This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2372753, IEEE Transactions on Cloud Computing

Partial Utility-driven Scheduling for Flexible SLA and Pricing Arbitration in Clouds

José Simão, Luís Veiga, *Member, IEEE*

Abstract—Cloud SLAs compensate customers with credits when average availability drops below certain levels. This is too inflexible because consumers lose non-measurable amounts of performance being only compensated later, in next charging cycles. We propose to schedule virtual machines (VMs), driven by range-based non-linear reductions of utility, different for classes of users and across different ranges of resource allocations: partial utility. This customer-defined metric, allows providers transferring resources between VMs in meaningful and economically efficient ways. We define a comprehensive cost model incorporating partial utility given by clients to a certain level of degradation, when VMs are allocated in overcommitted environments (Public, Private, Community Clouds). CloudSim was extended to support our scheduling model. Several simulation scenarios with synthetic and real workloads are presented, using datacenters with different dimensions regarding the number of servers and computational capacity. We show the partial utility-driven driven scheduling allows more VMs to be allocated. It brings benefits to providers, regarding revenue and resource utilization, allowing for more revenue per resource allocated and scaling well with the size of datacenters when comparing with an utility-oblivious redistribution of resources. Regarding clients, their workloads' execution time is also improved, by incorporating an SLA-based redistribution of their VM's computational power.

Index Terms—Cloud Computing, Community Clouds, Service Level Agreements, Utility-driven Scheduling, VM allocation, VM scheduling

1 Introduction

Currently cloud providers provide a resource selection interface based on abstract computational units (e.g. EC2 compute unit). This business model is known as Infrastructure-as-a-Service (IaaS). Cloud users rent computational units taking into account the estimated peak usage of their workloads. To accommodate this simplistic interface, cloud providers have to deal with massive hardware deployments, and all the management and environmental costs that are inherent to such a solution. These costs will eventually be reflected in the price of each computational unit.

Today, cloud providers' SLAs already establish some compensation in consumption credits when *availability*, or uptime, fall below a certain threshold.[1] The problem with *availability* is that, from a quantitative point of view, it is often equivalent to all-or-nothing, i.e. either availability level fulfills the agreed uptime or not. Even so, to get their compensation credits, users have to fill a form and wait for the next charging cycle.

Some argue that although virtualization brings key benefits for the organizations, full migration to a public cloud is sometimes not the better option. A middle ground approach is to deploy workloads in a private (or hybrid) cloud. Doing so has the potential to limit costs on a foreseeable future and, also important, keeps private data in-premises. Others propose to bring private clouds even closer to users to provide a more environmentally reasonable, or cheaper to cool and operate, cluster [1], [2].

1.1 Overcommitted environments

Figure 1 shows what means to bring the cloud closer to the user. Small, geo-distributed near-the-client datacenters (private, shared) save money, the environment, and reduce latency by keeping data on premises. This kind of vision is sometimes referred as Community Cloud Computing (C3) [3], which can take advantage of previous research in peer-to-peer and grid systems [4]. Nevertheless, many of the fundamental research and the technological deployments are yet to be explored.

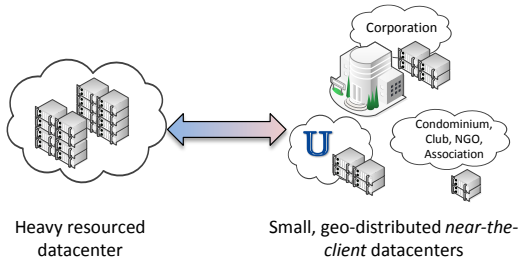


Fig. 1: Cloud deployments: From heavy clouds to small, geo-distributed near-the-client datacenters

The work reported in this article was supported by national funds through FCT – Fundação para a Ciência e a Tecnologia, under projects PTDC/EIA-EIA/113613/2009, contract Pest-OE/EEI/LA0021/2013 and the PROTEC program of the Polytechnic Institute of Lisbon (IPL).

- *J. Simão is with INESC-ID Lisboa and Instituto Superior de Engenharia de Lisboa (ISEL). E-mail: see www.gsd.inesc-id.pt/∼jsimao/.*
- *L. Veiga is with INESC-ID Lisboa / Instituto Superior Técnico, Universidade de Lisboa. E-mail: see http://www.gsd.inesc-id.pt/∼lveiga/.*

1. http://aws.amazon.com/ec2-sla/

2168-7161 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2372753, IEEE Transactions on Cloud Computing

From a resource management point of view, these new approaches highlight two issues. In one hand, the deployment sites are more lightly resourced [5], [6], either because the hardware is intrinsically less powered or the hardware layer is made of unused parts of deployments already used for other tasks. So, overcommitment, which is commonly used in virtualized environments [7], [8], [9], will become more frequent. Techniques such as dynamic resource allocation and accurate cost modeling must be researched to manage this kind of clouds. Because of the federated and low-cost nature, overcommitment of resources is perhaps a more common (and needed) scenario than in public clouds. Second, in such environments there will be many classes of users which, in most cases, are willing to trade the performance of their workloads for a lower (or even free) usage cost.

To overcommit with minimal impact on performance and maximum cost-benefits ratio, cloud providers need to relate how the partial release of resources will impact in the workload performance and user satisfaction. While users can easily decide about their relative satisfaction in the presence of resource degradation, they cannot easily determine how their workloads react to events such as peak demands, hardware failures, or any reconfiguration in general.

As private clouds become more frequent in medium and large scale organizations, it is necessary to promote a fair use of the available resources. Usually, these organizations are made of several departments, working on different projects. Each project has a budget to rent computational shared resources. For example, Intel owns a distributed compute farm that is used for running its massive chip-simulation workloads [10]. The various Intel's projects that need to use the infrastructure *purchase* different amount of servers. Also in this context, it is relevant to know how each department values or prioritizes each of its workloads, which will influence the *price* they are willing to pay for the execution environment.

All-or-nothing resource allocation is not flexible enough for these multi-tenant multi-typed user environments, especially when users may not know exactly how many resources are actually required. While no one complaints because there is no real market, it does not mean there is no room for improvements in more flexible pricing models that can foster competition and entry of smaller players. Like other telecommunications and commodity markets before, such as electricity, the still emergent Cloud market is still seen by some as an oligopoly (hence not a real market with an even playing field) because it still lacks a large number of big suppliers [11]. From the provider or owner point of view, this is important if there can be cost reductions and/or there are environmental gains by restricting resources, which will still be more favorable than simply delaying or queuing their workloads as a whole.

Both memory and CPU/cores [12], [9], [13] are common targets of overcommitment. The two major approaches consist of adapting the resources based on current observation of the system performance or using predictive methods that estimate the best resource allocation in the future based on past observations. Others incorporate explicit or implicit risk-based QoS requirements and try do decide which requested VMs should be favored but depend on non-deterministic parameters (e.g. client's willingness to pay) and make uncommon assumptions about the requested VM characteristics (e.g. homogeneous types) [14], [15]. Moreover they do not consider the *partial utility* of applying resource allocation, i.e. that reducing shares equally or in equal proportion may not yield the best overall result.

1.2 Scheduling Based on Partial-Utility

In this work we propose to schedule CPU processing capacity to VMs (the isolation unit of IaaS) using an algorithm that strives to account for user's and provider's potentially opposing interests. While the users want their workloads to complete with maximum performance and minimal cost, the provider will eventually need to consolidate workloads, overcommitting resources and so inevitably degrading the performance of some of them.

The proposed strategy operates when new VM requests are made to the provider, and takes the user's partial utility specification, which relates the user's satisfaction for a given amount of resources, and correlates it with the provider analysis of the workload progress given the resources applied. This gives an operational interval which the provider can use to maximize the user satisfaction and the need to save resources. Resources can be taken from workloads that use them poorly, or do not mind in having an agreed performance degradation (and so pay less for the service), and assign them to workloads that can use them better, or belong to users with a more demanding satisfaction rate (and so are willing to pay more).

We have implemented our algorithm as an extension to scheduling policies of a state of the art cloud infrastructures simulator, CloudSim [8], [16]. After extensive simulations using synthetic and real workloads, the results are encouraging and show that resources can be taken from workloads, while improving global utility of the user renting cost and of the provider infrastructure management.

This paper extends a previous one [17] by (i) enhancing and detailing the cost model and discussing how different utility matrices can be compared; (ii) comparing the proposed strategy with a more comprehensive list of utility-oblivious algorithms; (iii) detailing the implementation in CloudSim; (iv) presenting the results of a larger set of datacenter configurations. In summary the contributions of this work are the following:

- An architectural extension to the current relation between cloud users and providers, particularly useful for private and hybrid cloud deployments;

2168-7161 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2372753, IEEE Transactions on Cloud Computing

- A cost model which takes into account the clients' partial utility of having their VMs release resources when in overcommit;
- Strategies to determine, in a overcommitted scenario, the best distribution of workloads (from different classes of users) among VMs with different execution capacities, aiming to maximize the overall utility of the allocation;
- Extension of state of the art cloud simulator. Implementation and evaluation of the cost model in the extended simulator.

1.3 Document roadmap

The rest of the paper is organized as follows. Section 2 starts by framing our contributions with other related works. In Section 3 we describe our utility model and in Section 4 the scheduling strategies are presented. Section 5 discusses the extensions made to the simulation environment in order to support our requisites. Section 6 discusses the development and deployment in the simulation environment of CloudSim, and presents the results of our evaluation in simulated workloads. Section 7 presents our conclusions and work to address in the future.

2 Related Work

With the advent of Cloud Computing, particularly with the Infrastructure-as-a-Service business model, resource scheduling in virtualized environments received a prominent attention from the research community [18], [19], [20], [21], [22], addressed as either a resource management or a fair allocation challenge. At the same time the research community has built simulation environments to more realistically explore new strategies while making a significant contribution to *repeatable science* [23], [16], [8].

The management of virtual machines, and particularly their assignment to the execution of different workloads, is a critical operation in these infrastructures [24]. Although virtual machine monitors provide the necessary mechanisms to determine how resources are shared, finding an efficiency balance of allocations, for the customer and the provider, is a non trivial task. In recent years a significant amount of effort has been devoted to investigate new mechanisms and allocation strategies, aiming to improve the efficiency of Infrastructure-as-a-Service datacenters. Improvements to allocation mechanisms at the hypervisor level, or in an application's agnostic way, aim to make a fair distribution of available resources to the several virtual machines running on top of an hypervisor, with intervention over CPU and memory shares or I/O-related mechanisms [7].

We can organize this research space in two main categories: (i) scheduling with energy awareness, which is usually transparent to the client; (ii) scheduling with negotiated service-level objectives, which has implications in the client and provider goals. In this paper we focus on the second category, but both topics can benefit by our approach. The following is a briefly survey of these two areas.

Scheduling with Energy Awareness: A low-level energy-aware hypervisor scheduler is proposed in [25]. The scheduler takes into account the energy consumption measured based on in-processor events. It considers the dynamic power, which can change with different scheduling decisions (unlike leakage power which is always constant). A common approach is to use dynamic voltage and frequency scaling (DVFS). Typically, a globally underloaded system will have its frequency reduced. But this will have a negative and unpredictable impact on other VMs that, although having a smaller share of the system, are using it fully. To avoid inflicting performance penalties on these VMs, recent work [26] proposes extensions to the credit scheduler so that the allocated share of CPU to these smaller but overloaded VMs remains proportionally the same after the adjustment. Nevertheless, recent findings [27] show that frequency scaling and dynamic voltage have a small contribution on the reduction of energy consumption. Instead, systems based on modern commodity hardware should favor the idle state.

Others determine which is the minimum number of servers that needs to be active in order to fulfill the workload's demand, without breaking the service level objectives [28], [8], [29]. Meng et al. [28] determine which are the best VM pairs to be co-located based on their past resource demand. Given historic workload timeseries and an SLA-based characterization of the VM's demand, they determine the number of servers that need to be used for a given set of VMs. Beloglazov et al. [8] detect over and under utilization peaks, and migrate VMs between hosts to minimize the power consumption inside the datacenter. Mastroianni et al. [29] have similar goals with their ecoCloud, but use a probabilistic process to determine the consolidation of VMs. These solutions usually impose constraints on the number of VMs that can be co-located and do not use client's utility to drive allocation, missing the opportunity to explore combinations with advantage to both parties, provider and clients, that is, higher revenue per resource (which is on the provider's interest) and more progress for each dollar payed (which is on the clients' interest).

Scheduling with Service-Level Objectives: Clouds inherit the potential for resource sharing and pooling due to their inherent multi-tenancy support. In Grids, resource allocation and scheduling can be performed mostly based on initially predefined, a priori and static, job requirements [20]. In clouds, resource allocation can also be changed elastically (up or down) at runtime in order to meet the application load and effective needs at each time, improving flexibility and resource usage.

To avoid strict service level objectives violations main research works can be framed into three methods: (i) statistical learning and prediction; (ii) linear optimization methods; (iii) and economic-oriented strategies.

2168-7161 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2372753, IEEE Transactions on Cloud Computing

Resource management can also be based on microeconomic game theory models, mostly in two directions: i) forecast the number of virtual machines (or their characteristics) a given workload will need to operate [30], [31] and ii) change allocations at runtime to improve a given metric such as workload fairness or the provider's energy costs [22]. Auction-based approaches have also been proposed in the context of provisioning VMs [32], [33], [34] when available resources are less abundant than requests. Commercial systems such as the Amazon EC2 Spot Instances have adopted this strategy. S. Costache et al. [35] proposes a market where the users bid for a VM with a certain amount of resources. To guarantee a steady amount of resources, their system migrates VMs between different nodes which has the potential to impose a significant performance penalty [36].

In [14] clients choose the SLA based on a class of risk, which has impact on the price the client will pay for the service - the lower the risk the higher the price. Based on this negotiation, an allocation policy would be used to allocate resources for each user, either minimizing the risk (of the client) or the cost (of the provider). They are however unable to explicitly select the VM or set of VMs to degrade. In [15] a method is presented to decide which VMs should release their resources, based on each client willingness to pay for the service. This approach is similar to our work but they assume that some amount of SLA violations will occur because they demand the victim VM to release its full resources. They try to minimize the impact on the user's satisfaction, based on a probabilistic metric, decided only by the provider. Moreover, they assume homogeneous VM's types and with explicitly assessed different reliability levels, which in uncommon in cloud deployments.

Seokho et al. [37] focus on datacenters that are distributed across several sites, and use SLAs to distribute the load among them. Their system selects a data center according to a utility function that evaluates the appropriateness of placing a VM. This utility function depends on the distance between the user and the datacenter, together with the expected response time of the workload to be placed. Therefore, a VM request is allocated in the physical machine that is closest to the user and has a recent history of low utilization. For network bounded workloads, their system could integrate our approach by also considering the partial assignment of resources, eventually exchanging locality (and so, smaller network delays) by, for example, a small deferment in the workload finish time.

SageShift [38] targets the hosting of web services, and uses SLAs to make admission control of new VMs (Sage) based on the expected rate of co-arrival requests. In addition, it presents an extension to an hypervisor scheduler (Shift) to control the order of execution of co-located VMs, and minimize the risk of failing to meet the negotiated response time. Also in the case of Sage, no alternative strategy exists when the system detects that a new VM cannot strictly comply with a given SLA.

Flexible SLAs: In conclusion, our work is the first that we are aware of that clearly accepts, and incorporates in the economic model, the notions of partial utility degradation in the context of VM scheduling in virtualized infrastructures, such as data centers, public, private or hybrid clouds. It demonstrates that it can render benefits for the providers, as well as reduce user dissatisfaction in a structured and principled-based way, instead of the typical all-or-nothing approach of queuing or delaying requests, while still able to prioritize user classes in an SLA-like manner.

3 A PARTIAL UTILITY MODEL FOR CLOUD SCHEDULING

Our model uses a non-linear, range-based, reduction of utility that is different for classes of users, and across different ranges of resource allocations that can be applied. We name it partial utility.

To schedule VMs based on the partial utility of the clients we have to define the several elements that constitute our system model. The provider can offer several categories of virtual machines, more compute or memory optimized. In each category (e.g. compute optimized) we consider that the various VM types are represented by the set $VM_{types} = \{VM_{t_1}, VM_{t_2}, VM_{t_3}, \dots VM_{t_m}\}$. Elements of this set have a transitive less-than order, where $VM_{t_1} < VM_{t_2}$ iff *VirtualPower*$(VM_{t_1}) <$ *VirtualPower*(VM_{t_2}). The function *VirtualPower* represents the provider's metric to advertise each VM computational power, along with details about a particular combination of CPU, memory and storage capacity. For example, Amazon EC2 uses the Elastic Compute Unit (ECU) which is an aggregated metric of several proprietary benchmarks. Other examples include the HP Cloud Compute Unit (CCU).

Currently, infrastructure-as-a-service providers rent virtual machines based on pays-as-you-go or pre-reserved instances. In either case, a price for a charging period is established, e.g. $ / hour, for each VM type. This value, determined by the function $Pr(VM_{t_i})$, is the monetary value to pay when a VM of type t_i is not in overcommit with other VMs (from the same type or not). Considering that for a given VM instance, vm, the type (i.e. element of the set VM_{types}) can be determined by the function $VMType(vm)$, and therefore the price can be determined by $Pr(VMType(vm))$.

3.1 Degradation factor and Partial utility

For each VM, the provider can determine which is the degradation factor, that is, which percentage of the VM *virtual power* is diminished because of resource sharing and overcommit with other VMs. For a given VM instance, vm, this is determined by the function $Df(vm)$. In scenarios of overcommit described in the previous section, each user can choose which fraction of the price

2168-7161 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2372753, IEEE Transactions on Cloud Computing

he/she will pay when workloads are executed. When the provider must allocate VMs in overcommitment, the client will be affected by having its VMs with less allocated resources, resulting in a potentially perceivable degradation of performance of its workload. So, overcommitment and the degradation factor refer to the same process but represent either the provider's or the client's view. We will use these expressions interchangeably throughout the paper.

When overcommit must be engaged, the same client will pay as described in Equation 1, where the function Pu represents the partial utility that the owner of the VM gives to the degradation. Both these terms are percentage values.

$$Cost(vm) = Pr(VMType(vm)) \cdot (1 - Df(vm)) \cdot Pu(Df(vm)) \quad (1)$$

Although this equation naturally shares the goals of many SLA-based deployment [15], it takes into account specific aspects of our approach, as it factors the elements taken from the partial utility specification (detailed in the following paragraphs). For example, if $Df(vm)$ is 20% and $Pu(Df(vm))$ is 100% it means that the client is willing to accept the overcommit of 20% and still pay a value proportional to the degradation. But if in the same scenario $Pu(Df(vm))$ is 50% it means the client will only pay half of the value resulting from the overcommit, i.e. $Pr(VMType(vm)) \times (1 - 0.2) \times 0.5 = Pr(VMType(vm)) \times 0.4$.

In general, overcommit can vary during the renting period. During a single hour, which we consider the charging period, a single VM can have more than one degradation factor. For example, during the first hour no degradation may be necessary while during the second and the third hour, the provider could take 20% of the computation power. So, because a VM can be hibernated or destroyed by their owners, and new VMs can be requested, Df must also depend on time. To take this into account, $Df_h(vm, i)$ is the i^{nth} degradation period of hour h. Jin et al. [11] also discusses a fine grained pricing schema although they focus on the *partial usage waste* problem, which is complementary to the work discussed in this paper.

Internally, providers will want to control the maximum overcommitment which, in average, is applied to the VMs allocated to a given client and, by extension, to the all datacenter. Equation 2 is able to measure this using the Aggregated Degradation Index (ADI) for a generic set of VMs. This metric can range from 0 (non degraded) to 1 (fully degraded).

$$ADI(VMSet) = 1 - \frac{\sum_{vm \in VMSet} (1 - Df(vm)) \cdot VirtualPower(vm)}{\sum_{vm \in VMSet} VirtualPower(vm)} \quad (2)$$

3.2 Classes for prices and partial utility

Clients can rent several types of VMs and choose the class associated to each one. Classes have two purposes. The first is to establish a partial utility based on the overcommit factor. The second is to set the base price for each VM type. Clients, and the VMs they rent, are organized into classes which are represented as a set $C = \{C_1, C_2, C_3, \ldots, C_n\}$. Elements of this set have a transitive less-than order ($<$) where $C_1 < C_2$ iff *base-price*(C_1) $<$ *base-price*(C_2). The function *base-price* represents the base price for each VM type. The class of a given virtual machine instance vm is represented by the function $class(vm)$, while the owner (i.e. the client who is renting the VM) can be determined by $owner(vm)$.

Each class determines, for each overcommit factor, the partial utility degradation. Because the overcommit factor can have several values we define R as a set of ranges: $R = \{]0..0.2[, [0.2..0.4[, [0.4..0.6[, [0.6..0.8[, [0.8..1]\}$. As a result of theses classes of SLAs, the Pu function must be replaced by one that also takes into account the class of the VM, along with the interval of the overcommit factor, as presented in definition 3. Doing so, Pu_{class} is a matrix of partial utilities. Each provider can have a different matrix which it advertises so that clients can choose the best option.

$$Pu_{class} : C \times R \rightarrow [0..1] \quad (3)$$

Note that, currently, our model assumes that the partial utility matrix is defined regarding the total virtual power of a VM, namely, CPU, memory and storage capacity. If some overcommitment must be done in any of these dimensions, we consider them equal or do a simple average of them. This value is then used to determine the overall partial utility of the VM's new allocation. However, a more generic (and complex) model could be used, where a matrix like the one defined in Equation 3 could be specified for each of the dimensions of the VM. This would result in a vector of partial-utility matrices, whose final value would have to be aggregated to be used in Equation 1. This is seen as future work.

The Pr function for each VM must also be extended to take into account the VM's class, in addition to the VM's type. We define a new function, Pr_{class}, as presented in Equation 4. Similarly to the matrix of partial utilities, each provider can have a different *price* matrix.

$$Pr_{class} : C \times VM_{types} \rightarrow \mathbb{R} \quad (4)$$

In summary, the proposed partial utility model and the associated cost structure is based on three elements: *i)* the base price of each VM type, *ii)* the overcommit factor, *iii)* the partial utility degradation class associated to each VM.

3.3 Total costs

For a given client, the total sum cost of renting is simply determined by the total cost of renting each VM, as

2168-7161 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2372753, IEEE Transactions on Cloud Computing

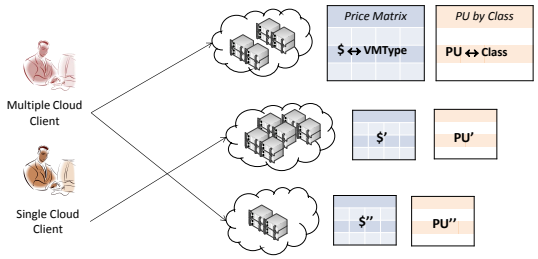


Fig. 2: A practical scenario of using flexible SLAs in a market-oriented environment

presented in Equation 5, where $RentVMs(c)$ represent the VMs rented by client c.

$$RentingCost(c) = \sum_{vm \in RentVMs(c)} VMCost(vm) \quad (5)$$

The cost of each VM is presented in Equation 6 where N is the number of hours the VM was running, and P the number of overcommitment periods in hour h. If after allocation the VM's degradation factor remains constant, then P equals 1.

$$VMCost(vm) = \sum_{h=1}^{N}\sum_{p=1}^{P} \frac{Pr_{class}(class(vm), VMType(vm))}{P} \cdot \cdot(1 - Df_h(vm,p)) \cdot \cdot Pu_{class}(class(vm), Df_h(vm,p)) \quad (6)$$

The provider's revenue is given by how much all clients pay for the VMs they rent. The provider wants to maximize the revenue by minimizing the degradation factor imposed to each virtual machine. Because there are several classes of VMs, each with a particular partial utility for a given degradation factor, the provider's scheduler must find the allocation that maximizes (6). There are different ways to do so which we analyze in Section 4.

3.4 Practical scenario

As a practical scenario we consider that the partial utility model has three classes of users (High, Medium, Low) according to their willingness to relinquish resources in exchange for a lower payment. More classes could be added but these three illustrate:

- **High** users that represent those with more stringent requirements, deadlines, and that are willing to pay more for a higher performance assurance but, in exchange, demand to be compensated if those are not met. Compensation may include not simply refund but also some level of significant penalization;
- **Medium** users who are willing to pay but will accept running their workloads in VMs with less resources for the sake of lesser payment, and other externalities, such as reduced carbon footprint impact, but have some level of expectation on execution time, and;
- careless **Low** users who do not mind waiting for their workloads to complete if they pay less;

Partial utility profiles could also be organized around cloud providers, and assume that each provider would be specialized in a given profile. For example, *flexible* would represent shared infrastructures with no obligations, and many well dimensioned private clouds; *business* public clouds or high-load private or hybrid clouds; *critical* clouds where budgets and deadlines of workloads are of high relevance, and penalties are relevant; *SLA-Oriented* top scenario where penalties should be avoided at all cost. For simplicity we focus on a single cloud provider that supports several classes of partial utility which clients can choose when renting VMs, as illustrated in Fig. 2.

For the three classes of our example, the cloud provider can define a partial utility matrix, represented by M in (7). This matrix defines a profile of partial utility for each level of resource degradation (resources released) that can be used to compare strictness or flexibility of the resource management proposed.

$$M = \begin{array}{c} \\ [0..0.2[\\ [0.2..0.4[\\ [0.4..0.6[\\ [0.6..0.8[\\ [0.8..1[\end{array} \begin{array}{c} \begin{array}{ccc} High & Medium & Low \end{array} \\ \begin{pmatrix} 1.0 & 1.0 & 1.0 \\ 0.8 & 1.0 & 1.0 \\ 0.6 & 0.8 & 0.9 \\ 0.2 & 0.6 & 0.8 \\ 0.0 & 0.4 & 0.6 \end{pmatrix} \end{array} \quad (7)$$

The provider must also advertise the base price for each type of VM. We assume there are four types of virtual machines with increasing *virtual power*, for example, **micro**, **small**, **regular** and **extra**. The matrix presented in (8) determines the base price ($/hour) for these types of VMs.

$$P = \begin{array}{c} \\ micro \\ small \\ regular \\ extra \end{array} \begin{array}{c} \begin{array}{ccc} High & Medium & Low \end{array} \\ \begin{pmatrix} 0.40 & 0.32 & 0.26 \\ 0.80 & 0.64 & 0.51 \\ 1.60 & 1.28 & 1.02 \\ 2.40 & 1.92 & 1.54 \end{pmatrix} \end{array} \quad (8)$$

3.5 Comparing flexible pricing profiles in a cloud market

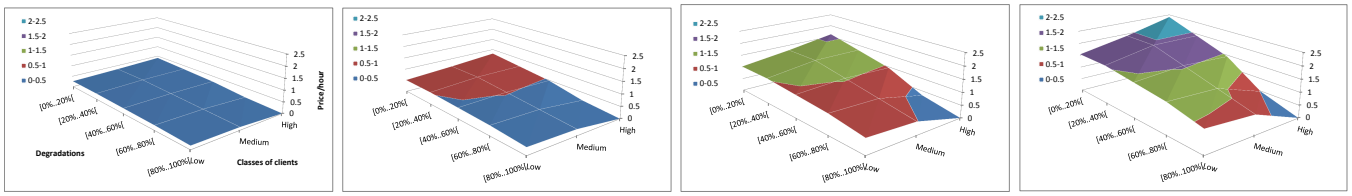
In a market of cloud providers that are nearer the client, such as the cloud communities that start to emerge [39], clients will be more mobile and independent of each provider. In this way, clients will more frequently have to look for best prices and partial utilities distributions. To this end, based on matrices P and M, Equation 9 defines a new set of matrices for each VM type. In this set, the matrices represent, for each VM type, the multiplication of a price's vector (a line in the P matrix) by the matrix of partial utilities of the provider. Note that C is the ordered set of user's classes.

$$PM_{type} = \forall classes \in C : P_{type,classes} \cdot M \quad (9)$$

Figure 3 illustrates an instance of this new set for the VM types previously described. The differences are increasingly significant as we increase the capacity (and

2168-7161 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2372753, IEEE Transactions on Cloud Computing



(a) PM_{Micro} (b) PM_{Small} (c) $PM_{Regular}$ (d) PM_{Extra}

Fig. 3: Matrices combining price and utility for the different VM types and partial utilities.

consequently the prices) of the VMs. While these matrices represent related pricing profiles, they can be used by costumers to compare and arbitrate over different providers, either for a given user class and VM size, or for global aggregate assessment. This further allows users to graphically navigate through the providers' pricing profiles. In particular, this make it possible to explore the pricing profile of a given provider, and determine the reallocation of resources a user is willingly to have, in order to fulfill a given level of cost constrains.

4 Partial Utility based Scheduling for IaaS Deployments

In general, the problem we have described is equivalent to a bin packing problem [40]. So, the scheduling process must impose constraints, on what would be a heavy search problem, and be guided by heuristics for celerity. We consider as relevant resources of a host, and requirements for a virtual machine, the following: number of cores, the processing capability of each core (expressed as *millions of instructions per second* - MIPS, MFLOPS, or any other comparative reference), and memory (in MB). The following algorithms focus on the first two requirements but a similar strategy could be used for memory. They allocate new requested VMs to these resources, taking into account the partial utility model described in the previous section.

Algorithm 1 presents what is hereafter identified as the base allocation algorithm. It takes a list of hosts and a virtual machine (with its resource requirements) that needs to be allocated to physical hardware or otherwise fail. It will search for the host with either more or less available cores, depending on the order criterion (Φ). When a greater-than ($>$) criterion is used, we call it First-Fit Increasing (FFI) since the host with more available cores will be selected. When a less-than ($<$) criterion is used we call it First-Fit Decreasing (FFD), since the host with less cores still available will be selected. This base allocation will eventually fail if no host is found with the number of requested MIPS, regardless of the class of each VM. In this situation a classic provider cannot fulfill further requests without using extra hardware, which may simple not be available.

Function ALLOCATE checks if a VM can be allocated in a given host (h). Current allocation strategies either i) try to find the host where there are still more physical cores than the sum of virtual ones, and each individually has

Algorithm 1 Generic base allocation: First-Fit Increasing/Decreasing

Require: $hosts$ list of available hosts
Require: vm VM to be allocated
Require: Φ order criterion
1: **function** BASESCHEDULING($hosts$,vm)
2: $currCores \leftarrow 0$ *or* $+\infty$ *depending on criterion*
3: $selectedHost \leftarrow null$
4: **for all** $h \in hosts$ **do**
5: **if** AVAILABLECORES(h) Φ $currCores$ **then**
6: **if** ISCOMPATIBLE(h, vm) **then**
7: $currCores \leftarrow$ AVAILABLECORES(h)
8: $selectedHost \leftarrow h$
9: **end if**
10: **end if**
11: **end for**
12: **if** $selectedHost \neq null$ **then**
13: ALLOCATE($selectHost, vm$)
14: **return** $true$
15: **end if**
16: **return** $false$
17: **end function**

enough capacity to hold the VM; ii) try to find the host with a core where the VM can fit even if shared with others; iii) degradate all the VMs in the host to fit the new VM until no more computational power is available in the host. In the first two cases, if the conditions are not met the allocation will fail. In this case, *unused cores* is used in the sense that they are still available to allocate without incurring in overcommit. All the physical cores will be in use, as usual, but they will not be used to 100% capacity. So if, for example, 4 cores have an average of 25% CPU ocupation, we consider it equivalent to saying there are 3 unused cores (i.e. still available to allocate without overcommit).

In the last case, the allocation will succeed but not taking the best choices for the new utility model proposed in Section 3. Function ISCOMPATIBLE uses the same strategies but only determines whether the conditions hold, leaving the effective allocation to the ALLOCATE function.

4.1 Analysis of the scheduling cost of the utility-oblivious scheduling

Algorithm 1 iterates over M hosts looking for the one with minimum or maximum available cores. In either case this algorithm determines a compatible host in O(M) iterations. The ISCOMPATIBLE function depends on the total number of cores, C, to determine, in case i) if there is any unused core and, in case ii) if any core still has

2168-7161 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2372753, IEEE Transactions on Cloud Computing

Algorithm 2 Partial utility allocation strategies

Require: $hosts$ hosts ordered by available resources
Require: vm new VM to be allocated
Require: $maxADI$ maximum aggregated degradation index
1: **function** VMUTILITYALLOCATION($hosts$,vm)
2: **if** BASESCHEDULING($hosts, vm$) $= true$ **then**
3: **return** $true$ ▷ No need to overcommit VM(s)
4: **end if**
5: $selection \leftarrow null$
6: $hosts \leftarrow$ *sort hosts in ascending order of available resources*
7: **for all** $h \in hosts$ **do**
8: $needed \leftarrow$ REQUESTED(vm) $-$ AVAILABLE(h)
9: $vmList \leftarrow$ ALLOCATEDVMS(h)
10: $selection \leftarrow$ SELECTVMS($vmList, needed$)
11: **if** ADINDEX($hosts, selection$) $< maxADI$ **then**
12: **for all** $(vm, df) \in selection$ **do**
13: CHANGEALLOCATION(vm, df)
14: **end for**
15: **return** $true$
16: **end if**
17: **end for**
18: **return** $false$
19: **end function**

Algorithm 3 Partial utility allocation by min/max VM type and minimum class price

Require: VM_{types} ascending/descending list of VM's types
Require: $vmList$ list of VMs allocated in host
Require: $target$ virtual power needed to fit all VMs
1: **function** SELECTVMS($vmList, target$)
2: $selection \leftarrow null$
3: $sum \leftarrow 0$
4: $vmList \leftarrow$ *sort* $vmList$ *in ascending order of price's class*
5: **while** $sum < target$ **do**
6: **for all** $t \in VM_{types}$ **do**
7: **for all** $vm \in vmList$: VMTYPE(vm) $= t$ **do**
8: $r_{vm} \leftarrow$ NEXTRANGE(vm)
9: $selection \leftarrow selection \cup (vm, r_{vm})$
10: $sum \leftarrow sum +$ VIRTUALPOWER(vm) $* (1 - r_{vm})$
11: **if** $sum \geq target$ **then**
12: **break**
13: **end if**
14: **end for**
15: **end for**
16: **end while**
17: **return** $selection$
18: **end function**

available MIPS. After determining the host where to allocate the requested VM, function ALLOCATE, can also complete with the same asymptotic cost. So, in summary, Algorithm 1 has a cost of $O(M \cdot C)$.

4.2 Partial utility-aware scheduling strategies

When there are no hosts that can be used to allocate the requested VM, some redistribution strategy must be used, while maximizing the *renting cost* as defined in Section 3. This means that the provider can use different strategies to do so, by giving priority to larger or smaller VMs (regarding their *virtual power*) or to classes with higher or lower base price.

We have extended the base algorithm so that, when a VM fails to be allocated, we then have to find a combination of degradation factors that makes it possible to fit the new VM. Four strategies/heuristics were implemented to guide our partial utility-driven algorithm. They differ in the way a host and victim VM is selected for degradation. They all start by taking the host with more resources available, that is, with more unitary available cores and with more total computation power (MIPS).

Algorithm 2 presents the modifications to the base algorithm to enable partial utility allocation strategies. After a host is selected, a set of VMs must be chosen from the list of allocated VMs in that host, i.e. operation SELECTVMS presented in Algorithm 3. These VMs are selected either by choosing the ones from the smallest size type (which we call min strategy) or the ones with the biggest size (which we call max strategy). This is controlled by using VM_{types} sorted in ascending or descending order. In both cases there are variants that combine with the lowest partial utility class (w.r.t. the definition of Section 3), either in ascending or descending order regarding its partial utility class, i.e. min-class and max-class.

4.3 Analysis of the partial-utility scheduling cost

Algorithm 2 goes through the list of hosts trying to find a combination of VMs whose resources can be reallocated. For each host, VMs are selected based on Algorithm 3. The cost of this procedure depends on a sort operation of N VMs, $O(N \lg(N))$, and a search in the space of minimum degradations to reach a target amount of resources. This search depends on r intervals in matrix (7) and t classes for prices (currently, three, as presented in Section 3.4), with a cost of $O(rtN)$. This results in an asymptotic cost of $O(rtN + N \lg(N)) = O(N \lg(N))$. Overall, the host and VMs selection algorithm cost belongs to $O(M \lg M + MN \lg N)$. Because there will be more VMs (N), across the datacenter, than hosts (M), the asymptotic cost is $O(MN \lg(N))$. In the next section, we briefly present the more relevant details of extending the CloudSim [8] simulator to evaluate these strategies.

5 IMPLEMENTATION DETAILS

We have implemented and evaluated our partial utility model on a state of the art simulator, CloudSim [16]. CloudSim is a simulation framework that must be

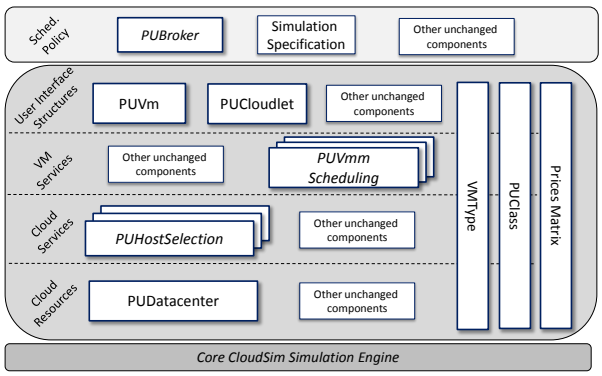


Fig. 4: Highlighted extensions to the CloudSim simulation environment

2168-7161 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2372753, IEEE Transactions on Cloud Computing

programmatically configured, or extended, to reflect the characteristics and scheduling strategies of a cloud provider.

The framework has an object domain representing the elements of a data center (physical hosts, virtual machines and execution tasks). Extensibility points include the strategy to allocate physical resources to VMs and allocation of workloads to resources available at each VM. Furthermore, at the data center level, it is possible to define how VMs are allocated to hosts (including energy-aware policies) and how execution tasks are assigned to VMs. Figure 4 highlights the new classes added to the simulation environment, which range from exploring extension points, like the virtual machine allocation to hosts, to enrichments of the object model to include partial utility related types (e.g. VM type, specification tables).

Regarding the CloudSim's base object model we have the `PUVm` type which incorporates information regarding its partial utility class. The scheduling algorithms were implemented as extensions of two main types: `VmAllocationPolicy` and `VmScheduler`. The former determines how a VM is assigned to a host while the latter determines how the virtual machine monitor (VMM) of each host allocates the available resources to each VM. It can use and re-use different matrices of partial utility classes and VM base prices, defined in the type that represents the partial utility-driven datacenter.

The type in CloudSim that represents the dynamic use of the available (virtual) CPU is the `Cloudlet` type. Because cloudlets represent work being done, each cloudlet must run in a VM with the appropriate type, simulating work being done on several VMs with different computational power. So, regarding the `Cloudlet` class, we added information about which VM type must be used to run the task. To ensure that each cloudlet is executed in the correct VM (degraded or not), we also created a new broker (extended from `DatacenterBroker`).[2]

6 Evaluation

In this section we evaluate the proposed scheduling based on partial utility. To do so, we first describe the datacenters used in the simulation and the VM types whose base price was already presented in Section 3.4. The datacenters are characterized by the number and type of hosts as described in Table 1. We used three types of datacenters hereafter known as *Size-1*, *Size-2* and *Size-3*. Each datacenter configuration is representative of a specific scenario we want to evaluate. The configuration *Size-1* represents a typical configuration of a cloud community datacenter [41], where low-end processors are used. Configuration *Size-2* represents a set of clusters owned by our research labs where raw computational capacity is around 300 cores. The simulation uses quad-core processes with hyper-threading and a computational capacity per core in the range used by Xeon processors with this number of cores. Configuration *Size-3* doubles the number of hosts, keeping their computational configuration.

2. Source code available at https://code.google.com/p/partial-utility-cloudsim/

DC size	Hosts	Cores	HT	MHz	Mem (Gbytes)
Size-1	10	2	no	1860	4
	10	2	no	2660	4
Size-2	20	4	yes	1860	8
	20	4	yes	2660	8
Size-3	40	4	yes	1860	8
	40	4	yes	2660	8

TABLE 1: Hosts configured in the simulation. Number of hosts per configuration, number of cores per host, computational capacity, hyper-threading, Memory capacity

	micro	small	regular	extra
Virtual CPU Power ($\times 10^3$ MIPS)	0.5	1	2	2.5
Memory (Gbytes)	1	1.7	2.4	3.5

TABLE 2: Characteristics of each VM type used in the simulation

Available VM types are presented in Table 2. To enrich the simulation scenario VMs have different *sizes*, simulating the request of heterogeneous virtual hardware. This is a common practice in the literature [23], [8], [16], [31]. The configurations chosen for each VM type will put our strategies to the test when a new VM request can not be fulfilled. The number of cores depends on the size of the datacenter. We simulate different scenarios where the number of cores per VM will increase as more physical resources are available. Configuration *Size-1* uses VMs with 1 core. Configuration *Size-2* and *Size-3* were simulated with VMs having 2 and 4 cores respectively. Each virtual core, of each VM type, will have the CPU power presented in Table 2.

We used an increasing number of VMs trying to be allocated. Each requested VM has a type (e.g. *micro*). We considered VM's *types* to be uniformly distributed (realistic assumption) and requested one type at a time. The following sections highlight the differences between the current allocation strategies and the ones that can cope with the proposed flexible SLAs.

6.1 Utility Unaware Allocation

Figures 5 and 6 show the effects of using two different allocation strategies for host selection, and other two regarding the use of cores, but still without taking into account each client's partial utility. Each x-axis value represents a total number of VMs requested, r, and the value in the corresponding left y-axis is the datacenter occupation (MIPS and Memory) obtained when $r - f$ number of VMs are able to run, with $f \geq 0$ being the number of not allocated VMs. The host selection is based on the First-Fit Increasing (FFI) and First-Fit Decreasing (FFD) algorithms, described in Section 4. In each of these approaches we present the total percentage of MIPS and memory allocated, in the left y-axis, for each

2168-7161 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2372753, IEEE Transactions on Cloud Computing

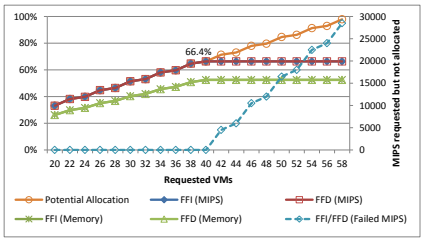


Fig. 5: Base algorithm with no core sharing between different VMs.

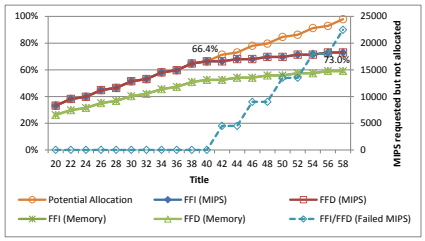


Fig. 6: Base algorithm and core sharing between different VMs.

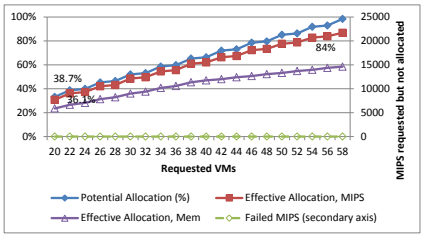


Fig. 7: Over subscription, equal degradation and unaware of client's classes.

set of requested VMs. Regarding the 6th series, FFI/FFD Failed MIPS, its results are plotted in the right y-axis.

In Figure 5 each VMM (one for each host) allocates one or more cores to each VM and does not allow any sharing of cores by different VMs. In Figure 6 each VMM (one for each host) allocates one or more cores to each VM and, if necessary, allocates a share of the same core to a different VM.

In both cases, the datacenter starts rejecting the allocation of new VMs when it is about at 66% of its raw processing capacity (i.e. MIPS) and at aproximatly 53% of its memory capacity. Although there are still resources available (cores and memory) they are not able to fit 100% the QoS of the requested VM. As expected, the core sharing algorithm promotes better resource utilization because the maximum effective allocation is 73% of the datacenter, regarding raw processing capacity, and 59%, regarding memory capacity. The effective allocation of core-based sharing still continues to increase, at a slower rate, because there are smaller VMs that can be allocated.

Figure 8 shows the counting of VM failures grouped by the VM type and VMM scheduling strategy. The simulation uses hosts with different capacities and heterogeneous VMs, for realism, as workloads are varied and resources not fully symmetric, as it happens in many real deployments in practice. The allocation strategy that enables sharing of resources is naturally the one with fewer failed requests. In the configured *Size-1* datacenter, the no-core sharing strategy starts rejecting VMs when a total of 40 is requested. In both cases, the bigger VMs (i.e. the ones requesting more computing power) are the ones with a higher rejection rate.

Table 3 (with results for an added number of VM requests) also shows, in the "Hosts" column, the number of extra hosts that would be necessary to fit the rejected VMs. These extra hosts are determined by summing all the resources not allocated and dividing by the resources of the type of host with more capacity (i.e., assuming a perfect fit and ignoring the computational cost of determining such a fit). The solution proposed in this paper avoids these extra hosts by readjusting the allocation of new and accepted VMs, following the utility and price matrices negotiated with the client.

Figure 9 shows the evolution of hosts' utilization. This figure presents the result of allocating a total of 76 VMs. It shows that when using FFD with a core sharing approach, the number of unused hosts drops more slowly, while with the FFI approach all hosts start being used with less VMs allocated. If the datacenter is running a number and type of VMs below its rejection point, the FFD scheduling is better because hosts can be turned off or put into an idle state.

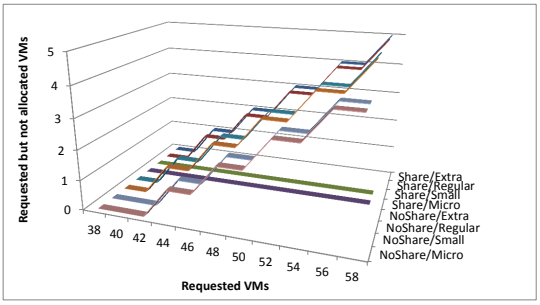


Fig. 8: Types, sizes and counting of requested but not allocated VMs

	Base No Core Sharing						Base Core Sharing					
# VMs	Failed	E	R	S	M	Hosts	Failed	E	R	S	M	Hosts
38	0 (0%)	0	0	0	0	+0	0 (0%)	0	0	0	0	+0
42	2 (5%)	1	1	0	0	+1	2 (5%)	1	1	0	0	+1
60	20 (33%)	5	5	5	5	+10	10 (17%)	5	5	0	0	+5
76	36 (47%)	9	9	9	9	+18	18 (24%)	9	9	0	0	+8

TABLE 3: Summary of VMs requested but not allocated and the number of additional hosts when cores are not shared

6.2 Over subscription

Looking again to Figures 5-6, at the time when 58 VMs are requested, both strategies leave a significant part of the datacenter unused.

Figure 7 shows the results for the over subscription algorithm (hereafter known as Base+OverSub), described in Section 4, that is oblivious to client's classes, because

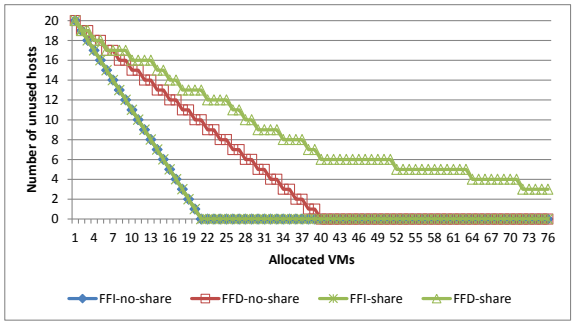
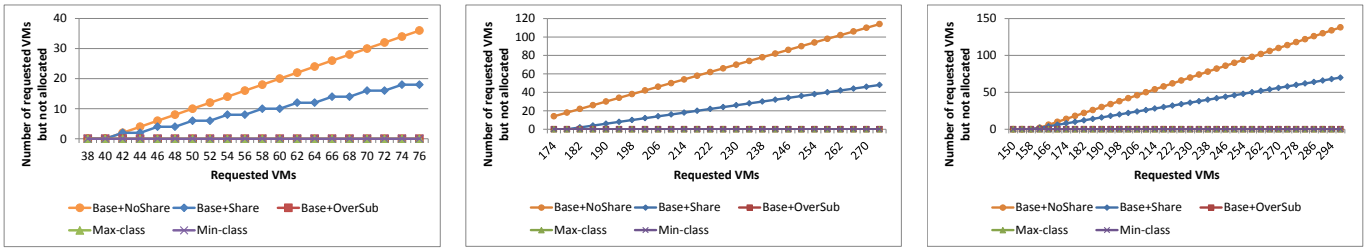


Fig. 9: Unused hosts

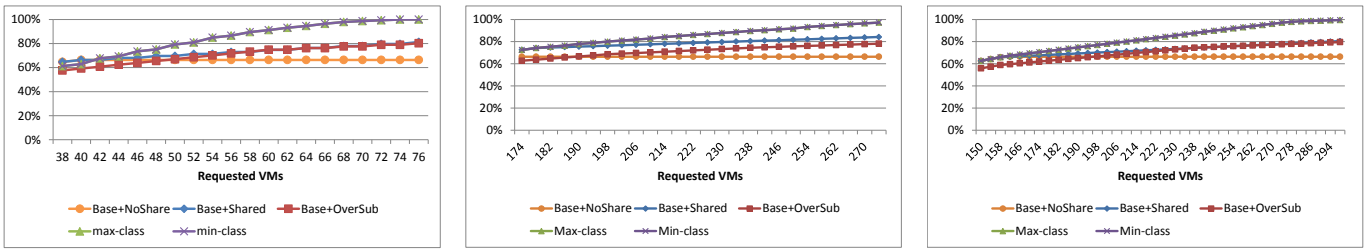
2168-7161 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2372753, IEEE Transactions on Cloud Computing



(a) 1 core/VM. Size-1 datacenter. *(b) 2 cores/VM. Size-2 datacenter.* *(c) 4 cores/VM. Size-3 datacenter.*

Fig. 10: Number of requested but not allocated VMs.



(a) 1 core/VM. Size-1 datacenter. *(b) 2 cores/VM. Size-2 datacenter.* *(c) 4 cores/VM. Size-3 datacenter.*

Fig. 11: Compared resource utilization.

it releases resources from all VMs until no more computational power is available in the host. Given that this strategy looks at the host with more cores, ignoring the total computational power, it departures immediately from the potential allocation, because resources are released from VMs even when there is computational power available in other hosts. However, when more than 40 VMs are requested, it will grow more than the previous two allocation strategies.

Differently from the previous two strategies, it will not fail allocations, as can be seen in the right y-axis regarding the series "Failed MIPS (sec. axis)". Nevertheless, the effective allocation still has margin to grow. More importantly, using this approach, there is no way to enforce the SLA negotiated with the clients. This has a significant impact in the provider's revenue as we will demonstrate next when we present the results for our strategies that take into account the type of VMs, their classes, and the partial utility negotiated.

6.3 Utility-driven Allocation

In utility-driven allocation, all requested VMs will eventually be allocated until the datacenter is overcommitted by a factor that can be defined for each provider. Along with the type, VMs are characterized by a partial utility class (e.g. *high*), as described in Section 3. In the following results, in each set of allocated VMs there are 20% of class *high*, 50% of class *medium* and 30% of class *low*.

In this section we will show how the proposed approach behaves, regarding two important set of metrics: a) allocation of VMs and b) execution of workloads by the allocated VMs. The first set of metrics is mainly important for the provider, while the second set of metrics is primarily of interest to the client. We compare utility-unware allocations with two heuristics presented in Section 4 - max-class and min-class.

6.3.1 Allocation of VMs

Regarding the provider side metrics, we measure the number of failed VM requests, the resource utilization percentage and the revenue (per hour). In all of the metrics, our strategies are at least as good as the Base+OverSub strategy, while specifically regarding revenue, we have average increases around 40%.

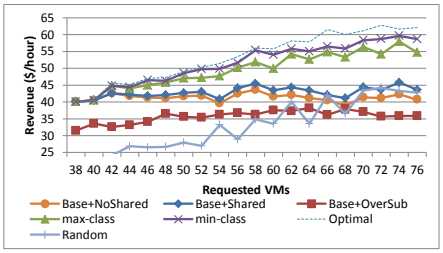
First, we compare our approaches with the base algorithm described in Section 4, regarding the number of VMs that were requested but not allocated. Figure 10.a shows that, while the base algorithm fails to allocate some VMs when 40 or more VMs are requested, the other five utility-driven strategies can allocate all requests in this configuration of the datacenter (note the collapsed series). Figure 10.b presents similar results for a *Size-2* datacenter. In this case, after 180 VMs, the Base allocation algorithm rejects VMs of type *extra* and *regular*.

Second, we evaluate how available resources are utilized. Regarding this metric, Figure 11 shows the percentage of *resource utilization* with an increasing number of VMs being requested for allocation. Three observations are worth noting: a) although with base allocation strategy some VMs are not scheduled, as demonstrated in Figure 10, others can still be allocated and can use some of the remaining resources; b) second, it is clear that our strategies achieve better resource utilization, while allocating all VMs; c) as the size of the datacenter increases, the strategy Base+OverSub lags behind to use all available resources. Our strategies can reach the peak in a similar fashion, across all sizes of datacenters.

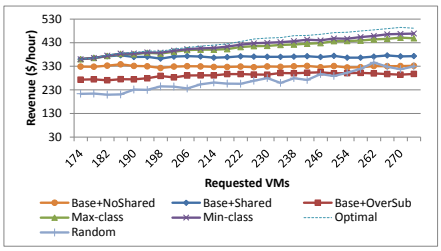
The third and last metric evaluated for the provider is the revenue. Figure 12 shows how the revenue progresses with an increasing number of total VM requests.

2168-7161 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

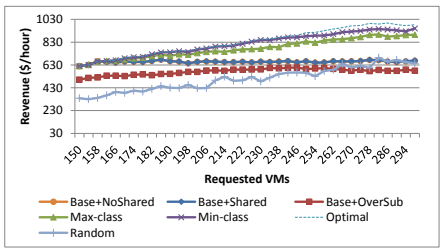
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2372753, IEEE Transactions on Cloud Computing



(a) 1 core/VM. Size-1 datacenter.

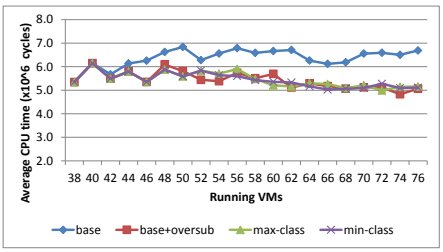


(b) 2 cores/VM. Size-2 datacenter.

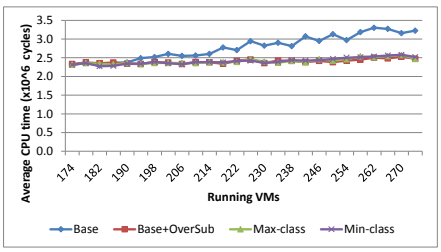


(c) 4 cores/VM. Size-3 datacenter.

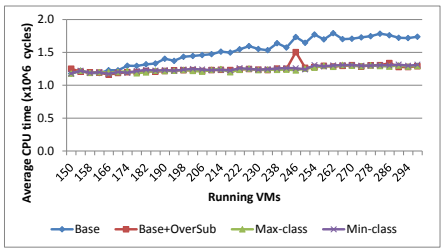
Fig. 12: Compared revenue.



(a) 1 core/VM. Size-1 datacenter.

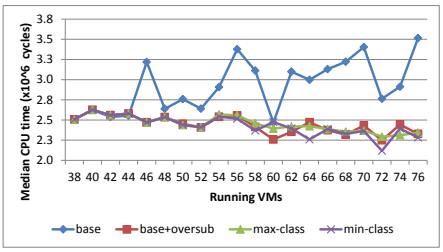


(b) 2 cores/VM. Size-2 datacenter.

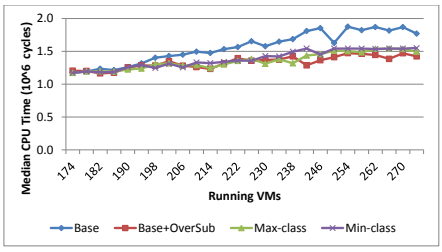


(c) 4 cores/VM. Size-3 datacenter.

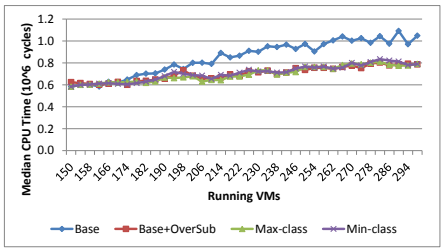
Fig. 13: Compared average execution time.



(a) 1 core/VM. Size-1 datacenter.



(b) 2 cores/VM. Size-2 datacenter.



(c) 4 cores/VM. Size-3 datacenter.

Fig. 14: Compared median execution time.

It clearly demonstrates the benefits of using a degradation and partial utility-driven approach, showing that the provider's revenue can indeed increase if the rejected VMs (above 40 in the *Size-1* datacenter and above 180 in the *Size-2* datacenter) are allocated, even if only with a fraction of their requested resources (i.e. subject to degradation driven by partial-utility ranges).

Comparing with the utility-oblivious redistribution, which also allocates all requested VMs (i.e. Base+OverSub), the increase of revenues in a *Size-1* type datacenter can go up to a a maximum of 65% ($35.8 to $59.0). In the case of a *Size-2* datacenter it can reach a maximum of 53% ($294.3 to $451.2), and 54% ($580.1 to $895.8) in a *Size-3* configuration. When the comparison is done starting from the point where VMs are rejected by the base strategy, the medium increase in revenue is 45%, 40% and 31%, for each datacenter configuration, which results in an average increase in revenue of 39% when considering all revenue increases across all datacenters.

We also compare the scheduling heuristics with a *random* and an *optimal* allocation. The *random* method chooses the server according to a random order. At a given server it will iterate over a random number of the available VMs (at most 50%), until it can take the necessary resources. This strategy stays below or slightly above Base+OverSub (which also does not rejects VMs) but exhibits worst results than any of our heuristics. The *optimal* allocation was determined by exhaustively testing all the combinations of resource reallocation (a very slow process) and at each step choosing the one with better revenue. Our two main partial utility-driven heuristics are the ones that come closer to this allocation.

6.3.2 Effects on workloads

Finally, and regarding the execution time, we have evaluated the scheduling of VM resources to each profile based on the partial utility. The data used was collected from workloads executed during 10 days by thousands of PlanetLab VMs provisioned for multiple users [8], [42]. Each of these workloads are represented by traces with the percentage of CPU usage of a given VM running in the PlanetLab network, during a day. We use n of these workloads where n is the number of requested VMs. In our simulation environment, each trace is assigned to a single VM allocated with each strategy.

The average execution time of the workloads in each VM is presented in Figure 13, while the median execution time of the workloads in each VM is presented in

2168-7161 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2372753, IEEE Transactions on Cloud Computing

Figure 14, for the three datacenter sizes. The CPU time used by the workloads running on the allocated VMs is based on the simulation clock managed by CloudSim.

In the base strategy, as some requested VMs will be rejected because no host can be found with the complete requirements, there will be VMs that receive more than one PlanetLab VM trace. In the simulation, when these PlantLab VMs are being reproduced, they receive a fraction of the available CPU, proportionally to the number of co-located PlanetLab VMs.

The results show that with more VMs allocated, even if with less allocated resources than the ones requested, as it is the case, both the average and the median execution time of tasks running on VMs allocated with our partial utility-driven scheduler is below the execution times achieved with the base strategy.

When comparing Base+OverSub with our best strategy (i.e. min-class), we can observe that the former has a marginally better average execution time while the latter has a slightly better median, rendering the differences seemingly non-significant. Nevertheless, as shown before in Section 6.3, the Base+OverSub strategy is unable to reach the best revenue for the provider and cannot provide any economic benefits for the clients given its utility unawareness.

7 Conclusion

There is an increasing interest in small, geo-distributed and near-the-client datacenters, what is sometimes known as Community Cloud Computing (C3). In these deployments, overcommitting resources is a relevant technique to lower environmental and operational costs. Nevertheless, users may be just as happy, or at least content, with slightly or even significantly reduced performance if they are compensated by lower cost or almost cost-free.

In this paper, we have proposed a cost model that takes into account the user's partial utility specification when the provider needs to transfer resources between VMs. We developed extensions to the scheduling policies of a state of the art cloud infrastructures simulator, CloudSim [8], [16], that are driven by this model. The cost model and partial utility-driven strategies were applied to the oversubscription of CPU. We have measured the provider's revenue, resource utilization and client's workloads execution time. Results show that, although our strategies partially degraded and release the computational power of VMs when resources are scarce, they overcome the classic allocation strategy which would not be able to allocate above a certain number of VMs.

We see an interesting path regarding future work on this topic. From an experimental point of view we plan to incorporate this approach in private cloud solutions such as OpenStack[3] and extend the evaluation of the model to other resources, namely the network bandwidth. We also want to enhance the scheduling process to incorporate progress information collected from workloads, eventually using historical data, such that resources can also be taken from workloads that use them less efficiently. This will need further extensions to the execution model of CloudSim.

3. http://www.openstack.org/

References

[1] J. Liu, M. Goraczko, S. James, C. Belady, J. Lu, and K. Whitehouse, "The data furnace: heating up with cloud computing," in *Proceedings of the 3rd USENIX conference on Hot topics in cloud computing*, ser. HotCloud'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 15–15.

[2] A. Khan, L. Navarro, L. Sharifi, and L. Veiga, "Clouds of small things: Provisioning infrastructure-as-a-service from within community networks," in *Wireless and Mobile Computing, Networking and Communications (WiMob), 2013 IEEE 9th International Conference on*, 2013, pp. 16–21.

[3] A. Marinos and G. Briscoe, "Community cloud computing," in *First International Conference on Cloud Computing*, 2007, pp. 472–484.

[4] J. N. Silva, P. Ferreira, and L. Veiga, "Service and resource discovery in cycle-sharing environments with a utility algebra," in *IPDPS*. IEEE, 2010, pp. 1–11.

[5] B. Saovapakhiran and M. Devetsikiotis, "Enhancing computing power by exploiting underutilized resources in the community cloud," in *Proceedings of IEEE International Conference on Communications, ICC 2011, Kyoto, Japan, 5-9 June, 2011*. IEEE, 2011, pp. 1–6.

[6] A. Khan, U. Buyuksahin, and F. Freitag, "Prototyping incentive-based resource assignment for clouds in community networks," in *Advanced Information Networking and Applications (AINA), 2014 IEEE 28th International Conference on*, May 2014, pp. 719–726.

[7] C. A. Waldspurger, "Memory resource management in VMware ESX server," *SIGOPS Oper. Syst. Rev.*, vol. 36, pp. 181–194, December 2002.

[8] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 13, pp. 1397–1420, 2012.

[9] O. Agmon Ben-Yehuda, E. Posener, M. Ben-Yehuda, A. Schuster, and A. Mu'alem, "Ginseng: Market-driven memory allocation," in *Proceedings of the 10th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, ser. VEE '14. New York, NY, USA: ACM, 2014, pp. 41–52. [Online]. Available: http://doi.acm.org/10.1145/2576195.2576197

[10] D. G. F. Ohad Shai, Edi Shmueli, "Heuristics for resource matching in intel's compute farm," in *Proceedings of the 17th Workshop on Job Scheduling Strategies for Parallel Processing (co-located with IPDPS)*, 2013.

[11] H. Jin, X. Wang, S. Wu, S. Di, and X. Shi, "Towards optimized fine-grained pricing of iaas cloud platform," *Cloud Computing, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2014.

[12] Y. Zhang, A. Bestavros, M. Guirguis, I. Matta, and R. West, "Friendly virtual machines: leveraging a feedback-control model for application adaptation," in *Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments*, ser. VEE '05. New York, NY, USA: ACM, 2005, pp. 2–12.

[13] Z. Gong, X. Gu, and J. Wilkes, "Press: Predictive elastic resource scaling for cloud systems," in *Network and Service Management (CNSM), 2010 International Conference on*, oct. 2010, pp. 9 –16.

[14] M. Macias and J. Guitart, "A risk-based model for service level agreement differentiation in cloud market providers," in *Distributed Applications and Interoperable Systems*, ser. Lecture Notes in Computer Science, K. Magoutis and P. Pietzuch, Eds. Springer Berlin Heidelberg, 2014, pp. 1–15.

[15] H. Morshedlou and M. R. Meybodi, "Decreasing impact of SLA violations: A proactive resource allocation approachfor cloud computing environments," *IEEE T. Cloud Computing*, vol. 2, no. 2, pp. 156–167, 2014.

2168-7161 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2372753, IEEE Transactions on Cloud Computing

[16] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw. Pract. Exper.*, vol. 41, no. 1, pp. 23–50, Jan. 2011.

[17] J. Simão and L. Veiga, "Flexible SLAs in the Cloud with a Partial Utility-Driven Scheduling Architecture," in *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, vol. 1, Dec 2013, pp. 274–281.

[18] L. M. Vaquero, L. Rodero-Merino, and R. Buyya, "Dynamically scaling applications in the cloud," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 1, pp. 45–52, Jan. 2011.

[19] R. Buyya, S. K. Garg, and R. N. Calheiros, "SLA-oriented resource provisioning for cloud computing: Challenges, architecture, and solutions," in *Proceedings of the 2011 International Conference on Cloud and Service Computing*, ser. CSC '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 1–10.

[20] J. N. Silva, L. Veiga, and P. Ferreira, "A^{2}HA - automatic and adaptive host allocation in utility computing for bag-of-tasks," *J. Internet Services and Applications*, vol. 2, no. 2, pp. 171–185, 2011.

[21] J. Simão and L. Veiga, "VM economics for Java cloud computing: An adaptive and resource-aware java runtime with quality-of-execution," in *CCGRID*. IEEE, 2012, pp. 723–728.

[22] V. Ishakian, R. Sweha, A. Bestavros, and J. Appavoo, "Cloudpack: Exploiting workload flexibility through rational pricing," in *Middleware 2012*, ser. Lecture Notes in Computer Science, P. Narasimhan and P. Triantafillou, Eds. Springer Berlin Heidelberg, 2012, vol. 7662, pp. 374–393.

[23] S. Malik, S. Khan, and S. Srinivasan, "Modeling and analysis of state-of-the-art vm-based cloud management platforms," *Cloud Computing, IEEE Transactions on*, vol. 1, no. 1, 2013.

[24] J. Simão and L. Veiga, "A classification of middleware to support virtual machines adaptability in iaas," in *Proceedings of the 11th International Workshop on Adaptive and Reflective Middleware*, ser. ARM '12. New York, NY, USA: ACM, 2012, pp. 5:1–5:6.

[25] N. Kim, J. Cho, and E. Seo, "Energy-credit scheduler: An energy-aware virtual machine scheduler for cloud systems," *Future Generation Computer Systems*, vol. 32, no. 0, pp. 128 – 137, 2014.

[26] D. Hagimont, C. Mayap Kamga, L. Broto, A. Tchana, and N. Palma, "Dvfs aware cpu credit enforcement in a virtualized system," in *Middleware 2013*, ser. Lecture Notes in Computer Science, D. Eyers and K. Schwan, Eds. Springer Berlin Heidelberg, 2013, vol. 8275, pp. 123–142.

[27] E. Le Sueur and G. Heiser, "Dynamic voltage and frequency scaling: The laws of diminishing returns," in *Proceedings of the 2010 International Conference on Power Aware Computing and Systems*, ser. HotPower'10. USENIX Association, 2010, pp. 1–8.

[28] X. Meng, C. Isci, J. Kephart, L. Zhang, E. Bouillet, and D. Pendarakis, "Efficient resource provisioning in compute clouds via vm multiplexing," in *Proceedings of the 7th International Conference on Autonomic Computing*, ser. ICAC '10. New York, NY, USA: ACM, 2010, pp. 11–20.

[29] C. Mastroianni, M. Meo, and G. Papuzzo, "Probabilistic consolidation of virtual machines in self-organizing cloud data centers," *Cloud Computing, IEEE Transactions on*, vol. 1, no. 2, pp. 215–228, July 2013.

[30] K. Tsakalozos, H. Kllapi, E. Sitaridi, M. Roussopoulos, D. Paparas, and A. Delis, "Flexible use of cloud resources through profit maximization and price discrimination," in *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering*, ser. ICDE '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 75–86.

[31] R. Mian, P. Martin, F. Zulkernine, and J. L. Vazquez-Poletti, "Estimating resource costs of data-intensive workloads in public clouds," in *Proceedings of the 10th International Workshop on Middleware for Grids, Clouds and e-Science*, ser. MGC '12. New York, NY, USA: ACM, 2012, pp. 3:1–3:6.

[32] H. Xu and B. Li, "Dynamic cloud pricing for revenue maximization," *Cloud Computing, IEEE Transactions on*, vol. 1, no. 2, pp. 158–171, July 2013.

[33] D. G. S. Zaman, "Combinatorial auction-based dynamic VM provisioning and allocation in clouds," in *Third International Conference on Cloud Computing Technology and Science (CloudCom)*, IEEE, Ed., 2011, pp. 107–114.

[34] A. Andrzejak, D. Kondo, and S. Yi, "Decision model for cloud computing under sla constraints," in *Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on*, 2010, pp. 257–266.

[35] S. Costache, N. Parlavantzas, C. Morin, and S. Kortas, "On the use of a proportional-share market for application slo support in clouds," in *Euro-Par 2013 Parallel Processing*, ser. Lecture Notes in Computer Science, F. Wolf, B. Mohr, and D. Mey, Eds. Springer Berlin Heidelberg, 2013, vol. 8097, pp. 341–352.

[36] F. Xu, F. Liu, H. Jin, and A. Vasilakos, "Managing performance overhead of virtual machines in cloud computing: A survey, state of the art, and future directions," *Proceedings of the IEEE*, vol. 102, no. 1, pp. 11–31, 2014.

[37] S. Son, G. Jung, and S. Jun, "An sla-based cloud computing that facilitates resource allocation in the distributed data centers of a cloud provider," *The Journal of Supercomputing*, vol. 64, no. 2, pp. 606–637, 2013.

[38] O. Sukwong, A. Sangpetch, and H. Kim, "Sageshift: Managing slas for highly consolidated cloud," in *INFOCOM, 2012 Proceedings IEEE*, March 2012, pp. 208–216.

[39] Peter Mell and Tim Grance, "The NIST Definition of Cloud Computing," http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf, 2009.

[40] M. R. Garey, R. L. Graham, and D. S. Johnson, "Resource constrained scheduling as generalized bin packing," *J. Comb. Theory, Ser. A*, vol. 21, no. 3, pp. 257–298, 1976.

[41] A. Khan, U. Buyuksahin, and F. Freitag, "Prototyping incentive-based resource assignment for clouds in community networks," in *Advanced Information Networking and Applications (AINA), 2014 IEEE 28th International Conference on*, May 2014, pp. 719–726.

[42] K. Park and V. S. Pai, "CoMon: a mostly-scalable monitoring system for planetlab," *SIGOPS Oper. Syst. Rev.*, vol. 40, no. 1, pp. 65–74, Jan. 2006.

José Simão is a lecturer at the Engineering School of the Polytechnic Institute of Lisbon (ISEL), where he teaches programming technologies. He is also a final year PhD student at the Engineering School of the University of Lisbon (IST), ULisboa, where he obtained his Master degree in 2008, and a junior researcher at INESC-ID Lisboa. His main research interests are resource management in clusters and cloud infrastructures, with a special focus on scheduling algorithms and virtualization technologies. He has also interest in computer security having participated in a project with the Portuguese National Security Office.

Luís Veiga is a tenured Assistant Professor at Instituto Superior Tcnico (IST), ULisboa, Senior Researcher at INESC-ID, and Group Manager of GSD for 2014-15. He coordinates locally the FP7 CloudForEurope project, participates in FP7 Timbus project on digital preservation and virtualization. He has lead 2 National funded research projects on P2P cycle-sharing and virtualization, and has coordinated 2 on distributed virtual machines and multicore programming, and evaluated FP7 and third-country project proposals (Belgium). He has over 85 peer-reviewed scientific publications in journals, conferences, book chapters, workshops (Best Paper Award at Middleware 2007 and Best-Paper Award Runner Up at IEEE CloudCom 2013). He was General Chair for Middleware 2011, and belongs to Middleware Steering and Program Committee. He was Virtualization track co-Chair for IEEE CloudCom 2013, and Local Chair for Euro-Par 2014 track on Distributed Systems and Algorithms. He was an "Excellence in Teaching in IST" mention recipient (2008, 2012, 2014), and awarded Best Researcher Overall at INESC-ID Prize (2014) and Best Young Researcher at INESC-ID Prize (2012). He is a member of the Scientific Board of Erasmus Mundus European Master and Joint Doctorate in Distributed Computing. He is Chair of IEEE Computer Society Chapter, IEEE Section Portugal for 2014-2015.

2168-7161 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.