On the Scaling of Virtualized Network Functions

Windhya Rankothge
Sri Lanka Institute of Information Technology
Malabe, Sri Lanka
windhya.r@sliit.lk

Helena Ramalhinho
Universitat Pompeu Fabra
Barcelona, Spain
helena.ramalhinho@upf.edu

Jorge Lobo
ICREA & Universitat Pompeu Fabra
Barcelona, Spain
jorge.lobo@upf.edu

***Abstract*—Offering Virtualized Network Functions (VNFs) as a service requires automation of cloud resource management to allocate cloud resources for the VNFs dynamically. Most of the existing solutions focus only on the initial resource allocation. However, the allocation of resources must adapt to dynamic traffic demands and support fast scaling mechanisms. There are three basic scaling models: vertical where re-scaling is achieved by changing the resources assigned to the VNF in the host server, horizontal where VNFs are replicated or removed to do re-scaling, and migration where VNFs are moved to servers with more resources. In this paper, we present an Iterated Local Search (ILS) based framework for automation of resource re-allocation that supports the three scaling models. We, then, use the framework to run experiments and compare the different scaling approaches, specifically how the optimization is affected by the scaling approach and the optimization objectives.**

***Index Terms*—Virtualized Network Functions, Cloud Resources Management, Iterated Local Search**

I. Introduction

Most of the existing work on automation of VNF resource management, deal with the initial resource allocation for new requests. In the initial resource allocation, clients provide requirements in the form of NFs chains with specification of expected traffic volume. Then the NF Cloud Service Provider (CSP) allocates resources to run the NFs as virtual machines and configure the network, so that the traffic passes through the required VNFs in the appropriate order [1]–[5]. The initial resource allocation for VNFs can be done in the order of minutes, and then the new VNFs can be deployed accordingly.

However, following the initial resource allocation, over time, without over-provisioning, resources have to be reallocated to adapt to dynamic changes in traffic volumes. When traffic increases, the CSP needs to adjust the resource allocation to fulfil service levels agreed with the client (scale out) and when traffic decreases, the CSP needs to recover underutilized resources that can be assigned to other clients (scale in). The resource re-allocation for the scaling requirements happens during the ongoing operations, where the already deployed VNFs are processing traffic. Therefore, it is a time critical on-line problem. Solutions must be provided in the order of seconds or even milliseconds, so that disturbances to current operations are minimal.

Jorge Lobo was partially supported by the Spanish Ministry of Economy and Competitiveness. Grant Numbers: TIN-2016-81032-P, MDM-2015-0502.

978-3-903176-15-7 © 2019 IFIP

In this paper we focus on issues related to automating the resource re-allocation to satisfy scaling requirements, and assume the existence of a module that triggers the re-allocation when needed. Despite some initial efforts [6], [7] the automation of dynamic scaling of VNFs still has open challenges. Deciding what scaling method to use, i.e., whether to use ***vertical scaling*** (i.e., allocation/release of host and bandwidth resources to/from a VNF instance), ***migration scaling*** (i.e., running VNFs are paused, its state is serialized and transferred to different servers with more resources), or ***horizontal scaling*** (i.e., installation/removal of VNF instances), is not obvious because of potentially conflicting optimization objectives: re-allocating resources in a way that minimizes changes to current configuration and therefore current network activities are minimally disturbed, and at the same time optimizing usage of server and network resources [8]. To our knowledge, there is no systematic study comparing the three scaling models.

The contributions of this paper are: (1) a Integer Linear Programming (ILP) model for automation of dynamic scaling of VNFs in situations where there might not be feasible solutions, i.e., there are not sufficient resources to cover the demands under the restrictions of resource re-allocations imposed by the scaling method. In practice, no feasible solution doesn't mean that no service will be provided. Instead the service will be provided with a lower quality, such as a lower throughput. Hence, our optimization goal is to fulfil as much as possible of the bandwidth requested. We have defined *bandwidth dropped* (presented in the results in percentage) as the amount of the bandwidth that has been requested, but cannot be covered by the resources allocated, and the optimization as the minimization of the bandwidth dropped. (2) an Iterated Local Search (ILS) method that efficiently solve the problem and can be implemented in real-life and (3) a comparison on the different scaling models. For automation of dynamic scaling of VNFs, the ILP models allows us to understand the complexity of the problem and evaluate the quality of the ILS heuristics proposed.

The ILS heuristics has proved to be one of the best approximation methods to solve many complex optimization problems efficiently [9]. For the comparison of scaling models, we compare the scaling approaches in terms of the resource allocation efficiency produced by each approach, the bandwidth dropped and changes required by the reconfiguration.

We developed the ILS based framework to handle the three scaling methods and various optimization objectives, including

bandwidth dropped minimization. With the framework implementation, we have conducted experimental simulations using the data generation process described in [10], [11] which is based on published information about the use of VNFs in enterprises and traffic patterns from a real network. Our simulations showed that, on average, the bandwidth dropped was 49.6% for vertical scaling, 3.89% for migration and 0.12% for horizontal scaling, pointing that in any autonomic scaling method selection, the less intrusive vertical scaling alone should be avoided. However, even though there is a clear difference in the average bandwidth dropped between migration and horizontal scaling, further results showed, not surprisingly, that horizontal scaling accepts more requests at the cost of one, allocating more CPUs, and two, causing more changes to the running configurations. The more changes are done, the more likely services will affected during re-allocation of resources.

The rest of the paper is organized as follows. Section II gives an insight to the existing work on the same domain. Section III formalizes the bandwidth dropped optimization problem within ILP and Section IV describes the implementation of the ILS based resource allocation framework. Section V presents the evaluation set-up and the results of our scaling model comparison. Our final remarks can be found in Section VI.

II. Related Work

The VNF placement problem, as it generalizes the VM allocation problem, is NP-hard [12]. Hence, instead of optimal solutions which take long time to find, heuristic-based solutions are preferred. [2] provides a dynamic programming based heuristic to solve larger instances of the VNF placement problem. [13] uses a greedy algorithm and tabu based local search techniques. But most work on automating cloud resource management focuses on the initial placement of VNFs [1]–[5], [13]–[15]. However, to offer a flexible service, following the initial placement, VNFs have to be rescaled over time.

Work on automation of the resource re-allocation for scaling VNFs is very limited [6], [7], [10]. The common strategy is to avoid the hard problem of finding a global optimal, partitioning the problem, and approximating to local optimal of the sub-problems. The approach in [6] works with VNF chains and can solve the initial placement problem by iterating over initial input chains. Scaling is achieved by duplicating full instances of VNF chains, and by reshuffling chains and migrating traffic. The optimization is very dependent on the data center architecture. The approach in [7] pushes the locality to the limit and tries to deal with the optimization of one VNF at a time, independent of the location in the chain of VNFs. They take as input a set of individual VNFs demands and they might migrate, duplicate or remove VNF instances, trying to optimize server and bandwidth utilization as well as the cost of making the modifications. Our previous work [10], [16] takes the middle approach. It works with NF chains, assuming that for a chain, it is likely that resources demand changes happen in isolated NFs, but the optimization is done in the context of the chain.

Furthermore, research work on scaling of VNFs has focused only on the complexities of the scaling technologies. They do not consider the resource allocation or optimization aspects of the scaling methods. [17], [18] studied the re-scaling of VMs using vertical approach, according to the dynamic changes in the demand for the applications that they host. [19]–[22] proposed optimizations for VM migrations.

Our work is inspired by these existing work, but it focuses on comparing scaling approaches based on the effects of the scaling method on the resource allocation.

III. ILP model for dynamic scaling of VNFs problem

In this section, we introduce our ILP model to solve the problem of dynamic scaling of VNFs which builds on the previous work done in VNF in [23] and in [6]. In our formalization, we consider a network function center (NFC) to be a cloud center providing VNF services, having M servers and L links.

A link l connects a server to a switch, or a switch to another switch. The amount of resource capacity (number CPUs as basic resource) of server m is denoted H_m, and network capacity of link l is denoted K_l. A path p between two servers (a source and a destination server pair), comprises two or more links (there is at least one switch between connected servers). P denotes the set of available paths between all source and destination server pairs in the NFC. Given a path p (in P) that connects servers $m1$ and $m2$, Q_p represents the source server ($m1$), and R_p represents the destination server ($m2$). The variable E_l^p indicates whether link l is used on path p.

We assume the existence of a module that triggers the re-allocation when needed. A resource allocation request has the following specifications: (1) the initial policy configuration given by the types of required VNFs, numbered 1 to N, and interconnectivity between them (policy), given by pairs (n, n'), indicating that VNF n' follows n in the policy[1]; (2) the maximum expected delay d, and (3) the expected input traffic load B_0 to be processed by these VNFs.

This traffic load is not necessarily the load that will flow through the paths between NFs of the policy. Depending on the type of NF, the traffic output by a VNF can decrease (e.g. WAN optimizer) or increase (e.g., VPN) with respect to the input traffic. This scaling factor will be known to the system. The traffic will be re-scaled accordingly and set in a constant B_n for each VNF n. The constant B_n is, therefore, computed as a function of the scaling factor of the VNF n: sf_n and the input traffic load expected by the VNF. For the special case of first VNF in the chain, the input traffic load, B_0, is part of the re-allocation specification. The input traffic load for the remaining VNFs will depend on the scaling factor of the preceding VNF and its input load.

Table I lists all the constants and dynamic variables relevant for the formalization.

[1]It is assumed the policy is a chain of NFs.

The minimal bandwidth dropped rate is given by the minimization of:

$$-B'_0 \tag{1}$$

subject to:

$$\sum_{m=1}^{M} Z_n^m = 1, \forall n \tag{2a}$$

$$\sum_{n=1}^{N} Z_n^m . S'_n \leq H_m . G_m, \forall m \tag{2b}$$

$$\sum_{p=1}^{P} A_n^p = 1, \forall n \tag{2c}$$

$$\sum_{p=1}^{P} \sum_{n=1}^{N} A_n^p . E_l^p . B'_n \leq K_l . F_l, \forall l \tag{2d}$$

$$\sum_{p=1}^{P} \sum_{n=1}^{N} A_n^p . E_l^p - F_l \geq 0, \forall l \tag{2e}$$

$$\sum_{p=1}^{P} A_n^p . Q_p - \sum_{m=1}^{M} Z_n^m . m = 0, \forall n \tag{2f}$$

$$\sum_{p=1}^{P} A_n^p . R_p - \sum_{m=1}^{M} Z_{n'}^m . m = 0, \forall (n, n') \tag{2g}$$

$$\sum_{m=1}^{M} d_{nf} . G_m + \sum_{l=1}^{L} d_{lk} . F_l \leq d \tag{2h}$$

$$S'_n - S_n < cpu_n, \forall n \tag{2i}$$

$$B'_{n-1} \leq cap_n S'_n, \forall n \tag{2j}$$

$$B'_n = sf_n . B'_{n-1}, \forall n \tag{2k}$$

$$B'_0 \leq B_0 \tag{2l}$$

Constraints (2a) and (2b) model the server resource constraints of the problem. Constraint (2a) guarantees that each VNF in a policy is placed on one and only one server. Constraint (2b) guarantees that the total capacity allocated to all VNFs placed on a server does not exceed total capacity of that server.

Constraints (2c) to (2h) model the network resource constraints of the problem. If a VNF is not the last VNF of a policy, constraint (2c) guarantees that the VNF has a path to its successor. Constraint (2d) guarantees that for each link allocated to transport traffic output by a VNF, the link has sufficient bandwidth capacity left to accommodate the traffic generated by the VNF. Constraint (2e) guarantees that a link is counted as "used", only if it is actually used in the configuration solution. Constraints (2f) and (2g) guarantee that the path selected for a VNF to send traffic to its successor, starts from the server where the VNF resides (source server), and ends in the server where the VNF's successor resides (destination server). Constraint (2h) guarantees that the delays introduced by traversing VNFs in VNF chain and traversing links in the path allocated inside the NFC, does not exceed the delay accepted by the client.

Constants characterizing the NFC architecture	
M	No. of servers, indexed by $m = 1, ..., M$
H_m	Capacity of server m
L	No. of links, indexed by $l = 1, ..., L$
K_l	Bandwidth of link l
P	No. of paths, indexed by $p = 1, ..., P$
Q_p	Source server of the path p
R_p	Destination server of the path p
E_l^p	Indicates whether the link l is used on path p, $(0, 1)$
w_1, w_2, w_3	Weighting factors
Constants characterizing the re-scaling policies to be served	
N	No. of VNFs, indexed by $n = 1, ..., N$
S_n	Server capacity required for VNF n
B_n	Bandwidth required for VNF n
B_0	Input bandwidth required by the re-scaling
d	Max delay
d_f	delay caused by traversing a VNF
d_k	delay caused by traversing a link
Dynamic variables for new policy requests provisioning	
Z_n^m	A binary decision for placing VNF n on server m
A_n^p	A binary decision for routing traffic of VNF n on path p
G_m	Server m is used/not
F_l	Link l is used/not
S'_n	Server capacity allocated for VNF n
B'_n	Bandwidth allocated for VNF n
B'_0	Input bandwidth allocated to the re-scaling

TABLE I
SUMMARY OF KEY NOTATIONS

If a new VNF needs to be allocated, there is a minimum number of CPU unites required and it depends on the VNF type. The minimum number of CPU unites required for a particular VNF type, also constrains the number of CPU unites that can be requested for scaling. The number of CPU unites that can be requested for scaling can only be multiples of minimum number of CPU unites required for a particular VNF type. For example, if an IP firewall implementation that handles 900 Mbs loads requires 4 CPUs, the minimum number of CPUs that can be allocated to handle a 1000 MBs load would be 4 more CPUs. This is a simplification of the typical offer made by VNF vendors of products with discrete jumps in capacities. This is also important to capture horizontal scale up where the replication is done for a new full VNF instance. The constant cpu_n denotes the number of CPU unites required by a single scaling (in or out) of VNF n. Constraint (2i) bounds the CPUs required for a VNF type. Constraint (2j) forces that the allocation of CPUs to a VNF is never more than the minimum necessary to handle the incoming traffic load – cap_n is the traffic load capacity associated to an instance of the type of VNF n. The constant sf_n is the traffic scaling factor associated to the type of VNF n. Given that the traffic output by a VNF can decrease (e.g. WAN optimizer) or increase (e.g., VPN) with respect to the input traffic based on the scaling factor, Constraint (2k) guarantees that, if the bandwidth reserved for VNF $n-1$ is B'_{n-1}, then the bandwidth reservation for VNF n is calculated considering the scaling factor of the VNF: sf_n. Constraint (2l) forces that the bandwidth allocation to the input traffic is never larger than the allocated VNF capacity. Examples of cpu_ns, cap_ns and sf_ns will be found in the

experiment section (Sec. V).

The optimization objective is to minimize the bandwidth dropped rate. Exact solutions to this optimization problem are difficult to obtain. It takes minutes to hours to solve when there are hundreds of servers connected through different paths using a typical data center architecture such as k-fat trees or BCubes.

For this purpose we build upon our previous work [10], [16], [23] where we developed an approximation algorithm to address the computationally complexity of finding optimal resource re-allocation during scaling. In [16], [23], we used Genetic Programming (GP) techniques for the approximation. Since then, we have developed a much effective algorithm based on Iterated Local Search (ILS) [9]. There are two (interrelated) benefits of using the new ILS algorithm. The first is that the space and time requirements of GP algorithms are proportional to the size of the genetic population times the space needed to represent a resource allocation state of the system (i.e., the resource allocation represented by each solution of the population). In contrast, ILS requires information about a single state, the current state of the system. The second is that the empirical evaluations show that ILS consistently returns better approximations than the GP algorithms. A third intangible but nevertheless crucial benefit of adopting the new implementation is the simplicity and flexibility of the ILS approach to adapt well to many situations. This allowed us to evaluate different scaling methods easily. Modifying our GP implementation would have been much harder.

In the next section we present our approximation algorithm based on Iterated Local Search.

IV. Iterated Local Search (ILS) Approach

The essence of the ILS meta-heuristic is to iteratively build a sequence of solutions generated by an embedded heuristic search, leading to far better solutions than if one were to use repeated random trials of that heuristic [9]. ILS extends a problem-specific local search method by introducing a perturbation at each new local optimal solution, before restarting the search for a new local optimal solution. As shown in Algorithm 1, ILS comprises four main steps: (1) Generation of an Initial Solution, (2) Local Search, (3) Perturbation and (4) Acceptance Criterion. ILS aims at avoiding the disadvantages of random restarts by exploring the region of feasible solutions using a walk that steps from one local optimal solution S^* to a nearby one. Given the current solution S^*, a change or perturbation is first applied leading to an intermediate feasible solution, S'. Then, a Local Search is applied to S' to obtain a new local optimal solution, $S^{*\prime}$. If $S^{*\prime}$ passes an acceptance test, it becomes the new solution; otherwise, one returns to the previous one, S^*.

The ILS process starts with the first phase of generating a feasible solution for the scaling requirement. We call our implemented solution for dynamic scaling of VNFs problem as the ***Cloud Resources Management Program (CRMP)***.

The CRMP starts with the current state and searches for the re-assignment of resources (new servers and paths) for the set of VNFs that are scaling using Depth First Search (DFS) method. The search is performed based on the scaling approach that is used. **In all three scaling approaches, the CRMP tries to find resources (server and bandwidth) to satisfy the total bandwidth demand, while ensuring the traffic of the policy request meet its deadline. If the required resources cannot be allocated, then the CRMP tries to allocate resources to accept as much as possible bandwidth demand.**

Algorithm 1 Iterated Local Search for CRMP

```
1: procedure ITERATED LOCAL SEARCH
2:     S_0 = GenerateInitialSolution
3:     S* = LocalSearch(S_0)
4:     Repeat:
5:         S' = Perturbation(S*, history)
6:         S*' = LocalSearch(S')
7:         S* = AcceptanceCriteria(S*, S*', history)
8:     until termination condition met
```

With Vertical scaling, when a policy has to be scaled out, for each VNF of the scaling policy, the CRMP checks whether the server and the path, that is currently used by the VNF, can handle the bandwidth demand. When a policy has to be scaled in, for each VNF of the scaling policy, the CRMP decreases server resources from the server where the VNF currently resides, and decreases bandwidth resources from the paths that VNF currently uses.

With migration scaling, when a policy has to be scaled out, we treat as if it is a new policy. Also, we assume that the new policy's bandwidth request is the total bandwidth demand: (current bandwidth plus traffic change). For each VNF of the scaling policy, the CRMP searches for a server and a path to handle the bandwidth demand. After the new policy has been implemented, the resources allocated to the existing policy are removed. This is done because, it is expected that both implementations would need to co-exist for sometime to be able to handle session affinity to ensure the correct behaviour of the VNF. When a policy has to be scaled in, for each VNF of the scaling policy, the CRMP decreases server resources from the server where the VNF currently resides, and decreases bandwidth resources from the paths that VNF currently uses.

With horizontal scaling, when a policy has to be scaled out, we again treat it as a new policy, and that the new policy's bandwidth request is the extra bandwidth demand (the traffic change) of the existing policy. We called it the "child policy". For each VNF of the policy that is scaling, the CRMP searches for a server and a path to handle the extra bandwidth demand, using a DFS method. When a policy has to be scaled in, for each VNF of the scaling policy, the CRMP decreases server resources from the server where the VNF currently resides, and decreases bandwidth resources from the paths that VNF currently uses. If this decrease means removing all the resources from a child policy, then we remove that child policy entirely.

The current solution that describes the current state of the system is modified according to the new servers and paths

found. The seconds phase of ILS is the local search. We apply a simple iterative improvement local search, i.e., as soon as a better solution is found, the process will restart using this solution as the current one. A neighborhood for the local search is defined by first choosing a type of transaction to obtain a new feasible solution from the current one: move, and then defining the neighborhood as the set of solutions that can be obtained from the current one by applying the same type of move. For migrations and horizontal scaling, we define a simple move – changing a scaling VNF from one server to another feasible server: "1-opt-VNF move", and afterwards changing paths of the VNF affected. For each VNF, its neighbors can be found by applying "1-opt-VNF moves".

The perturbation process in our algorithms is implemented by applying n "1-opt-VNF moves", where $2 < n <$Number of VNFs. It works by randomly selecting n VNFs in randomly selected scaling policies, and changing each selected VNF from one server to another feasible server by applying "1-opt-VNF move", and afterwards changing paths of the VNF affected. The acceptance criteria phase uses the objective function given in equation 1. A solution obtained after the perturbation and local search is accepted only if it improves the current solution. The acceptance criteria (step 7 in Algorithm 1) is an improvement in the objective function.

For the initial resource allocation (before any scaling happens), the CRMP performs a simple Depth First Search (DFS), and selects a server and a path for each VNF in each initial policy request considering the expected traffic load by the policy. The initial resource allocation generated by DFS is further improved by an ILS algorithm, with the following modifications: within the local search procedure, CRMP tries to perform a "global ILS" to the given solution, where it applies "1-opt-VNF move", to all VNFs of any policy (a standard full local search). In the perturbation process, "1-opt-VNF move" is applied to n randomly selected VNFs from all the policies. In the acceptance criteria phase, the objective function is as in [10] where the objective functions is to minimize the number of servers and links used.

A. Verification of the ILS implementation

Before we move to the comparison of scaling approaches we would like to say a few words about the ILS implementation. We have conducted a comprehensive evaluation of the performance of the ILS CRMP. We followed the same experimental regime reported in [10] used to evaluate our resource allocation system based on Genetic Algorithms (GA). The experiments verified the stability of the ILS system when parameters such as initial conditions or order of processing requests were varied.

With regards to its performance when compared to the optimal solutions we needed to limit our experiments to small network cloud scenarios to get the ILP implementation to finish. We used a small network with 4 servers, and ran simulations to allocate resources for up to 10 VNFs. Our ILS based algorithm was able to find the optimal solutions within few milliseconds while the ILP implementation in CPLEX [24] took several minutes.

For the comparison to the performance of the GA implementation, we ran experiments in which we, first, found initial solutions with DFS, and then improved the initial solutions using ILS with 20 repeated procedures and GP with 200 generations separately in three different clould center architectures, k-fat trees, BCubes and VL2s. The ILS approximations were always as good or better than the approximations found by the GA system. It was observed in [23] that after the 200 generations, the improvements are very rare for the GP approach. However, for the ILS approach, if one continue repeating procedures the solutions improved.

As the main goal of the optimization in [23] was also to reduce the average link utilization, so that the network is less congested, we repeated the tests and the results produced by ILS reduced the average link utilization by 32%, 27.2% and 7% when compared to the initial DFS solution for the three architectures, while the results produced by GP were 28.7%, 14.9% and 3.2%. In addition, ILS was faster than GP.

V. Scaling methods evaluation

There are two different aspects to look at when selecting a scaling method: (1) the impact of its implementation and (2) the optimization goals. Existing work has shown that each scaling method has advantages and disadvantages [25].

Vertical scaling: (1) needs less time for reconfiguration as it needs only metrics adjustment, (2) does not need additional software licenses, (3) does not affect the quantity of VNF instances and (4) does not introduce a coordination or a traffic distribution among multiple VNF instances. However, vertical scaling is limited by the capacity of individual servers and links, because it cannot increase the resources more than their maximum capacity. The duration of reconfiguration for migration scaling is higher, because it includes the time to duplicate the scaling VNF in a new server (VNF migration) and then initialize the instance [26], [27]. Even though, horizontal scaling has more freedom in terms of space, it needs to create new VNF instances and therefore, it needs a workload balancer to distribute the traffic to multiple VNF instances [28], [29].

The second aspect is to select a configuration solution based on the optimization goals. This has not been studied, hence, we decided to run simulations to explore on how the optimization is affected by the scaling approach and the optimization objectives.

A. Experimental set-up

The work in this section is based on our previous experimental setting reported in [16] and [10]. For a more realistic evaluation, we used the data generation process described in [10] which is based on published information about the use of VNFs in enterprises and traffic patterns from a real network. The data generation and data modeling programs can be found in [11], [30].

Following [10], we assumed that we are providing services for policy requests of 6 large enterprises. Therefore, the system

NF type	Processing capacity	CPU cores required
IP firewall	900 Mbps	4
Application firewall	900 Mbps	4
WAN optimizer	900 Mbps	4
Proxy	900 Mbps	4
Gateway	900 Mbps	4
VPN	900 Mbps	2
Load balancer	900 Mbps	4
IDS/ IPS	600 Mbps	8

TABLE II
PROCESSING CAPACITIES AND CPU REQUIREMENTS OF VNFS

NF type	Traffic scaling	Traffic scaling factor
IP firewall	No	-
Application firewall	No	-
WAN optimizer	Yes	0.65
Proxy	Yes	0.65
Gateway	No	-
VPN	Yes	1.5
Load balancer	No	-
IDS/ IPS	Yes	1.5

TABLE III
BANDWIDTH DEMAND TRANSFORMATIONS OF VNFS

handles around 130 policies with 600 NFs which were chains with at least 2, but no more than 7 VNFs, that followed a truncated power-low distribution with exponent 2. We assumed a 128 server environment in a 4-fat tree architecture where each server has an initial capacity of 24 cpu cores and each link has an initial capacity of 10 Gbps.

We considered 7 types of VNFs: IP firewalls, Application firewalls, WAN optimizers, Proxies, Gateways, VPNs, Load balancers, and IDS/IPS as used in [10]. We also made more precise specifications of the VNFs. The computational requirements for each VNF to serve a specific amount of traffic, can be derived using the inbound traffic rate and the resource profile of each VNF type. A resource profile for each NF includes: (1) processing capacity of the VNF in Mbps, (2) required minimum number of CPU cores and (3) bandwidth demand transformation by the VNF (compress or amplify traffic).

Following the existing works [2], [31], Table II shows the assumed processing capacities and the required number of CPU cores for different VNFs in our experiments.

The bandwidth demand transformations are associated with traffic-scaling VNFs. Following the existing works [32], Table III shows the assumed traffic scaling factors for different VNFs in our experiments. A traffic scaling factor smaller than 1 implies traffic compression while a traffic scaling factor greater than 1 implies traffic amplification.

As in [33], we take the end-to-end delay requirement (deadline) for a packet within a CSP network to be 1 ms. In our model, we assumed that the end-to-end delay experienced by each packet (inside the NFC) is composed of: (1) processing delays of VNFs in the policy request, and (2) transmission delays between VNFs of the policy request [6]. We assumed that the processing delay of a VNF is equal to $\frac{1}{cap}$, where cap is the processing capacity of that VNF [6]. The transmission delay between two VNFs is calculated based on the allocated path, where the delay caused by a link in the path is assumed to be 0.001 ms [34]. Even though we limited the end-to-end delay to these two components and considered only the delay inside the cloud, one can easily add more delay components such as the extra delay caused by moving NFs to the cloud or migrating traffic from one VNF instance to another.

For the VNFs scaling methods evaluation, we considered traffic changes of every hour, to get the scaling triggers and derived the scaling requirements. We will refer to the scaling requirements of each hour a "scaling event" and therefore, we have 20 scaling events in a full day.

B. Evaluation

All experiments were carried out in a machine with an Intel core i3 processor and 20GB of RAM. For each experiment round, we started with two initial resource allocations: (1) Depth First Search (DFS) and (2) random. Then we used the solutions of the two methods separately to test resource re-allocation for scaling over time. We ran the algorithm with the three scaling methods (vertical, migration and horizontal) separately. We repeated this process for 10 experiment rounds, where for each experiment round, we derived a fixed set of policies (average of 130 policies) that included a total of 600 VNFs. For each experiment round, scaling events were derived based on the policy set used and the traffic data. All the results are the averages of the 10 experiment rounds.

1) Percentage of bandwidth dropped: Our optimization goal was to minimize the bandwidth dropped rate of scaling requests with a constraint that ensures the delay experienced by each packet of accepted scaling requests, did not exceed their deadlines. Therefore, we checked the percentage of bandwidth dropped, for each scaling approach, starting with DFS and random based initial allocations to have a better comparison.

There are two reasons for dropping bandwidth requests: (1) the cloud infrastructure does not have enough resources or (2) the cloud infrastructure has enough resources, but the resources cannot satisfy the deadline constraint. We calculated the percentage of bandwidth dropped for each event as (Total of dropped bandwidth for all the scaling policies of the event / Total bandwidth requested by all the scaling policies of the event * 100). Vertical scaling had the highest percentage of bandwidth dropped: average of 49.6%. The next was migration scaling: average of 3.89%, while horizontal scaling had the lowest: average of 0.12%, making it the best scaling approach, in terms of the optimization goal of minimizing the bandwidth dropped rate.

Figure 1 shows the percentage of bandwidth dropped for the vertical scaling, over scaling events of the day. The vertical scaling is always limited by the spare computational resources of the VNF's current server. However, starting with a random initial allocation and continuing the day with vertical scaling, provided significant better results than starting with a DFS initial allocation. Since DFS follows a bin-packing strategy, it did not leave free CPU units in the servers to be used for

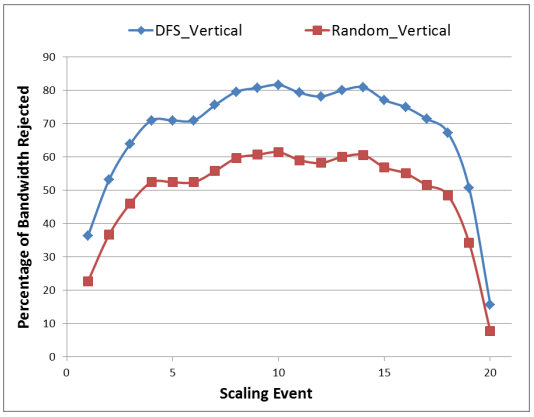

Fig. 1. Percentage of bandwidth dropped: Vertical

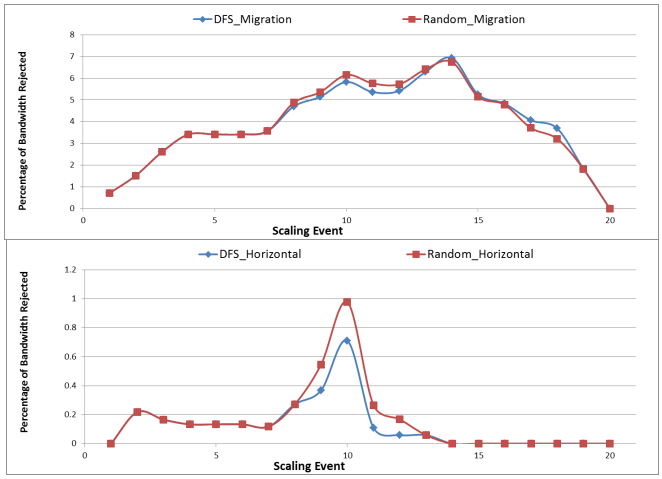

Fig. 2. Percentage of bandwidth dropped: Migration and Horizontal

future bandwidth demands. On the other hand, with a random initial allocation, servers were not fully packed, therefore there were CPU units left in servers to be used for future bandwidth demands.

Figure 2 shows the percentage of bandwidth dropped for the migration and horizontal scaling, over the scaling events of the day. Note that the X axes are in different orders of magnitude in the two plots of the figure 2 and in the plot of figure 1, making the horizontal scaling, the best minimizing bandwidth dropped rate. The first observation with the migration scaling is that, as expected, when the bandwidth demand increased over time, the migration scaling faced the problem of physical resources limitation of a server, thus, limiting the resources that can be allocated to a VNF. Therefore, the CRMP was not able to find servers to satisfy the total bandwidth demand. The second observation is that for the scaling events at the beginning of the day (until the 8th event), there was no significant difference between starting with a DFS or a random initial allocation. However, when the system gets tighter over the time, the percentage of bandwidth dropped tended to be slightly different for the two methods. For some events, starting with the DFS initial allocation gave better results while for other events, starting with a random initial allocation gave better results. In other words, a DFS or a random based initial allocation has not significant effect on average within the migration scaling. The factors affecting the migration scaling were the tightness of the system and physical resources limitations of the servers.

The main observation with the horizontal scaling is that most of the time, regardless of whether the system was tight or not, the CRMP was able to find resources to satisfy the total extra demand, and the percentage of dropped bandwidth was minimal. For the scaling events at the beginning of the day (until the 8th event), there was no significant difference between starting with a DFS or a random initial allocation. However, when the system gets tighter over the time, starting with DFS initial allocation gave better or equal results, having a low percentage of bandwidth dropped.

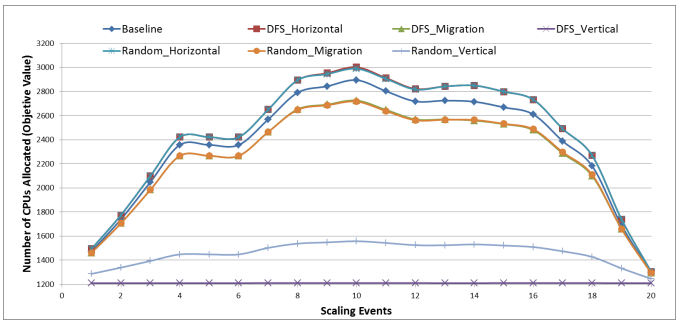

Fig. 3. Comparison of allocated CPU units

2) Allocated CPU units: The optimization goal of our algorithm was to minimize the bandwidth dropped rate of scaling requests. We can equate that value with earning and contrast that against the CPU units allocated for the VNFs to process the accepted bandwidth as a measurement of expenses. This implies an approximated measurement of profit.[2]

The number of allocated CPU units for the VNFs to process the accepted bandwidth when following different approaches are shown in Figure 3. We compared the allocated CPU units by these approaches with a base line: the minimum number of CPU units required to satisfy the total bandwidth demand of the system (for all policies) at each scaling event. This is calculated without considering the maximum capacities of servers or links, making it an over-optimistic scenario.

As described in the previous section, the vertical scaling has the highest percentage of dropped bandwidth. Therefore, it has the lowest number of CPU units allocated. However, starting with a random initial solution was better than starting with DFS. The migration scaling's number of CPU units allocated is closer to the baseline.

According to the baseline, the horizontal scaling has allo-

[2]We haven't explicitly converted accepted bandwidth requests and allocated CPU units to monetary values, but once this is done, we can calculate profit.

cated more than required. With the horizontal scaling, when a policy has to be scaled out, we assumed it as a new policy. Also, we assumed that the new policy's bandwidth request was the extra bandwidth demand (the traffic change) of the existing policy. We have observed that some VNFs in the existing policy, had enough resources to satisfy the total bandwidth demand, while some VNFs did not. As the maximum processing capacity of a VNF for a given amount of resources depends on the VNF type, the capability to handle the total bandwidth demand with already allocated resources also depends on the VNF type.

A strategy of re-using VNFs of the existing policy that could satisfy the total bandwidth is complicated. We might need to create new instances for some VNFs of the policy that can't satisfy the total bandwidth, and distribute the traffic by having workload balancers within the policy chain [28], [29]. Therefore, even though there were VNFs in the existing policy that could satisfy the total bandwidth demand with the already implemented VNF instance, we still created a new VNF instance, because the concept was to implement a new policy to satisfy the extra bandwidth demand, as it needs a single workload balancer to distribute the traffic over policy instances.

For both the migration scaling and horizontal scaling, starting with a DFS or a random initial solution did not provide significant difference in the results.

3) Changes to the current configuration: As we mentioned earlier, when scaling VNFs to satisfy the new traffic amount, there might be different conflicted objectives. When re-allocating resources to minimize the bandwidth dropped, we might also want to minimize changes to the current configurations.

We use the term "a server change", to refer to the event when the placement of a VNF changed. For the migration scale, "a server change" means the existing VNF instance has to migrated to a new server and traffic has to be re-directed to the new server. For the horizontal scale, "a server change" means, that a new VNF instance has to be instantiated and traffic has to be balanced between multiple VNF instances. When using vertical scale, there are no server changes, because scaling is done by adjusting CPU and memory metrics in the server that the VNF is currently residing.

Figure 4 shows the comparison of server changes when horizontal and migration scaling are used over the scaling events. For both horizontal and migration scaling, there was no significant effect when starting with a DFS or a random initial allocation, with respect to the number of server changes. Furthermore, horizontal scaling caused more server changes than migration scaling. This is because the horizontal scaling was accepting more bandwidth requests than the migration scaling, at the cost of changing servers. However, the consequence is that during the system adjustments, the horizontal scaling might cause more traffic loss, directly affecting earnings and profit.

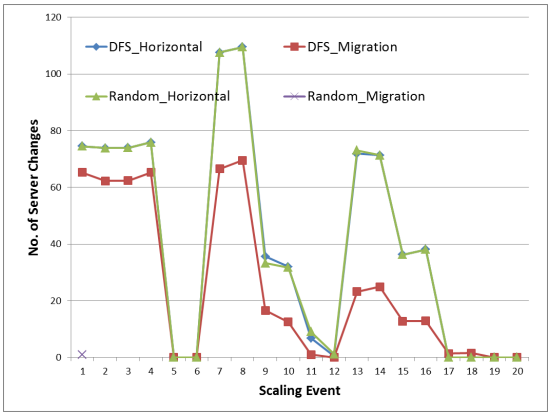

Fig. 4. Number of server changes

VI. Final Remarks

In this paper we proposed: (1) a Integer Linear Programming (ILP) model for automation of dynamic scaling of VNFs, (2) an Iterated Local Search (ILS) method that efficiently solve the problem and can be implemented in real-life and (3) a comparison on the different scaling models: vertical, migration and horizontal.

First, our results showed that For a small network with 4 servers, when trying to implement 10 VNFs, our ILS based algorithm was able to find optimal solutions within few milliseconds where the Integer Linear Programming (ILP) implementation in CPLEX [24] took several minutes. Next we explored how the optimization is effected by the different scaling approach and the optimization objectives. We compared the different characteristics of the solutions provided by scaling approaches such as accepted bandwidth ratio and resource utilization. To our knowledge, this is the first study that focuses in the effect on the optimization goals of the different scaling methods.

As future work, we are planning to explore more on automating resource management for scaling VNFs in two aspects. First, to explore the problem of monitoring the resources and determining when to scale the VNFs according to the dynamic traffic changes. Next, how to handle scaling out/in situations, where it might need VNFs migration and traffic splitting or re-direction. Specifically we are interested in how to split the traffic load between VNFs instances with minimum traffic loss and delay. We are working on a consistent hashing based session-aware load balancing algorithm.

References

[1] M. Luizelli, L. Bays, L. Buriol, M. Barcellos, and L. Gaspary, "Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions," in *IEEE IM '15*.

[2] M. Bari, S. Chowdhury, and at el, "On orchestrating virtual network functions," in *IEEE/ACM/IFIP CNSM 2015*.

[3] T. Lukovszki, M. Rost, and S. Schmid, "Its a match! near-optimal and incremental middlebox deployment," in *ACM SIGCOMM Computer Communication Review: Jan 2016*.

[4] R. Cohen, L. Lewin-Eytan, and at el, "Near optimal placement of virtual network functions," in *INFOCOMM 2015*.

[5] M. Ghaznavi, N. Shahriar, and at el, "Service function chaining simplified," in *eprint arXiv:1601.0075*.
[6] Y. Li, L. Phan, and B. T. Loo, "Network functions virtualization with soft real-time guarantees," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2016.
[7] G. Milad, K. Aimal, S. Nashid, A. Khalid, A. Reaz, and B. Raouf, "Elastic virtual network function placement," in *Cloud Networking (CloudNet), 2015 IEEE 4th International Conference on*, 2015.
[8] M. Ghaznavi, A. Khan, N. Shahriar, and at el., "Elastic virtual network function placement," in *IEEE CloudNet*, 2015.
[9] H. R. Lourenco, O. Martin, and T. Stutzle, *Iterated Local Search: Framework and Applications. In Handbook of Metaheuristics*, 2010.
[10] W. Rankothge, F. Le, A. Russo, and J. Lobo, "Experimental results on the use of genetic algorithms for scaling virtualized network functions," in *IEEE SDN/NFV 2015*.
[11] W. Rankothge, F. Le, and at el., "Data modelling for the evaluation of virtualized network functions resource allocation algorithms." in *arXiv:1702.00369*.
[12] X. Meng, V. Pappas, and at el, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *GIIS '12*.
[13] R. Mijumbi, J. Serrat, and at el, "Design and evaluation of algorithms for mapping and scheduling of virtual network functions," in *NetSoft 2015*.
[14] A. Gember, R. Grandl, A. Anand, and at el, "Stratos: Virtual middleboxes as first-class entities," *Technical Report TR1771*, 2013.
[15] S. Clayman, E. Maini, and at el, "The dynamic placement of virtual network functions," in *NOMS 2014*.
[16] W. Rankothge, J. Ma, F. Le, A. Russo, and J. Lobo, "Towards making network function virtualization a cloud computing service," in *IEEE IM 2015*.
[17] N. Bobroff, A. Kochut, and at el., "Dynamic placement of virtual machines for managing sla violations." in *IEEE IM 2007*.
[18] U. Sharma, P. Shenoy, and at el., "Kingfisher: Cost-aware elasticity in the cloud." in *Infocom 2011*.
[19] S. Bazarbayev, M. Hiltunen, and at el., "Content-based scheduling of virtual machines (vms) in the cloud." in *ICDCS 2013*.
[20] M. Hines and K. Gopalan, "Post-copy based live virtual machine migration using adaptive prepaging and dynamic self-ballooning," in *VEE 2009*.
[21] H. Liu, H. Jin, and at el., "Live migration of virtual machine based on full system trace and replay." in *HPDC 2009*.
[22] V. Mann, A. Vishnoi, and at el., "Crossroads: seamless vm mobility across data centers through software defined networking." in *NOMS 2012*.
[23] W. Rankothge, F. Le, A. Russo, and J. Lobo, "Optimizing resources allocation for virtualized network functions in a cloud center using genetic algorithms." in *IEEE TNSM 2017*.
[24] "Cplex," http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/.
[25] W. Wang, H. Chen, and X. Chen, "An availability-aware virtual machine placement approach for dynamic scaling of cloud applications," in *Conference on Autonomic and Trusted Computing*, 2012.
[26] "Xensource inc," http://www.xensource.com.
[27] "Vmware esx server," http://www.vmware.com/products/esx.
[28] J. Ma, W. Rankothge, C. Makaya, C. Morales, F. Le, and J. Lobo, "An overview of a load balancer architecture for vnf chains horizontal scaling," in *IEEE IFIP CNSM 2018*.
[29] J. Ma, W. Rankothge, C. Makaya, C. Morales, and at el, "A comprehensive study on load balancers for vnf chains horizontal scaling." in *arXiv:1810.03238*.
[30] W. Rankothge and J. Lobo, "Test data," 2016, https://github.com/windyswsw/DataForNFVSDNExperiments.
[31] J. Martins, M. Ahmed, and at el, "Clickos and the art of network function virtualization," in *NSDI '14*, 2014.
[32] D. Dietrich, A. Abujoda, and at el., "Network service embedding across multiple providers with nestor," in *IFIP Networking 2015*.
[33] P. Shoumik, L. Chang, H. Sangjin, and at el, "E2: a framework for nfv applications," in *Proceedings of the 25th Symposium on Operating Systems Principles*, 2015.
[34] "Design best practices for latency optimization," https://www.cisco.com.