Flexible SLAs in the Cloud with a Partial Utility-driven Scheduling Architecture

José Simão
INESC-ID Lisboa
Instituto Superior de Engenharia de Lisboa (ISEL)
Lisboa, Portugal
Email: jsimao@cc.isel.ipl.pt

Luís Veiga
INESC-ID Lisboa
Instituto Superior Técnico (IST)
Lisboa, Portugal
Email: luis.veiga@inesc-id.pt

***Abstract*—Current clouds SLAs include compensation for customers (i.e. resource renters) with credits when average availability drops below a certain point. However, this credit scheme is too inflexible because consumers lose a non measurable quantity of performance and are only compensated later (i.e. in the next charging cycle). We propose to schedule cloud isolation and execution units, i.e. virtual machines (VMs), driven by the partial utility of applying a certain amount of resources (CPU, memory or bandwidth) to a given VM. This partial utility metric, specified by the customer, allows the provider to transfer resources between VMs. This is particularly relevant for private clouds where resources are not so abundant. We have defined a cost model that incorporates the partial utility the client gives to a certain level of depreciation when VMs are allocated in an overcommit environment. CloudSim, a state of the art cloud simulator, was extended to support our partial utility-driven scheduling model. Using simulation scenarios with synthetic and real workloads, we show that our proposed scheduling strategy brings benefits to providers (i.e. revenue, resource utilization) and clients (i.e. workloads' execution time) by incorporating a SLA-based depreciation of computational power, allowing for more VMs to be allocated.**

***Index Terms*—Cloud Computing, Community Clouds, Service Level Agreements, Utility-driven Scheduling**

I. Introduction

Currently cloud providers provide a resource selection interface based on abstract computational units (e.g. EC2 computational 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' SLA 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.

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

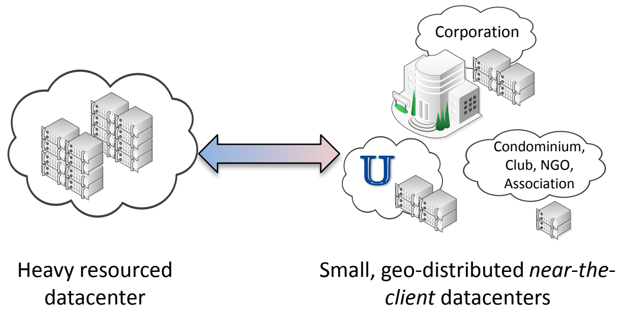
Corporation
Condominium, Club, NGO, Association
U
Heavy resourced datacenter
Small, geo-distributed *near-the-client* datacenters

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

Some argue that although virtualization brings key benefits for the organizations, migrating all to a public cloud is not the better option.[2] 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 [1] to provide a more environmentally reasonable, and cheaper to cool and operate, cluster.

A. 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) [2], which can take advantage of previous research in peer-to-peer and grid systems [3]. Nevertheless, many of the fundamental research and the technological deployments are yet to be explored. From a resource management point of view, these new approaches highlight two issues. In one hand, because the deployment sites are more lightly resourced, overcommitment 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

[2]Adopt the cloud, lose money. Virtualize your datacenter instead. http://www.theregister.co.uk/2009/04/15/mckinsey_cloud_report/

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.

In a public cloud, overcommitting can be used to reduce the number of machines requiring power when aiming to reduce energy consumption [4]. In private clouds, given the potential physical resource scarcity, the problem is even more critical. To overcommit with minimal impact on performance and maximum cost-benefits ratio, cloud providers need to have a depreciation rationale relating how the depreciation 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 depreciation, they cannot easily determine how their workloads react to events such as peak demands, hardware failures, or any reconfiguration in general.

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. 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. From the provider or owner point of view, this is important if there can be cost reductions and/or 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 [5], [6] are common targets of overcommitment. The two major approaches consist of adapting the resources based on current observation of the system performance [5], [7] or using predictive methods that estimate the best resource allocation in the future based on past observations [8]. These systems try to reach equilibrium in the share allocated to each user targeting previously established performance goals, based on offline profiling. 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. Others build a model to estimate the costs of running workloads in the cloud but they assume a specific kind of workloads (e.g. master-slave [9]) or assume users are only satisfied by a certain amount of resources [10], [11].

B. 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' 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 depreciating the performance of some of them.

The proposed strategy 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 used 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 [4], [12]. After extensive simulations using synthetic and real workloads, the results are encouraging and show that resources can be taken from workloads, improving global utility of the user renting cost and of the provider infrastructure management.

In summary the contributions of this paper are the following:

- An architectural extension to the current relation between cloud users and providers, particularly useful for private and hybrid cloud deployments;
- A cost model which takes into account the clients' partial utility of having their VMs depreciated 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 utility of the allocation;
- Extension of state of the art cloud simulator. Implementation and evaluation of the cost model in the extended simulator.

C. Document roadmap

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

II. 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 [13], [6], [14], [15], [16], [17], addressed as either a resource management or a fair allocation challenge.

The management of virtual machines, and particularly their assignment to the execution of different workloads, is a critical operation in these infrastructures. Although virtual machine monitors provide the necessary mechanisms to determine how resources are shared [18], finding an efficiency balance of allocations, for the customer and the provider, is a non trivial task.

Early[3] work of Zhang et al. [5] uses a feedback control model to equalize resource demand in a set of VMs (running on top of a VMM). Their distinct idea is that VMs should regulate themselves based on a single congestion signal (the real time needed to update the virtual time clock), instead of using complex resource management strategies. Profiling VMs for cloud placement has been used in [19]. In e-science scenarios, performance predictability in scheduling has been addressed in [20] and [21].

The current trend is towards cloud computing infrastructures (public, private, hybrid). 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 [15]. 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.

In [22] a novel business model is proposed where unused resources, i.e. system VM instances, can be rented to secondary users whose workloads may execute intermittently. Kingsher [23] is a cost-aware elasticity provisioning system targeting application owners who want to select the provider with the best configuration to support their peak workloads (i.e. maximum number of requests/second successfully handled), minimizing the application owner cost.

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 [9], [11] and ii) change allocations at runtime to improve a given metric such as workload fairness or provider energy costs [24], [17]. Auction-based approaches have also been proposed in the context of provisioning VMs [22], [25] when available resources are less abundant than requests. Commercial systems such as the Amazon EC2 Spot Instances have adopted this strategy.

Still, most works until now have been focused on finding resource allocations in the most efficient way for the provider [13], [26], [8], [10], usually based on energy and/or cooling and environmental costs. Few works consider that the customer accepts a negotiable performance during the workload execution. This type of flexibility usually requires the adoption of an economic or cost theoretical model. Besides the work in [8], Cloudpack [17] provides support for users to specify workloads in a way they can declare their quantitative resource requirements and temporal flexibilities.

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]The word *early* is to be understood as in the beginning of the modern virtualization era, after a hibernation of more than 30 years following IBM's virtual machine in the 70's

III. A PARTIAL UTILITY MODEL FOR CLOUD SCHEDULING

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 VMs are represented by the set $VM_{types} = \{VM_{t_1}, VM_{t_2}, VM_{t_3}, \ldots VM_{t_m}\}$. Elements of this set have a transitive less-than order, where $VM_{t_1} < VM_{t_2}$ iff *virtual-power*$(VM_{t_1}) <$ *virtual-power*(VM_{t_2}). The function *virtual-power* represents the provider's metric to advertise each VM computational power. For example, Amazon EC2 uses the Elastic Computation Unit (ECU) which is an aggregated metric of several proprietary benchmarks. Other examples include the HP Cloud Compute Unit (CCU).

Current cloud providers determine a price for a charging period, e.g. \$ / hour, for each VM type. This value, determined by the function $Price(VM_{t_i})$, 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 $VMType(vm)$, the price can be determined by $Pr(VMType(vm))$.

A. Depreciation factor and Partial utility

For each VM the provider can determine which is the depreciation 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, the depreciation factor 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 he will pay when workloads are executed. When overcommit must be used, the same client will pay as described in Equation 1, where the function Pu represents the partial utility the owner of the VM gives to the depreciation. Both the depreciation factor and the partial utility are percentage values.

$$\begin{aligned} &Cost(vm) = \\ &\quad Pr(VMType(vm)) \cdot (1 - Df(vm)) \cdot Pu(Df(vm)) \qquad (1) \end{aligned}$$

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 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 depreciation factor as depicted in Figure 2. In this example, during the first hour no

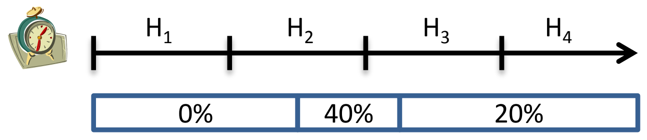

Fig. 2: Scenario where depreciation varies during renting period

depreciation is necessary while during part of the third hour and the fourth hour, the provider needs to take 20% of the computation power. So, because a VM can be hibernated or destroyed by their owners, and new VMs can be requested, the depreciation factor, given by Df, must also depend on time. To take this into account, $Df_h(vm, i)$ is the i^{nth} depreciation period of hour h. This is similar to the *epoch* concept of [17] but more fine grained and usable in the partial utility model as described next.

B. Classes for prices and partial utility

Clients can rent several types of VMs and choose the class associated to each one. Classes have two proposes. 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: $R = \{]0..0.2[, [0.2..0.4[, [0.4..0.6[, [0.6..0.8[, [0.8..1]\}$. The Pu function must be rewritten to take into account the class of the VM and the interval of the overcommit factor, as presented in definition 2. Doing so, Pu is a matrix of partial utilities. Each provider can have a different matrix which it advertises so that clients can choose the best option.

$$Pu : C \times R \rightarrow [0..1] \tag{2}$$

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 redefine the Pr function as presented in Equation 3. Similarly to the matrix of partial utilities, each provider can have a different *price* matrix.

$$Pr : C \times VM_{types} \rightarrow \Re \tag{3}$$

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.

C. Total costs

For a given client, the total cost of renting is simply determined by the total cost of renting each VM, as presented in Equation 4, where $RentVMs(c)$ represent the VMs rented by client c.

$$RentingCost(c) = \sum_{vm}^{RentVMs(c)} VMCost(vm) \tag{4}$$

The cost of each VM is presented in Equation 5 where N is the number of hours the VM was running, and P the number of depreciation periods in hour h. If after allocation the VM's depreciation factor remains constant, the P equals 1.

$$\begin{aligned} VMCost(vm) = & \\ & \sum_{h=1}^{N}\sum_{p=1}^{P} \frac{Pr(class(vm), VMType(vm))}{P} \\ & \cdot(1 - Df_h(vm, p)) \\ & \cdot Pu(class(vm), Df_h(vm, p)) \end{aligned} \tag{5}$$

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 depreciation factor imposed to each virtual machine. Because there are several classes of VMs, each with a particular partial utility for a given depreciation factor, the provider scheduler must find the allocation that maximizes (5). There are different ways to do so which we analyze in Section IV.

D. Practical scenario

As a practical scenario we consider that the partial utility model has three classes (High, Normal, 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 they are not met. Compensation may include not simply refund but also some level of significant penalization.
- **Normal** users who are willing to pay but will accept some depreciation 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.

For the three classes of our example, the cloud provider can define a partial utility matrix, represented by M in (6).

$$M = \begin{array}{c|ccc} & High & Medium & Low \\ \hline [0..0.2[& 1.0 & 1.0 & 1.0 \\ [0.2..0.4[& 0.6 & 1.0 & 1.0 \\ [0.4..0.6[& 0.4 & 0.8 & 1.0 \\ [0.6..0.8[& 0.0 & 0.6 & 0.8 \\ [0.8..1] & 0.0 & 0.0 & 0.6 \end{array} \tag{6}$$

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 (7) determines the base price ($/hour) for these types of VMs.

$$P = \begin{array}{c|ccc} & High & Medium & Low \\ \hline micro & 0.40 & 0.32 & 0.26 \\ small & 0.80 & 0.64 & 0.51 \\ regular & 1.60 & 1.28 & 1.02 \\ extra & 2.40 & 1.92 & 1.54 \end{array} \tag{7}$$

IV. Partial Utility based Scheduling for IaaS Deployments

In general, the problem we have described is equivalent to a bin packing problem [27]. So, the schedules must impose constrains on what would be a heavy search problem. Algorithm 1 presents what is hereafter identified as the base allocation algorithm. It looks for the host with more available cores and checks if it has available computational power, i.e. available number of *millions of instructions per second* (MIPS). It will eventually fail if no host is found with the number of requested MIPS, regardless of the class of each VM.

Algorithm 1 Base allocation

```
Require: $hosts$ list of available hosts
Require: $vm$ VM to be allocated
 1: function BASESCHEDULING($hosts$,$vm$)
 2:     $allocated \leftarrow false$
 3:     $tmpHosts \leftarrow hosts$
 4:     while $allocated \neq true$ & SIZE($tmpHosts$) $\geq 1$ do
 5:         $maxCores \leftarrow 0$
 6:         for all $h \in tmpHosts$ do
 7:             if AVAILABLECORES($h$) $> maxCores$ then
 8:                 $maxCores \leftarrow$ AVAILABLECORES($hosts[i]$)
 9:                 $freeHost \leftarrow h$
10:             end if
11:         end for
12:         $allocated \leftarrow$ TRYALLOCATE($freeHost, vm$)
13:         $tmpHosts \leftarrow tmpHosts - \{freeHost\}$
14:     end while
15:     return $allocated$
16: end function
```

Algorithm 2 checks if a VM can be allocated in a given host (h), that is, it returns $true$ if there are still available computational power in host h. If allocation cannot be done it returns $false$. Function $allocVmMipsInCores$ tries to allocate vm in the available cores. It will fail if the computational power (MIPS) of the VM is bigger than the cores of the host. It will end successfully if the VM can fit in any core, eventually shared with other VMs.

Algorithm 2 Try to allocate VM in host

```
Require: $host$ host used for allocation
Require: $vm$ VM to be allocated
 1: function TRYALLOCATE($host$,$vm$)
 2:     $allocated \leftarrow false$
 3:     $availableMips \leftarrow$ AVAILABLEMIPS($host$)
 4:     $requestedMips \leftarrow$ REQUESTEDMIPS($vm$)
 5:     if $availableMips \geq requestedMips$ then
 6:         ALLOCVMMIPSINCORES($host, requestedMips$)
 7:         $allocated \leftarrow true$
 8:     end if
 9:     return $allocated$
10: end function
```

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

We have developed four strategies/heuristics to guide our utility-driven algorithm. They differ in the way a host and victim VM is selected for depreciation. All start by looking for the host with more resources available, that is, with more unitary available cores and with more total computation power (MIPS). After a host is selected a VM must the chosen from the list of allocated VMs in that host. This VM (the $victim$) is selected either by choosing the smallest VM (which we call min strategy) or the one with the biggest size (which we call max strategy). There is also the possibility to instead look for VMs smaller than the VM which is trying to be allocated.

In resume there are four allocation strategies: min, max, min-class, max-class which we evaluate in Section VI.

V. Implementation details

We have implemented and evaluated our partial utility model on a state of the art simulator, CloudSim [12]. CloudSim is a simulation framework that must be 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, allocate execution tasks 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 [4]) and how execution tasks are assigned to VMs.

Regarding the CloudSim's base object model we have added information to the `VM` type regarding its partial utility class.

Number of hosts	Cores	Hz	Memory (Gbytes)
10	2	1860	4
10	2	2660	4

TABLE I: Hosts configured in the simulation

VM type	Virtual power	Memory (Gbytes)
micro	0.5	0.63
small	1	1.7
regular	2	3.75
extra	2.5	0.85

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

The scheduling algorithms were implemented as extensions of the type that determines how a VM is assigned to a host (`VmAllocationPolicy`). It can 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 (depreciated or not), we also created a new broker (extended from `DatacenterBroker`).

VI. Evaluation

In this section we evaluate the proposed scheduling based on partial utility. To do so, we first describe the small datacenter used in the simulation and the VM types whose base price was already presented in Section III-D. The datacenter is characterized by the number and type of hosts as described in Table I. Available VM types are presented in Table II.

A. *Utility Unware Allocation*

Figures 3 and 4 show the effects of using two different allocations strategies but still without taking into account each client's partial utility. Common to both experiences is the algorithm of allocating VMs, which chooses the host with more cores available, as described in Algorithm 1. The difference is the VMM scheduler. In Figure 3 each VMM (one for each host) allocates one or more cores to each VM and does not allow sharing of cores by different VMs. In Figure 4 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.

As expected, the core sharing algorithm promotes better resource utilization because the maximum effective allocation is 75% of the datacenter which compares with 71% maximum utilization of the other strategy. Nevertheless, in both cases, the datacenter starts rejecting the allocation of new VMs when it

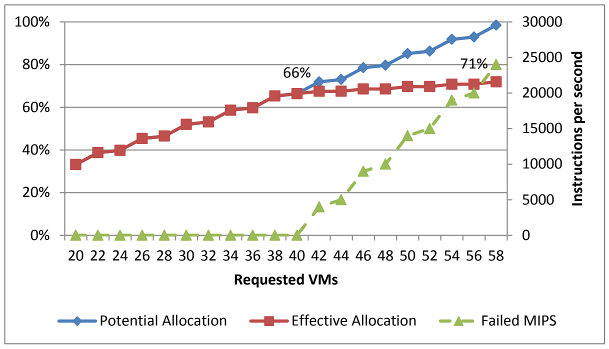

Fig. 3: Base algorithm with cores not shared between different VMs. Resource utilization ratio (effective allocation), potential allocation and requested but not allocated MIPS.

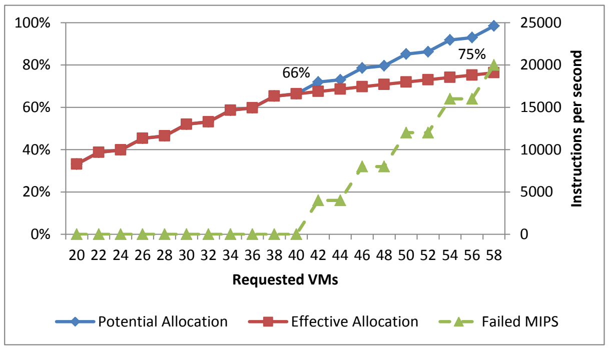

Fig. 4: Base algorithm when cores are shared between different VMs. Resource utilization ratio (effective allocation), potential allocation and requested but not allocated MIPS.

is about at 66% of its capacity, as can be observed by the difference between the potential allocation and the effective allocation series. The effective allocation yet continues to increase, at a slower rate, because there are still VMs that can be allocated. Figure 5 shows the counting of VM failures grouped by the VM type and VMM scheduling strategy. The datacenter rejects VMs of type *high* and *small*.

B. *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. We used several synthetic workloads with an increasing number of VMs trying to be allocated, as in the experimentations of the previous section. Each requested VM has a type (e.g. *micro*) and a partial utility class (e.g. *high*). We considered VM's *types* to be uniformly distributed (realistic assumption). Regarding the partial utility class distribution profile, in each workload, there are 20% of *high*, 50% of *medium* and 30% of *low*.

First we compare our approaches with the base algorithm described in Section IV regarding the number of VMs that were requested but not allocated. Figure 6 shows that while the base algorithm fails to allocate some VMs when 40 or more VMs are requested, the other four utility-driven strategies can allocate all requests.

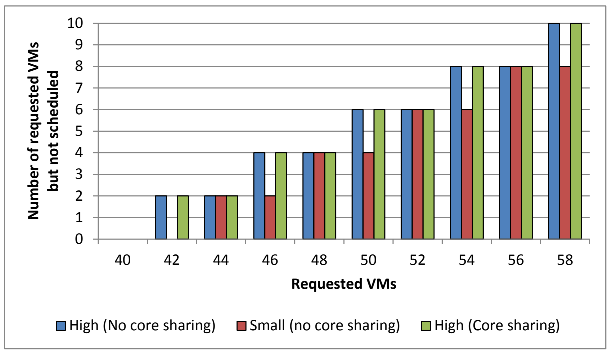

Fig. 5: Types and counting of requested but not allocated VMs

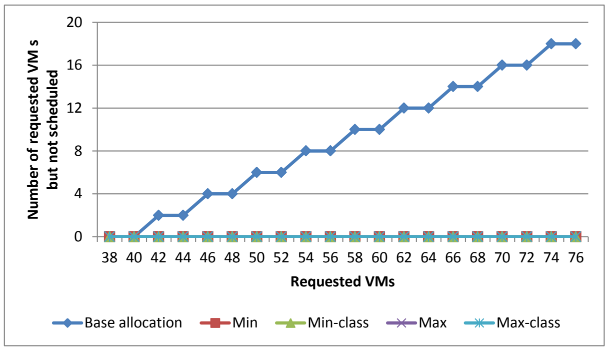

Fig. 6: Number of requested but not allocated VMs using different strategies

To further investigate the benefits of the proposed approaches we must evaluate how available resources are utilized, the revenue of the provider and the execution time of workloads. Figure 7 shows the percentage of *resource utilization* with an increasing number of VMs being requested for allocation. Two observations are worth noting. First, although with base allocation strategy some VMs are not scheduled, as demonstrated in Figure 6, others can still be allocated and can use some of the remaining resources. Second, it is clear that our four strategies achieve better resource utilization, while allocating all VMs, and that strategy min is the one that can better occupy available resources. Regarding revenue for the provider, Figure 8 further demonstrates the benefits of using a depreciation and utility-driven approach, showing that the provider's revenue can indeed increase if the rejected VMs (above 40) are allocated.

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 is from workloads executed during 10 days by thousands of PlanetLab VMs provisioned for multiple users [4], [28]. The average execution times are presented in Figure 9. The results shows that with more VMs allocated, even if depreciated, as it is the case, the average execution time of tasks running on those VMs is below the execution times achieved with the base strategy.

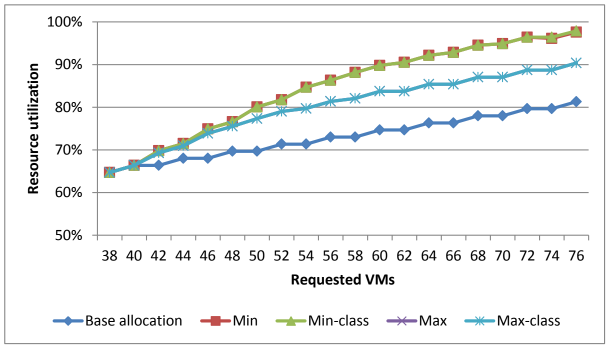

Fig. 7: Compared resource utilization

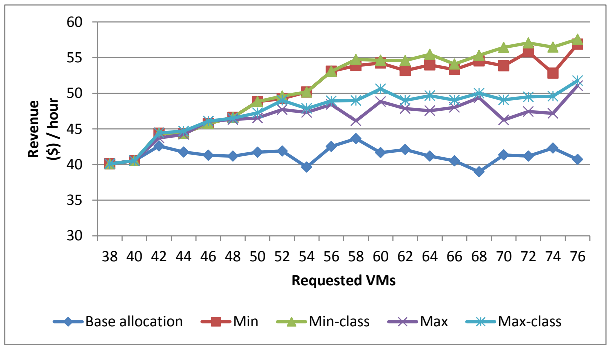

Fig. 8: Compared revenue

VII. 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 take 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 [4], [12]. The cost

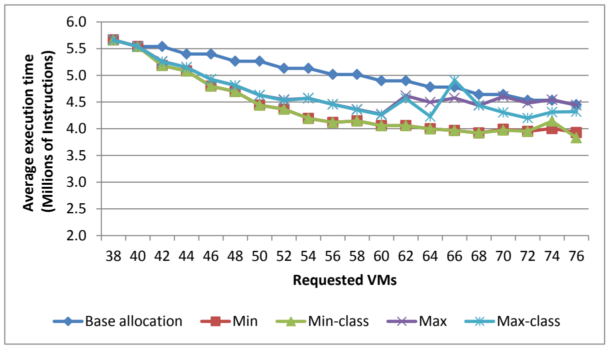

Fig. 9: Compared average execution time

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 depreciate 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. As future work we want to extend the scheduling process to incorporate progress information collected from workloads, such that resources can also be taken from workloads that use them less efficiently. This will need some extensions to the execution model of CloudSim. We also plan to incorporate this approach in private cloud solutions such as Eucalyptus.[4]

Acknowledgment

This work was partially supported by national funds through FCT – Fundação para a Ciência e a Tecnologia, projects PTDC/EIA-EIA/113613/2009, PTDC/EIA-EIA/102250/2008, PEst-OE/EEI/LA0021/2013 and the PROTEC program of the Polytechnic Institute of Lisbon (IPL).

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

[3] 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.

[4] 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.

[5] 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.

[6] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: fair allocation of multiple resource types," in *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, ser. NSDI'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 24–24.

[7] A. G. Michael Hines, M. Silva, D. D. Silva, K. D. Ryu, and M. Ben-Yehuda, "Applications know best: Performance-driven memory overcommit with ginkgo," in *CloudCom '11: 3rd IEEE International Conference on Cloud Computing Technology and Science*, 2011.

[8] J. Li, K. Shuang, S. Su, Q. Huang, P. Xu, X. Cheng, and J. Wang, "Reducing operational costs through consolidation with resource prediction in the cloud," in *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, ser. CCGRID '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 793–798.

[9] 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.

[10] A. Fölling and M. Hofmann, "Improving scheduling performance using a q-learning-based leasing policy for clouds," in *Proceedings of the 18th international conference on Parallel Processing*, ser. Euro-Par'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 337–349.

[11] 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.

[12] 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.

[13] 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.

[14] A. Gulati, G. Shanmuganathan, A. Holler, and I. Ahmad, "Cloud-scale resource management: challenges and techniques," in *Proceedings of the 3rd USENIX conference on Hot topics in cloud computing*, ser. HotCloud'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 3–3.

[15] 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.

[16] 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.

[17] 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.

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

[19] A. V. Do, J. Chen, C. Wang, Y. C. Lee, A. Y. Zomaya, and B. B. Zhou, "Profiling applications for virtual machine placement in clouds," in *IEEE CLOUD*, L. Liu and M. Parashar, Eds. IEEE, 2011, pp. 660–667.

[20] S.-M. Park and M. Humphrey, "Self-tuning virtual machines for predictable escience," in *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, ser. CCGRID '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 356–363.

[21] Z. Shao, H. Jin, and Y. Li, "Virtual machine resource management for high performance computing applications," *Parallel and Distributed Processing with Applications, International Symposium on*, vol. 0, pp. 137–144, 2009.

[22] J. P. X. W. J. L. F. H. Q. L. W. Z. X. F. S. K. Qin Yuan, Zhixiang Liu, "A leasing instances based billing model for cloud computing," in *6th International Conference on Grid and Pervasive Computing*, Springer, Ed., 2011, pp. 33–41.

[23] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh, "A cost-aware elasticity provisioning system for the cloud," in *Proceedings of the 2011 31st International Conference on Distributed Computing Systems*, ser. ICDCS '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 559–570.

[24] X. Len and L. Navarro, "A stackelberg game to derive the limits of energy savings for the allocation of data center resources," *Future Generation Computer Systems*, vol. 29(1), pp. 74–83, 2013.

[25] 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.

[26] A. Verma, P. Ahuja, and A. Neogi, "pMapper: power and migration cost aware application placement in virtualized systems," in *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, ser. Middleware '08. New York, NY, USA: Springer-Verlag New York, Inc., 2008, pp. 243–264.

[27] 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.

[28] 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.

[4]http://www.eucalyptus.com/