Free elasticity and free CPU power for scientific workloads on IaaS Clouds

Etienne Michon and Julien Gossa and Stéphane Genaud
LSIIT-ICPS - UMR 7005, Université de Strasbourg, CNRS
Pôle API Blvd S. Brant, 67400 Illkirch
email: etienne.michon@etu.unistra.fr, gossa@unistra.fr, genaud@unistra.fr

***Abstract*—Recent Infrastructure as a Service (IaaS) solutions, such as Amazon's EC2 cloud, provide virtualized on-demand computing resources on a pay-per-use model. From the user point of view, the cloud provides an inexhaustible supply of resources, which can be dynamically claimed and released. In the context of independent tasks, the main pricing model of EC2 promises two exciting features that drastically change the problem of resource provisioning and job scheduling. We call them *free elasticity* and *free CPU power*. Indeed, the price of CPU cycles is constant whatever the type of CPU and the amount of resources leased. Consequently, as soon as a user is able to keep its resources busy, the cost of one computation is the same using a lot of powerful resources or few slow ones. In this article, we study if these features can be exploited to execute bags of tasks, and what efforts are required to reach this goal. Efforts might be put on implementation, with complex provisioning and scheduling strategies, and in terms of performance, with the acceptance of execution delays. Using real workloads, we show that: (1) Most of the users can benefit from free elasticity with few efforts; (2) Free CPU power is difficult to achieve; (3) Using adapted provisioning and scheduling strategies can improve the results for a significant number of users; And (4) the outcomes of these efforts is difficult to predict.**

***Keywords*-cloud; provisioning; resource management;**

I. Introduction

Cloud computing sets a new paradigm for hardware infrastructure management by offering unprecedented possibilities to deploy software in distributed environments. Amazon EC2 is one of the most well-known solution to provide a utility computing model. Their solution has contributed to popularizing IaaS paradigm, which enables on-demand *provisioning* of computational resources, operated by virtual machines (VMs) in cloud providers' data centers. Resources can be reserved on a pay-per-use basis, thereby eliminating capital and maintenance costs for customers. Resource provisioning consists of finding how many VMs are needed, for how long, and the type of hardware configuration fitting the job needs. From a technical view point, prior to request a resource, the client must have prepared and transferred a customized VM image at the provider's site. Instances of these VM images can then be started, becoming ready to accept work tasks, and eventually released once they are no longer needed. Afterwards, the provider invoices the client according to the billing model, the VM type, and the up-time of the VMs.

Supported by the french ministry of defence - Direction générale de l'armement

This work focuses on the Amazon's pricing model called *on-demand*, in which users pay by the hour for active VMs. It is now widely adopted by other provider such as GoGrid, OVH or Rackspace... Although per-hour billing is widely spread, some companies propose billing time units (BTU) different from the hour, or alternative pricing models.

This type of pricing model presents two interesting features. First, the cost of deployment is linear in the BTUs spent, independently of the number of started VMs (modulo the deployment and shutdown overheads). Hence, one VM instance running for two hours costs the same as two instances for one hour. We call this property **free elasticity**. Second, the cost of deployment is linear in the number of available CPU cycles, independently of the speed of the CPU rented. Amazon expresses the power of the resources in EC2CU units, 1EC2CU being the power equivalent to what a 1-core Xeon 1.7GHz is able to deliver. A *small* instance at Amazon's costs 0.08$/hour for a power of 1 EC2CU, while the *extra-large* instance costs 0.64$/hour for a total power of 8 EC2CU: 8 times the price for 8 times the CPU power. They both share the same price per hour per EC2CU. Hence, running an extra-large instance during one hour costs the same as running the small instance during eight hours. However, this instance is actually 4 virtual cores of 2 EC2CU each. We call this property **free CPU power**.

Nevertheless, theses features present an important drawback: because BTUs are relatively coarse grain (e.g 1 hour) and each started unit of time is due, any job whose runtime does not last exactly this time unit leaves idle time on the deployed instances. However, this idle time, that would be charged anyhow, can be recycled to run jobs for no extra-cost. This leaves room for different type of provisioning strategies, which could exploit unused CPU time. A previous work of ours [1] explored the effectiveness of various strategies designed to scale the amount of VMs according to users' preferences. Our results showed how using queues on the provisioned VM allows to delay the execution of the jobs in order to both recycle idle time and make provisioning decisions. We assessed a dozen provisioning strategies and found out that: (1) the cheapest strategy, using only one VM in order to optimally reuse the idle time is not an option, because of the enormous waiting time it incurs for the waiting jobs; (2) the most expensive strategy, using one

VM per job to obtain the best execution time, often does not imply significant cost increase and is thus very interesting; and (3) when this last strategy is not interesting, a trade-off between cost and performance can be achieved using bin-packing based algorithms, provisioning a new VM only if the queues of the already provisioned ones are filled enough to cover their BTUs.

While we focused on the provisioning strategies in the previous work, we now concentrate on assessing whether the features of such pricing models allow actual elasticity with more powerful CPUs at no extra cost.

The paper is organized as follows. Section II explains our assumptions and the problem we address in this paper. In Section III, we present the different provisioning and scheduling strategies, while Section IV evaluates the pricing model. The results are further discussed in Section V. Finally, we present some related work in Section VI and conclude with our future work plans.

II. Assumption and Problem Statement

Assumptions: First, we assume that jobs are independent scientific computations with known durations. This is for example the case when users submit their jobs through a local resource management system, which requires users to specify a maximum runtime. Second, we consider that tasks are not preemptible, e.g the migration of a running task is not possible and jobs cannot be suspended to run another one. Third, we assume each user has its own VM, i.e a job from a given user cannot be run on a VM from another user. Fourth, we assume a semi-online scheduling system: tasks are dynamically scheduled but transit through waiting queues, their final assignment to a resource being computed at each scheduling round. Last, the pricing model is a piece-wise linear pricing model, such as *on demand* instances at Amazon. The billing period is discretized, each started billing time unit (BTU) being fully charged.

Problem statement: In this article, we focus on two types of EC2 on-demand standard instances: small instance with $1\,EC2CU$ and medium instance with $2\,EC2CU$. We choose these instances because they both provide a single core, and hence allow a direct comparison in the power of the CPUs. More powerful instances proposed make use of several cores. Job runtimes are normalized according to the small instance power. Therefore, during one BTU, one small instance can handle $3\,600$ seconds of computation whereas one medium instance can handle $7\,200$ seconds. In other words, one job with normalized runtime $3\,600\,s$ occupies one full BTU of small instance, but only half the BTU of a medium instance. Their respective prices are $0.08\$/h$ and $0.16\$/h$ as specified by Amazon on May 2012. Thus, for $0.16\$$ one can get either:

a) 2 BTUs of one small instance.
b) 1 BTU of 2 small instances.
c) 1 BTU of one medium instance.

The most interesting choices in general might seem b) and c) because the computations will finish sooner, and overall c) because it is the quickest choice. However, to benefit from this feature, one must be able to keep the BTU fully occupied. For instance, executing only one single $1\,000\,s$ job costs $0.08\$$ with one small instance, but $0.16\$$ with one medium instance. Therefore, only VM idle times induce extra-cost, and taking advantage of free CPU power and free elasticity requires to be able to keep the leased VMs busy. Our main objective is to check whether this requirement can be met and what are the implications in terms of workload management.

Let us introduce the problem of VM provisioning in the example of Fig. 1. The workload is composed of four job requests according to the chronology indicated on the time-line. The cheapest solution is to provision a single resource and to execute the jobs sequentially in the order they were submitted, as shown on the row labelled *1VM4All*. In this case, enqueuing the jobs for execution on one single VM leads to minimize the idle time, and thus the cost. An alternative is to reserve simultaneously several resources in order to execute some jobs in parallel, as exemplified on the other rows on Fig. 1. These executions will complete earlier but the cost is likely to be higher since the chances to waste the remaining time after the last job's end are greater.

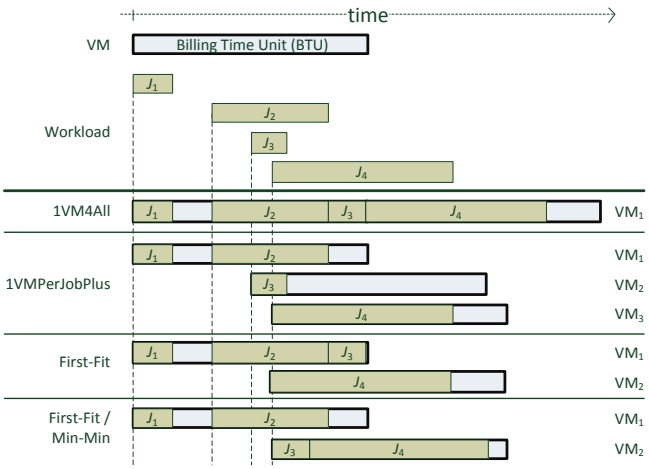

Figure 1. Illustration of the provisioning strategies

III. Definition of Strategies

A. Provisioning Strategies

We propose four strategies, illustrated in Fig. 1:

- ***1VM4All*** : The first strategy provisions a single VM and put all the jobs in its queue. It gives a lower bound on cost for the given workload because idle time is minimized.
- ***1VMperJobPlus*** : On the opposite side of the spectrum, we devise an "expensive" strategy, which minimizes the

waiting time thanks to a new VM deployment each time a job is submitted. We include to this strategy a straightforward optimization, which is to reuse a running VM if one is immediately available. On the example, the job J_2 is executed on VM_1 with no extra-cost, on the contrary to J_3 and J_4. This strategy is the most attractive because it is efficient in terms of performance (minimum waiting time), and its implementation has a low time complexity and only requires the knowledge of idle VMs.

- ***FirstFit*** : In our previous paper, we implemented adapted versions of the classic heuristics used for the online bin-packing problem [2]. Their general objective is to map the submitted jobs to already running VMs in priority, so as to minimize the number of started BTUs. As these strategies all showed to produce similar results, we choose only one of them here, namely *FirstFit*. It scans the list of already deployed VMs and maps the job to the first VM that does not require to extend the lease time over a new BTU, i.e. we map the job for no extra-cost. If no such already started VM exists, a new VM is deployed immediately. On the example, the execution of J_3 is delayed in order to reuse the idle time of VM_1, while a new VM is deployed to run J_4 because it cannot be handled at constant cost otherwise. While the *FirstFit* strategy (like the other bin-packing strategies) aims to minimize the rental cost, it results in longer wait times for jobs.
- ***Min-min*** : *FirstFit*, by design, makes an online scheduling decision, i.e it decides in which VMs BTU the job will fit as soon as the job arrives. This is not necessarily the best decision and this is the reason why a considerable average waiting time can be observed when processing real workloads. Better decisions can be made by relaxing the online constraint. Meanwhile a job waits in the queue, the scheduling algorithm can recompute, each time a new event happens, to which VM the job should be best assigned. This results in the reduction of the slowdown at constant cost. This semi-online scheduling is complementary to *FirstFit* and can be chosen among the heuristics described in [3]. Among these, only *Min-min* has been kept as a representative since our experiments showed similar results with the others. On the example, J_3 is initially enqueued on VM_1 before VM_2 is deployed to handle J_4. At the time of this new deployment, *Min-min* schedules J_3 on VM_2, reducing the slowdown induced. This strategy has a higher complexity, which might impose tight constraints to schedule a large number of arriving jobs in real time.

These two last strategies require more details from the workload. The duration of the jobs must be known, and *Min-min* requires to know what is the queue of waiting jobs for each VM.

B. Model

As stated previously, we consider an online scheduling system. To model the dynamic state of the system, we essentially need to account at a given instant, for the active VM we have started. We also maintain a *queue* of jobs assigned to each active VM.

The notations used are:

- V: Set of active virtual machines
- q_v: The job queue of $v \in V$
- b_v: The boot date of $v \in V$ (s)
- s_v: The shutdown date of $v \in V$ (s)
- i_v: The date when $v \in V$ becomes idle (s) (when q_v becomes empty)
- J: Set of arriving jobs
- r_j: The run time of $j \in J$ (s)
- w_j: The average wait time of $j \in J$ (s)
- $c(x)$: Cost of one virtual machine for x seconds of up-time. For EC2: $c(x) = pph * \lceil x/3600 \rceil$ where 3600 is the BTU and pph is its cost.

C. Common Algorithmic Phases

The strategies we propose can be expressed through algorithms sharing a common structure. These algorithms have two phases:

1) a *deploy* phase, invoked at each job submission. It consists of deciding (1) whether or not a new VM must be deployed, and (2) which active VM the job must be mapped to. It is described in Algorithm 1.
2) a *release* phase, triggered at a parametrized frequency. This release procedure is common to all strategies. It consists of deciding which active VMs must be shutdown and released. Each running VM is examined in turn, and an idle VM is kept running as long as it does not increase the cost. A shutdown occurs when it would incur additional charges.

Algorithm 1 Deploy(j,t)

// a new job j is submitted, at date t
$C \leftarrow \emptyset$ // C is the set of candidate VMs ($C \subset V$)
for $v \in V$ **do**
  if $eligible(v, j)$ **then**
    $C \leftarrow C \cup \{v\}$
  end if
end for
if $C \neq \emptyset$ **then**
  $v \leftarrow optimum(C)$
else
  $v \leftarrow deploy()$ // Create and run a new VM
  $V \leftarrow V \cup \{v\}$
end if
$enqueue(q_v, j)$ // Map the job to the VM

Table I
THE PROVISIONING STRATEGIES WITH THEIR RESPECTIVE PARAMETERS FOR ALGORITHM 1.

strategy	$eligible(v, j)$ returns $true$	$optimum(C)$ returns $v \in C$ such that ...	comment
1VM4All	always	$v = v_0$	Slowest/Cheapest - Reference
1VMperJobPlus	if $q_v = \emptyset$	any	Fastest/Most expensive
FirstFit	if $c(s_v - b_v) = c(s_v - b_v + r_j)$	any	Regular bin-packing strategy

Table II
WORKLOAD TRACE

Trace	#jobs	#CPUs	#user	Arrival	Runtime	Diameter
LCG	188 041	24 115	190	0 / 16 282 / 265 584	47 / 14 126 / 262 484	1 / 20 / 279.2
AuverGrid	336 086	475	337	1 / 265 542 / 6 222 766	1 / 13 726 / 134 973	1 / 9.5 / 141.5
NorduGrid	781 356	2 000	350	8 / 319 708 / 11 854 876	43 / 54 003 / 1 223 884	1 / 18.3 / 1 327
SharcNet	180 376	6 828	355	0 / 219 997 / 6 481 390	5 / 45 855 / 1 673 102	1 / 22.6 / 3 753
Unistra	306 605	1 000	74	0 / 117 006 / 4 638 583	3 / 51 906 / 638 346	1 / 26.4 / 142.7

Where:

- $eligible(v, j)$ is true if j can be assigned to q_v,
- $optimum(C)$ returns the virtual machine to which a job j is to be assigned,
- $deploy()$ provisions and starts a new VM and returns its identifier,
- $enqueue(q_v, j)$ adds the job to the queue of a given VM v. If v is available (i.e q_v is empty) the job actually starts immediately on v without being queued.

$eligible$ and $optimum$ allow us to define all our provisioning strategies. $eligible$ filters out the set of active VMs to which a job can be assigned depending on the current state of VMs. If this set is empty, then a new VM is deployed, otherwise $optimum$ selects the VM to assign the job to among the set of candidate VMs. These definitions are summarized in Table I. The *Min-min* scheduling algorithm is executed after this provisioning phase when *FirstFit* is applied.

Figure 1 shows an example use case. We can see that, whatever the strategy is, the VMs are only released at the end of the BTU, even after the execution of J_1 when there is no more job to run.

The efficiency of these strategies essentially depends on the characteristics of the workload, which is precisely what we evaluate in the next section.

IV. EVALUATION

Our study is based on the analysis of real workloads. We used four datasets from production grids, publicly available from the Grid Workload Archive [4] and a dataset from a local computing center at our University. The datasets' characteristics are presented in Table II, which lists the total numbers of jobs, the number of distinct CPUs used, and the number of users. Remaining columns present the minimum / average / maximum for the inter-arrival time between jobs, the average runtime of jobs and the diameter (the number of concurrent jobs). All times are in seconds.

LCG is a data storage and computing infrastructure for the high-energy physics community using the Large Hadron Collider at CERN. This production Grid has about 180 sites with around 30,000 CPUs. The traces collected include only high-energy physics (HEP) data processing. Eleven days of activity starting from Nov. 20, 2005 were logged. AuverGrid is a multi-site grid, part of the EGEE project. This grid is mainly used for biomedical and HEP applications. The logs account for one year of activity starting from Jan. 2006. NorduGrid is a production grid for academic researchers composed of over 75 non-dedicated clusters contributed mostly by academic but also industrial, scientific or private organizations. Applications are from the areas of chemistry, graphics, biomed, and HEP. The traces used here contain the grid jobs for three years starting from March 2003. SharcNet is a consortium of Canadian academic institutions who share a network of high performance computers. The traces analyzed were produced for a setup with 10 clusters over one year of activity starting from Dec. 2005. They where originally provided by the Parallel Workload Archive and analyzed in [5]. The trace from the computing center of *University of Strasbourg (unistra)* represents two years of activity (March 2009-March 2011) originating from different labs of the University. It consists of jobs reserving one or several processors, some being parallel jobs (e.g. MPI jobs). To respect our assumptions, we consider one job per reserved processor.

A. Evaluation Objectives

The objective of the evaluation is to highlight whether reducing the makespan, through the use of more powerful instances (CPU) or more instances simultaneously (elasticity), can be achieved for free. To that end, we have implemented the aforementioned provisioning and scheduling algorithms in a simulator of our own. We have extracted from the workload the submissions of each user (1 306 users). Afterwards, we have simulated the provisioning for each of these datasets

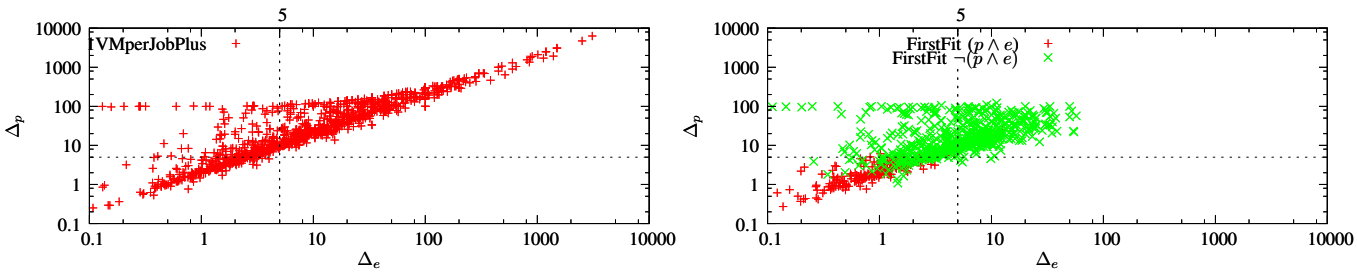

Figure 2. Distribution of users according to the difference between the cheapest strategy and *1VMperJobPlus* (left) and *FirstFit* (right)

and computed a wide range of metrics, both about simulation results and workload characteristics. Only the most relevant are presented in the following.

Total cost: Costs for the leasing of the provisioned resources, in $.

Slowdown: Average ratio of the wait time to the runtime for each job. It is defined as $sd_j = \frac{r_j + w_j}{r_j}$ [6], and measures the user's satisfaction. For instance, given one job j, $sd_j = 1$ means $w_j = 0$, and $sd_j = 3$ means $w_j = 2 \times r_j$.

From now on, we note S_{1x} and S_{2x}, the execution of strategy S using respectively small instances $(1x)$ and medium instances $(2x)$.

B. User Segmentation

In the following, we are mainly interested in near cost-optimal solutions defined as not exceeding the optimal by a factor ε. Our goal is to present a segmentation map of the 1 306 users of the 5 traces, depending on the extra-cost involved by using more elasticity and more powerful CPUs. We define *1VM4All*$_{1x}$ as the reference strategy. As explained in section III-A, this strategy gives the minimum cost, and hence used with small instances, represents the lower bound on cost. We consider the gain obtained for a very small cost increase is perceived by the user as "free", hence we term it in the following ε-free. We note R the reference cost: $R = (1+\varepsilon) \cdot cost(\textit{1VM4All}_{1x})$. This cost is then used to isolate the users in different exclusive categories with the following predicates:

$$
\begin{aligned}
e &= cost(\textit{1VMperJobPlus}_{1x}) \leq R \\
e' &= \neg e \wedge (cost(\textit{FirstFit}_{1x}) \leq R) \\
\bar{e} &= \neg(e \vee e') \\
p &= cost(\textit{1VMperJobPlus}_{2x}) \leq R \\
p' &= \neg p \wedge (cost(\textit{FirstFit}_{2x}) \leq R) \\
\bar{p} &= \neg(p \vee p')
\end{aligned}
$$

Fig. 2 shows, for each user, two extra-cost values as a couple of points (Δ_e, Δ_p): Δ_e represents the cost increase implied by using a given strategy S on small instances, while Δ_p shows the cost increase due to the application of the same strategy but with medium instances. The extra-costs are expressed in percentage of the reference cost. On the figure, results are shown for $S =$ *1VMperJobPlus* (left plot) and $S =$ *FirstFit* (right plot). Precisely, the cost incresases plotted are defined as

- $\Delta_p = \dfrac{cost(S_{2x}) - R}{R} \times 100$,
- $\Delta_e = \dfrac{cost(S_{1x}) - R}{R} \times 100$

The dashed lines on the plots represent the ε thresholds chosen in the analysis.

For instance, a point at position (10,100) (resp. x-axis, y-axis) means the corresponding user would see a 10% cost increase if he/she increases elasticity, and 100% if, in addition, CPUs with twice the normal power (medium instances) are used.

First, we observe that a large majority of users stay within a budget that is twice the lowest reference cost. However, *1VMperJobPlus* yields for some users, costs that are in the order of ten times the optimal (maximum of 3100%). In contrast, *FirstFit* is cheaper, with a cost always lower than twice the optimal. Second, we highlight that *FirstFit* reduces the price for most users (93.49%). Last, we see that for all users, the additional costs due to the increase in CPU power are always greater than those due to elasticity increase. In the following, we consider that a 5% increase is reasonable as no user get 0% price increase. So we focus on the case where ε=0.05.

Table III presents an overview of this segmentation process for ε=0.05. Columns represent the effort to achieve ε-free elasticity: for the population of users in column (e), using *1VMperJobPlus* is enough; for the population in column (e'), *FirstFit* must be used to reach this goal; column $(\bar{e})$ counts the users for who none of the strategy can offer free elasticity. Rows represent the effort to achieve ε-free CPU power: (p) when *1VMperJobPlus* is enough; otherwise (p') when *FirstFit* must be used; otherwise $(\bar{p})$ when it cannot be achieved.

Each table cell represents the share of users who match the conjunction of predicates in corresponding row and column. Values between brackets are the absolute number of these users. For instance, the second row left cell says that 4.29% of the users can benefit either from ε-free elasticity applying the strategy *1VMperJobPlus*$_{1x}$ or ε-free CPU power applying *FirstFit*$_{2x}$. It yields, in both cases, a cost not greater than ε the lower bound. Notice also, that though not shown here, the distribution of users in the different categories is roughly

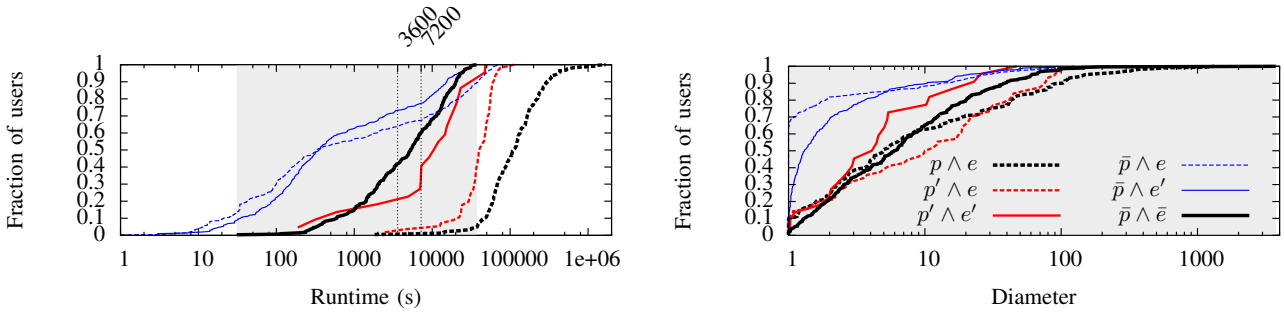

Figure 3. Cumulative distribution functions of users according tothe average runtime (left) and diameter (right) for strategies achieving ε-free elasticity ($\varepsilon = 0.05$).

the same whatever the workload is.

Table III
USER SEGMENTATION FOR $\varepsilon = 0.05$. SHARE OF USERS (AND NUMBER) IN CATEGORIES.

CPU \ Elasticity	e	e'	$\bar{e}$
p	**17.08%** (223)	**0.08%** (1)	**0%** (0)
p'	**4.29%** (56)	**1.68%** (22)	**0.23%** (3)
$\bar{p}$	**31.16%** (407)	**23.28%** (304)	**22.21%** (290)

By summing up the values in the first column, we notice that 52.53% of the users would take advantage of using *1VMperJobPlus*$_{1x}$, thereby getting elasticity for free. If we extend to the two first columns, this share grows up to 77.57%. Reading the table row-wise shows that only a small share of the users is able to use medium instances at the cost of the small ones. There are 17.16% such users with no extra waiting time, and 23.36% if we include the *FirstFit* strategy. From these figures, we conclude that choosing *FirstFit* is valuable in two cases: it allows 23.28% of the users to reach the near-optimal cost with small instances, and for another 6.2% (4.29%+1.68%+0.23%), using medium instances is possible for no extra-cost.

However, if *FirstFit* allows to keep the price low, it also induces unacceptable slowdowns very often. This effect can be lowered by applying the *Min-min* scheduling, which in our experiment, reduces slowdowns to acceptable values: the median slowdown falls down to 2.14. Out of the 386 concerned users (those in the grayed out central cells in the table), 252 see no improvement using *Min-min*, while for the 134 remaining ones (34%) the slowdowns are reduced of 32% on average. But just 21 users (5.4%) fall down to a slowdown less than 3. Consequently, to stay within the 5% cost increase, users in these categories must overall accept considerable wait times. It is not enough to compete with *1VMperJobPlus* as it represents a significant effort in term of performance.

C. Per-characteristics Analysis

We now try to answer the question: "Is there a good workload characterization which would allow us to predict the effects of strategies?". In our previous work [1], we tested the strategies with the objective to put forward good cost-wait trade-offs, and we found out a direct relationship between the diameter, the runtime and the cost. Therefore, we examine if the runtime and the diameter are also key factors regarding the categories. For this analysis, we leave out the three table cells containing only a few users. We plot in Fig. 3 the average runtimes and diameters for the six categories left. For a sake of comparison, the particular category of users who cannot keep their additional cost within 5% of the reference cost, with any strategy (i.e. $\bar{p} \wedge \bar{e}$) is highlighted by the grayed background which delimits the range of values for runtimes and diameters observed for this category. The label of the curves (only on the right figure) are common to both figures. On left figure, each curve corresponds to the cumulative distribution function (CDF) of per-user average runtimes in a category. For instance, about 70% of the users belonging to the category $\bar{p} \wedge e'$ have a job average runtime greater or equal to 3600 s. Similarly, the right figure shows the CDF of per-user average diameter, i.e the number of VMs running simultaneously.

The first observation from the runtime CDF is that the users from the category $p \wedge e$ are characterized by very long runtimes (160 610 s on average), allowing to keep medium BTU busy.

Second, users from the category $p' \wedge e$ show slightly shorter runtimes (42 813 s on average). However, 40% of the users have an average runtime in the highlighted zone. This implies that the VMs cannot be kept busy using only *1VMperJobPlus*$_{2x}$, as some jobs become small according to the medium BTU (7 200 s), leading to idle times. However, *FirstFit* succeeds to reuse these idle times in some cases.

Third, we see that the runtime is not a decisive characteristics for the four remaining categories. A user with a given average runtime could be in either $\bar{p} \wedge \bar{e}$, $p' \wedge e'$, $\bar{p} \wedge e$, or $\bar{p} \wedge e'$.

Finally, we expected that a high diameter would increase the possibilities to reuse idle time. However, the CDF of diameter surprisingly shows that whatever the diameter, we are not able to prognosticate the outcome of a strategy given

a workload. For instance, the user sharcnet-U111 has a diameter of 3 753 and cannot benefit from either free elasticity or free CPU power. The only exception is the diameter equals to one which works fine with *1VMperJobPlus*$_{1x}$.

The most interesting observation is that it is impossible to predict in which pool a given workload will be, except for *1VMperJobPlus*$_{2x}$ which concerns very long runtimes. Furthermore, most of the pools have workloads sharing exactly the same characteristics, especially the not ε-free case ($\bar{p} \wedge \bar{e}$) which cannot be separated from the rest, as the highlighted zones show. We conducted an intensive study of each individual workload and strategy behaviour, using chart of submission and provisioning, together with more advanced metrics, like the heterogeneity of runtimes and diameters. Our conclusion is that the thresholds due to coarse grained BTU impose micro management and micro analysis of each single submission. It actually makes it impossible to predict the outcome of each strategy according to macro metrics like average runtime or diameter. For instance, let us consider two jobs of 3 599 s and 3 601 s. The first fits perfectly in one small BTU, while the second leads to double the cost. Moreover, if they are submitted simultaneously, *FirstFit* performs well, but not *1VMperJobPlus*; if they are submitted consequently *1VMperJobPlus* performs well, but not *FirstFit*; but none works if there is a gap of 1 s between the two submissions.

D. Conclusions

While achieving free elasticity is easy, taking advantage of free CPU power is more difficult. Only very peculiar workloads, with very large runtimes, can achieve both with few efforts using *1VMperJobPlus*$_{2x}$. Users with slightly lower runtimes might use *FirstFit*$_{2x}$.

With small instances, free elasticity is achievable, since for most of the users *1VMperJobPlus*$_{1x}$ is sufficient, and *FirstFit*$_{1x}$ works well for a large share of them.

Whatever the CPU power, using *FirstFit* implies important slowdowns. They can however be significantly reduced with the *Min-min* scheduling algorithm.

However, predicting the outcome of these choices given global workload characteristics only is impossible. Our recommendation is that, either some peculiar conditions are gathered (very long runtimes), or assessment tools should be used as decision support.

V. Discussion

We have shown that even though IaaS pricing model promises free CPU power and elasticity, it is impossible to go by the characteristics of the trace to make a good choice of the strategy. However, our study is limited on several aspects.

First, certain technical details have been abstracted from the simulation, like the startup times (deployment, boot) and stop times (shutdown, release) of VM. These details do not impact the relevance of our study, as they are negligible face to the BTU length. For example, the time to instantiate and start a VM instance at Amazon's EC2 is 60 to 130 seconds according to a comprehensive study [7], while the BTU is 3 600 seconds. Moreover, we suppose that the SLA is respected. According to Dejun et al. [8], this is not always the case as the computing power can vary from a VM to another. Such details, together with the economic model of communications, have to be taken into account by a real brokering system, and should be handled carefully to optimize the solutions.

Also, we have considered the same runtimes on the cloud platform as in the workload traces because these last do not include information about the used CPU. This does not change the conclusions of this study, but only implies that the computed costs and waiting times are not realistic.

Next, we filtered some of the workloads to consider that a job reserving several processors is one job per reserved processor. However, some are parallel jobs (e.g. MPI). Moreover, some instances are composed of several virtual cores. These issues will be addressed with one stone thanks to *bound BTUs*,starting and stopping together: while x cores of one VM can be mapped to x *bound BTUs*, y parallel jobs can lead to provision y *bound BTUs*.

Amazon EC2 split the different types of instances into five categories. Instances among a category share the same price per hour per EC2CU. Our study focuses on standard instances as they are widely used. Using instances from another category would change the price per hour per EC2CU and thus, changing the paradigm of free CPU power.

Last, our study focused on bags-of-tasks only. This is one of the hardest application field as it presents hardly no specific submission pattern. Our results do not apply to other fields of application. For instance, our strategies are not suitable for web hosting applications, having very numerous and very short jobs that cannot wait. They can fit to a HPC usage although network traffic should be taken into account. However, long jobs should suit with our best case (*1VMperJobPlus*$_{2x}$). Similarly, our study does not apply to pricing models based on fine grained BTU.

VI. Related Work

The problem of provisioning IaaS cloud resources has been addressed by a number of works, but a minority of them specifically address the client cost/benefit concern. For example, Deelman et al. [9] illustrate with a concrete example the need to analyze the cost-benefit of supplementing in-house computers with cloud resources. In [10] a batch scheduling system able to submit jobs onto clusters and/or a cloud is proposed. The authors investigate how classic scheduling algorithms behave in terms of cost when external cloud resource can be granted to increase the performance. In contrast to these works, which consider a mixed type of resources, other studies as ours consider only cloud resources. In this category, [11] addresses the problem of provisioning resources for independent tasks with known durations, so as to maximize the speedup under

a budget constraint. Workloads made of independent tasks of unknown durations are considered by Oprescu and Kielmann [12]. In the context of a multi-cluster system (or multi-cloud), they propose a batch scheduler which continuously estimates the task execution runtimes on each cluster using statistical inference based on previous observations. This process drives the choice of the resources to use in order to best meet a budget constraint. Closest to our previous work is the recent paper by Villegas et al. [13]. They present a similar performance-cost analysis for a set of provisioning and allocation strategies. The provisioning heuristics they propose can be seen as extensions to those proposed in [14] to start an appropriate number of VMs face to different workload types. They test their strategies both through simulation and on reduced size cloud infrastructures. Differently from us, they choose to work with synthetic workloads consisting in jobs with very small runtimes ($47\,s$ on average), which correspond to specific MapReduce workloads.

VII. Conclusion and Future Work

In this article, we have put forward two properties induced by the common pricing model for IaaS, which we call free elasticity and free CPU power. We have shown, through the simulation of different scheduling stategies using real traces of bag-of-tasks applications, that free elasticity is possible for a majority of user's workloads, while free CPU power applies in a few cases only. This latter property is only observed with peculiar workloads having very long runtimes.

We have shown that for 54% of the users, *1VMperJobPlus* is sufficient to achieve at least free elasticity; and that *First-Fit*, can be efficient in 25% of the cases inducing wait times. We have also shown that applying scheduling strategies to queued waiting jobs can improve the performance and the user satisfaction by reducing the slowdown for 35% of them.

Nevertheless, we found out that predicting the outcome of these efforts is very difficult in the general case. Indeed, the thresholds implied by coarse grained BTU have a large impact on provisioning strategies behaviour, that cannot be predicted using macro metrics like average runtime or diameter.

Therefore, our future work will focus on developing an assessment tool to be used as a decision support. The main idea is to create a simulator based on the SimGrid [15] toolkit to simulate the different strategies each time a job is submitted in order to make a more suitable choice.

Acknowledgment

This work is partially supported by the ANR project SONGS (11-INFRA-13).

References

[1] S. Genaud and J. Gossa, "Cost-wait trade-offs in client-side resource provisioning with elastic clouds," in *4th IEEE International Conference on Cloud Computing (CLOUD 2011)*. IEEE, Jul. 2011.

[2] E. G. Coffman, M. R. Garey, and D. S. Johnson, *Approximation algorithms for bin packing: a survey*. Boston, MA, USA: PWS Publishing Co., 1997, pp. 46–93.

[3] M. Maheswaran, S. Ali, H. J. Siegel, D. A. Hensgen, and R. F. Freund, "Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems," in *8th Heterogeneous Computing Workshop (HCW '99)*. IEEE Computer Society, Apr. 1999, pp. 30–44.

[4] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, and D. H. J. Epema, "The grid workloads archive," *Future Generation Comp. Syst.*, vol. 24, no. 7, pp. 672–686, 2008.

[5] D. G. Feitelson, "Locality of sampling and diversity in parallel system workloads," in *21th Annual International Conference on Supercomputing*. ACM, 2007, pp. 53–63.

[6] A. Iosup, C. Dumitrescu, D. H. J. Epema, H. Li, and L. Wolters, "How are real grids used? the analysis of four grid traces and its implications," in *GRID*. IEEE, 2006.

[7] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "A performance analysis of ec2 cloud computing services for scientific computing," in *Cloud Computing: 1st Intl Conf. on Cloud Computing (CloudComp 2009)*, ser. Lecture Notes (LNICST), vol. 34. Springer, 2010, pp. 115–131.

[8] J. Dejun, G. Pierre, and C. Chi, "EC2 performance analysis for resource provisioning of Service-Oriented applications," in *3rd Workshop on Non-Functional Properties and SLA Management in Service-Oriented Computing*, Nov. 2009.

[9] E. Deelman, G. Singh, M. Livny, G. B. Berriman, and J. Good, "The cost of doing science on the cloud: the montage example." in *Proceedings of the ACM/IEEE Conference on High Performance Computing, SC 2008*. IEEE/ACM, 2008, p. 50.

[10] M. de Assunção, A. di Costanzo, and R. Buyya, "A cost-benefit analysis of using cloud computing to extend the capacity of clusters," *Cluster Computing*, vol. 13, 2010.

[11] J. N. Silva, L. Veiga, and P. Ferreira, "Heuristic for resources allocation on utility computing infrastructures," in *6th International Workshop on Middleware for Grid Computing (MGC 2008)*. ACM, Dec. 2008.

[12] A. Oprescu and T. Kielmann, "Bag-of-Tasks scheduling under budget constraints," *Proc. 2nd IEEE International Conference on cloud Computing Technology and Science (CloudCom 2010)*, Nov. 2010.

[13] D. Villegas, A. Antoniou, S. Sadjadi, and A. Iosup, "An analysis of provisioning and allocation policies for Infrastructure-as-a-Service clouds," *Cloud and Grid Computing (CCGrid)*, May 2012.

[14] P. Marshall, K. Keahey, and T. Freeman, "Elastic site: Using clouds to elastically extend site resources," in *Cloud and Grid Computing (CCGrid 2010)*. IEEE, 2010.

[15] H. Casanova, A. Legrand, and M. Quinson, "SimGrid: a Generic Framework for Large-Scale Distributed Experiments," in *10th IEEE International Conference on Computer Modeling and Simulation*, Mar. 2008.