426.637	0.271	1.616	0.758	14.437
Netsolver-ILP	310	194.811	0.440	0.594	0.559	0.868
Physical Data Center of 2000 servers with 16 cores each						
Z3-AR	52	3566.684	64.684	68.590	68.715	87.359
SecondNet	567	1953.754	0.074	3.446	3.509	4.112
Netsolver-SMT	398	3595.583	1.418	9.034	4.576	92.755
Netsolver-ILP	1041	3595.818	2.295	3.400	3.383	3.900

Table 8
Experiments on FatTree and BCube data center topologies from [9]. This table shows the total number of VDC allocations (#VDC) and the total CPU time in seconds required by each algorithm to allocate those VDCs. Additionally, we show the minimum (Min), Average (Avg), median (Mdn) and Maximum (Max) CPU time in seconds per VDC allocation. The full experimental setup is described in Section 5. Results continue on following pages.

	VDCs with 6 VMs					
	#VDC	Total (s)	Min (s)	Avg (s)	Mdn (s)	Max (s)
FatTree Physical Data Center of 128 servers with 16 cores each						
SecondNet	209	0.012	<0.001	<0.001	<0.001	<0.001
Netsolver-SMT	342	96.260	0.189	0.281	0.263	1.535
Netsolver-ILP	333	113.660	0.235	0.341	0.349	0.445
FatTree Physical Data Center of 432 servers with 16 cores each						
SecondNet	718	0.054	<0.001	<0.001	<0.001	<0.001
Netsolver-SMT	1152	1249.125	0.609	1.084	0.992	5.310
Netsolver-ILP	1152	1415.767	0.848	1.229	1.271	1.720

(*continued on next page*)

Table 8 (*continued*)

	VDCs with 6 VMs #VDC	Total (s)	Min (s)	Avg (s)	Mdn (s)	Max (s)
FatTree Physical Data Center of 1024 servers with 16 cores each						
SecondNet	1595	0.118	<0.001	<0.001	<0.001	<0.001
NETSOLVER-SMT	1251	3597.369	1.501	2.876	2.538	12.986
NETSOLVER-ILP	1274	3597.346	2.018	2.823	2.890	3.499
BCube Physical Data Center of 512 servers with 16 cores each						
SecondNet	1360	0.194	<0.001	<0.001	<0.001	<0.001
NETSOLVER-SMT	1355	2573.113	0.735	1.899	1.229	22.152
NETSOLVER-ILP	1365	1929.239	1.011	1.413	1.452	1.919
BCube Physical Data Center of 1000 servers with 16 cores each						
SecondNet	2660	0.511	<0.001	<0.001	<0.001	0.002
NETSOLVER-SMT	214	3571.800	1.475	16.691	4.206	135.497
NETSOLVER-ILP	1258	3597.378	2.008	2.859	2.968	3.451

Table 9
Continuation of Table 8, showing results for VDCs with 9 VMs.

	VDCs with 9 VMs #VDC	Total (s)	Min (s)	Avg (s)	Mdn (s)	Max (s)
FatTree Physical Data Center of 128 servers with 16 cores each						
SecondNet	114	0.009	<0.001	<0.001	<0.001	<0.001
NETSOLVER-SMT	226	114.566	0.266	0.507	0.478	2.518
NETSOLVER-ILP	215	118.594	0.352	0.551	0.566	0.746
FatTree Physical Data Center of 432 servers with 16 cores each						
SecondNet	347	0.034	<0.001	<0.001	<0.001	<0.001
NETSOLVER-SMT	761	1731.223	0.903	2.275	1.958	11.452
NETSOLVER-ILP	768	1393.752	1.152	1.815	1.847	2.417
FatTree Physical Data Center of 1024 servers with 16 cores each						
SecondNet	785	0.094	<0.001	<0.001	<0.001	<0.001
NETSOLVER-SMT	461	3597.070	2.286	7.803	6.073	62.231
NETSOLVER-ILP	779	3595.779	2.931	4.616	4.643	6.058
BCube Physical Data Center of 512 servers with 16 cores each						
SecondNet	501	0.059	<0.001	<0.001	<0.001	<0.001
NETSOLVER-SMT	507	3596.435	1.173	7.094	3.805	75.544
NETSOLVER-ILP	910	2000.569	1.477	2.198	2.249	3.508
BCube Physical Data Center of 1000 servers with 16 cores each						
SecondNet	1653	0.501	<0.001	<0.001	<0.001	0.003
NETSOLVER-SMT	46	3537.341	2.216	76.899	16.144	389.057
NETSOLVER-ILP	801	3596.705	2.826	4.490	4.581	5.896

Table 10
Continuation of Table 8, showing results for VDCs with 12 VMs.

	VDCs with 12 VMs #VDC	Total (s)	Min (s)	Avg (s)	Mdn (s)	Max (s)
FatTree Physical Data Center of 128 servers with 16 cores each						
SecondNet	60	0.007	<0.001	<0.001	<0.001	<0.001
NETSOLVER-SMT	150	222.579	0.392	1.484	1.163	5.355
NETSOLVER-ILP	148	135.489	0.693	0.915	0.920	1.163
FatTree Physical Data Center of 432 servers with 16 cores each						
SecondNet	196	0.027	<0.001	<0.001	<0.001	<0.001
NETSOLVER-SMT	482	3578.014	1.366	7.423	5.072	56.351
NETSOLVER-ILP	562	1761.317	2.312	3.134	3.172	6.418
FatTree Physical Data Center of 1024 servers with 16 cores each						
SecondNet	435	0.071	<0.001	<0.001	<0.001	<0.001
NETSOLVER-SMT	229	3569.541	3.206	15.588	11.930	74.653
NETSOLVER-ILP	485	3595.888	5.742	7.414	7.505	8.589
BCube Physical Data Center of 512 servers with 16 cores each						
SecondNet	267	0.048	<0.001	<0.001	<0.001	<0.001
NETSOLVER-SMT	450	3598.757	1.759	7.997	6.658	41.041
NETSOLVER-ILP	654	2567.049	3.099	3.925	3.778	31.076
BCube Physical Data Center of 1000 servers with 16 cores each						
SecondNet	466	0.095	<0.001	<0.001	<0.001	<0.001
NETSOLVER-SMT	61	3580.793	3.237	58.702	30.438	335.214
NETSOLVER-ILP	491	3595.093	6.042	7.322	7.389	8.332

Table 11
Continuation of Table 8, showing results for VDCs with 15 VMs.

	VDCs with 15 VMs					
	#VDC	Total (s)	Min (s)	Avg (s)	Mdn (s)	Max (s)
FatTree Physical Data Center of 128 servers with 16 cores each						
SecondNet	38	0.006	<0.001	<0.001	<0.001	<0.001
NETSOLVER-SMT	112	530.380	0.525	4.736	2.738	35.364
NETSOLVER-ILP	127	173.856	1.022	1.369	1.340	3.572
FatTree Physical Data Center of 432 servers with 16 cores each						
SecondNet	117	0.021	<0.001	<0.001	<0.001	<0.001
NETSOLVER-SMT	176	3596.489	2.001	20.435	15.689	83.388
NETSOLVER-ILP	453	2208.468	3.592	4.875	4.644	27.885
FatTree Physical Data Center of 1024 servers with 16 cores each						
SecondNet	262	0.057	<0.001	<0.001	<0.001	<0.001
NETSOLVER-SMT	81	3583.999	4.547	44.247	31.697	171.315
NETSOLVER-ILP	327	3598.403	7.867	11.004	10.751	13.091
BCube Physical Data Center of 512 servers with 16 cores each						
SecondNet	144	0.027	<0.001	<0.001	<0.001	<0.001
NETSOLVER-SMT	127	3574.325	3.347	28.144	18.470	296.543
NETSOLVER-ILP	530	3514.333	4.366	6.631	5.285	232.300
BCube Physical Data Center of 1000 servers with 16 cores each						
SecondNet	302	0.090	<0.001	<0.001	<0.001	0.002
NETSOLVER-SMT	46	3423.108	4.765	74.415	58.549	258.313
NETSOLVER-ILP	345	3589.471	8.069	10.404	10.169	12.470

Table 12
Experiments on commercial data center topologies. This table shows the total number of VDC allocations (#VDC) and the total CPU time in seconds required by each algorithm to allocate those VDCs. Additionally, we show the minimum (Min), Average (Avg), median (Mdn) and Maximum (Max) CPU time in seconds per VDC allocation. The full experimental setup is described in Section 5. Results continue on following pages.

	VDCs with 4 VMs					
	#VDC	Total (s)	Min (s)	Avg (s)	Mdn (s)	Max (s)
US-Middle1 Data Center, 4 clusters, 24 racks, 384 servers						
SecondNet	100	0.006	<0.001	<0.001	<0.001	<0.001
NETSOLVER-SMT	144	53.654	0.279	0.373	0.328	2.409
NETSOLVER-ILP	158	68.919	0.404	0.436	0.435	0.473
US-Middle2 Data Center, 1 cluster, 40 racks, 800 servers						
SecondNet	204	0.009	<0.001	<0.001	<0.001	<0.001
NETSOLVER-SMT	307	243.606	0.631	0.794	0.721	4.981
NETSOLVER-ILP	331	297.718	0.805	0.899	0.903	0.989
US-West1 Data Center, 2 clusters, 60 racks, 1200 servers						
SecondNet	305	0.015	<0.001	<0.001	<0.001	<0.001
NETSOLVER-SMT	462	586.680	1.004	1.270	1.155	7.584
NETSOLVER-ILP	499	680.962	1.255	1.364	1.374	1.487
US-West2 Data Center, 1 cluster, 14 racks, 280 servers						
SecondNet	71	0.003	<0.001	<0.001	<0.001	<0.001
NETSOLVER-SMT	104	27.990	0.201	0.269	0.228	1.738
NETSOLVER-ILP	112	33.808	0.280	0.302	0.303	0.327

Table 13
Continuation of Table 12, showing results for VDCs with 6 VMs.

	VDCs with 6 VMs					
	#VDC	Total (s)	Min (s)	Avg (s)	Mdn (s)	Max (s)
US-Middle1 Data Center, 4 clusters, 24 racks, 384 servers						
SecondNet	82	0.005	<0.001	<0.001	<0.001	<0.001
NETSOLVER-SMT	116	73.146	0.400	0.631	0.548	3.669
NETSOLVER-ILP	137	90.664	0.614	0.662	0.668	0.688
US-Middle2 Data Center, 1 cluster, 40 racks, 800 servers						
SecondNet	146	0.011	<0.001	<0.001	<0.001	<0.001
NETSOLVER-SMT	249	360.209	0.879	1.447	1.190	8.529
NETSOLVER-ILP	287	385.045	1.189	1.341	1.348	1.400

(*continued on next page*)

Table 13 (*continued*)

	VDCs with 6 VMs					
	#VDC	Total (s)	Min (s)	Avg (s)	Mdn (s)	Max (s)
US-West1 Data Center, 2 clusters, 60 racks, 1200 servers						
SecondNet	217	0.019	<0.001	<0.001	<0.001	<0.001
NetSolver-SMT	379	905.496	1.342	2.389	1.975	13.741
NetSolver-ILP	437	917.736	1.883	2.100	2.105	2.208
US-West2 Data Center, 1 cluster, 14 racks, 280 servers						
SecondNet	53	0.003	<0.001	<0.001	<0.001	<0.001
NetSolver-SMT	84	42.917	0.299	0.511	0.384	2.649
NetSolver-ILP	97	42.933	0.407	0.442	0.445	0.464

Table 14
Continuation of Table 12, showing results for VDCs with 8 VMs.

	VDCs with 8 VMs					
	#VDC	Total (s)	Min (s)	Avg (s)	Mdn (s)	Max (s)
US-Middle1 Data Center, 4 clusters, 24 racks, 384 servers						
SecondNet	53	0.005	<0.001	<0.001	<0.001	<0.001
NetSolver-SMT	93	103.128	0.531	1.109	0.725	6.957
NetSolver-ILP	111	94.669	0.809	0.853	0.855	0.891
US-Middle2 Data Center, 1 cluster, 40 racks, 800 servers						
SecondNet	120	0.008	<0.001	<0.001	<0.001	<0.001
NetSolver-SMT	186	478.705	1.183	2.574	1.619	26.338
NetSolver-ILP	230	403.041	1.655	1.752	1.754	1.843
US-West1 Data Center, 2 clusters, 60 racks, 1200 servers						
SecondNet	180	0.012	<0.001	<0.001	<0.001	<0.001
NetSolver-SMT	286	1171.278	1.819	4.095	2.563	42.955
NetSolver-ILP	346	978.825	2.632	2.829	2.834	2.984
US-West2 Data Center, 1 cluster, 14 racks, 280 servers						
SecondNet	42	0.003	<0.001	<0.001	<0.001	<0.001
NetSolver-SMT	67	48.530	0.385	0.724	0.513	4.105
NetSolver-ILP	83	49.793	0.555	0.600	0.601	0.647

Table 15
Continuation of Table 12, showing results for VDCs with 10 VMs.

	VDCs with 10 VMs					
	#VDC	Total (s)	Min (s)	Avg (s)	Mdn (s)	Max (s)
US-Middle1 Physical DC, 4 clusters, 24 racks, 384 servers						
SecondNet	48	0.005	<0.001	<0.001	<0.001	<0.001
NetSolver-SMT	67	69.605	0.598	1.039	0.837	6.019
NetSolver-ILP	92	99.813	1.022	1.085	1.087	1.133
US-Middle2 Physical DC, 1 cluster, 40 racks, 800 servers						
SecondNet	80	0.006	<0.001	<0.001	<0.001	<0.001
NetSolver-SMT	143	308.535	1.333	2.158	1.832	12.597
NetSolver-ILP	193	445.922	2.115	2.310	2.331	2.404
US-West1 Physical DC, 2 clusters, 60 racks, 1200 servers						
SecondNet	120	0.009	<0.001	<0.001	<0.001	<0.001
NetSolver-SMT	209	815.794	2.163	3.903	3.294	24.980
NetSolver-ILP	287	990.454	3.218	3.451	3.466	3.596
US-West2 Physical DC, 1 cluster, 14 racks, 280 servers						
SecondNet	28	0.002	<0.001	<0.001	<0.001	<0.001
NetSolver-SMT	50	38.824	0.446	0.776	0.538	4.362
NetSolver-ILP	66	49.241	0.693	0.746	0.749	0.779

Table 16
Continuation of Table 12, showing results for VDCs with 12 VMs.

	VDCs with 12 VMs					
	#VDC	Total (s)	Min (s)	Avg (s)	Mdn (s)	Max (s)
US-Middle1 Physical DC, 4 clusters, 24 racks, 384 servers						
SecondNet	48	0.005	<0.001	<0.001	<0.001	<0.001
Netsolver-SMT	65	96.038	0.728	1.478	1.166	7.146
Netsolver-ILP	81	104.445	1.179	1.289	1.297	1.350
US-Middle2 Physical DC, 1 cluster, 40 racks, 800 servers						
SecondNet	80	0.007	<0.001	<0.001	<0.001	<0.001
Netsolver-SMT	134	475.928	1.752	3.552	2.639	23.144
Netsolver-ILP	168	464.098	2.563	2.762	2.778	2.899
US-West1 Physical DC, 2 clusters, 60 racks, 1200 servers						
SecondNet	120	0.011	<0.001	<0.001	<0.001	<0.001
Netsolver-SMT	191	3195.215	2.904	16.729	4.516	800.931
Netsolver-ILP	254	1078.694	3.755	4.247	4.250	4.550
US-West2 Physical DC, 1 cluster, 14 racks, 280 servers						
SecondNet	28	0.002	<0.001	<0.001	<0.001	<0.001
Netsolver-SMT	46	970.648	0.564	21.101	0.853	919.691
Netsolver-ILP	59	53.903	0.845	0.913	0.918	0.958

References

[1] M.R. Prasad, A. Biere, A. Gupta, A survey of recent advances in SAT-based formal verification, Int. J. Softw. Tools Technol. Transf. 7 (2) (2005) 156–173, https://doi.org/10.1007/s10009-004-0183-4.
[2] C. Cadar, D. Dunbar, D. Engler, KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs, in: Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation, USENIX Association, Berkeley, CA, USA, 2008, pp. 209–224, http://dl.acm.org/citation.cfm?id=1855741.1855756.
[3] J. Rintanen, Madagascar: efficient planning with SAT, in: The 2011 International Planning Competition, 2011, p. 61.
[4] A. Biere, D. Kröning, Sat-based model checking, in: Handbook of Model Checking, Springer, 2018, pp. 277–303.
[5] Gurobi, Gurobi Optimizer Reference Manual, 2014.
[6] S. Bayless, N. Bayless, H. Hoos, A. Hu, SAT modulo monotonic theories, in: Proceedings of the 29th AAAI Conference on Artificial Intelligence, 2015.
[7] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, I. Stoica, Faircloud: sharing the network in cloud computing, in: Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM), ACM, 2012, pp. 187–198.
[8] J. Lee, Y. Turner, M. Lee, L. Popa, S. Banerjee, J.-M. Kang, P. Sharma, Application-driven bandwidth guarantees in datacenters, ACM SIGCOMM Comput. Commun. Rev. 44 (2014) 467–478.
[9] C. Guo, G. Lu, H.J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, Y. Zhang, Secondnet: a data center network virtualization architecture with bandwidth guarantees, in: Proceedings of the 6th International Conference on Emerging Networking Experiments and Technologies, Co-NEXT '10, ACM, 2010, p. 15.
[10] S. Bayless, N. Kodirov, I. Beschastnikh, H.H. Hoos, A.J. Hu, Scalable constraint-based virtual data center allocation, in: Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI), 2017.
[11] F. Hutter, H.H. Hoos, K. Leyton-Brown, Sequential model-based optimization for general algorithm configuration, in: Proceedings of the 5th Learning and Intelligent Optimization Conference (LION '11), vol. 5, 2011, pp. 507–523.
[12] A.N. Tantawi, A scalable algorithm for placement of virtual clusters in large data centers, in: 2012 IEEE 20th International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), IEEE, 2012, pp. 3–10.
[13] H. Ballani, P. Costa, T. Karagiannis, A. Rowstron, Towards predictable datacenter networks, ACM SIGCOMM Comput. Commun. Rev. 41 (2011) 242–253.
[14] M.F. Zhani, Q. Zhang, G. Simona, R. Boutaba, Vdc planner: dynamic migration-aware virtual data center embedding for clouds, in: 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), IEEE, 2013, pp. 18–25.
[15] M. Rost, C. Fuerst, S. Schmid, Beyond the stars: revisiting virtual cluster embeddings, ACM SIGCOMM Comput. Commun. Rev. 45 (3) (2015) 12–18.
[16] M. Yu, Y. Yi, J. Rexford, M. Chiang, Rethinking virtual network embedding: substrate support for path splitting and migration, ACM SIGCOMM Comput. Commun. Rev. 38 (2) (2008) 17–29.
[17] X. Cheng, S. Su, Z. Zhang, H. Wang, F. Yang, Y. Luo, J. Wang, Virtual network embedding through topology-aware node ranking, ACM SIGCOMM Comput. Commun. Rev. 41 (2) (2011) 38–47.
[18] N.M.K. Chowdhury, M.R. Rahman, R. Boutaba, Virtual network embedding with coordinated node and link mapping, in: Proceedings of the IEEE International Conference on Computer Communications (INFOCOM), IEEE, 2009, pp. 783–791.
[19] J. Lischka, H. Karl, A virtual network mapping algorithm based on subgraph isomorphism detection, in: Proceedings of the 1st ACM Workshop on Virtualized Infrastructure Systems and Architectures, ACM, 2009, pp. 81–88.
[20] T. Huang, C. Rong, Y. Tang, C. Hu, J. Li, P. Zhang, Virtualrack: bandwidth-aware virtual network allocation for multi-tenant datacenters, in: 2014 IEEE International Conference on Communications (ICC), IEEE, 2014, pp. 3620–3625.
[21] Y. Yuan, A. Wang, R. Alur, B.T. Loo, On the feasibility of automation for bandwidth allocation problems in data centers, in: Formal Methods in Computer-Aided Design (FMCAD), 2013, IEEE, 2013, pp. 42–45.
[22] X. Meng, V. Pappas, L. Zhang, Improving the scalability of data center networks with traffic-aware virtual machine placement, in: Proceedings of the IEEE International Conference on Computer Communications (INFOCOM), IEEE, 2010, pp. 1–9.
[23] D. Kakadia, N. Kopri, V. Varma, Network-aware virtual machine consolidation for large data centers, in: Proceedings of the Third International Workshop on Network-Aware Data Management, ACM, 2013, p. 6.
[24] H. Ballani, D. Gunawardena, T. Karagiannis, Network Sharing in Multi-Tenant Datacenters, Tech. rep., https://www.microsoft.com/en-us/research/publication/network-sharing-in-multi-tenant-datacenters/, February 2012.
[25] K. LaCurts, J.C. Mogul, H. Balakrishnan, Y. Turner, Cicada: Introducing Predictive Guarantees for Cloud Networks, vol. 14, 2014, pp. 14–19.
[26] S. Angel, H. Ballani, T. Karagiannis, G. O'Shea, E. Thereska, End-to-end performance isolation through virtual datacenters, in: Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2014, pp. 233–248.

[27] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, M. Handley, Improving datacenter performance and robustness with multipath TCP, ACM SIGCOMM Comput. Commun. Rev. 41 (2011) 266–277.
[28] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, F. Matus, R. Pan, N. Yadav, G. Varghese, et al., Conga: distributed congestion-aware load balancing for datacenters, ACM SIGCOMM Comput. Commun. Rev. 44 (2014) 503–514.
[29] A. Vahdat, A look inside Google's data center network, youtube.com/watch?v=FaAZAII2x0w. (Accessed 26 November 2017).
[30] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, S. Lu, BCube: a high performance, server-centric network architecture for modular data centers, ACM SIGCOMM Comput. Commun. Rev. 39 (4) (2009) 63–74.
[31] A. Greenberg, J.R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D.A. Maltz, P. Patel, S. Sengupta, VL2: a scalable and flexible data center network, ACM SIGCOMM Comput. Commun. Rev. 39 (2009) 51–62.
[32] TRILL: transparent interconnection of lots of links, en.wikipedia.org/wiki/TRILL_(computing). (Accessed 26 November 2017).
[33] N.G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K.K. Ramakrishnan, J.E. van der Merive, A flexible model for resource management in virtual private networks, ACM SIGCOMM Comput. Commun. Rev. 29 (1999) 95–108.
[34] A. Belbekkouche, M.M. Hasan, A. Karmouch, Resource discovery and allocation in network virtualization, IEEE Commun. Surv. Tutor. 14 (4) (2012) 1114–1128.
[35] A. Fischer, J.F. Botero, M.T. Beck, H. De Meer, X. Hesselbach, Virtual network embedding: a survey, IEEE Commun. Surv. Tutor. 15 (4) (2013) 1888–1906.
[36] A. Fischer, J.F. Botero Vega, M. Duelli, D. Schlosser, X. Hesselbach Serra, H. De Meer, Alevin—a framework to develop, compare, and analyze virtual network embedding algorithms, in: Electronic Communications of the EASST, 2011, pp. 1–12.
[37] M.F. Bari, R. Boutaba, R. Esteves, L.Z. Granville, M. Podlesny, M.G. Rabbani, Q. Zhang, M.F. Zhani, Data center network virtualization: a survey, IEEE Commun. Surv. Tutor. 15 (2) (2013) 909–928.
[38] A. Gupta, J. Kleinberg, A. Kumar, R. Rastogi, B. Yener, Provisioning a virtual private network: a network design problem for multicommodity flow, in: Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing, ACM, 2001, pp. 389–398.
[39] W. Szeto, Y. Iraqi, R. Boutaba, A multi-commodity flow based approach to virtual network resource allocation, in: Global Telecommunications Conference (GLOBECOM), vol. 6, IEEE, 2003, pp. 3004–3008.
[40] S. Even, A. Itai, A. Shamir, On the complexity of time table and multi-commodity flow problems, in: Proceedings of the 16th Annual Symposium on Foundations of Computer Science, IEEE, 1975, pp. 184–193.
[41] M.Y. Sir, I.F. Senturk, E. Sisikoglu, K. Akkaya, An optimization-based approach for connecting partitioned mobile sensor/actuator networks, in: 2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), IEEE, 2011, pp. 525–530.
[42] D.E. Kaufman, J. Nonis, R.L. Smith, A mixed integer linear programming model for dynamic route guidance, Transp. Res., Part B, Methodol. 32 (6) (1998) 431–440.
[43] IBM, ILOG CPLEX Optimization Studio 12.6.1: multi-commodity flow cuts, https://www.ibm.com/support/knowledgecenter/SSSA5P_12.6.1/ilog.odms.cplex.help/CPLEX/UsrMan/topics/discr_optim/mip/cuts/37_mcf.html.
[44] H. Marchand, A. Martin, R. Weismantel, L. Wolsey, Cutting planes in integer and mixed integer programming, Discrete Appl. Math. 123 (1–3) (2002) 397–446.
[45] S. Bayless, SAT Modulo Monotonic Theories, Ph.D. thesis, University of British Columbia, 2017.
[46] R. Bar-Yehuda, S. Even, A local-ratio theorem for approximating the weighted vertex cover problem, North-Holl. Math. Stud. 109 (1985) 27–45.
[47] N. Eén, N. Sorensson, Translating pseudo-Boolean constraints into SAT, J. Satisf. Boolean Model. Comput. 2 (2006) 1–26.
[48] N. Sorensson, N. Een, Minisat – a SAT solver with conflict-clause minimization, in: SAT 2005, vol. 53, 2005, pp. 1–2.
[49] L. De Moura, N. Bjørner, Z3: an efficient SMT solver, in: Tools and Algorithms for the Construction and Analysis of Systems, 2008, pp. 337–340.
[50] F. Hutter, M. López-Ibánez, C. Fawcett, M. Lindauer, H.H. Hoos, K. Leyton-Brown, T. Stützle, Aclib: a benchmark library for algorithm configuration, in: International Conference on Learning and Intelligent Optimization, Springer, 2014, pp. 36–40.
[51] P. Bodík, I. Menache, M. Chowdhury, P. Mani, D.A. Maltz, I. Stoica, Surviving failures in bandwidth-constrained datacenters, in: Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, ACM, 2012, pp. 431–442.
[52] A. Andreyev, Introducing data center fabric, the next-generation Facebook data center network, code.facebook.com/posts/360346274145943. (Accessed 26 November 2017).
[53] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, A. Goldberg, Quincy: fair scheduling for distributed computing clusters, in: Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles, ACM, 2009, pp. 261–276.
[54] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, I. Stoica, Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling, in: Proceedings of the 5th European Conference on Computer Systems, ACM, 2010, pp. 265–278.
[55] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, V. Sekar, Making middleboxes someone else's problem: network processing as a cloud service, ACM SIGCOMM Comput. Commun. Rev. 42 (4) (2012) 13–24.