UiO : Department of Informatics
University of Oslo

# Intelligent Traffic-aware Consolidation of Virtual Machines in a Data Center

Akaki Jobava

Master's Thesis Spring 2015

# Intelligent Traffic-aware Consolidation of Virtual Machines in a Data Center

Akaki Jobava

May 18, 2015

This thesis is lovingly dedicated to my mother, Tina, for her unconditional love and support throughout my life.

# Abstract

Cloud computing is growing fast and becoming more and more popular. The computing resources such as CPU, memory and storage are becoming cheaper and the servers grow more and more powerful by the time. This enables clouds to host more virtual machines (VMs) than ever. As a result many modern data centers experience very high internal traffic inside the data centers due to the servers belonging to the same tenants communicating with each other. Since the modern VM deployment tools are not traffic-aware, the VMs with high mutual traffic often end up running far apart in the data center network and have to communicate over unnecessarily long distance. The resulting traffic bottlenecks negatively affect application performance and the network in whole and are posing important challenges for cloud and data center administrators.

This thesis investigates how this problem can be resolved by consolidating VMs in clusters in different data center network architectures and deploy the produced clusters on the available server racks in a traffic-aware way. In order to achieve this the paper breaks the problem down in two parts. The VMs are consolidated with a VM clustering algorithm, successfully reducing the total cost of communication with 34 to 85%, and the resulting clusters are assigned to the server racks with a cluster placement algorithm, which further reduces the total cost of communication with 89 to 99%. The analysis shows that the optimization is done in a fast and computationally efficient way.

# Contents

# List of Figures

# List of Tables

# Acknowledgements

I would like to express my sincere grattitude and appreciation to the following people for their help and support during my studies and my work on this thesis:

- To my supervisor Anis Yazidi for his dedication, guidance, support and kindness both during my master studies and the thesis project that helped me overcome the difficulties during this project and made this thesis possible.
- To Kyrre Bengnum for his kind advices and guidance during my studies and while working on this project.
- To all my colleagues and friends who kindly reviewed my thesis in progress in order to give me useful advices and recommendations.
- To the University of Oslo and the Oslo and Akershus University College of Applied Sciences for giving me the opportunity to take this master's program and to all the lecturers, professors and teachers for working hard to teach the students and make the education process interesting and fun.
- To all of my fellow students with whom I share so many good memories for being supportive, friendly, kind and for teaching me so much.
- To my family and all my friends both in Georgia and in Norway for support and motivation in spite of the long distance between us.
- To my little daughter Nea Sofia, who's not yet able to read this, for giving me strength and inspiration every day since the day she was born.

# Part I

# Introduction

# Chapter 1

# Introduction

Cloud computing is a relatively new concept referring to an environment where physical and virtualized computing resources are distributed and accessed over the network. Cloud computing is becoming a very central paradigm in computing. Its robustness, increasing user-friendliness, high flexibility and scalability combined with cost efficiency [12, 36, 44] make it increasingly popular amongst enterprises. According to the Intel's survey of 200 IT Managers [12] 80% of them are in the process of deploying or have already adopted private and/or public cloud by moving parts of their IT environment to it, while the remaining 20% plan to do so in the near future.

One of the main reasons behind cloud computing's success are the properties of virtualization technology which is very central in cloud computing as it allows the virtual machines (VMs) to be created, cloned, migrated, restored, etc. in a time-effective manner with little effort from the system administrator. Live migration allows VMs to be moved from one physical host to another without the customer noticing it as the service is never interrupted before, during or after the process. These characteristics of virtualization give cloud computing the robustness and flexibility enabling dynamic scaling of the infrastructure in a much more rapid and effective way compared to the traditional systems. As a result cloud computing is becoming one of the major driving forces behind the rapid growth of the data centers around the world [19].

Due to the exponential growth of the data centers and the growing computational power of the modern computers the data centers are constrained

not merely by the computational power, storage or any other computing resource but increasingly so by the networking limitations [7]. Large data centers are hosting hundreds and thousands of VMs for different cloud computing service providers. The VMs are usually consolidated with resource usage in mind with various tools, such as VMWare Capacity planner [63], Microsoft Assessment and Planning (MAP) Toolkit for Hyper-V [48] or IBM Workload Deployer [30] that help plan and carry out VM consolidation with regards to CPU, memory and disk usage. However, these tools don't take in account network usage or VM intercommunication which often results in VMs that communicate extensively with each other being placed far away from one another and having to communicate over long distances unnecessarily overloading the higher levels of the network which contains the most expensive enterprise grade equipment. Facebook experiences roughly 1000 times higher traffic usage inside its data center compared to the incoming and outgoing traffic from and to its users [42]. Bandwidth becomes a bottleneck resource in the higher layers of the network decreasing communication performance[61] for applications and increasing workload for network elements on the aggregation and core layers which in turn often results in higher power consumption of a data center [19], more greenhouse emissions and increased business costs.

These problems pose a significant challenge not only for the environment and in terms of high power usage business costs but also for the network-dependent application performance and the scalability and the growth of data centers. The 2009 study by Benson et al.[5] has shown that the link utilization in the lower layers of data centers for most of the time is very low. Thus, it is reasonable to assume that the link utilization can be optimized by traffic-aware VM deployment eliminating traffic bottlenecks and ensuring high communication performance between applications.

The aim of this thesis is to investigate an important aspect of the resource provisioning which has not received enough attention yet, namely traffic-aware virtual machine placement. In most cases the applications communicating extensively with each other in the cloud environment will belong to the same tenant. It would be beneficial for the whole network if the VMs hosting applications with high mutual traffic were deployed in closer proximity to each other. Such placement is assumed to relieve

the network elements in the upper layers of the networking infrastructure where the most expensive equipment is usually operated and fully utilize the links at the lower levels of the network. This project aims to investigate how the VMs with high mutual communication can be consolidated in clusters in order to reduce the total cost of communication. One approach to this problem could be to attempt all possible combinations of VM placements and choose the most optimal configuration. However, since data centers usually host hundreds and thousands of VMs in order to find the best possible placement for the VM number greater than 20 it would require to test astronomical number of different permutations and the task would be computationally infeasible. Therefore this project aims to break down the problem in two main parts. First the VM clusters should be detected with a graph partitioning algorithm which will consolidate VMs with high mutual traffic in clusters. The resulting clusters should thereafter be assigned to the physical hosts in the server racks in the data center. As the number of the groups will usually be significantly less than the number of VMs it will become computationally feasible to find the best possible way to assign these clusters to the server racks in the data center. An algorithm for quadratic assignment problem should be able to handle this task.

Since several new data center network architectures have been proposed in recent years the thesis will test the VM consolidation and cluster assignment on a number of different architectures in order to see what the effect of the data center network topology is on the traffic-aware VM consolidation through graph partitioning and on which of the topologies the algorithms yield the best results.

## 1.1 Problem statement

The goals of this paper are to investigate how a graph partitioning algorithm can be used in order to optimize VM placement in an intelligent traffic-aware way and also to investigate how a quadratic assignment algorithm can be implemented in order to further optimize the VM placement on the available server racks so that the VMs with high mutual traffic are placed in closer proximity to each other effectively decreasing the total cost of communication in any data center.

The paper addresses the following questions:

1. *How can graph partitioning be used in order to consolidate VMs in VM clusters in a traffic-aware way?*
2. *How can the resulting VM clusters be placed on the available server racks in order to minimize the total communication cost in any data center network architecture?*

*Algorithm* refers to a step-by-step set of operations designed to solve specific problems in computer science.

The term *Optimization* in the problem statement refers to the concept often used in the computer sciences describing the process of improving a process or a system making it more efficient.

*Graph partitioning* refers to the technique of dividing a graph, which is a representation of data, into sub-partitions where the communication between the nodes inside the sub-partitions is maximized while the communication between the sub-partitions is minimized.

The concept of *traffic-aware* virtual machine placement refers to considering traffic characteristics when making decisions on where to place the virtual machines.

## 1.2 Thesis outline

This thesis is organized in the following way:

**Chapter 1** (Introduction) provides a short overview of the current growth of the cloud computing in data centers, what challenges this is posing for internal bandwidth usage and application performance and how the thesis is aiming to address these challenges.

**Chapter 2** (Background) describes the technologies and concepts relevant in this project.

**Chapter 3** (Approach) gives a thorough description of the planned steps needed to address the problem statement, describes the experiment design, project methodology and results evaluation strategy.

**Chapter 4** (Results) describes the implemented algorithms, the experiment process and the results obtained through the experiments along with basic statistical data and the visualized output.

**Chapter 5** (Analysis) goes through the results obtained through the experiments as described in the results section and analyses them, compares them to each other and explains the observed results and behavior of the algorithms.

**Chapter 6** (Discussion and Future work) critically reflects on the course of the project, the obtained results and the analysis, discusses the approach and the alternative methods, considers the bigger picture and suggests several improvements and future work.

**Chapter 7** (Conclusion) presents the summary of the thesis by explaining how the problem statement was addressed and what the actual outcome of the research was.

**Chapter 8** (Appendix) provides the algorithm, the experiment management and the plotting and analysis scripts developed during the project.

# Chapter 2

# Background

## 2.1 Cloud computing

Cloud computing generally refers to delivering computing services over the network or the internet. Cloud consists of number of interconnected computers providing platforms or applications to the users. Virtualization technology is one of the most important technologies powering cloud computing by allowing computing resources to be shared across the cloud completely transparent to the user. As demonstrated in the introduction chapter cloud computing is gaining popularity extensively and is spreading rapidly all over the world with more and more IT professionals either in process or planning to implement private or public clouds in the near future [12], a strategy which is expected to cut considerable amount of IT expenses [36].

Some of the key features that make cloud computing an attractive choice are:

**Flexible pricing** This pricing model is often called *pay-per-use* or *pay-as-you-go* and means that customers get to pay only for what resources they have used.

**Service on demand** The resources are provided according to the needs of the customer.

**High availability** Cloud computing systems consist of numerous redundant components hidden from customer. These components make

applications, networking, storage and other services and resources redundant and highly available.

**Scalability** One of the main strengths of cloud systems is their scalability. Virtualization technologies further make scaling up or down easy and transparent to the system users.

As previously mentioned, there are four main cloud deployment models:

- Private cloud
- Public cloud
- Community cloud
- Hybrid cloud

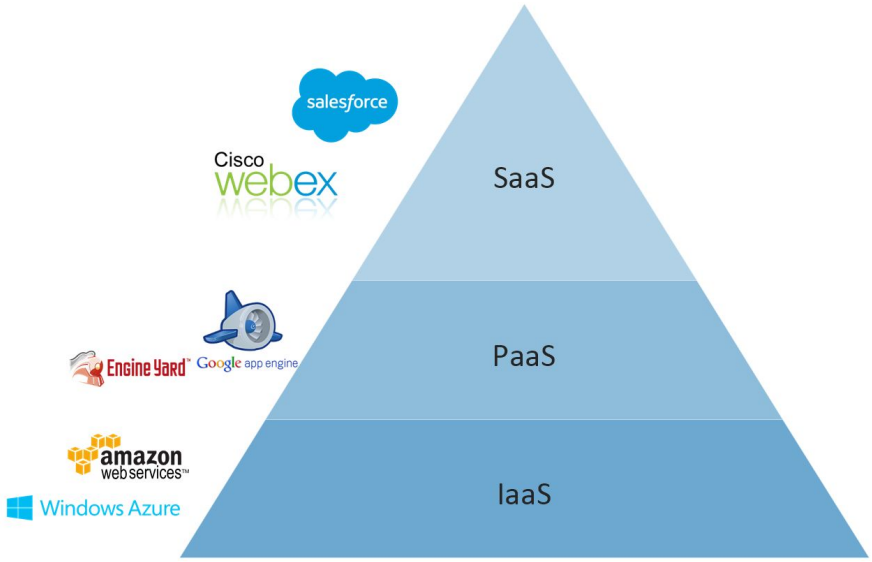


Figure 2.1: Cloud computing service models

**Private cloud** is usually a cloud environment which consists of the hardware and software owned by the company that uses it, hosted either on the premises of the organization or externally. The private cloud is normally managed, maintained, supported and utilized either by the owner or by a third party.

**Public cloud** is usually a commercial cloud environment hosted off the company premises and providing free or pay-per-usage based services over network that's available for public use. Some of the best known examples of public clouds are Amazon Elastic Compute Cloud (EC2), Microsoft Azure, IBM's Blue Cloud, Google AppEngine.

**Community cloud** is a cloud environment which is shared by two or more organizations and hosted either internally or by a third party. Hybrid cloud is a combination of two or more clouds (private, community or public) that remain unique parts but are interconnected enabling data and application portability.[16]

The main distinguishing characteristic of cloud service is that the services are often sold per use, per minute or hour. Usually the cloud provider is responsible for maintenance of the underlying software and hardware whereas the customer simply connects to the service over the internet totally oblivious of the multiple network layers and complex machinery behind the cloud.

Recent years have seen both the increase in the new cloud services providing various services as well as businesses moving their infrastructures or parts of it to the cloud in some cases presumably saving up to 37% of infrastructure expenditures over the next five years and at the same time eliminating up to 21% of support calls for their systems [36]. KPMG's 2014 Cloud Computing Survey conducted a study where 500 interviews of global business executives from over a dozen industries showed that 75% of the enterprises are experiencing improved business performance after adopting cloud-based applications and strategies [40]. The American information technology research company Gartner Group predicts that cloud computing will be the bulk of new IT expenditures by 2016 [58]. Public cloud is expected to increase from the estimated $58 billion to $191 billion by 2020 [20].

### 2.1.1 Cloud computing service models

There are three main cloud computing service models offering different types of services to their users.

- Infrastructure-as-a-Service (IaaS)
- Platform-as-a-Service (PaaS)
- Software-as-a-Service (SaaS)

### IaaS

Infrastructure-as-a-Service model enables the companies to outsource their computing equipment and other resources such as servers, networking devices, storage devices, etc. usually offers virtual machines and network components such as load balancers, switches, firewalls, etc. to customers who wish to outsource their equipment or infrastructure. The cloud provider hosts physical machines and is responsible for maintenance, monitoring and support of their equipment. Customer usually subscribes to appropriate quality of service of their choice and pays according to the agreed service level. IaaS is normally easily scaled up or down depending on the customer's requirements. Customer gets to manage applications, data, platform and operating system. The customer manages all the components except the virtualization layer, hardware and the infrastructure behind it.

Several examples of IaaS providers are Amazon AWS, Windows Azure, Google Compute Engine and Rackspace Open Cloud.

### PaaS

In Platform-asa-Service resources such as operating systems, storage, network, programming language execution environments, databases, etc. are provided over the network. This service is for example useful for developers who work on the same project from different parts of the world. The hardware behind the PaaS scales automatically to match the demands of the application used by the customers. PaaS allows the users to control the data and the applications, but not the underlying layers such as operating system, hardware, etc.

Some notable PaaS providers are Google App Engine, Engine Yard, Amazon AWS and AppFog.

### SaaS

In Software-as-a-Service model application software such as for example webmail or virtualized desktop is provided over the network by the software or service provider. Cloud providers maintain the underlying infrastructure and hardware and is responsible for the quality of service. Scaling the underlying infrastructure is completely transparent to the cloud clients. The customer has minimal control of the service and no access to the underlying components.

A few examples of SaaS are Salesforce, Cisco WebEx and Gmail.

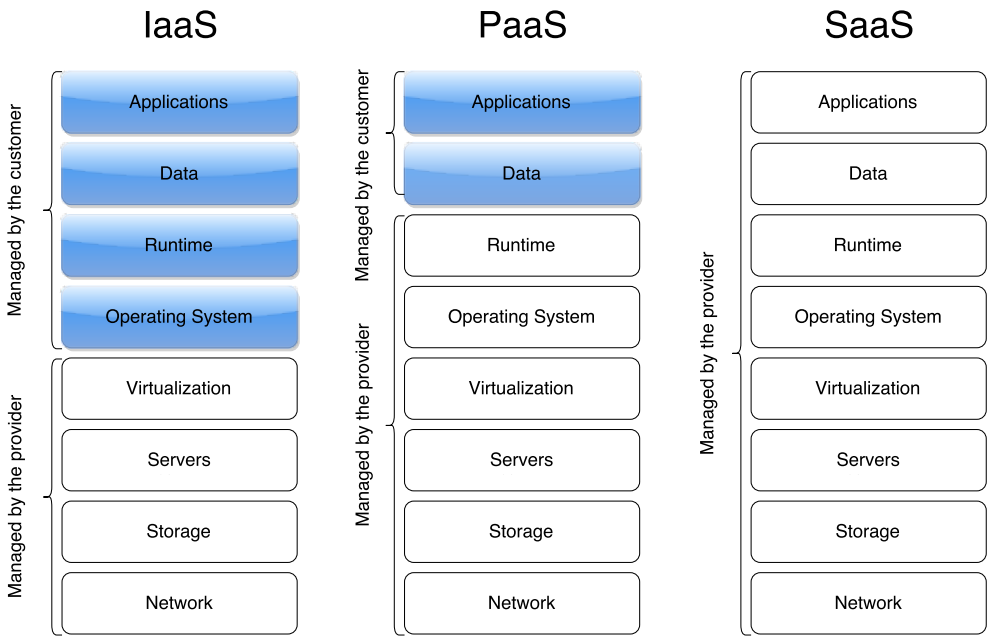


Figure 2.2: Cloud computing service models: IaaS, PaaS and SaaS. (Source: MSDN/Microsoft Azure)

### 2.1.2 Cloud platforms

Some of the leading cloud platforms have emerged since the evolution of the virtualization and cloud computing started in 1960's. Most of the major cloud computing platforms are commercial, however there are open-source alternatives as well.

### VMWare

VMWare[1] is one of the leading virtualization platforms and was founded in 1998. In 1999 VMware introduced VMware Virtual Platform and the year after VMware GSX Server 1.0 for Linux and Windows. VMWare claims to be the first to create a commercially successful x86 virtualization. VMWare's ESX and ESXi servers are bare-metal hypervisors that run directly on hardware and don't require operating system layer to function. VMWare is free to to some degree, however the advanced features require purchase of costly licenses.

### Microsoft Hyper-V

Microsoft started experimenting with virtualization back in 2003 when it acquired Connectix VPC and Virtual Server [43]. In 2004 Microsoft released Microsoft Virtual Server 2005 and then Microsoft Virtual Server 2005 R2 the following year. Microsoft Hyper-V[2] was first shipped with some versions of Windows Server 2008. Finally Microsoft introduced its Hyper-V server 2008 in October 2008. Hyper-V is a native hypervisor which creates VMs on x86-64 architecture systems. A stand-alone Hyper-V Server offers

### OpenStack

OpenStack[3] is a open-source and free cloud computing software platform which is mainly used as an infrastructure-as-a-service deployment. The development of OpenStack started in 2010 jointly by RackSpace Hosting and NASA and is currently managed by a non-profit organization OpenStack. Numerous organizations have joined the project including Cisco, Dell, AT&T, Hewlett-Packard, IBM, Intel, Linux, VMWare, etc. Multiple research and academic institutions, non-profit and commercial companies have adopted OpenStack.

OpenStack has modular architecture consisting on different components with their own codenames. Some of the main components in OpenStack are:

[1] http://www.vmware.com/
[2] https://technet.microsoft.com/en-us/windowsserver/dd448604.aspx
[3] http://www.openstack.org/

- Compute (Nova)
- Object Storage (Swift)
- Block Storage (Cinder)
- Networking (Neutron)
- Dashboard (Horizon)
- Identity Service (Keystone)
- Image Service (Glance)
- Database (Trove)

## 2.2 Virtualization

The term virtualization means creating a virtual version of something, whether it's hardware platform, operating system, network resources, storage device or server virtualization. Virtualization is the technology that allows multiple virtual machines ("guests") to share the resources of the same ("host") physical hardware. The technology was developed by International Business Machines Corporation (IBM) in the mid 1960's [62] in order to consolidate several systems into one mainframe and spare the mainframe resources. Virtualization makes it possible to set up complex computer networks consisting of multiple guest virtual machines that run all sorts of different operating systems and utilize virtualized networking and security devices, switches, routers and firewalls. The technology that gives the cloud computing systems high-scalability, reduces costs and saves computing resources, is the main driving force behind the success of cloud computing.

### 2.2.1 Types of virtualization

There are three main types of virtualization.

#### Partial virtualization

Partial virtualization refers to when some parts of the hardware are simulated. It provides a partial or a sectional simulation of the hardware in

the physical host and especially address space. As a result the operating system can't run in the virtual machine in the same way as in the full virtualization. Some of the running software needs modification in order to run. Partial virtualization was a very important milestone ultimately leading to development of full virtualization. The term partial virtualization can also be used to describe an operating system which provides address spaces for individual users or processes regardless of whether they can be considered virtual machine systems or not.

#### Para-virtualization

In para-virtualization the hardware is not necessarily simulated. Instead the guest programs run in their separate, isolated environments. The hypervisor is called Type 2 hypervisor in paravirtualization and the guest operating systems are modified in order to function as they are aware of the fact that they are being virtualized. Sometimes a dedicated VM called dom0 needs to be running in order to accommodate the management tools and device drivers. This technique is used in products such as UML and Xen.

#### Full virtualization

Full virtualization is a type of virtualization when the hypervisor runs directly on the hardware. This type of hypervisor is also called a bare-metal or Type 1 hypervisor. The guest operating systems run on top of the Type 1 hypervisor in full virtualization. The guest OS is unaware of the virtualization and requires no modifications in order to function. The hypervisor's job is to emulate device hardware at the lowest level [31]. Some examples of full virtualization are VMWare, KVM, Xen, VirtualBox, Hyper-V.

### 2.2.2 Hypervisors

The physical machine, so called "host" that runs virtual machines is called hypervisor. There are several popular hypervisors:

- Microsoft's Hyper-V
- VMware ESX/ESXi

- KVM
- Xen

There are two main types of hypervisors: type 1 and type 2 hypervisors. Type 1 hypervisors, also called bare-metal hypervisors, run straight on the hardware without an operating system in the middle (VMware ESXi, Hyper-v, Xen), while type 2 hypervisors (Oracle VirtualBox, VMware Virtual Workstation) run on top of the pre-installed operating system such as Windows or Linux.

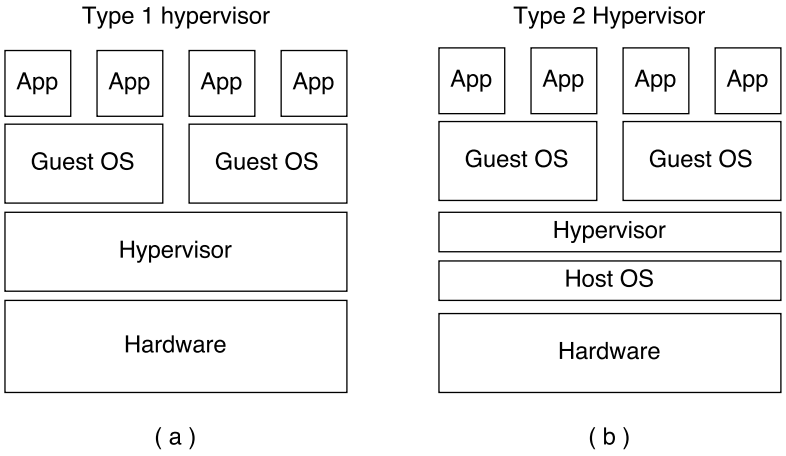


Figure 2.3: Full virtualization (a) and Paravirtualization (b)

## 2.3 Data centers

Data center, also called server farm or computer room, is a facility where majority of an organization's servers, computer systems and IT equipment are located, managed and operated. It is where the organization stores and disseminates its data from.

According to Gartner's IT Glossary page:

> *"The data center is the department in an enterprise that houses and maintains back-end information technology (IT) systems and data stores—its mainframes, servers and databases. In the days of large, centralized IT operations, this department and all the systems resided in one physical place, hence the name data center."*[4]

[4]http://www.gartner.com/it-glossary/data-center/

Data centers have evolved extensively since the so called dot-com bubble[5] of the 1990s' [3] when companies saw the need for being present on the internet and started to look for efficient ways to deploy their IT systems in a way that gave them fast internet connectivity and non-stop IT operations. By 2007 the average data center consumed as much energy as a small town [3, 22] with over five million new servers deployed each year. In 2010 it was estimated [38] that 2% of all electricity in the United States of America and about 1.3% of the electricity worldwide was consumed by data centers.

Data centers typically consist of four main components[8]:

**White space:** Usually refers to the usable raised floor environment. For the data centers that don't use the raised floor environment the term can still be used to refer to the usable area.

**Support infrastructure:** Refers to the space and equipment which is needed in order to support the data center operations. This includes power supply, uninterruptible power source (UPS), cooling systems, air distribution systems, etc. Support infrastructure can occupy much larger space compared to the white space.

**IT equipment:** Refers to all the IT equipment needed to operate the data center. This includes server racks, servers, network elements, storage systems, cabling, etc.

**Operations:** Refers to the staff that is responsible for managing, monitoring, maintaining and when required repairing and upgrading the data center. Both IT systems and the underlying physical infrastructure.

The rapid rise in popularity and the consequential expansion of cloud computing is fueling the growth of the data centers both in numbers and in size around the world. As of 2010 the data centers are estimated to consume about 2% of all electricity in the United States of America and about 1.3% of the electricity worldwide [38]. By 2007 it was estimated that the emissions from data centers accounted for about 14% of all the emissions caused by the ICT systems generally, including telecommunications devices and infrastructure and PCs and peripherals [25], and it's presumed that data cen-

[5]http://www.techopedia.com/definition/26175/dot-com-boom

ters will be responsible for 18% of emissions of all the ICT systems by 2020.

### 2.3.1 Data center architecture

Data center network is traditionally based on the *layered* [14] [56] or a three-tier approach. Three-tier network architecture consists of three layers of switches and routers (see Fig.2.4). The layered approach is designed to enhance scalability, high performance and flexibility and improve maintenance of data center networks.

**Access layer:** This is where the servers are physically connected to the network by connecting to the layer 2 switches called access or edge switches.

**Aggregation layer:** This layer provides functions such as service module integration, Layer 2 domain definitions, spanning tree and default gateway redundancy.

**Core layer:** Handles all the incoming and outgoing traffic that comes in and leaves the data center. This layer provides connectivity to various aggregation modules and it handles the layer 3 networking with access and border routers.

### 2.3.2 Top of Rack (ToR) and End of Rack (EoR) designs

Typical data centers consist of rows of server racks. A server rack, sometimes referred to as server cabinet, is usually a metal frame designed to hold various IT equipment such as servers, blade chassis, switches, routers, network patch panels, and provide power, connectivity and cooling to these components. Each rack typically contains ethernet switches and patch panels on the top, however, the switch doesn't actually have to be physically on top of the rack. These switches are referred to as Top of Rack (ToR) switches and provide non-blocking bandwidth for the directly connected nodes [50]. The advantages of ToR design are less cabling, flexible "per rack" architecture and fiber infrastructure. Main disadvantages are more

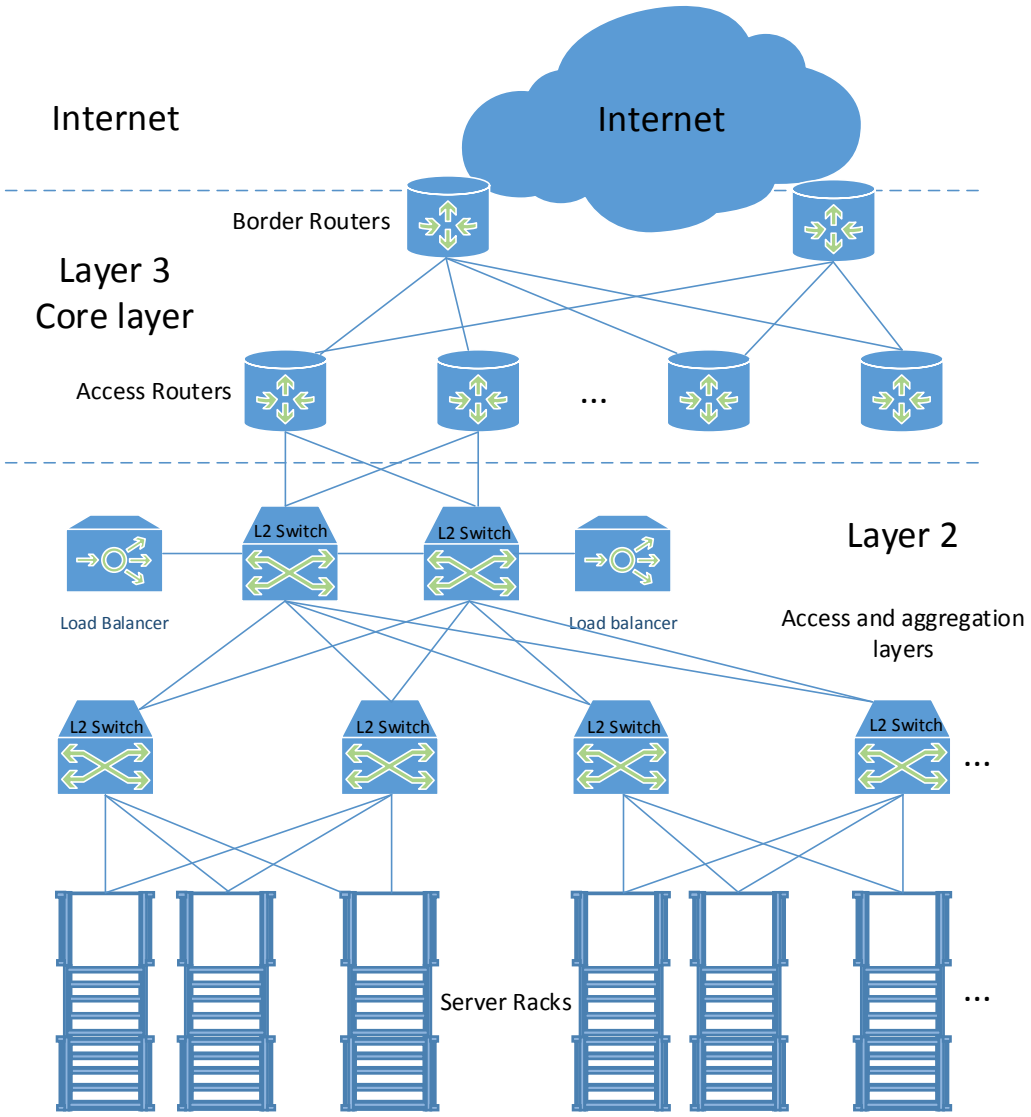


Figure 2.4: The traditional layered data center architecture

switches involved in the design and more server-to-server traffic in the aggregation layer.

An alternative design is called *end of row* (EoR) design where the hosts in the server racks are connected to a dedicated rack which is called End of Row (EoR) rack. The switches in this scenario are called End of Row (EoR) switches. The EoR switches don't actually need to be situated in the end of each row. This approach requires fewer access switches and there are fewer ports involved on the aggregation layer. On the other side expensive and bulky copper cabling is required. More patching and cable management

and less flexibility are other cons of this approach.[27]

### 2.3.3 Data center network architectures

Due to the exponential growth of the cloud in data centers and the evolution of the computers the computing power is no longer the constraining factor in the data centers. The servers are becoming increasingly powerful and as the cloud computing grows and with it the number of VMs explodes the data centers are faced with the inherent problems in the traditional data center network (DCN) architecture. The bandwidth bottlenecks, oversubscription in the higher layers and the underutilization in the lower layers of the data center network are becoming real issues [7]. Several new approaches to data center network topology have been proposed in recent years.

#### Tree topology

As previously mentioned the current data centers usually follow the traditional three-tier (or three-layer) network architectures. At the lowest level, which is called *access tier* hosts connect to one or multiple access switches. Each of the access switches is connected to one or multiple aggregate switches at the aggregation layer. The aggregation switches in turn connect to multiple core switches at the core layer. This design creates a tree-like (see Fig.2.5) topology where packets are forwarded according to a layer 2 logical topology [47]. The higher level network elements are usually enterprise-level devices and often highly oversubscribed.

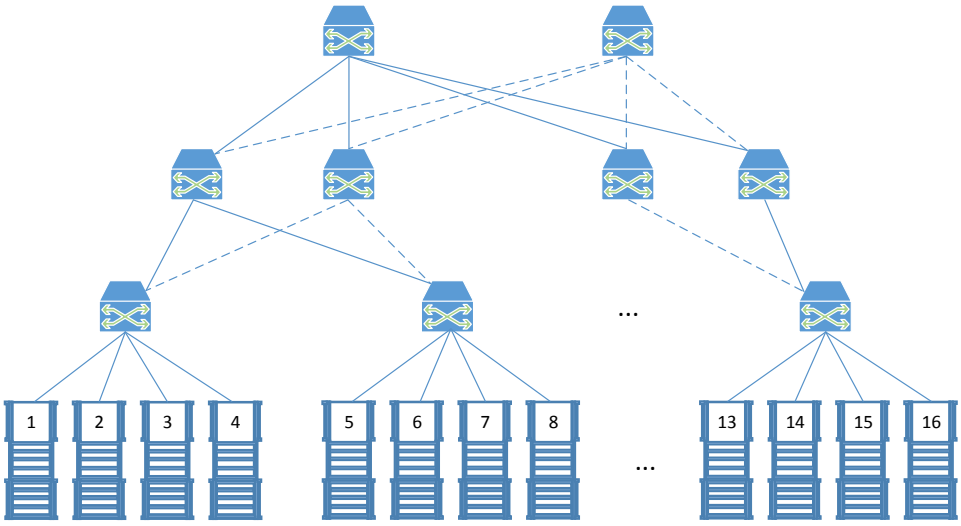


Figure 2.5: Tree (three-tier) topology

### 2.3.4 Recently proposed DCN architectures

Several new data center network architectures have been proposed as alternatives to the legacy DCN architecture.

#### PortLand (Fat-tree)

PortLand data center network architecture is an attempt to solve the cross-section bandwidth challenges of the tree-topology and makes use of the Fat-tree network topologies. The network elements in PortLand DCN follow hierarchical organization similar to the tree-topology and form a Clos topology. Fat tree is organized in pods (see Fig.2.6). Pod refers to a group of access and aggregation switches forming a complete Clos (or a bipartite) graph. In Fat tree each pod is connected to all of the core switches.

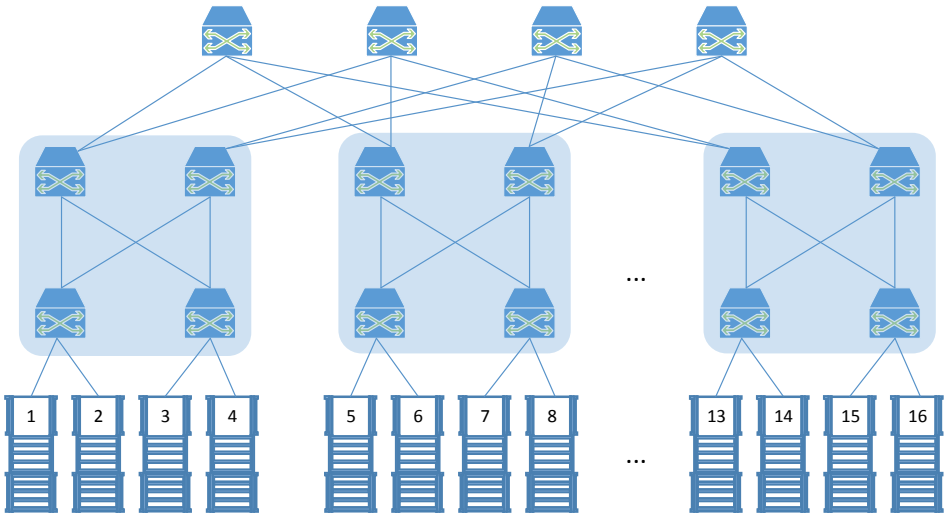


Figure 2.6: PortLand (Fat-tree) topology

Number of available ports on each switch decides the number of pods. If $k$ is the number of available ports on each switch there will be $k$ number of pods, $\frac{k}{2}$ number of access switches and $\frac{k}{2}$ number of aggregation switches in each pod. Each pod is connected to the $\frac{k^2}{4}$ core switches on the higher level and with $\frac{k^2}{4}$ server on the bottom layer. Totally, there are $\frac{5k^2}{4}$ switches connecting $\frac{k^3}{4}$ servers to each other.

### VL2

VL2 network architecture (see Fig.2.7) resembles the traditional three-tier tree architecture. It is also a three-layer architecture, however the core and the aggregation layers compose a Clos[6] topology [24].

In VL2 the data packets originating from the access switches are forwarded to the aggregation and the core layers with the use of valiant load balancing. The traffic is first forwarded to a randomly elected core switch and then forwarded back to the access layer to its actual destination switch. The idea behind this method is to provide smoother load balancing on all available links when the traffic is unpredictable.

[6]http://www.networkworld.com/article/2226122/cisco-subnet/clos-networks–what-s-old-is-new-again.html

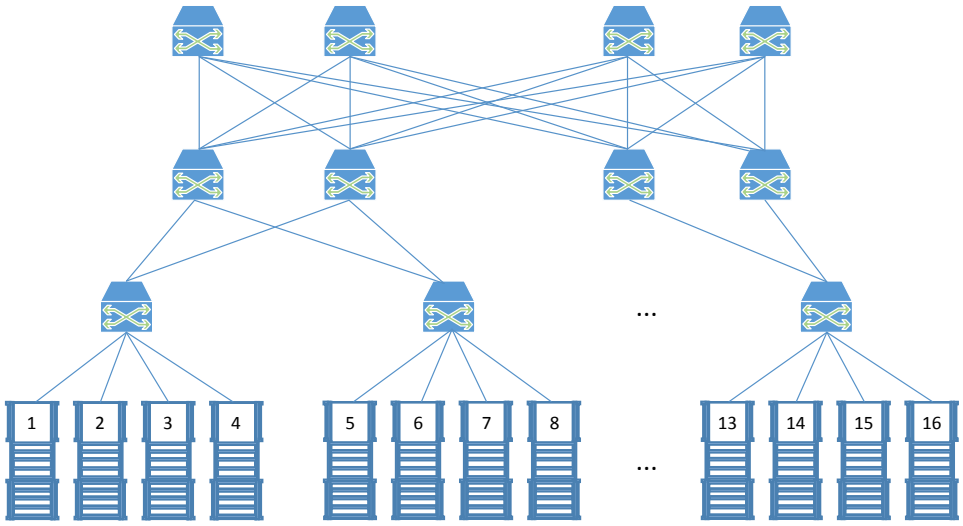


Figure 2.7: VL2 topology

### BCube

BCube (see Fig.2.8) is a multi-level server-centric DCN architecture. Server-centric refers to an architecture where servers become part of the networking infrastructure and participate in packet forwarding for other servers.

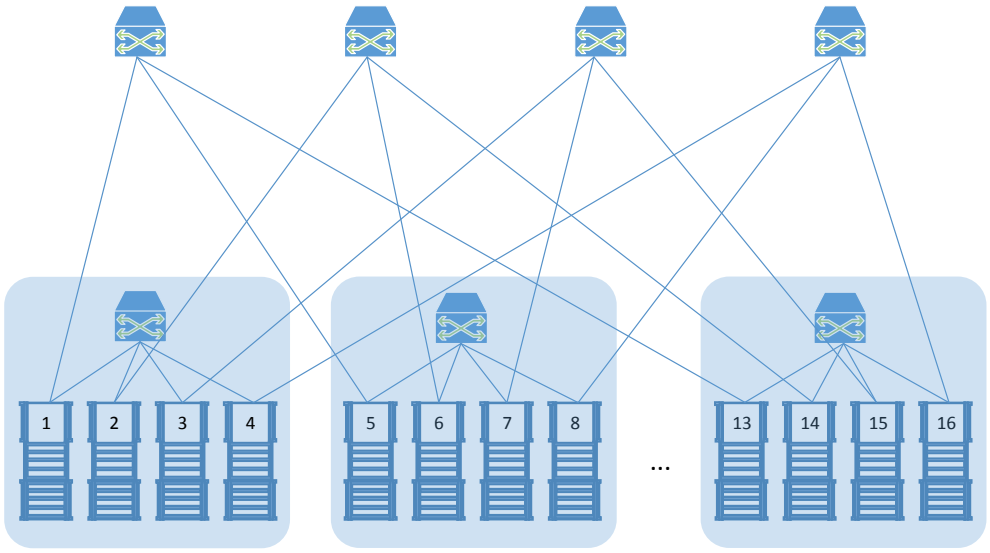


Figure 2.8: BCube topology

### 2.3.5 Cost matrix

A cost matrix (or a distance matrix) is a two dimensional array which contains information about the communication cost (or the distance) between the pairs of nodes in a set of nodes. The matrix usually has a $NxN$ dimension where $N$ is the number of the nodes in the set of nodes. Each row in the matrix corresponds to a single node denoted by $i$ and each column also represents a single node and is denoted by $j$.

$$C_{ij} = \begin{bmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,j} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,j} \\ \vdots & \vdots & \ddots & \vdots \\ c_{i,1} & c_{i,2} & \cdots & c_{i,j} \end{bmatrix} \tag{2.1}$$

In the sample matrix displayed above each element of the matrix represents cost of communication or a distance from the node $i$ to node $j$.

Cost matrix should not be confused with adjacency matrix. The main difference is that the adjacency matrix merely shows which nodes are connected to each other ignoring the communication costs between them. Cost matrix can be either asymmetric or symmetric. In some cases first symmetric matrix is constructed when connection costs between nodes are different depending on the "direction". After obtaining an asymmetric cost matrix a symmetric cost matrix can easily be calculated by computing average costs between the nodes.

## 2.4 Graph partitioning

Graph partitioning (GP) refers to division of data into sub-partitions (see Fig.2.9) so that the communication between the sub-partitions is minimized while inter-partition communication is maximized [17]. If data is represented as a graph $G = (V, E)$, where $V$ are vertices and $E$ are edges, graph partition is dividing the graph G into smaller partitions with specific characteristics according to given constraints.

Uniform graph partitioning refers to graph partitioning where sub-partitions are about the same size and where connections between the par-

titions is minimized.

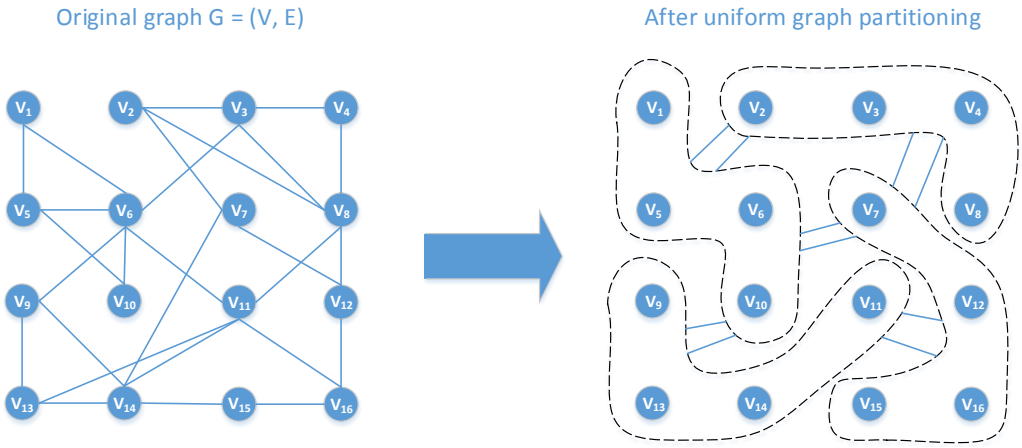


Figure 2.9: Example of partitioning graph $G = (V, E)$

The figure 2.9 illustrates how the original graph $G = (V, E)$ is partitioned into four uniform sub-partitions. The inter-partition communication (communication between the nodes inside partition) is maximized while the communication between the partitions is minimized.

Computer scientists frequently use graphs as data abstractions when constructing problem models [9]. Even if the ultimate problem is something else graph partitioning can still be used as a sub-problem for complexity reduction or parallelization.

If we assume graph $G = (V, E)$ to be any graph with an even number of vertices, V. The graph partitioning problem (GPP) involves partitioning V into two node sets (or groups) $V_1$ and $V_2$ (where $|V| = |V_1| + |V_2|$ and $V_1 \cap V_2 = \emptyset$) such that the sum of the edge-cost having endpoints in different sets is minimized. If $C_{ij}$ is the symmetric cost of the edge connecting nodes i and j, the graph partitioning problem is the following nonlinear optimization problem:

$$Minimize \sum_{i \in V} \sum_{j \in V} c_{ij} \cdot x_i \cdot (1 - x_j) \tag{2.2}$$

$$Subject\ to \sum_{i \in V} x_i = N \tag{2.3}$$

$$where\ x_i \in \{0,1\}\ for\ all\ i \in V;\ and\ x_i = 1 \implies i$$

$$is\ in\ set\ V_1;\ x_i = 0 \implies i\ is\ in\ set\ V_2$$

The graph partitioning problem can be rewritten in an unconstrained form:

$$Minimize \sum_{i \in V} \sum_{j \in V} c_{ij} \cdot x_i \cdot (1 - x_j) + \Pi \tag{2.4}$$

Where $\Pi$ is a penalty measure associated with 2.3. The 2.4 formulation was utilized by Johnson et al. [33] in 1989.

Due to the fact that there is no exact algorithm to find the solution to the GPP the most optimal way is to use an exhaustive search of the whole solution space. However, the solution space can be too large depending on the number of vertices. In case of $|V| = 100$ the solution space has more than $10^29$ solutions. Thus, it not be computationally feasible to find the exact solution and instead it might be reasonable to go for near-optimal solution.

### 2.4.1 GPP problem complexity

Usually graph partitioning problem (GPP) belongs to NP-hard problems and is resolved by employing heuristic, approximation or optimization algorithms. However, uniform GPP (also called *balanced* graph partitioning problem) is known to be NP-complete to approximate.

In computational complexity theory [54, 64] NP refers to *nondeterministic polynomial time* and is the most fundamental complexity class. Problems are assigned to NP class when they are solvable in polynomial time by a nondeterministic Turing machine [60]. An algorithm is said to be solvable

in polynomial time if the number of steps required to complete the algorithm for a given input is $O(n^k)$ for some non-negative integer $k$, where $n$ is the complexity of the input[7]. Polynomial algorithms are considered to be fast and efficient. Some examples of mathematical operations that can be completed in polynomial time are addition, subtraction, multiplication, division, square roots, powers and logarithms.

A problem is known to be NP-hard if the algorithm for solving it can be translated into an algorithm for solving any NP-problem. If a problem is both verifiable in nondeterministic polynomial time (an NP-problem) and is also an NP-hard problem is known to be an NP-complete problem.

### 2.4.2 Graph partitioning algorithms

Over the years of research various approaches have been proposed by different researchers in order to come up with the best possible solution for the GPP with the use of different algorithms.

**Kerninghan-Lin Algorithm** [35].

The Kerninghan-Lin (KL) graph partitioning algorithm was developed in 1969 and has been considered as one of the best heuristic algorithms for years. The strength of the KL algorithm is the ability to quickly find "good" solutions. The KL algorithm is based on the idea that some nodes are more strongly connected than others and need to be moved between potential solution sets. This approach uses the observation that the connected vertices with weighty edge costs tend to form clusters. The researchers concluded that swapping groups of vertices between the temporary solution sets was more probable to yield better results compared to swapping individual pairs of nodes. The disadvantage of the KL algorithm is the fact that the results are not consistently of high quality [59].

**The Extended Local Search Algorithm**

The extended local search algorithm (XLS) was developed in 1991 by Rolland et al. [59]. The XLS is related to a local search scheme. The Algorithm LS works on a current partitioning and modifies it by moving a pair of nodes between the sub-partitions. The algorithm obtain a solution

[7] http://mathworld.wolfram.com/PolynomialTime.html

by swapping pairs of nodes between two sub-partitions with an additional constraint, that each node should note be moved more than once. The LS searches the neighborhood of the current partition in attempt to locate a local optimum. Given any possible solution a better solution can be found by swapping a single pair of nodes. Due to the large solution space size repeated invocation of Algorithm LS may not succeed in finding the optimal solution, however the Algorithm LS works well for dense graphs when invoked repeatedly due to the multiple "almost optimal" solutions.

### 2.4.3 Graph partitioning using learning automata

Oommen introduced a learning automata-based graph partitioning algorithm in 1996 in the paper "Graph Partitioning Using Learning Automata" [51]. The work proposes a novel approach to solving GPP by using learning automata and viewing the problem not as a searching or a parameter-based training, but as an object partitioning problem.

#### Learning automata

Research of learning automata goes back to the work of Tsetlin in 1960s in Soviet Union, however the *learning automaton* term was first used by Narendra and Thathachar in a survey paper of 1974 [49].

Learning automaton is a decision-making device, an algorithm that adaptively chooses from a set of different actions on a random environment. The automata approach to learning represents the determination of the optimal action from the set of finite available actions [45, 46]. After learning automata applies an action to the random environment feedback is generated by the environment which is used by learning automata in order to learn the optimal action. Learning automata can be useful in resolving optimization problems or for statistical decision-making.

Learning automata can be useful when addressing graph partitioning problems (GPP) when the graph is being divided into sub-graphs according to the "similarities" and "dissimilarities" of the graph nodes [51]. The environment is constantly being changed and the automata makes decisions according to the pre-programmed decision set based on the altered

graphs.

**Object partitioning based on learning automata**

As previously mentioned Oommen's approach views the GPP not as a searching or parameter-based training, but rather as an object partitioning problem. The algorithm checks random pairs of nodes and compares them to each other in order to determine which of them are "similar" or "dissimilar". The similarity of nodes is determined by how strongly interconnected they are and how small their corresponding edge is. This information is used to decide whether or not the pairs of nodes belong in the same sub-partition. The migrations are done in the pairwise mode and this is achieved by using previous subpratition patterns in order to intelligently partition the entire graph. The work is the first one to not only group the nodes but also quantify the "closeness of fit" of how well the nodes belong to the assigned sub-partitions. This is achieved by intelligently pushing the vertices further and further "deep" into their corresponding sub-partitions or by doing the opposite, depending on to what degree they belong in the current sub-partition. This approach also helps nominating the "best" node for each sub-partition which is referred to as the *nucleus* of the respective sub-partition.

**Algorithm performance**

The algorithm is invoked repeatedly and at each invocation the nodes in the randomly chosen pair are either penalized or rewarded depending on whether or not they're "similar" or "dissimilar" and whether or not they belong to the same sub-partition. Oommen's algorithm is space-inexpensive (doesn't use temporary sub-partitions, but rather sorts the nodes in place) and fast in finding good solutions which can be further improved. The automata based graph partitioning algorithm [51] outperforms previously suggested Kerninghan-Lin's and Rolland et al. algorithms [35, 59]

## 2.5 Facility location problem

Facility location problem (FLP) refers to a problem of placing facilities and allocating customers to the facilities in a way that minimizes of total service

cost [6].

The study of location theory goes back to 1909 and it started out when Alfred Weber studied how to position a single warehouse in a way that would minimize distance between it and several customers [53].

In a virtualization-based data center facility location problem is relevant when making decisions about placing tenants or VMs on various racks as it is often of high importance that the VMs are placed in such way that minimizes the cost of communication between them and ultimately decreases oversubscription in the upper layers of the network.

### 2.5.1 Quadratic assignment problem

The quadratic assignment problem (QAP), which was first proposed by Koopmans and Beckman in 1955[39], is a well-known NP-hard combinatorial optimization problem from the facility location problems category in mathematics. Facility location problem is the most common application area for QAP. However QAP is also applied to problems in statistical analysis, chemistry, parallel and distributed computing, archeology, chemistry, scheduling, etc [32, 57].

Given $N$ facilities $f_1$, $f_2$ ... $f_N$ and $N$ locations $l_1$, $l_2$... $l_N$

Let $T^{NxN} = (t_i, t_j)$ be a positive real matrix, where $t_{i,j}$ is the flow between facilities $f_i$ and $f_j$.

Let $C^{NxN} = (C_{i,j}$ be a positive real matrix, where $c_{i,j}$ is the distance between locations $l_i$ and $l_j$.

Let $p : \{1, 2, ..., N\} \rightarrow \{1, 2, ..., N\}$ be an assignment of the N facilities to the N locations.

Cost of the assignment is defined as follows:

$$c(p) = \sum_{i=1} \sum_{j=1} a_{i,j} \cdot b_{p(i)p(j)} \tag{2.5}$$

The quadratic assignment problem: Find a permutation vector $p$ which minimizes the cost of assignment.

$$Minimize\ c(p) : subject\ to\ p \in \Pi_N$$

QAP is known to be computationally one of the most difficult problems in NP-hard class [39] and there is a general consensus that finding the optimality of QAP problems with size > 20 is practically impossible [47]. Various heuristic methods have been developed to solve the QAP.

### 2.5.2 Simulated annealing

Simulated annealing (SA) is a generic probabilistic metaheuristic for the global optimization problem presented by Kirkpatrick et al.[37] in 1983 and Cerny [13] in 1985. SA belongs to the general iterative algorithms in the approximation algorithms class [65]. SA algorithms don't guarantee finding an optimal solution. They don't know when such solution is reached and need to be stopped with some mechanism at some determined point.

SA is often implemented when the search space is discrete or when it is acceptable to find a good enough solution in a certain fixed amount of time instead of finding the best possible solution. The name of the metaheuristic is inspired by annealing in metallurgy [18]. A technique involving heating metals in order to alter their physical or chemical properties and then cooling them in a controlled way. In SA probability of accepting worse solutions slowly decreases as the SA explores the solution space. This resembles the controlled and gradual decrease of the temperature during annealing in metallurgy, hence the name.

The simulated annealing process involves following steps:

1. Randomly alter the state
2. Assess the energy of the new state
3. Compare the energy of the current state to the previous state and decide whether or not to move to the new state.
4. Repeat until the acceptable solution is found

In order to avoid being trapped in the local minima one of the following conditions must be met for the altered state to be accepted:

- The alteration caused decrease of the energy
- The alteration caused increase of the energy, but within the bounds of the energy (which gradually decreases)

## 2.6 Related research

Due to the exponential growth of the cloud computing a more efficient resource provisioning in data centers has become an increasingly critical issue and has been attracting attention from researchers. There has been proposals for a more efficient and scalable data center network architectures such as VL2 [24] and PortLand [50]. However, some researchers have suggested a different, traffic-oriented VM consolidation approach to the problem.

### 2.6.1 Network-aware Virtual Machine Consolidation for Large Data Centers

Kakadia, Kopri and Varma address the internal bandwidth optimization problem in a data center by identifying the virtual machine groups based on the network traffic in the Network-aware Virtual Machine Consolidation for Large Data Centers paper [34]. The paper proposes a greedy consolidation algorithm to ensure small number of migrations and fast placement decisions. The work proposes algorithms to form VMClusters, to select VMs for migration and to place them using the cost tree. The experiment is evaluated in an extended NetworkCloudSim [21] with software defined network (SDN) functionality support and Floodlight[8] as the SDN controller. Performance improvement in runtime of jobs were measured and it was concluded that I/O intensive jobs had been benefited the most. However, the short jobs also showed significant improvements. In terms of traffic localization the results compared to other approaches showed significant improvements. The ToR traffic showed $\sim$60% increase while $\sim$70%

[8]http://www.projectfloodlight.org/floodlight/

reduction was measured in core traffic.

### 2.6.2 A Network-aware Virtual Machine Placement and Migration Approach in Cloud Computing

Piao and Yan [55] use a hypothetical scenario where a customer requests a data storage space and VMs from a cloud service provider in order to host the applications and process data. In this scenario the resources are arbitrarily provisioned without taking in account traffic usage and as a result the data has to travel unnecessarily long distance. The paper proposes VM placement and migration approach to be deployed in the host broker which is responsible for resource allocation. The VM placement algorithm makes sure that the new VMs are placed intelligently so that the communication occurs over the shortest possible path while the VM migration algorithm is triggered when the communication between existing resources suffers due to some latency issues on the network. The latter algorithm is triggered when the predefined service level agreement (SLA) based on the execution time of the application is breached. The VM migration algorithm relocates the affected VM(s) intelligently to the physical host with better network status. The experiment was conducted on the CloudSim 2.0 [11] data center simulation environment and the results showed improved task completion time.

### 2.6.3 Improving the Scalability of Data Center Networks with Traffic-aware Virtual Machine Placement

Meng, Pappas and Zhang [47] address the network scalability problem by formulating the VM placement as an optimization problem and propose a two-tier approximation algorithm to solve it for very large problems. The paper takes in account recently proposed data center network architectures. The real-life production data center traffic traces are used in the experiment and significant improvements are shown compared to existing methods that don't take in account traffic patterns and data center architectures. The paper specifies the network-aware VM placement problem (TVMPP) and attempts to optimize it by minimizing average traffic latency which is caused by the network infrastructure assuming that each network element

causes equal delay of communication between the VMs. Cluster-and-Cut algorithm, which leverages the unique features of the traffic patterns and network topologies is used to optimize the problem. The algorithm has two major components: 1) SlotClustering and VMMinKcut. The results of Cluster-and-Cut and two benchmark algorithms LOPI [1] and SA [10] are compared with each other in an experiment where 1024 slots and VMs are used. It is concluded that the function value given by the Cluster-and-Cut algorithm is $\sim$10% smaller compared the two benchmarks.

### 2.6.4 Starling: Minimizing Communication Overhead in Virtualized Computing Platforms Using Decentralized Affinity-Aware Migration

Sonnek et al. [61] introduce a decentralized affinity-aware migration technique for allocating virtual machines on the available physical resources. The technique monitors the network affinity between the pairs of the virtual machines and uses a distributed bartering algorithm together with VM migration in order to dynamically move VMs in a way that ensures that the communication overhead is minimized. This is achieved by placing the VMs with high mutual traffic as close to each other as possible, whether putting them in the same server rack, cluster or local network. The main contributions of the paper are: Affinity-based VM placement and migration, implicit inference of dynamic job dependencies and decentralized control. The affinity-aware migration algorithm runs on each node and consists of the traffic monitoring and fingerprinting, affinity inference and bartering and migration components. The experiment is conducted on a 7-node Xen-based cluster. The Intel MPI benchmark suite[9] and Cube MHD Jet (Cube)[10] were used for simulation and benchmarking. The results showed 42% improvement in the runtime of the application over a no-migration technique and up to 85% reduction in network communication overhead.

[9]https://software.intel.com/en-us/articles/intel-mpi-benchmarks
[10]http://www.astro.umn.edu/groups/compastro/?q=node/1

### 2.6.5 Net-cohort: Detecting and managing vm ensembles in virtualized data centers

Liting Hu et al. [29] presents 'Net-Cohort', a lightweight system which continuously monitors a system to identify potential virtual machine ensembles, evaluates the degree of communication (or as the paper calls it 'chattiness') among the VMs in the potential ensembles and enables optimized VM placement to reduce the stress on bi-section bandwidth of the data center network. Net-Cohort uses commonly available VM-level statistics in order to create VM subsets (or ensembles) using correlation values and a hierarchical clustering algorithm. In the second step a statistical packet sniffer is used in order to identify VMs as members of a misplaced ensemble using the statistical algorithm proposed by Golab and De Haan in [23] and to finally make new VM placement decisions. The experiment was conducted on 15 Xen-based hosts and 225 VMs. Net-Cohort showed the ability to detect VM ensembles at low cost with about 90% accuracy. The experiment results showed that the new VM placement improved application throughput with 385% for a RUBiS instance, while application throughput for a Hadoop instance improved with 56.4%. The quality of service (QoS) for a SIPp instance showed 12.76 times improvement.

### 2.6.6 Cicada: Introducing Predictive Guarantees for Cloud Networks

LaCurts et al. [41] introduce predictive guarantees, a new abstraction for bandwidth guarantees in cloud networks, which is achieved by analyzing traffic traces gathered over six months from an HP Cloud Services data center and developing a prediction algorithm which is used by the cloud provider in order to suggest appropriate bandwidth guarantees to the tenants. Cicada's prediction algorithm adapts Herbster and Warmuth's "tracking the best expert" idea [28]. In order to predict traffic the paper uses all previously observed traffic matrices. This method doesn't require extensive amount of data in order to make predictions. For VM placement a two-stage "virtual oversubscribed cluster" (VOC) algorithm introduced in Ballani et al. [2] is used. The algorithm is designed to place clusters on the smallest subtree. Cicada's greedy algorithm tries to place the VM pairs with most intercommunication on the highest-bandwidth paths, typically

on the same rack, in the same subtree. Cicada's performance is compared to VOC algorithm's results on a simulated physical infrastructure with 71 racks with 16 servers each. Results show that Cicada's placement algorithm leaves more inter-rack bandwidth available.

### 2.6.7 Application-Driven Bandwidth Guarantees in Datacenters

Lee et al. [42] introduces CloudMirror, a solution that provides bandwidth guarantees to cloud applications by deriving a network abstraction based on application communication structure, called Tenant Application Graph or TAG. CloudMirror provides a new workload placement algorithm that meets bandwidth requirements by TAGs while taking in account high availability considerations. TAG model is introduced as a graph, where each vertex represents an application component or a tier, set of VMs performing the same function. A tenant can simply map each tier onto a TAG vertex. For example web, business logic and database tiers. Users can either specify a matching TAG model and tune the bandwidth guarantees by themselves or cloud orchestration systems like OpenStack Heat or AWS CloudFormation could be extended to generate TAG models. The simulation environment is written in Python and both CloudMirror placement algorithm (CM) efficiency and accepting more tenant requests compared to other models is evaluated in it. The results showed that CloudMirror outperforms the performance of the existing solutions. CloudMirror was able to handle 40% more bandwidth demand compared to the Oktopus [2] system and also improved high availability from 20% to 70%.

### 2.6.8 VMPlanner: Optimizing virtual machine placement and traffic flow routing to reduce network power costs in cloud data centers

The main focus in Fang et al. [19] is to consolidate VMs in a way that allows a number of network elements to become redundant and be removed or put in a power-saving state. The paper proposes VMPlanner, a novel approach for network power reduction in cloud-based data centers. VMPlanner tries to manage not only VM placement but traffic flow routing as well by implementing three approximation algorithms: traffic-aware VM

grouping algorithm, distance-aware VM-group to server-rack mapping algorithm and power-aware inter-VM traffic flow routing algorithm. The VMPlanner system for data centers consists of three modules: analyzer, optimizer and controller and can be implemented as a NOX application [26] to run atop a network of OpenFlow switches. The performance of VMPlanner is evaluated on a simulator developed in C++ using simulation parameters and traffic conditions from real cases from a private data center test-bed [15]. The experiment was conducted with 2000 VMs. The results were very preliminary but at the same time succeeded in demonstrating the potential of reducing power usage by consolidating VMs in a traffic-aware way and intelligently routing the traffic.

### 2.6.9 Tools for implementation

Several tools essential to the implementation of this project will be used during the research.

#### Python scripts

Python [11] is powerful and widely used high-level programming language. The idea of Python was conceived in 1989 by Guido van Rossum at CWI in the Netherlands and the version 1.0 was released in January 1994.

Python supports object-oriented, imperative and functional programming paradigms and provides a comprehensive built-in library as well as numerous useful add-on libraries for different purposes. Python is known for its user friendly and easily readable code and the ability to resolve complex problems using very few lines.

#### Pseudocode

Pseudocode is an informal description of an operating unit, a computer program or an algorithm in a high-level fashion. Pseudocode doesn't use a specific programming language syntax, instead it is written in a way which is easier understandable for a reader regardless the technical background and programming skills or absence of it. Pseudocode usually uses the

[11] https://www.python.org/

structure of a typical programming language, however it is not intended to be read by a machine, but rather by humans.

Below is an example of how pseudocode can be written:

**Algorithm 1:** An example pseudocode

```
Result: The result of the algorithm
preprocessing;
while While condition do
    instructions;
    if condition then
        instruction 1;
        instruction 2;
    else
        instruction 3;
    end
end
```

# Part II

# The project

# Chapter 3

# Approach

The approach chapter describes what actions will be taken and what methods will be used in order to address the problem statement defined earlier. Experiment design, tools and methods used in the experiment and methods for evaluation of the experiment results will also be described in this chapter.

## 3.1 Objectives

As described in the problem statement (see section 1.1) the main objectives of this project are to investigate how graph partitioning can be used in order to consolidate VMs in an intelligent traffic-aware way and how a quadratic assignment algorithm can be implemented in order to assign the partitions to the available racks in the datacenter to further optimize the internal bandwidth usage.

In order to achieve the above objectives several steps must be taken:

1. **Experiment design** should be planned in order to prepare testbeds for the experiments.
   (a) **Data center models** should be designed.
   (b) **Corresponding cost matrices** should be calculated.
2. **Virtual machine communication data** should be collected, parsed and stored for the use in the experiment.

3. **VM clustering algorithm** should be designed and implemented in order to consolidate VMs with highest mutual traffic.

4. **Quadratic assignment algorithm** should be designed and applied to the VM clusters obtained in the previous step in order to place them on the available server racks in the data center model in the most optimal way.

5. **Experiments** should be conducted in order to measure the effect of the implemented VM clustering and quadratic assignment algorithms.

6. **Evaluation** of the impact of the VM consolidation through clustering and quadratic assignment on the internal bandwidth usage should be conducted with the use of the results obtained during the experiments.

## 3.2 Experiment design

In order to address the problem statement an effective experiment design should be planned and implemented. It's essential that the models used in the experiment, the assumptions taken and the methods utilized ensure that the experiments mimic the real world scenarios as closely as possible. Naturally, the resources available for the research might in some cases impose various limits to the experiments. It's important to be aware of these limitations and what it means for the results of the experiments. Three different data center models will be simulated in this thesis instead of testing on live data centers. This approach has its advantages as well as the disadvantages which will be discussed later in the thesis.

Three identical experiments will be conducted on three DCN models with the help of scripts written in Python. During the three experiments baseline costs of communication will be computed based on the randomly scattered VMs. The proposed graph partitioning and quadratic placement algorithms will alter the locations of the VMs with the purpose of optimizing the communication between them and the resulting costs of communication will be calculated and stored for later comparison and analysis.

A fixed number of IP addresses will be randomly picked from the traffic information in order to simulate VM traffic for the experiments. The chosen IP addresses will be assumed to represent VMs. The VMs will be divided into fixed number of clusters of equal size. These clusters will be optimized during the experiment by migrating VMs in and out of them altering the cluster populations, however the number of VMs in each cluster will remain constant before and after each experiment. It will be assumed that each VM has identical specifications with regards to CPU, memory, disk, etc. It will be assumed that the link capacity is the same for all the links in the whole data center for all the models used in the experiments.

The first set of the three experiments will be conducted based on the fixed number of VMs randomly chosen from the obtained traffic trace. The second set of the three experiments will be conducted with exactly the same conditions with one difference: this time the fixed number of VMs to be optimized will be chosen manually in order to test the effectiveness of the proposed algorithms on a different set of VMs with different traffic patterns.

### 3.2.1 VM communication data

The communication data used in this project is obtained via third party source which made the data available for the public use. Three actual data center traffic traces are published on the Computer Sciences User Pages of the University of Wisconsin-Madison[1] by the assistant professor Theophilus Benson of the Duke University. The data sets are dated from 2009 and represent three different university data centers studied in 2010 paper titled Network Traffic Characteristics of Data Centers in the Wild [4].

UNI1 data center traces are chosen for the data center traffic simulation in this work. The traffic traces are originally stored in the binary packet capture (PCAP) files. Roughly one hour of traffic data is stored in 20 PACP files. The start timestamp of the data used in this project is *2009-12-17 17:26:04* and the end timestamp is *2009-12-17 18:31:19*.

[1] http://pages.cs.wisc.edu/ tbenson/IMC10_Data.html

The important assumption is that the chosen traces, even though they represent a short period of time, reflect the traffic behavior of an average data center over longer periods of time and can be generalized for other data centers as well.

For the use in the experiments in the project the binary PCAP files will be converted to a human-readable format with the *tcpdump* tool as shown in the example below:

tcpdump -ttttnnr univ1_pt1.pcap

```
2009-12-17 17:26:04.398500 IP 41.177.117.184.1618 >
41.177.3.224.51332: Flags [P.], seq 354231048:354231386,
ack 3814681859, win 65535, length 338

2009-12-17 17:26:04.398601 IP 90.218.72.95.10749 >
244.3.160.239.80: Flags [P.], seq 1479609190:1479610159,
ack 3766710729, win 17520, length 969

2009-12-17 17:26:04.398810 IP 244.3.160.239.80 >
90.218.72.95.10749: Flags [.], ack 969, win 24820,
length 0

2009-12-17 17:26:04.398879 IP 41.177.3.224.51332 >
41.177.117.184.1618: Flags [P.], seq 1:611, ack 338,
win 65535, length 610
```

The results will be stored in output text files in plain text format for later access. The Python script will parse the text files and using regular expressions line by line search for the IP addresses and the amount of traffic exchanged between them. The matched results will be sorted in a comma-separated format in a table with three columns: *source_ip, destination_ip, bytes_transmitted*. This list of communicating pairs will include both transmissions from IP address A to IP address B and vice versa. The output data will be stored in the `traffic_rates_list.txt` file which will be accessed later on in the experiment.

Example of the contents of the traffic_rates_list.txt file:

Example of contents of traffic_rates.txt file

```
41.177.67.75,244.3.41.84,15
```

```
41.177.26.15,68.159.161.47,1307818
244.3.160.248,41.177.26.141,37350
41.177.26.46,111.98.75.211,405272
194.66.108.214,244.3.41.84,1262
41.177.247.145,244.3.160.239,21877
244.3.160.239,215.224.91.90,8260038
168.22.108.153,244.3.210.197,702
```

The IP addresses will be extracted from `traffic_rates_list.txt` file and stored separately in the `all_ips.txt` file by a python script in order to be used later to populate a traffic matrix and also to be able to be easily referenced later. Each IP address will automatically be assigned a unique and a constant *ID* at this stage. The list index will determine the ID. The first IP address 41.177.67.75 for example will get an $ID = 0$, 244.3.41.84 will get an $ID = 1$, etc.

Example of contents of all_ips.txt file

```
41.177.67.75
244.3.41.84
41.177.26.15
68.159.161.47
244.3.160.248
41.177.26.141
41.177.26.46
111.98.75.211
```

For this work two sets of 1600 IP addresses will be chosen. The first set of 1600 VMs will be chosen on a random basis while the other set of 1600 VMs will be chosen in a more controlled way. Both sets will be used to conduct the identical experiments. Each IP address is assumed to correspond to a single virtual machine. This will allow the 16 clusters to be created with 100 VMs in each cluster. Each cluster then will be assigned to a single server rack so that the total cost of communication for the whole data center model can be calculated.

It is also assumed for this experiment that each VM has identical configuration in terms of the number of CPUs, memory and disk capacity and other specifications. Each data center model will contain 16 server racks

and it is assumed that each server rack is able to host 100 VMs.

### 3.2.2 VM traffic matrix

Communication rates between all the virtual machines should be made easily available as frequent calculations will be done throughout the experiments when detecting VM clusters, determining the amount of traffic flow between them or searching for the best possible placement. It would be preferable to conduct experiments on a data collected over a long period of time from real, modern data center with full overview of the hardware and software. However in this project a third party data will be used. One of the best ways to store and access this kind of information is to use a two dimensional matrix.

A matrix will be constructed with the use of the built-in array functionality of Python and the library `pickle` will be used in order to store and access the VM traffic matrix in a binary format later during the experiments.

When the list of communicating VM pairs and corresponding traffic rates is built and each VM has been assigned a unique identifier VM traffic matrix with 1600 rows and 1600 columns will be created and populated with the values corresponding to the traffic rates. The matrix should be symmetric and contain 2560000 elements or edges. The matrix values diagonally where matrix row is equal to matrix column will be equal to 0. First an asymmetric VM traffic matrix $A$ is created by iterating through all the IP addresses and updating the corresponding matrix elements (by row and column) with the edge values. An example of an asymmetric matrix (see Fig.3.1) is displayed below:

$$A_{ij} = \begin{bmatrix} 0 & 1324980.0 & 31102812.0 & \cdots & 0 \\ 62291730.0 & 0 & 0 & \cdots & 0 \\ 639871.0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 252940.0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix} \tag{3.1}$$

As mentioned before each row and column ID corresponds to a unique VM documented with an IP address in the `all_ips.txt` file. Thus the value 1324980.0 in the asymmetric traffic matrix A (see Fig. 3.1) corresponds to the edge between $V_0$ and $V_1$ ($V_0$ refers to the VM with ID 0; $V_1$ refers to the VM with ID 1) where $V_0$ is the transmitter and $V_1$ is the receiver and $A_{01}$ is the edge between them in the asymmetric VM traffic matrix $A$. Similarly 62291730.0 is the edge between $V_1$ and $V_0$ where $V_1$ is the transmitter and $V_0$ is the receiving end and $A_{10}$ is the edge between them in the asymmetric VM traffic matrix $A$.

After obtaining the asymmetric VM traffic matrix a new symmetric VM traffic matrix $D$ will be built based on the asymmetric one. The script will iterate through the rows and the columns of the matrix and whenever an element $A_ij$ has a value greater than 0 the value of $A_ji$ will be added to $A_ij$ and the sum will be divided by 2. Both $D_ij$ and $D_ji$ will be updated with the new value. Thus a new symmetric VM traffic matrix $D$ will be created. An example of such a matrix (see Fig. 3.2) is displayed below:

$$D_{ij} = \begin{bmatrix} 0 & 31808355.0 & 31742683.0 & \cdots & 0 \\ 31808355.0 & 0 & 0 & \cdots & 0 \\ 31742683.0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 252940.0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix} \tag{3.2}$$

In the symmetric VM traffic matrix D above (see Fig. 3.2) the value 31808355.0 for the edge between $V_0$ and $V_1$ (where $V_0$ refers to the VM with ID 0 and $V_1$ refers to the VM with ID 1) is derived from the asymmetric VM traffic matrix $A$ by adding the edge between $V_0$ and $V_1$ to the edge between $V_1$ and $V_0$ and dividing the sum by 2. Hence $D_{01} = D_{10} = 31808355.0 = \frac{A_{01}+A_{10}}{2}$. Therefore in the symmetric VM traffic matrix edge between $VM_0$ and $VM_1$ is equal to the edge between $V_1$ and $V_0$ and can be denoted as edge $D_{01}$ or $D_{10}$.

### 3.2.3 Data center models

Since no live data center is available for this research data center models will be implemented. The purpose of a data center model is to simulate a network architecture used by the virtual machines in order to communicate with each other and to provide the basis for the calculation of cost of communication. Simulating the network elements and the links the traffic from one VM has to travel to reach its destination VM is necessary in order to compute the communication cost between VMs in any given virtual machine pair and to ultimately calculate the total cost of communication between all the existing communicating VM pairs in the data center for the given period of time.

There are several different data center network architectures (DCN) in use in the world today as discussed in the background section [14, 24, 47, 50]. In this project three of the data center network architectures will be simulated in order to test the impact of the proposed algorithms.

The three DCN architectures chosen for this project are: the legacy Tree data center network architecture, the Fat-tree and the VL2 data center network architecture.

### 3.2.4 Cost matrices

One of the main methods for simulating a data center network will be calculation of the associated cost matrix. Each row and the column with the corresponding index will be associated with a single server rack in the data center. The matrix with 16 rows and 16 columns will contain 256 elements

where each element will correspond to the cost associated with the communication between two server racks.

The cost of communication between two nodes can be determined by the link speed between the nodes or by the number of network elements (switches or routers) the packets have to travel through (also referred to as number of hops) on their way to the destination. In this thesis number of hops will be used to determine the communication cost. For example if a data packet from server rack number 1 ($R_1$) has to travel through one switch before it reaches the destination rack number 2 ($R_2$) the cost of communication between $R_1$ and $R_2$ will be 1 and the corresponding edge will be found in the cost matrix in row 1 and column 1.

The cost matrix will be calculated by picking every server rack one by one and comparing its communication cost with the rest of the server racks one by one while evaluating how many hops a data packet has to go through on its way from one rack to another. In this way all the possible permutations will be taken in account and the result of the calculation will be a two dimensional symmetric matrix.

## 3.3 Proposed VM clustering algorithm

The proposed VM clustering algorithm will be based on Oommen's Graph Partitioning Using Learning Automata (GPLA) [51] algorithm with several minor adjustments and tweaks. GPLA attempts to solve the Graph Partitioning Problem (GPP) [9, 17, 33] by using stochastic learning automata (LA) which is designed to learn the optimal action offered by a random environment. Learning is achieved by interacting with the environment as it constantly changes and by processing the response of the environment to the actions taken. Since this thesis is dealing with a GPP where all the subpartitions are of equal size the problem is referred to as equi-partitionig problem (EPP). The best solution to EPP is Object Migrating Automaton (OMA) proposed by Oommen and Ma [52]. This technique will be adapted for the GPP and used in the proposed VM clustering algorithm.

The algorithm adapted for this project will read the set of 1600 nodes

or vertices distributed over 16 sub-partitions, also referred to as groups or arms, and output the final solution of the graph partitioning problem. This will be achieved by adopting the object migration automata (OMA) used in Oommen's algorithm. The randomly picked pairs of vertices will be checked by the algorithm in order to find out whether or not they're significantly connected and then they will be either rewarded or penalized depending on what conditions they satisfy.

In order to determine whether the nodes are significantly connected two important thresholds, *similarity threshold* and *dissimilarity threshold* will be calculated by the following formulae:

$$SimilarityThreshold = (1 + \rho) * MeanEdge \tag{3.3}$$

$$SimilarityThreshold = (1 - \rho) * MeanEdge \tag{3.4}$$

$\rho$ will be set to the fixed value of 0.25 and the *MeanEdge* value will be calculated by computing the average edge value based on all the nonzero elements (or edges between the nodes) of the symmetric VM traffic matrix $D$.

When two random vertices $V_i$ and $V_j$ are picked and their corresponding edge $D_{ij}$ is higher than the *SimilarityThreshold* the two nodes will be regarded as *similar*. If the nodes are found to be in the distinct sub-partitions they will be penalized since this state is unfavorable. If the nodes are found in the same sub-partitioned they will be rewarded since this scenario is favorable. The penalize action will move the nodes closer to the *MinimumCertainty* state towards the outer boundary of the sub-partition while the reward action will push the nodes deeper into their sub-partitions towards the *MaximumCertainty* state. When the nodes reach the outer boundaries of their sub-partitions they might be expelled from their current sub-partitions and moved to a better one. This process will be repeated until the maximum number of iterations is reached.

## 3.4 Assigning VM clusters to server racks

Assigning 16 VM clusters obtained after VM clustering procedure to 16 available server racks is the next optimization problem in this thesis. As some of the VMs in the clusters are expected to communicate with other VMs in other clusters the intercluster traffic is expected to play a significant role in the total cost of communication. The intercluster traffic is expected to decrease after the VM clustering algorithm partitions the clusters consolidating highly talkative VMs in the same clusters. However, it is assumed that the optimal assignment of the clusters to the server racks will further decrease the total cost of communication. Assigning VM clusters with high mutual traffic to server racks close to each other will be beneficial for the data center traffic optimization as less traffic will flow over the costly links.

### VM cluster communication matrix

In order to be able to quickly access the connection rates between the previously established 16 clusters an associated symmetric VM cluster traffic matrix $S$ (see Fig. 3.5) will be constructed consisting of 16 rows and 16 columns. The 256 elements of the VM cluster traffic matrix $S$ will represent the 256 edges between the partitioned VM clusters. In order to calculate the edges between two VM clusters traffic rates between every element of cluster 1 and every element of cluster 2 will be found in the associated symmetric VM traffic matrix $D$ and the sum of the values will determine the edge between the two clusters.

$$S_{ij} = \begin{bmatrix} 0 & 11107855.5 & 23063975.0 & \cdots & 10806454.0 \\ 11107855.5 & 0 & 224874.0 & \cdots & 945003.0 \\ 23063975.0 & 224874.0 & 0 & \cdots & 1933362.0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 31876358.0 & 1800978.0 & 389940.5 & \cdots & 2827934.0 \\ 28356699.5 & 10705004.5 & 11557148.0 & \cdots & 5128501.0 \\ 27724267.0 & 38285.0 & 780987.0 & \cdots & 245885.0 \\ 10806454.0 & 945003.0 & 1933362.0 & \cdots & 0 \end{bmatrix} \tag{3.5}$$

## 3.5 Proposed cluster placement algorithm

Since the main objective is to assign the VM clusters to the server racks in a way which decreases the total cost of communication a cluster placement algorithm will be designed to handle this task. The assignment problem will be treated as a Quadratic Assignment Problem (QAP) [32, 39, 47, 57] which is known to be one of the most difficult combinatorial optimization problems in the NP-hard class. The assignment of the 16 clusters to the available 16 server racks that gives the lowest total communication cost will be considered the best assignment. In order to find such an assignment an algorithm will be implemented by the use of the Python scripting language. The task of the cluster placement algorithm will be to conduct a search of the best assignment in the possible solution space. Since the solution space for 16 groups is an astronomical number of 16! the exhaustive search approach in order to find the best solution is not computationally feasible. Instead the most optimal solution has to be found in a specific pool of solutions. In order to find such an optimal solution to QAP in this project simulated annealing (SA) [13, 37] technique will be used. Simulated annealing makes sure that the algorithm doesn't get trapped in a local minimum and that it'll be given a chance to explore wider range of possible solutions by visiting even the inferior solutions with certain constantly decreasing probability [18].

#### Setting initial cluster placement

The cluster placement algorithm will read the set of nodes previously partitioned by the VM clustering algorithm and the VM cluster traffic matrix $S$ in order to check all the possible cluster pairs and sort them by the corresponding edge values $S_{ij}$ in the descending order after which the total cost of communication will be calculated using the VM cluster traffic matrix $S$ and the communication cost matrix $C$. The result of this step will be set as the initial and the current best state of the VM clusters. The initial placement will be an already improved placement compared to the randomly aligned VM clusters and this is expected to help the cluster placement algorithm to find an even more optimal solution.

The total cost of communication will be calculated by summing all the

edges multiplied by their corresponding communication costs using the following formula:

$$\sum_{i,j=\cdots,n} D_{ij} \cdot C_{\pi(i)\pi(j)} \tag{3.6}$$

Where $D_{ij}$ denotes a traffic rate between nodes $V_i$ and $V_j$ and $C_{\pi(i)\pi(j)}$ denotes the cost of communication between the server racks the nodes $V_i$ and $V_j$ are assigned to.

**Simulated annealing process**

When the initial placement is established and the initial total cost of communication is calculated the algorithm will start executing the $N$ number of iterations by starting at a predefined value $T$ (temperature) and decreasing the temperature gradually. During each iteration two distinct clusters will be chosen and they will swap with places.

After each swap the total cost of communication will be calculated and the new state will be stored temporarily. If the new state yields total cost of communication which is superior to the previous (or the initial) total cost of communication the algorithm will set is as the current best state. If the new state is inferior to the previous state the algorithm will move to it with a certain probability P. This probability will be calculated with the following formula:

$$P = e^{-\frac{\Delta}{T}} \tag{3.7}$$

Where $\Delta = TotalCost_{new} - TotalCost_{old}$, the difference between the total communication cost yielded by the new state and the total communication cost of the old state, and $T$ is the temperature.

This process (see Fig.3.1) will ensure that the algorithm won't get stuck in the local minimum and falsely assume that the optimal result has been obtained. In the beginning the probability $P$ will have a higher value

meaning that the algorithm will accept inferior results more frequently. However as the temperature $T$ decreases over time the $P$ value will gradually decrease and the algorithm will be less and less likely to accept inferior results. The simulated annealing technique will give the cluster placement algorithm possibility to explore wider range of the possible solutions space. In the end the most optimal solution will be chosen.

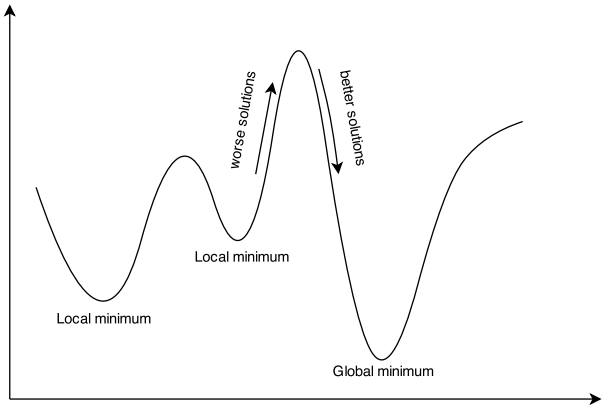


Figure 3.1: Simulated annealing process

## 3.6 Experiment set A

In this experiment set three experiments will be conducted on each of the simulated data center networking architecture. A separate experiment will be conducted in order to observe changes in the intracluster and the intercluster traffic caused by the VM clustering algorithm with the use of graph partitioning. In the experiment set A the set of 1600 VMs chosen randomly from the collected traffic trace will be used. It is expected that this set of 1600 VMs will contain several VMs who have rather high mutual traffic while most of the VMs communicate with each other at a significantly lower rate.

### 3.6.1 Experiment a1: Tree DCN

First the tests will be run on the most widely used legacy three-tier Tree DCN model. The Tree DCN model (see Fig.3.2) will contain 16 server racks. Each server rack is assumed to be able to accommodate 100 VMs. The server racks will form four groups, where each group will consist of four server racks connected to a single access layer (or layer 1) switch. The

four access switches will be connected to the layer 2 - the aggregation layer switches. The aggregation layer will consist of four switches. However only two of the four switches will be presumed active, while the other two will be in standby mode. Finally there will be one active and one standby switch on the core layer at the top level of the data center network.

Some of the important results obtained through the experiment will be abbreviated. The average total cost of communication with the randomly placed VMs for the Tree DCN will be noted as $T_{RandTreeA}$, the average total communication cost after the optimization with the VM clustering algorithm for the Tree DCN will be shortened as $T_{GpTreeA}$ and the average total communication cost after executing the cluster placement algorithm will be noted as $T_{QapTreeA}$. "A" in these notations refers the experiment set A.

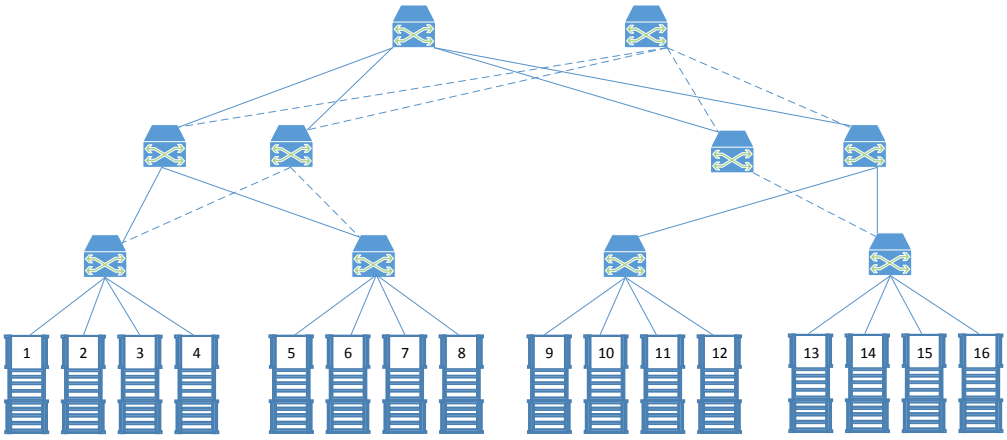


Figure 3.2: The Tree data center network model used in the project

The simulated model of a three-tier data center networking architecture (see Fig. 3.2) allows for the cost matrix to be constructed for later use in the calculations of the cost of communication between the VMs placed in the specific server racks.

### Cost matrix for the Tree DCN

The following communication cost matrix (see Fig. 3.8) will be used for the Tree DCN in this project:

$$C_{ij} = \begin{bmatrix} 0 & 1 & 1 & 1 & 3 & 3 & 3 & 3 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 1 & 0 & 1 & 1 & 3 & 3 & 3 & 3 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 1 & 1 & 0 & 1 & 3 & 3 & 3 & 3 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 1 & 1 & 1 & 0 & 3 & 3 & 3 & 3 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 3 & 3 & 3 & 3 & 0 & 1 & 1 & 1 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 3 & 3 & 3 & 3 & 1 & 0 & 1 & 1 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 3 & 3 & 3 & 3 & 1 & 1 & 0 & 1 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 3 & 3 & 3 & 3 & 1 & 1 & 1 & 0 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 0 & 1 & 1 & 1 & 3 & 3 & 3 & 3 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 1 & 0 & 1 & 1 & 3 & 3 & 3 & 3 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 1 & 1 & 0 & 1 & 3 & 3 & 3 & 3 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 1 & 1 & 1 & 0 & 3 & 3 & 3 & 3 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 3 & 3 & 3 & 3 & 0 & 1 & 1 & 1 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 3 & 3 & 3 & 3 & 1 & 0 & 1 & 1 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 3 & 3 & 3 & 3 & 1 & 1 & 0 & 1 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 3 & 3 & 3 & 3 & 1 & 1 & 1 & 0 \end{bmatrix} \tag{3.8}$$

Each row and each column in the cost matrix corresponds to a single server rack. For example row 1 (the first row of the matrix) and column 1 (the first column of the matrix) correspond to the rack number 1 (see Fig.3.2), whereas row 16 (the last row of the matrix) and the column 16 (the last column of the matrix) correspond to the rack number 16. Thus the communication cost for the traffic between server rack number 1 and 10 can be found in the row number 1 and the column number 10 of the communication cost matrix and equals to 5.

### 3.6.2 Experiment a2: Fat-tree DCN

The next experiment will be conducted on a relatively recently proposed data center network architecture PortLand [50] which is based on a so called Fat-tree network topology. In the Fat-tree DCN model (see Fig.3.3) four pods will be formed out of 16 switches. Each pod will contain 4 switches and will be connected to all the available 4 core switches. The

traffic between 1600 VMs will be simulated in the Fat-tree model. The 1600 VMs will be divided in 16 sub-partitions of equal sizes containing 100 VMs each. Each sub-partition will be assigned to one of the 16 server racks.

The average total cost of communication with the randomly placed VMs for the Fat-tree DCN will be noted as $T_{RandFtreeA}$, the average total communication cost after the optimization with the VM clustering algorithm for the Fat-tree DCN will be shortened as $T_{GpFtreeA}$ and the average total communication cost after executing the cluster placement algorithm will be noted as $T_{QapFtreeA}$ where "A" in the abbreviation refers to the experiment set A.

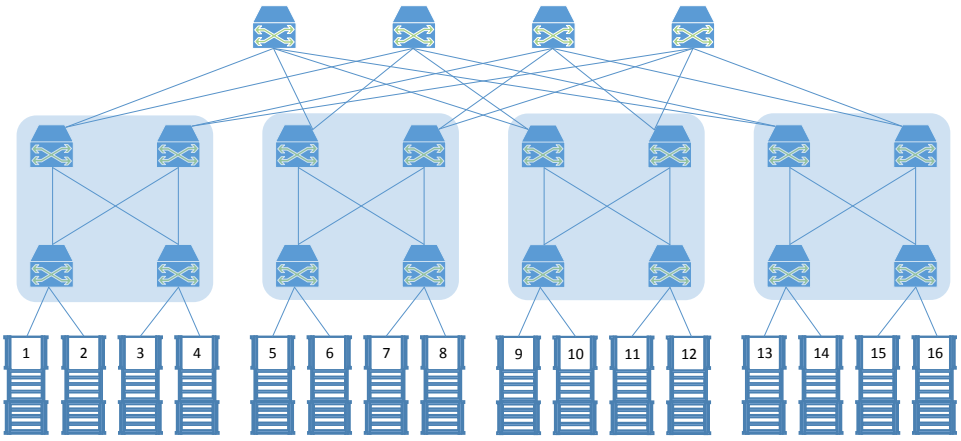


Figure 3.3: The Fat-tree data center network model used in the project

### Cost matrix for the Fat-tree DCN

The cost of communication between the neighbor pairs of racks $R_1$ and $R_2$ will be 1. However the cost across the neighboring pairs, for example between $R_2$ and $R_3$, in the same pod will be 3, whereas the cost of communication across the pods will be 5. A cost matrix $C$ will be calculated for the experiment with Fat-tree DCN based on the number of the network elements (switches) the traffic has to travel through in order to reach its destination from one server rack to another. The cost matrix which will be used for the experiment is displayed in fig.3.9 below:

$$C_{ij} = \begin{bmatrix}
0 & 1 & 3 & 3 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\
1 & 0 & 3 & 3 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\
3 & 3 & 0 & 1 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\
3 & 3 & 1 & 0 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\
5 & 5 & 5 & 5 & 0 & 1 & 3 & 3 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\
5 & 5 & 5 & 5 & 1 & 0 & 3 & 3 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\
5 & 5 & 5 & 5 & 3 & 3 & 0 & 1 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\
5 & 5 & 5 & 5 & 3 & 3 & 1 & 0 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\
5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 0 & 1 & 3 & 3 & 5 & 5 & 5 & 5 \\
5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 1 & 0 & 3 & 3 & 5 & 5 & 5 & 5 \\
5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 3 & 3 & 0 & 1 & 5 & 5 & 5 & 5 \\
5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 3 & 3 & 1 & 0 & 5 & 5 & 5 & 5 \\
5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 0 & 1 & 3 & 3 \\
5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 1 & 0 & 3 & 3 \\
5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 3 & 3 & 0 & 1 \\
5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 3 & 3 & 1 & 0
\end{bmatrix} \tag{3.9}$$

### 3.6.3 Experiment a3: VL2 DCN

The third and the last experiment will be run on the simulated VL2 [24] data center network architecture. VL2 is a newly proposed DCN and it shares many similarities with the traditional Tree DCN. However, the main difference between VL2 (see Fig.3.4) and tree with regards to the cost of communication is that the traffic in VL2 is forwarded all the way to the core layer before it's routed back to the access layer to its destination. This difference will increase the cost of communication between the neighboring access layer switches and respectively between the groups of the server racks associated with the given access switches.

The VL2 model will consist of 12 switches and 16 server racks. The racks will form four groups each consisting of 4 racks. Each group will be connected to a single access layer switch. 1600 VMs will be accommodated by the VL2 model. The VMs will be divided into 16 groups of 100 VMs each. Each server rack will be assumed to be able to host 100 VMs.

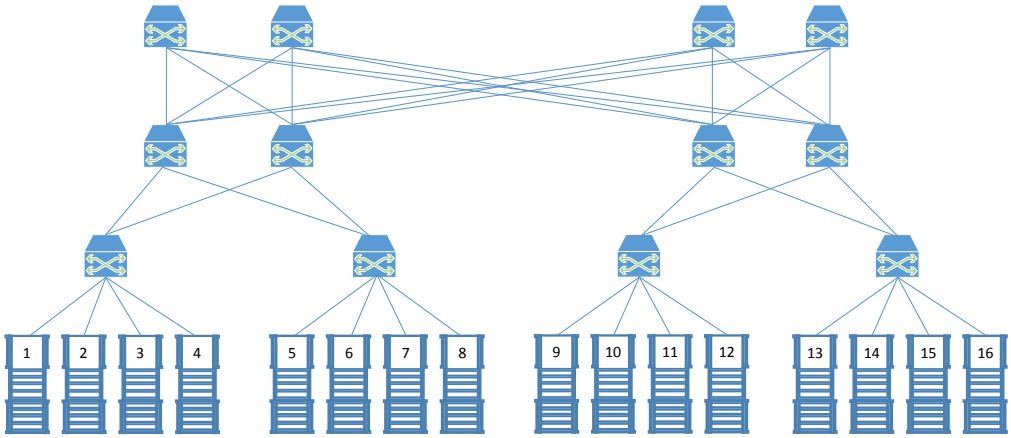


Figure 3.4: The VL2 data center network model used in the project

The average total cost of communication with the randomly placed VMs for the VL2 DCN will be noted as $T_{RandVl2A}$, the average total communication cost after the optimization with the VM clustering algorithm for the VL2 DCN will be shortened as $T_{GpVl2A}$ and the average total communication cost after executing the cluster placement algorithm will be noted as $T_{QapVl2A}$. The "A" in the abbreviations refers to the experiment set A.

### Cost matrix for the VL2 DCN

The resulting cost matrix for VL2 (see Fig.3.10) will be similar to the cost matrix for the Tree reflecting the similarities and the differences between the two network topologies. The matrix clearly describes the relatively higher cost compared to the previous data center models associated with the communication across the rack groups belonging to the different access

layer switches.

$$C_{ij} = \begin{bmatrix} 0 & 1 & 1 & 1 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 1 & 0 & 1 & 1 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 1 & 1 & 0 & 1 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 1 & 1 & 1 & 0 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 0 & 1 & 1 & 1 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 1 & 0 & 1 & 1 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 1 & 1 & 0 & 1 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 1 & 1 & 1 & 0 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 0 & 1 & 1 & 1 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 1 & 0 & 1 & 1 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 1 & 1 & 0 & 1 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 1 & 1 & 1 & 0 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 0 & 1 & 1 & 1 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 1 & 0 & 1 & 1 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 1 & 1 & 0 & 1 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 1 & 1 & 1 & 0 \end{bmatrix} \tag{3.10}$$

### 3.6.4 Intracluster traffic experiment: set A

The purpose of this experiment will be to observe the effects of the graph partitioning algorithm on the intracluster and intercluster communications. In order to achieve this goal the available 1600 nodes will be placed in random states in the 16 clusters. After this step the intracluster and intercluster traffic rates will be calculated. This will be done multiple times in order to collect reliable data. When the baseline values are calculated the graph partitioning algorithm will optimize the VM placement and the intracluster and intercluster traffic rates will be calculated once again. The optimization will take place multiple times in order to obtain reliable average values. The results will be stored to be plotted and analyzed later.

## 3.7 Experiment set B

The experiment set B will consist of the following experiments:

1. Experiment b1: Tree DCN

2. Experiment b2: Fat-tree DCN

3. Experiment b3: VL2 DCN

The experiment set B will repeat the experiments described in the previous sections under *Experiment set A*. The difference will be the set of 1600 VMs. For the experiment set B different VMs will be chosen in order to better observe the effects of the graph partitioning and quadratic assignment algorithms. After the three experiments an intercluster traffic experiment for the set B will be conducted.

The purpose of repeating these three experiments and the intracluster experiment is to observe how the optimization algorithms developed in this project behave with a different set of VMs with a different traffic pattern. The new set used in the experiment set B will be deliberately chosen to be IP addresses who talk to each other with a relatively more even traffic rates compared to the VM set used in the experiment set A. This will be achieved by sorting the `traffic_rates_list.txt` file by the traffic rates in the descending order and then removing the communicating pairs with significantly high traffic rates. This will smoothe the graph of the distribution of the edges between the VMs participating in the experiments.

Three experiments experiment b1 on Tree, experiment b2 on Fat-tree and the experiment b3 on VL2 data center networking architectures will be conducted with the exact same parameters as in the experiment set A. After this the intracluster trafic experiment will be conducted also with the same parameters as in the experiment set A. The total cost of communication notations will be similar to the ones used in the experiment set A. The only difference will be the letter "B" in the end. For example total cost of communication with randomly assigned VMs in the Tree experiment for the experiment set B (experiment b1) will be denoted as $T_{RandTreeB}$.

## 3.8 Measurement and Evaluation

In order to be able to assess how the problem statement has been addressed it is important to reliably evaluate the performance of the proposed VM clustering and cluster placement algorithms and their impact on the total

cost of the communication in the data center models. Since the experiments in this thesis are conducted on virtual models of data center networking architectures there is no possibility of connecting to physical devices and directly measuring real-time bandwidth usage on real links prior and after the optimization. Instead other methods of measurement and evaluation will be implemented. It's expected that the total cost of communication for the whole data center will decrease after both VM clustering and cluster placement algorithms are implemented. It is also expected that the average intracluster (the traffic between the member VMs inside a group) traffic will increase after VM clustering algorithm as the VMs with high mutual traffic are grouped in the same clusters. While the intercluster traffic decreases as the result of the optimization.

In order to compare initial and optimized states of the system first a baseline will be established by randomly assigning VMs to the clusters and then computing and storing both intracluster and intercluster traffic data and the total cost of communication for the whole system. There is however a probability of obtaining extreme results by chance since the highly communicating VM pairs can by chance end up in the same clusters hence yielding relatively low total cost of communication already in the baseline placement. In this case the clustering and assignment algorithms might not seem to have significant enough effect. In order to avoid this scenario and obtain a baseline which can be considered as a reliable average unoptimized system multiple ($N = 35$) randomly distributed VM states will be generated. Since at least 30 test samples is usually required in order to obtain reliable statistics and due to the time constraints of the project the test sample size will be set to 35. The average values for the total cost of communication will be calculated and stored. Average intracluster and intercluster traffic will also be computed and stored for later analysis. This approach is expected to reduce the chance of random error distorting the results. In order to reduce the variation in the mean values and obtain as reliable data as possible the tests will be repeated 35 times and the obtained 35 results will be used in order to calculate statistical values.

In order to establish a baseline total cost of communication 35 randomly distributed VM placements will be generated. The VM clustering and cluster placement algorithms will be run 35 times for each random system. This

is especially important since both algorithms rely on randomly picking the nodes for object migration and swapping. Therefore there is a chance of producing slightly different results due to partly relying on chance. It is expected that the algorithms will converge to the same results most of the time. Multiple tests will prove whether or not this will be the fact. The results of the experiments will be stored in comma-separated text files with timestamps and experiment names so that they're later easily accessed and used for analysis and plotting. The output of the console window of Python showing the status and the progress of the experiments will also be stored. The algorithms will optimize the randomly generated systems and each time calculate the new total cost of communication and the difference between the unoptimized and the optimized system together with other indicators such as the time used on graph partitioning process, the number of times reward or penalty procedures had to be invoked, time used on the simulated annealing process, the number of times the process chose an inferior state, etc.

One of the indicators of the graph partitioning algorithm's performance will be the intracluster communication before and after graph partitioning. The traffic between each element inside a cluster will be summed up for each cluster in order to determine the intracluster traffic values. Another indicator to measure the performance of the VM clustering and cluster placement algorithms will be the time needed for them to converge and the stability in variation of the results.

### 3.8.1 Testbed for the experiments

Since the experiments will be conducted on simulated data center network topologies with the help of the static traffic traces obtained through the third party and the tests don't require special equipment, real networking infrastructure, significant computing power, extensive memory or storage the tests will be conducted on a single desktop workstation.

The workstation will be running 64-bits Microsoft Windows 8.1 Enterprise. It will be equipped with a 64-bits Intel Xeon 3.20GHz processor with 4 cores and 16GB of physical memory. Win32 version of Python 2.7.9 will be used for all the experiments. The scripts running the tests during all the experiments will be launched from the Python's Integrated DeveLopment

Environment (IDLE) graphical user interface (GUI).

### 3.8.2 Plotting and analysis

In order to visualize the results of the experiments the data will be plotted as graphs. The graphs will be used for later analysis of the results. The `matplotlib` library of Python will be used as the plotting tool. Python's `numpy` library will provide the means for the descriptive statistical analysis. Average values, standard deviations, maximums, minimums and medians will be calculated and used in the analysis phase.

# Chapter 4

# Results

This section will describe the experiments conducted as well as the outcome of the tests and the obtained results.

## 4.1 Implementation of the algorithms

During the project a VM clustering and a cluster placement algorithms were designed and implemented in order to solve the graph partitioning and the quadratic assignment problems described in the approach chapter. This section will describe how the algorithms were developed using Python scripting language and how they solved the GPP and the QAP.

### 4.1.1 VM clustering algorithm

After parsing the traffic traces data the list of totally 5488 IP addresses (VMs) and the dictionary of the communicating pairs of IPs and the respective traffic rates were obtained and stored in `all_ips.txt` and `traffic_rates_list.txt` files. After this 1600 IPs were chosen randomly and the associated traffic matrix was built. The VM clustering algorithm was implemented by the help of Python scripts as discussed in the approach chapter (see 3.5). In order to simulate VM groups Python's built in `list` functionality was used.

The 1600 nodes were divided into 16 sub-partitions. Each sub-partition was assigned a *memory depth* property (M) which described how many different states or positions a vertex in the group could have. One position could be occupied either by one vertex or shared by multiple vertices.

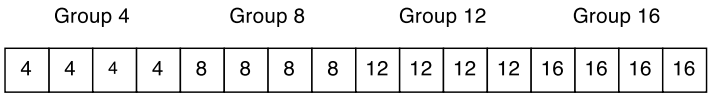


Figure 4.1: An example of four sub-partitions containing four VMs each

The example (see Fig.4.1) illustrates four sub-partitions or groups with memory depth of 4. Each group contains four VMs. The cell numbers refer to the VM states. The first four VMs belong to the *Group 4* and all the four VMs in the sub-partition are in the state 4, which is the external boundary state of the Group 4. The external boundary is also called *MinimumCertainty* state. The innermost state in a group was called *MaximumCertainty* state. The cells are numbered from 0 to 15 and the cell numbers correspond to the VM IDs that can be translated to IP addresses by referring to the `all_ips.txt` reference.

In Python the above example was implemented as follows:

```
Example set of 16 vertices to be partitioned in 4 sub-partitions
groups = [4, 4, 4, 4, 8, 8, 8, 8, 12, 12, 12, 12, 16, 16, 16, 16]
```

Each element of a Python list *groups* has an index depending on the placement of the element in the list and is constant in that regard. Each element has also a value, which will be changing throughout the graph partitioning process as the vertices are moved from one position to another and migrate between the sub-partitions. Hence the element value in this project referred to the position inside a specific sub-partition and was a subject to change whereas the index of the list element referred to the $ID$ of a specific VM and was constant.

**Object migration automata**

In order to move a node from one sub-partition to another the value of the list element with the index corresponding to the $ID$ of the VM was changed. For example in order to move a VM with the $ID = 3$ from its current sub-partition *4* to a new sub-partition *12* the following changes were

done in Python:

```
Example of moving vertices between sub-partitions
groups = [4, 4, 4, 12, 8, 8, 8, 8, 12, 12, 12, 12, 16, 16, 16, 16]
```

Note that in the example below it's clear that the sub-partition 4 now contains only 3 vertices (*ID*: 0, 1, 2) while sub-partition 12 consists of 5 (*ID*: 3, 8, 9, 10, 11) nodes. One way to even out the size of the vertices is to choose one vertex from the sub-partition 12 and move it to sub-partition 4. This is done in the example below:

```
Example of moving vertices between sub-partitions
groups = [4, 4, 4, 12, 8, 8, 8, 8, 4, 12, 12, 12, 16, 16, 16, 16]
```

Thus the VMs with *IDs* 3 and 8 have swapped places.

**Reward and penalty mechanisms**

After calculating the *similarity* and the *dissimilarity* thresholds the algorithm placed all the randomly chosen 1600 vertices in the *MinimumCertainty* states of their corresponding groups by assigning random VMs to the random groups. This state was assumed to correspond to a typical data center containing 1600 VMs who were scattered across the 16 physical hosts across the 16 server racks without taking in account the traffic between the VMs.

The algorithm then calculated the number of maximum iterations by counting the number of edges between the 1600 VMs and multiplying the result by 20. During each iteration two random and distinct nodes were picked from the total number of the nodes and the corresponding traffic rate was pulled from the `traffic matrix`. If the value was over the *similarity threshold* the vertices were be regarded as *similar*. If the traffic rate is lower than *dissimilarity threshold* the vertices were considered as *dissimilar*. The next step of the algorithm was to check whether the two chosen vertices were currently situated in the same or in distinct sub-partitions. After this check one of the four following procedures were invoked:

### RewardSimilarNodes

This procedure was invoked when the two chosen vertices are found to be similar and they were currently in the same sub-partition. The procedure rewarded both vertices by decreasing their current state by 1, effectively moving both 1 position towards the state of *MaximumCertainty*. This process was called a reward procedure because as the result of repetitive rewarding of a vertex it would be moved deeper in its sub-partition and would have higher probability of staying in that sub-partition.

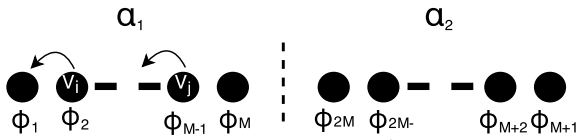


Figure 4.2: Reward transitions for the RewardSimilarNodes

### PenalizeSimilarNodes

The *PenalizeSimilarNodes* procedure was invoked when the two vertices were found to be similar but they resided in two distinct sub-partitions. In this case both vertices were penalized by moving both of them one position towards the external boundaries of their groups, closer to the *MinimumCertainty* state. However, if one of the vertices already was in the external boundary state it would be migrated over to the group of the second vertex and in order to compensate for a missing vertex a node closest to the external boundary (*MinimumCertainty*) of its new group would be moved to the old group of the first vertex where it would be placed in the external boundary position. If both vertices were already in the external boundary state then one of them was moved to the group of the other one and a vertex closest to the *MinimumCertainty* was migrated back to the sub-partition missing one vertex and placed in the *MinimumCertainty* state.

### PenalizeDissimilarNodes

If the two vertices were found to be dissimilar and they were also in the same sub-partition the *PenalizeDissimilarSame* procedure was invoked

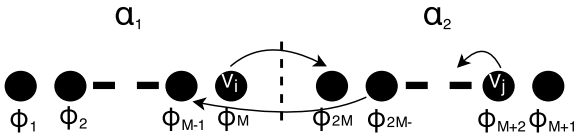


Figure 4.3: Penalty transitions for the PenalizeSimilarNodes

which would move the two vertices with one position towards the *MinimumCertainty* state.

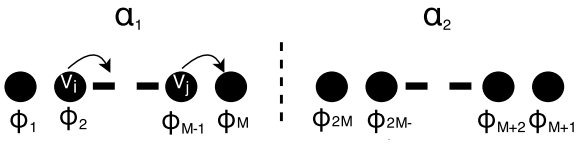


Figure 4.4: Penalty transitions for the PenalizeDissimilarNodes

If both vertices were found in the state of *MinimumCertainty* no action was taken. If one of the vertices was already in the MinimumCertainty state it was moved to the group containing a vertex with which it had the highest edge in the innermost state of its own group. This resulted in an excess of vertices in the new sub-partition and scarcity in the old one. One candidate was chosen in the new sub-partition in order to move it to the old one. The vertex closest to the state of *MinimumCertainty* was moved.

**Changes in Oommen's approach**

Two changes were done to the original approach of Oommen in order to tweak the algorithm adapted for this project and make it more suitable for the data used in this project.

The *RewardDissimilarNodes* was originally used in Oommen's algorithm [51] when the two chosen vertices were found to be dissimilar and they were at the same time found to be in two distinct sub-partitions. In this scenario both vertices were moved towards their most internal (*MaximumCertainty*) states. However in this project this procedure was removed. When dissimilar vertices were found in distinct sub-partitions they remained in their current states and no action was taken. The reason behind this deci-

sion was that there were expected to be too many dissimilar nodes which would cause overwhelming majority of the iterations to pick the dissimilar pairs and push the nodes towards the MaximumCertainty most of the times. This would significantly lower the chances for the few vertices with strong connections who happened to be in distinct sub-partitions to be able to get out of their sub-partitions and group together.

The formula for calculating the number of iterations was changed from multiplying the number of edges by 100 to multiplying the number of edges by 20.

### VM clustering algorithm pseudocode

The designed and implemented VM clustering algorithm is described by the following pseudocode:

- $V = V_1, V_2, ..., V_{\{}KN\}$: The set of vertices to be partitioned
- $(\alpha_1, \alpha_2, ..., \alpha_K)$: Set of actions a node can fall into (K sub-partitions)
- $\Phi_1, \Phi_2, ..., \Phi_{KM}$: Set of memory states or memory depth (M)
- $E$: Edges between the nodes with the associated traffic matrix D
- $\beta = \{0, 1\}$: Input set, where 0 is reward and 1 is penalty
- $Q$: Transition function, which explains how the vertices should be moved between states
- $G$: Function, which partitions the set of states for the sub-partitions

**Procedure** RewardSimilarNodes(i,j)

**Data**: Node indices i and j, where $\omega_i$ and $\omega_j$ are the state indices of similar nodes in the same sub-partition.

**if** $\omega_i \mod M \neq 1$ **then** `/* i is not in the most internal state */`

$\omega_i = \omega_i - 1$

**if** $\omega_j \mod M \neq 1$ **then** `/* j is not in the most internal state */`

$\omega_j = \omega_j - 1$

**Procedure** PenalizeSimilarNodes(i,j)

**Data**: Node indices i and j, where $\omega_i$ and $\omega_j$ are the state indices of similar nodes in the different sub-partitions.

**if** $(((w_i \mod M) \neq 0) and ((w_i \mod M) \neq 0))$ **then**
    $\omega_i = \omega_i + 1$ /* both are in internal states */
    $\omega_j = \omega_j + 1$
    **else**
    **if** $\omega_i \mod M \neq 0$ **then** /* $v_i$ is in internal state */
        $\omega_i = \omega_i + 1$ /* update state of $v_i$ */
        temp = $\omega_j$ /* store the state of $v_j$ */
        $\omega_j = (\omega_i divM) \cdot M$ /* move $v_j$ to $v_i$'s sub-partition */
        t := index of a node in $v_i$'s sub-partition with $v_t \neq v_i$ and $v_t$ closest to the boundary state of $\omega_i$
        $\omega_t$ = temp /* move $v_t$ to the old state of $v_j$ */
        **else**
        **if** $\omega_j \mod M \neq 0$ **then** /* $v_j$ has to be moved */
            $\omega_j = \omega_j + 1$ /* update state of $v_j$ */
            temp = $\omega_i$ /* store the state of $v_i$ */
            $\omega_i = (\omega_j divM) \cdot M$ /* move $v_i$ to $v_j$'s sub-partition */
            t := index of a node in $v_j$'s sub-partition with $v_t \neq v_j$ and $v_t$ closest to the boundary state of $\omega_j$
            $\omega_t$ = temp /* move $v_t$ to the old state of $v_i$ */

**Algorithm 2: ClusterVMs**

**Input:** The set $V = \{v_1, v_2, ..., v_{KN}\}$ to be partitioned into K sub-partitions.

D is adjacency traffic matrix and $V_1$, $V_2$ ... $V_K$ are current feasible sub-partitions.

$\rho$ is a parameter used to determine the similarity or dissimilarity of the vertices. M=100 for the experiments in this project.

The algorithm is run for certain fixed numbers of iterations.

**Output:** The final partitions $\{V_1, V_2, ..., V_K\}$

**Preprocess:**

Compute Mean_Edge.

Randomly partition V into $\{V_1, V_2, ..., V_K\}$

Assign all nodes to the boundary state of the actions

**Data**: Set of nodes to be partitioned: $V = \{v_1, v_2, ..., v_{KN}\}$

**Result**: The final solution to the GPP

**Method**:

```
for Iteration :=1 to Max_Iterations do
    for a random edge E_ij do
        if C_ij > (1 + ρ) · Mean_Edge then
            if v_i and v_j are in same sub-partition then
                RewardSimilarNodes(i,j)
            else
                PenalizeSimilarNodes(i,j)
        else
            if C_ij < (1 − ρ) · Mean_Edge then
                if v_i and v_j are in same sub-partition then
                    PenalizeDissimilarNodes(i,j)
            else
                Pass
```

**Procedure** PenalizeDissimilarNodes(i,j)

**Data**: Node indices i and j where $\omega_i$ and $\omega_j$ are the state indices of dissimilar nodes in the same sub-partition

**if** $(((w_i \mod M) \neq 0) and ((w_i \mod M) \neq 0))$ **then**

- $\omega_i = \omega_i + 1$ /* both are in internal states */
- $\omega_j = \omega_j + 1$

**else**

- **if** $\omega_i \mod M \neq 0$ **then** /* $v_j$ is in internal state */
  - $\omega_i = \omega_i + 1$ /* update state of $v_i$ */
  - TempState1 = EvaluateCost of current partitioning /* store the state of $v_j$ */
  - Prev_Cost = EvaluateCost of current partitioning
  - **for** *all remaining* $K-1$ *partitions* **do**
    - $\omega_p$ = state of node closest to boundary in this current sub-partition
    - TempState2 = $\omega_p$
    - $\omega_j = (\omega_p divM + 1) \cdot M$/* move $v_j$ to new sub-partition */
    - $\omega_p = TempState1$ /* move $v_p$ to $v_j$'s old state */
    - New_Cost = EvaluateCost of current partitioning
    - **if** *New_Cost* $>$ *Prev_Cost* **then**
      - $\omega_p = TempState2$ /* change is not superior */
      - $\omega_j = TempState1$ /* undo it */
    - **else** /* this change is superior */
      - Prev_Cost = New_Cost /* retain it */
  - **else** /* $v_j$ is in internal state */
    - $\omega_j = \omega_j + 1$ /* update state of $v_j$ */
    - TempState1 = $\omega_i$ /* store state of $v_i$ */
    - Prev_Cost = EvaluateCost of current partitioning
    - **for** *all remaining* $K-1$ *partitions* **do**
      - $\omega_p$ = state of node closest to boundary in this current sub-partition, $\alpha_Z$
      - TempState2 = $\omega_p$
      - $\omega_i = (\omega_p divM + 1) \cdot M$ /* move $v_i$ to new sub-partition */
      - $\omega_p$ = TempState1 /* move $v_p$ to old state of $v_i$ */
      - New_Cost = EvaluateCost of current partitioning
      - **if** *New_Cost* $>$ *Prev_Cost* **then**
        - $\omega_p = TempState2$ /* change is not superior */
        - $\omega_i = TempState1$ /* undo it */
      - **else** /* this change is superior */
        - Prev_Cost = New_Cost /* retain it */
      - /* move $v_t$ to the old state of $v_i$ */

## 4.2 Implementation of the cluster placement algorithm

After partitioning the initial set of nodes in a way which placed the nodes with the high mutual traffic in the same sub-partitions a cluster placement algorithm was developed using simulated annealing (SA) as described in the section 3.5 in the approach chapter in order to assign the produced 16 sub-partitions to the available 16 server racks in the most optimal way.

The algorithm was implemented as two main parts:

1. Initiate the most optimal start position
2. Swap random clusters to find a better position

In order to quickly obtain the traffic between any given clusters the cluster matrix $S$ was generated by creating a static cluster reference list and then creating a matrix with 16 rows and 16 columns where each row and column corresponded to a cluster in the cluster reference. The edge values between the clusters was calculated by summing the traffic rates between every single member VM of the two clusters. The first part of the algorithm was implemented by reading the list of partitioned nodes produced by the VM clustering algorithm, generating all the possible pair combinations and creating the list of the pairs with the corresponding edge values taken from the cluster matrix $S$. After this the pairs were sorted by the traffic in the descending order. This created an initial state which already resulted in lower total cost of communication compared to the randomly placed cluster set. The initial cluster placement at this stage was regarded as the *best state*.

The second part of the algorithm attempted to further optimize the cluster placement by randomly choosing two clusters and swapping them as displayed below:

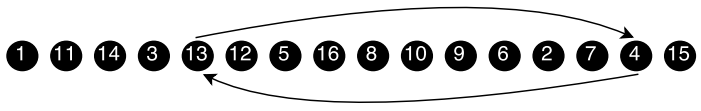


Figure 4.5: Set of clusters after swapping two random clusters

In Python this was achieved by picking two random indices of the list representing the set of clusters and then swapping them. The index of the list corresponded to a server rack number. The element of the list with the index 0 corresponded to the rack number 1 while the element with the index 15 corresponded to the rack 16. The values of the list elements corresponded to the cluster name as defined in the cluster reference:

Example of choosing two random elements from the list

```
groups = [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100,
1200, 1300, 1400, 1500, 1600]
```

In the examples above list elements with index 3 and 8 are randomly picked. The first list shows that the rack 4 (list[3] = $Rack_4$) has the cluster 400 assigned while the rack 9 (list[8] = $Rack_9$) has the cluster 900 assigned.

Example of swapping two random clusters

```
groups = [100, 200, 300, 900, 500, 600, 700, 800, 400, 1000, 1100,
1200, 1300, 1400, 1500, 1600]
```

In the example displayed above the list elements are swapped. The list now contains 900 as the list[3] element meaning that the server rack number 4 now has cluster 900 assigned, while list[8]=400 meaning that the rack 9 has the cluster 400 assigned.

The execution of the algorithm was controlled by establishing the initial temperature $T$ with a certain value which was gradually decreased after each iteration. The $T$ value was also used at each iteration to calculate the probability $P$ value with the formula (see 3.7) described in the approach section.

After each time two random clusters were swapped the total cost of communication was computed and if the new value was lower than the previous total cost of communication the algorithm chose the new placement as the *best state*. If the new total cost of communication was however higher than the previously set *best state* the algorithm conducted a check to compare the current probability $P$ value with a random value between 0 and 1. If the $P$ value was greater than the random value the algorithm chose the current state as the *best state* in order to explore broader

range of the available solution space and avoid assuming a local minimum to be the global minimum (see Fig. 3.1) as explained in the approach chapter.

### Cluster placement algorithm pseudocode

The implemented cluster placement algorithm is described in detail in the pseudocode below:

**Algorithm 3: Place_Clusters**

**Input:** Set of $N$ partitioned VM clusters $G = \{g_1, ...g_{KN}\}$ to be assigned to $K$ server racks.
**Output:** Final solution to QAP.

**Preprocess:**
Compute the cluster communication matrix $S$.
Find the highest mutual traffic cluster pairs and sort the set of clusters accordingly. Store the initial state as *BestState*
Calculate the corresponding total cost of communication, $TotalCost_{BestState}$

```
for Temperature := T to 0 do
    Decrease T
    for random distinct clusters G_i and G_j do
        TempState = SwapPositions
        Calculate TotalCost_TempState
        if TotalCost_TempState < TotalCost_BestState then
            BestState = TempState /* go to the new state      */
            BestTotalCost = TotalCost_BestState
        else
            Retain BestState
        if TotalCost_TempState > TotalCost_BestState then
            P = e^(-Δ/T) /* Calculate probability P            */
            if P < RandomValue then
                BestState = TempState /* go to the new state */
                BestTotalCost = TotalCost_BestState
            else
                retain BestState
```

## 4.3 Developed Python scripts

Several Python scripts were developed in order to implement the VM clustering and cluster placement algorithms and to handle the tasks needed to conduct the experiments involving these algorithms in the way which is described in the approach section. The scripts were also implemented to collect the experiment results and organize them in output files in the cleanest possible way to make the plotting and analysis process as uncomplicated as possible.

### 4.3.1 Script: parse_data.py

`Parse_data.py` script was developed in order to automatically parse the obtained traffic traces, find and extract the relevant data, set up the IP address reference and the dictionary of communicating pairs and the number of transmitted bytes, to construct the asymmetric traffic matrix and then to create and populate the symmetric traffic matrix $D$ heavily used during the experiments.

Console output of the parse_data.py script

```
2015-05-02 19:22:03 : started reading input

2015-05-02 19:22:03 : parsing: ../univ1_pt1.txt
2015-05-02 19:22:10 : finished parsing: ../univ1_pt1.txt
2015-05-02 19:22:11 : parsing: ../univ1_pt2.txt
2015-05-02 19:22:18 : finished parsing: ../univ1_pt2.txt
...
...
...
2015-05-02 19:24:21 : finished parsing: ../univ1_pt18.txt
2015-05-02 19:24:21 : parsing: ../univ1_pt19.txt
2015-05-02 19:24:30 : finished parsing: ../univ1_pt19.txt
2015-05-02 19:24:30 : parsing: ../univ1_pt20.txt
2015-05-02 19:24:39 : finished parsing: ../univ1_pt20.txt
2015-05-02 19:24:40 : saved file: traffic_rates_list.txt

----------------------------------------
Number of unique IPs: 5488
Number of communicating pairs: 12567
2015-05-02 19:24:40 : finished getting input
```

```
----------------------------------------

2015-05-02 19:24:40 : saved file: all_ips.txt
2015-05-02 19:24:40 : populating the asymmetric matrix.
Matrix dimensions: 1600  by 1600
2015-05-03 08:07:41 : wiping 1600_asym_matrix.pkl ...
2015-05-03 08:07:41 : writing data to 1600_asym_matrix.pkl

done.
2015-05-03 08:07:48
2015-05-03 08:07:48 : populating the symmetric matrix.
Matrix dimensions: 1600  by 1600
2015-05-03 08:07:49 : finished populating the symmetric matrix...
2015-05-03 08:07:49 : wiping 1600_sym_matrix.pkl ...
2015-05-03 08:07:49 : saving symmetric matrix...

done.
2015-05-03 08:07:57
```

The `parse_data.py` script read 2.57GB of data in 20 text files in roughly 3 minutes and generated several output files: `all_ips.txt`, `traffic_rates_list.txt`, `1600_asym_matrix.pkl` and `1600_sym_matrix.pkl`. The script took approximately 12 hours to go through the 2560000 elements of the matrix, search for each edge in the `traffic_rates_list.txt` list referring to the `all_ips.txt` reference.

### 4.3.2 Script: cluster_vms.py

The `cluster_vms.py` script was developed in order to solve the graph partitioning problem (GPP) by implementing the VM clustering algorithm described in the previous sections. The script generated a set of 16 sub-partitions and randomly assigned the available 1600 VMs to the different sub-partitions so that each contained exactly 100 VMs. The script then read `1600_sym_matrix.pkl` file in order to scan the traffic matrix and to calculate the mean edge value. The mean edge value was used to compute the similarity (see Fig. 3.3) and dissimilarity (see Fig. 3.4) thresholds as described in the section 3.3. The `cluster_vms` script additionally calculated the number of iterations (number of times it would pick a random pair for

partitioning) by counting the number of edges and calculating it by 20. After this the script proceeded to pick two random nodes, compared their corresponding edge $D_{ij}$ (by looking it up in the symmetric traffic matrix $D$) to the similarity and dissimilarity thresholds and either invoked a reward or penalize procedure or took no action depending on whether the nodes appeared to be in distinct or the same sub-partitions.

Console output of the cluster_vms.py script

```
threshold similar 1020756.33985
threshold dissimilar 612453.803907
starting the graph partitioning algorithm with 82920 iterations

2015-05-07 13:46:43

0 iterations done. 0.0% finished (2015-05-07 13:46:43)
16584 iterations done. 20.0% finished (2015-05-07 13:46:45)
33168 iterations done. 40.0% finished (2015-05-07 13:46:45)
49752 iterations done. 60.0% finished (2015-05-07 13:46:46)
66336 iterations done. 80.0% finished (2015-05-07 13:46:46)
-----------------
```

The `cluster_vms.py` script returned a set of values: the set of the graph partitioned nodes, number of times the RewardSimilarNodes, PenalizeSimilarNodes and PenalizeDissimilarNodes procedure had been invoked, time used for partitioning (seconds), number of the edges over the similarity threshold and number of the edges under the dissimilarity threshold.

### 4.3.3 Script: place_clusters.py

The `place_clusters.py` script was developed in order to implement the cluster placement algorithm described in the previous sections. It read the set of graph partitioned nodes produced by the `cluster_vms.py` script and to solve the quadratic assignment problem (QAP) as discussed in the section 3.5. After reading the set of partitioned nodes the script first ranked the pairs of clusters by intercommunication rates and sorted them in descending order in order to generate a new sequence of the nodes with lower total cost of communication. After this step the script proceeded to use the simulated annealing technique on the set by swapping two random clusters.

After each swap the script checked the state of the system. If the new state was superior to the previous state the script would adopt it as the current best state. However, if the new state was inferior to the previous state the script would calculate the probability value and would move to the inferior state with that probability. The probability was calculated by constantly decreasing the initially set $T$ (temperature) value and using it in the formula (see 3.7) described in section 3.5.

Console output of the place_clusters.py script

```
starting VM clustering algorithm with T= 500000 :
2015-05-07 13:46:48

initial total cost is: 1502038700.0
T=500000: 100.0% hot
chosen inferior placement as P = 0.301634720373 >
0.0061017362148 random
chosen inferior placement as P = 0.779886007112 >
0.242585755024 random
chosen inferior placement as P = 0.480433392584 >
0.0263669433088 random
chosen inferior placement as P = 0.240029743912 >
0.161003559517 random
...
T=400000: 80.0% hot
chosen inferior placement as P = 0.0773042938568 >
0.0592923019254 random
chosen inferior placement as P = 0.150254013297 >
0.0899942025324 random
chosen inferior placement as P = 0.158967675765 >
0.0415338339246 random
...
T=300000: 60.0% hot
chosen inferior placement as P = 0.777697575015 >
0.250850162891 random
chosen inferior placement as P = 0.391799794669 >
0.32120927418 random
...
T=200000: 40.0% hot
chosen inferior placement as P = 0.27493613225 >
0.242596981898 random
```

```
T=100000: 20.0% hot
best total cost is: 1493339638.0
finished VM clustering algorithm: 2015-05-07 13:49:51
```

The algorithm finished working when the temperature became 0 after gradually "cooling down".

### 4.3.4 Script: generate_random_placements.py

This script was developed with the sole purpose to create the necessary number of randomly assigned set of nodes. The script used the Python's `random` library to achieve this and output 35 or 40 different .pkl files with the use of `pickle` library.

Console output of the generate_random_placements.py script

```
created directory

2015-05-03 16:52:46 : writing data to random_placements/
1_random_placement_vms.pkl
2015-05-03 16:52:46 : writing data to random_placements/
2_random_placement_vms.pkl
2015-05-03 16:52:46 : writing data to random_placements/
3_random_placement_vms.pkl
2015-05-03 16:52:46 : writing data to random_placements/
4_random_placement_vms.pkl
...
2015-05-03 16:52:47 : writing data to random_placements/
37_random_placement_vms.pkl
2015-05-03 16:52:47 : writing data to random_placements/
38_random_placement_vms.pkl
2015-05-03 16:52:47 : writing data to random_placements/
39_random_placement_vms.pkl
2015-05-03 16:52:47 : writing data to random_placements/
40_random_placement_vms.pkl
```

These files were later used by the other scripts in order to establish a baseline from the average values generated by unoptimized, randomly dispersed sets of VMs.

### 4.3.5 Script: cluster_and_calculate_tot_cost.py

The `cluster_vms.py` and the `place_clusters.py` scripts were managed in the larger `cluster_and_place_vms.py` script which was developed in order to automate the experiments by generating 35 randomly placed VM sets and for each placement run graph partitioning and consecutively quadratic assignment algorithms, measure the total cost before and after optimization, calculate the impact and organize and store the output data. The `cluster_and_place_vms.py` was developed to only require the set of the input files containing the randomly generated VM sets. The rest of the processes were managed by the script and didn't require human assistance.

The script was run with the same input files for all three experiments. The only difference was the parameter which specified the cost matrix the script was supposed to use in order to calculate the total cost of communication after each optimization. The script would either use the *tree_cost_matrix*, the *fattree_cost_matrix* or the *vl2_cost_matrix* depending on the experiment data center model.

The output of the script was a comma-separated plain text file, one for each DCN model. The file was named according to the DCN model. For example the output file for the Tree data center model experiment was `tree_dcn_total_cost.txt` while the output file name for the Fat-tree experiment was `fattree_dcn_total_cost.txt` and the output file for the VL2 DCN model experiment was called `vl2_dcn_total_cost.txt`. The contents of the output files looked as follows:

Example contents of the tree_dcn_total_cost.txt file

```
2015-05-04 22:13:54,1_random_placement_vms.pkl,12235176726.0,
1921962355.0,-84.291503114,4,82920,360,3628,5492,1628,2239,
1270398209.0,-89.6168380936,37,1,100000

2015-05-04 22:14:37,1_random_placement_vms.pkl,12235176726.0,
1861169022.0,-84.7883764683,4,82920,360,3628,5392,1807,
1981,1291657024.0,-89.4430864962,38,5,100000

2015-05-04 22:15:20,1_random_placement_vms.pkl,12235176726.0,
1575026771.0,-87.1270615352,3,82920,360,3628,5807,1419,
2276,1227001125.0,-89.9715292024,37,2,100000
```

```
2015-05-04 22:16:02,1_random_placement_vms.pkl,12235176726.0,
1480767988.0,-87.8974532108,3,82920,360,3628,5817,1457,
1637,1209675742.0,-90.113132249,36,0,100000
```

Each row in the output file produced by the `cluster_and_place_vms.py` script and demonstrated above contained following 16 values:

1. Timestamp
2. Random set file name
3. Total cost of communication of the random set
4. Total cost of communication after graph partitioning
5. Percent change in total cost of communication after graph partitioning
6. Seconds used on clustering the set of random VMs
7. Number of iterations used by the clustering algorithm
8. Number of the edges over the similarity threshold
9. Number of the edges under the dissimilarity threshold
10. Number of RewardSimilarNodes invoked
11. Number of PenalizeSimilarNodes invoked
12. Number of PenalizeDissimilarNodes invoked
13. Total cost of communication after cluster placement
14. Percent change of the total cost of communication after quadratic assignment compared to the randomly placed nodes
15. Seconds used by the cluster placement algorithm
16. Number of times the simulated annealing mechanism chose inferior state
17. The initial temperature used for simulated annealing.

### 4.3.6 Script: intracluster_comm.py

The `intracluster_communications.py` script was developed to handle the execution of the intracluster and intercluster experiments and write the results in a comma-separated format in the output `intracluster_comm_35.txt` file.

Example contents of the intracluster_comm_35.txt file

```
2015-05-08 23:06:51,1_random_placement_vms.pkl,2189549.4375,
91044283.5625,4058.12870005,100,0,100,11380010.0,13814912.6917,
1967614.80833,-85.7573127515

2015-05-08 23:06:51,1_random_placement_vms.pkl,2189549.4375,
91044283.5625,4058.12870005,200,2552.0,200,20682398.5,
13814912.6917,1967614.80833,-85.7573127515

2015-05-08 23:06:51,1_random_placement_vms.pkl,2189549.4375,
91044283.5625,4058.12870005,300,6931704.0,300,318643080.5,
13814912.6917,1967614.80833,-85.7573127515

2015-05-08 23:06:51,1_random_placement_vms.pkl,2189549.4375,
91044283.5625,4058.12870005,400,1236674.5,400,40330603.0,
13814912.6917,1967614.80833,-85.7573127515
```

### 4.3.7 Script: analyze_and_plot.py

The `analyze_and_plot.py` script was developed in order to read the output files generated by the `cluster_and_place_vms.py` script and plot the results with the `matplotlib` library.

Example console output of the analyze_and_plot.py script

```
Tree experiment. Number of experiments: 35
average tot cost before optimization ( N = 35 ): 11744193654.7
stdev: 1422547350.22
min: 8309073280.0
max: 13930703965.0
median: 12126586036.0
------------------------------------------------------------

average tot c after graph partitioning: 1742925049.17
N = 35
stdev: 51529292.9456
```

```
min: 1638339362.17
max: 1885657993.69
median: 1741854170.29
------------------------------------------------------------

average tot c after QAP: 1268149583.85
N = 35
stdev: 15796579.674
min: 1231258618.91
max: 1302712967.63
median: 1268084325.77
------------------------------------------------------------

improvement of tot c after GP: -85.1592616707 %
improvement of tot c after QAP: -89.2019016278 %
------------------------------------------------------------

average number of reward_similar_nodes: 5614.82122449
stdev: 31.3908384569

average number of penalize_similar_nodes: 1584.49632653
stdev: 30.0001291237

average number of penalize_dissimilar_nodes: 2208.02040816
stdev: 140.558790965
...
```

The script generated graphs in .pdf file format and stored them in the working directory.

Two sets of experiments were conducted with two different set of 1600 VMs picked from the obtained traffic traces.

## 4.4 Experiment set A

Three instances of the `cluster_and_place_vms.py` were run in parallel mode in order to simultaneously conduct the Tree, Fat-tree and VL2 experiments.

### 4.4.1 Experiment a1: Tree results

The experiment a1 was conducted on the Tree data center network architecture. The `cluster_and_place_vms.py` was run by specifying the cost matrix for the Tree DCN. The script read the 35 .pkl files containing the 35 randomly placed VM sets and started running the VM clustering and cluster placement algorithms 35 times for each of the 35 .pkl files. Thus the optimization algorithms were run totally 1225 times for the Tree experiment. The number of iterations for the VM clustering algorithm was calculated to be 82920 each time. The *similarity threshold* was calculated to be 1020756.33985 and the *dissimilarity threshold* was 612453.803907. The initial temperature sent to the cluster placement algorithm for the simulated annealing process was $T = 100000$. Each test took roughly 3 minutes. The whole experiment with 1225 tests took approximately 19 hours starting 09.05.2015 at 17:44:30 and finishing 10.05.2015 at 08:28:26.

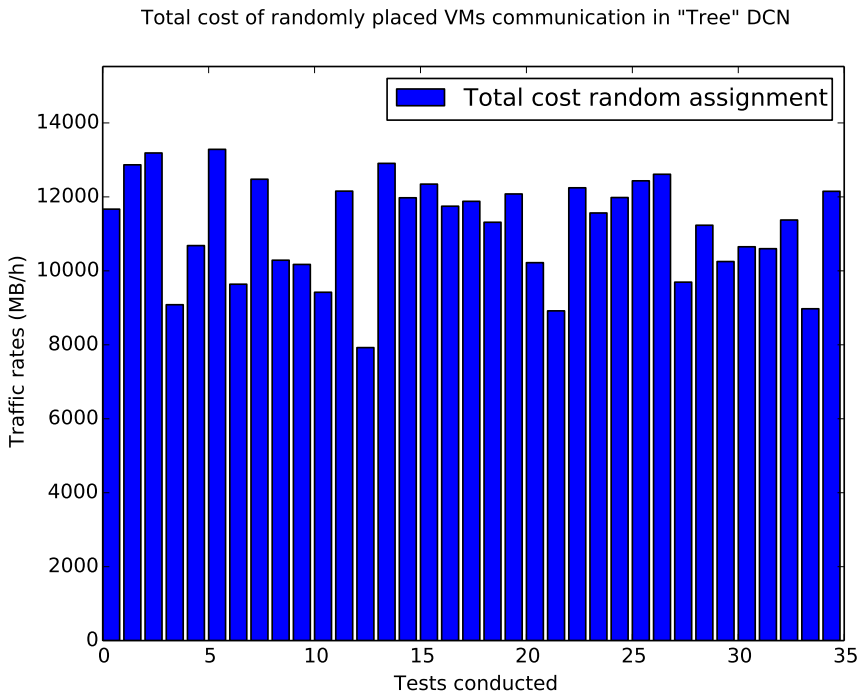


Figure 4.6: Total cost of communication in Tree with random assignments in set A

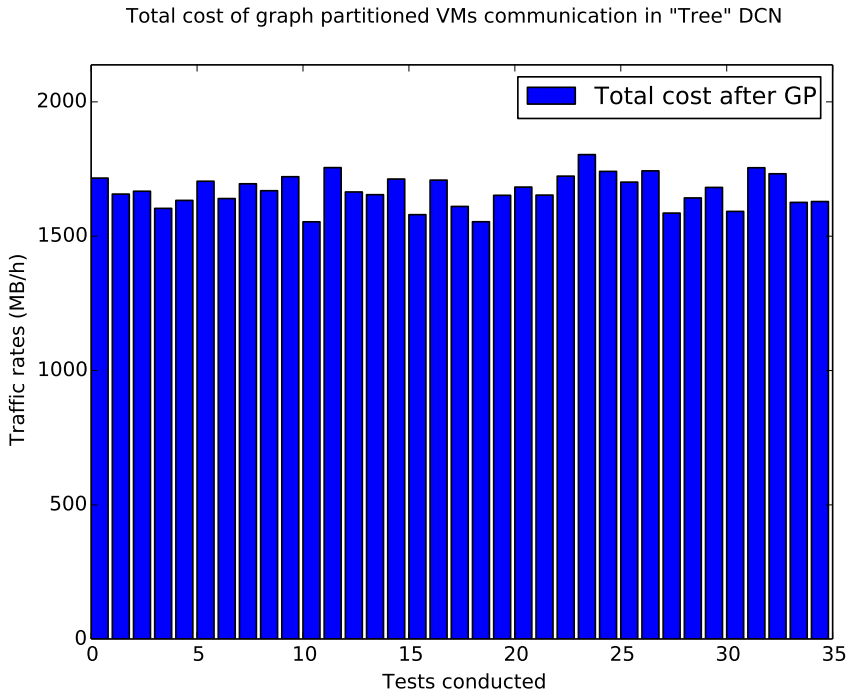


Figure 4.7: Total cost of communication in Tree after VM clustering in set A

The average total cost of communication of the randomly distributed VMs in a Tree data center ($T_{RandTreeA}$) was measured to be 11744193654.7 bytes (see Fig. 4.6) or 10.9376 GB for the sample size of 35 with the standard deviation of 1422547350.22 bytes or 1.3249 GB.

The average total cost of communication after the VM clustering algorithm ($T_{GpTreeA}$) was measured 1751027376.48 bytes or 1.6308 GB (see Fig. 4.7) with standard deviation 62550001.98 bytes (59.6523 MB). The results after cluster assignment showed further decrease in the total cost of communication. The average total cost of communication after cluster assignment ($T_{QapTreeA}$) algorithm (see Fig. 4.8) was 49621895.5551 bytes (47.3231 MB) with standard deviation 3293752.44 bytes (3.1412 MB).

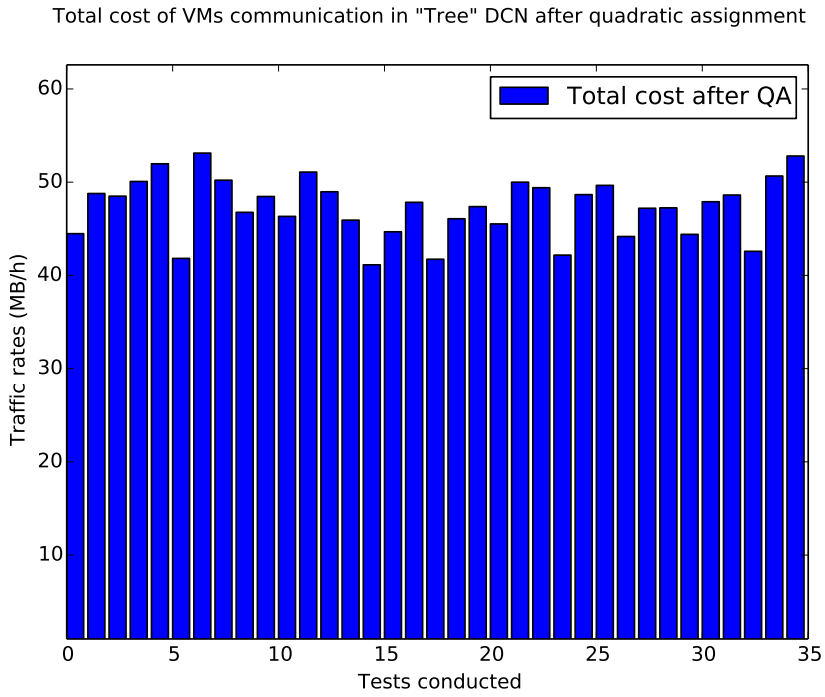


Figure 4.8: Total cost of communication in Tree after cluster placement in set A

### 4.4.2 Experiment a2: Fat-tree results

The experiment 2 was conducted on the Fat-tree data center network architecture. The `cluster_and_place_vms.py` script was launched in parallel mode with the experiment 1 by specifying the cost matrix for the Fat-tree DCN. The script read the same 35 .pkl files containing the 35 randomly placed VM sets and started running the VM clustering and cluster placement algorithms 35 times for each of the 35 .pkl files. Similar to the script instance running the experiment 1. The optimization algorithms were run totally 1225 times for the Fat-tree experiment with the same threshold and maximum iteration values. The number of iterations for the VM clustering algorithm was set to be 82920 each time. The *similarity threshold* was calculated to be 1020756.33 and the *dissimilarity threshold* was 612453.80 identically to the experiment 1.

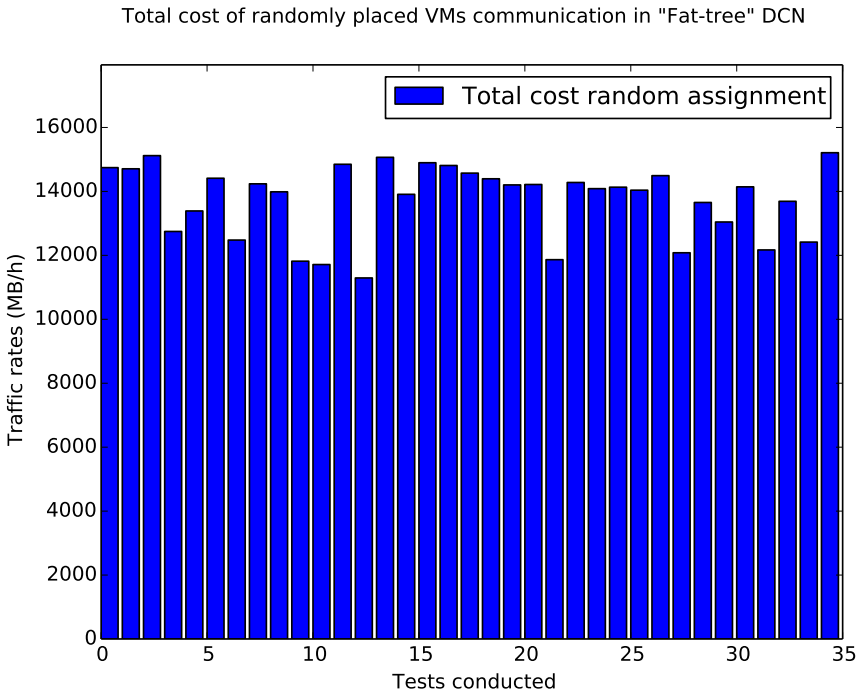


Figure 4.9: Total cost of communication in Fat-tree with random assignments in set A

The initial temperature sent to the cluster placement algorithm for the simulated annealing process was $T = 100000$. Each test took roughly 3 minutes during this experiment as well. The whole experiment with 1225 tests took roughly 16 hours starting 09.05.2015 at 17:44:24 and finishing 10.05.2015 at 10:22:57.

The average total cost of communication (see Fig.4.9) for the randomly placed VMs in the Fat-tree data center ($T_{RandFtreeA}$) was measured 14406082632.7 bytes (13.4167 GB) with standard deviation 1158102049.49 bytes (1.0786 GB).

The results showed that the average total cost of communication measured for the 35 scenarios after optimizing the VM placements (see Fig.4.10) with the use of VM clustering algorithm ($T_{GpFtreeA}$) was calculated to be 2140643564.18 bytes (1.9936 GB) with standard deviation 74675979.95 bytes (71.2166 MB). The average total cost of communication after further optimizing the clusters' placement (see Fig.4.11) with the cluster placement algorithm ($T_{QapFtreeA}$) was calculated to be 68162675.71 bytes (65.005 MB) with standard deviation 3101849.88 bytes (2.9582 MB).

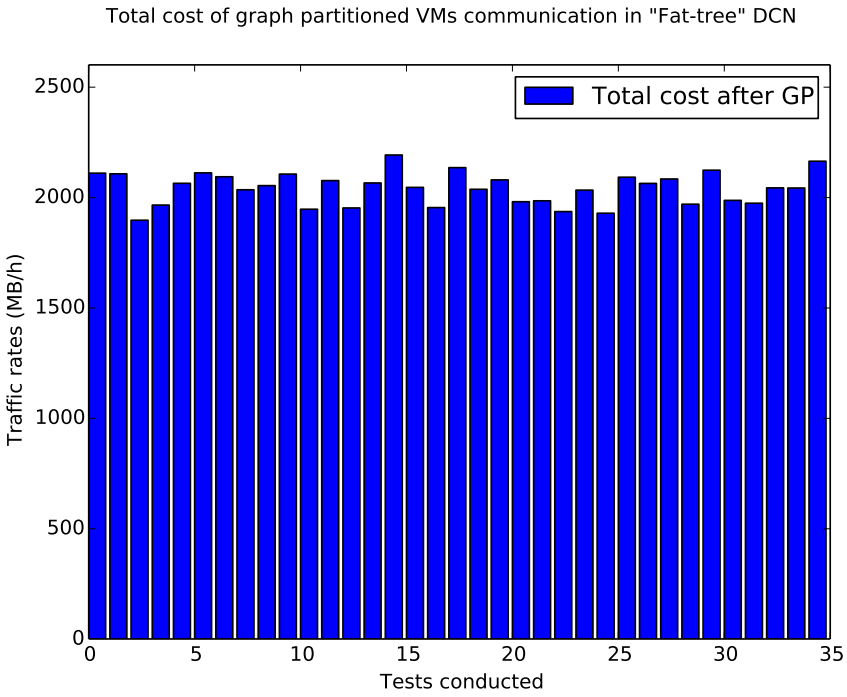


Figure 4.10: Total cost of communication in Fat-tree after VM clustering in set A

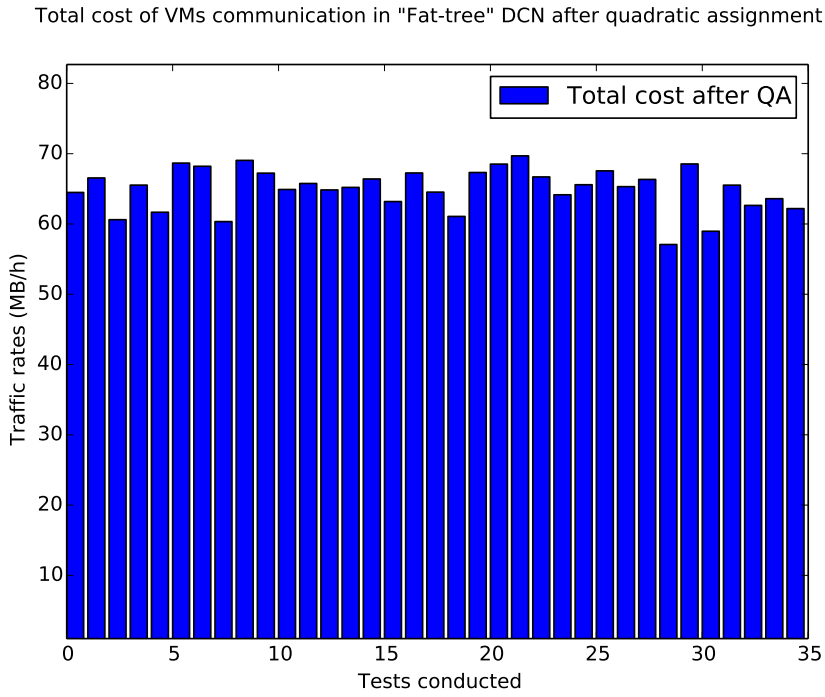


Figure 4.11: Total cost of communication in Fat-tree after cluster placement in set A

### 4.4.3 Experiment a3: VL2 results

The experiment 3 was conducted on the VL2 data center network architecture. The third parallel instance of the `cluster_and_place_vms.py` script was launched by specifying the cost matrix for the VL2 DCN. The script read the same 35 .pkl files containing the 35 randomly placed VM sets and started executing the VM clustering and cluster placement algorithms 35 times for each of the 35 .pkl files identically to the script instances running the experiment 1 and experiment 2. The optimization algorithms were run totally 1225 times for the VL2 experiment with the same threshold and maximum iteration values as in the previous two experiments. The number of iterations for the VM clustering algorithm was set to be 82920 each time in this experiment as well. The *similarity threshold* was calculated to be 1020756.33 and the *dissimilarity threshold* was 612453.80 identically to the experiment 1 and experiment 2. The initial temperature sent to the cluster placement algorithm for the simulated annealing process was $T = 100000$. Each test took roughly 3 minutes during this experiment as well. The whole experiment with 1225 tests took roughly 16 hours starting 09.05.2015 at 17:44:35 and finishing 10.05.2015 at 10:05:29.

According to the results the average total cost of communication for the VL2 DCN ($T_{RandVl2A}$) with randomly placed VMs (see Fig.4.12) was 13586759288.7 bytes (12.6537 GB) with standard deviation 1414884915.62 bytes (1.3177 GB). After the VM clustering (see Fig.4.13) the results showed the total cost of communication ($T_{GpVl2A}$) 2010382164.52 bytes (1.8723 GB) with standard deviation 53688282.77 bytes (51.2011 MB) whereas after the cluster placement (see Fig.4.14) the average total cost of communication ($T_{QapVl2A}$) was measured 59993636.63 bytes 57.2144 MB) with standard deviation 3292518.29 bytes (3.14 MB).

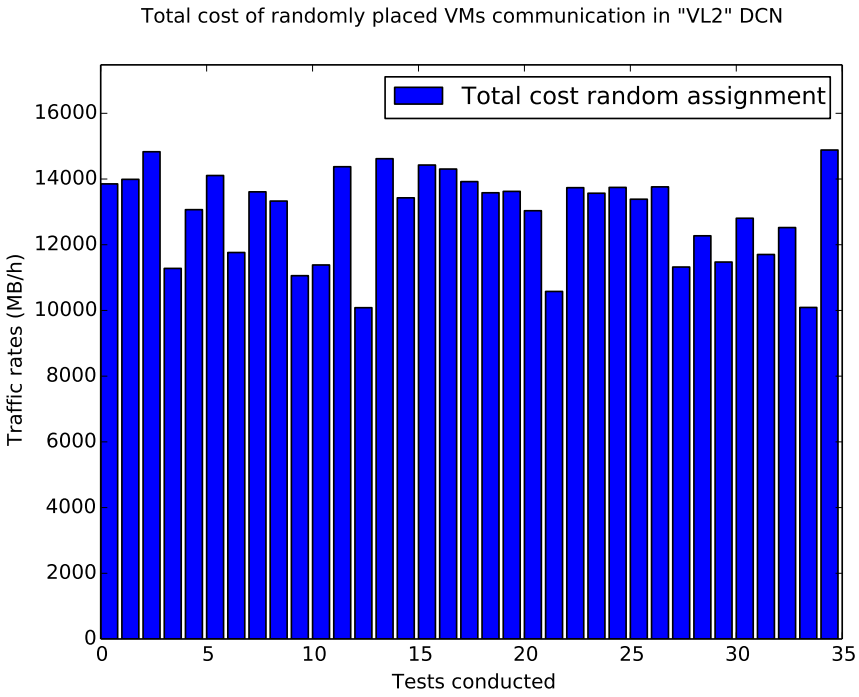


Figure 4.12: Total cost of communication in VL2 with random assignments in set A

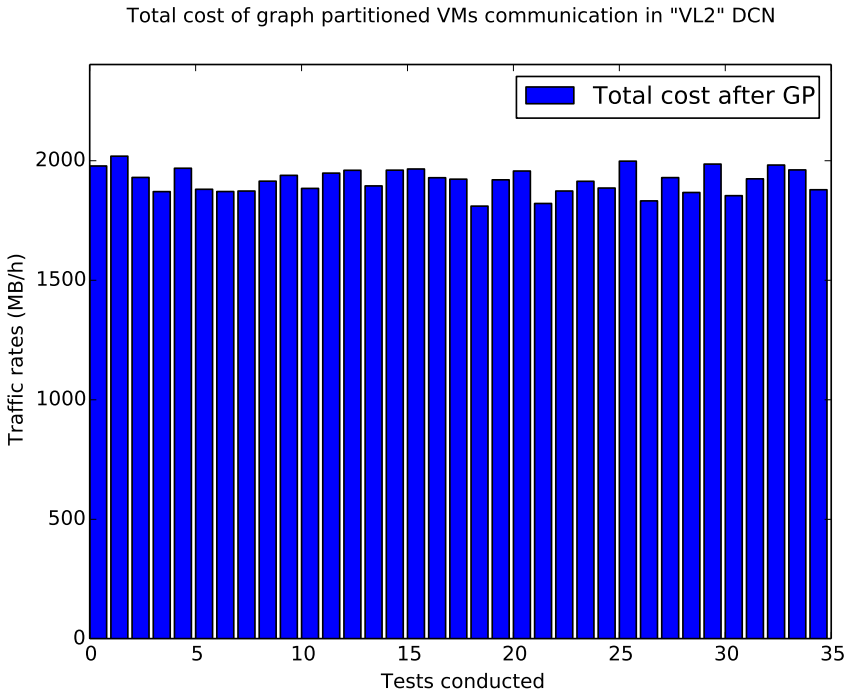


Figure 4.13: Total cost of communication in VL2 after VM clustering in set A

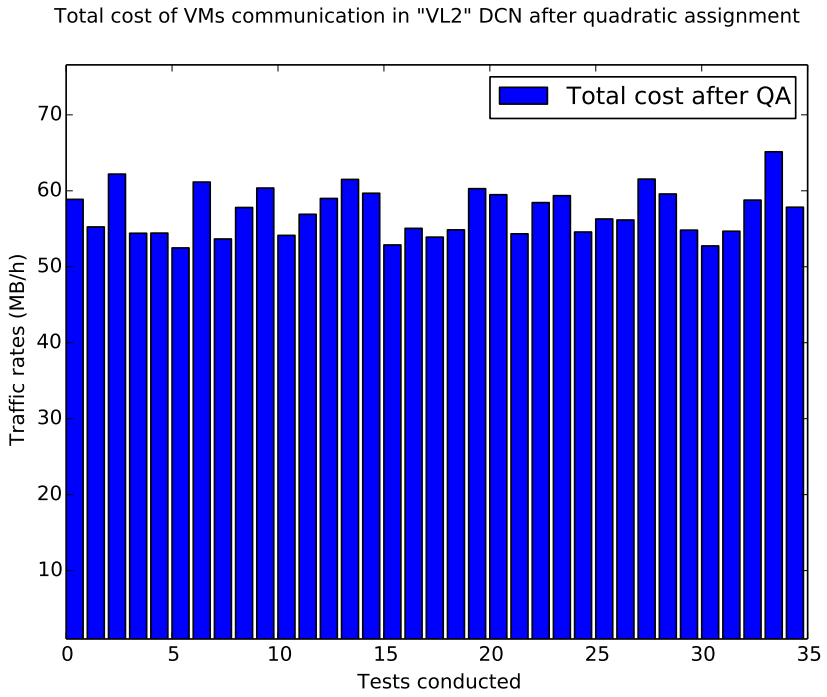


Figure 4.14: Total cost of communication in VL2 after cluster placement in set A

### 4.4.4 Intra and intercluster traffic experiment A

In order to closer observe the effect of the graph partitioning through the VM clustering algorithm a separate experiment was conducted with the same 35 randomly generated VM placement sets used in the previous 3 experiments. In the intra and intercluster traffic experiment 35 tests were executed for each of the 35 random assignment .pkl files. During each test the set of randomly assigned nodes were graph partitioned and several important values were calculated and stored such as: timestamp, file name, average aggregate intracluster communication, average aggregate intracluster communication after graph partitioning, intracluster communication of each of the 16 clusters both before and after each optimization with the VM clustering algorithm, the average aggregate intercluster communication with randomly assigned VMs, the average aggregate intercluster communication after VM clustering and the percent change.

The intercluster communications were calculated by generating every possible cluster pair combination and adding their corresponding edges found in the associated cluster matrix. The mean value of the collected

communication rates was used.

Console output example intracluster communications experiment

```
working with: ./random_placements/35_random_placement_vms.pkl
threshold similar 1020756.33985
threshold dissimilar 612453.803907
starting the graph partitioning algorithm with 414600 iterations

2015-05-09 01:39:33

0 iterations done. 0.0% finished (2015-05-09 01:39:33)
82920 iterations done. 20.0% finished (2015-05-09 01:39:36)
165840 iterations done. 40.0% finished (2015-05-09 01:39:36)
248760 iterations done. 60.0% finished (2015-05-09 01:39:37)
331680 iterations done. 80.0% finished (2015-05-09 01:39:37)
-----------------
2015-05-09 01:39:37
-------- Summary intracluster communications --------

Total average unoptimized intracluster communications: 1752782.75
Total average optimized intracluster communications: 91992765.375
percent change intracluster comm: 5148.38377004 %

-------- Summary intercluster communications --------

Total unoptimized intercluster communications: 13873148.25
Total optimized intercluster communications: 1841150.56667
percent change: -86.7286751825 %
-----------------------------------------------------
```

The average cluster communication matrices with both randomly distributed VMs and then graph partitioned VMs were calculated and stored in order to observe the intra and intercluster communications for the VM clustering analysis.

The results showed that the average intracluster traffic for the 16 clusters with randomly assigned VMs (see Fig.4.15) was 6178109.44 bytes (5.8919 MB) with standard deviation 5712565.77 bytes (5.4479 MB).

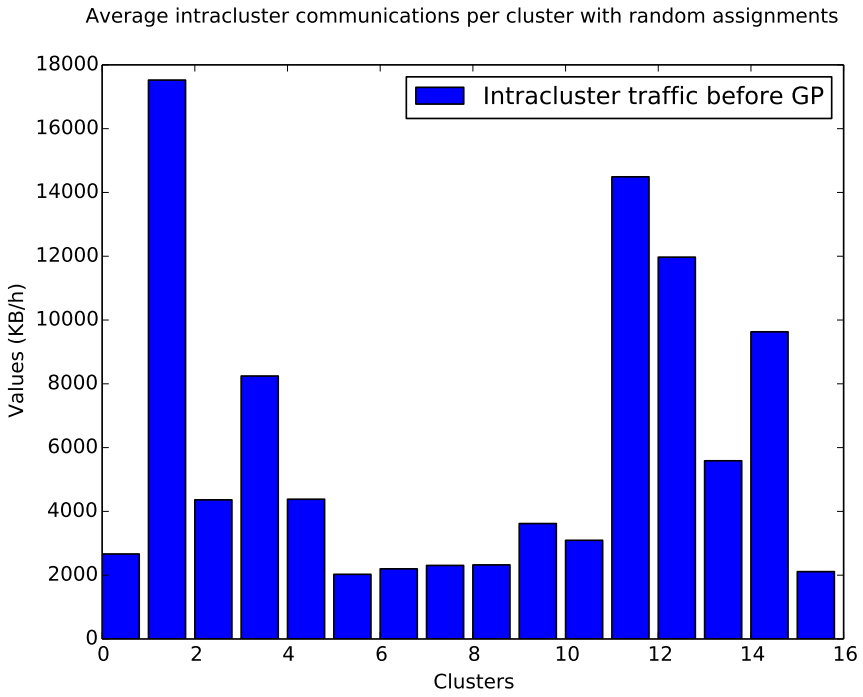


Figure 4.15: Average total intracluster traffic in 16 clusters with randomly assigned VMs in set A

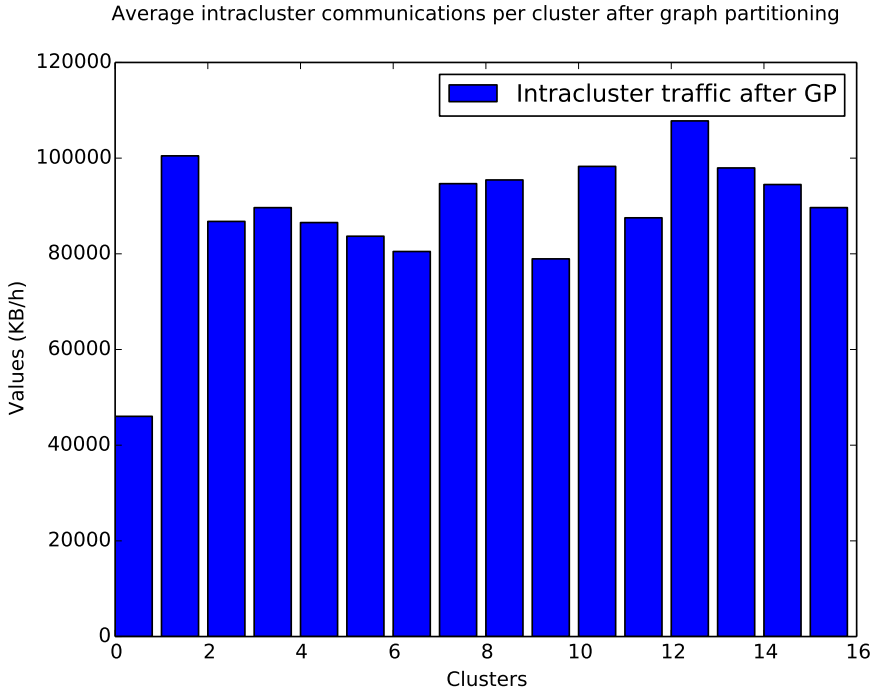


Figure 4.16: Average total intracluster traffic in 16 clusters after GP in set A

After the VM clustering algorithm optimized the clusters the average

intracluster communication for the clusters (see Fig.4.16) was shown to be 90774221.85 bytes (86.569 MB) with standard deviation 2642162.77 bytes (2.5198 MB)

## 4.5 Experiment set B

The three experiments described in the previous section and the intracluster traffic experiment were repeated once again with a new set of 1600 VMs chosen more carefully in order to test on a different type of traffic.

### 4.5.1 Experiment b1: Tree results

The experiment b1 on Tree DCN was conducted on the same 35 .pkl files used in the experiment set A. The number of maximum iterations for the VM clustering algorithm during this experiment was calculated to be 87040. The *similarity threshold* was 133695.24 and the *dissimilarity threshold* was calculated to 80217.14. In this way 1316 edges ended up over the similarity threshold while 1966 edges were below the dissimilarity threshold.

According to the results the average total cost of communication with randomly assigned VMs $T_{RandTreeB}$ was 1601453698.57 bytes (1.4915 GB) with standard deviation 18116631.36 bytes (0.0169 GB). After the VM clustering algorithm the average total cost of communication $T_{GpTreeB}$ was measured 1061026520.0 bytes (0.9882 GB) with standard deviation 12608363.74 bytes (12.0243 MB). After the cluster placement algorithm the total cost of communication $T_{QapTreeB}$ was 24244865.90 bytes (23.1217 MB) with standard deviation 373481.77 bytes (0.3562 MB).

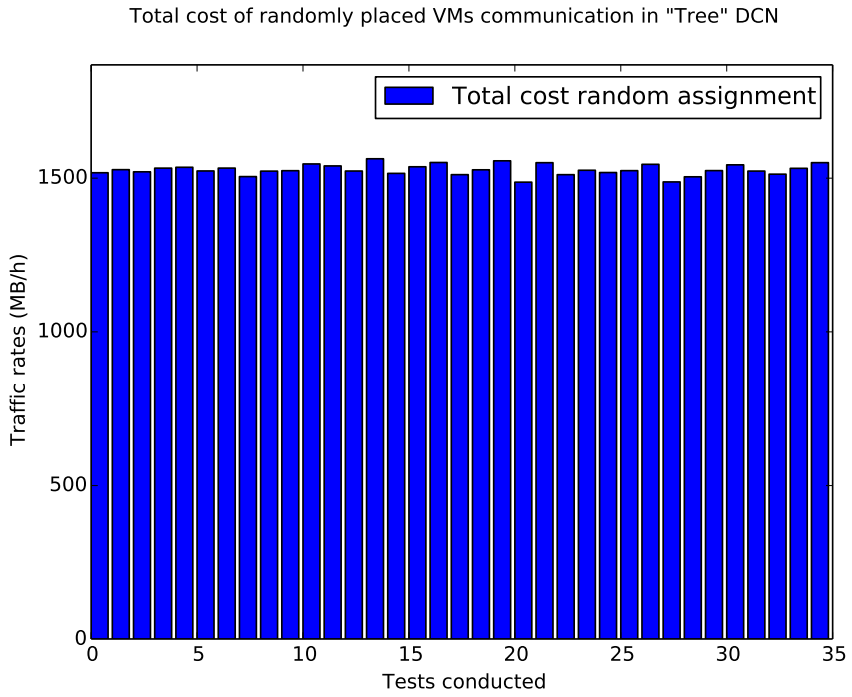


Figure 4.17: Total cost of communication in Tree with random assignments in set B

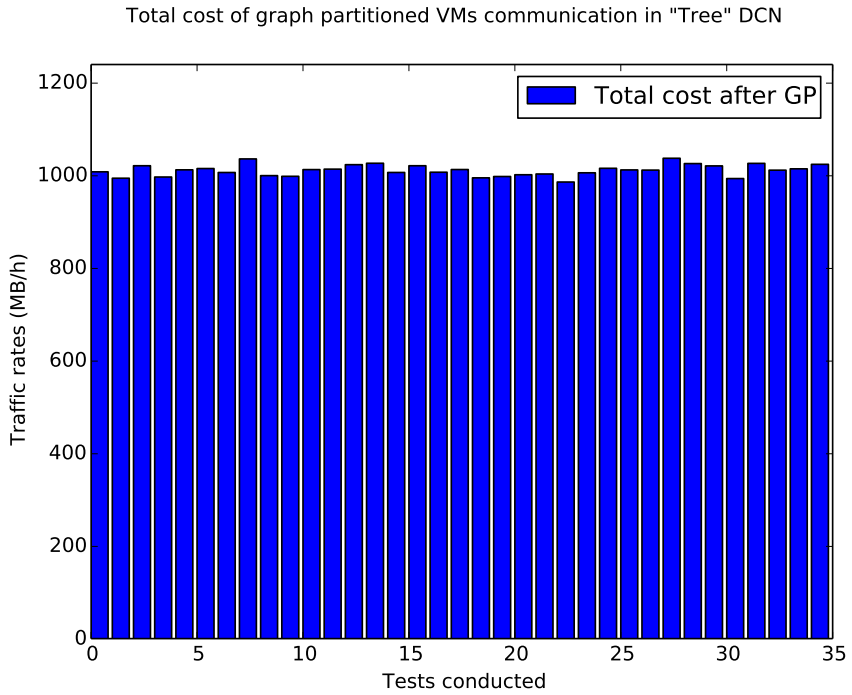


Figure 4.18: Total cost of communication in Tree after VM clustering in set B

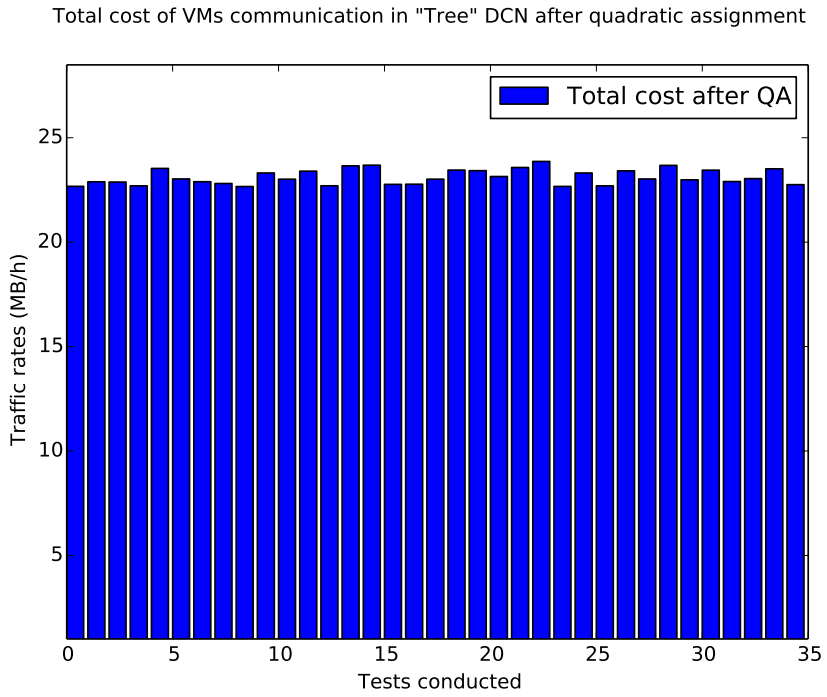


Figure 4.19: Total cost of communication in Tree after cluster placement in set B

### 4.5.2 Experiment b2: Fat-tree results

The same 35 .pkl files were used to conduct the experiment b2 on Fat-tree DCN. The same mean edge value was used in order to calculate the similarity and the dissimilarity thresholds in this experiment as in the previous b1 experiment.

According to the results the average total cost of communication with randomly assigned VMs $T_{RandFtreeB}$ was 1950752236.0 bytes (1.8168 GB) with standard deviation 17762337.15 bytes (0.0165 GB). After the VM clustering algorithm the average total cost of communication $T_{GpFtreeB}$ was measured 1288877475.49 bytes (1.2004 GB) with standard deviation 12621967.57 bytes (12.0372 MB). After the cluster placement algorithm the total cost of communication $T_{QapFtreeB}$ was 30825934.92 bytes (29.3979 MB) with standard deviation 361225.03 bytes (0.3445 MB).

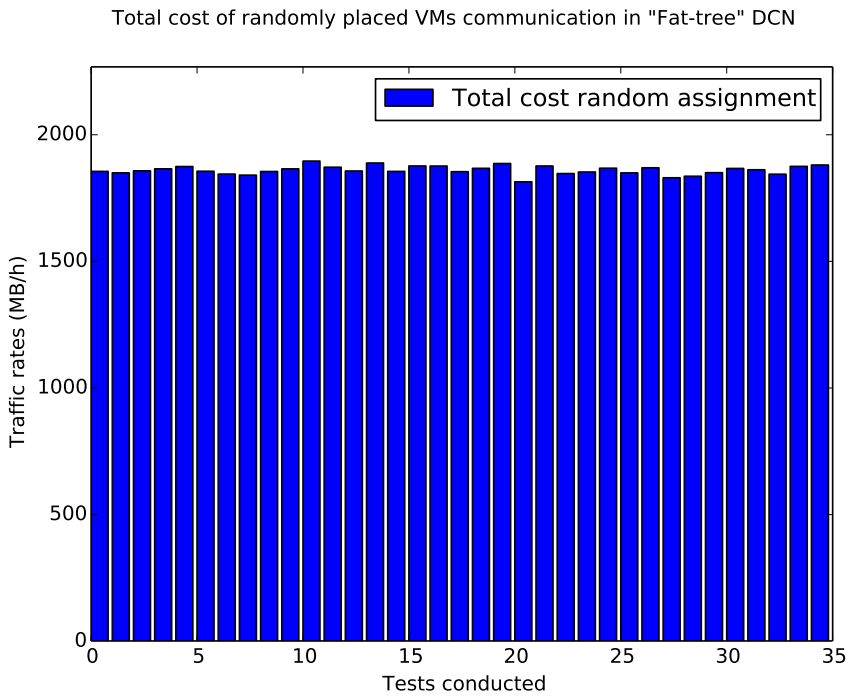


Figure 4.20: Total cost of communication in Fat-tree with random assignments in set B

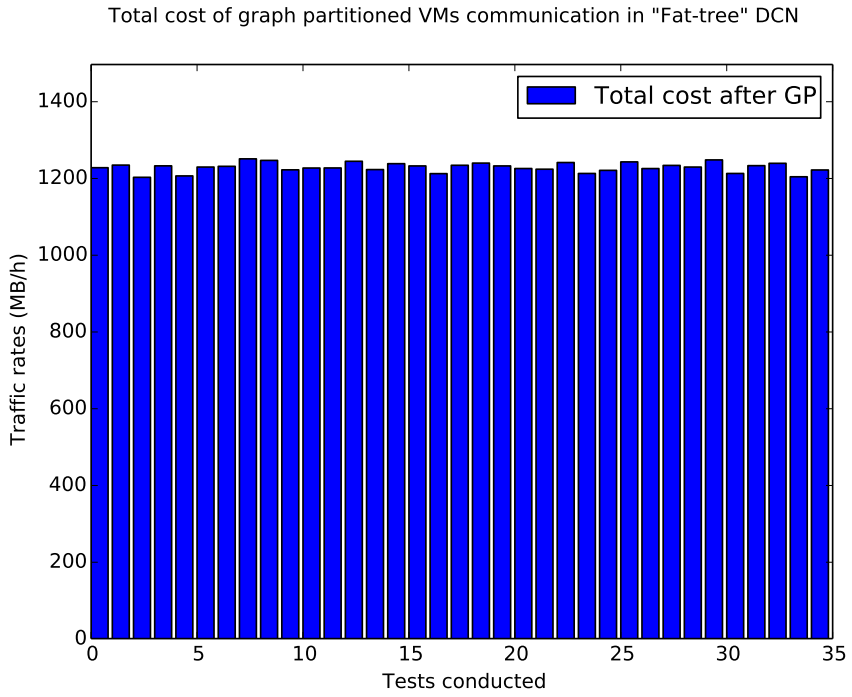


Figure 4.21: Total cost of communication in Fat-tree after VM clustering in set B

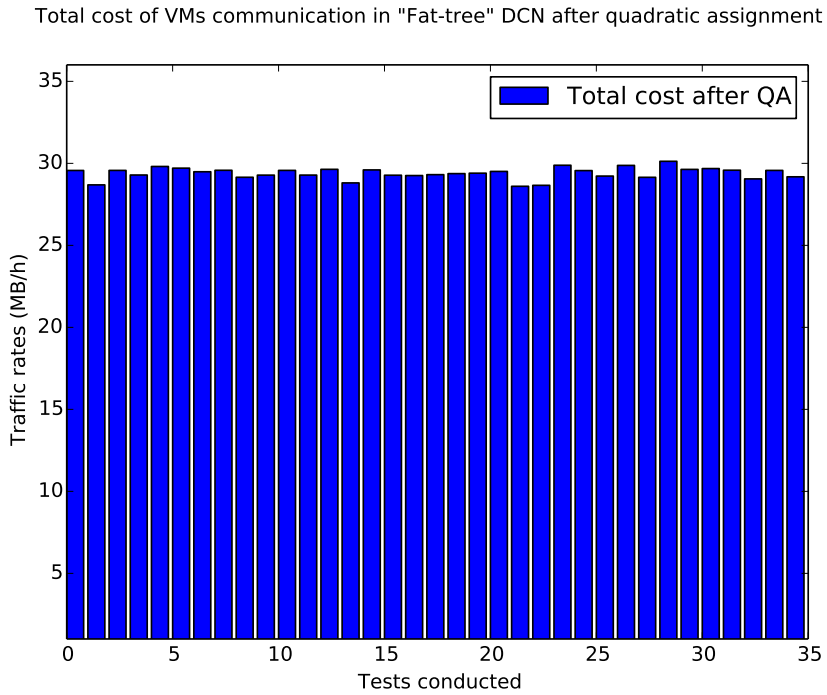


Figure 4.22: Total cost of communication in Fat-tree after cluster placement in set B

### 4.5.3 Experiment b3: VL2 results

Once again the same 35 .pkl files used in b1 and b2 were used to conduct the experiment b3 on VL2 DCN. The same mean edge value was used in order to calculate the similarity and the dissimilarity thresholds in this experiment as in the previous b1 and b2 experiments.

The results showed that the average total cost of communication with randomly assigned VMs $T_{RandVl2B}$ was 1835909748.69 bytes (1.7098 GB) with standard deviation 22100449.20 bytes (0.0206 GB). After the VM clustering algorithm the average total cost of communication $T_{GpVl2B}$ was measured 1211796514.7 bytes (1.1286 GB) with standard deviation 10158873.22 bytes (9.6883 MB). After the cluster placement algorithm the total cost of communication $T_{QapVl2B}$ was 28061769.69 bytes (26.7618 MB) with standard deviation 410242.89 bytes (0.3912 MB).

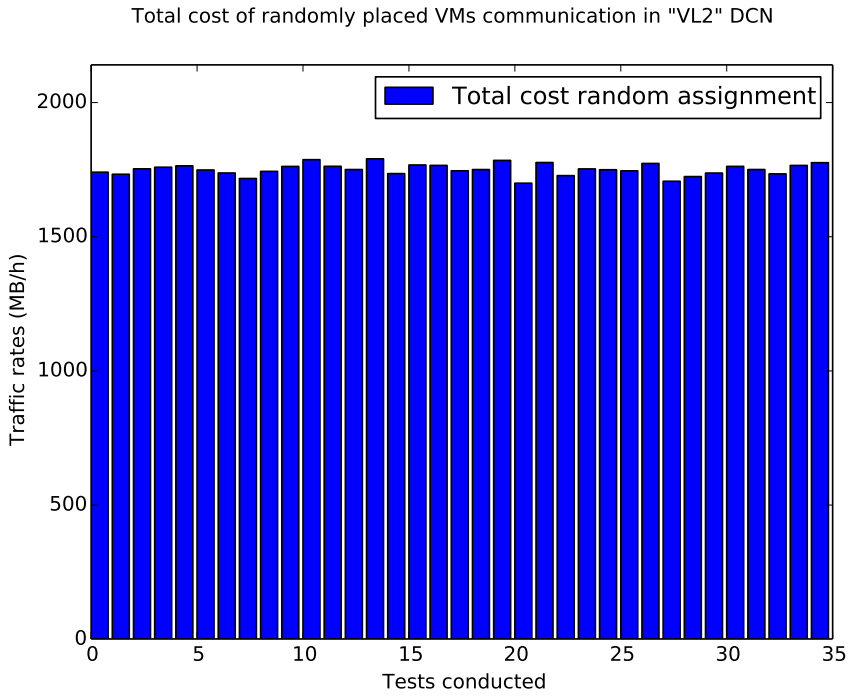


Figure 4.23: Total cost of communication in Fat-tree with random assignments in set B

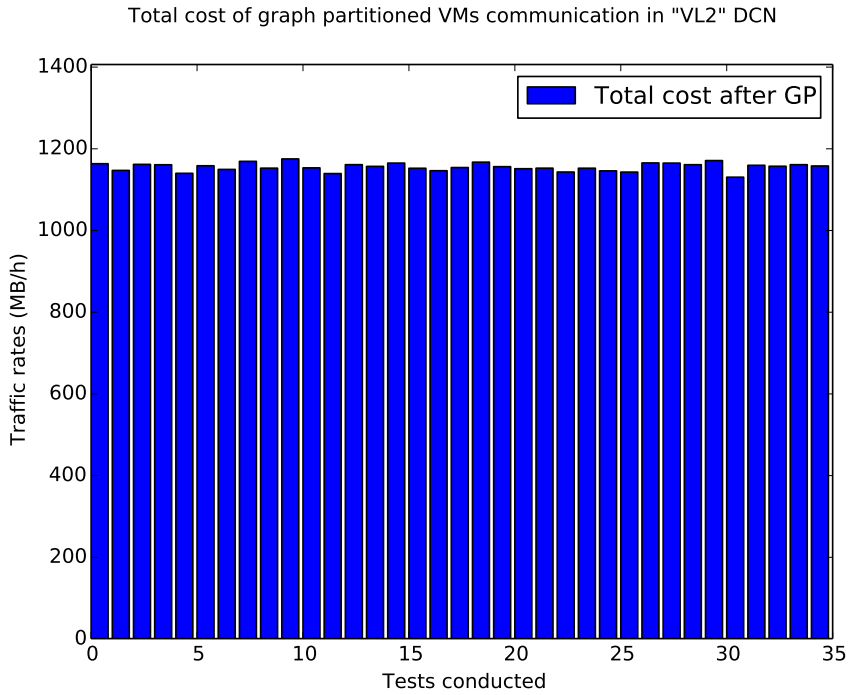


Figure 4.24: Total cost of communication in Fat-tree after VM clustering in set B

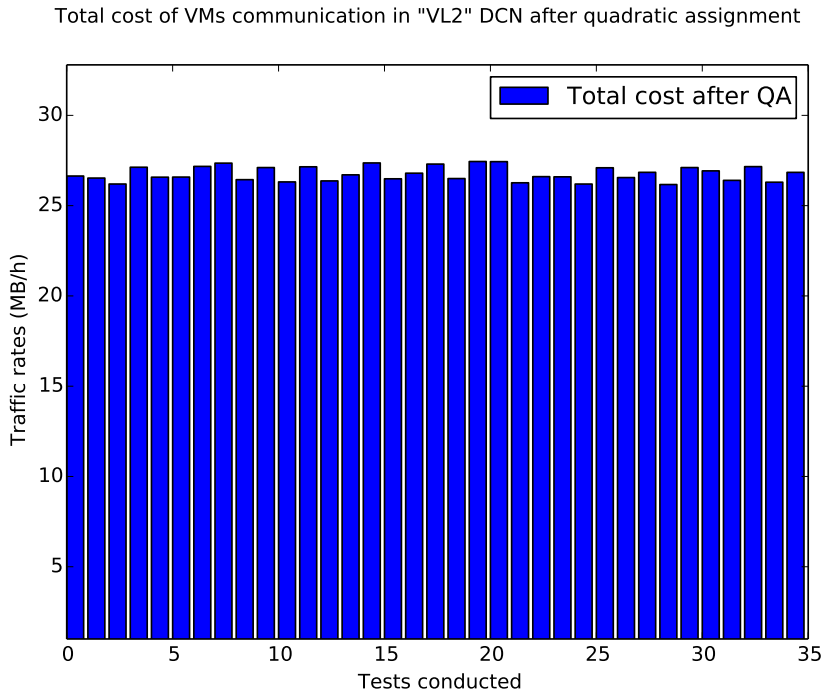


Figure 4.25: Total cost of communication in Fat-tree after cluster placement in set B

### 4.5.4 Intra and intercluster traffic experiment B

The 35 randomly generated VM placement sets used in the previous 3 experiments were used in order to observe the effect of the VM clustering algorithm on intracluster and intercluster traffic. In the intra and intercluster traffic experiment 35 tests were executed for each of the 35 random assignment .pkl files. Once again during each test the set of randomly assigned nodes were graph partitioned and the results were stored in the output files.

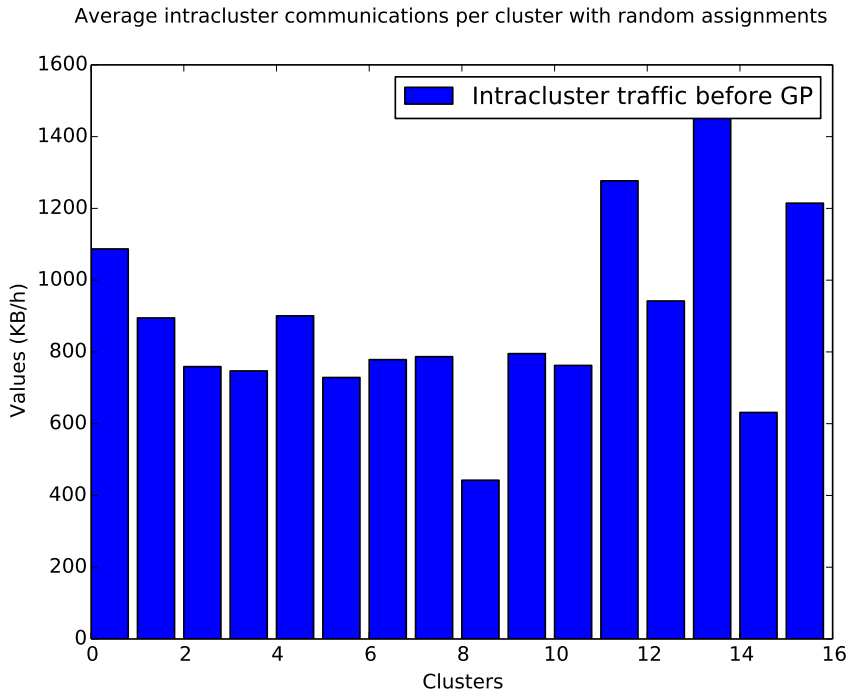


Figure 4.26: Average total intracluster traffic in 16 clusters with randomly assigned VMs in set B

.

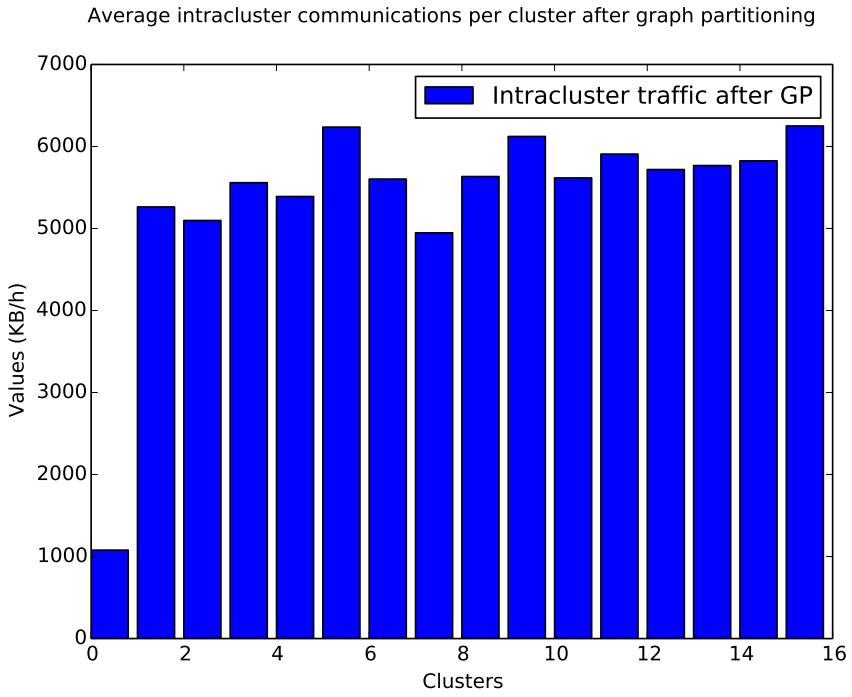


Figure 4.27: Average total intracluster traffic in 16 clusters after GP in set B

# Chapter 5

# Analysis

In this chapter the results of the implementation of VM clustering and cluster placement algorithms and the experiments on the three data center network architectures will be analyzed thoroughly by going through the plots and the statistical data obtained during the experiment.

## 5.1 VM clustering and cluster placement: set A

The results showed that the traffic-aware consolidation of the VMs had significant impact on the total cost of communication. The data also shows that the cluster placement algorithm further decreases the total cost of communication. This section goes through and analyzes the results of each of the three experiments by using the statistical data and the data visualization provided by the analysis tools developed in Python.

### 5.1.1 Experiment a1: Tree analysis

When looking at the baseline total cost of communication the first noticeable thing is the high variance (std is 12.11% of the mean) in the distribution of mean total costs (see Fig.5.1 and 4.6). This is can be explained by the fact that the cost matrix for the Tree DCN (see Fig.3.8) can cause higher variation in the cost of communication as the result of moving clusters with significantly high traffic slightly away or closer to their pairs with whom they exchange significantly high traffic. The cost matrices for Fat-tree and VL2 (see Fig.3.9 and 3.10) are more uniform in comparison.

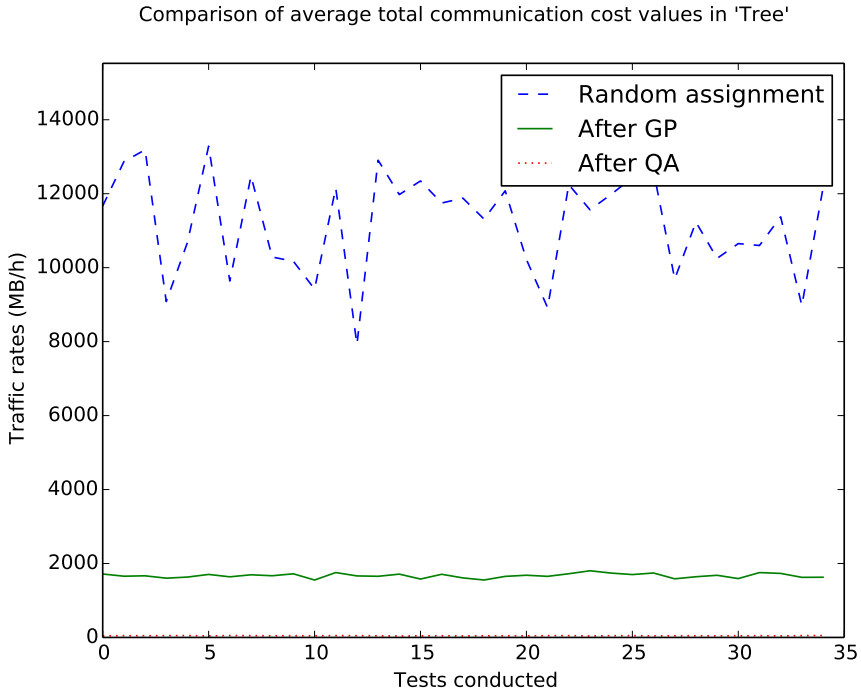


Figure 5.1: Total cost of communication in Tree in set A

The impact of the VM clustering algorithm using Oommen's graph partitioning technique is obvious when observing the plotted graphs. VM consolidation decreases the total cost of communication with 85.09% (from 11744193654.7 to 1751027376.48 bytes) and stabilizes the variance as well (see Fig.5.1 and 4.7). At this point the clusters are not assigned to the racks in the most optimal way yet. After the cluster assignment with simulated annealing the total cost of communication drops further to 49621895.55 bytes which is 97.17% decrease compared to the total cost of the consolidated (clustered) VMs and 99.58% overall decrease compared to the total cost of communication of the randomly distributed VMs.

| | mean | st.dev | $\Delta_{Prev.mean}$ | $\Delta_{Overall}$ |
|---|---|---|---|---|
| $T_{RandTreeA}$ | 11744193654.7 | 1422547350.22 | — | — |
| $T_{GpTreeA}$ | 1751027376.48 | 62550001.98 | -85.09% | -85.09% |
| $T_{QapTreeA}$ | 49621895.55 | 3293752.44 | -97.17% | -99.58% |

Table 5.1: Change in the total cost of communication in Tree in set A

During the 82920 iterations the VM clustering algorithm invoked the *RewardSimilarNodes* procedure on average 5622 times, while the *PenalizeSimilarNodes* was invoked 1576 times on average and the average number of the times *PenalizeDissmiilarNodes* was invoked was 2213. The graph partitioning process took on average 3.4 seconds to complete.

This means that for most of the time algorithm picked nodes that were not significantly connected. This occured in 9411 out of total 82920 iterations. Thus, the graph partitioning algorithm was "idle" 88.65% of the total iterations when the picked edges were regarded neither similar nor dissimilar.

The data also shows that the cluster placement algorithm used the inferior configurations 149.4 times on average (standard deviation 16.8) out of 100000 iterations. It took the simulated annealing process on average 37.7 seconds to complete the 100000 iterations which is significantly longer time compared to what the VM clustering algorithm used. This can be explained by the fact that the VM clustering algorithm for most of the time (88.65%) didn't have to conduct any time consuming operations whereas the cluster placement algorithm had calculation jobs to execute for each of the 100000 iterations.

### 5.1.2 Experiment a2: Fat-tree analysis

The results of the Fat-tree experiment reveal (see Fig.5.2 and 4.10) that the baseline total cost of communication in this DCN model was more stable compared to the baseline total cost of communication for the Tree in terms of the variance. The standard deviation of the 35 tests is on average 8.04% of the mean. This can be explained by the cost matrix for the Fat-tree (see Fig.3.9) which is relatively uniform compared to the cost matrix of Tree which might be a good explanation for why there is less variation in total cost as there is no difference caused in the cost of communication between two VMs if one of the VMs is moved from one rack to another, when the pair is already communicating to each other from the server rack groups belonging to distinct access layer switches. There's greater change in cost of communication for VM pairs migrated from one rack to another in cases of

the VMs communicating to each other within the same access layer switch rack group (see Fig.3.9)

The total cost of communication measured with the randomly distributed VMs decreased significantly after the set of nodes was graph partitioned by the VM clustering algorithm. The average total cost was reduced with 85.14% from 14406082632.7 to 2140643564.18 bytes.

The VM clustering algorithm executed 82920 iterations and invoked *RewardSimilarNodes* procedure on average 5619 times. The *PenalizeSimilarNodes* procedure was invoked 1582 times on average while the *PenalizeDissimilarNodes* was invoked on average 2189 times. The average time used no the graph partitioning was 3.33 seconds. The VM clustering algorithm behaved in the same way as during the Tree analysis as expected. In this experiment too it was idle most of the time as the majority of the randomly picked VM pairs weren't considered either similar or dissimilar. The algorithm was busy 11.32% of the time rewarding and penalizing the nodes. 40.16% of the picked VMs were penalized while the remaining 59.84% were rewarded.

The cluster placement algorithm was executed after the VM consolidation which further decreased the average total cost of communication with -96.82% from 2140643564.18 to 68162675.71 bytes. Totaly the average total cost was decrease with 99.52% from the initial 14406082632.7 to the 68162675.71 bytes after the cluster placement.

| | mean | st.dev | $\Delta_{Prev.mean}$ | $\Delta_{Overall}$ |
|---|---|---|---|---|
| $T_{RandFtreeA}$ | 14406082632.7 | 1158102049.49 | — | — |
| $T_{GpFtreeA}$ | 2140643564.18 | 74675979.95 | -85.14% | -85.14% |
| $T_{QapFtreeA}$ | 68162675.71 | 3101849.88 | -96.82% | -99.52% |

Table 5.2: Change in the total cost of communication in Fat-tree in set A

The cluster placement algorithm accepted on average 1733.76 worse configurations during the simulated annealing process which took 38.85

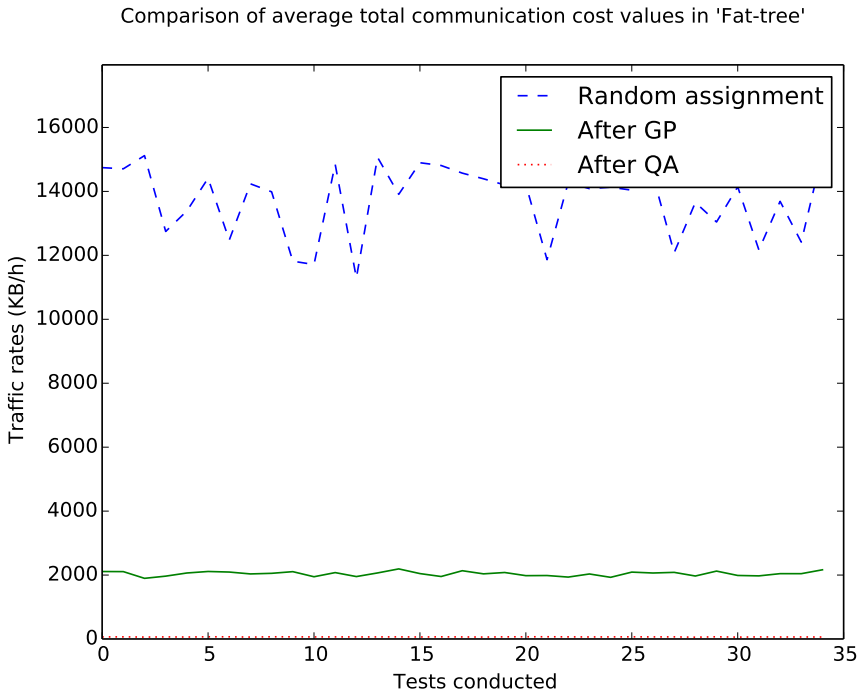


Figure 5.2: Total cost of communication in Fat-tree in set A

seconds to complete on average.

### 5.1.3 Experiment a3: VL2 analysis

The observed average total communication cost for the randomly distributed VMs in the VL2 was 13586759288.7 with standard deviation 1414884915.62. Variance was higher compared to Fat-tree but lower compared to Tree as it is the 10.41% of the mean. This result can be explained again by comparing the cost matrices of the three data center architecture models displayed in the approach section.

| | mean | st.dev | $\Delta_{Prev.mean}$ | $\Delta_{Overall}$ |
|---|---|---|---|---|
| $T_{RandVl2A}$ | 13586759288.7 | 1414884915.62 | — | — |
| $T_{GpVl2A}$ | 2010382164.52 | 53688282.77 | -85.20% | -85.20% |
| $T_{QapVl2A}$ | 59993636.63 | 3292518.29 | -97.02% | -99.56% |

Table 5.3: Change in the total cost of communication in VL2 in set A

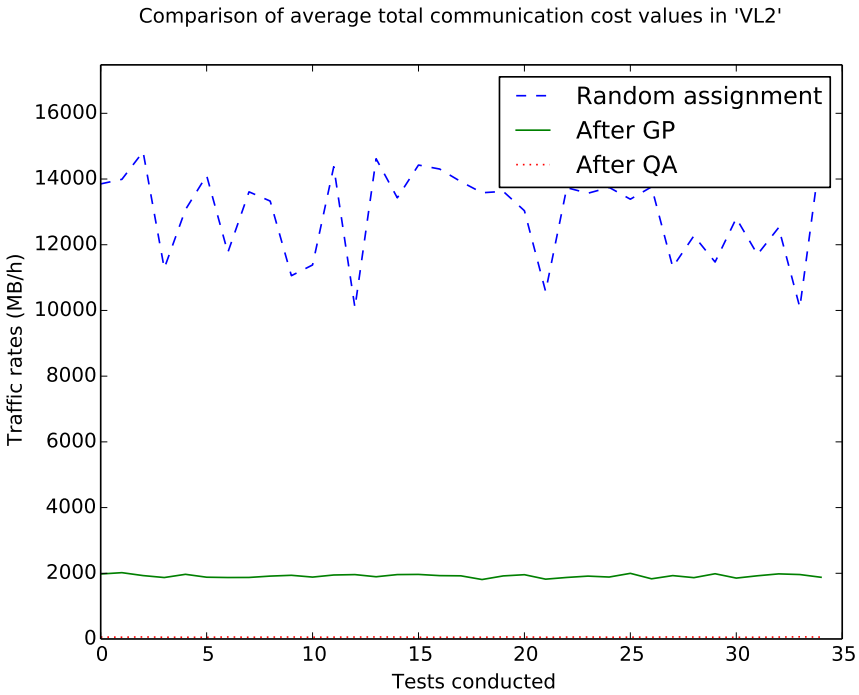


Figure 5.3: Total cost of communication in VL2 in set A

The average total cost of communication decreased with 85.20% after VM clustering from 13586759288.7 to 2010382164.52 bytes. During the VM clustering the graph partitioning process invoked *RewardSimilarNodes* on average 5619 times. The *PenalizeSimilarNodes* procedure was invoked 1581 times and the *PenalizeDissimilarNodes* procedure was invoked 2204 times on average. The graph partitioning process took average of 3.34 seconds to complete.

The cluster placement algorithm further decreased the total cost of communication with 97.02% compared to the total cost of communication achieved after the graph partitioning. The new total cost went down to average of 59993636.63 bytes which, compared to the initial total cost of communication was a 99.56% decrease. The simulated annealing took on average 39.6 seconds while 1409 times out of 100000 the algorithm chose an inferior state.

## 5.2 Intracluster and intercluster communication: set A

It is clear that the traffic-aware consolidation of the VMs had significant impact on the total cost of communication which was greatly decreased through both VM clustering and quadratic assignment. In order to understand what caused this significant improvement it's important to observe the changes in the traffic between the VMs inside the clusters (intracluster traffic) and also the change in traffic exchanged between (intercluster traffic) the clusters.

The data collected during the intracluster and intercluster experiments shows clearly how the intracluster traffic was increased as a result of the graph partitioning process conducted by the VM clustering algorithm. The figure 5.4 illustrates the average intracluster communications inside the 16 clusters before the VM consolidation when the clusters were populated by randomly distributed VMs for the 35 randomly generated VM sets.

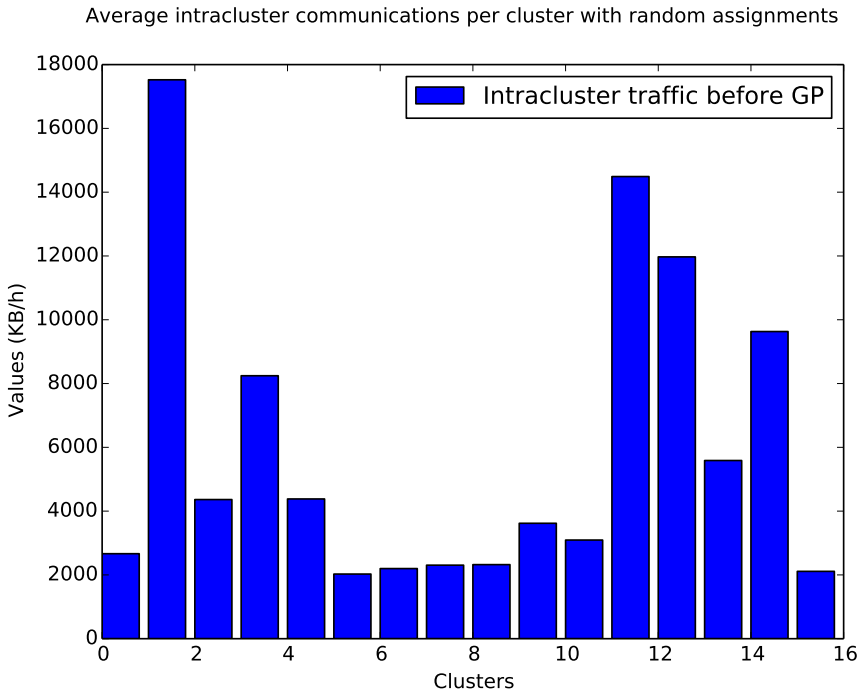


Figure 5.4: Intracluster traffic in the 16 clusters before GP in set A

The figure 5.5 illustrates how the intracluster traffic looked in the same

16 clusters shown in the figure 5.4 after the VMs were consolidated with the VM clustering algorithm using the graph partitioning technique.

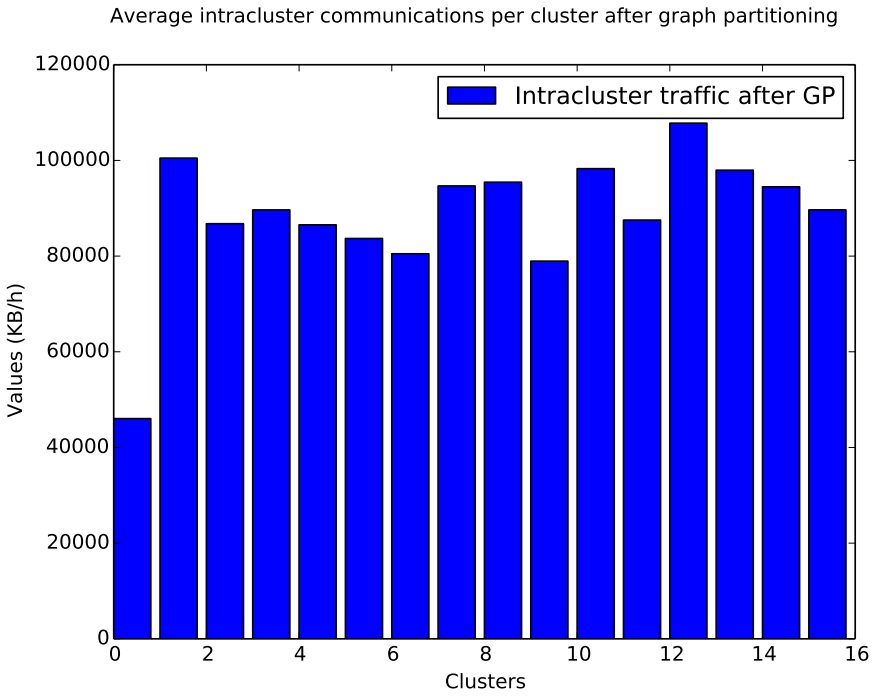


Figure 5.5: Intracluster traffic in the 16 clusters after GP in set A

With the randomly assigned VMs the aggregate average intracluster communication was 6178109.44 bytes (std: 5712565.77) which was increased with 1369.28% up to 90774221.85 bytes after the graph partitioning. Already at this stage even without intelligent assignment of the clusters to the available server racks the data center traffic is significantly optimized compared to the state prior to consolidating the virtual machines with the VM clustering algorithm. This effect is observed consistently through all the 35 tests conducted for the 35 randomly generated VM sets (1225 times totally) which indicates the stable and reliable performance of the VM clustering algorithm.

While the intracluster traffic increased the traffic between the clusters decreased at the same time. The average initial aggregate intercluster traffic was measured 13283104.69 bytes and the traffic between the clusters was not optimized as shown in the figure 5.7, where it's evident that clusters are communicating with each other in a more or less chaotic way with vari-

able traffic rates. The average aggregate intercluster traffic decreases with 84.92% to 2003623.03 bytes after the VM clustering as the result of consolidating highly communicative VMs in the same clusters.

This is illustrated by the figure 5.6. The cells diagonally represent the traffic inside the clusters (intracluster communication) while all the other cells refer to the traffic between (intercluster communication) the 16 clusters. The colors correspond to the values each cells represent. The light blue colors represent low values while the values get greater as the cell color gets darker and darker shade of blue.

The figure 5.6 shows for example that the cell in row 1 and column 13 is considerably darker blue compared to the neighboring cell in row 1 and column 13. This cell represents the traffic between clusters 1 and 13. The figure reveals that on average 4 clusters had especially high mutual traffic compared to the rest of the clusters. The diagonal of the figure 5.6 reveals that none of the clusters has high internal traffic judging by the light blue color of the diagonal cells.

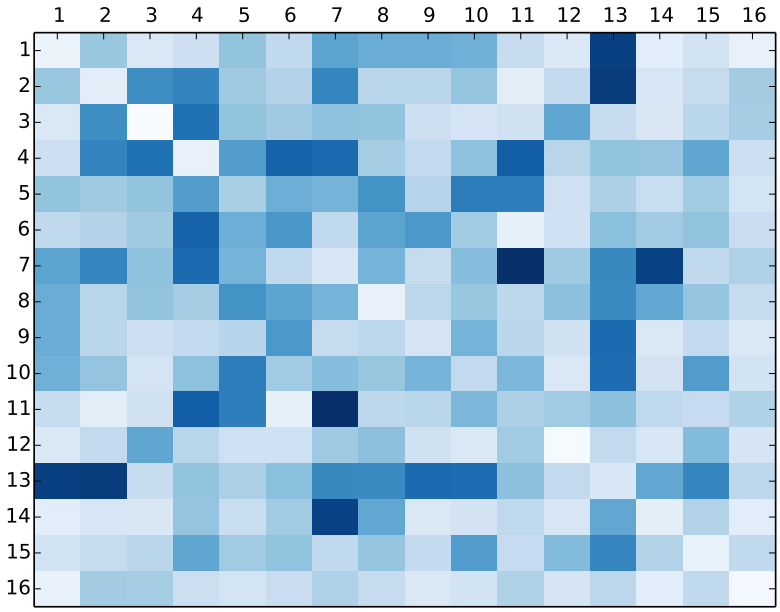


Figure 5.6: Intra and intercluster traffic heatmap before GP in set A

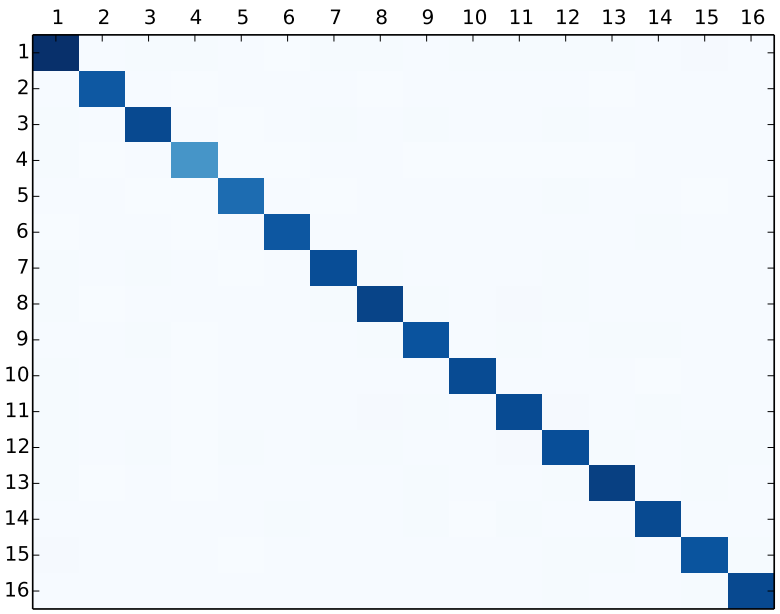


Figure 5.7: Intra and intercluster traffic heatmap after GP in set A

The figure 5.7 shows significantly different picture after VM clustering. The diagonal cells representing the intracluster traffic are dark blue signifying the high values whereas the rest of the cells are of much lighter color. The non-diagonal graph is smooth and uniform signifying the decrease in intercluster communication over the whole matrix.

### 5.2.1 Overall comparison: set A

As shown in the previous sections due to the VM clustering algorithm consolidating VMs with high mutual traffic in the same clusters the intracluster communication increased with 1369.28% while the intercluster traffic decreased with 84.92% at the same time. These changes caused the decrease of the total cost of communication with with with 97.17% in Tree, with 96.82% in Fat-tree and with 97.02% in VL2. The smart assignment of the clusters to the server racks with the use of the simulated annealing implemented in the cluster placement algorithm further decreased the total cost of communication with 99.58% in Tree, 99.52% in Fat-tree and with 99.56% in VL2 data center network architecture models. The figure 5.8 illustrates the total cost of communication with randomly assigned VMs, after

VM clustering and after cluster placement in all three data center network architecture models experimented on in this project.

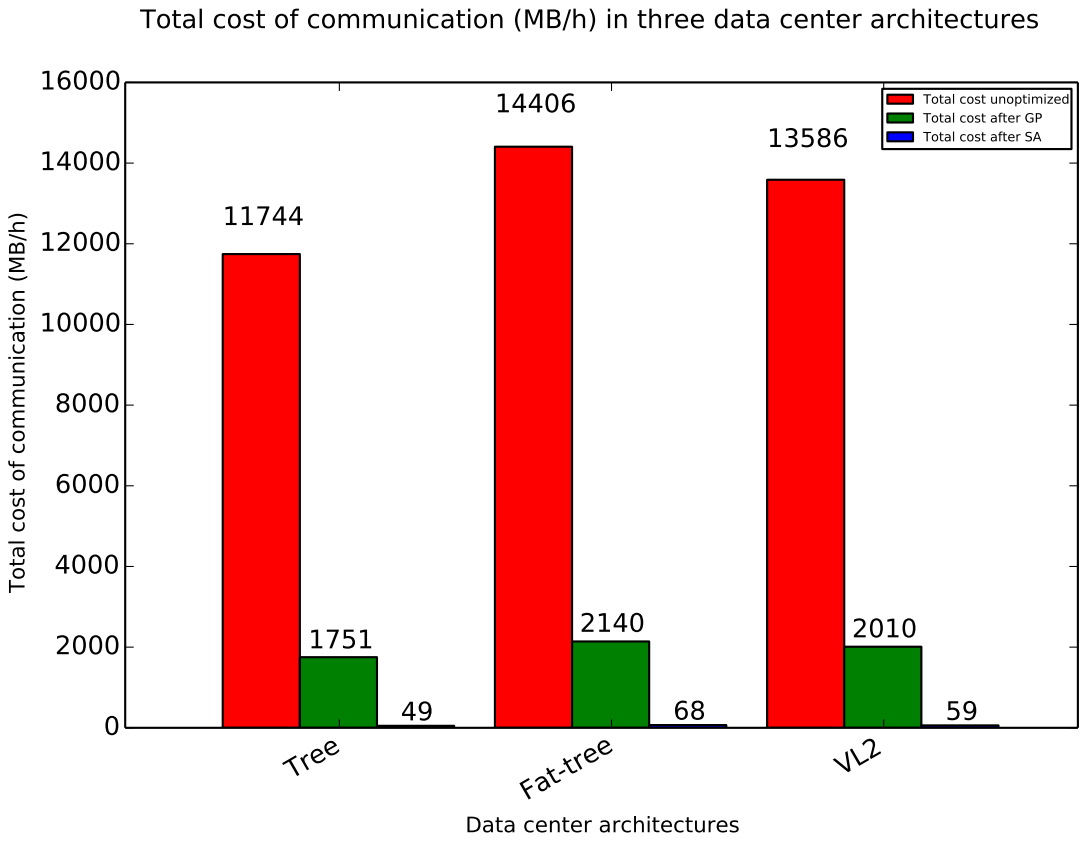


Figure 5.8: Total cost of communication in all three experiments in set A

During all the three experiments the effect of the VM clustering and the cluster placement was shown to be rather effective in consolidating the strongly connected nodes in the same clusters and ultimately greatly decreasing the total cost of communication in the data center models. The optimization results were stable and consistent in all the tests conducted.

### 5.2.2 Traffic matrix characteristics: set A

Deeper analysis of the traffic matrix used to conduct all three experiments shows that not only are the 2555854 out of 2560000 (99.84% of the total) values equal to zero in the matrix but the remaining 4146 are rather unevenly distributed.

The figure 5.9 shows the distribution of the 4146 nonzero elements (or the edges between the communicating VMs) of the traffic matrix. The graph is extremely skewed meaning that there are very few high values while most of the edges are considerably lower. Thus the mean edge is far apart from the median value and therefore merely 8.68% (360 edges) of the total number of the edges end up over the similarity threshold calculated by using the mean edge value while the majority (87.51%) of the edges, which is 3628 values, end up below the dissimilarity threshold.

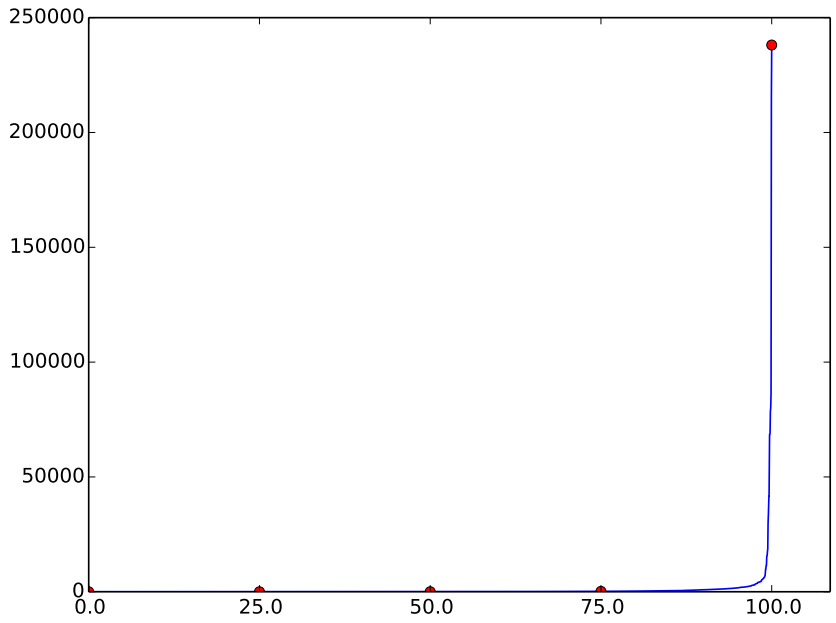


Figure 5.9: All edge values in set A in the traffic matrix in 25% percentiles

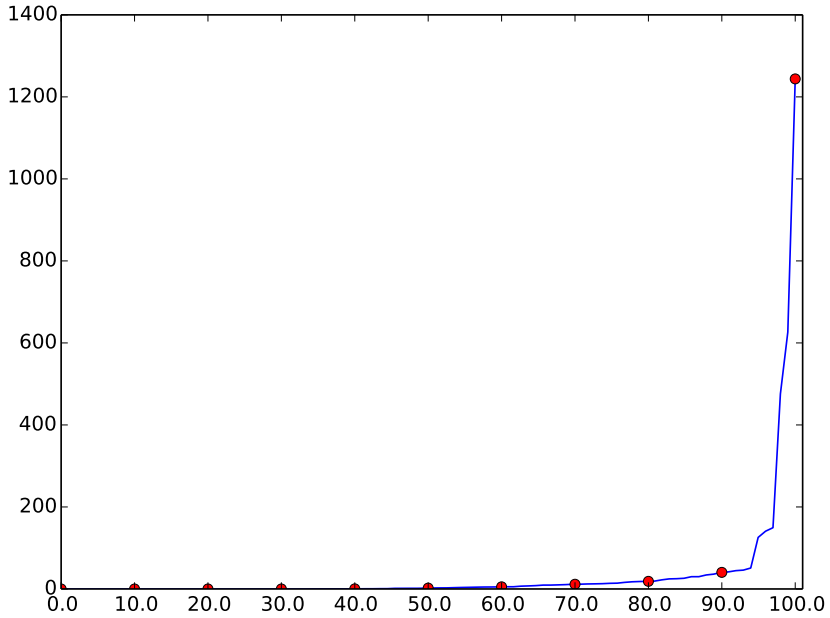


Figure 5.10: Top 100 edge values in set A in the traffic matrix shown with 10% percentiles

## 5.3 Experiment set B: analysis

As discussed in the approach section a different set of 1600 VMs was used in order to conduct experiment B otherwise with the same parameters. The purpose of this was to intentionally experiment on the set of VMs with a "smoother" communication patterns. The figures 5.9 and 5.10 show that the mere 10% of the edges constituted for most of the traffic in the set while the remaining 90% of the edges were significantly lower values. This means that there are very few VMs who communicate immensely with one another while the rest of the VMs have moderate mutual communication.

The set B was intentionally picked to exclude the VMs with extremely high intercommunication compared to the rest of the nodes. The purpose of this method was to test how the developed algorithms would perform in a different environment.

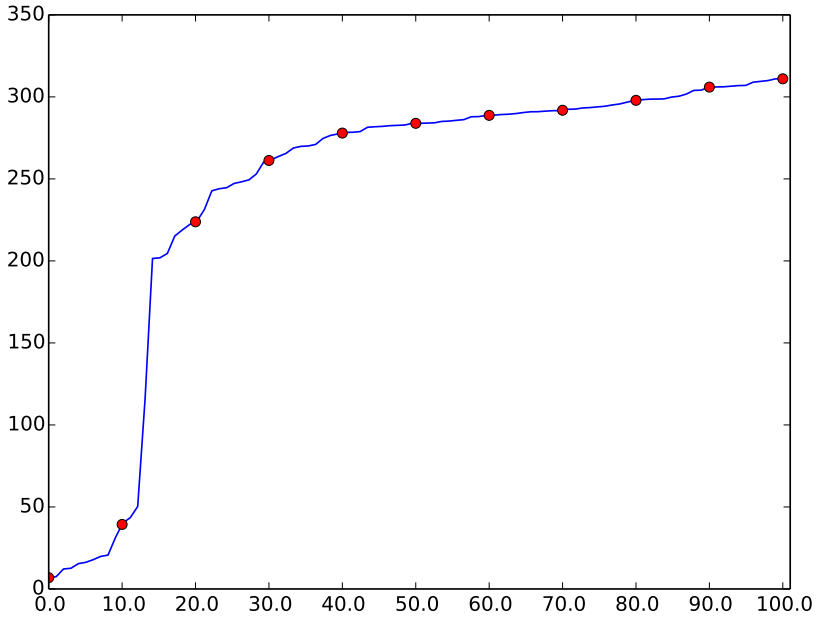


Figure 5.11: Edge values in the traffic matrix in set B shown with 10% percentiles

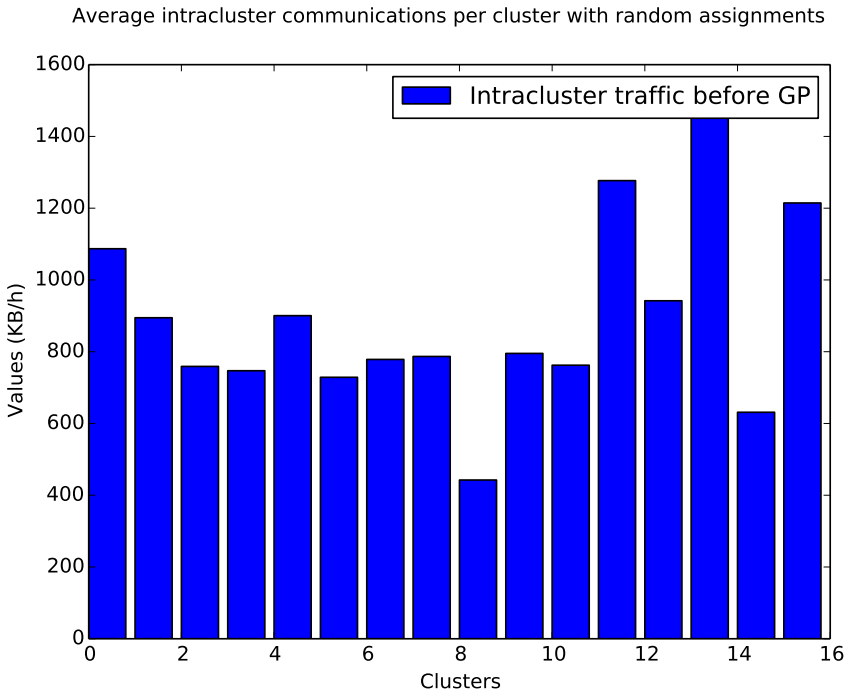


Figure 5.12: Intracluster traffic in the 16 clusters before GP in set B

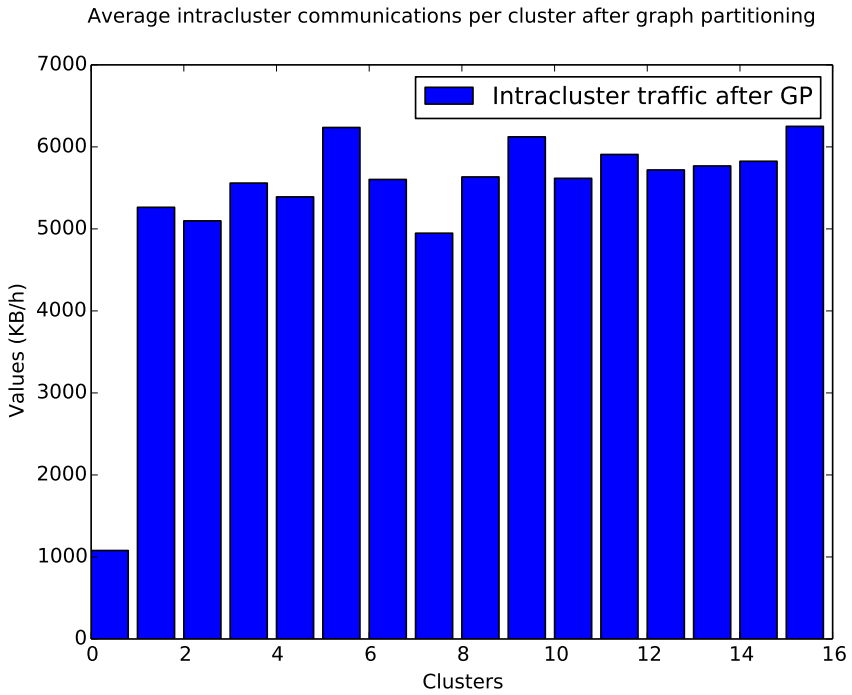


Figure 5.13: Intracluster traffic in the 16 clusters after GP in set B

The figure 5.11 shows the distribution of the edges in the symmetric traffic matrix for the experiment set B. It is evident that the there was a more even distribution of edges in this experiment.

Due to the fact that the communication between the communicating VM pairs is much more evenly distributed compared to the experiment set A the increase in the intracluster traffic and the decrease in the intercluster traffic is less compared to the experiment set A. The results showed that the intracluster traffic increased with 502.18% (from 914155.61 to 5504944.75 bytes) while the intercluster traffic decreased with 33.67% (from 1817584.94 to 1205479.72 bytes).

As a result the decrease in total cost of communication after VM clustering was also moderate compared to the results seen in the experiment set A. Total cost of communication for the Tree experiment (b1) after VM clustering decreased with 33.74%. After cluster placement algorithm the total cost of communication decreased with 98.48%. The VM clustering algorith decreased the total cost of communication with 33.92% in the Fat-tree experiment (b2) and with 33.99% in the VL2 experiment (b3). The cluster

placement algorithm improved the results with 98.41% in the Fat-tree (b2) and with 98.47% in the VL2 (b3) experiments.

Average number of *RewardSimilarNodes* invoked increased to 13306.48 in the Tree experiment (b1). Average number of *PenalizeSimilarNodes* was 13013.23 while the average number of *PenalizeDissimilarNodes* was 868.82. During the Fat-tree experiment (b1) the number of *RewardSimilarNodes* was 13330.65 while the average number of *PenalizeSimilarNodes* was 12994.87 and the *PenalizeDissimilarNodes* was 873.34. The average number of the *RewardSimilarNodes* increased during the VL2 experiments (b3) as well and was 13296.16 while the average number of *PenalizeSimilarNodes* invoked was 13022.85. The average number of *PenalizeSimilarNodes* invoked was 875.69.

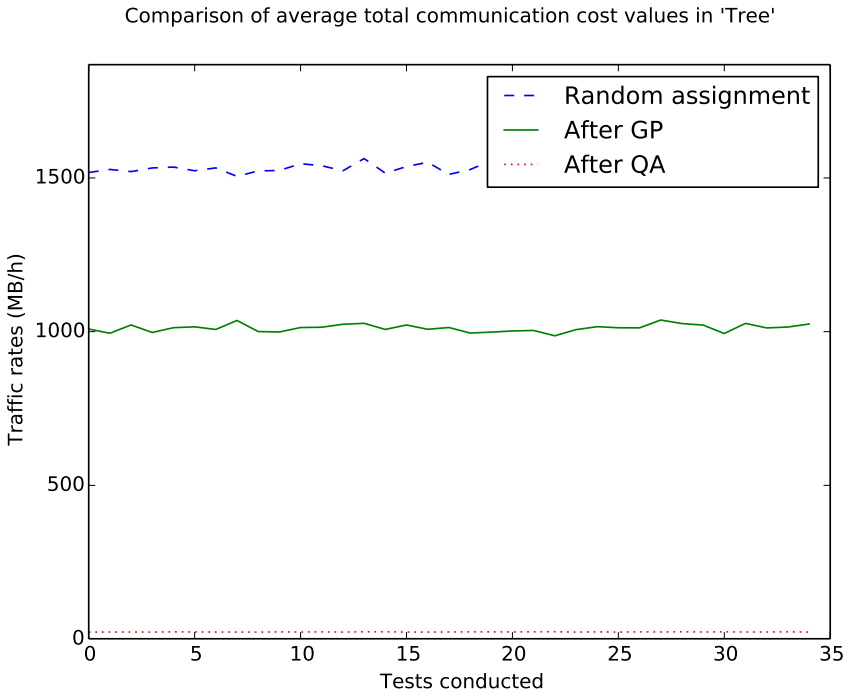


Figure 5.14: Total cost of communication in Tree in set B

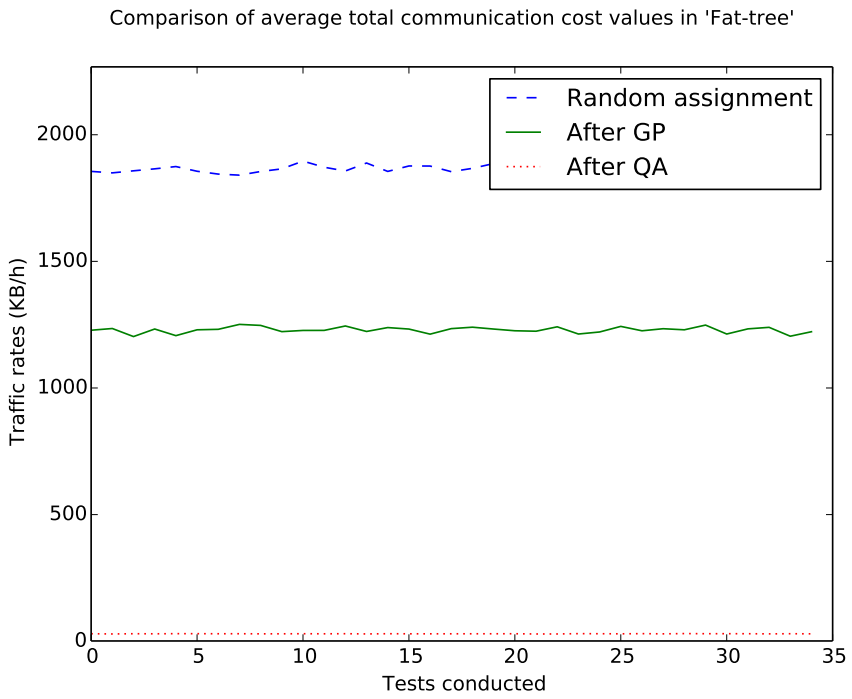


Figure 5.15: Total cost of communication in Fat-tree in set B

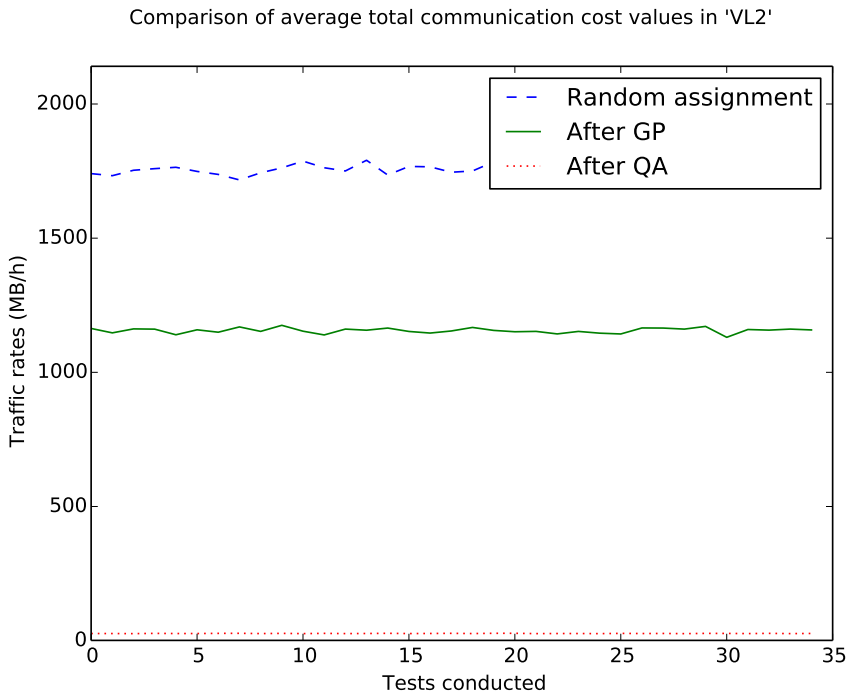


Figure 5.16: Total cost of communication in VL2 in set B

The figures 5.14, 5.15 and 5.16 show the average total cost of communi-

cation in the Tree, Fat-tree and VL2 experiments with set B with randomly assigned VMs, after graph partitioning with the VM clustering algorithm and after the cluster placement algorithm. The figures demonstrate the positive effect of graph partitioning and further improvement in total cost of communication after quadratic assignment with the use of the developed cluster placement algorithm.

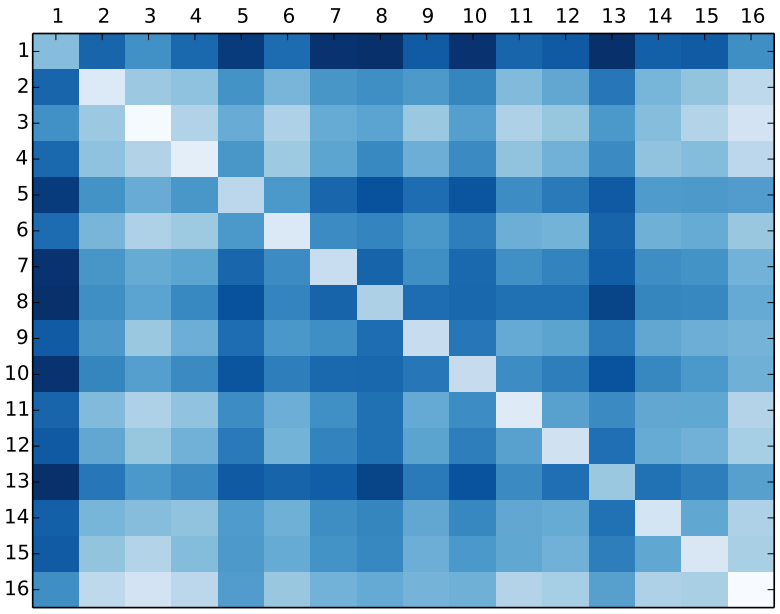


Figure 5.17: Intra and intercluster traffic heatmap before GP in set B

The heatmap plots of the intra and intercluster communications demonstrate how the traffic patterns look before and after graph partitioning with the VM clustering algorithm in the experiment set B. The figure 5.17 shows that the traffic rates between the clusters is on average higher compared to the experiment set A, while the intracluster communication (the diagonal cells) is considerably less. The figure 5.18 shows how the VM clustering algorithm optimizes the traffic. The diagonal cells show increase in intracluster traffic while the intercluster traffic is reduced.

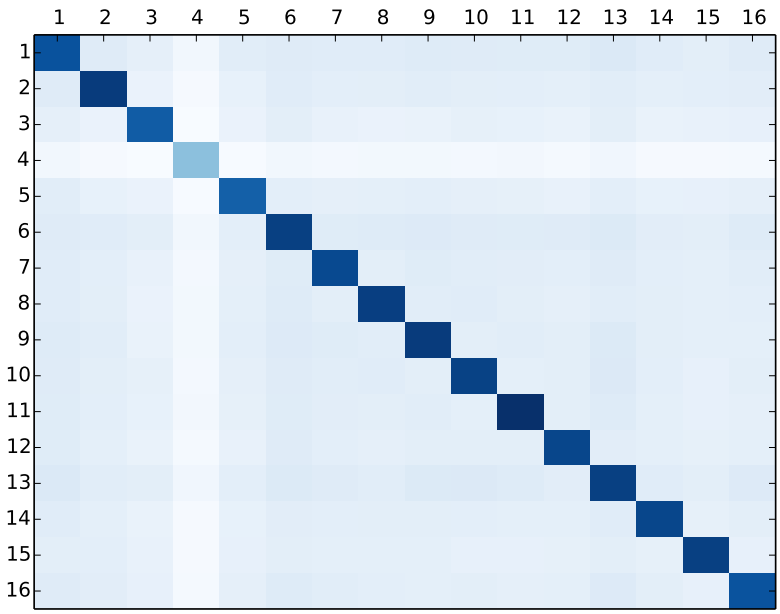


Figure 5.18: Intra and intercluster traffic heatmap after GP in set B

The figure 5.19 illustrates how the total cost of communication was gradually decreased first with the use of the VM clustering algorithm and after that with the use of the cluster placement algorithm. It is obvious that the VM clustering with graph partitioning had considerable effect on the total cost of communication. However, due to the nature of the data set used in the experiment set B the improvement was less compared to the improvement achieved in the experiment set A.

The table 5.4 shows how the VM clustering and the cluster placement algorithms reduced the total cost of communication in Tree (b1), Fat-tree (b2) and VL2 (b3) experiments for the experiment set B.

| | mean | st.dev | $\Delta_{Prev.mean}$ | $\Delta_{Overall}$ |
|---|---|---|---|---|
| $T_{RandTreeB}$ | 1601453698.57 | 18116631.36 | — | — |
| $T_{GpTreeB}$ | 1061026520.0 | 12608363.74 | -33.75% | -33.75% |
| $T_{QapTreeB}$ | 24244865.90 | 373481.77 | -97.71% | -98.49% |
| $T_{RandFtreeB}$ | 1950752236.0 | 17762337.15 | — | — |
| $T_{GpFtreeB}$ | 1288877475.49 | 12621967.57 | -33.93% | -33.93% |
| $T_{QapFtreeB}$ | 30825934.92 | 361225.03 | -97.61% | -98.42% |
| $T_{RandVl2B}$ | 1835909748.69 | 22100449.20 | — | — |
| $T_{GpVl2B}$ | 1211796514.7 | 10158873.22 | -33.99% | -33.99% |
| $T_{QapVl2B}$ | 28061769.69 | 410242.89 | -97.68% | -98.47% |

Table 5.4: Changes in the total cost of communication in set B

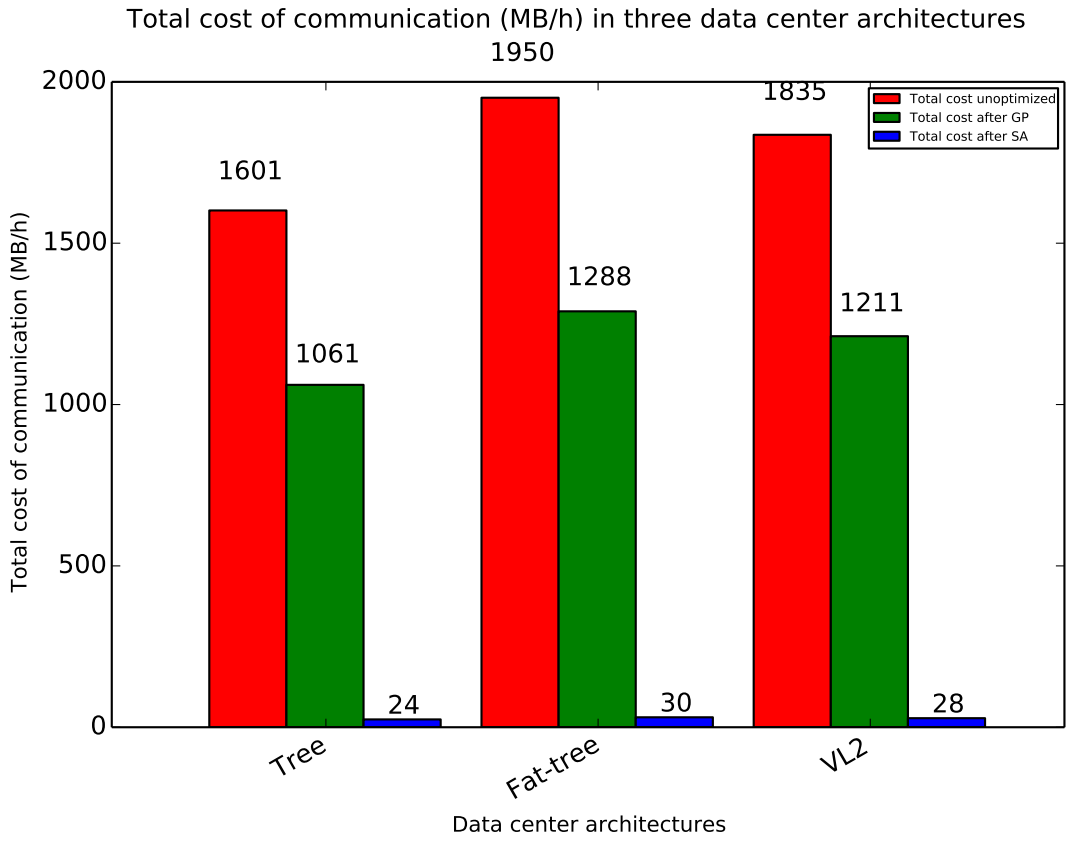


Figure 5.19: Total cost of communication in all three experiments in set B

The figure 5.19 shows the overall comparison of the average total cost of communication with the randomly assigned VMs in the three data center

network architectures experimented on, the average total cost of communication after the VM clustering with the graph partitioning technique and lastly the average total cost of communication after the clusters were assigned to the server racks in a traffic-aware way with the cluster placement algorithm which used the simulated annealing technique.

# Part III

# Conclusion

# Chapter 6

# Discussion and future work

This chapter reflects on the results obtained through the project, the development and the course of the project, the feasible alternative approaches and suggests improvements for the future work.

## 6.1 Implementation of the algorithms

The goal of this paper was to investigate how a graph partitioning algorithm proposed by Oommen [51] could be implemented in order to be used to consolidate VMs in a traffic-aware way and how the resulting VM clusters could be assigned to the available server racks in a way which would minimize the total cost of communication in the developed data center model. These questions have been answered by studying Oommen's algorithm and implementing it in Python on a hypothetical data center using traffic traces as communication data. Simulated annealing-based quadratic assignment algorithm has been implemented in order to place the resulting clusters on the server racks. The effect of the traffic aware consolidation and the cluster assignment have been measured and analyzed with the help of supporting scripts.

The results clearly showed the expected improvement by significantly decreasing the total cost of communication in all the conducted experiments with the stable and the expected rates. The validity of the results were confirmed by closely observing the effect of the traffic-aware consolidation through analyzing other observable effects of such consolidation,

namely the intracluster and intercluster traffic pattern changes. The observations confirmed the expected increase in the intracluster traffic and the simultaneous decrease in the intercluster traffic. The research showed that the effects of optimization were greater in the environment with few VMs communicating extensively with each other while the rest of the VMs had significantly less mutual traffic. The optimization effect was relatively less in the environment where most of the VMs communicated with each other at roughly the same rates. However this result was not achieved easily as several challenges had to be faced in order to get the developed system working properly according to the plan.

### 6.1.1 Challenges during the implementation

One of the most challenging problems during the project was developing and debugging the considerable amount of code needed for the implementation of the algorithms, for managing the experiments, collecting the resulting data and analyzing and plotting the results. By the end of the thesis over 1500 lines of code was developed. Even though the pseudocode was available from the start through Oommen's paper [51] the actual implementation proved to be more challenging than previously assumed. The main reason for this turned out to be a number of low level operations that had to be implemented in the code in order for the algorithm to function. Numerous different scenarios had to be taken in account when developing the graph partitioning algorithm. The learning automata had do respond to the dynamic changes in VM clusters by invoking different procedures and these procedures had to be precise, fast and effective. Another challenge was the difficulty associated with referring to list element values and the list indices in the whole set of nodes as well as in the separate subpartitions during development of the code. There were many small details that had to be remembered in order to write the code and it was easy to make a mistake during the process.

Several small changes were made to the original algorithm during the implementation. One of the biggest changes was the deactivation of the *RewardDissimilarNodes* procedure. The reason behind this decision was the nature of the input data the experiments (especially the experiment set A)

was conducted on. The number of edges over the similarity threshold was significantly lower compared to the number of edges under the dissimilarity threshold. This fact caused most of the VMs to be rewarded and pushed inwards in their clusters and as a result very few VMs managed to actually migrate from one cluster to another. After deactivation of the *RewardDissimilarNodes* the number of migrations increased and the results were more favorable as expected.

### 6.1.2 Obstacles encountered

The code measuring the total cost of communication was developed at the later stage of the project and revealed some unexpected results. It was suspected that the reason for this was a bug somewhere in the several hundred lines of code amongst the low level operations. At this stage the project was halted and couldn't continue without the cause of the highly unexpected results could be discovered and dealt with. After scrutinizing the code thoroughly an error was discovered and corrected. During the debugging process more discoveries were made which helped further improve the functionality of the algorithm. Several adjustments had to be done to the algorithm in order to get it to work optimally. Due to this obstacle the conducting of the experiments was delayed for several days, however better understanding of the algorithms involved, the data used in the project and the whole process of graph partitioning and object migration was achieved through the trial.

It was also discovered that the initial communication cost measured after randomly distributing the 1600 VMs in the data center network model could vary significantly due to chance. In order to avoid distorted and unreliable results the 35x35 experiment scheme was designed and later used to conduct the three experiments. It was assumed that conducting over 30 experiments on the multiple baseline node sets would increase the validity and the reliability of the results.

### 6.1.3 Changes in the initial approach

As the task of developing both the algorithms needed for the experiments and the code needed to manage the complex experiments proved to be more challenging than initially assumed the initial plans for the project had to be adjusted in order to better handle the amount of work within the strict deadline boundaries. The initial approach included further expanding the project by developing several additional features to the VM clustering algorithm making it more robust and implementing additional custom constraints handling. As it became clear that development of these mentioned additional features in addition to the core features of the algorithms could imperil the overall quality of the project these exciting plans had to be reluctantly altered and postponed for the future research. As a result of changing the initial plans it became possible to re-allocate the available resources in order to expand the experiments and test on three different data center models instead of only one. It also became possible to conduct two different sets of the experiments with the three data center models. Each set of three experiments was conducted with a different set of 1600 VMs and this approach allowed for two different data center traffic environments to be simulated which in turn helped to better test the effects of the developed algorithms.

### 6.1.4 Alternative approaches

In retrospect several alternative approaches can be considered in order to address the problem statement questions. First of all the traffic traces used for the simulation of the VM traffic in a data center could be diversified and the quality of the data can be further improved. In the ideal scenario comprehensive information about data traffic patterns for the virtual machines collected over longer periods of time (days and perhaps weeks) from different real life data centers could be used as the traffic data for adapting the clustering and placement algorithms. This would provide a more reliable overview of the real traffic between the VMs in the real world. Since data collection of this type and scope might be difficult an alternative approach could also simply be data traces collected from one data center over longer period of time, such as 24 hours or more.

Another alternative method of experimenting would be to keep the traffic traces used in this project but make sure to choose a different random sets of 1600 VMs and run experiments to test if the results obtained show the similar effect of the traffic-aware consolidation most of the time which would provide stronger proof for the VM clustering algorithm's positive impact on different traffic types.

As demonstrated in the analysis section the randomly chosen 1600 VMs had edges forming a rather skewed graph, where 89% of the edges were below the average (3712 edges below and 434 edges over the mean edge of 816605) which could have significant impact on the outcome of the optimization with the developed algorithms. An alternative could be to pick the 1600 VMs in a more controlled way as to form a more evenly distributed graph and experiment on the new set to see if the outcome of the VM consolidation is the same in this case.

**"Top-down" vs "bottom-up" clustering**

In this thesis the VM set was partitioned by randomly distributing the all the available 1600 VMs over the 16 clusters and then by optimizing them by migrating the VMs between the clusters in a sort of "bottom-up" clustering. A different approach could have been tested. Namely dividing the total number of VMs in two clusters ($\frac{1600}{2}$) and optimizing these two clusters with the graph partitioning technique. Then dividing each of these two clusters into two and optimizing the resulting 4 clusters by migrating the VMs between the clusters and repeating these steps until 16 clusters were produced and graph partitioned. This approach can be seen as a "top-down" clustering.

**Potential improvements for VM clustering**

The VM clustering algorithm proved to be rather fast and effective tool for traffic-aware VM consolidation through repeated experiments throughout this research. However further improvements might be possible through fine-tuning and more testing by focusing on the object migration processes through step-by-step analysis of the actual movements of the nodes along

the available states in the group and the eventual migrations across the groups. This would help gain deeper understanding of the behavior of the object migration mechanisms in the VM clustering algorithm and could potentially help further optimize it. It could be useful to test the VM clustering algorithm's capabilities with variable-sized sub-partitions.

**Further improvements for cluster placement**

The cluster placement algorithm turned out to be more resource-consuming than originally expected due to the constant calculations needed for each iteration no matter the picked pair of clusters. The simulated annealing further increased the amount of workload needed to find the optimal solution to the quadratic assignment problem. The algorithm can potentially be improved through more detailed analysis, testing and fine-tuning.

The function chosen for the gradual decrease of the initially set temperature could be substituted with other functions in order to change the temperature decrease pattern and test to learn how the algorithm responds. An alternative to the implemented design would be to skip the initial cluster ranking step prior to the simulated annealing. This would potentially increase the probability of the algorithm choosing more inferior states in the early stages of the simulated annealing causing the algorithm to spend more time on searching the optimal solution.

Due to the high number of the teste needed in order to collect statistically reliable data the initial temperature was lowered in order to speed up the algorithm. This might have caused the algorithm to underperform. Experiments with higher temperature would be useful in order to learn more about the capabilities of the cluster placement algorithm.

**Optimization of traffic matrices**

As mentioned in the thesis the generation of the traffic matrices heavily used during the project was the most time consuming part. However, when once generated the matrices were stored and later easily read and used in different experiments. Different approaches could have been used in

order to store the VM traffic rates data and dynamically and continuously update them. For example a database could have been used to serve this purpose. The initial population of the database can take significantly long time, however it can be easier maintained and incrementally updated as the traffic picture changes in the data center.

### 6.1.5 Thesis contributions

The project demonstrates the high potential of the traffic-aware VM consolidation with the use of the adapted version of John Oommen's algorithm while the simulated annealing-powered placement algorithm further improves the results. The graph partitioning algorithm impresses with the speed with which it converges to an impressively optimal solution given the relatively high number of the nodes it's given to partition.

Even though some questions still remain and more testing should be done in order to test the algorithm with different data traces and different numbers of VMs and VM groups or server racks it is still possible to conclude that this technique has a high potential and further research could benefit the cloud computing industry. The findings of this thesis show that complex problem of consolidating VMs with high mutual traffic and placing them optimally in any of the three given topologies can be achieved in a relatively short time. Given the explosive growth of cloud computing and virtualization inside the data centers the traffic optimization is becoming a growingly important issue and this research could hopefully be a small step in finding robust and resource-effective solutions.

## 6.2 Suggestions for future work

There is always potential for improvement and this work certainly isn't an exception. Several features and functions can be developed to further improve and expand the capabilities of the VM clustering and cluster placement algorithms.

### 6.2.1 Constraints

Future work could focus more on expanding the VM clustering algorithm by developing custom constraints handling in case of cloud tenants whose VMs can't freely be moved from one server rack to another due to various reasons. For example in some cases VMs who communicate immensely with one another are the VMs who shouldn't be hosted on same physical servers due to strict redundancy or security requirements. The algorithm could be further developed to take such constraints in account.

In the simulated data center architecture models in this project it was assumed that the link capacity was same for all the links. This is usually not the case in the real world. The VM clustering and cluster placement system could be further enhanced by considering various link capacities in the data center network topologies.

### 6.2.2 Minimizing migrations

One important future improvement would be to improve the VM clustering algorithm by developing smart functionality which ensures that the minimal number of migrations is needed for the final optimization. This feature is rather important as migrating large numbers of VMs is quite resource consuming and could be difficult to plan and execute in a large and complex cloud environment.

### 6.2.3 From static to dynamic optimization

The algorithms developed in this work are of "offline" nature due to the fact that the static traces are read once and the decision to move multiple VMs at once is taken through the VM consolidation process. Future research can further develop the clustering and placement algorithms by streamlining them and adapting them to a live environment. The dynamic version of the improved system would constantly monitor the changes in the data center traffic and generate suggestions for VM migrations after reliably detecting VM clusters. The future work could also focus on incremental change where the VMs with most impact on the overall traffic picture are singled out and taken care of proactively. It could be useful to

further develop the intelligent traffic-aware VM consolidation system by building a graphical user interface (GUI) with informative dashboards giving a clearer overview of the bigger picture and providing ease of use for the users of different technical expertise.

# Chapter 7

# Conclusion

The aim of this project was to investigate how a graph partitioning algorithm could be used in order to consolidate VMs in a traffic-aware way and to explore how a quadratic assignment algorithm would help assign the produced VM clusters to the server racks in order to reduce the total cost of communication in any data center.

The problem statement was addressed by developing a VM clustering algorithm based on Oommen's Learning Automata based Graph Partitioning Algorithm (GPLA) and a cluster placement algorithm using simulated annealing technique. The two algorithms were used to partition 1600 VMs into 16 clusters and assign the clusters to 16 server racks. Experiments were conducted on three different data center networking architecture models simulated for the project using publicly available traffic traces from a live data center. The two algorithms were tested extensively with two different data sets in order to strengthen the reliability of the results.

The analysis of the results of over 2500 tests conducted in this project revealed that the VM clustering algorithm decreased the total cost of communication from 34% to 85% depending on the original input data characteristics. The cluster placement algorithm further decreased the total cost of communication with the total improvement of 98% to 99%. The analysis showed that the VM clustering algorithm was fast, resource-effective and rather effective at consolidating the VMs with high mutual traffic in clusters while the cluster placement algorithm managed to find a significantly improved placement for the resulting clusters in all the data center network

topologies tested in this thesis.

Further testing with more diverse data traces and several improvements have been suggested for the future work, such as support for the custom constraints for VM locations, minimization of the needed VM migrations and transformation of the system into a more dynamic solution.

# Part IV

# Appendix

# Chapter 8

# Appendix

All the scripts developed and used in this thesis have been uploaded to an online repository where they can be accessed via the hyperlinks presented in this section.

## 8.1 Experiment management scripts

Three different scripts were used to manage the course of the different experiments.

Scripts `cluster_and_place_vms.py`, `intracluster_comm.py` and `cost_matrix.py`: (http://bit.ly/1EQsfIA)

## 8.2 Algorithm implementations

Script `cluster_vms.py`: (http://bit.ly/1B5OVOa)
Script `place_clusters.py`: (http://bit.ly/1FmasY5)

## 8.3 Intracluster experiment

Script `calc_intracluster_comm.py`: (http://bit.ly/1JRnkFq)

## 8.4 Analysis and plotting scripts

Scripts `analyze_and_plot.py`, `plot_intracluster_and_heatmap.py`,

`plot_overall_groupbar.py`: (http://bit.ly/1bYotj1)

# Bibliography


[1] Gordon C Armour and Elwood S Buffa. "A heuristic algorithm and simulation approach to relative location of facilities." In: *Management Science* 9.2 (1963), pp. 294–309.

[2] Hitesh Ballani et al. "Towards predictable datacenter networks." In: *ACM SIGCOMM Computer Communication Review*. Vol. 41. 4. ACM. 2011, pp. 242–253.

[3] Angela Bartels. "Data Center Evolution: 1960 to 2000." In: *Rackspace* 2011 (Aug. 2011). URL: http://www.rackspace.com/blog/datacenter-evolution-1960-to-2000/.

[4] Theophilus Benson, Aditya Akella, and David A Maltz. "Network traffic characteristics of data centers in the wild." In: *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM. 2010, pp. 267–280.

[5] Theophilus Benson et al. "Understanding data center traffic characteristics." In: *ACM SIGCOMM Computer Communication Review* 40.1 (2010), pp. 92–99.

[6] Milena Bieniek. "A note on the facility location problem with stochastic demands." In: *Omega* (2015).

[7] Kashif Bilal et al. "A taxonomy and survey on Green Data Center Networks." In: *Future Generation Computer Systems* 36 (2014), pp. 189–208.

[8] Michael Bullock. "Data Center Definition and Solutions." In: *CIO* 2009 (Aug. 2009). URL: http://www.cio.com/article/2425545/data-center/data-center-definition-and-solutions.html.

[9] Aydin Buluç et al. "Recent advances in graph partitioning." In: *Preprint* (2013).

[10] Rainer E Burkard and Franz Rendl. "A thermodynamically motivated simulation procedure for combinatorial optimization problems." In: *European Journal of Operational Research* 17.2 (1984), pp. 169–174.

[11] Rajkumar Buyya, Rajiv Ranjan, and Rodrigo N Calheiros. "Modeling and simulation of scalable Cloud computing environments and the CloudSim toolkit: Challenges and opportunities." In: *High Performance Computing & Simulation, 2009. HPCS'09. International Conference on*. IEEE. 2009, pp. 1–11.

[12] Intel IT Center. *Cloud Computing Research for IT Strategic Planning*. http://www.intel.com/content/www/us/en/cloud-computing/next-generation-cloud-networking-storage-peer-research-report.html?wapkw=peer+research. [Online; accessed 15-February-2015]. 2012.

[13] Vladimír Černỳ. "Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm." In: *Journal of optimization theory and applications* 45.1 (1985), pp. 41–51.

[14] Cisco. *Data Center Architecture Overview*. http://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Data_Center/DC_Infra2_5/DCInfra_1.html.

[15] Intel Corporation. *A Cloud Test Bed for China Railway Enterprise Data Center*. http://www.intel.com/content/dam/doc/case-study/cloud-computing-xeon-test-bed-china-railway-study.pdf. [Online; accessed 18-March-2015]. 2009.

[16] Tharam Dillon, Chen Wu, and Elizabeth Chang. "Cloud computing: issues and challenges." In: *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*. Ieee. 2010, pp. 27–33.

[17] Chris HQ Ding et al. "A min-max cut algorithm for graph partitioning and data clustering." In: *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*. IEEE. 2001, pp. 107–114.

[18] RW Eglese. "Simulated annealing: a tool for operational research." In: *European journal of operational research* 46.3 (1990), pp. 271–281.

[19] Weiwei Fang et al. "VMPlanner: Optimizing virtual machine placement and traffic flow routing to reduce network power costs in cloud data centers." In: *Computer Networks* 57.1 (2013), pp. 179–196.

[20] Forrester. *The Public Cloud Market Is Now In Hypergrowth: Sizing The Public Cloud Market, 2014 To 2020*. https://www.forrester.com/The+Public+Cloud+Market+Is+Now+In+Hypergrowth/fulltext/-/E-RES113365. [Online; accessed 15-February-2015]. 2014.

[21] Saurabh Kumar Garg and Rajkumar Buyya. "Networkcloudsim: Modelling parallel applications in cloud simulations." In: *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*. IEEE. 2011, pp. 105–113.

[22] James Glanz. "Power, pollution and the internet." In: *The New York Times* 2012 (Sept. 2012). URL: http://www.nytimes.com/2012/09/23/technology/data-centers-waste-vast-amounts-of-energy-belying-industry-image.html.

[23] Lukasz Golab et al. "Identifying frequent items in sliding windows over on-line packet streams." In: *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*. ACM. 2003, pp. 173–178.

[24] Albert Greenberg et al. "VL2: a scalable and flexible data center network." In: *ACM SIGCOMM computer communication review*. Vol. 39. 4. ACM. 2009, pp. 51–62.

[25] Greenpeace.org. *Make IT Green: Cloud Computing and its Contribution to Climate Change*. http://www.greenpeace.org/usa/Global/usa/report/2010/3/make-it-green-cloud-computing.pdf. [Online; accessed 28-January-2015]. 2010.

[26] Natasha Gude et al. "NOX: towards an operating system for networks." In: *ACM SIGCOMM Computer Communication Review* 38.3 (2008), pp. 105–110.

[27] Brad Hedlund. *Top of Rack vs End of Row Data Center Designs*. http://bradhedlund.com/2009/04/05/top-of-rack-vs-end-of-row-data-center-designs/. [Online; accessed 22-February-2015]. 2009.

[28] Mark Herbster and Manfred K Warmuth. "Tracking the best expert." In: *Machine Learning* 32.2 (1998), pp. 151–178.

[29] Liting Hu et al. "Net-cohort: Detecting and managing vm ensembles in virtualized data centers." In: *Proceedings of the 9th international conference on Autonomic computing*. ACM. 2012, pp. 3–12.

[30] IBM. *IBM Workload Deployer*. http://www-03.ibm.com/software/products/en/workload-deployer. [Online; accessed 29-January-2015]. 2015.

[31] IBM. *Virtio: An I/O virtualization framework for Linux*. http://www.ibm.com/developerworks/library/l-virtio/. [Online; accessed 21-February-2015]. 2010.

[32] Tabitha James, Cesar Rego, and Fred Glover. "A cooperative parallel tabu search algorithm for the quadratic assignment problem." In: *European Journal of Operational Research* 195.3 (2009), pp. 810–826.

[33] David S Johnson et al. "Optimization by simulated annealing: An experimental evaluation; part I, graph partitioning." In: *Operations research* 37.6 (1989), pp. 865–892.

[34] Dharmesh Kakadia, Nandish Kopri, and Vasudeva Varma. "Network-aware virtual machine consolidation for large data centers." In: *Proceedings of the Third International Workshop on Network-Aware Data Management*. ACM. 2013, p. 6.

[35] Brian W Kernighan and Shen Lin. "An efficient heuristic procedure for partitioning graphs." In: *Bell system technical journal* 49.2 (1970), pp. 291–307.

[36] Ali Khajeh-Hosseini, David Greenwood, and Ian Sommerville. "Cloud migration: A case study of migrating an enterprise it system to iaas." In: *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*. IEEE. 2010, pp. 450–457.

[37] S Kirkpatrick, CD Gelatt Jr, and MP Vecchi. "Optimization by simulated annealing." In: *Neurocomputing: foundations of research*. MIT Press. 1988, pp. 551–567.

[38] Jonathan Koomey. "Growth in data center electricity use 2005 to 2010." In: *A report by Analytical Press, completed at the request of The New York Times* (2011).

[39] Tjalling C Koopmans and Martin Beckmann. "Assignment problems and the location of economic activities." In: *Econometrica: journal of the Econometric Society* (1957), pp. 53–76.

[40] KPMG. *2014 Cloud Survey Report: Elevating Business in the Cloud*. http://www.kpmginfo.com/EnablingBusinessInTheCloud/downloads/2014. [Online; accessed 15-February-2015]. 2014.

[41] Katrina LaCurts et al. “Cicada: Introducing predictive guarantees for cloud networks.” In: *Proceedings of the USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*. 2014.

[42] Jeongkeun Lee et al. “Application-driven bandwidth guarantees in datacenters.” In: *Proceedings of the 2014 ACM conference on SIGCOMM*. ACM. 2014, pp. 467–478.

[43] David Marshall, Wade A Reynolds, and Dave McCrory. *Advanced server virtualization: VMware and Microsoft platforms in the virtual data center*. CRC Press, 2006.

[44] Sean Marston et al. “Cloud computing—The business perspective.” In: *Decision Support Systems* 51.1 (2011), pp. 176–189.

[45] Shahrzad Motamedi Mehr et al. “A new recommendation algorithm using distributed learning automata and graph partitioning.” In: *Hybrid Intelligent Systems (HIS), 2011 11th International Conference on*. IEEE. 2011, pp. 351–357.

[46] Shahrzad Motamedi Mehr et al. “Determining web pages similarity using distributed learning automata and graph partitioning.” In: *Artificial Intelligence and Signal Processing (AISP), 2011 International Symposium on*. IEEE. 2011, pp. 129–134.

[47] Xiaoqiao Meng, Vasileios Pappas, and Li Zhang. “Improving the scalability of data center networks with traffic-aware virtual machine placement.” In: *INFOCOM, 2010 Proceedings IEEE*. IEEE. 2010, pp. 1–9.

[48] Microsoft. *Microsoft Assessment and Planning (MAP) Toolkit for Hyper-V*. https://technet.microsoft.com/en-us/solutionaccelerators/dd537570.aspx. [Online; accessed 29-January-2015]. 2015.

[49] Kumpati S Narendra and MLAA Thathachar. “Learning automata-a survey.” In: *Systems, Man and Cybernetics, IEEE Transactions on* 4 (1974), pp. 323–334.

[50] Radhika Niranjan Mysore et al. “Portland: a scalable fault-tolerant layer 2 data center network fabric.” In: *ACM SIGCOMM Computer Communication Review*. Vol. 39. 4. ACM. 2009, pp. 39–50.

[51] B. John Oommen, De St Croix, et al. “Graph partitioning using learning automata.” In: *Computers, IEEE Transactions on* 45.2 (1996), pp. 195–208.

[52] B. John Oommen and Daniel C. Y. Ma. "Deterministic learning automata solutions to the equipartitioning problem." In: *IEEE Transactions on Computers* 37.1 (1988), pp. 2–13.

[53] Susan Hesse Owen and Mark S Daskin. "Strategic facility location: A review." In: *European Journal of Operational Research* 111.3 (1998), pp. 423–447.

[54] Christos H Papadimitriou. *Computational complexity*. John Wiley and Sons Ltd., 2003.

[55] Jing Tai Piao and Jun Yan. "A network-aware virtual machine placement and migration approach in cloud computing." In: *Grid and Cooperative Computing (GCC), 2010 9th International Conference on*. IEEE. 2010, pp. 87–92.

[56] Cisco Press. "Cisco Data Center Infrastructure 2.5 Design Guide." In: (2007).

[57] AS Ramkumar, SG Ponnambalam, and N Jawahar. "A new iterated fast local search heuristic for solving QAP formulation in facility layout design." In: *Robotics and Computer-Integrated Manufacturing* 25.3 (2009), pp. 620–629.

[58] Press Release. "Gartner Predicts Infrastructure Services Will Accelerate Cloud Computing Growth." In: *Forbes* 2013 (Oct. 2013). URL: http://www.gartner.com/newsroom/id/2613015.

[59] Erik Rolland and Hasan Pirkul. *Heuristic solution procedures for the graph partitioning problem*. College of Business, Ohio State University, 1991.

[60] Satu Elisa Schaeffer. "Graph clustering." In: *Computer Science Review* 1.1 (2007), pp. 27–64.

[61] Jason Sonnek et al. "Starling: Minimizing communication overhead in virtualized computing platforms using decentralized affinity-aware migration." In: *Parallel Processing (ICPP), 2010 39th International Conference on*. IEEE. 2010, pp. 228–237.

[62] Tsuyoshi Tanaka, Toshiaki Tarui, and Ken Naono. "Investigating suitability for server virtualization using business application benchmarks." In: *Proceedings of the 3rd international workshop on Virtualization technologies in distributed computing*. ACM. 2009, pp. 43–50.

[63] VMWare. *Capacity Planner*. http://www.vmware.com/products/capacity-planner/. [Online; accessed 29-January-2015]. 2015.

[64] Eric W. from MathWorld Weisstein. *NP-Problem*. http://mathworld.wolfram.com/NP-Problem.html. [Online; accessed 12-March-2015]. 2010.

[65] Habib Youssef, Sadiq M Sait, and Hakim Adiche. "Evolutionary algorithms, simulated annealing and tabu search: a comparative study." In: *Engineering Applications of Artificial Intelligence* 14.2 (2001), pp. 167–181.