2010 IEEE International Conference on Cluster Computing

Energy-aware Scheduling in Virtualized Datacenters

Íñigo Goiri, Ferran Julià, Ramón Nou, Josep Ll. Berral, Jordi Guitart and Jordi Torres
Universitat Politècnica de Catalunya and Barcelona Supercomputing Center
Jordi Girona 31, 08034 Barcelona, Spain
{igoiri,fjulia,rnou,berral,jguitart,torres}@ac.upc.edu

***Abstract*—The reduction of energy consumption in large-scale datacenters is being accomplished through an extensive use of virtualization, which enables the consolidation of multiple workloads in a smaller number of machines. Nevertheless, virtualization also incurs some additional overheads (e.g. virtual machine creation and migration) that can influence what is the best consolidated configuration, and thus, they must be taken into account. In this paper, we present a dynamic job scheduling policy for power-aware resource allocation in a virtualized datacenter. Our policy tries to consolidate workloads from separate machines into a smaller number of nodes, while fulfilling the amount of hardware resources needed to preserve the quality of service of each job. This allows turning off the spare servers, thus reducing the overall datacenter power consumption. As a novelty, this policy incorporates all the virtualization overheads in the decision process. In addition, our policy is prepared to consider other important parameters for a datacenter, such as reliability or dynamic SLA enforcement, in a synergistic way with power consumption. The introduced policy is evaluated comparing it against common policies in a simulated environment that accurately models HPC jobs execution in a virtualized datacenter including power consumption modeling and obtains a power consumption reduction of 15% with respect to typical policies.**

***Index Terms*—Virtualization, HPC, Energy, SLA, Scheduling**

I. Introduction

Energy-related costs have become a major economical factor for IT infrastructures and datacenters because of the power's price escalation. Companies are now focusing more than ever on the need to improve energy efficiency. A new challenge, namely the reduction of the carbon footprint, has appeared besides the energy cost due to many EU regulations and campaigns demanding greener businesses. Energy costs are rising, datacenter equipment is stressing power and cooling infrastructures, and the main issue is not the energy consumption of current datacenters but the fact that this consumption is increasing faster than any other.

The reduction of energy consumption in large-scale datacenters is being accomplished through an extensive use of virtualization. That not only derives in less consumption but in more flexible, secure and configurable systems. Virtualization is also the enabler technology to consolidate multiple workloads, thus reducing overall datacenter power consumption. Nevertheless, these consolidation strategies must also take into account additional parameters of utmost importance for datacenters, such as QoS, reliability, and global revenue.

The use of virtualization for consolidation, encapsulating jobs on virtual machines (VMs), has been already presented in different works. Starting from the results of some of these works, we propose a novel scheduling that addresses the important problem of data center machine management so as to meet application Service Level Agreements (SLAs) while at the same time, reducing power consumption. This job scheduling policy is able to consolidate workloads for reducing power consumption while preserving the tasks quality of service (QoS) agreed on the SLA, and taking into account virtualization overheads such as VM creation, checkpointing, and migration. Furthermore, the introduced scheduling policy is able to unify different provider's requirements in addition to power consumption, namely reliability and dynamic SLA enforcement (be able to recover from an SLA violation during the execution). Our policy decides what the best location for executing a new job is, depending on the resources it requires in order to fulfill its SLA and according with the provider's interests on power efficiency, reliability, etc. These restrictions are derived from the information of the system, including job execution and node status, and are stored in a score matrix. Furthermore, the proposed policy periodically calculates whether to move jobs in order to improve global system utility. For instance, this applies when more resources are needed in order to enforce an SLA that is being violated.

This paper states the whole proposed policy and evaluates a first implementation which includes virtualization overheads and power consumption. It is compared against common policies in a simulated environment that models a virtualized datacenter. This simulator mainly focuses on CPU and memory simulation and measures power consumption. This first proof of concept is based in HPC jobs and uses deadlines as QoS metric in order to define the SLA.

The remainder of the paper is organized as follows. Some related work and discussion of typical approaches is presented in Section II. Section III describes the internal architecture of our *Score-based Scheduler*. Section IV presents the evaluation environment including a detailed description of the simulation and Section V evaluates the presented approach. Finally, some conclusions and future work in Section VI.

II. Related work

Power management in cluster-based systems is an emerging topic in the resource management area. There are several works proposing energy management for servers that focus on applying energy optimization techniques in multiprocessor environments, such as [1] and [2]. Another proposal on load balancing for power and performance optimization in this kind of environment can be found in [3]. Economical approaches

978-0-7695-4220-1/10 $26.00 © 2010 IEEE
DOI 10.1109/CLUSTER.2010.15

are also used for managing shared server resources in e.g. [4], where authors use a greedy resource allocation algorithm that allows distributing a web workload among different servers assigned to each service. This technique demonstrates to reduce server energy usage by 29% or more for a typical Web workload. [5] proposes a hybrid datacenter architecture that mixes low power systems and high performance ones.

There have been several proposals into resource capacity planning and dynamic provisioning issues for QoS control (e.g. [6], [7], [8]). [9] states that new power saving policies, such as Dynamic Voltage/Frequency Scaling (DVFS), or turning off idle servers can increase hardware problems as well as the problem to meet SLAs in this reduced environment. Following this idea, we show how scheduling policies can take into account such problems. On the other hand, DVFS is one of the techniques that can be used to reduce the consumption of a server and minimize the total energy expenditure [10]. We rely on the node's underlying technology which automatically changes the frequency according to the load.

We propose adding smarter scheduling policies to dynamically turn off idle machines and reduce the overall consumption. Khargharia et al. [11] introduce a theoretical framework and methodology for autonomic power and performance management in e-business datacenters. They optimize the performance/watt by using a mathematically-rigorous optimization approach that minimizes wasted power while meeting performance constraints. Other optimization techniques have been applied for online scheduling algorithms, which has lead to the use of popular heuristic algorithms such as MET, Min-Min, Max-Min, OLB, or the fast greedy [12], [13]. Meta-heuristic algorithms such as Tabu search and Simulated Annealing have been also proposed [12], [14], [15].

The use of virtualization for consolidation is presented in [16], which proposes a dynamic configuration approach for power optimization in virtualized server clusters and outlines an algorithm to dynamically manage it. In addition, they developed a mixed integer programming (MIP) formulation to dynamically configure the consolidation of multiple services/applications in a virtualized server cluster [17]. The approach is power efficient centered and takes into account the cost of turning on/off the servers. However, it can lead to a too slow decision process for an online scheduler like the one we are interested in. In addition, this work is highly centered in HPC jobs while our proposal, although based in HPC is also extensible to heterogeneous applications.

We propose the use of VM for executing HPC applications taking into account virtualization overheads. Following the same idea, [18] aims to reduce virtualized datacenter power consumption by supporting VM migration and VM placement optimization while reducing the human intervention, they do not provide any evaluation of their solution. Finally, we state an extensible methodology that includes new trends such as fault tolerance and SLA enforcement. Fault tolerance in clusters has been researched in [19] where they use virtualization and design a reconfigurable distributed virtual machine (RDVM) infrastructure. Despite it is focused on failure management, they use a similar node selection approach taking into account nodes reliability and tasks deadlines. Nevertheless, this approach is not focused on the aggregation of other costs such as virtualization overheads.

Newer trends presented in [20] propose the usage of different data centers with distributed locations in order to distribute workload among those according to its power consumption and its source. Our framework can be applied to this model in order to give it a more detailed and precise vision.

III. Score-based Scheduling

There are many issues to be considered about VM scheduling in virtualized datacenters, e.g., power consumption, meet SLAs, reliability, etc. Focusing in the two first issues, our approach uses two different mechanisms in order to reduce the power consumption of a datacenter while preserving the different SLAs. One of the mechanisms that allows saving more power is turn off idle machines, which saves more than 200W in our testbed machines, and turn them on again if they are needed when a peak load occurs. A complementary mechanism, consolidation, is trying to execute all the VMs containing the users' tasks with the minimum amount of physical machines, but without degrading excessively the execution of these tasks. Therefore, scheduling takes a main role in order to achieve this power consumption reduction while maintaining QoS.

Here, several scheduling policies, such as Backfilling supporting migration, could be applied in order to assign new VMs in the system to available machines and redistribute VMs being executed in order to make some machines idle and then turn them off [21]. Nevertheless, scheduling policies using consolidation and turning on/off machines must also take into account additional parameters of utmost importance for datacenters, such as QoS, reliability, and global revenue. For this purpose, we propose a new power-aware scheduling technique that is able to take advantage of virtualization features while taking into account also its involved overheads (e.g. VM creation, migration, checkpointing). In addition, it is prepared to consider other parameters like node reliability or dynamic SLA enforcement. Based on all these parameters, our technique assigns different scores for each possible ⟨host,VM⟩ allocation and tries to find the best possible schedule.

A. Scheduling algorithm

Our scheduling policy consists of trying to find, at each change on the system status, the optimal combination of ⟨host,VM⟩ using as input information: the hardware and software requirements of the VM, the amount of resources required, the resources offered by the host machine (i.e. those that are available), the energy consumption of the machine, the user SLA constraints, and the reliability of the host. The policy is prepared to extend this list of parameters depending on the provider's interests. It gives each machine a dynamic score depending on this parameters and solves a dynamic optimization problem that assigns each VM to the best machine taking into account the different factors.

A scheduling round is started when a new VM enters in the system, finishes its execution, a violation in its SLA is detected, or the reliability of a node changes. Then, the best ⟨host,VM⟩ combination is found by creating a score matrix that maps all the tentative VM allocations by using the benefit of each VM being hold on each host machine. Each score indicates the reward (R) and the cost of holding a VM in a host by adding different penalties (P) such as migration, occupying an empty machine, SLA violations or any other needed constraint. At this point, a *hill climbing*-like search algorithm [22] finds the set of movements optimizing ⟨host,VM⟩ matrix. And finally, the system performs the set of operations decided by the new schedule (creation, migration...).

The score matrix is a $(M+1) \times N$ matrix, having N VMs and M machines plus a *virtual host* on the scheduler which holds the VMs requests in queue (all the new VMs and those which were running in a failed node and need to be scheduled again are maintained in a queue waiting to be allocated). Each value in the matrix represents the score (penalization) of hosting a VM in a specific host, including the costs involved due to virtualization, power consumption, reliability, and dynamic SLA enforcement. Those hosts that cannot hold a VM, due to insufficient free resources or hardware/software constraints have an ∞ value, and those which have the lowest value for a VM are supposed to be the most suitable. Then, we propose using a mathematical model to find a better combination of ⟨host,VM⟩ by choosing, at each step, the movement that improves the given requirements. When we consider to have found the best solution (if we cannot find any better one, or we have reached the maximum number of algorithm iterations), we apply the changes on the system by means of the actuators.

The *virtual host* acts as a queue where not allocated VMs are temporary scheduled and VMs entering in the system are held in that queue with infinite score, so the penalty of keeping them without real allocation is the maximum one. The operations with maximum benefit will be those involving the allocation of a new VM into a real host able to handle it.

As commented before, our scheduling policy is ready to be extended in order to support different provider's interests. Our current design considers hardware and software requirements, resource requirements to fulfill SLA, virtualization overhead (creation and migration costs), power consumption, dynamic SLA enforcement, and reliability. The first four parameters have been already implemented and tested in a real environment while the experimentation with the other two is part of our future work. Following subsections explain the different penalties and values applied for each parameter and constraint, and also the algorithm for matrix solving.

1) Hardware and software requirements: For each VM, we check if each host is able to hold it, by evaluating the hardware (it has the required system architecture, the required type and number of CPUs, etc.), the software (it has the capability to execute a given software, it uses a given hypervisor, e.g. Xen, KVM, ...). In case the host is not available to execute that VM, the score is set to infinity. So, the first penalty applied to the cost matrix discards all those ⟨host,VM⟩ combinations that are not viable.

$$P_{req}(h, vm) = \begin{cases} \infty & \text{if } h \text{ cannot fulfill } Req_{vm} \\ 0.0 & \text{otherwise} \end{cases}$$

2) Resource requirements: Once the hardware and software requirements have been checked, it is needed to ensure that the VM can get the amount of resources it requires to fulfill the SLA. For this reason, the scheduling policy checks if the occupation of every host after allocating a VM ($O(h, vm)$) is lower than 100%. This occupation is the most occupied resource on that host, calculated from the VMs hardware requirements in that machine. For example, if a host has a VM with a requirement of 10% of memory and 50% of CPU and another one with 65% and 30% respectively, then the global occupation would be 80% because of the CPU which is the most used resource.

$$\begin{aligned} O(h, vm) &= \text{occupation of } h \text{ allocating } vm \\ P_{res}(h, vm) &= \begin{cases} \infty & O(h, vm) > 1 \\ 0.0 & O(h, vm) \leq 1 \end{cases} \end{aligned}$$

In other words, it checks if the host will have enough resources to execute all the VMs in that host after adding this new one. Hence, in case the host is not able to allocate that VM, the score is set to infinity, disabling this allocation.

3) Virtualization overhead: One of the strengths of our proposal is its capability to deal with virtualization overheads. One is creation overhead, which is the time required to create and start a VM before it is ready to run tasks. The other is the migration overhead, which is the one incurred when moving a running VM between two different nodes. In the cost matrix, we calculate a creation time penalty for a new VM on each candidate host, and a migration time penalty for an allocated VM being moved to each candidate host. The migration penalties reduce the number of migration and so, prevents the same VM from moving too often. Furthermore, this migration penalty considers an estimation of the remaining execution time according to the user initial requirement. This is done for penalizing the migration of those VMs which remaining execution time is small; therefore they will finish soon and there is no need for migration.

Furthermore, in order to prevent possible VM migrations or other actions while a node is creating or already migrating the given VM, we set an infinity penalty while any action is being performed in a given VM.

$$\begin{aligned} C_c(h, vm) &= \text{cost of creating } vm \text{ in } h \\ C_m(h, vm) &= \text{cost of migrating } vm \text{ to } h \\ T_u(vm) &= vm \text{ execution time according to user} \\ t(vm) &= \text{time since } vm \text{ submission} \\ T_r(vm) &= vm \text{ remaining time according to user} \\ &= T_u(vm) - t(vm) \\ P_m(h, vm) &= \begin{cases} 2 \cdot C_m(h, vm) & T_r(vm) < C_m(h, vm) \\ \frac{C_m(h,vm)^2}{T_r(vm)} & T_r(vm) \geq C_m(h, vm) \end{cases} \end{aligned}$$

$$P_{virt}(h,vm) = \begin{cases} 0.0 & \text{if } vm \text{ is in } h \\ \infty & \text{if operation on } vm \\ C_c(h,vm) & \text{if } vm \text{ is new} \\ P_m(h,vm) & \text{otherwise} \end{cases}$$

Finally, one important factor when performing any action to a VM is the overhead produced in the node. In particular, performing more than one action at the same time can generate a race for the resources (e.g. disk, CPU) which will add an additional overhead. In order to consider this situation, we calculate a concurrency penalty for each host that indicates whether it is already creating or migrating a VM. This penalty is applied to those VMs which are not running in that host.

$$C_{conc}(h,vm) = \begin{cases} C_c(h,vm) & \text{if } h \text{ is creating } vm \\ C_m(h,vm) & \text{if } h \text{ is migrating } vm \\ 0.0 & \text{otherwise} \end{cases}$$

$$P_{conc}(h,vm) = \begin{cases} 0.0 & \text{if } vm \text{ is in } h \\ \sum_{vm \in h} C_{conc}(h,vm) & \text{otherwise} \end{cases}$$

4) Power efficiency: As we look for consolidating VMs in real hosts, we reward those operations that move a VM to a *fillable host*, those that are barely full but there have still room for another VM. This makes them more attractive to host VMs. On the other hand, those nodes that host few VMs and have many unused resources, are considered *emptiable hosts*, so we punish the movements to those hosts and VMs being hosted in them, since we want these VMs to move away and also avoid other VMs to get room in them.

$$\begin{array}{lcl} O(h,vm) & = & \text{occupation of } h \\ \#VM(h) & = & \text{number of } vm \text{ of } h \\ C_f & = & \text{benefit of filling a machine} \\ C_e & = & \text{cost of keeping a host under-used} \\ TH_{empty} & = & \#\text{VM empty threshold} \\ T_{empty}(h) & = & \begin{cases} 1 & \text{if } \#VM(h) \leq TH_{empty} \\ 0 & \text{otherwise} \end{cases} \\ P_{pwr}(h,vm) & = & T_{empty}(h) \cdot C_e - O(h,vm) \cdot C_f \end{array}$$

In the previous equations we can see how TH_{empty} defines when a machine should be considered to be mostly empty, therefore, when to penalize a machine with the empty cost. This cost, C_e, represents the cost of having these nodes with a lower number of VMs running, typically it can be set to the creation time in order to prevent running VMs in these nodes. On the other hand, C_f tries to compensate the migration cost and reward those nodes with big occupation in order to force the operation in nodes with bigger occupations. Finally, the aggregated formula, prevents migrating nodes with big usage ratios but small number of VMs.

5) Dynamic SLA enforcement: To ensure that SLAs of each VM are being accomplished during their execution, we add a maximum penalty (P_{SLA}) for each ⟨host,VM⟩ combination according to its SLA fulfillment ($SLA(h,vm)$). In addition, we increase the amount of needed resources for that VM if this is needed to preserve the SLA, so the VM will be rescheduled in another node with more available resources, compensating the SLA violation (whether it is possible). Furthermore, if increasing the VM resource requirements results on not fitting in some other hosts, these hosts get an infinity value.

$$\begin{array}{lcl} C_{sla}(vm) & = & \text{cost of breaking } vm \text{ SLA} \\ TH_{SLA} & = & \text{tolerance threshold} \\ SLA(h,vm) & = & \text{SLA fulfillment of } vm \text{ in host } h \\ P_{SLA}(h,vm) & = & \begin{cases} 0.0 & \text{if } SLA(h,vm) = 1 \\ C_{sla}(vm) & \text{if } TH_{SLA} < SLA(h,vm) < 1 \\ \infty & \text{if } SLA(h,vm) \leq TH_{SLA} \end{cases} \end{array}$$

6) Reliability: Each host has a given reliability factor ($F_{rel}(h)$) between 0 and 1 that represents the amount of time the node is up (1 implies it is always running). In order to support fault tolerance, we use the complement to the reliability factor ($1 - F_{rel}(h)$) as a probability of failure. Hence, the nodes which are always up will not have any penalty (P_{fault}), while the nodes that have a probability of being down during execution will. Also, some VMs have some tolerance or permissiveness to faults ($F_{tol}(vm)$), so a not strict VM in a host with some probability of failure will be less punished than a VM that requires a full uptime machine. Note that when a node fails, the VMs that were running there are moved to the *virtual host* to be rescheduled.

$$\begin{array}{lcl} C_{fail}(vm) & = & \text{cost of failing } vm \\ F_{rel}(h) & = & \text{reliability factor of } h \\ F_{tol}(vm) & = & \text{tolerance to fail factor of } vm \\ P_{fault}(h,vm) & = & ((1 - F_{rel}(h)) - F_{tol}(vm)) \cdot C_{fail}(vm) \end{array}$$

7) Merging scores: The final score for each cell is the result of the sum of all the individual scores (see next equation). This score merges the different constraints for each combination and it is incorporated to the final matrix. Therefore, each scheduling has a score according to different weighted parameters. Note that a high score means a high cost of maintaining a VM in that host.

$$Score(h,vm) = P_{req}(h,vm) + P_{res}(h,vm) + P_{virt}(h,vm)$$

$$P_{conc}(h,vm) + P_{pwr}(h,vm) + P_{SLA}(h,vm) + P_{fault}(h,vm)$$

Penalties which can take infinity value may make all the other penalties insignificant. For instance, if a node do not have the right architecture, this allocation will be impossible without taking into account the node occupation or its reliability.

B. Matrix solving

Once having the cost matrix, where each value represents the cost of hosting a specific VM_i in a given host H_j, the optimization process can start. First, we must subtract from each cell $\langle H_j, VM_i \rangle$ the cost of maintaining VM_i in the current host (i.e. this is the value of cell $\langle H_{cur}, VM_i \rangle$ if VM_i is running in H_{cur}). After this, we obtain for each cell the difference (improvement or degradation) of moving a VM from its current host to the host corresponding to this cell. Positive scores mean degradation and negative scores mean improvement, as each score is equivalent to the penalization of the ⟨host,VM⟩ combination. For example, after calculating the previously presented penalties, we could obtain the following

score matrix, where VM_1 is currently running in H_M, VM_2 in H_3, VM_3 in H_5, VM_4 in H_1, and VM_N in H_6:

	VM_1	VM_2	VM_3	VM_4	...	VM_N
H_1	15.2	15.2	∞	15.2		10
H_2	∞	7.8	7.8	7.8		∞
H_3	10.3	10.3	∞	10.3		10.5
...					$\ddots$	
H_M	11.0	∞	11.0	11.0		∞
H_V	∞	∞	∞	∞		∞

Starting from this initial cost matrix, we must the center column VM values according to the current host of each VM (i.e. subtracting from each cell the cost of maintaining the VM in its current host), obtaining the following matrix:

	VM_1	VM_2	VM_3	VM_4	...	VM_N
H_1	4.2	4.9	∞	0.0		9.5
H_2	∞	-2.5	-0.9	-7.4		∞
H_3	-0.7	0.0	∞	-4.9		10.0
...					$\ddots$	
H_M	0.0	∞	2.3	-4.2		∞
H_V	∞	∞	∞	∞		∞

Having the cost matrix preprocessed, we can start optimizing it by selecting on each iteration the smallest value of the matrix representing the best movement to be done in all the system. We move the corresponding VM to the corresponding new host machine, and we refresh the column values, the old host and new host row values, having the cost matrix updated. The main idea is to iterate until there cost matrix has no negative values When the matrix reaches a state where all values are positive (no improvements can be done) or the number of movements has reached a given limit, we consider to have found a suboptimal solution for the current system configuration. Algorithm 1 shows the matrix optimization algorithm.

Algorithm 1 Algorithm for Allocation Matrix Optimization

```
CM    := Cost Matrix [hosts][VMs];

- Fill CM with values and penalties;

While CM has negative values do:
    <h,v>  := smallest position on CM;
    o      := current host for v;

    - Re-schedule VM v from Host o to Host h;
    - Recalculate CM values;

    If (iterations limit reached) then:
        break;
    End If
End While
```

Note that Hill Climbing algorithm is greedy, but in this situation it finds a suboptimal solution much faster and cheaper than evaluating all possible configurations. Each step brings to a more optimal configuration until there are no better configurations or an iteration limit is reached. In our study case, some of the constraints help to reduce the search space, i.e. the resource requirement constraint discards a great amount of ⟨host,VM⟩ combinations at the beginning of the algorithm.

The algorithm complexity has an upper boundary of $O(\#Hosts \cdot \#VMs) \cdot C$ since it iterates over the ⟨host,VM⟩ matrix C times. In addition, as not all the hosts are fillable or emptiable, and the task requirements, the algorithm can skip many hosts and reduce the number of hosts to visit.

C. Actuators

Once the scheduling policy has decided the host allocation for each VM, changes needs to be performed in the system using the facilities that virtualization offers, such as VM creation and migration. In case the VM has never been running in the system, the scheduler invokes the selected node to create this VM. If a given VM has been moved from a node to another, the scheduler asks the current executing node to migrate it to its new location. Furthermore, if the VM was running in a failed node, the new executing node tries to recover it from the more recent checkpoint, and if there is not available checkpoint, it recreates the VM.

In addition to scheduling itself, one of the key decisions is determining the amount of operative nodes or in other words, when a node can be turned off in order to save power consumption, or turned on again in order to be used to fulfill the tasks SLAs. This decision is driven by the final score matrix and two thresholds: the minimum *Working hosts* threshold λ_{min} and the maximum *Working hosts* threshold λ_{max}. When the ratio of working nodes goes over λ_{max}, the scheduler must start turning on stopped nodes. The nodes to be turned on are selected according to a number of parameters, including its reliability, boot time, etc. On the other hand, when the ratio of working nodes goes below λ_{min}, the scheduler can start turning nodes off. The scheduler selects those idle machines according to the score of the host in the matrix. This score results from the aggregation of scores (matrix row) and taking into account the number of infinity scores. Those nodes with a higher score are selected to be turned off. Finally, in order to define a minimum set of operative machines, the scheduler can use the $\min_{exec}$ parameter.

IV. Power-aware Simulator

The development of scheduling techniques for power saving in large-scale clusters raises different handicaps, some related with the decision making and others related with the testing and validation process. The latter requires the use of large amount of machine power and time that sometimes makes the process itself not feasible. For this reason, in such cases it is usual to use simulators that behave in the same way than the real infrastructure, but saving lots of time, power, and effort, while maintaining the levels of stringency in the results [23].

We have developed a framework for evaluating the power efficiency of a datacenter executing a given workload. With this framework, we can evaluate the effectiveness in terms of power consumption while making the scheduling techniques able to take advantage of different capabilities such as migration of tasks between nodes, dynamic turn on/off, and consolidation. A first version of this simulation framework has

been presented in [24]. The simulator has been now improved adding virtualization support.

The use of virtualization provides more flexibility to the datacenters but complicates the design of a simulator. Managing VMs has overheads that must be simulated and they are different depending on the real hardware. Our simulator is based on the real Cloud middleware described in [25]. This middleware has also checkpointing and caching capabilities, with low contribution to power consumption, and for this reason, they have not been simulated for this paper. However, the creation and migration overheads, which are the most relevant for power efficiency, are implemented in the simulator.

Our simulator is based on the OMNet++ [26] framework, which is used to build an event-driven simulation where we control and replicate the behavior of the several layers in a node (from the PowerSupply to the Operating System Scheduler in the virtual host). It has been designed following the procedure in [27], but focused on the energy consumption and on the CPU power scheduling among VMs. The simulator loads a workload trace, simulates the execution on several machines with different configuration each one and generates the output results using different global scheduling policies. In addition, these machines can be dynamically turned on/off. The simulation takes into account both the physical machine boot time and power consumption and the VM creation and migration power consumption.

As there is a lack of real traces for cloud, we use slightly modified real Grid traces. They contain the arrival sequence of the jobs and their characteristics, mainly the CPU and memory consumption. As we are interested in large-scale workloads, the simulator does not try to simulate the exact execution times; this would not be usable and would take too long to get the results reducing the benefits of simulation. The simulator is designed in a way that the time scale can be accelerated with enough precision thus we can simulate a large virtualized datacenter executing a workload for a week using one machine during an hour approximately.

The architecture proposed for this framework is splitted in three parts: *Workload Generator*, *Scheduler* and *VHost*. The *Workload Generator* loads the trace information and simulates the job arrival with its properties into the *Scheduler*. The *Scheduler* is a "real" part in our simulator, it is not simulated. We have several scheduling policies that take into account consolidation, SLA fulfillment, etc. The *Scheduler* decides the job location and the result of this decision process is sent to the *VHost*, which simulates the execution and generates the result traces with all the execution information.

We have followed the standard development cycle for designing the simulator. Firstly, different applications with different typologies and profiles are executed (on a real, not simulated, machine) and their resource usage and power consumption are monitored. We also have executed different number of VMs with different configuration parameters to model both the power consumption and the behavior of the Xen HyperScheduler. In this case, as we use Xen [28] as our virtualization platform, we have developed a simulator based on its behavior including characteristics like Virtual Machine Weights and Capabilities. Using this information, we have built a model which is then used to simulate a virtualized datacenter with many "identical" machines. Validations are applied to refine the model and the simulator. Finally, the simulator is executed to provide the experimental data described here.

The machines that are simulated are identical in behavior but not in characteristics. We can simulate machines with different amounts of CPUs or memory, which in turn will be able to run a different number of VMs. The internal resource scheduling follows the same algorithm for all the machines, the Xen's resource scheduler. We also have measured the CPU overload that is produced when creating new VMs or at migration time and this has been implemented into the simulator.

We have also included some statistical distributions to mimic variability found in the real measures, i.e. a normal distribution (μ 40, σ 2.5), as observed in the real environment, has been used in VM creations. Including more complex distributions for other behaviors may produce more accurate results. Nonetheless, we obtain satisfactory results with the actual distributions as seen in subsection IV-B.

A. Power modeling

The first stage of the development of our simulator requires measuring the power consumption behavior in a real computer. In our case, we used a 4-way machine, with some energy-efficient policies in its kernel, executing different workloads and with different VM configurations. Our aim was to discover how Xen really manages the CPU and how its energy-efficient policies perform under very different configurations.

The power consumption of the machine was gathered using digital methods. In the past, analogical methods via oscilloscope where used [29]. The resolution of the measurements is below 0.1 Watts with a measured latency of 1 second. Our measurements show that there is a minimum consumption when the machine is turned on. This is the idle machine consumption (230W in our machines), which corresponds to a machine with none VMs hosted on it and executing nothing. We measured the power consumption having different number of VMs in the host and changing also the CPU consumption that each VM did. The results, which are displayed in Table I indicating the number of virtual CPUs used by each VM (the + sign indicates more than one VM at the host), the average CPU consumed by each VM, and the measured power for each configuration, show that there is no dependence in the number of VMs and in how they are configured. The only real dependence is with the total CPU consumed by the VMs.

#VCPUs	%CPU	Power	#VCPUs	%CPU	Power
1	100%	259 W	1+1	$2 \times 100\%$	273 W
2	200%	273 W	1+2	$100\% + 200\%$	291 W
3	300%	291 W	1+1+1+1	$4 \times 100\%$	304 W
4	400%	304 W	1+1+1+1	$4 \times 0\%$	230 W

TABLE I
VIRTUALIZED SERVER POWER USAGE

Although the measured values do not take care of disk usage, in the future we should include different jobs with I/O load that will obtain benefits of a hard disk power usage modeling. Furthermore, there are some other machines where the power usage does not change with the load and it is always constant. These machines should be avoided because no wattage reduction can be obtained even adding energy efficiency capabilities to the kernel or applying DVFS. Idle wattage level should be decreased in the industry as it is one of the most used states and it is not energy efficient [30].

B. Model validation

The simulator validation is done by comparing real results with simulated ones executed with the same workload and in the same environment. In particular, the validation process consists of comparing CPU usage and power consumption in a node that executes a 1300 seconds workload that is composed by seven different tasks that explore the most typical situations we can have in a real cloud execution. Figure 1 shows the power usage in both cases: real and simulated. The total power consumption for the execution in the real environment is 99.9 ± 1.8 Wh, while the simulator predicts a total power consumption of 97.5 Wh, which means an underestimation of 2.4%. We can also calculate the instantaneous error, which is 8.62 with a standard deviation of 8.06 W. Notice that for us is important to have more total power consumption accuracy than instantaneous one. We can see in the validation graphic that executions have not the exact sequence in power load, this is because the simulator does not imitate the global behavior of the real framework and that leads to quite different instantaneous behaviors. That difference is translated into a difference in the instantaneous CPU consumption, and is for that reason why the consumption can be slightly different during the execution but the total results are almost equal. With these results we can conclude that our simulator has been validated, however there is place to decrease the instantaneous error increasing the simulation cost or introducing newer components into the simulation as separate disk usage simulation.

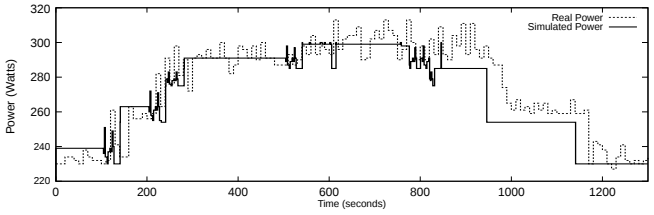

Fig. 1. Simulator validation

Future work may include fine-tuning and a more detailed validation, as some smaller overheads and other issues that can show in the real world should be modeled and introduced in the simulator. As for example the use of transactional jobs or the simulation of checkpointing and caching process inside the data-center or detailed disk usage as mentioned before.

V. Evaluation

In this section, we evaluate a first version of our proposal against different common scheduling techniques, comparing their energy efficiency and demonstrating that our policy is able to manage virtualization overhead better than typical proposals. The experiments consist of the simulation of a whole virtualized datacenter with 100 nodes. The datacenter is configured to have three different types of nodes according to their virtualization overheads. There are 15 fast nodes, with C_c = 30s and C_m = 40s, 50 medium nodes with C_c = 40s and C_m = 60s, and 35 slow nodes with C_c = 60s and C_m = 80s. Taking into account these speeds for VM creation and migration, our policy is set up theoretically with medium values. From these values, we derive $TH_{empty} = 1$, $C_{empty} = 20$, and $C_{fill} = 40$. The first one starts penalizing if the node has only one VM or none; the second one represents the cost of having an empty node with few VMs; the last cost rewards those nodes with big occupation and prevents migrating nodes with big usage ratios but small number of VMs.

The presented approach intends to take benefit of consolidation in large virtualized datacenters executing HPC jobs. For this reason, the evaluation of our proposal has been performed using a Grid Workload, which has been obtained from Grid5000 [31] on the week that starts on Monday first of October of 2007. The set of policies is evaluated according to different metrics including number of used nodes, CPU usage, power consumption, and SLA fulfillment. On the one hand, the consolidation of the system is reflected in the average number of working (those which are executing a VM) and online nodes (those which are turned on) and the power consumption.

On the other hand, in order to measure the QoS, we use the typical metric to define an SLA in a HPC environment: deadline accomplishment, which shows the client satisfaction (S). We will define this satisfaction as a percentage between 0 and 100%, 100% if job execution time (T_{exec}) is less than the agreed deadline (T_{dead}) and 0% if completing the task takes longer than twice this time. This is defined by this equation:

$$S = \begin{cases} 100 & \text{if } T_{exec} < T_{dead} \\ 100 \cdot max\{1 - \frac{T_{exec} - T_{dead}}{T_{dead}}, 0\} & \text{if } T_{exec} \geq T_{dead} \end{cases}$$

Therefore, having higher percentages will mean the applications are less delayed because of the virtualization overheads, the scheduling or the contention of too loaded machines. Hence, lower execution times will mean more client satisfaction. In addition, we will also use this execution time delay ($delay$) in our results in order to measure the quality of scheduling policies.

In this experimentation, we have setup the deadlines for the Grid5000 jobs multiplying their execution times in a dedicated machine by a factor between 1.2 and 2 depending on the job and user typology. For example, a job with a factor of 1.5 that takes 100 minutes in a dedicated environment will have a deadline of 150 minutes. Therefore, if it would take more than 300 minutes to be finished, it would have a client satisfaction of 0% and a $delay$ of 200%.

Finally, in this work we focus our scheduling policies in the hardware/software requirements, resource requirements, virtualization costs, and power factors. Dynamic SLA enforcement, reliability, and revenue factors are not included in the experimentation at this moment, but they will be introduced in next works evaluating the capability of predicting SLAs from a ⟨host,VM⟩ combination, introducing an environment with failures, and entering more in detail with revenue elements.

A. Power consumption vs. SLA fulfillment trade-off

As it has been already presented, consolidation is applied in order to be able to get idle nodes to be turned off. Nevertheless, a too aggressive node turning off policy will incur in not enough resources to execute tasks while a passive one will have bigger power consumption. This trade-off depends on the λ_{min} and λ_{max} thresholds.

The effect of these two thresholds has been tested by executing the Grid workload on top of the simulated datacenter using the score-based policy, which is the one that makes a more aggressive consolidation. This allows evaluating the influence of the turning on/off thresholds by showing the client satisfaction and the power consumption respectively.

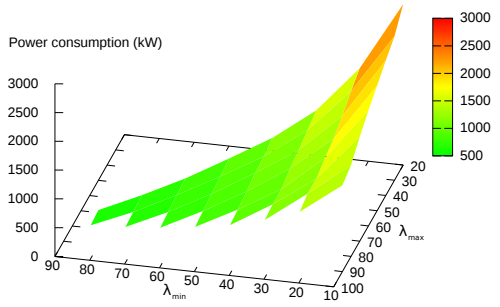

Fig. 2. Power consumption using different turn on/off thresholds

Figure 2 shows that waiting the nodes to reach a high utilization before adding new nodes (high λ_{max}) makes the power consumption smaller. In the same manner, the earlier the system shutdowns a machine (high λ_{min}), the smaller the power consumption is. It demonstrates how turning on and off machines in a dynamic way can be used to dramatically increase the energy efficiency in a consolidated datacenter.

On the other hand, client satisfaction decreases, as shown in Figure 3, when the turn on/off mechanism is more aggressive and it shuts down more machines (in order to increase energy efficiency). Therefore, this is a trade-off between the fulfillment of the SLAs and the reduction of the power consumption, whose resolution will eventually depend on the provider's interests. For instance, if the provider is having a high client satisfaction, it could decide to reduce it slightly in order to allow for a greater power reduction (by shutting down more nodes). In addition, this experiment has shown the capability of the presented scheduling policy to consolidate the load and reduce the amount of used nodes.

Fortunately, average threshold values give a balanced trade-off between energy and QoS. Experimentally, we have found

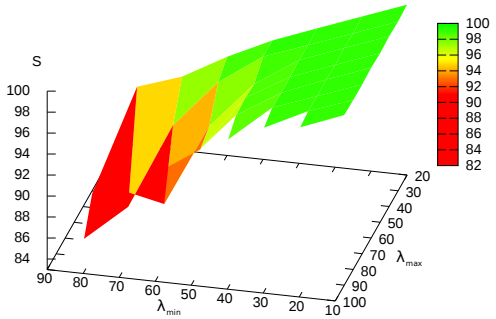

Fig. 3. Client satisfaction using different turn on/off thresholds

that our environment best values are λ_{min} = 30% and λ_{max} = 90% to ensure almost complete fulfillment of the SLAs while getting substantial power reduction. A next step would be to dynamically adjust these thresholds, which is part of our future work. However, these static values will be used in these experiments.

B. Static allocation

Once we have set up the parameters to turn on/off nodes according with the number of loaded nodes (λ_{min} = 30% and λ_{max} = 90%), we compare the energy efficiency and SLA fulfillment of four static scheduling algorithms which do not use migration; *Random* (RD), which assigns the tasks randomly; *Round Robin* (RR), which assigns a task to each available node, which implies a maximization of the amount of resources to a task but also a sparse usage of the resources; *Backfilling* (BF), which tries to fill as much as possible the nodes, thus solving the former problem; and finally, a basic version of our policy (SB0) which just takes into account hardware and software requirements P_{req}, the resource requirements P_{res}, and power efficiency P_{pwr}. In addition, we have set up our policy to not perform migration in order to establish a fair comparison with the other static policies.

	Work/ON	CPU (h)	Pwr (kW)	S (%)	$delay$ (%)
RD	24.3 / 41.7	14597.2	1952.1	33.2	474.5
RR	23.5 / 51.9	11844.2	2321.0	60.4	338.4
BF	10.1 / 22.2	6055.3	1007.3	98.0	10.4
SB0	9.9 / 22.4	6055.3	1016.3	98.2	10.4

TABLE II
SCHEDULING RESULTS OF POLICIES WITHOUT MIGRATION

The results are presented in Table II, showing the power consumption (Pwr) and different metrics such as the average number of nodes that are actually working ($Work$), the average number of nodes running (ON), the client satisfaction (S) and the $delay$. It shows that non-consolidating policies such as *Random* and *Round-Robin* give poor energy efficiency while violating a significant amount of SLAs: they give the worst results on both criteria. *Backfilling* gets better SLA fulfillment with substantially lower cost as it uses fewer nodes. Finally, our proposal, which works with no penalties on virtualization overheads, behaves very similar to the *Backfilling* policy.

C. Impact of virtualization overheads

Using the non-migrative approach, we have tested our policy with different configurations (SB1 = SB0 + P_{virt}, SB2 = SB1 + P_{conc}) in order to test the impact of considering virtualization overheads (creation and concurrency). Table III shows that SB1, which adds VM creation overheads, makes a better use of the resources because it takes into account the time to create VMs and selects better nodes to perform that. In addition, it gets worse SLA fulfillment which is solved by the SB2 by taking care about concurrency overheads which also causes a small increment on the power consumption. This is because of considering the cost of concurrent creation of VMs reduces the consolidation ratio but gets better SLA fulfillment since it produces faster VM creation.

	λ	Work/ON	CPU	Pwr	S	$delay$
SB0	30-90	9.85 / 22.4	6055.3	1016.3	98.2	10.4
SB1	30-90	10.2 / 22.2	6055.3	1006.7	97.9	10.7
SB2	30-90	10.2 / 23.0	6068.5	1038.5	99.2	8.8
SB2	40-90	10.4 / 19.0	6055.1	880.5	98.1	10.2

TABLE III
SCHEDULING RESULTS OF SCORE-BASED POLICIES WITHOUT MIGRATION

Even though it implies a power consumption increment regarding the basic configuration, the client satisfaction has been increased and it allows the provider making a more aggressive turn on/off policy what derives in higher consolidation and lower power consumption. Following the idea presented in Section V-A, we tweaked the lambda parameters in order to get the same 98% SLA fulfillment, as it is got in the basic configuration. Using $\lambda_{min} = 40\%$ and $\lambda_{max} = 90\%$ we get the results in the last row of Table III, which show a reduction of more than 12% regards to the *Backfilling* policy while getting a similar SLA fulfillment.

D. Impact of migration

This section demonstrates the behavior of our scheduler when introducing the capability to migrate VMs in order to get a better consolidation. Table IV shows the results of *Dynamic Backfilling* (DBF) policy, which applies *Backfilling* and migrates VMs between nodes in order to provide a higher consolidation level, and our score-based proposal (SB) using all the penalties and including the migration capability.

	λ	Work/ON	CPU	Pwr	S	$delay$	Mig
DBF	30-90	9.7 / 21.3	6056.0	970.6	98.1	12.9	124
SB	30-90	9.7 / 21.0	6055.8	956.4	99.1	9.0	87
SB	40-90	9.7 / 18.3	6055.8	850.2	98.4	9.9	87

TABLE IV
SCHEDULING RESULTS OF POLICIES WITH MIGRATION

Results for DBF show a small improvement in power efficiency with respect to non-migration variation while getting much better consolidation. This is caused by the overhead introduced by migrating VMs. In addition, the SLA fulfillment is maintained in a medium level as in the non-migration approach. On the other hand, our SB policy takes virtualization overheads like creation and migration into account, which makes it getting more client satisfaction.

As we did in the previous experiment, in order to give a measure of what is the improvement in client satisfaction terms, we set a similar SLA fulfillment target for DBF and the best of the score-based configurations, which is $C_{empty} = 20$ and $C_{fill} = 40$, and we set more aggressive turn on/off parameters of $\lambda_{min} = 40\%$ and $\lambda_{max} = 90\%$. Using this configuration we get a reduction in the datacenter power consumption of 15% with regard to *Backfilling* and 12% compared with the dynamic variant. These experiments demonstrate how our proposal gets the best power consumption and SLA fulfillment as it takes into account the migration overheads.

E. Impact of consolidation parameters

One of the advantages of our policy is it can be easily configured according to the provider's requirements. In this experiment, we show some variants of our policy: without penalizing empty hosts ($C_{empty} = 0$), using typical parameters ($C_{empty} = 20$ and $C_{fill} = 40$), and using more aggressive parameters for consolidation.

Table V shows the results of this parameter tweaking. The first variant does not penalize empty hosts ($C_{empty} = 0$), which implies lower consolidation and worst power performance. In fact, the first one does not migrate any VM since the fillable reward is not worthwhile. The second variant uses the values used in previous experiments, which include the empty host penalization. It gets better consolidation while maintaining similar client satisfaction as it performs an accurate number of migrations. Finally, the third variant has been set up with aggressive parameters, getting the best consolidation in terms of working nodes, but getting poor energy efficiency and lower SLA fulfillment, which is mainly because it rewards the occupation and penalties too much empty hosts which implies a big amount of migrations. This experiment also demonstrates how our proposal can also be set up to favor different interests according to the provider's features and requirements. Nevertheless, future work will include an automatic setting according with economical parameters.

C_e	C_f	Work/ON	CPU	Pwr	S	$delay$	Mig
0	40	10.4 / 22.9	6055.2	1036.4	99.3	8.6	0
20	40	9.7 / 21.0	6055.8	956.4	99.1	9.0	87
60	100	9.3 / 22.0	6057.8	998.8	97.7	11.2	432

TABLE V
SCORE-BASED SCHEDULING RESULTS WITH DIFFERENT COSTS

VI. CONCLUSIONS

In order to obtain good energy efficiency the key issue is consolidating. Nevertheless, this is not the only factor scheduling policies have to take into account; it also has to care about other parameters such as QoS, revenue and virtualization overheads. For this reason, in this paper we have presented a new scheduling policy that takes advantage of virtualization

in order to consolidate multiple workloads, keeping in mind performance and power consumption simultaneously.

The matrix formulation of the problem is intuitive, captures many important considerations in managing a virtualized data center, and lends itself easily to extension. In addition, the results obtained in this paper shows a power consumption reduction of 15% with respect to typical policies using our new scheduling policy which is based on consolidation in order to decide the movements and operations to be done within scheduling functions. In addition, it has demonstrated this policy deals better with virtualization costs and our experiments, performed using real workloads, exemplify that these techniques can offer substantial improvements in energy and performance efficiency in these scenarios. The experiments using the Grid workload demonstrates how non-consolidation aware policies give poor energy efficiency.

Our future work will focus on extending the proposed policy by evaluating parameters such as SLA enforcement or reliability in a real scenario. Furthermore, new enhancements to the scheduling policy such as dynamic thresholds or economical decision making will be included, as well as, the capabilities to provide checkpointing management and extended SLA management.

ACKNOWLEDGMENT

This work is supported by the Ministry of Science and Technology of Spain under contracts AP2008-0264, TIN2007-60625, and TIN2008-06582-C03-01, and by the Generalitat de Catalunya under contracts 2009-SGR-980 and 2009-SGR-1428.

REFERENCES

[1] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. Keller, "Energy Management for Commercial Servers," *Computer*, vol. 36, no. 12, pp. 39–48, 2003.

[2] R. Bianchini and R. Rajaniony, "Power and Energy Management for Server Systems," *Computer*, vol. 37, no. 11, pp. 68–76, 2004.

[3] E. Pinheiro, R. Bianchini, E. Carrera, and T. Heath, "Load Balancing and Unbalancing for Power and Performance in Cluster-based Systems," in *Workshop on Compilers and Operating Systems for Low Power, September 9, 2001, Barcelona, Spain*, vol. 180, 2001, pp. 182–195.

[4] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle, "Managing Energy and Server Resources in Hosting Centers," *ACM SIGOPS Operating Systems Review*, vol. 35, no. 5, pp. 103–116, 2001.

[5] B. Chun, G. Iannaccone, G. Iannaccone, R. Katz, L. Gunho, and L. Niccolini, "An Energy Case for Hybrid Datacenters," *Workshop on Power Aware Computing and Systems (HotPower'09), October 10, Big Sky, MT, USA*, 2009.

[6] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar, S. Krishnakumar, D. Pazel, J. Pershing, and B. Rochwerger, "Oceano - SLA based management of a computing utility," in *7th IFIP/IEEE International Symposium on Integrated Network Management, May 14–18, 2001, Seattle, USA*, 2001.

[7] K. Shen, H. Tang, T. Yang, and L. Chu, "Integrated Resource Management for Cluster-based Internet Services," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 225–238, 2002.

[8] S. Ranjan, J. Rolia, H. Fu, and E. Knightly, "Qos-driven Server Migration for Internet Data Centers," in *10th International Workshop on Quality of Service (IWQoS), Miami, USA, May 15–17*, 2002, pp. 3–12.

[9] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam, "Managing Server Energy and Operational Costs in Hosting Centers," *ACM SIGMETRICS Performance Evaluation Review*, vol. 33, no. 1, pp. 303–314, 2005.

[10] Y. Lee and A. Zomaya, "Minimizing Energy Consumption for Precedence-Constrained Applications Using Dynamic Voltage Scaling," in *9th IEEE/ACM International Symposium on Cluster Computing and the Grid, Shanghai, China, May 18-21*, 2009, pp. 92–99.

[11] B. Khargharia, S. Hariri, and M. Yousif, "Autonomic Power and Performance Management for Computing Systems," *Cluster Computing*, vol. 11, no. 2, pp. 167–181, 2008.

[12] T. Braun, H. Siegel, N. Beck, L. Bölöni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, B. Yao, D. Hensgen *et al.*, "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems," *Journal of Parallel and Distributed Computing*, vol. 61, no. 6, pp. 810–837, 2001.

[13] R. Armstrong, D. Hensgen, and T. Kidd, "The Relative Performance of Various Mapping Algorithms is Independent of Sizable Variances in Run-time Predictions," in *7th IEEE Heterogeneous Computing Workshop (HCW'98), March 30, 1998, Orlando, FL, USA*, 1998.

[14] A. Abraham, R. Buyya, and B. Nath, "Nature's Heuristics for Scheduling Jobs on Computational Grids," in *8th International Conference on Advanced Computing and Communications, December 14-16, Cochin, India*, 2000.

[15] M. Mika, G. Waligora, and J. Weglarz, "A Metaheuristic Approach to Scheduling Workflow Jobs on a Grid," pp. 295–318, 2004.

[16] V. Petrucci, O. Loques, B. Niteroi, and D. Mossé, "Dynamic Configuration Support for Power-aware Virtualized Server Clusters," in *21th Conference on Real-Time Systems, Dublin, Ireland, July 1–3*, 2009.

[17] V. Petrucci, O. Loques, and D. Mossé, "A Dynamic Configuration Model for Power-efficient Virtualized Server Clusters," in *11th Brazillian Workshop on Real-Time and Embedded Systems (WTR), Recife, Brazil, May 25, 2009*, 2009.

[18] L. Liu, H. Wang, X. Liu, X. Jin, W. He, Q. Wang, and Y. Chen, "GreenCloud: a New Architecture for Green Data Center," in *6th International Conference on Autonomic Computing and Communications, Industry Session, Barcelona, Spain, June 15–19, 2009*, 2009, pp. 29–38.

[19] S. Fu, "Failure-Aware Construction and Reconfiguration of Distributed Virtual Machines for High Availability Computing," in *9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2009, pp. 372–379.

[20] K. Le, R. Bianchini, M. Martonosi, and T. Nguyen, "Cost-and Energy-Aware Load Distribution Across Data Centers," *Workshop on Power Aware Computing and Systems (HotPower'09), October 10, Big Sky, MT, USA*, 2009.

[21] V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour, "Evaluation of Job-scheduling Strategies for Grid Computing," *1st IEEE/ACM International Workshop on Grid Computing, Bangalore, India, December 17, 2000*, pp. 191–202, 2000.

[22] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.

[23] D. Menasce, M. Bennani, and H. Ruan, "On the Use of Online Analytic Performance Models in Self-Managing and Self-Organizing Computer Systems," *Self-star Properties in Complex Information Systems*, vol. 3460, pp. 128–142.

[24] J. Berral, I. Goiri, R. Nou, F. Julia, J. Guitart, R. Gavalda, and J. Torres, "Towards energy-aware scheduling in data centers using machine learning," *1st International Conference on Energy-Efficient Computing and Networking (eEnergy'10)*, 2010.

[25] I. Goiri, J. Guitart, and J. Torres, "Elastic Management of Tasks in Virtualized Environments," in *XX Jornadas de Paralelismo (JP 2009), A Coruña, Spain, September 16–18, 2009*, 2009, pp. 671–676.

[26] "Omnet," 2009, http://www.omnet.org.

[27] R. Nou, S. Kounev, F. Julià, and J. Torres, "Autonomic QoS Control in Enterprise Grid Environments using Online Simulation," *Journal on Systems Software*, vol. 82, no. 3, pp. 486–502, 2009.

[28] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer, "Xen and the Art of Virtualization," in *19th ACM Symposium on Operating Systems Principles, Bolton Landing, NY, USA, October 19–22, 2003*, 2003, pp. 164–177.

[29] R. Nou, "Energy Efficiency: A Case Study," Technical University of Catalonia (UPC) - Computer Architecture Department, Tech. Rep. UPC-DAC-RR-CAP-2009-14, 2009.

[30] L. Barroso and U. Hölzle, "The Case for Energy-Proportional Computing," *Computer*, vol. 40, no. 12, pp. 33–37, 2007.

[31] "The Grid Workloads Archive," 2009, http://gwa.ewi.tudelft.nl.