Elastic, on-line and network aware virtual machine placement within a data center

Federico Larumbe
Ecole Polytechnique de Montréal
federico.larumbe@polymtl.ca

Brunilde Sansò
Ecole Polytechnique de Montréal
brunilde.sanso@polymtl.ca

***Abstract*—This article presents a new model and a resolution algorithm, based on Tabu Search, for the assignment of Virtual Machines (VMs) to servers in a data center. We propose a Mixed Integer Programming (MIP) model that optimizes the Quality of Service (QoS) and power consumption of applications, taking into account their communication traffic and dynamic aspects. A hierarchic method and a Tabu Search heuristic that considers the network topology are developed to solve cases with realistic sizes—e.g., a data center with 1600 servers per pod, for up to 128,000 total servers—. The method specifically considers the optimal mapping of the application graph into the data center network. The proposed scheduler is compared with 1) a static method that does not consider workload variations, and 2) the first-fit policy as a sample of methods that do not consider communication traffic among VMs.**

***Index Terms*—Data center management, cloud computing, scheduling, Tabu Search, virtualization, virtual machine placement, service function chaining, virtual network embedding.**

I. Introduction

With the popularity of cloud computing services, a large amount of data is being managed in large data centers making use of virtualized resources. Virtualization adds flexibility to data center management but that flexibility comes at the expense of combinatorial complexity when allocating the virtual and physical resources to the different applications. The allocation problem consisting in assigning physical resources (CPU, storage, memory) to virtual machines (VMs), has been extensively studied in the literature under the name of *VM placement problem* (VMPP) (see surveys [1], [2], [3], [4]). Most of the work on the VMPP has exploited its bin-packing nature [5], where the objective is to reduce the energy consumption by packing as many VMs as possible in as few servers as possible. However, as demand for applications such as social-networking continues to grow, there has been a shifting of the demand from being "north-south" (inside-outside the data center), to "east-west" (intra-data center). This shifting has the potential of creating important congestion bottlenecks inside a data center. As a result, the *network awareness* of the VM placement is gaining importance [6], [7], [8], [9], [10], according to which the VMs that "talk to each other" should be placed as closed as possible to reduce their usage of network resources. The strategy, however, can create a tight packing of VMs in the same racks, which can induce scalability and elasticity problems when the demand increases. The VM placement can also be classified as "on-line" or "off-line" depending on wether the placement decision is taken or not in an on-line, dynamic fashion [11], [12]. When the decision is on-line, the placement strategies are usually quite simple (such as first-fit), given the need to keep the resolution time of the scheduling algorithm as short as possible. This paper presents a modelling framework and a very efficient resolution approach that deals with the above mentioned issues. The method is quick and efficient, so that it can be integrated to on-line management engines. It can also be used off-line for longer term optimization placement and diagnostic. The article presents the following original contributions:

1) A detailed Mixed Integer Programming (MIP) model to assign VMs to servers, minimize power consumption, and communication delay among VMs considering: a)multiple server models with different capacities and energy efficiency, b)multiple VM types that require different amount of server resources, c)network topology, delay, and link utilization, and d)the elasticity of applications as the workload varies over time.
2) A very efficient Tabu Search heuristic that quickly solves the proposed MIP model for networks of hundreds of thousands of servers and Virtual Machines (VMs), adds a new application to the network and resizes existing applications each hour of the day.

The rest of this article is structured as follows. Section II presents a short overview of the literature, including important surveys. Section III explains the proposed strategy to assign VMs to servers. Section IV describes the MIP model. In Section V a Tabu Search based heuristic to solve the VM placement is presented. Section VI describes a case study of a data center with 128,000 servers including network topology, application topology, workload, and power consumption parameters. Section VII presents the results obtained by comparing the proposed scheduler with static VM placement and dynamic VM placement with first-fit policy. Finally, the article is concluded in Section IX.

II. Literature review

The VMPP is well-known to be NP-hard as it can be reduced to the multi-dimensional bin-packing [13]. [2] surveys energy aware problems inside the datacenter, including VM placement. [3] focuses on a comprehensive overview of resource management issues in clouds, among them, the scheduling of the VMs. [4] deals with the VM placement

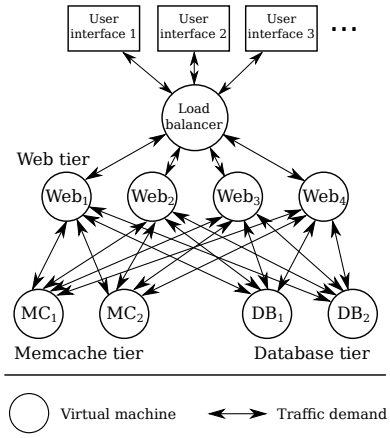

Fig. 1. Graph of a multi-tier application.

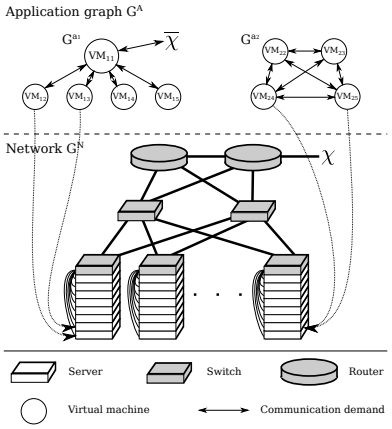

Fig. 2. Application and network mapping.

problem optimization approaches, classifying the literature based on the type objective functions, the type of solution approaches, the cloud architecture and the type of experiments being carried out. [14] proposes a detailed management of the capacity while dealing with scalability of an SDN (Software Defined Networking) solution. [15] treats the temporal and spatial aspects of the problem by introducing the notion of complimentary VMs which are VMs with different temporal resource patterns that can be combined to optimize the local resources. [16] tackles the problem of assigning tasks to servers by adapting a multi-dimensional bin packing where scalability and network awareness has been incorporated. [17], [18], [19] have taken into account the traffic among VMs at the time of placing VMs. They differ on the strategy to handle a dynamic workload and on the solution methods. [20] presents a method that provides bandwidth guarantees by considering a two-layered problem.The multilayered approach, that we introduced in our previous work on data center location [21], is similar to the one presented here, however, the optimization features and the way to handle the bandwidth guarantees are different. Moreover, they require a specific tree-like topology whereas our method is topology independent. Finally, we mention the latest literature on Service Function Chaining (SFC) and Virtual Network Embedding (VNE) [22], [23], [24] that, even though differ from the VMPP, present a very similar structure to our problem and could benefit from our mapping approach.

III. Proposed Placement Strategy

For each application request, the decision to make is which server will host the requested VM for that application. After the decision is made, the hypervisors of the target servers create the VMs in real time. The objective is to minimize delay among VMs, maximize the network throughput available among VMs, and minimize the server energy consumption. Our placement strategy can be characterized as: a) based on application graph, b) communication-aware, c) energy-efficient and d) elastic. The requirements of an application are modeled as an application graph where each node is a VM and each arc indicates a VM sending traffic to another one. The application developer defines the type of VMs needed and the expected traffic between each pair of VMs. The scheduler takes that information as an input to place the VMs. Figure 1 shows an example of a multi-tier application graph, a popular architecture for Internet applications [25]. Note that our method is in no way restricted to the example, as it can be applied to any application graph made up of communicating software components. Thus, since the application's VMs communicate among themselves, the application performance depends on which server of the network will host each VM.The proposed model is network-aware because the model includes the delay among VMs in the objective function so that VMs that communicate among themselves will be placed close to each other. This reduces the load on the links between switches, thus improving quality of service and avoiding bottlenecks to increase the throughput available among VMs. The proposed approach is also energy-efficient because an active server with enough capacity will be preferred rather than using an inactive server. Furthermore, between two servers of different models or manufacturers, the one that consumes less power will be used. The application elasticity is also considered for the placement policy. Depending on the hour and the day, applications receive different workload sizes. Adding and removing VMs to existing applications can handle a varying workload. The proposed approach keeps a fraction of each rack empty to be used by VMs added in peak periods, in this way these applications benefit of a reduced delay and high throughput. VM migration is not considered in the proposed approach because even if live migration can be done in a short time interval [26], that period could negatively impact the programs and network protocols executing on the VM. However, if live migration is requested, additional constraints could be easily incorporated into the framework described below.

IV. Problem description

A cloud data center network is represented as a graph $G^N(V^N, A^N)$, as shown in Figure 2. The set of nodes V^N

contains the servers $\mathcal{S}$, switches $\mathcal{Z}$, and a special node χ that represents destinations outside the data center network. The data center contains a set of racks $\mathcal{R} = \{1, \ldots, |\mathcal{R}|\}$, and each rack $r \in \mathcal{R}$ accommodates a set of servers $\mathcal{S}_r \subseteq S$. The servers of a rack are connected to a Top of Rack (ToR) switch, and all the switches are connected through a particular topology. Each link direction is an arc in A^N.

A program called *scheduler* receives requests to add, remove, and resize applications. Let a_n represent a particular application and $G^{a_n}(V^{a_n}, A^{a_n})$ the topology of that application. As can be seen in Figure 2, each node in V^{a_n} is a VM, and each arc $(i, j) \in A^{a_n}$ represents that VM i sends information to VM j. Similarly to the network layer, there is a special node $\overline{\chi}$ in V^{a_n} that represents programs outside the data center that exchange information with the VMs.

$G^A_t(V^A_t, A^A_t)$ is the graph that contains all the applications being executed at time t. Therefore, the approach is valid not only for a single application request, but for several requests at the same time, as they can be considered by this graph. The nodes in V^A_t are the VMs of all applications, and the arcs in A^A_t are the communication demands. If the system receives a request to add a new application a_n at time $t + 1$ then:

$$G^A_{t+1} = (V^A_t \cup V^{a_n}, A^A_t \cup A^{a_n})$$

When a request to remove an application a_n is received at time $t + 1$ then:

$$G^A_{t+1} = (V^A_t - V^{a_n}, A^A_t - A^{a_n})$$

M_t: $V^A_t \rightarrow \mathcal{S}$ is a mapping that specifies which server is assigned to each VM at time t. Then the scheduler has to define the mapping M_{t+1} starting from M_t.

The VMs of an application can communicate among themselves and with Internet hosts to achieve the application's purpose.The parameters related to the communication traffic are the following:

- b_d Average throughput of demand $d \in A^A_t$ (in bps).
- $\overline{c_e}$ Capacity of link $e \in A^N$ (in bps).
- ζ Average packet size (in bits).

Data centers typically have different server models. Each server model has capacities associated to each server resource:

- $\mathcal{M}$ Set of server models, e.g., 1 = Dell PowerEdge R420, 2 = HP ProLiant Dl360, 3 = IBM System x3750.
- μ_k Model of server $k \in \mathcal{S}$.
- $\mathcal{K}$ Set of resources that each server has, which include $1 =$ CPU, $2 =$ network card, $3 =$ RAM, and $4 =$ storage.
- c_{pm} Capacity of resource $p \in \mathcal{K}$ on a server of model $m \in \mathcal{M}$ in the corresponding units (GHz, Gbps, GB).

Different VM types require more or less of each server resource. Each application defines the type of VMs needed depending on the VM purpose. The following are the parameters that define the VMs:

- $\mathcal{T}$ Set of VM types, e.g., 1 = small instance, 2 = medium instance, 3 = large instance, 4 = high RAM instance, 5 = high CPU instance.
- ψ_i Type of VM $i \in V^A_t$.
- r_{pv} Requirement of resource $p \in \mathcal{K}$ for a VM of type $v \in \mathcal{T}$ in the corresponding units (GHz, Gbps, GB).
- $\mathcal{P}$ Set of possible assignments (v, m) indicating that a VM of type $v \in \mathcal{T}$ can be hosted on a server of model $m \in \mathcal{M}$.

The power consumption of a server is related to its model and its resource utilization. We assume that a server that does not host any VM can be put in suspended-to-ram state, and thus consume a very low amount of power. Because VMs can be added and removed dynamically, we add a parameter to avoid activating and deactivating a server too often. If a server is turned on to host a VM, and then the VM is removed in a short interval of time, the server is kept on for the next VM to be added. Without that condition, the second VM could arrive while the server was entering in suspension state and would wait longer to start. The following parameters define the power consumption aspect:

- $w_m : \mathbb{R}^{|\mathcal{K}|} \rightarrow \mathbb{R}$ Power consumed by a server of model $m \in \mathcal{M}$ as a function of CPU, network bandwidth, RAM, and disk allocated.
- π_k Time of activation of server $k \in \mathcal{S}$ (in minutes); this value is -1 if the server was never activated.
- h Minimal time interval that a server must be kept in active state (in minutes).

In order to group VMs of the same application together, we keep a fraction of each rack available for future VMs. That is defined by an expected utilization threshold on the rack resources (sum of the server resources). When the resources of a rack are higher than the threshold, a penalty term is added to the objective function.

- u_{rpt} Utilization threshold of resource $p \in \mathcal{K}$ in rack $r \in \mathcal{R}$ $(0 \leq u_{rpt} \leq 1)$.

The variables for the problem are:

- x_{ikt} 1 if VM $i \in V^A_t$ is located in server $k \in \mathcal{S}$; 0 otherwise.
- y_{kt} 1 if server $k \in \mathcal{S}$ is in active state; 0 if the server is in suspend-to-ram state.
- f_{det} Amount of traffic demand $d \in A^A_t$ carried by link $e \in A^N$ (in bps). This variable is in charge of the optimal mapping and physical link allocation between the application and the physical network layers.
- q_{kpt} Consumption of resource $p \in \mathcal{K}$ in server $k \in \mathcal{S}$ in the corresponding units.
- l_{rpt} Consumption of resource $p \in \mathcal{K}$ in rack $r \in \mathcal{R}$ $(0 \leq l_{rpt} \leq 1)$.
- o_{rpt} Overflow of resource $p \in \mathcal{K}$ in rack $r \in \mathcal{R}$ $(0 \leq o_{rpt} \leq 1)$.

The first part of the multi criteria objective considers the communication delay among VMs, that is proportional to the packet size ζ and inversely proportional to the link capacity $\overline{c_e}$. Each link delay is weighted by the amount of traffic it carries. The advantage of this formula when compared with

the simple hop count is that it differentiates links with low and high capacity—e.g., 1 Gbps and 10 Gbps links.

$$C^D = \frac{\sum_{e \in A^N} (\zeta / \overline{c_e}) \sum_{d \in A_t^A} f_{det}}{\sum_{d \in A_t^A} b_d}$$

The second part of the objective to minimize is the server power consumption, that is calculated as the sum of each server power consumption:

$$C^E = \sum_{k \in \mathcal{S}} w_{\mu_k} (q_{k,1}, q_{k,2}, q_{k,3}, q_{k,4})$$

The third term in the objective is an elasticity penalty that aims at leaving unused servers for new VMs created at the peak times. It is defined as the sum of the overflow of each rack, which starts to be positive when the allocated resources of a rack are above a threshold.

$$C^L = \sum_{r \in \mathcal{R}} \sum_{p \in \mathcal{K}} o_{rpt}$$

The objective function to minimize is the weighted sum:

$$z = \alpha C^D + \beta C^E + \gamma C^L \tag{1}$$

The problem constraints are given by:

- *VM placement*

$$\sum_{\substack{k \in \mathcal{S} / \\ (\psi_i, \mu_k) \in \mathcal{P}}} x_{ikt} = 1 \qquad \forall \quad i \in V^{a_n} \tag{2}$$

$$\sum_{\substack{k \in \mathcal{S} / \\ (\psi_i, \mu_k) \notin \mathcal{P}}} x_{ikt} = 0 \qquad \forall \quad i \in V^{a_n} \tag{3}$$

$$x_{ikt} = 1 \qquad \forall \quad (i,k) \in M_{t-1} \tag{4}$$

$$x_{ikt} = 0 \qquad \forall \quad i \in V_{t-1}^A, k \in \mathcal{S}, (i,k) \notin M_{t-1} \tag{5}$$

Equations (2) and (3) define that each VM must be placed in one server with a model that is feasible for that type of VM. Equations (4) and (5) state that the VMs previously placed remain in the same servers.

- *Active servers*

$$y_{kt} \geq x_{ikt} \qquad \forall \quad i \in V_t^A, k \in \mathcal{S} \tag{6}$$

$$y_{kt} = 1 \qquad \forall \quad k \in \mathcal{S} / t \in [\pi_k, \pi_k + h] \tag{7}$$

Constraint (6) states that the servers hosting VMs must be in active state. Equation (7) says that servers must remain active at least h minutes.

- *Flow conservation*

$$\sum_{e \in \Gamma_k^-} f_{det} + x_{ikt} b_d = \sum_{e \in \Gamma_k^+} f_{det} + x_{jk} b_d \tag{8}$$
$$\forall \quad d = (i,j) \in A_t^A, \quad k \in \mathcal{S}$$

$$\sum_{e \in \Gamma_k^-} f_{det} = \sum_{e \in \Gamma_k^+} f_{det} \tag{9}$$
$$\forall \quad d = (i,j) \in A_t^A, \quad k \in \mathcal{Z}$$

$$\sum_{e \in \Gamma_\chi^-} f_{det} = b_d \qquad \forall \quad d = (\overline{\chi}, j) \in A_t^A \tag{10}$$

$$\sum_{e \in \Gamma_\chi^+} f_{det} = b_d \qquad \forall \quad d = (i, \overline{\chi}) \in A_t^A \tag{11}$$

Equation (8) relates VM placement variables and routing of traffic demands. It associates the application and the network layers. For a demand from a VM $i \in V_t^A$ placed in server $k_1 \in \mathcal{S}$ to a VM $j \in V_t^A$ placed in server $k_2 \in \mathcal{S}$, the flow of traffic b_d starts in k_1 and ends in k_2. Equation (9) states that in each switch k of the path from k_1 to k_2 the flow is conserved, that is the incoming flow of k is equal to its outgoing flow. Equation (10) states that the traffic coming from outside the data center enters by node χ. Equation (11) states that the traffic going outside the data center exits by node χ. In those data centers where routing is restricted, additional constraints should be added.

- *Link capacity*

$$\sum_{d \in A_t^A} f_{det} \leq \overline{c_e} \qquad \forall \quad e \in A^N \tag{12}$$

In Constraint (12), the flow of each link is lower than its capacity.

- *Server resource utilization*

$$q_{kpt} = \sum_{i \in V^A} r_{p\psi_i} x_{ikt} \qquad \forall \quad k \in \mathcal{S}, p \in \mathcal{K} \tag{13}$$

$$q_{kpt} \leq c_{p\mu_k} \qquad \forall \quad k \in \mathcal{S}, p \in \mathcal{K} \tag{14}$$

Equation (13) calculates the amount of resource p reserved in server k. Constraint (14) requires the capacity of each resource to be respected.

- *Rack resource utilization*

$$l_{rpt} = \frac{\sum_{k \in \mathcal{S}_r} q_{kpt}}{\sum_{k \in \mathcal{S}_r} c_{p\mu_k}} \qquad \forall \quad r \in \mathcal{R}, p \in \mathcal{K} \tag{15}$$

$$o_{rpt} \geq \frac{l_{rpt} - u_{rpt}}{1 - u_{rpt}} \qquad \forall \quad r \in \mathcal{R}, p \in \mathcal{K} \tag{16}$$

$$o_{rpt} \geq 0 \qquad \forall \quad r \in \mathcal{R}, p \in \mathcal{K} \tag{17}$$

Equations (15) aggregate the server resource consumption for a whole rack and normalize it between 0 and 1. Constraints (16) and (17) define the overflow variable for each rack and resource. That variable starts to be positive when the rack load l_{rpt} is above the utilization threshold u_{rpt} and reaches the value 1 when the rack load is 1.

- *Domain of variables*

$$x_{ikt} \in \{0,1\} \qquad \forall \quad i \in V_t^A, k \in \mathcal{S} \tag{18}$$

$$y_{kt} \in \{0,1\} \qquad \forall \quad k \in \mathcal{S} \tag{19}$$

$$f_{det} \in \mathbb{R}_{\geq 0} \qquad \forall \quad d \in A_t^A, e \in A^N \tag{20}$$

$$q_{kpt} \in \mathbb{R}_{\geq 0} \qquad \forall \quad k \in \mathcal{S}, p \in \mathcal{K} \tag{21}$$

$$l_{rpt} \in \mathbb{R}_{\geq 0} \qquad \forall \quad r \in \mathcal{R}, p \in \mathcal{K} \tag{22}$$

$$o_{rpt} \in \mathbb{R}_{\geq 0} \qquad \forall \quad r \in \mathcal{R}, p \in \mathcal{K} \tag{23}$$

V. Solution approach

Given that a VM scheduler in a cloud data center could need to scale to more than 100,000 servers and VMs, we developed a very efficient hierarchical heuristic based on Tabu Search. We name the proposed method Cloptimus Scheduler (CS). We exploit the fact that data center topologies are split in clusters and sub clusters (pods) and the symmetry of the topology.

Algorithm 1 Add, remove, or resize an application.

function CLOPTIMUSSCHEDULER(*Request*, M)
  $a_n \leftarrow$ *Request* application
  if *Request* type is add **then**
    $M \leftarrow$ INITIALGREEDYSOLUTION(a_n, M)
    $M \leftarrow$ TABUSEARCH(a_n, M)
  end if
  if *Request* type is remove **then**
    Remove a_n's VMs from M
  end if
  if *Request* type is resize **then**
    Remove a_n's VMs from solution M
    Add or remove VMs from a_n
    Place existing VMs in the same servers than before in M
    if a_n has new VMs to schedule **then**
      $M \leftarrow$ INITIALGREEDYSOLUTION(a_n, M)
      $M \leftarrow$ TABUSEARCH(a_n, M)
    end if
  end if
end function

function INITIALGREEDYSOLUTION(a_n, S)
  for each vm in $vmsToSchedule(a_n)$ **do**
    Move vm to the server that least increases $z(S)$
  end for
  return S
end function

function TABUSEARCH(a_n, *Current*)
  $S \leftarrow$ *Current*
  Best $\leftarrow$ *Current*
  $L \leftarrow \{\}$
  $i \leftarrow 0$
  repeat
    Choose the pair $(c, s) \in \mathcal{N}(S) - L$ that minimizes z when moving vm to s in S.
    Move c to s in S
    Add (c, s) to tabu list L for q iterations.
    $i \leftarrow i + 1$
    if $z(S) <$ *z*(*Best*) **then**
      Best $\leftarrow S$
      $i \leftarrow 0$
    end if
  until $i =$ *MAX_ITERATIONS*
  return *Best*
end function

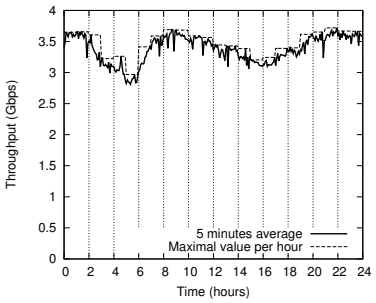

Fig. 3. San Jose CAIDA's traffic monitor during August 3, 2013 [27].

In this problem, a solution S is a mapping between VMs and servers. The resolution approach, portrayed in Algorithm1, is composed of three main functions: the initial greedy solution, the Tabu Search and the Scheduler that receives requests to add, remove or resize an application. When the request is to add an application a_n, then the complete graph of the application is passed to the initial greedy solution and next to the Tabu Search. If the request is to remove the application, then all the VMs belonging to the application are removed. If the request is to resize the application, then either some VMs will be removed or will be added, with the consequent call to the greedy and then the Tabu Search. The initial greedy solution places the VMs in the server that least increases the objective function. With respect to the Tabu, the neighborhood $\mathcal{N}(S)$ is the set of solutions that differ in the placement of one of the VMs that is being scheduled. To take advantage of the symmetry between solutions, each possible movement is to move each VM to the first available server in each rack. Thus, the number of possible movements is reduced to the number of VMs to schedule multiplied by the number of racks in a pod. The reduction of movements is valid because the servers in the same rack are equal in terms of delay. Furthermore, choosing the first one available optimizes the second objective that is the power consumption because a server will be activated only if the previous ones were full. This assumes that servers are sorted by energy efficiency in each rack. Thus, the first available server will also be the most efficient one among all.

The number of iterations q to keep a movement in the tabu list is the square root of the neighborhood size. In this case, a second restriction was added to the tabu movements. Each VM that is moved to a server is then kept in that server for a number of iterations, that is half of the number of VMs to schedule. Keeping a VM fixed in a new rack for a number of iterations allows the other VMs to be moved to that rack, and that solution with the whole application in a different rack can be evaluated by the algorithm. When the solution does not improve for *MAX_ITERATIONS* the algorithm stops. That value was set to the number of VMs to schedule multiplied by the number of racks.

VI. Case study

The datacenter features and network topology chosen for the tests are portrayed in Figure 4

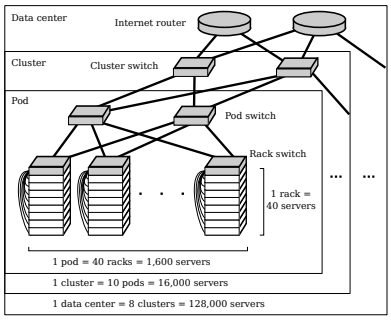

Fig. 4. Data center network topology.

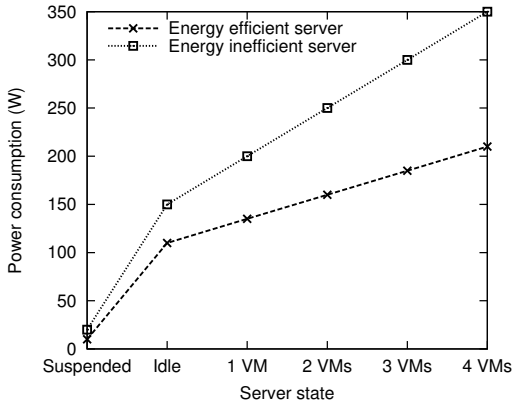

Fig. 5. Server power consumption.

The proposed scheduler is executed as multiple parallel processes, each one handling a pod of 1,600 servers. The experiments shown in this article use one scheduler managing the first pod of the topology. The delay of each link was set as the time to transmit one packet of 1,500 bytes at the line capacity.

Each application to be scheduled is a multi-tier application with a different number of VMs. Each application is composed of 1 load balancer, 1 memcache, 2 databases, and a number of web servers that changes depending on the workload size. That is the same architecture as that of Figure 1. Each web server VM uploads up to 100 Mbps of traffic to its load balancer. The load balancer forwards the traffic of its web VMs to the Internet. 50% of the traffic uploaded by the web server is retrieved from the database, and the other 50% is retrieved from memcache.

The number of web VMs of each application was drawn from a Pareto distribution with parameters $\alpha = 1.6$ and $\beta = 1$. Generating 800 applications gave between 1 and 68 web VMs per application.

The workload variation is based on the load of a whole day in an Internet link of a data center in San Jose, California, publicly available by the CAIDA initiative [27]. As shown in Figure 3, that link has periods of high utilization (3.72 Gbps) and low utilization (3.22 Gbps). The relative variation in each period was used to multiply the number of web VMs of each application each hour of the day.

Each server has 8 cores at a frequency of 2 Ghz and 64 GB of RAM. Each core is shared by two virtual CPUs. Each VM requires 4 virtual CPUs of 1 GHz and 16 GB RAM, the same requirements as the extra large instances of Amazon EC2 [28]. With these settings, each server can host up to 4 VMs.

We considered two server models with the same specifications but different power consumption. One server model is energy-efficient and consumes 10 W in suspended state, 110 W in idle state, and an average of 210 W when it hosts 4 VMs. The other server model is less efficient and consumes 20 W in suspended state, 150 W in idle state, and an average of 350 W when it hosts 4 VMs [29], [30], [31]. That average power consumption can be seen in Figure 5 as a function of the number of VMs in a server.

TABLE I
CLOPTIMUS SCHEDULER VS FIRST-FIT AFTER ADDING 800 APPLICATIONS.

Solution:	CS	FF
Avg. delay	11.3 µs	19.2 µs
Avg. link utilization	13.1%	23.0%
Avg. inter-switch link util.	10.5%	11.9%
Max. inter-switch link util.	32.2%	36.7%
Number of VMs	5,027	5,027
Number of servers	1,258	1,257
Total power	267.5 kW	267.4 kW

Each rack has either all energy-efficient servers, or all energy-inefficient servers. One of the cases tested had all racks with energy-efficient servers. Another case had energy-inefficient servers in racks numbered with odd numbers, and energy-efficient servers in racks numbered with even numbers.

VII. RESULTS

This section analyzes the proposed Cloptimus Scheduler (CS) for the case study, and compares CS with First Fit (FF) policy. We first analyze an initial period when the applications are deployed, and then the whole day with a varying workload.

A. Initial period

We first added 800 applications with a total of 5,027 VMs corresponding to the workload between 5 PM and 6 PM to a pod of 1,600 servers. For each application, CS uses the Tabu Search algorithm to place its VMs. In another experience with the same applications and VMs, the policy to place each VM was FF, that is to choose for each VM the first server with enough capacity.

Table I shows the results obtained by the two policies. The average transmission delay was 70% higher in FF than in CS, and the average link utilization was 9.9% higher in FF than CS. That is because CS has the minimization of delay as first priority, and FF does not take traffic into account to place VMs. When an application with multiple VMs is deployed, FF picks the first the server for the first VM. If that server has not enough space for the second VM, then it will be placed in

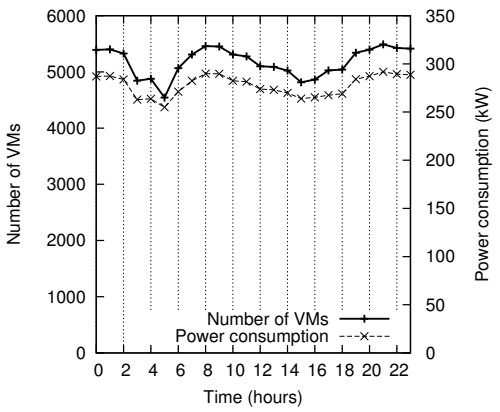

Fig. 6. Number of VMs used and power consumption in each period.

a second server. The traffic between these VMs will use the links between the servers and the rack switch. On the other hand, CS will pick a server with enough space for two VMs, and the traffic between both VMs will be kept within the server and will not use any link.

Considering the links that connect rack and pod switches, the average link utilization was 1.4% higher in FF than CS. That improvement is because CS minimizes delay by placing the VMs of each application in a single rack, and reduces the use of inter-rack links. Because FF places the VMs of an application in consecutive servers, they will often be placed in the same rack. However, when a rack is almost full, an application will be split and the links between rack and pod switches will be used. As we will see, the difference between CS and FF is larger when applications are resized over the day.

With respect to power consumption, both mechanisms are power-efficient because they tend to fill an active server with VMs before turning on an inactive server. In some cases, CS will temporally consume more power because it will prefer to turn on a server to place VMs of the same application in the same rack. However, the VMs of the next applications will use the server that had been kept partially occupied, and the total number of servers used is almost the same.

CS minimizes power consumption by taking into account the consumption of each type of server. In the case shown, all the racks have servers with the same type of energy consumption. When racks of energy-efficient servers and energy-inefficient servers are alternated, CS consumes 332 kW and FF, 357 kW, an 8% advantage for CS.

B. Workload

Once the 800 applications are deployed, we consider a workload that varies over the day. Each application is resized each hour according to the maximal workload expected for that hour. New VMs are deployed when the workload increases in size, and VMs are removed in periods where the workload decreases. Figure 6 shows the number of VMs used to match the workload in each period and the power consumption achieved by CC in a pod of 1,600 servers.

TABLE II
CLOPTIMUS SCHEDULER VS FIRST-FIT IN THE HIGHEST PERIOD (9 PM).

Solution:	CS	FF
Avg. delay	13.5 µs	19.9 µs
Avg. link utilization	20.7%	31.0%
Avg. inter-switch link util.	15.2%	27.3%
Max. inter-switch link util.	47.0%	80.0%
Number of VMs	5,490	5,490
Number of servers	1,385	1,373
Total power	291.8 kW	290.6 kW

First, we compare how much energy is saved by considering the variation of workload instead of dimensioning for the peak period. In this case, the peak period is at 9 PM when 291.8 kW are consumed by the VMs placed by CS in a pod, or 23.3 MW if the pod power consumption is extrapolated to the whole data center. If the servers used in that peak period consume that power during a year, 204.4 millions kWh would be consumed. Multiplying by an electricity price of $0.16 / kWh [32] gives a total of $32.7 millions for the electricity employed by the data center during a year. Using CS to resize applications during the day and suspending unused servers saves 4.9% of energy consumption, that becomes $1.6 millions per year. These savings are from the cloud provider point of view. Users that deploy applications would achieve higher savings in the cost paid to the provider to host VMs. Applications whose workload have more variability during the day benefit the most.

CS is then compared to the FF policy at the time of resizing applications each hour of the day. Table II presents results of both policies in the peak period. The first remark that can be highlighted is that FF produces 47% more delay than CS. That also implies that the links are 10.3% more used, thus increasing delay and degrading the Quality of Service (QoS). In particular, links that connect rack switches with pod switches reach up to 80% utilization in FF. That is because 32 racks are used and 8 racks remain free after placing the applications in the initial period. VMs added in high activity periods are then placed in the last racks. The traffic of the new VMs is directed to the load balancers placed in the first racks using the links between rack and pod switches. On the other hand, CS uses 40 racks from the starting period leaving available space in each rack for VMs that arrive in high periods. Then, CS places each new VM in the rack that hosts the corresponding application and thus reduces the link utilization between racks and improves the QoS. The cost to pay is that 12 more servers are used in the pod of 16,000 servers, and power consumption increases 0.4% compared to FF in the case where all servers are equally energy-efficient. The energy consumed in the whole day is 0.3% higher in CS than FF because a few more servers were used to improve the QoS.

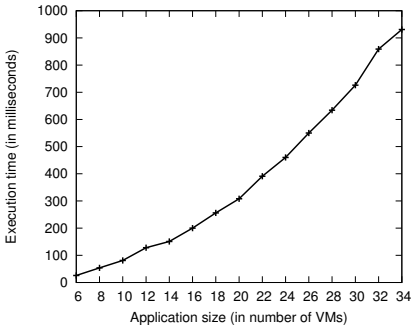

Fig. 7. Average scheduler execution time over the application size.

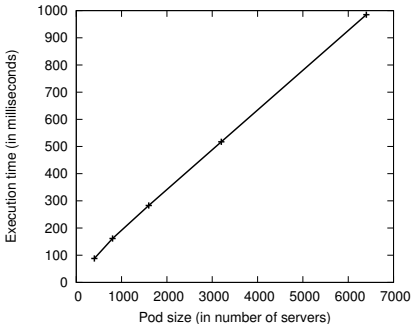

Fig. 8. Average scheduler execution time over the network size.

C. Execution time

The execution time of CS was evaluated for different application sizes and number of servers. To vary the application size, an empty pod with 1,600 servers was gradually filled with applications with between 6 and 40 VMs each. For each application, the time to schedule the application was recorded. With that information, the average execution time for each application size was calculated as seen in Figure 7.

To analyze the variation of scheduling time as a function of the network size, the number of VMs in each application was set as 20 and we varied the number of racks, with 40 servers per rack. For each network size, applications are scheduled until all the servers are full with VMs. The average scheduling time was calculated for each test. Figure 8 shows that the scheduling time of an application grows linearly as a function of the number of racks or servers. In particular, scheduling an application with 20 VMs in a pod of 1,600 servers take an average of 283 ms.

Concerning the workload variation over the day, the scheduler must resize each application each hour of the day. That implies choosing the servers to place new VMs in the periods when an application expands. The fact that only the new VMs are deployed makes the operation very quick compared to the operation of initial application deployment. Adding 800 applications with their 5,027 VMs in a pod of 1,600 servers took 33 seconds in the initial period. Resizing those same applications only took a total of half a second in each period.

VIII. Discussion

The proposed method shows the importance of taking into account the traffic among VMs to improve the QoS. An alternative to reduce delay and increase throughput among VMs is to augment the link capacity between switches. However, that approach has a high cost in number of switches and power consumption not justifiable when not all applications require high throughput among VMs. These topologies are indeed useful for specific applications that do require an all to all communication pattern such as Hadoop clusters. Experiences in this article study communication delay, throughput, and link utilization, metrics that vary depending on which server host each VM. The server processing time was not included because the processing power is a fixed requirement of each VM type, e.g., in number of virtual CPUs in a specific frequency. The number of VMs that will be assigned to an application will determine the number of requests per unity of time that each VM will handle and its processing time. However, the total response time is composed of the processing time of each VM that participates, e.g., load balancer, web server, database, memcache, and the communication delay. The way these times are combined is difficult to predict precisely because that will depend on how the application is implemented. For instance, if a web VM makes requests to multiple databases in parallel or in a sequential fashion. We believe that the relation between communication delay and total response time should be analyzed through measurements in specific implementations. The fact that the transmission time is expressed in microseconds should not mislead to think that is negligible compared to a total response time in milliseconds because each application request can require multiple internal requests among VMs and because if congestion is not avoided the queuing delay could increase. As this article shows, our approach reduces communication delay and the total response time as a result.

IX. Conclusion

This article presented an online/offline network-aware method to place VMs in servers, optimizing communication among VMs and power consumption. It also considered server heterogeneity and VMs with different types of requirements. Taking advantage of topology regularity made possible to scale in the delay calculation and parallelization on multiple clusters. The proposed scheduler, formalized through a MIP model, also considers the elasticity of applications to improve the QoS of VMs that arrive at peak periods. The developed Tabu Search heuristic scaled to more than 100,000 servers and applications with a resolution time in the miliseconds. Applications can be added and removed on the fly, and the number of VMs of an application can be changed to match the workload that varies over the day. A case study shows the advantages of the proposed approach. Compared to first-fit policy, the delay among VMs is reduced by 70% and the most used network link utilization is decreased 33%. We also found that a 4.9% of power consumption is saved compared to statically provisioning of resources for the peak period.

REFERENCES

[1] M. Bari, R. Boutaba, R. Esteves, L. Granville, M. Podlesny, M. Rabbani, Q. Zhang, and M. Zhani, "Data center network virtualization: A survey," *IEEE Communications Surveys and Tutorials*, vol. 15, no. 2, pp. 909–928, jan 2013. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6308765

[2] F. Kong and X. Liu, "A survey on green-energy-aware power management for datacenters," *ACM Computing Surveys*, vol. 47, no. 2, pp. 1–38, nov 2014. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2658850.2642708

[3] B. Jennings and R. Stadler, "Resource management in clouds: Survey and research challenges," *Journal of Network and Systems Management*, vol. 23, no. 3, pp. 567–619, mar 2014. [Online]. Available: http://link.springer.com/10.1007/s10922-014-9307-7

[4] F. Lopez-Pires and B. Baran, "Virtual Machine Placement Literature Review," no. 1, jun 2015. [Online]. Available: http://arxiv.org/abs/1506.01509

[5] R. Camati, A. Calsavara, and L. Lima, "Solving the virtual machine placement problem s a multidimensional knapsack problem," in *ICN14: The Thirteenth International Conference on Networks*, 2014, pp. 1–8.

[6] M. Ferdaus, M. Murshed, R. Calheiros, and R. Buyya, "Network-aware virtual machine placement and migration in cloud data centers," in *Emerging Research in Cloud Distributed Computing Systems*. IGI Global, 2015, pp. 42–91. [Online]. Available: http://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/978-1-4666-8213-9.ch002

[7] X. Li, J. Wu, S. Tang, and S. Lu, "Let's stay together: Towards traffic aware virtual machine placement in data centers," in *Proceedings - IEEE INFOCOM*. IEEE, apr 2014, pp. 1842–1850. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6848123

[8] K. Shahzad, A. Umer, and B. Nazir, "Reduce VM migration in bandwidth oversubscribed cloud data centres," in *2015 IEEE 12th International Conference on Networking, Sensing and Control*. IEEE, apr 2015, pp. 140–145. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7116024

[9] R. da Silva and N. da Fonseca, "Topology-aware virtual machine placement in data centers," *Journal of Grid Computing*, no. November 2014, sep 2015. [Online]. Available: http://link.springer.com/10.1007/s10723-015-9343-x

[10] H. Matsuba, K. Joshi, M. Hiltunen, and R. Schlichting, "Airfoil: A topology aware distributed load balancing service," in *IEEE 8th International Conference on Cloud Computing*, 2015, pp. 325–332.

[11] Z. Zhang, C. Hsu, and M. Chang, "Cool cloud: A practical dynamic virtual machine placement framework for energy aware data centers," in *2015 IEEE 8th International Conference on Cloud Computing*. IEEE, jun 2015, pp. 758–765. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7214115

[12] X. Wang, X. Wang, H. Che, K. Li, M. Huang, and C. Gao, "An intelligent economic approach for dynamic resource allocation in cloud services," *IEEE Transactions on Cloud Computing*, vol. 3, no. 3, pp. 275–289, jul 2015. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7097011

[13] L. Kou and G. Markowsky, "Multidimensional binpacking algorithms," *IBM Journal of Research and Development*, vol. 21, no. 5, pp. 443–448, 1977.

[14] S. Marcon and M. Marcellos, "Predictor: providing fine grain management and predictability in multi-tenant datacenter networks," in *IFIP/IEEE Integrated Network Management Symposium*, 2015.

[15] L.Chen and H. Shen, "Consolidating complementary VM with spatial/temporal awareness in cloud datacenters," in *INFOCOM*, 2014, pp. 1033–1041.

[16] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, "Multi-resource packing for cluster scheduling," in *SigComm*. ACM, 2014.

[17] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *Proc. IEEE INFOCOM*, 2010.

[18] 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*, vol. 57, no. 1, pp. 179–196, 2013.

[19] C. Guo, G. Lu, H. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "Secondnet: a data center network virtualization architecture with bandwidth guarantees," in *Proc. 6th ACM International Conference on Emerging Networking EXperiments and Technologies (CoNEXT)*, 2010, pp. 15:1–15:12.

[20] J.Lee, Y. Turner, M. Lee, S. Banerjee, J.-M. Khang, and P. Sharma, "Application driven bandwidth guarantees in datacenters," in *SigComm*. ACM, 2014.

[21] F. Larumbe and B. Sansò, "A tabu search heuristic for the location of data centers and software components in green cloud computing networks," *IEEE Transactions on Cloud Computing*, vol. 1, no. 1, pp. 22–35, 2013.

[22] Y. Li and M. Chen, "Software-defined network function virtualization: A survey," *IEEE Access*, vol. 3, pp. 2542–2553, 2015.

[23] X. Li and C. Qian, "Survey of network function placement," in *13th IEEE Annual Communications and Networking Conference*, 2016, pp. 1–6.

[24] Y. Xie, Z. Lieu, S. Wang, and Y. Wang, "Service function chaining resource allocation: A survey," ArXiV, Jul. 2016.

[25] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, D. Stafford, T. Tung, and V. Venkataramani, "Scaling memcache at Facebook," in *Proc. 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, Apr. 2013, pp. 385–398.

[26] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proc. 2nd USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2005, pp. 273–286.

[27] C. Walsworth, E. Aben, K. Claffy, and D. Andersen, "The CAIDA anonymized 2013 internet traces," 2013. [Online]. Available: http://www.caida.org/data/passive/passive_2013_dataset.xml

[28] Amazon, "Amazon Elastic Compute Cloud (EC2)," Jan. 2013. [Online]. Available: http://aws.amazon.com/ec2/

[29] D. Meisner, B. Gold, and T. Weinisch, "Powernap: Eliminating server idle power," in *Proceeding of the 14th international conference on Architectural support for programming languages and operating systems*, 2009, pp. 205–216.

[30] A. Gandhi, M. Harchol-Balter, and M. A. Kozuch, "The case for sleep states in servers," in *Proc. 4th ACM Workshop on Power-Aware Computing and Systems (HotPower)*, 2011, pp. 2:1–2:5.

[31] A. Beloglazov, R. Buyya, Y. C. Lee, and A. Zomaya, "A taxonomy and survey of energy-efficient data centers and cloud computing systems," *Advances in Computers*, vol. 82, no. 2, pp. 47–111, 2011.

[32] U.S. Energy Information Administration, "Average retail price of electricity to ultimate customers by end-use sector," May 2013. [Online]. Available: http://www.eia.gov/todayinenergy/detail.cfm?id=4530