Computer Networks 91 (2015) 508–527

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Network-aware virtual machine placement in cloud data centers with multiple traffic-intensive components

Amir Rahimzadeh Ilkhechi [a,*], Ibrahim Korpeoglu [b], Özgür Ulusoy [c]

[a] Computer Engineering Department, Duke University, 1505-Duke University Road, Apt #4k, Durham, North Carolina, ZIP code: 27701, United States
[b] Computer Engineering Department, Bilkent University, office at Engineering EA-401, Ankara, Turkey
[c] Computer Engineering Department, Bilkent University, office at Engineering EA-402, Ankara, Turkey

ARTICLE INFO

Article history:
Received 27 November 2014
Revised 14 August 2015
Accepted 27 August 2015
Available online 11 September 2015

Keywords:
Cloud computing
Virtual machine placement
Sink node
Predictable flow
Network congestion

ABSTRACT

Following a shift from computing as a purchasable product to computing as a deliverable service to consumers over the Internet, cloud computing has emerged as a novel paradigm with an unprecedented success in turning utility computing into a reality. Like any emerging technology, with its advent, it also brought new challenges to be addressed. This work studies network and traffic aware virtual machine (VM) placement in a special cloud computing scenario from a provider's perspective, where certain infrastructure components have a predisposition to be the endpoints of a large number of intensive flows whose other endpoints are VMs located in physical machines (PMs). In the scenarios of interest, the performance of any VM is strictly dependent on the infrastructure's ability to meet their intensive traffic demands. We first introduce and attempt to maximize the total value of a metric named "satisfaction" that reflects the performance of a VM when placed on a particular PM. The problem of finding a perfect assignment for a set of given VMs is NP-hard and there is no polynomial time algorithm that can yield optimal solutions for large problems. Therefore, we introduce several off-line heuristic-based algorithms that yield nearly optimal solutions given the communication pattern and flow demand profiles of subject VMs. With extensive simulation experiments we evaluate and compare the effectiveness of our proposed algorithms against each other and also against naïve approaches.

1. Introduction

The problem of appropriately placing a set of Virtual Machines (VMs) into a set of Physical Machines (PMs) in distributed environments has been an important topic of interest for researchers in the area of cloud computing. The proposed approaches often focus on various problem domains with different objectives: initial placement [1–3], throughput maximization [4], consolidation [9,10], Service Level Agreement (SLA) satisfaction versus provider operating costs minimization [11], etc. [5]. Mathematical models are often used to formally define the problems of that category. Then, they are normally fed into solvers operating based on different approaches including but not limited to greedy, heuristic-based or approximation algorithms. There are also well-known optimization tools such as CPLEX [12], Gurobi [15] and GLPK [17] that are predominantly utilized in order to solve placement problems of small size.

There is also another way of classifying the works related to VM placement based on the number of cloud environments: *Single-cloud* environments and *Multi-cloud* environments.

* Corresponding author. Tel.: +1 9196997984.
E-mail addresses: ilkhechi@cs.duke.edu (A.R. Ilkhechi), korpe@cs.bilkent.edu.tr (I. Korpeoglu), oulusoy@cs.bilkent.edu.tr (Ö. Ulusoy).

http://dx.doi.org/10.1016/j.comnet.2015.08.042
1389-1286/

The first category is mostly concerned with service to PM assignment problems which are often NP-hard in complexity. That is, given a set of PMs and a set of services that are encapsulated within VMs with fluctuating demands, design an online placement controller that decides how many instances should run for each service and also where the services are assigned to and executed in, taking into account the resource constraints. Several approximation approaches have been introduced for that purpose including the algorithm proposed by Tang et al. in [16].

The second category, namely the VM placement in multiple cloud environments, deals with placing VMs in numerous cloud infrastructures provided by different Infrastructure Providers (IPs). Usually, the only initial data that is available for the Service Provider (SP) is the provision-related information such as types of VM instances, price schemes, etc. Without any information about the number of physical machines, the load distribution, and other such critical factors inside the IP side mostly working on VM placement across multi-cloud environments are related to cost minimization problems. As an example of research in that area, Chaisiri et al. [18] propose an algorithm to be used in such scenarios to minimize the cost spent in each placement plan for hosting VMs in a multiple cloud provider environment.

To begin with, our work falls into the first category that pertains to single cloud environments. Based on this assumption, we can take the availability of detailed information about VMs and their profiles, PMs and their capacities, the underlying interconnecting network infrastructure and all related for granted. Moreover, we concentrate on network rather than data center/server constraints associated with VM placement problem.

This paper introduces nearly optimal placement algorithms that map a set of virtual machines (VMs) into a set of physical machines (PMs) with the objective of maximizing a particular metric (named *satisfaction*) which is defined for VMs in a special scenario. The details of the metric and the scenario are explained in Section 3 while also a brief explanation is provided below. The placement algorithms are offline and assume that the communication patterns and flow demand profiles of the VMs are given. The algorithms consider network topology and network conditions in making placement decisions.

Imagine a network of physical machines in which there are certain nodes (physical machines or connection points) that virtual machines are highly interested in communicating. We call these special nodes "sinks", and call the remaining nodes "Physical Machines (PMs)". Despite the fact that sink usually is a receiver node in networks, we assume that flows between VMs and sinks are bidirectional.

As illustrated in Fig. 1, assuming a general unstructured network topology, some small number of nodes (shown as cylinder-shaped components) are functionally different than the rest. With a high probability, any VM to be placed in the ordinary PMs will be somehow dependent on at least one of the sink nodes shown in the figure. By dependence, we mean the tendency to require massive end-to-end traffic between a given VM and a sink that the VM is dependent on. With that definition, the intenser the requirement is, the more dependent the VM is said to be.

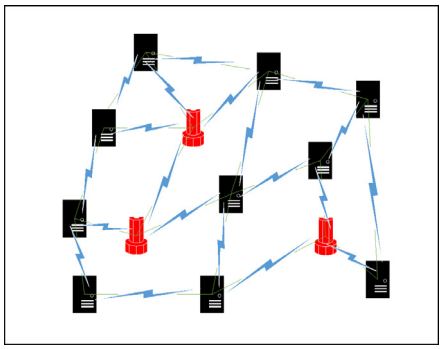


Fig. 1. Interconnected physical machines and sink nodes in an unstructured network topology.

The network connecting the nodes can be represented as a general graph $G(V, E)$ where E is the set of links, V is the set of nodes and S is the set of sinks (note that $S \in V$). On the other hand, the number of normal PMs is much larger than the number of sinks (i.e. $|S| \ll |V - S|$).

Each link consisting of end nodes u_i and u_j is associated with a capacity c_{ij} that is the maximum flow that can be transmitted through the link.

Assume that the intensity of communication between physical machines is negligible compared to the intensity of communication between physical machines and sinks. In such a scenario, the quality of communication (in terms of delay, flow, etc.) between VMs and the sinks is the most important factor that we should focus on. That is, placing the VMs on PMs that offer a better quality according to the demands of the VMs is a reasonable decision. Before advancing further, we suppose that the following a priori information is given about any VM:

- *Total Flow*: the total flow intensity that the VM will demand in order to achieve perfect performance (for sending to and/or receiving data from sinks).
- *Demand Weight*: for a particular VM (vm_i), the weights of the demands for the sinks are given as a demand vector $V_i = (v_{i1}, v_{i2}, \ldots, v_{i|S|})$ with elements between 0 and 1 whose sum is equal to 1. (v_{ik} is the weight of demand for sink k in vm_i).

Moreover, suppose that each PM-sink pair is associated with a numerical cost. It is clearly not a good idea to place a VM with intensive demand for sink x in a PM that has a high cost associated with that sink.

Based on these assumptions, we define a metric named satisfaction that shows how "satisfied" a given virtual machine v is, when placed on a physical machine p.

By maximizing the overall *satisfaction* of the VMs, we can claim that both the service provider and the service consumer sides will be in a win-win situation. From consumer's point of view, the VMs will experience a better quality of service which is a catch for users. Similarly, on the provider side, the links will be less likely to be saturated which enables serving more VMs.

The placement problem in our scenario is the complement of the famous Quadratic Assignment Problem (QAP) [19] which is NP-hard. On account of the dynamic nature of

the VMs that are frequently commenced and terminated, it is impossible to arrange the sinks optimally in a constant basis, since it requires physical changes in the topology. So, we instead attempt to find optimal placement (or actually nearly-optimal placement) for the VMs which is exactly the complement of the aforementioned problem. We propose greedy and heuristic based approaches that show different behaviors according to the topology (Tree, VL2, etc.) of the network.

We introduce two different approaches for the placement problem including a greedy algorithm and a heuristic-based algorithm. Each of these algorithms have two different variants. We test the effectiveness of the proposed algorithms through simulation experiments. The results reveal that a closer to optimal placement can be achieved by deploying the algorithms instead of assigning them regardless of their needs (random assignment). We also provide a comparison between the variants of the algorithms and test them under different topology and problem size conditions.

The rest of this paper includes a literature review (Section 2) followed by the formal definition of the problem in hand (Section 3). In Section 4, 4 algorithms for solving the problem are provided. Experimental results and evaluations are included in Section 5. Finally, Section 6 concludes the paper and proposes some potential future work.

2. Related work

There are several studies in the literature that are closely related to our work in the sense that they attempt to improve the performance of a given data center by choosing which physical machines accommodate which virtual machines. In this section, we mention such past works categorized according to their relevance to our work as well as their relevance to each other. To the best of our knowledge, there are no past works that study the scenario of our interest.

2.1. Network-aware placement related work

The most relevant past work is [25] by R. Cohen et al. In their work, they concentrate merely on the networking aspects and consider the placement problem of virtual machines with intense bandwidth requirements. They focus on maximizing the benefit from the overall traffic sent by virtual machines to a single point in the data center which they call root. In a storage area network of applications with intense storage requirements, the scenario that is described in their work is very likely. They propose an algorithm and simulate on different widely used data center network topologies. We realized that the defined problem in the mentioned work is very limited though the scenario itself in its general form is significant. Then, we came up with the problem that is studied in this paper, by generalizing the mentioned problem into a scenario in which there can be more than one root or sink.

The following works also consider network related constraints of the placement problem, but their defined scenarios are less related to our work.

Kuo et al. [6] introduces VM placement algorithms for a scenario that is related to MapReduce/Hadoop architecture. In this paper, the scenario is as follows: suppose that we have a data center consisting of many data nodes (DNs) and computation nodes (CN). Each computation node has several available VMs. Users data chucks are stored in some DNs and they may request VMs to process their data whose location is fixed and given in advance. The problem is to assign VMs to DNs such that the maximum access latency between the DNs and VMs and also between the VMs is bounded. There are several notable differences between [6] and our work: to begin with, the cost function defined in the mentioned work takes only delay into account while in our work we are concerned with both bandwidth and delay (i.e., the cost in our work is defined as a function of both delay and bandwidth, and each can be given weights). The other difference is that [6] does not assume that VMs compete for bandwidth in order to access a given data node while in our work we make such an assumption (the VMs compete for sinks instead of data nodes). In other words, in our scenario, any placement decision can potentially affect the performance of other VMs as well. Moreover Kuo et al. [6] assume that each VM is interested in only one DN and each DN is only accessed by a single VM. In our scenario however, the sinks that are equivalent to DNs can be requested by many competing nodes simultaneously.

In another work [21], Biran et al. contend that VM placement has to carefully consider the aggregated resource consumption of co-located VMs in order to be able to honor Service Level Agreements (SLA) by spending comparatively fewer costs. Biran et al. [21] are focused on both network and CPU-memory requirements of the VMs, but it only takes general constraints of the network such as network cuts into account while we believe that bandwidth related factors need to be studied as well.

Teyeb et al. [7] study VM placement problem in geographically distributed data centers with tenants requiring a set of networking VMs. In their work, an ILP formulation of the placement problem is provided that takes location and system performance constraints into account. In such a placement problem, there is a trade-off between efficiency and user experience since there may be delay between users and data centers. The objective is to minimize the traffic generated by networking VMs circulating on the backbone network. The mentioned work employs the simplified form of the formulation for Hub Location problem discussed in [8] to find an optimal placement. Teyeb et al. [7] make some assumptions about the distributed data centers that seem unrealistic. For example, by assuming that the distributed data center parts do not send traffic to each other they simplify the problem.

Similarly, [14] is another work (very closely related to [7]) that studies VM placement in geographically distributed data centers. The mentioned work aims to minimize IP-traffic within a given backbone network by placing VMs in data centers that are connected over an IP-over-WDM network. Again, Teyeb et al. [14] make assumptions that simplify the original problem of placing VMs in geographically distributed data centers.

2.2. Other VM placement related work

There are some less related works that also focus on network aspects of cloud computing but from different standpoints such as routing, scalability, connectivity, load balancing and alike.

In [13] the problem of sharing-aware VM maximization in a general sharing model is studied. The objective is to find a subset of potential VMs that can be hosted by a server with a given memory capacity with the goal of maximizing the total profit. In the mentioned paper, a greedy approximation algorithm is proposed for solving the problem.

The scalability of data centers has been carefully studied by X. Meng et al. in their work [22]. They propose a traffic-aware Virtual Machine placement to improve the network scalability. Unlike past works, their proposed methods do not require any alterations in the network architecture and routing protocols. They suggest that traffic patterns among VMs can be better matched with the communication distance between them. They formulate the VM placement as an optimization problem and then prove its hardness. In the mentioned work, a two-tier approximate algorithm is proposed that solves the VM placement problem efficiently.

3. Formal problem definition

We are interested in the problem of finding an optimal assignment of a set of Virtual Machines (VMs) into a set of Physical Machines (PMs) (assuming that the number of PMs is greater than or at least equal to that of VMs) in a special scenario with the objective of maximizing a metric that we define as *satisfaction*. In the following sections, the scenario of interest, assumptions, the defined metric, and mathematical description of the problem is provided, respectively.

3.1. The scenario

Heterogeneity of interconnected physical resources in terms of computational power and/or functionality is not too unlikely in cloud computing environments [27]. If we refer to any server (or any connection point) in Data Center Network (DCN) as a node, assuming that the nodes can have different importance levels is also a reasonable assumption in some situations. Note that here, since we are concerned with network constraints and aspects, by importance level we mean the intensity of traffic that is expected to be destined for a subject node. In other words, if VMs have a higher tendency to initiate traffics to be received and processed by a certain set of nodes (call it S), we say that the nodes belonging to that set have a higher importance (e.g., the cylinder-shaped servers shown in Fig. 1). Throughout the paper, those special nodes are called sinks. Besides, a sink can be a physical resource such as a supercomputer or it can be a virtual non-processing unit such as a connection point.

One can think of a sink as a physical resource (as is the case of Fig. 1) that other components are heavily dependent on. A powerful supercomputer capable of executing quadrillions of calculations per second [28] can be considered a physical resource of high importance from network's point of view. Such resources can also be functionally different from each other. While a particular server X is meant to process visual information, server Y might be used as a data encrypter.

However, in our scenario, a sink is not necessarily a processing unit or physical resource. In other words, it can also be a connection point to other clouds located in different regions meant for variety of purposes including but not limited to replication (Fig. 2). Suppose that in the mentioned scenario, every VM is somehow dependent on those sinks in that sense that there exists reciprocally intensive traffic transmission requirement between any VM-sink pair.

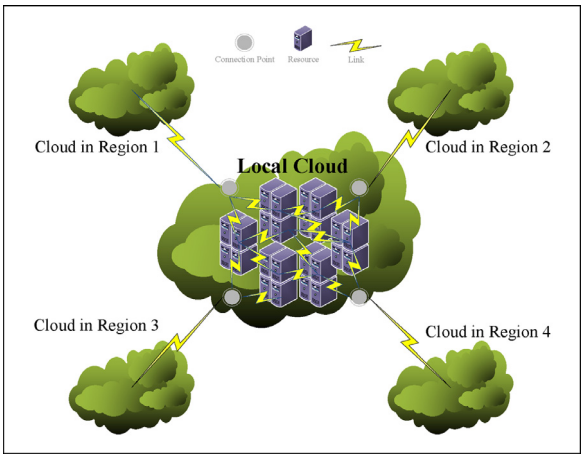


Fig. 2. Non-resource sinks.

Table 1
Sink demands of three VMs.

VM/Sink	S_1	S_2	S_3
VM_1	0.1	0.2	0.7
VM_2	0.5	0.05	0.45
VM_3	0.8	0.18	0.02

Regardless of the types of the sinks (resource or non-resource) the overall traffic request destined for them is assumed to be very intense. However, functional differences might exist between the sinks that can in turn result in a disparity on the VM demands. We assume that any VM has a specific demand weight for any given sink.

In the subject scenario, the tendency to transmit unidirectional and/or bidirectional massive traffic to sinks is so high that it is the decisive factor in measuring a VM's efficiency. Also Service Level Agreement (SLA) requirements are satisfied more suitably if all the VMs have the best possible communication quality (in terms of bandwidth, delay, etc.) with the sinks commensurating their per sink demands. For example, in Table 1 three virtual machines are given associated with their demands for each sink in the network. An appropriate placement must honor the needs of the VMs by placing any VM as close as possible to the sinks that they tend to communicate with more intensively (e.g., require a tenser flow).

3.2. Assumptions

The scenario explained in Section 3.1 is dependent upon several assumptions that are explained below.

- *Negligible Inter-VM Traffic*: the core presupposition that our scenario is based on is assuming that the sinks play a significant role as virtual or real resources that VMs in hand attempt to acquire as much as possible. Access to the resources is limited by the network constraints and

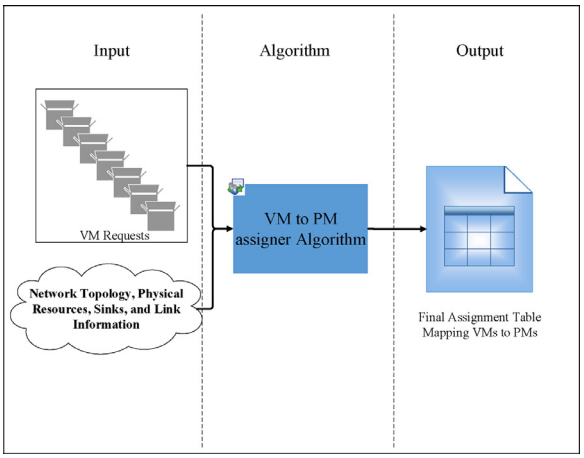


Fig. 3. Off-line virtual machine placement.

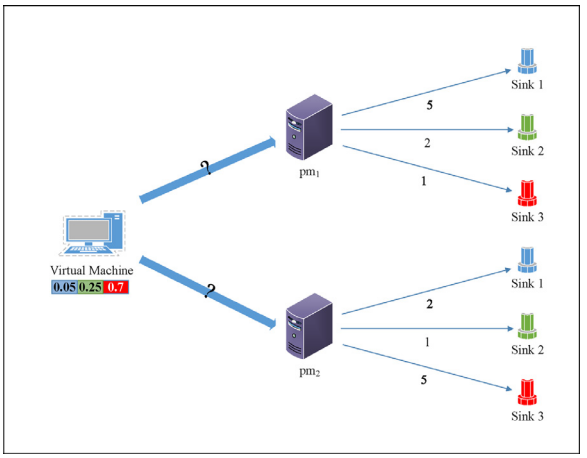


Fig. 4. A simple example of placement decision.

from a virtual machine's point of view, proximity of its host PM (in terms of cost) to the sinks of its interest matters the most. Therefore, we implicitly make an assumption on the negligibility of inter-VM dependency meaning that VMs do not require to exchange very huge amounts of data between themselves. If we denote the amount of flow that VM vm_i demands for sink s_j by $D(i, j)$, and similarly denote the amount of flow that VM vm_k demands for another VM vm_l by $D'(k, l)$, then Relation 1 must hold where i, j, k, and l are possible values (i.e., $i \leq$ number of VMs, and $j \leq$ number of sinks):

$$D'(k, l) \ll D(i, j), \quad \forall i, j, k, l \tag{1}$$

- *Availability of VM Profiles*: whether by means of long term runs or by analyzing the requirements of VMs at the coding level, we assume that the sink demands of the VMs based on which the placement algorithms operate are given. In other words, associated with any VM to be placed is a vector called demand vector that has as many entries as the number of sinks.

  Suppose that the sinks are numbered and each entry on any demand vector corresponds to the sink whose number is equal to the index of the entry. Entries in the demand vectors are the indicators of relative importance of corresponding sinks. The value of each entry is a real value in the range [0,1] and the summation of entries in any demand vector is equal to 1. In addition to demand vector, we suppose that a priori knowledge about the total sink flow demand of any VM is also given. Sink flow demand for a particular VM vm_x is defined as the total amount of flow that vm_x will exchange with the sinks cumulatively.
- *Off-line Placement*: the placement algorithms that we propose are off-line meaning that given the information about the VMs and their requirements, network topology, physical machines, sinks, and links, the placement happens all in once as shown in Fig. 3.
- *One VM per a PM*: we suppose that our proposed algorithms take a group of consolidated VMs as input so that each PM accommodates only one big VM. Although this assumption may sound unrealistic, it is always possible to consolidate several VMs as a single VM [25]. In other words, by allowing the VM placement to be performed in two different stages, we can achieve a better result using machine level placement algorithms (e.g., [9,10,23,24]) alongside network related algorithms that we propose. In real world scenarios, CPU and memory capacity limits of each host determine the number of VMs that it can accommodate. Therefore, a different and straightforward approach is to bundle all VMs that can be placed in a single host into one logical VM with accumulated bandwidth requirements. In both cases we can thus assume without the loss of generality throughout the paper that each PM can accommodate a single VM.

3.3. Satisfaction metric

The placement problem in our scenario can be viewed from two different stakeholders' perspectives: from a Service Provider's standpoint, an appropriate placement is the one that honors the virtual machines' demand vectors. Comparably, Infrastructure Provider tries to maximize the locality of the traffics and minimize the flow collisions. Fortunately, in our scenario, the desirability of a particular placement from both IP and SP viewpoints are in accordance: any placement mechanism that respects the requirements of the VMs (their sink demands basically), also provides more locality and less congestion in the IP side.

We define a metric that shows how satisfied a given VM vm_i becomes when it is placed on PM pm_j. In our scenario, satisfaction of a virtual machine depends on the appropriateness of the PM that it is placed on according to its demand vector. Any PM-sink pair is associated with a cost. Likewise, there is a demand between any VM-sink pair that shows how important a given sink for a given VM is. A proper placement should take into account the proximity of VMs to the sinks according to their significance. Here, by proximity we mean the inverse of cost between a PM and a sink: a lower cost means a higher proximity.

As an example, suppose that we have one VM and two options to choose from (Fig. 4): pm_1 or pm_2. In this example, there are three sinks in the whole network. The VM is given

together with its total flow demand and demand vector. The costs between pm_1 and all the other sinks supports the suitability of that PM to accommodate the given VM because more important sinks have a smaller cost for pm_1. Sinks 3, 2 and 1 with corresponding significance values 0.7, 0.25, 0.05 are the most important sinks, respectively. The cost between pm_1 and *sink 3* is the least among the three cost values between that PM and the sinks. The next smallest costs are coupled with *sink 2* and *sink 1*, respectively. If we compare those values with the ones between pm_2 and the sinks, we can easily decide that pm_1 is more suitable to accommodate the requested VM. If we sum up the values resulted by dividing the value of each sink in the demand vector of a VM to the cost value associated with that sink in any potential PM, then we can come up with a numerical value reflecting the desirability of that PM to accommodate our VM. For now, let's denote this value by $x(vm, pm)$ which means the desirability of physical machine pm for virtual machine vm. The desirability of pm_1 and pm_2 for the given VM request in our example can be calculated as follows (Eqs. 2 and 3):

$$x(vm, pm_1) = \frac{0.05}{5} + \frac{0.25}{2} + \frac{0.7}{1} = 0.835 \tag{2}$$

$$x(vm, pm_2) = \frac{0.05}{2} + \frac{0.25}{1} + \frac{0.7}{5} = 0.415 \tag{3}$$

From these calculations, it is clearly understandable that placing the requested VM on pm_1 will satisfy the demands of that VM in a better manner.

Based on that intuition, given a VM vm with demand vector V including entries $v_1, \ldots, v_{|S|}$, a set of PMs $P = \{pm_1, \ldots, pm_{|P|}\}$, the set of sinks $S = \{s_1, \ldots, s_{|S|}\}$, a static cost table D (we also call it function D interchangeably) with entries $D(i, j) = d_{ij}$ indicating the static cost between pm_i and s_j, and a dynamic cost function $G(pm, s, vm)$ that returns the dynamic cost between PM pm and sink s when vm is placed on pm, we define the *satisfaction* metric as a function of demand vector, static and dynamic costs. For now, let's suppose that we have a single sink s_x, a VM vm and a PM pm_y. The satisfaction of vm when placed on pm_y, is inversely proportional to static and dynamic costs between pm_y and s_x. If we represent satisfaction of vm on pm_y as $sat(vm, pm_y)$, then we have the following (Relation 4):

$$Sat(vm, pm_y) \propto 1/f(G(pm_y, s_x, vm), d_{yx}) \tag{4}$$

In Relation 4, function f is any linear or nonlinear and non-decreasing function of G and D. The choice of f is dependent upon many factors like the sensitivity of a VM's performance to static and dynamic costs. Without the loss of generality, we suppose that $f = G.D$ (it means dynamic cost of a placement multiplied by the corresponding static cost) throughout the paper. One may define f in different way (e.g., a linear combination of G and D) but still it should be a function of G and D that increases by increasing either static or dynamic costs. f can also be defined separately for different VMs depending on their applications. If f is properly defined, the proportionality in Relation 4 can be turned into equality (Relation 5):

$$Sat(vm, pm_y) = 1/f(G(pm_y, s_x, vm), d_{yx}) \tag{5}$$

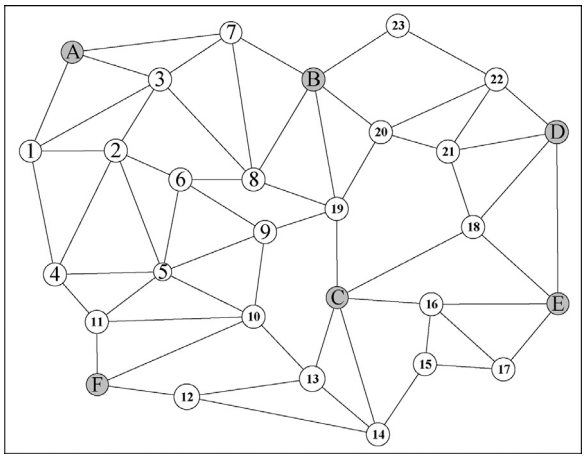


Fig. 5. A graph representing a simple data center network without an standard topology. The nodes named by alphabetic letters are the sinks.

More specifically, in this paper, for the provided example, we define Relation 6:

$$Sat(vm, pm_y) = \frac{1}{d_{yx} \times G(pm_y, s_x, vm)} \tag{6}$$

In a more realistic scenario, we may have more than one sink. Therefore, the *satisfaction* of a given VM can be defined as the weighted average of values returned by the function in Relation 5 for different sinks. If we define f as $f = G.D$, and use the demand vector of each VM for calculating the weighted average, then the general *satisfaction* function can be defined as Relation 7 for a VM vm with demand vector V that is placed on pm_i:

$$Sat(vm, pm_i) = \sum_{j=1}^{|S|} \frac{v_j}{d_{ij} \times G(pm_i, s_j, vm)} \tag{7}$$

The details of D table (or function D) and G function in Relation 7 are provided in the next section (Mathematical Description). Note that for the sake of simplicity but without the loss of generality, we assume that the static costs are as important as the dynamic costs in our scenario (e.g., according to the Relation 7, a PM p with static cost c_s and dynamic cost c_d associated with a sink s is as desirable as another PM p' with static cost $c'_s = \frac{1}{2}.c_s$ and dynamic cost $c'_d = 2.c_d$ associated with s, for any VM that has an intensive demand for s). Static and dynamic costs are of different natures and their combined effect must be calculated by a precisely defined function (i.e., function f) that depends on the sensitivity of the VMs to delay, congestion, and so on.

3.4. Mathematical description

The problem in hand can be represented in mathematical language. First of all, topology of the network is representable as a graph $G(V, E)$ where V is the set of all resources (including PMs and sinks) and E is the set of links (associated with some values such as capacity) between the resources (Fig. 5). In addition to the topology, we have the following information in hand.

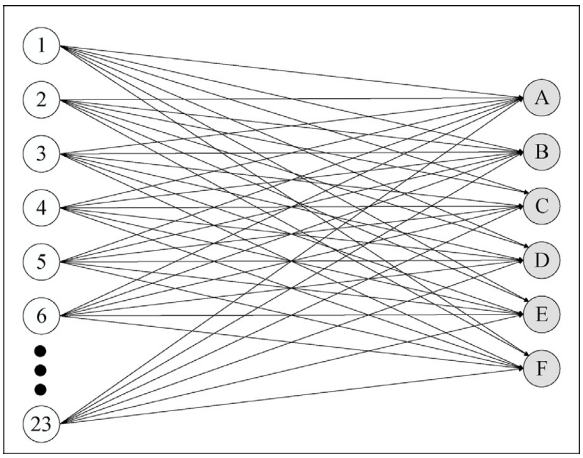


Fig. 6. A bipartite graph version of Fig. 5 representing the costs between PMs and sinks.

- Set $N = pm_1, pm_2, \ldots, pm_n$ consisting of physical machines.
- Set $M = vm_1, vm_2, \ldots, vm_m$ consisting of virtual machine requests.
- Set $S = s_1, s_2, \ldots, s_z$ consisting of sinks that are functionally not identical.

where Relations 8 and 9 hold:

$$|N| \geq |M| \tag{8}$$

$$|N| \gg |S| \tag{9}$$

In addition, any VM request has a sink demand vector and a total sink flow:

- f_i = total sink flow demand of vm_i. In other words, it is a number that specifies the amount of demanded total flow for vm_i that is destined for the sinks.
- $V_i = (v_{i1}, v_{i2}, \ldots, v_{i|S|})$ which is the demand vector for vm_i. In this vector,
  v_{ik} = the intensity of flow destined for s_k initiated from vm_i.

For any demand vector, we have the following relations (10 and 11):

$$\sum_{j=1}^{|S|} v_{ij} = 1, \qquad \forall i \tag{10}$$

$$0 \leq v_{ij} \leq 1, \qquad \forall i, j \tag{11}$$

All of the resources in the graph G can be separated into two groups, namely, normal physical machines and sinks (that can be special physical machines or virtual resources like connection points as explained in Section 3.1). With that in mind, as illustrated in Fig. 6, we can think of a bipartite graph $Gp = (N \cup S, E_p)$ whose:

- Vertices are the union of physical machines and sinks.
- Edges are weighted and represent the costs between any PM-sink pair.

The cost associated with any PM-sink pair is in direct relationship with static costs such as physical distance (e.g., it can be the number of hops or any other measure) and dynamic costs such as congestion as a result of link capacity saturation and flow collisions. Note that the use of the word congestion in our paper is different than its general meaning in Computer Networks jargon, in the sense that it never happens if the VMs do not initiate flows as much as they really need to (i.e. they back off if congestion really happens). Depending on different assignments, the cost value on the edges connecting the physical machines to the sinks can also change. For example, according to Figs. 5 and 6, if a new virtual machine is placed on PM #18 that has a very high demand for the sink C, then the cost between PM #23 and the sink C will also change most likely. Suppose that PM #23 uses two paths to transmit its traffic to sink C: $P_1 = \{22 - 21 - 18\}$ and $P_2 = \{22 - 20 - 19\}$ (excluding the source and destination). Placing a VM with an extremely high demand for sink C on 18 can cause a bottleneck in the link connecting 18 to the mentioned sink. As a result, PM #23 may have a higher cost for sink C afterwards, since the congested link is on P_1 which is used by PM #23 to send some of its traffic through. Accordingly, we define:

- *D Matrix*: a matrix representing the static costs between any PM-sink pair which is an equivalent of the example bipartite graph shown in Fig. 6 in its general case. An entry D_{ij} stores the static cost between pm_i and s_j.
- *G Function*: for any VM, the desirability of a PM is decided not only according to static but also dynamic costs. $G(pm_i, s_j, vm_k)$: $N \times S \times M \rightarrow R^+$ = congestion function that returns a positive real number giving a sense of how much congestion affects the desirability of pm_i when vm_k is going to be placed on it, taking into account the past placements. Congestion happens only when links are not capable of handling the flow demands perfectly. The G function returns a number greater than or equal to 1 which shows how well the links between a PM-sink pair are capable of handling the flow demands of a particular VM. If the value returned by this function is 1, it means that the path(s) connecting the given PM-sink pair won't suffer from congestion if the given VM is placed on the corresponding PM. Because the value returned by G is a cost, a higher number means a worse condition. Implicitly, G is also a function of past placements that dictate how network resources are occupied according to the demands of the VMs. While there is no universal algorithm for G function as its output is totally dependent on the underlying routing algorithm that is used, it can be described abstractly as shown in Fig. 7. According to that figure, the G function has four inputs out of which two of them are related to the assignment that is going to take place (VM Request and PM-sink pair) and the rest are related to past assignments and their effects on the network (the occupation of link capacity and so on).

Based on the underlying routing algorithm used, the inner mechanism of G function can be one of the followings.

- *Oblivious routing with single shortest path*: for such a routing scheme, the G function simply finds the most occupied link and divides the total requested flow over the capacity of the link (refer to Algorithm 1). If the value is less than or equal to 1, then it returns 1.

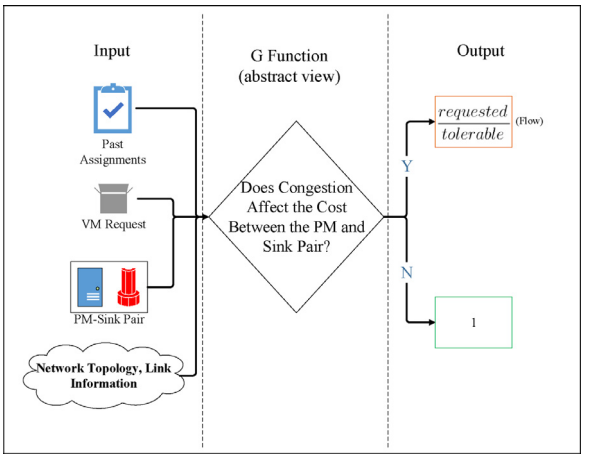


Fig. 7. The abstract working mechanism of *G* function.

Algorithm 1 $G(pm_i, s_j, vm_k)$: The congestion function for oblivious routing with single path.

1: *Path* ← the path connecting pm_i and s_j
2: *MinLink* ← the link in the *Path* that is occupied the most
3: *totReq* ← total flow request destined to pass through *MinLink*
4: c ← Total Capacity of *MinLink*
5: $G = \frac{totReq}{c}$
6: return max $(G, 1)$

Otherwise, it returns the value itself. According to Fig. 8, physical machines *X* and *Y* use static paths (1-2-3 and 1-2-4, respectively) to send their traffic to the sink. Suppose that among the links connecting *X* to the sink, only the link 1–2 is shared with a different physical machine (*Y* in that case). Link 1–2 is therefore the most occupied link and if we call the *G* function for a given VM, knowing that another VM is placed on *Y* beforehand, according to the demands of the previously placed VM and the VM that is going to be assigned to *Y*, *G* will return a value greater than or equal to 1 showing the capability of the bottleneck link of handling the total requested flow.

- *Oblivious routing with multiple shortest (or acceptable) paths*: if there are more than one static path between the PM-sink pairs and the load is equally divided between them, then the *G* function can be defined in a similar way with some differences: every path will have its own bottleneck link and the *G* function must return the sum of requested over total capacity of the bottleneck links in every path divided by the number of paths (Algorithm 2). Hence, if congestion happens in a single path, the overall congestion will be worsened less than the single path case. In Fig. 9, two different static paths (1-2-3, 6-7-8-9 for *X*, and 5-4, 1-7-8-9 for *Y*) have been assigned to each of the physical machines *X* and *Y*. The total flow is divided between those two paths and the congestion that happens in the links that are colored green, affects only one path of each PM.

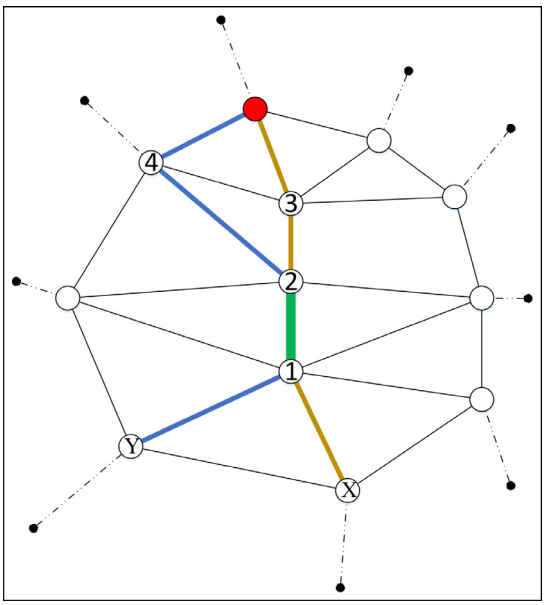


Fig. 8. A partial graph representing part of a data center network. The colored node represents a sink. Physical machines *X* and *Y* use oblivious routing to transmit traffic to the sink. The thickest edge (1–2) is the shared link. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Algorithm 2 $G(pm_i, s_j, vm_k)$: The congestion function for oblivious routing with multiple paths.

1: n ← number of paths connecting pm_i to $sink_j$
2: *TotG* ← 0
3: **for all** *Path* between pm_i and $sink_j$ **do**
4: *MinLink* ← the link in the *Path* that is occupied the most
5: *totReq* ← total flow request passing from *MinLink*
6: c ← Total Capacity of *MinLink*
7: $G = \frac{totReq}{c}$
8: *TotG* ← *TotG* + *G*
9: **end for**
10: return max $(\frac{TotG}{n}, 1)$

- *Dynamic routing*: defining a *G* function for dynamic routing is more complex and many factors such as load balancing should be taken into account. However, the heuristic that we provide for placement is independent from the routing protocol. *G* functions provided for oblivious routing can be applied to two famous topologies, namely Tree and VL2 [20].

We are now ready to give a formal definition of the assignment problem. The problem can be formalized as a 0–1 programming problem, but before we can advance further, another matrix must be defined for storing the assignments.

- *X matrix*: X: $M \times N \rightarrow \{0, 1\}$ is a two dimensional table to denote assignments. If $x_{ij} = 1$ it means that vm_i is assigned to pm_j.

The maximization problem given below (12) is a formal representation of the problem in hand as an integer (0–1) programming. Given an assignment matrix *X*, VM requests,

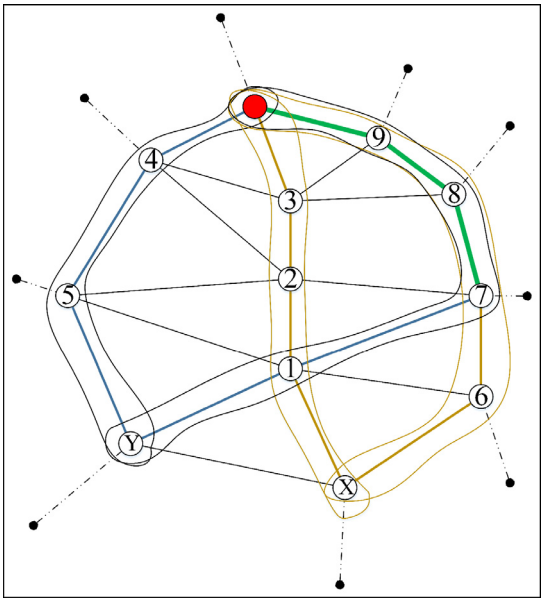


Fig. 9. The same partial graph as shown in Fig. 8, this time with a multi-path oblivious routing. Physical machines *X* and *Y* use two different static routes to transmit traffic to the sink. The thicker edges (7-8, 8-9, and 9-sink) are the shared links. The paths for *X* and *Y* are shown by light (brown) and dark (black) closed curves, respectively. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

topology, PMs, sinks and link related information the challenge is to fill the entries of the matrix X with 0s and 1s so that it maximizes the objective function and does not violate any constraint.

$$Maximize \sum_{i=1}^{|M|}\sum_{j=1}^{|N|} Sat(vm_i, pm_j).x_{ij} \tag{12}$$

$$Sat(vm_i, pm_j) = \sum_{k=1}^{|S|} \frac{v_{ik}}{d_{jk} \times G(pm_j, s_k, vm_i)}$$

Such that

$$\sum_{i=1}^{|M|} x_{ij} = 1, \quad \text{for all } j = 1, \ldots, |N| \tag{1}$$

$$\sum_{j=1}^{|N|} x_{ij} = 1, \quad \text{for all } i = 1, \ldots, |M| \tag{2}$$

$$x_{ij} \in \{0, 1\}, \quad \text{for all possible values of } i \text{ and } j \tag{3}$$

The constraints (1) and (2) ensure that each VM is assigned to exactly one PM and vice versa. Constraint (3) prohibits partial assignments. As explained before, function G gives a sense of how congestion will affect the cost between pm_j and s_k if vm_i is about to be assigned to pm_j.

We can represent the assignment problem as a bipartite graph that maps VMs into PMs. The edges connecting VMs to PMs are associated with weights which are the satisfaction of each VM when assigned to the corresponding PM. The weights may change as new VMs are placed in the PMs. It depends on the capacity of the links and amount of flow that each VM demands. Therefore, the weights on the mentioned bipartite graph may be dynamic if dynamic costs affect the decisions. If so, after finalizing an assignment, the weights of other edges may require alteration. Because congestion is in direct relationship with number of VMs placed, after any assignment we expect a non-decreasing congestion in the network. However, the amount of increase can vary by placing a given VM in different PMs. According to the capacity of the links, we expect to encounter two situations.

3.5. First case: no congestion

If the capacity of the links are high enough that no congestion happens in the network, the assignment problem can be considered as a linear assignment problem which looks like the following integer linear programming problem [29]. Given two sets, A and T (assignees and tasks), of equal size, together with a weight function C: $A \times T \to R$. Find a bijection f: $A \to T$ such that the cost function $\Sigma_{a \in A} C(a, f(a))$ is minimized:

$$Minimize \sum_{i \in A}\sum_{j \in T} C(i, j).x_{ij} \tag{13}$$

Such that

$$\sum_{i \in A} x_{ij} = 1, \quad \text{for } i \in A$$

$$\sum_{j \in T} x_{ij} = 1, \quad \text{for } j \in T$$

$x_{ij} \in \{0, 1\}$, for all possible values of i and j

In that case, the only factor that affects the satisfaction of a VM is static cost which is distance. The assignment problem can be easily solved by the Hungarian Algorithm [29] by converting the maximization problem into a minimization problem and also defining dummy VMs with total sink flow demand of zero if the number of VMs is less than the number of PMs.

3.6. Second case: presence of congestion

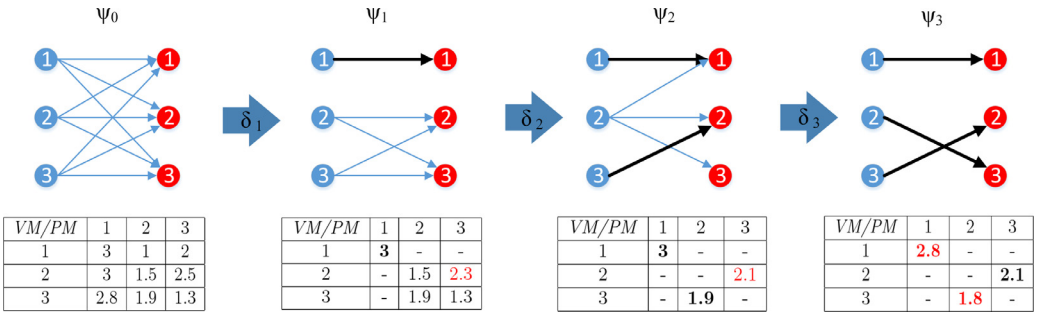
In that case, the maximization problem is actually nonlinear, because placing a VM is dependent on previous placements. From complexity point of view, this problem is similar to the Quadratic Assignment Problem [19], which is NP-hard. In Section 4, greedy and heuristic-based algorithms have been introduced to find an approximate solution for the defined problem when dynamic costs such as congestion are taken into account.

4. Proposed algorithms

In this chapter, we introduce two different approaches, namely *Greedy-based* and *Heuristic-based*, for solving the problem that is defined in Section 3.

4.1. Polynomial approximation for NP-hard problem

As explained in Section 3, the complexity of the problem in hand in its most general form is NP-hard. Therefore, there



VM/PM	1	2	3
1	3	1	2
2	3	1.5	2.5
3	2.8	1.9	1.3

VM/PM	1	2	3
1	**3**	-	-
2	-	1.5	2.3
3	-	1.9	1.3

VM/PM	1	2	3
1	**3**	-	-
2	-	-	2.1
3	-	**1.9**	-

VM/PM	1	2	3
1	2.8	-	-
2	-	-	**2.1**
3	-	1.8	-

Fig. 10. An example of sequential decisions.

is no possible algorithm constrained to both polynomial time and space boundaries that yields the best result. So, there is a trade-off between the optimality of the placement result and time/space complexity of any proposed algorithm for our problem.

With that in mind, we can think of an algorithm for placement task that makes sequential assignment decisions that finally lead to an optimal solution (if we model the solver as a non-deterministic finite state machine). In the scenario of interest, m virtual machines are required to be assigned to n physical machines. Since resulted by any assignment decision there is a dynamic cost that will be applied to a subset of PM-sink pairs, any decision is capable of affecting the future assignments. Making the problem even harder is the fact that even future assignments if not intelligently chosen, can also disprove the past assignments optimality.

On that account, given a placement problem $X = (M, N, S, T)$ in which $M =$ the set of VM requests, $N =$ the set of available PMs, $S =$ the set of sinks, $T =$ topology and link information of the underlying DCN, we can define a solution Ψ for the placement problem X as a sequence of assignment decisions: $\Psi = (\delta_1, \ldots, \delta_{|M|})$. Each δ can be considered as a temporally local decision that maps one VM to one PM. Let's assume that the total satisfaction of all the VMs is denoted by $TotSat(\Psi)$ for a solution Ψ. A solution Ψ_o is said to be an optimal solution if and only if $\nexists \Psi_x$, $suchthatTotSat(\Psi_x) > TotSat(\Psi_o)$. Note that it may not be possible to find Ψ_o in polynomial time and/or space.

Although we don't expect the outcome (a sequence of assignment decisions) of any algorithm that works in polynomial time and space to be an optimal placement, it is still possible to approximate the optimal solution by making the impact of future assignments less severe by intelligently choosing which VM to place and where to place it in each step. In other words, given VM–PM pairs as a bipartite graph $G_{vp} = (M \cup N, E_{vp})$ in which an edge connecting VM x to PM y represents the satisfaction of VM x when placed on PM y, any decision δ depending on the past decisions and the VM to be assigned, will possibly impact the weights between VM–PM pairs. The impact of δ can be represented by a matrix such as $I(\delta) = (i_{11}, \ldots, i_{1|N|}, \ldots, i_{|M||N|})$. Each entry i_{xy} represents the effect of decision δ on the satisfaction of VM x when placed on PM y. At the time that decision δ is made, if some of the VMs are not assigned yet, the impact of δ may change their preferences (impact of δ on future decisions). Likewise, given that before δ, possibly some other decisions such as δ' have been already made, the satisfaction of assigned VMs can also change (impact of δ on past decisions).

Let's denote a sequence of decisions $(\delta_1, \ldots, \delta_r)$ by a partial solution Ψ_r in which $r < |M|$. At any point, given a partial solution Ψ_r, it is possible to calculate $Sat(\Psi_r)$. If we append a new decision δ_{r+1} to the end of the decision sequence in Ψ_r, we can advance one step further (Ψ_{r+1}) and calculate the satisfaction of the new partial solution. If some of the elements in $I(\delta_{r+1})$ pertain to the already assigned VMs, then the satisfaction of these VMs will be affected. On the other hand, a new decision assigns a new VM to a new PM and the satisfaction of newly assigned VM must also be considered when calculating the $Sat(\Psi_{r+1})$. Briefly, if $Sat_\Delta(\delta_{r+1}|\Psi_r)$ denotes the additional satisfaction that decision δ_{r+1} brings, and similarly $Sat_I(I(\delta_{r+1}|X_{G_{vp}}))$ denotes the amount of loss of total satisfaction because of decision δ_{r+1} given past assignments of graph G_{vp} as matrix $X_{G_{vp}}$, then the Relation 14 exists:

$$Sat(\Psi_{r+1}) = Sat(\Psi_r) + Sat_\Delta(\delta_{r+1}|\Psi_r) - Sat_I(I(\delta_{r+1})|X_{G_{vp}}) \tag{14}$$

Fig. 10 delineates the decision process for a simple placement problem in which three VMs are supposed to be assigned to three PMs. The tables below each bipartite graph show the total weight of the edges connecting the VMs to the PMs (satisfaction of the VMs in other words). At the beginning where assignments are yet to be decided ($\Psi_0 = \varnothing$), the potential satisfaction of the VMs is at their maximum amount (i.e., there is no congestion). Since no decision has been made in Ψ_0, we have: $Sat(\Psi_0) = 0$. To make a transition from Ψ_0 to Ψ_1, decision δ chooses VM #1 and assigns it to PM #1. The new table below Ψ_1 shows that the weight between VM #2 and PM #3 is affected ($2.5 \rightarrow 2.3$) meaning that the congestion caused by VM #1 will degrade the potential satisfaction of VM #2 if it is placed on PM #3. In that level, $\Psi_1 = (\delta_1)$ and $Sat(\Psi_1) = 3$ since we have only one assigned VM whose satisfaction is equal to 3. Decision δ_2 assigns VM #3 into PM #2. This time, the potential satisfaction of VM #2 when placed on PM #3 is again declined ($2.3 \rightarrow 2.1$). At that point $Sat(\Psi_2) = 3 + 1.9 = 4.9$. Finally decision δ_3 assigns the only remaining VM–PM pair (#2 to #3). After the final assignment, the congestion caused by VM #2 affects the satisfaction of VMs #1 and #3 meaning that the decision δ_3 affects the past decisions. In other words, $Sat_I(\delta_3|\#1 \rightarrow \#1, \#3 \rightarrow \#2) \neq 0$. $Sat(\Psi_3)$ which is the total satisfaction of final solution Ψ_3

can be calculated as follows:

$$\begin{aligned} Sat(\Psi_3) &= Sat(\Psi_2) + Sat_{\Delta}(\delta_3|\Psi_2) - Sat_I(I(\delta_{r+1})|\#1 \\ &\quad \rightarrow \#1, \#3 \rightarrow \#2) \\ &= 4.9 + 2.1 - ((3 - 2.8) + (1.9 - 1.8)) \\ &= 6.7 \end{aligned}$$

4.2. Greedy approach

As the name suggests, in the Greedy approach we try to approximate the optimal solution Ψ_o for a placement problem X by making the best temporally local decisions expecting that the aggregated satisfaction of the VMs will be near to the maximum when all of them are assigned.

Each decision δ, assigns one VM to exactly one PM in a greedy manner. However, there is more to it than this: when making a decision, the selection of which VM to be assigned is also important. In our greedy approach, we sort the VMs according to their sink demands and then decisions are made by processing the sorted sequence of the VMs. Therefore, for any decision δ, selecting the next VM is straightforward. The sorting can be done according to:

- *Total Sink Flow Demand*: the VMs are sorted in descending order according to their total sink flow demands. Then, the VMs with higher sink flow demands are assigned first starting from the VM with the highest demand.
- *Sink-Specific Flow Demand*: the VMs are sorted according to their demands for different sinks: a descending ordered list of VMs according to their flow demands are created for every sink. In that case, the assignment starts by processing one VM at a time from the lists until no unassigned VM remains.

4.2.1. Intuitions behind the approach

The Greedy approach assumes that assigning the VMs with higher demands prior to the ones with lower demands alleviates the severity of negative effects that those highly demanding VMs will induce in the potential satisfaction of future VMs that wait to be assigned. In other words, if we try to assign the VMs with more intensive bandwidth/flow demand first, they will stay as local as possible and have a more moderate impact of the dynamic costs between the PMs and the sinks. Fig. 11 demonstrates how assignment of a particular VM can affect the overall congestion and enhance/diminish the performance of other VMs.

In the figure, VM #1 has a higher total sink demand and also a tendency to transmit most of its traffic to sink #1. If we let the decider module assign this VM first, then the mentioned VM will take the most appropriate PM for itself according to its demand. Another possibility is to let the VM #2 be assigned first. In that case, VM #1 will be assigned to a PM which has a higher static cost associated with the sinks of its interest. As a result, the overall congestion of the links will be higher because of less traffic locality. The thicker links represent a higher congestion. The link embraced by ellipses, might be required by some other PMs to transmit their traffics to other sinks. Accordingly, a lower overall congestion in the links can mean a lower dynamic cost for other PM-sink pairs.

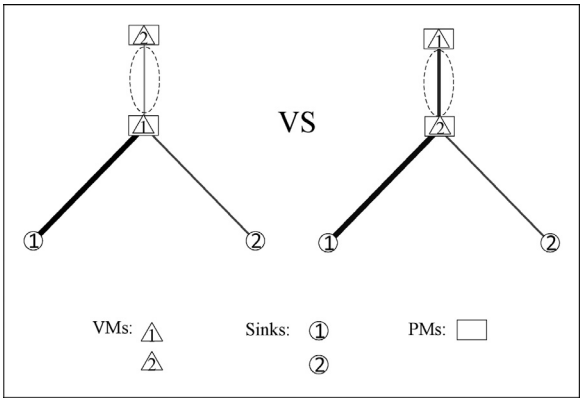


Fig. 11. Impact of assignment on the overall congestion.

4.2.2. The algorithm

Algorithm 3 shows the steps that are taken in greedy placement with total sink demand request sorting approach until all of the VMs are assigned.

Algorithm 3 Greedy Assignment Algorithm 1. Input: a list of VM requests *VMR*, DCN network information including link details, sinks and PMs.

1: $X \leftarrow$ the assignment matrix
2: **if** *VMR.length* $<$ # of PMs **then**
3: fill the *VMR* with dummy VMs
4: **end if**
5: sort the VMs in *VMR* according to their total sink demands in descending order
6: **while** there is VM left in *VMS* **do**
7: $v \leftarrow$ *VMS.removeHead()*
8: $p \leftarrow$ the PM that offers the highest satisfaction for v
9: $X_{vp} = 1$ (update the entry corresponding to v and p in the assignment table)
10: **end while**
11: return X

Another version of the greedy approach that sorts the VMs in different lists is given in Algorithm 4. This algorithm makes as many sorted list of VMs as the number of sinks. For a sink s, the sorted list corresponding to s contains the VMs sorted according to their total demand for sink s. Then, the algorithm finds the list with a head entry that has the highest demand, assigns the VM and removes it from any list. The idea is supported by the same intuition that is explained in Section 4.2.1 in a more strong way because this time we compare the VMs competing for common sinks. In both of the algorithms matrix X represents the assignment table with entries that can take values 0 or 1. Algorithm 3 takes a list of VM requests as input, adds some dummy request if required (i.e, if the number of VMs is less than number of PMs), and sorts the list according to the requests' total bandwidth demand. Then, it processes one request at a time and assigns it to a physical machine (line 8). The algorithm terminates when all of the VMs are assigned. Similarly, Algorithm 4 takes a list of VM placement requests as its input with the difference that it keeps a list of lists defined in the line 3 ($r[0..z]$) of the algorithm to store per-sink sorted

Algorithm 4 Greedy Assignment Algorithm 2. Input: a list of VM requests *VMR*, DCN network information including link details, sinks and PMs.

```
1: X ← the assignment matrix
2: z ← # of sinks in the DCN
3: r[0..z] ← an array of z lists of VMs
4: i ← 0
5: if VMR.length < # of PMs then
6:    fill the VMR with dummy VMs
7: end if
8: while i ≤ z do
9:    r[i]=sorted list [in descending order] of VMs in VMR according to their demands for sink i
10: end while
11: while there are all-zero rows in X do
12:    max ← 0
13:    maxIndex← 0
14:    for any VM list L_j in r \*(j=1,...,size(r))*\ do
15:       t ← L_j.getHead()
16:       if demand of t for sink j > max then
17:          max = demand of t for sink j
18:          maxIndex = j
19:       end if
20:    end for
21:    v ← r[maxIndex].removeHead()
22:    p ← the PM that offers the highest satisfaction for v
23:    X_vp = 1 (update the entry corresponding to v and p in the assignment table)
24:    remove v from any list in r
25: end while
26: return X
```

list of VM placement requests. Similar to Algorithm 3, it adds some dummy VM placement requests if necessary,and then, it fills the entries of the list r in lines 8–10 with sorted lists of per-sink total bandwidth demand. Note that entry i of r represents the list of VM placement requests sorted according to their bandwidth demands for sink i. The *while* loop statement in line 11 of the Algorithm 4 in each loop selects the VM-sink pair (v, s) in which v has the maximum demand (call it d) for s meaning that for any other pair (v', s') with demand d', we have: $d \geq d'$. The algorithm will terminate when all of the virtual machines are assigned (i.e., no all zeros rows remain in X) since in each loop a single VM request is processed and deleted from every list in r (line 24).

4.3. Heuristic-based approach

The greedy approach can be improved by ensuring that the temporally local decisions do not degrade the potential satisfaction of the VMs that will be assigned in future by considering their demand as a whole. In other words, the greedy approach does not take the demands of the unassigned VMs into account. Whenever an assignment decision is made, one VM goes to the group of assigned VMs and the remaining ones can be considered as another group with holistically defined demands. The core idea in heuristic-based approach is to make decisions that are best for the VM to be placed and at the same time the best possible for the remaining VMs. That is, when finding the most appropriate PM for a given VM, a punishment cost must be associated with any decision that tries to assign the VM to a PM that will cause congestion in the links connected to a highly requested sink. We use two different methods to measure the effect of a decision on the holistic satisfaction of the unassigned VMs.

- *Mean value VMs*: we can calculate the mean value of the total sink demands and also sink-based demands and come up with a virtual VM $\overline{vm}$ that represents a typical unassigned VM. When making a decision, we can assume that any remaining PM accommodates a $\overline{vm}$ and calculate the effect of assignment on the satisfaction of the virtual VMs. The more the degradation, the more severe the decision is penalized.
- *Greedily assigned unassigned VMs*: another way of letting the unassigned VMs play their role in assignment decisions is to virtually assign them with the greedy approach and then try to find the PM that maximizes the satisfaction of the VM to be placed, by taking into account the punishment cost that the VM receives for degrading the satisfaction of greedily assigned unassigned VMs. Intuitively, this approach has a better potential to reach a more proper placement at the end. However, the complexity of this method is higher.

4.3.1. Intuitions behind the approach

In many situations especially in data center networks that are based on a general non-standard topology, the greedy approach can be improved by selecting among best choices that will affect the future assignments with the least negative impact. According to Fig. 12, while two possible assignments maximize the satisfaction of the VM (shown as a triangle), the unassigned VMs may suffer more if the assignment decision opts to place the VM as shown in the left part of the figure.

The assignment decision at the left hand side has selected the PM that maximizes the satisfaction of the given VM using the greedy algorithm that is provided in the previous section. Although it seems like the most suitable PM to choose, it will affect the potential satisfaction of the VMs that will be placed in the remaining PMs in following decisions. Therefore, it is a better option to choose a different PM such that while maximizing the satisfaction of the subject VM, it also imposes the minimum performance degradation over the remaining VMs.

4.3.2. The algorithm

The Algorithm 5 shows the steps taken by the heuristic-based approach (with mean value VMs method) in a more detailed manner. It first starts by sorting the VMs according to one of the methods described in the greedy approach section. Afterwards, the VMs are processed one by one until no unassigned VM remains. Each time the remaining VMs are *virtually* assigned to the free PMs when evaluating the desirability of a particular PM for the given subject VM. Virtual assignment of the VMs can be done using one of the methods that have been discussed in this section (Mean value VMs or Greedily assigned unassigned VMs). Lines 1–21 are the same in Algorithms 5 and 4. In line 22 of the Algorithm 5, a virtual average VM request is calculated according to the demands of the remaining virtual machines and is stored in the variable $\overline{vm}$. A copy of the virtual VM request $\overline{vm}$ is then placed in

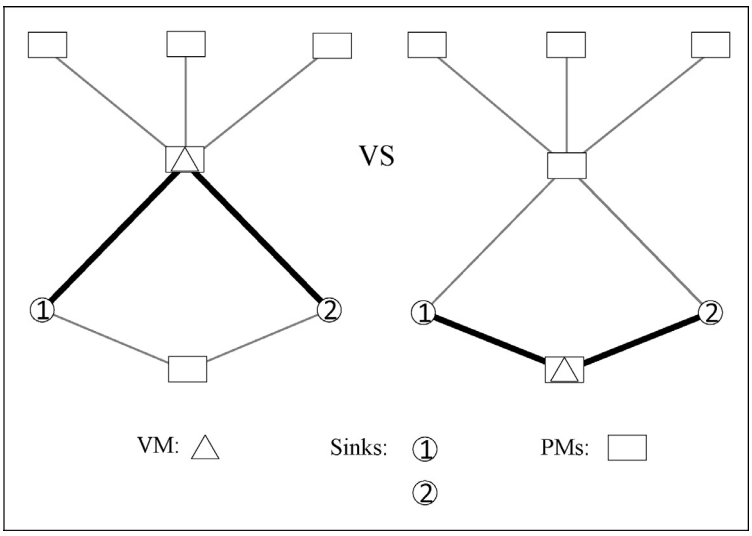


Fig. 12. Impact of an assignment decision on unassigned VMs.

Algorithm 5 Heuristic-based Assignment Algorithm. Input: a list of VM requests *VMR*, DCN network information including link details, sinks and PMs.

```
1: X ← the assignment matrix
2: z ← # of sinks in the DCN
3: r[0..z] ← an array of z lists of VMs
4: i ← 0
5: if VMR.length < # of PMs then
6:    fill the VMR with dummy VMs
7: end if
8: while i ≤ z do
9:    r[i] = sorted list of VMs in VMR according to their demands for sink i
10: end while
11: while there are all-zero rows in X do
12:    max ← 0
13:    maxIndex ← 0
14:    for any VM list L_j in r \*(j=1,...,size(r))*\ do
15:       t ← L_j.getHead()
16:       if demand of t for sink j > max then
17:          max = demand of t for sink j
18:          maxIndex = j
19:       end if
20:    end for
21:    v ← r[maxIndex].removeHead()
22:    vm̄ ← a virtual VM that is resulted by taking the mean value of total sink demands and sink-specific demands of all the remaining VMs
23:    virtually place vm̄ copies in all the remaining PMs
24:    p ← the PM that offers the highest satisfaction for v without vm̄
25:    release the virtually assigned vm̄(s)
26:    X_vp = 1 (update the entry corresponding to v and p in the assignment table)
27:    remove v from any list in r
28: end while
29: return X
```

every remaining PM. The PM that offers the best satisfaction without considering the $\overline{vm}$ assigned to it is selected. Then, in line 25 of the algorithm, the $\overline{vm}$ instances are released from the PMs to prepare the variables for the next loop. Finally, the assignment decision is recorded in the assignment matrix X and the current VM request is removed from any list in r. The loop continues until no unassigned VM remains and eventually it terminates.

4.4. Worst case complexity

The worst case time complexity analysis of the introduced algorithms are discussed in this section. Our algorithms include some operations that are either preprocessing or have a constant time complexity. To begin with, the shortest paths between any PM-sink pair can be found using famous methods like Floyd-Warshall algorithm which is of $\Theta(n^3)$ if n is the number of all nodes in the network graph. Besides, each time that the algorithm attempts to find the most appropriate PM, it calls the *G function* as many times as the number of sinks (which is strictly smaller than n). The G function needs to calculate the maximum possible flow capacity between two nodes. Even this operation can be considered of constant time because it examines only a subset of the edges that are less than a constant (e.g. if the oblivious routing is used, the number of checked links is less than or equal to the diameter of the network). Accordingly, the time complexity of the two approaches and their two variants can be formulated as follows:

- *Greedy Algorithm (first variant)*: if m denotes the number of VMs to be placed, then processing time for sorting the VMs according to their total sink demands is $O(m \log m)$. The algorithm takes one VM at a time from the sorted list and finds the most appropriate PM. If the number of the PMs is n, since the number of VMs is less than or equal to the number of PMs, we can conclude that the asymptotic time complexity of the algorithm is of $O(n^2)$.

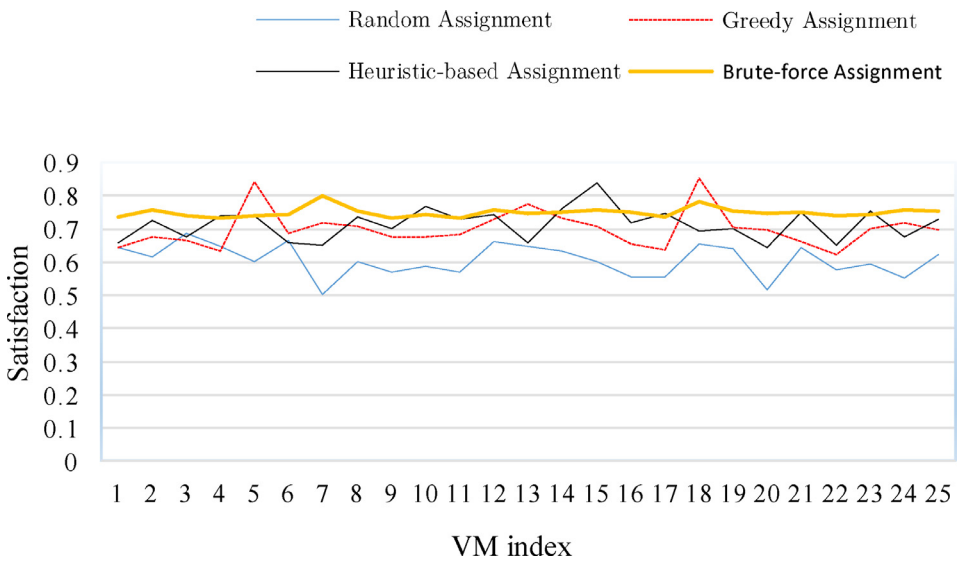


Fig. 13. A comparison of assignment methods in a data center with tree topology having 25 normal PMs and 5 sinks.

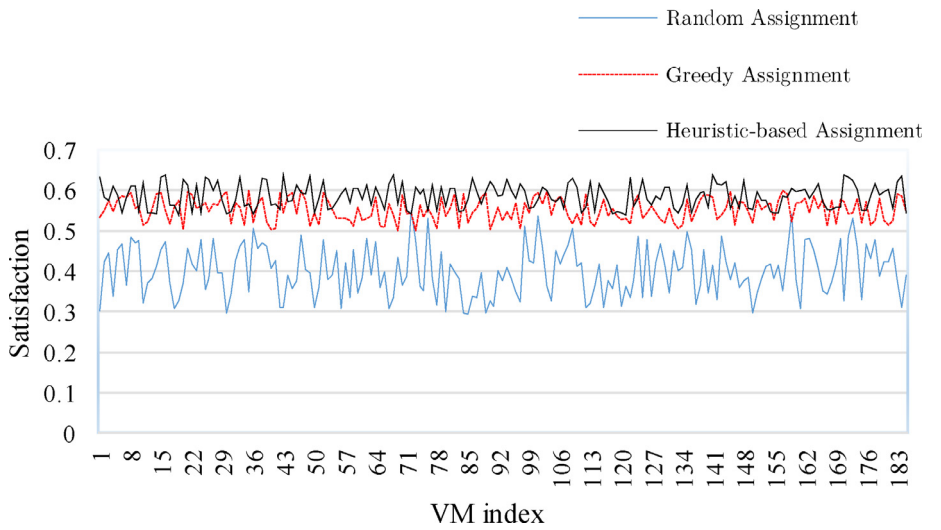


Fig. 14. A comparison of assignment methods in a data center with VL2 topology ($D_A = 4$, $D_I = 10$) having 185 normal PMs and 15 sinks.

- *Greedy Algorithm (second variant)*: if m denotes the number of VMs to be placed, then processing time for sorting the VMs z times according to their total sink demands is $O(zm\log m)$ where z is the number of sinks. The rest of the algorithm is the same as the first variant meaning that the asymptotic time complexity is of $O(n^2)$.
- *Heuristic-Based Algorithm (first variant)*: if m denotes the number of VMs to be placed, then processing time for sorting the VMs according to their total sink demands is $O(m\log m)$. The algorithm processes every VM only once and each time calculates a *mean VM* that is the average VM of remaining VM requests. It means that it does $O(n)$ operations each time it tries to assign a VM. Then, it places each copy of the average VM on the remaining machines which is of time complexity $O(n)$. Put together, the asymptotic complexity of this algorithm is $O(n^2)$.
- *Heuristic-Based Algorithm (second variant)*: if m denotes the number of VMs to be placed, then processing time for sorting the VMs according to their total sink demands is $O(m\log m)$. The algorithm processes every VM only once and each time assigns the remaining VMs virtually. The placement of the remaining VMs is of $O(n^2)$ time complexity. Therefore, the asymptotic time complexity of the algorithm is of $O(n^4)$ as it processes every VM only once, tries to find the most suitable PM for the VM being processed, while placing the remaining VMs virtually using the greedy algorithm in each trial.

5. Simulation experiment

We tested the effectiveness of our proposed approaches discussed in Section 4 using extensive simulation experiments. In this chapter, we report and discuss the

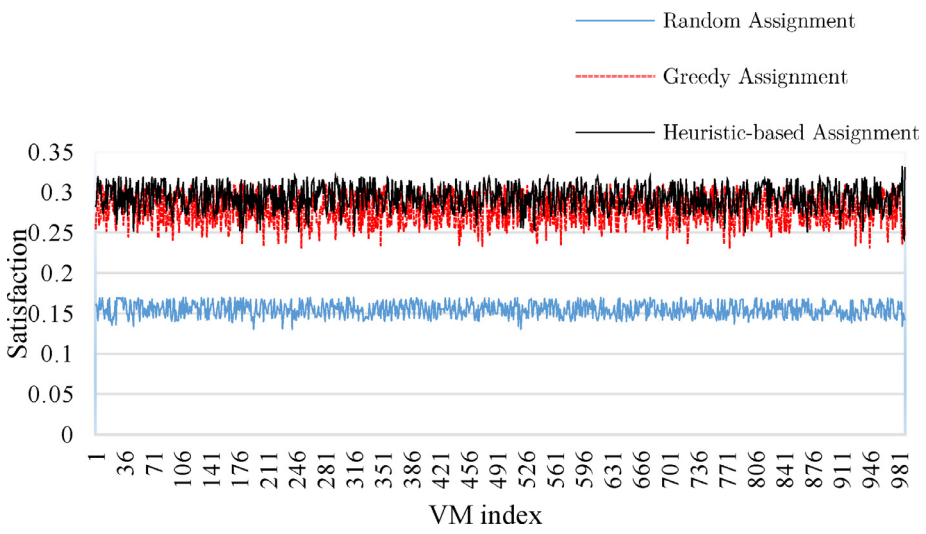


Fig. 15. A comparison of assignment methods in a data center with VL2 topology ($D_A = 4$, $D_I = 50$) having 990 normal PMs and 10 sinks.

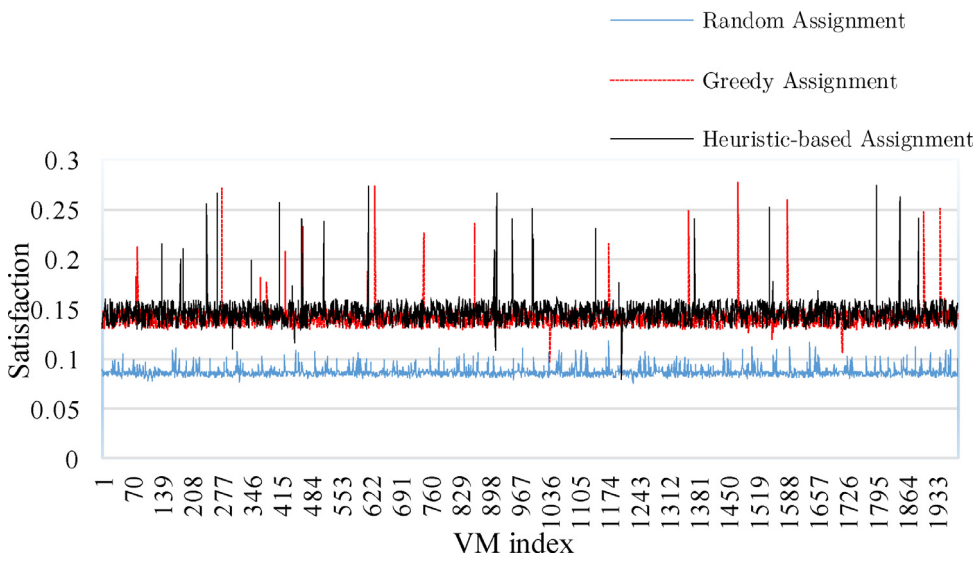


Fig. 16. A comparison of assignment methods in a data center with VL2 topology ($D_A = 8$, $D_I = 50$) having 1985 normal PMs and 15 sinks.

results of these experiments. To do simulations, we developed a customized simulator using Java by utilizing the JUNG open source library [30] to model and analyze graphs. We model the physical DCN infrastructure using graphs.

In the following sections, our Greedy and Heuristic-based assignment algorithms (their first variant) are compared against random and brute-force assignments (for small problem sizes). Brute-force assignment enumerates all possible assignments and selects the best one and as a result, it finds the optimal solution. It is, however, computationally expensive and is applied only for small-size networks. To the best of our knowledge, there are no past works that propose efficient algorithms for solving the placement problem in the scenario of interest. Therefore, we compare our algorithms against random and brute-force (for small instances of the problem) placements.

In Sections 5.1, 5.2 and 5.3, we provide a comparison of various assignment approaches for various problem sizes and topologies, demand distributions, and algorithm variants used, respectively. In Section 5.4, we show the correlation between total *satisfaction* and the overall congestion in the network.

5.1. Comparison based on problem sizes and topologies

In this section, we compare the behavior of the proposed algorithms in different problem sizes and topologies. To begin with, we test the algorithms on a very basic and simple data center with tree topology consisting of 25 physical machines and 5 sinks. The height of the tree is 2 and the links have a capacity of 1 Gbps as in [26]. While real and modern data centers do not usually use the simple tree structure, we use this test case as an example only. Fig. 13 shows the

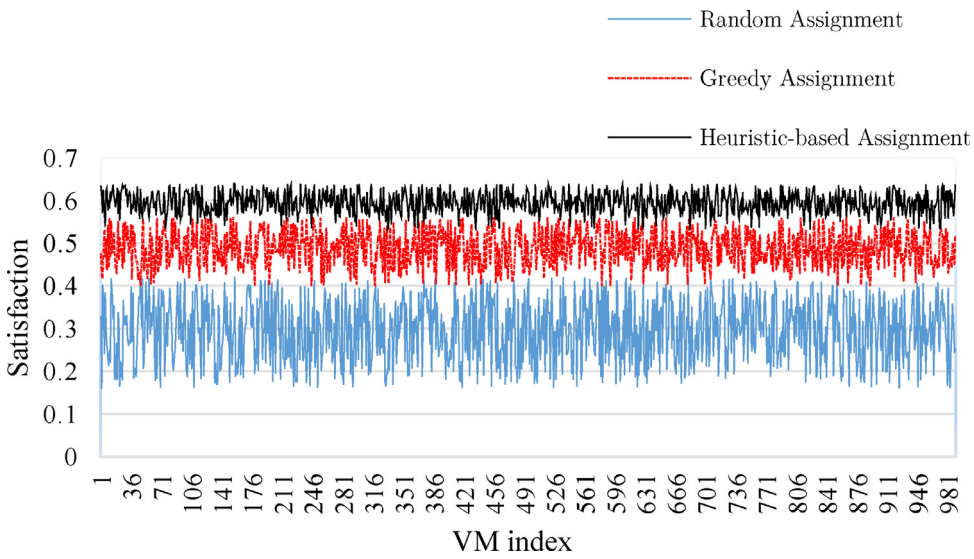


Fig. 17. A comparison of assignment methods in a data center with general topology having 990 normal PMs and 10 sinks.

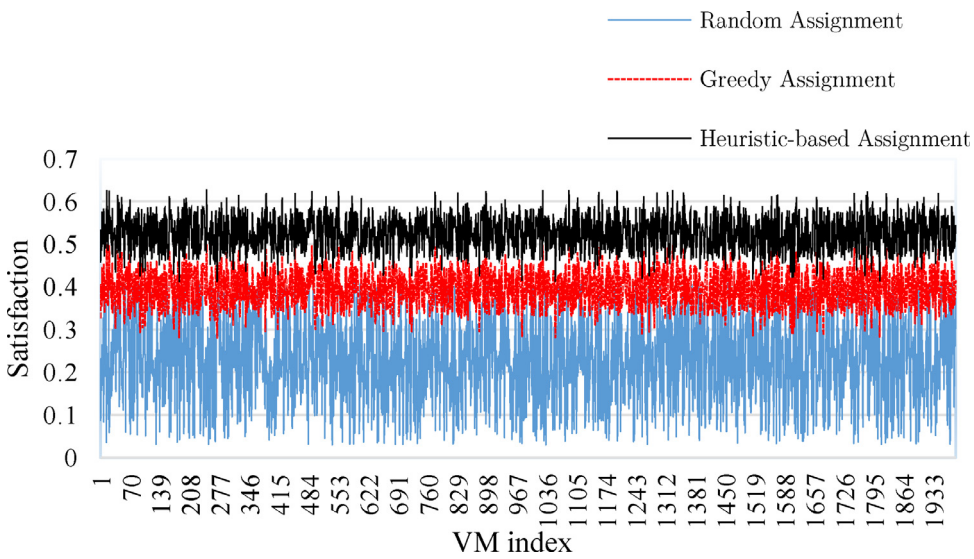


Fig. 18. A comparison of assignment methods in a data center with general topology having 1985 normal PMs and 15 sinks.

relative *satisfaction* of 25 VMs when placed on 25 physical machines using four different methods. While other placement methods yield a higher satisfaction for some machines, the overall satisfaction is obviously the highest using brute-force placement method. Similarly, the average *satisfaction* of the VMs are shown in Fig. 19a which clearly indicates that among three assignment methods (random, greedy, and heuristic-based) heuristic-based algorithm yields the closest result to the most optimal assignment attained by using the brute-force method. The Greedy algorithm also yields an approximated optimal assignment though it is slightly (about 4% in terms of average *satisfaction* achieved) less optimal than that of the heuristic-based algorithm.

The other topologies that we test our algorithms on, include VL2 [20] and non-structured topology. Different problem sizes (200-1000-2000) with various numbers of sinks (15-10-15, respectively) are used when comparing our methods in those two topologies. We chose the same links capacities as in [20] (10 Gbps) for both of the cases. The capacity of the leave links (connecting ToRs to PMs) are set to 1 Gbps in VL2 test cases. Figs. 14–16 illustrate the amount of *satisfaction* that any given VM experiences using different methods when placed on physical machines interconnected in a VL2-based infrastructure. The comparison between the average *satisfaction* of the VMs is provided in Fig. 19b–d.

We also performed similar experiments by applying our algorithms into two placement problems in which physical machines are interconnected without a standard topology. Figs. 17 and 18 demonstrate the experiment results in a DCN with general topology consisting of 990 and 1985 non-sink with 10 and 15 sink physical machines, respectively. According to the corresponding average *satisfaction* bar graph

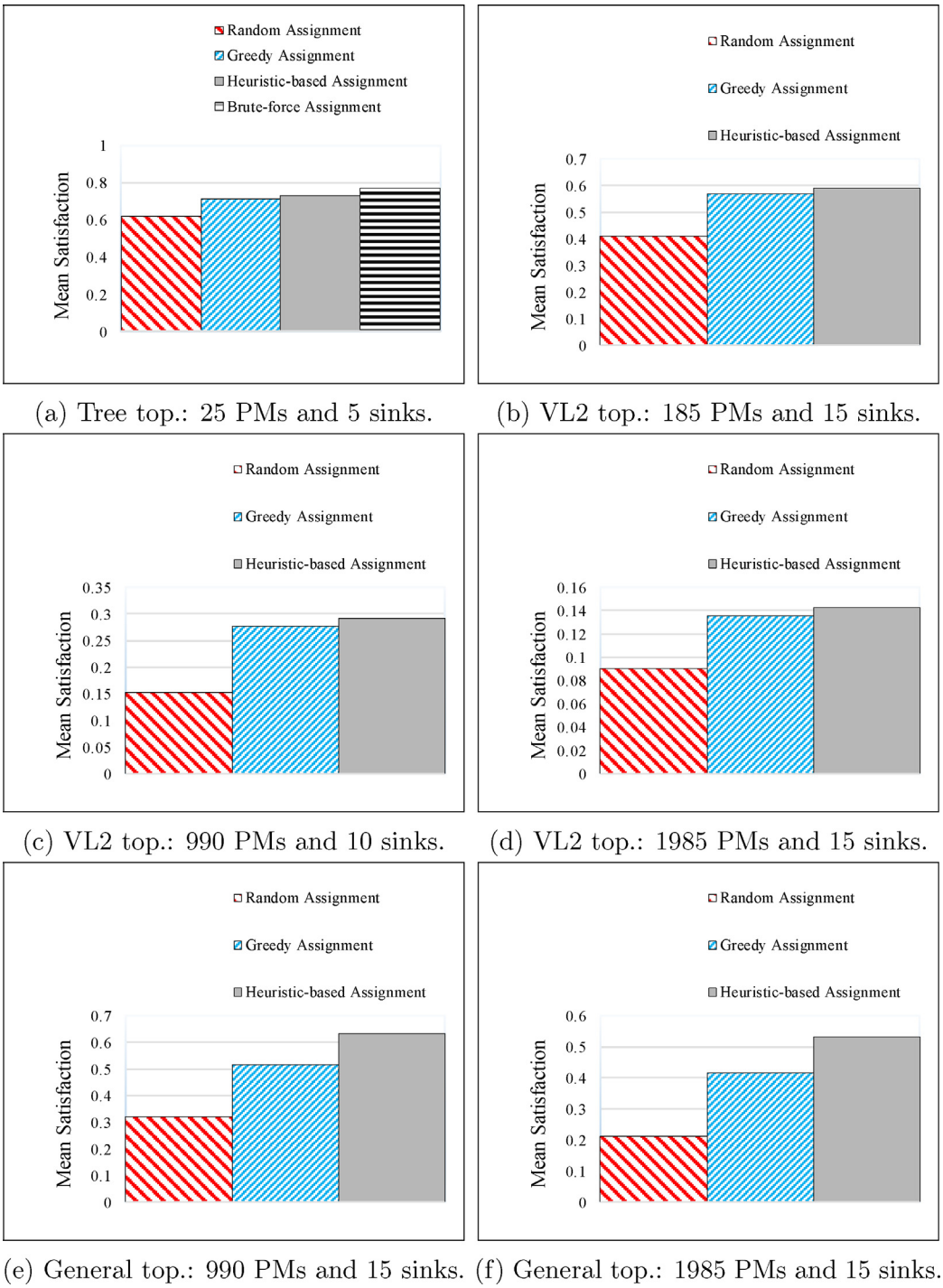


Fig. 19. A comparison between different assignment methods based on average *satisfaction* achieved.

depicted in Fig. 19e and f it can be understood that there is a more significant improvement when heuristic-based algorithm is applied to a problem whose underlying network topology is of general type (i.e., not Tree or Fat-tree based topology).

A comparison between the overall average *satisfaction* of the VMs in two different topologies with the same number of physical machines and sinks can also reveal an interesting relationship between the underlying topology and the amount of average *satisfaction* achieved. According to Fig. 19c–f, when

the machines are interconnected with a structureless topology, the VMs will be serviced with a higher *satisfaction* level. The less amount of overall *satisfaction* in DCNs with tree-based topologies can be justified by the symmetry that exists in such topologies. If a sink is located in one branch of the tree, the congestion will be inevitable after the PMs in the vicinity of the sink (in the same branch) are occupied, especially if the remaining VMs still have a high level of demand for that particular sink. Now, compare to the flexibility of VM placement in DCNs with structureless topologies where there are usually multiple links to the sinks that gives more space for optimization.

5.2. Comparison based on demand distribution

The statistical distribution of the sink demands might affect the behavior of the assignment approaches used. In this section, we compare the outcomes of the algorithms in two different situations: (1) VMs with uniformly distributed total demands between 0.2 and 1 Gbps, (2) VMs with total demands having normal distribution with $\mu = 0.6$ and $\sigma = 0.1$. To that end, we compare the random, greedy and heuristic-based algorithms when applied to two identical DCNs (the general topology DCN with 990 PMs and 10 sinks) with the difference in the distribution of total sink demands in VM requests. Fig. 20 shows the differences between the effectiveness of greedy and heuristic-based algorithms in two different situations. According to that figure, it can be concluded that when the VM demands have a more diverse distribution, it makes more sense to use the proposed algorithms. In Uniform Distribution case, the chances of having VMs from a wide spectrum are high, while in Normal Distribution case, the majority of the VMs have much closer demands. The closeness of the demands makes the sorting less effective especially in the first variant of the greedy algorithm. As a result, the mean satisfaction of our placement algorithms can be closer to that of random placement.

5.3. Comparison between variants of Greedy and Heuristic-based approaches

As discussed in Sections 4.2 and 4.3, there are two variants of each algorithm that might have considerable excellence over each other given different topologies and distributions. In this section, we compare the optimality of the assignments resulted by two different variants of each algorithm. We applied both variants of the two approaches to the general topology DCN with 1985 non-sink PMs and 15 sink PMs. Fig. 21 illustrates the effectiveness of the two methods used for the same DCN with identical VM requests. It can be concluded from the figure that the second variants of both of the algorithms outperform the first variant to some extent. However, there is a trade-off between precision and the complexity of the algorithm.

5.4. Satisfaction vs. congestion

In our simulation experiments, we also compare the overall congestion against total satisfaction. As discussed in Section 5.1, we did an experiment with a DCN whose PMs

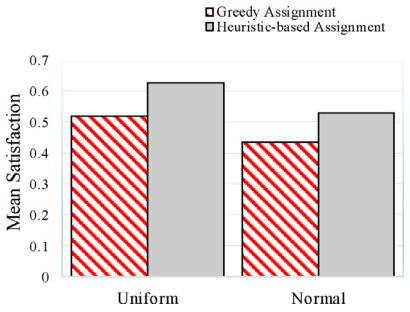


Fig. 20. Sink demand distribution effect on the effectiveness of the algorithms.

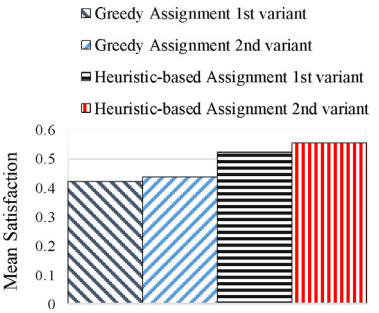


Fig. 21. A comparison between the two variants of greedy and heuristic-based algorithms when applied to the same placement problem.

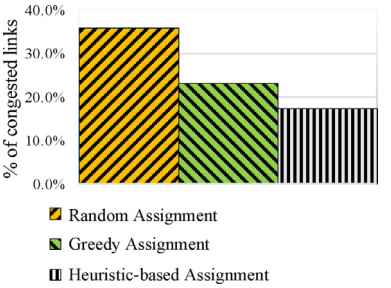


Fig. 22. The relationship between congestion and *satisfaction*.

are interconnected like vertices in a general graph (e.g. consider the graph shown in Fig. 5). For the experiment with 1985 non-sink PMs we compare the results shown in Fig. 19f with the number of links that are able to simultaneously tolerate less than 50% of the traffic demands that are supposed to be transmitted through them according to the routing algorithm used which is multi-path oblivious routing in our experiment.

As shown in Fig. 22, with random assignment deployed, almost 36% of the links connecting the DCN components to each other will experience a congestion level such that less than 50% of the flows that are supposed to pass through each of those links will be transmittable. Now, compare this percentage (36%) to 23% and 17% that are similar results when greedy and heuristic-based approaches are deployed, respectively. Based on those results, it can be concluded that when our algorithms are deployed, the links will be shared with less intense flows compared to the random assignment.

6. Conclusion and future work

In this work, we present greedy and heuristic-based algorithms for assigning a set of Virtual Machines (VMs) to a set of Physical Machines (PMs) in a specific scenario of Cloud Computing with the objective of maximizing a metric which is also defined in this work. In the scenario of interest, VMs are intensively inclined to exchanging traffic with special fixed nodes (called sinks) in the data center network while the inter-VM traffic is negligible. Therefore, the ability of the Infrastructure Provider (IP) to meet the scenario-specific demands of the VMs in the best possible way, is a decisive factor in the performance of the VMs. We introduce a metric named *satisfaction* that reflects the relational suitability of a PM for any VM assigned to it. Intuitively, this metric is also correlated to the overall congestion in the entire network. The problem of maximizing the mentioned metric by trying to find the most appropriate assignment is similar to the Quadratic Assignment problem when congestion is taken into account. Therefore, it is NP-hard and there is no polynomial time algorithm that yields an optimal solution. With that in mind, we introduce several heuristic-based offline algorithms that yield nearly optimal solutions in general case and for large problem sizes. The placement algorithms assume that the communication pattern and flow demand profiles of the VMs are given.

We compare the performance of the introduced algorithms by simulation experiments and according to the results, our algorithms are significantly more effective (the difference depends on the network topology, distribution of demands, etc.) when compared to random assignment. We also, compare the algorithms by applying them into a variety of problem sizes and observe a consistency in the good performance of their outcome. Each approach (greedy and heuristic-based) consists of two variants that are also compared to each other in our simulations. Moreover, the experiments also reveal that the overall data center network congestion and *total satisfaction* are correlated.

Future directions for our work may include studying similar scenarios with different limitations and assumptions, and applying relevant concepts for modeling the problem.

- One important limitation is often imposed not only by the network but also by the sinks (e.g., because of having limited resources) especially when they are processing units and physical machines. In future works, this sort of assumptions can be also taken into account when designing algorithms.
- The algorithms designed for the scenario that we study, assume the availability of a priori information about the demands of the all VMs which is an unrealistic assumption in some situations. Therefore, proposing a similar variant of the algorithms that are designed for on-line purposes can be another candidate for future work.
- A comparatively novel concept in Game Theory named Graphical Congestion Games [31,32] has drawn considerable attention in recent years because of its ability to model problems that consist of rivaling components that try to select the best strategy that maximizes their own interests while having a negative impact on the other components depending on their local distance. In other words, two components may be using the same resource while having no effect on each other because of the long distance between them. Although the concept has been applied to problems such as Wireless Spectrum Sharing as in [33], it also suits our problem because in our scenario a decision's impact is capable of affecting only a subset of the components (VMs) depending on their location (the PM that they are placed on).

Acknowledgments

We thank TÜBİTAK (The Scientific and Technological Research Council of Turkey) for supporting this work with project 113E274. We also thank Professor Hande Yaman for her comments on the problem defined in this paper.

References

[1] X. Meng, V. Pappas, L. Zhang, Improving the scalability of data center networks with traffic-aware virtual machine placement, in: Proceedings of the 29th Conference on Information Communications, INFOCOM '10, IEEE Press, (Piscataway, NJ, USA), 2010, pp. 1154–1162.
[2] J. Xu, J. Fortes, Multi-objective virtual machine placement in virtualized data center environments, in: Proceedings of the 2010 IEEE/ACM Conference on Green Computing and Communications, 2010, pp. 179–188.
[3] U. Bellur, C. Rao, M. Kumar, Optimal placement algorithms for virtual machines, in: Proceedings of CoRR, 2010.
[4] A. Verma, P. Ahuja, A. Neogi, PMAPPER: power and migration cost aware application placement in virtualized systems, in: Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware, ser., Middleware '08, (New York, NY, USA), SpringerVerlag New York, Inc., 2008, pp. 243–264.
[5] K. Mills, J. Filliben, C. Dabrowski, Comparing VM-placement algorithms for on-demand clouds, in: Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science, CLOUDCOM '11, (Washington, DC, USA) IEEE Computer Society, 2011, pp. 91–98.
[6] J. Kuo, H. Yang, M. Tsai, Optimal approximation algorithm of virtual machine placement for data latency minimization in cloud systems, in: Proceedings IEEE INFOCOM 2014, April 27 2014–May 2, 2014, pp. 1303–1311.
[7] H. Teyeb, A. Balma, N.B. Hadj-Alouane, S. Tata, A.B. Hadj-Alouane, Traffic-aware virtual machine placement in geographically distributed Clouds, in: Proceedings of 2014 International Conference on Control, Decision and Information Technologies (CoDIT), 3–5, November 2014, pp. 024–029.
[8] A.T. Ernst, M. Krishnamoorthy, Solution algorithms for the capacitated single allocation hub location problem, Ann. Oper. Res. 86 (1999) 141–159.
[9] S. Das, M. Kagan, D. Crupnicoff, Faster and efficient VM migrations for improving SLA and ROI in cloud infrastructures, DC CAVES, 2010.
[10] S. Lee, R. Panigrahy, V. Prabhakaran, V. Ramasubrahmanian, K. Talwar, L. Uyeda, U. Wieder, Jan. 2011, Validating Heuristics for Virtual Machine Consolidation, Microsoft Research, MSR–TR–2011–9, pp. 14.
[11] N. Bobroff, A. Kochut, K. Beaty, Dynamic placement of virtual machines for managing SLA violations, in: Proceedings of the 10th IFIP/IEEE Symposium on Integrated Network Management, 2007, pp. 119–128.
[12] IBM ILOG CPLEX Optimizer, http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/, last visited May 2014.
[13] S. Rampersaud, D. Grosu, A sharing-aware greedy Algorithm for virtual machine maximization, in: Proceedings of the 2014 IEEE 13th International Symposium on Network Computing and Applications (NCA), 21–23, August 2014, pp. 113–120.
[14] H. Teyeb, A. Balma, N.B. Hadj-Alouane, S. Tata, Optimal virtual machine placement in large-scale cloud systems, in: Proceedings of the 2014 IEEE 7th International Conference on Cloud Computing (CLOUD), June 27–July 2, 2014, pp. 424–431.
[15] Gurobi Optimization, http://www.gurobi.com, last visited May 2014.
[16] C. Tang, M. Steinder, M. Spreitzer, G. Pacifici, A scalable application placement controller for enterprise data centers, in: Proceedings of the 16th International Conference on World Wide Web, WWW '07, (New York, NY, USA), ACM, 2007, pp. 331–340.
[17] GNU Linear Programming Kit, http://www.gnu.org/s/glpk/, last visited May 2014.

[18] S. Chaisiri, B.-S. Lee, D. Niyato, Optimal virtual machine placement across multiple cloud providers, in: Proceedings of Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific, December 2009, pp. 103–110.
[19] E.M. Loiola, N.M.M. de Abreu, P.O. Boaventura-Netto, P. Hahn, T. Querido, A survey for the quadratic assignment problem, Eur. J. Oper. Res. 176 (2) (2007) 657–690.
[20] A. Greenberg, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, S. Sengupta, VL2: A scalable and flexible data center network, in: Proceedings of SIGCOMM, Association for Computing Machinery, Inc., August 2009.
[21] O. Biran, A. Corradi, M. Fanelli, L. Foschini, A. Nus, D. Raz, E. Silvera, A stable network-aware vm placement for cloud systems, in: Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012), IEEE Computer Society, 2012, pp. 498–506.
[22] X. Meng, V. Pappas, L. Zhang, Improving the scalability of data center networks with traffic-aware virtual machine placement, in: Proceedings of the INFOCOM, 2010 IEEE, 2010, IEEE, pp. 1–9.
[23] C.A.M. Fanelli, L. Foschini, VM consolidation: A real case based on OpenStack Cloud, Future Gener. Comput. Syst. 32 (2014) 118–127.
[24] K. Sunil Rao, P. Santhi Thilagam, Heuristics based server consolidation with residual resource defragmentation in cloud data centers, Future Gener. Comput. Syst. 50 (2014) 87–98.
[25] R. Cohen, L. Lewin-Eytan, J.Seffi Naor, D. Raz, Almost optimal virtual machine placement for traffic intense data centers, in: Proceedings of INFOCOM, 2013 IEEE, 2013, IEEE, pp. 355–359.
[26] K.C. Webb, A.C. Snoeren, K. Yocum, Topology switching for data center networks, in: Hot-ICE Workshop, 2011.
[27] S. Crago, K. Dunn, P. Eads, L. Hochstein, D.-I. Kang, M. Kang, D. Modium, K. Singh, J. Suh, J.P. Walters, Heterogeneous cloud computing, in: Proceedings of IEEE International Conference on Cluster Computing (CLUSTER), 2011, IEEE, pp. 378–385.
[28] Top 500 Supercomputers in the World, http://www.top500.org/, last visited April 2014.
[29] J. Munkres, Algorithms for the assignment and transportation problems, J. Soc. Indus. Appl. Math. 5 (1) (1957) 32–38.
[30] JUNG, Java Universal Network/Graph Framework, http://jung.sourceforge.net/, last visited June 2014.
[31] V. Bilò, A. Fanelli, M. Flammini, L. Moscardelli, Graphical congestion games, Algorithmica 61 (2) (2011) 274–297.
[32] R. Southwell, Y. Chen, J. Huang, Q. Zhang, Convergence dynamics of graphical congestion games, in: Game Theory for Networks, Springer, 2012, pp. 31–46.
[33] S. Ahmad, C. Tekin, M. Liu, R. Southwell, J. Huang, Spectrum sharing as spatial congestion games, arXiv preprint arXiv:1011.5384 (2010).

Amir Rahimzadeh Ilkhechi was born in Ilkhechi, East Azerbaijan, Iran in 1989. He received his B.S. degree in Information Technology from the University of Tabriz, East Azerbaijan, Iran, in 2011, and then joined as an M.S. student the Department of Computer Engineering at Bilkent University (co-supervised by Associate Professor Ibrahim Korpeoglu and Professor Özgür Ulusoy), Ankara, Turkey, in 2012. He is currently a PhD student at the Department of Computer Science at Duke University (supervised by Assistant Professor Shivnath Babu), North Carolina, United States. His research interests cover Computer Networks, Distributed Systems, Cloud Computing, Big Data, Graph Theory and Computational Geometry.

Ibrahim Korpeoglu received his Ph.D. and M.S. degrees from University of Maryland at College Park, both in Computer Science, in 2000 and 1996, respectively. He received his B.S. degree (summa cum laude) in Computer Engineering, from Bilkent University in 1994. Since 2002, he is a faculty member in the Department of Computer Engineering of Bilkent University. Before that, he worked in several research and development companies in USA including Ericsson, IBM T.J. Watson Research Center, Bell Laboratories, and Bell Communications Research (Bellcore). He received Bilkent University Distinguished Teaching Award in 2006 and IBM Faculty Award in 2009. His research interests include computer networks and systems, wireless networks, and cloud computing.

Özgür Ulusoy received his Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign. He is currently a Professor in the Computer Engineering Department of Bilkent University in Ankara, Turkey. His current research interests include web databases and web information retrieval, multimedia database systems, social networks, and cloud computing. He has published over 120 articles in archived journals and conference proceedings.