Flexible Use of Cloud Resources through Profit Maximization and Price Discrimination

Konstantinos Tsakalozos [#1], Herald Kllapi [#2], Eva Sitaridi [*3],
Mema Roussopoulos [#4], Dimitris Paparas [*5] and Alex Delis [#6]

[#]University of Athens, GR15748, Athens, Greece
{*k.tsakalozos*[1], *herald*[2], *mema*[4], *ad*[6]}*@di.uoa.gr*

[*]Columbia University, New York, NY 10027
{*eva*[3], *paparas*[5]}*@cs.columbia.edu*

***Abstract*—Modern frameworks, such as Hadoop, combined with abundance of computing resources from the cloud, offer a significant opportunity to address long standing challenges in distributed processing. Infrastructure-as-a-Service clouds reduce the investment cost of renting a large data center while distributed processing frameworks are capable of efficiently harvesting the rented physical resources. Yet, the performance users get out of these resources varies greatly because the cloud hardware is shared by all users. The value for money cloud consumers achieve renders resource sharing policies a key player in both cloud performance and user satisfaction. In this paper, we employ microeconomics to direct the allotment of cloud resources for consumption in highly scalable master-worker virtual infrastructures. Our approach is developed on two premises: the cloud-consumer always has a budget and cloud physical resources are limited. Using our approach, the cloud administration is able to maximize per-user financial profit. We show that there is an equilibrium point at which our method achieves resource sharing proportional to each user's budget. Ultimately, this approach allows us to answer the question of how many resources a consumer should request from the seemingly endless pool provided by the cloud.**

I. Introduction

Cloud computing introduces a number of challenges regarding both performance and financial issues. On the one hand, consumers of cloud services try to minimize the execution time of their submitted tasks without exceeding a given budget and on the other, cloud providers are keen on maximizing their financial gain while keeping their customers satisfied. With this work we focus on virtual infrastructures following the master-worker paradigm hosted in a cloud. In this type of infrastructure, there is a master node that dispatches jobs to worker nodes and increasing performance is a matter of adding extra worker nodes. This ease of expansion has been put into use by a number of programming frameworks, such as MapReduce[1], [2], and resource allocation management tools, including Condor[3] and TORQUE[4].

Much of the previous work on master-worker architectures targets scalability bottlenecks. Such bottlenecks may develop due to two factors: first, each infrastructure has its own hardware limitations. Second, the data processing algorithms implemented by the jobs cannot always be efficiently parallelized and therefore are not suited for this type of distributed environment. Research on the aforementioned bottleneck factors has resulted in solutions that can efficiently harvest the hardware resources of infrastructures of any size. Experience in Grids has shown that the combination of a resource allocation tool with a programming library for distributed programming (e.g., MPI) can fully utilize small to medium computer clusters [3], [4]. For larger computing infrastructures, frameworks such as MapReduce are shown to display outstanding scalability[1]. Yet, purchasing and maintaining such large physical infrastructures involves a high investment risk. *Infrastructure as a Service* (*IaaS*) clouds have greatly reduced the investment risk of owning an infrastructure, but introduce a new performance scaling factor: the user's financial capacity to rent virtual resources. This additional factor complicates the deployment and management of cloud architectures.

The value for money spent in a multi-tenant, elastic cloud renders resource sharing policies a key player in both cloud performance and user satisfaction. These policies must take into account the fact that cloud providers need strategies to evaluate and reduce possible financial risks while maximizing their profit. In addition, resource sharing policies must take into consideration the user's budget. Such policies often employ either auctioning [5], [6] or attempt to estimate user demand for resources [7]. In this way, the consumer needs are quantified based on their willingness to pay for the resources available. Microeconomic-based resource sharing policies thus allow for the distributed computing infrastructure to reach an equilibrium where the quality of service provided reflects the money spent.

In this paper, we propose a virtual-machine provision policy based on marginal cost and revenue functions. Each cloud customer announces her budget as a function of the execution time of the tasks she submits. Knowledge of this function, combined with the machine-hour cost, allows for educated decisions regarding the amount of virtual resources

This work has been partially supported by the *D4Science I & II* FP7-projects funded by the European Commission.

allocated per customer in the context of an *IaaS-Cloud*. The main contribution of our approach is that we provide an answer to the question of exactly how many virtual machines (VMs) a consumer should request from a cloud within a budget. In light of scalable, master-worker-based, virtual infrastructures and the seemingly endless resources of a physical cloud, specifying the exact amount of resources needed must be based on a) the consumer's budget and b) the performance bottlenecks which are known only at runtime. We propose a mechanism that continuously monitors user application performance and either "removes" or "adds" VMs in response to the observed performance fluctuations serving the needs of autonomic systems [8]. Our approach automatically adjusts to the ever-changing equilibrium point caused by dynamic workloads and thus ensures that resources are shared proportionally to money spent by the users. In addition, our approach is applicable to a wide range of computational environments since it does not enforce the use of a specific job submission tool. We allow each user to select any virtual infrastructure equipped with the tools of her preference.

The rest of this paper is organized as follows: Section II describes the problem our system targets. Section III gives an overview of the related work. In Section IV we describe the main aspects of our system in a single user environment. In Section V we present how our approach performs when multiple users coexist in a dynamic environment with ever-changing conditions. Section VI outlines both simulation and prototype experimental results while concluding remarks are found in Section VII.

II. Problem Statement

The master-workers paradigm is used by numerous programming frameworks and resource management tools to implement a number of parallel algorithms [9]. The master node acts as the coordinator assigning jobs to the workers. Since workers reside on different machines, they can process data simultaneously and collectively produce the output results. A user can effectively increase application performance by adding more worker nodes.

This paradigm is constrained by the physical resources available as well as the application-specific requirements. Satisfying the needs of such applications routinely involves setting and maintaining high-end, large data centers. The latter either exceed the financial capabilities of the interested parties or constitute a risky investment. *IaaS*-clouds, with their on-demand elasticity, are able to reduce such financial risks; they provide a seemingly-endless pool of computational resources needed to exploit the scalability of master-worker frameworks such as Map-Reduce [1]. Users contacting an *Iaas*-cloud request a number of virtual machines (VMs) to host the software framework of their preference. These VMs make up the elastic virtual infrastructure that will expand on-demand with the addition of extra worker nodes.

The decision on the exact number of worker-node VMs to be requested from an *IaaS*-Cloud is not a trivial one. *IaaS*-Clouds offer VMs at a specific monetary cost. Overconsumption of virtual resources must be avoided since underutilized worker-nodes reduce the value-for-money the cloud client achieves. Similarly, a conservative policy that underestimates the need for worker nodes may result in long processing times thus hampering user satisfaction.

When deciding on the number of worker VMs, one must consider both:

1) **The level of user satisfaction:** User satisfaction can be quantified through a budget function. However, low budget does not necessarily indicate that the user is satisfied with fewer VMs, rather a low budget may be the result of limited financial means on the user's part. In a multi-user environment, administrators may need to consider establishing a social scheduling policy that would prevent "poor" users from being entirely deprived of virtual resources. The risk here is that one may manipulate such policies and reduce the cloud's financial gain.
2) **The overall performance of the virtual infrastructure:** The number of worker VMs must never exceed the threshold over which extra workers become an overhead. As more nodes are added to the virtual infrastructure, the performance penalty incurred by the communication between the workers and the master node increases. The scalability displayed by the virtual infrastructure is often subject to this communication cost. In turn, this communication cost is subject to the algorithms used, the amount of data processed and the characteristics of the virtual and physical nodes [10], [11]. Moreover, neither the user nor the cloud administration are able to estimate the infrastructure's performance by themselves. The software stack installed and the data sets processed are known only to the user, whereas the exact physical node characteristics are known only to the cloud administration and are typically never revealed to the user.

Ideally, one could use an analytical model to predict the optimal number of worker VMs. Such a model would take into account *a)* the performance limitations set by the hardware, *b)* the interactions among all user applications simultaneously served by the cloud, *c)* the input data used by the applications in execution, and *d)* the performance constraints set by the software stack on both the physical and virtual infrastructure. Unfortunately, implementing this model is impractical if not infeasible especially considering the fact that several properties are known only at runtime. Our approach requires from each user to specify only her budget as a function of the total execution time of her workload. Assuming that such workloads are recurring, we employ an autonomic system that continuously monitors the infrastructure's performance and adjusts to any changing conditions. We handle the infrastructure as a black box –with the number of VMs as input and the total execution time as output–, thus our approach can be used in any master-

workers "elastic" architecture [2], [3].

III. Related Work

The idea of using economics in resource sharing policies is an old one. Two types of market-oriented resource sharing policies have been examined in terms of market stability and resource efficiency [12] . The two types are: a) markets where resources are shared through auctions and b) markets where resources are treated as interchangeable commodities[1]. In the latter market type, resources are given a fixed price determined according to their supply and demand [7]. Markets where resources are treated as commodities are applicable whenever the group of interested users does not allow the creation of an auction. Most often in commodity selling markets, user demand reduces as resource price increases. This kind of user behavior is assumed to study the conditions that maximize the resource provider's profit [15]. Markets with more than one type of commodity (such as CPU, network, storage) are well-suited for use in multi-user computing clusters. Here, policies set the price of all sold resources considering their correlations [7]. Policies that take into account only high-level performance metrics, such as the application response time [16], seem more appealing to the end users since less intervention is required on their part.

Resource allocation based on auctioning [5], [6], [17] is easy-to-implement and well-justified. Users quantify their need for resources through bids while at the same time resource providers offer their "products" for a price. In Mirage [18], a sensor network with limited resources, bids involving virtual currency in a closed loop market are used to express the user need for resources and prioritize requests. Similarly, in [17], auctioning is employed to provide access to low level resources such as CPU and RAM. Here resources are limited and users use proxies to bid for combinations of resources in periodic auctions.

Often, both consumers and producers are able to game auctions in their favor. In many cases, market gaming is an unwanted implication. An approach in tackling this implication is presented in [19]. Here, the agent matching the bids with the resource offers is allowed to reduce the resource supply thus protecting the producers from teaming consumers.

Due to its publicly available resources and the large user base, the Grid is used by many researchers as a testbed for various resource sharing policies [6], [12], [15], [16], [20], [21]. However, the nature of the Grid –publicly-funded and freely available– does not allow such resource allocation methodologies to be widely acceptable. Systems such as Tycoon [22] and Libra [23] are built on lower-level resource scheduling and allocation tools –as the resource manager of Torque system [4]– to bypass the complexity of Grids. In [21], resources of different grid sites are shared through a bidding process.

The management of *IaaS*-Clouds must tackle both technical and theoretical issues to achieve profit maximization for the cloud providers and satisfaction of QoS expectations set by the customers. Such infrastructures offer elastic services in a way that they can be measured, thus allowing for a billing policy to be enforced. In this way, clouds inherently provide ground for applying various economic models [24]. Service consumers almost always have some form of a utility function that corresponds to their budget while providers try to maximize their financial gain. For instance, in [25], a budget-based financial model is used to manage an adaptive cloud environment. Yet, unlike our approach, their model is not adaptive in terms of the number of cloud virtual machines deployed. Instead, their approach responds to user QoS requirements by building new data structures and/or choosing the most appropriate query execution plan.

By combining clouds with high resource availability [13] and a massively scalable programming framework [1], a user can process large amounts of data. *Dynamic Hadoop* [2] and Condor [3] integrate with "elastic" infrastructures to acquire resources upon demand. These frameworks add a software layer that efficiently manages administrative concerns of scaling virtual infrastructures. We build on the success of such highly scalable frameworks and infrastructures and enhance them with the application of microeconomic models for profit maximization. Contrary to much prior work [6], [12], [15], [16], [20], [21], our approach targets *IaaS*-Clouds and not the Grid. Our system acts as a gateway to the cloud for the users, taking into consideration high-level user requirements (i.e. application response time) and finally, designates a suitable number of virtual machines. To do so, we employ microeconomics to converge to a proportional sharing equilibrium [26]. At this convergence point, resources are allocated proportionally to the amount of money each user spends. Contrary to traditional auctioning-based systems [5], [6], [18], [17] our approach does not require users to bid. Rather, it takes as input a user budget function that portrays the money she is willing to pay for certain response time guarantees. The periodic auctions of [17] allow for resource sharing adjustments, but the resource mapping produced is not optimal, instead, in our case we achieve proportional share (fairness). The user budget functions we employ resemble the decaying value function of the execution tasks described in [21]. However, unlike our work, [21] proposes heuristics to balance the risks and gains involved. In addition, in our approach all tasks (corresponding to users) are served simultaneously, thus there is no need for any task scheduling (queueing and prioritize). This also comes in contrast to Mirage [18] where the goal is to prioritize requests according to their importance.

In a multi-user virtualized environment, as the one we find in the cloud, quality of service and efficient resource management are often contradicting high level goals. In [27],

[1]In Amazon Elastic Computing Cloud [13] we find both market types: users are allowed to buy resources at a fixed price or bid for them at a lower price [14]

the impact of resource overbooking in quality of service is examined. Autonomic systems [28], [29] are also shown to be able of efficiently managing resources under time-varying requests and multi-tier environments.

Our approach in resource provisioning resembles that of a monopoly where the cloud provider is able to enforce price discrimination [30]. In such a market the monopolist charges different prices to different consumers. We also employ an autonomic mechanism that monitors the user's applications and continuously adjusts the number of VMs to be used. This mechanism is required since the user application response time may change over time in ways we cannot predict. This approach allows us to treat the cloud as a black box providing us with resources. Unlike [2], [23], [31], we do not enforce the use of a specific job-submission tool. Instead, we let cloud consumers use the job management system of their preference and we ensure that the instantiated VMs register with it.

IV. Adaptive Cloud Management

In this section, we present a microeconomic model we use to designate a suitable amount of resources for the cloud consumers. Resources are dynamically requested on-the-fly from an *IaaS*-cloud in the form of Virtual Machines (VMs).

A. Overview of Our Approach

The main operational aspects of our approach are depicted in Figure 1. The *resource-sharing layer* implements the microeconomic model we propose. In the figure, we show two users who contact two separate virtual infrastructures, namely *Hadoop* and *Condor*. The interaction among the users, our framework, and the cloud involves three phases: first, each user sets a budget function. Second, jobs are submitted and their response time is monitored or announced to our framework. Third, worker VMs are added to or removed from the virtual infrastructures. The user has no involvement during the last phase. Also, note that the second and third step have to be repeated more than once to find the optimal number of VMs. Therefore, the workloads served by our resource sharing layer have to be recurring [9].

Two key factors in our approach are the cloud revenue (R) and the cloud cost (C). The cloud revenue is estimated using the user budget function and the monitored response times. The cloud cost is the cost of owning the cloud resources. In the case of a single user, our approach maximizes the difference between the revenue and the cost. In the case of multiple users, our approach allocates resources fairly; i.e., proportionally to the money each user is willing to pay.

B. Single User Adaption via Profit Maximization

When the user contacts our framework, she announces a budget that will accompany all her job submission requests. This budget $B(t)$ is a function of the amount of money per hour she can afford relative to her average job response time.

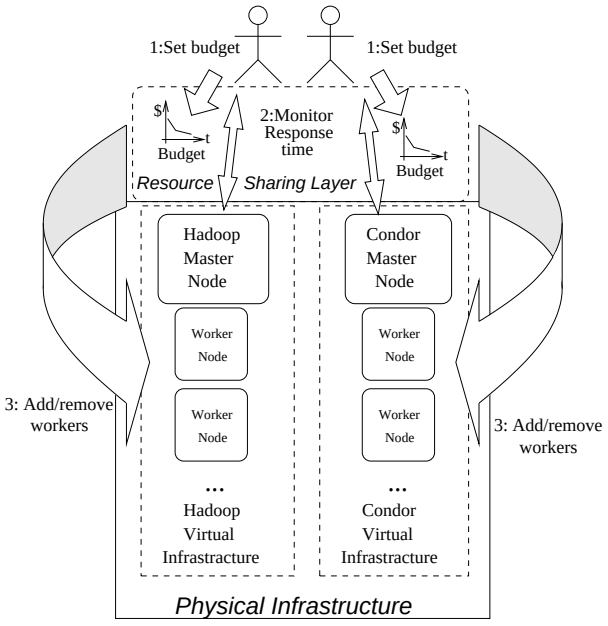


Fig. 1. Operational overview of our approach

As we will discuss later, a budget function would better be monotonically decreasing[2], i.e: $B(t_1) \leq B(t_2), \forall\ t1 > t2$. This means that users are not going to pay less if results are to arrive sooner. For each job submission, we keep track of the response time. By applying the average response time to the user's budget function we determine the amount of money available for buying virtual resources. This amount becomes the revenue the cloud will get per hour.

We expect the master-worker architecture present in the virtual infrastructure to decrease the application's response time when we deploy additional VMs. In turn, when we reduce the response time, the cloud's revenue (R) normally increases, since the user is expected to pay more. This is the reason why we suggest that budget functions be decreasing.

In practice, physical resources of clouds are limited and a performance bottleneck will eventually develop. Therefore, regardless the nature of this bottleneck (network communication and/or computation), the response time reaches a minimum for a specific number of VMs. This in turn means that there is an upper bound on the revenue. In Figure 2, we show a hypothetical revenue function. Note that the number of VMs offering a lower bound on the response time and an upper bound on the revenue may not be the most profitable in terms of value for money the user achieves.

A major factor influencing profit is the cost (C) of providing a VM. From the cloud's perspective, this cost includes server hardware and software, IT staff training and salary, power consumption and cooling costs, insurance, and downtime penalties. Although the virtual machine cost function is,

[2]Monotonically decreasing budget functions combined with a fixed price per VM ensures that our approach will not get trapped at a local minimum

undoubtedly, complex, we assume it is well-studied and known to the cloud financial administration. From the user's point of view, each VM usually has a fixed price. This price is the VM cost for the users. In any case, this cost per VM is known to our approach. Figure 2 depicts –apart from the revenue– the cost involved in purchasing VMs. We choose to display a cost function that grows linearly with the number of VMs. This fixed price per VM policy is common among *IaaS*-cloud providers [13]. When combining this VM billing policy with a monotonically decreasing user budget function, we get a single point where the profit R-C is maximized.

Figure 2 also presents the Marginal Revenue (*MR*) and the Marginal Cost (*MC*) that an additional unit of product (in our case, a VM) will bring:

$$MR = dR/dVM, \quad MC = dC/dVM$$

The profit is maximized when *MC*=*MR* at point A. Here, we should provide B VMs. Beyond B, each VM costs more than the revenue offered (*MC*>*MR*). Until we reach B VMs, the cost for each unit of product (VM) is less than the revenue we get from it (*MC*<*MR*). For B VMs, the total cost and revenue are E and D respectively; so the maximum profit P is: $P = D - E$.

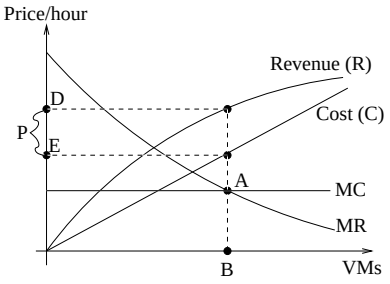


Fig. 2. Maximize profit using Marginal Revenue and Cost

As discussed in Section II, identifying the equilibrium point of maximum profit must be done through an iterative process during the consumer's interaction with the cloud. The reason for this is that response times, and thus the total and marginal revenue, are known only at runtime.

Algorithm 1 computes the number of VMs to be used in the virtual infrastructure by finding the equilibrium point where *MC*=*MR*. This algorithm takes as input the marginal cost (*MC*) and estimates at runtime the marginal revenue (*MR*). We choose to have *MC* as an input parameter as we expect either the cloud administration or the *IaaS* consumer to set the cost function. *MR* estimation is done through the user's budget function and the average response time T, both of which are provided as input. The final input, *step*, is the initial number of VMs to be used to bootstrap the algorithm. In lines 10–12, we compute the Marginal Revenue. In doing so, we use the average response time and the designated VM count of the last two runs. We keep this information in the static variables of the algorithm (lines 1–5). Note that for the first two runs we do not have enough data to compute *MR*. Therefore, for these runs, we simply increase the number of VMs by *step*.

Depending on the difference between *MR* and *MC*, we either increase or decrease the number of VMs we need active. The exact number of VMs to be added or removed plays a significant role in the number of iterations our approach needs to converge to the $MC = MR$ equilibrium. In Algorithm 2, we show how we decide on the number of VMs. The input of this algorithm is a variable named *direction*, indicating whether *MC* is greater than *MR* (or not) and the number of VMs which are currently active. There are two policies in changing the VM number: *a)* exponentially and *b)* linearly. Our approach is inspired by the congestion avoidance and control [32] employed by various TCP implementations.

The idea behind the exponential and linear increase/decrease is the same as with the slow start and congestion avoidance modes we find for the TCP congestion window. We start with the exponential VM-increase mode until we pass the maximum profit point. As soon as we are past the *MC* = *MR*, we start decreasing VMs linearly. In case we have *start_exp_move_after* steps of the same "*direction*", that is we have *start_exp_move_after* successive decreases or *start_exp_move_after* successive VM additions, we move again to the exponential mode. In this way, if the equilibrium point changes, we readjust quickly using the exponential change mode. Nevertheless if the equilibrium point stays the same, we perform a slight pivot around it using the linear mode. Lines 6 to 11 of Algorithm 2 count the successive iterations of the same "*direction*". Using this count, we determine which mode to follow (if clause of line 12). In line 18, we employ a sanity check that prevents us from suggesting too many VMs. We check this number of VMs against a threshold (*VM_threshold*) and if this threshold is exceeded, we change our increase mode to linear. *VMs_threshold* and *start_exp_move_after* constants are set to be 50 VMs and 5 iterations respectively. However, these values are bound to change according to the specific characteristics of the cloud infrastructure. Large infrastructures are able to sustain a higher number of VMs thus, the *VMs_threshold* should be set accordingly. Similarly, for infrastructures where VM instantiation is costly, higher values of *start_exp_move_after* may be appropriate.

C. Cloud versus User Profit

Our approach can be used either as a module "inside" the cloud or as a user-gateway to the cloud's resource provisioning mechanism. The difference between these two uses is essentially what the cost and revenue functions represent: when our approach is implemented as a cloud component, the revenue function is the amount of money the user pays while the resource cost is the internal operational cost of the cloud. In this case, the profit is the cloud's financial gain. When consumers employ our approach as a gateway to the cloud resource provisioning mechanism, the cost is the resource

Algorithm 1 Compute Number of VMs

Input: `B()`: User budget function
`MC()`: Marginal cost function
`step`: Step in changing the VM count
T: Average response time
Output: Number of VMs to provide

```
1:  static last_resp_time = 0
2:  static second_to_last_resp_time = 0
3:  static last_VM_count = 0
4:  static second_to_last_VM_count = 0
5:  static run = 1;
6:  cur_VM_count = 0;
7:  if (run == 1) OR (run == 2) then
8:     cur_VM_count = last_VM_count + step
9:  else
10:    dVM = second_to_last_VM_count - last_VM_count
11:    dR = B(second_to_last_resp_time) - B(last_resp_time)
12:    MR = dR / dVM
13:    if MR ≥ MC(last_VM_count) then
14:       cur_VM_count = VMs_Deviation("Up", last_VM_count)
15:    else
16:       cur_VM_count = VMs_Deviation("Down", last_VM_count)
17:    end if
18: end if
19: second_to_last_resp_time = last_resp_time
20: second_to_last_VM_count = last_VM_count
21: last_VM_count = cur_VM_count
22: last_resp_time = T
23: run++
24: return cur_VM_count
```

Algorithm 2 VMs_Deviation

Input: `direction`: "Up" or "Down" if MR $\geq$ MC or MR $<$ MC
`last_VM_count` Current number of VMs
Output: Number of VMs to provide

```
1:  constant VMs_threshold = 50
2:  constant start_exp_move_after = 5
3:  static last_direction = "Unknown"
4:  static same_direction = 0
5:  static VMs_increase = 1;
6:  if direction != last_direction then
7:     same_direction = 0
8:  else
9:     same_direction++
10: end if
11: last_direction = direction
12: if same_direction > start_exp_move_after then
13:    VMs_increase = 2 * VMs_increase;
14: else
15:    VMs_increase = 1;
16: end if
17: if direction == "Up" then
18:    if last_VM_count + VMs_increase > VMs_threshold then
19:       VMs_count = last_VM_count + 1
20:       same_direction = 0
21:    else
22:       VMs_count = last_VM_count + VMs_increase
23:    end if
24: else
25:    VMs_count = last_VM_count - VMs_increase
26: end if
27: return VMs_count
```

purchase price and the revenue is the money users are willing to pay. However, in this setting, the revenue is not paid to the cloud; instead, it is used as a measure of user satisfaction. Maximizing profit results in an optimum degree of satisfaction (revenue) per virtual machine. In brief, consumers maximize the value-for-money ratio.

V. Multiuser Dynamic Environments

Our approach thus far assumes a single user environment, an assumption that is not true when it comes to *IaaS*-clouds. Clouds are multi-tenant environments where users do not directly interact with each other, yet, the performance they get out of the infrastructure is greatly influenced by other users. In this section, we show how our model functions in such dynamic environments to reach equilibrium (*MC=MR*).

A. Multiple Users

Adding more *IaaS*-cloud consumers results in increased demand for cloud resources. The VMs of different users will compete over physical resources increasing the virtual infrastructure's average response time. In turn, increased response times decrease revenue. Figure 3 shows how the maximum profit equilibrium, point A, is shifted to the left as revenue and marginal revenue decrease. The shift from A to A' suggests that fewer VMs (B' instead of B) should be deployed to maximize the profit from a specific user. The interpretation of this per-user optimal VM provision point (B') depends on

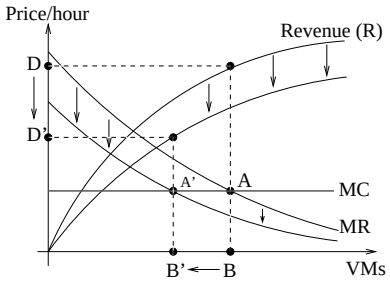


Fig. 3. Equilibrium shift in light of revenue reduction

the entity (consumer or cloud administration) that employs our approach:

- **Cloud:** When the cloud administration employs our approach it achieves maximum profit per user. We expect lower-level mechanisms, such as CPU scheduling and network bandwidth quota, to favor well-paying customers.
- **Users:** In case *MC=MR* is reached for each user separately, cloud's physical resources are assigned in such a way that any deviation from that would result in placing at least one of the consumers at a disadvantage (Pareto optimality). If we were to provide more physical resources at the same cost –without changing the consumers VM number– to a specific consumer we would be decreasing her jobs' response times. However, the rest of the consumers would be deprived of some portion of their

resources due to the cloud's resource sharing policies. These consumers will be at a disadvantage as their value-for-money ratio would deteriorate.

B. Proportional Share – Price Discrimination

The equilibrium point our approach reaches displays a number of properties examined both in the context of economics and resource sharing. With our approach the cloud "sells" resources at a different price to each user. This type of market resembles a variation of the monopoly where the monopolist enforces price discrimination among buyers [30]. Selling at different prices allows for maximizing profit per consumer. In this case, the total financial gain is greater than the financial gain the provider would have with a single price for all consumers. The main difference between our approach and monopolistic markets with price discrimination is that in our approach, we do not exert control over the resource/product supply, the amount of resources is fixed and we only need to distribute them among users/consumers. In contrast, the monopolist in markets with price discrimination is able to produce more products to satisfy the demand up to the point where profit is maximized for each customer separately.

We now show how our approach manages to share resources proportionally to the funds each consumer is willing to pay. This resource sharing policy property indicates that our approach treats users in a fair way. To show that our approach performs this type of fair resource sharing, we have to make certain assumptions. The overall goal of these assumptions is to model the user behavior and the cloud's runtime performance. These assumptions are:

Assumption 1: Each VM has a fixed cost K.

Assumption 2: The response time a single user would get out of the cloud if she were using v VMs is:

$$T(v) = \frac{M}{\ln(v)+1}, \tag{1}$$

where M corresponds to the time it would take the application to complete with only one VM. With this function we model the fact that response time will reach a saturation point and will improve only marginally after reaching a certain number of VMs [10].

Assumption 3: Each user i has a budget of the form

$$B_i(t) = \frac{\lambda_i}{t} \tag{2}$$

where λ_i is a per user-constant and t is the response time.

From assumption 1, we have that the cost is fixed per VM therefore:

$$MC = K \tag{3}$$

From equations 2 and 1, we can compute the revenue for user i given that she uses v VMs:

$$B_i(t) = B_i(T(v)) = B_i(\frac{M}{\ln(v)+1}) = \frac{\lambda_i}{M} * (\ln(v)+1) \tag{4}$$

Starting from the maximum profit point and using equations 3 and 4:

$$\begin{aligned} MC(v) &= MR(v) \Rightarrow \\ \frac{dC(v)}{dv} &= \frac{dR(v)}{dv} \Rightarrow^{(3)} \\ K &= \frac{dR}{dv} \Rightarrow^{(4)} \\ K &= \frac{d(\frac{\lambda_i}{M} * (\ln(v)+1))}{dv} \Rightarrow \\ K &= \frac{\lambda_i}{M} * \frac{1}{v} \Rightarrow \\ v &= \frac{\lambda_i}{M} * \frac{1}{K} \end{aligned} \tag{5}$$

In case we have n users each one with her own budget function ($B_i(t) = \frac{\lambda_i}{t}$) the ratio of the VMs allotted to user k is:

$$\frac{v_k}{\sum_{i=1}^{n} v_i} = \frac{\lambda_k/(M*K)}{\sum_{i=1}^{n} \lambda_i/(M*K)} = \frac{\lambda_k}{\sum_{i=1}^{n} \lambda_i} \tag{6}$$

Equation 6 shows that this VM allotment is proportional to the maximum amount of money each user is willing to pay. We acknowledge that the above modeling is inaccurate. Nevertheless, the modeling and its assumptions are used only for the needs of demonstrating the resource sharing properties. In practice, our approach makes no such assumptions as it constantly monitors the cloud's performance through the response times of the applications. The proportional share properties are also verified through our experimentation portion of which we discuss in Section VI.

C. Rapidly Changing Conditions

As is the case with any autonomic system [8], the period length chosen to monitor the response time plays a key role. In light of rapidly changing average response times, our method might not converge to the point of maximum profit. As a remedy, we could consider increasing the monitoring period length. Factors influencing the response time include:

- Variations in the workloads/jobs submitted to the virtual infrastructure. These variations include not only different execution tasks but also changing input parameters.
- Abrupt fluctuations in the amount of concurrent *IaaS* cloud consumers.

Our approach adapts to mild changes in the response time since Algorithm 1 computes MR at runtime. The period length between two successive suggestions must be such that the average response time indicates changes in workloads and resource availability.

VI. Evaluation

We have implemented our framework as a Java library so that it can be embedded in either consumer or cloud resource management systems. Our evaluation is two-fold.

First, we simulate a number of different cloud infrastructures serving multiple users. Second, we use our framework to assist resource management in a real cloud infrastructure setup in our lab. With the simulation we examine the behavior of our framework when used in large data centers while in the real cloud we show how our approach enables users to submit jobs to an "elastic" Hadoop infrastructure capable of growing or shrinking on demand.

A. Simulated Cloud Environment

We have simulated the behavior of a typical cloud middleware that serves user requests for VMs. The cloud middleware assigns the requested VMs evenly to physical machines using a load balancing algorithm. We assume that all users arrive at the same time and they submit the same workload continuously for a pre-defined number of periods. During the first period all users are assigned the same number of VMs. This may change according to the suggestions our approach provides. Of key importance to our simulation is the model used to estimate the workload execution time based on the resources provided by the cloud. The turnaround time of a job execution is given by the following formula:

$$Runtime(u) = \frac{1}{\sum_{i=0}^{N} \frac{VMs_{(u,i)}}{TotalVMs_{(i)}}} + Gauss(\mu, \sigma)$$

With N being the number of VMs user u owns, *VMs(u, i)* returns the portion of the VMs deployed on the physical node i. *TotalVMs(i)* is the total number of VMs deployed on node i. We also apply a Gaussian noise. The optimal amount of VMs per user is calculated at the end of every period and is used in the upcoming execution.

Table I shows all parameters of our simulation. We have three types of users, Richies, Bobs and Scrooges. Richies pay more than Bobs, and Bobs pay more than Scrooges. Users 1, 4, 7, 10 are Richies, users 2, 5, 8 are Bobs and Users 3, 6, 9 are Scrooges.

TABLE I
SIMULATION PARAMETER SETTINGS

Parameter	Value
Number of physical nodes	100
Number of users	10
Initial number of VMs per user	3
Maximum number of VMs per user	100
VM cost	0.15 $
Gauss	$\mu = 0$, $\sigma = 0.001$
Richie budget function	$B(t) = \frac{5}{\sqrt{t}}$
Bob budget function	$B(t) = \frac{3}{\sqrt{t}}$
Scrooge budget function	$B(t) = \frac{1}{\sqrt{t}}$

Figure 4 shows 10 plots, each one corresponding to a different user. The figure shows the percentage of time the user has at her discretion a specific number of VMs. For example, Ritchie User 10 spends 25% of her time using on average 50 VMs. When the system reaches the equilibrium point, we see that the number of VMs assigned to each user is proportional to the amount of money she is willing to pay. Thus, more VMs are assigned to Richies and fewer to Scrooges. The deviation from the equilibrium point increases as more VMs are used in our virtual infrastructure. This behavior is expected since in large virtual infrastructures the addition or removal of a single VM has minor impact on the overall performance.

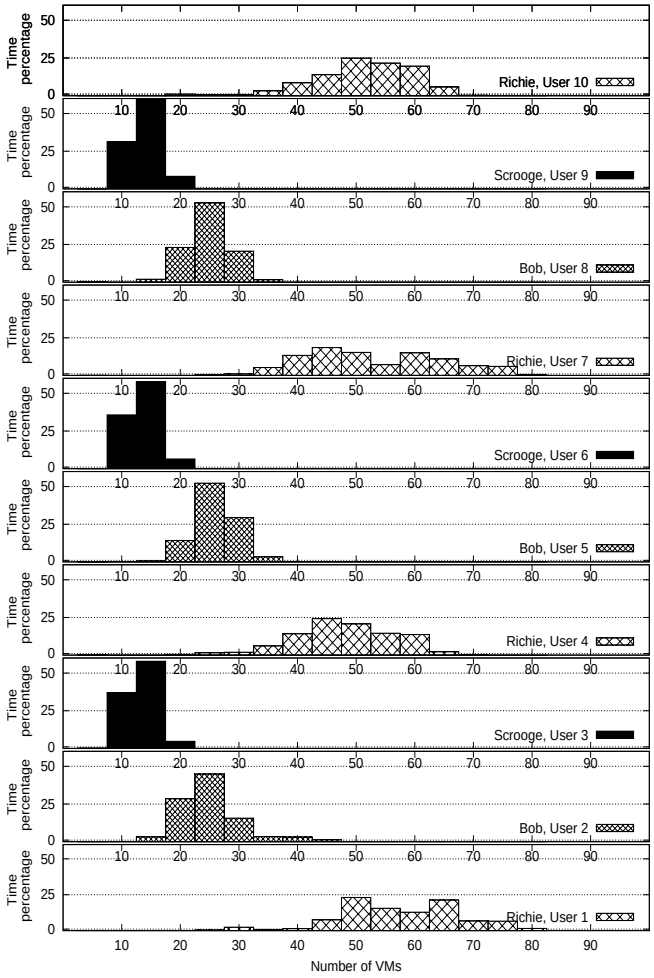


Fig. 4. Percentage of time spent with each number of VMs

In any proportional sharing policy the following formula holds for every user i.

$$\frac{Y_i}{\sum_{u=1}^{U} Y_u} - \frac{R_i}{\sum_{u=1}^{U} R_u} = 0 \ ,$$

where U is the number of users, Y_i is the amount of resources provided to user i and R_i is the revenue we get from her. Figure 5 shows that metric for every user during the simulation period after the equilibrium point is reached. In this way, we establish that indeed our approach converges to a proportional share state. The computed proportional share metric is almost 0 for all users with slightly negative values for the Riches.

The reason for this is that Riches deviate more than the rest of the users from their equilibrium point thus they are charged slightly more.

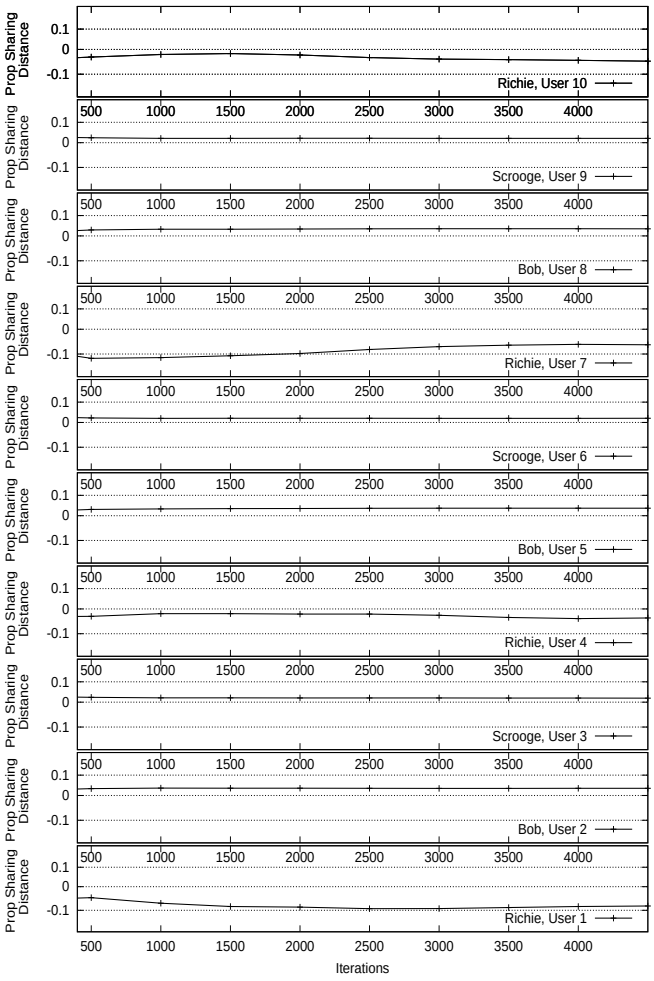


Fig. 5. User VMs proportional sharing

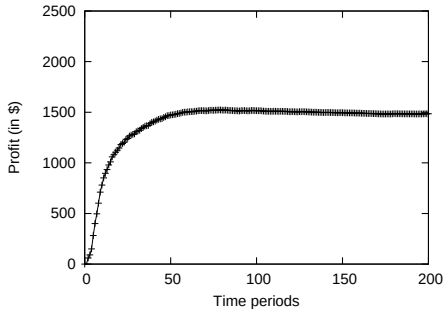


Fig. 6. Total Profit

Figure 6 illustrates the total profit of the system, when the latter reaches its equilibrium point. For nearly 70 time periods the cloud's profit increases. From that point on, our framework gradually reduces the deviation from the *MC=MR* equilibrium point.

Using the simulated environment we were able to test how long it takes our approach to reach a steady state with respect to the number of users simultaneously entering the cloud. Figure 7 shows the number of time periods our approach takes to reach a steady state as we increase the number of users contacting our simulated cloud. The estimation of the steady state is based on when the profit stops increasing. For each iteration, we compute the profit in our infrastructure from all workers. By applying linear regression on the latest five profit measurements we construct a straight line indicating the evolution of the profit in the recent past. When the slope of this line is below 0.2 for three consecutive iterations, we assume the system has reached a steady state. Figure 7 shows that the system becomes more stable when more users use it. The reason for this is that the impact of a single user's VM decisions on the rest of the cloud's users is greater when fewer users are present. In addition, a large number of users correspond to more VM adjustment operations per time period, and thus the steady state point is reached within fewer iterations.

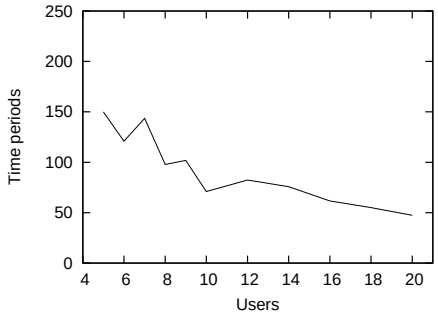


Fig. 7. Convergence to Proportional Sharing

B. A Private Cloud

To evaluate our approach in a real environment, we tested it in a private cloud setup in our lab. We have setup two types of "elastic" virtual infrastructures one using Condor [3] and a second with Hadoop [33]. In both infrastructure types there is a master node acting as a gateway. Through this gateway the user is able to submit jobs to be executed on the worker nodes.

Worker nodes register with the master node as soon as they become online. Both Condor and Hadoop feature a mechanism to remove unreachable nodes from their worker pools. Therefore, removing a worker involves shutting down the VM hosting the respective worker services and perhaps running additional shutdown scripts to ensure a graceful depart of the worker from the "elastic" infrastructure.

Each user has at her disposal her own private "elastic" infrastructure. When she contacts our cloud –apart from her budget– she specifies the type of virtual infrastructure she is going to use. Our cloud instantiates a gateway featuring a public IP and informs the user about it. Each user is aware

only of her own gateway. Users do not interact with each other, they are entirely isolated within their set of virtual machines. During operation, worker nodes register only with the gateway of the user that triggered their instantiation; there are no workers shared among users. With this setup, the resources shared are only those of the physical nodes.

In our lab, the cloud's hardware is setup on a rack where six dedicated nodes serve VM instantiation requests. All physical systems are connected through a $1\ GBps$ Ethernet switch. Each such node is equipped with $8\ GB$ of RAM. Two nodes are each equipped with an Intel(R) Core(TM)2 CPU 6600 at 2.40GHz while the other four have an Intel(R) Xeon(R) CPU X3220 at 2.40GHz. Live migration is not available and VM disk images are fetched on-demand from a file server when they are to be deployed.

The VM hypervisor we use is *Xen* 3.2-1 [34]. In order to treat all hardware nodes as a cloud we have setup the *OpenNebula v.1.2.0* [35] middleware. With *OpenNebula* we are able to automate the instantiation, migration and shutdown of VMs. These operations include the transparent copy or removal of disk images to/from the proper physical hosting nodes and the issue of respective *Xen* commands. To harvest physical resources in the best possible way, virtual worker nodes should be evenly distributed among physical VM hosting nodes. To this end, we do not rely on the default VM scheduling and placement policy of *OpenNebula*; instead we use *Nefeli* [36]. *Nefeli* allows us to set VM deployment hints and constraints when entire virtual infrastructures are to be instantiated. All instantiated VMs use 512 MB of RAM and a single CPU core. To achieve this, we make use of the *Xen* VCPU-option to restrict the utilization of CPU cores.

In the experiments presented here, we focus on the Hadoop infrastructure. As with the simulated workloads, user interaction with the cloud consists of fixed time periods during which a job is submitted. The response time of each submission is tracked so as to set the exact number of the deployed VMs. At the beginning of each period, we request the addition or removal of VMs. As soon as a new node comes on-line it automatically registers with the Hadoop JobTracker running on the user's gateway using a predefined IP address.

A real job submission on a real cloud:
With this first use case of our approach we show: a) how a long lasting job can be split into several smaller ones to decide the optimal number of VMs to use, and b) how our approach functions in a real cloud environment.

The jobs submitted to our framework perform data processing on portions of *Wikipedia*. First, we make sure that data are properly stored on Hadoop's file-system (HDFS) so that subsequent data processing operations will not fail. The CPU intensive part of the job a executes the `grep` program on a 500 *MB Wikipedia* corpus file. `Grep` is shipped as part of the Hadoop example programs. Here, two map/reduce operations are executed in sequence. The first operation counts the occurrences of the query regular expression and the second operation sorts the matching strings according to their frequency. The submitted job proceeds with the distributed creation of a full text *Lucene* [37] index using *Katta* [38]. This operation functions on a different segment of *Wikipedia*, a 100 *MB* file, although it could use the results of the first `grep` operation.

Repeated job submissions of the aforementioned formation are used to index parts of *Wikipedia* while at the same time use our approach to efficiently use cloud resources.

During our experimentation, the budget function we choose to quantify the user need for performance is set to $B(t) = 1000 * 1/t$, where t is the response time. The cost per VM is set to $\$0.085$, the amount Amazon EC2 [13] charges for a small VM instance.

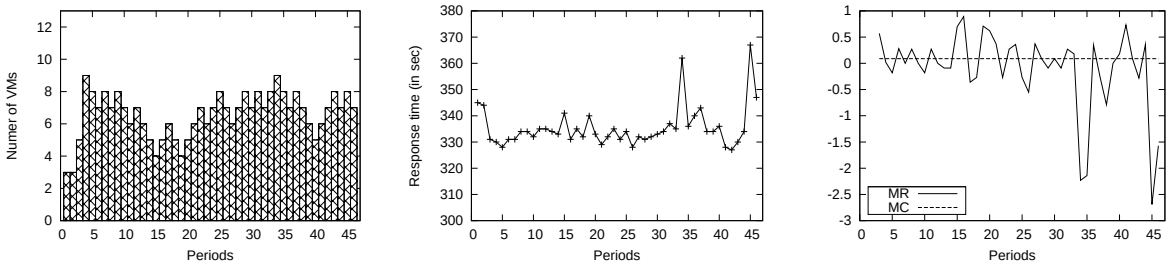
Figure 8(a) shows the number of VMs our approach designates to the user's job. Starting from 3 VMs our algorithm quickly converges to an average of around 7 VMs, with a maximum of 9 and a minimum of 4 VMs. In Figure 8(b) we present the response times the user gets from the cloud. Rapid deviations in response time such as the ones measured during periods 34 and 45 are the product of other users consuming cloud's resources.

Figure 8(c) shows the internal operation of Algorithm 1. Using the user's budget function and the response times we present in Figure 8(b) we are able to compute the marginal revenue. Figure 8(c) shows how *MR* pivots around *MC* in an attempt to maximize profit.

Impact of interaction among users:
With this second experiment we show what is the reaction of our approach to shortage of resources caused by the sharing of the clouds hardware. In order to have a clear view of the shared resource we have used a CPU intensive job. This job estimates the value of π by executing a MapReduce program on our "elastic" Hadoop infrastructure. The MapReduce program used is shipped as part of Hadoop examples and employs a quasi-Monte Carlo method to distribute computation. Random points are placed within a unit square. Comparing the amount of points inside the inscribed circle to those outside gives us an estimate of π.

We start with a cloud where the only tenant is the user submitting the above job. Eventually the maximum profit equilibrium point is reached and the number of VMs designated pivots around it. Then we initiate a CPU intensive operation consuming a considerable amount of the cloud's resources. This second job causes the equilibrium point of the first job to change and we monitor how our approach adjusts to this change.

In this experiment the cost per VM is also set to $\$0.085$. We use a step budget function presented in Figure 9. Note that with this function the user states that she is not willing to pay extra for any time improvements in the range of 0 to 200 secs. As a consequence our approach does not fully utilize all cloud



(a) VMs allocated per period (b) Response time per period (c) Marginal Revenue & Cost

Fig. 8. Evaluation in a real private cloud infrastructure.

resources.

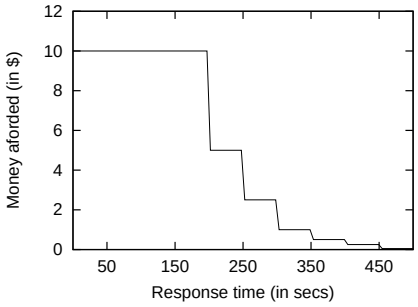


Fig. 9. Step budget function

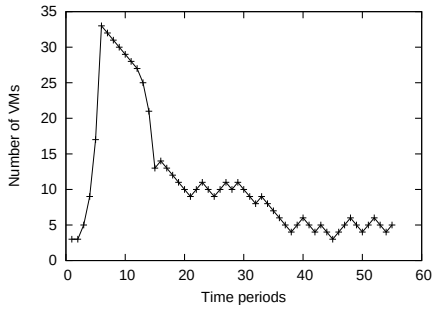


Fig. 10. Our approach following changes in the equilibrium point

In Figure 10 we show the VMs assigned to the user in each execution period. There are 3 phases: From period 1 to 18 our approach converges to the first equilibrium point. There is an exponential increase in the VM count until period 6. Then a linear decrease (periods 7 to 12) and then again an exponential decrease. From period 19 to 34 our approach suggests the use of 9 VMs on average, this is the first equilibrium point. On period 34 we start the background process. That process uses more than half of the the cloud's CPU resources. This causes the consumption not only of the idle CPU cycles but also part of the resources our job requires. Therefore, a new equilibrium point develops around the 5 VMs margin. Since the second maximum profit point is close to the first one our approach does not enter the exponential decrease mode, rather it decreases the number of VMs linearly. From period 38 to 56 our approach has reached the second equilibrium point.

C. Discussion on "Tiny" Clouds

Although the experiments shown here reached a stable state around the equilibrium point, our prior experience with a private cloud comprising of fewer physical nodes shows that such a point is not easily found. Often, in clouds with limited resources, the equilibrium point either exceeds the cloud's capacity or suggests that the users should not use the cloud at all since their profits are maximized for less than 1 VM. Moreover, in small cloud infrastructures, deploying VMs unevenly on physical systems results in seemingly unreasonable performance penalties. For example, three VMs co-deployed on the same physical node will perform worse than two VMs running on different hosting machines. Similar abnormalities emerge in the case of VM failures. Due to job re-submissions, a failing VM would stall the overall execution. However, in large infrastructures both uneven VM deployment and node failure have a reduced impact. Large numbers of VMs tend to be evenly distributed even under random-based scheduling policies. Also, VM failures can be effectively handled by the simultaneous submission of the same job to multiple workers.

VII. Conclusions - Future work

Combining resources from *IaaS*-Clouds with modern distributed computing frameworks allows for the effective handling of massively parallel problems. However, this combination introduces new challenges regarding both efficient use of cloud resources as well as user satisfaction. In this paper, we provide an answer to the key question of how many virtual machines (VMs) a user should request from an *IaaS*-Cloud given that users have a limited budget and that there are speed-up barriers set by the available physical resources. We follow a microeconomic-inspired approach to determine the number of VMs alloted to each user according to her financial capacity. Since the underlying physical resources are shared among all cloud tenants, the performance the users get out of the cloud may significantly vary over time. Therefore, our approach continuously monitors the response time of user applications and adjusts the amount of resources accordingly.

At its equilibrium point, the suggested approach maximizes profit. From the provider's point of view this profit corresponds to financial benefit whereas from the consumer's point of view, the same profit corresponds to quality of service received. Our experimental evaluation with both a detailed prototype and a simulator demonstrates that our proposed method converges to a fair resource sharing equilibrium point. At that point, the number of VMs provided to each user is proportional to the amount of money the same user is willing to pay.

In the future, we plan to investigate methods that reduce the time required to reach the maximum profit point and enrich pertinent approaches from the fields of both autonomic systems and economics. We also plan to examine the effects of open- and closed-loop markets on our approach. Finally, we wish to apply the same ideas on other resource sharing frameworks that do not rely on clouds to accommodate user needs.

REFERENCES

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, San Francisco, CA, Dec. 2004.

[2] HP Labs, "Dynamic Hadoop Clusters," http://wiki.smartfrog.org/wiki/display/sf/Dynamic+Hadoop+Clusters, Feb. 2010.

[3] http://www.cs.wisc.edu/condor/, "The Condor Project," 2010.

[4] http://www.clusterresources.com/pages/products/torque-resource manager.php, "TORQUE Resource Manager," 2010.

[5] D. Grosu and A. Das, "Auctioning resources in Grids: model and protocols: Research Articles," *Concurrent Computation : Practice and Experience*, vol. 18, no. 15, pp. 1909–1927, 2006.

[6] C. Chen, M. Maheswaran, and M. Toulouse, "Supporting Co-allocation in an Auctioning-based Resource Allocator for Grid Systems," in *Proc. of the International Parallel and Distributed Processing Symposium*, Fort Lauderdale, Florida, USA, Apr. 2002, pp. 89–96.

[7] K. Subramoniam, M. Maheswaran, and M. Toulouse, "Towards a Micro-Economic Model for Resource Allocation," in *In IEEE Canadian Conference on Electrical and Computer Engineering*. IEEE Press, 2002, pp. 782–785.

[8] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," *IEEE Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[9] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi, "Characterization of Scientific Workflows," in *3rd Workshop on Workflows in Support of Large-Scale Science*, Austin, TX, November 2008, pp. 1–10.

[10] D. L. Eager, J. Zahorjan, and E. D. Lozowska, "Speedup Versus Efficiency in Parallel Systems," *IEEE Transactions Computers*, vol. 38, no. 3, pp. 408–423, 1989.

[11] D. J. DeWitt and J. Gray, "Parallel Database Systems: The Future of High Performance Database Systems," *Commun. ACM*, vol. 35, no. 6, pp. 85–98, 1992.

[12] R. Wolski, J. S. Plank, J. Brevik, and T. Bryan, "G-commerce: Market Formulations Controlling Resource Allocation on the Computational Grid," in *International Parallel and Distributed Processing Symposium (IPDPS 01)*. San Francisco: IEEE, April 2001.

[13] Amazon, "Elastic Cloud," http://aws.amazon.com/ec2/, 2010.

[14] http://aws.amazon.com/ec2/spot instances/, "Amazon EC2 Spot Instances," 2010.

[15] V. Marbukh and K. Mills, "Demand Pricing & Resource Allocation in Market-Based Compute Grids: A Model and Initial Results," in *ICN '08: Proceedings of the Seventh International Conference on Networking*. Cancun, Mexico: IEEE Computer Society, Apr. 2008, pp. 752–757.

[16] C. E. Volker, V. Hamscher, and R. Yahyapour, "Economic Scheduling in Grid Computing," in *JSSPP '02: Revised Papers from the 8th International Workshop on Job Scheduling Strategies for Parallel Processing*. London, UK: Springer, 2002, pp. 128–152.

[17] M. Stokely, J. Winget, E. Keyes, C. Grimes, and B. Yolken, "Using a Market Economy to Provision Compute Resources Across Planet-wide Clusters," in *Proc. for the International Parallel and Distributed Processing Symposium*, Rome, Italy, May 2009.

[18] B. N. Chun, P. Buonadonna, A. AuYoung, C. Ng, D. C. Parkes, J. Shneidman, A. C. Snoeren, and A. Vahdat, "Mirage: A Microeconomic Resource Allocation System for Sensornet Testbeds," in *Proc. of the 2nd IEEE Workshop on Embedded Networked Sensors*, May 2005.

[19] A. Danak and S. Mannor, "Resource Allocation with Supply Adjustment in Distributed Computing Systems," in *Proceedings of the 30th IEEE International Conference on Distributed Computing Systems (ICDCS)*, Genoa, Italy, June 2010.

[20] J. Broberg, S. Venugopal, and R. Buyya, "Market-oriented Grids and Utility Computing: The state-of-the-art and future directions," *Journal of Grid Computing*, vol. 6, no. 3, pp. 255–276, Sep. 2008.

[21] D. E. Irwin, L. E. Grit, and J. S. Chase, "Balancing Risk and Reward in a Market-Based Task Service," in *Proc. of the 13th IEEE International Symposium on High Performance Distributed Computing*. Munich, Germany: IEEE Computer Society, 2004, pp. 160–169.

[22] K. Lai, L. Rasmusson, E. Adar, L. Zhang, and B. A. Huberman, "Tycoon: An Implementation of a Distributed, Market-based Resource Allocation System," *Multiagent Grid Systems*, vol. 1, no. 3, 2005.

[23] J. Sherwani, N. Ali, N. Lotia, Z. Hayat, and R. Buyya, "Libra: A Computational Economy-Based Job Scheduling System For Clusters," *Software: Practice and Experience*, vol. 34, no. 6, pp. 573–590, 2004.

[24] R. Buyya, C. S. Yeo, and S. Venugopal, "Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities," in *Proc of the 10th International Conference on High Performance and Communications(HPCC 08)*. Dalian, China: IEEE Computer Society, Sept 2008.

[25] D. Dash, V. Kantere, and A. Ailamaki, "An Economic Model for Self-Tuned Cloud Caching," in *Proc of the 25th IEEE International Conference on Data Engineering*, Shanghai, China, Mar. 2009.

[26] B. N. Chun and D. E. Culler, "Market-based Proportional Resource Sharing for Clusters," University of California at Berkeley, Berkeley, CA, USA, Tech. Rep., 2000.

[27] B. Urgaonkar, B. Urgaonkar, P. Shenoy, P. Shenoy, T. Roscoe, and T. Roscoe, "Resource Overbooking and Application Profiling in Shared Hosting Platforms," in *Proc. of the 5th USENIX Symposium on Operating Systems Design and Implementation*, 2002, pp. 239–254.

[28] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant, "Automated Control of Multiple Virtualized Resources," in *Proceedings of the 4th ACM European Conference on Computer Systems*. New York, USA: ACM, 2009, pp. 13–26.

[29] P. Padala, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem, "Adaptive Control of Virtualized Resources in Utility Computing Environments," in *Proc. of the European Conference on Computer Systems*, Nuremberg, Germany, 2007, pp. 289–302.

[30] H. R. Varian, *Intermediate Microeconomics : A Modern Approach*, 7th ed. W. W. Norton and Company, Dec. 2005, ch. 25, Monopoly Behavior.

[31] T. Sandholm and K. Lai, "MapReduce Optimization Using Regulated Dynamic Prioritization," in *SIGMETRICS '09: Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*. New York, NY, USA: ACM, 2009, pp. 299–310.

[32] V. Jacobson, "Congestion avoidance and control," in *SIGCOMM '88: Symposium proceedings on Communications architectures and protocols*. Stanford, California, United States: ACM, 1988, pp. 314–329.

[33] Apache, "Hadoop," http://hadoop.apache.org/, July 2010.

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

[35] "OpenNebula," http://www.opennebula.org, March 2010.

[36] K. Tsakalozos, M. Roussopoulos, V. Floros, and A. Delis, "Nefeli: Hint-based Execution of Workloads in Clouds," in *Procedings of the 30th IEEE International Conference on Distributed Computing Systems (ICDCS 2010)*, June 2010.

[37] Apache, "Lucene," http://lucene.apache.org/, July 2010.

[38] J. Zillman, M. Bauhardt, M. Schaaf, and S. Groschupf, "Katta," http://katta.sourceforge.net/, July 2010.