HAL open science

DVFS Aware CPU Credit Enforcement in a Virtualized System

Daniel Hagimont, Christine Mayap Kamga, Laurent Broto, Alain Tchana, Noel de Palma

▶ **To cite this version:**

Daniel Hagimont, Christine Mayap Kamga, Laurent Broto, Alain Tchana, Noel de Palma. DVFS Aware CPU Credit Enforcement in a Virtualized System. 14th International Middleware Conference (Middleware), Dec 2013, Beijing, China. pp.123-142, 10.1007/978-3-642-45065-5_7 . hal-01151030v2

HAL Id: hal-01151030

https://inria.hal.science/hal-01151030v2

Submitted on 2 Mar 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

DVFS aware CPU credit enforcement in a virtualized system

D. Hagimont[1], C. Mayap Kamga[1], L. Broto[1], A. Tchana[2], and N. De Palma[2]

[1] University of Toulouse,
first.last@enseeiht.fr
[2] Joseph Fourier University,
first.last@imag.fr

Abstract. Nowadays, virtualization is present in almost all computing infrastructures. Thanks to VM migration and server consolidation, virtualization helps reducing power consumption in distributed environments. On another side, Dynamic Voltage and Frequency Scaling (DVFS) allows servers to dynamically modify the processor frequency (according to the CPU load) in order to achieve less energy consumption. We observed that these two techniques have several incompatibilities. For instance, if two virtual machines VM1 and VM2 are running on the same physical host (with their respective allocated credits), VM1 being overloaded and VM2 being underloaded, the host may be globally underloaded leading to a reduction of the processor frequency, which would penalize VM1 even if VM1's owner booked a given CPU capacity. In this paper, we analyze the compatibility of available VM schedulers with DVFS management in virtualized environments, we identify key issues and finally propose a DVFS aware VM scheduler which addresses these issues. We implemented and evaluated our prototype in the Xen virtualized environment.

Keywords: DVFS, virtual machines, scheduling

1 Introduction

Nowadays, many organizations tend to outsource the management of their physical infrastructure to hosting centers. They subscribe for a quality of service (QoS) and expect providers to fully meet it. By acting this way, companies aim at reducing their costs by paying only for what they really need. The providers, instead, are interested in saving resources while guaranteeing customers QoS requirements.

On the provider side, virtualization was introduced in order to facilitate resource management. Virtualization is a software-based solution for building and running simultaneously several operating systems (called *guest OS* or *Virtual Machine*) on top of an underlying OS (called *host OS* or *hypervisor*). In hosting centers, virtualization is a means to implement server consolidation. Indeed, servers being underutilized most of the time (below 30% of processor utilization [17]), VM (Virtual Machine) migration helps achieving better server utilization by migrating VMs on a minimal set of machines, and switching unused machines off in order to save energy.

However, powerful computers with high processor frequency, multiple cores and multiple CPUs are an important factor contributing to the continuous increase of energy consumption in computing infrastructures. To reduce power consumption of such infrastructures, processor manufacturers have developed a hardware technology called *Dynamic Voltage and Frequency Scaling (DVFS)*. DVFS [12] allows dynamic processor frequency control, and hence, helps in reducing power consumption.

DVFS is largely used in non-virtualized systems, but its implementation in virtualized architectures reveals some incompatibilities with VM schedulers. In virtualized systems, VMs are generally created and configured with a fixed CPU share. DVFS, according to the host's global CPU load, dynamically scales the processor frequency regardless of the VM local loads. In a scenario with some overloaded VMs, but a globally underloaded host, DVFS will scale down processor frequency, which will penalize overloaded VMs.

The contributions of this paper are two folds. First, we analyze and highlight the incompatibility of available VM schedulers with DVFS management in virtualized systems. Second, we identify the key issues and propose a DVFS aware VM scheduler which addresses these issues. To demonstrate the effectiveness of our approach, we implemented and evaluated a prototype in the Xen [1] virtualized environment.

The rest of this paper is structured as follows. Section 2 presents the context of our work. Section 3 analyzes VM schedulers and DVFS principles and pinpoints their incompatibilities. In Section 4, we present our DVFS aware VM scheduler prototype. Section 5 presents our experiments and results. After a review of related works in section 6, we conclude the article in Section 7.

2 Context

In this section, we introduce the basic concepts of virtualization and DVFS.

2.1 Virtualization

Virtualization is a software and/or hardware-based solution for building and running simultaneously several *guest OS* on top of an underlying *host OS*. The key technique is to share hardware resources safely and efficiently between these guest OS. In the host OS, a *hypervisor* implements several execution environments also known as *Virtual Machines (VM)* in which guest OS can be executed. Thanks to the hypervisor, guest OS are isolated from each other, and have the illusion of being in the presence of several separate machines. The hypervisor is also the program that ensures good sharing of hardware resources among multiple guest OS. It emulates the underlying hardware for VMs[3] and enables communication between guest OS and real devices.

The hypervisor is responsible for scheduling VMs on the processor. Initially, VMs are created and configured in order to have, among other parameters, an execution priority and a CPU credit. The hypervisor scheduler chooses the VM to execute, according

[3] Each guest OS has the illusion of having host's processor, memory and other resources all to itself.

to its scheduling algorithm and the VM parameters. It ensures dynamic and fair allocation of CPU resources to VMs. However, in each VM, the execution of a guest OS implies the execution of processes scheduled by another process scheduler. Therefore, the execution of an application in a virtualized environment involves different levels of scheduler, but the hypervisor is not conscious of it. From its point of view, VMs are just processes which are scheduled in such a way that each VM receives its associated credit of the CPU [18] (a percentage of the CPU time).

In the rest of the article, we will use the term VM to refer to a guest OS running in a virtual machine.

2.2 Dynamic Voltage and Frequency Scaling (DVFS)

Today, all processors integrate dynamic frequency scaling (also known as CPU throttling) to adjust frequency at runtime. The system service which adapts frequency in the operating system is called a *governor*. Different governors can be implemented with different policies regarding frequency management.

In the Linux kernel, the *Ondemand* governor changes frequency depending on CPU utilization. It changes frequency between the lowest level (when CPU utilization is less than 20% [22]) and the highest level (when CPU load is higher). The *Performance* governor always keeps the frequency to the highest value while the *Powersave* governor keeps it at the lowest level. The *Conservative* governor decreases or increases frequency by one level through a range of values supported by the hardware, according to the CPU load. Finally, the *Userspace* governor allows user applications to manually set the processor frequency [13].
In order to control the CPU frequency, governors rely on an underlying subsystem inside the kernel called cpufreq [22]. Cpufreq provides a set of modularized interfaces to allow changing the CPU frequency.

As aforementioned, effective usage of DVFS brings the advantage of reducing power consumption by lowering the processor frequency. Moreover, almost all computing infrastructures rely on multi-core and high frequency processors. Therefore, the benefit from using DVFS has been experienced in many different systems.

2.3 Consolidation and DVFS

Virtualization allows VMs to be dynamically migrated between hosts, generally according to the CPU load on the different hosts, and to switch unused machines off. Ideally, a consolidation system should gather all the VMs on a reduced set of machines which should have a high CPU load, and DVFS would therefore be useless.

However, as argued in [16], an important bottleneck of such consolidation systems is memory. Any VM, even idle, needs physical memory, which limits the number of VMs that can be executed on a host. Therefore, even if consolidation can reduce the number of active machines in a hosting center, it cannot optimally guarantee full usage of CPU on active machines as it is memory bound. Consequently, DVFS is complementary to consolidation.

The next section analyzes the incompatibilities between virtualization and DVFS for energy saving.

3 Analysis

The overall goal of this paper is to show that combining virtualization and DVFS management may raise incompatibilities and that it is required a smart coordination between DVFS management and VM management (i.e., VM scheduling). In this section, we first review VM schedulers, especially those that are effectively used in virtualization solutions (such as Xen). We then analyze the issues that are raised when these schedulers are combined with DVFS management.

3.1 VM Schedulers

VM schedulers are in charge of allocating CPU to VMs. Commonly, schedulers are classified into three categories: *share*, *credit allocation* and *preemption* [6]. In our work, we focus on credit allocation schedulers as they aim at allocating a portion of processor to a VM. This portion of the processor corresponds to a SLA (service level agreement) negotiated between the provider and the customer, i.e. this portion of the CPU was bought by the client and has to be guarantee by the provider.

In the credit scheduler category, we distinguish: **fix credit** and **variable credit** schedulers.

With fix credit scheduler[4], the CPU credit of each VM is guaranteed, which means that the VM always obtains the time slices corresponding to this credit (a percentage of the processor). For instance, if two VMs are running on the same physical host with the same priority and CPU credit (50%), then each of them will receive at most 50% of the CPU time even if one of them becomes inactive.

With variable credit scheduler[5], the CPU credit of each VM is also guaranteed, but only if the VM has a computation load to effectively use it. In the case of unused CPU time slices, they are redistributed among active VMs. It means that the processor is idle only if there is no more runnable VM. For example, with two VMs with the same priority and CPU credit (50%), each of them will receive 50% of the CPU time slices if they are both fully using their time slices, but if one of them becomes inactive, the active VM may receive up to 100% of the CPU.

In this paper, we conducted our experiments with the Xen system (version 4.1.2). Xen has three schedulers called *Credit, Simple Earliest Deadline First (SEDF) and Credit2*. Credit2 scheduler is an updated version of Credit scheduler, with the intention of solving some of its weaknesses. This scheduler is currently available in a beta version. Credit is the default Xen scheduler and SEDF is about to be removed from Xen sources. In the following, we only consider credit and SEDF schedulers as they allow illustrating the incompatibilities (with DVFS) we aim at addressing.

Xen Credit scheduler is primarily a fix credit scheduler. A VM can be created with a given priority and a given credit, and the VM credit is always guaranteed. The only exception is when allocating a VM with a null credit. In this latter case, the VM will not have any credit limit and it can use any CPU time slices that are not used by other VMs

[4] Also called *non-work conserving* scheduler

[5] Also called *work conserving* scheduler

(such a VM behaves as with a variable credit scheduler, except that it does not have any guaranteed credit).

With Xen SEDF scheduler, each VM is configured with a triplet *(s,p,b)*, where *s* represents *the lowest slice* of time during each *period* of length *p* where the VM will use the CPU. The boolean flag *b*, indicate whether the VM is eligible or not to receive extra CPU time slices that are not used by other VMs. Therefore SEDF can be both used as a fix or variable credit scheduler (according to the *b* flag), and the credit allocated to a VM can be defined with the *s* and *p* parameters.

In our experiments, we create VMs with a given fraction of the CPU capacity (a credit corresponding to a SLA) and we illustrate the incompatibilities between DVFS and VM schedulers by using Xen Credit scheduler as fix credit scheduler and Xen SEDF scheduler as variable credit scheduler.

In the rest of the paper, we will not consider different VM priorities as we assume that the overall goal of a hosting center provider is to allocate portions of a processor (with VMs) to customers, without any priority between customers.

3.2 Combining DVFS and VM Scheduling

In version 4.1.2, Xen supports four governors (as described in Section 2.2): *ondemand, performance, powersave* and *userspace*.

The Ondemand governor is the most used for DVFS. Depending on the global host CPU load, the governor adjusts the processor frequency between the highest and the lowest level. However, the VM scheduler selects and executes VMs regardless of processor frequency, and therefore a processor frequency reduction influences VM performance.

Let us consider a Xen virtualized system with two VMs (V20 and V70) running on the same physical host. They are respectively configured with 20% and 70% of credit. The Ondemand governor will set the suitable processor frequency according to the load. We assume in the illustrative example used below that reducing the processor frequency slows down the processor by 50%.

Then, let us consider, the two following scenarios (with different schedulers).

Scenario 1 - Fix Credit Scheduler We assume that the host is working with a fix credit scheduler. If V20 is overloaded (100% of its 20% CPU credits) and V70 is underloaded (0% of its 70% CPU credits), then the host is globally underloaded (the load is theoretically 20%). The Ondemand governor scales down the processor frequency. This reduction saves energy, but V20 is heavily penalized. Indeed, instead of receiving its percentage (20%) of the computing capacity, V20 receives less (50% of 20%) because of the frequency reduction. When a credit is allocated to a VM as a percentage of the total host CPU capacity, this percentage is a fraction of the processor capacity at the maximum frequency. If the processor frequency is decreased because V70 is not using its allocated credit (and the host is therefore globally underloaded), V20 does not obtain its initially allocated credit.

In summary in this scenario, scaling down frequency decreases V20's performance and therefore its applications QoS, because the fix credit scheduler is not aware of processor frequency scaling.

Scenario 2 - Variable Credit Scheduler Now, consider that our host is working with a variable credit scheduler. With the same assumptions than in the previous section (V20 overloaded, V70 underloaded), V70's unused CPU time slices can be given to V20, because of variable credit allocation, which counterbalances the effects described with fix credit scheduler if we would have a frequency reduction. However, we will not have any frequency reduction. All the V70 unused time slices can be given to V20 (without any limit), which leads to a globally overloaded host, which in turn prevents the processor frequency be scaled down.

In this scenario, the problem with the variable credit scheduler is that by giving unused time slices to V20, it will prevent frequency scaling, thus wasting energy from the point of view of the provider.

Design Principles of Power Aware Scheduling Because of the independence of VM schedulers and DVFS governors, either the provider cannot guarantee the QoS required by the customers (with the fix credit scheduler) or a VM will be allowed to use more CPU than its allocated credit, preventing DVFS scaling (with the variable credit scheduler). The previous scenarios reveal incompatibilities between schedulers and governors.

Our proposal is to take advantage of DVFS to lower power consumption while guaranteeing the credits allocated to VMs. Concretely, when the processor frequency is modified, we reconsider the credit associated with VMs in order to counterbalance the effect of the frequency modification. The consequence is that:

- the initially configured credit of a VM is a percentage of the computing capacity of the processor at the maximum frequency (20% for V20 and 70% for V70 in our scenario)
- a VM will see its credit increased (resp. decreased) whenever the processor frequency is decreased (resp. increased), this credit (at the new frequency) being equivalent to the initial credit (at the maximum frequency). In the previous scenario, when the processor frequency is decreased (slowing down the processor by 50%), V20 will be given 40% of credit to counterbalance the frequency reduction.
- a VM is never given more computing capacity than its allocated credit, enabling frequency reductions

The next section details this contribution.

4 Contributions

As previously argued, DVFS and VM schedulers have incompatibilities. DVFS was introduced for power reduction, but cannot directly be exploited for the same purpose in virtualized systems. Our contribution aims at managing DVFS in a virtualized environment while (i) benefiting from power reduction and (ii) guaranteeing allocated CPU credits.

We implemented our *Power Aware Scheduler* (*PAS* for short) in the Xen environment as an extension of the Xen Credit scheduler, which is the default and most achieved VM scheduler. The following subsections present our implementation choices and the implementation of the PAS scheduler.

4.1 Implementation Choices

We considered three possible implementations for our PAS scheduler:

- *user level - credit management*. In this design, we let the Ondemand governor manage the processor frequency. Then, a user level application monitors the processor frequency, and periodically computes and sets VM credits in order to guarantee initially allocated credits.
- *user level - credit and DVFS management*. In this design, a user level application monitors the VM loads. Periodically, it computes and sets the processor frequency which can accept the load, and it also computes and sets the updated VM credits. With this solution, the VM credits can be updated each time the processor frequency is modified.
- *in the Xen system - credit and DVFS management*: A user level implementation can be quite intrusive because of system calls and it may lack reactivity. Another possibility is to implement it as an extension of the VM scheduler. DVFS and VM credit computations and adaptations are then performed each time a scheduling decision is made.

We experimented with these three solutions. The results reported in this paper are based on the third implementation.

4.2 PAS Scheduler Implementation

In our implementation of the PAS scheduler, we rely on two main assumptions:

- proportionality of frequency and performance. This property means that if we modify the frequency of the processor, the impact on performance is proportional to the change of the frequency.
- proportionality of credit and performance. This property means that if we modify the credits allocated to a VM, the impact on performance is proportional to the change of the credits.

Proportionality of frequency and performance

This proportionality is defined by:

$$\frac{L_{max}}{L_i} = \frac{F_i}{F_{max}} \times cf_i \quad (cf_i \text{ is very close to 1}) \tag{1}$$

which means that if we decrease the frequency from F_{max} down to F_i, the load will proportionally increase from L_{max} to L_i. For instance, if F_{max} is 3000 and F_i is 1500, the frequency ratio is 0.5 which means that the processor is running 50% slower at F_i compared to F_{max}. So if we consider a load (L_{max}) of 10% at F_{max}, the load (L_i) should be $\frac{10\%}{0.5} = 20\%$ at F_i.
Even if cf_i is very close to 1, we kept this variable in our equations as we observed that it may vary according to the machine architecture and the considered frequency F_i.
A similar proportionality is defined for execution times. But we add here that execution

times depend on the credit (*j*) allocated to the VM which hosts the computation (VM credits are considered below)[6].

$$\frac{T_{max}^{j}}{T_{i}^{j}} = \frac{F_i}{F_{max}} \times cf_i \tag{2}$$

We define the frequency ratio as $ratio_i = \frac{F_i}{F_{max}}$.

Proportionality of credit and performance

This proportionality is defined by:

$$\frac{T_{i}^{init}}{T_{i}^{j}} = \frac{C^j}{C^{init}} \tag{3}$$

which means that if we increase the credits of a VM from C^{init} up to C^j, the execution time will proportionally decrease from T_i^{init} to T_i^j. Here, the execution time also depends on the frequency (*i*) of the processor. For instance, if we increase the credits allocated to a VM from 10% to 20%, we double the computing capacity of the VM. Then the execution time should become half of the initial execution time.

These proportionality rules are validated at the beginning of the evaluation section (Section 5).

In our algorithms, the first equation (1) is used to estimate, for a given CPU load, what would be the load at a different processor frequency. Therefore, if we measure a load L_i at frequency F_i, we can compute the **absolute load**, i.e., the equivalent load at F_{max} which is $L_i * ratio_i * cf_i$. And if we want to check whether such an absolute load can be supported at a different frequency (i), we will check whether this absolute load is less than $100 * ratio_i * cf_i$.

The two other equations (2 and 3) are used to compute the modification of VM credits, which can compensate the performance penalty incurred by a frequency reduction. Assume that a VM is initially allocated credit C^{init} (which is a fraction of the processor at frequency F_{max}). Then assume that the frequency of the processor is reduced down to F_i. We are looking for the new credit C^j to assign to this VM, so that its execution time would be the same as with credit C^{init} and frequency F_{max}, i.e. so that $T_i^j = T_{max}^{init}$.

According to equation 3, $C^j = \frac{T_i^{init} * C^{init}}{T_i^j}$

According to equation 2, $T_i^j = \frac{T_{max}^j}{ratio_i * cf_i}$, so $T_i^{init} = \frac{T_{max}^{init}}{ratio_i * cf_i}$

Therefore, $C^j = \frac{T_{max}^{init} * C^{init}}{ratio_i * cf_i * T_i^j}$

and we want that $T_i^j = T_{max}^{init}$

So $C^j = \frac{T_{max}^{init} * C^{init}}{ratio_i * cf_i * T_{max}^{init}} = \frac{C^{init}}{ratio_i * cf_i}$

[6] in the following, frequencies are show as subscripts and credits as exponents

In summary:

$$C^j = \frac{C^{init}}{ratio_i \ * \ cf_i} \tag{4}$$

This means that with our assumptions, we can compensate the performance penalty incurred by a frequency reduction as follows:

- If we run a computation in a VM with 20% credit at the F_{max} frequency.
- If we reduce the processor frequency from F_{max} to F_i, for instance half the maximum frequency, so that $ratio_i$ is 0.5.
- we can change the credit to $20\% \div 0.5 = 40\%$ in order to have the same computing capacity, and we will have the same computation time or the same computation load under this new frequency (assuming that cf_i=1).

We now describe the implementation of the PAS scheduler.

The PAS scheduler relies on a set of variables that are used for the computation of the processor frequency to be used and the credits to be associated with VMs. We also define additional variables that are used to explain the behavior of our PAS scheduler in the evaluation section.

- *VM[]* is a table of the VMs managed by the scheduler and *nbVM* the number of VMs.
- *Credit[]* is a table of the credit associated with each VM.
- *Freq[]* is a table of the possible processor frequencies and *fmax* is the number of frequencies (so *Freq[fmax]* is the maximum frequency).
- *CF[]* is a table of the variables (cf_i) associated with the different frequencies.
- *CurrentFreq* is the current frequency of the processor.
- *VM_load* is the observed load of a VM (e.g., V20 has a VM_load of 100% in our previous scenario).
- *VM_global_load* is the contribution of a VM to the load of the processor (e.g., V20 which is allocated a credit of 20% and which has a *VM_load* of 100%, contributes to the processor load for 100% of 20%, that is 20%). If a VM is allocated a credit of *VM_credit*, then *VM_global_load* = *VM_load* * *VM_credit*.
- *Global_load* is the load of the processor. Therefore, *Global_load* = $\sum$*VM_global_load*. [7]
- *Absolute_load* is the processor load that we would have if processor was running at the maximum frequency. According to our previous assumption regarding frequencies, we have: *Absolute_load* = *Global_load* * $\frac{CurrentFreq}{Freq[max]}$ * $cf_{CurrentFreq}$.

As mentioned in Section 4.1, the PAS scheduler has been implemented both at user level and at system level. In the following, we rely on the system implementation.
At each tick in the VM scheduler, we compute the appropriate processor frequency according to the *Absolute_load*, as depicted in the algorithm below (Listing 1.1). We

[7] Note that, each time we consider the *Global_load*, it represents an average of three successive processor utilization.

iterate on the processor frequencies (line 2). Following our assumption regarding frequencies, we compute for each frequency the frequency ratio (line 3) and check if the computing capacity of the processor at that frequency can absorb the current absolute load (line 4).

```
int computeNewFreq() {
 for (i=1; i<=fmax; i++) {
    int ratio = Freq[i]/Freq[fmax];
    if (ratio * 100 * CF[i] > Absolute_load)
      return Freq[i];
      }
 return Freq[fmax];
 }
```

Listing 1.1. Algorithm for computing the next processor frequency

At each tick, we need to compute the new credits associated with VMs and to modify VM credits and the processor frequency. This is described in the algorithm below (Listing 1.2). For the new frequency of the processor, we compute the frequency ratio (line 3) and for each VM, we compute the new credit that has to be associated with each VM (line 5) and assign it (line 6). The credit of a VM increases when the frequency of the processor decreases. Finally, we modify the processor frequency (line 7).

An important remark is that with this algorithm, when the processor frequency is low, the sum of the VM credits may be more than 100%, because we computed the new credit limit for each VM.

```
void updateDvfsAndCredits() {
  int newFreq = computeNewFreq();
  int ratio = newFreq/Freq[fmax];
  for (vm=1; i<=nbVM; vm++) {
    in newCredit = Credit[vm]/(ratio * CF[newFreq]);
    setCredit(VM[vm], newCredit);
  }
  setFrequency(newFreq);
}
```

Listing 1.2. Algorithm for computing VM credits, and setting VM credits and processor frequency

Some of these VM are active and some are lazy. For active VM, this new limit extends their computing capacities and compensate the frequency reduction. For lazy VM, this new limit is meaningless as it will not be reached (if the load of lazy VMs increases, the processor frequency will increase and VM credits will be decreased).

5 Evaluation

5.1 Environment

Our experiments were performed on a DELL Optiplex 755, with an Intel Core 2 Duo 2.66GHz with 4G RAM. We run a Linux Debian Squeeze (with the 2.6.32.27 kernel)

in a single processor mode. The Xen hypervisor (in his 4.1.2 version) is used as virtualization solution.

The evaluation described below were performed with two applications:

- when we aim at measuring an execution time, we use an application which computes an approximation of π. This application is called π-*app*.
- when we aim at measuring a CPU load, we use a web application (a Joomla CMS server) which receives a load generated by httperf [14]. This application is called *Web-app*. The Joomla server consists of Joomla 1.7, relying on Apache 2.2.16, PHP 5.3.3 installed into Apache with modphp5 and MySQL 5.1.49. This application is called *Web-app* in the following.

5.2 Verification of our Assumptions

As said at the beginning of Section 4, the implementation of our PAS scheduler relies on two main assumptions: proportionality of frequency and performance (equation 1 & 2) and proportionality of credit and performance (equation 3).

In order to validate these two assumptions, we conducted the following experiments:

- Proportionality of frequency and performance. We ran different Web-app workloads at the different processor frequencies. For each workload, we measured the loads *L(freq)* at the different *freq* processor frequencies and we drew for each workload the ratios $\frac{L(freqmax)}{L(freq)} and \frac{freq}{freqmax}$, in order to compute the cf_i values for each frequency and to verify that they were constant under various workloads (thus validating equation 1). We also ran different π-app workloads at different processor frequencies and measured the execution times, allowing us to verify the proportionality of frequency ratios and execution time ratios (equation 2).
- Proportionality of credit and performance. We ran different π-app workloads on VMs configured with different credits (with the Xen credit scheduler). For each workload and credit (index j), we measured the execution time and computed the credit ratio ($\frac{C^j}{C^{init}}$) and the execution time ratio ($\frac{T_i^{init}}{T_i^j}$), in order to verify equation 3.

These experiments allowed to validate these proportionality assumptions and to compute the cf_i values (more details are given in Section 5.8).

Finally, in order to verify the accuracy of equation 4 which is used to compensate (with a credit allocation) the performance penalty incurred by a frequency reduction, we executed π-app at the maximum frequency (2667 MHz) with different initial credits (10, 20, 30 . . .), then we ran the same experiment at frequency 2133 MHz, but computed with equation 4 what should be the associated credits which compensate this frequency reduction. Figure 1 shows our results. The X axis at the bottom gives the initial credits, the X axis at the top gives the computed credits, and the Y axis gives the execution times. This experiments shows that we can effectively compensate a frequency reduction with a credit allocation.

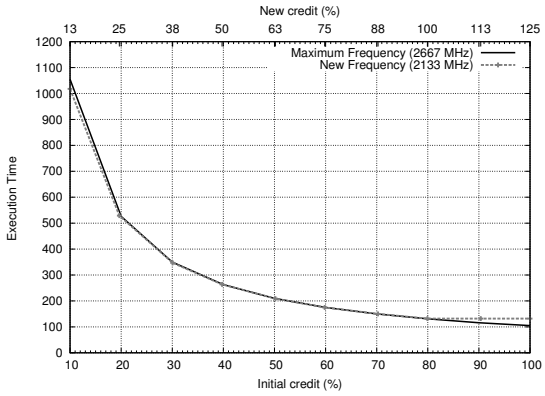

Fig. 1. Compensation of Frequency Reduction with Credit Allocation

5.3 Execution Profile

In the rest of this evaluation section, we rely on the Web-app application previously described and we consider 2 virtual machines called V20 and V70, with respectively 20% and 70% of initially allocated credit. The remaining 10% of credit are allocated for the hypervisor (the Dom0 in Xen) which is configured with the highest priority in the VM scheduler. The two VMs have the same priority.

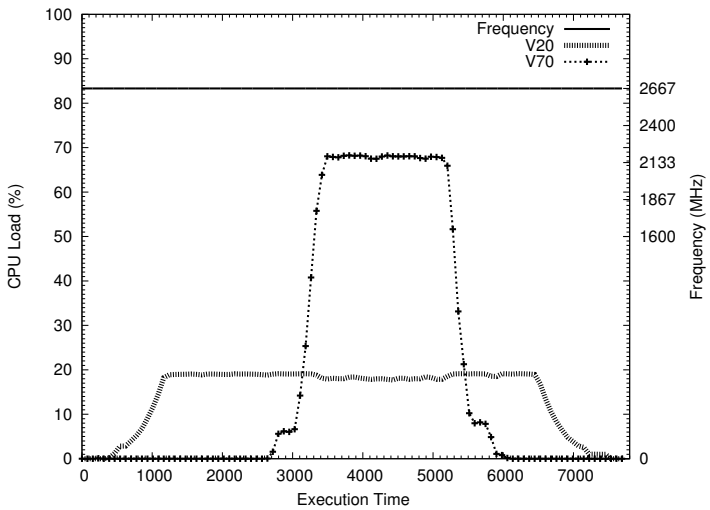

Fig. 2. Load profile (at the maximum frequency)

The objective of the execution profile we are considering is to reveal the scheduling problem we are addressing and to demonstrate the effectiveness of our solution (the PAS scheduler). Both VMs have a three-phase profile: inactive-active-inactive:

- inactive. During the inactive phase, the VM does not receive any load from the load injector (httperf).
- active. During the active phase, the VM may receive two types of load: either the injector is configured to generate a load which represents 100% of the VM capacity but not more (we call such a load an *exact load*), or it is configured to generate a load which exceeds the VM capacity (we call such a load a *thrashing load*).

Figure 2 shows the VM_global_load (as defined in Section 4, the contribution of the VM to the load of the processor) for both VMs when executing this profile with the credit scheduler, and with the processor frequency being kept at its maximum value (the frequency is shown on the Y axis on the right side). Notice here that the same performance figure is obtained with an exact load or a thrashing load, since the credit scheduler limits the amount of CPU that a VM may use (according to its initially allocated credit). This figure characterizes the execution profile we will use in the rest of the article.

5.4 Credit Scheduler in Default

We now run the same execution profile (with an exact load) with the credit scheduler, but with the Ondemand DVFS management governor.

As observed on Figure 3, the default Ondemand governor is quite aggressive and unstable. Therefore, we implemented our own (ondemand) governor, which is less aggressive and more stable, and consequently saves less energy. We performed the same experiments with both governors and observed the same overall behaviors (Figure 4), but without such oscillations with our governor (that we use in rest of this evaluation for readibility of figures).

In the two previous figures, when a VM is in the active phase, its VM_global_load is 70% for V70 and 20% for V20 (its contribution to the load of the processor).

While the previous Figures gave the observed VM_global_loads, Figure 5 shows the Absolute_load (defined in Section 4 as the processor load that it represents with a processor running at the maximum frequency, or more precisely Absolute_load = Global_load * $\frac{CurrentFreq}{Freq[max]} * cf_{CurrentFreq}$). We observe that in the first phase (when V20 is active and V70 inactive), the absolute load of V20 is close to 10%. This is due to the lowered processor frequency, since the global load of the processor is only 20%. However, as soon as V70 becomes active, the global load of the processor becomes high enough to scale up the processor frequency at the maximum level, and then the absolute load of V20 climbs to 20%. In summary, V20 is only granted its allocated (absolute) credit (20%) when the processor frequency is at the maximum level.

5.5 SEDF Scheduler Brings a Solution

We ran the same experiment with the SEDF scheduler. Remind that with SEDF, unused CPU time slices can be given to active VMs. Therefore, as observed on Figure 6, in

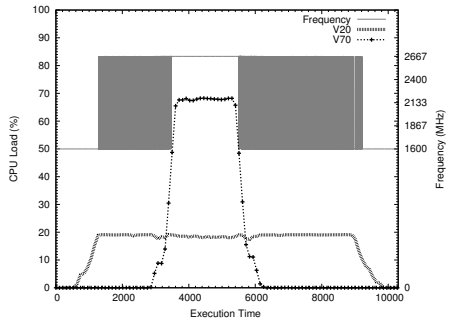

Fig. 3. Global loads with Ondemand governor / Credit scheduler / exact load

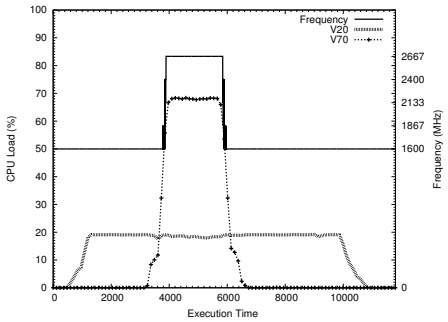

Fig. 4. Global loads with our governor / Credit scheduler / exact load

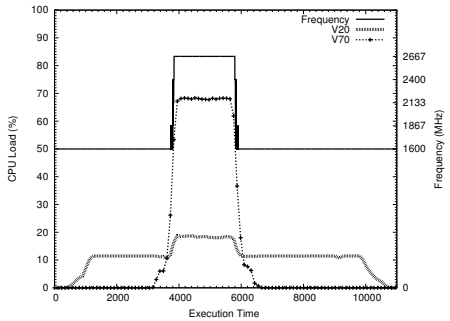

Fig. 5. Absolute loads with our governor / Credit scheduler / exact load

the first phase (when V20 is active and V70 inactive), V20 has a global load of 35%, because it is given time slices which are not used by V70. And when V70 becomes active, then the initially allocated credits are respected and V20 ends up with 20% of global load (at the maximum processor frequency).

And if we observe the absolute load in this experiment (Figure 7), we see that unused time slices that were given to V20 allowed to compensate the penalty of the lowered processor frequency. V20 has a 20% absolute load during the entire experiment. Therefore, SEDF brings a solution to our identified issue, i.e., the fact that an active VM can be victim of a frequency reduction (due to other VM laziness).

5.6 SEDF Scheduler in Default

However, the SEDF scheduler does not actually solve the problem. In the previous experiments, we used exact loads (which represents 100% of the VM capacity but not more). If we use thrashing loads (which exceed VM capacities), we observe (Figure 8) that in the first phase (when V20 is active and V70 inactive), the SEDF scheduler gives unused time slices to V20, which in turn brings the processor frequency at the highest level. In this first phase, V20 is allowed to consume 85% of the processor. This is not

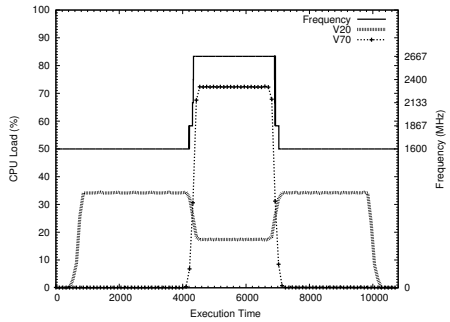

Fig. 6. Global loads with our ondemand governor / SEDF scheduler / exact load

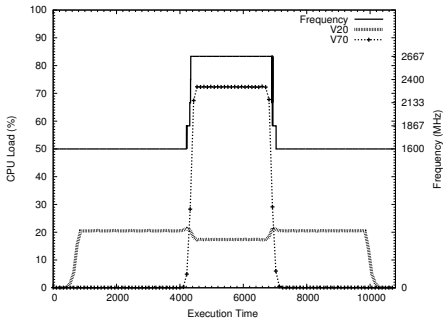

Fig. 7. Absolute loads with our governor / SEDF scheduler / exact load

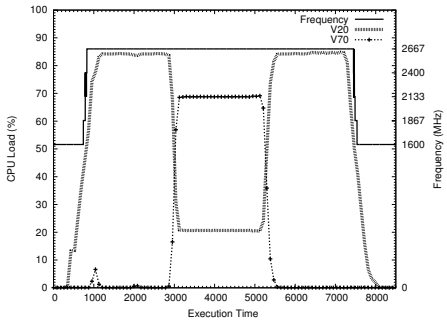

Fig. 8. Global or absolute loads with our governor / SEDF scheduler / thrashing load

consistent from the point of view of the provider in a hosting infrastructure, since V20 was initially allocated 20% of credit and the provider does not benefit from a frequency reduction due to V70 inactivity.

In the second phase, when V70 becomes active, the SEDF scheduler guarantee the initially allocated credits and V20 cannot benefit from unused time slices anymore.

Notice here that in this experiment, the global and absolute load figures are the same (we only show a single figure) since the processor frequency is kept at the highest level during the whole experiment.

5.7 PAS Scheduler Solves the Problem

Our PAS scheduler recomputes credits allocated to VMs according to the frequency of the processor. Therefore, it provides the same benefits than the SEDF scheduler with the exact load, but also guarantees the respect of credits under thrashing loads. In Figure 9, the PAS scheduler computes that in the first phase, V20 should be granted 33% of credit in order to compensate the low processor frequency (1600 MHz). In the second phase, V20 is granted 20% of credit as the processor frequency reaches the maximum value.

With this strategy, the absolute loads of each VM is consistent with credit allocations (Figure 10).

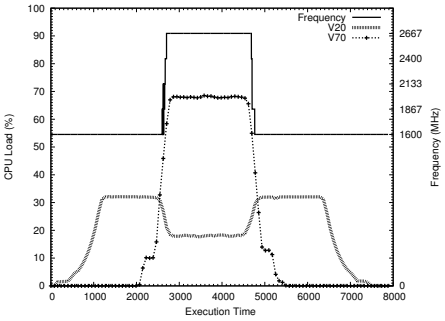

Fig. 9. Global loads with the PAS scheduler / thrashing load

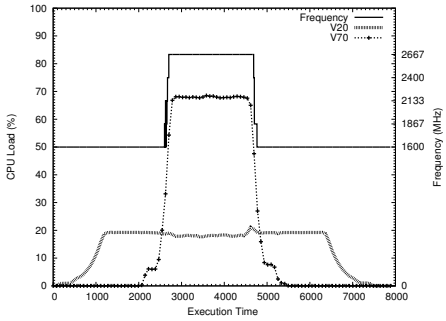

Fig. 10. Absolute loads with the PAS scheduler / thrashing load

5.8 Other Environments

In order to study the applicability of this approach in other environments, we experimented with different hardware architectures and different virtualization systems.

Other Hardware We verified that our proportionality assumptions, described in Section 4.2 and validated in Section 5.2 are valid on other hardware architectures. Therefore, we measured for different workloads the introduced variable cf_i, which may depend on the hardware architecture, for different types of machines available on Grid5000, the french national grid (Table 1). We report the measurements (cf_i) only for the minimal frequency, as many processors only have 2 available frequencies. We observed that even is cf_{min} is most of the time equal to one, it may significantly vary on particular architectures (e.g. Intel Xeon E5-2620).

	Intel Xeon X3440	**Intel Xeon L5420**	**Intel Xeon E5-2620**	**AMD Opteron 6164 HE**	**Intel Core i7-3770**
cf_{min}	0,94867	0,99903	0,80338	0,99508	0,86206

Table 1. cf_{min} on different processors

Other Virtualization Platforms We verified that the issue we address is relevant in other virtualization platforms. Therefore, we ran the same scenario as in Section 5.3 on different virtualization environments and measured the execution time of the V20

virtual machine (Table 2). V20 can be penalized by a frequency reduction when V70 is lazy. These measurements were performed on the leading virtualization products (commercial or open-source) that we installed on the same hardware configuration, a HP compaq Elite 8300 (with an Intel Core i7-3770 3.4GHz with 8G RAM). This machine embeds hardware assisted virtualization technologies that were enabled in all our experiments. On the left part of the table, we compare solutions with a fix VM credit scheduler. The V20 virtual machine is significantly penalized on Hyper-V Server 2012, Vmware ESXi 5 and Xen (with its credit scheduler), and our PAS scheduler in Xen cancels this degradation by allocating additional credits to V20. On the right part of the table, solutions with variable credit schedulers have a much faster execution time, since the CPU capacity of V70 is given to V20 when V70 is lazy. However, V20 may consume any amount of unused CPU capacity, which prevents a reduction of the processor frequency, thus wasting energy.

	Fix credit scheduler				Variable credit scheduler		
	Hyper-V	**VMware**	**Xen/credit**	**Xen/PAS**	**Xen/SEDF**	**KVM**	**Vbox**
Performance	1601	1550	1559	1559	616	599	625
OnDemand	3212	2132	2599	1560	616	599	625
Degradation(%)	50	27	40	0	0	0	0

Table 2. Execution Times on Different Virtualization Platforms

6 Related Work

In recent years, we observed the rapid development of hosting infrastructures and their energy consumption became an important issue.

Energy saving in hosting centers.

In order to better manage hosting center energy consumption, the *Green Grid* [2] association defined metrics, such as *Power Usage Effectiveness (PUE)*. *PUE* is a measure of how efficiently a hosting center uses its power; specifically, how much of the power is actually used for computing (in contrast to cooling and other overheads). It is computed as follows: PUE $=\frac{TotalFacilityPower}{ITEquipmentPower}$ where $TotalFacilityPower$ and $ITEquipmentPower$ represent respectively the global power consumption of the hosting center and the power associated with all the IT equipment (computers, storage, network equipments, etc.). The ideal value of PUE should be 1, meaning that all the power consumed by the hosting center is dedicated for computing. Such metrics allow the estimation of the energy efficiency of hosting centers, to compare the results against other hosting centers, and to determine if any energy management improvements can be made. For example, in [10], James Hamilton exploits the PUE metrics to determine the power distribution of his computing infrastructure in order to reduce high-scale data center costs.

Energy saving for computing servers.

Many research projects have focused on reducing the energy consumed by servers in hosting centers. The general orientation is to rely on dynamic resource allocation. In hosting centers, hardware resources are mutualized among multiples customers, which is a means to use less resources while fulfilling the requirements of customers. Customers subscribe for resources, but those resources are made available to customers only if effectively used. Therefore the amount of active resources can be reduced, thus leading to energy saving. Such energy management policies are generally implemented at the level of servers.

In 2001, Chase et al. [4] showed that hosting center servers used at least 60% of their peak power in idle state. Therefore, it is beneficial to gather computations on a reduced set of servers and to switch idle servers off. In this vein, servers Vary-On/Vary-Off (VOVO) [19] strategies have been proposed and adopted by many researchers. They consist in load-balancing a computing load on a set of servers, and according to the load, increasing or decreasing the number of active servers in that set [20]. Chen at al. [5] investigated the use of this strategy for power saving in a HPC system.

However, such VOVO approaches require applications to be structured following a master-slave model where a load-balancer balances the load between a number of slave servers, which can be adapted according to the received load. This is an important constraint on the design of applications.

Energy saving in virtualization environments.

If virtualization technologies were first introduced about 30 years ago [9], they are now increasingly used for resource management in hosting centers. In this context, the main advantage of virtualization is to relax the previous constraints on applications [21]. Application services can be deployed on separate virtual machines and a global resource manager is responsible for the allocation of resources to these VMs according to the load. This global manager can notably rely on VM migration [8] to gather VMs on fewer physical machines and to switch unused machines off. Such an approach is generally known as *server consolidation* [3, 15]. Another important advantage of virtualization is isolation of applications, as consolidation may collocate VMs from different applications on the same physical host [1].

Energy saving with frequency scaling.

Beside the reduction of the number of active machines in a hosting center, another way to reduce energy consumption is to dynamically adapt the frequency of active machines according to the CPU load on these machines. Such techniques are known as Dynamic Voltage and Frequency Scaling (DVFS). Several studies showed that DVFS allows significant energy consumption reductions [7, 11]. Moreover, recent works studied the impact of DVFS on applications performance and their Quality of Service. Chengjian Wen et al. [23] proposed to combine DVFS management and VM scheduling in a cluster in order to ensure fairness in the energy consumption of VMs, by accounting VMs power usage and prioritizing VMs accordingly. In the same vein, Laszewski et al. [12] investigated a similar approach while ensuring QoS in terms of execution times.

Positioning our contribution.

Our contribution shares many objectives with the works mentioned in the previous paragraph. Similarly, our goal was to guarantee a QoS allocated to VMs while saving

energy thanks to DVFS. However, these projects didn't consider that a VM is allocated a computing capacity (a credit) at creation time and that it has to be managed as a Service Level Agreement (SLA). Our Power-Aware Scheduler (PAS) allows DVFS management while guaranteeing that the computing capacity allocated to a VM (and bought by a customer) is available. We are not aware of any similar contribution to this issue.

7 Conclusion and Perspective

With the emergence of cloud computing environments, large scale hosting centers are being deployed and the energy consumption of such infrastructures has become a critical issue. In this context, two main orientations have been successfully followed for saving energy:

- Virtualization which allows to safely host several guest operating systems on the same physical machines and more importantly to migrate guest OS between machines, thus implementing server consolidation.
- DVFS which allows adaptation of the processor frequency according to the CPU load, thus reducing power usage.

We observed that these two techniques suffer from incompatibilities, as DVFS governors are implemented in the hypervisor and don't take into account the existence of different VMs with allocated credits and different loads. If a machine is globally underloaded but hosts a loaded VM (which consumes a significant part of its credit), then the frequency of the processor may be scaled down, thus affecting the computing capacity of the loaded VM.

In this paper, we proposed a Power-Aware Scheduler (PAS) which addresses this issue. A credit is associated with a VM at creation time and represents its allocated computing capacity. If the machine which hosts the VM is underloaded and its frequency is therefore scaled down, the credit associated with the VM is recomputed in order to maintain its computing capacity.

Our PAS scheduler was implemented in the Xen hypervisor and evaluated through different scenarios which demonstrate its advantage over the Credit and SEDF schedulers, the two schedulers available in Xen.

Our main perspective is to address the issue presented in Section 2.3. Memory is the main limitation factor for an efficient consolidation system. We are investigating energy aware resource management strategies which would coordinate VM scheduling, frequency scaling and memory management in a hosting center. Furthermore, we plan to extend our scheduler and take into account other technology factors such as hyperthreading, multi-core, per-socket DVFS, and per-core DVFS.

Acknowledgements The work reported in this article benefited from the support of the French National Research Agency through projects Ctrl-Green (ANR-11-INFR-0012).

References

1. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. Proceedings of the 9th ACM symposium on Operating systems principles. (2003)
2. Belady, C., Rawson, A., Pfleuger, J., Cader, T.: The green grid data center power efficiency metrics: PUE and DCiE. White paper. (2007) http://www.thegreengrid.org/en/Global/Content/white-papers/The-Green-Grid-Data-Center-Power-Efficiency-Metrics-PUE-and-DCiE
3. Beloglazov, A., Buyya, R.: Energy efficient resource management in virtualized cloud data centers. Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing. (2010)
4. Chase, J.S., Anderson, D.C., Thakar, P.N., Vahdat, A.M.: Managing energy and server resources in hosting centers. Proceedings of the eighteenth ACM symposium on Operating systems principles. (2001)
5. Chen, W., Jiang, F., Zheng, W., Zhang, P.: A dynamic energy conservation scheme for clusters in computing centers. Proceedings of the Second international conference on Embedded Software and Systems. (2005)
6. Cherkasova, L., Gupta, D., Vahdat, A.: Comparison of the three CPU schedulers in Xen. SIGMETRICS Performance Evaluation Review, 35(2). (2007)
7. Chung-hsing, H., Wu-chun, F.: A Feasibility Analysis of Power Awareness in Commodity-Based High-Performance Clusters. Proceedings of the 7th IEEE International Conference on Cluster Computing. (2005)
8. Clark, C., Fraser, K., Hand, S., Hansen, J.G., Jul, E., Limpach, C., Pratt, I., Warfield, A.: Live migration of virtual machines. Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation. (2005)
9. Gum, P.H.: System/370 extended architecture: facilities for virtual machines. IBM Journal of Research and Development, 27(6). (1983)
10. Hamilton, J.: Cooperative Expendable Micro-Slice Servers (CEMS): Low Cost, Low Power Servers for Internet-Scale Services. Proceedings of the fourth Biennial Conference on Innovative Data Systems Research. (2009)
11. Hsu, C., Feng, W.: A Power-Aware Run-Time System for High-Performance Computing. Proceedings of the ACM/IEEE conference on Supercomputing. (2005)
12. Laszewski, G.V., Wang, L., Younge, A.J., He, X.: Power-aware scheduling of virtual machines in DVFS-enabled clusters. Proceedings of the IEEE International Conference on Cluster Computing and Workshops. (2009)
13. Miyakawa, D., Ishikawa, Y.: Process Oriented Power Management. Proceedings of the 2nd International Symposium on Industrial Embedded Systems. (2007)
14. Mosberger, D., Jin, T.: httperf - a tool for measuring web server performance. SIGMETRICS Performance Evaluation Review, 26(3). (1998)
15. Nathuji, R., Schwan, K.: VirtualPower: coordinated power management in virtualized enterprise systems. Proceedings of the 21th ACM SIGOPS symposium on Operating systems principles. (2007)
16. Norris, C., Cohen, H.M., Cohen, B.: Leveraging IBM eX5 Systems for Breakthrough Cost and Density Improvements in Virtualized x86 Environments. White paper. (2011) ftp://public.dhe.ibm.com/common/ssi/ecm/en/xsw03099usen
17. Padala, P., Zhu, X., Wang, Z., Singhal, S., Shin, K.G.: Performance evaluation of virtualization technologies for server consolidation. HP Laboratories Palo Alto. (2007) http://www.hpl.hp.com/techreports/2007/HPL-2007-59.pdf

18. Padhy, R.P., Patra, M.R., Sarapathy, S.C.: Virtualization Techniques & Technologies: State of The Art. Journal of Global Research in Computer Science, 2(12). (2011)
19. Pinheiro, E., Bianchini, R., Carrera, E.V., Heath, T.: Compilers and operating systems for low power. In the book Dynamic cluster reconfiguration for power and performance, Kluwer Academic Publishers.
20. Rajamanin, K., Lefurgy, C.: On evaluating request-distribution schemes for saving energy in server clusters. Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software. (2003)
21. Soltesz, S., Potzl, H., Fiuczynski, M.E., Bavier, A., Peterson, L.: Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems. (2007)
22. Pallipadi, V., Starikovskiy, A.: The ondemand governor: past, present and future. Proceedings of Linux Symposium. (2006)
23. Wen, C., He, J., Zhang, J., Long, X.: PCFS: Power Credit Based Fair Scheduler Under DVFS for Muliticore Virtualization Platform. Proceedings of the IEEE/ACM International Conference on Green Computing and Communications. (2010)
24. Motahari-Nezhad, Hamid R and Stephenson, Bryan and Singhal, Sharad: Outsourcing business to cloud computing services: Opportunities and challenges. IEEE Internet Computing, Palo Alto. (2009)