Balancing the Load Across Virtual Links from Virtual Machine Requests in Distributed Clouds

Glauco Estácio Gonçalves, André Vitor de Almeida Palhares, Marcelo Anderson Batista dos Santos, Patricia Takako Endo, Judith Kelner, Djamel Sadok
Federal University of Pernambuco (UFPE)
Networking and Telecommunications Research Group (GPRT)
Recife, Pernambuco, Brazil
{glauco, andre.vitor, marcelo, patricia, jk, jamel}@gprt.ufpe.br

***Abstract*—Cloud providers should be able to attend to different types of requests from their users. Generally, requests are composed of virtual machine and virtual link restrictions. When users request only virtual machines, the Cloud provider should interconnect them – by creating a virtual network - in order to allow the communication. In this paper, we propose a strategy to create virtual links between virtual machines, aiming for load balancing, and at same time, link consolidation in order to optimize resource utilization in Distributed Clouds. The results show that our algorithm usually hits the optimum solution for small requests for virtual machines.**

***Index Terms*— Distributed Clouds, virtual link allocation, load balancing, link consolidation.**

I. INTRODUCTION

Current Cloud Computing setups involve a huge amount of investment in the datacenter, which is the common underlying infrastructure of Clouds. This centralized infrastructure brings many well-known challenges such as the need for resource over-provisioning and the high cost of heat dissipation and temperature control. Considering well-known challenges in centralized infrastructure, industry and academic researchers have presented indications that small datacenters can be sometimes more attractive since they offer a cheaper and lower-power consumption alternative while also reducing the infrastructure costs of centralized Clouds [2]. These small datacenters can be built in different geographical regions and connected by dedicated or public (provided by Internet Service Providers) networks, configuring a new type of Cloud, referred as **Distributed Clouds** [3], or just D-Clouds. Such D-Clouds can exploit the possibility of virtual link creation and the potential of sharing resources across geographic boundaries to provide latency-based allocation of resources to fully utilize this emerging distributed computing power.

When clients make a request for virtual machines, it is expected that virtual machines are able to communicate with each other, composing something like a virtual local network. Then, the D-Cloud provider could apply some optimization strategy to create a virtual network for intercommunication between the virtual machines in order to obtain better usage of its network resources.

Working with this scenario, this paper investigates the problem of creating a virtual network when a request for virtual machines is submitted without any link constraints, and also proposes a strategy to solve it in D-Clouds. Regarding virtual link creation, the algorithm focuses on minimizing the maximum number of virtual paths allocated over a physical link while performing load balancing. We assume that the virtual machines were already allocated in a given region, due to geographical restrictions, for example. Then, analyzing the graph of the physical network infrastructure, our strategy will calculate the need for creation of one or more virtual hubs in order to concentrate virtual links. The virtual hubs positioning is similar to the well-known Replica Placement (RP) problem.

To the best of our knowledge, there is no other similar work that optimizes the utilization of physical links trying to minimize the energy consumption and, at the same time, perform link load balancing. It is an interesting question that will be discussed in Section VI.

The rest of the paper is organized as follow: The related works are described in Section II. The problem and our solution proposal are defined in Section III. Section IV shows the simulation methodology. Section V shows evaluations and results. Section VI discusses results and applications of the proposed algorithm. Finally, conclusions and future works are delineated in Section VII.

II. RELATED WORK

The main objective of the **Replica Placement** (RP) problem is to decide where, when, and by whom servers or their content should be positioned in order to improve performance. The correspondent existing solutions to these problems are generally known as **Replica Placement Algorithms** (RPA) [8].

The general RP problem is modeled as a physical topology (represented by a graph), a set of clients requesting services, and some servers to place on the graph (costs per server can be considered instead). As pointed out by [8], an RPA groups these aspects into two different components: the problem definition, which consists of a cost function to be minimized under some constraints, and a heuristic, which is used to search for near-optimal solutions in a feasible time frame, since the defined problems are usually NP-complete. Different versions

978-3-901882-50-0

of this problem can be mapped onto resource allocation problems in D-Clouds. A very simple mapping can be defined considering an IaaS service where virtual machines can be allocated in a geo-distributed infrastructure.

Presti et al. [9] treat an RP variant considering a trade-off between the load of requests per content and the number of replica additions and removals. Their solution considers that each server in the physical topology decides autonomously, based on thresholds, when to clone overloaded contents or to remove the underutilized ones. Such decisions also encompass the minimization of the distance between clients and the respective accessed replica. A similar problem is investigated in [10], but considering constraints on the QoS perceived by the client. The authors propose a mathematical offline formulation and an online version that uses a greedy heuristic. The results show that the heuristic presents good results with minor computational time.

Qiu et al. [11] propose three different algorithms to solve the K-median problem in a CDN (Content Delivery Network) scenario: a Tree-based algorithm, a Greedy algorithm, and a Hot Spot algorithm. The Tree-based solution assumes that the underlying graph is a tree that is divided into several small trees, placing each server in each small tree. The Greedy algorithm places servers one at a time in order to obtain a better solution in each step until all servers are allocated. Finally, the Hot Spot solution attempts to place servers in the vicinity of clients with the greatest demand. The results showed that the Greedy Algorithm for replica placement could provide CDNs with performance that is close to optimal.

Zhu and Ammar [12] propose a set of four algorithms with the goal of balancing the load on the physical links and nodes, but their algorithms do not consider capacity aspects. Their algorithms perform the initial allocation and make adaptive optimizations to obtain better allocations. The key idea of the algorithms is to allocate virtual nodes considering the load of the node and the load of the neighbor links of that node. Thus, one can say that they perform the allocation in a coordinated way. For virtual link allocation, the algorithm tries to select paths with few stressed links in the network.

In our work, the difference is the consumer request, since there is no specification of link, only virtual machines. Furthermore, our solution executes load balancing in physical links together with link aggregation, decreasing the possibility of bottlenecks and simultaneously using a smaller number of physical links in a D-Cloud scenario. Finally, the RP problem is solved by a Greedy algorithm, reducing the problem to a Steiner Tree.

III. Problem Definition and Algorithms

Usually, links between virtual machines are specified in a virtual network request. For this case, there are well-known allocation algorithms (such as [1], [4], [5], and [6]). On the other hand, it is possible that a client makes a request specifying only virtual machines with hardware features and location constraints. In this case, a D-Cloud provider is free to allocate links between all virtual machines without restrictions and make them accessible for each other. Working with this scenario, this section investigates the problem of creating a virtual network when a request is submitted only with virtual machine restrictions (e.g. memory, CPU, location and storage).

For us, the problem consists of how to create a virtual network that connects a subset of the virtual nodes with the objective of **balancing the load** across physical links and, secondly, **minimizing the energy consumption** through consolidation of virtual links over physical links [13]. Note that the problem considers that virtual machines were previously allocated by any allocation algorithm.

This problem can be formulated as follows: find an interconnection with minimum length (number of links) for a subset of nodes such that the maximum weight of its links is minimal. Weight corresponds to the stress in the physical link, i.e., the number of virtual links already allocated over that physical link.

The minimum Steiner tree problem can be reduced to our problem by considering minimizing the length in number of links and with all link weights with the same arbitrary value. On the other hand, if the value of the minimal maximum weight is given, the graph can be pruned, and then the problem reduces to finding a minimum length Steiner tree. As one can note, problems are polynomial-time equivalent.

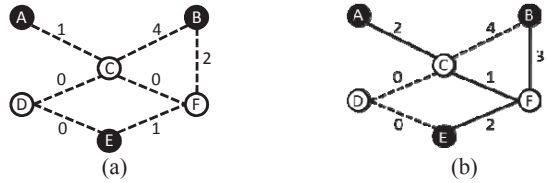


Figure 1. Example creating a virtual network: (a) before the creation; (b) after the creation

Figure 1(a) shows an example of a given physical topology where the virtual machines are allocated by unique user in nodes A, B, and E (black nodes), and the number in each physical link indicates the number of current virtual links crossing this physical link. Figure 1(b) shows the virtual network (the black lines) created to interconnect those nodes. The created virtual network reaches the two objectives of balancing the load in the network and reducing the number of used links, which leads to the reduction of the energy consumption.

The proposed problem is NP-hard. Therefore, approximated solutions will be proposed to solve it. The reduction of the problem to a Steiner tree gives an interesting idea for our heuristic. Such heuristic consists in executing a binary search in a vector ranging from the current minimum link weight to the maximum link weight of the graph. During each iteration, a weight k is selected and all links with weight greater than k are removed from the graph. An approximate minimum length Steiner tree algorithm [1] is executed in this subgraph to verify if there is a tree between the nodes. If there is a Steiner tree in this graph, the upper bound is adjusted to the current k. Adversely, if there is no path in this graph, the lower bound is adjusted to k. The binary search is repeated until the vector is reduced to two elements. Thus, k is chosen as the element in the upper bound, and the Steiner tree approximation gives the virtual network for the problem.

The binary search provides the load balance searching by minimal maximum link weight, while the Steiner tree approximation gives the minimum length virtual network that minimizes energy.

Next, the algorithms used to solve the minimum length Steiner tree problem in a graph are shown. The first algorithm (named STA) is a well-known approximation based on the minimum spanning tree in the distance graph [1]. The second (called GHS) and third (called OA) algorithms are proposed by this paper: one uses a greedy heuristic that searches for the better hubs in the network in order to minimize the path lengths, and the other is an exponential algorithm that finds the minimum Steiner tree through successive tries of link removal.

A. Steiner Tree Approximation (STA)

Basically, the STA algorithm consists of transforming a general Steiner tree problem into a metric Steiner tree problem, which is a variant of the general problem where the graph is complete and the links satisfy the triangle inequality $cost(u,v) \leq cost(u,w) + cost(v,w)$. The transformation is made by calculating the all-pairs shortest path in the original graph G and generating a new connected graph G', whose nodes are the nodes of the original graph and the links are the cost of the shortest path between the nodes in the original graph.

In *G'* one can find a minimum spanning tree *T'* that is a 2-approximation of the minimum Steiner tree in this distance graph. Given this tree, one can replace their links by the original paths to obtain a subgraph in G. If this subgraph contains cycles, removing links will generate a 2-approximation of the minimum Steiner tree in the general graph. More details and proofs of this process can be found in [1]. The complexity of the STA algorithm is $O(N^3)$, where N is the number of physical nodes.

B. Greedy Hub Selection (GHS)

The solution of the Steiner tree contains a subset of nodes of the graph that acts like a group of hubs interconnecting two or more nodes of the Steiner tree (the nodes C and F in Figure 1(b)). Thus, given a graph and a subset of physical nodes containing the requested virtual nodes, the objective of our GHS algorithm is to find the hubs of the minimum length Steiner tree interconnecting the virtual nodes.

The GHS algorithm initiates with a tree formed by the physical nodes where a virtual node was allocated. One of these nodes is chosen as the root of the tree and as the first hub. The procedure for selecting the root node tries to select the virtual node that minimizes the summing of the distances between all the allocated virtual nodes, where the links' weights are all equal to one and the summation is made only in a subset of nodes. Note that this problem is equivalent to solving the 1-median problem, which is seen as a very simple form of the replica problems. This problem can be solved calculating the all-pairs shortest path in a weighted graph which can be obtained with the Floyd-Warshall algorithm whose complexity is $O(N^3)$[7], where N is the number of nodes. Using the distance from each node to each other, the sum of the distances can be calculated for each node in the graph that is a candidate for positioning the root. The node that has the minimum sum of distances is the solution to the root.

Following an iterative approach, a new hub node is placed at the best location in the network, defined as the one which achieves the minimal number of used links (the cost) among all the possible positions. This location is then fixed (`Search Procedure`). The new hub is connected to other nodes in the tree through the shortest path, but a heuristic is used to maintain the tree (`Placement Procedure`). The positioning of a new hub and the immediate link rewiring reduces the cost and these processes follow while the positioning of a new hub reduces the cost.

The selection characterizes GHS as **greedy**: it selects the best configuration possible for the current iteration. Thus, GHS searches for a solution with a relevant degree of optimality, calculating the new value of the cost for each selected candidate physical node, and selecting the one that achieves the best cost. The complexity of GHS is $O(N^3)$.

Algorithm 1: Search Procedure

```
1   Inputs: nodeList;
2   Outputs: selectedNode, best;
3   best = infinite;
4   for each node in nodeList(available) do
5      cost = placementProcedure(node);
6      if(cost < best)
7         selectedNode = node;
8         best = cost;
9      end
10     undoInsertion(node);
11  end
```

Figure 2. Search procedure used by the GHS algorithm

The pseudo-code of this search procedure is presented in Figure 2. Such procedure simply selects any available physical node as a candidate. The variable `nodeList` can be indexed with available, hub, root, or requested in order to return respectively: the available nodes; the hub nodes other than the root; the root node; or the other requested nodes. Thus, `nodeList(available)` returns the nodes of the physical network that are not already a hub, the root, or a requested node. After that, the algorithm calls the `Placement Procedure` (Figure 3) adding a new hub in this candidate and rewiring the virtual links (line 5). The cost achieved by placing this new hub is calculated and if better than others it is selected as the best candidate node (lines 7 and 8). Line 10 returns the network to the state before the hub insertion in order to prepare the network for the next candidate hub.

The placement strategy (Figure 3) is a heuristic approach that links a candidate hub to a parent and children maintaining the tree structure and guaranteeing optimality. The parent is selected firstly as the nearest hub considering the shortest path length (line 3), but if the virtual link from the current parent of this nearest hub crosses the candidate (line 4) the parent of the candidate will be the parent of the nearest hub, and the candidate will be the new parent of this hub. After that, the children of the new hub are selected amongst the other hub and requested nodes (line 10). A node is chosen as a child if the new hub is a better parent (nearer) than the current parent and if

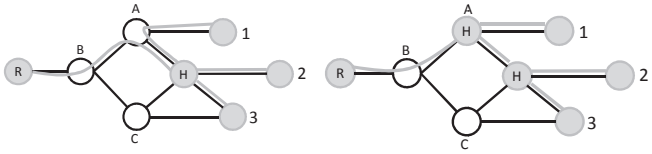
the node is not an ancestor of the new hub. After placing the new hub, the new number of used links is returned.

Algorithm 2: Placement Procedure

```
1   Inputs: candidateNode
2   Outputs: newCost
3   nearest = nearestHub(candidate);
4   if(path(nearest,parent(nearest)).contains(candid
    ate))
5     parent(candidate) = parent(nearest);
6     parent(nearest) = candidate;
7   else
8     parent(candidate) = nearest;
9   end
10  for each node in nodeList(hub or requested) do
11    if(distance(node, parent(node)) >
      distance(node, candidate) and not
      isAncestor(node, candidate))
12       parent(node) = candidate;
13    end
14  end
15  newCost = calculateNewCost();
```

Figure 3. Placement procedure used by the GHS algorithm

In order to clarify the proposed heuristic, let's consider the current virtual tree in Figure 4(a), which is formed by the grey nodes, with the R node representing the selected root node, the H node as a hub selected in a previous iteration, and the grey lines indicating the path of the current virtual links. The white nodes are available for adding new hubs. Considering the current candidate as the node A, one must select the nearest one as node H, since it is the nearest hub to A. However, as the path that goes from H to its parent R passes through A, the parent of A is set as R and H as a child of A. Finally, one must set all the other children of A, as any node that is not an ancestor of the new hub (as the parent of A was already set, its ancestor is well defined as the node R) which has a distance to A of less than its distance to its own parent. Notice that the nodes that can be its children are 1, 2, and 3. For such nodes, only node 1 is nearer to node A than to its own parent; the others are nearer to node H. So, the new children of A are 1 and H. Figure 4 (b) depicts the new configuration. The cost function should be calculated now and compared to other candidate results.



(a) (b)

Figure 4. Example of the placement procedure: (a) before and (b) after placement

C. Optimal Algorithm (OA)

Because a Steiner tree never contains a cycle, then there exists a subgraph of the original graph which is a tree and contains a minimal Steiner tree for any given subset of nodes. Observe that, in this subgraph there is only one path between any two nodes. Thus, if the graph is already a tree, the minimal Steiner tree between the given subset of nodes can be found on linear time through a depth-first search.

Considering such property, an optimal algorithm to find the minimum length Steiner tree can be proposed: for the connected component of the graph in which the subset of nodes is – which can be found by a breadth-first search after removing the links with weight greater than k –, count the minimal number of links that are needed to be removed in order to turn this graph into a tree. If N is the number of nodes and L the number of links of the considered component, this number is given by $m = L - N + 1$ [6]. Then, in this component, for each subset of m links, remove them and find the optimal Steiner tree in this subgraph, considering that this graph is a tree, through the depth-first search, as observed previously. As the algorithm tries every possible way of removing these links, one of them will find the tree which leads to an optimal Steiner tree. The complexity of this algorithm is $O(\binom{L}{m}(N+L))$.

IV. Evaluation Methodology

The evaluation was made through Monte Carlo simulations considering a fixed physical topology with the random positioning of the virtual nodes. In each simulation sample, the stress of each physical link is drawn from a uniform distribution and the virtual nodes to be connected are positioned in a uniform way. Note that, each sample is independent, and the physical network is returned to its initial values before a new set of requested nodes is attempted.

The adopted physical network was the backbone of the RNP *(Rede Nacional de Ensino e Pesquisa*), a Brazilian academic ISP, which is currently composed of 28 nodes and 33 links as showed at Figure 5.

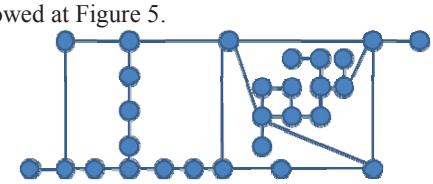


Figure 5. The network topologies of RNP used in simulations

At each run of the simulation, each algorithm (STA, GHS, and OA) is submitted to the same scenario and the number of used physical links (the cost) in the tree is measured for each algorithm independently. The goal is to evaluate to what extent each algorithm achieves the optimal case in the allocation of virtual links while minimizing the cost of using physical links and performing load balancing.

Table I - Factors and levels used in the GHS's evaluation

Factors	Levels
Number of requested virtual nodes	3 to 28
Algorithms	Optimal Algorithm Greedy Hub Selection Steiner Tree Approximation

The factors varied in the experiment are showed in Table I. The number of requested virtual nodes in the network is varied though the simulations from 3 to 28 nodes, which is equivalent to 10.7% and 100% of the physical nodes, respectively. For each run, from a total of 1000, the requested nodes were

positioned in a different physical node in a random way, with every subset of the physical nodes equally likely to be selected without repetitions. For each sample, the relative error between the costs of the GHS and STA algorithms was calculated against the optimal cost. All the results showed use a confidence level of 95%, which are showed in the graphs.

V. Results

This section evaluates the minimum length Steiner tree algorithms discussed in Section IV. We do not evaluate the overall solution for the creation of the virtual network, which has the binary search as the outer procedure, but we evaluate only the inner Steiner tree procedure, i.e., this evaluation only covers the energy reduction objective.

As shown in Figure 6, the GHS algorithm achieves the optimum cost in 100% of the samples for virtual networks with 3 nodes and 95% to 99% of the samples for virtual networks of 4, 5, 6 and 7 nodes. However, the performance of GHS tends to decrease when the number of requested nodes increases. In the worst case, about 53% of the samples of GHS reach the optimum. In the cases from 16 to 26 requested nodes the GHS achieves the lowest percentage to optimum from 53% to 69%.

Moreover, for small virtual networks up to 11 nodes, GHS statistically outperforms STA, with the best cases occurring with 6 and 7 nodes where the difference is about 12%. The performance of STA is high for few requested nodes, decreases in the middle of the range of virtual network size, and reaches the optimum when 27 nodes are requested.

As the number of nodes in the virtual network tends to the total number of nodes, the performance of STA is improved and outperforms GHS since the problem tends to compute the minimum spanning tree in the physical network. On the other hand, GHS is better for small networks because the placement strategy is designed to find the hubs in the physical network whereas the STA strategy is to find the common links in the shortest paths between the requested nodes; if there are no common links in these shortest paths, STA cannot find the hubs minimizing the cost. Looking only for the samples that reached the optimum, one can conclude that the GHS algorithm is not adequate for bigger virtual networks.

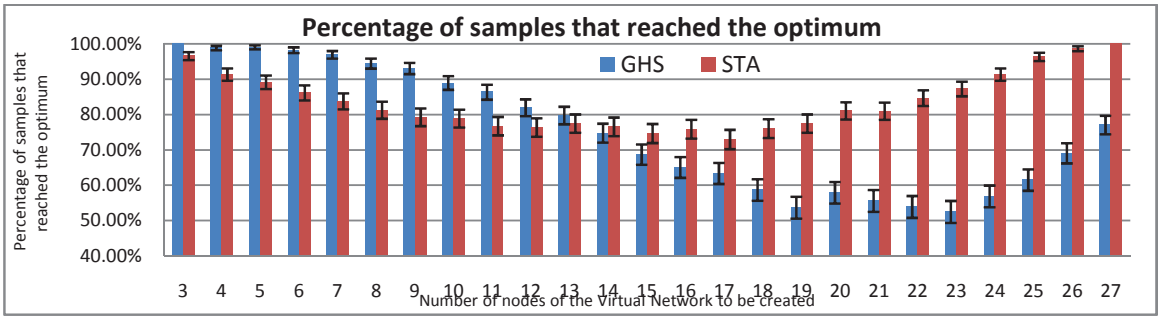


Figure 6. Percentage of optimal samples for GHS and STA

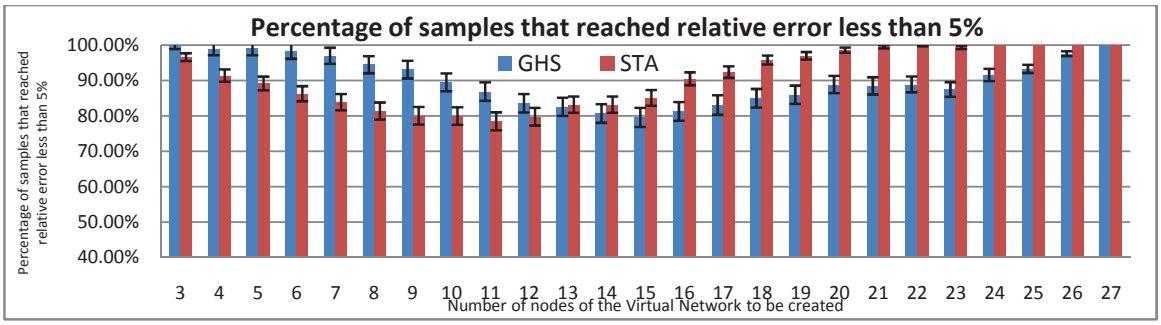


Figure 7. Percentage of samples reaching relative error <= 5

However, Figure 7 shows the performance of each algorithm considering the samples that reached a relative cost error less than 5% in relation to the optimum. In this case, the performance of GHS has significantly improved for virtual networks greater than or equal to 16 nodes. For example, in the scenario with 19 nodes the optimality of the GHS algorithm increased from 53% (considering only optimum samples) to 86%. Furthermore, the STA algorithm has improved for virtual networks but remains below the GHS for requests for up to 11 VMs. This shows that, considering all virtual networks' sizes, the GHS algorithm reached better optimality in requests with few virtual machines, whereas STA reached better optimality in bigger requests. Moreover, the worst case for GHS was for 53.7% of the samples with 19 nodes. For STA, the worst case was 73% with 17 virtual nodes.

Figure 8 shows the relationship of the optimal cost to experiments with STA and GHS algorithms. Note that regardless of the number of virtual nodes required, in both cases the cost focuses on values greater than 15. This behavior is explained by the randomness where the nodes were placed in the topology of RNP. Thus, for example, if 3 VMs are requested and their positions are distant from each other, the number the physical links used will be increased so that there is a connection between each VM requested.

Another important aspect to be noted in Figure 8 is that, for small costs, the GHS algorithm adheres better to the optimal than the STA. Moreover, even in high costs, in most of the experiments the GHS reaches optimum values.

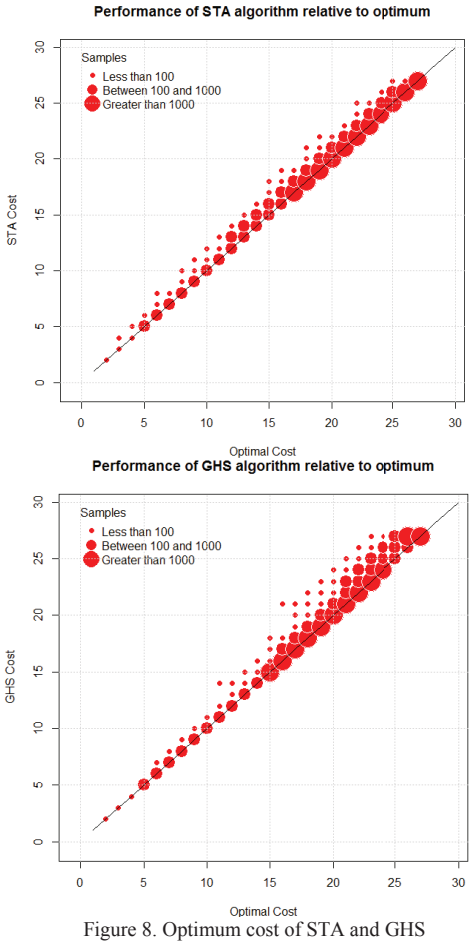


Figure 8. Optimum cost of STA and GHS

VI. Discussions

The allocation of several virtual links focusing on minimizing the number of virtual links on each physical link is an NP-hard optimization problem (see [12] to verify this aspect); this is the reason why the solution presented here is a greedy approach based on some heuristics. Another studied problem was to build a virtual network for a given set of virtual nodes in order to minimize the energy consumption while balancing the load of physical links. Such a problem considers that the developer requests only virtual nodes with no virtual links, and thus, the problem tries to capture the provider's requirements on load balancing and energy consumption.

It is an interesting scenario, due to the antagonism of these two objectives. In general cases, when one wants to save energy, the strategy is to consolidate resources, for example, to allocate requests in a minimal number of available servers. Then, resources not used can be turned off, aiming at the energy saving goal. In a contrary way, when the goal is to balance the load, the strategy distributes the requests across the available servers, and the idleness of resources is reduced, ensuring the load balancing.

Our proposal tries to obtain the better of these two objectives, since we focus on load balancing when we create virtual links to connect virtual machines considering the number of virtual links already allocated over the physical link, and focus on energy reduction when we choose hubs to aggregate virtual links. However, the impact of this blend should be better analyzed. We proposed two algorithms to create virtual links; one an optimum algorithm (OA) and the other one based on heuristics (GHS). The heuristic approach was compared with a known approximation algorithm for the Steiner tree problem (STA), and the results showed that the proposed heuristic is better suited for small size virtual networks whereas the traditional approximation algorithm is better for bigger virtual networks. One possible implementation of this algorithm in a production scenario could consider alternating between these algorithms according to the size of the virtual network in order to obtain the best solution in each case.

The optimum algorithm was used in the experiments as a baseline that references the performance of the other algorithms. Observe that the optimum algorithm is adapted for the physical network used in the test scenario (28 nodes and 33 links). In this case the optimum algorithm could be used for virtual network creation since the number of possible combinations is low, lowering the computing time required.

Finally, it is noteworthy that the algorithm proposed is not entwined with a specific technology. For example, in the core network it is possible to use MPLS or OpenFlow to define forwarding rules.

VII. Conclusions and Future Works

This paper presented the problem of allocating tree-based virtual links for connecting nodes, when the request does not contain any link requirements. The load balancing is the main objective, with the energy reduction as a secondary objective.

A two-step algorithm was designed for this problem, with an outer loop responsible for load balancing and an inner loop for energy reduction. The inner loop problem can be reduced to a minimum length Steiner tree problem and two algorithms are proposed for this step. One uses a greedy strategy for hub placement and the other uses a combinatorial approach to find the optimum. The heuristic-based algorithm is compared against a Steiner tree approximation algorithm with the optimum algorithm as the baseline.

The results showed that the hub placement heuristic is better suited for small virtual networks, while the Steiner approximation has better results in scenarios with larger virtual networks. Overall, in the experiments, our algorithm reached the optimum in most cases, having the worst case of 53.7% for requests with 19 VMs; however, 86% of cases were close to optimum assuming a 5% error.

Future work to follow this proposal includes considering an implementation of the proposed solution in a testbed, measuring, for example, the variation of power consumption by turning off the network devices that are not being used.

ACKOWLEDMENTS

This work was supported by the Innovation Center, Ericsson Telecomunicações S.A., Brazil.

REFERENCES

[1] V. V. Vazirani. Approximation Algorithms, 2nd Edition, Springer-Verlag, 2003.

[2] K. Church, A. Greenbreg, and J. Hamilton. "On Delivering Embarrassingly Distributed Cloud Services", Workshop on Hot Topics in Networks (HotNets), 2008.

[3] P. T. Endo, A. V. A. Palhares, N. N. Pereira, G. E. Gonçalves, D. Sadok, J. Kelner, B. Melander, and J. E. Mangs. "Resource allocation for distributed cloud: concepts and research challenges", IEEE Network Magazine, vol. 25, pp. 42-46, July 2011.

[4] A. Belbekkouche, M. Hasan, A. Karmouch. "Resource Discovery and Allocation in Network Virtualization", IEEE Communications Surveys & Tutorials, n. 99, pp. 1-15, 2012.

[5] N. M. M. K., Chowdhury, M. R. Rahman, and R. Boutaba. "Virtual Network Embedding with Coordinated Node and Link Mapping", IEEE INFOCOM, 2009.

[6] A. Razzaq and M. S. Rathore, "An approach towards resource efficient virtual network embedding", in Proc. HPSR, 2010.

[7] C. E. Leiserson, C. Stein, R. L. Rivest, and T. H. Cormen. Algoritmos: Teoria e Prática. Campus, ed. 1, 2002.

[8] M. Karlsson, C. Karamanolis, and M. A. Mahalingam. "Framework for Evaluating Replica Placement Algorithms". Technical Report HPL-2002, HP Laboratories, July 2002.

[9] F. L. Presti, C. Petrioli, and C. Vicari. "Distributed dynamic replica placement and request redirection in content delivery networks", MASCOTS, pp. 366–373, 2007.

[10] T. A. Neves, L. M. A. Drummond, L. S. Ochi, C. Albuquerque, and E. Uchoa. "Solving Replica Placement and Request Distribution in Content Distribution Networks", Electronic Notes in Discrete Mathematics, Volume 36, pp. 89-96, ISSN 1571-0653, 2010.

[11] L. Qiu, V. Padmanabhan, and G. Voelker. "On the Placement of Web Server Replicas". Proceedings of IEEE INFOCOM, pages 1587–1596, April 2001

[12] Y. Zhu, and M. Ammar. "Algorithms for assigning substrate network resources to virtual network components", IEEE INFOCOM, 2006.

[13] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown. ElasticTree: saving energy in data center networks. In NSDI, 2010.