Comparison of Auto-scaling Techniques for Cloud Environments

Tania Lorido-Botrán
University of Deusto
6 PUBLICATIONS **68** CITATIONS

Jose Miguel-Alonso
Universidad del País Vasco / Euskal Herriko…
85 PUBLICATIONS **641** CITATIONS

Comparison of Auto-scaling Techniques for Cloud Environments

Tania Lorido-Botran[1] Jose Miguel-Alonso[2] Jose A. Lozano[3]

Abstract— **Cloud computing environments offer the user the capability of running their applications in an elastic manner, using only the resources they need, and paying for what they use. However, to take advantage of this flexibility, it is advisable to use an auto-scaling technique that adjusts the resources to the incoming workload, both reducing the overall cost and complying with the Service Level Objective. In this work we present a comparison of some auto-scaling techniques (both reactive and proactive) proposed in the literature, plus two new approaches based on rules with dynamic thresholds. Results show that dynamic thresholds avoid the bad performance derived from a bad threshold selection.**

Keywords— **Cloud computing, scalable applications, auto-scaling, service level objectives**

I. Introduction

CLOUD computing is a popular technology, characterized by its elastic nature. This means that users only acquire the resources they need (on-demand) and pay only for what they use (a pay-as-you-go scheme). Resources are provided in the form of VMs. There are two types of scaling: horizontal and vertical. Horizontal scaling (or scaling out/in) consists of adding or removing VMs, and vertical scaling (scaling up/down) consists of dimensioning the resources assigned to a particular VM (e.g. CPU or memory). Vertical scaling is not supported for all operating systems; for this reason, many cloud providers only offer horizontal scaling.

Adapting resources to application needs is a challenge, due to the constant changes in the input workload. Manual scaling would require constant supervision and, probably, will cause a bad performance in the application. Instead, an auto-scaling technique is a suitable way to automate resource adaptation. Cloud providers usually offer a rule-based auto-scaling mechanism, which is simple to set up but difficult to tune. The main drawback is its reactive nature: it is unable to cope with sudden workload bursts and also with the boot up time of VMs (up to 10 minutes). A proactive approach tries to overcome these problems, forecasting future resource needs.

Authors in the literature have proposed many reactive and proactive auto-scaling techniques, related to control theory, time series analysis, reinforcement learning and so on. Due to the heterogeneity of the techniques and the testing environment they use, it is impossible to carry out a proper comparison to decide the best auto-scaling technique for a particular scenario. For this reason, we have developed a cloud simulator based on CloudSim [1], specifically designed as a workbench for studying auto-scaling techniques. In this paper we use it to test several representative techniques, analyze the impact of parameter definition and compare techniques in terms of VM cost and SLO violations. Note that this is a preliminary, not exhaustive study.

Cloud providers usually bill on an hourly basis and partial hours are accounted as full hours. In this case, VM cost is accounted as the number of VM running hours, multiplied by the hour fee. The Service Level Agreement (SLA) is the contract between the cloud provider and the customer, in which the Quality of Service (QoS) is defined. The SLA is defined in terms of one or more Service Level Objectives (SLO). In this paper, a single SLO is considered, based on the service time required by incoming requests.

The rest of the paper is organized as follows. The target scenario is described in Section II. Then, each of the auto-scaling techniques is described and tested on the cloud simulator (Section III). Section IV presents some performance results and makes a comparison of the different auto-scaling techniques. Finally, the conclusions extracted, along with some future work lines, are presented in Section V.

II. Scenario

Our target scenario is a web-type application, deployed over a pool of homogeneous VMs. We will focus on the load balancer and the business tier. User requests arrive the application following a given pattern (in this case, based on a real workload). The load balancer will distribute requests among the VMs, based on different policies: random, round-robin or least-connection. Round-robin policy distributes requests in turns, whilst the least-connection one consists of assigning requests to the least loaded VM, i.e., the VM that is executing the least number of tasks.

Each task is assigned to a single VM. Request execution time is denoted as its *expected service time*, and in our simulated environment it is known *a-priori*. VMs adopt a time-sharing policy, so all incoming tasks will be accepted, but service time will increase accordingly to the number of tasks in the CPU. In this context, the *service time* is the lapse since a task is assigned to a VM, until the response is received back.

The auto-scaler will perform horizontal scaling actions: adding (scaling out) or removing VMs (scaling in). Many performance metrics could be applied to

[1]Dep. of Computer Architecture and Technology, U. of the Basque Country, UPV/EHU, e-mail: `tania.lorido@ehu.es`

[2]Dep. of Computer Architecture and Technology, U. of the Basque Country, UPV/EHU, e-mail: `j.miguel@ehu.es`

[3]Dep. of Computer Science and Artificial Intelligence, U. of the Basque Country, UPV/EHU, e-mail: `ja.lozano@ehu.es`

trigger auto-scaling actions. The most typical ones in the literature are CPU load, service time and input request rate, but many others have been proposed such as memory used by the VMs, or available bandwidth. Scaling decision may be taken based on a single metric or a combination of two or more. For simplicity, this study focuses on the use of a single metric.

The performance of each technique may be evaluated in terms of the total VM cost and the total number (or ratio) of SLO violations. Each cloud provider uses its own billing scheme. VM hours can be charged per minute or per hour (and in that case, partial hours are accounted as full hours). The cost may include not only VM running hours, but also other resource usage (e.g. disk or bandwith) and services (e.g. monitoring system).

Figure 1 shows an example of a real workload of one week duration. The number of VMs (represented with a thin line) varies according to the system load (number of requests per minute), from 10-15 VMs, up to 50 VMs at workload peaks.

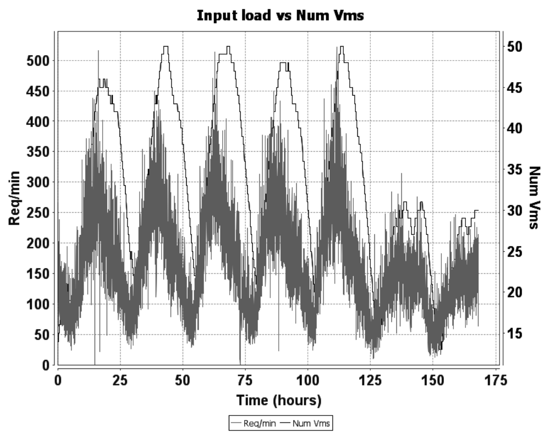

Fig. 1. ClarkNet trace (the workload) vs. number of VMs assigned

III. Description of the Auto-scaling Techniques

Many diverse auto-scaling techniques have been proposed in the literature. In [3], a taxonomy consisting of five categories was defined: static threshold-based rules, time series analysis, control theory, reinforcement learning and queuing theory. In the current study we compare techniques from the first three categories.

Static threshold-based rules are typically used by cloud providers such as Amazon EC2. A simple example: if $CPU > 70\%$, then scale out; if $CPU < 30\%$, then scale in. It is quite difficult to set the correct thresholds and this must be done manually. An incorrect adjustment will cause oscillations in the number of VMs, and therefore, lead to bad performance. After each scaling decision, a cooldown period can be applied, during which no scaling will be performed. This cooldown period reduces the oscillations in the number of VMs and can be applied not only to rules, but to any auto-scaling technique. In RightScale's [4] variation of rules, each VM votes independently, based on rules like those explained before, whether to scale or not. Then, a simple democratic voting is performed to decide the scaling action.

Time series analysis includes a number of methods that use a past history window of a given performance metric in order to predict its future values (*proactive techniques*). In this case, we consider three methods: moving average, exponential smoothing and linear regression. Moving average calculates the mean of the n last values. Exponential smoothing assigns exponentially decreasing weight to each value in the time series. Last, linear regression tries to fit a linear equation to the last values (where x is the time and y is the performance metric value), and then, it estimates a future value.

Control theory has been applied to automate the management of different systems. The typical controller is the Proportional Integral Derivative (PID) controller. In this study, we have implement a simplified version called I (Integral) controller that obeys to the following equation:

$$u_k = u_{k-1} + K_i(y_k - y_{ref}) \tag{1}$$

where u_k is the new value for the manipulated variable (new number of VM); y_k is the performance metric value (CPU load), y_{ref} is the set point or target value; and K_i is the integral gain parameter. In other words, the integral controller will adjust the number of VMs in order to maintain the performance metric value (e.g. CPU load) as closer as possible to the target value.

Lastly, we have proposed a new auto-scaling technique, based on rules, that tries to overcome the limitation of using static thresholds. In this case, the upper and lower thresholds will be adjusted based on another set of rules. First, if SLO violations occurs for a certain time (e.g. 5 minutes), the threshold range will be *widened.* For example, a 60-40% configuration for upper-lower thresholds may be adapted to 80-20%. This makes the system less reactive to workload changes, the auto-scaler will be less eager to remove VMs, and thus, the number of SLO violations will be reduced. The opposite case is when no SLO violations have taken place during the last period of time, so the threshold range is *narrowed.* Then, the system becomes more reactive. The modification to use dynamic thresholds has also been applied to RightScale's voting system.

The dynamic threshold-based rules constitute a novel contribution of this work. All the remaining auto-scaling techniques discussed in this paper, and many additional ones, are described in detail in [3].

IV. Experiments

This section presents the results for the different auto-scaling techniques and analyzes them. We have extended the functionality of the CloudSim cloud simulator to carry out the experiments. As described

TABLE I

Performance values of load balancing policies with static threshold-based rules

Rules		R-Robin		Random		Least-con.	
Upper Thres.	Down Thres.	Cost	SLOv	Cost	SLOv	Cost	SLOv
60	20	5756.70	0.15	5967.40	4.53	5727.40	0.13
70	30	4677.00	0.86	4799.50	10.36	4625.90	0.81
80	30	4330.10	1.46	4335.30	15.50	4294.40	1.39
80	40	4037.20	3.24	4053.40	21.21	4004.70	3.15
90	30	4026.70	2.66	3907.00	23.71	4002.50	2.35
90	10	5172.80	0.49	4855.80	15.46	5151.50	0.34

TABLE II

Results for reactive techniques, based on CPU load, for two different values of boot up time

Technique	Parameters			0 min		10 min	
				VMcost	SLOv	VMcost	SLOv
Rules	UT: 60	DT: 20		5756.70	0.15	5575.80	0.33
	UT: 70	DT: 30		4677.00	0.86	4402.60	2.91
	UT: 80	DT: 30		4330.10	1.46	4155.80	4.92
	UT: 80	DT: 40		4037.20	3.24	3764.10	8.89
	UT: 90	DT: 30		4026.70	2.66	3914.90	7.13
	UT: 90	DT: 10		5172.80	0.49	5136.20	0.79
Dynamic thresholds	UT: 60	DT: 20		5175.50	0.50	5127.60	0.87
	UT: 70	DT: 30		5169.50	0.50	5127.50	0.87
	UT: 80	DT: 30		5168.90	0.51	5127.50	0.87
	UT: 80	DT: 40		5168.30	0.51	5126.80	0.90
	UT: 90	DT: 30		5168.70	0.51	5127.40	0.88
	UT: 90	DT: 10		5173.50	0.48	5136.30	0.79
RighstScale	UT: 60	DT: 20	Votes 75%	5813.70	0.14	5741.20	0.23
	UT: 60	DT: 20	Votes 95%	5712.00	0.11	5665.70	0.19
	UT: 70	DT: 30	Votes 75%	4656.90	0.78	4467.00	2.63
	UT: 70	DT: 30	Votes 95%	4631.10	0.84	4523.00	2.27
	UT: 80	DT: 40	Votes 75%	4001.70	3.12	3771.00	8.54
	UT: 80	DT: 40	Votes 95%	3940.20	3.14	3777.80	8.76
	UT: 90	DT: 10	Votes 75%	5134.60	0.61	5089.40	1.10
	UT: 90	DT: 10	Votes 95%	5090.40	0.87	5092.50	1.34
Dynamic thr. RightScale	UT: 60	DT: 20	Votes 75%	5138.30	0.58	5094.40	1.04
	UT: 60	DT: 20	Votes 95%	5095.00	0.78	5105.10	1.22
	UT: 70	DT: 30	Votes 75%	5136.50	0.58	5094.40	1.04
	UT: 70	DT: 30	Votes 95%	5087.90	0.81	5105.10	1.22
	UT: 80	DT: 40	Votes 75%	5114.90	0.60	5092.90	1.05
	UT: 80	DT: 40	Votes 95%	5086.00	0.82	5105.10	1.22
	UT: 90	DT: 10	Votes 75%	5131.50	0.59	5079.40	1.05
	UT: 90	DT: 10	Votes 95%	5085.30	0.83	5105.10	1.22
IController	K: -0.01	Target: 60%		6474.60	0.04	6538.50	0.29
	K: -0.01	Target: 65%		5658.00	0.16	5492.60	0.71
	K: -0.01	Target: 70%		5089.20	0.44	4906.00	1.81
	K: -0.01	Target: 75%		4602.30	1.17	4467.20	4.07
	K: -0.01	Target: 80%		4207.70	2.40	4098.40	7.55

in Section II, a web-type application is simulated, which is composed of a business tier (15 initial VMs), and a load balancer. All VMs are homogeneous: one core of 1GHz; other resources such as memory, disk or bandwith have not been considered.

Request execution time is generated based on a uniform distribution, that ranges from 3 to 7 seconds. The arrival time is taken from a real workload trace, the ClarkNet trace [5]. It contains 1654276 requests, that arrive in a clear cyclic pattern (see Figure 1): daytime has more workload than the night, and the workload on weekends is lower than that taking place on weekdays.

CPU load (mean load of all VMs) is used as the performance metric. The monitoring interval of one minute and scaling decisions are taken based on the performance metric value from the last minute. Scaling actions add or remove a single VM (other options could be considered, such as adding/removing a percentage of the current number of running VMs). Two different values of boot-up time (time to effectively put to work a new VM) are considered: 0 and 10 minutes. After a scaling decision, a cooldown period is applied. This cooldown period is equal to the boot-up time, plus an extra period of 5 minutes (that is necessary to see the effect of an scaling decision on the system). When a scale-in decision is made, the chosen VM is not removed until all tasks that are being serviced on it finish. Therefore, in a subsequent scale-out action, instead of creating a new VM, we can use one of these VMs that are pending to be removed and avoid the boot-up time.

Each experiment is repeated 10 times. Auto-scaling techniques are compared attending to the mean VM cost (*VMcost*) and the mean number of SLO violations (*SLOv*). In this scenario, the Amazon EC2 [2] billing scheme is applied: VM hours are charged per running hour, and partial hours are accounted as full hours. Boot up time is not charged. An SLO violation occurs when the service time of a task is greater than or equal to $5 * expectedtime$.

Results are divided into several subsections. Focusing on the rules, we analyze the impact of applying different load balancing policies, and also compare static vs. dynamic thresholds. Then, several reactive and proactive techniques are compared.

A. Impact of load balancing policy

Three load balancing policies have been tested: round-robin, random and least-connection. For the comparison, we will focus on the results for static threshold-based rules, and instant boot up of VMs. Results are gathered in Table I.

Random policy shows the worst results, up to 23.71% of SLO violations. Its nature may cause an unbalanced distribution of tasks among the VMs. Round-robin and least-connection policies both show similar results, but the latter performs a little better. The performance of these two policies will depend on the homogeneity in the duration of the tasks. If all tasks have the same execution time, round robin will be the best option. Otherwise, if tasks have different execution times, least-connection policy is able to distribute the load more evenly among the VMs. For the rest of the experiments, we will use round robin as the load balancing policy.

B. Reactive techniques

Reactive techniques considered in this paper include rules, RightScale's voting system, two proposals based on dynamic thresholds and an integral controller. Results for two different values of boot-up time (0 and 10 minutes) are shown in Table II.

B.1 Static vs dynamic threshold-based rules

In this section we compare typical rules based on static thresholds (*Rules*), also combined with a voting system (*RightScale*), against our proposal of adapting thresholds depending on the number of SLO violations. Results are shown for the round-robin load balancer policy, 0 minute boot up time and CPU load as the performance metric (see Table II).

Dynamic thresholds consider 50% as the mid range, 90% as the upper limit and 10% as the lower limit. These limits have been established in order to avoid a 100-0 and/or 95-5% threshold-configurations, that lead to a situation where few scaling actions are performed and the number of VM remains mainly constant. Variations of thresholds are of 5%. Cooldown time is applied for scaling rules, but not for threshold adaptation (this is checked every minute).

Regarding both simple rules and RightScale's voting system based on static thresholds, the performance depends highly on the selection of threshold values. Some configurations result in low VM cost, but in consequence, the number of SLO violations is excessive. Despite the fact that the lowest number of SLO violations is obtained using certain values of static thresholds, our proposal based on dynamic values is able to maintain an acceptable SLO compliance (0.50% for simple rules and 0.58-0.83% for the RightScale's approach), regardless the initial threshold values. Combining dynamic thresholds with simple rules seems to be the best option (rather than using a voting system), since the number of SLO violations remains nearly constant.

B.2 Integral controller

The integral controller is different from the rest of reactive auto-scaling techniques, because it considers a target CPU value, instead of trying to keep CPU within an acceptable range (delimited by upper and lower thresholds). It directly calculates the required number of VMs (this value is rounded up). K value has been selected using a trial and error method. This parameter is really difficult to set, and a bad choice causes high oscillations in the number of VMs. Although it is the technique that achieves the lowest number of SLO violations (0.04%), it does so at a high cost and, therefore, we can not consider integral controller as the best choice.

As a general conclusion, reactive techniques cannot anticipate to changes and a boot-up time of 10 minutes causes an increase in both the VM cost and the number of SLO violations. The performance of each auto-scaling technique highly depends on parameter selection, which is difficult to be done manually. Besides, a bad configuration causes a high number of SLO violations and dissatisfied clients. Our proposal based on dynamic thresholds is able to improve the general performance, by adjusting the thresholds automatically, without human intervention.

C. Proactive techniques

Three proactive methods (based on time series analysis) are considered: moving average (MA), linear regression (LR) and exponential smoothing (ES). They predict the mean CPU load for the next period (1 minute ahead) and this value is used with threshold-based rules to decide the next scaling action. Among the threshold configurations for the scaling rules, we have selected the three that obtained the lowest number of SLO violations in the reactive case: 60-20%, 70-30% and 90-10%. Table

TABLE III
Results for proactive techniques, based on CPU load, for two different values of boot up time

T.	UT LT	Par.	0 min VMcost	0 min SLOv	10 min VMcost	10 min SLOv
MA	60 20	W:2	5699.70	0.15	5638.80	0.24
		W:3	5659.80	0.16	5600.90	0.31
		W:5	5572.30	0.18	5551.20	0.36
		W:10	5479.00	0.23	5393.70	0.41
	70 30	W:2	4585.80	0.78	4414.00	2.59
		W:3	4535.70	0.90	4423.20	2.73
		W:5	4539.30	1.07	4502.20	2.27
		W:10	4505.20	1.43	4469.80	2.62
	90 10	W:2	5063.00	0.88	5090.90	1.27
		W:3	5008.70	1.25	5066.20	1.57
		W:5	5079.30	1.57	5050.70	2.13
		W:10	4945.20	2.37	5003.40	3.41
ES	60 20	α:.1	5364.50	0.30	5371.20	0.45
		α:.3	5544.50	0.18	5548.20	0.33
		α:.6	5751.00	0.13	5715.10	0.23
		α:.9	5735.70	0.15	5603.00	0.29
	70 30	α:.1	4453.30	1.85	4428.30	3.14
		α:.3	4565.70	1.00	4514.40	2.26
		α:.6	4596.20	0.71	4474.00	2.22
		α:.9	4662.70	0.78	4436.80	2.72
	90 10	α:.1	4801.40	3.74	4844.60	4.65
		α:.3	5092.10	1.68	5144.60	2.32
		α:.6	4997.70	1.02	5039.30	1.40
		α:.9	5099.00	0.58	5090.80	0.90
LR	60 20	W:2	5602.30	0.71	4993.90	2.23
		W:3	5594.70	0.45	5057.50	1.76
		W:5	5604.70	0.26	5345.70	0.53
		W:10	5605.60	0.15	5471.60	0.26
	70 30	W:2	4666.90	2.69	4144.70	9.70
		W:3	4691.00	1.88	4183.30	8.06
		W:5	4695.50	1.20	4261.50	5.87
		W:10	4525.30	0.97	4380.30	2.91
	90 10	W:2	4734.60	1.42	4293.70	6.23
		W:3	4836.50	1.09	4581.60	3.52
		W:5	4901.80	0.73	4794.60	1.64
		W:10	4994.70	0.98	4944.00	1.51

III contains the results for two boot-up times: 0 and 10 minutes.

Again, results highly depend on parameter configuration, history window and smoothing factor. In case of MA, the best result is obtained for a history window W equal to 2; higher values of this parameter lead to an increase in the number of SLO violations. On the contrary, for LR case, it is better to use a larger history window ($W = 10$). Last, ES is applied with different values of the smoothing factor α. The lowest number of SLO violations is obtained with $\alpha = 0.6$ (in two out of three cases), which balances the weight given to new values (0.6), and the weight assigned to the past values.

MA and ES techniques are less dependent on the parameter values than LR. The latter can vary from 9.70% of SLO violations with a $W = 2$ to 2.91% with $W = 10$ (with 70-30% thresholds and 10 minutes of boot-up time).

Looking at the number of SLO violations for both instant and 10-minute boot-up times, the lowest results are obtained by ES with $\alpha = 6$ (0.13/0.23%), nearly followed by MA with $W = 2$ (0.15/0.24%) and last, LR with $W = 10$ (0.15/0.26%). All three techniques improve the results of static threshold-based rules (0.15/0.33% for 60-20% threshold values).

D. Comparing Reactive vs. Proactive techniques

Figure 2 shows a global perspective of auto-scaling techniques. Regarding proactive techniques, only results for 60-20% threshold configuration are presented, as they obtained the lowest number of SLO violations.

In general, the performance worsens when the boot-up time of VMs is 10 minutes in terms of SLO violations. In contrast, the overall cost is quite similar for the two values of boot-up time. In almost all cases, for both reactive and proactive approaches, the number of SLO violations highly depends on the parameter configuration. The lowest number of SLO violations is achieved by the integral controller and RightScale's voting system, but both require a fine tuning of two or more parameters. However, proactive techniques (MA and ES), combined with 60-20% threshold-based rules, are less dependent on parameters, and still improve the performance of simple rules. Finally, proactive techniques have been tested with rules for simplicity, but they may be combined with any method such as an integral controller or RightScale's voting system.

V. Conclusions

Cloud computing environments have a great potential for running scalable applications, thanks to their elasticity nature. Auto-scalers are the key to the efficient use of elastic resources, because they enable the user to adjust resources to needs, while complying with SLO and reducing the cost. Using simulation, we have tested some representative auto-scaling techniques, comparing them in terms of overall cost and SLO violations. We have also proposed a method based on rules with dynamic thresholds, that improves the performance of simple, static threshold-based rules. The main conclusion extracted is that any auto-scaling method is very dependent on the parameter tuning.

As future work, we plan to improve our initial idea of using dynamic thresholds. The main reason is that nowadays, cloud providers that offer auto-scaling capabilities are based on rules. But we also want to explore the potential of other proactive techniques, such as ARMA models or reinforcement learning. ARMA techniques are able to model complex time series such as input workload to an application, in order to perform more accurate forecasts. Reinforcement learning is a different approach that learns based on experience the best scaling action to take at a given system state.

Acknowledgements

This work has been partially supported by the Saiotek and Research Groups 2013-2018 (IT-609-13) programs (Basque Government), TIN2010-

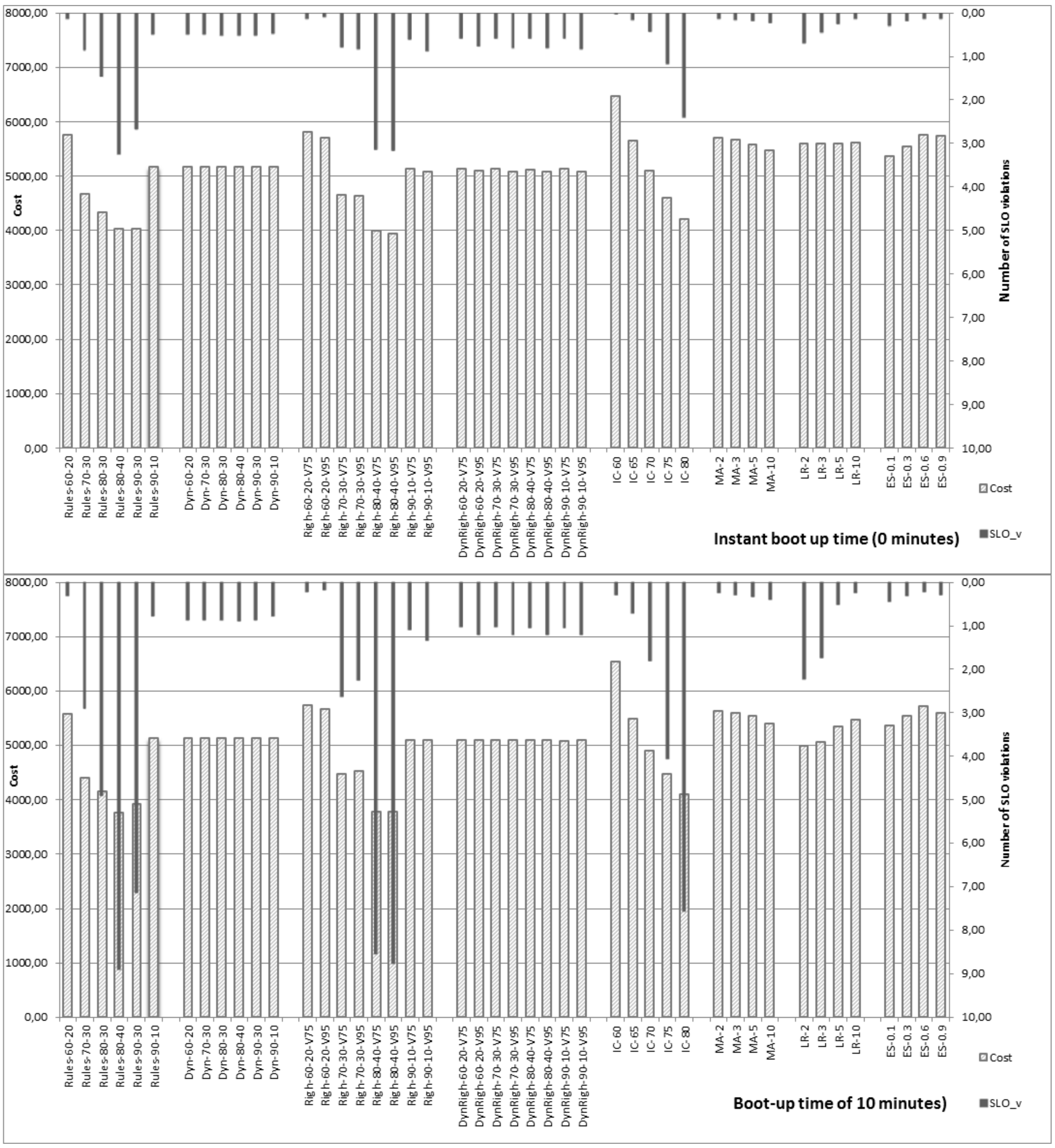

Fig. 2. VM cost and number of SLO violations for auto-scaling techniques, with instant boot-up time (top) and 10 minutes of boot-up time (bottom).

14931 (Ministry of Science and Technology), COMBIOMED network in computational biomedicine (Carlos III Health Institute), and by the NICaiA Project PIRSES-GA-2009-247619 (European Commission). Dr. Miguel-Alonso is member of the HiPEAC European Network of Excellence. Mrs Lorido-Botrán is supported by a doctoral grant from the Basque Government.

References

[1] "CloudSim: A Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services.," `http://www.cloudbus.org/cloudsim/`, 2012, [Online; accessed 18-September-2012].

[2] "Amazon Elastic Compute Cloud (Amazon EC2)," `http://http://aws.amazon.com/ec2/`, 2012, [Online; accessed 15-May-2013].

[3] T Lorido-Botrán, J Miguel-Alonso, and JA Lozano, "Auto-scaling Techniques for Elastic Applications in Cloud Environments," Tech. Rep., University of the Basque Country UPV/EHU; EHU-KAT-IK-09-12, 2012.

[4] "RightScale Cloud Management," `http://www.rightscale.com/`, 2012, [Online; accessed 15-May-2013].

[5] "ClarkNet HTTP Trace (From the Internet Traffic Archive)," `http://ita.ee.lbl.gov/html/contrib/ClarkNet-HTTP.html`, 2012, [Online; accessed 15-May-2013].