Multi-Cloud Distribution of Virtual Functions and Dynamic Service Deployment: OpenADN Perspective

Deval Bhamare
Qatar University,
Doha, Qatar
devalb@qu.edu.qa

Raj Jain
Washington University,
St. Louis, USA
jain@wustl.edu

Mohammed Samaka
Qatar University,
Doha, Qatar
samaka.m@qu.edu.qa

Gabor Vaszkun
Washington University,
St. Louis, USA
vaszkun@gmail.com

Aiman Erbad
Qatar University,
Doha, Qatar
aerbad@qu.edu.qa

Abstract: Network Function Virtualization (NFV) and Service Chaining (SC) are novel service deployment approaches in the contemporary cloud environments for increased flexibility and cost efficiency to the Application Service Providers and Network Providers. However, NFV and SC are still new and evolving topics. Optimized placement of these virtual functions is necessary for acceptable latency to the end-users. In this work we consider the problem of optimal Virtual Function (VF) placement in a multi-cloud environment to satisfy the client demands so that the total response time is minimized. In addition we consider the problem of dynamic service deployment for OpenADN, a novel multi-cloud application delivery platform. [1]

Keywords — Virtual Function Distribution, OpenADN, Multi-Cloud, Application Delivery, Inter-Cloud, Optimal Placement, Network Function Virtualization, Service Chaining.

I. Introduction

OpenADN is a novel approach to facilitate multi-cloud service deployment and application delivery by extending the concept of control and data plane separation proposed by the "*Software-Defined Networking*" (SDN) architecture [1, 2]. Cloud management platforms such as OpenStack and OpenDayLight consider the same problem from single-cloud perspective. These platforms need to be complemented by the sophisticated algorithms for the selection of sites to deploy Network Virtual Functions (NVFs or just VFs) and dynamic VF allocation on the already deployed VMs for the optimal performance. Network Function Virtualization (NFV) is an enabler for the Network Service Chaining (NSC) in the recent years which allows network Services to be deployed at software level contrary to the ad-hoc hardware implementation [19, 23]. The problem of VF placement has been widely considered in the literature with the focus primarily on a single-cloud environment.

If one considers the same problem from the multi-cloud perspective, new dimensions get introduced to the problem that need to be addressed. For example, one needs to consider a 3-dimensional (3-D) model for service requirements comprising computational capacity, storage capacity as well as network capacity. On contrary, 2-dimensional model has been considered for intra-cloud solution, where only computational capacity and storage capacity are considered. This is justified since network capacity is not a limiting factor for a single-cloud environment. In this work, we revisit the problem to solve it in multi-cloud environment.

A good example of multi-cloud application deployment is the "*Network Function Virtualization*" (NFV) being planned at ETSI [19]. NFV allows Internet Service Providers (ISPs) to implement key function modules, such as, BRAS (Broadband Remote Access Server), IMS (Internet Multimedia System), etc. in virtual machines in a cloud environment. One key problem in the NFV implementation is that of connecting various virtual network functions (VNFs). This is called "Service Chaining." For VNFs located inside a single cloud, this consists of programming the data center network so that the traffic flows through the various VMs according to the policies of the tenant. The network links in the data center can be programmed accordingly. Similarly, for VNFs located in different cloud data centers, the tenants would like their traffic to be handled according to their policies. Since the WAN link capacities are extremely limited and expensive, the available capacity often dictates the VNF placement to a subset of available clouds in order to meet the strict throughput and delay guarantees. The same problem is faced by non-ISP multi-cloud applications and is called service chaining of virtual functions (VFs).

From the *Application Service Provider* (ASP) perspective, an application is a set of interdependent services forming a "*workflow*". ASPs need to start multiple instance of a single workflow depending on the client demands. ASPs confront a problem while optimally placing the workflow instances considering the user demand density across multiple autonomous systems in various regions and distribution of the data-centers as potential deployment sites. Since these workflows are eventually mapped to the VFs, we will be referring the problem as a VF placement problem. In this work we propose a scheme for optimal distribution of VFs in multi-cloud environment and allocation of the clients to the respective service chains so as to minimize total response time to the clients. We also propose a heuristic approach for the dynamic VF allocation to the hosts in multi-cloud environment. We then compare our proposed heuristic against standard Max-Min approach.

[1] This work has been supported under the grant ID NPRP 6 - 901 - 2 - 370 for the project entitled "Middleware Architecture for Cloud Based Services Using Software Defined Networking (SDN)", which is funded by the Qatar National Research Fund (QNRF). The statements made herein are solely the responsibility of the authors.

II. Prior Work

There has been some recent work in the similar domain. For example, Bakiras presents a constrained mirror placement problem and approximate server selection problem in the context of the web-servers [4]. Laoutaris, et al, consider a similar problem but from the perspective of optimizing storage capacity only [13]. Tang and Xu try to address a similar problem form the QoS perspective in [3]. Work by Seshan, et al. propose a server selection scheme based on end-to-end performance measurements collected from clients in the same network [14]. Guyton and Schwartz use shortest distance method between clients and servers based on the routing tables [7].

We try to minimize the total latency (delay) for the user demands while total number of VFs to be deployed is given as an input. In addition, we are proposing a heuristic for dynamic service deployment on geographically disparate hosts with varying set of capacities, by considering a 3-dimensional (3-D) capacity model. It is to be noted that we are focusing only on transmission delays in the network for the current work and other delays such as queuing delay or propagation delay are not considered. We aim to extend the proposed model in our next work.

We aim to complement already proposed schemes to accommodate a new dimension which are introduced due to multi-cloud environment with dynamic client requests arrivals. Randles et al. proposes an "*Active Clustering*" approach in [16] and Li and Xu, propose a scheme based on "*Ant Colony Optimization"* for load balancing in clouds [17]. In addition, Wu proposes hierarchical approach to achieve load balancing, especially from single cloud perspective [18]. We aim to minimize the total number of VFs to satisfy dynamic client demands as well as minimize the network, storage and computational resources in multi-cloud environment.

Rest of the paper is organized as follows. In Section III we present an optimization model for the optimal placement of the workflows and their VFs so as to minimize the total response time to the clients spread across multiple Autonomous Systems (ASs) or regions. Section IV represents a novel heuristic approach in order to deploy the VFs dynamically on the already chosen set of hosts so as to minimize the total computational, storage and network resources. In Section V we describe the simulation setup and present the simulation results for the evaluation of the proposed optimization model and the heuristic. Section VI concludes the paper.

III. Optimal Distribution of Virtual Functions

In this section, we formulate an optimization model to deploy the workflows on the VFs and assign client requests to these workflows so as to minimize the response time (latency) to the clients. Let $G = \{V, E\}$ be a graph to represent the network in consideration where V is a set of nodes representing the clusters or ASs in the network and E be set of the edges such that $E \subseteq V \times V$. The Virtual Functions (VFs) of the workflows will be deployed per cluster which will be picked from the set of vertices V.

Total number of such instances to be deployed, λ, is given as an input to the optimization model. We vary this number from some minimum threshold (λ_{min}) till maximum threshold (λ_{max}) and observe the variation in the performance in terms of the total delay in the network. Let be $H \subseteq V$ the set of clients. For the sake of convenience, we are assuming that the set of clients and hosts are disjoint sets. A capacity of vector matrix C represents the capacities of the sites in a vector format with $C_i = C^1_i + C^2_i + C^3_i$ being the capacity of site i. As mentioned earlier, we are referring to a 3-D vector to represent the capacity, that is, CPU, Storage and Network Capacity. $|C_i| = 0$ indicates that the site i is a client site.

Let P be the transmission delay matrix with P_{ij} being the delay between nodes i and j in the graph G. This can be calculated with the help of simple ping requests between two nodes i and j [11]. Let W be the matrix to represent the volume of traffic originating from the client sites, that is, W_i be the traffic getting generated at node i. The total response time (latency), R_{ij} for a client i which is allocated to the host j, is a function of P_{ij} and W_i. That is, $R_{ij} = f(P_{ij}, W_i)$.

Let M be total number of VF. Let D be the demand matrix for the VFs with $D_m = D^1_m + D^2_m + D^3_m$ being the demand of VF m. Let Γ be a representing the processing limit of VFs, that is, maximum client traffic a single host VF can handle with Γ_m being the processing limit of the VF m. Please note that more than one instances of a VF may be deployed at any deployment site depending on the processing capacity of the VF and total traffic demand getting generated at the sites which are allocated to that particular VF. Let T be the instance matrix with T_{mk} representing how many instances of a VF m need to be deployed at site k.

Let A be an allocation matrix such that $A_{ij} = 1$ if client request i is assigned to the host j. Note that $A_{kk} = 1$ means node k has been assigned a client request. In other words, a workflow instance has been deployed on a VF at node k. We have assumed no-split of the client requests amongst the hosts or VFs, that is, one client request will be processed at a single node only (single-allocation model). The constraint may be modeled as:

$$\sum_{j \in |V|} A_{ij} = 1, \ \forall i \in |V| \qquad (1)$$

As mentioned earlier, for the sake of convenience, we are assuming that the set of clients and hosts are disjoint set. Hence, we need to make sure that the clients requests are forwarded to VF nodes only (and not to the other client nodes). It is ensured with the help of following constraint:

$$A_{ik} \leq A_{kk}, \ \forall i, k \in |V| \qquad (2)$$

Also, we will be providing number of host nodes to be installed as an input, λ. λ varies from λ_{min} to λ_{max}. λ_{min} may start from 1 however we provide some feasible number to start with. Also let f be fixed cost associated with the installation of a single host and F be the total cost limit. Hence λ_{max} can be calculated as $\lambda_{max} = F/f$. We need to make sure that the total number of VFs hosting the workflows should be equal to λ. That is:

$$\sum_{k \in |V|} A^{kk} = \lambda \quad (3)$$

Maximum number of instances of a VF which may be deployed at a given site is bounded by the capacity of the site and demands of the VF. Similarly, minimum number of a VF is bounded by the total client traffic from all the sites assigned to that VF. We formulate the capacity constraints as follows.

$$T_{mk} \leq A_{kk} \times (C_k/D_m),\ \forall k \in |V|,\ m \in M \quad (4)$$
$$T_{mk} \geq \sum_{j \in |V|} A_{jk} \times (W_j/\Gamma_m),\ \forall k \in |V|,\ m \in M \quad (5)$$

We formulate our optimization function as follows:

$$\textbf{Minimize}: \sum_{i \in C} \sum_{j \in |V|} R_{ij} A_{ij} \quad (6)$$

We seek to minimize the total response time to the clients in the network. We solve the above formulation using *Integer Linear Program* (ILP) tool. The results are presented later in Section V. In the next section we present a heuristic approach for dynamic VF deployment in multi-cloud scenario.

IV. Dynamic Virtual Function Deployment — Heuristic Approach

We have proposed an optimal solution in the previous section to place the VFs for specific set of client demands. In the cloud, a single *Application Service Provider* (ASP) may have a set of services (VFs) to be deployed and every VF has specific requirements such as computational power, storage capacity and others. OpenADN platform which deploys services over multiple clouds imposes additional constraints such as network bandwidth requirement. Each Virtual Machine (VM) may host a specific set of VFs depending on its computational, storage and network capabilities.

Locating suitable VMs from a given pool to minimize the number of VMs so that all the instances of the VFs can be satisfied is a NP-complete problem. It can be reduced to the "*Set Cover*" problem in a polynomial time. In this section, we describe an approximate heuristic approach for the VF deployment on the preselected VMs across the multiple clouds. We will be introducing a 3-D model for service requirements and VM capacities which is necessary for multi-cloud scenario. As we have mentioned, the already existing cloud management platforms are good for service deployment within a single cloud and hence the model considered for service requirements is a 2-D model, that is, computational capacity and storage capacity [8, 15]. However, we will be introducing a new parameter which we refer as *network capacity*. Each VF in a workflow demands some minimum network infrastructure to be able to communicate across the data-centers with other VF as well.

We assume that each VM on a physical server gets equal share of the host bandwidth. ASPs need to provide configuration files for the VF requirements after doing a careful VF profiling. As no logs or patterns of VF or service requests for data-center or clouds are publicly available due to privacy and/or security concerns, we generate the service requirement patterns along the lines of published traffic distributions to emulate typical cloud workloads to evaluate the proposed schemes [9, 10, 12]. We have assumed the VM configurations from Amazon EC2 cloud service [5].

We propose *Minimum-Residue* heuristic approach for this variant of set-cover problem. As mentioned earlier, we model each ASP service as a 3-D vector. Also, each VM is represented in term of the same 3-D vector, that is, computational capacity, storage capacity and network capacity. We then try to fit all the instances of the VFs for each ASP on minimum number of VMs. For a given set S of the VFs for a given ASP, we calculate a new set $\ddot{U}$, which is a set of all the subsets for S (except a *null* set). For example, if $S = \{s1, s2, s3\}$ (that is an ASP has three services to be deployed), then the corresponding $\ddot{U}$ will be: $\ddot{U} = \{\{s1\}, \{s2\}, \{s3\}, \{s1, s2\}, \ldots, \{s1, s2, s3\}\}$. Each VM will be able to satisfy some of these subsets, that is, can accommodate some VFs at a given time instance.

Then we try to find out a set-cover with the given VMs so that all the VFs in S will be covered with the minimum number of VMs. To do this, we iterate through all the subsets of S in $\ddot{U}$. We find total CPU, storage and network bandwidth requirement for each subset. While considering the requirements for a particular VF, we allow it to run on a VM even if a VM cannot satisfy its complete CPU and Network requirements, but instead can satisfy some percentage of it. However, the storage requirements need to be satisfied completely for a VF to be started. It may be justified since an ASP may prefer a service to be a bit slower rather than not running at all. We have assumed the percentage level to be 80% (i.e. $\lambda = 0.8$), though this parameter is configurable and depends on the end-user demands. We then iterate through each VM sequentially. For each VM, we subtract the 3-D vector calculated above from the 3-D vector of that particular VM. We discard all the subsets where a negative term is introduced (that is the VM cannot satisfy one of three requirement of that subset). For the remaining subsets, we choose a subset which results in minimum remaining resources on that VM. In other words, of all the possible subsets of the VFs, we fit that subset on the VM which will result in the maximum utilization of the VM. If there are two or more subsets with the same remaining resources, we choose the subset with minimum size since smaller the size of the subset, more difficult it will be to split the subset later on. In other words, there will be less possible combinations to try for a subset with smaller size.

We repeat this process till all the services for a given ASP are deployed. This procedure is repeated for all the ASPs. We notice that a similar approach is used in the field of Computer Architecture and Operating Systems to avoid the fragmentation in the computer memory or storage [21, 22]. It must be noted that, though we start the heuristic with static knowledgebase of the service requirements, the dynamic service allocation is also possible with the proposed heuristic. If a completely new set of services from an ASP arrives, we just need to execute the heuristic again. If an ASP adds to its existing set of services, a VM can be chosen with the maximum residue to satisfy the newly added services. Listing 1 below outlines the proposed heuristic.

Listing 1: Outline of the proposed heuristic

```
Algorithm: Minimal-Residue
1.  Let S be the set of services the given ASP
2.  Ü ← Set of all Subsets of S (except Null set)
3.  foreach set Ş in Ü
4.  foreach service ṡ in Ş
      1.  Ş ($CPU_{total}$)+= ṡ(CPU) X λ
      2.  Ş ($Storage_{total}$)+= ṡ(Storage)
      3.  Ş ($NW_{total}$) += ṡ(NW) X λ
5.  foreach V in VMs
6.  foreach set Ş in Ü
      If V(CPU) >= Ş ($CPU_{total}$) && V(Storage) >= Ş ($Storage_{total}$)
      && V(NW) >= Ş ($NW_{total}$)
          V(residue) ← ($3\_D$ Vector)$_{remaining\_cap}$
7.  Sort the set of V(residue) for all VMs in non-decreasing order
8.  Select first Ş as a set of services to be deployed on the selected VM V.
9.  Remove all the services ṡ in Ş from the set of services to be allocated
    (set S).
10. Repeat until all services are allocated to the VMs
11. Repeat Step 1 through Step 10 for all ASPs.
```

V. Results and Analysis

In this section, first, we analyze the performance of the ILP presented in Section III for optimal VF deployments so as to minimize the total response time in the network. Random cluster graphs generated employing the "*networkx*" library [6] using Python have been used for the purpose of simulation. We have considered mesh topology with average degree of three or four. Such networks are prevalent in the contemporary metro and access networks. Networks with sizes varying from 10-nodes to 100-nodes have been considered for the simulation purpose. Client requests are generated at these nodes in the Autonomous Systems. A sample 40-node topology for data-center interconnectivity (data from a data-center service provider in USA) considered in the simulations is displayed in Fig. 1 below.

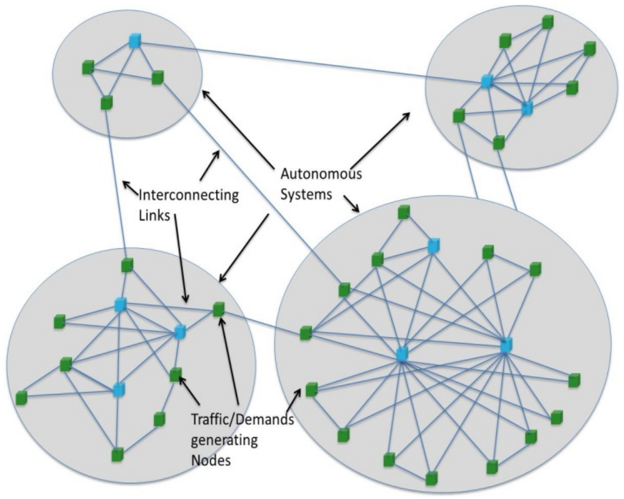

Fig. 1: A sample 40-node topology for the inter-datacenter connectivity

Fig. 2 below displays the same network topology as in Fig.1 using a tool "*GraphDraw*" which we have used for the visualization of the various topology and generating the adjacency matrix for the given topology.

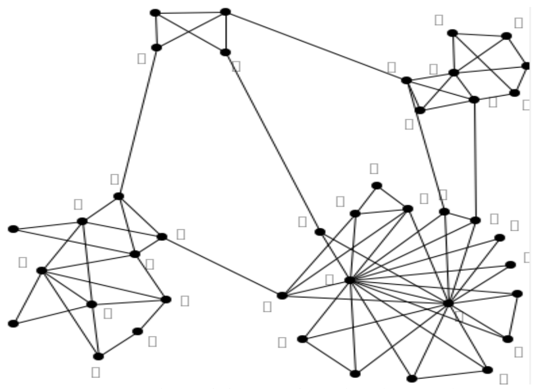

Fig. 2: Representation of the topology in Fig. 1 using a graph generator tool "*GraphDraw*"

We have used MOSEK optimizing tool [24] for the solution. We have provided number of VFs to be installed as an input parameter and the range (λ_{min} to λ_{max}) is calculated as mentioned in the Section III earlier. Service requests are generated randomly with the service granularities in Mbps and are chosen from the set G = {100, 200, 500, 1000}. Fig. 3 below displays variation of the total response time for topology from 10-nodes till 40-nodes with number of VFs installed varying from 4 to 40. Fig. 4 displays the same graph for 50-nodes till 100-nodes topology with number of VFs installed varying from 10 to 100.

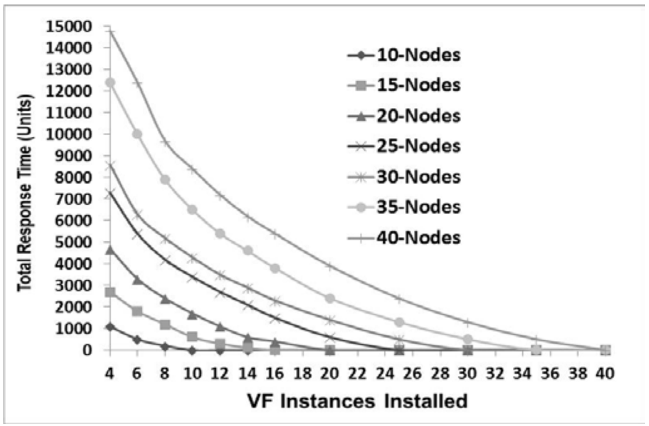

Fig. 3: Total response time against number of VFs installed (15-nodes till 40-nodes)

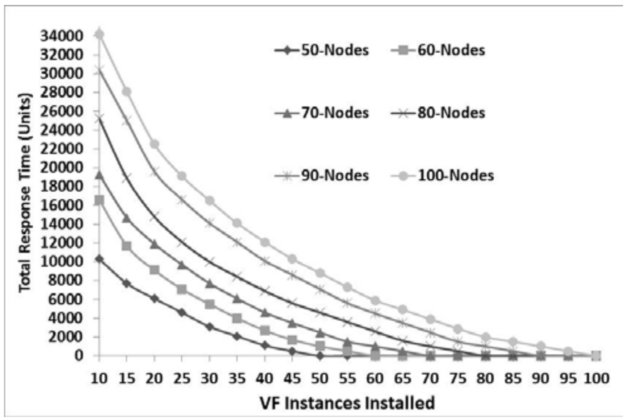

Fig. 4: Total response time against number of VFs installed (50-nodes till 100-nodes)

We observe from both the graphs that the response time reduces as the number of VFs deployed increases till certain limit, however, after that there is no significant reduction in the total response time. For example, the total response time dips approximately from 35k units to 10k units for 100-nodes as total number of VFs installed vary from 20 to 50 (approximate 25k units of improvement). However the change in total response time for same topology is from 8k units to 2k units with VFs varying from 60 to 90 (improvement is mere 6k units). We now demonstrate the performance of the proposed heuristic "*Minimum-Residue"* for the dynamic service deployment on the preselected pool of VMs so as to satisfy all the VF demands on minimum number of VMs. In addition we compare the performance of the heuristic against well-known "*Max-Min*" heuristic approach for task allocation. We have used a variant of the Max-Min approach proposed in [17, 20] to suit our input data-sets. As mentioned earlier, we generate the service requirement patterns along the lines of traffic distributions in published work or on the Internet to emulate typical cloud workloads to evaluate the proposed schemes [9, 10, 12, 19]. A simple example is presented below where our proposed heuristic outperform the standard *Max-Min* approach. We consider a set of five VFs. Let there be two resources, r_1 and r_2, analogous to the CPU and Storage demands (we are considering 2-D resource model in this example for the simplicity purpose, however implementation is for 3-D model as mentioned earlier. The example can easily be extended for 3-D model). Table 1 shows the demands of the VFs for r_1 and r_2. Table 2 displays the available VM configurations with r_1 and r_2 available.

Table 1: VF requirements

Services	S1	S2	S3	S4	S5
r_1	10	80	10	10	40
r_2	10	50	10	10	90

Table 2: VM configurations

VMs	V1	V2	V3
r_1	50	100	100
r_2	100	100	100

The allotment scheme by both the heuristics is displayed in Table 3 below. As we observe, the Max-Min heuristic needs all the three VMs. While our proposed scheme can accommodate all the VFs with only two VMs. This is because we search for minimum remaining capacity for each VM rather than greedily allocating the demands, which might get stuck in local-minima.

Table 3: Allotment schemes by both heuristics

Max-Min	***Minimum Residue***
V1 → S1, S3, S4	*V1 → S1, S5*
V2 → S2	*V2 → S2, S3, S4*
V3 → S5	*V3 → free*

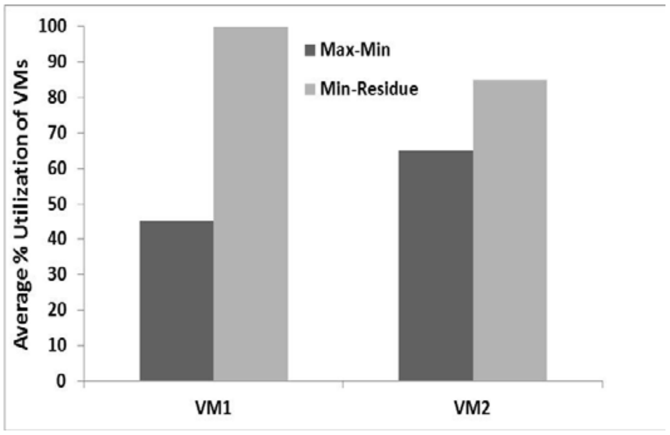

Fig. 6: Average Utilization of the VMs with the two schemes

This also ensures maximum *Average Utilization* of the VMs. The graph in Fig. 6 displays the average utilization (as per the definition in [20]) of the VMs in the above example. As we observe, the utilization of both the used VMs is higher with the proposed heuristic against that of *Mix-Min* approach. We have generated various sets of VFs with total number varying from 100 till 1000. We observe from the graph displayed in Fig. 7 that the maximum number of VMs needed to satisfy all the incoming service requests with the proposed heuristic is always less than that of standard *Max-Min* heuristic. Please note we have considered only *micro* and *small* VM configurations from [5] to generate this result for better demonstration. We observe in the graph that the gap between two schemes keeps on increasing as number of services increases. Though the gap in both schemes is small, we expect a better performance with more VM configurations. Overall we observe that the performance of the proposed heuristic gets better with increased load.

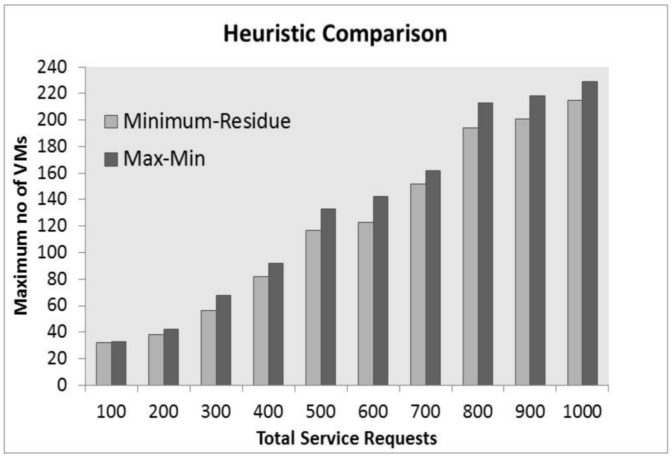

Fig. 7: Comparison between proposed Minimum-Residue and standard Max-Min Heuristic

VI. Conclusions

In this work we propose an optimization model to place the Virtual Functions in multi-cloud environment so as to minimize the total transmission delay in the network and solve it using Integer Linear Program tool. We also propose a heuristic scheme for dynamic Virtual Function deployment at the sites which are selected in the optimization model. We deploy the instances of VFs on Virtual Machines at pre-selected sites, which may be deployed on the OpenADN platform, a multi-cloud application delivery tool. The problem is NP-Complete and we propose a polynomial time heuristic approach "*Minimum Residue*" to solve the problem for 3 dimensional (3-D) capacity model. The proposed scheme is compared with the standard *Max-Min* approach in the literature and it is showed that the proposed heuristic out-performs standard approach. Simulation results are presented to evaluate the optimal solution and the proposed heuristic with larger topology and greater number of VF instances.

VII. References

[1] S. Paul, R. Jain, M. Samaka, J. Pan, "Application Delivery in Multi-Cloud Environments using Software Defined Networking," Computer Networks Special Issue on cloud networking and communications, February 2014, pp. 166-186.

[2] S. Paul and R. Jain, "OpenADN: Mobile Apps on Global Clouds Using OpenFlow and Software Defined Networking," 1st Int'l. Wksp. On Management and Security Technologies for Cloud Computing, December 2012, pp. 719-723.

[3] X. Tang, J. Xu, "QoS-aware replica placement for content distribution", IEEE Transactions on Parallel and Distributed Systems, August 2005, pp. 921-932.

[4] S. Bakiras, "Approximate server selection algorithms in content distribution networks," IEEE International Conference on Communications, 2005, pp.1490-1494.

[5] Amazon EC2 Virtual Function Instances: http://aws.amazon.com/ec2/instance-types/

[6] Online reference: http://networkx.github.com/

[7] A. Guyton and M. Schwartz, "Locating nearby copies of replicated internet servers", ACM SIGCOMM, August 1995, pp. 288–298.

[8] K. M. Hanna, N. Natarajan, and B. N. Levine, "Evaluation of a novel two-step server selection metric", IEEE International Conference on Network Protocols (ICNP), November 2001, pp. 290–300.

[9] T. Benson, A. Anand, A. Akella, M. Zhang, "Understanding Data Center Traffic Characteristics", ACM SIGCOMM Computer Communication Review, 2009, pp. 92-99.

[10] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, R. Chaiken, "The Nature of Data Center Traffic: Measurements & Analysis", ACM SIGCOMM conference on Internet measurement, 2009, pp. 202-208.

[11] TraceRT consulting service. [Online]: http://www.tracert.com

[12] X. Kuai, F. Wang, G. Lin, "Profiling-as-a-Service in Multi-tenant Cloud Computing Environments", Distributed Computing Systems Workshops (ICDCSW), June 2012, pp. 461-465.

[13] N. Laoutaris, V. Zissimopoulos, I. Stavrakakis, "On the optimization of storage capacity allocation for content distribution", Computer Networks, February 2005, pp. 409-428.

[14] S. Seshan, M. Stemm, and R. Katz, "Shared passive network performance discovery", USENIX Symposium on Internet Technologies and Systems, December 1997, pp. 1-18.

[15] A. Lenk, M. Klems, J. Nimis, S. Tai, T. Sandholm, "What's inside the Cloud? An architectural map of the Cloud landscape", ICSE Workshop on Software Engineering Challenges of Cloud Computing, May 2009, pp. 23-31.

[16] M .Randles, D. Lamb, A. Bendiab, "A Comparative Study into Distributed Load Balancing Algorithms for Cloud Computing", IEEE 24th International Conference on Advanced Information Networking and Applications Workshops (WAINA), 2010, pp. 551-556.

[17] K. Li, G. Xu, G. Zhao, Y. Dong, D. Wang, "Cloud task scheduling based on load balancing ant colony optimization", IEEE Sixth Annual ChinaGrid Conference, 2011, pp. 3-9.

[18] Z. Wu, X. Liu, Z. Ni, D. Yuan, A. Yang, "Market-oriented hierarchical scheduling strategy in cloud workflow systems", Journal of Super Computing, 2013, pp. 256-293.

[19] ETSI, "NFV – Update White Paper," Oct 2013, http://www.tid.es/es/Documents/NFV_White_PaperV2.pdf.

[20] J. Cao, D. P. Spooner, S. A. Jarvis, G. R. Nudd, "Grid Load Balancing Using Intelligent Agents", Future Generation Computer Systems, 2005, pp. 135-149.

[21] P. Denning, "The Locality Principle", Communications of the ACM, 2005, pp. 19-24.

[22] Q. Zhu, Y. Qiao, "A Survey on Computer System Memory Management and Optimization Techniques", American Journal of Computer Architecture, 2012, pp. 37-50.

[23] W. John, K. Pentikousis, G. Agapiou, E. Jacob, M. Kind, A. Manzalini, F. Risso, D. Staessens, R. Steinert, C. Meirosu, "Research Directions in Network Service Chaining," IEEE SDN for Future Networks and Services (SDN4FNS), November 2013 , pp. 11-13.

[24] MOSEK, https://mosek.com