On Achieving Intelligent Traffic-aware Consolidation of Virtual Machines in a Data Center Using Learning Automata*

Akaki Jobava†, Anis Yazidi‡, B. John Oommen§, and Kyrre Begnum¶

Abstract

Unlike the computational mechanisms of the past many decades, that involved individual (extremely powerful) computers or clusters of machines, Cloud Computing (CC) is becoming increasingly pertinent and popular. Computing resources such as CPU and storage are becoming cheaper, and the servers themselves are becoming more powerful. This enables clouds to host more Virtual Machines (VMs). A natural consequence of this is that many modern-day data centers experience very high *internal* traffic within the data centers themselves. This is, of course, due to the occurrence of servers that belong to the same tenant, communicating between themselves. The problem is accentuated when the VM deployment tools are not traffic-aware. In such cases, the VMs with high mutual traffic often end up being far apart in the data center network, forcing them to communicate over unnecessarily long distances. The consequent traffic bottlenecks negatively affect both the performance of the application and the network in its entirety, posing non-trivial challenges for the administrators of these cloud-based data centers.

The problem, and consequently the solution, can, quite naturally, be compartmentalized into two phases which follow each other. In the first, the task is to consolidate VMs into clusters, where those that communicate with each other fall into the same cluster. The second phase assigns these clusters onto the available server racks. Both of these phases must be executed in a traffic-aware manner. This paper provides efficient intelligent solutions for both these phases. First of all, the VMs are consolidated with a VM clustering algorithm, and this is achieved by utilizing the toolbox involving Learning Automata (LA). By mapping the clustering problem onto the Graph Partitioning (GP) problem, our LA-based solution successfully reduces the total communication cost by amounts that range between 34% to 85%. Thereafter, the resulting clusters are assigned to the server racks using a cluster placement algorithm that involves a completely different intelligent strategy, i.e., one that invokes Simulated Annealing (SA). This phase further reduces the total cost of communication by amounts that range between 89% to 99%. The analysis and results for different models and topologies demonstrate that the optimization is done in a fast and computationally-efficient way.

*An abridged and preliminary version of this paper, which merely made a claim on the result, appeared in the *Proceedings of NTMS'16, the 2016 IFIP International Conference on New Technologies, Mobility and Security*, Larnaca, Cyprus, November 2016. We are extremely grateful to the anonymous Referees of the earlier version of this paper for their feedback. Their input significantly improved the quality of this current version.

†Former student at: Dept. of Computer Science, University College of Oslo and Akershus, Oslo. E-mail: akakijobava@gmail.com.

‡This author can be contacted at: Dept. of Computer Science, University College of Oslo and Akershus, Oslo. E-mail: anis.yazidi@hioa.no.

§Author's status: *Chancellor's Professor*, *Fellow: IEEE* and *Fellow: IAPR*. This author can be contacted at: School of Computer Science, Carleton University, Ottawa, Canada : K1S 5B6. The author is also an Adjunct Professor with University of Agder, Grimstad, Norway. E-mail: oommen@scs.carleton.ca.

¶This author can be contacted at: Dept. of Computer Science, University College of Oslo and Akershus, Oslo. E-mail:kyrre.begnum@hioa.no.

Keywords: *Cloud Computing, Virtual Machines, Graph Partitioning, Learning Automata, Quadratic Assignment, Simulated Annealing*

1 Introduction

Cloud Computing (CC) is a relatively new phenomenon. It refers to an environment and computational model in which physical and virtualized computing resources are distributed and accessed over the network. CC is maturing to become a very central paradigm within the theory and applications of computation. Its robustness, increasing user-friendliness, high flexibility and scalability, combined with its cost efficiency [11, 32, 38], make it an increasingly popular model[1] in real-life enterprises.

One of the main reasons behind the success of CC is that the concept of "virtualization", central to this computational model, allows the overall system to create, clone, migrate, restore, etc. Virtual Machines (VMs) in a time-effective manner with minimal effort from the system administrator. Live migration allows VMs to be moved from one physical host to another without the client/customer noticing it. This is because the service is never interrupted before, during or after the process. These characteristics of virtualization provide CC with the required robustness and flexibility, thus enabling the dynamic scaling of the infrastructure in a much more rapid and effective way when compared to systems that invoke a "traditional" model of computation. Consequently, CC is becoming one of the major driving forces behind the rapid growth of data centers around the world [17].

This is the arena in which we operate. The goal of this paper is to see how the VMs can be optimally placed within a data center in a traffic-aware manner. Viewed from a traffic-aware perspective, the resource of bandwidth becomes a bottleneck in the higher layers of the network, decreasing the performance when it concerns communication [60] between the applications. This also increases the workload for the network elements on the aggregation and core layers, which, in turn, often results in higher power consumption within the data center [17], more greenhouse emissions, and the increased business costs. Resolving this is far from trivial, as explained below, and accomplishing this is the goal of this paper.

First of all, the reader must observe that due to the exponential growth of data centers and the growing computational power of modern computers, these data centers are *not merely constrained* by the computational power, storage or any other computing resource, as in the past decades. Rather, they are increasingly constrained by the limitations of the systems' *networking* capabilities [7]. Large data centers are typically hosting hundreds/thousands of VMs for different CC service providers. The VMs are usually consolidated with resource usage in mind with various tools, such as VMWare Capacity planner [62], Microsoft Assessment and Planning (MAP) Toolkit for Hyper-V [40] or the IBM Workload Deployer [27], that can help plan and carry out VM consolidation with regards to CPU, memory and disk usage. However, these tools do not take into 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. The consequence of this is the overloading of the higher levels of the network[2] that contain the most expensive enterprise-grade equipment.

These problems pose a significant challenge not only for the environment and in terms of the increased power

[1] As per Intel's survey of 200 IT Managers [11], 80% of them are in the process of deploying or have already adopted private and/or public cloud facilities by moving parts of their IT environment onto it. The remaining 20% plan to do so in the near future.

[2] *Facebook*, for example, experiences roughly 1,000 times higher traffic usage inside its data center, when compared to the incoming and outgoing traffic from and to its users [36].

usage and business costs, but also for the performance of the network-dependent applications and the scalability and the growth of data centers. The 2009 study by Benson *et al.* [6] has shown that for the most part, the link utilization in the lower layers of data centers 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.

Problem description: The aim of this paper is to investigate an important aspect of the resource provisioning which, as far as we can see, has not received enough attention in the literature. This is, indeed, the placement of traffic-aware VMs. The problem is two-pronged.

First of all, it is clear that, in most cases, the applications communicating extensively with each other in the cloud environment will belong to the same tenant. It would thus be beneficial for the whole network if the VMs hosting applications with high mutual traffic were deployed in the close proximity with each other. To accomplish this, we would like the VMs that communicate much with each other to be "clustered" together. Such a placement would relieve the network elements in the upper layers of the networking infrastructure where the most expensive equipment usually operate, and fully utilize the links at the lower levels of the network. However, this, in and of itself, is far from trivial because the traffic patterns are not known *a priori*.

Secondly, once we have identified the VMs that really should be in the close proximity of each other, the task is to assign them to the available server racks.

This paper addresses both these issues. Firstly, it investigates how the VMs with high mutual communication can be consolidated into clusters in order to reduce the total communication cost. It then explains how these clusters can be assigned to the racks.

One approach to resolve this problem could be to attempt all possible combinations of VM placements and choose to choose the configuration that is "closest" to being optimal. However, since data centers usually host hundreds/thousands of VMs, this would require us to test an astronomical number of different permutations in order to find the best possible placement when the number of VMs is greater than 20 – the task would be computationally infeasible. The *modus operandus* suggested in this paper breaks down the problem in two main parts - each associated with one of the above distinct phases of solving the problem. We first determine the VM clusters using a Graph Partitioning (GP) algorithm. This is achieved by utilizing the toolbox involving Learning Automata (LA). By mapping the clustering problem onto the GP problem, our LA-based solution successfully reduces the total communication cost since it succeeds in consolidating VMs with high mutual traffic into distinct clusters. We then address the second phase of assigning the resulting clusters to the physical hosts in the server racks in the data center. This problem could be as computationally hard as the previous phase inasmuch as any algorithm that resolves the quadratic assignment problem should be able to handle it. We have opted to solve this phase by invoking the tools in the toolbox of Simulated Annealing (SA).

The methods that we propose have been rigorously tested. Indeed, since several new data center network architectures have been proposed in recent years, we will test our VM consolidation and cluster assignment schemes on a number of different architectures in order to see the effect that the network topology has on the traffic-aware VM consolidation. The paper also reports the topologies for which the algorithms yield the best results. Our overall solution, which combines these two phases, reduces the communication cost by as much as 99%, and conclusively proves the strength of the techniques we have used.

The novelty of this paper is that it presents the application of LA in the traffic-aware consolidation of virtual

machines in data centers, and also presents a strategy which serializes the tools in LA and SA to optimize CC. As far as we know, the use of these methods in this domain, individually and in tandem, are unreported in the literature.

2 Related Research and Background

Due to the exponential growth of CC, achieving a more efficient resource provisioning in data centers has become an increasingly critical issue that has attracted research interest. This has led to proposals for more efficient and scalable data center network architectures such as VL2 [21] and PortLand [47]. However, some researchers have suggested a different, traffic-oriented VM consolidation approach to the problem. The material[3] in this section surveys the field.

2.1 Network-aware Approaches

In this section, we shall briefly review the available research avenues presented in the literature when it concerns network-aware approaches.

Network-aware Virtual Machine Consolidation: Kakadia, Kopri and Varma address the internal bandwidth optimization problem in a data center by identifying groups of virtual machines based on the network traffic in the data center in [31]. The paper, which presents a network-aware consolidation strategy for VMs for large data centers, proposes a greedy consolidation algorithm to ensure a small number of migrations and fast placement decisions. The work includes algorithms to form VMClusters, to select VMs for migration and to place them using the cost tree. The paper reports experimental results that evaluate the scheme in an extended simulated cloud environment (NetworkCloudSim [19]) with its associated Software Defined Network (SDN) functionality support. It also uses Floodlight[4] as the SDN controller. The paper measured the runtime performance improvement for the jobs that were executed, and based on these results the authors conclude that I/O intensive jobs benefited the most. Besides these, short jobs also showed significant improvements. In terms of traffic localization, the results presented demonstrated significant superiority to other approaches. The ToR traffic displayed ∼60% increase, while the core traffic yielded ∼70% improvement.

VM Placement and Migration in CC: 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 [10] data center simulation environment and the results showed improved task completion time.

[3] The literature survey is quite detailed and comprehensive. The Referees have, generally speaking, recommended including it.
[4] http://www.projectfloodlight.org/floodlight/

Scalability of Data Center Networks Traffic-aware VMs: Meng, Pappas and Zhang [39] address network scalability 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 authors use real-life production data center traffic traces and prove that they can obtain significant improvements when compared to existing methods that ignore the traffic patterns and data center architectures. The paper also specifies the network-aware VM placement problem and attempts to optimize it by minimizing the average traffic latency caused by the network infrastructure with the assumption that each network element causes an equal delay of communication between the VMs. The so-called Cluster-and-Cut algorithm, which leverages the unique features of the traffic patterns and network topologies is used to optimize the solution. The algorithm has two major components, namely SlotClustering and VMMinKcut. The authors compare the results of Cluster-and-Cut and the two associated benchmark algorithms (i.e., LOPI [2] and SA [9]) involving an environment with 1,024 slots and VMs are used. From the results provided, one can conclude that the function value given by the Cluster-and-Cut algorithm is ∼10% smaller than the measures obtained by the two benchmarks.

Starling: Minimizing Communication Using Decentralized Affinity-Aware Migration: Sonnek *et al.* [60] introduce a decentralized affinity-aware migration technique for allocating VMs on the available physical resources. The technique monitors the network affinity between the pairs of VMs 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 a high mutual traffic as close to each other as possible, and this could involve placing them in the same server rack, cluster or local network. The salient contributions of the paper include affinity-based VM placement and migration, the implicit inference of dynamic job dependencies, and an efficient decentralized control mechanism. The affinity-aware migration algorithm runs on each node and incorporates traffic monitoring and fingerprinting, affinity inference and bartering and migration components. The experiment was conducted on a 7-node Xen-based cluster. The Intel MPI benchmark suite[5] and Cube MHD Jet (Cube)[6] were used for simulation and benchmarking. The results displayed about 42% improvement in the application's runtime over a technique that included no migration, and up to 85% reduction in the associated network communication overhead.

Detecting and managing vm ensembles: Liting Hu *et al.* [26] presents an application called 'Net-Cohort', which is a lightweight system that continuously monitors a system to identify potential VM ensembles, evaluates the degree of communication (or so-called 'chattiness') among the VMs in the potential ensembles, and enables optimized VM placement to reduce the stress on the 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, it invokes a statistical packet sniffer in order to identify VMs as members of a misplaced ensemble using the statistical algorithm proposed by Golab and De Haan in [20], and to thus finally make new VM placement decisions. The experiment was conducted on 15 Xen-based hosts and 225 VMs. Net-Cohort displayed the ability to detect VM ensembles at low cost with about 90% accuracy. The experimental results showed that the new VM placement improved the application throughput by 385% for an instance of RUBiS , while application throughput for an instance of Hadoop improved by 56.4%. The quality of service (QoS) for a SIPp instance displayed an improvement by a

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

factor of 12.76.

Introducing Predictive Guarantees: In their system Cicada, LaCurts *et al.* [35] introduce predictive guarantees, which represents a new abstraction for bandwidth guarantees in CC networks. This 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" concept [24]. In order to predict the traffic and the underlying patterns, they utilize all the previously observed traffic matrices. The advantage of this method is that it does not require an extensive amount of data in order to make predictions. For VM placement, they invoke a two-stage "virtual oversubscribed cluster" (VOC) algorithm introduced in Ballani *et al.* [4] which is designed to place clusters on the smallest subtree. Cicada's greedy algorithm tries to place the VM pairs with the most intercommunication on the highest-bandwidth paths, typically on the same rack, within the same subtree. Cicada's performance was compared to VOC algorithm on a simulated physical infrastructure with 71 racks with 16 servers each. The reported results show that Cicada's placement algorithm leaves more inter-rack bandwidth available.

Application-Driven Bandwidth Guarantees in Datacenters: Lee *et al.* [36] introduce CloudMirror, a solution that provides bandwidth guarantees to cloud applications by deriving a network abstraction based on the application communication structure, referred to as the "Tenant Application Graph" or TAG. CloudMirror provides a new workload placement algorithm that meets bandwidth requirements by using TAGs while taking into account high availability considerations. The 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. Example of such tiers are the web, business logic and database tiers. Users can either specify a matching TAG model and tune the bandwidth guarantees by themselves. On the other hand, they can resort to cloud orchestration systems like OpenStack Heat or AWS CloudFormation to generate TAG models. The simulation environment was written in Python and both the efficiency and the metric of accepting more tenant requests by the CloudMirror placement algorithm (when compared to other schemes) were evaluated in it. The results showed that CloudMirror outperforms the performance of the existing solutions. CloudMirror was able to handle 40% more bandwidth demand when compared to the Oktopus [4] system. It also improved the high availability from 20% to 70%.

Reducing Network Power Costs in Cloud Data Centers: The main focus in the paper by Fang *et al.* [17] 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 authors propose VMPlanner, a novel approach for network power reduction in cloud-based data centers. VMPlanner does not merely try manage the VM placements but also the traffic flow routing by implementing three approximation algorithms, namely a traffic-aware VM grouping algorithm, a distance-aware VM-group to a server-rack mapping algorithm, and power-aware inter-VM traffic flow routing algorithm. The VMPlanner system consists of three modules: an analyzer, an optimizer and a controller that can all be implemented as NOX applications [22] to run on top of a network of OpenFlow switches. The performance of VMPlanner was evaluated on a simulator developed in C++ using simulation parameters and traffic conditions from real cases obtained from a private data center test-bed [14]. The experiment was conducted with 2,000 VMs. The results reported were very preliminary but, at the same time, the paper succeeded in demonstrating the potential of reducing power usage by consolidating VMs in a traffic-aware way

and intelligently routing the traffic.

2.2 Three-tier network architecture

A data center network is traditionally based on a *layered* [13] [56] or a three-tier approach. Such a three-tier network architecture consists of three layers of switches and routers (see Fig.1). The layered approach is designed to enhance scalability, high performance and flexibility and to also improve the maintenance associated with data center networks. These layers are explained below.

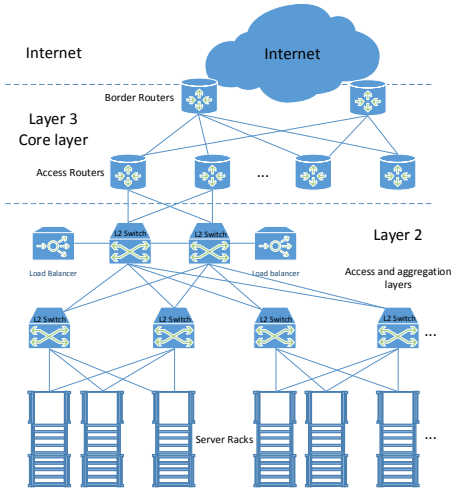


Figure 1: The architecture of a traditional layered data center.

Access layer: This is where the servers are physically connected to the network by connections to the Layer 2 switches, also called the 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: This layer handles all the incoming and outgoing traffic that comes in and leaves the data center. This layer provides the connectivity required to various aggregation modules. It handles the Layer 3 networking with the access and border routers.

2.3 Top of Rack (ToR) and End of Row (EoR) Designs

Typical data centers consist of racks of rows of server. A server rack, sometimes referred to as server cabinet, is usually a metal frame designed to hold various pieces of IT equipment such as servers, blade chassis, switches, routers, network patch panels, and to provide power, connectivity and cooling to these components. Each rack typically contains ethernet switches and patch panels on the top, although the switch does not not necessarily have to, physically, be on the 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 [47]. The advantages of such a ToR design are that it requires less cabling, flexible "per rack" architecture and fiber infrastructure. The primary disadvantages, however, are that one requires more switches in the design. Further, it causes more server-to-server traffic in the aggregation layer.

Optimally, bandwidth is over-provisioned in the ToR switch and to a lesser extent in the aggregation layer. However, an injudicious placement of "chatty VM" in different racks leads to cross-rack communication that increases 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 the EoR rack. The switches in this scenario are called the EoR switches. Physically, however, the EoR switches don't necessarily have to be situated at the end of each row. This approach requires fewer access switches and there are also fewer ports involved on the aggregation layer. On the other hand, it requires expensive and bulky copper cabling. The other cons of this approach are that it requires more patching and cable management. It also provides less flexibility [23].

2.4 Data Center Network Architectures

Due to the exponential growth of the cloud in data centers and the evolution of the computers in an of themselves, computing power is no longer the constraining factor in the data centers. The servers are becoming increasingly powerful and as the phenomenon of CC grows,the number of VMs correspondingly explodes. Thus, data centers are faced with inherent problems in the traditional data center network (DCN) architecture. This leads to real problematic issues such as bandwidth bottlenecks, oversubscription in the higher layers and the under-utilization of the lower layers of the data center network are becoming [7]. To resolve this, several new approaches to designing data center network topologies have been proposed in the recent years, one of which is the "tree topology" discussed below (see Fig. 5).

A tree topology: As mentioned previously, modern-day data centers usually follow traditional three-tier (or three-layer) network architectures. At the lowest level, referred to as the *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 topology where packets are forwarded according to a Layer 2 logical topology [39]. The higher level network elements are usually enterprise-level devices and are often highly oversubscribed.

2.5 Recently proposed DCN architectures

Several new data center network architectures have been proposed as alternatives to the legacy DCN architecture. We describe them briefly here.

PortLand (Fat-tree)

The PortLand data center network architecture is an attempt to solve cross-section bandwidth challenges of the tree-topology. It makes use of the so-called Fat-tree network topologies. The network elements in a PortLand DCN follow a hierarchical organization similar to the tree-topology and form a Clos[7] (or a bipartite graph) topology (see Fig. 6). The Fat-tree is organized in pods, where a Pod refers to a group of access and aggregation switches forming a complete Clos topology. In a Fat-tree, each pod is connected to all of the core switches.

The 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 pods, $\frac{k}{2}$ access switches and $\frac{k}{2}$ aggregation switches in each pod. Each pod

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

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. In total, there will be $\frac{5k^2}{4}$ switches connecting $\frac{k^3}{4}$ servers to each other.

VL2

The VL2 network architecture (see Figure 7) resembles the traditional three-tier tree architecture. Although it is also, by definition, a three-layer architecture, the core and aggregation layers are a Clos topology [21].

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.

BCube

BCube (see Figure2) is a multi-level server-centric DCN architecture. The term "server-centric" refers to an architecture in which the servers become part of the networking infrastructure and participate in packet forwarding for the other servers.

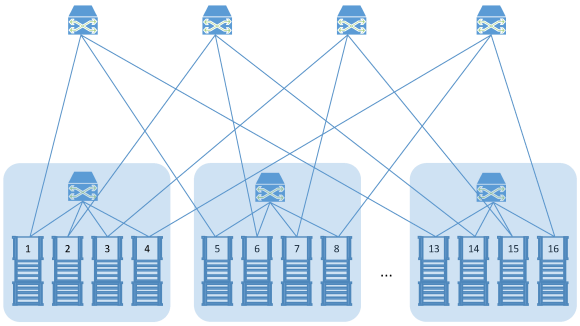


Figure 2: The BCube topology.

2.6 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 size of $N \times N$, 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, denoted by j.

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

In the example matrix displayed above, each element of the matrix represents cost of communication from node i to node j, or quantifies the "distance" from node i to node j.

The Cost matrix should not be confused with the so-called 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. The Cost matrix can be either asymmetric or symmetric. In some cases, an asymmetric matrix is first constructed when the connection costs between the nodes are different depending on the "direction" of the traffic. After obtaining an asymmetric cost matrix, a symmetric cost matrix can easily be calculated by computing the average costs between the nodes.

3 Learning Automata (LA)

Our solution to the traffic-aware consolidation involves LA. LA have been used to model biological learning systems and to learn the optimal action which a random environment offers. The learning is accomplished by actually interacting with an environment and processing its responses to the actions that are chosen, while gradually converging toward an ultimate goal. There are various applications that use LA including parameter optimization, statistical decision making, telephone routing, pattern recognition, game playing, natural language processing, modeling biological learning systems, string taxonomy, detection and tracking, distributing the process of a parallel application, routing in communication networks and object partitioning [18, 25, 42, 45, 46, 48, 49, 51, 52, 53, 63]. More recently, LA have been utilized in cloud computing [43], adaptive Petri-nets [61], optimization: [44], [37], vehicular systems [3], graph problems [41], social network analysis [58], cognitive radio networks [29] and peer-to-peer networks [59]. Since the literature on LA is extensive, we refer the reader to [45] for a good reference that also provides an overview of the field.

The functionality of the LA can be described as a sequence of repetitive feedback cycles. The feedback cycle involves two entities, the *Random Environment* and the LA. During each cycle the automaton chooses an action, which triggers a response from the Environment, and uses the received response - that can be either a reward or a penalty- with the knowledge gained from the previous cycles to determine which is the next action to be chosen. By the process of learning, the automaton adapts itself to the Environment and determines the optimal action, i.e., the action which has the minimum penalty probability, or has a maximum reward probability.

Incorporating LA in any application domain is an evidence of the power of the philosophy. Basically, LA learn from the random environment. The actual technique involved in applying the LA philosophy in the different applications involves modeling the actions, simulating the transforming functions, and representing the system's output in order to have reward or penalty responses. This is where the creativity of the researcher becomes apparent.

Stochastic LA can be classified into two main classes:

1. Fixed Structure Stochastic Automata ($FSSA$): These $FSSA$ have the property that their transition and output functions do not change with time. These LA seem to possess powerful properties useful for solving different NP-hard problems.

2. Variable Structure Stochastic Automata ($VSSA$): These $VSSA$ have a dynamically changing structure, because their transition and output matrices are time varying. In practice, however, they are merely defined in terms of action probability updating rules which are either of a continuous or discrete nature [1, 50]. Automata with a variable structure are generally much faster in their convergence.

3.1 Fundamentals of FSSA

A $FSSA$ is a quintuple ($\underline{\alpha}$, $\underline{\Phi}$, $\underline{\beta}$, F, G) where:

- $\underline{\alpha} = \{\alpha_1, \ldots, \alpha_R\}$ is the set of actions that it must choose from.
- $\underline{\Phi} = \{\phi_1, \ldots, \phi_S\}$ is a set of states.
- $\underline{\beta} = \{0, 1\}$ is its set of inputs. The '1' represents a penalty, while the '0' represents a reward.
- F is a map from $\Phi \times \beta$ to Φ. It defines the transition of the internal state of the automaton on receiving an input. F may be stochastic.
- G is a map from Φ to α, and it determines the action taken by the automaton if it is in a given state. With no loss of generality, G is deterministic.

As discussed above, the automaton is offered a set of actions, and it is constrained to choose one of them. When an action is chosen, the Environment gives out a response $\beta(t)$ at a time 't'. The automaton is either penalized or rewarded with an unknown probability c_i or $1 - c_i$, respectively. On the basis of the response $\beta(t)$, the state of the automaton $\phi(t)$ is updated and a new action is chosen at (t+1). The penalty probability c_i satisfies:

$$c_i = Pr[\beta(t) = 1 | \alpha(t) = \alpha_i] \qquad (i = 1, 2, \ldots, R).$$

The basic idea used to solve the traffic consolidation problem is based on a sub-class of LA solutions that has been used to solve the object partitioning problem [52, 18], and more particularly the graph partitioning problem. As documented in the literature, the object partitioning problem involves partitioning a set of $|\mathbb{P}|$ objects into $|\mathbb{N}|$ groups or classes, where the main aim is to partition the objects into groups that mimic an underlying unknown grouping. In other words, the objects which are accessed together must reside in the same group [52]. In the special case when all the groups are required to contain the same number of objects, the problem is also referred to as the Equi-Partitioning Problem (EPP). Many solutions involving LA have been proposed to solve the EPP, but the most efficient algorithm is the *Object Migrating Automaton* (OMA) [52]. The latter was first proposed by Oommen and Ma [52], and some modifications were added by Gale *et.al.* [18] to create the *Adaptive Clustering Algorithm* (ACA). Since the OMA is, in one sense, the prior art on which our present solution is built, we briefly describe its design here, before detailing the graph partitioning algorithm later in the paper.

3.2 Object Migrating Automaton (OMA)

The Object Migrating Automaton (OMA) is an ergodic automaton that has R actions $\{\alpha_1, \ldots, \alpha_R\}$ representing the possible underlying classes. Each action α_i has its own set of states $\{\phi_{i1}, \phi_{i2} \ldots, \phi_{iM}\}$, where M is the depth of memory, and $1 \leq i \leq R$ represents the number of classes. ϕ_{i1} is called the most internal state and ϕ_{iM} is the boundary (or most external) state.

A set of W physical objects $\{A_1, A_2, \ldots, A_W\}$ is accessed by a random stream of queries, and the objects are to be partitioned into groups so that the frequently *jointly*-accessed objects are clustered together. The OMA utilizes W abstract objects $\{O_1, O_2, \ldots, O_W\}$ instead of migrating the physical objects. Each abstract object is assigned to a state belonging to an initial random group but in its boundary state. The objects within the automaton move from one action to another, and so, in this case, all the W abstract objects move around in the automaton. If the abstract objects O_i and O_j are in the action α_h, and the request accesses $< A_i, A_j >$, then the OMA will be rewarded by moving them towards the most internal state ϕ_{h1} (Figure 3(a)).

On the other hand, a penalty arises if the abstract objects O_i and O_j are in different classes, say α_h and α_g, respectively. Assuming O_i is in $\zeta_i \; \varepsilon \; \{\phi_{h1}, \phi_{h2}, \ldots, \phi_{hM}\}$ and O_j is in $\zeta_j \; \varepsilon \; \{\phi_{g1}, \phi_{g2}, \ldots, \phi_{gM}\}$, they will be moved as follows:

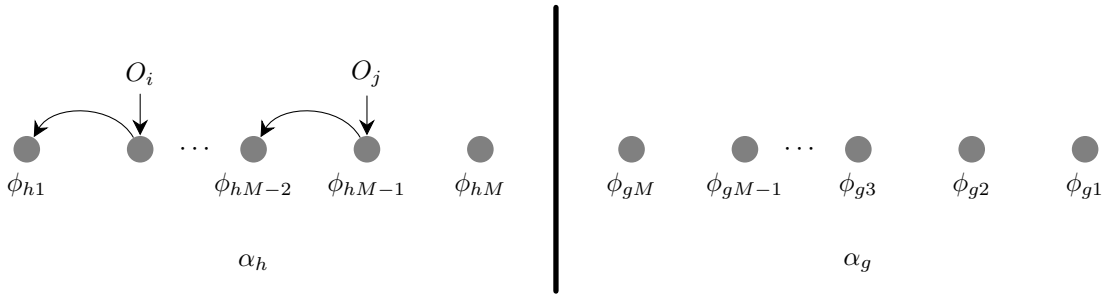
- If $\zeta_i \neq \phi_{hM}$ and $\zeta_j \neq \phi_{gM}$, O_i and O_j are moved one state toward ϕ_{hM} and ϕ_{gM}, respectively (Figure 3(b)).
- If exactly one of them is in the boundary state, the object which is not in the boundary state is moved towards *its* boundary state (Figure 3(c)).
- If both of them are in their boundary states, one of them, say O_i is moved to the boundary state of the other object ϕ_{gM}. In addition, the closest object to them, given in the figure as O_i, is moved to the boundary state ϕ_{hM}, so as to preserve an equal number of objects in each group (Figure 3(d)).

It is important to point out that the random stream of queries contains information about an optimal partition, and the OMA attempts to converge to it. The automaton is said to have converged when all the objects in a class are in the deepest (or second deepest) most-internal state.

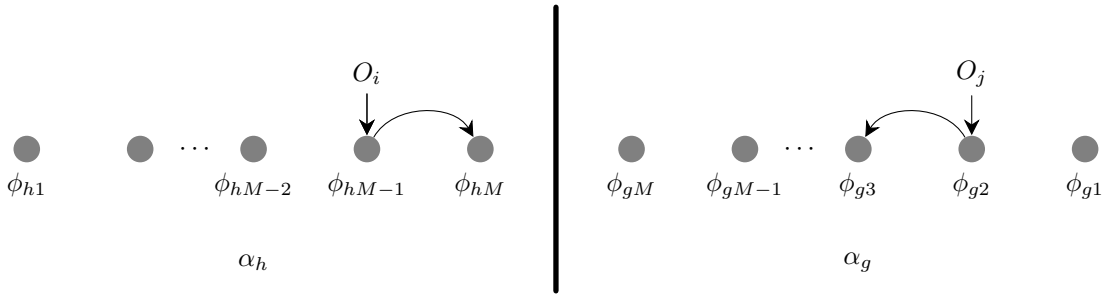
The OMA can be improved by the following: Assume that a pair of objects $< A_i, A_j >$ is accessed, where O_i is in the boundary state, while O_j is in a non-boundary state. In this case, a general check should be made to locate another object in the boundary state of the partition containing O_j. If there is an object, then swapping is done between this object and O_i in order to bring the two accessed objects into the same partition. In turn, instead of waiting for a long time to have these accessed objects in the same partition, the convergence speed can be increased by swapping the objects into the right partitions.

The formal algorithm for the OMA is found in [18, 52], and omitted here in the interest of space.

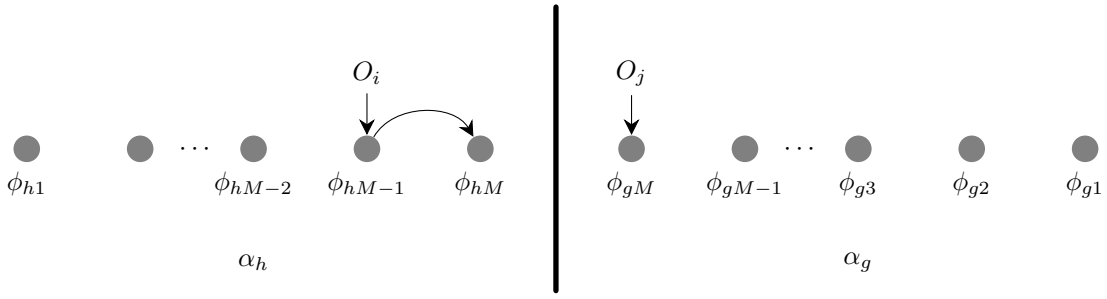
The question that begs attention is that of explaining why we have chosen to use LA, and in particular, the OMA to solve the problem at hand. The reason is, quite simply, the following: The underlying problem that we deal with is that of "partitioning" based on a "infinite" data stream. This problem is *NP*-Hard because of the number of possible partitions. Further, it involves estimating an exponential number of joint access probabilities. The first prior solution to this involved a hill-climbing technique and required hundreds of thousands of query pairs. The algorithm that succeeded this was the so-called Basic Adaptive Method, and it required tens of



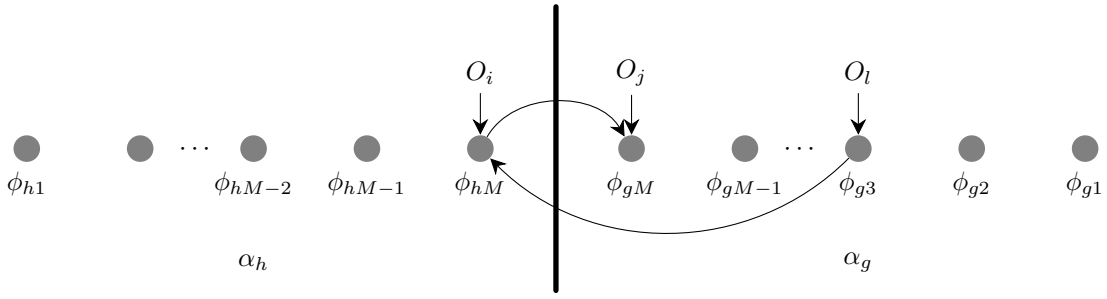
(a) On reward: Move the abstract objects towards the extreme states.



(b) On penalty: Move the objects towards their boundary states (Case 1).



(c) On penalty: Move the objects towards their boundary states (Case 2).



(d) On penalty: Move the object to the boundary state of the other action (Case 3).

Figure 3: Transition rules for a two-action OMA. On reward, we move the abstract objects towards the extreme states. On penalty, we move the abstract objects towards their boundary states or move them to the boundary state of the other action. All these scenarios are depicted in the four sub-diagrams.

thousands of query pairs. The solution that has been the benchmark for three decades is LA-based, and is, indeed, the OMA, which is orders of magnitude faster than the BAM. This motivates the choice of LA-based and OMA-based solutions for this problem because the application of *any* version of the OMA to this application domain has been unreported.

4 Existing Solution

4.1 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 [13, 39, 21, 47]. In this paper, 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 paper are: the legacy Tree data center network architecture, the Fat-tree and the VL2 data center network architecture.

4.2 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 N rows and N columns 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 paper 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.

5 Proposed VM Clustering Algorithm

We shall now explain the strategy that we use to resolve the assignment of VMs. The reader must first of all appreciate that the assignment of VMs is essentially a clustering exercise. Indeed, since the traffic patterns are not known *a priori*, the assignment algorithm must learn the best assignment by inferring this from the real-time traffic. In other words, the VMs that communicate much with each other must be in the close proximity of each other, while those that communicate less could be, potentially, placed further apart. With a little insight, one can see that this is precisely equivalent to the problem of partitioning the nodes of a graph into subsets based on some pre-defined similarity criteria. This is exactly the paradigm that we invoke.

Our proposed VM clustering algorithm is based on Oommen's Graph Partitioning Using Learning Automata (GPLA) [54] algorithm. That being said, the GPLA, in and of itself, is not directly applicable to our application domain. Rather, we shall see that it has to be adapted to resolve VM assignment. The GPLA attempts to solve the Graph Partitioning Problem (GPP) [8, 15, 30] by using the toolbox that incorporate stochastic Learning Automata (LA), which 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. In this paper we deal with a version of the GPP in which all the sub-partitions are of equal size, and this is precisely the so-called Equi-Partitioning Problem (EPP)[8]. The best solution to the EPP is the so-called 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.

As we will explain later in the section explaining the experimental results, the algorithm adapted for this work will read the set of 1,600 nodes or vertices distributed over 16 sub-partitions, also referred to as groups or arms, and deliver as its output the final solution of the corresponding graph partitioning problem. This will be achieved by adopting the OMA used in Oommen's algorithm. The strategy will involve checking pairs of vertices that are randomly selected by the algorithm in order to determine whether they are connected "significantly", based on which they will be either rewarded or penalized depending on the corresponding conditions ofconnectivity.

We assume that we are given the symmetric VM traffic matrix, D. In order to determine whether the nodes are connected "significantly", we specify two important thresholds, *SimilarityThreshold* and *DissimilarityThreshold*, calculated by the following formulae, both of which involve a user-defined constant ρ:

$$SimilarityThreshold = (1 + \rho) * MeanEdge, \text{and} \tag{2}$$

$$DisimilarityThreshold = (1 - \rho) * MeanEdge. \tag{3}$$

As mentioned above, ρ is a user-defined parameter. With regard to its value, the literature pertaining to the OMA recommends a value of ρ equal to 0.25. Clearly, ρ controls the similarity and dissimilarity measure, which in turn control the decisions for reward, penalty and "inaction" of the different pairs of objects. Thus, in all our experiments, it was set to the fixed value of 0.25. An interesting avenue of research is that of designing a meta-algorithm that can control the value of ρ. Further, the *MeanEdge* value was calculated by computing the

[8]By assuming that the "graph" is equi-partitioned, we can invoke a solution to the EPP to resolve it. The problem remains unsolved if the relative sizes of the respective sub-graphs are unknown. Clearly, any new solution to a non-equi-partitioned version of the OPP will also be applicable for the corresponding VM assignment problem.

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 distinct sub-partitions they will be penalized since this state is unfavorable. If, however, the nodes are found in the same sub-partitions 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, i.e., towards the *MaximumCertainty* state. When the nodes reach the outer boundaries of their sub-partitions they could be made to migrate from their current sub-partitions and moved to a better one. This process will be repeated until the maximum number of iterations is reached.

Pseudocode for the VM Clustering Algorithm

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 the states
- G: Function, which partitions the set of states for the sub-partitions

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.

ρ is a parameter used to determine the similarity or dissimilarity of the vertices. M=100.

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)
            end
            else
                PenalizeSimilarNodes(i,j)
            end
        end
        else
            if C_ij < (1 − ρ) · Mean_Edge then
                if v_i and v_j are in same sub-partition then
                    PenalizeDissimilarNodes(i,j)
                end
            end
            else
                Pass
            end
        end
    end
end
```

Algorithm 1: The Pseudocode for the Function ClusterVMs

Data: Node indices i and j, where ω_i and ω_j are the state indices of similar nodes in the same sub-partition.

```
if ω_i mod M ≠ 1 then /* i is not in the most internal state */
    ω_i = ω_i − 1
end
if ω_j mod M ≠ 1 then /* j is not in the most internal state */
    ω_j = ω_j − 1
end
```

Procedure The Pseudocode for the Function RewardSimilarNodes(i,j)

Data: Node indices i and j, where ω_i and ω_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
  end
  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 */
    $\omega_j = (\omega_i div M) \cdot M$ /* 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 ω_i
    ω_t = temp /* move v_t to the old state of v_j */
    else
    end
    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 */
      $\omega_i = (\omega_j div M) \cdot M$ /* 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 ω_j
      ω_t = temp /* move v_t to the old state of v_i */
    end
  end
end

Procedure The Pseudocode for the Function PenalizeSimilarNodes(i,j)

5.1 Enhancement of the OMA algorithm for solving our problem

Our primary objective is to assign the VM clusters to the server racks in a manner that decreases the total cost of communication. *This* assignment problem will be treated as a Quadratic Assignment Problem (QAP) [28, 34, 39, 57], known to be one of the most difficult combinatorial optimization problems. The assignment of the 16 clusters to the available 16 server racks that gives the lowest total communication cost will be considered as the best assignment. 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 astronomically large number (i.e., 16!) the exhaustive search approach in order to find the best solution is computationally infeasible. Instead, we seek a solution that is "closest" to being optimal from among a specific pool of solutions. In order to find such an optimal solution to QAP, we will invoke a simulated annealing (SA) phase [12, 33]. SA ensures that the algorithm does not get trapped in a local minimum and that it will be given a chance to explore a wider range of possible solutions by visiting even the inferior solutions with constantly decreasing probability [16].

Setting the Initial Cluster Placements

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. To be more specific, the S matrix denotes the cluster-to-cluster traffic matrix. The reader must observe that the clusters are obtained in the first phase via the OMA.

```
Data: Node indices i and j where ω_i and ω_j are the state indices of dissimilar nodes in the same
      sub-partition
if (((w_i mod M) ≠ 0)and((w_i mod M) ≠ 0))) then
    ω_i = ω_i + 1 /* both are in internal states                                   */
    ω_j = ω_j + 1
end
else
    if ω_i mod M ≠ 0 then /* v_j is in internal state                              */
        ω_i = ω_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
            ω_p = state of node closest to boundary in this current sub-partition
            TempState2 = ω_p
            ω_j = (ω_p divM + 1) · M/* move v_j to new sub-partition            */
            ω_p = TempState1 /* move v_p to v_j's old state                       */
            New_Cost = EvaluateCost of current partitioning
            if New_Cost > Prev_Cost then
                ω_p = TempState2 /* change is not superior                       */
                ω_j = TempState1 /* undo it                                      */
            end
            else /* this change is superior                                      */
                Prev_Cost = New_Cost /* retain it                               */
            end
        end
        else /* v_j is in internal state                                         */
            ω_j = ω_j + 1 /* update state of v_j                                 */
            TempState1 = ω_i /* store state of v_i                               */
            Prev_Cost = EvaluateCost of current partitioning
            for all remaining K − 1 partitions do
                ω_p = state of node closest to boundary in this current sub-partition, α_Z
                TempState2 = ω_p
                ω_i = (ω_p divM + 1) · M /* move v_i to new sub-partition       */
                ω_p = TempState1 /* move v_p to old state of v_i                 */
                New_Cost = EvaluateCost of current partitioning
                if New_Cost > Prev_Cost then
                    ω_p = TempState2 /* change is not superior                   */
                    ω_i = TempState1 /* undo it                                  */
                end
                else /* this change is superior                                  */
                    Prev_Cost = New_Cost /* retain it                           */
                end
                /* move v_t to the old state of v_i                             */
            end
        end
    end
end
```

Procedure The Pseudocode for the Function PenalizeDissimilarNodes(i,j)

It is thus straightforward to compute the S matrix based on the D matrix describing the VM traffic and the results of the OMA partitioning. Subsequently, 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 states of the VM clusters. Observe that the initial placement will be an already-improved placement when compared to randomly-assigned VM clusters, and this helps the cluster placement algorithm to find an even more superior solution. In this regard. the total cost of communication will be calculated by summing all the edges multiplied by their corresponding communication costs using the following formula:

$$Comm_{Total} = \sum_{i,j=\cdots,n} D_{ij} \cdot C_{\pi(i)\pi(j)}, \tag{4}$$

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 that the nodes V_i and V_j are assigned to.

5.2 The Simulated Annealing Process

Once the initial placement has been established and the initial total cost of communication has been 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, calculated as below:

$$P = e^{-\frac{\Delta}{T}}, \tag{5}$$

where $\Delta = TotalCost_{new} - TotalCost_{old}$, is 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 Figure 4) will ensure that the algorithm does not get stuck in the local minimum and falsely assume that the optimal result has been obtained. Initially, the probability P will have a higher value implying that the algorithm will accept inferior results more frequently. However, as the temperature T decreases over time, the value of P will gradually decrease and the algorithm will be less and less likely to accept inferior results. The simulated annealing technique will render to the cluster placement algorithm the potential of exploring a wider range of the possible solutions space. Ultimately, it will yield the most superior solution encountered.

Pseudocode of the Cluster Placement Algorithm

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

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 */` $BestTotalCost = TotalCost_{BestState}$
        end
        else
            Retain $BestState$
        end
        if $TotalCost_{TempState} > TotalCost_{BestState}$ **then**
            $P = e^{-\frac{\Delta}{T}}$`/* Calculate probability P */`
            if $P < RandomValue$ **then**
                $BestState = TempState$ `/* go to the new state */`
                $BestTotalCost = TotalCost_{BestState}$
            end
            else
                retain $BestState$
            end
        end
    end
end

Algorithm 2: The Pseudocode for the algorithm to place clusters.

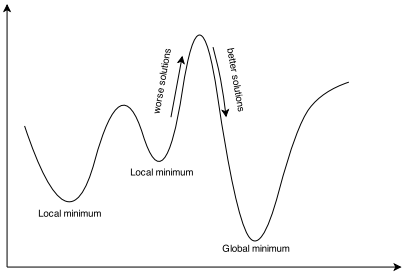


Figure 4: This diagram displays the process followed by Simulated Annealing.

6 Experiment Settings

In order to test the proposed algorithms on various *kinds* of data sets and to be able to retrieve reliable results, we performed two sets of experiments[9]. They were conducted with two different sets of 1,600 VMs selected from the obtained traffic traces. We also understood the importance of having a plan by which one could perform the measurement and evaluation of the experimental results.

6.1 Experiment Set A

In this experiment set, we conducted three experiments on each of the simulated data center networking architectures. In each case, we conducted a separate experiment 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 experiments titled "Set A", we randomly selected the set of 1,600 VMs from the collected traffic traces, with the expectation that this set of 1,600 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.

Experiment A1: Tree DCN

The first set of tests were run on the most widely-used legacy three-tier Tree DCN model. The Tree DCN model (see Figure 5) contained 16 server racks. Each server rack was assumed to be able to accommodate 100 VMs. The server racks constituted four groups, where each group consisted of four server racks connected to a single access layer (or Layer 1) switch. The four access switches were connected to Layer 2 - the aggregation layer switches. The aggregation layer, in turn, consisted of four switches. However only two of the four switches were presumed to be active, while the other two were in a so-called "Standby" mode. Finally, there was one active and one standby switch on the core layer at the top level of the data center network.

In the interest of brevity and space, we merely cite some of the important results obtained in the experiment. In particular, we record $T_{RandTreeA}$, the average total cost of communication with the randomly placed VMs for the Tree DCN, $T_{GpTreeA}$, the average total communication cost after the optimization with the VM clustering algorithm for the Tree DCN, and $T_{QapTreeA}$, the average total communication cost after executing the cluster placement algorithm, where the 'A' in these notations refers to the Experiment Set A.

The simulated model of a three-tier data center networking architecture (see Figure 5) 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 In this work, we utilized the following communication cost matrix (see Eq. (6)) for the Tree DCN:

[9]One anonymous Referee had requested that we put the data in the public domain, because it could help someone else working independently to apply the findings, and to, maybe, subsequently contribute to this optimization. We are grateful for this request, and the data is available at http://pages.cs.wisc.edu/~tbenson/IMC10_Data.html. The traces from three of the data centers given in the above link are studied in [5].

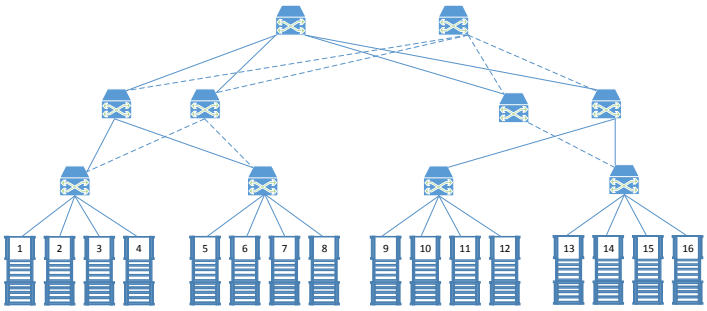


Figure 5: The Tree data center network model used in the simulations.

$$C = \begin{bmatrix}
0 & 1 & 1 & 1 & 3 & 3 & 3 & 3 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\
1 & 0 & 1 & 1 & 3 & 3 & 3 & 3 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\
1 & 1 & 0 & 1 & 3 & 3 & 3 & 3 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\
1 & 1 & 1 & 0 & 3 & 3 & 3 & 3 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\
3 & 3 & 3 & 3 & 0 & 1 & 1 & 1 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\
3 & 3 & 3 & 3 & 1 & 0 & 1 & 1 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\
3 & 3 & 3 & 3 & 1 & 1 & 0 & 1 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\
3 & 3 & 3 & 3 & 1 & 1 & 1 & 0 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\
5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 0 & 1 & 1 & 1 & 3 & 3 & 3 & 3 \\
5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 1 & 0 & 1 & 1 & 3 & 3 & 3 & 3 \\
5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 1 & 1 & 0 & 1 & 3 & 3 & 3 & 3 \\
5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 1 & 1 & 1 & 0 & 3 & 3 & 3 & 3 \\
5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 3 & 3 & 3 & 3 & 0 & 1 & 1 & 1 \\
5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 3 & 3 & 3 & 3 & 1 & 0 & 1 & 1 \\
5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 3 & 3 & 3 & 3 & 1 & 1 & 0 & 1 \\
5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 3 & 3 & 3 & 3 & 1 & 1 & 1 & 0
\end{bmatrix} \tag{6}$$

Each row and each column in the cost matrix corresponds to a single server rack. For example, Row i and Column j correspond to the respective rack numbers (see Figure 5), and thus, for example, the communication cost for the traffic between server rack number 1 and 10 can be found in corresponding entry of the communication cost matrix and equals 5.

Experiment A2: Fat-tree DCN

The next experiment on which we conducted experiments was the relatively-recently-proposed data center network architecture PortLand [47], based on a so-called Fat-tree network topology. In the Fat-tree DCN model (see Figure 6) we used four pods out of 16 switches. Each pod contained 4 switches and were connected to all the available 4 core switches. The traffic between the 1,600 VMs was then simulated as per this Fat-tree model. The 1,600 VMs were divided in 16 sub-partitions of equal sizes each containing 100 VMs. Each sub-partition was 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-tı ommunication cost
after executing the clus in the abbreviation
refers to the Experimen

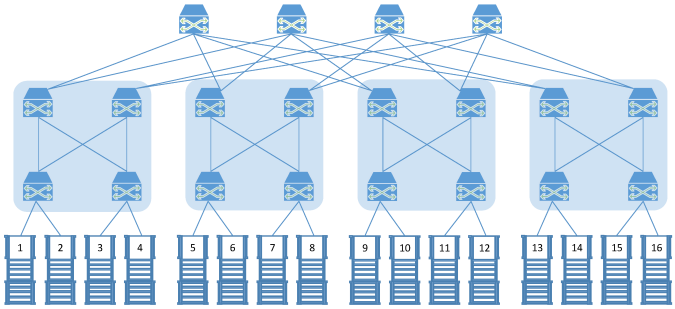


Figure 6: The Fat-tree data center network model used in the simulations.

Cost matrix for the Fat-tree DCN In this case, the cost of communication between the neighbor pairs of racks R_1 and R_2 was 1. However, the cost across the neighboring pairs, for example, between R_2 and R_3, in the same pod was 3. As opposed to this, the cost of communication *across* the pods was 5. A cost matrix C was calculated for the experiment with Fat-tree DCN based on the number of the network elements (switches) that the traffic has to travel through in order to reach its destination from one server rack to another. Thus, the cost matrix used for the experiment is displayed in Eq. (7) below:

$$C_{ij} = \begin{bmatrix} 0 & 1 & 3 & 3 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 1 & 0 & 3 & 3 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 3 & 3 & 0 & 1 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 3 & 3 & 1 & 0 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 0 & 1 & 3 & 3 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 1 & 0 & 3 & 3 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 3 & 3 & 0 & 1 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 3 & 3 & 1 & 0 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 0 & 1 & 3 & 3 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 1 & 0 & 3 & 3 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 3 & 3 & 0 & 1 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 3 & 3 & 1 & 0 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 0 & 1 & 3 & 3 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 1 & 0 & 3 & 3 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 3 & 3 & 0 & 1 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 3 & 3 & 1 & 0 \end{bmatrix} \quad (7)$$

Experiment A3: VL2 DCN

The third and the last experiment that was conducted was on a simulated VL2 [21] data center network architecture. The VL2 is a newly proposed DCN and it shares many similarities with the traditional Tree DCN.

However, the main difference between VL2 (see Figure 7) and the 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 is routed back to the access layer and then to its destination. This difference will increase the cost of communication between the neighboring access layer switches and also between the groups of the server racks associated with the given access switches.

The VL2 model that we used consisted of 12 switches and 16 server racks. The racks constituted four groups each consisting of 4 racks. Each group was connected to a single access layer switch. The 1,600 VMs were thus accommodated by the VL2 model, and were divided into 16 groups of 100 VMs each, with each server rack being able to host 100 VMs.

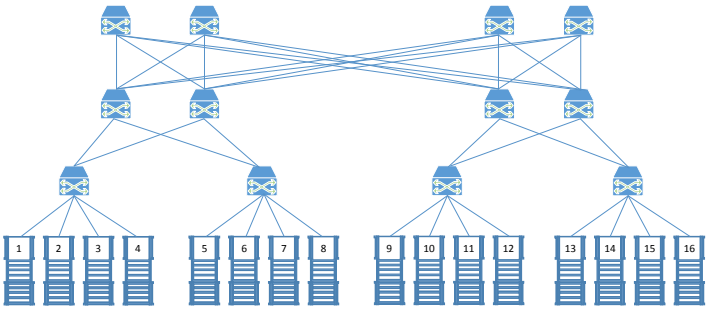


Figure 7: The VL2 data center network model used in the simulations.

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

Cost matrix for the VL2 DCN The resulting cost matrix for VL2 (see Eq. (8)) was 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.

$$C_{ij} = \begin{bmatrix} 0 & 1 & 1 & 1 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 1 & 0 & 1 & 1 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 1 & 1 & 0 & 1 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 1 & 1 & 1 & 0 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 0 & 1 & 1 & 1 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 1 & 0 & 1 & 1 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 1 & 1 & 0 & 1 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 1 & 1 & 1 & 0 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 0 & 1 & 1 & 1 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 1 & 0 & 1 & 1 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 1 & 1 & 0 & 1 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 1 & 1 & 1 & 0 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 0 & 1 & 1 & 1 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 1 & 0 & 1 & 1 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 1 & 1 & 0 & 1 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 1 & 1 & 1 & 0 \end{bmatrix} \quad (8)$$

6.2 Experiment Set B

The Experiment Set B consisted of the following experiments:

1. Experiment B1: Tree DCN
2. Experiment B2: Fat-tree DCN
3. Experiment B3: VL2 DCN

The Experiment Set B repeated the experiments described in the previous section refereed to as *Experiment Set A*. The difference between the two experiments consisted in 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 here behave with a different set of VMs and with distinct traffic patterns. The new set used in the Experiment Set B was specifically chosen to contain the IP addresses of those who "talk" to each other with relatively higher traffic rates compared to the VM set used in the Experiment Set A. This was achieved by sorting (in descending order) the file containing the list of traffic rates (i.e., D_i, j), in terms of these rates, and by then removing the communicating pairs with significantly high traffic rates. This had the effect of smoothing the graph of the distribution of the edges between the VMs participating in the experiments.

We conducted three experiments in this scenario: Experiment B1 on a Tree, Experiment B2 on a Fat-tree, and Experiment B3 on VL2 data center networking architectures, and in each case we conducted the experiments with the exact same parameters as in Experiment Set A. Subsequently, the intracluster traffic experiment was

also conducted with the same parameters as in the Experiment Set A. The notations used for tge total cost of communication is analogous to the ones used in the Experiment Set A, except that they will have suffix 'B' instead of 'A'. Thus, for example, the total cost of communication with randomly assigned VMs in the Tree experiment for the experiment set B (Experiment B1) will be denoted as $T_{RandTreeB}$.

6.3 Measurement and Evaluation

In order to be able to assess how the problem statement has been addressed, it is important for us to reliably evaluate the performance of the proposed VM clustering and cluster placement algorithms, and their respective impacts on the total cost of the communication in the data center models. Since the experiments done in this paper are conducted on virtual models of data center networking architectures, there is no possibility of connecting the networks to physical devices and directly measuring real-time bandwidth usage on "real" links prior to and and after the optimization. Instead, we will have to resort to other methods of measurement and evaluation. Indeed, it is our expectation that that the total communication cost of for the whole data center will decrease after both the VM clustering and cluster placement algorithms have been implemented. We also expect the average intracluster (the traffic between the member VMs inside a group) traffic will increase after invoking the VM clustering algorithm because the goal of this phase was to place VMs with high mutual traffic within the same clusters. Simultaneously, we would expect the intercluster traffic to decrease as the result of our optimization.

In order to compare initial and optimized states of the system, we first established a baseline setting by randomly assigning VMs to the clusters and by thereafter computing and storing both the intracluster and intercluster traffic data, and the total cost of communication for the whole system. There is, however, a probability that we could obtain extreme results due to the sampling errors, since the highly communicating VM pairs could end up in the same clusters hence yielding a relatively low total communication cost in the baseline placement itself. This would have the effect of implying that the clustering and assignment algorithms were not sufficiently expedient. In order to avoid such a scenario, and to allow us to obtain a baseline that could be considered as a reliable-average un-optimized system, we generated multiple[10] ($N = 35$) randomly-distributed VM states. The average values for the total cost of communication was thus calculated and stored. The average intracluster and intercluster traffic was also computed and stored for subsequent analysis. Such an approach was expected to reduce the chance of random sampling error distorting the results. In order to reduce the variation in the mean values and to obtain data that was as reliable as possible, the tests were repeated 35 times, and the obtained sets of (35) results were subsequently used in order to calculate the statistical values.

The results of the 35 experiments were stored in comma-separated text files with timestamps and experiment names so that they could be subsequently easily accessed and used for analysis and plotting. The goal was that the algorithms would have been able to optimize the randomly-generated systems and thereafter calculate the new total cost of communication and the difference between the un-optimized and the optimized systems, along with various other indications such as the number of times the reward or penalty procedures had to be invokedetc. One of the indicators of the graph partitioning algorithm's performance was the intracluster communication before and after the graph partitioning. To evaluate this, the traffic between each element inside a cluster was summed up. Another indicator to measure the performance of the VM clustering and

[10]Since at least 30 test samples is, usually, required in order to obtain reliable statistics, our test sample size was set to be 35.

cluster placement algorithms was the time needed for them to converge and the stability in variation of the results.

7 Experiment Results and Analysis

After running the two sets of experiments each with three sub-experiments for the three chosen data center network architectures the results were obtained and stored for further analysis.

7.1 Analysis: Experiment Set A

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

Experiment A1 was conducted on the Tree data center network architecture. The simulator was run by specifying the cost matrix for the Tree DCN. The simulator read the 35 .pkl files containing the 35 randomly placed VM sets and executed the VM clustering and cluster placement algorithms 35 times for each of the 35 .pkl files. Thus the optimization algorithms were run totally 1,225 times for the Tree experiment. The number of iterations for the VM clustering algorithm was calculated to be 82,920 each time. The *SimilarityThreshold* was calculated to be 1020,756.33, and the *DissimilarityThreshold* was 612,453.80. The initial temperature sent to the cluster placement algorithm for SA was $T = 100,000$. Each test took roughly 3 minutes. The whole experiment with 1,225 tests took approximately 19 hours.

Experiment A2 was conducted on the Fat-tree data center network architecture. The simulator was launched in a parallel mode with Experiment A1 by specifying the cost matrix for the Fat-tree DCN. The simulator read the same 35 .pkl files containing the 35 randomly placed VM sets and executed the VM clustering and cluster placement algorithms 35 times for each of the 35 .pkl files. Again, the optimization algorithms were run 1,225 times for the Fat-tree experiment with the same values for the threshold and the maximum number of iteration.

Experiment 3 was conducted on the VL2 data center network architecture. The third parallel instance of the simulator was launched by specifying the cost matrix for the VL2 DCN. The simulator read the same 35 .pkl files containing the 35 randomly placed VM sets. All the settings and parameters were identical to the ones used in Experiment A1 and Experiment A2. The whole experiment with its 1,225 tests took approximately 16 hours.

The results showed that the traffic-aware consolidation of the VMs had a significant impact on the total communication cost. The results also demonstrated that the cluster placement algorithm further decreased the total communication cost. 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.

Experiment A1: Tree Analysis

When one considers at the baseline total communication cost, the first observation is the high variance (the standard deviation is is as high as 12.11% of the mean) in the distribution of total costs (see Table 1)[11]. This can be explained by the fact that the cost matrix for the Tree DCN (see Eq. (6)) can cause a higher variation in the communication cost as the result of moving clusters with significantly high traffic slightly away or closer

[11] The first row in this table and all the other similar tables corresponds to the "random placement" setting.

to their pairs with whom they exchange significantly high traffic. By way of comparison, the cost matrices for the Fat-tree and VL2 configurations (see Eq. (7) and (8)) are more uniform.

The impact of the VM clustering algorithm using Oommen's graph partitioning technique is obvious when observing the plotted graphs. The VM consolidation decreases the total communication cost by 85.09% (from 11,744 to 1,751 GB), which is a significant improvement. It further stabilizes the variance as well (see Table 1). One should also observe that at this juncture, the clusters are not assigned to the racks in manner that is "closest" to being optimal. Indeed, after the cluster assignment with simulated annealing the total communication cost drops further to 0.49 GB which amounts to a decrease of 97.17% when compared to the total cost of the consolidated (clustered) VMs, and to a overall decrease of 99.58% when compared to the total communication cost associated with randomly distributed VMs.

The significant advantage of using our algorithms is obvious!

	Mean	St.dev	$\Delta_{Prev.mean}$	$\Delta_{Overall}$
$T_{RandTreeA}$	11,744	1,422	—	—
$T_{GpTreeA}$	1,751	0.062	-85.09%	-85.09%
$T_{QapTreeA}$	0.49	0.003	-97.17%	-99.58%

Table 1: Change in the total cost of communication in the Tree set-up in Set A.

During the 82,920 iterations, the VM clustering algorithm invoked the *RewardSimilarNodes* procedure, on average, 5,622 times, while the *PenalizeSimilarNodes* was invoked 1,576 times, on average. The average number of the times *PenalizeDissmiilarNodes* was invoked was 2,213. The graph partitioning process took on average 3.4 seconds to complete.

This means that for most of the time, the algorithm picked nodes that were not significantly connected. This occurred in 9,411 out of the total 82,920 iterations. Thus, the graph partitioning algorithm was "idle" 88.65% of the total number of iterations when the picked edges were regarded to be neither similar nor dissimilar.

The results also showed that the cluster placement algorithm used the inferior configurations 149.4 times on average (standard deviation 16.8) out of 100,000 iterations. It took the SA process (on average) 37.7 seconds to complete the 100,000 iterations which was a significantly longer time compared to what the VM clustering algorithm used. This can be explained by the fact that for most of the time (88.65%), the VM clustering algorithm did not have to conduct any time consuming operations, while the cluster placement algorithm executed time intensive calculation jobs for each of the 100,000 iterations.

Experiment A2: Fat-tree analysis

The results of the Fat-tree experiment revealed (see Table 2) that the baseline total communication cost in this DCN model was more stable when compared to the baseline total communication cost for the Tree (in terms of the variances). The standard deviation of the 35 tests was, on average, 8.04% of the mean. This can be explained by examining the cost matrix for the Fat-tree set-up (see Eq. (7)), which was relatively uniform when compared to the cost matrix of Tree. This observation might also be a good explanation for the fact why there is a lesser 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 with each other from the server rack groups belonging to distinct access layer switches. However, there is greater change in cost of communication for VM pairs migrated from one rack to another in cases of the VMs communicating with each other within the same access layer switch their respective rack groups (see Eq. (7)).

In this case, the total cost of communication measured with the randomly distributed VMs decreased significantly after the set of nodes were graph partitioned by the VM clustering algorithm. The average total cost was reduced by 85.14%, i.e., from 14,406 to 2,140 GB (Gigabyte).

The VM clustering algorithm executed 82920 iterations and invoked the *RewardSimilarNodes* procedure, on average, 5,619 times. The *PenalizeSimilarNodes* procedure was invoked 1,582 times, on average, while the *PenalizeDissimilarNodes* was invoked on average 2,189 times. The average time used to achieve the GP 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 were not considered to be 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. This further decreased the average total communication cost to 96.82% from 2,140 to 0,68 GB. Thus, the overall average total cost was decreased by 99.52% from its initial value of 14,406 to the 0,68 GB after the cluster placement. Again, the power of using our algorithms is clear.

	mean	st.dev	$\Delta_{Prev.mean}$	$\Delta_{Overall}$
$T_{RandFtreeA}$	14,406	1,158	—	—
$T_{GpFtreeA}$	2,140	0.074	-85.14%	-85.14%
$T_{QapFtreeA}$	0.068	0.003	-96.82%	-99.52%

Table 2: Change in the total cost of communication in the Fat-tree set-up in set A.

The cluster placement algorithm accepted on average 1,733.76 worse configurations during the SA process which took 38.85 seconds to complete, on average.

Experiment A3: VL2 Analysis

The observed average total communication cost for the randomly distributed VMs in the VL2 set-up was 13,586,759,288.7 with a standard deviation of 1,414,884,915.62. The variance (10.41% of the mean) was thus higher than the Fat-tree set-up, but lower than the Tree set-up. This result can be explained again by comparing the cost matrices of the three data center architecture models displayed in the approach section.

The average total cost of communication decreased by 85.20% after the VM clustering from 13,586 to 2,010 GB. During the VM clustering the GP process invoked *RewardSimilarNodes*, on average, 5,619 times. The *PenalizeSimilarNodes* procedure was invoked 1,581 times and the *PenalizeDissimilarNodes* procedure was invoked 2,204 times, on average. The graph partitioning process took an average of 3.34 seconds to complete.

The cluster placement algorithm further decreased the total cost of communication by 97.02% compared to the total cost of communication achieved after the GP. The new total cost went down to an average of 0.59

	mean	st.dev	$\Delta_{Prev.mean}$	$\Delta_{Overall}$
$T_{RandVl2A}$	13,586	1,414	—	—
T_{GpVl2A}	2,010	0.53	-85.20%	-85.20%
$T_{QapVl2A}$	0.59	0.03	-97.02%	-99.56%

Table 3: Change in the total cost of communication in the VL2 set-up in Set A.

GB which, when compared to the initial total cost of communication represented a decrease of an incredible 99.56% decrease. The SA took, on average, 39.6 seconds while in 1,409 times out of the 100,000 iterations, the algorithm chose an inferior state.

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 the VM clustering and the subsequent quadratic assignment. In order to understand the causes for this significant improvement, it is crucial to observe the changes in the intracluster traffic and the changes in the intercluster traffic.

The results obtained during the intracluster and intercluster experiments clearly demonstrate how the intracluster traffic was increased as a result of the GP process achieved using the GPLA algorithm. Indeed, Figure 8 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.

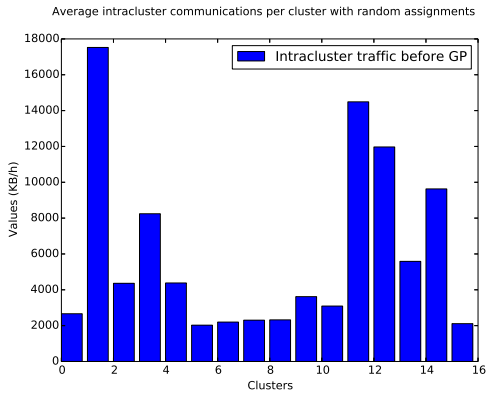


Figure 8: Intracluster traffic in the 16 clusters *before* executing the GP in Set A.

Figure 9 illustrates how the intracluster traffic looked within the same 16 clusters shown in the Figure 8, after the VMs were consolidated with the VM clustering algorithm using the GP technique.

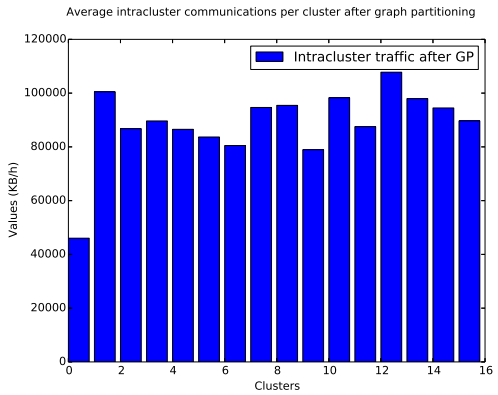
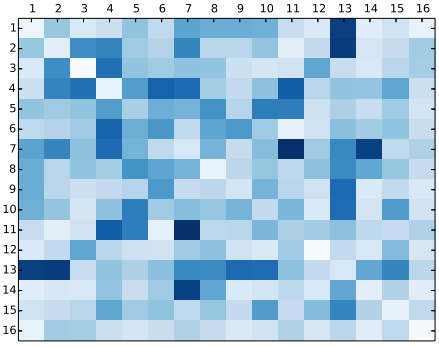


Figure 9: Intracluster traffic in the 16 clusters *after* executing the GP in Set A.

Using the randomly assigned VMs, the aggregate average intracluster communication was 0.06 GB (with a std of 0.0057) which was increased by 1,369.28% up to 0.907 GB after the GP. The reader should thus observe the strength of the GP phase. Already at this stage, without even attempting the intelligent assignment of the clusters to the available server racks, the data center traffic was significantly optimized compared to the state prior to consolidating the virtual machines with the VM clustering algorithm. Such an effect was consistently observed through all the 35 tests conducted for the 35 randomly generated VM sets (1225 times totally). The stability and reliability of the VM clustering algorithm is remarkable.

As expected, while the intracluster traffic increased, the traffic *between* the clusters decreased. The average initial aggregate intercluster traffic was measured to be 0.13 GB, and the traffic between the clusters was not optimized as shown in Figure 11, from which it is evident that the clusters are communicating with each other in a more or less chaotic manner with variable traffic rates. The average aggregate intercluster traffic decreased by 84.92% to 0.002 GB after the VM clustering as the result of consolidating highly communicative VMs in the same clusters.

This is further observed by considering Figure 10. 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 that each cells represent, where the light blue colors represent low values, while greater values are represented by darker shade of blue. Figure 10 shows, for example, that the cell in row 1 and column 13 has a considerably darker hue of blue when compared to the neighboring cell in row 1 and column 13. The figure also reveals that, on averagem 4 clusters had an especially high mutual traffic compared to the rest of the clusters. The diagonal of the figure 10 reveals that none of the clusters had a high internal traffic judging by the light blue color of the diagonal cells.



x

Figure 10: Intra and intercluster traffic heatmap *before* executing the GP in Set A. Here, darker blue regions denote higher intra/inter cluster traffic while lighter blue regions denote lower intra/inter cluster traffic, where the intra cluster traffic is along the diagonal terms while inter cluster traffic is along the non-diagonal terms.

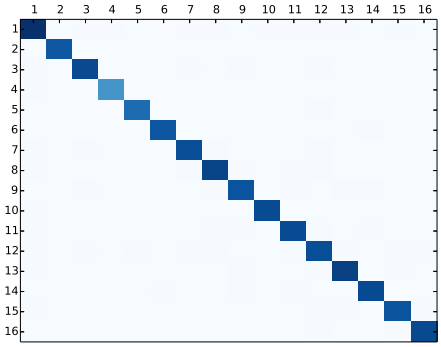


Figure 11: Intra and intercluster traffic heatmap *after* executing the GP in Set A. Again, darker blue regions denote higher intra/inter cluster traffic while lighter blue regions denote lower intra/inter cluster traffic, where the intra cluster traffic is along the diagonal terms while inter cluster traffic is along the non-diagonal terms.

On the other hand, Figure 11 displays a significantly different picture, i.e., the one obtained after the VM clustering was invoked. 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 areas of the graph appear to be more smooth and uniform signifying a decrease in the intercluster communication over the whole matrix.

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 by 1,369.28% while the intercluster traffic decreased by 84.92% at the same time. These changes caused the decrease of the total communication cost by 97.17% in the Tree set-up, by 96.82% in the Fat-tree set-up, and by 97.02% in the VL2 set-up. 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 communication cost by 99.58% in Tree set-up, by 99.52% in the Fat-tree set-up, and by 99.56% in the VL2 data center network architecture models. Figure 12 illustrates the total cost of communication with randomly assigned VMs, after the VM clustering phase and after cluster

placement in all three data center network architecture models.

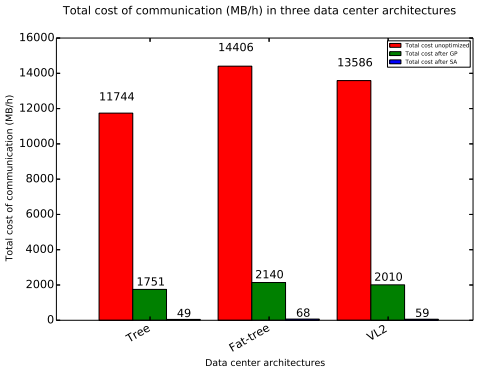


Figure 12: Total cost of communication in all three experiments in Set A.

Without a single exception, in all the three experiments, the effect of the VM clustering and the cluster placement was shown to be extremely effective in consolidating the strongly connected nodes in the same clusters and ultimately greatly decreasing the total communication cost in the data center models. The optimization results were both stable and consistent in all the tests conducted.

Traffic Matrix Characteristics: Set A

A deeper analysis of the traffic matrix used to conduct all three experiments shows that not only are the 2,555,854 out of 2,560,000 (99.84% of the total) values equal to zero in the matrix but the remaining 4,146 are rather unevenly distributed. Indeed, Figure 13 displays the distribution of the 4,146 nonzero elements (or the edges between the communicating VMs) of the traffic matrix. The graph is extremely skewed implying that there are very few values that are "high", while most of the edges are considerably lower. Thus the mean edge is far apart from the median value, implying that merely 8.68% (360 edges) of the total number of the edges terminate over the similarity threshold calculated by using the mean edge value, while the majority (87.51%) of the edges, i.e., 3,628 values, end up below the dissimilarity threshold.

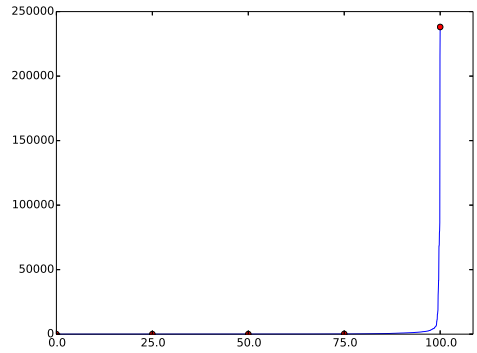


Figure 13: Graph displaying all the edge values in the traffic matrix in Set A in terms of their 25% percentiles.

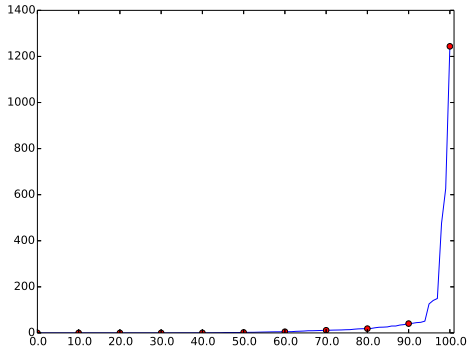


Figure 14: The top 100 edge values in in the traffic matrix in Set A shown with their 10% percentiles.

7.2 Analysis: Experiment Set B

As discussed earlier, the experiments reported above were also conducted on a different set of 1,600 VMs referred to as Set B, but with the same parameters. The purpose of this was to examine how the algorithms would behave when they encounter VMs with a "smoother" communication patterns. In this case, Figures 13 and 14 show that only 10% of the edges were responsible for most of the traffic, while the remaining 90% of the edges contained significantly lower values. This implied that there were very few VMs who communicated heavily with each another while the rest of the VMs demonstrated a moderate amount of mutual communications.

The so-called Set B was intentionally chosen to exclude the VMs that possessed an extremely high level of intercommunication compared to the rest of the nodes. The purpose of this study was to consider how the algorithms that we had developed would perform in a markedly-different environment.

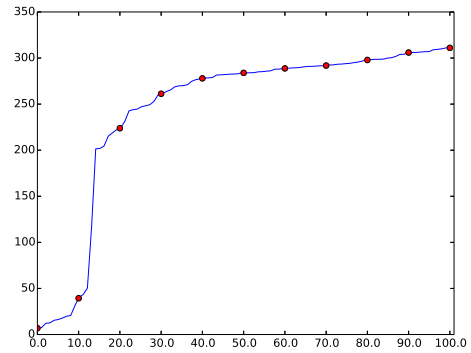


Figure 15: Edge values in the traffic matrix in Set B shown with 10% percentiles.

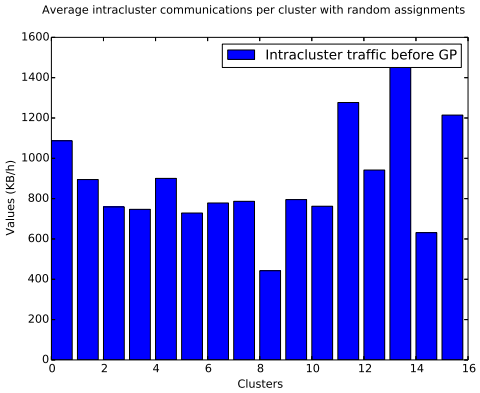


Figure 16: Intracluster traffic in the 16 clusters *before* executing the GP in Set B.

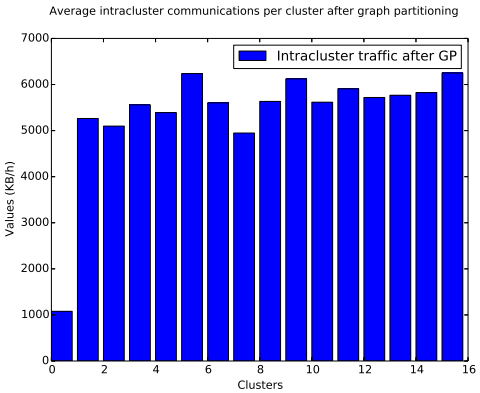


Figure 17: Intracluster traffic in the 16 clusters *after* executing the GP in Set B.

Figure 15 shows the distribution of the edges in the symmetric traffic matrix for the data in 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 case of the VMs in Set A) the increase in the intracluster traffic and the decrease in the intercluster traffic was less pronounced when compared to the corresponding results obtained from those obtained in the case of Set A. The results showed that the intracluster traffic increased by 502.18% while the intercluster traffic decreased by 33.67%.

Consequently, the decrease in total cost of communication after the VM clustering was also moderate compared to the results obtained in the case of the data in Set A. The total cost of communication for the Tree experiment (b1) after VM clustering decreased by 33.74%. After the cluster placement algorithm the total cost of communication decreased by 98.48%. Similarly, the VM clustering algorithm decreased the total communication cost by 33.92% in the Fat-tree experiment (referred to as B2) and by 33.99% in the VL2 experiment (referred to as B3). The cluster placement algorithm improved the results by 98.41% in the Fat-tree set-up (B2), and by 98.47% in the VL2 (B3) set-up.

Without repeating the extensive details we mention thAT In this case, average number of *RewardSimilarNodes* invoked increased to 13,306.48 in the Tree experiment (B1). The average number of *PenalizeSimilarNodes* was 13,013.23 while the average number of *PenalizeDissimilarNodes* was 868.82. During the Fat-tree set-up B2) the number of *RewardSimilarNodes* was 13,330.65 while the average number of *PenalizeSimilarNodes* was

12,994.87 and the *PenalizeDissimilarNodes* was 873.34. Finally, the average number of the *RewardSimilarNodes* increased during the VL2 experiments (B3) as well and was 13,296.16 while the average number of *PenalizeSimilarNodes* invoked was 13,022.85. The average number of *PenalizeSimilarNodes* invoked was 875.69.

Table 4 shows the average total cost of communication in the Tree, Fat-tree and VL2 experiments respectively with the data Set B with randomly assigned VMs, after the GP with the VM clustering algorithm was executed, and finally after the cluster placement algorithm was done. The Table demonstrates the positive effect of the GP and the further improvement in the total cost of communication after the quadratic assignment was done with the use of the cluster placement algorithm.

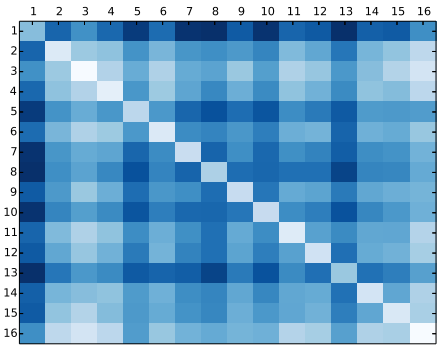


Figure 18: Intra and intercluster traffic heatmap *before* executing the GP in Set B. Here, darker blue regions denote higher intra/inter cluster traffic while lighter blue regions denote lower intra/inter cluster traffic, where the intra cluster traffic is along the diagonal terms while inter cluster traffic is along the non-diagonal terms.

Using the identical notation as in the case of Set A, the heatmap plots of the intra and intercluster communications demonstrate how the traffic patterns look before and after GP was executed in Set B. Figure 18 shows that the traffic rates between the clusters, is on average, higher when compared to the results obtained for Set A, while the intracluster communication (the diagonal cells) is considerably less. Figure 19 shows how the VM clustering algorithm optimizes the traffic. The diagonal cells display how the intracluster traffic was increased while the intercluster traffic was reduced.

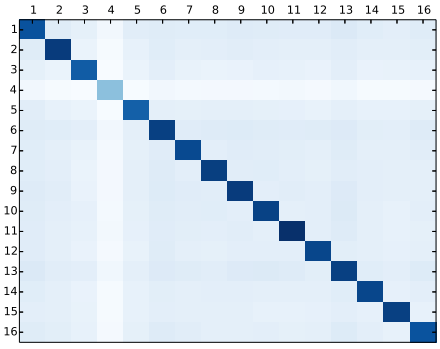


Figure 19: Intra and intercluster traffic heatmap *after* executing the GP in Set B. Again, darker blue regions denote higher intra/inter cluster traffic while lighter blue regions denote lower intra/inter cluster traffic, where the intra cluster traffic is along the diagonal terms while inter cluster traffic is along the non-diagonal terms.

Again, the graph in Figure 20 illustrate how the total cost of communication gradually decreased first by utilizing the VM clustering algorithm and thereafter with the use of the cluster placement algorithm. It is obvious that the VM clustering achieved by GP had a considerable effect on the total communication cost. However, due to the nature of the data set used in the the data Set B, the improvement was less pronounced when compared to Set A.

Table 4 shows how the VM clustering and the cluster placement algorithms reduced the total cost of communication in the Tree (B1), Fat-tree (B2) and VL2 (B3) set-ups for the data in Set B.

	mean	st.dev	$\Delta_{Prev.mean}$	$\Delta_{Overall}$
$T_{RandTreeB}$	1,601	0.18	—	—
$T_{GpTreeB}$	1,061	0.012	-33.75%	-33.75%
$T_{QapTreeB}$	0.024	0.000373	-97.71%	-98.49%
$T_{RandFtreeB}$	1,950	0.017	—	—
$T_{GpFtreeB}$	1,288	0.012	-33.93%	-33.93%
$T_{QapFtreeB}$	0.030	0.00036	-97.61%	-98.42%
$T_{RandVl2B}$	1,835	0.022	—	—
T_{GpVl2B}	1,211	0.010	-33.99%	-33.99%
$T_{QapVl2B}$	0.028	0.000410	-97.68%	-98.47%

Table 4: Changes in the total cost of communication for the various set-ups in the case of the data in Set B.

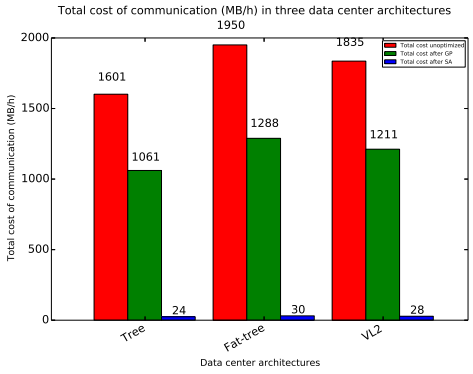


Figure 20: Total cost of communication in all three experiments in in the case of the data in Set B.

Figure 20 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 GP technique, and finally, the average total cost of communication after the clusters were assigned to the server racks in a traffic-aware manner utilizing simulated annealing.

The efficiency of the approach in general cases and for purely random patterns is unknown. However, our solution is advantageous when there are some "chatty" VMs. Indeed, in the realm of data centers, it is becoming

increasingly more common to have dependencies in terms of intercommunication between different applications. One such typical example is the *Map Reduce* application that create a significant amount of inter-traffic or multi-tier web applications running on different VMs.

We now consider the issue of how we could resolve the problem if the size of a few partitions is greater than capacity of a rack. This can be solved via a "Bin Packing" in the initialization phase and small modification of the OMA to accommodate the capacity constraints. To be more specific, in this article, we chose the size of the partition based on the capacity of a rack and the total number of racks in the data center. In this sense, the problem is rather an equi-partitioning problem. However, it is possible to add extra constraints and suppose that the racks do not have equal capacity, and that the VMs do not have neither equal size. In such case, the initialization of the OMA needs to start from a feasible solution, and each swapping of objects between two clusters needs to ensure that the constraints are not violated. In order to ensure starting the OMA algorithm from an initial feasible assignment, it is possible to deploy a "Bin Packing" algorithm to place the different VMs in different racks so as to satisfy the different constraints. Intuitively, the initial solution is feasible in terms of the constraints but it is not optimal in terms of traffic, thus requiring the invocation of the OMA. Thereafter, the OMA algorithm can proceed (not equi- partitioning, in this case) as normal but rather with an extra-check operation to ensure that swapping elements do not to violate these constraints. In the worst case, where no feasible solution is available, some over-provisioning can be allowed – which is a known paradigm in the cloud.

8 Conclusion and Future Work

The aim of this paper was to demonstrate how a Learning Automaton-based Graph Partitioning (GP) algorithm could be used to consolidate VMs in a traffic-aware manner, and to also show how a subsequent solution to a quadratic assignment algorithm could help in assigning the produced VM clusters to the server racks in order to reduce the total communication cost in a data center.

The first phase of the solution involved developing a VM clustering algorithm based on Oommen's Learning Automata based Graph Partitioning Algorithm (GPLA). The subsequent cluster placement algorithm involved the paradigm of simulated annealing. The two algorithms were utilized, one after the other, to partition 1,600 VMs into 16 clusters and to assign the clusters to 16 server racks. The strength of the algorithms was demonstrated by performing rigorous experiments on three different data center networking architecture models simulated using publicly available traffic traces from a live data center. The GPLA and SA 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 2,500 tests conducted revealed that the VM clustering algorithm decreased the total cost of communication from 34% to 85% depending on the characteristics of the original input data. Thereafter, the SA-based cluster placement algorithm further decreased the total cost of communication by a total improvement of more than 98%. The analysis showed that the VM clustering algorithm was fast, resource-effective and extremely capable of 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.

With regard to future work we mention that several features and functions can be developed to further improve and expand the capabilities of the VM clustering and cluster placement algorithms. These include:

- **Constraints**: One avenue for future work could focus more on expanding the VM clustering algorithm by developing custom constraints-handling for tenants on the cloud where VMs are not allowed to be freely moved from one server rack to another due to various reasons. This could occur, for example, in cases when VMs who communicate excessively with one another are the VMs who should not be hosted on same physical servers due to strict redundancy or security requirements. One could also consider architecture models it in which the link capacity is not the same for all the links.

- **Minimizing migrations**: One could also investigate how the VM clustering algorithm can be improved by requiring the minimal number of migrations is needed for the final optimization. Such a feature would be very important as migrating large numbers of VMs a resource-consuming task, and could be difficult to plan and execute in a large and complex cloud environments.

- **From static to dynamic optimization**: Rendering the current "offline" algorithms to be dynamic (i.e., to work in an on-line manner) would be a fascinating avenure for future research. A dynamic version would constantly monitor the changes in the traffic of the data center and generate suggestions, possibly incremental, for VM migrations after reliably detecting the VM clusters.

References

[1] M. Agache and B. Oommen. Generalized Pursuit Learning Schemes: New Families of Continuous and Discretized Learning Automata. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 32(6):738–749, 2002.

[2] Gordon C Armour and Elwood S Buffa. A heuristic algorithm and simulation approach to relative location of facilities. *Management Science*, 9(2):294–309, 1963.

[3] Rasmeet Singh Bali and Neeraj Kumar. Learning automata-assisted predictive clustering approach for vehicular cyber-physical system. *Computers & Electrical Engineering*, 52:82–97, 2016.

[4] Hitesh Ballani, Paolo Costa, Thomas Karagiannis, and Ant Rowstron. Towards predictable datacenter networks. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 242–253. ACM, 2011.

[5] 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*, pages 267–280. ACM, 2010.

[6] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Understanding data center traffic characteristics. *ACM SIGCOMM Computer Communication Review*, 40(1):92–99, 2010.

[7] Kashif Bilal, Saif Ur Rehman Malik, Osman Khalid, Abdul Hameed, Enrique Alvarez, Vidura Wijaysekara, Rizwana Irfan, Sarjan Shrestha, Debjyoti Dwivedy, Mazhar Ali, et al. A taxonomy and survey on green data center networks. *Future Generation Computer Systems*, 36:189–208, 2014.

[8] Aydin Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. Recent advances in graph partitioning. *Preprint*, 2013.

[9] Rainer E Burkard and Franz Rendl. A thermodynamically motivated simulation procedure for combinatorial optimization problems. *European Journal of Operational Research*, 17(2):169–174, 1984.

[10] 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*, pages 1–11. IEEE, 2009.

[11] 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`, 2012. [Online; accessed 15-February-2015].

[12] Vladimír Černỳ. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of optimization theory and applications*, 45(1):41–51, 1985.

[13] Cisco. Data Center Architecture Overview . `http://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Data_Center/DC_Infra2_5/DCInfra_1.html`. [Online; accessed 22-February-2015].

[14] 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`, 2009. [Online; accessed 18-March-2015].

[15] Chris HQ Ding, Xiaofeng He, Hongyuan Zha, Ming Gu, and Horst D Simon. A min-max cut algorithm for graph partitioning and data clustering. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 107–114. IEEE, 2001.

[16] RW Eglese. Simulated annealing: a tool for operational research. *European journal of operational research*, 46(3):271–281, 1990.

[17] Weiwei Fang, Xiangmin Liang, Shengxin Li, Luca Chiaraviglio, and Naixue Xiong. Vmplanner: Optimizing virtual machine placement and traffic flow routing to reduce network power costs in cloud data centers. *Computer Networks*, 57(1):179–196, 2013.

[18] W. Gale, S. Das, and C. Yu. Improvements to an Algorithm for Equipartitioning. *IEEE Trans. Comput.*, 39(5):706–710, 1990.

[19] 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*, pages 105–113. IEEE, 2011.

[20] Lukasz Golab, David DeHaan, Erik D Demaine, Alejandro Lopez-Ortiz, and J Ian Munro. Identifying frequent items in sliding windows over on-line packet streams. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 173–178. ACM, 2003.

[21] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: a scalable and flexible data center network. In *ACM SIGCOMM computer communication review*, volume 39, pages 51–62. ACM, 2009.

[22] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martín Casado, Nick McKeown, and Scott Shenker. Nox: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110, 2008.

[23] 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/`, 2009. [Online; accessed 22-February-2015].

[24] Mark Herbster and Manfred K Warmuth. Tracking the best expert. *Machine Learning*, 32(2):151–178, 1998.

[25] G. Horn and B. Oommen. A Fixed-Structure Learning Automaton Solution to the Stochastic Static Mapping Problem. In *19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*, pages 297–304, 2005.

[26] Liting Hu, Karsten Schwan, Ajay Gulati, Junjie Zhang, and Chengwei Wang. Net-cohort: Detecting and managing vm ensembles in virtualized data centers. In *Proceedings of the 9th international conference on Autonomic computing*, pages 3–12. ACM, 2012.

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

[28] Tabitha James, Cesar Rego, and Fred Glover. A cooperative parallel tabu search algorithm for the quadratic assignment problem. *European Journal of Operational Research*, 195(3):810–826, 2009.

[29] Lei Jiao, Xuan Zhang, B John Oommen, and Ole-Christoffer Granmo. Optimizing channel selection for cognitive radio networks using a distributed bayesian learning automata-based approach. *Applied Intelligence*, 44(2):307–321, 2016.

[30] David S Johnson, Cecilia R Aragon, Lyle A McGeoch, and Catherine Schevon. Optimization by simulated annealing: An experimental evaluation; part i, graph partitioning. *Operations research*, 37(6):865–892, 1989.

[31] 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*, page 6. ACM, 2013.

[32] 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*, pages 450–457. IEEE, 2010.

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

[34] Tjalling C Koopmans and Martin Beckmann. Assignment problems and the location of economic activities. *Econometrica: journal of the Econometric Society*, pages 53–76, 1957.

[35] Katrina LaCurts, Jeffrey C Mogul, Hari Balakrishnan, and Yoshio Turner. Cicada: Introducing predictive guarantees for cloud networks. In *Proceedings of the USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2014.

[36] Jeongkeun Lee, Yoshio Turner, Myungjin Lee, Lucian Popa, Sujata Banerjee, Joon-Myung Kang, and Puneet Sharma. Application-driven bandwidth guarantees in datacenters. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 467–478. ACM, 2014.

[37] Mahshid Mahdaviani, Javidan Kazemi Kordestani, Alireza Rezvanian, and Mohammad Reza Meybodi. Lade: Learning automata based differential evolution. *International Journal on Artificial Intelligence Tools*, 24(06):1550023, 2015.

[38] Sean Marston, Zhi Li, Subhajyoti Bandyopadhyay, Juheng Zhang, and Anand Ghalsasi. Cloud computing: The business perspective. *Decision Support Systems*, 51(1):176–189, 2011.

[39] 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*, pages 1–9. IEEE, 2010.

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

[41] Mehdi Rezapoor Mirsaleh and Mohammad Reza Meybodi. A new memetic algorithm based on cellular learning automata for solving the vertex coloring problem. *Memetic Computing*, 8(3):211–222, 2016.

[42] S. Misra and B. Oommen. Dynamic Algorithms for the Shortest Path Routing Problem: Learning Automata-Based Solutions. *IEEE Trans Syst Man Cybern B Cybern*, 35(6):1179–1192, 2005.

[43] Hossein Morshedlou and Mohammad Reza Meybodi. Decreasing impact of sla violations: a proactive resource allocation approachfor cloud computing environments. *IEEE Transactions on Cloud Computing*, 2(2):156–167, 2014.

[44] Milad Mozafari, Mohammad Ebrahim Shiri, and Hamid Beigy. A cooperative learning method based on cellular learning automata and its application in optimization problems. *Journal of Computational Science*, 11:279–288, 2015.

[45] K. Narendra and M. Thathachar. *Learning Automata: An Introduction*. New Jersey: Prentice-Hall, 1989.

[46] K. Narendra, E. Wright, and L. Mason. Application of Learning Automata to Telephone Traffic Routing and Control. *IEEE Trans Syst Man Cybern B Cybern*, SMC-7(11):785–792, 1977.

[47] Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat. Portland: a scalable fault-tolerant layer 2 data center network fabric. In *ACM SIGCOMM Computer Communication Review*, volume 39, pages 39–50. ACM, 2009.

[48] B. Oommen and S. Croix. String Taxonomy Using Learning Automata. *IEEETSMC: IEEE Transactions on Systems, Man, and Cybernetics*, 27(2):354–365, 1997.

[49] B. Oommen and C. Fothergill. Fast Learning Automaton-Based Image Examination and Retrieval. *The Computer Journal*, 36(6):542–553, 1993.

[50] B. Oommen and J. Lanctot. Discretized Pursuit Learning Automata. *Systems, Man and Cybernetics, IEEE Transactions on*, 20:931–938, 1990.

[51] B. Oommen and D. Ma. Fast Object Partitioning Using Stochastic Learning Automata. In *SIGIR '87: Proceedings of the 10th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 111–122, New York, NY, USA, 1987. ACM Press.

[52] B. Oommen and D. Ma. Deterministic Learning Automata Solutions to the Equipartitioning Problem. *IEEE Transaction Computer*, 37(1):2–13, 1988.

[53] B. Oommen and T. Roberts. Continuous Learning Automata Solutions to the Capacity Assignment Problem. *IEEE Transactions on Computers*, 49(6):608–620, 2000.

[54] B. John Oommen, De St Croix, et al. Graph partitioning using learning automata. *Computers, IEEE Transactions on*, 45(2):195–208, 1996.

[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*, pages 87–92. IEEE, 2010.

[56] Cisco Press. Cisco data center infrastructure 2.5 design guide. 2007.

[57] AS Ramkumar, SG Ponnambalam, and N Jawahar. A new iterated fast local search heuristic for solving qap formulation in facility layout design. *Robotics and Computer-Integrated Manufacturing*, 25(3):620–629, 2009.

[58] Alireza Rezvanian and Mohammad Reza Meybodi. Stochastic graph as a model for social networks. *Computers in Human Behavior*, 64:621–640, 2016.

[59] Ali Mohammad Saghiri and Mohammad Reza Meybodi. An approach for designing cognitive engines in cognitive peer-to-peer networks. *Journal of Network and Computer Applications*, 70:17–40, 2016.

[60] Jason Sonnek, James Greensky, Robert Reutiman, and Abhishek Chandra. Starling: Minimizing communication overhead in virtualized computing platforms using decentralized affinity-aware migration. In *Parallel Processing (ICPP), 2010 39th International Conference on*, pages 228–237. IEEE, 2010.

[61] S Mehdi Vahidipour, Mohammad Reza Meybodi, and Mehdi Esnaashari. Learning automata-based adaptive petri net and its application to priority assignment in queuing systems with unknown parameters. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(10):1373–1384, 2015.

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

[63] Y. Yang, J. Carbonell, R. Brown, T. Pierce, B. Archibald, and X. Liu. Learning approaches for detecting and tracking news events. *IEEE Intelligent Systems*, 14(4):32–43, 1999.