Experimental Results on the use of Genetic Algorithms for Scaling Virtualized Network Functions

Windhya Rankothge
Universitat Pompeu Fabra
windhya.rankothge@upf.edu

Franck Le
IBM Research
fle@us.ibm.com

Alessandra Russo
Imperial College
a.russo@imperial.ac.uk

Jorge Lobo
ICREA-Universitat Pompeu Fabra
jorge.lobo@upf.edu

***Abstract*—Network Function Virtualization (NFV) is bringing closer the possibility to truly migrate enterprise data centers into the cloud. However, for a Cloud Service Provider to offer such services, a key question is how and when to scale out/in resources to satisfy dynamic traffic demands. In previous work [1], we have proposed a platform called Network Function Center (NFC) to study research issues related to NFV and Network Functions (NFs). In a NFC, we assume NFs to be implemented on virtual machines that can be deployed in any server in the network. In this paper we present further experiments on the use of Genetic Algorithms (GAs) for scaling out/in NFs when the traffic changes dynamically. We combined data from previous empirical analyses [2], [3] to generate NF chains and for getting traffic patterns of a day and run simulations of resource allocation decision making. We have implemented different fitness functions with GA and compared their performance when scaling out/in over time.**

I. Introduction

With Network Functions Virtualization (NFV) [4], the possibility of outsourcing enterprise Network Function (NFs) processing to the cloud has gained lot of attention. When the NFs of an enterprise are outsourced to a cloud service provider, the cloud service provider is responsible for: (1) where initial virtual NFs should be instantiated, (2) what, when and where additional virtual NFs should be instantiated to satisfy the dynamic traffic demands (scaling); and (3) how to update the network configurations to minimize latency, packet loss, and the impact impact on network performance.

For cloud resource allocations, researchers have often relied on mixed Integer Linear Programming (ILP) to optimize VM allocation and network management [5]. However, this approach can be applied only if adjustments to traffic demands are made in the order of hours [1]. In addition, we can make better resource allocations if both computing resources and network configuration are managed concurrently [6]. Although several works have considered both VM allocation and network management jointly [7], [8], most of them have assumed that only one of the two components could be modified. In particular, several proposals [5], [7], [8], [9] assume that the computing resources allocation can be updated but not the network configuration. We can identify (at least) two obstacles behind these assumptions. One has been the difficulty of making dynamic updates to the network components. The second one is that the complexity of the optimization model increases by the addition of the parameters corresponding to the network components. Software Defined Networks (SDN) naturally arises as a tool to overcome the first obstacle, but we need to look for suitable approximations for the optimization. As such, we have been developing an experimental platform that we call Network Function Center (NFC) to study issues related to NFV and NFs management as a service. In particular, we have proposed a new Genetic Programming based resource allocation algorithm [1], and we have assumed a Software-Defined/OpenFlow infrastructure [10] to have programmatic control over the traffic flow and easy reconfiguration of the physical network.

Our initial performance evaluation has shown encouraging results [1]. However our initial experiments have several limitations. First we evaluated the performance of our proposed resource allocation algorithm under a fixed network architecture, k-fat trees. Second, we ran all our experiments under purely synthetic data. And third, we did not fully analyse how fast we could generate a new NFC configuration given a current configuration and information about new traffic demands, nor evaluated the quality of the configurations when we repeat the procedure over time. This is perhaps the largest limitation given that our main goal is to perform resource allocations to satisfy dynamic traffic demands over time and given that we are not working with optimal but approximated solutions. Hence it is crucial to verify that the sub-optimality does not deteriorate over time.

Extending our previous work [1], this paper explores the results of the resource allocation algorithms over a full day of traffic based on more realistic traffic data under three different network architectures. We assume small data centers having 64 servers with : (1) a k-fat tree architecture [11], (2) a BCube architecture [12] and (3) a VL2 architecture [13]. We focus on Genetic Programming based resource allocation algorithms with fitness function that in addition to reducing server and link utilization, also minimize the number of changes in server allocation and links configuration. The reason is that configuration changes (e.g., moving NFs) can cause performance degradation. Public traffic data specifically referring to NFs chains is not readily available. Hence, we have combined the data about the use of NFs in different kinds of enterprises from [2] and the HTTP traffic data observed by an ISP [3] to create, although limited, a traffic model based on real data. We study the quality of the solutions provided by such fitness functions over time. We compare the results with those of a fitness function that solely focuses on reducing server and link utilization. The concern was that whether trying to also minimize the number of changes in server and link configurations would lead to local optima at each time point, and gradually diverge over time from the global optimal solution. However, our experimental results proved otherwise: the difference between the quality of an optimization for scaling that minimizes changes and the global optimization solution, does not increase over time. In other words, the quality of the scaling optimization solutions does not degrade over time. In addition, we consider a fitness function that solely tries to minimize server and link changes. Such solutions should minimally disturb the traffic (e.g., packet drops, latency) and can be used as a baseline for comparison. We found that the proposed fitness function that simultaneously tries to minimize server and link changes as well as minimizing servers usage and links congestion provides solutions where the the number of changes in the network required for scaling is very close to those required by the fitness function that solely tries to minimize server and link changes. In other words, the approach produces solutions that are close to the optimal in terms of minimally disturbing the network for scaling. We explore the fitness functions behaviour for different data center architectures over several days and our experimental results show that the fitness function values and their changes, highly depend on the network architecture, specially on the number of links and paths of the network.

The rest of the paper is organized as follows. Section II gives a brief description of our experimental NFV platform and its management system. Section III describes the implementation of GA based NFs placement and dynamic scaling out/in algorithms. Section IV presents the evaluation set-up and Section V shows the results of the evaluation. Related work is briefly described in Section VI. Our final remarks can be found in Section VII.

II. Experimental Platform

We are developing a Network Function Center (NFC) as an experimental platform [1], to study research issues related to NFV and NFs. We assume that NFC will deliver virtualized NFs to clients on a subscription basis. To receive services from our NFC, a client needs to provide the following two specifications: (1) types of required NFs and interconnectivity between them (policy chain) and (2) initial expected traffic load to be processed by these NFs.

Figure 1 represents a snapshot of a NFC. It shows the placement of NFs that implements the two policy chains in Table 1. Table 2 shows the physical sequences of switches and NFs the client's traffic will go through. Client1 wants his traffic coming from 10.1.0.0/24 to any destination to go through the policy chain of Firewall-IDS-Proxy NFs. To satisfy his request, a firewall and a IDS are implemented on two VMs at Server1 and a Proxy is implemented on a VM at Server2.

The NFC Management System is built around five key modules: (1) Resource Manager, (2) Topology Manager, (3) Flow Manager, (4) Elasticity Manager and (5) Rules Generator.

Once a new client request is submitted, the Resource Manager takes decisions on the placement of NFs and paths for the client's traffic to follow inside the NFC. The Resource Manager is also called by the Elasticity Manager. The Elasticity Manager monitors the resources utilization. The Elasticity

Table 1

Traffic Flow	Policy	Expected Traffic
10.1.0.0/24 - *, HTTP	Firewall – IDS - Proxy	300 GB
20.1.0.0/24 – 172.0.0.0/24, HTTP	IDS - Proxy	100 GB

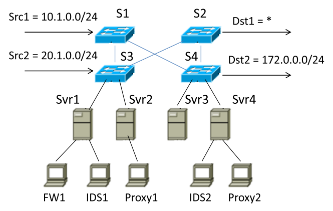

Table 2

Policy	Physical Sequence
Firewall – IDS - Proxy	S1-S3-Svr1-FW1-Svr1-IDS1-Svr1-S3-Svr2-Proxy1-Svr2-S3-S2
IDS - Proxy	S3-S2-S4-Svr4-IDS2-Svr4-Proxy2-Svr4-S4

Fig. 1: NFC Snapshot

Manager takes decisions on when to increase/decrease the capacity of the instances of NFs and paths for the traffic flows. The Resource Manager then determines the reallocation of server and network resources to satisfy the demands. The Topology Manager, Flow Manager, and Rules Generator configure the network according to decisions taken by the Resource Manager and Elasticity Manager. More details of the architecture of our NFC can be found in [1].

III. Resource Manager Module

The Resource Manager has two main responsibilities:

(1) New function provisioning: upon receipt of a new set of policies, the Resource Manager takes into account the physical network, servers constraints, and already allocated resources, to identify the resources where to instantiate the new function.

(2) Scaling out/in: upon receiving requests from the Elasticity Manager, the Resource Manager decides the reallocation of resources in order to the satisfy the traffic demand changes.

Integer Linear Programming (ILP) optimization has been a popular technique for VM allocation. However, our experimental results with ILP show [1] that using ILP to find an optimal configuration can take a long time even for a small number of NFs. So as we have explained in [1], we have explored finding approximations by means of finding the *best fitted* solution according to a Genetic Algorithmic model of the problem that explores a fixed amount of generations. Genetic Algorithm (GA)s are a part of evolutionary computing and were introduced as a computational analogy of adaptive systems [14].

The GAs [14]:

1) Randomly generate an initial population F(0) with n full solutions f
2) Compute and save the fitness u(f) for each individual full solution f in the current population F(t)
3) Generate F(t+1) by selecting i full solutions from F(t)
4) Produce offspring by applying genetic operators to population F(t+1)
5) Repeat step 2 until satisfying solution is obtained.

Following the GA terminology, a possible configuration state (represented by servers and paths assignments) of the NFC is considered as a full solution f, if it is an allocation of resources for all the policies in the system. The population F(t) consists of n full solutions which represents different possible configuration states for the NFC. If there are m policies in the NFC, then each full solution contains m partial solutions, each partial solution representing the allocation of resources (i.e., servers and paths) for a policy.

$$\begin{aligned} F1 = \quad & w_1 \tfrac{1}{M}.T_s + w_2 \tfrac{1}{L}.U_l + w_3(1 - \tfrac{1}{L}.T_l) \\ & + w_4 \tfrac{1}{M}.C_s + w_5 \tfrac{1}{L}.C_l \end{aligned}$$

TABLE I: Fitness Function

M	Total no. of servers
T_s	No. of servers used
L	Total no. of links
T_l	No. of links used
U_l	Avg. % of total links capacity used
C_s	Total servers changed from previous state
C_l	Total links changed from previous state
w_1 to w_5	Weighting factors

TABLE II: Parameters used in fitness functions

For new services provisioning, the Resource Manager uses network's traffic, topology data, server constraints and client requirements as inputs. In step 1, the Resource Manager generates the initial population F(t). It performs a selection (we have used Depth First Search (DFS)) for the initial assignment of NFs and paths for each new policy request. The configuration state (NFs and paths) that the Resource Manager comes up with after the DFS for a new policy request is considered a partial solution that combined with the partial solutions of each of the existing policies form a full solution. After the initial population is generated, the fitness function ($F1$) given in Table I with weights $w_4 = w_5 = 0$ is used to measure how good a full solution is (step 2). $F1$ can take into account: servers capacity, links capacity, number of links not used and number of servers used with respect to the total physical usage of the network and network resources available. As we are trying to maximize the server and network utilization, fittest solutions are those for which the function returns the smallest value. So full solutions that return smaller values are preferred and in step 3 they are selected as the best solutions for the next population generation. In step 4, the Resource Manager performs mutations and crossovers for randomly selected partial solutions of a full solution and generates a new full solution. We have considered two types of genetic operators to produce mutations:(1) Re-placement where we try to place the NF in a different server and (2) Re-wiring where we try to find a different path between given two NFs. For crossovers, first we select two random full solutions and a random partial solution from each selected full solution. Then we try to check whether the configuration given in the first partial solution can be applied to the second partial solution and vise versa. If so, then the configurations of the partial solutions will be changed accordingly. The newly generated full solution is added to the existing set of full solutions, which is known as the current population. This process is continued for a fixed x number of generations. In the final generation, the full solution with the best fitness value is selected as the configuration for the new policy implementation.

For the optimization related to scaling, the procedure is different since we care about changes. When the Elasticity Manager decides that a NF or a path has to be scaled out/in (i.e., a new VM needs to be created for the NF, or an existing VM can be removed), the Resource Manager starts with the current state and performs an initial selection using a DFS for the re-assignment of resources (new servers and paths) of the set of NFs and paths that are scaling. The partial solutions relevant to the scaling are modified according to the results of the DFS. The fitness function $F1$ with none zero values for at least w_4 and w_5 is used to measure how good a full solution is. Parameters related to w_4 and w_5 represent the changes to the current system. While trying to maximize the server and network utilization, we want to minimize the changes to the current system because drastic re-arrangements of the system configuration will cause unacceptable deterioration of performances during the transition time. In contrast to the "global optimization" performed during the initial resource allocation process, when scaling out/in, the mutations and crossovers are carried out only to the partial solutions which were changed because of the scaling out/in. The process is continued for x number of generations and the best full solution is selected as the configuration for re-assignment of the policy. The question then is, what is the effect of doing local resource allocation optimizations that also minimizes changes when compared with an optimization of resource allocation that is done globally without regard to changes? Does the allocation of resources drifts away from an optimal allocation? As the experiments in the following sections show this will not be the case.

IV. Evaluation Data

For the evaluation of the Resource Manager, we needed data on: (1) potential NFs chains (policies), (2) traffic flows passing through these NFs chains and (3) different data center architectures for NFC. In the following section we describe the data we used and assumptions we have made to conduct the experiments.

A. *Policies and traffic flows passing through them*

We have combined a data set from a study about physical middle-boxes in enterprises [2] to generate our policies and a data set from a study about HTTP traffic [3] on internet to generate our traffic, since there are no publicly available real data sets on NF chains and traffic that might pass through them.

The policies used in the following sets of experiments are generated based on a study about physical middle-boxes used in enterprise networks [2]. This paper includes figures about types of enterprise networks, number and types of middle-boxes used in them. Following [2], we have assumed large enterprise networks with an average of 100 NFs and with each policy having from 2 to 7 NFs in a chain. The number of NFs in a policy follows a truncated power-low distribution with exponent 2, minimum 2 and maximum 7.

The traffic load that each client is expecting is modelled according to the applications [15]. We consider web based applications and for the traffic, we rely on empirical data from previous studies [3]. The data set includes an HTTP traffic breakdown of 30,000 users for a day which is measured at three different vantage points of an Italian ISP. The traffic breakdown reports traffic for every 2 hours. We focus on the traffic statistics of Megaupload, LeaseWeb, Level3 and Limelight for our experiments.

In a data center, traffic changes happen throughout the day and according to the amount of these changes, the NFs should be scaled out/in to satisfy the dynamic demands. A limitation of the HTTP traffic data we are using is that, information was collected at every two hours. So the first challenge is interpreting the pattern of traffic change over two hours. Other studies (e.g., [16]) show that traffic changes on usual days happen gradually over time. From times when traffic may increase significantly, changes may still increase gradually over 15 minutes time periods [17]. As such, although sudden traffic changes may occur within few minutes, we have assumed a uniform traffic increase/decrease over the 2 hours time intervals. To reflect scaling requirements of all situations, we spread the increase/decrease of number of NFs (needed for the full 2 hour traffic change) over 2 hours and increase/decrease the capacity one NF at a time.

The second challenge is identifying the policies affected by each enterprise traffic change. For each enterprise we have x number of policies generated and each policy has a unique traffic flow passing through its NFs. When there is a change in the total traffic for that enterprise, it is very unlikely that traffic passing through all the policies of that enterprise contributed to the traffic change. So we select randomly a subset of policies from that enterprise, as the policies affected by the traffic change.

The third challenge is deciding which NF from each policy, needs to be scaled out/in to satisfy the new traffic demands. [9] shows that in general no two NFs will be simultaneously and equally bottlenecked and scaling one NF in the policy at a time is the best strategy. Hence assuming the conditions in [9], we randomly select a NF from each policy as the bottlenecked NF for which the resource allocation needs to be increase/decrease.

The fourth challenge is, from the identified NF instance to scale, how many instances we should add/remove to satisfy the new traffic demand. Here, we are making an assumption: the traffic flowing through the NF instance is proportional to the capacity of the NF instance and it is the same for all types of NFs. [18] shows that if we add more than one instance at a time, we are usually adding more than what is needed and wasting resources. So we calculated a traffic change threshold to find how many instances we should add/remove to accommodate traffic change and add/remove one instance at a time. This resulted in 36 significant events over the 24 hours of traffic data. These are events where either the allocation of resources for at least one NF needs to be increased or reduced, or the traffic in at least one link needs to be modified.

B. Data center architectures for NFC

We evaluated the performances of the resource allocation algorithm, assuming three different data center network architectures for NFC: (1) k fat tree, (2) VL2 and (3) BCube shown in the Figure 2. We have assumed architectures of a small data center having 64 servers.

A typical k-ary fat-tree network [11] has three layers: a core layer, an aggregation layer and a Top-of-Rack (ToR) layer. It consists of $(k/2)^2$ core layer switches and k pods of k switches, half of them aggregation switches and the other half ToR. Each switch in pod has k ports. The ToR switches are at the bottom of the pod, and the aggregation switches in the middle. In one pod, each ToR switch is connected to every aggregation switch and $(k/2)$ servers. Each aggregation switch connects to $(k/2)^2$ switches on the core layer. We have used a 4 fat-tree architecture which has 20 switches: 4 pods of 4 switches, each with 8 servers in each ToR switch and 4 switches in the core layer. The network consists of 96 links and 35776 paths connecting all source destination server pairs with maximum number of hops for a path of 6.

The VL2 architecture [13] shares many features with an k-ary fat-tree architecture, but the main difference is the core tier and aggregation tier form a Clos topology [19], i.e., the aggregation switches are connected with the core ones by forming a complete bipartite graph. We have used a VL2 architecture with 12 switches. The network consists of 96 links and 33760 paths connecting all source destination server pairs with maximum number of hops for a path of 6.

In the BCube architecture [12], servers are considered part of the network infrastructure, i.e., they forward packets on behalf of other servers. A BCube is a recursively defined structure. At level 0, $BCube_0$ consists of n servers that connect together with a n-port switch. A $BCube_k$ consists of n $BCube_{(k-1)}$ connected with n^k n-port switches. We have used a $BCube_1$ architecture where there are 8 $BCube_0$s, each connected to 8 switches in the next level switches and form the $BCube_1$. Each s server of $BCube_0$s are connected to switch s of $BCube_1$. The network consists of 128 links and 7168 paths connecting all source destination server pairs with maximum number of hops for a path of 4.

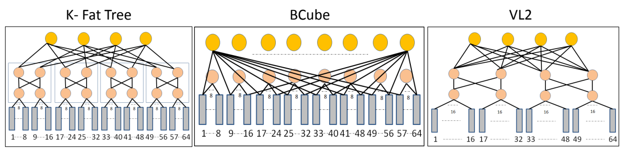

Fig. 2: Architectures used for NFC

V. Evaluation Results

As described in Section III, once a new client request is submitted, the Resource Manager takes decisions on the initial placement of NFs and paths for the traffic. Then, after the initial configuration, according to the dynamic changes of traffic over time, the Resource Manager responds to the requests from the Elasticity Manager to scale the resources and decide a new set of NFs assignments and paths for existing traffic flow. These two activities are implemented using a GA with the fitness function given in Section III. We explore the evolution of the solutions deriving from the GA fitness function over a full day of traffic. In this section, we describe the results of the experiments we conducted to compare and understand the performances of different fitness functions when continuously scaling out/in.

For VM allocation, an ILP can give us the best optimal configuration solution. However, as we showed in previous work [1], ILP takes a long time to find an optimal configuration even for a small number of NFs. And, although GA may not provide the optimal solution, GA approach can compute configurations two to three orders of magnitude faster than ILP [1].

Although global optimization may provide better resource allocations, the solutions may require drastic re-arrangements of the current configurations, hence making them impractical in real scenarios. However, we can use this method to provide us with a baseline of how the local optimization behaves. We conducted experiments computing the results of both global and local optimizations to compare their performances with respect to how well resources are allocated. The comparison is done by comparing the value of the fitness function of the implemented allocations assuming that changes don't count (i.e., $w_4 = w_5 = 0$). For local optimization, we have used different weights for parameters in the fitness function to find allocation. Due to space limitation we will show the results for 2 different usages of fitness function as specified in Table 3. We call global optimization the baseline, i.e. the minimization of the parameters relevant to server and links usage. For local optimization we show the 2 limited cases: (1) all parameters are considered and (2) only parameters relevant to changes are used. Since the second case tries to minimize the changes to the servers and links, it represents the scaling solutions that in theory minimally disturb the traffic (e.g., packet drops, latency).

Fitness Function Usage	w1	w2	w3	w4	w5
Global Optimization	1	1	1	0	0
Local Optimization 1	1	1	1	1	1
Local Optimization 2	0	0	0	1	1

TABLE III: Different usages of fitness function

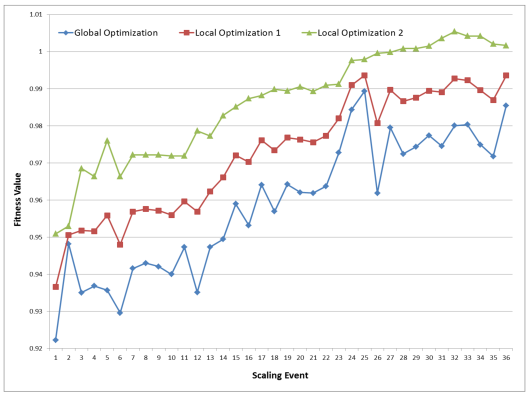

Fig. 3: Fitness Value Comparison

A. *Experiments with k-fat tree architecture*

For the first set of experiments we have used the k-fat tree as the architecture for NFC. Figure 3 shows a comparison of fitness values of the baseline fitness function obtained from configuration solutions given by (1) the global optimization, (2) the local optimization 1 and (3) the local optimization 2 for dynamic traffic changes over the 36 events that Elasticity Manager will report. As expected the global optimization produces better resource allocations than the other two, but the important observation is that the figure clearly shows that optimizations (2) and (3), if we smooth the curves, will follow essentially the same behavior (module a translation in the y axis) that the behavior of the baseline (1).

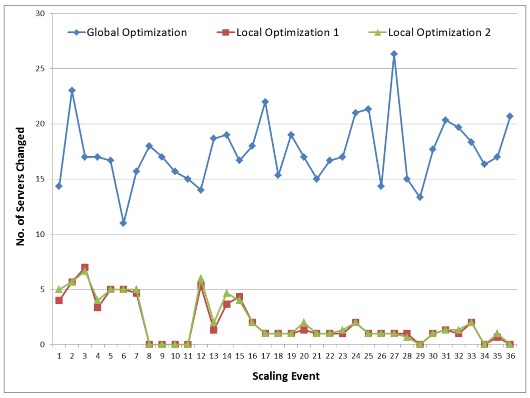

Fig. 4: Server Changes Comparison

Figure 4 shows a comparison of server changes needed in the configuration solutions given by (1) global optimization, (2) local optimization 1 and (3) local optimization 2 after processing each event. Again, as expected, the global optimization is the one causing the largest number of changes. On the other hand, since the local optimizations allows genetic operations only on the partial solutions that are scaling, the solutions given by the local optimization has fewer server changes from their previous configuration. The interesting part is that both local optimization methods have the same number of server changes most of the time, making the two methods essentially the same. We have observed that these server changes are not necessarily caused by the genetic operations, but rather they are the unavoidable changes due to the scaling requirements.

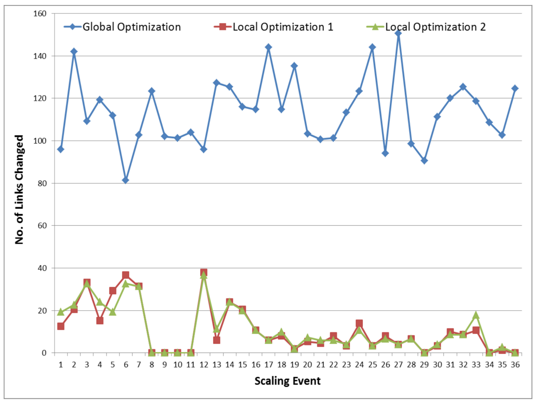

Fig. 5: Links Changes Comparison

Figure 5 shows a similar comparison to Figure 4 but for links changes needed in the configuration solutions given by (1) global optimization, (2) local optimization 1 and (3) local optimization 2 from their previous configuration to current configuration in the solution. Following the pattern in server changes, the solutions given by the global optimization has most links changes from their previous configuration. Both local optimization methods have the same number of links changes most of the time.

When it comes to the number of changes in servers and links, there is no much difference between local optimization 1 and local optimization 2 methods. But as we have shown earlier, local optimization 1 (which additionally minimizes usage of servers and links congestion) gives better fitness values than local optimization 2. Therefore, local optimization 1 leads to a better server and network resources utilization without incurring in a larger number of changes. In the rest of the experiments, we have used local optimisation 1 as the method for local optimization, as it performs better than local optimization 2.

B. Experiments with different architectures for NFC

After the initial set of experiments conducted with k-fat tree architecture, we explored the fitness function behaviour with global optimization and local optimization over different data center architectures that we have described in the section IV-B. In the same time, we wanted to check how the fitness function would behave over several days, so we repeated the data for single day for several times. Because of the space limitations we have included the results for 2 repetitive days.

Figure 6 shows the fitness values for k-fat tree, BCube and VL2 architectures for 2 repetitive days. As expected for all three topologies, the global optimization produces better resource allocations than the local optimization. Also they follow essentially the same behaviour (module a translation in space) of the baseline: global optimization. The fitness values grow during most of the scaling events of each day, and this is because, in the traffic model we are using, the traffic is increasing until late night of each day.

We observed that for each architecture's fitness values are effected by different parameters of the fitness function. For all three architectures, the number of servers used are very much similar while the number of links used and the links utilization make the difference in the fitness values.

In the BCube architecture, local optimization always uses fewer links than global optimization and this creates the difference between two optimizations. Since we are trying to minimize number of server and links changes in the local optimization, it hesitates to use more links where the global optimization freely use more links over time. So the solutions given by the global optimization are less congested than the solutions given by the local optimization.

In the VL2 and k-fat tree architectures, for both the local optimization and the global optimization, the number of links used is similar while the link utilization makes the difference. When comparing the fitness values increases for each day, the VL2 architecture's fitness values for the local optimization increase fast with respect to k-fat tree and BCube. This is because, in the VL2 architecture servers are more compact and it has fewer paths between servers inside the same pod. This makes the links more congested and when traffic is increasing, links utilization also increases fast.

The k-fat tree architecture has more smooth effect on the parameters of the fitness function. Since it has more paths and servers are not compact, it tries to use more links and make the links less congested.

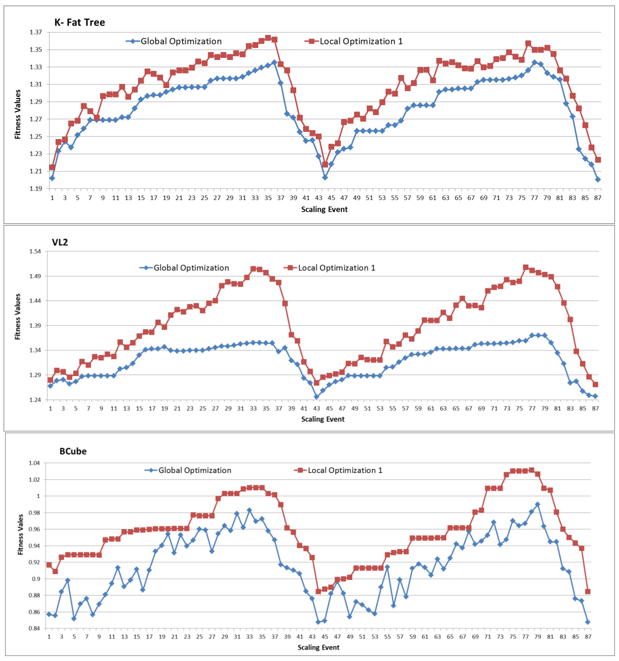

Fig. 6: Fitness Values Comparison for Two Repetitive Days

VI. Related Work

Initial work on the VM placement problem assumed that VMs are assigned static shares of servers (CPU and/or memory). The placement of VMs onto servers was related to the vector bin packing problem in [20] and heuristic algorithms were discussed in [21]. Now a days, provisioning requests from cloud users involve sets of VMs. So the placement is constrained by resource requirements and placement constraints indicated by the cloud user [22], [23]. Bandwidth allocation for communicating VMs have been modelled as a Stochastic Bin Packing problem [24] and a Min Cut Ratio-aware problem [25]. [5] uses an ILP apparoach, and takes in the order of minutes to decide the placement of 1024 VMs in the data center of 16 servers. [8] argues that it is important to optimize the placement of VMs and routing between VMs jointly. They have considered the VM placement as deciding the location of a VM each time a request is received using a Markov approximation technique. [7] focuses on network interface of machines as the network resource to optimize with the server resources. [26] proposes a heuristic based approach for initial NFs placement and chaining problem with the goal of minimizing number of NFs instances used in the cloud.

Following the initial placement, VMs can be rescaled as demanded by the applications and agreements made with the clients. [27] proposes a fuzzy-logic based controller, [28] formulates multiple-knapsack problem in which the objectives are to maximize the satisfied demand for the collection of VMs and [29] brings a decentralized solution for VM placement, using a round-based gossip protocol. AGILE [30] proposes a distributed resource scaling system for IaaS clouds and uses wavelets to provide medium-term performance predictions.

VII. Final Remarks

In this paper, we have presented a summary of the GA based resource allocation algorithm which we proposed in [1] and going further, explored the evolution of the algorithm, over a full day traffic patterns based on more realistic data.

We evaluated the behaviour of the resource allocation algorithm when scaling out/in, over different fitness functions and compared their performances. Even though for these experiments, we have only considered the number of servers used, links used, links congestion, number of server changes and links changes in the fitness, we can change the fitness function easily and add different factors to be considered for the optimization. In fact we can add parameters to the fitness function such as traffic lost, delay, cost of NFs software license [26], power consumption etc. The biggest advantage of using GA is, since the fitness function does not need to be linear, we can introduce parameters which do not have linear dependencies. So in the future we are planing to explore more on factors that effect the NFC and evaluate fitness functions more comprehensively.

We have evaluated the behaviour of the resource allocation algorithm over two days assuming three different architectures for NFC. We have observed that the architecture of the NFC affects the fitness function heavily and therefore the parameters and weights of those parameters in the fitness function should be defined based on the architectures.

We must point out that, the traffic model we have used here is limited to HTTP traffic and we have made many assumptions with regard to deciding when to scale and how much to scale. Also we have only looked at the NFs that are TCP/IP based, but there are many other types of NFs. In particular, NFs in the telecom networks are very different from the ones reported in [2] and traffic going through a telecom network can pass through more than 20 different NFs. Hence, building a more realistic model is an open challenge, that needs to be addressed.

Acknowledgment

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

References

[1] W. Rankothge, J. Ma, F. Le, A. Russo, and J. Lobo, "Towards making network function virtualization a cloud computing service," in *IM 2015*.

[2] J. Sherry, S. Hasan, C. Scott, and at el, "Making middleboxes someone elses problem: network processing as a cloud service," in *ACM SIGCOMM'12*.

[3] M. M. Vinicius G., Alessandro F. and at el., "Uncovering the big players of the web," in *ICTMA 2012*.

[4] ETSI, "Network functions virtualisation white paper," *SDN and OpenFlow World Congress*, 2013.

[5] X. Meng, V. Pappas, and at el, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *GIIS'12*.

[6] S. Jain, A. Kumar, S. Mandal, J. Ong, and at el, "B4: Experience with a globally-deployed software defined wan," in *ACM SIGCOMM'13*.

[7] F. Wuhib, R. Yanggratoke, and at el, "Allocating compute and network resources under management objectives in large-scale clouds," in *JNSM 2013*.

[8] J. Jiang, T. Lan, S. Ha, M. Chen, and at el, "Joint vm placement and routing for data center traffic engineering," in *INFOCOM'12*.

[9] A. Gember, R. Grandl, A. Anand, and at el, "Stratos: Virtual middleboxes as first-class entities," *TR1771*, 2013.

[10] "Openflow 1.4 specifications," https://www.opennetworking.org/sdn-resources/onf-specifications/openflow.

[11] C. E. Leiserson, "Fat-trees: universal networks for hardware-efficient supercomputing," in *IEEE Transactions on Computers*, 1999.

[12] C. Guo, G. Lu, D. Li, and at el, "Bcube: a high performance, server-centric network architecture for modular data centers," in *ACM SIGCOMM 2009*.

[13] A. Greenberg, J. R. Hamilton, N. Jain, and at el, "Vl2: a scalable and flexible data center network," in *ACM SIGCOMM 2009*.

[14] M. Melanie, *An Introduction to Genetic Algorithms*, 1999.

[15] S. Gebert, R. Pries, and at el, "Internet access traffic measurement and analysis," in *ICTMA 2012*.

[16] S. Kandula, S. Sengupta, and at el., "The nature of data center traffic: Measurement and analysis," in *ACM SIGCOMM Internet measurements*, 2009.

[17] Y. Tarui, "Analyzing impact of major social events on internet exchange traffic," 2009.

[18] X. C. Wenting Wang, Haopeng Chen, "An availability aware virtual machine placement approach for dynamic scaling of cloud applications," in *ICATC 2012*.

[19] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*, 2003.

[20] G. Jung, Joshi, and at el., "Generating adaptation policies for multi-tier applications in consolidated server environments," in *ICAC '08*.

[21] R. Panigrahy, K. Talwar, and at el, "Heuristics for vector bin packing." in *Microsoft Research (2011)*.

[22] Z. A. Qazi, C.-C. Tu, L. Chiang, and at el, "Simple-fying middlebox policy enforcement using sdn," in *ACM SIGCOMM'13*.

[23] L. Shi, B. Butler, Botvich, and at el, "Provisioning of requests for virtual machine sets with placement constraints in iaas clouds." in *IM 2013*.

[24] M. Wang, X. Meng, and L. Zhang, "Consolidating virtual machines with dynamic band- width demand in data centers," in *Infocom 2011*.

[25] V. Shrivastava, P. Zerfos, K. Lee, Jamjoom, and at el, "Application aware virtual machine migration in data centers," in *Infocom 2011*.

[26] 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 *IM 2015*.

[27] D. Gmach, S. Krompass, Scholz, and at el, "Adaptive quality of service management for enterprise services," in *ACM Transactions:Web 2*, 2008.

[28] C. Tang, M. Steinder, M. Spreitzer, and G. Paci
ci, "A scalable application placement con- troller for enterprise data centers," in *International conference on World Wide Web*, 2007.

[29] F. Wuhib and R. a. e. Stadler, "Gossip-based resource management for cloud environments," in *CNSM 2010*.

[30] H. Nguyen, Z. Shen, X. Gu, and et el, "Agile: elastic distributed resource scaling for infrastructure-as-a-service," in *ICAC '13*.