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

Akaki Jobava
Oslo and Akershus University College of Applied Sciences
Departement of Computer Science
Oslo, Norway

Anis Yazidi
Oslo and Akershus University College
Oslo, Norway
Departement of Computer Science
Email: anis.yazidi@hioa.no

B. John Oommen
Carleton University
School of Computer Science
Ottawa, Canada
Email: oommen@scs.carleton.ca

Kyrre Begnum
Oslo and Akershus University College
Oslo, Norway
Departement of Computer Science
Email: Kyrre.Begnum@hioa.no

***Abstract*—Cloud Computing (CC) is becoming increasingly pertinent and popular. A natural consequence of this is that many modern-day data centers experience very high *internal* traffic within the data centers themselves. 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 can, quite naturally, be compartmentalized into two phases which follow each other. 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% when tested on real-life data center traffic traces. 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. Indeed, as far as we know, this paper pioneers the application of LA in the traffic-aware consolidation of virtual machines in data centers, and also pioneers a strategy which serializes the tools in LA and SA to optimize CC.**

***Index Terms*—Cloud Computing, Virtual Machines, Graph Partitioning, Learning Automata, Quadratic Assignment, Simulated Annealing**

## I. 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 [8], [23], [28], make it an increasingly popular model 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. Consequently, CC is becoming one of the major driving forces behind the rapid growth of data centers around the world [13]. 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 [36] 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 [13], more greenhouse emissions, and the increased business costs. 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

Third author status: *Chancellor's Professor*, *Fellow: IEEE* and *Fellow: IAPR*. The author is also an Adjunct Professor with University of Agder, Grimstad, Norway.

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 the most optimal configuration. 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 is not as computationally hard as the previous phase 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).

## II. 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 [16] and PortLand [30]. However, some researchers have suggested a different, traffic-oriented VM consolidation approach to the problem. The material[1] in this section surveys the field.

### A. Network-aware Approaches

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

[1]The literature survey is quite detailed and comprehensive. It can be abridged if recommended by the Referees.

**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 [22]. 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 [14]) with its associated Software Defined Network (SDN) functionality support. It also uses Floodlight[2] 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 $\sim 60\%$ increase, while the core traffic yielded $\sim 70\%$ improvement.

**VM Placement and Migration in CC**: Piao and Yan [33] 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 [7] data center simulation environment and the results showed improved task completion time.

**Scalability of Data Center Networks Traffic-aware VMs**: Meng, Pappas and Zhang [29] 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.

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

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 [1] and SA [6]) 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 $\sim 10\%$ smaller than the measures obtained by the two benchmarks.

**Starling: Minimizing Communication Using Decentralized Affinity-Aware Migration**: Sonnek *et al.* [36] 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[3] and Cube MHD Jet (Cube)[4] 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.* [19] 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 [15], 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 factor of 12.76.

**Introducing Predictive Guarantees**: In their system Cicada, LaCurts *et al.* [26] 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 [18]. 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.* [2] 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.* [27] 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

[3]Please see https://software.intel.com/en-us/articles/intel-mpi-benchmarks.

[4]Please see http://www.astro.umn.edu/groups/compastro/?q=node/1.

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 [2] 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.* [13] 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 [17] 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 [10]. 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.

## III. Three-tier network architecture

A data center network is traditionally based on a *layered* [34] 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.

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.

## IV. 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

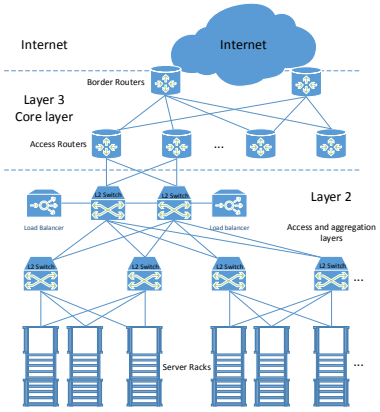


Fig. 1: The architecture of a traditional layered data center.

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 [4]. 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.

**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 [29]. The higher level network elements are usually enterprise-level devices and are often highly oversubscribed.

### A. 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 $j$.

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

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 $j$.

## V. 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) [32] 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) [5], [11], [21] 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). The best solution to the EPP is the so-called Object Migrating Automaton (OMA) proposed by Oommen and Ma [31]. 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. In order to determine whether the nodes are connected "significantly", we specify two important thresholds, *SimilarityThreshold* and *DissimilarityThreshold*, calculated by the following formulae:

$$SimilarityThreshold = (1 + \rho) MeanEdge$$
$$DisimilarityThreshold = (1 - \rho) MeanEdge$$

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

When two random vertices $V_i$ and $V_j$ are picked and their corresponding edge $D_{ij}$ is higher than the *SimilarityThreshold* the two nodes will be regarded as *similar*. If the nodes are found to be in 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.

### A. 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

## VI. 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) [20], [25], [29], [35], 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

**Data:** Node indices i and j, where $\omega_i$ and $\omega_j$ are the state indices of similar nodes in the same sub-partition.
**if** $\omega_i \mod M \neq 1$ **then** /* i is not in the most internal state */
  $\omega_i = \omega_i - 1$
**end**
**if** $\omega_j \mod M \neq 1$ **then** /* j is not in the most internal state */
  $\omega_j = \omega_j - 1$
**end**

**Procedure** The Pseudocode for the Function Reward-SimilarNodes(i,j)

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 "most 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 [9], [24]. 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 [12].

*1) 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. 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-aligned 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:

$$Comm_{Total} = \sum_{i,j=\cdots,n} D_{ij} \cdot C_{\pi(i)\pi(j)}, \qquad (2)$$

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.

### A. 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

**Data:** Node indices i and j where $\omega_i$ and $\omega_j$ are the state indices of dissimilar nodes in the same sub-partition
**if** $(((w_i \mod M) \neq 0) and ((w_i \mod M) \neq 0))$ **then**
  $\omega_i$ = $\omega_i$ + 1
  $\omega_j$ = $\omega_j$ + 1
**end**
**else**
  **if** $\omega_i \mod M \neq 0$ **then**
    $\omega_i$ = $\omega_i$ + 1
    TempState1 = EvaluateCost of current partitioning
    Prev_Cost = EvaluateCost of current partitioning
    **for** *all remaining* $K-1$ *partitions* **do**
      $\omega_p$ = state of node closest to boundary in this current sub-partition
      TempState2 = $\omega_p$
      $\omega_j = (\omega_p div M + 1) \cdot M$
      $\omega_p = TempState1$
      New_Cost = EvaluateCost of current partitioning
      **if** $New\_Cost > Prev\_Cost$ **then**
        $\omega_p = TempState2$
        $\omega_j = TempState1$
      **end**
      **else**
        Prev_Cost = New_Cost
      **end**
    **end**
    **else**
      $\omega_j = \omega_j + 1$
      TempState1 = $\omega_i$
      Prev_Cost = EvaluateCost of current partitioning
      **for** *all remaining* $K-1$ *partitions* **do**
        $\omega_p$ = state of node closest to boundary in this current sub-partition, $\alpha_Z$
        TempState2 = $\omega_p$
        $\omega_i = (\omega_p div M + 1) \cdot M$
        $\omega_p$ = TempState1
        New_Cost = EvaluateCost of current partitioning
        **if** $New\_Cost > Prev\_Cost$ **then**
          $\omega_p = TempState2$
          $\omega_i = TempState1$
        **end**
        **else**
          Prev_Cost = New_Cost
        **end**
      **end**
    **end**
  **end**
**end**

**Procedure** The Pseudocode for the Function Penalize-DissimilarNodes(i,j)

**Data:** Node indices i and j, where $\omega_i$ and $\omega_j$ are the state indices of similar nodes in the different sub-partitions.
**if** $(((w_i \mod M) \neq 0) and ((w_i \mod M) \neq 0))$ **then**
  $\omega_i = \omega_i + 1$ /* both are in internal states */
  $\omega_j = \omega_j + 1$
  **else**
  **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 $v_j$ */
    $\omega_j = (\omega_i divM) \cdot M$ /* move $v_j$ to $v_i$'s sub-partition */
    t := index of a node in $v_i$'s sub-partition with $v_t \neq v_i$ and $v_t$ closest to the boundary state of $\omega_i$
    $\omega_t$ = temp /* move $v_t$ to the old state of $v_j$ */
    **else**
    **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 $v_i$ */
      $\omega_i = (\omega_j divM) \cdot M$ /* move $v_i$ to $v_j$'s sub-partition */
      t := index of a node in $v_j$'s sub-partition with $v_t \neq v_j$ and $v_t$ closest to the boundary state of $\omega_j$
      $\omega_t$ = temp /* move $v_t$ to the old state of $v_i$ */
    **end**
  **end**
**end**

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

**Input:** The set $V = \{v_1, v_2, ..., v_{KN}\}$ to be partitioned into K sub-partitions.
D is adjacency traffic matrix and $V_1$, $V_2$ ... $V_K$ are current feasible sub-partitions.
$\rho$ is a parameter used to determine the similarity or dissimilarity of the vertices.
M=100.
**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** $C_{ij} > (1 + \rho) \cdot Mean\_Edge$ **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** $C_{ij} < (1 - \rho) \cdot Mean\_Edge$ **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**

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:

$$P = e^{-\frac{\Delta}{T}}, \tag{3}$$

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 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.**

## VII. 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. They were conducted with two different sets of 1,600 VMs selected from the obtained traffic traces.

The communication data used in this work is obtained via third party source which made the data available for the public use. Three actual data center traffic traces are published on the Computer Sciences User Pages of the University of Wisconsin-Madison[5]. The data sets are dated from 2009 and represent three different university data centers studied in the paper titled Network Traffic Characteristics of Data Centers in the Wild [3].

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

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

We also understood the importance of having a plan by which one could perform the measurement and evaluation of the experimental results. 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. 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 2 illustrates the total cost of communication with randomly assigned VMs, after the

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

VM clustering phase and after cluster placement in all three data center network architecture models.

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

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

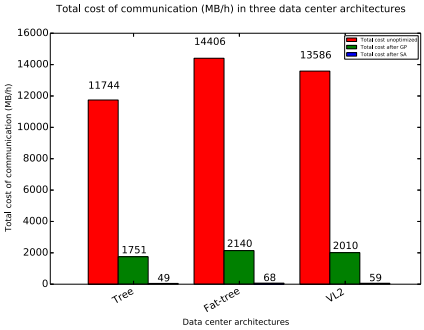


Fig. 2: Total cost of communication in all three experiments in Set A.

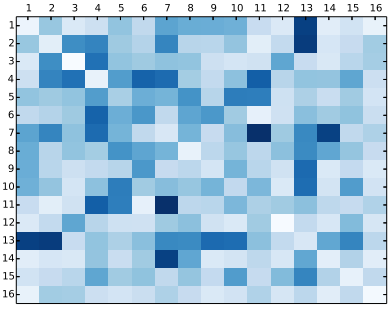

Fig. 3: Intra and intercluster traffic heatmap *before* executing the GP in Set A.

## VIII. Conclusion

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.

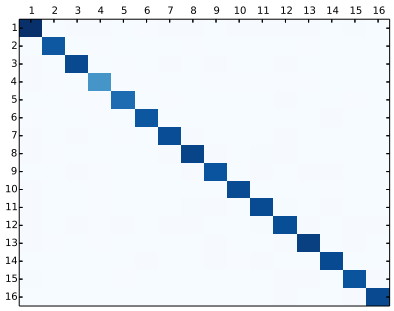

Fig. 4: Intra and intercluster traffic heatmap *after* executing the GP in Set A.

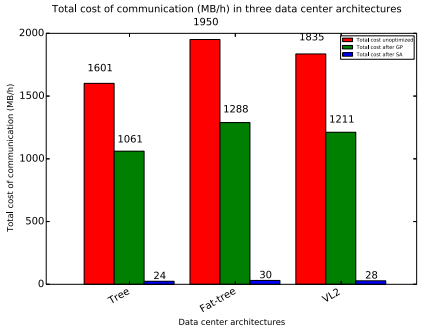


Fig. 5: Total cost of communication in all three experiments in in the case of the data in Set B.

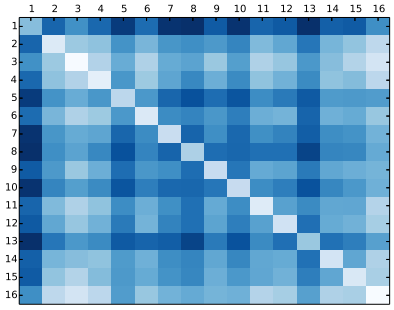

Fig. 6: Intra and intercluster traffic heatmap *before* executing the GP in Set B.

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.

## References


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

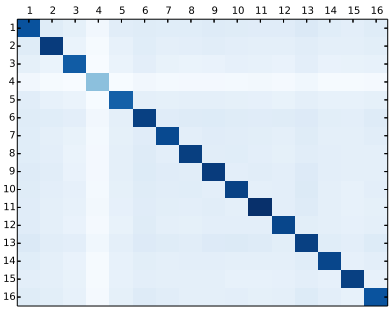


Fig. 7: Intra and intercluster traffic heatmap *after* executing the GP in Set B.


[2] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. Towards predictable datacenter networks. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 242–253. ACM, 2011.

[3] T. Benson, A. Akella, and D. 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.

[4] K. Bilal, S. U. R. Malik, O. Khalid, A. Hameed, E. Alvarez, V. Wijaysekara, R. Irfan, S. Shrestha, D. Dwivedy, M. Ali, et al. A taxonomy and survey on green data center networks. *Future Generation Computer Systems*, 36:189–208, 2014.

[5] A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz. Recent advances in graph partitioning. *Preprint*, 2013.

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

[7] R. Buyya, R. Ranjan, and R. 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.

[8] I. I. 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].

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

[10] I. 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].

[11] C. H. Ding, X. He, H. Zha, M. Gu, and H. D. Simon. A min-max cut algorithm for graph partitioning and data clustering. In *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, pages 107–114. IEEE, 2001.

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

[13] W. Fang, X. Liang, S. Li, L. Chiaraviglio, and N. 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.

[14] S. K. Garg and R. 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.

[15] L. Golab, D. DeHaan, E. D. Demaine, A. Lopez-Ortiz, and J. I. Munro. Identifying frequent items in sliding windows over online packet streams. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 173–178. ACM, 2003.

[16] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. Vl2: a scalable and flexible data center network. In *ACM SIGCOMM computer communication review*, volume 39, pages 51–62. ACM, 2009.

[17] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. Nox: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110, 2008.

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

[19] L. Hu, K. Schwan, A. Gulati, J. Zhang, and C. 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.

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

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

[22] D. Kakadia, N. Kopri, and V. 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.

[23] A. Khajeh-Hosseini, D. Greenwood, and I. Sommerville. Cloud migration: A case study of migrating an enterprise it system to iaas. In *IEEE International Conference on Cloud Computing (CLOUD)*, pages 450–457. IEEE, 2010.

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

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

[26] K. LaCurts, J. C. Mogul, H. Balakrishnan, and Y. Turner. Cicada: Introducing predictive guarantees for cloud networks. In *Proceedings of the USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2014.

[27] J. Lee, Y. Turner, M. Lee, L. Popa, S. Banerjee, J.-M. Kang, and P. Sharma. Application-driven bandwidth guarantees in datacenters. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 467–478. ACM, 2014.

[28] S. Marston, Z. Li, S. Bandyopadhyay, J. Zhang, and A. Ghalsasi. Cloud computing: The business perspective. *Decision Support Systems*, 51(1):176–189, 2011.

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

[30] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. 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.

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

[32] B. J. Oommen, D. S. Croix, et al. Graph partitioning using learning automata. *IEEE Transactions on Computers*, 45(2):195–208, 1996.

[33] J. T. Piao and J. 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.

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

[35] A. Ramkumar, S. 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.

[36] J. Sonnek, J. Greensky, R. Reutiman, and A. Chandra. Starling: Minimizing communication overhead in virtualized computing platforms using decentralized affinity-aware migration. In *International Conference on Parallel Processing (ICPP)*, pages 228–237. IEEE, 2010.