Enforcing CPU allocation in a heterogeneous IaaS

Boris Teabe, Alain Tchana, Daniel Hagimont

▶ **To cite this version:**

Boris Teabe, Alain Tchana, Daniel Hagimont. Enforcing CPU allocation in a heterogeneous IaaS. Future Generation Computer Systems, 2015, vol. 53, pp. 1-12. 10.1016/j.future.2015.05.013 . hal-01343004

HAL Id: hal-01343004
https://hal.science/hal-01343004

Submitted on 7 Jul 2016

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.

Open Archive TOULOUSE Archive Ouverte (OATAO)

OATAO is an open access repository that collects the work of Toulouse researchers and makes it freely available over the web where possible.

This is an author-deposited version published in : http://oatao.univ-toulouse.fr/
Eprints ID : 15389

To link to this article : DOI:10.1016/j.future.2015.05.013
URL : http://dx.doi.org/10.1016/j.future.2015.05.013

To cite this version : Teabe, Boris and Tchana, Alain and Hagimont, Daniel *Enforcing CPU allocation in a heterogeneous IaaS.* (2015) Future Generation Computer Systems, vol. 53. pp. 1-12. ISSN 0167-739X

Any correspondence concerning this service should be sent to the repository administrator: staff-oatao@listes-diff.inp-toulouse.fr

Enforcing CPU allocation in a heterogeneous IaaS

Boris Teabe, Alain Tchana *, Daniel Hagimont

University of Toulouse, Toulouse, France

HIGHLIGHTS

- We identify issues related to changing underlying hardware in public IaaS.
- We propose a solution for DVFS and VM migration across heterogeneous nodes.
- We provide an implementation of our solution within the Xen hypervisor.
- The evaluation of our prototype confirms the accuracy of our solution.

ABSTRACT

In an Infrastructure as a Service (IaaS), the amount of resources allocated to a virtual machine (VM) at creation time may be expressed with relative values (relative to the hardware, i.e., a fraction of the capacity of a device) or absolute values (i.e., a performance metric which is independent from the capacity of the hardware). Surprisingly, disk or network resource allocations are expressed with absolute values (bandwidth), but CPU resource allocations are expressed with relative values (a percentage of a processor). The major problem with CPU relative value allocations is that it depends on the capacity of the CPU, which may vary due to different factors (server heterogeneity in a cluster, Dynamic Voltage Frequency Scaling (DVFS)). In this paper, we analyze the side effects and drawbacks of relative allocations. We claim that CPU allocation should be expressed with absolute values. We propose such a CPU resource management system and we demonstrate and evaluate its benefits.

Keywords:
SLA enforcement
Virtualization
Cloud
DVFS
Heterogeneity

1. Introduction

Nowadays, many organizations tend to outsource the management of their physical infrastructure to hosting centers. This way, companies aim to reduce their costs by paying only for what they really need. This trend is commonly called cloud computing. In this context, virtualization [1] plays an increasing role. A majority of cloud platforms implement the Infrastructure as a Service (IaaS) model where customers buy virtual machines (VM) with a set of reserved resources. This set of resource corresponds to a Service Level Agreement (SLA) [2–4] and customers expect providers to fully meet it. On their side, providers are interested in saving resources (notably energy) while guaranteeing customers SLA requirements.

In such platforms, two main techniques are used by providers for energy management: VM migration [1] and device speed scaling [5–9]. VM migration is used to gather VMs on a reduced set of machines (according to VM loads) in order to switch unused machines off, thus implementing a consolidation [10] strategy. Device speed scaling (for underloaded devices) is also a means to save energy as reducing a device speed generally reduces its consumption. For example, in the CPU case, processor manufacturers have developed a hardware technology called Dynamic Voltage and Frequency Scaling (DVFS). DVFS allows dynamic processor frequency control, and hence, helps in reducing power consumption. DVFS, according to the hosts global CPU load, dynamically scales the processor frequency. From a more abstract point of view, VM migration (between heterogeneous machines) or speed scaling on one machine can be both considered as changing the nature and performance of the underlying hardware.

The amount of resources allocated to a VM at creation time (in a SLA) may be expressed with relative values (relative to the hardware, i.e., a fraction of the capacity of a device, e.g., 30% of a device) [11] or absolute values (i.e., a performance metric which is independent from the capacity of the hardware, e.g., a throughput). Resource allocation expressed with relative values (as a fraction of a device) can be problematic, since the capacity of a device may change as a consequence of the 2 above techniques. The negotiated SLA should not vary according to energy management decisions

* Corresponding author.
E-mail addresses: boris.teabe@enseeiht.f (B. Teabe), alain.tchana@enseeiht.fr (A. Tchana), daniel.hagimont@enseeiht.f (D. Hagimont).

from the hosting center provider. As example, consider two virtual machines VM_1 and VM_2 running on the same physical host (with their respective allocated capacity expressed as a relative value: 30% and 60%), VM_1 being overloaded and VM_2 being underloaded, the host may be globally underloaded leading to a reduction of the processor frequency, which would penalize VM_1. Similarly, if the consolidation policy migrates VM_1 to another machine with a different CPU capacity, VM_1 should not be given 30% of the new machine capacity.

Surprisingly, while disk or network resource allocations are expressed with absolute values (bandwidth) [12], CPU resource allocations are most of the time expressed with relative values (a percentage of a processor) [13], or whenever absolute values are used, the implementation of scheduling transforms them into relative values [14,15] thus losing the benefit from absolute values. Few systems [16–18] addressed this issue as SLA enforcement was mainly considered without DVFS and in homogeneous environments [19].

In this article, we consider resource allocation in a heterogeneous IaaS environment where VM migration and device speed scaling can both be used to save energy. We claim and motivate that resource allocation should be based on absolute values for all devices. This is already the case for disk and network devices. Since current CPU resource allocations fail to implement a truly absolute value based system [14], we propose an absolute allocation unit for CPU and we show that such absolute allocations can be translated into relative values, which are generally understood by today's operating systems schedulers [20–23,14]. Such translations take place when a VM is migrated or when the frequency of the processor is changed. We implemented this resource allocations system in Xen, by improving its default Credit Scheduler. The Credit Scheduler is a fair share algorithm based on proportional scheduling; it relies on a credit system to fairly share processor resources while minimizing the wasted CPU cycles.

We make the following contributions in this paper:

- We analyze resource allocation in public IaaS platforms and identify issues related to changing underlying hardware.
- We propose a solution which addresses the problem raised by processor allocation based on relative values, when DVFS and VM migration (across heterogeneous machines) are enabled.
- We provide an implementation of our solution within the Xen hypervisor.
- We experimentally evaluate our solution in a private IaaS composed of heterogeneous machines with DVFS and VM consolidation enabled. These experiments confirm the effectiveness of our approach. We also show that this is an important issue in popular IaaS platforms such as EC2 and Rackspace.

The rest of the article is structured as follows. Section 2 introduces the context of our work. Section 3 analyzes the issue we are addressing. Section 4 provides an assessment of this issue based on the study and evaluation of existing solutions. Section 5 presents our contribution to this problem. An evaluation of our prototype is reported in Section 6. Then, a review of related works is presented in Section 7. Finally, we present our conclusions and perspectives in Section 8.

2. Context

In the context of the development of cloud computing, IaaS platforms are seeing widespread adoption. Such platforms provide to customers an API for dynamically allocating VMs with a given SLA. The latter generally takes the form of a type of VM (among several pre-defined types). Each type of VM is characterized by the fraction of resource associated with the VM (CPU capacity, memory and disk spaces, network and disk IO capacities). Customers expect providers to fully meet SLAs [3,2].

The goal of the provider, who manages the platform, is to make profit, i.e., to reduce the cost of management of the platform. The resources that are effectively used in the platform are varying a lot. VMs may be allocated and freed dynamically, and the resources used internally by each VM are also variable [24]. Therefore, providers often implement a consolidation strategy which relies on VM migration in order to gather VMs on a reduced set of machines (according to VM loads). This kind of VM packing is a way to host more VMs with fewer servers, and it is also a way to save energy [25,26] by switching unused machines off. Another way to save energy is to adapt the speed of the CPU (actually its frequency and voltage) according to the load. More generally, device speed scaling (for underloaded devices) can be used to save energy [27,28], with different type of device, e.g. network interfaces [29,30] or disks [9,6].

The main issue for a provider is to enforce SLAs while reducing energy consumption. Two elements in the previous techniques may lead to SLA violations:

- a migration of a VM from a source to a target machine, if the two machines are of different types, may result in a loss of performance for that VM.
- a reduction of the speed of a device, if the device is underloaded on the current machine, may result in a loss of performance for VMs running on that machine.

These issues are analyzed in the next section. The rest of this section introduces key technologies we rely on.

2.1. Xen hypervisor

Xen is a virtualization 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 this paper, we are particularly interested in CPU management. In order to share CPU between guest OS, Xen relies on a scheduler called the Credit scheduler. It is a fair share algorithm based on the proportional scheduling. Each VM is assigned a credit value for the VM. The credit value represents the CPU share that the VM is expected to have. Therefore, the VMs should have an equal fraction of processor resources if each VM is given the same credit value.

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.

3. Problem statement

The main problem we are addressing is resource allocation in a IaaS environment which includes varying speed devices. The speed of a device may vary since (i) VMs can be migrated between heterogeneous physical machines and (ii) devices often have a dynamically adjusted speed in order to adapt their energy consumption according to the load [5–7]. In such a context, since resource allocations in a SLA must correspond to a fixed amount of resource, they must be expressed with absolute values.

Therefore, a memory capacity can be expressed in Bytes and an IO capacity can be expressed in Bytes/s for a disk and Bit/s for a network. These metrics are totally independent from any hardware architecture or any operating system. In the next section, we show

that cgroup [13] is able to enforce an allocated network IO capacity expressed in Bit/s.

Regarding CPU capacity allocation, the situation is much more sensitive. If we consider an allocation expressed in MIPS (millions of instructions per second), this is not an absolute computing capacity metric as it depends on the architecture of the processor. A given MIPS capacity would not provide the same performance on a RISC or a CISC processor. But since we are considering the execution of a given VM with a given code format, we can assume that it will be run on processors with the same instruction set (e.g., x86) and MIPS can therefore be considered as an absolute metric. However, schedulers in today's operating systems are not relying on MIPS allocation [14], and it would be very hard (difficult to implement and/or intrusive at execution time [31]) to keep track of executed instructions for each VM.

Many virtualization systems and/or IaaS systems rely on GHz (cycles per second) for CPU allocations. As for MIPS, a GHz allocation depends on the architecture of the processor. But with GHz, even if we assume a unique instruction set, different processor architectures will lead to different performances as these processors may execute a different number of instructions per cycle (e.g., if processor pipelines have different depths). Now, let us consider a homogeneous hardware environment (a single type of machines), but with frequency scaling (DVFS) enabled (3 GHz maximum frequency processors). GHz may seem to be an absolute metric, as when you allocate 1 GHz, you are granted 1 million cycles per second, be the current processor frequency set to 2 GHz or 3 GHz. This is theoretically true, but not when we take a look at the implementation. Today's scheduler's CPU allocation is based on weight or a percentage of the processor time. Most of the time, when a VM is given a 1 GHz CPU capacity on a 3 GHz processor, the VM is configured to be given 33% of the processor time. Therefore, this allocation is not absolute, but relative to the speed of the processor (if the processor frequency is scaled down to 2 GHz, the VM will get 33% of 2 GHz).

Overall, our analysis of the situation is that:

- memory, disk and network IOs can be allocated with absolute values and their allocations can be effectively enforced with today's operating system mechanisms that are also present in today's VM hypervisors.
- CPU allocations are most of the time (always according to our knowledge) expressed with relative values, and if they are expressed with absolute values their enforcement implementation makes them relative.
- MIPS or MFLOPS would be ideal absolute metrics for CPU allocation, but they would be very difficult to implement and/or intrusive in today's kernels.

In the next section, we validate our analysis of the situation with actual measurements of network IOs and CPU.

4. Problem assessment

This section presents a set of experiments that we performed in order to assess the problems highlighted earlier.

4.1. Experimental context

All the experimental results reported in this section were gathered on three heterogeneous physical machines (PMs) whose characteristics are presented in Table 1. All the PMs are configured with a single processor/core. When necessary, they are virtualized with the Xen system, version 4.1.0. All VMs are always configured with a single vCPU. Each result is an average of 10 executions. For all these experiments, we used the π-app CPU intensive benchmark and we computed the tenth decimal of π. All the

Table 1
Experimental PMs.

(P_0) DELL	Intel Core 2 Duo CPU E7300 @ 2.66 GHz	Ubuntu 12.04
(P_1) HP	Intel Core i7-3770 CPU @ 3.40 GHz	–
(P_2) ASUS	AMD A6-6400K APU 3.919 GHz	–

PMs were configured (when necessary) to use the on-demand governor [32] DVFS policy. This governor changes the frequency between the lowest level (when CPU usage is less than a threshold) and the highest level (when CPU load is higher). We configured the threshold to 40% for all experiments.

4.2. The heterogeneity problem

As we argued in Section 3, processor allocations based on GHz are relative to the architecture of the hardware when the IaaS is heterogeneous. Fig. 1 shows the results of the experiments we performed to illustrate that. Firstly, we show that running an application (π-app) on two heterogeneous processors may lead to different execution times even if these processors provide the same GHz capacity. The left histogram in Fig. 1 shows the results with the two PMs P_0 and P_1 (we configured their processor to run at the same speed, 1.6 GHz). We observe a difference of performance of about 21% between the two machines. The right curves in Fig. 1 present the results when the experiments are performed on VMs. For different CPU capacities expressed in GHz, we allocate a VM and translate this capacity in credits in the Xen scheduler (as described in the previous section). The top right figure presents the execution times when we keep the same credit translation regardless the type of the machine. Obviously the execution times vary according to the type of the machines. Notice that execution times on P_3 (the ASUS/AMD) are higher than on P_1 (HP) while P_3 (3.9 GHz) has a higher frequency than P_1 (3.40 GHz). This confirms that GHz cannot be considered as an absolute allocation unit and should at least be adapted according to the frequency of the processor. The bottom right figure presents the results when the credits in Xen are computed according to the processor frequency of the machine (a rule of three is applied). Even in this case, performance significantly depend on the processor architecture. A smarter translation scheme would be required.

4.3. The speed scaling problem

This section illustrates the effect of dynamic device speed scaling on relative value based resource allocation.

4.3.1. The problem may exist for network

Knowing that commonly used solutions for network are based on absolute values, we implemented a solution which relies on relative values in order to illustrate the importance of absolute values. We updated the original implementation of tc (traffic control in cgroup [13]) in order to introduce the notion of relative values (instead of absolute values). To this end, we consider that the network card is able to send/receive each second a fixed number of packets, seen as a buffer with a fixed length. A relative value for a VM corresponds here to a percentage of that buffer. The network scheduler (HTB[1] in our case) wakes up every second and fills the buffer with VMs requests, according to their relative value. If a VM has more requests than its buffer's ratio, the remaining requests are kept and delayed until the next scheduling time. We also implemented a dynamic speed scaling which adapts the speed of the network card according to the global utilization of the buffer, which corresponds to the global saturation of the card. A modification of the card speed (its throughput) modifies the

[1] Hierarchical Token Bucket.

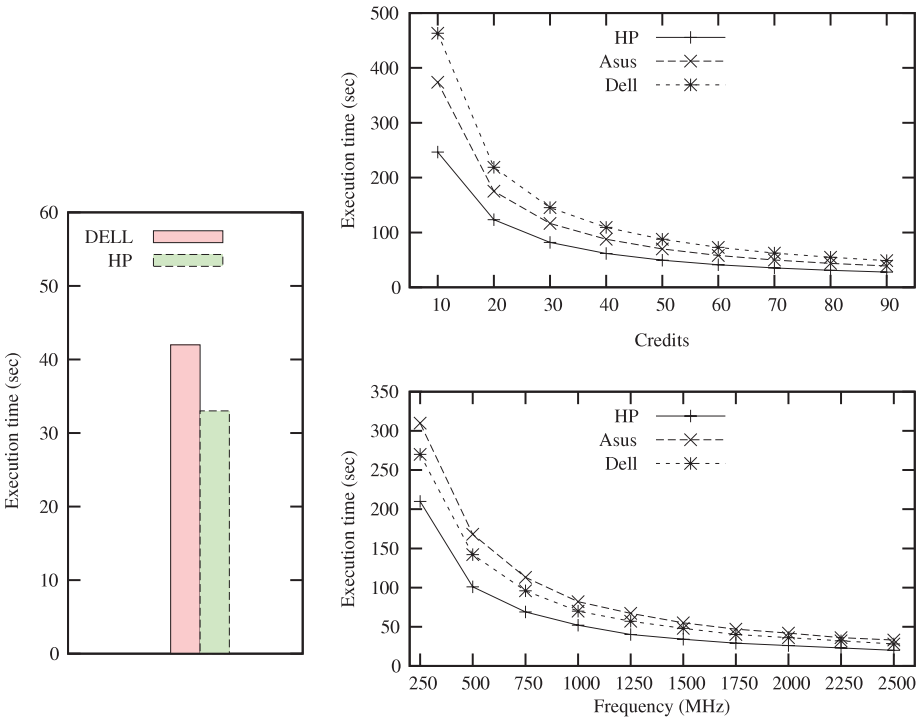

Fig. 1. CPU capacity booked for a VM in terms of GHz is not always guaranteed when dealing with heterogeneous machines or relative values.

Table 2
Experimental context to illustrate that the implementation of network allocation may based on relative values.

PMs	2 PMs: first one for requests injection (CLIF [34]), and the second hosts VMs
Network cards	2 levels of speed 1000 and 100 Mb/s
HTTP workload	Constant
Micro bench workload	Constant upper load and constant lower load (as a decreased stair)
Metrics	TPC-W throughput

size of packets, but does not modify the number of packets in the buffer.

For this experiment, we used the TPC-W [33] benchmark and the PMs P_0 and P_1 (equipped with the same type of network card). We defined a scenario with two VMs, the first VM hosting TPC-W servers (web and database) and having 10% ratio, while the second VM runs a micro network benchmark with 80% ratio. 10% ratio is reserved for the Xen dom0 (the host system). TPC-W servers receive a constant workload while the second VM receives a two phases workload: constant upper level workload until the middle of the experiment and then a very low level until the end of the experiment. When the load falls at the end of the first phase, the network card speed is decreased. The first VM (with TPC-W) is our indicator and the SLA violation will be observed on its throughput. Table 2 summarizes the parameters of this experiment. Fig. 2 shows the comparison of both implementations (original tc and the modified one). Note that the low peak in the figure occurs during the modification of the speed of the network card. It corresponds to the time needed by the network card driver to reconfigure itself. We can see that the SLA of TPC-W is maintained when the scheduler is based on absolute values while it is not the case with relative values. In the latter case, the throughput of TPC-W is affected (reduced) by the reduction of the speed of the network card. Indeed, the VM is assigned a ratio of the buffer which limits the number of packets the VM can send/receive. Since the card speed reduction reduces the size of packets, the throughput of the VM is also affected. The original tc uses as many packets as needed to enforce the throughput specified with an absolute value.

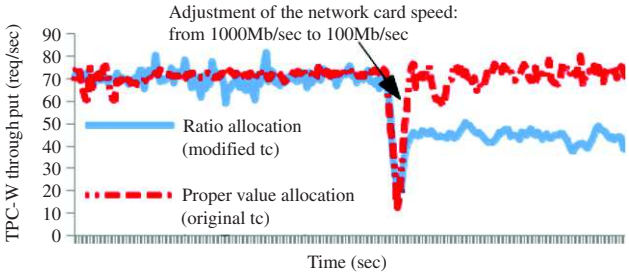

Fig. 2. Resource allocations based on relative and absolute values facing variable device speed.

4.3.2. The problem exists with the CPU

In this section, we allocate CPU capacities to VMs as relative values and we want to illustrate the issues due to frequency scaling (DVFS). Let us consider our two PMs P_0 and P_1 virtualized with Xen. Let us consider the allocation of four VMs, $VM20_1$, $VM70_2$, $VM20_3$, $VM70_4$. We assume that $VM20_1$ and $VM20_3$ have the same capacity, and so do $VM70_2$ and $VM70_4$.

The IaaS initially distributes the VMs on PMs and computes their corresponding credits for the scheduler (to take into account the different frequencies of P_0 and P_1). $VM20_1$ and $VM70_2$ are deployed on P_0 and use respectively 20% and 70% of its capacity, $VM20_3$ and $VM70_4$ are deployed on P_1 and use respectively 15% and 55% of its capacity (P_0 is more powerful than P_1). Fig. 3 shows the workload run by each VM during the experiment. The different peak load phases of $VM20_1$ are equal. Note that the CPU load shown for each VM is the contribution of the VM to the global CPU load of its PM, e.g., 20 for $VM20_1$ implies that it uses all its capacity. At time "a" and "b", $VM70_2$ and $VM20_3$ respectively end their job. We assume that when a PM has all its VMs at very low level in terms of CPU activity (less than 5%), the consolidation system will move them to another PM which can run them (its CPU load is so that it can accept the load of the incoming VMs). Thus at time "c" $VM20_1$ and $VM70_2$ are migrated from P_0 to P_1 in order to switch P_0 off. In this experiment, reducing the processor speed on P_0 slows down the processor by 50%.

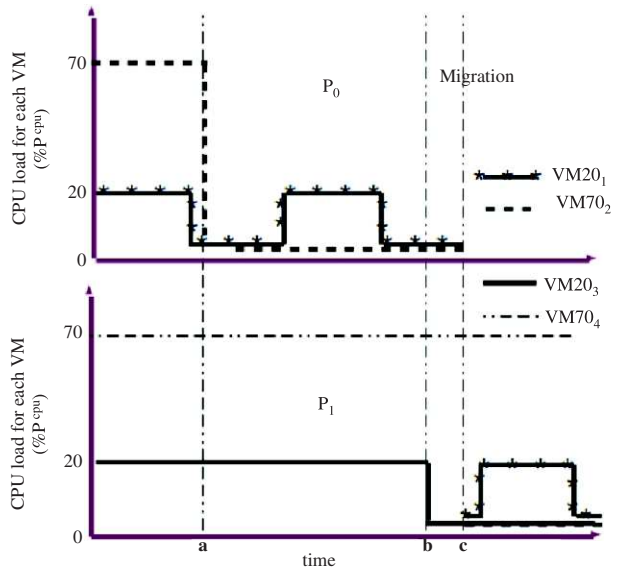

Fig. 3. Global scenario to illustrate the effects of CPU allocation based on relative values when DVFS is enabled or when migrating VMs across heterogeneous PMs.

Fig. 4 presents the monitored load of each VM. It is interpreted as follows:

- At time "a" the DVFS manager decreases the speed of P_0's processor since its global utilization falls under the threshold (40%). This operation leads to performance degradation on $VM20_1$: the second peak load phase of $VM20_1$ is larger than the first one (the expected duration). The SLA is not respected.
- As we shown in Section 4.2 the initial deployment of $VM20_3$ and $VM70_4$ on P_1 impacts their performance (as the computation of their credits based on the processors' frequencies is not correct). This can be observed on $VM20_3$ which ends its jobs earlier (before time "b") than expected. The provider does not optimize his infrastructure. After time "c" where $VM20_1$ is migrated to P_1, we observe the same phenomenon (its final peak is shorter than its first one).

4.4. The problem in popular IaaS

We ran the π-app application in different IaaS platforms. Regarding the DVFS problem, there is no way to externally monitor the activity of DVFS governors on IaaS's PMs. Therefore, we only investigated how they address the problem of hardware heterogeneity. All results reported here have been validated with several executions.

Rackspace: Fig. 5 top presents the results of this experiment, performed in Virginia, Dallas and Sydney with a standard VM instance type. In the morning, the VMs in Virginia and Dallas provide the same performance, which is not the case in Sydney (up to 34% of difference). We can explain that by the heterogeneity of the processors. We found (using */proc/cpuinfo*) that VMs in Virginia and Dallas used the same type of processor for this particular experiment: Intel Xeon E5-2670 2.6 GHz. It was an AMD Opteron 4332 in Sydney. We repeated the experiments at a different time (afternoon) of the day and we observed different types of processors as in the first time: AMD Opteron 4332 in Virginia and Sydney, and AMD Opteron 4170 in Dallas. According to these results, Rackspace does not address the problem raised by the heterogeneity of processors. Referring to the documentation they provide, the allocation unit for CPU is a vCPU. No more information is given about the real computing capacity of this unit. The actual computing capacity of a VM on this IaaS depends not only on the VM type (for each type of VM corresponds a number of vCPU), but also on its location, as we experimented.

Amazon EC2: It is the only IaaS (to the best of our knowledge) which attempts to solve the issues we address. It proposes a unique allocation unit for CPU which is claimed to be independent from the hardware: ECU. For instance, a m3.medium VM type is configured with 3ECU. Since we do not have access to ECUs implementation, our evaluations study its ability to be consistent (a small VM should lead to less performance than a big one, the same type of VM should lead to the same performance whatever the context). To this end, we ran in EC2 two VM types: m3.medium and t1.micro. We performed the same experiment within different geographical zones (Oregon and Ireland). Fig. 5 bottom presents the results of these experiments. From Fig. 5 bottom left we can see that EC2 provides the same performance for m3.medium whatever the context (zone, type of virtualization, and VMs colocation). This can

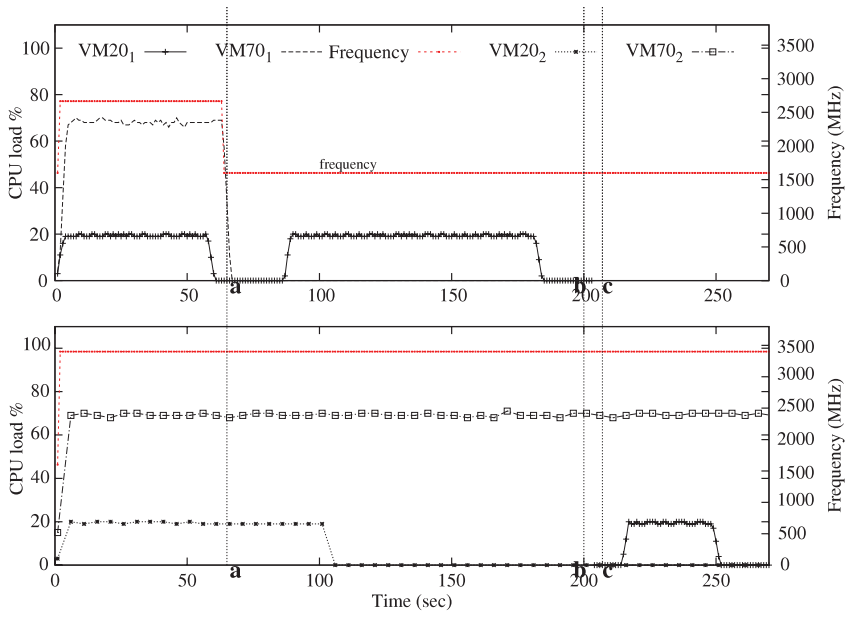

Fig. 4. The effects of CPU allocation based on relative value when DVFS and VM migration are performed.

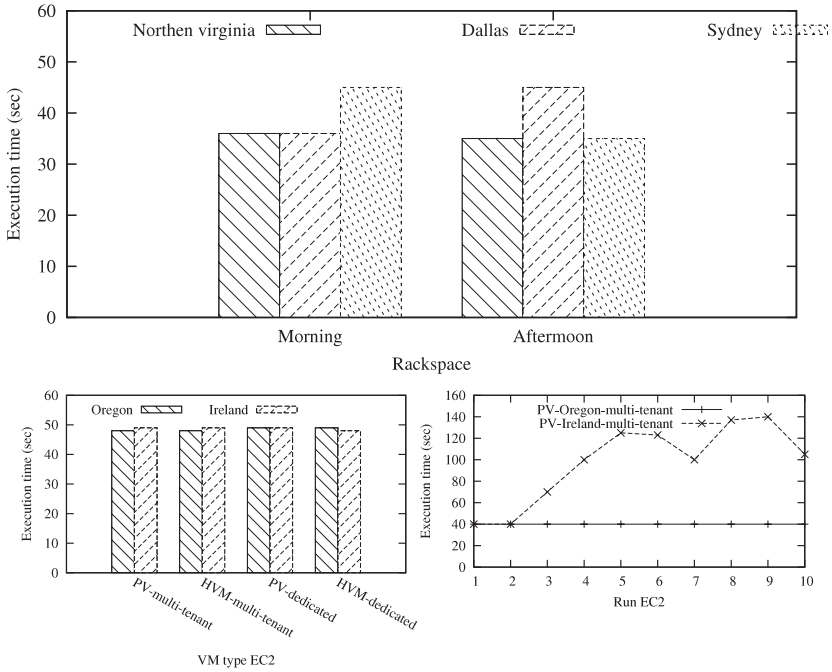

Fig. 5. The capacity booked for a VM is not always satisfied in Amazon EC2 and Rackspace.

be explained by the fact that EC2 always runs that type of VM on the same type of processor (Intel Xeon E5-2670v2 2.5 GHz) where the single vCPU of the VM is always pinned to a dedicated hyperthread. This strategy is announced in advance by EC2. By doing that, EC2 avoids the heterogeneity problem for the m3.medium type. However, this approach may limit VM consolidation possibilities. For example, a physical machine without an Intel Xeon E5-2670v2 2.5 GHz processor cannot host a VM of type m3.medium even if the physical machine has enough resources.

However, regarding the t1.micro VM type, the results (Fig. 5 bottom right) show that EC2 provides no guarantee at all. Although performance is stable in Oregon, we have an uncertain behavior in Ireland. According to EC2 documentation, a t1.micro can run with up to 2ECU, with one vCPU. Unlike the m3.medium type, no information is given about the mapping of that vCPU onto real resources. For this type of VM, EC2 reserves itself the right to use heterogeneous processors and also the right to allocate one or two ECU. Thus, the capacity of that type of VM is unknown. This is not the problem we want to highlight since this behavior is announced in advance to clients. But, what is even more surprising is the fact that t1.micro instances can provide better performance (execution time of about 40 s in Oregon) than m3.medium instances (execution time of about 47 s). VMs of type m3.medium are always allocated 3ECU while t1.micro instances receive 2ECU at maximum. A VM with 3ECU should be at least one-third more powerful than a VM with less than 2ECU. Such an inconsistent behavior makes performance predictability very sensitive. One possible explanation is that the t1.micro instance in our experiment (within Oregon) was deployed on a more powerful PM than an Intel Xeon E5-2670 hyperthread (used for m3.medium instances) so that the translation of 2ECU on that powerful machine leads to more performance than the translation of 3ECU on an Intel Xeon E5-2670 hyperthread.

5. Contributions

This paper addresses the issue raised by CPU allocation when dealing with variable speed processors in a IaaS (heterogeneous or homogeneous processors with DVFS). Although our contributions can be generalized to multi-core systems, we consider in this paper IaaS with single-core machines. This section is organized as follows. We first introduce useful notations that are used in the rest of the section. We then present a resource mapping (of booked capacities onto real resources) solution based on absolute values which guarantees the SLA.

5.1. Model

5.1.1. IaaS model

In this paper we assume there are m PMs in the IaaS denoted by P_i, where $1 \leq i \leq m$. To represent the heterogeneity of the IaaS, we note $P_{i(t)}$ to say P_i is of type "t". In addition we consider the fact that a PM can run at different CPU speeds. Frequencies are discrete values. We note $F_i = \{f_i^j, 1 \leq j \leq \max, f_i^j < f_i^{j+1}\}$, the set of available frequencies on P_i.

According to these notations, the activity of DVFS on P_i can change the frequency from f_i^{cur} to $f_i^{cur'}$. Thus, when a DVFS governor (e.g. on-demand governor [32]) is enabled on a PM, that PM should be noted $P_{i(t,f_i^{cur})}$: P_i is of type "t" and runs at frequency f_i^{cur}. From a more abstract point of view, the IaaS manages different PMs $P_{i(t,f_i^{cur})}$ and both VM migration and frequency scaling are operations which modify the PM which hosts a VM. To ease reading, PMs will be noted $P_{i(cur)}$ knowing that P_i itself tells us about its type.

5.1.2. Resource selling model

A CPU allocation capacity is noted C_b. We claim that IaaS providers should not use real units of allocation (e.g. GHz) when dealing with customers. This is motivated by the fact that the type of the PM which will run a VM is not always known in advance at negotiation time (when the customer expresses his needs) and may change during the lifetime of the VM. For these reasons, we propose to define a unique unit of allocation which is independent from any type of hardware. We call it "Virtual Unit (VU)". In addition, the latter should make sense to the customer so that he knows without any ambiguity to which capacity it corresponds. Therefore, a VU should be illustrated by its capacity to run some

well known benchmarks. A computation capacity C_b for a VM is an amount of VUs, which may belong to a set of IaaS proposed capacities (VM types or sizes).

Because resources are shared between VMs on a PM, the main questions we need to answer (considered in this paper as the SLA enforcement problem) are the following: How to translate VU onto PM resources and how to guarantee the constancy of VU allocations as the translation should take into account the change of PM during the VM lifetime?

5.2. Virtual resources translation to PMs

5.2.1. Overall design

Our translation scheme is based on a reference PM P_{ref} (P_{ref} can be an arbitrarily PM type chosen from the IaaS). P_{ref} defines an absolute CPU capacity and a VU is defined as a fraction of this capacity (a relative value of an absolute value is an absolute value). From now on, such a fraction is called a credit.

The main problem we need to address is to guarantee that an allocated computation power stays the same whenever the VM moves to different PMs in the IaaS: this is how we define "absolute allocations" which do not depend on any PM. The main idea behind our solution is to rely on the calibration of the IaaS using a reference CPU intensive benchmark (e.g. SPEC CPU2006 or SPLASH-2 LU [35]) that we run on each PM. We internally consider the execution times of that benchmark in our resource management system (see sections below). This solution is not intrusive comparing to a solution which continuously monitors the execution of each VM in order to count executed instructions. Based on this calibration, we translate the allocated VUs (a credit on $P_{ref(max)}$) to its equivalent on the target PM (a credit on that PM) either at VM creation time, or at migration time. Also, we dynamically translate the credits of each VM whenever the processor frequency is modified by DVFS.

Therefore, we propose a three level translation solution described below, which guarantees the VU allocations.

5.2.2. First level translation

Firstly, given a booked CPU capacity C_b (an amount of VU), we provide a first level translation function $V2R$ defined as: $V2R(C_b) = C$ where C corresponds to the credit on P_{ref} that corresponds to the allocated VUs.

5.2.3. Calibration of the IaaS

This operation is performed only once for each type of PM in the IaaS (when a PM with a different architecture is introduced in the IaaS). We chose a CPU intensive benchmark and we determine its execution time for each credits (sizes of VM) allowed by the provider and for each type of PM in the IaaS. The calibration is done at f_i^{max} for any P_i (including P_{ref}). At the end of this step we have $T_{i(max)}^{C'}$ which gives the execution time of the benchmark within a VM with credits C' when it runs on top of the PM $P_{i(max)}$. As motivated earlier, we are addressing the problem of CPU allocation. Therefore the calibration benchmark should respect the following characteristics: (1) it should be CPU bound only, (2) all the data it uses should fit within cache memories. According to these characteristics, the Cloud provider can either write its own benchmark or rely on a well configured reference benchmark such as the SPLASH-2 LU [35] application as we have done.

5.2.4. Second level translation

If C is the translated credits on P_{ref}, corresponding to the capacity booked for a VM, and the IaaS manager (the module of the IaaS which is responsible for the placement of VMs on PMs) chooses PM P_i to run that VM (first instantiation of the VM or migration), we first compute credits C' to give to that VM on P_i as if the latter ran at its maximum frequency f_i^{max}. To do that we rely on the results of the calibration. The execution time of the benchmark with credit C on P_{ref} should be the same as with credit C' on $P_{i(max)}$. So, we choose C' such that:

$$T_{i(max)}^{C'} = T_{ref}^{C}. \tag{1}$$

5.2.5. Third level translation

When the speed of the processor is modified on a PM, we need to recompute the credits associated with VMs on that PM in order to counterbalance the effect of the frequency modification. We need to compute the actual credit C'' on P_i which takes into account its actual frequency f_i^{cur}, set by the DVFS activity. C'' should be computed so that "**absolute allocation constraint**" is enforced:

$$T_{i(cur)}^{C''} = T_{i(max)}^{C'} = T_{ref(max)}^{C}. \tag{2}$$

We implemented a *Power Aware Scheduler* (*PAS* for short) at the hypervisor level (although it could have been implemented separately). Actually it is implemented as an extension of the Xen Credit scheduler [14], which is the default and most achieved VM scheduler. It computes and sets at each scheduling tick the new credits (C'') associated with VMs. The computation of a new credit relies 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 [5].
- proportionality of credit and performance. This property means that if we modify the capacity allocated to a VM, the impact on performance is proportional to the change of the capacity.

We validated these proportionality rules in one of our previous work [19].

Proportionality of frequency and performance.

This proportionality is defined by:

$$\frac{T_{i(max)}^{C'}}{T_{i(cur)}^{C'}} = \frac{f_i^{cur}}{f_i^{max}} \times cf_{i(cur)} \quad (cf_{i(cur)} \text{ is close to 1}) \tag{3}$$

which means that on PM P_i, if we decrease the frequency from f_i^{max} down to f_i^{cur}, the execution time of a VM with credits C' will proportionally increase from $T_{i(max)}^{C'}$ to $T_{i(cur)}^{C'}$. For instance, if f_i^{max} is 3000 MHz and f_i^{cur} is 1500 MHz, the frequency ratio is 0.5 which means that the processor is running at half of its capacity at f_i^{cur} compared to f_i^{max}. So if we consider a program with execution time $T_{i(max)}^{C'} = 10$ s at f_i^{max}, its execution time $T_{i(cur)}^{C'}$ at f_i^{cur} should be $\frac{10}{0.5} = 20$ s. We define the frequency ratio on P_i as $ratio_{i(cur)} = \frac{f_i^{cur}}{f_i^{max}}$.

Even if $cf_{i(cur)}$ is very close to 1, we keep it in our equations as we observed that it may vary according to the machine architecture and the considered frequency f_i^{cur}.

Proportionality of credit and performance.

This proportionality is defined by:

$$\frac{T_{i(f)}^{C'}}{T_{i(f)}^{C''}} = \frac{C''}{C'} \times cs \quad (cs \text{ is close to 1}) \tag{4}$$

which means that if we increase the credits of a VM from C' up to C'' on $P_{i(f)}$, the execution time of that VM will proportionally decrease from $T_{i(f)}^{C'}$ to $T_{i(f)}^{C''}$. 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. As for the first proportionality, we have a variable *cs* introduced in the equation. It is very close to 1 and may vary according to the implementation of the scheduler which schedules VMs according to their credits. As we observed that $cs = 1$ in the case of Xen credit scheduler, we ignore *cs* in the rest of the paper.

Credits C″ computation in response to DVFS.

Eqs. (3) and (4) are used to compute the modification of VM credits, which can compensate the performance penalty incurred by a frequency modification. Based on C' (the translated credits for P_i at f_i^{max}), we provide in this section a way to compute the credits C'' to give to the VM, taking into account the actual frequency f_i^{cur} of P_i. Remember that C'' should be computed so that the absolute allocation constraint is respected (Eq. (2)).

According to Eq. (4), $C'' = \frac{T_{i(cur)}^{C'} \times C'}{T_{i(cur)}^{C''}}$.

According to Eq. (3), $T_{i(cur)}^{C'} = \frac{T_{i(max)}^{C'}}{ratio_{i(cur)} * cf_{i(cur)}}$, and $T_{i(cur)}^{C''} = \frac{T_{i(max)}^{C''}}{ratio_{i(cur)} * cf_{i(cur)}}$

$\Rightarrow$

$$C'' = \frac{T_{i(max)}^{C'} \times C'}{T_{i(max)}^{C''}} \tag{5}$$

but from Eq. (3), $T_{i(max)}^{C''} = T_{i(cur)}^{C''} \times ratio_{i(cur)} \times cf_{i(cur)}$ and we want $T_{i(cur)}^{C''} = T_{i(max)}^{C'}$ (the absolute allocation constraint)

$\Rightarrow$

$$C'' = \frac{C'}{ratio_{i(cur)} * cf_{i(cur)}}. \tag{6}$$

5.2.6. Calibration step optimization

Remember that the objective of calibration is to provide a way to determine the credit C' of a VM (which corresponds to an allocated credit C on $P_{ref(max)}$) on a PM $P_{i(max)}$ (chosen by the IaaS manager).

The solution we proposed earlier (called brute force calibration) has a complexity of $O(\#credits \times m)$, with #credits be the number of available VM sizes (with different credits) the IaaS provider allows, and m the number of PMs. This solution is fastidious for two reasons. Firstly it needs to calibrate all the available credits for all types of PMs. Secondly if during the lifetime of the IaaS the provider needs to consider a new credit (corresponding to a new type of VM), he will need to calibrate this credit for all the types of PM. Here we propose an optimization (called direct calibration) which requires a reduced number of calibration, with a complexity of $O(m)$. This method consists in choosing arbitrarily a unique credit C_{ref} to calibrate, called the reference credit. The calibration (as described in Section 5.2.3) is only performed with this credit. Thus we have $T_{i(max)}^{C_{ref}}\ \forall\ i \in \{1, \ldots, m\}$. The calibration of another credit C' is done as follows.

According to Eq. (4):

$$\frac{T_{ref(max)}^{C}}{T_{ref(max)}^{C_{ref}}} = \frac{C_{ref}}{C} \tag{7}$$

$\Rightarrow$

$$T_{ref(max)}^{C} = \frac{C_{ref} \times T_{ref(max)}^{C_{ref}}}{C} \tag{8}$$

$$\forall\ i \in \{1, \ldots, m\} \text{ we have } \frac{T_{i(max)}^{C'}}{T_{i(max)}^{C_{ref}}} = \frac{C_{ref}}{C'} \tag{9}$$

$\Rightarrow$

$$T_{i(max)}^{C'} = \frac{C_{ref} \times T_{i(max)}^{C_{ref}}}{C'}. \tag{10}$$

Since we want to guarantee absolute allocation (Eq. (1): $T_{i(max)}^{C'} = T_{ref(max)}^{C}$), from Eqs. (8) and (10) we have:

$$T_{i(max)}^{C'} = T_{ref(max)}^{C} = \frac{C_{ref} \times T_{ref(max)}^{C_{ref}}}{C} = \frac{C_{ref} \times T_{i(max)}^{C_{ref}}}{C'}. \tag{11}$$

Therefore

$$C' = \frac{T_{i(max)}^{C_{ref}}}{T_{ref(max)}^{C_{ref}}} \times C. \tag{12}$$

Computing C' using Eq. (12) is the fastest solution and provides the same results.

In summary, from Eqs. (6) and (12), the actual capacity C'' which takes into account the actual frequency f_i^{cur} on P_i is given by the following equation

$$C'' = \frac{T_{i(max)}^{C_{ref}} \times C}{T_{ref(max)}^{C_{ref}} \times ratio_{i(cur)} \times cf_{i(cur)}}. \tag{13}$$

6. Evaluations

This section presents both the evaluations of our solution applied in a private hosting center and an evaluation of the ability of two popular IaaS (Amazon EC2 and Rackspace) to guarantee a SLA. The experimental context of these evaluations is the same as presented in Section 4.1 (with P_{ref} be P_0). As mentioned in the previous section, a prototype of our solution (called PAS) is implemented within the Xen credit scheduler. This implementation incurs a low overhead and a good scalability. Indeed the complexity of our solution is O(#VMs) and given that the number of VMs a machine can host simultaneously is not usually high, the CPU time required by our solution to compute new credits is negligible (time for a division).

6.1. VM capacities calibration

The first type of experiment compares and validates the two solutions (brute force solution (S1) and the direct solution (S2)) that we proposed for the calibration. Fig. 6 presents the results of these experiments. For each curve, X1-axis presents the considered credits on P_{ref}, X2-axis presents the computed credits on P_1 which should lead SPLASH-2 LU to the same execution time, and Y-axis presents the execution time of the benchmark. Corresponding credits are computed using the considered solution ((S1) or (S2)). For each credit on X1-axis, the reference curve (solid line) gives the execution time on P_{ref}, and the other curves (for (S1) and (S2)) give the execution times on P_1 with the computed credits. These results confirm the fact that solutions (S1) and (S2) (calibrated with a single credit $C_{ref} = 90$) are equivalent and accurate.

6.2. Computation of $cf_{*(*)}$

The second type of experiments is dedicated to the computation of $cf_{*(*)}$ for the three types of PMs we used (DELL, HP, and ASUS). Recall that $cf_{i(f)}$ represents the variable which corrects the imperfection of the proportionality of frequency and performance on P_i at frequency f. The results of these experiments are reported in Table 3. We can see that except $cf_{1(*)}$ (Intel core i7-3770), the other variables are very close to 1.

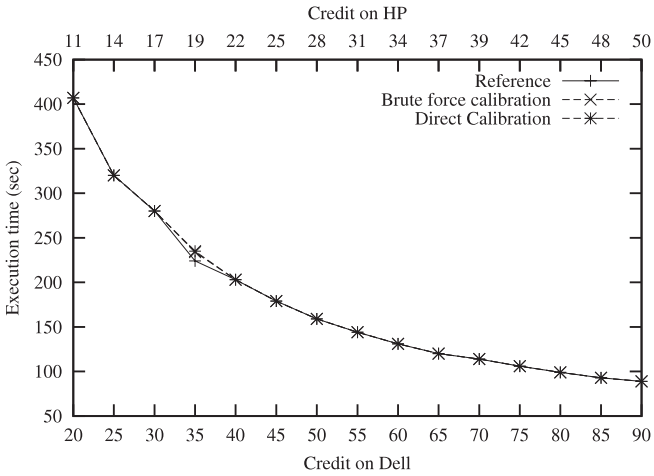

Fig. 6. VMs capacities calibration on P_1 using the two solutions we proposed.

Table 3
Computation of $cf_{*(*)}$.

PM	Freq. (GHz)	$cf_{*(*)}$	PM	Freq. (GHz)	$cf_{*(*)}$
P_{ref}	2.667, 2.133, 1.6	1	P_1	3.4–3.3	0.87
	2.4, 1.867	0.99		3.1–2.2	0.86
P_2	3.9	1		2–1.7	0.85
	1.8	0.96		1.6	0.84
	3.6, 3.1, 2.2	0.95			
	2.6	0.96			

6.3. Heterogeneity and DVFS

The third type of experiment validates the effectiveness of our solution. We ran the workload scenario (Fig. 3) presented in Section 4.3 where the Xen credit scheduler is replaced by our PAS scheduler (which implements our solution) and a IaaS manager which gathers VMs with migrations whenever it can free a PM. In this scenario, P_0 is the reference machine, P_{ref}. Therefore, any CPU capacity is booked relatively to P_{ref}. Fig. 7 presents the results of these experiments. The first two curves in Fig. 7 show the expected results, corresponding to the execution of the scenario on two machines of the same type as P_{ref}, without DVFS (homogeneous environment). The last two curves in Fig. 7 show the results of the experiment when it is performed under the same conditions as in Section 4.3 (two PMs of different types, with our solution).

We can see that the experiment runs within the same expected time. $VM20_3$ and $VM70_4$ are assigned the appropriate credits on P_1 which is more powerful than P_{ref} (resp. 11 and 40). This allows the provider to avoid resource waste as seen in Section 4.3. Regarding our $VM20_1$, it is assigned much more credits (from 20 to about 35) when the DVFS decreases the speed of the processor on P_{ref} (because of the termination of $VM70_2$'s job at time "a"). After the migration of $VM20_1$ and $VM70_2$ to P_1 (at time "c"), their credits are recomputed. We can see that during the experiment, the length of each upper peak load of $VM20_1$ is the same (about 55 s) as expected. The sporadic peaks observed on P_1 frequency curve are explained by the fact that the CPU load on that machine is near the DVFS threshold (40%). This is not the case on P_{ref}. In summary, we can see that our solution addresses the problems we identified in Section 3.

6.4. Comparison with popular IaaS

As argued in Section 4.4, Rackspace does not address the problem of hardware heterogeneity. Referring to Rackspace documentations, the allocation unit for CPU is a vCPU. No more information is given about the real computing capacity of this unit. The actual computing capacity of a VM on this IaaS depends not only on the VM type, but also on its location, as we experimented. Amazon EC2 proposes ECU as allocation unit (Section 4.4). In order to provide performance guarantee for some of its VM types (m3.medium), EC2 always run them on the same type of processor. This approach may significantly limit VM consolidation possibilities.

7. Related work

We discuss related work with respect to different aspects of cloud computing and resource allocation and place them in the context of our work: SLA enforcement.

SLA model. The benefits of Cloud computing come at the cost of fully trusting cloud providers. [36] presents a framework for building collaborative applications that run within an untrusted environment, while meeting SLA constraints. Our work is part of a larger body of work (e.g. [3,4,37]) on SLA enforcement in shared hosting centers. [2,38] propose a cloud computing negotiation model which formalizes the negotiation process between the customers and the provider. In our case, we focus on the selling model (i.e., the metric used in the SLA). [17] presents Quasar, a cluster manager where resource reservation are left to the responsibility of the provider (instead of the customer itself). He allocates just enough resource to meet customers application QoS. [39] uses the same approach. Neither resource presentation to customers nor booked capacity translation onto real resources (as we did) are considered by these works.

Resource accounting in the Cloud. SLA enforcement as we studied in this paper for CPU could be addressed by effectively counting CPU cycles used by each VM. [40] presents ALIBI, a monitoring tool which counts effective memory and CPU cycles used by a guest VM. Monitoring stats are sent to the customer so that he is able to check if his VM behaves consistently (detecting intrusions or SLA violations). [41,42] present similar works. [42] presents an implementation of accountable VM and hypervisor. There have been many efforts to add support for hardware event counters in virtualized environments [43–47]. In our research work, some of these tools could be used by the provider in order to implement an absolute CPU allocation metric but such tools introduce non negligible overhead and they often depend on a particular type of hardware.

Relative and absolute values based scheduling. Many scheduling algorithms use relative values [11] to allocate resources [48,49,14, 20–23]. [24] studies IO bandwidth reservation in a IaaS. It presents mClock, an IO resource allocation in a hypervisor. mClock supports proportional-share fairness subject to a minimum and a maximum limit on the IO allocations. It combines absolute and relative allocations in order to address workload fluctuation. Absolute values allow it to guarantee the minimum and the maximum limits. [15] does a similar work but relies exclusive on relative values. Unlike [15], mClock may not suffer from the dynamic speed scaling and heterogeneity problems.

Heterogeneity aware resource allocation. Many research works have investigated the problem of heterogeneity in shared hosting centers [50–53]. Heterogeneity can be divided into two categories: hardware and workload heterogeneity. [18] evaluates the impact of assuming a homogeneous data center while it is heterogeneous. It proposes a metric to express the sensibility of an application facing heterogeneity. According to their respective documentation, Amazon EC2 and Microsoft Azure avoid the problem of hardware heterogeneity by dedicating the same type of hardware for each VM of a given type, even if we highlighted a problem of performance guarantee when moving from one geographic zone to another. No information is given about Rackspace where

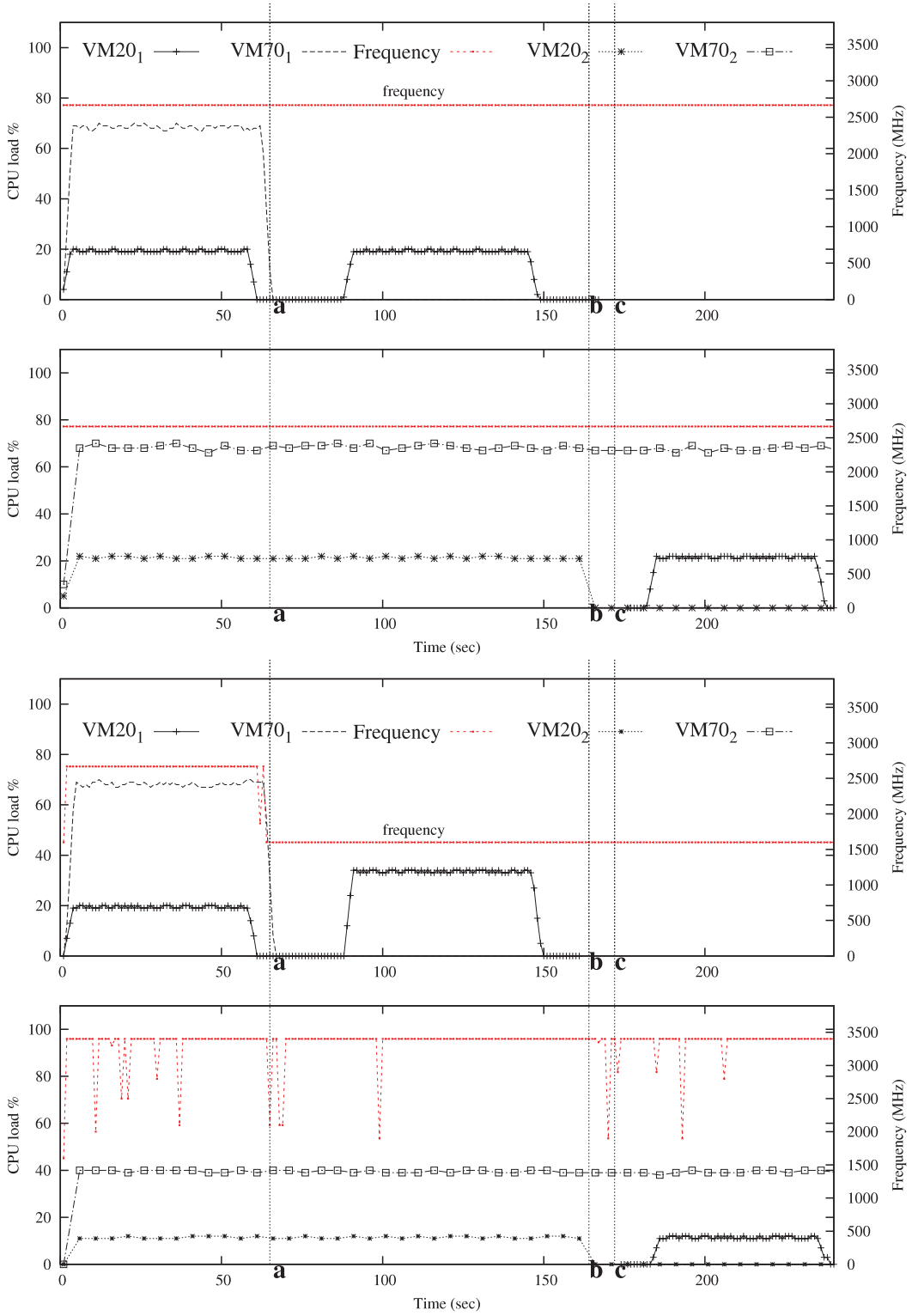

Fig. 7. Two left curves (top and bottom): execution of the scenario on two P_{ref} PMs (baseline). It corresponds to what we expect for each VM during its lifetime in the IaaS. Two rightmost curves: evaluation of our solution.

we encountered the same problem. [54] studies all the facets of heterogeneity in EC2 and observes the same behavior as us. Instead of providing a solution to guarantee performance, [54] proposes a gaming based placement which places VMs according to their analyses of the EC2 environment. [55–57] investigate the same approach. [16] presents Paragon, a QoS-aware scheduling for heterogeneous workload in a datacenter. Its objective is to minimize performance degradation while we present a way to enforce an SLA defined in terms of resources allocated to a VM. [58–60] present similar works.

Speed scaling aware resource scheduling. Dynamic speed scaling is one of the commonly used technology to save energy [27,28, 61,29,30]. Driven on the success of DVFS for processors, [7,8] present a DVFS solution for memory. [62] presents an approach which combines service selection (replicated across many clusters

of the same provider) and dynamic speed scaling in web service systems in order to achieve high energy efficiency while meeting performance requirements. [63] presents CoScale, a system which coordinates CPU and memory power management in order to improve energy savings compared to existing approaches which manage these devices separately. However, none of them investigated the issue of SLA enforcement while relying on variable speed devices.

8. Conclusion

In this paper, we studied resource allocation in a IaaS environment. We showed that existing resource allocation systems which rely on relative values may lead to SLA violations in the context of a IaaS with heterogeneous machines or variable speed devices. While disk or network resource allocations are expressed with absolute values, CPU allocations are expressed with relative values (a percentage of a processor). We proposed an absolute allocation system for CPU and showed how it can be dynamically mapped onto physical resources. We implemented this solution in the Xen virtualization system and evaluated it in a private IaaS composed of heterogeneous machines with DVFS and VM consolidation enabled. These evaluations validated the effectiveness of our solution (no SLA violation). We also showed that this is an important issue in popular IaaS platforms such as Rackspace or EC2. EC2 proposes a unique allocation unit for CPU which is independent from the hardware: ECU, but to guarantee the performance EC2 always runs a type of VM on the same type of processor. This approach reduces VM consolidation possibilities. Rackspace provides vCPU as the allocation unit for CPU. Results obtained with Rackspace showed that they do not address the problem raised by the heterogeneity of processors. In the near future, we plan to generalize our solution for multi-core machines. We are also considering the implementation of such an absolute value CPU allocation based on hardware instruction counters.

References

[1] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, Andrew Warfield, Xen and the art of virtualization, in: Proceedings of the Symposium on Operating Systems Principles, SOSP'03, pp. 164–177.

[2] Linlin Wu, Saurabh Kumar Garg, Rajkumar Buyya, Chao Chen, Steven Versteeg, Automated SLA negociation framework for cloud computing, in: Cluster, Cloud and Grid Computing 2013, CCGrid'13, pp. 235–244.

[3] Lionel Eyraud-Dubois, Hubert Larcheveque, Optimizing resource allocation while handling SLA violations in cloud computing platforms, in: IEEE 27th International Symposium on Parallel & Distributed Processing, IPDPS'13, pp. 79–87.

[4] Jonathan Lejeune, Luciana Arantes, Julien Sopena, Pierre Sens, Service level agreement for distributed mutual exclusion in cloud computing, in: Cluster, Cloud and Grid Computing, CCGRID'12, pp. 180–187.

[5] Kihwan Choi, Ramakrishna Soma, Massoud Pedram, Dynamic voltage and frequency scaling based on workload decomposition, in: Low Power Electronics and Design 2004, ISLPED'04, pp. 174–179.

[6] Tom Bostoen, Sape Mullender, Yolande Berbers, Power-reduction techniques for data-center storage systems, ACM Comput. Surv. 2011 (2011) 45.

[7] Howard David, Chris Fallin, Eugene Gorbatov, Ulf R. Hanebutte, Onur Mutlu, Memory power management via dynamic voltage/frequency scaling, in: ICAC 2011, pp. 31–40.

[8] Qingyuan Deng, David Meisner, Luiz Ramos, Thomas F. Wenisch, Ricardo Bianchini, MemScale: Active low-power modes for main memory, in: ASPLOS 2011, pp. 225–238.

[9] Kester Li, Roger Kumpf, Paul Horton, Thomas Anderson, A quantitative analysis of disk drive power management in portable computers, in: Proceedings of the USENIX Winter 1994 Technical Conference, WTEC'94, pp. 22–22.

[10] Hui Lv, Yaozu Dong, Jiangang Duan, Kevin Tian, Virtualization challenges: a view from server consolidation perspective, in: Proceedings of the 8th ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments, VEE'12, pp. 15–26.

[11] Antonio Nicolo, Efficiency and truthfulness with Leontief preferences. A note on two-agent, two-good economies, Rev. Econ. Des. (2004).

[12] http://lartc.org/howto/lartc.qdisc.classful.html visited on April 2014.

[13] Patrick Bellasi, Giuseppe Massari, William Fornaciari, Exploiting Linux Control Groups for Effective Run-time Resource Management, HiPEAC 2013. https://www.kernel.org/doc/Documentation/cgroups/blkio-controller.txt visited on April 2014.

[14] Ludmila Cherkasova, Diwaker Gupta, Amin Vahdat, Comparison of the three CPU schedulers in Xen, ACM SIGMETRICS Perform. Eval. Rev. 35 (2007) 42–51. SIGMETRICS Performance Evaluation Review.

[15] Ajay Gulati, Irfan Ahmad, Carl A. Waldspurger, PARDA: Proportional allocation of resources in distributed storage access, in: Proccedings of the 7th Conference on File and Storage Technologies, Usenix FAST'09, pp. 85–98.

[16] Christina Delimitrou, Christos Kozyrakis, Paragon: QoS-aware scheduling for heterogeneous datacenters, in: Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS'13, pp. 77–88.

[17] Christina Delimitrou, Christos Kozyrakis, Quasar: Resource-efficient and QoS-aware cluster management, in: Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2014, pp. 127–144.

[18] Jason Mars, Lingjia Tang, Whare-map: heterogeneity in "homogeneous" warehouse-scale computers, in: Proceedings of the 40th Annual International Symposium on Computer Architecture, ISCA'13, pp. 619–630.

[19] Daniel Hagimont, Christine Larissa Mayap Kamga, Laurent Broto, Alain Tchana, Noel Depalma, DVFS aware CPU credit enforcement in a virtualized system, in: ACM/IFIP/USENIX International Middleware Conference, Middleware'13, 8275, 2013, pp. 123–142.

[20] Carl A. Waldspurger, William E. Weihl, Lottery scheduling: flexible proportional-share resource management, in: Proceedings of the First USENIX Symposium on Operating System Design and Implementation, OSDI'94.

[21] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, Ion Stoica, Mesos: A platform for fine-grained resource sharing in the data center, in: Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation, NSDI'11, pp. 295–308.

[22] Zhibo Chang, Jian Li, Ruhui Ma, Zhiqiang Huang, Haibing Guan, Adjustable credit scheduling for high performance network virtualization, in: Cluster Computing, CLUSTER'12, pp. 337–345.

[23] Sudipto Das, Vivek Narasayya, Feng Li, Manoj Syamala, CPU sharing techniques for performance isolation in multi-tenant relational database-as-a-service, in: Proceedings of the VLDB Endowment, VLDB'13, Vol. 7.

[24] Ajay Gulati, Arif Merchant, Peter Varman, mClock: Handling throughput variability for hypervisor IO scheduling, in: Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation, OSDI'10.

[25] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin M. Vahdat, Managing energy and server resources in hosting centers, in: Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles, SOSP'01, pp. 103–116.

[26] Can Hankendi, Sherief Reda, Ayse K. Coskun, vCap: Adaptive power capping for virtualized servers, in: IEEE International Symposium on Low Power Electronics and Design, ISLPED'13, pp. 415–420.

[27] Stijn Eyerman, Lieven Eeckhout, Fine-grained DVFS using on-chip regulators, in: ACM Transactions on Architecture and Code Optimization, TACO 2011, Vol. 8.

[28] Michael Butler, Leslie Barnes, Debjit Das Sarma, Bob Gelinas, Bulldozer: An approach to multithreaded compute performance, in: MICRO 2011, pp. 6–15.

[29] Chamara Gunaratne, Ken Christensen, Ethernet adaptive link rate: System design and performance evaluation, in: Local Computer Networks 2006, pp. 28–35.

[30] Sergiu Nedevschi, Lucian Popa, Gianluca Iannaccone, Sylvia Ratnasamy, David Wetherall, Reducing network energy consumption via sleeping and rate-adaptation, in: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation, NSDI'08, pp. 323–336.

[31] Vincent M. Weaver, Dan Terpstra, Shirley Moore, Non-determinism and overcount on modern hardware performance counter implementations, in: Performance Analysis of Systems and Software 2013, ISPASS'13, pp. 215–224.

[32] Venkatesh Pallipadi, Alexey Starikovskiy, The ondemand governor: past, present and future, in: Proceedings of Linux Symposium 2006, 2, 2006, pp. 223–238.

[33] Daniel A. Menascé, TPC-W: a benchmark for e-commerce, Internet Comput. 6 (3) (2002) 83–87.

[34] CLIF is a Load Injection Framework, http://clif.ow2.org/ visited on April 2014.

[35] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, Anoop Gupta, The splash-2 programs: Characterization and methodological considerations, in: SIGARCH 1995, pp. 24–36.

[36] Ariel J. Feldman, William P. Zeller, Michael J. Freedman, Edward W. Felten, SPORC: group collaboration using untrusted cloud resources, in: Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation, OSDI'10.

[37] Douglas B. Terry, Vijayan Prabhakaran, Ramakrishna Kotla, Mahesh Balakrishnan, Marcos K. Aguilera, Hussam Abu-Libdeh, Consistency-based service level agreements for cloud storage, in: Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, SOSP 2013, pp. 309–324.

[38] Xianrong Zheng, Patrick Martin, Kathryn Brohman, Cloud service negotiation: Concession vs. tradeoff approaches, in: Cluster, Cloud and Grid Computing, CCGRID'12, pp. 515–522.
[39] Nedeljko Vasic, Dejan Novakovic, Svetozar Miucin, Dejan Kostic, Ricardo Bianchini, DejaVu: Accelerating resource allocation in virtualized environments, in: Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2012, pp. 423–436.
[40] Chen Chen, Petros Maniatis, Adrian Perrig, Amit Vasudevan, Vyas Sekar, Towards verifiable resource accounting for outsourced computation, in: Proceedings of the 9th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, VEE'13, pp. 167–178.
[41] Vyas Sekar, Petros Maniatis, Verifiable resource accounting for cloud computing services, in: Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop, CCSW'11, pp. 21–26.
[42] Andreas Haeberlen, Paarijaat Aditya, Rodrigo Rodrigues, Peter Druschel, Accountable virtual machines, in: Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation, OSDI'10.
[43] Ruslan Nikolaev, Godmar Back, Perfctr-Xen: A framework for performance counter virtualization, in: Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, VEE'11, pp. 15–26.
[44] Jiaqing Du, Nipun Sehrawat, Willy Zwaenepoel, Performance profiling of virtual machines, in: Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, VEE'11, pp. 3–14.
[45] Aravind Menon, Jose Renato Santos, Yoshio Turner, G. (John) Janakiraman, Willy Zwaenepoel, Diagnosing performance overheads in the Xen virtual machine environment, in: Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, VEE'05, pp. 13–23.
[46] Vmkperf utility for VMWare ESX 4.0, 2011.
[47] Shirley Browne, Jack Dongarra, Nathan Garner, Kevin London, Philip Mucci, A scalable cross-platform infrastructure for application performance tuning using hardware counters, in: Proceedings of the 2000 ACM/IEEE Conference on Supercomputing, Supercomputing'00.
[48] Wei Jin, Jeffrey S. Chase, Jasleen Kaur, Interposed proportional sharing for a storage service utility, in: ACM SIGMETRICS Performance Evaluation Review 2004, pp. 37–48.
[49] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, Ion Stoica, Dominant resource fairness: Fair allocation of multiple resource types, in: Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation, NSDI'11, pp. 323–336.
[50] Phitchaya Mangpo Phothilimthana, Jason Ansel, Jonathan Ragan-Kelley, Saman Amarasinghe, Portable performance on heterogeneous architectures, in: Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS'13, pp. 431–444.
[51] David Koufaty, Dheeraj Reddy, Scott Hahn, Bias scheduling in heterogeneous multi-core architectures, in: Proceedings of the 5th European Conference on Computer Systems, EuroSys'10, pp. 125–138.
[52] Marco Canini, Vojin Jovanovic, Daniele Venzano, Dejan Novakovic, Dejan Kosti, Online testing of federated and heterogeneous distributed systems, in: Proceedings of the ACM SIGCOMM 2011 Conference, SIGCOMM 2012, pp. 434–435.
[53] Alexandra Fedorova, David Vengerov, Daniel Doucette, Operating system on heterogeneous core systems, in: Proceedings of 2007 Operating System Support for Heterogeneous Multicore Architectures, ASPLOS'13.
[54] Benjamin Farley, Venkatanathan Varadarajan, Kevin D. Bowers, Ari Juels, Thomas Ristenpart, Michael M. Swift, More for your money: Exploiting performance heterogeneity in public clouds, in: Proceedings of the Third ACM Symposium on Cloud Computing. SoCC'12.
[55] Zhonghong Ou, Hao Zhuang, Jukka K. Nurminen, Antti Yla-Jaaski, Pan Hu, Exploiting hardware heterogeneity within the same instance type of Amazon EC2, in: Proceedings of the 4th USENIX Conference on Hot Topics in Cloud Ccomputing, HotCloud'12, pp. 4–4.
[56] Zhonghong Ou, Hao Zhuang, Andrey Lukyanenko, Jukka K. Nurminen, Pan Hu, Vladimir Mazalov, Antti Yla-Jaaski, Is the same instance type created equal? Exploiting heterogeneity of public clouds, in: IEEE Transactions on Cloud Computing, TCC 2013, 1, 2013, pp. 201–214.
[57] Alexander Lenk, Michael Menzel, Johannes Lipsky, Stefan Tai, Philipp Offermann, What are you paying for? Performance benchmarking for infrastructure-as-a-service offerings, in: IEEE International Conference on Cloud, pp. 484–491.
[58] Christina Delimitrou, Christos Kozyrakis, QoS-aware scheduling in heterogeneous datacenters with paragon, ACM Trans. Comput. Syst. 31 (2013) 17–30.
[59] Gunho Lee, Byung-Gon Chun, Randy H. Katz, Heterogeneity-aware resource allocation and scheduling in the cloud, in: Proceedings of the 3rd USENIX Conference on Hot Topics in Cloud Computing, HotCloud'11, pp. 4–4.
[60] Alexey Tumanov, James Cipar, Michael A. Kozuch, alsched: Algebraic scheduling of mixed workloads in heterogeneous clouds, in: Proceedings of the Third ACM Symposium on Cloud Computing, SOCC'12, article No. 25.
[61] Rustam Miftakhutdinov, Eiman Ebrahimi, Yale N. Patt, Predicting performance impact of DVFS for realistic memory systems, in: MICRO 2012.
[62] Jiwei Huang, Chuang Lin, Agent-based green web service selection and dynamic speed scaling, in: IEEE 19th International Conference on Web Services, ICWS 2013, pp. 91–98.
[63] Qingyuan Deng, David Meisner, Abhishek Bhattacharjee, Thomas F. Wenisch, Ricardo Bianchini, CoScale: Coordinating CPU and memory system DVFS in server systems, in: 45th Annual IEEE/ACM International Symposium Microarchitecture, MICRO, 2012, pp. 143–154.

Boris Teabe received his M.S. in National Advanced School of Engineering Cameroon in 2013. Since January 2014 he is carrying out a 6 month internship at IRIT lab, Toulouse France. He is a member of SEPIA research group. His main research interests are in Virtualization, Cloud Computing, and Operating System.

Alain Tchana received his Ph.D. in Computer Science in 2011, at the IRIT laboratory, Polytechnic National Institute of Toulouse, France. Since September 2013 he is a Associate Professor at Polytechnic National Institute of Toulouse, France. He is a member of SEPIA research group at IRIT laboratory, Toulouse. His main research interests are in Virtualization, Cloud Computing, and Operating System.

Daniel Hagimont is a Professor at Polytechnic National Institute of Toulouse, France and a member of the IRIT laboratory, where he leads a group working on operating systems, distributed systems and middleware. He received a Ph.D. from Polytechnic National Institute of Grenoble, France in 1993. After a postdoctorate at the University of British Columbia, Vancouver, Canada in 1994, he joined INRIA Grenoble in 1995. He took his position of Professor in Toulouse in 2005.