Simulation of Techniques to Improve the Utilization of Cloud Elasticity in Workload-aware Adaptive Software

Diego Perez-Palacin
Dip. di Elettronica, Informazione e Bioingegneria
Politecnico di Milano, Italy
diego.perez@polimi.it

Raffaela Mirandola
Dip. di Elettronica, Informazione e Bioingegneria
Politecnico di Milano, Italy
raffaela.mirandola@polimi.it

Marco Scoppetta
Dip. di Elettronica, Informazione e Bioingegneria
Politecnico di Milano, Italy
marco.scoppetta@mail.polimi.it

ABSTRACT

More and more software owners consider moving their IT infrastructure to the cloud. At present, cloud providers offer easy manners to deploy software artifacts. Therefore, the profile of cloud clients is no longer limited to computing experts. However, an appropriate configuration of the elasticity offered by cloud computing is still complicated. To help these clients, this work presents a simulator of the behavior of software services that run on the cloud and use the cloud elasticity for adapting their infrastructure in order to accommodate their workload in each moment. This work identifies techniques that are used to help mitigating at runtime the lack of predictability of workload changes. The presented simulator implements the identified techniques and allows users to execute scenarios where a combination of these techniques is enabled.

Keywords

Performance, Simulation, Workload, Elasticity

1. INTRODUCTION

Many organizations are daily moving their computing infrastructure to the cloud, or at least studying this option as a plausible alternative. This is true not only for companies that provide software services as main activity but also for generic organizations that prefer using cloud infrastructures to host the internal IT services supporting their main business.

In this context, clients of cloud services are no longer computing experts that can take informed decisions about the configuration of the infrastructure. Now, cloud clients hold different types of roles with less expertise in computing. However, for the deployment on the cloud of a cost-efficient application that also runs clients' software with good performance, it is necessary a deep knowledge on cloud elasticity behavior and on system analysis field. Unfortunately, a correct utilization of the elasticity offered by cloud providers that allows clients to use only the necessary resources in each moment is among the most intricate concepts when deploying a software application on the cloud. Therefore, in order to make cloud computing appealing to this new mass of clients, an accurate decision support should be offered.

ICPE'16 Companion, March 12-18, 2016, Delft, Netherlands

DOI: http://dx.doi.org/10.1145/2859889.2859897

In this direction, related works have directed an effort towards the development of tools that simulate the entire layered cloud architecture [3], tools assessing the cost of running applications under concrete elasticity plans [8, 12, 2] and tools that, using models of the application and environment and applying formal analysis techniques, can identify the theoretical optimal software configuration for the given properties of its own components and characteristics of the changing environment [11, 10, 4, 6, 5, 7]. However, the application of these theoretical optimal configurations may not always succeed in the cost- and quality-effective management of the application because, at runtime, when the application faces the real execution environment and cloud properties, some phenomena which were not considered in the analysis may occur. For instance, a gradient never seen before in the increment of application load when a burst of service request arrives.

This work identifies and accounts some of these phenomena that hinder scaling engines from an appropriate management of the application elasticity. It also identifies techniques whose application improves the usage of elasticity and the quality of the application or the cost of running it on the cloud, for instance the well-known hysteresis technique.

We experiment the usage of these techniques through a simulation tool called PERFSCALE[1] which is intended to help cloud clients to unveil expected availability and performance properties of their application. PERFSCALE allows simulating the behavior of an elastic application executing in the cloud, under the identified phenomena and variable workload. PERFSCALE also simulates the execution of the application when any combination of the identified techniques improving the behavior of scaling engines is applied. PERFSCALE requires as input: the expected execution time of requests, a workload trace with the application requests, which is usually easy to obtain from application logs, and an initial scaling plan that contains the optimal workload thresholds for which the scaling engine should modify the application infrastructure by activating or deactivating resources. These threshold values are represented in the output format of SCOAP tool [10], which is a tool that can compute such optimal threshold based on the tradeoff between cost and performance requirements.

The rest of the work is organized as follows: Section 2 presents the identified phenomena that may prevent the appropriate execution of scaling engines. Section 3 presents the identified techniques to be included in the synthesis of a scaling plan. Section 4 describes the experiments carried out with PERFSCALE, the results obtained and the type of information that PERFSCALE can offer to application owners to estimate the effect of moving their software to cloud computing infrastructures. Section 5 concludes the work.

[1]Detailed description of the simulation tool is not reported here for space reasons.

2. INFRASTRUCTURE AND WORKLOAD PROPERTIES

This section discusses factors that have an impact on the goodness of a scaling plan when executing in a real scenario. There are phenomena at runtime that prevent the system from executing optimally if the scaling plans are only based on the threshold workload values of optimal infrastructure configuration. We separate the identified phenomena into two groups according to: the workload nature and the cloud infrastructure nature.

2.1 Workload nature

A phenomenon that continuously happens to applications offered as publicly accessible service is the variation in its workload. In order to show good behavior, simple scaling plans expect the trend of this workload variation to be easily predictable and smooth. However, these expectations are not usually met in variable workloads. On the contrary, it frequently happens that workloads show the following characteristics.

Fast fluctuation: the number of requests received over the recently measured time intervals fluctuates around a specific value. If this phenomenon is not managed by the scaling plan, the infrastructure controller can continuously acquire and release cloud resources, which can cause that the infrastructure costs soar while the application still shows poor performance since many of the active resources are in their booting phase instead of serving requests.

Burstiness: the application notices a fast grow in the number of legitimate incoming requests. Although scaling plans are intended to avoid congestion even in cases of large differences in the arrival rate of requests, the unexpected speed with which the peak of incoming requests is reached together with the required time to adapt the infrastructure lead the application to showing a degraded performance.

2.2 Infrastructure nature

Every cloud infrastructure and cloud provider shows its own particularities, which entails that a scaling plan that is suitable for a certain type of infrastructure can behave poorly in other types. PERFSCALE considers the following particularities:

Startup time: cloud resources are not operative right after their acquisition but they need a startup time that includes, among others, operative system booting, application load and binding to application components running on other machines. These booting times are not constant among different providers nor for different types of computing resources within the same provider. Startup time variations have been identified [9] depending on: the cloud provider, the operative system of the virtual machine (VM) to start, VM characteristics and VM acquisition method (e.g,. spot [1] or on-demand).

Billing model: each cloud provider follows a different and sometimes complex template to calculate its clients bill for the utilization of its resources. Among the particularities of each template, there is the common property of minimum billing period for the utilization of a VM (e.g., the utilization of a VM is charged in the number of complete hours, rounding up the real resource utilization time to the next full hour). The effect of this property on the pricing model is that scale-in operations shortly followed by some scale-out operations make clients to pay twice for the same computational resource for a given time period.

3. TECHNIQUES FOR PERFORMANCE IMPROVEMENT

We have identified a set of techniques, which are already used in computing areas, to face the effect of the phenomena presented in Section 2 on the system performance. We have adapted these techniques for the IaaS management environment, and we have implemented their behavior in PERFSCALE. The objective is to allow cloud clients to simulate their execution with different parameter values in order to realize the parameterization leading to the best tradeoff between application performance and infrastructure cost. Hereafter, we focus on two of techniques called *hysteresis* and *deactivation decision interval*.

The necessity of some techniques that enhance the behavior of the application with respect to the direct application of a scaling plan is depicted through an example in Figure 1. This figure shows an example of workload trace with 2.5 millions of request to an application. It has been taken from [13]. The arrival rate shows frequent fluctuations around a value of 50 *requests/s*. The darkest line in the figure depicts the arrival rate values that the infrastructure configuration in each moment (i.e., servers running and with their booting period finished) is able to satisfy with appropriate performance when no mitigation technique is used. The frequent fluctuation in the infrastructure configuration is the consequence of a scaling plan that only takes into account the current workload and current configuration to decide to reconfigure the application infrastructure. Furthermore, despite this frequent fluctuation, we can also see in the figure several intervals where the application is under-provisioned; those intervals where the incoming arrival rate (lightest line) line is above the darkest line.

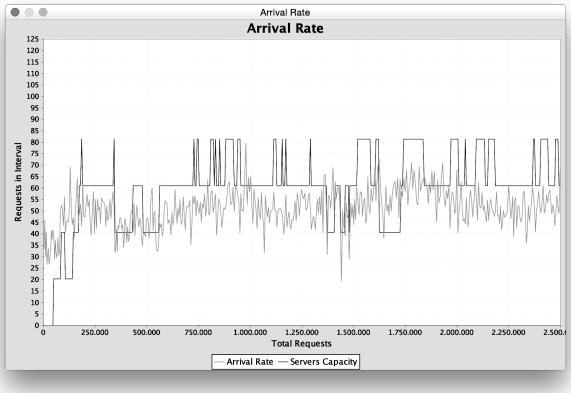

Figure 1: Fast fluctuating workload and arrival rate that can be served by the active servers proposed by a basic scaling plan

The following subsections describe the techniques that PERFSCALE can simulate.

3.1 Hysteresis technique

The hysteresis is a well-known technique in scaling plans. It is based on having different threshold values for executing the infrastructure scale-out or scale-in operations. Its aim is avoiding continuous scaling actions when the workload fluctuates around the optimal threshold value.

In our concrete problem, we change from using the single threshold workload value w_i given in the scaling plan for defining the change between activating or deactivating the i-*th* resource (i.e., activate the i-*th* when the workload is above w_i and deactivate the i-*th* when it decreases below w_i) to use two values. These values are an activation threshold w_i^a that defines the activation of the i-*th* resource and a deactivation threshold w_i^d that defines the deactivation of the i-*th* resource.

The effectiveness of this technique to reduce the switching rate depends on the hysteresis width. Larger values of width produce lower switching rates, at the cost of wasting more resources along time since the system will tend to keep more resources active even when they are not completely necessary to prevent additional activations if they become necessary in the near future.

The optimal width value for a scaling plan depends on the workload pattern and on the arrival rate that each resource can handle while executing with appropriate performance. For this reason, we experiment with tailored values for the hysteresis width that take into account the capacity of a resource to serve requests, rather than with absolute hysteresis width values based on workload. Therefore, in PERFSCALE, the activation threshold is $w_i^a = w_i$ and the deactivation threshold $w_i^d = w_{i-h}$, meaning the latter that the deactivation threshold of the i-*th* resource coincides with the activation threshold of the (i-h)-*th* resource. In this way we avoid experimenting with hysteresis width values that are unrelated to the infrastructure and workload properties.

Cloud clients can hardly calculate optimal hysteresis values in their scaling plans, avoiding both wasting resources (e.g., not too wide hysteresis value) and prematurely turning off resources that will be probably needed again in the very near future (e.g., not too narrow hysteresis value). For helping them in this setting activity, PERFSCALE allows cloud clients to simulate the application expected performance and expected reconfiguration rate of the infrastructure under different hysteresis width values h. After having simulated different scenarios, cloud clients will know what is the best kind of hysteresis for their application and can better set parameters values for the scaling plans of their applications.

3.2 Deactivation decision interval

This technique proposes that the decision regarding the deactivation of a resource is not taken right after the workload decreases under the calculated threshold. By including a "deactivation decision interval", the scaling plan proposes to wait an interval of time in which the workload should continue under the calculated threshold for finally deciding the deactivation of the resource.

This interval improves the application performance in two cases of workload variation. The first case happens when a peak of requests fades out and the arrival rate goes back to a common value. In this situation, almost all resources would be instantaneously turned off. Therefore, when two burst of requests occur close in time requests of the second peak will experience very long response times. An interval for the deactivation decision can avoid this situation if the interval length is larger than the time between bursts.

The second case is caused by the fact of a sudden drop in the number of requests in a certain moment. A very abrupt drop in the number of requests may indicate problems in reaching the application that will be solved in a few moments (e.g., caused by network congestion or downtime of some networking infrastructure) rather than meaning that users suddenly stopped using the application. Therefore, in this situation, the technique of waiting some time for deciding whether to scale-in the application is a good solution.

This technique brings particularly useful improvements when used together with the hysteresis technique. For example, notice that the previous hysteresis technique was unable to mitigate the effect of the illustrated sudden and short-lived drop in the application requests. Working together, once the deactivation threshold has been crossed, the resources will not be switched off immediately but instead a deactivation interval will be waited before deciding. If the arrival rate recovers over the threshold value within the waiting interval, resources will not be switched off and then on again.

4. EXPERIMENTATION

This section presents our experimentation with PERFSCALE. It shows the results that can be obtained by using the simulator, its usefulness to evaluate applications that are affected by external factors such as the quantity of time that VMs need to boot, and how the identified techniques for performance improvement affect the behavior of the scaling plan and performance of the application.

4.1 Environment of the experiments

We have used for experimentation a simple application that offers a stateless service which requires in average 50ms to execute. The queue of requests waiting for being served can contain a maximum of 1000 requests. For the expected workload, we have used a trace from the World Cup 1998 [13] which contains more than 140 million of requests and covers 34.7 continuous days. This trace, despite its age, represents well the variability in the arrival rate regarding the fast fluctuation and burstiness phenomena and it is still one of the finest grain publicly available since its timestamps for requests have the granularity of a second. Therefore, by counting the requests with the same timestamp, we can arrive to an arrival rate variability that changes every second.

The scaling plan has been created by SCOAP tool [10] using the same workload trace for the plan synthesis. The generated plan provides arrival rate thresholds up to 24 different configurations. The first six thresholds for activating VMs are the following:

$$w_1 = 0,\ w_2 = 15.15,\ w_3 = 30.4,\ w_4 = 44.37,$$

$$w_5 = 57.9,\ w_6 = 73.4$$

meaning that the first VM is always active (its activation threshold is 0); between 15.15 and 30.4 requests/s the optimal configuration is the one that uses two VMs; between 30.4 and 44.37 requests/s the optimal configuration is the one that uses three VMs; between 44.37 and 57.9 the optimal configuration is the one that uses four VMs; and so on.

4.2 Experiments setup

We have executed several experiments for each different performance improvement technique by testing them with different parameters, in different combinations and in different scenarios of the environment parameters (e.g., for different expected startup times of VMs). Since the application performance estimation of PERFSCALE is based on discrete event simulator following probability distributions, each of these experiments has been run 25 times. We report the average value of the simulations in the centered boldface result in next tables cells together with its minimum and maximum values in the smaller font values below the average. Table 1 shows this format.

4.3 Base case experiments

We first experimented the behavior of the scaling plan and performance of the application when the scaling plan is directly executed without applying any of the presented performance improvement techniques. Table 2 provides the results.

We have performed six experiments, each of them consisting on 25 runs, by variating the *Startup Time* parameter from 60 to 900 seconds in order to ensure a realistic set of startup times. We used this range of times based on work [9], which performs an analysis of startup times in different cloud computing providers and reported measured data of startup times from 44 to more than 800 seconds.

Results in Table 2 show how the characteristics of scaling plan improve when the startup time increases (less cost and less reconfigurations) while the quality of the application deteriorates (less

Average value
MiniumValue
MaximumValue

Table 1: Example of experiment results in a generic cell

Startup Time	*Average time in queue*	*Peak time in queue*	*Avai-lability*	*Cost*	*Num of Reconfs*
60 s	**0,4752 s**	**50,87 s**	**99,865%**	**16317,15 $**	**24735**
	0,0005 s	9,98 s	99,384%	16291,53 $	24614
	1,8149 s	105,28 s	99,998%	16395,68 $	24851
120 s	**0,6420 s**	**48,73 s**	**99,726%**	**12217,61 $**	**18128**
	0,0064 s	14,16 s	98,935%	12101,27 $	18006
	2,2859 s	99,74 s	99,988%	12319,08 $	18251
360 s	**0,7671 s**	**61,94 s**	**99,538%**	**6909,48 $**	**9595**
	0,0350 s	20,72 s	96,502%	6822,36 $	9412
	2,4692 s	101,42 s	99,917%	7002,86 $	9750
480 s	**0,8686 s**	**49,63 s**	**99,330%**	**5888,15 $**	**7960**
	0,0217 s	13,42 s	94,687%	5671,26 $	7837
	3,4992 s	119,29 s	99,935%	5974,50 $	8085
600 s	**1,4104 s**	**62,56 s**	**98,923%**	**5205,47 $**	**6865**
	0,0229 s	22,76 s	96,846%	5152,28 $	6780
	4,6200 s	113,09 s	99,947%	5278,00 $	6973
900 s	**2,1704 s**	**64,35 s**	**97,995%**	**4173,56 $**	**5219**
	0,0930 s	22,13 s	92,558%	4100,46 $	5134
	5,6954 s	124,89 s	99,748%	4241,36 $	5332

Table 2: Results of the direct application of the scaling plan

availability, more waiting time in queue); nonetheless they show that the scaling plan is not generally able to perform well for any startup value and some adjustment would be advisable. We do not enter into a deep evaluation of the absolute values of results in this moment because they will be discussed in comparison with the rest of results when adjustments are applied. However, we can already see that all metrics are quite sensitive to the variation of startup time. In the real execution of the application, even knowing the expected values for the startup time, the time required by a concrete activation order of a resource is hardly predictable.

The rest of the experiments show the influence of the techniques for performance improvement on the quality properties of the application and the scaling engine. These techniques allow solving the trade-off between performance and availability and, up to a certain point, they also solve the trade-off between quality properties of the application and quality properties of the scaling engine (i.e., its cost and reconfiguration rate). Above such point, improvements in the quality properties of the application can be reached at the cost of executing scaling plans that increments in the operational cost of the application. All the following experiments use a maximum queue of 1000 requests.

4.4 Deactivation decision interval application

We first experiment the expected properties of the system by using only the technique called deactivation decision interval. Table 3 shows the results obtained for a representative subset of startup times: a short startup of 120s, a medium startup time of 360s and a slow startup of 900s. They show that the utilization of this technique improves the overall system efficiency with respect to the base case. All waiting times in queue, availability, cost of the deployment and number of reconfigurations improve with respect to the base case in Table 2.

Startup Time	*Average time in queue*	*Availability*	*Cost*	*Num of Reconfs*
Deactivation decision interval: 1800 s				
120 s	**0,0082 s**	**99,994%**	**2286,08 $**	**1585**
	0,0004 s	99,943%	2210,29 $	1569
	0,0842 s	99,998%	2300,89 $	1607
360 s	**0,0246 s**	**99,985%**	**2212,56 $**	**1493**
	0,0005 s	99,898%	2208,22 $	1478
	0,1353 s	99,998%	2225,02 $	1515
900 s	**0,0918 s**	**99,888%**	**2087,84 $**	**1350**
	0,0062 s	99,202%	2051,18 $	1325
	1,5446 s	99,989%	2104,78 $	1386
Deactivation decision interval: 3600 s				
120 s	**0,0072 s**	**99,994%**	**2266,08 $**	**1010**
	0,0004 s	99,943%	2200,29 $	999
	0,0641 s	99,998%	2291,89 $	1013
360 s	**0,0186 s**	**99,986%**	**2201,51 $**	**990**
	0,0005 s	99,891%	2199,26 $	981
	0,1053 s	99,993%	2205,02 $	1001
900 s	**0,0588 s**	**99,887%**	**2078,84 $**	**943**
	0,0062 s	99,200%	2051,18 $	920
	1,2436 s	99,979%	2104,78 $	956
Deactivation decision interval: 14400 s				
120 s	**0,0035 s**	**99,996%**	**2378,76 $**	**508**
	0,0006 s	99,994%	2374,10 $	504
	0,0265 s	99,998%	2382,14 $	513
360 s	**0,0287 s**	**99,984%**	**2371,08 $**	**500**
	0,0002 s	99,945%	2355,20 $	491
	0,1622 s	99,996%	2381,84 $	513
900 s	**0,0606 s**	**99,879%**	**2350,40 $**	**489**
	0,0058 s	99,747%	2333,37 $	480
	0,2760 s	99,986%	2359,38$	499

Table 3: Experiment results using a deactivation decision interval of 1800,3600 and 14400 seconds

As expected, the reconfiguration rate always decreases as the interval length increases. It is noticeable that, in the table rows with results for the case of 120 seconds of startup time, for intervals lengths of 1800s 3600s and 14400s, the reconfiguration rate using a deactivation time interval is only 8.7%, 5.6% and 2.8% of the rate in the base case, respectively. The relation of the decrement in the reconfiguration rate shows that it decreases in 30∼35% each time that the deactivation interval doubles its length.

Availability has largely increased its value with respect to the base case, passing from 99.72% to 99.99% of requests served in the case of 120 seconds of startup time, from 99.53% to 99.98% in the case of 360 seconds of startup time, and from 97.99% to 99.88% in the case of 900 seconds of startup time. It has achieved at least one nine more in every case, demonstrating the usefulness of this technique. Besides, we can see little variance of availability results for different interval lengths for a given startup time value. It means that this technique cannot be used in isolation to achieve any value of quality but that beyond a certain interval length this technique stops improving the quality of the application.

Waiting time in queue has also decreased with respect to the base case, concretely for the case of VM startup time of 120 seconds it dramatically decreased to the 1.3%, 1.12%, and 0.54% of the base case when using deactivation interval lengths of 1800s, 3600s, and 14400s, respectively. As it is reasonable, even if applying this technique, higher VM startup times still entail that requests wait higher

time in queue. Regarding the costs of running the application, they have also dramatically decreased when applying the deactivation decision interval technique. In this case, it was found that the lowest costs were found for interval lengths of 3600s. It means that, for the characteristics of workload received, an interval length of 3600 realizes the variations of the workload not related to its *fast fluctuation* better than both 1800s -which filters out less variations coming from the *fast fluctuation*- and14400s -which misses more true variations of the workload due to its longer decision time.

Comparing these results with the ones in the base case in Table 2, we see that the application of this technique reduces both the costs and the average waiting times of requests. From these facts, we can deduce that, in the base case, many of the active resources were in their booting period instead of serving requests, and that this situation is mitigated using deactivation decision interval technique. Note that, although PERFSCALE does not compute the optimal value of interval length, it shows its usefulness by allowing users to execute *what-if* scenarios with different interval length values.

Although the properties of the application and the scaling plan have improved with the application of this technique, we can see that their values are still quite sensitive to the variation in the startup time of VMs.

4.5 Application of hysteresis

Next we experimented PERFSCALE on the simulation of the application when only the hysteresis technique is applied. We have experimented with hysteresis width values h from 2 to 7, and VMs startup times of 120, 360 and 900s. Table 4 presents the experimental results, which are discussed in the following.

Results confirm that this commonly used technique allows to improve the performance and availability of the application and can greatly reduce the reconfiguration rate. The higher the hysteresis width the better they are. The counterpart is an increment in the cost of running the application because, in order to increase the stability of the infrastructure along time, the scaling plan accepts the situation of continuous slight over-provisioning, i.e., within the hysteresis width. Contrarily, in the previous deactivation decision interval technique, the scaling engine always aimed at being in the optimal configuration specified in the scaling plan, even if it allowed itself for an interval of decision time.

In comparison with the base case, we can see that applying some hysteresis is certainly advisable because it improves all properties of the application and scaling engine. The engineer should consider which is the best value for the hysteresis width since, depending on the requirements, high width values improve performance and availability but they also increase the cost of running the application with respect to $h = 2$. From the obtained results, we can also observe that the application properties achieved by this technique are still quite sensitive to the startup time of VMs. For instance, the average waiting time in queue of experiments with startup time of 900s are 4, 12 and 17 times higher than those ones with startup time of 120 seconds for the case of hysteresis width equal to 2, 4, and 7, respectively.

4.6 Combination of deactivation decision interval and hysteresis

Our next experiments show PERFSCALE results when it applies the two previous techniques together. When deciding the type of scaling plan of an application, it is interesting to know whether the application of more than one technique to the concrete application leads them to a symbiotic behavior or to interfere each other, then causing more running costs and not improving the application properties. In this technique, the decision time interval to deactivate the i-th resource is considered after the workload decreases below w_i^d. Table 5 shows the experimental results. They show that, for the example application and expected workload variations, techniques help each other in achieving better system properties.

Startup Time	*Average time in queue*	*Availability*	*Cost*	*Num of Reconfs*
Hysteresis width: 2				
120 s	**0,0221 s**	**99,994%**	**2092,79 $**	**990**
	0,0008 s	99,984%	2088,99 $	983
	0,2992 s	99,998%	2100,03 $	1005
360 s	**0,0280 s**	**99,925%**	**2028,74 $**	**884**
	0,0013 s	99,875%	2016,81 $	870
	0,0669 s	99,995%	2036,56 $	899
900 s	**0,0856 s**	**99,784%**	**1941,01 $**	**774**
	0,0028 s	99,676%	1925,88 $	757
	0,0782 s	99,795%	1952,36 $	788
Hysteresis width: 4				
120 s	**0,0034 s**	**99,992%**	**2218,58 $**	**362**
	0,0004 s	99,985%	2216,87 $	358
	0,0148 s	99,998%	2220,51 $	366
360 s	**0,0092 s**	**99,975%**	**2202,54 $**	**339**
	0,0005 s	99,897%	2199,33 $	334
	0,0451 s	99,998%	2207,94 $	347
900 s	**0,0405 s**	**99,887%**	**2174,51 $**	**319**
	0,0004 s	99,200%	2167,96 $	308
	0,0948 s	99,979%	2178,55 $	327
Hysteresis width: 7				
120 s	**0,0014 s**	**99,996%**	**3413,71 $**	**178**
	0,0001 s	99,990%	3411,29 $	177
	0,0093 s	99,998%	3415,90 $	179
360 s	**0,0033 s**	**99,995%**	**3406,15 $**	**173**
	0,0004 s	99,985%	3399,03 $	172
	0,0097 s	99,998%	3409,90 $	177
900 s	**0,0239 s**	**99,973%**	**3392,37 $**	**168**
	0,0008 s	99,891%	3386,26 $	165
	0,0559 s	99,994%	3398,61$	172

Table 4: Experiment results using hysteresis technique with width values $h = \{2, 4, 7\}$

Waiting time and availability values have improved from the previous cases where technique were actuated in isolation. For instance, in the case of VM startup time of 360 s, the combination of techniques can achieve an expected waiting time of 0.004 at a cost of 1934.45$ while the utilization of the deactivation decision interval technique in isolation, whose cost for this case was always over 2200$, did not achieve such low waiting time. The same happens for startup time of 900s. The combination of techniques for deactivation decision interval of 1800s and hysteresis width $h = 2$ provides better waiting time (0.039s) and lower cost (1909.8$) than any of the results shown in 3 when the expected startup time of VMs is 900s. Regarding the comparison with the application of the hysteresis technique in isolation, for each row of the experiments shown in Table 4 we can find a row in Table 5 with better value of waiting time in queue and lower cost. Results also show that availability values are not largely improved with the combination of techniques. The reason is that, once we have reached 3 and 4 nines for availability, the requests that are still lost come from moments of sudden bursts of requests that fill the queue size, but none of these two techniques can completely eliminate the effect of the burstiness in the workload nature. Looking at the the difference in

Startup Time	*Average time in queue*	*Availability*	*Cost*	*Num of Reconfs*
Timeout: 1800 s - Hysteresis length: 2				
360 s	**0,0040 s**	**99,993%**	**1934,45 $**	**353**
	0,0003 s	99,967%	1929,92 $	352
	0,0363 s	99,999%	1939,18 $	357
900 s	**0,0391 s**	**99,940%**	**1909,86 $**	**349**
	0,0029 s	99,741%	1903,39 $	347
	0,3039 s	99,993%	1914,81 $	355
Timeout: 3600 s - Hysteresis length: 3				
360 s	**0,0117 s**	**99,986%**	**2197,09 $**	**262**
	0,0028 s	99,921%	2189,86 $	260
	0,1268 s	99,997%	2205,40 $	264
Timeout: 7200 s - Hysteresis length: 4				
120 s	**0,0015 s**	**99,997%**	**2559,71 $**	**218**
	0,0001 s	99,995%	2556,29 $	217
	0,0067 s	99,999%	2562,90 $	221
360 s	**0,0052 s**	**99,991%**	**2552,15 $**	**217**
	0,0252 s	99,950%	2549,03 $	216
	0,0097 s	99,998%	2556,90 $	219
900 s	**0,0091 s**	**99,963%**	**2535,41 $**	**216**
	0,0008 s	99,891%	2528,37 $	215
	0,0256 s	99,994%	2540,90$	219
Timeout: 3600 s - Hysteresis length: 5				
120 s	**0,0018 s**	**99,998%**	**2695,03 $**	**186**
	0,0004 s	99,997%	2682,33 $	186
	0,0059 s	99,999%	2702,75 $	187
360 s	**0,0023 s**	**99,994%**	**2684,52 $**	**186**
	0,0005 s	99,977%	2674,37 $	185
	0,0151 s	99,999%	2697,26 $	188
900 s	**0,0089 s**	**99,976%**	**2672,14 $**	**186**
	0,0004 s	99,922%	2658,95 $	185
	0,0198 s	99,990%	2690,57 $	188

Table 5: Experiment results using both techniques deactivation decision interval and hysteresis.

the reconfiguration rate of columns with $h = 4$ (with deactivation interval of 7200 s) and $h = 5$ (with deactivation interval of 3600 s) with respect to the reconfiguration rate observed for $h = 3$ (with deactivation interval of 3600 s), we can deduce that the hysteresis width value has a more profound effect on the reconfiguration rate than the decision interval length, at least for the example system and for large values of interval length.

5. CONCLUSIONS

Everyday, software services are being migrated to cloud computing infrastructures. Some of the reasons are the pay-as-you go policies, and the provided elasticity. The achievement of the most beneficial management of elasticity is one of the hardest issues for non experts in cloud computing.

To help cloud costumers, this work presents PERFSCALE, a simulator based on queueing models of application sand scaling engines. PERFSCALE uses as input scaling plans as defined by [10] and traces of expected workload of the application whose cloud deployment is under consideration. It allows to simulate the application properties when their elasticity is driven by scaling plans and the identified techniques for applicatin performance improvement. The application properties for which PERFSCALE offers simulation results are: average waiting time of request to the application before they are served, availability, expected cost for running the application and reconfiguration rate of virtual machines. This work presents examples of the output of PERFSCALE and the usefulness of its results for cloud costumers.

As future work, we plan to continue experimenting PERFSCALE with other possible workloads and we consider integrating the techniques for elasticity utilization improvement in more detailed cloud simulators like CloudSim [3]. We also plan to extend our research with more techniques for elasticity improvement and with different queueing models in order to first simulate the common three-tier applications and later by allowing the user to specify their particular application topology.

Acknowledgments

The work has been partially supported by the FP7 European project Seaclouds.

6. REFERENCES

[1] Amazon. AWS EC2 Spot Instances. https://aws.amazon.com/ec2/spot/.

[2] Amazon Web Services. Simple Monthly Calculator, 2015. http://calculator.s3.amazonaws.com/index.html.

[3] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya. Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw. Pract. Exper.*, 41(1):23–50, Jan. 2011.

[4] R. Calinescu, S. Gerasimou, and A. Banks. Self-adaptive software with decentralised control loops. In A. Egyed and I. Schaefer, editors, *Fundamental Approaches to Software Engineering*, volume 9033 of *LNCS*, pages 235–251. 2015.

[5] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli. Dynamic QoS Management and Optimization in Service-Based Systems. *IEEE Transactions on Software Engineering*, 37(3):387–409, 2011.

[6] R. Calinescu and M. Kwiatkowska. Cads*: Computer-aided development of self-* systems. In M. Chechik and M. Wirsing, editors, *Fundamental Approaches to Software Engineering*, volume 5503 of *LNCS*, pages 421–424. 2009.

[7] V. Cardellini, E. Casalicchio, V. Grassi, S. Iannucci, F. L. Presti, and R. Mirandola. MOSES: A Framework for QoS Driven Runtime Adaptation of Service-Oriented Systems. *IEEE Trans. Software Eng.*, 38(5):1138–1159, 2012.

[8] A. Khajeh-Hosseini, D. Greenwood, J. W. Smith, and I. Sommerville. The cloud adoption toolkit: supporting cloud adoption decisions in the enterprise. *Software: Practice and Experience*, 2011.

[9] M. Mao and M. Humphrey. A Performance Study on the VM Startup Time in the Cloud. In *IEEE 5th International Conference on Cloud Computing*, pages 423–430, 2012.

[10] D. Perez-Palacin, R. Mirandola, and R. Calinescu. Synthesis of adaptation plans for cloud infrastructure with hybrid cost models. In *EUROMICRO Conf. on Software Engineering and Advanced Applications*, pages 443–450, 2014.

[11] D. Perez-Palacin, R. Mirandola, and J. Merseguer. QoS and energy management with Petri nets: A self-adaptive framework . *Journal of Systems and Software*, 85(12):2796 – 2811, 2012.

[12] PlanForCloud. PlanForCloud, 2014. http://www.planforcloud.com/.

[13] World Cup 1998 Access logs, 1998. http://ita.ee.lbl.gov/html/contrib/WorldCup.html.