NETWORK-AWARE VIRTUAL MACHINE PLACEMENT IN CLOUD DATA CENTERS WITH MULTIPLE TRAFFIC-INTENSIVE COMPONENTS

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Amir Rahimzadeh Ilkhechi

July, 2014

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. İbrahim Körpeoğlu (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Özgür Ulusoy (Co–advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Uğur Güdükbay

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Ahmet Coşar

Approved for the Graduate School of Engineering and Science:

Prof. Dr. Levent Onural
Director of the Graduate School

ABSTRACT

NETWORK-AWARE VIRTUAL MACHINE PLACEMENT IN CLOUD DATA CENTERS WITH MULTIPLE TRAFFIC-INTENSIVE COMPONENTS

Amir Rahimzadeh Ilkhechi
M.S. in Computer Engineering
Supervisors: Assoc. Prof. Dr. İbrahim Körpeoğlu and Prof. Dr. Özgür Ulusoy
July, 2014

Following a shift from computing as a purchasable product to computing as a deliverable service to the consumers over the Internet, Cloud Computing emerged as a novel paradigm with an unprecedented success in turning utility computing into a reality. Like any emerging technology, with its advent, Cloud Computing also brought new challenges to be addressed. This work studies network and traffic aware virtual machine (VM) placement in Cloud Computing infrastructures from a provider perspective, where certain infrastructure components have a predisposition to be the sinks or sources of a large number of intensive-traffic flows initiated or targeted by VMs. In the scenarios of interest, the performance of VMs are 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 physical machine (PM). The problem is NP-hard and there is no polynomial time algorithm that yields an optimal solution. Therefore we introduce several off-line heuristics-based algorithms that yield nearly optimal solutions given the communication pattern and flow demand profiles of VMs. We evaluate and compare the performance of our proposed algorithms via extensive simulation experiments.

Keywords: Cloud Computing, Virtual Machine Placement, Sink Node, Predictable Flow, Network Congestion.

ÖZET

YOĞUN TRAFİĞE SAHİP ÇOK SAYIDA BİLEŞENDEN OLUŞAN BULUT VERİ MERKEZLERİ İÇIN MEVCUT AĞ KOŞULLARINI GÖZ ÖNÜNE ALAN SANAL MAKİNE YERLEŞTİRME

Amir Rahimzadeh Ilkhechi
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticileri: Doç. Dr. İbrahim Körpeoğlu ve Prof. Dr. Özgür Ulusoy
Temmuz, 2014

Hesaplamanın satın alınabilir bir üründen İnternet üzerinde kullanıcılara sunulabilir bir hizmete dönüşmesinin ardından, Bulut Bilişim yeni bir model olarak hizmetli hesaplamayı gerçekleştirmede eşsiz bir başarısıyla ortaya çıkmıştır. Gelişmekte olan herhangi bir teknoloji gibi, Bulut Bilişim de gelişimiyle beraber çözülmesi gereken yeni zorlukları ortaya çıkarmıştır. Bu çalışmada mevcut ağ koşullarını göz önüne alan Sanal Makine (SM) yerleştirme problemini sağlayıcı açısından özel bir senaryoda inceliyoruz. Adı geçen senaryoda, belli altyapı parçaları SM'lerden kaynaklanan yoğun trafik akımlarının son hedefi olmaktadır. Odaklandığımız senaryoda, SM'lerin verimliliği yüksek oranda mevcut altyapı tarafından yoğun trafik isteklerinin karşılanmasına bağlıdır. İlk olarak, SM'lerin verimliliğini yansıtan memnuniyet (satisfaction) olarak adlandırdığımız bir metriği tanımlayıp, bu metriği en yükseğe çıkarmaya çalışıyoruz. Tanımlanan problem NP-hard olup, probleme en uygun çözümü sağlayan polinom zamanda çalışan bir algoritma mevcut değildir. Bu nedenle, SM'lerin iletişim örneği ve akım istek profillerine dayanarak tahmini optimum çözüm sağlayan bir kaç çevrim dışı sezgisel algoritma öneriyoruz. Son bölümde, simülasyon deneyleri ile, önerilen algoritmaların etkisini değerlendirip performanslarını karşılaştırıyoruz.

Anahtar sözcükler: Bulut Bilişim, Sanal Makine Yerleştirme, Gider Düğüm, Tahmin Edilebilir Akım, Ağ Tıkanıklığı.

Acknowledgement

It would not have been possible to write this thesis without the help and support of the kind people around me, to only some of whom it is possible to give particular mention here:

First of all, I would like to gratefully and sincerely thank my first advisor Dr. İbrahim Körpeoğlu for his guidance, understanding, patience, and most importantly, his friendship during my graduate studies at Bilkent University.

I would also like to thank my second advisor for his irreplaceable assistance throughout my studies. The heartwarming advice, support and friendship of Dr. Özgür Ulusoy, has been invaluable on both academic and personal level, for which I am extremely grateful.

I cannot express enough thanks to other jury members for their invaluable contribution in such a vital transitional stage of my academic life: Prof. Dr. Uğur Güdükbay and Prof. Dr. Ahmet Coşar.

I also thank TÜBİTAK (The Scientific and Technological Research Council of Turkey) for supporting this work with project 113E274.

Finally, I would like to acknowledge the financial, academic and technical support of the Department of Computer Engineering at Bilkent University. I would like to express my special appreciations and thanks to the faculty and staff of the department, especially Dr. Öznur Taştan for her kindness, friendship and support, and the respected department chair Prof. Dr. Altay Güvenir and administrative assistant Elif Aslan for their kind helps.

Contents

1 Introduction **1**

2 Background **9**

2.1 Cloud Computing 9

2.1.1 Hardware Virtualization 10

2.1.2 The XaaS Service Models 11

2.1.3 Cloud Computing Scenarios 13

2.2 Service Assignment Problems 16

2.2.1 Parameters and Criteria 16

2.2.2 Challenges 18

2.3 The Scope of This Thesis 19

2.4 Related Work 20

3 Formal Problem Definition **23**

3.1 Scenario 23

3.2 Assumptions 25

3.3 Satisfaction Metric 28

3.4 Mathematical Description 30

3.4.1 First Case: No congestion 38

3.4.2 Second Case: Presence of Congestion 39

3.5 Chapter Summary 40

4 Proposed Algorithms 41

4.1 Polynomial Approximation for NP-hard problem 41

4.2 Greedy Approach 44

4.2.1 Intuitions Behind the Approach 45

4.2.2 The Algorithm 46

4.3 Heuristic-based Approach 48

4.3.1 Intuitions Behind the Approach 49

4.3.2 The Algorithm 50

4.4 Worst Case Complexity 50

4.5 Chapter Summary 52

5 Simulation Experiments and Evaluation 54

5.1 Comparison Based on Problem Sizes and Topologies 55

5.2 Comparison Based on Demand Distribution 58

5.3 Comparison Between Variants of Greedy and Heuristic-based approaches 59

5.4 Satisfaction vs. Congestion 60

5.5 Chapter Summary 61

6 Conclusion and Future Work 64

Bibliography 66

List of Figures

1.1 Interconnected physical machines and sink nodes in an unstructured network topology. 4

1.2 The dependence of any VM on any sink given as demand vector. . 5

1.3 The costs between any PM-Sink pair. 6

1.4 The Placement Problem. 7

2.1 Temporal Partitioning. 11

2.2 Virtualized Execution. 12

2.3 Virtual Machine. 13

2.4 Private Cloud Scheme. 14

2.5 Cloud Bursting Scheme. 15

2.6 Federated Cloud Scheme. 16

2.7 Multi-Cloud Scheme. 17

3.1 Non-resource Sinks. 24

3.2 Incoming VM Request. 26

3.3 Off-line Virtual Machine Placement. 27

3.4 A simple example of placement decision. 29

3.5 A graph representation of a simple data center network without an standard topology. 31

3.6 A bipartite graph version of Figure 3.5 32

3.7 The abstract working mechanism of G function. 34

3.8 Single path oblivious routing 35

3.9 Multi-path oblivious routing 37

4.1 An example of sequential decisions. 43

4.2 Impact of assignment on the overall congestion. 45

4.3 Impact of an assignment decision on unassigned VMs. 49

5.1 A comparison of assignment methods in a data center with tree topology 56

5.2 A comparison of assignment methods in a data center with VL2 topology 57

5.5 A comparison of assignment methods in a data center with general topology 59

5.7 A comparison between different assignment methods based on average *satisfaction* achieved. 62

5.8 Sink demand distribution effect on the effectiveness of the algorithms. 63

5.9 Comparison of algorithms variants 63

5.10 The relationship between congestion and *satisfaction*. 63

List of Tables

3.1 Sink demands of three VMs. 25

Chapter 1

Introduction

The problem of placing a set of Virtual Machines (VMs) in a set of Physical Machines (PMs) in distributed environments has been an important topic of interest for researchers in the realm of cloud computing. The proposed approaches often focus on various problem domains: primary placement, throughput maximization, consolidation, Service Level Agreement (SLA) satisfaction versus provider operating costs minimization, etc. [1]

Mathematical models are often used to formally define the problems of virtual machine placement. The problems are then 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 [2], Gurobi [3] and GLPK [10] that are predominantly utilized in to solve placement problems of small size.

There is also another way of classifying the works related to VM placement according to the number of cloud environments in two different environments:

1. Single-cloud environments.
2. Multi-cloud environments.

The firstly mentioned 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 service is assigned to and executed in, taking into account the resource constraints.

Normally, a reverse relation exists between the computational cost and the precision of the solutions. Therefore, finding a tradeoff point where both quality and complexity of the solution is acceptable, is a challenge. Several approximation approaches have been introduced for that purpose including the algorithm proposed by Tang et al. [4] that can come up with an efficient solution for immense placement problems with thousands of machines and services. The goal of the aforementioned algorithm is to maximize the total satisfied application demand and to minimize the number of application starts and stops as well as balancing the load among machines.

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 having access to information about the number of physical machines, the load distribution, and other such critical factors inside the IP (Infrastructure Provider) side, most works on VM placement across multi-cloud environments are related to cost minimization problems.

As an example of research in that area, Chaisiri et al. [11] 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. The algorithm assumes that the future demand and price are uncertain and is based on Stochastic Integer Programming (SIP).

By examining multi-cloud scenario, Vozmediano et al. [5, 6] propose utilizing a computing cluster on top of a multi-cloud infrastructure to solve Many-Task Computing (MTC) applications problems that are loosely coupled.

One advantage of this method is improving the effectiveness of deployment when cluster nodes can be provisioned with resources from different clouds. Another advantage can be enabling implementation of high-availability strategies in such scenarios.

In a different but related work, Hermenier et al. [7] propose the Entropy Resource Manager designed for homogeneous clusters. By taking the migration overhead into account, the proposed resource manager consolidates the VMs dynamically based on constraint programming. In this approach, migrations that cause lower performance overhead are chosen by the entropy. In order to solve the problem, CHOCO constraint programming solver is utilized. Also, a considerable amount of effort has been devoted to the energy management aspects of VM placement. Le et al. [8] investigate the effect of VM placement mechanisms on cooling and maximum data center temperatures. The objective is to reduce the electricity cost in geographically distributed data centers that are designed for performing high performance computing. They develop a model of data center cooling for a realistic scenario and design VM distribution and migration policies across data centers to benefit from time-based differences in the temperature and electricity costs.

To begin with, our work falls into the first category that pertains to single cloud environments. Based on this assumption, we can take the access to detailed information about the 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 thesis 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 Chapter 3 while also a brief explanation is provided below. The placement algorithms are off-line 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 with. We call these special nodes “sinks”, and call the remaining nodes “Physical Machines (PMs)”. Although we call the special nodes as sinks, we assume the communication between VMs and sinks is bidirectional.

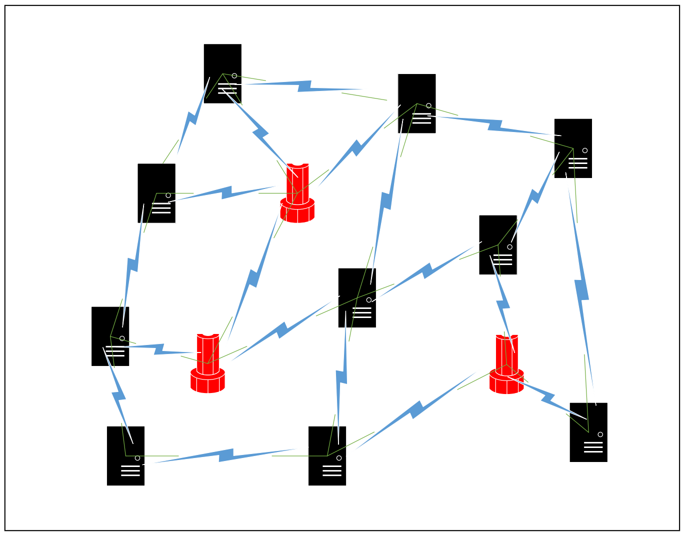


Figure 1.1: Interconnected physical machines and sink nodes in an unstructured network topology.

As illustrated in Figure 1.1, 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.

The network connecting the nodes can be represented as a general graph $G(V, E)$ where E is the set of links and V is the set of nodes (including PMs and sinks as shown in relation 1.1). On the other hand, the number of normal PMs is much larger than the number of sinks (relation 1.2):

$$S \in V \tag{1.1}$$

$$|S| \ll |V - S| \tag{1.2}$$

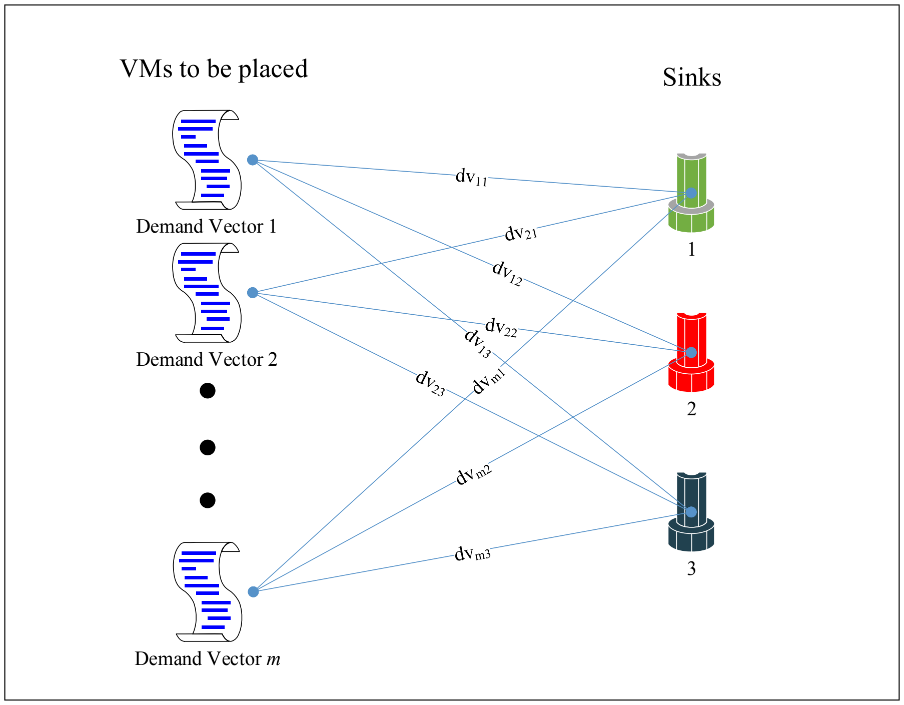


Figure 1.2: The dependence of any VM on any sink given as demand vector.

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, it makes sense to assume that 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 prior information is given about any VM:

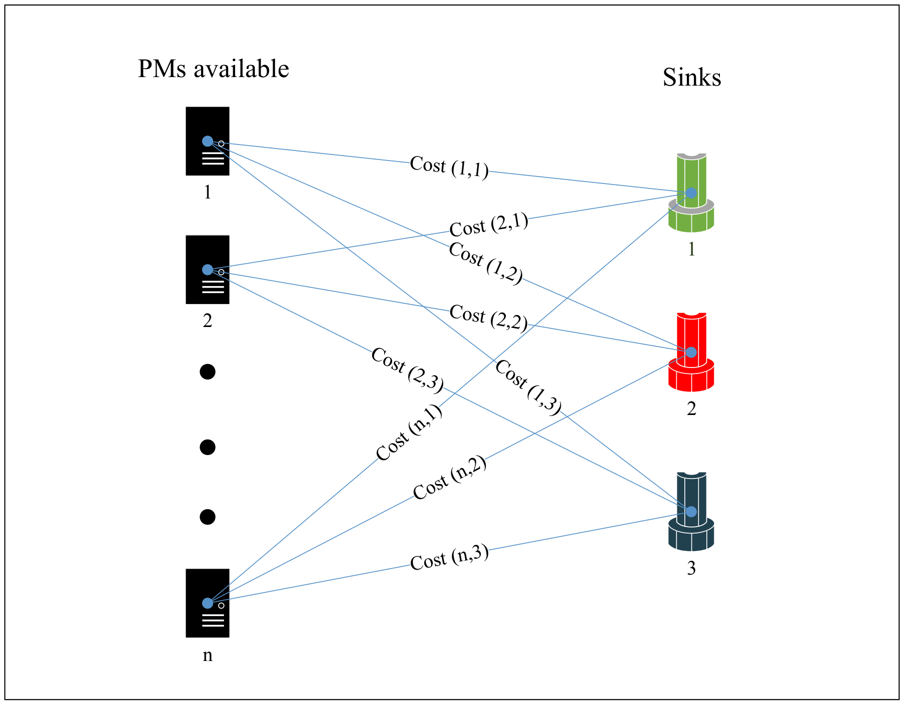


Figure 1.3: The costs between any PM-Sink pair.

- **Total Flow:** The total flow that the VM will demand in order to send to and/or receive 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). Figure 1.2 illustrates what demand vector means.

Suppose that each PM-Sink pair is associated with a numerical cost as depicted in Figure 1.3 (to be explained in detail later). 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 those 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

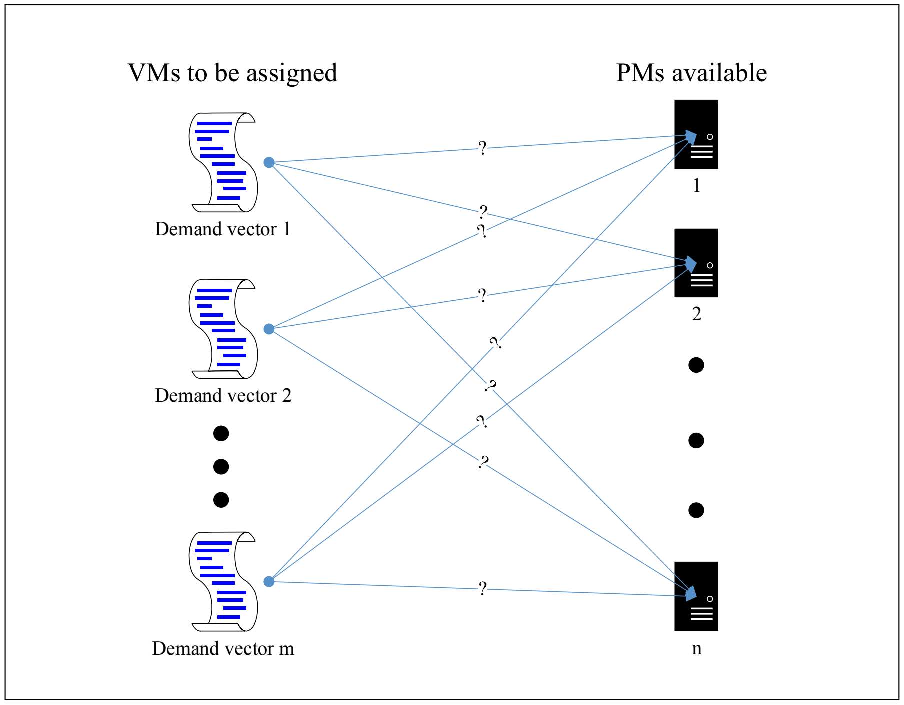


Figure 1.4: The Placement Problem that can be represented as an Assignment Problem that maps VMs to PMs.

the service provider and the service consumer side will be in a win-win situation. From consumer's point of view, the VMs will experience a better quality of service. Similarly, on the provider side, the links will be less likely to be saturated which enables serving more VMs.

The placement problem (Figure 1.4) in our scenario is the complement of the famous Quadratic Assignment Problem (QAP) [36] 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 behavior 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 thesis includes a brief background about cloud computing together with literature review (Chapter 2) followed by the formal definition of the problem in hand (Chapter 3). In Chapter 4, some algorithms for solving the problem are provided. Experimental results and evaluations are included in Chapter 5. Finally Chapter 6 concludes the thesis and proposes some potential future work.

Chapter 2

Background

2.1 Cloud Computing

Nowadays, Cloud Computing is becoming a famous buzzword. As a brand new infrastructure to offer services, Cloud Computing systems have many superiorities in comparing to those existed traditional service provisions, such as reduced upfront investment, expected performance, high availability, infinite scalability, tremendous fault-tolerance capability and so on, and consequently chased by most of the IT companies, such as Google, Amazon, Microsoft, Salesforce.com [14]. Cloud Computing provides a paradigm shift following the shift from mainframe to client–server architecture in the early 1980s [9, 12] and rather than a product, in this novel paradigm computing is delivered as a service. In such a paradigm, resources, software, or information are services that are provided to customers over networks.

Cloud Computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the data centers that provide those services [16].

Nevertheless, there are quite a few different definitions for Cloud Computing and there is no consensus on the whatabouts of Clouds. Besides, Cloud

Computing is not a completely novel concept as it is conceptually connected to some relatively new paradigms such as Grid Computing, utility computing, cluster computing, and distributed systems in general. [24].

2.1.1 Hardware Virtualization

Literally, hardware virtualization means that an application executes on virtualized hardware as opposed to physical hardware [15]. Virtualization is a technology that draws a separation line between computation and physical hardware. This technology is often referred to as the groundwork for Cloud Computing for its ability to isolate software from hardware, and in turn isolating users and processes/resources. Traditional operating systems are not capable of providing such degree of isolation that suits well for Cloud Computing. Hardware virtualization approaches include Full Virtualization, Partial virtualization and Paravirtualization [17]. With virtualization, software capable of execution on the raw hardware can be run in a virtual machine. Cloud systems deployable services can be encapsulated in virtual appliances (VAs) [18], and deployed by instantiating virtual machines with their virtual appliances [19].

In [15] Plessl and Platzner compare the three different approaches of hardware virtualization arguing the motives behind each approach:

- **Temporal Partitioning:** Partially deployed but insufficient resources may be sequentially utilized to run a specific splittable application. Temporal Partitioning maps an application of arbitrary size to a device with insufficient resources having the reconfigurability feature (Figure 2.1).
- **Virtualized Execution:** Virtualized execution is meant to achieve a certain level of device-independence within a device family. An application is mapped to or specified directly in a programming model (Figure 2.2).
- **Virtual Machine:** The motivation for this virtualization approach is to achieve an even higher level of device-independence. Instead of mapping an application directly to a specific architecture, the application is mapped

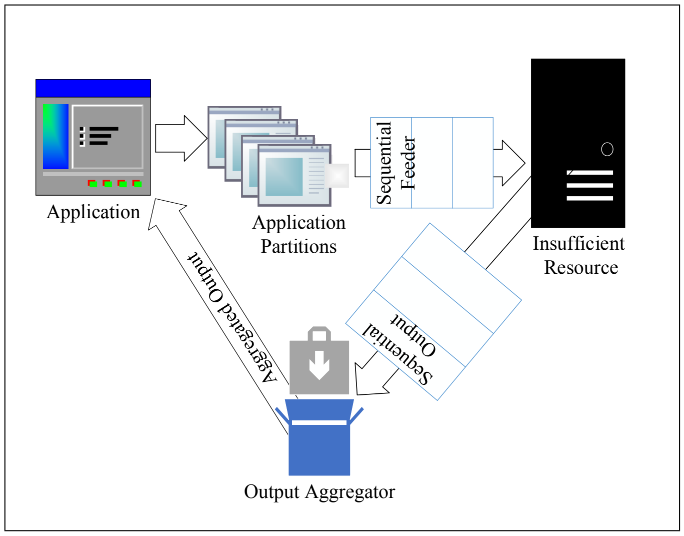


Figure 2.1: Temporal Partitioning.

to an abstract computing architecture. Java Virtual Machine [25, 26] is probably the most renown example of this virtualization category (Figure 2.3).

2.1.2 The XaaS Service Models

There are several service models that are related to cloud computing out of which some are very important and they are worthy to be mentioned here. Furthermore, these services fill into three levels: hardware level, system level and application level [14].

- **Software as a Service (SaaS):** In this model, software applications are the actual services that are executed on infrastructures that are managed by a vendor. Customers often use certain family of clients such as web browsers and programming interfaces to access the provided services. Normally, users are charged on a subscription basis [20]. Customers have often no idea of the working mechanisms in the underlying infrastructure where their applications are serviced. In other words, there is a level of transparency between customers and the infrastructure.

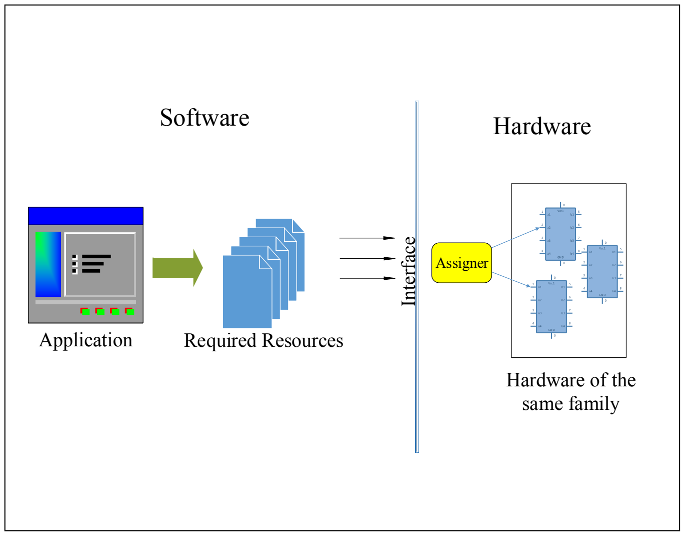


Figure 2.2: Virtualized Execution.

- **Platform as a Service (PaaS):** In the PaaS model, computing platform or solution stack normally consisting of operating system, programming language IDE, database, and web server are delivered as services to consumers [21]. Without a necessity to oversee and control the bottom layer of software and hardware (e.g. Network, OS, physical data storage) users or customers can simply develop and run their software and also manage their applications (e.g. configuration settings for the hosting environment) if required [22].

- **Network as a Service (NaaS):** Examples of the services that fall into NaaS category abound. Amongst, there are some popular ones such as Internet services from carriers (both wired and wireless network service), mobile services and alike. The Internet services are mainly concerned with providing broadband bandwidth-on-demand services [14].

- **Data as a Service (DaaS):** The DaaS refers to the family of services provided by means of software as a service or web service that offer access and analytics to a set of proprietary set of aggregated data. Several renown companies such as DataDirect [27] and Strikeiron [28] are providing that service to customers.

- **Infrastructure as a Service (IaaS):** In Clould Computing jargon, a set of

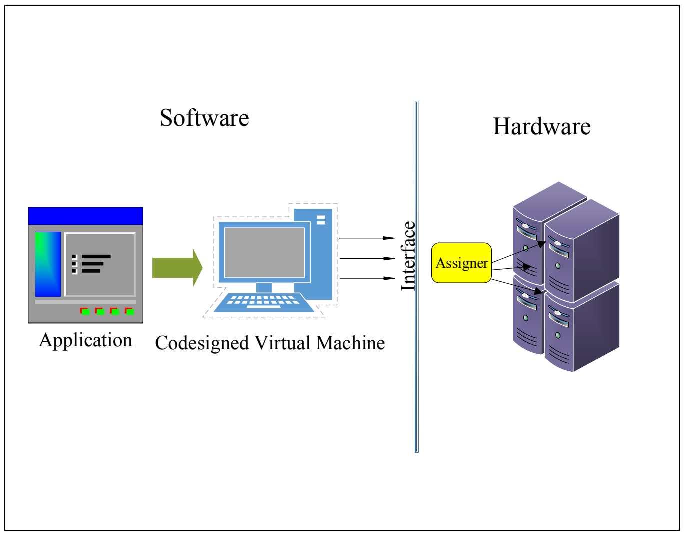


Figure 2.3: Virtual Machine.

hardware resources including components like storage capacity, memory, CPU cycles, and Network facilities are called infrastructure. IaaS is a service model in which infrastructure is delivered as service, typically over the Internet. Some popular companies with huge investments on hardware that offer such services include Amazon [29], ServePath [30], Mosso [31], and Skytap [32]. Those services often charge in terms of customer usage [14]. In this model, the customers have more flexibility as they are able to run their desired software that can be ordinary applications or even operating systems. Nevertheless, the underlying cloud infrastructure is not under direct control of the users. Customers can only manipulate and control their own virtual infrastructure that is formed by virtual machines hosted by physical IaaS vendors. In this thesis, although we are not confined to a particular service model, we will mostly focus on infrastructure as a service model.

2.1.3 Cloud Computing Scenarios

Regardless of the service model that is used, one can think of two major stakeholders: 1- Infrastructure Provider (IP), and 2- Service Provider (SP). The first refers to the party that offers infrastructure and resources such as data centers, storage capacities, networks, etc, while the latter refers to the party that delivers the services provided by the IP to the end users. The provided services can be of any family (e.g. SaaS, NaaS, ...) that in most cases are deployed using PaaS tools. According to [23], cloud scenarios can be roughly classified into four categories:

- **Private Cloud:** In such a scenario, the SP and IP are integrated as a single provider and the corresponding organization provisions services depending on its own internal infrastructure. One obvious advantage of this scheme is the lack of transparency between the IP and SP enabling a better control and higher performance as the whole cloud is administered within a united capsule. Higher security can be counted as another advantage of the mentioned scenario. Figure 2.4 provides a graphical illustration of private cloud scheme.

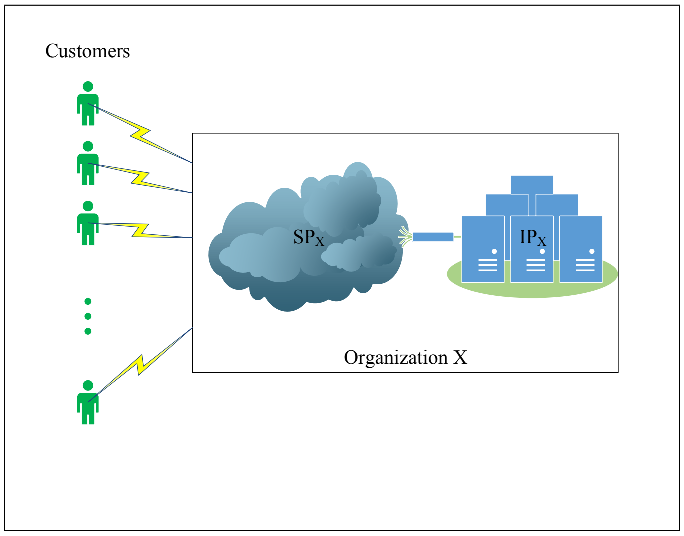


Figure 2.4: Private Cloud Scheme.

- **Cloud Bursting:** Figure 2.5 depicts a different scenario where one or more IPs are deployed as backup Infrastructure Providers along with a local one. In some unavoidable situations, the local IP may not be capable of handling the demands due to a burst workload and/or it might be facing periodical failures. The usage of multiple back up IPs guarantees that the scheduled jobs can be offloaded to a different IP in any unpredicted or anomalous circumstance.

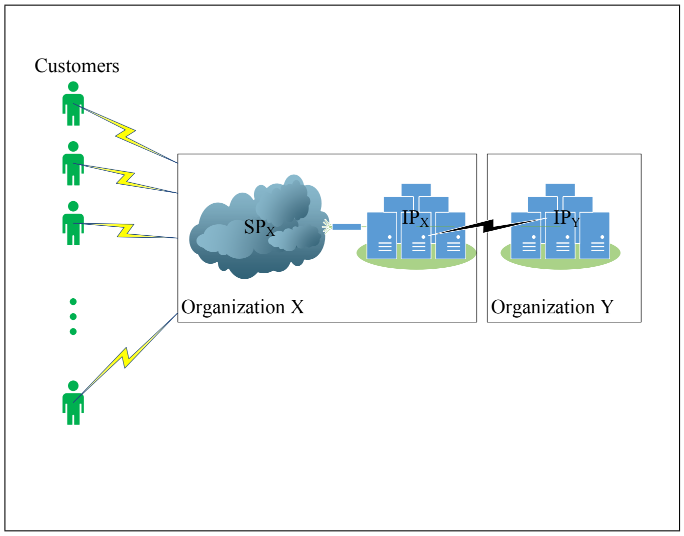


Figure 2.5: Cloud Bursting Scheme.

- **Federated Cloud:** According to [33], a federated cloud (Figure 2.6) is a collection of a Service Provider and multiple collaborative Infrastructure Providers that try to balance the load between themselves. The federation is often transparently associated with the IP level and SP has no control over it. Put differently, an SP that assigns a job to an IP in a federation, is not aware of the fact that the job might be/have been offloaded to a different IP for load balancing purposes. Nonetheless, the SP is sometimes able to enforce location constraints on the service (e.g. by requiring it to be provisioned in particular IP(s)).

- **Multi-Cloud:** A federated cloud without transparency between the SP and IPs is equivalent to Multi-Cloud scenario. As shown in Figure 2.7, in such scenarios, the SP is responsible for coordinating the IPs and deal with load balancing and optimality challenges. We can think of a similarly defined special case scenario where the organization has no internal IP and depends totally on the external ones. In this case as the SP depends totally on external infrastructures, cost-related optimization problems often draw attention.

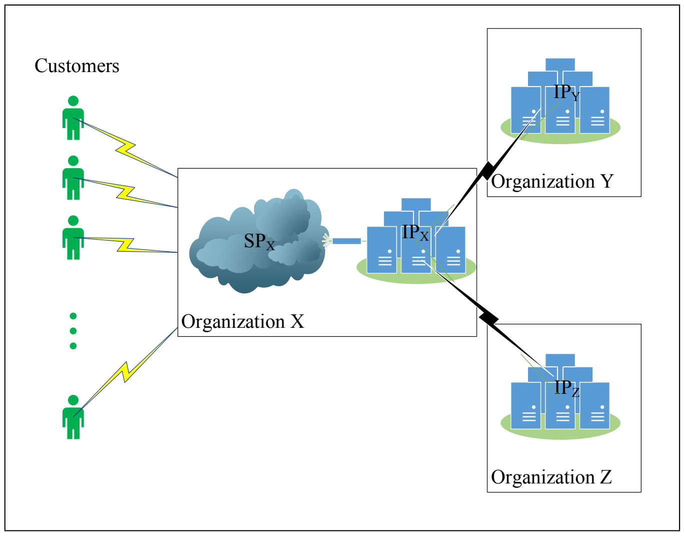


Figure 2.6: Federated Cloud Scheme.

2.2 Service Assignment Problems

Simply put, the Cloud assignment problems are related to finding an optimal solution for determining how to map a set of requested services to a set of local or in some cases remote resources. The requested services may consist of several components whose assignment should be considered separately. Moreover, the scenario and service model is also important in deciding where to assign a given demand. The characteristics of the demands also are dependent upon the service model and scenario. In the following subsections, a brief introduction to some factors and criteria as well as challenges when dealing assignment problems are given.

2.2.1 Parameters and Criteria

Numerous factors and parameters including (but not limited to) Data Center capacities, network constraints, pricing, locality, etc. should be taken into account when designing an effective assignment algorithm. Otherwise, the output can be unsatisfactory or it can yield contradictory outcomes for different stakeholders. The assignment algorithm should be intricately designed to honor the advantages

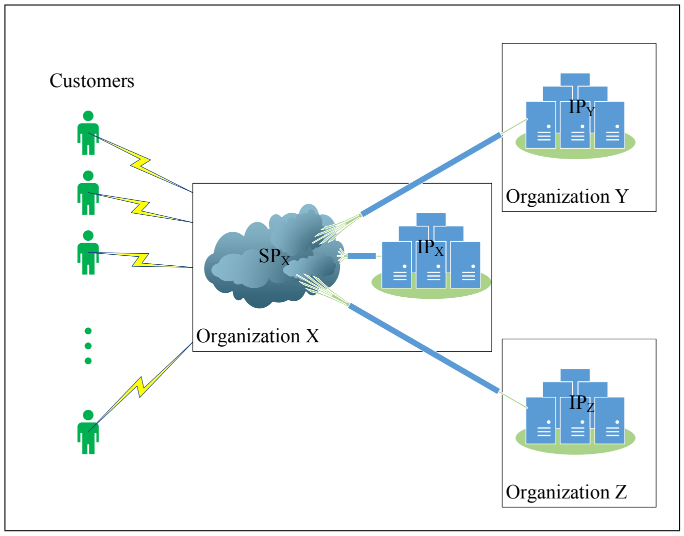


Figure 2.7: Multi-Cloud Scheme.

of the stakeholder (IP or SP) that it is meant to be associated with. That being the case, typically there is a price-performance trade-off intrinsic to assignment problems in Cloud systems that should be alleviated. Below, some of the most important factors that are mostly used as criteria for evaluating an assignment algorithm are given.

- **Performance:** Virtualization and consolidation are the most popular techniques to enhance the utilization of physical resources such as data centers and network bandwidth enabling the server platforms to run possibly heterogeneous applications. However, the significance of placement algorithms are coequally essential meaning that deploying different VM placement strategy can affect the performance substantially in an unvarying scenario [34].

- **Cost:** Although fixed schemes used to be dominant for modelling Cloud prices in the infancy days of Cloud technology, it has gradually shifted to the dynamic pricing schemes [35]. Without any undesired impact on the performance of a given service, it is possible to diminish the investment amounts by reconfiguring the services dynamically [37]. One example of this reconfiguration is resizing VMs without any disadvantageous consequences. Besides, the reciprocal competitive effect of VMs (e.g. intervention and rivalry of different VMs trying to access the same resource such as CPU cycles) should also be taken into consideration when modelling the price for a Cloud service.

- **Reliability and Availability of the Services:** In some scenarios, objective VM placement objective is guaranteeing a reliable service with highest possible availability being cognizant of resource constraints. Several techniques can be applied to achieve such goals including replication and migration of VMs across diverse physical machines possibly located in remote geographical zones. A higher level of reliability and availability depends on the reliability of hosting data centers, the significance of service and any associated data that is enclosed in VMs, the access frequency are the most important factors that one should bear in mind when designing a VM placement algorithm for maximizing the reliability and availability of the service [45].
- **Locality:** Locality is a beneficial quality unless some other more vital factors in stake such as security are conflictingly affected [43]. The closeness of a VM (or service) to its corresponding user is referred to as locality [42]. Maximizing this factor is specifically related to metrics such as congestion in the deployed networks. Our thesis is closely tied to VM placement in a scenario that locality should be highlighted in order to achieve a better performance in the underlying physical network.

2.2.2 Challenges

Developing a general-purpose placement mechanism is infeasible due to the magnitude of different scenarios, the range of parameters, conflict of different stakeholders' interests, etc. Below, some of the challenges that make the placement problem even more demanding are listed.

- To begin with, in some situations it is very difficult to model the requirements of the users when there is no defined prototype model that reflects the exact needs and all the thresholds for the users. In other words, the ever changing service models and scenarios may be evolving too fast to catch up with and adapt the placement method accordingly.

- Moreover, it is often a very taxing job to find proper parameters when trying to do model parameterization especially when the size of the problem is too large.

- Last but not least, the VM placement problems can often be reduced to well-known NP-hard problems such as multiple-knapsack [4]. Therefore, finding a perfect solution that can be run in a reasonable amount of time specially in online problems with huge size is almost impossible (e.g., Amazon EC2 [29], the leading cloud provider, has approximately 40,000 servers and schedules 80,000 VMs every day [44]). The solution is to use approximation algorithms and heuristic-based approaches to be able to achieve an acceptable result in a practicable time.

2.3 The Scope of This Thesis

Considering the complexity and variety of Virtual Machine placement problems, the subject and/or scenario orientedness of any proposed placement algorithm is something inevitable. That being said, the placement problem studied in this thesis is also limited to a specific scenario and assumes the availability of some apriori knowledge such as VM bandwidth demand profile information. In this work, we are concerned with optimal usage of network resources rather than any other Cloud assets including physical machines, or even some microscopic level resources like CPU cycles, memories, storage capacities and alike. The constraints that we consider in developing our VM placement algorithm are also related to the underlying network infrastructure that is deployed as a part of Cloud. We propose off-line VM placement approaches having a reasonable time complexity and yielding near optimal results.

2.4 Related Work

There are several studies in the literature that are closely related to our work. In [46] consolidation has been viewed from a different angle: consolidation is primarily meant to require VMs be packed tightly while they also receive resources commensurate with their demands. However, network bandwidth demands of VMs may be too dynamic that will make it difficult to distinguish demands by a fixed number and try to apply conventional consolidation schemes. In their work, they capture the bandwidth demand by random variables obeying probabilistic distributions. The mentioned study is focused on consolidating VMs according to bandwidth constraints enforced by network devices including Ethernet adapters and edge switches. They formulate the problem as a Stochastic Bin Packing problem and then propose an online packing algorithm.

In a different work [47] carried out by O. Biran et al. focused on Virtual Machine placement problem, researchers contend that 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 the least or comparatively fewer costs. In their work, they focus on both network and CPU-memory requirements of the VMs. Their proposed methods that not only satisfies the anticipated communication demands of the VMs, but is also resilient to variations happening over time.

The scalability of data centers have been carefully studied by X. Meng et al. in their work [48]. 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.

Another work that also focuses on consolidation of virtual machines is a research [49] followed through with D. Breitgand and A. Epstein. They suggest that consolidating VMs should be realized without quality of service degradation. The study is related to the problem of consolidating VMs on the minimum number of containers interconnected using a network where there is a bottleneck possibility.

To the best of our knowledge the most relevant past work is [50] 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 communication 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.

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 [38], M. Al-Fares et al. argue that the aggregate bandwidth requirements of an immense number of computers that are contained in a data center network can be huge. The typical infrastructure of the utilized networks consists of tree of routing and switching elements with progressively more specialized and expensive components moving upwards in the network hierarchy. Their work contends that this architecture after even deploying the higher-end IP switches or routers may only support half of the aggregate bandwidth available at the edge of the network. They try to interconnect the commodity switches in different ways to deliver more performance at less cost than available from popular higher-end solutions. Their approach does not require any alterations on the end host network interface, OS, and applications.

A network architecture is proposed by [39] by A. Greenberg et al. to allow dynamic resource allocation across large server pools. They declare that the data center network should allow any server to be assigned to any service, and VL2 meets that requirement. They have also made a real working prototype of VL2.

Similarly, in [40], C. Guo et al. propose a network architecture named BCube that is specifically aimed for shipping-container based and modular data centers. In this architecture, servers with multiple network ports connect to multiple layers of COTS (commodity the-shelf) mini-switches, relaying packets to other servers which is against what happens in other architectures where servers are only end hosts. According to their experiments, among the most important features of BCube is accelerating bandwidth intensive applications.

Finally, in [41] a study of application demands from a production data center of 1500 servers by S. Kandula et al. reveals that in many cases application demands can be generally met by a network that is slightly oversubscribed. In their work, they advocate a hybrid architecture claiming that eliminating over-subscription is a needless overkill. In their approach, after the base network is provisioned for the average case, for the hotspots they add extra links on an on-demand basis. The additional links are called flyways that provide reinforcement capacity in required situations.

Chapter 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 are provided, respectively.

3.1 Scenario

Heterogeneity of interconnected physical resources in terms of computational power and/or functionality is not too unlikely in Cloud Computing environments [52]. 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 reasonable 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 will say that the nodes belonging to that set have a higher importance (e.g., the cylinder-shaped servers shown in Figure 1.1).

Throughout the thesis, 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 in Figure 1.1) that other components are heavily dependent on. A powerful supercomputer capable of executing quadrillions of calculations per second [53] 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.

In our scenario, a sink is not necessarily a processing unit or physical resource. 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 (Figure 3.1). 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.

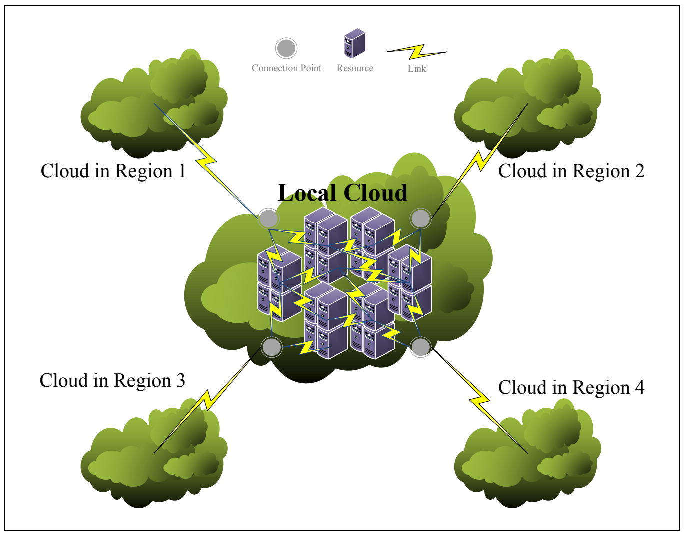


Figure 3.1: Non-resource Sinks.

Regardless of the types of the sinks (resource or non-resource) the overall traffic request destined for them is assumed to be very intense. Having said that, 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 commensurate to their per sink demands. For example, in Table 3.1 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).

Table 3.1: 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

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 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 3.1 must hold where i, j, k, and l are possible values (i.e., $i \leq$ number of VMs, and $j \leq$ number of sinks):

$$D'(k,l) \ll D(i,j), \quad \forall i,j,k,l \tag{3.1}$$

- **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 (Figure 3.2).

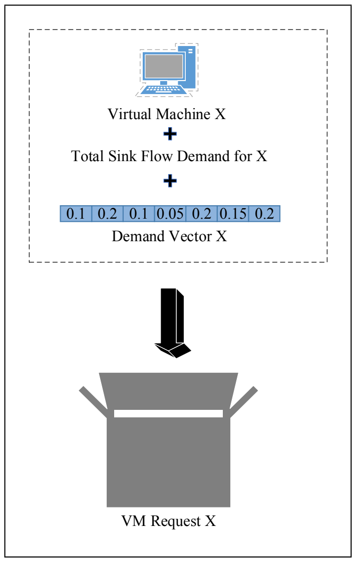


Figure 3.2: Incoming VM Request.

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 Figure 3.3.

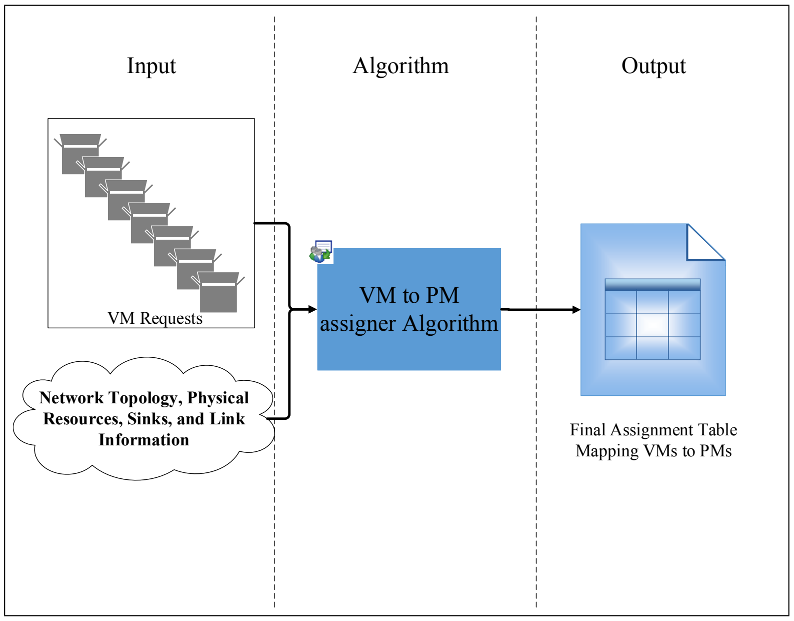

Input
Algorithm
Output
VM Requests
Network Topology, Physical Resources, Sinks, and Link Information
VM to PM assigner Algorithm
Final Assignment Table Mapping VMs to PMs

Figure 3.3: Off-line Virtual Machine Placement.

- **One VM per a PM:** We suppose that every PM can accommodate only one VM. Although this assumption may sound unrealistic, it is always possible to consolidate several VMs as a single VM [50]. In a real world scenarios, CPU and memory capacity limits of each host determines the number of VMs that it can accommodate. A different approach is to is to bundle all VMs that can be placed in a single host into one logical VM, with the accumulated bandwidth requirements. In both cases we can thus assume with the loss of generality throughout the thesis 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' perspective: 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. As it is illustrated on Figure 1.3, there is a cost associated between any PM-Sink pair. Likewise, there is a demand between any VM-Sink pair that shows how important a given sink for a VM is (Figure 1.2). A proper placement should take into account the proximity of VMs to the sinks proportionate 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 (Figure 3.4): 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

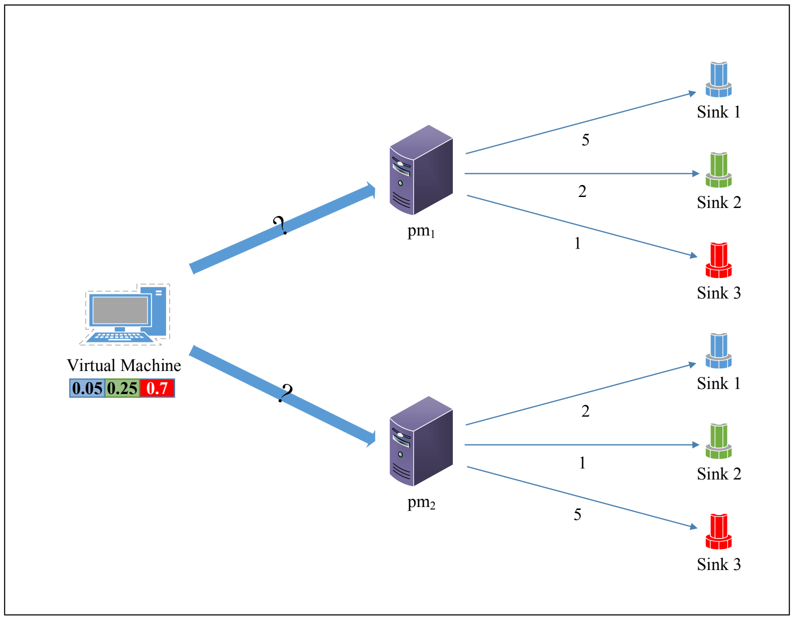


Figure 3.4: A simple example of placement decision.

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 (Equations 3.2 and 3.3):

$$x(vm, pm_1) = \frac{0.05}{5} + \frac{0.25}{2} + \frac{0.7}{1} = 0.835 \tag{3.2}$$

$$x(vm, pm_2) = \frac{0.05}{2} + \frac{0.25}{1} + \frac{0.7}{5} = 0.415 \tag{3.3}$$

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 $v_1, ..., v_{|S|}$, a set of PMs $P = \{pm_1, ..., pm_{|P|}\}$, the set of sinks $S = \{s_1, ..., s_{|S|}\}$, a static cost table D with entries 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* function $Sat(vm, pm)$ as:

$$Sat(vm, pm_i) = \sum_{j=1}^{|S|} \frac{v_j}{d_{ij} \times G(pm_i, s_j, vm)} \tag{3.4}$$

The details of D table and G function in Relation 3.4 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 (i.e., according to the Relation 3.4, 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 parameterized formula 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 (Figure 3.5). In addition to the topology, we have the following information in hand:

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

where Relations 3.5 and 3.6 hold:

$$|N| \geq |M| \tag{3.5}$$

$$|N| \gg |S| \tag{3.6}$$

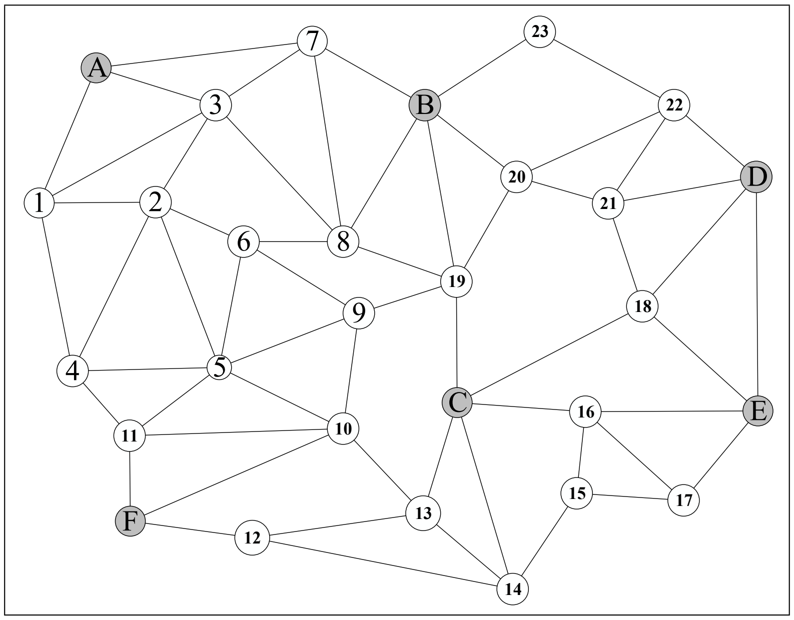


Figure 3.5: A graph representing a simple data center network without an standard topology. The nodes named by alphabetic letters are the sinks.

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 (3.7 and 3.8):

$$\sum_{j=1}^{|S|} v_{ij} = 1, \qquad \forall i \tag{3.7}$$

$$0 \leq v_{ij} \leq 1, \qquad \forall i, j \tag{3.8}$$

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 Figure 3.6, we can think of a bipartite graph $Gp = (N \cup S, E_p)$ whose:

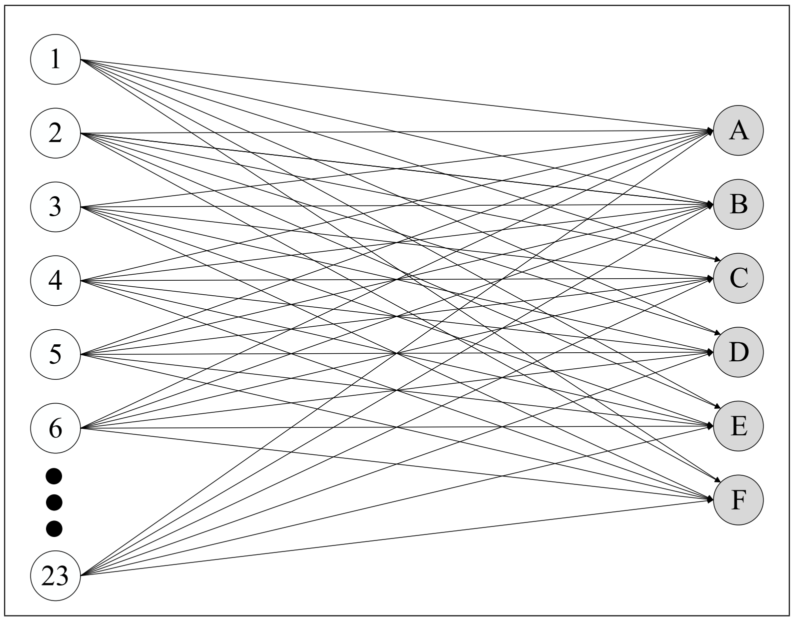


Figure 3.6: A bipartite graph version of Figure 3.5 representing the costs between PMs and sinks.

- 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. It means that depending on different assignments, the cost value on the edges connecting the physical machines to the sinks can also change. For example, according to Figures 3.5 and 3.6, 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 Figure 3.6 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 Figure 3.7. 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).

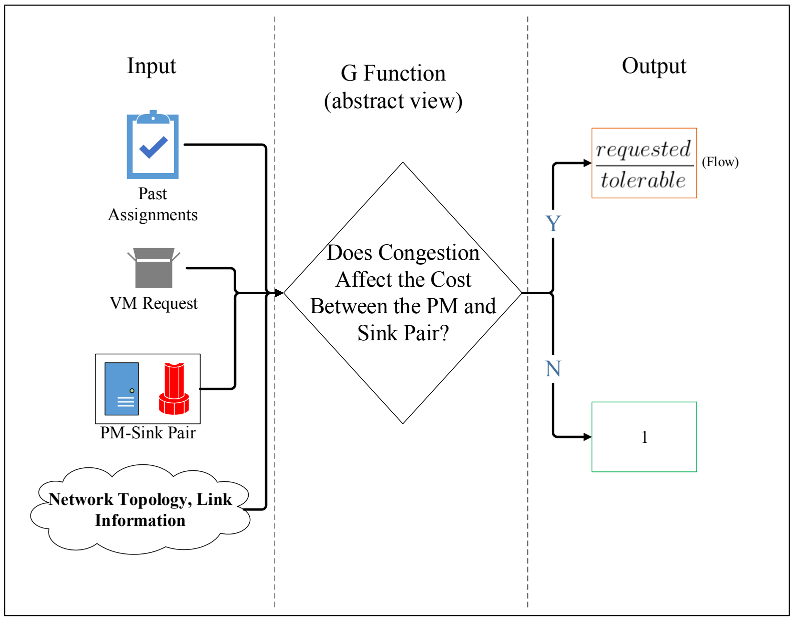

Input
G Function
(abstract view)
Output
Past Assignments
VM Request
PM-Sink Pair
Network Topology, Link Information
Does Congestion Affect the Cost Between the PM and Sink Pair?
Y
N
$\frac{requested}{tolerable}$
(Flow)
1

Figure 3.7: The abstract working mechanism of G function.

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. Otherwise, it returns the value itself. According to Figure 3.8, 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.

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

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

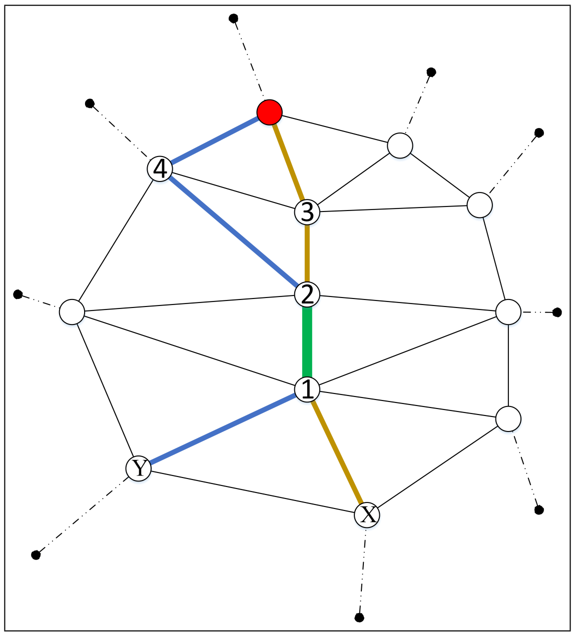


Figure 3.8: A partial graph representing part of a data center network. The colored node represents a sink. Physical machines X and Y use oblivious routing to transmit traffic to the sink. The thickest edge (1-2) is the shared link.

- **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 Figure 3.9, 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.

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

```
1: $n \leftarrow$ number of paths connecting $pm_i$ to $sink_j$
2: $TotG \leftarrow 0$
3: for all $Path$ between $pm_i$ and $sink_j$ do
4:     $MinLink \leftarrow$ the link in the $Path$ that is occupied the most
5:     $totReq \leftarrow$ total flow request passing from $MinLink$
6:     c $\leftarrow$ Total Capacity of $MinLink$
7:     $G = \frac{totReq}{c}$
8:     $TotG \leftarrow TotG + G$
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 [39].

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.

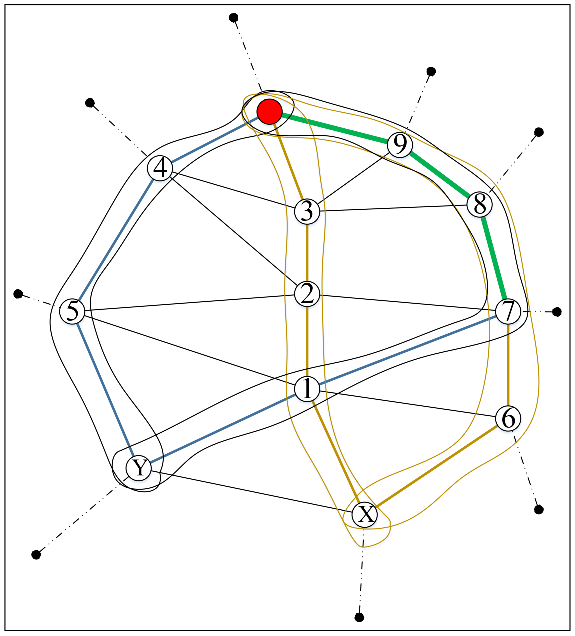


Figure 3.9: The same partial graph as shown in Figure 3.8, this time with a multi-path oblivious routing. Physical machines X and Y use two different static routes to transmit traffic to the sink. The thicker edges (7-8, 8-9, and 9-sink) are the shared links. The paths for X and Y are shown by light (brown) and dark (black) closed curves, respectively.

The maximization problem given below (3.9) is a formal representation of the problem in hand as an integer (0-1) programming. Given an assignment matrix X, VM requests, 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.

$$Maximize \sum_{i=1}^{|M|} \sum_{j=1}^{|N|} Sat(vm_i, pm_j).x_{ij} \tag{3.9}$$

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

Such that

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

$$\sum_{j=1}^{|N|} x_{ij} = 1, \text{ for all } i = 1, ..., |M| \qquad (2)$$

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

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:

3.4.1 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 [54]. 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 $\sum_{a \in A} C(a, f(a))$ is minimized:

$$Minimize \sum_{i \in A} \sum_{j \in T} C(i,j).x_{ij} \tag{3.10}$$

Such that

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

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

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

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 Hungarian Algorithm [54] 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.

3.4.2 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 [36], which is NP-hard. In Chapter 4, greedy and heuristic-based algorithms have been introduced to solve the defined problem when dynamic costs such as congestion are taken into account.

3.5 Chapter Summary

In this chapter, we represent the problem in hand using a formal description. We first start by describing the scenario and assumptions that we make. Then, we define the *Satisfaction* metric and describe the rationale behind it. After defining some network related functions such as G function, we provide a mathematical description of the problem in form of integer programming.

Chapter 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 Chapter 3.

4.1 Polynomial Approximation for NP-hard problem

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

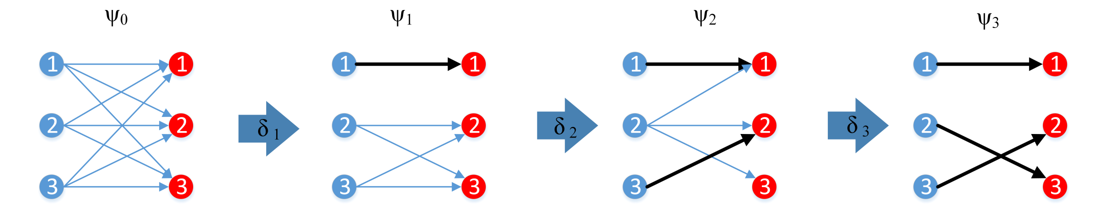
With that in mind, we can think of an algorithm for placement task that makes sequential assignment decisions that finally lead to an optimal solution (if we model the solver as a non-deterministic finite state machine). In the scenario of interest, m virtual machines are required to be assigned to n 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: $\Psi = (\delta_1, ..., \delta_{|M|})$. 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(Ψ)* for a solution Ψ. A solution Ψ_o is said to be an optimal solution if and only if $\nexists \Psi_x$, *such that* $\mathit{TotSat}(\Psi_x) > \mathit{TotSat}(\Psi_o)$. 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 $G_{vp} = (M \cup N, E_{vp})$ (as illustrated in Figure 1.4) 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 $I(\delta) = (i_{11}, ..., i_{1|N|}, ..., i_{|M||N|})$. 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 $(\delta_1, ..., \delta_r)$ by a partial solution Ψ_r in which $r < |M|$. At any point, given a partial solution Ψ_r, it is possible to calculate $\mathit{Sat}(\Psi_r)$. 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 $Sat_{\Delta}(\delta_{r+1})$ 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 4.1 exists:

$$Sat(\Psi_{r+1}) = Sat(\Psi_r) + Sat_{\Delta}(\delta_{r+1}|\Psi_r) - Sat_I(I(\delta_{r+1})|X_{G_{vp}}) \tag{4.1}$$



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	-

Figure 4.1: An example of sequential decisions.

Figure 4.1 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 are 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→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:

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

4.2 Greedy Approach

As the name suggests, in 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. Figure 4.2 demonstrates how assignment of a particular VM can affect the overall congestion and enhance/diminish the performance of other VMs.

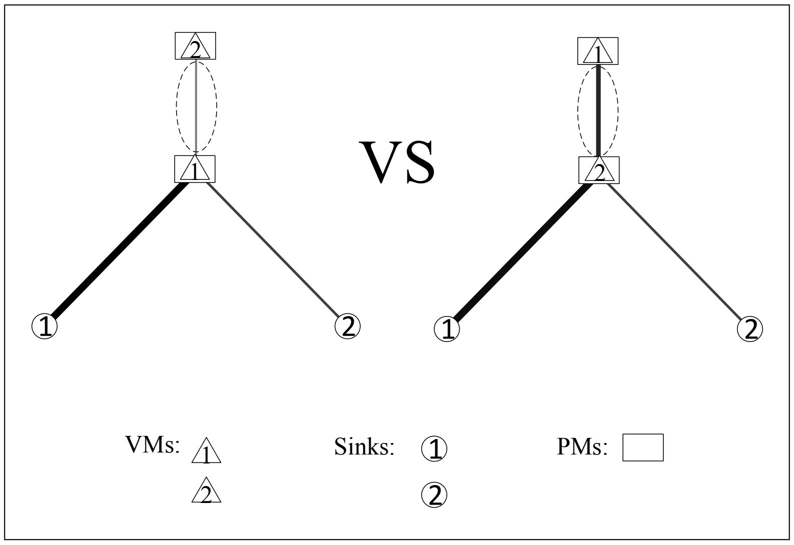


Figure 4.2: Impact of assignment on the overall congestion.

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.

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. 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 Subsection 4.2.1 in a more strong way because this time we compare the VMs competing for common sinks.

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

```
1: $X \leftarrow$ the assignment matrix
2: $z \leftarrow$ # of sinks in the DCN
3: $r[0..z] \leftarrow$ an array of $z$ lists of VMs
4: $i \leftarrow 0$
5: if $VMR.length <$ # of PMs then
6:    fill the $VMR$ with dummy VMs
7: end if
8: while $i \leq z$ 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:    $max \leftarrow 0$
13:    $maxIndex \leftarrow 0$
14:    for any VM list $L_i$ in r do
15:       $t \leftarrow L_i.getHead()$
16:       if demand of $t$ for sink $i > max$ then
17:          $max$ = demand of $t$ for sink $i$
18:          $maxIndex = i$
19:       end if
20:    end for
21:    $v \leftarrow r[maxIndex].removeHead()$
22:    $p \leftarrow$ 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
```

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 Figure 4.3, 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.

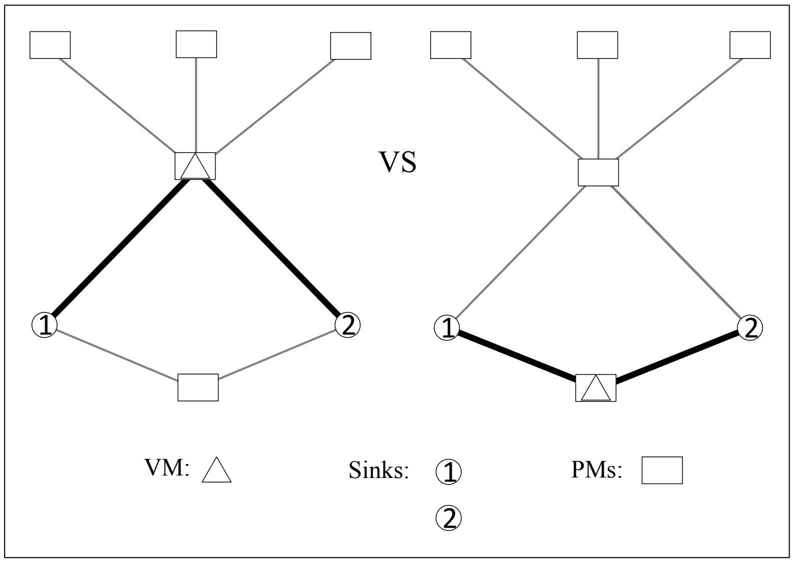


Figure 4.3: Impact of an assignment decision on unassigned VMs.

The assignment decision at the left hand side has selected the PM that maximizes the satisfaction of the given VM using 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 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).

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. For example, 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

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

```
1:  $X \leftarrow$ the assignment matrix
2:  $z \leftarrow \#$ of sinks in the DCN
3:  $r[0..z] \leftarrow$ an array of $z$ lists of VMs
4:  $i \leftarrow 0$
5:  if $VMR.length < \#$ of PMs then
6:      fill the $VMR$ with dummy VMs
7:  end if
8:  while $i \leq z$ 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:     $max \leftarrow 0$
13:     $maxIndex \leftarrow 0$
14:     for any VM list $L_i$ in r do
15:         $t \leftarrow L_i.getHead()$
16:         if demand of $t$ for sink $i > max$ then
17:             $max$ = demand of $t$ for sink $i$
18:             $maxIndex = i$
19:         end if
20:     end for
21:     $v \leftarrow r[maxIndex].removeHead()$
22:     $\overline{vm} \leftarrow$ 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 $\overline{vm}$ copies in all the remaining PMs
24:     $p \leftarrow$ the PM that offers the highest satisfaction for $v$
25:     release the virtually assigned $\overline{vm}(s)$
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
```

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 $O(n^2)$.

- **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 greedy algorithm in each trial.

4.5 Chapter Summary

In this chapter, we propose two algorithms (Greedy and Heuristic-based), each with two variants for solving the problem defined in Chapter 3. We first start by defining a general algorithm modeled as a non-deterministic finite state machine that returns a sequence of assignments that lead to the perfect solution. Then, we discuss that such an ideal algorithm may not exist but we can propose approximative approaches. Then, we explain the algorithms and the supporting intuitions. At the end of the chapter, we provide the worst case complexity analysis of the proposed algorithms.

Chapter 5

Simulation Experiments and Evaluation

We tested the effectiveness of our proposed approaches discussed in Chapter 4 using extensive simulation experiments. In this chapter, we report and discuss the results of these experiments. To do simulations, we developed a customized simulator using Java by utilizing the JUNG open source library [55] 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). Brut-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.

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 [51]. While real and modern data centers don not usually use the simple tree structure, we use this test case as an example only. Figure 5.1 shows the relative *satisfaction* of 25 VMs when placed on 25 physical machines using four different methods. Similarly, the average *satisfaction* of the VMs are shown in Figure 5.7a 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 brute-force method. 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 [39] 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 [39] (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. Figures 5.2, 5.3, and 5.4 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 Figures 5.7b, 5.7c and 5.7d.

We also performed similar experiments by applying our algorithms into two placement problems in which physical machines are interconnected without a standard topology. Figures 5.5 and 5.6 demonstrate the experiment results in a DCN with general topology consisting of 990 and 1985 non-sink with 10 and

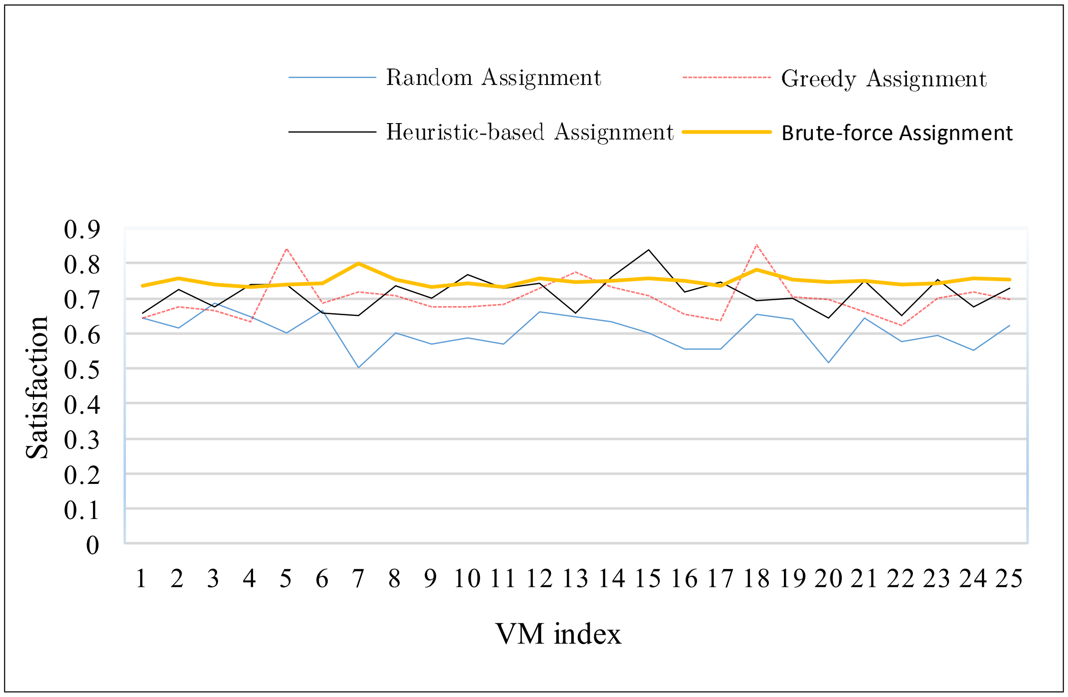


Figure 5.1: A comparison of assignment methods in a data center with tree topology having 25 normal PMs and 5 Sinks.

15 sink physical machines, respectively. According to the corresponding average *satisfaction* bar graph depicted in Figures 5.7e and 5.7f 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 Figures 5.7c, 5.7d, 5.7e, and 5.7f, 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

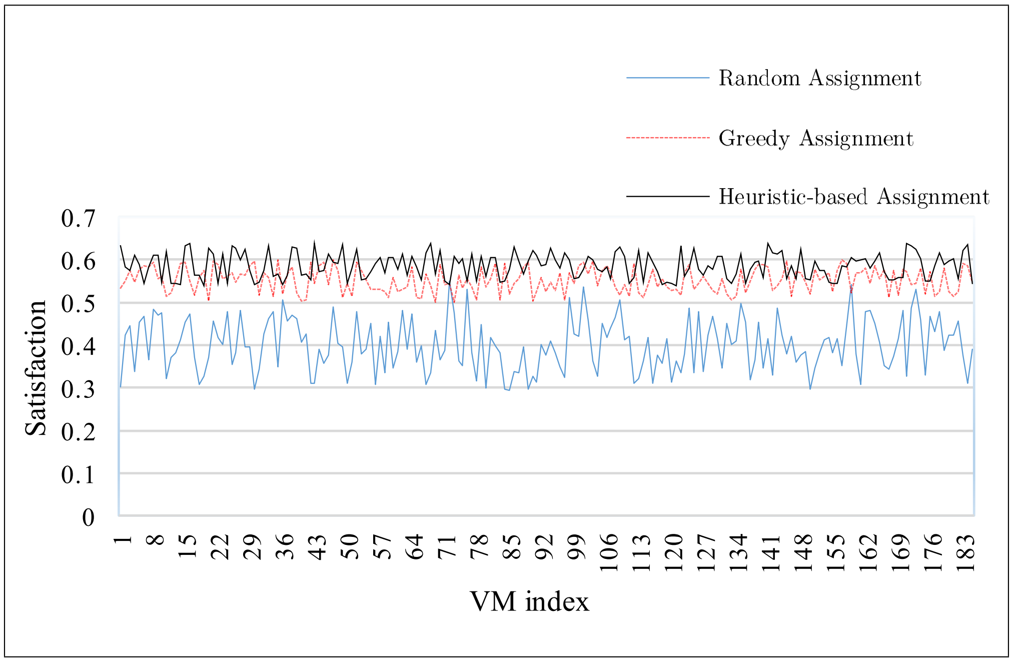


Figure 5.2: A comparison of assignment methods in a data center with VL2 topology ($D_A = 4$, $D_I = 10$) having 185 normal PMs and 15 Sinks.

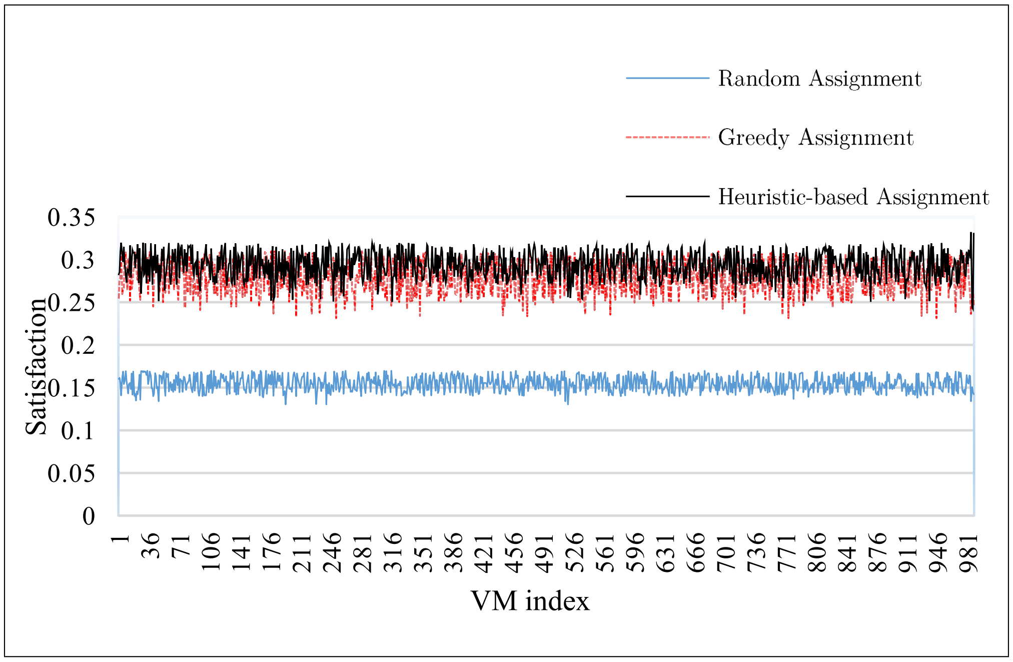


Figure 5.3: A comparison of assignment methods in a data center with VL2 topology ($D_A = 4$, $D_I = 50$) having 990 normal PMs and 10 Sinks.

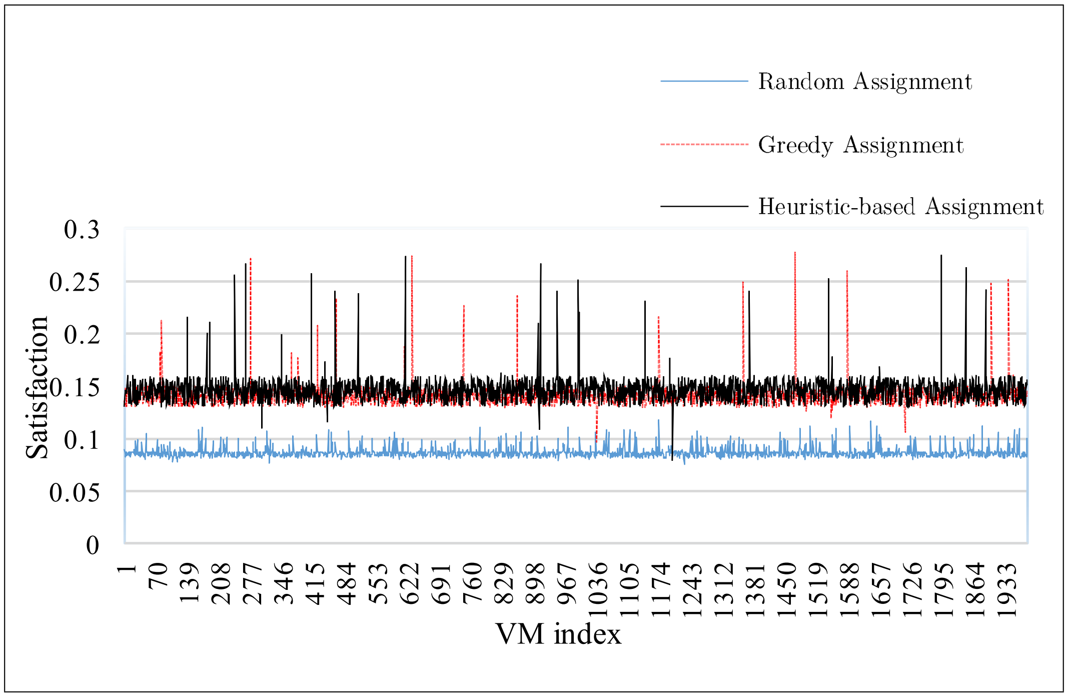


Figure 5.4: A comparison of assignment methods in a data center with VL2 topology ($D_A = 8$, $D_I = 50$) having 1985 normal PMs and 15 Sinks.

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. Figure 5.8 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

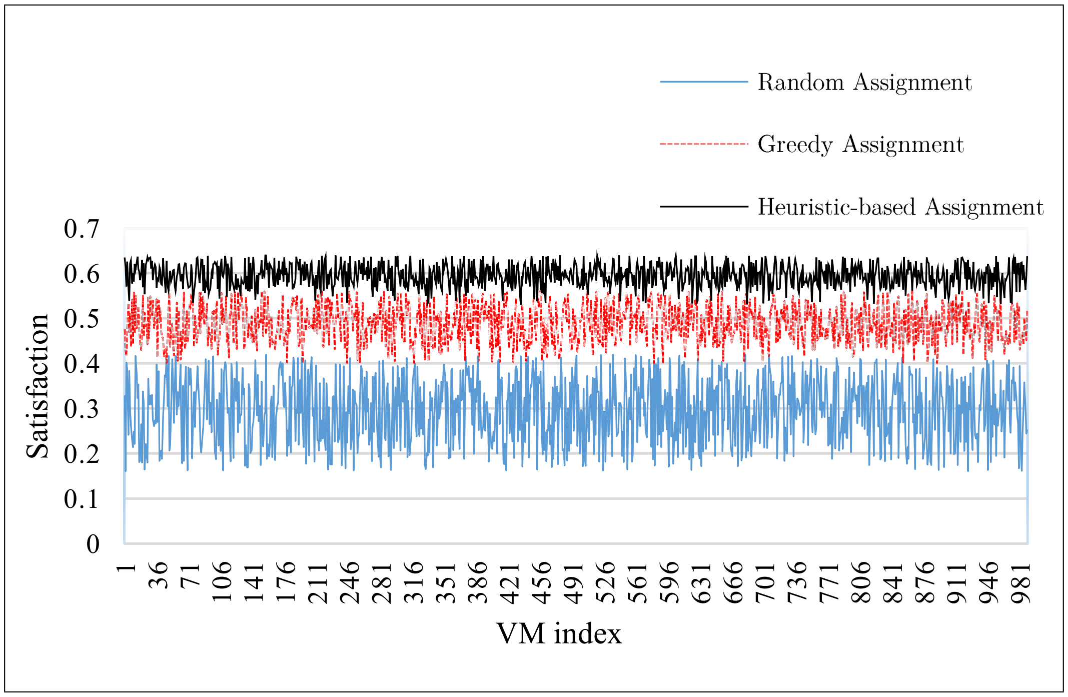


Figure 5.5: A comparison of assignment methods in a data center with general topology having 990 normal PMs and 10 Sinks.

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 is 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

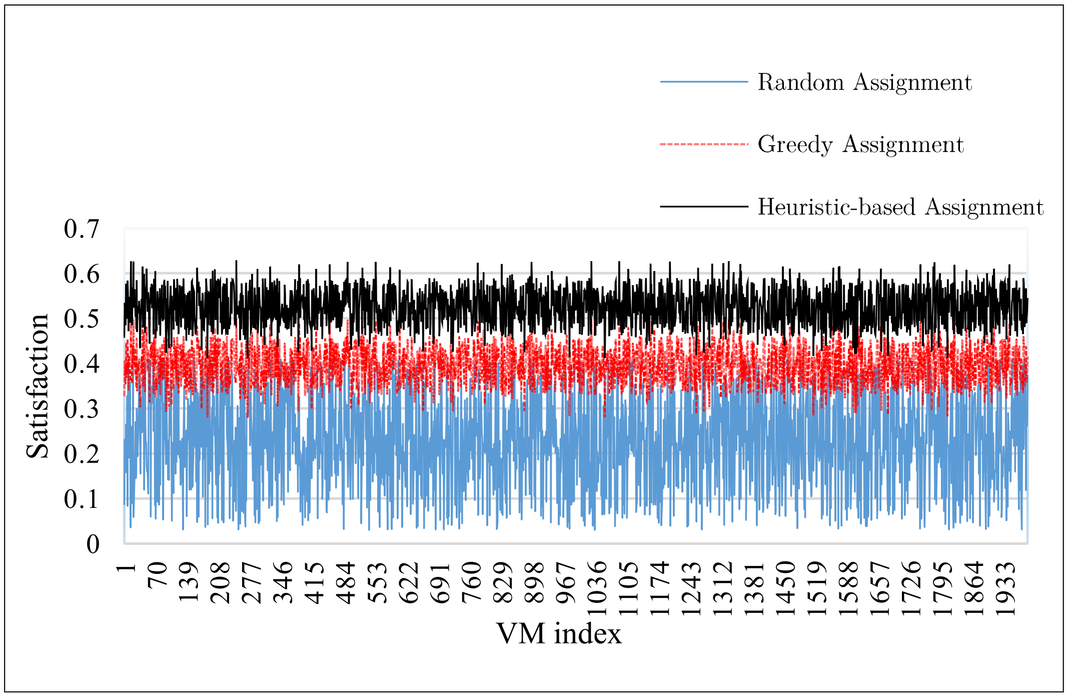


Figure 5.6: A comparison of assignment methods in a data center with general topology having 1985 normal PMs and 15 Sinks.

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. Figure 5.9 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.

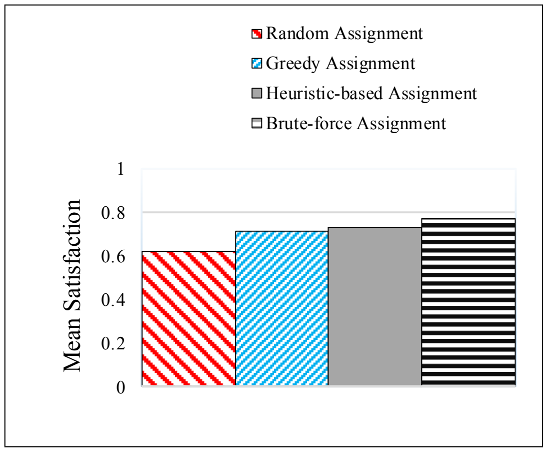
5.4 Satisfaction vs. Congestion

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 are interconnected like vertices in a general graph (e.g. consider the graph shown in Figure 3.5). For the experiment with 1985 non-sink PMs we compare the results shown in Figure 5.7f 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.

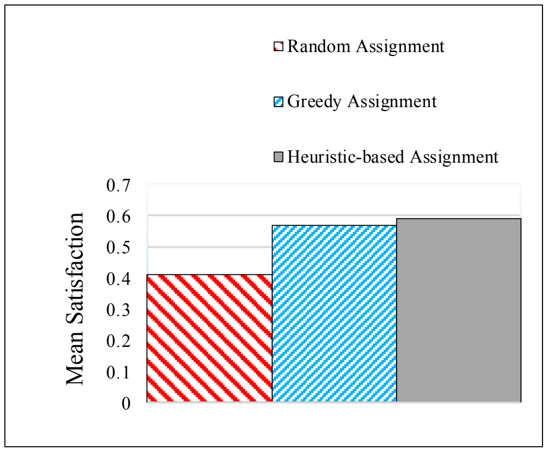
As shown in Figure 5.10, with random assignment deployed, almost 36% of the links connecting the DCN components to each other will experience a congestion level such that less than 50% of the flows that are supposed to pass through each of those links will be transmittable. Now, compare this percentage (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.

5.5 Chapter Summary

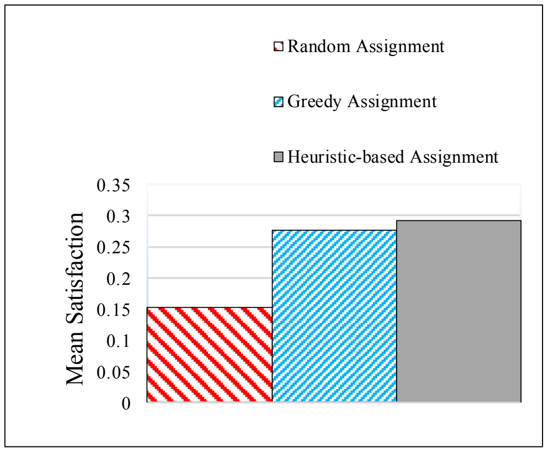
In this chapter, we compare the effectiveness of our algorithms when applied to various problems with different sizes, demand distributions, and topologies. The experiment results support the effectiveness of the algorithms in different cases. We also compare the two variants of the algorithms against each other. The correlation between *total satisfaction* and overall congestion is also put to the test.



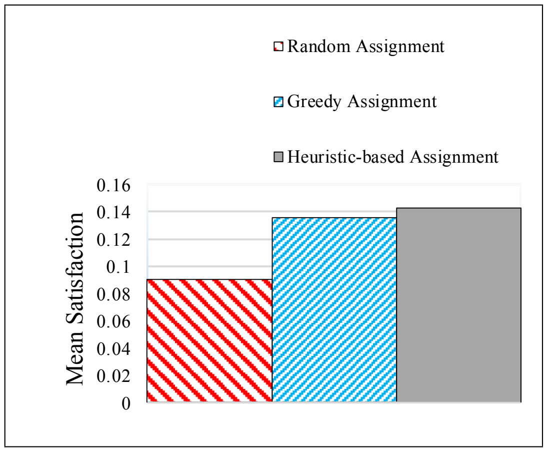
(a) Tree topology: 25 PMs and 5 sinks.



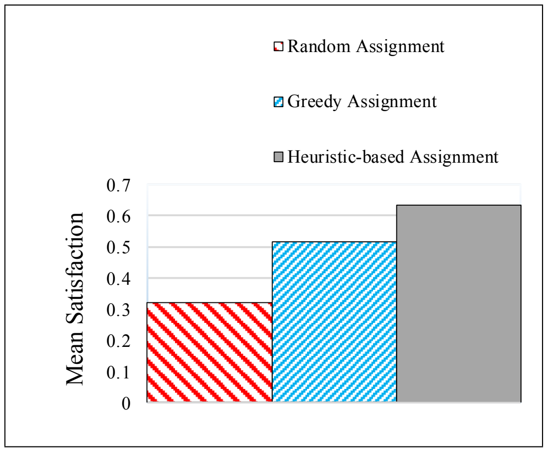
(b) VL2 topology: 185 PMs and 15 sinks.



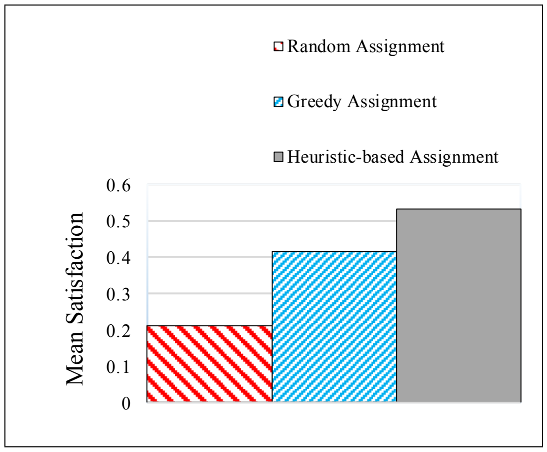
(c) VL2 topology: 990 PMs and 10 sinks.



(d) VL2 topology: 1985 PMs and 15 sinks.



(e) General top.: 990 PMs and 15 sinks.



(f) General top.: 1985 PMs and 15 sinks.

Figure 5.7: A comparison between different assignment methods based on average *satisfaction* achieved.

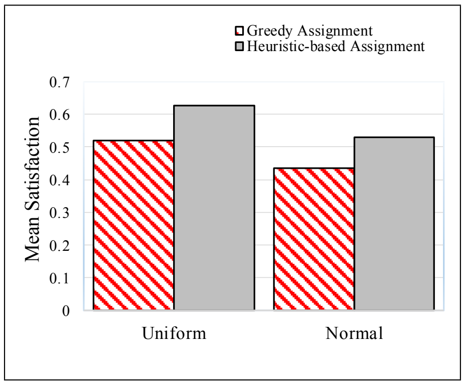


Figure 5.8: Sink demand distribution effect on the effectiveness of the algorithms.

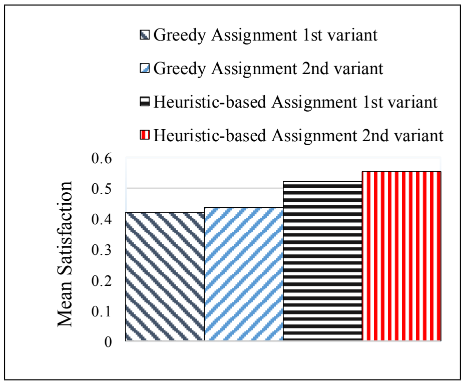


Figure 5.9: A comparison between the two variants of greedy and heuristic-based algorithms when applied to the same placement problem.

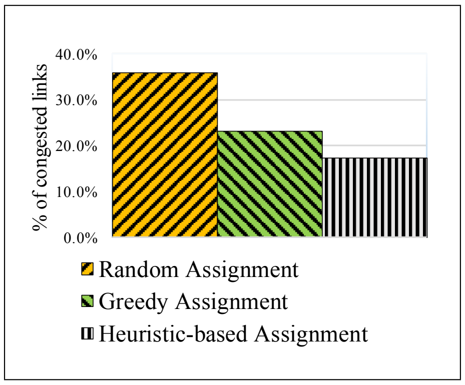


Figure 5.10: The relationship between congestion and *satisfaction*.

Chapter 6

Conclusion and Future Work

In this thesis, 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 off-line 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 especially when they are processing units and physical machines. In future works, those assumptions can be 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 [56, 57] 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 [58], 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).

Bibliography

[1] K. Mills, J. Filliben, and 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), pp. 91–98, IEEE Computer Society, 2011.

[2] IBM ILOG CPLEX Optimizer, `http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/`, last visited May 2014.

[3] Gurobi Optimization, `http://www.gurobi.com`, last visited May 2014.

[4] C. Tang, M. Steinder, M. Spreitzer, and 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), pp. 331–340, ACM, 2007.

[5] R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente, "Elastic management of web server clusters on distributed virtual infrastructures," *Concurrency and Computation: Practice and Experience*, vol. 23, pp. 1474–1490, September 2011.

[6] R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente, "Multicloud deployment of computing clusters for loosely coupled MTC applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 6, pp. 924–930, 2011.

[7] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. Lawall, "Entropy: A consolidation manager for clusters," in *Proceedings of the 2009*

ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, VEE '09, (New York, NY, USA), pp. 41–50, ACM, 2009.

[8] K. Le, R. Bianchini, J. Zhang, Y. Jaluria, J. Meng, and T. D. Nguyen, "Reducing electricity cost through virtual machine placement in high performance computing clouds," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, (New York, NY, USA), pp. 22:1–22:12, ACM, 2011.

[9] B. Hayes, "Cloud computing," *Communications of the ACM*, vol. 51, pp. 9–11, July 2008.

[10] GNU Linear Programming Kit, `http://www.gnu.org/s/glpk/`, last visited May 2014.

[11] S. Chaisiri, B.-S. Lee, and D. Niyato, "Optimal virtual machine placement across multiple cloud providers," in *Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific*, pp. 103–110, December 2009.

[12] P.-C. Yang, J.-H. Chiang, J.-C. Liu, Y.-L. Wen, and K.-Y. Chuang, "An efficient cloud for wellness self-management devices and services," in *Fourth International Conference on Genetic and Evolutionary Computing (ICGEC)*, pp. 767–770, December 2010.

[13] A. N. Toosi, R. N. Calheiros, and R. Buyya, "Interconnected cloud computing environments: Challenges, taxonomy, and survey," *ACM Computing Surveys*, vol. 47, pp. 7:1–7:47, May 2014.

[14] M. Zhou, R. Zhang, D. Zeng, and W. Qian, "Services in the cloud computing era: A survey," in *4th International Universal Communication Symposium (IUCS)*, pp. 40–46, October 2010.

[15] C. Plessl and M. Platzner, "Virtualization of hardware - introduction and survey," in *ERSA*, pp. 63–69, 2004.

[16] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," February 2009.

[17] VMware. “Understanding full virtualization, paravirtualization, and hardware assist.,” http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf.

[18] G. Kecskemeti, G. Terstyanszky, P. Kacsuk, and Z. Neméth, “An approach for virtual appliance distribution for service deployment,” *Future Generation Computer Systems*, vol. 27, no. 3, pp. 280 – 289, 2011.

[19] G. Kecskemeti, P. Kacsuk, T. Delaitre, and G. Terstyanszky, “Virtual appliances: a way to provide automatic service deployment,” in *Remote Instrumentation and Virtual Laboratories*, pp. 67–77, Springer, 2010.

[20] Software as a Service (SaaS). Cloud Taxonomy. http://cloudtaxonomy.opencrowd.com/taxonomy/software-as-a-service/, last visited April 2014.

[21] Platform as a Service. http://en.wikipedia.org/wiki/Platform_as_a_service, last visited April 2014.

[22] P. Mell and T. Grance, “The NIST definition of cloud computing,” Tech. Rep. 800-145, National Institute of Standards and Technology (NIST), Gaithersburg, MD, September 2011.

[23] M. Ahronovitz et al, “Cloud computing use cases white paper”, v4.0. www.cloudusecases.org, last visited May 2014.

[24] I. Foster, Y. Zhao, I. Raicu, and S. Lu, “Cloud computing and grid computing 360-degree compared,” in *Grid Computing Environments Workshop, 2008. GCE ’08*, pp. 1–10, November 2008.

[25] B. Venners, “Inside the Java Virtual Machine,” *McGraw-Hill Professional*, 1st ed., 1999.

[26] R. F. Stark, E. Borger, and J. Schmid, “Java and the Java virtual machine: definition, verification, validation with Cdrom,” Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2001.

[27] D. Technologies, “Datadirect technologies,” http://www.datadirect.com/, last visited April 2014.

[28] StrikeIron, “StrikeIron,” `http://www.strikeiron.com`, last visited April 2014.

[29] Amazon, “EC2,” `http://aws.amazon.com/ec2/`, last visited April 2014.

[30] ServePath, “ServePath,” `http://www.gogrid.com/servepath/`, last visited April 2014.

[31] Mosso, “Mosso,” `http://www.mosso.com/`, last visited April 2014.

[32] Skytap, “Skytap,” `http://www.skytap.com/`, last visited April 2014.

[33] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben-Yehuda, W. Emmerich, and F. Galan, “The RESERVOIR model and architecture for open federated cloud computing,” *IBM Journal of Research and Development*, vol. 53, pp. 4:1–4:11, July 2009.

[34] O. Tickoo, R. Iyer, R. Illikkal, and D. Newell, “Modeling virtual machine performance: Challenges and approaches,” *SIGMETRICS Performance Evaluation Review*, vol. 37, pp. 55–60, January 2010.

[35] J. Lucas Simarro, R. Moreno-Vozmediano, R. Montero, and I. Llorente, “Dynamic placement of virtual machines for cost optimization in multi-cloud environments,” in *International Conference on High Performance Computing and Simulation (HPCS)* , pp. 1–7, July 2011.

[36] E. M. Loiola, N. M. M. de Abreu, P. O. Boaventura-Netto, P. Hahn, and T. Querido, “A survey for the quadratic assignment problem,” *European Journal of Operational Research*, vol. 176, no. 2, pp. 657 – 690, 2007.

[37] H. Qian and D. Medhi, “Server operational cost optimization for cloud computing service providers over a time horizon,” in *Proceedings of the 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, Hot-ICE’11, (Berkeley, CA, USA), pp. 4–4, USENIX Association, 2011.

[38] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *SIGCOMM Computer Communication Review*, vol. 38, pp. 63–74, August 2008.

[39] A. Greenberg, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, and S. Sengupta, "VL2: A scalable and flexible data center network," in *SIGCOMM*, Association for Computing Machinery, Inc., August 2009.

[40] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "BCube: A high performance, server-centric network architecture for modular data centers," in *ACM SIGCOMM*, Association for Computing Machinery, Inc., August 2009.

[41] S. Kandula, J. Padhye, and P. Bahl, "Flyways to de-congest data center networks," *ACM Workshop on Hot Topics in Networks*, 2009.

[42] J. Park, D. Lee, B. Kim, J. Huh, and S. Maeng, "Locality-aware dynamic VM reconfiguration on MapReduce clouds," in *Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '12, (New York, NY, USA), pp. 27–36, ACM, 2012.

[43] S. Subashini and V. Kavitha, "A survey on security issues in service delivery models of cloud computing," *Journal of Network and Computer Applications*, vol. 34, no. 1, pp. 1 – 11, 2011.

[44] D. Erickson, B. Heller, S. Yang, J. Chu, J. Ellithorpe, S. Whyte, S. Stuart, N. McKeown, G. Parulkar, and M. Rosenblum, "Optimizing a virtualized data center," *SIGCOMM Computer Communication Review*, vol. 41, pp. 478–479, August 2011.

[45] B. Yang, F. Tan, Y.-S. Dai, and S. Guo, "Performance evaluation of cloud service considering fault recovery," in *Cloud Computing*, pp. 571–576, Springer, 2009.

[46] M. Wang, X. Meng, and L. Zhang, "Consolidating virtual machines with dynamic bandwidth demand in data centers," in *INFOCOM, 2011 Proceedings IEEE*, pp. 71–75, IEEE, 2011.

[47] O. Biran, A. Corradi, M. Fanelli, L. Foschini, A. Nus, D. Raz, and 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)*, pp. 498–506, IEEE Computer Society, 2012.

[48] X. Meng, V. Pappas, and L. Zhang, “Improving the scalability of data center networks with traffic-aware virtual machine placement,” in *INFOCOM, 2010 Proceedings IEEE*, pp. 1–9, IEEE, 2010.

[49] D. Breitgand and A. Epstein, “Improving consolidation of virtual machines with risk-aware bandwidth oversubscription in compute clouds,” in *INFOCOM, 2012 Proceedings IEEE*, pp. 2861–2865, IEEE, 2012.

[50] R. Cohen, L. Lewin-Eytan, J. Seffi Naor, and D. Raz, “Almost optimal virtual machine placement for traffic intense data centers,” in *INFOCOM, 2013 Proceedings IEEE*, pp. 355–359, IEEE, 2013.

[51] K. C. Webb, A. C. Snoeren, and K. Yocum, “Topology switching for data center networks,” in *Hot-ICE Workshop*, 2011.

[52] S. Crago, K. Dunn, P. Eads, L. Hochstein, D.-I. Kang, M. Kang, D. Modium, K. Singh, J. Suh, and J. P. Walters, “Heterogeneous cloud computing,” in *IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 378–385, IEEE, 2011.

[53] Top 500 Supercomputers in the World, `http://www.top500.org/`, last visited April 2014.

[54] J. Munkres, “Algorithms for the assignment and transportation problems,” *Journal of the Society for Industrial and Applied Mathematics*, vol. 5, no. 1, pp. 32–38, 1957.

[55] JUNG, Java Universal Network/Graph Framework, `http://jung.sourceforge.net/`, last visited June 2014

[56] V. Bilò, A. Fanelli, M. Flammini, and L. Moscardelli, “Graphical congestion games,” *Algorithmica*, vol. 61, no. 2, pp. 274–297, 2011.

[57] R. Southwell, Y. Chen, J. Huang, and Q. Zhang, "Convergence dynamics of graphical congestion games," in *Game Theory for Networks*, pp. 31–46, Springer, 2012.

[58] S. Ahmad, C. Tekin, M. Liu, R. Southwell, and J. Huang, "Spectrum sharing as spatial congestion games," *arXiv preprint arXiv:1011.5384*, 2010.