Collaborative Agents for Distributed Load Management in Cloud Data Centers using Live Migration of Virtual Machines

J. Octavio Gutierrez-Garcia and Adrian Ramirez-Nafarrate

Abstract— Load management in Cloud data centers must take into account 1) hardware diversity of hosts, 2) heterogeneous user requirements, 3) volatile resource usage profiles of virtual machines (VMs), 4) fluctuating load patterns, and 5) energy consumption. This work proposes distributed problem solving techniques for load management in data centers supported by VM live migration. Collaborative agents are endowed with a load balancing protocol and an energy-aware consolidation protocol to balance and consolidate heterogeneous loads in a distributed manner while reducing energy consumption costs. Agents are provided with 1) policies for deciding when to migrate VMs, 2) a set of heuristics for selecting the VMs to be migrated, 3) a set of host selection heuristics for determining where to migrate VMs, and 4) policies for determining when to turn off/on hosts. This paper also proposes a novel load balancing heuristic that migrates the VMs causing the largest resource usage imbalance from overloaded hosts to underutilized hosts whose resource usage imbalances are reduced the most by hosting the VMs. Empirical results show that agents adopting the distributed problem solving techniques are efficient and effective in balancing data centers, consolidating heterogeneous loads, and carrying out energy-aware server consolidation.

Index Terms— Cloud computing, Distributed systems, Intelligent agents, Multiagent systems

1 Introduction

A Cloud data center consists of a network of heterogeneous commodity servers [24] providing virtualized computing services. Users access resources of data centers by allocating virtual machines (VMs) to hosts. A VM allocation request is composed of a set of functional requirements, e.g., custom software installed, and a set of non-functional requirements, namely, number of virtual cores and size of memory to be allocated. Both data centers and user requirements are commonly heterogeneous [22]. In addition, the resource usage profiles of VMs and the overall load pattern of data centers vary significantly over time [37]. Furthermore, data centers should increase hosts' utilization ratio and reduce the number of idle hosts in order to reduce energy consumption and operating costs [31]. Moreover, a VM suffers from resource usage imbalance when the utilization of the resources allocated to the VM is uneven. For instance, by making intensive use of CPU, but making little use of memory (as the applications for image texture sampling presented in [17]), or vice versa. Aggregate resource usage imbalance of VMs results in an overall resource usage imbalance of hosts, and as a consequence, in poor utilization of resources [20], violations of service level agreements [4], and waste of energy [31]. In addition, some hosts of a data center may be more suitable to host a given VM with a certain resource usage profile due to their underlying hardware architecture and available computing resources [14]. Due to the above reasons, a load management system of a data center must take into account 1) the heterogeneity of user requirements, 2) the hardware diversity of hosts, 3) the volatility of resource usage profiles of VMs, 4) the fluctuations of load patterns, and 5) energy consumption. However, the inherent heterogeneity of data centers and the dynamicity of their load are often disregarded, see [1], [4], [36]. In addition, centralized load balancing mechanisms where a single entity is in charge of balancing loads across hosts have been proposed, see [10], [23]. In a similar manner, energy-aware server consolidation for data centers is commonly tackled using centralized approaches, see [18], [29], [32]. Nevertheless, centralized approaches are not scalable because a central node has to monitor multiple distributed hosts, becoming a bottleneck (as well as a single point of failure) when the data center is under stress.

VM live migration has been used to enable dynamic load management in data centers, see [11], [33], [38]. VM live migration allows migrating VMs from a host to another host with almost imperceptible downtime [34].

This paper contributes distributed problem solving techniques for load management in data centers supported by VM live migration. Collaborative agents are endowed with a load balancing protocol and an energy-aware consolidation protocol to balance and consolidate heterogeneous loads (e.g., migrating memory-intensive loads to memory-intensive hosts) in a distributed manner while reducing energy consumption costs. When a host is overloaded, agents collaborate with each other to sample resource usage of hosts and determine the best destination host for a VM according to server-centric load management policies. In addition, hosts are managed by serv-

- *J.O. Gutierrez-Garcia is with the Department of Computer Science, Instituto Tecnológico Autónomo de México, Mexico City, D.F. 01080, Mexico. E-mail: octavio.gutierrez@itam.mx.*
- *A. Ramirez-Nafarrate is with the Department of Industrial & Operations Engineering, Instituto Tecnológico Autónomo de México, Mexico City, D.F. 01080, Mexico. E-mail: adrian.ramirez@itam.mx.*

xxxx-xxxx/0x/$xx.00 © 200x IEEE Published by the IEEE Computer Society

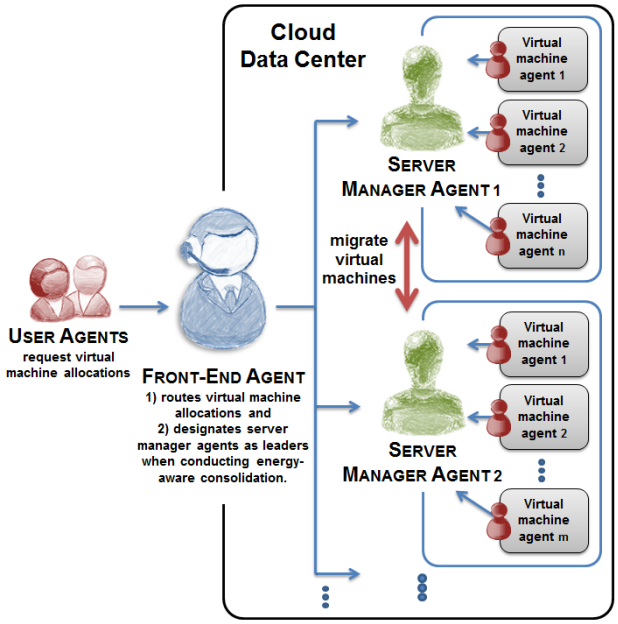

Fig. 1. Agent-based Cloud data center architecture.

er manager agents, which collaborate among each other to consolidate VMs (deployed in potentially underutilized hosts) into fewer hosts. A server manager agent deployed in an underutilized host interacts with a front-end agent in order to be designated as a leader, meaning the server manager agent can start migrating VMs to other hosts and autonomously turning itself off afterward. Additionally, agents deployed in front-end servers autonomously turn on hosts when VM allocation requests cannot be fulfilled due to insufficient resources. Moreover, agents are also provided with 1) policies for deciding when to migrate VMs, 2) a set of heuristics for selecting the VMs to be migrated, 3) a set of host selection heuristics for determining where to migrate VMs, and 4) policies for determining when to turn off/on hosts.

The structure of the paper is as follows. Section 2 presents the agent-based architecture. Section 3 describes the distributed problem solving techniques. Section 4 provides empirical evidence on the effectiveness and efficiency of agents in balancing data centers, consolidating heterogeneous loads, and carrying out energy-aware server consolidation. Section 5 includes a comparison with related work, and Section 6 presents some concluding remarks and future research directions.

2 Agent-based Architecture for Distributed Load Management

The agent-based architecture for distributed load management (Fig. 1) consists of user agents, a front-end agent, server manager agents, and virtual machine agents.

User agents (UAs) submit VM allocation requests to a front-end agent. A VM allocation request consists of number of virtual cores and size of memory (in GBs) to be allocated for a VM.

The front-end agent (FA) is in charge of requesting VM allocations to server manager agents in response to VM allocation requests from UAs. The FA receives heartbeat messages from server manager agents containing information about their available computing resources to determine whether a server can host a given VM. A heartbeat message contains the number of available cores, the amount of available memory, and the percentage of CPU and memory usage of a given host. The FA is endowed with a best-fit heuristic and a random load balancing heuristic for the initial allocation of VMs to one of the hosts with sufficient computing resources. The best-fit heuristic consists of allocating VMs to the hosts with the minimum number of available cores, see [30] for an example of a best-fit heuristic. The random heuristic consists of randomly allocating VMs to hosts. If the selected server is unable to host the VM, the procedure is repeated until a selected server is able to host it. Once the FA has selected an initial destination host, the FA adopts a *request* interaction protocol [13] to request the destination host to reserve memory and virtual cores for a VM. The *request* interaction protocol is used for synchronization purposes because the destination server manager agent may be concurrently interacting with other server manager agents to host another VM. In addition, when carrying out energy-aware server consolidation, the FA is in charge of designating server manager agents as leaders when they are about to be turned off in order to start migrating VMs (see Section 3.5 for details).

It is acknowledged that other load balancing heuristics can potentially be programmed to distribute VM allocation requests across hosts. However, this work focuses on distributed load management and thus centralized load management techniques are out of the scope.

Server manager agents (SMAs) are in charge of balancing the load across hosts and carrying out energy-aware server consolidation. In addition, SMAs are also in charge of consolidating heterogeneous loads (e.g., migrating memory-intensive loads to memory-intensive hosts) by using a set of load management policies. When carrying out energy-aware server consolidation, SMAs interact with the FA to be designated as leaders in order to be able to migrate VMs to other hosts and eventually turned themselves off (see Section 3.5 for details). Other functions of SMAs are:

- Receiving VM allocation requests from the FA.
- Monitoring host resource usages, namely, CPU and memory usage, which are sent to the FA. The monitoring information is a snapshot of the host's resource usage at a given time based on an administrator-defined monitoring frequency. The resource usage of a host is determined by the aggregate resource usage of its VMs. The resource usage of a VM is a function of its allocated resources, e.g., number of virtual cores, and the current usage (e.g., the overall percentage of CPU usage).
- Replying to *call-for-proposals* messages sent by overloaded or underutilized SMAs (who are attempting to migrate VMs) when carrying out collaborative load balancing and energy-aware server consolidation respectively.
- Allocating memory and virtual cores to host VMs.
- Triggering VM migrations when migration thresholds

are exceeded.
- Determining the VMs to be migrated when the host is overloaded or is about to be turned off.
- Determining the destination hosts for VMs.

Virtual machine agents (VMAs) are in charge of monitoring resource usages of VMs and sending the information (enclosed in heartbeat messages) to SMAs. The monitoring information is a snapshot of a VM's resource usage at a given time based on an administrator-defined monitoring frequency.

It should be noted that FAs, SMAs, and VMAs are designed to be deployed in front-end servers, in hosts at the hypervisor level and in VMs respectively.

3 Distributed Problem Solving Techniques

In order to collaboratively manage the load of a data center, SMAs are provided with load management policies, VM migration heuristics, host selection heuristics, the collaborative load balancing protocol, and the energy-aware consolidation protocol.

3.1 Load Management Policies

To take into account server heterogeneity, server-centric load management policies are defined for each host.

To perform load management, SMAs are endowed with *load-related migration thresholds* and a *load-related activation threshold*. The *load-related migration thresholds* define the maximum level of CPU and memory usage of a host at which its performance declines. The *load-related activation threshold* is the number of times the resource usage has to exceed the thresholds of a host (within a time window) before triggering a VM migration. The time window is variable because it depends on a counter, if the resource usage thresholds are exceeded, the counter is incremented by one, otherwise, the counter is decreased by one until zero is reached. In doing so, it is guaranteed that the resource usage thresholds have to be exceeded several times within a time window. It is worth mentioning that hosts have independent load-related migration thresholds for memory and CPU usage to take into account server heterogeneity. For instance, some hosts may be more suitable to handle memory-intensive VMs; consequently, their memory migration thresholds may be set relatively high.

To perform energy-aware server consolidation, SMAs are endowed with an *energy-related migration threshold* and an *energy-related activation threshold*. The *energy-related migration threshold* is the minimum power consumption allowed for a host, which is a function of the percentage of CPU usage. This is because there is a quasi-linear relationship between power consumption and CPU usage [6], [12]. The *energy-related activation threshold* is the number of times the power consumption of a host must fall below the energy-related migration threshold to start migrating VMs to other hosts.

It is worth mentioning that low *activation thresholds* may trigger more VM migrations than needed, whereas high *activation thresholds* may delay VM migrations unnecessarily causing performance deterioration. In addition, it should be noted that the values for the thresholds should be defined by a data center administrator based on the characteristics of hosts as in the Red Hat enterprise virtualization suite [10]. Automatic threshold adjustment requires host profiling, which is out of the scope of this paper.

3.2 VM Migration Heuristics

The *VM migration heuristics* for selecting the VMs to be migrated are as follows.

1) The *CPU-based migration heuristics* select the VMs to be migrated based only on their CPU usage. SMAs can be configured to select the VM with either the highest CPU usage (H_{CPU}) or the lowest CPU usage (L_{CPU}). On the one hand, by migrating the VM with the highest CPU usage, an overloaded host may quickly reduce its load. However, the destination host may be soon overloaded by hosting a VM with substantial computing requirements. On the other hand, by migrating the VM with the lowest CPU usage, an overloaded host may require more VM migrations to reduce its load. However, more potential destination hosts can be found and the VM may not be migrated again due to its low computing requirements.

2) The *memory-based migration heuristics* select the VMs to be migrated based only on their memory usage. SMAs can be configured to select the VM with either the highest memory usage (H_{MEM}) or the lowest memory usage (L_{MEM}). The rationale behind the memory-based migration heuristics is similar to the rationale of the CPU-based migration heuristics.

3) The *combined migration heuristics* select the VMs to be migrated based on both memory and CPU usage. There are four combined migration heuristics: H_{CPU}-H_{MEM}, H_{CPU}-L_{MEM}, L_{CPU}-H_{MEM}, and L_{CPU}-L_{MEM}.

The H_{CPU}-L_{MEM} migration heuristic selects the VM with the highest CPU usage and the lowest memory usage. This is determined by sorting the VMs hosted in a host in *ascending* order according to their CPU usage. Then, a *CPU* weight of k is assigned to the VM in the kth position of the sorted list. In doing so, the VM with the highest CPU usage has the heaviest weight. Afterward, the VMs are sorted in *descending* order according to their memory usage and a *memory* weight of k is assigned to the VM in the kth position of the sorted list. In doing so, the VM with the lowest memory usage has the heaviest weight. Finally, the VM with the highest sum of *CPU* weight and *memory* weight is selected for migration.

The H_{CPU}-H_{MEM}, L_{CPU}-H_{MEM}, and L_{CPU}-L_{MEM} migration heuristics are implemented similarly to the H_{CPU}-L_{MEM} migration heuristic.

4) The *imbalance-based migration heuristic* selects the VMs that cause the largest resource usage imbalance to hosts. The resource usage imbalance of a host is the absolute difference between its CPU usage and its memory usage. Then, to calculate the resource usage imbalance caused by a VM, see (3), the absolute difference between the CPU and memory usage contributions of the VM to the overall resource usage of its host is determined. Equations (1) and (2) calculate the CPU and memory usage contributions of a VM to the overall CPU and memory usage of its host respectively.

IF the initiator SMA is adopting a CPU-based migration heuristic **THEN**
  Select the participant SMA with the lowest CPU usage
ELSE IF the initiator SMA is adopting a *memory-based migration heuristic* **THEN**
  Select the participant SMA with the lowest memory usage
ELSE IF the initiator SMA is adopting a *combined migration heuristic* **THEN**
  Sort participant SMAs by CPU usage in descending order
  FOR EACH participant SMA **DO**
    $SMA_i.weight_{CPU} = weight_{CPU}$
    $weight_{CPU} = weight_{CPU} + 1$
  Sort participant SMAs by memory usage in descending order
  FOR EACH participant SMA **DO**
    $SMA_i.weight_{MEM} = weight_{MEM}$
    $weight_{MEM} = weight_{MEM} + 1$
    $SMA_i.weight = SMA_i.weight_{CPU} + SMAi.weight_{MEM}$
  Select the participant SMA with the highest total weight

Fig. 2. Greedy selection of destination hosts.

$$VM_{CPU} = Host.CPU_{usage} - \frac{VM.vCores * (VM.CPU_{usage})}{Host.vCores} \quad (1)$$

$$VM_{MEM} = Host.Memory_{usage} - \frac{VM.Memory * (VM.Memory_{usage})}{Host.Memory} \quad (2)$$

$$VM.\text{ResourceUsageImbalance} = \left| VM_{CPU} - VM_{MEM} \right| \quad (3)$$

Where *Host.vCores* stands for overall number of virtual cores of the host; *Host.Memory* stands for overall amount of memory of the host; *VM.vCores* stands for number of virtual cores allocated to the VM; *VM.Memory* stands for amount of memory allocated to the VM; $Host.CPU_{usage}$ and $Host.Memory_{usage}$ stand for overall CPU and memory usage of the host respectively; and $VM.CPU_{usage}$ and $VM.Memory_{usage}$ stand for CPU and memory usage of the VM respectively.

The rationale behind the imbalance-based migration heuristic is that by migrating VMs causing the largest resource usage imbalance from hosts, the hosts will eventually have similar CPU and memory usage.

In addition to the above VM migration heuristics, SMAs are also capable of selecting VMs for migration at random.

3.3 Host Selection Heuristics

The host selection heuristics used by SMAs are: greedy selection and imbalance-based selection.

The *greedy selection of destination hosts* is based on rankings of CPU and memory usage of SMAs playing the *participant* role of the CProtocol (see Section 3.4). If the initiator SMA adopts a CPU-based migration heuristic, the initiator SMA selects the participant SMA with the lowest CPU usage as the destination host. In a similar manner, if the initiator SMA adopts a memory-based migration heuristic, the initiator SMA selects the participant SMA with the lowest memory usage as the destination host. If the initiator SMA adopts a combined migration heuristic, the initiator SMA selects the participant SMA with a combination of the lowest CPU usage and the lowest memory usage. See Fig. 2 for a detailed definition of the greedy selection of destination hosts.

The *imbalance-based selection of destination hosts* selects the host of the participant SMA whose resource usage imbalance is reduced the most by hosting the VM to be migrated by the initiator SMA. To calculate the resource usage imbalance of a host before hosting a VM, (see (6)), the absolute difference between the CPU and memory usage contributions of the VM to the overall resource usage of the host is determined. Equations (4) and (5) calculate the CPU and memory usage contributions of a VM to the overall CPU and memory usage of a host respectively.

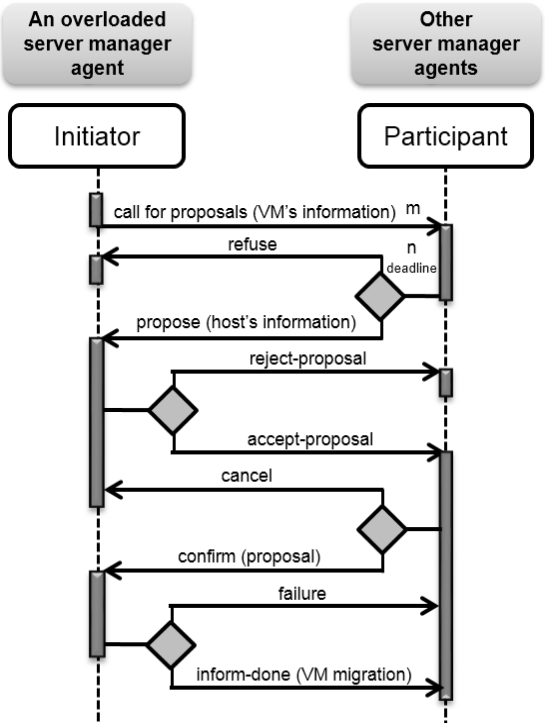

Fig. 3. Collaborative load management protocol (CProtocol).

$$Host_{CPU} = Host.CPU_{usage} + \frac{VM.vCores * (VM.CPU_{usage})}{Host.vCores} \quad (4)$$

$$Host_{MEM} = Host.Memory_{usage} + \frac{VM.Memory * (VM.Memory_{usage})}{Host.Memory} \quad (5)$$

$$Host.\text{ResourceUsageImbalance} = \left| Host_{CPU} - Host_{MEM} \right| \quad (6)$$

The imbalance-based migration heuristic can be combined with the imbalance-based selection of destination hosts. The rationale behind this combination is that by migrating VMs causing the largest resource usage imbalance from overloaded hosts to underutilized hosts whose resource usage imbalances are reduced the most, the load of the data center will eventually be balanced.

3.4 Collaborative Load Management Protocol

SMAs are endowed with a collaborative load management protocol (CProtocol) in order to select destination hosts for VMs in a distributed manner (Fig. 3). Agents, in this work, make use of the CProtocol to 1) balance the load of a data center and to 2) consolidate heterogeneous loads, e.g., mi-

grating memory-intensive VMs to memory-intensive hosts. Determining whether the CProtocol is used to consolidate or balance loads depends on the selected load management policies, VM migration heuristics, and host selection heuristics. See Sections 4.1 and 4.2 for examples of load balancing and load consolidation respectively.

The CProtocol (Fig. 3) has two roles: initiator and participant. Whereas the *initiator* role is played by an overloaded SMA, the *participant* role is played by *m* potentially underutilized SMAs. The CPU and memory migration thresholds of an SMA determine whether the SMA is overloaded or underutilized.

When the *load-related activation threshold* of an SMA is exceeded, the overloaded SMA adopts the *initiator* role and sends a *call-for-proposals* message (containing the computing requirements and resource usages of a VM to be migrated) to *m* SMAs playing the *participant* role. From the *m* participant SMAs, *n* SMAs with sufficient computing resources to host the VM reply with a *propose* message containing information about their hosts, e.g., available virtual cores and current CPU usage. The other *m-n* participant SMAs reply with a *refuse* message either because they do not have sufficient resources to host the VM or because their resource usage thresholds would be exceeded by hosting the VM. Afterward, from the proposals received, the initiator SMA selects a destination host based on a host selection heuristic and sends an *accept-proposal* message to the selected participant SMA and *n-1 reject-proposal* messages to the remaining participant SMAs. Then, the selected participant SMA may reply with either a *confirm* message indicating that sufficient resources have been allocated for the VM or a *cancel* message indicating that it has no longer sufficient resources to host the VM due to a sudden increase in CPU or memory usage. If the initiator SMA receives a *confirm* message, the initiator SMA migrates the VM to the host of the selected SMA and sends an *inform-done* message to report success; otherwise, the initiator SMA sends a *failure* message.

An example scenario of distributed load management involving four SMAs adopting the CProtocol is as follows. SMA_1, SMA_2, and SMA_3 are potentially underutilized agents playing the *participant* role. SMA_0 is an overloaded agent (playing the *initiator* role) using the $H_{CPU}|Greedy$ load balancing heuristic, i.e., SMA_0 is configured to select the VM with the highest CPU usage for migration to the host of the participant SMA with the lowest CPU usage. In order to reduce its load, SMA_0 sends a *call-for-proposals* message to SMA_1, SMA_2, and SMA_3 containing information about the VM, namely, its number of virtual cores and size of memory allocated as well as its most recent CPU and memory usages. SMA_3 does not have sufficient resources to allocate the VM and thus it sends a *refuse* message to SMA_0. SMA_1 and SMA_2 have sufficient resources to allocate the VM and thus send a *propose* message to SMA_0 containing information about their resource usages. The proposal of SMA_1 indicates a CPU usage of 50% and a memory usage of 60%. The proposal of SMA_2 indicates a CPU usage of 20% and a memory usage of 10%. Given that SMA_0 is adopting both a CPU-based migration heuristic and the greedy selection of destination hosts, it selects the proposal of the SMA with the lowest CPU usage, i.e., the proposal of SMA_2. Then, SMA_0 sends a *reject-proposal* message to SMA_1 and an *accept-proposal* message to SMA_2, which replies with a *confirm* message. In response, SMA_0 migrates the VM to the host of SMA_2 and sends an *inform-done* message to report success.

It should be noted that SMAs are capable of adopting *n* concurrent instances of the CProtocol. Each underlying agent conversation of the CProtocol is identified by means of unique conversation identifiers, which are used as message filters in order to keep a context for each agent conversation. In addition, concurrent resource allocation is protected by synchronized critical sections to prevent the overcommitment of resources.

3.5 Energy-Aware Server Consolidation Protocol

SMAs are endowed with an energy-aware consolidation protocol (EProtocol) to consolidate VMs deployed in potentially underutilized hosts into fewer hosts. The EProtocol involves the FA and SMAs and consists of four phases:

1) *Leader selection*. When both the *energy-related migration threshold* and the *energy-related activation threshold* of an SMA are exceeded simultaneously, the SMA attempts to promote itself as the leader by using a *request* interaction protocol [13] with the FA. By being a leader, the SMA is enabled to migrate its VMs to other hosts. By selecting a leader, scenarios where two or more SMAs (about to be turned off) continuously migrate VMs among each other are prevented. If the energy-related thresholds of other SMAs are exceeded while there exists a leader SMA, the other SMAs periodically attempt to promote themselves as leaders.

2) *Consecutive VM migrations*. The leader SMA migrates VMs one by one to other SMAs by adopting the *initiator* role of the CProtocol. The leader SMA uses the H_{CPU} migration heuristic (see Section 3.2). This is because it is assumed that the VM with the highest CPU usage is the most difficult to accommodate in terms of capacity, and as a consequence, those VMs should be the first to be migrated. In addition, the leader SMA uses the *greedy selection of destination hosts* (see Section 3.3). By allocating the VMs to the SMA with the lowest CPU usage, the selected SMAs are prevented from being turned off, and as a consequence, the number of VM migrations is reduced by delaying further server consolidation.

3) *Turning off servers*. Once the leader SMA has no VMs, it sends a message to the FA to indicate that it is no longer the leader, and it autonomously turns itself off (i.e., it goes into *sleep* mode) afterward.

4) *Turning on servers*. When a UA's VM allocation request cannot be fulfilled by the FA due to insufficient resources, the FA verifies whether there is a host in *sleep* mode. If it is the case, the FA turns on the host selected at random and allocates the VM to the recently turned on server. If there is no host in *sleep* mode, the FA rejects the VM allocation request.

It is acknowledged that in order to consolidate hosts, SMAs require a model of power consumption of the hosts where they are deployed. In this work, the power consumption of a host is modeled as follows:

$$P(Host.CPU_{usage}) = \begin{cases} \alpha, & \text{if the host is in } sleep \text{ mode} \\ \beta + \gamma Host.CPU_{usage}, & \text{if the host is } on \end{cases} \quad (7)$$

TABLE 1
LOAD-RELATED INPUT DATA FOR EXPERIMENTS 4.1 AND 4.2

Input parameter	Possible values
Avg. inter-arrival time of VMs	1000 ms
Avg. inter-departure time of VMs	35000 ms
Monitoring frequency of VMs	100 ms
Resource usage profiles of VMs	{$CPU_{50\%}$, $CPU_{75\%}$, $CPU_{100\%}$} x {$Mem_{50\%}$, $Mem_{75\%}$, $Mem_{100\%}$}
Number of virtual cores of VMs	{1, 2, 4, 6, 8}
Memory size of VMs	From 1 to 20 GBs in increments of 1
Number of VMs	300

TABLE 2
DATA CENTER'S INPUT DATA FOR EXPERIMENTS 4.1 AND 4.2

Input parameter	Possible values		
Number of UAs	1		
Number of FAs	1		
FA's allocation heuristic	Random		
Monitoring frequency of SMAs	100 ms		
Load-related activation threshold	300		
Specifications for SMAs of Experiment 4.1	Group 1	Group 2	Group 3
Number of SMAs	6	6	6
Number of virtual cores	20	40	60
Memory (GBs)	48	96	144
CPU migration threshold	50%		
Memory migration threshold	50%		
Load balancing heuristics: VM migration heuristic | Selection of destination host	{*Imbalance* | *Imbalance*, H_{CPU}-H_{MEM}| *Greedy*, L_{CPU}-L_{MEM}| *Greedy*, *Random* | *Random*}		
Specifications for SMAs of Experiment 4.2	Group 1	Group 2	Group 3
Number of SMAs	6	6	6
Number of virtual cores	40	40	40
Memory (GBs)	96	96	96
CPU migration threshold	50%	25%	50%
Memory migration threshold	25%	50%	50%
Load balancing heuristics: VM migration heuristic | Selection of destination host	L_{CPU}-H_{MEM}| *Greedy*	H_{CPU}-L_{MEM}| *Greedy*	L_{CPU}-L_{MEM}| *Greedy*

Where α denotes the power consumption when the server is in *sleep* mode; β denotes the power consumption when the server is *on* but *idle*; γ denotes a linear growth factor; and $Host.CPU_{usage}$ denotes overall CPU usage of a host at a given time.

The power consumption model is based on data reported in [6], which (in conjunction with Fan et al. [12]) indicates that there is a quasi-linear relationship between power consumption and CPU usage, see for instance the power consumption of the Intel servers reported in [6]. It is worth mentioning that other power consumption models can be integrated into SMAs. However, defining and integrating other power consumption models is out of the scope of this paper.

4 EMPIRICAL EVALUATION AND RESULTS

Three groups of experiments were conducted using the agent-based testbed defined in Sections 2 and 3, which was implemented using the java agent development framework [3]. Live migration of VMs was simulated by using live migration of agents.

The specifications of the computer on which the experiments were carried out are as follows: 2.7 GHz 12-core Intel Xeon E5, 64 GB 1867 MHz DDR3 ECC, OS X 10.9.5.

4.1 Distributed Load Balancing

Objective. A series of experiments was performed to evaluate the effectiveness and efficiency of agents endowed with the CProtocol in balancing a heterogeneous data center.

Experimental settings. The agent-based testbed has two sets of input parameters: load-related input data (Table 1) and data center's input data (Table 2).

The agents involved in the experiment were 1 UA, 1 FA, 18 SMAs, and 300 VMAs. The UA submitted 300 VM allocation requests. The arrival and departure of VMs were modeled as Poisson processes with an average inter-arrival time of 1000 ms and an average inter-departure time of 35000 ms respectively. It should be noted that Poisson processes are commonly used to model arrivals and departures in a myriad of systems [15]. The values of the load-related input parameters (Table 1) were determined based on an experimental tuning to gradually overload the data center and force SMAs to migrate VMs while having sufficient resources to allocate 300 VMs. A VM allocation request indicates the number of virtual cores and the memory size to be allocated for a VM. A VM, regardless of its resources allocated, can have nine different resource usage profiles (see Table 1), e.g., ($CPU_{50\%}$, $Memory_{100\%}$) meaning the VM has a relatively low CPU usage profile with a long-term mean of 50% and a relatively high memory usage profile with a long-term mean of 100%. CPU and memory usages of VMs were modeled using a first order autoregressive process (AR(1)), which is commonly used to model time series and has been used to model computing resource usages [36]. In addition, the AR(1) is a stochastic process with a long-term mean, which was set to 50%, 75%, and 100% for relatively low, relatively medium, and relatively high resource usages respectively. The number of virtual cores, memory size, and resource usage profile were randomly assigned from the possible values presented in Table 1 in order to generate a heterogeneous load.

The FA selected the destination hosts for VMs at random. In doing so, the SMAs had to balance the load by themselves in a collaborative manner without initial assistance from the FA.

Both the CPU migration threshold and the memory migration threshold of SMAs were set to 50%. In addition, the *load-related activation threshold* was set to 300. In doing so, the SMAs were forced to migrate VMs constantly during the simulations. The data center consisted of eighteen SMAs divided into three groups of six SMAs each. Each group had different specifications regarding the number of virtual cores and memory size to simulate

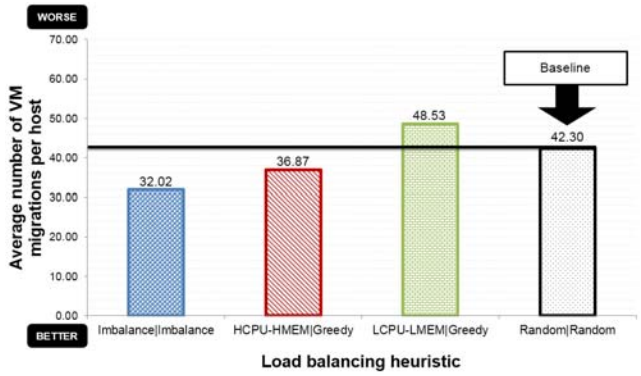

Fig. 4. Load balancing results: number of VM migrations.

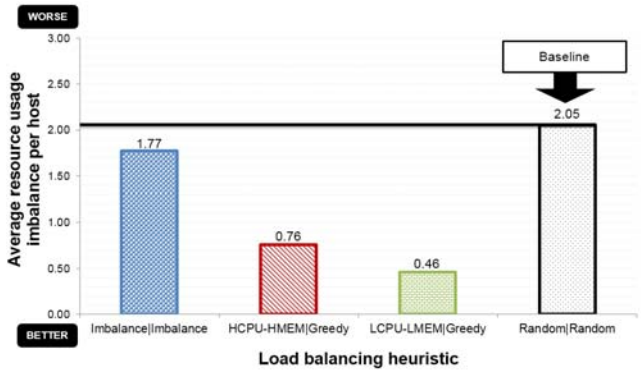

Fig. 5. Load balancing results: resource usage imbalance.

a heterogeneous data center.

Experiment runs were conducted with SMAs using the following combinations of migration heuristics and selection of destination hosts: H_{CPU}-H_{MEM}|*Greedy*, L_{CPU}-L_{MEM}|*Greedy*, *Imbalance*|*Imbalance*, and *Random*|*Random*.

The *Random*|*Random* load balancing heuristic is used as a baseline for benchmarking because random heuristics are commonly used as a baseline to compare other heuristics, e.g., see [39], p. 638.

Performance measures. The performance measures are 1) average number of VM migrations per host, 2) average resource usage imbalance per host, and 3) average sample standard deviation of percentage of resource usage. The *average number of VM migrations per host* is calculated as the overall number of VM migrations divided by the number of SMAs. The *average resource usage imbalance per host* is calculated as the arithmetic mean of the absolute differences between CPU and memory usage samples of each SMA through an experiment run. In a similar manner, the *average sample standard deviation of percentage of resource usage* is calculated as the arithmetic mean of the standard deviations of all the resource usage samples of each SMA through an experiment run.

For each configuration of the agent-based testbed (presented in Table 2), 10 experiment runs were carried out. Each experiment run consisted of balancing the load derived from 300 VMs.

Results. Empirical results are presented in Figs. 4, 5, and 6. From these results, three observations are drawn.

Observation 1. On average, SMAs using the *Imbalance*|*Imbalance* heuristic migrated fewer VMs than SMAs using the H_{CPU}-H_{MEM}|*Greedy*, the L_{CPU}-L_{MEM}|*Greedy* or the *Random*|*Random* heuristics.

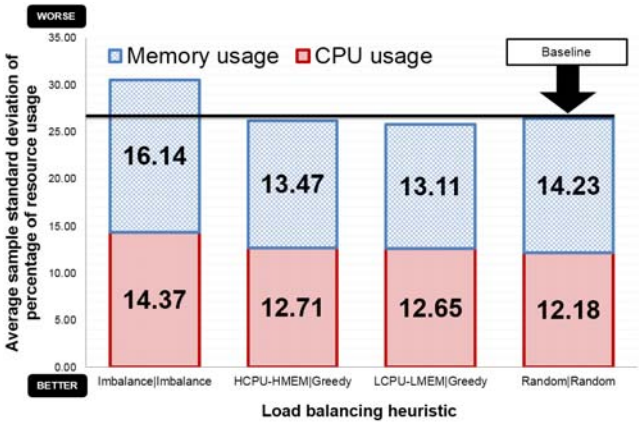

Fig. 6. Load balancing results: standard deviation of resource usage.

Analysis. As shown in Fig. 4, SMAs balancing loads using the *Imbalance*|*Imbalance* heuristic migrated 24.30% fewer VMs than SMAs using the *Random*|*Random* heuristic. This is because SMAs using the *Imbalance*|*Imbalance* heuristic migrated VMs (contributing the most to the resource usage imbalance of the hosts of the initiator SMAs) to destination hosts whose resource usage imbalances were reduced the most.

SMAs balancing loads using the H_{CPU}-H_{MEM}|*Greedy* heuristic migrated 12.83% fewer VMs than SMAs using the *Random*|*Random* heuristic, see Fig. 4. The SMAs using the H_{CPU}-H_{MEM}|*Greedy* heuristic by migrating VMs with substantial computing requirements required fewer VM migrations to keep their load below their migration thresholds than SMAs using the L_{CPU}-L_{MEM}|*Greedy* or the *Random*|*Random* heuristics.

SMAs balancing loads using the L_{CPU}-L_{MEM}|*Greedy* heuristic migrated more VMs than SMAs using the *Random*|*Random* heuristic, see Fig. 4. This is because SMAs migrating VMs with the lowest CPU and memory usages required more migrations of VMs that contributed little to keep their load below their migration thresholds. In contrast, the SMAs using the *Random*|*Random* heuristic (which is blind to resource usage) randomly selected VMs that contributed, to some extent, to keep their load below their migration thresholds by sometimes selecting VMs with regular or high resource usages.

In conclusion, collaborative SMAs endowed with the CProtocol and using the *Imbalance*|*Imbalance* or the H_{CPU}-H_{MEM}|*Greedy* heuristics are capable of efficiently balancing loads in a distributed manner with a relatively low overhead caused by VM migrations.

Observation 2. SMAs endowed with the CProtocol attained a relatively low resource usage imbalance (i.e., a resource usage imbalance equal or less than 2.05) regardless of the load balancing heuristic.

Analysis. As observed in Fig. 5, SMAs using the *Imbalance*|*Imbalance*, the H_{CPU}-H_{MEM}|*Greedy*, the L_{CPU}-L_{MEM}|*Greedy* heuristics attained a smaller resource usage imbalance than SMAs using the *Random*|*Random* heuristic. This is because the *Random*|*Random* heuristic is blind

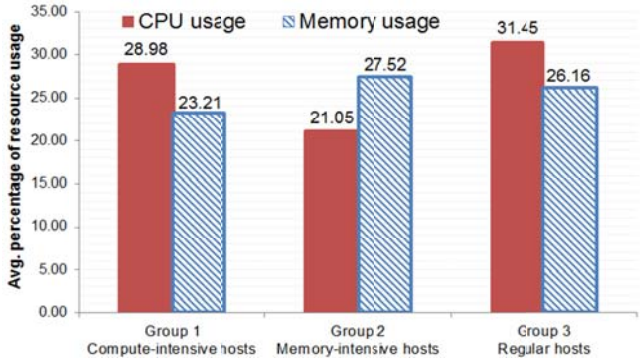

Fig. 7. Avg. percentage of resource usage grouped by host type.

to resource usage. Nevertheless, SMAs using the *Random*|*Random* heuristic attained a relatively low resource usage imbalance of 2.05. For instance, the resource usage imbalance of 2.05 may have resulted from the absolute difference of a memory usage of 90.00% and a CPU usage of 92.05%. This negligible resource usage imbalance was attained because the load introduced by the UA consisted of VM allocation requests containing randomly and uniformly assigned requirements, e.g., number of virtual cores and resource usage profile of VMs. Then, the random generation of the VM allocation requests may have introduced a similar number of regular, compute-intensive, and memory-intensive VMs. In addition, all the SMAs 1) were increasingly overloaded, 2) had the same CPU and memory migration thresholds, and 3) as soon as their load was below their migration thresholds, they hosted other VMs by adopting the CProtocol.

In summary, the SMAs through collaboration can handle heterogeneous loads while keeping a negligible resource usage imbalance.

Observation 3. Whereas SMAs using the L_{CPU}-L_{MEM}|*Greedy* heuristic attained the most balanced data center, SMAs using the *Imbalance*|*Imbalance* heuristic attained the least balanced data center.

Analysis. Fig. 6 shows that the average sample standard deviation of percentage of resource usage of hosts ranged from 30.51 (with SMAs using the *Imbalance*|*Imbalance* heuristic) to 25.76 (with SMAs using the L_{CPU}-L_{MEM}|*Greedy* heuristic). SMAs using the L_{CPU}-L_{MEM}|*Greedy* heuristic attained the most balanced data center because they caused relatively low variations in the load of the hosts by migrating VMs with low resource usages. However, this also caused a considerably high number of VM migrations as analyzed in Observation 1. In contrast, SMAs using the *Imbalance*|*Imbalance* heuristic attained the least balanced data center, but only with a relatively small difference of 4.75 and migrating considerably fewer VMs than SMAs using the L_{CPU}-L_{MEM}|*Greedy* heuristic.

As analyzed in Observations 1, 2, and 3, in general, SMAs through collaboration can efficiently and effectively distribute heterogeneous loads achieving balance within and across hosts. In addition, in most of the cases, SMAs using the *Imbalance*|*Imbalance*, the H_{CPU}-H_{MEM}|*Greedy*, and the L_{CPU}-L_{MEM}|*Greedy* heuristics outperformed SMAs using the *Random*|*Random* heuristic, see Figs. 4, 5, and 6.

4.2 Server-centric Load Management Policies

Objective. A series of experiments was performed to evaluate the effectiveness of SMAs using server-centric load management policies (integrated into the CProtocol) in dynamically consolidating heterogeneous loads.

Experimental settings. The data center consisted of 18 SMAs (provided with the same number of cores and memory size) divided into three groups of 6 SMAs each, see Table 2. Each group had SMAs with different CPU and memory migration thresholds in order to consolidate different load types into different groups of hosts.

The SMAs of group 1 (compute-intensive hosts) were configured to consolidate CPU load by setting a CPU migration threshold of 50% and a memory migration threshold of 25%, and providing them with the L_{CPU}-H_{MEM}|*Greedy* heuristic. In doing so, the SMAs of group 1 were configured to keep compute-intensive VMs and migrate memory-intensive VMs.

The SMAs of group 2 (memory-intensive hosts) were configured to consolidate memory load by setting a CPU migration threshold of 25% and a memory migration threshold of 50%, and providing them with the H_{CPU}-L_{MEM}|*Greedy* heuristic. In doing so, the SMAs of group 2 were configured to keep memory-intensive VMs and migrate compute-intensive VMs.

The SMAs of group 3 (regular hosts) were configured to consolidate both CPU and memory load by setting a CPU migration threshold of 50% and a memory migration threshold of 50%, and providing them with the L_{CPU}-L_{MEM}|*Greedy* heuristic. In doing so, the SMAs of group 3 were configured to have no preference for memory-intensive or compute-intensive VMs.

See Section 4.1 for detailed descriptions and justifications of the load-related input parameters (Table 1) and the remaining input parameters of the testbed.

Performance measures. The performance measures are average percentage of CPU usage and average percentage of memory usage grouped by host type, namely, compute-intensive, memory-intensive, and regular hosts.

For each configuration of the agent-based testbed (presented in Table 2), 10 experiment runs were carried out.

Results. Empirical results are presented in Fig. 7. From these results, one observation is drawn.

Observation 4. Compute-intensive hosts had a relatively high CPU usage and a relatively low memory usage. In contrast, memory-intensive hosts had a relatively high memory usage and a relatively low CPU usage. In the case of regular hosts, they consolidated both CPU-intensive and memory-intensive loads.

Analysis. SMAs of compute-intensive hosts consolidated a CPU-intensive load (see Fig. 7) by 1) migrating VMs with the lowest CPU usage and the highest memory usage, and 2) triggering VM migrations when their relatively low memory migration thresholds were exceeded. In addition, SMAs of compute-intensive hosts migrated memory-intensive VMs mostly to either memory-intensive or regular hosts because, in most of the cases, the SMAs of compute-intensive hosts may have not even

TABLE 3
LOAD-RELATED INPUT DATA FOR EXPERIMENT 4.3

Input parameter	Possible values
Number of VMs	1500
Number of virtual cores of VMs	{1, 2, 4, 6, 8}
Memory size of VMs	From 1 to 20 GBs in increments of 1
Monitoring frequency of VMs	100 ms
CPU usage profile of VMs	{$CPU_{50\%}$, $CPU_{75\%}$, $CPU_{100\%}$}
Avg. inter-arrival time of VMs	100 ms
Avg. inter-departure time of VMs	15000 ms

TABLE 4
DATA CENTER'S INPUT DATA FOR EXPERIMENT 4.3

Input parameter	Possible values
Number of UAs	1
Number of FAs	1
Energy-related migration threshold	{90%, 80%, 70%, 60%, 50%}
FA's load balancing heuristics	{Best-fit, Random} heuristics
Number of SMAs	20
Number of cores of SMAs	40
Memory size of SMAs	96 GBs
Selection of destination host	Greedy
VM migration heuristic	H_{CPU}
SMAs' monitoring frequency	1000 ms
Energy-aware activation threshold	10 threshold violations
Simulation time rate:	1 s in simulation is equal to 1 m

sent proposals to host memory-intensive VMs, which may have exceeded their low memory migration thresholds. Similar arguments explain why SMAs of memory-intensive hosts consolidated a memory-intensive load (see Fig. 7). SMAs of regular hosts consolidated more load than SMAs of either compute-intensive or memory-intensive hosts (see Fig. 7) because they migrated fewer VMs due to the fact that they had higher memory and CPU migration thresholds, and as a consequence they also sent more proposals to host VMs.

It is noteworthy that SMAs of regular hosts consolidated more CPU load than memory load, see Fig. 7. This is because the load introduced by the UA was unbalanced with respect to computing requirements, i.e., the VM allocation requests required a higher share of the virtual cores than the share of memory of the data center.

In summary, on average SMAs of compute-intensive and regular hosts consolidated more CPU load than SMAs of memory-intensive hosts. In addition, on average, SMAs of memory-intensive and regular hosts consolidated more memory load than SMAs of compute-intensive hosts.

It should be noted that even though the load management policies and load balancing heuristics were defined in a per server basis, different load types can be effectively consolidated into different groups of hosts in a distributed manner.

4.3 Distributed Energy-Aware Server Consolidation

Objective. A series of experiments was designed to explore the efficiency of agents adopting the EProtocol in carrying out energy-aware server consolidation.

Experimental settings. The testbed has two different sets of input parameters: load-related input data (Table 3) and data center's input data (Table 4).

The agents involved in the experiment were 1 UA, 1 FA, 20 SMAs, and 1500 VMAs. The number of VM allocations was set to 1500 with an average inter-arrival time and an average inter-departure time of 100 ms and 15000 ms respectively. The number of cores and the memory size of the VM allocation requests were randomly assigned from the values reported in Table 3. These values were determined based on an experimental tuning of the testbed to overload the data center, but to prevent rejection of VM allocations due to insufficient resources. In addition, to create a heterogeneous load, the VMs were provided with a randomly assigned CPU usage profile. The possible long-term means of CPU usage were 50%, 75%, and 100% for low, medium and high usage respectively.

The FA was endowed with the best-fit and random load balancing heuristics for the initial allocation of VMs. Both heuristics are commonly used as benchmarks for energy-aware server consolidation, see [8] for an example.

The SMAs were provided with the $H_{CPU}|Greedy$ heuristic for the selection of VMs and destination hosts, see Section 3.5 for a justification of the selection of this heuristic.

Only CPU usage was taken into account because this experiment explore energy-aware server consolidation, and according to Chen et al. [6] and Fan et al. [12], there is a quasi-linear relationship between power consumption and CPU usage.

The *energy-related migration threshold* of SMAs was varied from 90% to 50% in steps of 10%. In doing so, the efficiency was explored when the SMAs were increasingly forced to migrate VMs (in order to be turned off) due to increasingly stringent thresholds.

The number of SMAs involved in the experiments was set to 20, each provided with 40 cores and 96 GBs based on the server specifications reported in [7]. It should be noted that both the number of servers as well as their specifications were sufficient to handle the load described in Table 3. The SMAs' *energy-related activation threshold* was set to 10 threshold violations so as to continuously trigger the distributed server consolidation process during most of the simulation time. In general, an *energy-related activation threshold* of 10 violations corresponds to an SMA continuously exceeding its *energy-related migration threshold* for 10 m according to the simulation time rate and the monitoring frequency reported in Table 4.

The servers' power consumption model is an instance of (7) with $\alpha = 6.0$, $\beta = 106.238$, and $\gamma = 0.528$. The values of the constants of the power consumption model were calculated using a linear regression of the relationship between CPU usage and power consumption for one of the Intel servers reported in [6].

Performance measures. The performance measures are 1) kilowatt-hour (kWh) consumption for the cloud data center, 2) average number of VM migrations, 3) average number of

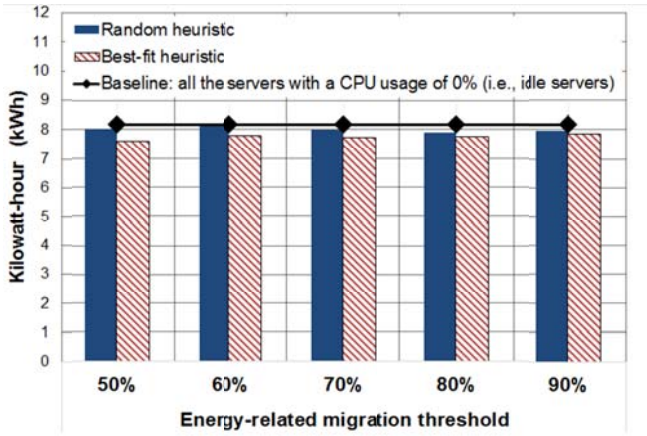

Fig. 8. kWh consumption for the Cloud data center.

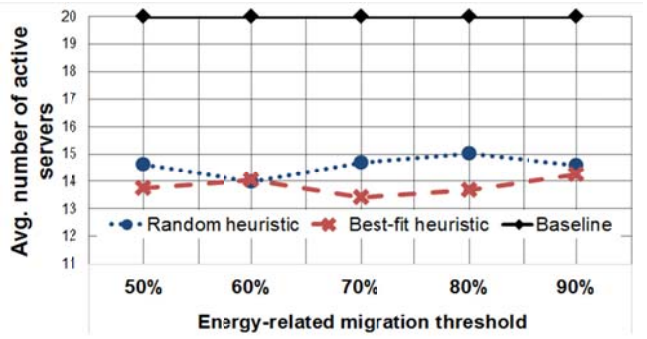

Fig. 9. Average number of active servers.

times a server was turned on, and 4) average number of active servers.

For each configuration of the agent-based testbed (presented in Table 4), 10 experiment runs varying the *energy-related migration threshold* were conducted.

Results. Empirical results are presented in Figs. 8, 9, 10, and 11. From these results, three observations are drawn.

Observation 5. In general, the agents endowed with the EProtocol attained a kWh consumption for the data center lower than the kWh consumption attained when the hosts were idle (see Fig. 8).

Analysis. The baseline (kWh consumption of all the servers with a CPU usage of 0%) shown in Fig. 8 was calculated taking into account the servers' wattage when they are idle (106.238 w) and the average simulated time (3.85 h) of the experiments runs. Having said that, SMAs using the EProtocol (in conjunction with the FA using either the best-fit or the random heuristic) attained a lower kWh consumption than the kWh consumption of the baseline because:

- The SMAs using the EProtocol (by autonomously migrating VMs when their power consumption thresholds were continuously exceeded) were capable of reducing the kWh consumption for the data center.
- The FA turned on hosts only when they were needed, i.e., when the current active servers were unable to host a given VM and there were hosts in *sleep* mode.

Observation 6. Overall, 75% of the servers (with respect to the baseline consisting of all the 20 servers being active) were sufficient for handling the load of the data center (Fig. 9).

Analysis. The FA and the SMAs by autonomously collaborating with each other were capable of consolidating the load of underutilized servers (but with a relative high kWh consumption) on a varying number of active servers according to the load.

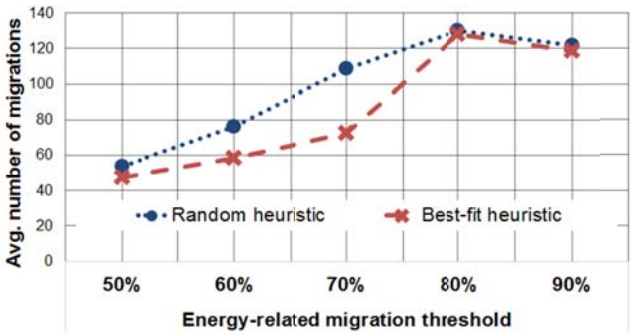

Fig. 10. Average number of VM migrations.

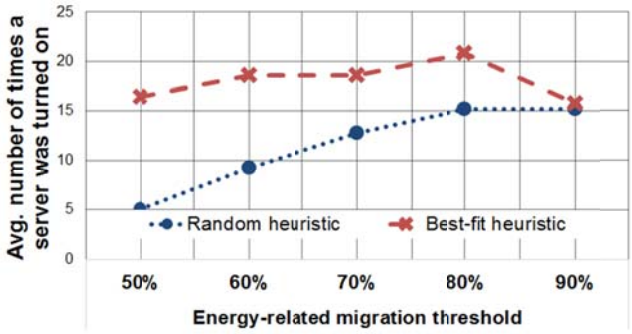

Fig. 11. Average number of times a server was turned on.

It is worth mentioning that the different *energy-related migration thresholds* of SMAs did not significantly influence the number of active servers (see Fig. 9). However, the different *energy-related migration thresholds* significantly influenced the number of VM migrations (see Fig. 10) because:

1) 75% of the servers were needed and sufficient to handle the load described in Table 3.

2) Given that the *energy-related migration threshold* indicates the minimum power consumption allowed for a host, a high *energy-related migration threshold* caused a relatively high number of VM migrations for the given input load as shown in Fig. 10.

Observation 7. The FA using the best-fit heuristic attained better performance than the FA using the random heuristic except for the average number of times a server was turned on.

Analysis. In general, the FA using the best-fit heuristic attained a lower kWh consumption (Fig. 8), a lower average number of active servers (Fig. 9), and a lower average number of VM migrations (Fig. 10) than the FA using the random heuristic.

The FA using the best-fit heuristic achieved the lowest number of active servers because the computer specifications of the hosts were the same, and as a consequence, the FA allocated VMs in an orderly manner by firstly allocating the VMs to SMA_1, then to SMA_2, and so on. This caused that, at the beginning of the simulations, the very last SMAs (e.g., SMA_{19} and SMA_{20}) were turned off for a brief period of time, and turned on again afterward due to the continuous arrival

of VM allocation requests. This explains the considerably higher average number of times a server was turned on when the FA used the best-fit heuristic with respect to when the FA used the random heuristic as shown in Fig. 11.

The FA by using the best-fit heuristic achieved a lower kWh consumption than when the FA used the random heuristic because the FA using the best-fit heuristic achieved the lowest average number of active servers (Fig. 9). In general, the lower the number of active servers, the lower the kWh consumption for the data center.

When the FA used the random heuristic, more VMs were migrated than when the FA used the best-fit heuristic. This is because the FA using the random heuristic allocated the VMs randomly using a uniform distribution. As a consequence, the VMs were allocated to all the servers uniformly, which had insufficient VMs to prevent their *energy-related migration threshold* and *energy-related activation threshold* from being exceeded. This caused that the SMAs had to migrate VMs to other hosts continuously.

5 Related Work Comparison

Load management mechanisms have been designed for different computing environments and with different emphasis ranging from mechanisms focused on balancing compute-intensive load for parallel processors [5] to mechanisms focused on balancing storage load [28]. Since this work focuses on energy-aware load management in distributed systems by means of VM live migration, related work includes 1) distributed load management mechanisms, 2) load management algorithms based on VM live migration, and 3) energy-aware server consolidation.

5.1 Distributed Load Management Mechanisms

Randles et al. [27] quantitatively compare three distributed load balancing mechanisms for Cloud environments: a honey bee foraging algorithm [21], an active clustering algorithm [9], and a random-walk algorithm [26]. The load balancing mechanism based on honey bee foraging behavior [21] uses a global advertisement board where hosts post advertisements regarding their performance. These advertisements are used to calculate the global colony's profit that guides the load balancing algorithm. In the active clustering algorithm presented in [9], hosts have to be aware of the specific services (e.g., image compression) of their neighbor hosts. The hosts self-organize themselves into homogeneous clusters and then distribute a specific load across the hosts belonging to the same cluster. The load balancing mechanism based on random walks proposed by Rahmeh et al. [26] creates a directed graph whose vertices represent computing nodes. The in-degree of a node indicates its number of available resources. On the one hand, each time a task is assigned to a node, its in-degree is decreased. On the other hand, when the task is executed by the node, its in-degree is increased. Tasks are assigned to the node of the highest in-degree, i.e., the least loaded, among a set of nodes sampled by means of a random walk influenced by communication delay. The random walk is performed on a network graph created from routing tables.

Unlike [9], [21], and [26], the present work does not require additional structures or services such as advertisement boards as in [21], service discovery systems as in [9] or routing tables as in [26]. In the present work, a *call-for-proposals* message is broadcasted to available hosts and only underutilized hosts participate in the bid for the allocation of a VM. Compared to [9], [21], and [26], the proposed agent-based load management algorithm takes into account both memory and CPU usage of hosts instead of time to serve requests as in [21] and number of tasks currently assigned to hosts as in [9] and [26], which may not represent the load of a computing node.

5.2 Load Management Mechanisms supported by VM Live Migration

Barak and Shiloh [2] propose a distributed load balancing mechanism supported by process migration. Each computing node keeps track of resource usages of n randomly selected computing nodes, which are constantly exchanging information with each other. The decision about where to migrate VMs is made based on load estimates of computing nodes. The selection of the process for migration is based on the process characteristics, e.g., size of the process. Compared to [2], where computing nodes increase overhead costs by constantly exchanging information every second, agents adopting the CProtocol only interact with each other when a host is overloaded. In addition, the load balancing mechanism proposed by Barak and Shiloh separates the VM migration heuristics from the host selection heuristics, i.e., there is not a rationale behind combining them. In contrast, in the present work, agents are provided with load balancing heuristics, e.g., the *Imbalance*|*Imbalance* heuristic, where the selection of VMs for migration is matched with the selection of destination hosts.

Zhao and Huang [38] propose a distributed load balancing mechanism supported by VM live migration. In [38], the VM with the highest combination of CPU and I/O usages is selected for migration. Destination hosts are selected based on the number of VMs allocated to them. The higher the number of VMs allocated to a host, the higher the probability of being selected as the destination host. The rationale behind this heuristic is that hosts that are capable of hosting multiple VMs are computationally powerful. Zhao and Huang show that this algorithm reduces the standard deviation of the average processor usage over time. However, the results reported do not include the number of VM migrations. In addition, unlike the present work, Zhao and Huang assumed that all the hosts have sufficient resources regarding number of cores and available memory to host VMs. Moreover, the load balancing heuristics of the present work take into account both the resource usage of hosts and the resource usage of VMs.

Fan et al. [11] propose an agent-based service migration framework for hybrid Cloud environments. Agents are deployed in both private and public Clouds. There are agents in charge of 1) monitoring computer resources, 2) allocating and releasing computing resources, 3) migrating services, and 4) keeping track of agents deployed in

the environment. The migration policies explored by Fan et al. are based on the load difference between the private and the public Cloud in terms of the number of jobs currently being executed, the size of jobs, and estimated completion times. The agent-based system proposed by Fan et al. elastically allocates and releases computing resources from the public Cloud based on the load of the private Cloud. It should be noted that Fan et al. assumed that all the jobs are initiated on resources of the private Cloud regardless of whether the private Cloud is overloaded or not. Their migration heuristics are only based on job characteristics disregarding server heterogeneity. In contrast, the present work takes into account the overall CPU and memory usage of hosts to determine the best destination host for a VM, e.g., the host whose resource usage imbalance would be reduced the most by hosting the VM.

Anderson et al. [1] propose a self-organizing agent-based load balancing mechanism supported by VM live migration. When an agent, based on a threshold, detects that a host is overloaded, the agent of the overloaded host migrates a VM to an underutilized host randomly selected from a pool of hosts. When, according to some thresholds, an agent is balanced, the agent stops migrating and allocating VMs. Anderson et al. use lightweight coordination calculus to define the agent interaction protocols that guide load balancing. These protocols allow only one-to-one interaction between an overloaded agent and any underutilized agent and thus the host of the underutilized agent may not be the best destination host for a given VM. In addition, unlike the present work, Anderson et al. do not propose a mechanism to select VMs for migration.

A different perspective on load management is provided by VMware [33] who focuses on relationships among VMs. Closely related VMs, e.g., VMs hosting services part of the same workflow, may exchange data among each other and thus the VMs may benefit from being hosted in the same host. The relationships between VMs must be identified by using affinity rules, which allow indicating that two VMs should be kept together. Affinity rules can also be used to indicate that two VMs should be kept apart in order to increase availability. Whereas in [33] the matching between VMs and hosts is conducted statically and with prior knowledge of the relationships between VMs, in the present work, the matching between VMs and hosts is performed dynamically without any prior knowledge of resource usage profiles of VMs.

The Red Hat Enterprise Virtualization suite [10] makes use of CPU usage thresholds to trigger VM migrations, which are defined in a per server basis. Nevertheless, load balancing is controlled by a central entity that determines what VMs should be migrated and to what destination hosts. Priority is given to the host with the lowest CPU usage. One of the main differences between Red Hat and the present work is that whereas in Red Hat the decisions about VM migrations are made by a central entity, in the present work, decisions about VM migrations result from the collaborative effort of agents in a distributed manner. Furthermore, the load balancing heuristics presented in this work not only takes into account CPU usage (as in Red Hat) but memory usage as well. However, both the present work and Red Hat, do not take into account disk I/O-intensive workloads, which may also cause performance deterioration of hosts. Nevertheless, managing disk I/O-intensive workloads is out of the scope of this work, which focuses on memory-intensive and compute-intensive loads. See [25] for a load balancing mechanism for disk I/O-intensive workloads.

5.3 Energy-Aware Server Consolidation Mechanisms

Server consolidation for data centers is commonly tackled using centralized approaches, see for instance [18], [29], and [32]. However, centralized consolidation approaches involve continuously monitoring the infrastructure of the data center, which increases communication overhead and limits the data center's ability to scale up. In this regard, the present work makes use of an event-based monitoring protocol that is only activated when the thresholds of SMAs are exceeded.

With respect to distributed approaches for server consolidation, Wang et al. [35] proposes a peer-to-peer system where hosts exchange resource usage information among them periodically, which may be unnecessary if no server consolidation is required. In a similar manner, Marzolla et al. [19] use a gossip protocol to update load vectors of neighbors (i.e., other servers) periodically. It should be noted that in [19] the update process also depends on a fixed update rate and despite the fact that the update process involves only neighboring nodes, maintaining the list of neighbors is not a trivial task in dynamic environments such as data centers. It is worth mentioning that both [19] and [35] do not take into account that data centers are concurrent systems where a host may be allocating resources for hosting VMs coming from either front-end servers or other hosts unlike the present concurrent distributed framework.

It is acknowledged that this present work is a considerably and significantly extended version of the work presented in [16]. This work enhances [16] by:

1. Defining an imbalance factor that indicates how much a VM contributes to the resource usage imbalance of a host.
2. Devising and implementing the imbalance-based migration heuristic and the imbalance-based selection of destination hosts.
3. Designing, implementing, and evaluating the EProtocol, a concurrent distributed framework for energy-aware server consolidation.
4. Conducting a new set of experiments to evaluate the effectiveness and efficiency of agents endowed with the CProtocol in balancing a data center.
5. Generalizing the results presented in [16] with respect to the efficiency of agents in dynamically consolidating load by conducting experiments using more stringent experimental settings.

6 Conclusion and Future Work

The significance of this work is that by using distributed problem solving techniques, heterogeneous loads can be efficiently balanced and effectively consolidated. Agents using both the CProtocol and the EProtocol manage loads in a distributed manner while reducing energy consumption costs and taking into account server heterogeneity.

The novelty of this work is that energy-aware load consolidation and load balancing in data centers can be collaboratively achieved by agents concurrently interacting with each other. In this work, agents are endowed with 1) server-centric load management policies, 2) VM migration heuristics, 3) host selection heuristics, and 4) policies for determining when to turn off/on hosts, which are integrated into the CProtocol and the EProtocol. In addition, this work also contributes a novel *Imbalance|Imbalance* load balancing heuristic that migrates VMs causing the largest resource usage imbalance from overloaded hosts to underutilized hosts whose resource usage imbalances are reduced the most. Moreover, the EProtocol allows for a combination of front-end energy-aware heuristics (e.g., the best-fit heuristic) and distributed server consolidation in a concurrent and collaborative manner. Furthermore, by adopting event-based load management mechanisms, the overhead is lower than the overhead of centralized approaches [10], [18], [23], [29], [32] and/or of other distributed approaches [19], [35]. Finally, the proposed distributed problem solving techniques allow the load management mechanisms of data centers to scale up.

It is acknowledged that the proposed distributed problem solving techniques for load management assume that agents are collaborative, which is a reasonable assumption in data centers owned by a single organization. Nevertheless, in data centers owned by multiple organizations, e.g., federated data centers, agents cannot be assumed to collaborate. Then, a mechanism to deal with self-interested agents should be devised to extend the scope of the proposed agent-based load management mechanisms to include data centers owned by multiple organizations. Other future research directions include 1) devising load balancing heuristics for the initial allocation of VMs to hosts in order to prevent unnecessary VM migrations; 2) constructing resource usage profiles of VMs and hosts using statistical forecasting for assisting the dynamic placement of VMs; 3) designing VM-centric management policies similar to affinity rules proposed in [33] to take into account relationships among VMs, e.g., when two VMs share storage; and 4) exploring the effects of resource overselling of hosts, e.g., memory overcommitment, to improve the decision-making of SMAs.

Acknowledgments

This work has been supported by Asociación Mexicana de Cultura A.C. In addition, the authors would like to thank the Guest Editors and the anonymous referees for their comments and suggestions.

References

[1] P. Anderson, S. Bijani, and A. Vichos, "Multi-agent Negotiation of Virtual Machine Migration Using the Lightweight Coordination Calculus," *Agent and Multi-Agent Systems. Technologies and Applications*, G. Jezic, M. Kusek, N.-T., Nguyen, R.J. Howlett and L.C. Jain, eds., Lecture Notes in Computer Science, vol. 7372, Springer Berlin Heidelberg, pp. 124-133, 2012.

[2] A. Barak and A. Shiloh, "A distributed load-balancing policy for a multicomputer," *Software: Practice and Experience*, vol. 15, no. 9, pp. 901-913, 1985.

[3] F. Bellifemine, F. Bergenti, G. Caire, and A. Poggi, "Jade — A Java Agent Development Framework," *Multi-Agent Programming*, R.H. Bordini, M. Dastani, J. Dix and A.E.F. Seghrouchni, eds., Multiagent Systems, Artificial Societies, and Simulated Organizations Series, Springer US, pp. 125-147, 2005.

[4] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing SLA violations," *Proc. of the 10th IFIP/IEEE International Symposium on Integrated Network Management*, pp. 119-128, 2007.

[5] R.K. Boel and J.H. van Schuppen, "Distributed load balancing," *Proc. of the 27th IEEE Conference on Decision and Control*, pp. 1486, 1988.

[6] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao, "Energy-aware server provisioning and load dispatching for connection-intensive Internet services," *Proc. of the 5th USENIX Symposium on Networked Systems Design & Implementation*, vol. 8, pp. 337-350, 2008.

[7] Cisco systems, "Cisco UCS B440 M2 high-performance blade server," available at http://www.cisco.com/c/en/us/products/servers-unified-computing/ucs-b440-m2-high-performance-blade-server/index.html. 2015.

[8] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "Toward energy-efficient cloud computing: Prediction, consolidation, and overcommitment," *IEEE Network*, vol. 29, pp. 56-61, March-April 2015.

[9] E. Di Nitto, D.J. Dubois, R. Mirandola, F. Saffre, and R. Tateson, "Applying self-aggregation to load balancing: experimental results," *Proc. of the 3rd International Conference on Bio-Inspired Models of Network, Information and Computing Systems*, pp. 14, 2008.

[10] Z. Dover, S. Gordon, and T. Hildred, "The Technical Architecture of Red Hat Enterprise Virtualization Environments – Edition 1," *Red Hat Enterprise Virtualization 3.2 - Technical Reference Guide*, available at https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Virtualization/3.2/pdf/Technical_Reference_Guide/Red_Hat_Enterprise_Virtualization-3.2-Technical_Reference_Guide-en-US.pdf. 2015.

[11] C.T. Fan, W.J. Wang, and Y.S. Chang, "Agent-based service migration framework in hybrid cloud," *Proc. of the 13th IEEE International Conference on High Performance Computing and Communications*, pp. 887-892, 2011.

[12] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," *Proc. of the 34th Annual International Symposium on Computer Architecture*, pp. 13-23, 2007.

[13] Foundation for Intelligent Physical Agents, "FIPA Request Interaction Protocol Specification," FIPA Specifications, available at http://www.fipa.org/specs/fipa00026/. 2015.

[14] S. Ghiasi, T. Keller, and F. Rawson, "Scheduling for heteroge-

neous processors in server systems," *Proc. of the 2nd Conference on Computing Frontiers*, pp. 199-210, 2005.

[15] D. Gross, J.F. Shortle, J.M. Thompson, and C.M. Harris, "*Fundamentals of Queueing Theory*," Boston, Massachusetts, John Wiley & Sons, 2008.

[16] J.O. Gutierrez-Garcia and A. Ramirez-Nafarrate, "Policy-Based Agents for Virtual Machine Migration in Cloud Data Centers," *Proc. of the IEEE International Conference on Services Computing*, pp. 603-610, 2013.

[17] A. Kerr, G. Diamos, and S. Yalamanchili, "A Characterization and Analysis of GPGPU Kernels," Technical Report GIT-CERCS-09-06, Georgia Institute of Technology, Atlanta, Georgia, May 2009.

[18] B. Li, J. Li, J. Huai, T. Wo, Q. Li, and L. Zhong, "Enacloud: An energy-saving application live placement approach for cloud computing environments," *Proc. of the IEEE International Conference on Cloud Computing*, pp. 17-24, 2009.

[19] M. Marzolla, O. Babaoglu, and F. Panzieri, "Server consolidation in clouds through gossiping," *Proc. of the IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, pp. 1-6, 2011.

[20] M. Mishra, A. Das, P. Kulkarni, and A. Sahoo, "Dynamic resource management using virtual machine migrations," *IEEE Communications Magazine*, vol. 50, no. 9, pp. 34-40, 2012.

[21] S. Nakrani and C. Tovey, "On honey bees and dynamic server allocation in internet hosting centers," *Adaptive Behavior*, vol. 12, no. 3-4, pp. 223-240, 2004.

[22] M.A. Netto, C. Vecchiola, M. Kirley, C.A. Varela, and R. Buyya, "Use of run time predictions for automatic co-allocation of multi-cluster resources for iterative parallel applications," *Journal of Parallel and Distributed Computing*, vol. 71, no. 10, pp. 1388-1399, 2011.

[23] J. Ni, Y. Huang, Z. Luan, J. Zhang, and D. Qian, "Virtual machine mapping policy based on load balancing in private cloud environment," *Proc. of the IEEE International Conference on Cloud and Service Computing*, pp. 292-295, 2011.

[24] C. Ortiz, "Building the Dynamic Data Center," *Dell Power Solutions*, available at http://www.dell.com/downloads/global/power/ps3q10-20100470-ortiz.pdf. 2010.

[25] X. Qin, H. Jiang, Y. Zhu, and D.R. Swanson, "Dynamic load balancing for I/O-intensive tasks on heterogeneous clusters, *High Performance Computing – HiPC 2003*, T.M. Pinkston V.K. Prasanna, eds., Lecture Notes in Computer Science, vol. 2913, Springer Berlin Heidelberg, pp. 300-309, 2003.

[26] O.A. Rahmeh, P. Johnson, and A. Taleb-Bendiab, "A dynamic biased random sampling scheme for scalable and reliable grid networks," *INFOCOMP Journal of Computer Science*, vol. 7, no. 4, pp. 1-10, 2008.

[27] M. Randles, D. Lamb, and A. Taleb-Bendiab, "A comparative study into distributed load balancing algorithms for cloud computing," *Proc. of the IEEE International Conference on Advanced Information Networking and Applications Workshops*, pp. 551-556, 2010.

[28] M.C. Riley and C. Scheideler, "Distributed Load Balancing," Ph.D. Dissertation, The Johns Hopkins University, 2004.

[29] S. Srikantaiah, A. Kansal, and F. Zhao, "Energy aware consolidation for cloud computing," *Proc. of the Conference on Power Aware Computing and Systems*, Vol. 10, pp. 1-5, 2008.

[30] Y. L. Tsai, K. C. Huang, H. Y. Chang, J. Ko, E. T. Wang, C. H. Hsu, "Scheduling multiple scientific and engineering workflows through task clustering and best-fit allocation," *Proc. of the IEEE Eighth World Congress on Services*, pp. 1-8, 2012.

[31] M. Uddin and A. A. Rahman, "Techniques to implement in green data centres to achieve energy efficiency and reduce global warming effects," *International Journal of Global Warming*, vol. 3, no. 4, pp. 372-389, 2011.

[32] A. Verma, G. Dasgupta, T. K. Nayak, P. De, and R. Kothari, "Server workload analysis for power minimization using consolidation," *Proc. of the USENIX Annual Technical Conference*, pp. 28-28, 2009.

[33] VMware, Inc., "Resource Management with VMware Distributed Resource Scheduler," *VMware Best Practices*, available at http://www.vmware.com/pdf/vmware_drs_wp.pdf. 2015.

[34] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, "Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation," *Cloud Computing*, M. Gilje Jaatun, G. Zhao and C. Rong, eds., Lecture Notes in Computer Science, vol. 5931, Springer Berlin Heidelberg, pp. 254-265, 2009.

[35] X. Wang, X. Liu, L. Fan, and X. Jia, "A decentralized virtual machine migration approach of data centers for cloud computing," *Mathematical Problems in Engineering*, Article ID 878542, pp. 1-10, 2013.

[36] T. Wood, P.J. Shenoy, A. Venkataramani, and M.S. Yousif, "Black-box and Gray-box Strategies for Virtual Machine Migration," *Proc. of the 4th USENIX Symposium on Networked Systems Design & Implementation*, pp. 229-242, 2007.

[37] F. Zhang, J. Wu, and Z. Lu, "PSRPS: A Workload Pattern Sensitive Resource Provisioning Scheme for Cloud Systems," *Proc. of the IEEE International Conference on Services Computing*, pp. 344-351, 2013.

[38] Y. Zhao and W. Huang, "Adaptive distributed load balancing algorithm based on live migration of virtual machines in cloud," *Proc. of the 5th IEEE International Joint Conference on Networked Computing*, pp. 170-175, 2009.

[39] A.Y. Zomaya and Y.C. Lee, "*Energy Efficient Distributed Computing Systems*," Hoboken, New Jersey, John Wiley & Sons, 2012.

J. Octavio Gutierrez-Garcia received his PhD in Electrical Engineering and Computer Science from CINVESTAV and Grenoble Institute of Technology respectively. Dr. Gutierrez-Garcia is an associate professor in the Department of Computer Science at Instituto Tecnológico Autónomo de México. He has served as a reviewer for numerous international conferences and journals. His current research interests include Cloud computing, distributed artificial intelligence, service-oriented computing, and agent-based modeling.

Adrian Ramirez-Nafarrate is an Associate Professor in the Department of Industrial & Operations Engineering at the Instituto Tecnológico Autónomo de México. His research interests include modeling, simulation and optimization of service systems. He received a PhD degree in Industrial Engineering from Arizona State University, an MS in Manufacturing Systems from ITESM and a BS in Industrial Engineering from Universidad de Sonora.