STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXII**, Number 1, 2017
DOI: 10.24193/subbi.2017.1.03

A FILTER-BASED DYNAMIC RESOURCE MANAGEMENT FRAMEWORK FOR VIRTUALIZED DATA CENTERS

CORA CRĂCIUN AND IOAN SALOMIE

Abstract. Data centers adapt their operation to changing run-time conditions using energy-aware and SLA-compliant resource management techniques. In this context, current paper presents a novel filter-based dynamic resource management framework for virtualized data centers. By choosing and combining properly software filters performing the scheduling and resource management operations, the framework may be used in what-if analysis. The framework is evaluated by simulation for deploying batch best-effort jobs with time-varying CPU requirements.

1. Introduction

High energy consumption and low Quality of service (QoS) are the main problems in data centers. Service providers aim to reduce the energy consumption, while the users demand high performance at low cost. Moreover, data centers are dynamic systems in which the users' requests and resource availability are time-varying. Therefore, data centers must adapt their operation to run-time conditions using appropriate scheduling and resource management strategies [22].

In this context, we present a novel filter-based dynamic resource management framework for virtualized data centers. The framework extends the Haizea lease scheduler (version 1.1) [15, 24, 26] and may assess the energy and performance efficiency of different resource management techniques. The framework uses software filters to perform the job scheduling, resource allocation, and virtual machines' migration operations in data centers. A data

Received by the editors: February 1, 2017.

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. Filter-based framework, Dynamic resource management, Virtualization.

center may adapt to run-time conditions by properly replacing and combining the filters. Current work uses CPU-related filters, but similar software components may be defined for other physical resources.

Presently, the framework has some limitations. For instance, it does not take into account the performance degradation when more virtual machines (VMs) are migrated simultaneously from or to the same physical machine. In addition, the framework considers the following simplified resource management scenarios and data center configurations: resource allocation policies with job queuing in which the jobs wait until they receive all required resources, single-core processing units, overloaded but not underused host management, small size homogeneous data centers, a single constraining physical resource (CPU), simultaneously arrived jobs at the data center, periodic change of the VMs' resource requirements, off-line VM migration as implemented in Haizea. The framework, however, is extensible with other filters, algorithms, or resource allocation policies.

The paper is structured as follows. Next section presents other investigations related to our dynamic resource management approach. Sect. 3 presents the filter-based framework and the scheduling, resource allocation, and host management filters. The framework is evaluated by simulation in Sect. 4, for different resource allocation policies including First Fit (FF), Best Fit (BF) and their decreasing forms, and the Gaussian-type policies defined in reference [7]. Final section summarizes current work.

2. Related work

Different resource management frameworks for clusters, grids, or cloud environments have been developed in the last years. Many of them use virtualization for dynamic resource provisioning. pMapper is a framework for virtualized heterogeneous clusters, which minimizes the electrical power and the VMs migration costs with performance guarantees [27]. The framework uses different VM-to-host mapping algorithms based on variants of the FFD heuristic. Unlike pMapper, our framework has been evaluated only for best-effort VMs, and for this case, it does not offer performance guarantees. Entropy consolidates the VMs in homogeneous clusters using constraint programming [16]. Compared to our work, this framework considers both CPU and memory as constraining resources, manages not only the overloaded but also the underused hosts, and has been evaluated in real environments. The framework presented in current paper, on the other hand, provides scheduling facilities, by means of Haizea, and queuing options. The GREEN-NET framework reduces the energy consumption in large scale distributed systems by switching

off the unused physical resources, by aggregating the reservations, or by predicting next reservations based on history [6]. OpenNebula is an open-source virtual infrastructure manager for private or public IaaS clouds [20, 24]. Two extensions of this framework address issues such as energy consumption (the Green Cloud Scheduler [14]) or advance reservation (the Haizea scheduler [15]) in data centers. Haizea is a resource manager used either as a simulator or as a backend VM scheduler for OpenNebula [15, 23, 24, 25, 26]. Haizea defines different types of leases implemented as virtual machines [26], uses off-line VMs migration, and considers the VM management time overheads. Our filter-based framework extends Haizea with new VM scheduling, resource allocation, and host management policies, and computes energy and performance-related quantities for virtualized data centers. CloudSim is an event-based simulation toolkit for private and hybrid virtualized cloud systems [4, 5]. The Cloudsim's core has been extended with a power package [1, 2, 3] and an interconnection network model of a cloud data center [11]. The power package contains different energy-aware VM placement algorithms and uses power models for specific server architectures. Our modular approach to solving the resource management problem in data centers is close to the work presented in [2] using CloudSim. OpenStack is a cloud operating system providing virtualized computing resources to the users [21]. The resource allocation method presented in current paper has similarities with the OpenStack's filtering procedure. This procedure selects the eligible hosts with the largest weighted cost scores for allocating the VMs.

3. The filter-based framework

3.1. **Framework design.** The filter-based dynamic resource management framework (Fig. 1) uses software filters to perform the VM scheduling, resource allocation, and host management activities. We assume that a virtual machine has already been provisioned to each job arrived at the data center. A job is a request addressed to a batch application or a workflow to a web service. All jobs arrive at the data center simultaneously. The jobs' CPU requirements are time-varying and not known in advance. The job-to-VM mapping is one-to-one and the resource requirements of the VM and the job coincide. The virtualized jobs have been assimilated to the Haizea's best-effort leases. Henceforth, we mainly refer to the virtual machines instead of jobs.

A virtual machine is mapped on a physical machine if it receives all resources required at mapping time. The CPU capacity needed for a specified duration is the single constraining resource for the VM-to-host mapping. More VMs may simultaneously share a physical machine if their cumulated resource

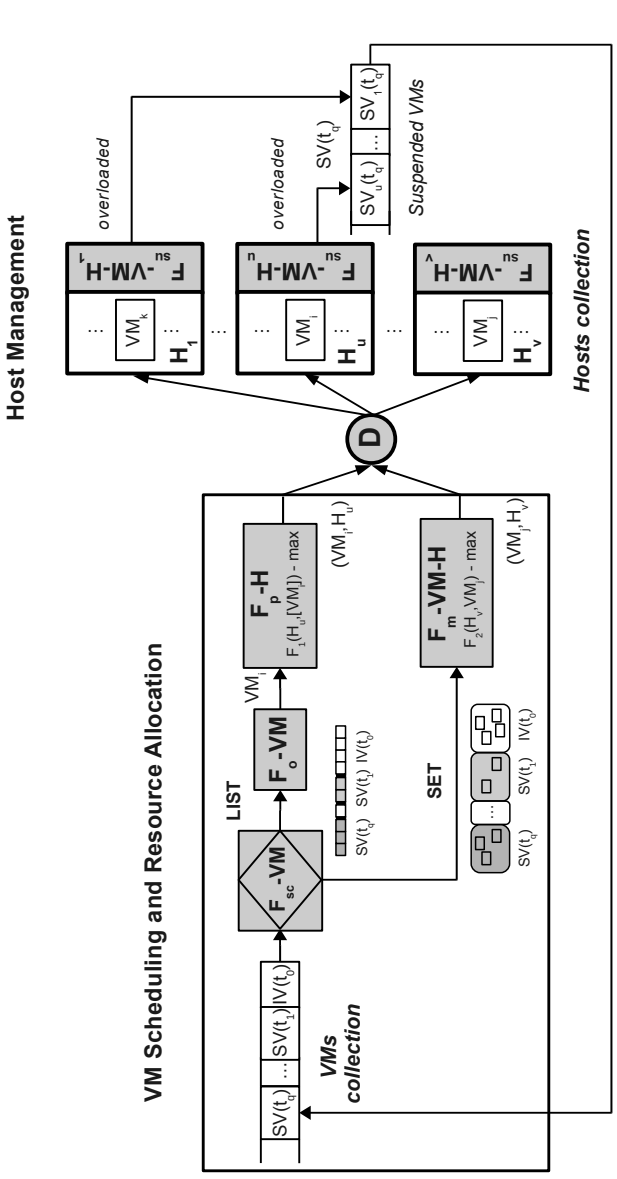

FIGURE 1. The filter-based dynamic resource management framework. Notations: $IV(t_0)$ - the subcollection of initial VMs; $SV(t_1)$, ..., $SV(t_q)$ - the subcollections of suspended VMs; H_1, H_u, H_v - physical machines (hosts); VM_k, VM_i, VM_j - virtual machines; $\mathbf{F_{sc}}$**-VM** - the VM scheduling filter, $\mathbf{F_o}$**-VM** - the VM ordering filter, $\mathbf{F_p}$**-H** - the host provisioning filter, $\mathbf{F_m}$**-VM-H** - the VM-to-host mapping filter, $\mathbf{F_{su}}$**-VM-H**$_\mathbf{u}$ - the VM suspending filter for host H_u; F_1 and F_2 - resource allocation functions; **D** - dispatcher

requirements do not exceed the available resources. At run-time, selected VMs from the overloaded hosts are suspended for rescheduling.

The framework gathers all virtual resources in a *VMs* collection, $\{VM_j\}_{j=1,\ldots,N_V}$, and the physical resources in a *Hosts* collection, $\{H_u\}_{u=1,\ldots,N_H}$ (Fig. 1). The individual VMs and hosts have unique identifiers (ID). The *VMs* collection is structured in time-labeled subcollections, which are scheduled in the increasing order of their time label. The first subcollection, $IV(t_0)$, contains the VMs provisioned to the jobs arrived at the data center at the initial time t_0. Next subcollections, $SV(t_q)$ $(q = 1, 2, \ldots)$, contain the VMs suspended at times t_q from all overloaded hosts.

The filters (Fig. 1) perform constraint or policy-based operations. The VM scheduling filter, $\mathbf{F_{sc}}$**-VM**, decides whether the VM subcollections are represented as lists or as sets. The VM ordering filter, $\mathbf{F_o}$**-VM**, optionally sorts the VM lists by a specified criterion. The difference between the VM lists and VM sets is that the VM lists are already ordered at scheduling time, while the VM sets are not. The VMs from VM lists are scheduled, if possible, in their queuing order. The VMs from VM sets, on the other hand, are mapped on hosts in an order depending on the required and the available resources. The host provisioning filter, $\mathbf{F_p}$**-H**, allocates resources to the VMs from VM lists, and the VM-to-host mapping filter, $\mathbf{F_m}$**-VM-H**, allocates resources to the VMs from VM sets. The $\mathbf{F_p}$**-H** filter selects the destination host for an already chosen VM, by maximizing a resource allocation function F_1 depending only on the host's usage. The $\mathbf{F_m}$**-VM-H** filter, on the other hand, selects the $(VM, host)$ pairs from a pool of VMs and a pool of hosts, such that to maximize a resource allocation function F_2. This two-dimensional function depends both on the VMs' required and the hosts' available resources. Finally, the VM suspending filter, $\mathbf{F_{su}}$**-VM-H**$_\mathbf{u}$, suspends VMs from the overloaded host H_u.

The framework executes the resource management activities in a repeated way. The timing for framework operation is presented in Table 1. Next sections describe the main activities of the filter-based framework.

3.2. **VMs scheduling.** At each scheduling time τ_s (Table 1), the framework tries to schedule the virtual machines present in the *VMs* collection. The scheduling policy is different for VM lists and VM sets.

3.2.1. *Scheduling of VM lists.* The VM lists are optionally sorted by the VM ordering filter $\mathbf{F_o}$**-VM** (Fig. 1). Examples of sorting the lists decreasingly by the VMs' CPU request are presented in the framework evaluation section. If the initial VM list $IV(t_0)$ is ordered, then its VMs having the same property according to the ordering criterion are sorted increasingly by their ID. If the

TABLE 1. The timing for framework operation

Time	Description
t_0	the arrival time for all jobs
$\{t_p\}_{p=1,2,...}$	the times when the VMs' CPU requirements are changed (periodic with time period T_0: $t_p = t_0 + pT_0$) and when the host management is performed
$t_d = nT_0$, $n \in \mathbb{Z}^+$	the CPU time required by each VM
$\{t_q\}_{q=1,2,...}$	the times when overloaded hosts are detected; $\{t_q\}_{q=1,2,...}$ is a subset of $\{t_p\}_{p=1,2,...}$
$\{\tau_s\}_{s=1,2,...}$	the VM scheduling times (periodic with time period $T_0/2$ or triggered by any event[1] in data center); include $\{t_p\}_{p=1,2,...}$

[1] Examples of events: VMs' suspension, migration, resumption, completion

suspended VM lists are ordered, on the other hand, then the ties are broken by sorting the VMs increasingly, first by their last host ID and then by the VMs' own ID value, in case of common host.

The framework uses two scheduling policies for VM lists, *FLIST* and *NLIST*. These policies behave differently when the next VM in the *VMs* collection can not be scheduled due to lack of physical resources. At each scheduling time τ_s, the framework using the *FLIST* policy schedules the VMs one by one, until it encounters a VM that can not be scheduled. In this case, the *FLIST* policy postpones the entire scheduling process for the next scheduling time. *FLIST* resembles the First-Come First-Served (FCFS) scheduling policy. However, for *FLIST*, the VMs from the same subcollection have a common time label and are optionally ordered, while FCFS considers the VMs in the order of their arrival time. The *FLIST* policy is useful when strict ordering is compulsory, for example when the VMs encapsulate the web services of a business process. For unrelated VMs, *FLIST* has the same drawbacks as FCFS: (a) the VMs with low resource requirements may be delayed by VMs with high requirements and (b) the VMs that need more processors may cause resource fragmentation [8, 18].

Unlike *FLIST*, when the next VM can not be scheduled, the *NLIST* policy tries to schedule the other VMs from the same subcollection as the first one. The process continues until the subcollection is completely scheduled and then is repeated for the next subcollections. The *NLIST* policy uses a list scanning procedure that resembles the List Scheduling algorithms for independent tasks, with no imposed partial ordering [13]. Nevertheless, in case of *NLIST*, the

VMs' ordering at subcollection level does not depend on time as it does for List Scheduling.

3.2.2. *Scheduling of VM sets.* At scheduling level, no ordering is imposed on VM sets. The VM-to-host mapping module presented in the next section decides the VMs' ordering in this case. As for VM lists, the VM sets are scheduled completely in the increasing order of their time label. The VM sets may model, for example, bag-of-tasks applications with independent tasks executed in parallel.

3.3. **Resource allocation.** The framework uses different policies (presented in Sect. 3.3.2) to provision physical resources to the virtual machines. For VM lists, the scheduling and ordering filters have already decided the order in which the VMs are mapped on hosts. On contrary, for VM sets, the order depends both on the VMs' requirements and the hosts' available resources, and is decided by the resource allocation filters.

3.3.1. *Resource allocation filters.* The host provisioning filter, $\mathbf{F_p}$**-H** (Fig. 1), allocates physical resources to the VMs from VM lists. This filter maximizes a resource allocation function $F_1(H_v, [VM_i])$, $v = 1, \ldots, N_H$, to find the destination host H_v for an already selected virtual machine VM_i. The VM-to-host mapping filter, $\mathbf{F_m}$**-VM-H**, on the other hand, provisions physical resources to the VMs from VM sets. For each VM set, the mapping filter chooses iteratively the $(VM, host)$ pairs that maximize a resource allocation function $F_2(H_v, VM_j)$, $v = 1, \ldots, N_H$ and $j = 1, \ldots, n$ (n is the set size). The searching process is exhaustive and finds a global solution to the optimization problem.

3.3.2. *Resource allocation policies.* This section presents the resource allocation policies currently used by the framework: (a) the reference First Fit and Best Fit policies and their decreasing forms (FFD and BFD), and (b) the Gaussian-type policies G1 and G2 defined in [7]. All policies may be used for VM lists. A variant of the BFD policy is also defined for VM sets and the G2 policy is mainly used for this type of VM subcollections.

Let 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 respectively available CPU resources ($T_{\text{CPU}} = U_{\text{CPU}} + A_{\text{CPU}}$). All hosts are identical and have a unique processor with $T_{\text{CPU}} = 100\%$ (the CPU quantities are considered in percents, which is numerically more efficient). A VM may be assigned only to a feasible host (a host with $A_{\text{CPU}} \geq R_{\text{CPU}}$) and only if it receives all required resources. If no such host exists, the VM remains in the waiting queue until a feasible host is found.

The First Fit policy assigns each VM from a list to the lowest-indexed feasible physical machine from the *Hosts* collection [12]. This policy uses an F_1-type resource allocation function, which takes the constant value 1 for any feasible host and the value 0 for the other hosts. The FFD policy uses the same resource allocation function as FF, but the VM lists are sorted decreasingly by the VMs' CPU request. The VM ordering filter, $\mathbf{F_o}$**-VM**, performs this sorting operation at scheduling level, by maximizing a function equal to R_{CPU}.

The Best Fit policy maps each VM from a list to the host with the minimal remained unused resources after allocation [12]. Since the current VM to be mapped on hosts has been chosen at scheduling time (its R_{CPU} value is known), we define the F_1-type BF resource allocation function as $1/A_{\text{CPU}}$ for the feasible hosts and 0 for the other hosts. The BFD and BF policies use the same resource allocation function for VM lists, but the lists are additionally sorted decreasingly for BFD. At its turn, the BFD policy for VM sets uses an F_2-type resource allocation function, which is $R_{\text{CPU}}/A_{\text{CPU}}$ for the feasible hosts and 0 for the other hosts. This function resembles the weight factor defined for some FFD and BFD-type heuristics proposed in the multi-dimensional Vector Bin Packing context [10, 17].

Two Gaussian-type resource allocation policies, G1 and G2, have been defined in reference [7]. These policies consolidate the VMs on physical resources in a less greedy way than FF and BF, but more tightly than load balancing methods. The G1 resource allocation policy is used by the host provisioning filter $\mathbf{F_p}$**-H** (Fig. 1) for VM lists. A given VM is assigned to the feasible host which maximizes the $G_1(U_{\text{CPU}})$ Gaussian function. This function depends only on the host usage and has adjustable location and width [7]. On contrary, the G2 policy may be used either by the host provisioning filter $\mathbf{F_p}$**-H**, for VM lists, or by the VM-to-host mapping filter $\mathbf{F_m}$**-VM-H**, for VM sets. In case of VM lists, the G2 policy finds the destination host for already selected VMs, by maximizing the F_1-type function $G_2([R_{\text{CPU}}], A_{\text{CPU}})$, with fixed R_{CPU} and variable A_{CPU}. In case of VM sets, the G2 policy selects the feasible $(VM, host)$ pairs that maximize the F_2-type two-variable function $G_2(R_{\text{CPU}}, A_{\text{CPU}})$ [7].

3.4. **Host management.** The resource requirements of the VMs deployed on physical resources are time-varying. Therefore, at run-time, some physical machines may be underused and others overloaded. VMs migrations have proved efficient for host management in both cases [2]. Currently, the framework considers only the case of overloaded hosts; the case of underused hosts remains as future work. A host is overloaded when the total CPU requirements of its VMs exceed the host's CPU resources ($U_{\text{CPU}} > T_{\text{CPU}}$). In real conditions,

however, the lower limit for hosts' overloading may be some percent from the total CPU capacity, such as $80\%T_{\text{CPU}}$.

The framework performs the host management at each time t_p (Table 1), when the VMs' CPU requirements are changed. The hosts are verified for possible overloading in increasing order of their ID. Then, the VM suspending filters (Fig. 1) select the VMs to be suspended from each identified overloaded host. For example, if the host H_u is overloaded at time t_q (Table 1), then the $\mathbf{F_{su}}$**-VM-H**$_u$ filter suspends a subset $SV_u(t_q)$ from its VMs. The VMs suspended from all overloaded hosts are collected into the $SV(t_q)$ subcollection. This subcollection is then appended to the *VMs* collection. When the suspended VMs are rescheduled, they are either resumed on the same hosts or are migrated to other hosts, depending on the result of the resource allocation process.

Haizea, the underlying scheduler of the filter-based framework, uses "cold" (off-line) VM migration. The VMs to be relocated are first suspended on the initial hosts, then migrated, and finally resumed on the new hosts. The applications encapsulated in the migrating VMs are completely stopped and restarted at the new location. The migration time is calculated as the ratio between the size of the VM's memory image and the network bandwidth.

The VM suspending filters may use different policies. In our previous work [7], we have evaluated the policy suspending the VMs with the lowest CPU requests from the overloaded hosts. This policy was combined with the FF, BF, and Gaussian-type resource allocation methods. Here we test two other policies, denoted by H and L. The H-policy suspends the VMs of an overloaded host in the decreasing order of their CPU required share. This policy reduces the number of VMs migrations, but the migrated VMs have high resource requirements at the destination hosts. Algorithms migrating VMs with high resource requirements or with high values of some metrics have been presented for example in references [2, 28, 29]. At its turn, the L-policy suspends some VMs with low CPU requirements, but not necessarily the lowest. To our knowledge, the L-policy has not been previously used in this form, but related variants exist in the literature, for example the iFFD algorithm in [27] or the HPG algorithm in [2].

The steps of the host management process at any time t_p, $p = 1, 2, \ldots$, are presented in Algorithm 1. As in reference [2], for both VM suspending policies, the algorithm tries first to suspend a single VM from each overloaded host (Algorithm 1, line 5), in order to minimize the VM migration number. Only the feasible VMs (the VMs not finishing their work in the next time period T_0) are potential candidates for suspension (line 4). The H-policy selects the VM with the highest CPU request and the L-policy the VM with

the lowest one, but greater than the overload. In each case, the ties are broken by selecting the VM with the smallest ID.

If a single VM is not able to eliminate the host overload (SV_u at line 5 in Algorithm 1 is the emptyset), then more VMs are suspended from that host. The H-policy sorts the host's VMs decreasingly by their CPU request (line 8, with $SUSP = H$) and selects as many VMs as necessary to eliminate the overload. On contrary, the L-policy sorts the VMs increasingly by their CPU request (line 8, with $SUSP = L$) and then uses two steps for VMs suspension. First, the policy selects the VMs in order until their cumulated CPU request exceeds the host's overload (lines 9–16). This means that by removing the selected VMs, A_{CPU} becomes greater than or equal to zero. Second, the list of selected VMs is scanned backwards and the VMs which are still not causing the host overload are restored (lines 17–28). The VMs suspended from each overloaded host are appended to the SV list (line 30). Finally, this list is sorted (lines 33–37) as explained in Sect. 3.2.1.

3.5. **Resource management compound filters.** A compound filter is a chain of filters able to perform all resource management operations for VMs deployment on physical resources. The compound filters evaluated in this paper are presented in Table 2. Their VM ordering, host provisioning, and VM-to-host mapping filters maximize the indicated objective functions, while the VM scheduling and VM suspending filters use the specified policies. The FF, BF, G1, and G2 compound filters for VM lists assign the VMs to the hosts by using the host provisioning filter $\mathbf{F_p}$**-H**. The FFD and BFD compound filters additionally sort the VM lists before resource allocation, with the VM ordering filter $\mathbf{F_o}$**-VM**. The BFD and G2 compound filters for VM sets allocate physical resources using the VM-to-host mapping filter $\mathbf{F_m}$**-VM-H**.

4. Framework evaluation

In this section, we present the results of evaluating the framework by simulation, for a set of batch jobs arrived simultaneously at the data center. A virtual machine was provisioned to each job. Simulation experiments consisted in processing 40 VMs with time-varying CPU requirements in two environments: one with sufficient physical resources (20 hosts) and the other with insufficient resources (8 hosts). In the (40VM,20H) case, the *NLIST* and *FLIST* scheduling policies were equivalent.

A trace of CPU requests lasting for $t_d = 500$ min was generated for each VM. The CPU requirements of the active VMs were changed periodically based on their trace, at times $t_p = t_0 + pT_0$, for $p = 1, 2, \ldots$ and $T_0 = 2$ min (Table 1). The active VMs were the ones deployed on physical machines and not waiting in queue for free resources. The VMs' CPU required shares (in percents) were

Algorithm 1 Management of overloaded hosts

Input: *Hosts* - the host collection, $\{H_u\}_{u=1,\ldots,N_H}$
VMs - the VM collection, $\{VM_j\}_{j=1,\ldots,N_V}$
ORDER - ordering option (Decreasing, None) for $\mathbf{F_o}$**-VM**
SUSP - suspending policy (L, H) for $\mathbf{F_{su}}$**-VM-H**
Output: SV - suspended VMs from all overloaded hosts

$SV \leftarrow \emptyset$
for all $H_u \in Hosts$ **do**
  if H_u.availableCPU < 0 **then**
    $AV_u \leftarrow H_u$.feasibleActiveVMs
    $SV_u \leftarrow$ choseOneVM(AV_u, $SUSP$)
    if $SV_u = \emptyset$ **then**
      sortIncreasingByVmID(AV_u)
      sortBySuspensionOption(AV_u,$SUSP$)
      for all $VM_i \in AV_u$ **do**
        if H_u.availableCPU < 0 **then**
          appendVM(SV_u, VM_i)
          H_u.availableCPU $\leftarrow H_u$.availableCPU $+ VM_i$.requiredCPU
        else
          break
        end if
      end for
      if $SUSP = L$ **then**
        $i \leftarrow SV_u$.size
        while $i \geq 1$ **do**
          $VM_i \leftarrow$ getVmByIndex(SV_u,i)
          $temp \leftarrow H_u$.availableCPU $- VM_i$.requiredCPU
          if $temp \geq 0$ **then**
            H_u.availableCPU $\leftarrow temp$
            removeVM(SV_u, VM_i)
          end if
          $i \leftarrow i - 1$
        end while
      end if
    end if
    appendVmList(SV,SV_u)
  end if
end for
sortIncreasingByVmID(SV)
sortIncreasingByHostID(SV)
if $ORDER =$ Decreasing **then**
  sortDecreasingByVmRequiredCPU(SV)
end if
return SV

TABLE 2. Resource management compound filters[1]

Compound filter \ Filter	Scheduling		Resource allocation		Host management
	F_{sc}-VM	F_o-VM	F_m-VM-H	F_p-H	F_{su}-VM-H
FF-fZ	*FLIST*	-	-	1	Z = *L* or *H*
FF-nZ	*NLIST*	-	-	1	Z = *L* or *H*
FFD-fZ	*FLIST*	R_{CPU}	-	1	Z = *L* or *H*
FFD-nZ	*NLIST*	R_{CPU}	-	1	Z = *L* or *H*
BF-fZ	*FLIST*	-	-	$1/A_{CPU}$	Z = *L* or *H*
BF-nZ	*NLIST*	-	-	$1/A_{CPU}$	Z = *L* or *H*
BFD-fZ	*FLIST*	R_{CPU}	-	$1/A_{CPU}$	Z = *L* or *H*
BFD-nZ	*NLIST*	R_{CPU}	-	$1/A_{CPU}$	Z = *L* or *H*
BFD-sZ	*SET*	-	R_{CPU}/A_{CPU}	-	Z = *L* or *H*
G1-fZ	*FLIST*	-	-	G_1	Z = *L* or *H*
G1-nZ	*NLIST*	-	-	G_1	Z = *L* or *H*
G2-fZ	*FLIST*	-	-	G_2	Z = *L* or *H*
G2-nZ	*NLIST*	-	-	G_2	Z = *L* or *H*
G2-sZ	*SET*	-	G_2	-	Z = *L* or *H*

[1] FF - First Fit, FFD - First Fit Decreasing, BF - Best Fit, BFD - Best Fit Decreasing, G1 and G2 - Gaussian-type filters (the G1 filter uses the G_1 resource allocation function and G2 uses G_2); F_{sc}-VM - the VM scheduling filter (uses the *FLIST*, *NLIST*, or *SET* policy), F_o-VM - the VM ordering filter (optionally orders the VMs decreasingly by their R_{CPU} value), F_m-VM-H - the VM-to-host mapping filter (uses an F_2-type resource allocation function), F_p-H - the host provisioning filter (uses an F_1-type resource allocation function), F_{su}-VM-H - the VM suspending filter (uses the *L* or *H* policy); R_{CPU} - the VM CPU request, A_{CPU} - the host available CPU.

random numbers generated uniformly between 10 and 40 and then rounded up to the closest integer value. This range of values ensured some variation among the VMs' CPU traces and favored the hosts' overloading, of interest for our study. In the migration process, the VM memory image was off-line migrated, with no disk image transfer. The VMs were suspended by saving their memory state on the filesystem at a rate of 32 MB/s [25, 15]. The VMs' resumptions were performed at the same rate. The suspension and resumption needed 4 s each and the copy of the VM memory image other 11 s. In simulations, the same total delay of 19 s was considered for all suspended VMs, either migrated or later resumed on the same hosts. The Haizea's restrictions have been relaxed, more VMs being allowed to migrate simultaneously from or to the same physical machine, with no performance overhead. Simulation experiments have been repeated 100 times. All compound filters used identical environment conditions and CPU traces within the same experiment.

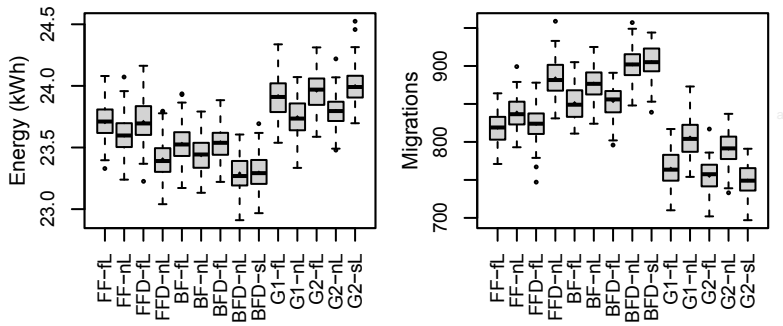

FIGURE 2. Boxplot representation of the consumed energy and the VM migration number for the (40VM,8H) configuration, *L*-suspending policy, and *FLIST*, *NLIST* and *SET* scheduling policies, in 100 simulation experiments.

The resource management policies presented in this paper were compared using the following metrics: the energy consumed by the physical machines, the VMs' total flow time (the sum of all VMs' processing times), the number of VMs migrations, and the mean number of active hosts for the entire makespan. The VMs migrations and the VMs suspensions with resumption on the same hosts were counted independently. We considered that the total electrical power of each physical machine depended linearly on its CPU usage [9]. Moreover, the idle power represented 70% of the total power of 250 W at full CPU utilization [1]. We assumed that the hosts were switched off when they became idle. In all experiments, the G_1 resource allocation function used the parameters $Thr_{\mathrm{L}} = 40$, $Thr_{\mathrm{H}} = 80$, and $a = 0.8$, as defined in reference [7], and the G_2 function used the parameters $\alpha = 0.5$ and $r = 0.001$.

Simulation results are presented in Figures 2 and 3. The boxplots contain boxes from the first to the third quartile of data, whiskers extending to the most extreme data point, but not further than 1.5 times the interquartile range [19], the data's median value (the horizontal line), the mean value (the full knot, possibly overlapped by the median's line), and outliers (the open knots). Based on these results, we made the following observations:

(a) *VM scheduling policy.* In the (40VM,8H) configuration, for the same type of compound filter, the average energy consumption was slightly smaller for the *NLIST* scheduling policy than for the *FLIST* policy, but with a slightly higher average VM migration number (Fig. 2). The *SET* scheduling policy was closer in outcome either to *NLIST* (for BFD) or to *FLIST* (for G2). For example, the median energy consumption and VM migration number were

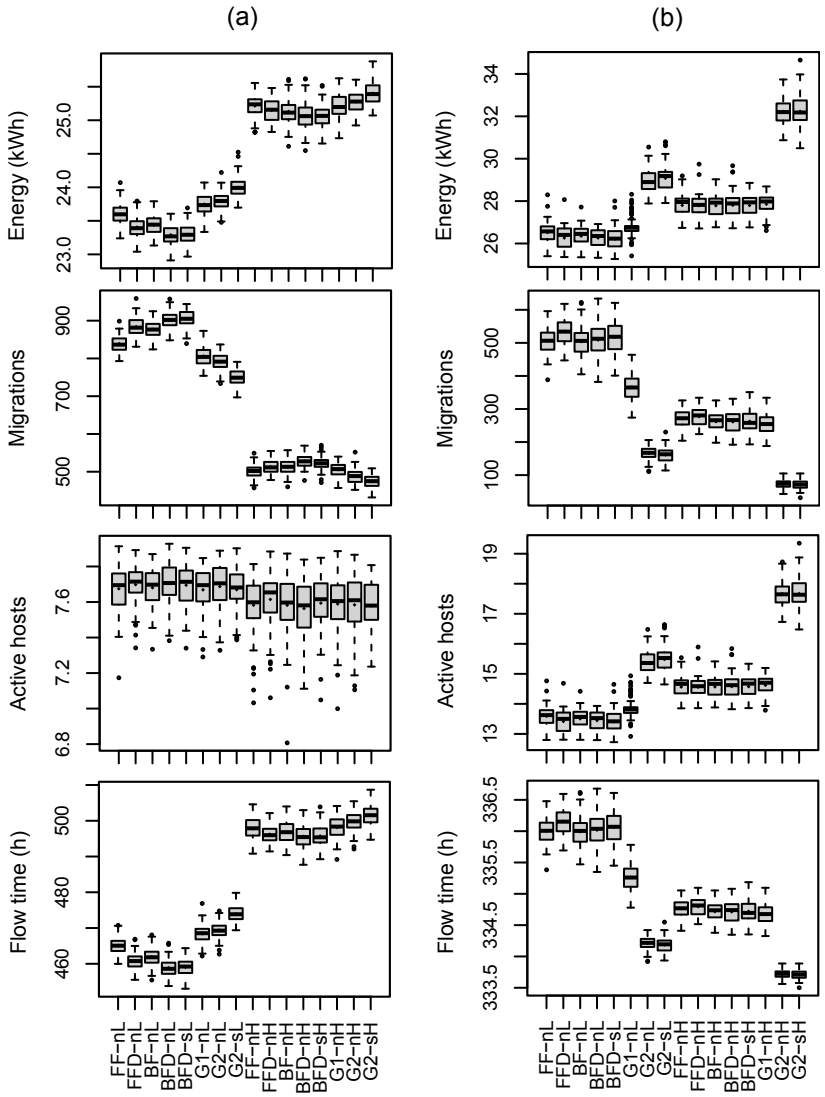

FIGURE 3. Boxplot representation of the energy and performance metrics for the (a) (40VM,8H) and (b) (40VM,20H) configurations, L and H suspending policies, and *NLIST* and *SET* scheduling policies, in 100 simulation experiments.

23.54 kWh and 856 for BFD-fL, 23.27 kWh and 902 for BFD-nL, and 23.29 kWh and 905 for BFD-sL.

(b) *Metrics.* The average VM migration number and the average energy consumption had opposite behavior. A more energy efficient compound filter was less efficient regarding the number of VMs migrations. Moreover, the relative average behavior of the compound filters was similar for the consumed energy and flow time in the (40VM,8H) configuration (Fig. 3a), but similar for the VM migration number and flow time in the (40VM,20H) configuration (Fig. 3b).

(c) *Compound filters.* The Gaussian-type compound filters were less energy-efficient than the FF, FFD, BF, and BFD compound filters, but generated a lower number of VMs migrations (Fig. 3). For example, in the (40VM,20H) configuration, the median energy consumption and VM migration number were 26.35 kWh and 513 for BFD-nL, 26.74 kWh and 365 for G1-nL, and 28.90 kWh and 168 for G2-nL.

(d) *Suspended VMs.* In the (40VM,8H) configuration, 11–14% from the total number of VMs suspensions were resumed later on the same hosts, without relocation. However, the compound filters' relative average behavior was not much affected qualitatively when considering all suspended VMs instead of only migrated ones.

(e) *VM suspending policy.* In both configurations and for all compound filters, the H-suspending policy caused a higher energy consumption and fewer VMs migrations than the L-suspending policy (Fig. 3). In the (40VM,20H) configuration, for instance, the median energy consumption and VM migration number were 26.35 kWh and 513 for BFD-nL, but 27.89 kWh and 267 for BFD-nH.

5. Conclusions

This paper has presented a filter-based dynamic resource management framework for virtualized environments. The framework uses resource management filters to perform the VM scheduling, resource allocation, and host management operations in data centers. The framework may be used to assess the energy and performance efficiency of different resource management techniques. The framework has been evaluated by simulation, for deploying virtual machines with time-varying CPU requirements, in small size environments with sufficient and insufficient physical resources. Since the resource management filters may be combined in the desired way, the framework may be included in autonomous systems and may be used in what-if analysis.

Acknowledgments

The authors would like to thank the anonymous reviewers for their suggestions and comments, which improved the paper.

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, 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.

[3] 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.

[4] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, R. Buyya, *CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms*, Software: Practice and Experience, 41 (2011), pp. 23-50.

[5] CloudSim. `http://www.cloudbus.org/cloudsim/`

[6] G. Da Costa, J.-P. Gelas, Y. Georgiou, L. Lefevre, A.-C. Orgerie, J.-M. Pierson, O. Richard, K. Sharma, *The GREEN-NET framework: Energy efficiency in large scale distributed systems*, in Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Processing (IPDPS'09), 2009, pp. 1-8.

[7] C. Crăciun, I. Salomie, *Gaussian-type resource allocation policies for virtualized data centers*, Studia Univ. Babeş-Bolyai, Informatica, LXI(2) (2016), pp. 94-109.

[8] L. Eyraud-Dubois, G. Mounié, D. Trystram, *Analysis of scheduling algorithms with reservations*, in Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS'07), 2007, pp. 1-8.

[9] 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.

[10] M. Gabay, S. Zaourar, *Variable size vector bin packing heuristics - Application to the machine reassignment problem*, Inria, TechReport hal-00868016 (OSP. 2013). Available online: `http://hal.archives-ouvertes.fr/hal-00868016`.

[11] S. K. Garg, R. Buyya, *NetworkCloudSim: Modelling parallel applications in cloud simulations*, in Proceedings of the 2011 4th IEEE International Conference on Utility and Cloud Computing (UCC'11), 2011, pp. 105-113.

[12] M. R. Garey, R. L. Graham, J. D. Ullman, *An analysis of some packing algorithms*, R. Rustin, ed., Combinatorial Algorithms, Algorithmics Press, New York, 1973, pp. 39-47.

[13] R. L. Graham, *Bounds for certain multiprocessing anomalies*, Bell System Technical Journal, 45 (1966), pp. 1563-1581.

[14] Green Cloud Scheduler. `http://coned.utcluj.ro/GreenCloudScheduler/`

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

[16] 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.

[17] S. Lee, R. Panigrahy, V. Prabhakaran, V. Ramasubramanian, K. Talwar, L. Uyeda, U. Wieder, *Validating heuristics for virtual machines consolidation*, Microsoft Research, TechReport MSR-TR-2011-9, Jan 2011. Available online: `http://research.microsoft.com/pubs/144571/virtualization.pdf`

[18] A. W. Mu'alem, D. G. Feitelson, *Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling*, IEEE Trans. Parallel Distrib. Syst., 12 (2001), pp. 529-543.
[19] The R Project for Statistical Computing. `http://www.r-project.org/`
[20] OpenNebula. `http://www.opennebula.org/`
[21] OpenStack. `http://www.openstack.org/`
[22] I. Salomie, T. Cioara, I. Anghel, D. Moldovan, G. Copil, P. Plebani, *An energy aware context model for green IT service centers*, Service-Oriented Computing. Lecture Notes in Computer Science 6568, Springer, Berlin, 2011, pp. 169-180.
[23] B. Sotomayor, K. Keahey, I. Foster, *Combining batch execution and leasing using virtual machines*, in Proceedings of the 17th International Symposium on High performance distributed computing (HPDC'08), 2008, pp. 87-96.
[24] 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.
[25] B. Sotomayor, R. S. Montero, I. M. Llorente, I. Foster, *Resource leasing and the art of suspending virtual machines*, in Proceedings of the 11th IEEE International Conference on High Performance Computing and Communications (HPCC-09), 2009, pp. 59-68.
[26] B. Sotomayor Basilio, *Provisioning computational resources using virtual machines and leases*, PhD Dissertation, Univ. of Chicago, Illinois, USA, 2010.
[27] 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.
[28] T. Wood, P. Shenoy, A. Venkataramani, M. Yousif, *Sandpiper: Black-box and gray-box resource management for virtual machines*, Comput. Netw., 53 (2009), pp. 2923-2938.
[29] H. Zhang, K. Yoshihira, Y.-Y. Su, G. Jiang, M. Chen, X. Wang, *iPOEM: A GPS tool for integrated management in virtualized data centers*, in Proceedings of the 8th IEEE/ACM International Conference on Autonomic Computing (ICAC'11), 2011, pp. 41-50.

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`