STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXI**, Number 2, 2016

GAUSSIAN-TYPE RESOURCE ALLOCATION POLICIES FOR VIRTUALIZED DATA CENTERS

CORA CRĂCIUN AND IOAN SALOMIE

Abstract. This paper defines two Gaussian-type resource allocation policies for virtualized data centers. The policies allocate physical resources to virtual machines in a more relaxed way than server consolidation methods, but in a more constraint way than load balancing methods. The Gaussian policies are compared with the First Fit and Best Fit heuristics regarding the energy consumption in data centers and some performance metrics. The assessment is performed by simulation, for virtualized data centers with sufficient physical resources and time-varying workloads.

1. Introduction

Jobs' performance and consumed energy in data centers depend on the scheduling and resource allocation methods used for jobs' deployment [4]. Time performance improves if the workload is evenly distributed between the available physical machines and the machines are run at their full processor capacity. This maximum performance design, however, becomes power inefficient when the physical machines are provisioned to work at their peak power, even when their utilization is low. Typical servers in data centers work at 10–50% from their capacity [1], while their optimum utilization is 70–80% [2]. Different Dynamic Power Management (DPM) techniques [17, 9] have been proposed to address this issue of servers' low utilization in data centers. DPM techniques consolidate the workload on few servers and switch the unused resources off or place them in low power-performance states [7]. In dynamic systems, however, the jobs' resource requirements are time-varying and thus, at run-time, the consolidated physical machines may become overused. In such

Received by the editors: October 14, 2016.

2010 *Mathematics Subject Classification.* 68M14, 68M20.

1998 *CR Categories and Descriptors.* C.2.4 [**Computer-Communication Networks**]: Distributed Systems – *Network operating systems*; C.4 [**Computer Systems Organization**]: Performance of Systems – *Design studies*.

Key words and phrases. Gaussian-type policies, resource allocation, virtual machine migration.

cases, the jobs receive less resources than required, wait for free resources, or migrate to other hosts. In all cases, the jobs' performance decreases.

In this paper, we propose an alternative approach to resource consolidation and load balancing methods, for virtualized data centers. For this, we define two Gaussian-type resource allocation policies for mapping jobs encapsulated in virtual machines (VMs) on physical hosts. These policies pack the VMs on hosts less tightly than the consolidation methods, but more tightly than the load balancing methods. With this approach, the VMs are assigned with a higher priority to those hosts which are neither overused nor underused. In the assignment process, the new policies maximize Gaussian-type resource allocation functions depending on the available and the required physical resources. The shape of the mapping functions may be adjusted by changing some parameters.

The Gaussian-type policies are evaluated by simulation, for deploying time-varying workloads in a virtualized data center with sufficient physical resources. The simulations are performed using a dynamic resource management framework built upon the Haizea lease management scheduler [11, 19, 18]. The aim of this work is to evaluate the effect of the proposed policies on the following metrics: the energy consumption in data centers, the number of VMs migrations, the mean number of active hosts, and the VMs' total flow time. The Gaussian policies are compared with the greedy First Fit (FF) and Best Fit (BF) heuristics and with a load balancing method.

Although the two Gaussian-type policies are evaluated here only by simulation, they may be tested in real environments. For example, Haizea works not only as a simulator, but also as a backend VM scheduler for the OpenNebula resource management framework [18, 15]. Extensions of Haizea may as well be adapted to work in such real conditions.

This paper is structured as follows. Next section presents other investigations related to current work. Section 3 defines the Gaussian-type resource allocation policies and illustrates their behavior with respect to FF and BF. In Section 4, the Gaussian and reference policies are evaluated by simulation in data centers with sufficient physical resources. Final section summarizes current work.

2. Related work

Static and dynamic resource allocation in data centers may be performed using exhaustive search, optimization methods based on variants of bin or vector packing, or approximation algorithms. Many resource allocations use heuristics, which are fast but lead to approximate and suboptimal solutions [5, 20, 10].

In order to achieve server consolidation in virtualized environments, the approach in [6] combines "effective VM sizing" with variants of First Fit Decreasing (FFD) and Best Fit Decreasing (BFD) heuristics, or with information about VM hosting history and correlation. "Effective sizing" is based on stochastic bin packing, which allows bin overflowing with a given probability. The pMapper framework [21] uses several VM-to-host mapping algorithms to solve a cost-performance-power optimization problem in virtualized heterogeneous clusters. Entropy [12] uses constraint programming in order to minimize the number of active servers and the VM migration number in homogeneous clusters. The work in [23] proposes a multi-objective method for dynamic placement of VMs in data centers. This method improves the power efficiency by up to 20% and performance by up to 30%, and decreases the number of VMs migrations by up to 80%.

The VM-to-host mapping functions used by the Gaussian-type policies defined in this paper are close to the assignment/migration probability functions presented in reference [14]. In their work, the authors define policies based on statistical Bernoulli trials, in order to reduce the energy consumption in cloud data centers, without performance degradation. Normal distributions are also used to model the VMs' resource requirements, as in references [13] and [6]. In our case, the VMs' requirements are uniformly distributed random numbers, while the Gaussian-type policies are deterministic and guide the VM-to-host mapping process.

3. The Gaussian-type resource allocation policies

Let consider that a data center contains N_H ordered physical machines and N_V VMs must be assigned to the hosts based on their CPU requirements. We denote by R_{CPU} a VM's required CPU share and by T_{CPU}, U_{CPU}, and A_{CPU} a host's total, used, and available CPU resources, respectively (all quantities are in percents and $T_{\text{CPU}} = U_{\text{CPU}} + A_{\text{CPU}}$). Feasible hosts for a given VM requiring R_{CPU} resources are those with $A_{\text{CPU}} \geq R_{\text{CPU}}$. Since the VMs' CPU requirements are time-varying, at run-time, some hosts may become overused. In such cases, selected VMs from the overloaded hosts migrate to other feasible physical machines.

The Gaussian-type policies described in the following use two resource allocation procedures. In one case, the VMs are queued and are mapped on feasible hosts by a policy depending only on the hosts' properties. In the other case, the VMs are not ordered and are assigned to hosts by a policy depending both on the VMs' and hosts' properties. The first resource allocation procedure is suitable for on-line conditions, because the search is performed only

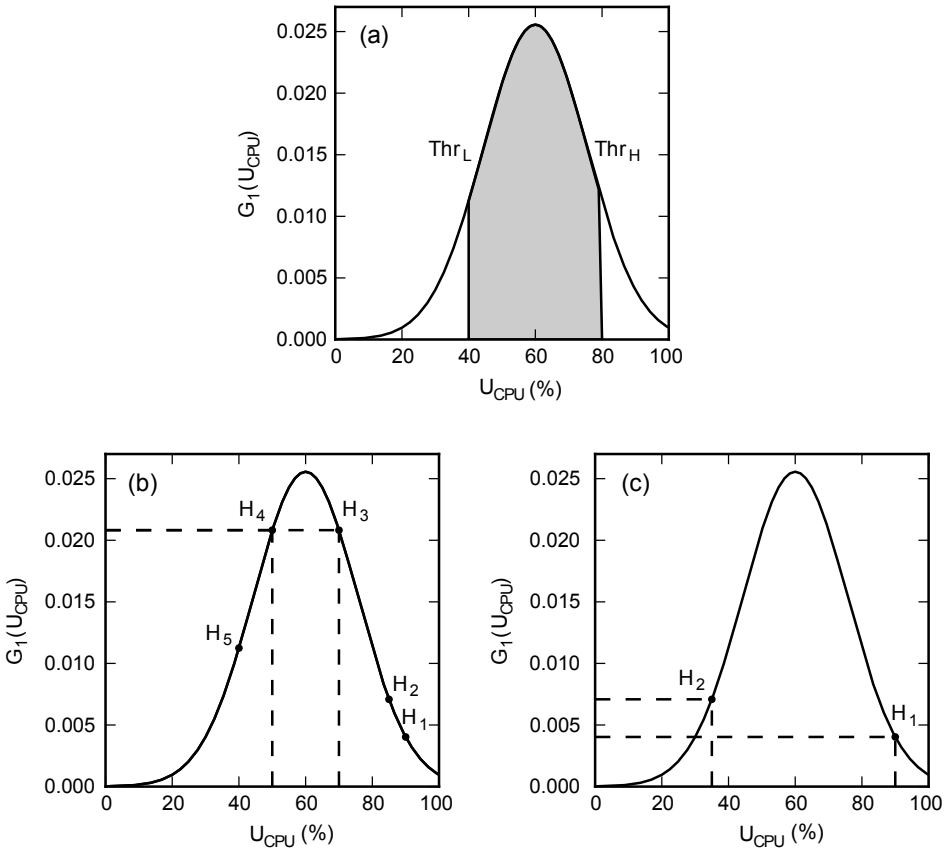

FIGURE 1. (a) The $G_1(U_{\mathrm{CPU}})$ Gaussian-type resource allocation function. The thresholds in formula (2) are $Thr_{\mathrm{L}} = 40$ and $Thr_{\mathrm{H}} = 80$, and the area of the shaded region is $a = 0.8$. (b-c) Examples of resource allocations using the G1 policy, for the hosts H_1–H_5

among the hosts. The second procedure, however, is more suitable for off-line conditions, since the search is both among VMs and hosts.

3.1. **The G1 policy.**

3.1.1. *The G_1 resource allocation function.* The G1 resource allocation policy assigns a given VM from a queue to the feasible host that maximizes the $G_1(U_{\mathrm{CPU}})$ function. This function resembles the t-distribution defined in reference [1], which models the hosts' utilization when they deploy VMs with randomly distributed resource requirements. There, the t-distribution is used

to compute upper and lower dynamic utilization thresholds for the hosts. For each host, the parameters of the t-distribution are estimated from the host's historical usage over some time period. Physical resources are allocated to the VMs using the power-aware Modified Best Fit Decreasing policy [1]. In our case, the resource allocation policy is of Gaussian-type and we map the VMs on hosts by comparing the hosts' $G_1(U_{\mathrm{CPU}})$ functions. As in reference [1], however, with this policy we aim to avoid the under or overusage of the physical resources.

For a feasible host, G_1 depends on the host's CPU utilization, U_{CPU}, and is a Gaussian centered at the middle point of the threshold interval $[Thr_{\mathrm{L}}, Thr_{\mathrm{H}}]$ (Fig. 1). The area below the Gaussian function and between the two thresholds is a specified value $a \in (0, 1)$. Then, the G_1 function is

$$G_1(U_{\mathrm{CPU}}) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{(U_{\mathrm{CPU}} - \mu)^2}{2\sigma^2}\right], \tag{1}$$

where

$$\mu = \frac{Thr_{\mathrm{L}} + Thr_{\mathrm{H}}}{2},\ \sigma = \frac{Thr_{\mathrm{H}} - Thr_{\mathrm{L}}}{2\sqrt{2}\,\mathrm{erf}^{-1}(a)}, \tag{2}$$

and erf^{-1} is the inverse error function. For the unfeasible hosts, the G_1 function is zero.

The thresholds have close meaning to that used in the literature [1, 13]. The hosts with higher utilization than Thr_{H} are overused and those with lower utilization than Thr_{L} are underused. In case of G1 policy, however, the VMs may still be mapped on feasible hosts with utilization falling outside the threshold interval, but with lower chances than to the other hosts.

The shape of the G_1 function may be adjusted by changing the thresholds or the a parameter. The mean value of the thresholds imposes the G_1 function's location. The distance between the thresholds and the a parameter, on the other hand, influence in opposite ways the Gaussian's width. At limit, when a is close to zero, G_1 degenerates in the zero function. In this case, the G1 policy treats all hosts equally and if G1 uses the $1 - G_1$ resource allocation function, then it behaves as the FF policy.

Each host is characterized by its G_1 function. The hosts with the same Thr_{L}, Thr_{H}, and a parameters own identical functions. The hosts also having the same CPU utilization, U_{CPU}, are equally likely to accept or reject a VM. All homogeneous physical machines from a data center may use the same resource allocation function G_1. On contrary, heterogeneous hosts from the same data center or identical hosts from different data centers may use distinct functions.

3.1.2. *Example using the G1 policy.* An example of resource allocation using the G1 policy is presented in Figure 1b. The thresholds have been chosen based on the values reported in the literature [3, 22]. The $a = 0.8$ parameter ensures that most part of the G_1 function lies inside the [0%,100%] utilization range. Only H_3, H_4, and H_5 are feasible hosts for a VM requiring an $R_{\text{CPU}} =$ 20% CPU share. Both H_3 and H_4 have the same maximal $G_1(U_{\text{CPU}})$ value for this VM. In case of ties, the resource allocation algorithm chooses the lowest-indexed host, here H_3. In general, the chosen host may be the one with a higher or a lower CPU utilization. To remove this ambiguity, an improved algorithm may select consistently one of the hosts, based on CPU utilization criteria. Even better, the algorithm may adapt to run-time conditions and choose the host with high CPU utilization for resource consolidation and the host with low CPU utilization for load balancing.

3.1.3. *Comparison of G1 with FF and BF policies.* Let now compare the G1 resource allocation policy with the FF and BF heuristics. If the hosts are identical and initially empty, they are all feasible and characterized by the same $G_1(0)$ value. Thus, the G1 policy assigns a given VM to the first host, H_1. Unless H_1 becomes unfeasible, next VMs in the queue are also allocated to this host, because its $G_1(U^1_{\text{CPU}})$ value is maximal among all hosts. Up to this point, G1, FF, and BF behave identically. Let assume now that the host H_1 is 90% used and the current VM requires a 35% CPU share. Since H_1 lacks resources, all policies assign the VM to the next host, H_2. Both H_1 and H_2 are feasible destination hosts for the following VM requiring a 10% CPU share. FF assigns the VM to the first feasible host, namely H_1. BF chooses the same host, because H_1 remains with less free resources than H_2 after VM assignment. On contrary, the G1 policy selects the host H_2, because $G_1(U^2_{\text{CPU}} = 35) > G_1(U^1_{\text{CPU}} = 90)$ (Fig. 1c).

In case of FF and BF policies, an increase in the VMs' CPU requirements on host H_1 would immediately trigger some VMs migrations from H_1 to other hosts. In case of G1, however, the host H_1 still has some free resources and may afford such resource requirements increases. Therefore, G1 promotes resource consolidation, but in a less greedy way than FF and BF. In addition, G1 may reduce the number of VMs migrations and avoid their drawbacks: the jobs' performance degradation, the electrical power overhead for the hosts involved in migration, and the network use.

3.2. **The G2 policy.**

3.2.1. *The G_2 resource allocation function.* The G2 policy favors those allocations in which the VMs' required resources are close to some fraction $\alpha \in (0, 1]$ from the hosts' available resources. To our knowledge, this policy has not been

previously used for resource allocation. Unlike G1, G2 uses a resource allocation function $G_2(R_{\text{CPU}}, A_{\text{CPU}})$, which depends both on the required and the available resources. For the unfeasible hosts, the G_2 function is zero. The G2 policy may allocate resources either for ordered VMs or for sets of VMs.

In case of VM queues, the next VM to be mapped on hosts is known at each scheduling time. As a result, for a given VM, the G2 policy finds the host that maximizes the one-variable function $G_2([R_{\text{CPU}}], A_{\text{CPU}})$, with fixed R_{CPU} value and variable A_{CPU}. From now on, we include a fixed variable of the G_2 function in square brackets. In case of VM sets, on the other hand, the G2 policy chooses iteratively the feasible $(VM, host)$ pairs that maximize the two-variable function $G_2(R_{\text{CPU}}, A_{\text{CPU}})$.

The G_2 function is defined as follows. For a host with A_{CPU} available resources, G_2 is a Gaussian centered at $R_{\text{CPU}} = \alpha A_{\text{CPU}}$. Moreover, G_2 decreases at a fraction $r \in (0, 1)$ from its peak value when $R_{\text{CPU}} = \alpha A_{\text{CPU}} \pm A_{\text{CPU}}/2$. Thus, the G_2 function is

$$G_2(R_{\text{CPU}}, A_{\text{CPU}}) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{(R_{\text{CPU}} - \alpha A_{\text{CPU}})^2}{2\sigma^2}\right], \tag{3}$$

where

$$\sigma = \frac{A_{\text{CPU}}}{2\sqrt{2\ln(1/r)}}. \tag{4}$$

Both the location and width of the G_2 function depend on the host's available resources. When A_{CPU} changes – because new VMs are mapped on the host, other VMs finish their work, or the CPU requirements of the running VMs are changing – the host's G_2 function, $G_2(R_{\text{CPU}}, [A_{\text{CPU}}])$, moves along the R_{CPU} axis and changes its width.

Figure 2 contains a three-dimensional representation of the $G_2(R_{\text{CPU}}, A_{\text{CPU}})$ function, when $A_{\text{CPU}} \in [10\%, 100\%]$, $R_{\text{CPU}} \in [10\%, A_{\text{CPU}}]$, and the parameters are $\alpha = 0.5$ and $r = 0.001$. With these parameters, the Gaussian of a host with fixed A_{CPU} value is centered at $R_{\text{CPU}} = A_{\text{CPU}}/2$, and decreases at 0.001 from its peak value when $R_{\text{CPU}} = 0$ or $R_{\text{CPU}} = A_{\text{CPU}}$. Therefore, the VMs and the hosts are better matched if the required resources are close to half the available ones.

3.2.2. *Example using the G2 policy.* Let exemplify the use of the G2 policy for mapping a set of four VMs on three hosts. The hosts have the following available resources: $A^1_{\text{CPU}} = 50\%$ (H_1), $A^2_{\text{CPU}} = 70\%$ (H_2), and $A^3_{\text{CPU}} = 100\%$ (H_3). The VMs require the CPU shares $R^1_{\text{CPU}} = 10\%$ (V_1), $R^2_{\text{CPU}} = 20\%$ (V_2), $R^3_{\text{CPU}} = 35\%$ (V_3), and $R^4_{\text{CPU}} = 30\%$ (V_4). The resource allocation functions G_2 of all hosts have the same parameters $\alpha = 0.5$ and $r = 0.001$.

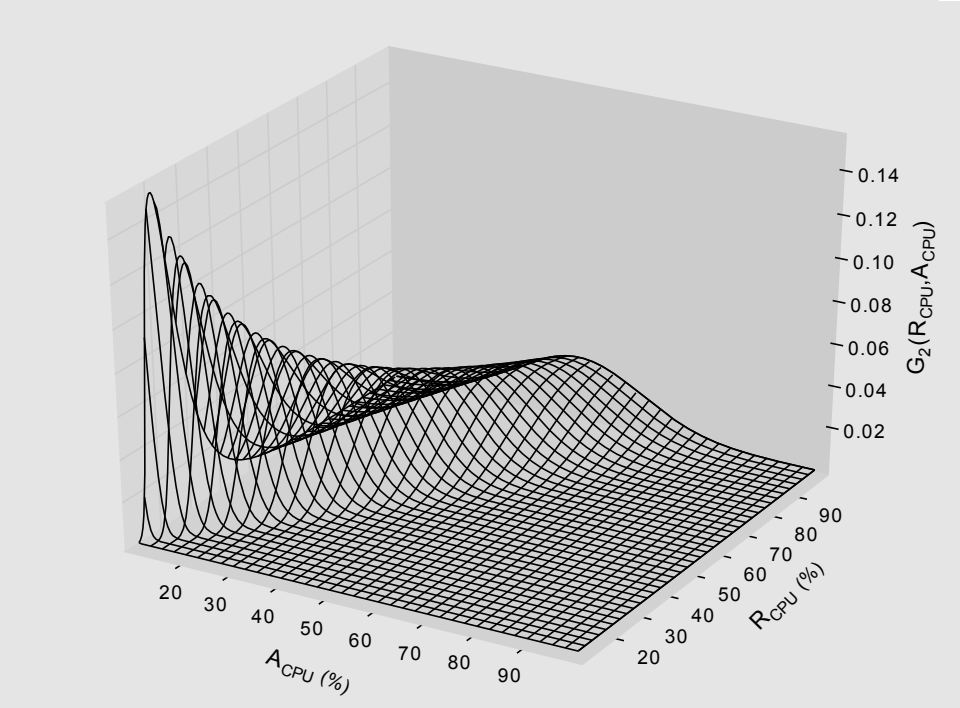

FIGURE 2. Three-dimensional representation of the $G_2(R_{\text{CPU}}, A_{\text{CPU}})$ Gaussian-type function, with parameters $\alpha = 0.5$ and $r = 0.001$ (formulas 3–4)

The VM-to-host mapping process is illustrated in Figure 3. Each subfigure corresponds to a resource allocation iteration step. Each step selects from the unassigned VMs and from the feasible physical machines the ($VM, host$) pair with the maximal $G_2(R_{\text{CPU}}, A_{\text{CPU}})$ value. Subfigures (a) to (d) are given in the order in which the G2 policy selects the ($VM, host$) pairs. Each Gaussian curve is the $G_2(R_{\text{CPU}}, [A_{\text{CPU}}])$ function of a host (continuous line for H_1, dashed line for H_2, and dotted line for H_3), at fixed A_{CPU} and variable R_{CPU}. The vertical lines located at 10%, 20%, 35%, and 30% R_{CPU} values indicate the VMs' required CPU shares. The vertical line for a VM with R_{CPU} requirements has a length equal with the maximal $G_2([R_{\text{CPU}}], A_{\text{CPU}})$ value among all hosts. The ($VM, host$) pair selected at each step is indicated with a dot on the host's Gaussian curve. The four steps of the mapping process are the following:

- Step 1: Figure 3a shows the Gaussian functions $G_2(R_{\text{CPU}}, [A^1_{\text{CPU}} = 50])$ for H_1, $G_2(R_{\text{CPU}}, [A^2_{\text{CPU}} = 70])$ for H_2, and $G_2(R_{\text{CPU}}, [A^3_{\text{CPU}} =$

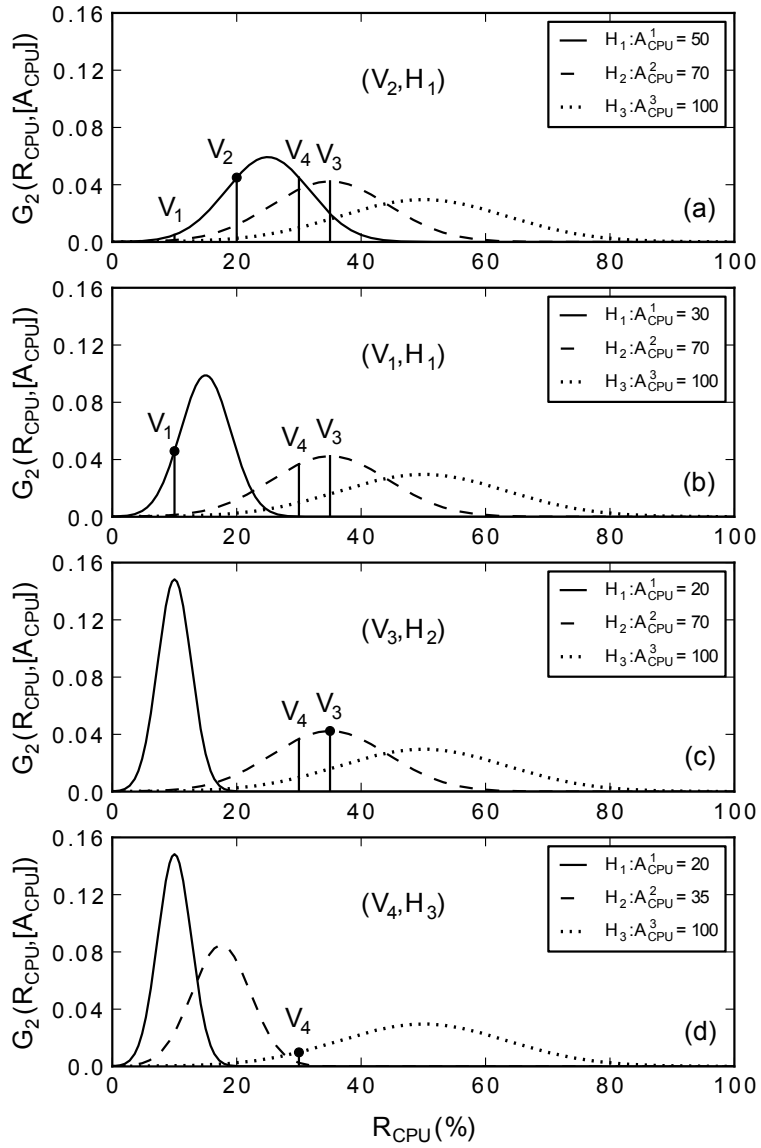

FIGURE 3. An example of mapping a set of four VMs (V_1, V_2, V_3, and V_4) on three hosts (H_1, H_2, and H_3), using the G2 policy

100]) for H_3. The $(VM, host)$ pair with the maximal G_2 value at this step is (V_2,H_1).

- Step 2: After Step 1, the host H_1 remains with $A^1_{\text{CPU}} = 30\%$ available resources. As a result, the G_2 function of H_1 moves towards lower values along the R_{CPU} axis. The Gaussian functions $G_2(R_{\text{CPU}}, [A^1_{\text{CPU}} = 30])$ for H_1, $G_2(R_{\text{CPU}}, [A^2_{\text{CPU}} = 70])$ for H_2, and $G_2(R_{\text{CPU}}, [A^3_{\text{CPU}} = 100])$ for H_3, as well as the required resources for the unmapped VMs V_1 (10%), V_3 (35%), and V_4 (30%) are shown in Figure 3b. The $(VM, host)$ pair selected at this step is (V_1,H_1).
- Step 3: After Step 2, the host H_1 remains with 20% available CPU. As before, the G_2 function of H_1 moves towards lower values along the R_{CPU} axis. The resource allocation functions $G_2(R_{\text{CPU}}, [A^1_{\text{CPU}} = 20])$ for H_1, $G_2(R_{\text{CPU}}, [A^2_{\text{CPU}} = 70])$ for H_2, and $G_2(R_{\text{CPU}}, [A^3_{\text{CPU}} = 100])$ for H_3, and the required resources of V_3 (35%) and V_4 (30%) are shown in Figure 3c. Now, the host H_1 lacks resources for both VMs. The final mapping at this step is (V_3,H_2).
- Step 4: After Step 3, the host H_2 remains with 35% available CPU. Figure 3d presents the Gaussian functions for the three hosts, $G_2(R_{\text{CPU}}, [A^1_{\text{CPU}} = 20])$ for H_1, $G_2(R_{\text{CPU}}, [A^2_{\text{CPU}} = 35])$ for H_2, and $G_2(R_{\text{CPU}}, [A^3_{\text{CPU}} = 100])$ for H_3. V_4 (30%) is the only VM still unmapped on a physical machine. The host H_1 is unfeasible for this VM and the host H_2 has a lower G_2 value than H_3. Finally, V_4 is assigned to the host H_3.

At Step 4, both FF and BF policies map the VM V_4 on host H_2 and keep the host H_3 unused. In conclusion, the G2 policy may use more physical machines than FF and BF, and thus may increase the energy consumption in data centers. A decrease in the VM migration number, however, compensates for this energy increase. Being less tightly packed with VMs, the hosts using the G2 policy have less chances to overflow at run-time, when the VMs' CPU requirements are changing. Therefore, less VMs migrate from one host to another.

4. Evaluation of the Gaussian-type policies

4.1. **Simulation experiments.** In this section, we compare by simulation the Gaussian-type policies with the greedy FF and BF heuristics and with a load balancing scenario. Simulation experiments consisted in deploying 40 VMs with time-varying resource requirements in a homogeneous data center with 20 identical hosts. The evaluation was performed using a resource management framework built upon the Haizea lease scheduler [11, 19, 18]. The framework was used for VM scheduling, resource allocation, and host management, and for computing several performance metrics and the energy consumption in virtualized data centers.

We enhanced the Haizea scheduler with the following facilities: (a) new resource allocation policies; (b) periodical change of the VMs' CPU requirements; (c) management of the overused hosts and of the suspended VMs; (d) computation of performance and energy metrics. Unlike reference [1], we considered only the overused hosts and not the underused ones when performing the VMs' migrations.

In the framework, the VMs have been modeled using Haizea's best-effort leases [19]. After their simultaneous arrival at the data center, the VMs were mapped on hosts according to their CPU requirements, the hosts' available resources, and the chosen resource allocation policy. For simulating a workflow, the VMs' CPU requirements were changed periodically at each 2 min. The VMs' CPU shares, R_{CPU} (in percents), were generated as uniformly distributed random numbers between 10 and 40, rounded up to the nearest integer values. The VMs' CPU traces lasted 300 min. The times were measured on the data center's clock.

All physical machines of the data center were single-processor and had the CPU capacity $T_{\mathrm{CPU}} = 100\%$. We considered that, while deploying VMs, the physical machines used a dynamic electrical power proportional to their CPU usage, U_{CPU} [8]. The idle power of each machine was 70% of the total power of 250 W at full CPU utilization [1]. Only the active machines consumed energy, the idle ones being switched off.

The framework performed the VM scheduling and resource allocation in a centralized way. The VMs were either queued at the data center, for their further mapping on hosts, or were selected from a VM set. After initial assignment, at run-time some VMs were suspended, because their resource requirements changed and their hosts became overused ($A_{\mathrm{CPU}} > T_{\mathrm{CPU}}$). Namely, the VMs of each overloaded host were first sorted increasingly by their CPU requests. Then, the VMs with the lowest requests were suspended in order, until the host's overload was cleared. Most suspended VMs were migrated to other hosts, but few VMs were resumed on the original hosts. The hosts were checked for overloading at each change of the VMs' CPU requirements. The VMs were relocated using the off-line migration procedure provided by the Haizea scheduler [11, 19, 18]. Unlike Haizea, we allowed more VMs to migrate simultaneously from the same host, without performance overhead. The migrated VMs were mapped on other hosts using the same resource allocation policy as the initial VMs.

Regardless of the resource allocation policy used, all suspended VMs, migrated or not, experienced a 19 s delay corresponding to a suspension / resumption rate of 32 MB/s. Other rates would have caused other delays. While suspended, the VMs did not consume CPU resources, neither on the initial nor on the final hosts. Because each VM required 300 min of effective processing,

its flow time was increased with the corresponding suspension delays. Each time the initial or suspended VMs were mapped on hosts, the Gaussian-type resource allocation functions were computed with the instantaneous R_{CPU}, U_{CPU}, and A_{CPU} values of the virtual or physical resources.

Simulation experiments were repeated 100 times. Within the same experiment, all compared resource allocation policies used a common set of 40 randomly generated VMs' CPU traces. A different random CPU trace was generated for each VM. Parameters of the G_1 resource allocation function were $Thr_{\mathrm{L}} = 40$, $Thr_{\mathrm{H}} = 80$, and $a = 0.8$. Parameters of the G_2 function were $\alpha = 0.5$ and $r = 0.001$.

4.2. **Results.** Figure 4 presents the following metrics computed from the 100 repeated experiments of deploying 40 VMs, with 300 min long CPU traces, in a data center with 20 hosts: the consumed energy, the mean number of active hosts in the entire makespan, the number of VMs migrations, the number of VMs suspensions without migration, the total number of VMs migrations and suspensions, and the VMs' total flow time (the sum of all VMs' processing times). Four scheduling and resource allocation policies were used for queued VMs (FFq, BFq, G1q, and G2q) and one policy for VM sets (G2s). Based on these results, we drew the following conclusions:

(a) In all experiments, the two G2 policies were less energy-efficient than FFq, BFq, and G1q, but generated a lower number of VMs migrations. The G2q and G2s policies performed similarly with respect to all metrics. In 52% of experiments, G2s consumed more energy than G2q, and in 59% of experiments, G2s generated less VMs migrations than G2q.

(b) In 67% of experiments, the G1q policy consumed more energy than FFq, but in 96% of experiments G1q generated less VMs migrations than FFq.

(c) The investigated policies showed similar relative behavior for the following metrics: 1. the energy consumption and the mean number of active hosts, and 2. the VM migration number and the flow time. The first similarity came from the fact that the energy consumption depended on how many hosts were switched-on and were deploying VMs. On the other hand, the VMs' total flow time followed the VM migration number because the CPU traces of all VMs were equally long, and the total migration time overhead depended on how many migrations occurred in the experiment.

(d) For all policies, the number of VMs suspensions without migration was very small (less than 3.2% from the total number of migrations and suspensions). These suspensions with resumption on the same hosts were caused by the host management procedure used in this work. From each overloaded host, the framework suspended in order the VMs with the lowest CPU request, until the overload was cleared. With this procedure, more VMs than necessary were

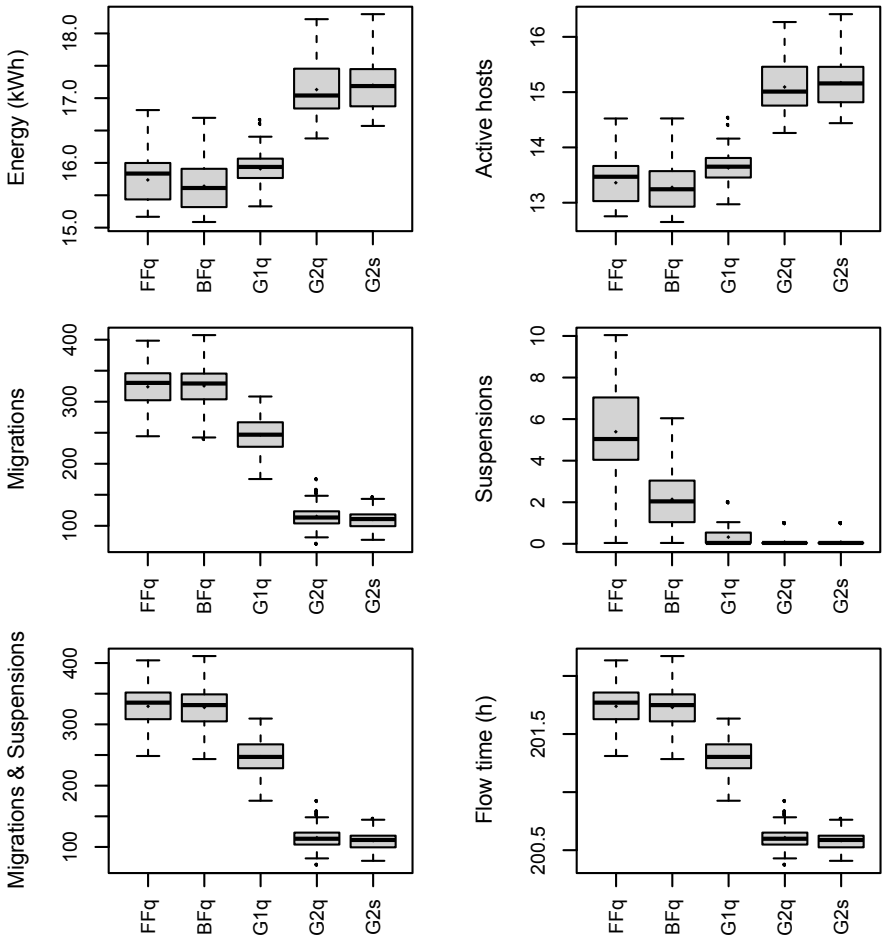

FIGURE 4. Boxplot representation [16] of the energy and performance metrics in 100 simulation experiments of deploying 40 VMs on 20 hosts. The horizontal lines inside the boxes indicate the data's median values and the full knots indicate the data's mean values.

suspended from some overloaded hosts. For few suspended VMs, the original hosts still had resources, and thus the VMs were resumed on the same hosts without migration. Other host management procedures, not considered here, may generate only VMs migrations.

(e) For each experiment, we computed the relative values (in percents) of the energy and VM migration number, for the G1q and G2q policies with

respect to the FFq policy. Then, we have calculated the mean and standard deviation of the relative values obtained for all 100 experiments. In case of G1q vs. FFq, the mean relative energy was 1.1% and the mean relative VM migration number was -23.4%. These values, however, had very high standard deviations (2.2% for energy and 10.5% for VM migration number). For G2q vs. FFq, the mean relative energy was 8.9% (sd = 3.1%) and the mean relative VM migration number was -64.5% (sd = 6.7%). Thus, the G2q policy decreased the VM migration number relative to FFq with more than 55%, with an energy overhead of up to 12%.

(f) Beside FF and BF, we have also compared the Gaussian-type policies with a load balancing scenario. This scenario used the same data center configuration (40 VMs and 20 hosts) and VMs' CPU traces as the other policies. Calculations were performed for the 100 simulation experiments considered previously. Two VMs were mapped on each host, in order to avoid the hosts' overloading at the VMs' maximal CPU requirements of 40%. In this setting, all VMs finished their work in 300 min, as required. The load balancing scenario led to a higher energy consumption (mean = 21.32 kWh and sd = 0.02 kWh, in 100 experiments), a smaller flow time (200 h), and no VM migration compared to all the other policies considered in present work (Fig. 4).

5. Conclusions

In this paper we have proposed two Gaussian-type resource allocation policies for virtualized data centers. In the resource allocation process, the two policies maximized Gaussian functions depending on some adjustable parameters and on the required and available resources. The Gaussian-type policies packed the VMs on hosts less tightly than the consolidation policies, but more tightly than the load balancing methods. Compared to First Fit and Best Fit, the Gaussian policies tended to use more physical machines, but they were more robust to the VMs' workload variations. As a result, the energy consumption in data centers was higher, but the number of VMs migrations was smaller. A drawback of the two policies is their exhaustive search in the VMs' or hosts' collections, which makes them suitable mostly for relatively small data centers. Similar Gaussian-type policies to those described in this paper may be defined for other physical resources than the CPU. For instance, the resource allocations constrained by multiple physical resources (CPU, memory, hard-disk, or network I/O) may use a policy based on a multidimensional Gaussian function. Solving such a resource allocation problem must consider which physical resources have correlated usage, which may be overprovisioned, shared between different VMs, or may be reclaimed from the VMs if not used.

Acknowledgments

The authors are grateful to the anonymous reviewers for their suggestions and comments, which improved the paper. The authors would like to thank a reviewer for suggesting to use a multidimensional Gaussian function in the future work.

References

[1] A. Beloglazov, R. Buyya, *Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers*, in Proceedings of the 8th International Workshop on Middleware for Grids, Clouds and e-Science (MGC'10), 2010, pp. 4:1-4:6.

[2] A. Beloglazov, R. Buyya, Y. C. Lee, A. Zomaya, *A taxonomy and survey of energy-efficient data centers and cloud computing systems*, M. Zelkowitz, ed., Advances in Computers 82, Elsevier, Amsterdam, 2011, pp. 47-111.

[3] A. Beloglazov, J. Abawajy, R. Buyya, *Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing*, Future Gener. Comput. Syst., 28 (2012), pp. 755-768.

[4] A. Beloglazov, R. Buyya, *Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers*, Concurr. Comput.: Pract. Exper., 24 (2012), pp. 1397-1420.

[5] M. Cardosa, M. R. Korupolu, A. Singh, *Shares and utilities based power consolidation in virtualized server environments*, in Proceedings of the 11th IFIP/IEEE International Symposium on Integrated Network Management (IM'09), 2009, pp. 327-334.

[6] M. Chen, H. Zhang, Y.-Y. Su, X. Wang, G. Jiang, K. Yoshihira, *Effective VM sizing in virtualized data centers*, in Proceedings of the 2011 IFIP/IEEE International Symposium on Integrated Network Management (IM'11), 2011, pp. 594-601.

[7] G. Dhiman, T. Simunic Rosing, *Dynamic power management using machine learning*, in Proceedings of the 2006 IEEE/ACM International Conference on Computer-Aided Design (ICCAD'06), 2006, pp. 747-754.

[8] X. Fan, W.-D. Weber, L. A. Barroso, *Power provisioning for a warehouse-sized computer*, in Proceedings of the 34th annual International Symposium on Computer architecture (ISCA'07), 2007, pp. 13-23.

[9] A. Gandhi, M. Harchol-Balter, R. Raghunathan, M. Kozuch, *Distributed, robust auto-scaling policies for power management in compute intensive server farms*, in Proceedings of the 2011 Sixth Open Cirrus Summit (OCS'11), 2011, pp. 1-5.

[10] L. Grit, D. Irwin, A. Yumerefendi, J. Chase, *Virtual machine hosting for networked clusters: building the foundations for "autonomic" orchestration*, in Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing (VTDC'06), 2006, pp. 7.

[11] Haizea. `http://haizea.cs.uchicago.edu/`

[12] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, J. Lawall, *Entropy: a consolidation manager for clusters*, in Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual execution environments (VEE'09), 2009, pp. 41-50.

[13] S. Kumar, V. Talwar, V. Kumar, P. Ranganathan, K. Schwan, *vManage: loosely coupled platform and virtualization management in data centers*, in Proceedings of the 6th International Conference on Autonomic computing (ICAC'09), 2009, pp. 127-136.

[14] C. Mastroianni, M. Meo, G. Papuzzo, *Self-economy in cloud data centers: statistical assignment and migration of virtual machines*, in Proceedings of the 17th International Conference on Parallel processing - Volume Part I (Euro-Par'11), 2011, pp. 407-418.
[15] OpenNebula. `http://www.opennebula.org/`
[16] The R Project for Statistical Computing. `http://www.r-project.org/`
[17] T. Simunic, *Energy efficient system design and utilization*, PhD Dissertation, Stanford University, Stanford, USA, 2001.
[18] B. Sotomayor, R. S. Montero, I. M. Llorente, I. Foster, *An open source solution for virtual infrastructure management in private and hybrid clouds*, IEEE Internet Computing, Special Issue on Cloud Computing, 2009.
[19] B. Sotomayor Basilio, *Provisioning computational resources using virtual machines and leases*, PhD Dissertation, Univ. of Chicago, Illinois, USA, 2010.
[20] M. Stillwell, F. Vivien, H. Casanova, *Virtual machine resource allocation for service hosting on heterogeneous distributed platforms*, in Proceedings of the 2012 IEEE 26th International Parallel & Distributed Processing Symposium (IPDPS'12), 2012, pp. 786-797.
[21] A. Verma, P. Ahuja, A. Neogi, *pMapper: power and migration cost aware application placement in virtualized systems*, in Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware (Middleware'08), 2008, pp. 243-264.
[22] VMware, Inc. *VMware® distributed power management concepts and use*, Technical White Paper, 2010. Available online: `http://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/Distributed-Power-Management-vSphere.pdf`
[23] J. Xu, J. Fortes, *A multi-objective approach to virtual machine management in datacenters*, in Proceedings of the 8th ACM International Conference on Autonomic computing (ICAC'11), 2011, pp. 225-234.

Department of Computer Science, Technical University of Cluj-Napoca, Romania

Faculty of Physics, Babeş-Bolyai University, Cluj-Napoca, Romania
E-mail address: `cora.craciun@phys.ubbcluj.ro`

Department of Computer Science, Technical University of Cluj-Napoca, Romania
E-mail address: `Ioan.Salomie@cs.utcluj.ro`