Cost-wait Trade-offs in Client-side Resource Provisioning with Elastic Clouds

Stéphane Genaud, Julien Gossa

Cost-wait Trade-offs in Client-side Resource Provisioning with Elastic Clouds

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

***Abstract*—Recent Infrastructure-as-a-Service offers, 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. This drastically changes the problem of resource provisioning and job scheduling. This article presents how billing models can be exploited by provisioning strategies to find a trade-off between fast/expensive computations and slow/cheap ones for indepedent sequential jobs. We study a dozen strategies based on classic heuristics for online scheduling and bin-packing problems, with the double objective of minimizing the wait time (and hence the completion time) of jobs and the monetary cost of the rented resources. We simulate these strategies on real grid workloads in two cases. First, we use the workloads as a whole, which is representative of a large community of users sharing some common resources. Second, we use the workloads extracted for each individual user. These lighter workloads correspond to users submitting work independently from others and paying for their own resources. Our findings show that on large workloads, a little budget increase allows to achieve optimal wait time, while trade-off heuristics may be largely beneficial for individual users with lighter workloads.**

***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's EC2 is one of the most well-know solution to provide a utility computing model. Their solution has contributed to popularizing the Infrastructure-as-a-Service (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 in finding how many VMs are needed, for how long, and the type of hardware configuration fitting the job needs. In this work, we address the problem of resource provisioning on the **client side** in IaaS clouds. 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.

Clients usually perform the resource provisioning process manually, i.e they decide all by themselves when and what resources should be claimed and released. This might lead to naïve and suboptimal solutions, especially when the number of jobs increases and several resources are proposed at different prices and performance capabilities. A good decision is even harder to make when the billing model is complex and several groups of users are submitting concurrent jobs to the same shared infrastructure. Therefore, a brokering system must be interposed between the clients and the provider of the cloud.

There are two contradictory metrics involved in the evaluation of the choices for a client: the monetary cost of the resources and the completion time of the execution. Generally, meeting short completion time constraints implies to provision powerful and numerous resources, and thus pay a higher price. On the contrary, a low budget implies more tolerance regarding the completion time. When using a cloud infrastructure this choice is typically an *online* problem, i.e. at the time the job is submitted, it must be decided immediately whether extra resources need to be provisioned or already running instances are sufficient.

Note that we focus here on a specific billing model, based on the main Amazon's pricing model called *on-demand*, in which users pay by the hour for active VMs. Although per-hour billing is widely spread, some companies propose different billing time units (BTU) or alternative pricing models. For instance, in the *reserved* model at Amazon, a resource is rented for 1-3 years for a fixed amount of money and in turn has a lower hourly price when used. A third pricing model called *spot* is based on bidding and requires different techniques to optimize its usage, as studied in [15]. The *on-demand* pricing presents two interesting characteristics. First, the cost of deployment is linear only in the BTUs spent, independently of the number of VMs started (modulo the deployment and shutdown overheads). Hence, one VM instance running for two hours costs the same as two instances for one hour. In other words: parallelization is free. Second, BTUs are relatively coarse grain (e.g 1 hour) and each started unit of time is due. As a result, any job whose runtime does not last exactly this time unit leaves idle time on the deployed instances, which can be recycled to run

other jobs. Our main investigation in this paper concerns how this remaining time can be reused. Indeed, one may decide to reuse it by delaying the execution of submitted jobs. It enables to lower the cost but involves waiting time for jobs and thus increases the overall completion time. Inversely, one can decide to ignore this reusable time and avoid any wait time by launching new VMs at the cost of extra billed idle time. However, there are intermediate *scenarii*, where the global price can be reduced while preserving an acceptable wait time. This implies to define strategies where certain jobs are delayed when it is worth, and new instances are deployed when it does not increase the price of the solution.

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 strategies, which are evaluated in Section IV. 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. Problem Statement

A. Assumptions

First, we assume that jobs are independent 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 tasks is not possible and jobs can not be suspended to run another one. Third, we assume an online scheduling system: tasks must be dispatched to a resource as soon as they are submitted. 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.

B. Problem

Let us introduce the problem on the example of Figure 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 labeled 1VM4All. In this case, enqueuing the jobs for execution on a single VM leads to an optimal cost since only the time between the end of J_4 and the end of the second BTU is wasted. An alternative is to reserve simultaneously several resources in order to execute some jobs in parallel, as exemplified on the other rows on Figure 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.

The problem of minimizing the cost is related to the *variable-sized bin packing with fixed cost* [5]. In this problem, we are given a collection of bin sizes and an infinite supply of bins of each size. The objective is to pack a set of items of given sizes by choosing appropriate bins so as to minimize the total size of the packing. This a generalization of the classical bin packing problem where all bins have a size S and every object has a size $s_i < S$. Both problems are NP-hard and many approximation algorithms have been proposed [1]. In our context, items are jobs and bins are BTUs of VMs. However, the solutions proposed for this problem cannot be used directly. First, many of the approximation algorithms proposed do not apply because of our *online* constraint. For example, the *FitDecrease* series of algorithms requires to sort the set of items by size. Second, and most important, the classical bin-packing problem does not take into account the dynamics of the resource provisioning problem. Contrary to a bin, which has a same occupation until new items are packed in it, a VM is, by analogy, continuously filled (or billed) from the time it is started until it is released, whether we assign jobs to it or not. Moreover, as explained in the introduction, our problem has two objectives. Our timing considerations must not be only relative to the number of BTUs spent but also to the completion time of the jobs. Finally, we will resort to adapted versions of the classic bin packing heuristics in the online case to minimize the cost of provisioning.

III. Provisioning Strategies

A. 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)
- $c(x)$: 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.

B. 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 in 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 parameterized frequency. This release procedure is common to all

strategies. It consists in 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

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.

Figure 1 shows an example use case. One 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.

C. Strategies

We propose four sets of strategies, illustrated on Figure 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, as idle time is reused at the maximum.
- ***1VMperJob*-based strategies**: On the opposite side of the spectrum, we devise three "expensive" strategies.

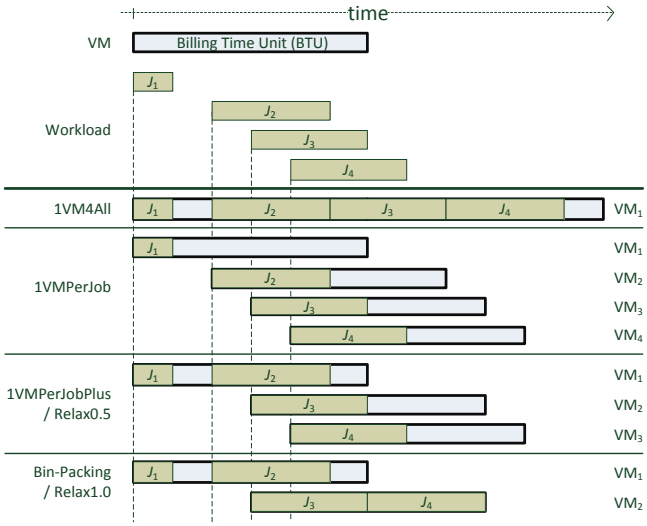

Figure 1. Illustration of the provisioning strategies

1VMperJob is a reference strategy for the lowest waiting time possible: it deploys a new VM for each new job request whatever the state of the other active VMs. As shown in the example with J_2, an obvious improvement is to deploy a new VM only if no active VM is currently idle which can be immediately reused at no cost. *1VMperJobPlus* implements this by returning the first idle VM found. *1VMperJobBest* and *1VMperJobWorst* return the VM that will shutdown the latest and soonest respectively, in order to maximize and minimize the remaining reusable idle time.

- ***Bin-Packing*-based strategies**: We have implemented three classic heuristics for the online bin-packing problem [1], namely *FirstFit*, *BestFit* and *WorstFit*. In our context, *FirstFit* scans the list of already deployed VMs and maps the job to the first VM that does not require to prolongate the rent time over a new BTU, i.e we map the job for no extra-cost. If no such already started VM exits, a new VM is deployed. On the example, a new VM is deployed to map the job J_3 because it can not be handled at constant cost otherwise. On the contrary, the execution of J_4 is delayed in order to reuse the idle time of this new VM.
  BestFit and *WorstFit* are identical to *FirstFit* except that they map the job to the VM that leaves the shortest and longest idle time respectively on the VM. Again, the objective is to respectively maximize and minimize the remaining reusable idle time.
  The above strategies have the objective to minimize the number of BTUs. Hence, it tends to minimize the global cost with little consideration to the completion time, which is absent from the original Bin-Packing problem. *EarliestFit* is a first approach to include this

strategy	$eligible(v, j)$ returns $true$	$optimum(C)$ returns $v \in C$ such that ...	comment
1VM4All	always	$v = v_0$	Slowest/Cheapest
1VMperJob	never	any	
1VMperJobPlus		any	Fastest/Most expensive
1VMperJobBest	if $q_v = \emptyset$	s_v is maximum	
1VMperJobWorst		s_v is minimum	
FirstFit		any	Regular BinPacking strategies
BestFit		$i_v - s_v$ is maximum	
WorstFit	if $c(s_v - b_v) = c(s_v - b_v + r_j)$	$i_v - s_v$ is minimum	
EarliestFit		i_v is minimum	+ wait time optimization
RelaxFirstFit x		any	Price optimization
RelaxEarliestFit x	if $c(s_v - b_v) = c(s_v - b_v + r_j)$	i_v is minimum	+ max wait time constraint
RelaxLastestFit x	and $(i_v - t) < (x \times r_t)$	i_v is maximum	

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

trace	#jobs	#CPUs	#user/#grp	arrival	runtime
LCG	188 041	24 115	216/28	5	8 970
AuverGrid	347 611	475	405/9	78	25 186
NorduGrid	78 1370	2 000	387/107	127	89 273
SharcNet	1 195 242	6 828	412/1	28	31 964

Table II
WORKLOAD TRACES

criteria. It is a *Fit* heuristic selecting the VM which minimizes the waiting time of the job.

- ***Relax*-based strategies**: *RelaxFirstFit* x, *RelaxEarliestFit* x and *RelaxLastestFit* x include a bound on the waiting time, which is expressed as a factor x. A new VM is deployed when no active VM can handle the job at constant cost or when the waiting time exceeds x times the runtime of the job. As shown on the example (Relax 0.5), a low value of x leads to a *1VMperJobPlus*-like behavior. Here, another VM is started for J_4 since using an already deployed VM would either increase the cost or make the execution end after the deadline. On the contrary, a high value of x leads to a Bin-Packing-like behavior, as the same delay is considered acceptable.
  The three heuristics only differ in the VM they select. They respectively select the first VM found, the VM that will be idle first and last.

IV. EVALUATION

To assess the performances of the different strategies, we used four datasets from real production grids publicly available from the Grid Workload Archive [6]. The main characteristics of the selected datasets are presented in table II: the total numbers of jobs, the number of distinct CPUs used, the number of users and groups, the average interarrival time between jobs, and the average runtime of 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 CAS, chemistry, graphics, biomed, and HEP. The traces used here contain the grid jobs for tree 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 [4]. The common data we can find in these traces are, for each job, its id, the submission time, the run time, the user id and the group id.

All strategies have been implemented in our own simulator which implements the algorithms described previously. The simulation uses the economic model of Amazon's EC2. The results of the simulations of the different provisioning strategies are plotted in Figure 2 and 3 along two performance metrics, the *job average wait time*, defined as the total wait time divided by the total number of jobs, and *total cost of the resources*, defined as the sum of the cost of each provisioned resource.

A. Results on complete traces

First, we make the assumption that each of the four workloads represents the requests from a whole group of users sent to the same submission system in order to mutualize cloud resources. We evaluate the strategies in this context where all requests are concurrent in the access to the resources. The simulations for these complete workloads are presented in Figure 2.

Very similar results are shown by the four workloads

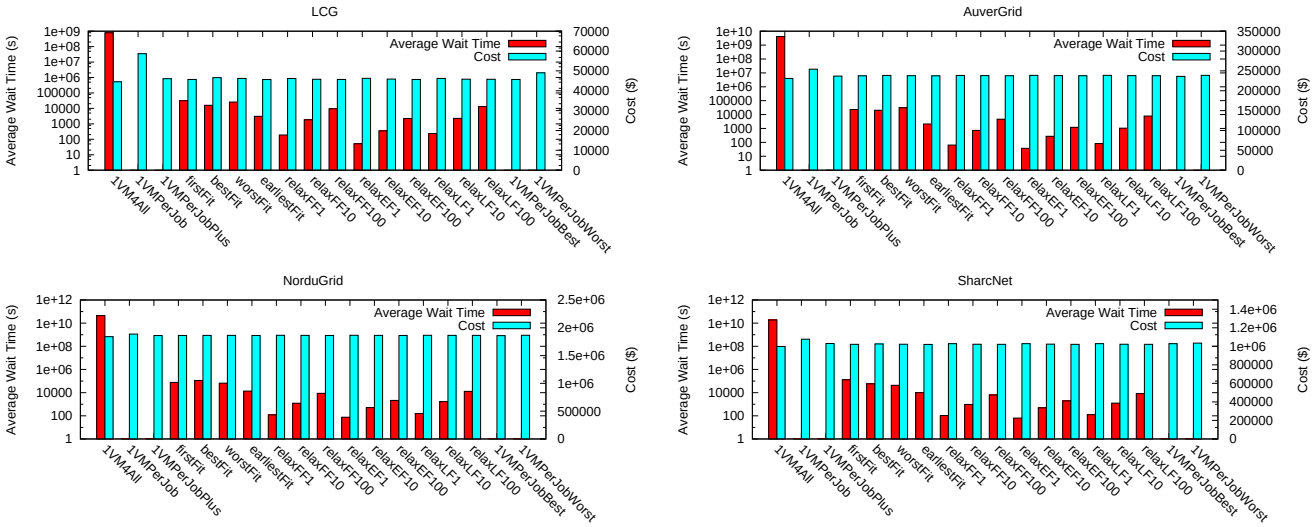

Figure 2. Average Wait Time (s) vs. Cost ($), for each strategy, on complete traces

LCG, AuverGrid, NorduGrid and Sharcnet. If we compare the strategy *1VM4All* designed to minimize the cost, and the set of *1VMperJob*-based strategies designed to minimize the waiting time, we see that the average waiting times differ by about 9 to 10 orders of magnitude, while the cost is more or less the same, especially for enhanced versions. *1VMperJob*-based strategies perform very well. For a small increase of price compared to *1VM4All* the average wait time is reduced to zero. For instance, *1VMperJobPlus* increases the price by only 3.5%, 2.5%, 1.1%, and 3.2% respectively on the four workloads. Among *1VMperJob*-based strategies, *1VMperJobBest* performs slightly better (respectively price increase by 2.7%, 2.2%, 1.0%, and 3.0%).

In other words, we learn from this simulation that there is little room to find a trade-off, as the cost can only be lowered by a few percents compared to the fastest/most expensive strategy. In consequence, "smartest" strategies are of no interest, as it is not possible to perform better than *1VMperJob*-based strategies in term of average wait time, while possible price improvements are negligible.

B. Results on individual user traces

Next, we evaluate the strategies on the workloads extracted for each individual user of LCG. Each simulation hence corresponds to the case where a user submits his requests through his own scheduler.

We found that users can be differentiated depending on the margin between the costs resulting from application of the cheapest and the most expensive strategies. This margin represents the room available to find a trade-off. Different margins lead to different winning strategies. To classify the LCG users we looked at this cost margin, defined as the ratio of the cost of *1VMperJobPlus* to the cost of *1VM4All*. A ratio of 1 means that both strategies produce a solution of equal price, i.e. there is no room for a trade-off. We have plotted in Figure 4 a cumulative distribution of the cost margins. For a sake of illustration, we arbitrarily separate this distribution in four parts, corresponding to different situations. For each situation, we have chosen a representative user workload. The results for these users are shown in Figure 3.

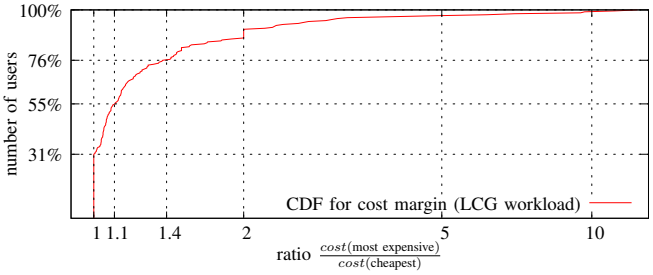

Figure 4. Cumulative distribution of cost margins on user traces

- 31% of the user workloads allow no cost margin (ratio=1). The job requests of user 96 in the LCG trace is an example of this situation. In that case, there is no cost difference between the cheapest *1VM4All* and the fastest *1VMperJobPlus* strategies. Moreover, there is no difference either in term of wait time for most of the users. Actually, this occurs with peculiar workloads having no concurrent jobs, or job runtimes greater than the BTU. This implies that all strategies perform the same. Some rare exceptions show an increase of wait time for *1VM4All* or cost for *1VMperJob*. In consequence, *1VMperJobPlus* should be used for this type of workload.
- 24% of the workloads allow a cost margin of up to 10% (ratio=1.1), like for instance LCG's user 155. In that case, the result is very close to our conclusion on

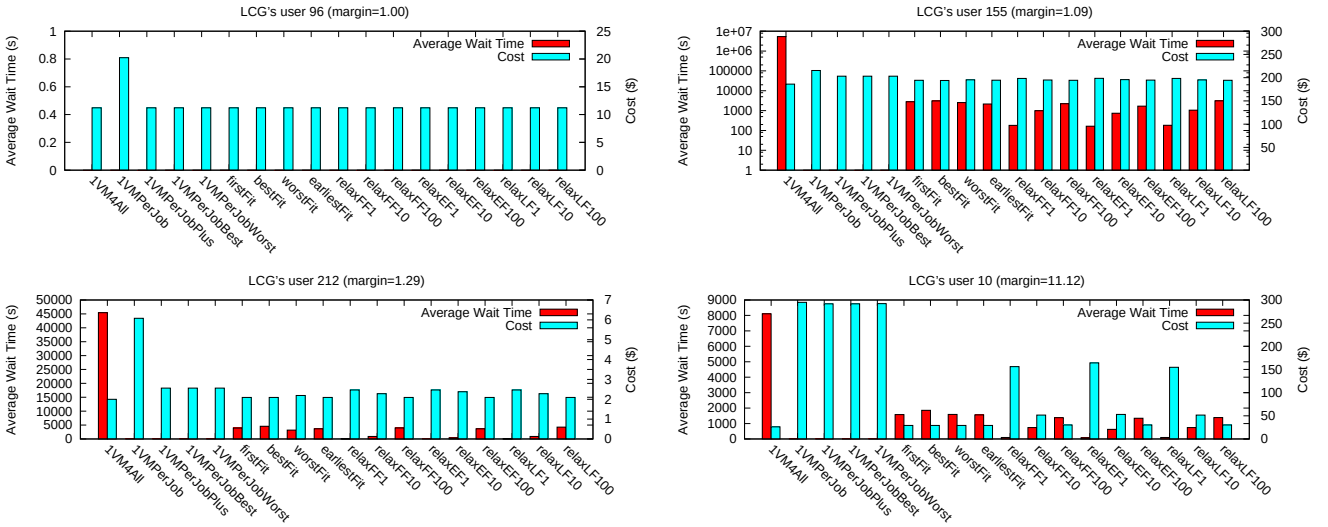

Figure 3. Average Wait Time (s) vs. Cost ($), for each strategy, on user's traces

complete traces. For less than 10% of cost increase, the wait time can be reduced to 0 with the *1VMperJobPlus* strategy while other strategies are of no use as they imply great increase of wait time for a small cost spare.

- 21% of the workloads allow a cost margin of 10 to 40%, like for instance LCG's user 212. In that case, there is room for a trade-off and Bin-Packing based strategies are useful, as they allow to match the price of the *1VM4All* strategy, while drastically reducing the wait time. Small differences appear between them: *BestFit* costs slightly less than *WorstFit* while having a slight increase of wait time, and *FirstFit* is in-between. But the most interesting strategies are the Relax-based ones. Indeed, they allow to control the trade-off thanks to their tolerance factor x. When x is low, these strategies behave like *1VMperJobPlus*, while they behave like the Bin-Packing based ones with high x. Moreover, moderating x allows to find a trade-off in-between. Unfortunately, this moderation is not easy as we have not been able to find any pattern of its impact among users workload. Similarly, *RelaxFirstFit* x, *RelaxLastestFit* x and *RelaxEarliestFit* x perform slightly differently, but no pattern have been discovered.
- Finally, 24% of the workloads allow a cost margin higher than 40%, like for instance LCG's user 10. The same observations apply as previously to these users, except that Bin-Packing based and Relax-based strategies are even more interesting and should be preferred to *1VMperJobPlus*.

C. Conclusions

The strategies dynamically provision and release VMs, and we call the number of active VMs at a given instant the *diameter* of the provisioning. There is a direct relationship

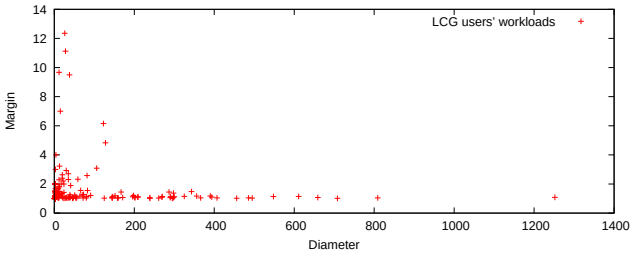

Figure 5. Cost margins vs. Provisioning diameter on user traces

between the diameter and the cost margin. On Figure 5 is plotted the maximum diameters for each user's workload on LCG. It can be observed that the cost margin is limited when the provisioning diameter is high. Indeed, no margin exceeds 1.5 when the diameter exceeds 170. This is because large diameters implies the multiplication of the opportunities to reuse idle VMs, and thus very good efficiency of *1VMperJob*-based strategies.

This explains why there is no room for "smarter" strategies with complete platform workloads, where the diameter is naturally high. Hence, when the number of concurrent jobs is high, the best provisioning strategy is *1VMperJobBest*. However, when this number is lower, *Bin-Packing*-based strategies are efficient as they allow to find a trade-off, and even to match the lowest cost while greatly improving the performances. The differences between each version of one same strategy (i.e. *First*/*Best*/*Early*) are not clear. Except for specific needs, *First* version can be chosen because of its lowest complexity. Nevertheless, *Relax*-based strategies are very interesting as they allow to control the trade-off between the *1VMperJob*-based and the *Bin-Packing*-based strategies, and should be used in most cases.

V. Discussion

We have shown that provisioning strategies can be designed regarding the billing models of cloud platforms, leading to different solutions in term of performance and price. 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 no impact the relevance of our study, as they are negligible face to the BTU. For example, the time to instantiate and start a VM instance at Amazon's EC2 is 60 to 130 seconds according to a recent study [10], while the BTU is 3600 seconds. However, such details have to be taken into account by a real brokering system, and should be handled carefully to optimize the solutions. For instance, if the startup time is greater than the time for one of the active VMs to become idle, then it is better to assign a job to this VM than to deploy a new one. This implies to modify the *eligible* condition in order to extend the set of candidates to this kind of VMs. Stop time must be handled carefully as well in order to avoid releases just after the BTU, leading to double cost with no performance improvement.

Next, we have considered the same runtimes on the cloud platform as in the workload traces because these last does not include information about the used CPU. However this does not change the conclusions of this study, but only implies that the presented prices and waiting time are not realistic.

Second, our work concerns only the provisioning problem, apart from the job scheduling problem. Indeed, the jobs are considered executed in a FIFO manner accordingly to our assumption of online scheduling. However, the same semi-online scheduling as those of batch schedulers such as conservative [8], aggressive [14] or selective back-filling [13] can be applied to the VM queues. This could improve the overall user experience. Such scheduling might have an impact on the provisioning strategy, for instance they might need a constant over-provisioning. Moreover, the presented provisioning strategies are designed for billing models based on coarse units of time and linear billing.

Finally, we only considered computation time and on-demand basic instances. Our model and strategies could be enhanced to take storage and data transfer costs into account, as well as different types of instance. Indeed, an efficient cloud-resources brokering system should be able to decide the characteristics of the instances to deploy, as well as to choose among the different billing models. Indeed, most of the cloud offers propose several types of instances, different in term of CPU speed, memory and allowed I/O. EC2's billing model is interesting as the cost of one simple-speed instance is the half of the cost of one double-speed instance. Consequently, just as parallelization is free, speed is free too. Thus, our approach can be adapted to find a trade-off between slow/cheap instances fully used and fast/expensive instances leaving more billed idle times.

Ultimately, our approach must be able to handle the decision between different cloud offers, by comparing the price and performances they can provide.

VI. Related Work

We have concentrated in this paper on the decisions a user can make in choosing a satisfactory trade-off between cost and speed to process a set of independent jobs. The need to choose between several *scenarii* is perfectly exemplified by Deelman et al. [3]. In this experimental study, users of a scientific application evaluate the benefits and costs of supplementing their in-house computers with cloud resources, taking into account computations and data transfer costs. From a more system oriented point of view, Marshall et al. [9] propose practical provisioning strategies implemented upon the Nimbus toolkit [7] in order to provide users with a responsive system. These strategies are designed empirically assuming simple job arrival patterns (one at a time, a stream of requests and a burst of requests). A more general study of scheduling strategies by de Assunção et al. [2], based on simulation from real traces, investigates how classic local resource management systems would perform when simultaneously scheduling jobs onto in-house clusters and/or cloud resources. The authors devise several strategies to mix this two types of resources and evaluate their performances depending on the heuristics used for scheduling. This work is the closest to ours, given three main differences: they assume mixed types of resources while we consider cloud-only resources; they test only classical scheduling strategies of local management resource systems (backfilling strategies); Although their objective is to a minimize the cost, the used strategies have no direct impact on the cost. On the contrary, the bin-packing heuristics we have investigated try to minimize the number of time units used.

By contrast, many works address the problem of VMs provisioning on the server side. The objective is to maximize the platform utilization. Some are worth mentioning because they try to improve the users' satisfaction by better serving their requests. Even though the aim is to improve *globally* the satisfaction, they employ techniques which could be transposed to a client-side approach. Perez et al. [11] propose a learning scheme to prioritize the scheduling in presence of mixed workloads (interactive and best-effort jobs), with a bi-objective optimization problem: fairness among users and responsiveness of job requests. A similar goal is targeted by Quiroz et al. in [12], which propose an online clustering approach to detect patterns in the stream of requests. The clustering process periodically outputs a set of VM classes, each related to the required resource configurations and numbers of VMs. For each class, the correlations between the user requirements and the real

executions are continuously computed to provide a feed-back to the provisioning process, which can adjust the resources to match users requirements.

VII. Conclusion and Future Work

In this paper, we have shown that the same workloads can lead to different performances and price according to the provisioning strategy with regards to the pay-per-use model. We have proposed and evaluated a dozen strategies based on classic scheduling and bin-packing algorithms. Experiments on real traces show that when the workload is massive, fast and expensive strategies like *1VMperJob* are very efficient because of the small price difference compared to the cheapest strategy, while offering the best performances. Moreover we have proposed enhanced alternatives to this strategy, presenting even better results. However, when the workload is lighter or the demand is specific, bin-packing based strategies are interesting. Especially, relax-based ones allow to constraint the waiting time according to the runtime of the job. They allow the user to tune the trade-off between performances and prices according to his specific needs.

The future work is twofold. First, we will investigate how extra scheduling strategies applied to queued waiting jobs can improve the current results, as discussed in Section V. We should also extend the work to consider the choice between different billing models and different types of resources. Second, we will work at giving the user a constant feed-back about the strategies. From a pragmatic point of view, the system should be able to tell the performance improvement per invested dollar for each candidate solution. We will also study how long-term workloads can be analyzed to provide the user with different possible budgets at month or year scale. Two issues might be investigated with this goal. First, the characterization of jobs submission patterns and their relationships with the strategies efficiency; Second, the use of our simulator to assess strategies efficiency without characterizing the job submission pattern. As a bottom line, our main objective is to design a user-side comprehensive and usable brokering system for cloud resources, taking the billing model at the core.

References

[1] E. G. Coffman, M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: a survey. pages 46–93, 1997.

[2] Marcos de Assunção, Alexandre di Costanzo, and Rajkumar Buyya. A cost-benefit analysis of using cloud computing to extend the capacity of clusters. *Cluster Computing*, 13:335–347, 2010. 10.1007/s10586-010-0131-x.

[3] Ewa Deelman, Gurmeet Singh, Miron Livny, G. Bruce Berriman, and John 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*, page 50. IEEE/ACM, 2008.

[4] Dror G. Feitelson. Locality of sampling and diversity in parallel system workloads. In *Proceedings of the 21th Annual International Conference on Supercomputing, ICS 2007*, pages 53–63. ACM, 2007.

[5] Donald K. Friesen and Michael A. Langston. Variable sized bin packing. *SIAM Journal on Computing*, 15(1):222–230, 1986.

[6] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, and D.H.J. Epema. The grid workloads archive. *Future Generation Computer Systems*, 24(7):672–686, 2008.

[7] Katarzyna Keahey, Ian T. Foster, Timothy Freeman, and Xuehai Zhang. Virtual workspaces: Achieving quality of service and quality of life in the grid. *Scientific Programming*, 13(4):265–275, 2005.

[8] David A. Lifka. The ANL/IBM SP scheduling system. In *Job Scheduling Strategies for Parallel Processing*, pages 295––303. Springer-Verlag, 1995.

[9] Paul Marshall, Kate Keahey, and Timothy Freeman. Elastic site: Using clouds to elastically extend site resources. In *10th IEEE/ACM Intnl Conf. on Cluster, Cloud and Grid Computing (CCGrid 2010)*, pages 43–52. IEEE, 2010.

[10] Simon Ostermann, Alexandru Iosup, Nezih Yigitbasi, Radu Prodan, Thomas Fahringer, and Dick Epema. A performance analysis of ec2 cloud computing services for scientific computing. In *Cloud Computing: 1st Intl Conf. on Cloud Computing (CloudComp 2009)*, volume 34 of *Lecture Notes (LNICST)*, pages 115–131. Springer, 2010.

[11] Julien Perez, Cecile Germain Renaud, Balázs Kégl, and Charles Loomis. Multi-objective reinforcement learning for responsive grids. *Journal of Grid Computing*, 8(3):473–492, 2010.

[12] A. Quiroz, H. Kim, M. Parashar, N. Gnanasambandam, and N. Sharma. Towards autonomic workload provisioning for enterprise grids and clouds. In *Proceedings of the 10 th IEEE/ACM International Conference on Grid Computing (Grid 2009)*, pages 50 – 57, 2009.

[13] Srividya Srinivasan, Rajkumar Kettimuthu, Vijay Subramani, and Ponnuswamy Sadayappan. Selective reservation strategies for backfill job scheduling. In *Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '02)*, volume 2537 of *LNCS*, pages 55–71. Springer, 2002.

[14] Ahuva Mu'alem Weil and Dror G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Transactions on Parallel and Distributed Systems*, 12(6):529–543, 2002.

[15] Sangho Yi, Derrick Kondo, and Artur Andrzejak. Reducing costs of spot instances via checkpointing in the amazon elastic compute cloud. In *3rd Intl Conf. on Cloud Computing (CLOUD'2010)*, pages 236–243. IEEE, July 2010.