An Architecture for Automatic Scaling of Replicated Services

Leonardo Aniello, Silvia Bonomi, Federico Lombardi, Alessandro Zelli, and Roberto Baldoni

Cyber Intelligence and Information Security Research Center and
Department of Computer, Control, and Management Engineering "Antonio Ruberti"
University of Rome "La Sapienza" - Via Ariosto 25, 00185, Rome, Italy
{last name}@dis.uniroma1.it

Abstract. Replicated services that allow to scale dynamically can adapt to requests load. Choosing the right number of replicas is fundamental to avoid performance worsening when input spikes occur and to save resources when the load is low. Current mechanisms for automatic scaling are mostly based on fixed thresholds on CPU and memory usage, which are not sufficiently accurate and often entail late countermeasures. We propose Make Your Service Elastic (MYSE), an architecture for automatic scaling of generic replicated services based on queuing models for accurate response time estimation. Requests and service times patterns are analyzed to learn and predict over time their distribution so as to allow for early scaling. A novel heuristic is proposed to avoid the flipping phenomenon. We carried out simulations that show promising results for what concerns the effectiveness of our approach.

Keywords: automatic scaling; performance modeling; traffic forecasting; QoS compliance; resource-saving

1 Introduction

While designing a replicated service to deliver target response time for a fixed workload is easily achievable by properly tuning the number and the specifics of replicas, it becomes really challenging for highly variable loads. Methods based on *over-provisioning* allow to cope with load peaks but entail huge waste of resources, which translates in to money loss. Nevertheless, *under-provisioning* systematically fails in delivering required performance when input spikes occur. The actual alternative to static provisioning is rendering the service *elastic*, so that it can adapt to fluctuating workloads by changing the number of replicas (*configuration*) on the fly. Such a functionality is called *auto scaling*. Amazon Web Services (`http://aws.amazon.com`) and Google App Engine (`https://appengine.google.com`) are among the most relevant XaaS providers offering the possibility to reconfigure at runtime. Both allow to define policies that trigger a reconfiguration on the basis of the variation of a set of off-the-shelf and custom metrics, like memory usage and CPU utilization.

This kind of solutions has two main issues. One concerns the difficulty to find an accurate relationship between the value of monitored metrics and the configuration needed to meet latency requirements. Indeed, an effective model should be employed in order to find the proper relationship between gathered measures and expected performance. The other regards the timeliness in reacting to load variations: spotting a problematic situation when it is already occurring brings temporary Quality of Service (QoS) violations, while reacting late to a load drop causes the same problems of over-provisioning. The reaction delay also includes the time required for the new configuration to be ready, which comprises additional factors like replica activation time and state transfer time.

One additional challenge of designing elastic solutions is the cost of elasticity, i.e. the cost that the service provider pays to activate/deactivate a replica like, for example, the bandwidth used to transfer the state from an active replica to a new one, the energy used to keep replicas running with low utilization and the tradeoff between buying the infrastructure or just renting it. Usually, such costs occur each time that the system moves from one configuration to another and they may grow up if the *flipping phenomenon* (i.e. a sequence of activation and deactivation of replicas) is not properly mastered.

We propose a solution aimed at facing these issues. We designed the *Make Your Service Elastic* module (MYSE) for the automatic horizontal scaling of a replicated service. By monitoring input requests patterns and the service times delivered by the replicated service, the MYSE module learns over time through neural networks how input load and service times vary, and produces estimations to enable early decisions about reconfiguration. A queuing model of the replicated service is used to compute the expected response time given the current configuration (number of replicas) and the distributions of both input requests and service times. A novel graph-based heuristic called *Flipping-reducing Scaling Heuristic* is employed that leverages this model to find the minimum number of replicas required to achieve the target performance and to reduce as much as possible the flipping phenomenon.

We carried out simulations by using a real dataset containing the requests to a website over the time. The results showed high accuracy in input traffic prediction and good effectiveness in taking proper scaling decisions. These promising outcomes of the MYSE module validation have driven us to start its real implementation on Amazon Web Services. With reference to the related work (see Section 2), the novelty of MYSE consists in (i) combining together traffic forecasting, done through artificial neural networks, and performance estimation through queuing models, and (ii) addressing the problem of flipping by employing the innovative Flipping-reducing Scaling Heuristic.

The rest of the paper is organized as follows: related works are presented in Section 2; Section 3 describes the MYSE architecture; Section 4 reports the preliminary results obtained from simulations and Section 5 outlines how the work is going to continue.

2 Related Work

According to a recent survey [19], there are several works on automatic scaling of elastic applications in the cloud. This survey proposes the following classification of the auto-scaling techniques existing in literature, on the basis of the approach they employ.

- **Static, threshold-based policies.** The configuration is changed according to a set of *rules*, some for scaling out and others for scaling in ([9] [14] [15] [20]). This is a completely *reactive* approach that is currently used by most of the cloud providers.
- **Reinforcement learning.** It is an automatic decision-making technique to learn online the performance model of the target system without any a priori knowledge. Such continuous learning is used to choose the best scaling decision according to the goals of decreasing response time and saving resources ([3] [10] [26] [30]).
- **Queuing theory.** The target system is modeled using techniques coming from the queuing theory with the aim of estimating its performance given a small set of parameters, like input rate and service time, that can be monitored at runtime ([30] [31] [32] [36]).
- **Control theory.** It is used to automate the management of scaling decisions through the employment of a feedback or feed-forward controller module whose objective is to meet performance requirements by adjusting the configuration of the target system ([5] [1] [24] [25] [34]). Feedback controllers correct their behavior by taking into account the error reported by the target system through a *gain parameter* that can be adapted dynamically. Feed-forward controllers are based on model predictive control (MPC) and aim at forecasting the future behavior of the system. The relationship between the input (the workload) and the output (the configuration to adopt) is embedded into the transfer function, which can be implemented in several ways (i.e., smoothing spline, Kalman filter, Fuzzy model).
- **Time-series analysis.** It can be used to spot recurring patterns of the workload over time, in order to forecast future workload so as to come to a scaling decision early. Several techniques can be used like averaging methods, regression and neural networks ([6] [7] [8] [13] [16] [17] [18] [21] [27] [29]).

The limitations of an approach based on static, threshold-based rules lies in its reactive nature: it only takes action after the recognition of a situation that requires scaling, and during the time needed for the scaling to complete either the system provides poor performance or resources are wasted. This problem can be addressed by using time-series analysis in order to forecast how the workload is likely to change over time so as to enable scaling decisions in advance and consequently avoid transient periods where the system is not properly configured.

The drawbacks of the techniques based on reinforcement learning are the excessive length of the training phase before reaching a point where it becomes effective, and the difficulty to adapt to workloads that change quickly.

Using control theory can actually be a valid choice, but choosing the right gain parameter is hard. In fact, considering a fixed gain parameter, its tuning is hard and cannot be adjusted at runtime; on the contrary, using an adaptive parameter, that is changed according to the workload, is likely to introduce flipping.

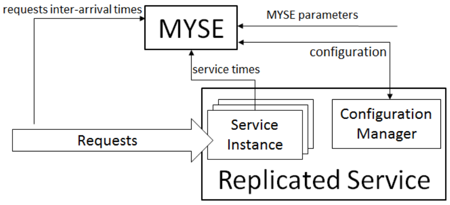

Fig. 1. Integration of MYSE module with target replicated service.

The employment of queuing models to estimate performance can turn out to be not reliable enough because the hard assumptions it requires could be not valid in a real scenario. Nevertheless, we chose to model the replicated service using a queue model because it doesn't require a long training as reinforcement learning does instead.

In addition to the works cited in the survey, also others employ a proactive approach for auto-scaling as we do, but none combines together traffic forecasting through artificial neural networks, and performance estimation through queuing models. Ghanbari et al. [12] present an auto-scaling approach based on MPC that aims both to meet SLA and save resources by framing the problem as an MPC problem. Moore et al. [22] describe an elasticity framework composed by two controllers operating in a coordinated manner: one works reactively on the basis of static rules and the other uses a time-series forecaster (based on support vector machines) and two Naive Bayes models to predict both the workload and the target system performance.

For what concerns how the flipping phenomenon is dealt with in literature, the survey [19] reports that such an issue is addressed in some of the threshold-based works by setting two distinct thresholds: one for scaling out and another for scaling in, so as to have a "tolerance band" that can absorb part of the oscillations. The survey also advises to set so called *calm periods* during which scaling decisions are suspended. Our approach is based on the concept of calm periods, as suggested in the survey, but is more refined as the length of such period is adapted on the basis of the amount of flipping experienced so far.

3 MYSE Architecture

Figure 1 shows how the MYSE module is expected to be integrated with the target service. We assume that a Configuration Manager module is available to receive external commands that update the configuration.

We modeled the service as a queuing system with s servers [11]. Without loss of generality, we considered that replicas are homogeneous (i.e. they have the same computational capabilities and can be used interchangeably) and a single class of service. The basic idea is to consider the replicated service as a black box and monitor requests patterns over time to identify the relevant characteristics of input traffic so as to properly reconfigure the service through the Configuration

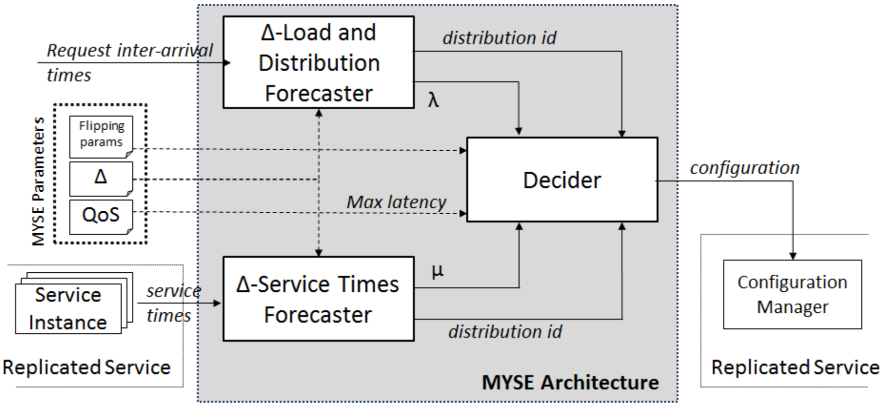

Fig. 2. MYSE Internal structure

Manager. To this aim, we assume that replicated service instances export, as performance metrics, their service times. This allows us to follow the black box approach as in [2, 33] by considering only observable parameters.

Monitoring requests arrival and service times over time enables to predict their probability distribution. The queuing model is then used to compute the expected latency in serving a single request, which can be compared to given QoS requirements to figure out whether the compliance is actually achieved. The same queuing model is employed to derive the minimum configuration allowing to meet the QoS and, at the same time, to avoid wasting resources. We employed four Artificial Neural Networks (ANNs), two ANNs to learn how request rate and request distribution vary over time, and other two to learn how service times and their distribution vary over time. In this way, we can conveniently update the configuration early enough to avoid temporary performance worsening or resource under-utilization. The timeline of these predictions is provided externally (Δ parameter).

Figure 2 details the submodules of MYSE. The Δ-*Load and Distribution Forecaster* is in charge of learning and forecasting request rate and request distribution (it includes Δ-*Load Forecaster* and Δ-*Distribution Forecaster* submodules). The Δ-*Service Times Forecaster* looks at service time patterns to extract the distribution of service times and its mean μ. The *Decider* determines the suitable configuration to meet QoS on the basis of the inputs supplied by the other two submodules and of the *Flipping Parameters* (see Section 3.4). Single submodules are described below.

3.1 Δ-Load Forecaster Submodule

It analyzes request rates over time and employs an ANN to provide predictions on expected request rate λ within Δ time units. A real dataset with 8000 hours provided by Google Analytics framework on our department services is used for training and testing the ANN. It has been divided in three parts: 70% for

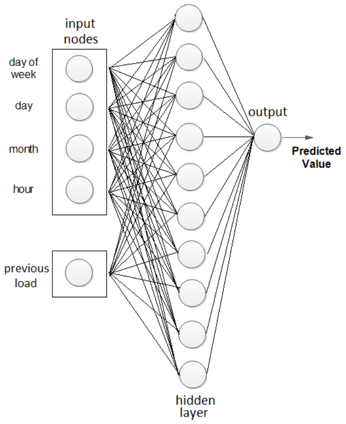

Fig. 3. The ANN implementing the Δ-Load Forecaster submodule.

training, 15% for validation and 15% for test using a k-fold cross validation with $k = 10$ in order to choose the right ANN parameters and address overfitting. We normalized the input parameters with the *Max-Min Normalization* $[0, 1]$ and employed the *Backpropagation Algorithm* [28] for learning, with a sigmoid as activation function. A general method to set the network parameters doesn't exist, so we empirically fixed *learning rate* and *momentum* (to 0.3 and 0.5, respectively), by executing several tests aimed at trading off the recognition error with the exposure to overfitting.

Several guidelines are available for choosing the number of hidden layers and nodes for obtaining good generalization and low overfitting. One hidden layer is sufficient to approximate any complex nonlinear functions to any desired accuracy. We implemented several ANNs to find out the best one, and it turned out to be the one with four input nodes for the date (day, day of the week, month, hour) and one input node for the current traffic. Again, we chose the number of hidden nodes empirically (as also suggested in [35]) and we found out that using 11 hidden neurons gives the best performances. The output node simply represents the predicted traffic values. Figure 3 shows the final architecture of the ANN used in the Δ-Load forecaster.

We also carried out some simulations to check how long it takes for the ANN to be properly trained as the number of backpropagation iterations varies. The obtained results show that using up to 100 iterations allows to keep the training time below 1.2 seconds.

3.2 Δ-Distribution Forecaster Submodule

It is composed by two parts, the *Distribution Recognizer* and the *Distribution Forecaster*; it analyzes requests inter-arrival times to produce predictions on request distribution Δ time units ahead. The Distribution Recognizer estimates the best-fitting continuous or discrete distribution by analyzing a set of samples

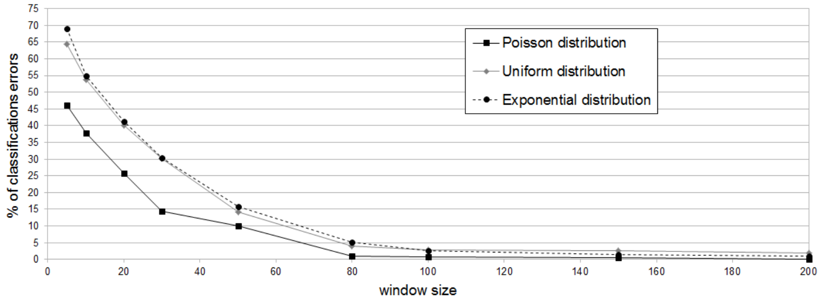

Fig. 4. Classification error in the Δ-Distribution Recognized while increasing the window size.

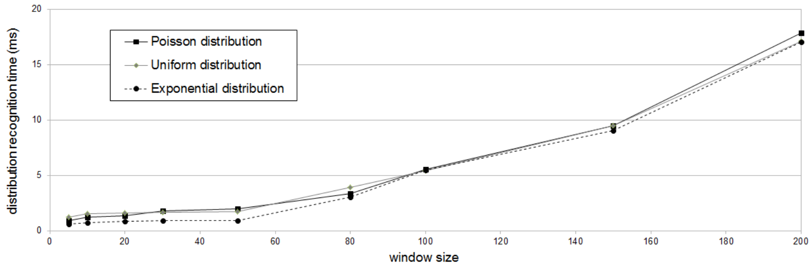

Fig. 5. Time needed by the Δ-Distribution Recognized to get the estimation of the distribution type by varying the window size. This experiment has been executed on an Intel Core 2 2.00 GHz, 2GB RAM.

given in input, which represent the requests inter-arrival times. These samples are analyzed with a fixed-length sliding window. We computed that the length has to be at least 80 samples to get an estimation error lower than or equal to 5% (see Figure 4); on the other hand, Figure 5 gives evidence that obtaining an estimation latency below 5 ms requires 80 samples or less.

The estimation of distribution parameters is made by using the *maximum-likelihood estimation method* [23]. This result is then used to perform “goodness-of-fit” tests, such as the *chi-squared test* (for discrete distributions, i.e. Poisson) and *Kolmogorov-Smirnov test* (for continuous distributions, i.e. Normal), for discriminating among distinct distributions [4]. In the current implementation, the submodule is able to recognize and classify the following distributions: Poisson, Uniform and Exponential. Obviously, in case the real distribution of the samples doesn’t correspond to any of those recognized by the Distribution Recognizer, then the most similar distribution is chosen. The Distribution Forecaster is able to predict the future distribution by using an ANN trained by taking as input the output produced by the Distribution Recognizer, which is an encoding of the recognized distribution. Such an encoding consists in assigning to each recognizable distribution a distinct numerical identifier, so that the identifiers are distanced enough to avoid possible conflicts. The choice to use two distinct parallel ANNs (this one and the one of the *Δ*-Load Forecaster) is made to improve the forecasting accuracy, as suggested in [35]. The ANN is built following the same empirical guidelines discussed in Section 3.1: it has as input four nodes for the date and one node for the previous distribution id, 150 hidden neurons and one output for the forecasted distribution. The learning rate is fixed to 0.9 and momentum to 0.4, while the other parameters are the same derived in Section 3.1.

3.3 *Δ*-Service Times Forecaster Submodule

It takes as input the service times provided by the replicated service and produces as output the estimation of their distribution and the mean μ of service times. The same techniques employed for the *Δ*-Load and Distribution Forecaster are used here.

3.4 Decider Submodule

The submodule computes the minimum configuration for guaranteeing QoS compliance in service provisioning (i.e. the latency in the specific case). It takes as input the latency threshold specified in the QoS, the predictions on request distribution (distribution id and λ), the predictions on the distribution of service times (distribution id and μ) and the *Flipping parameters*, which are the tuning parameters of the Flipping-reducing Scaling Heuristic, described in detail later in this section. The Decider submodule first applies the well-known queuing model techniques to compute the expected latency T in the current configuration (i.e. with s replicas), given the request rate and the service times provided by the forecasters. Then, it applies the Flipping-reducing Scaling Heuristic to decide whether scaling out (in case QoS is violated), scaling in (in case a configuration with less replicas can still guarantee QoS compliance) or keeping the current configuration.

The Flipping-reducing Scaling Heuristic. If QoS compliance can be achieved with the current configuration (with s replicas), then the algorithm evaluates if switching off replicas still makes it possible to satisfy QoS requirements. It computes the maximum number n of replicas that can be removed without violating the QoS, and moves from a configuration with s servers to a configuration with $s - n$. On the contrary, if the expected latency is higher than QoS threshold, then the algorithm computes the minimum number n of replicas to activate in order to comply with the QoS, and moves from a configuration with s servers to a configuration with $s + n$. Highly variable traffic and prediction errors can make the configuration oscillate very often, introducing a lot of overhead due to frequent activation/deactivation of replicas. This phenomenon is referred to as *flipping*, and we dealt with it by introducing a *cost function* that prevents the algorithm from moving back in a certain configuration in case such configuration was set too recently.

To this aim, we defined an edge-weighted directed graph $G = (V, E, w(e, t))$ where (i) the set of vertex V represents all the possible configurations (i.e. number of active replicas) i.e. $V = \{1, 2, 3, \ldots s_{max}\}$, (ii) there exists an edge between any pair of vertexes (iii) for any edge $e_{s,s'} \in E$ the edge weight $w(e_{s,s'}, t)$ represents the cost of moving from the configuration s to the configuration s' at the current time t.

The cost function is defined as $w(e_{s,s'}, t) = FlippingCost \cdot flipping\%$, where $FlippingCost$ is one of the Flipping parameters, while $flipping\%$ is computed as the number of the flippings detected from the beginning divided by the number of heuristic executions. The detection of a flipping is triggered when two scaling decisions in opposite directions (i.e., a scaling out and a scaling in) are executed within a configurable window (the $FlippingWindow$, another Flipping Parameter). In this way, the $flipping\%$ decreases in time in case no flipping occurs. When the algorithm moves from the configuration s to the configuration s' at time t, the edge $e_{s,s'}$ is assigned with the value $w(e_{s,s'}, t)$. Then, such weight is decreased linearly in time until it comes back to 0. The transition from a configuration s to a configuration s' is allowed at time t only if $w(e_{s,s'}, t) = 0$. The flipping phenomenon is very likely to involve additional costs because of the high number of machine activations. On the other hand, forcing to keep a certain configuration for a period of time regardless of the load can raise temporary over-provisioning and/or under-provisioning, and both lead to increased costs. The $FlippingCost$ parameter is indeed aimed at tuning such a tradeoff, on the basis of both the real costs of replica activation and QoS violation, and the expected variability of the input traffic.

We evaluated the effectiveness in avoiding the flipping phenomenon by carrying out a simulation that compares the behavior of the heuristic that uses the cost function with a heuristic that doesn't put any cost on the edges (i.e., $FlippingCost = 0$). We set $FlippingCost$ to 15000 and $FlippingWindow$ to 100 seconds. The results shown in Figure 6 give evidence that our heuristic is successful, indeed it manages to keep the configuration fixed during the intervals when the other oscillates instead. It is to note that at the beginning the algorithm actually introduces flipping, but this is due to the fact the no flipping was occurred before, so $flipping\%$ is zero, yet.

4 Simulations

In this section we describe a set of simulations aimed at assessing the effectiveness of the proposed architecture. In particular, we first evaluated the ability of the Δ-Load Forecaster and Δ-Distribution Forecaster to adapt to different types of loads and then we evaluated globally the goodness of the approach by evaluating the evolution of the configurations in time. In all these simulations, Δ is fixed to one hour.

Evaluation of the Δ-Load Forecaster. In this experiments, we used a real dataset including one year of statistics collected through the Google Analytics framework on our Department services. In order to show the adaptation to

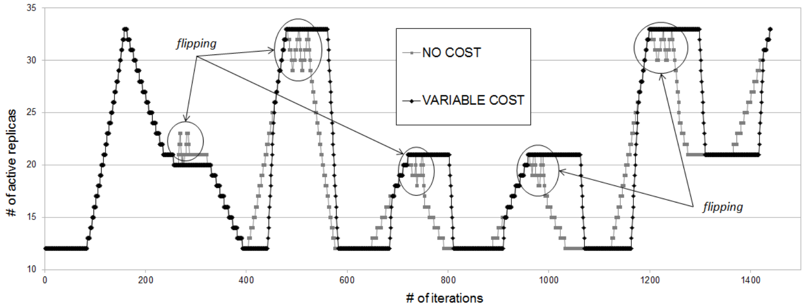

Fig. 6. A comparison between the heuristic employing a cost function and another one that doesn't use any cost function. Over time, it is shown that putting costs on the edges allows to limit the flipping phenomenon.

changes in the load pattern, this dataset has been integrated with a synthetic one. In particular, starting from the real Google Analytics dataset, we appended five days of the same dataset that has been scaled (amplified), and a sine function. We didn't simulate the Service Times Estimator because of the unavailability of traces describing service times over time, but we considered it as fixed and known to the Decider Module. Anyway, we believe that the effectiveness of the results obtained for the Δ-Load and Distribution Forecaster can also hold for this module, since the employed techniques are the same. In order to make the ANN really adaptive to traffic changes, online learning is employed as follows. We store the training set in a fixed-length sliding window containing the last 100 inputs, and at each new input the ANN is trained using all the inputs in the window (by executing 100 iterations of the backpropagation algorithm, as explained in Section 3.1). Figure 7 shows the comparison between the real number of requests over time and the predictions. As we can see, the predictions follow the real pattern and converge quite quickly after a request pattern change. The weekly *Root Mean Square Error* (RMSE) of predictions is 4%, and the *Mean Average Error* (MAE) is 3%.

Evaluation of the Δ-Distribution Forecaster We evaluated the capability of this module to recognize a distribution by using a synthetic dataset where the distribution of inter-arrival times changes very often over time, from Uniform to Poisson and viceversa. We measured the number of iterations required to correctly recognize distribution changes, where one iteration corresponds to the analysis of the samples of a single window, whose length was fixed to 80 samples (see Section 3.2). The results showed that 62 iterations are required on average to detect the transition from Uniform to Poisson, and 30 are needed instead for the opposite transition. Since the window slides at each sample, these results indicate that distribution changes can be recognized before the window gets totally renewed.

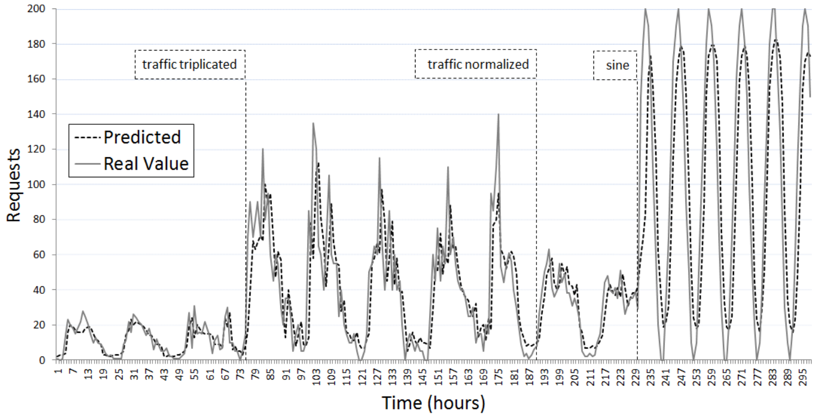

Fig. 7. Comparison between dataset traffic and forecaster predictions. The instants in time are indicated where traffic pattern changes: from normal to triplicated, to normal again and finally to a sine function with amplitude 100 is used.

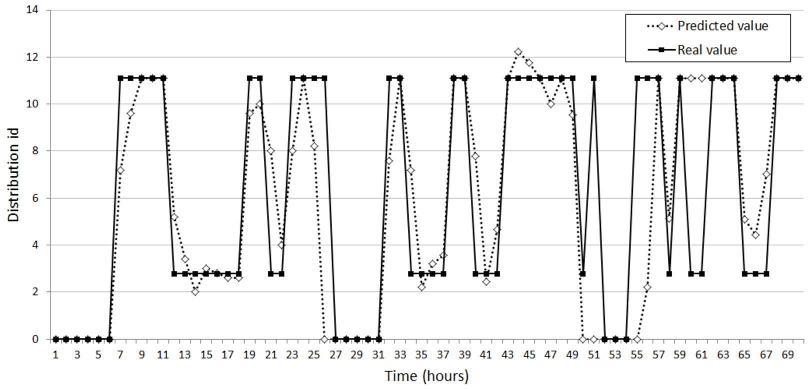

Fig. 8. Comparison between real and predicted request arrival distributions. Three distributions are recognizable: Uniform (id=0), Exponential (id=3) and Poisson (id=11). The average error in 72 hours is 3%

We also evaluated the predictive accuracy of the ANN, by comparing real and forecasted request distributions over time. Three distinct distributions are recognized, each mapped to distinct ids chosen so that classification errors get minimized. We modified the dataset used for the evaluation of the Δ-Load Forecaster in such a way that the distribution of inter-arrival times changes very often over time among the recognizable distributions. Figure 8 shows that predictions are notably accurate (3% error over 3 days).

Evaluation of the overall architecture. This evaluation is aimed to highlight the relevant added value of employing traffic predictions for correctly issuing re-

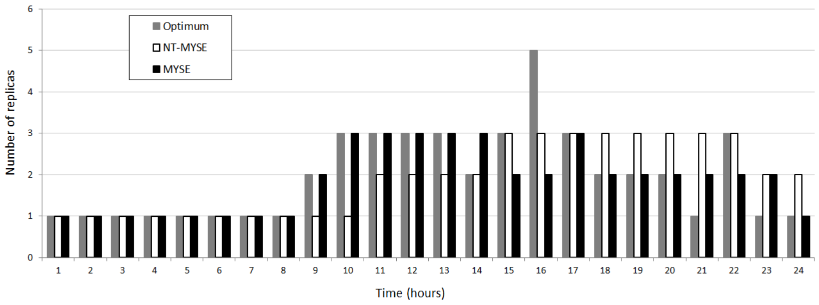

Fig. 9. Comparison between the number of replicas requested by MYSE and NT-MYSE, together with the indication of the optimum, that is the minimum number of replicas required to meet QoS requirements.

source provisioning. The dataset used here is the Google Analytics one. Figure 9 shows how the number of requested replicas varies over time on a hour-basis during a whole day. These results have been obtained by simulating three distinct scenarios: (i) the optimal configuration, that is the minimum number of replicas required to meet QoS requirements, (ii) the configuration produced without the contributions of Δ-Load and Distribution Forecaster, referred to as *Non-Trained MYSE* (NT-MYSE) and (iii) the configuration requested by the complete MYSE module. In the simulation of NT-MYSE, the Decider is fed with traffic details that are produced in real-time by the Distribution Recognizer on the basis of the current traffic only.

Until 8:00 the traffic is stable and both the approaches behave correctly, then traffic begins changing and the use of predictions shows its effectiveness by contributing to generate configurations that are nearer to the optimum compared to NT-MYSE approach. Another important advantage of employing predictions is rendering the system more robust to unexpected peaks. This can be seen by observing the effect of the isolated peak occurring at 16:00 (a peak in the number of the optimal number of replicas corresponds to a peak in traffic load): NT-MYSE is biased by such occurrence and hereafter keeps to over-provision, while MYSE correctly recognizes it as an outlier. We quantified numerically the error of each approach by averaging over the entire day the number of replicas above (over-provisioning) and below (QoS violation) the optimum. NT-MYSE provided on average 0.29 replicas in excess and 0.33 replicas less than the minimum required, for a total of 0.62. MYSE allows on average to over-provision 0.13 replicas (55% more accurate) and under-provision 0.21 replicas (36% more accurate), that is a total error of 0.34 replicas (45% more accurate).

5 Conclusions and Future Directions

Avoiding both performance worsening and over-provisioning for replicated services is the goal of the architecture we propose in this work and, according to the results obtained by the simulations we carried out, we believe that forecasting requests and service times, and carefully modeling how performance gets affected, are the proper building blocks for achieving that objective. We claim that this approach is novel within the context of auto-scaling works. An additional original contribution concerns the Flipping-reducing Heuristic, which allows to address the problems due to quick oscillations in the load. Along this line, we are carrying on this work by exploring distinct directions.

At the time of this writing we are deploying the MYSE architecture on Amazon Web Services in order to concretely assess strengths and weaknesses of the model. Once a target service for the prototype implementation is chosen, we will be able to provide both a validation of the MYSE module and the specification of the analytical model for the actual service in order to compare the results. The deployment of MYSE in a real cloud infrastructure will allow to estimate the delays due to replicas activation/deactivation, which will contribute to the accuracy of the model itself. Another important aspect we want to investigate is the economics behind this elastic model. What is the best model to price the elastic replication system? Who are the customers for this kind of service? The answers to these questions depend on the definition of a precise cost model that can help to identify the market sectors where such replication mechanism could be beneficial.

Finally, we are investigating how to make the MYSE architecture completely non-intrusive, i.e., so that the MYSE module does not need any information provided by the replicated service, such as the service times. This would enable and ease its employment in a wider range of applications.

6 Acknowledgments

This work has been partially supported by the TENACE PRIN Project (n. 20103P34XC) funded by the Italian Ministry of Education, University and Research and by the academic project C26A133HZY funded by the University of Rome “La Sapienza”.

References

1. Ali-Eldin, A., Tordsson, J., Elmroth, E.: An adaptive hybrid elasticity controller for cloud infrastructures. In: Network Operations and Management Symposium (NOMS), 2012 IEEE. pp. 204–212 (2012)
2. Baldoni, R., Lodi, G., Montanari, L., Mariotta, G., Rizzuto, M.: Online black-box failure prediction for mission critical distributed systems. In: SAFECOMP. pp. 185–197 (2012)
3. Barrett, E., Howley, E., Duggan, J.: Applying reinforcement learning towards automating resource allocation and application scalability in the cloud. Concurrency and Computation: Practice and Experience 25(12), 1656–1674 (2013)
4. Biswas, S., Ahmad, S., Molla, M.K.I., Hirose, K., Nasser, M.: Kolmogorov-smirnov test in text-dependent automatic speaker identification. Engineering Letters 16(4), 469–472 (2008)

5. Bodík, P., Griffith, R., Sutton, C., Fox, A., Jordan, M., Patterson, D.: Statistical machine learning makes automatic control practical for internet datacenters. In: Proceedings of the 2009 Conference on Hot Topics in Cloud Computing. HotCloud'09, USENIX Association, Berkeley, CA, USA (2009)
6. Cardosa, M., Chandra, A.: Resource bundles: Using aggregation for statistical large-scale resource discovery and management. Parallel and Distributed Systems, IEEE Transactions on 21(8), 1089–1102 (2010)
7. Caron, E., Desprez, F., Muresan, A.: Forecasting for Cloud Computing On-demand Resources Based on Pattern Matching. Research RR-7217, Inria (2010)
8. Chen, G., He, W., Liu, J., Nath, S., Rigas, L., Xiao, L., Zhao, F.: Energy-aware server provisioning and load dispatching for connection-intensive internet services. In: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation (NSDI). pp. 337–350. USENIX Association (2008)
9. Dutreilh, X., Rivierre, N., Moreau, A., Malenfant, J., Truck, I.: From Data Center Resource Allocation to Control Theory and Back. In: Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on. pp. 410–417 (2010)
10. Dutreilh, X., Kirgizov, S., Melekhova, O., Malenfant, J., Rivierre, N., Truck, I.: Using reinforcement learning for autonomic resource allocation in clouds: Towards a fully automated workflow. In: ICAS 2011, The Seventh International Conference on Autonomic and Autonomous Systems. pp. 67–74. Venice/Mestre, Italy (2011)
11. Garlan, D., Cheng, S.W., Schmerl, B.: Architecting Dependable Systems. chap. Increasing System Dependability Through Architecture-based Self-repair, pp. 61–89. Springer-Verlag, Berlin, Heidelberg (2003)
12. Ghanbari, H., Simmons, B., Litoiu, M., Barna, C., Iszlai, G.: Optimal autoscaling in a iaas cloud. In: Proceedings of the 9th International Conference on Autonomic Computing. pp. 173–178. ICAC '12, ACM, New York, NY, USA (2012)
13. Gong, Z., Gu, X., Wilkes, J.: Press: Predictive elastic resource scaling for cloud systems. In: Network and Service Management (CNSM), 2010 International Conference on. pp. 9–16 (2010)
14. Han, R., Guo, L., Ghanem, M., Guo, Y.: Lightweight resource scaling for cloud applications. In: Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on. pp. 644–651 (2012)
15. Hasan, M., Magana, E., Clemm, A., Tucker, L., Gudreddi, S.: Integrated and autonomic cloud resource scaling. In: Network Operations and Management Symposium (NOMS), 2012 IEEE. pp. 1327–1334 (2012)
16. Huang, J., Li, C., Yu, J.: Resource prediction based on double exponential smoothing in cloud computing. In: Consumer Electronics, Communications and Networks (CECNet), 2012 2nd International Conference on. pp. 2056–2060 (2012)
17. Iqbal, W., Dailey, M.N., Carrera, D., Janecek, P.: Adaptive resource provisioning for read intensive multi-tier applications in the cloud. Future Gener. Comput. Syst. 27(6), 871–879 (Jun 2011)
18. Islam, S., Keung, J., Lee, K., Liu, A.: Empirical prediction models for adaptive resource provisioning in the cloud. Future Gener. Comput. Syst. 28(1), 155–162 (Jan 2012)
19. Lorido-Botrán, T., Miguel-Alonso, J., Lozano, J.A.: Auto-scaling Techniques for Elastic Applications in Cloud Environments. Research EHU-KAT-IK, Department of Computer Architecture and Technology, UPV/EHU (2012)
20. Maurer, M., Brandic, I., Sakellariou, R.: Enacting slas in clouds using rules. In: Proceedings of the 17th International Conference on Parallel Processing - Volume Part I. pp. 455–466. Euro-Par'11, Springer-Verlag, Berlin, Heidelberg (2011)

21. Mi, H., Wang, H., Yin, G., Zhou, Y., Shi, D., Yuan, L.: Online self-reconfiguration with performance guarantee for energy-efficient large-scale cloud computing data centers. In: Services Computing (SCC), 2010 IEEE International Conference on. pp. 514–521 (2010)
22. Moore, L.R., Bean, K., Ellahi, T.: Transforming reactive auto-scaling into proactive auto-scaling. In: Proceedings of the 3rd International Workshop on Cloud Data and Platforms. pp. 7–12. CloudDP '13, ACM, New York, NY, USA (2013)
23. Myung, I.J.: Tutorial on Maximum Likelihood Estimation. Journal of Mathematical Psychology 47(1), 90–100 (Feb 2003)
24. Padala, P., Hou, K.Y., Shin, K.G., Zhu, X., Uysal, M., Wang, Z., Singhal, S., Merchant, A.: Automated control of multiple virtualized resources. In: Proceedings of the 4th ACM European Conference on Computer Systems. pp. 13–26. EuroSys '09, ACM, New York, NY, USA (2009)
25. Park, S.M., Humphrey, M.: Self-tuning virtual machines for predictable escience. In: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid. pp. 356–363. CCGRID '09, IEEE Computer Society, Washington, DC, USA (2009)
26. Rao, J., Bu, X., Xu, C.Z., Wang, L., Yin, G.: Vconf: A reinforcement learning approach to virtual machines auto-configuration. In: Proceedings of the 6th International Conference on Autonomic Computing. pp. 137–146. ICAC '09, ACM, New York, NY, USA (2009)
27. Roy, N., Dubey, A., Gokhale, A.: Efficient autoscaling in the cloud using predictive models for workload forecasting. In: Cloud Computing (CLOUD), 2011 IEEE International Conference on. pp. 500–507 (2011)
28. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representations by error propagation. Tech. rep., DTIC Document (1985)
29. Shen, Z., Subbiah, S., Gu, X., Wilkes, J.: Cloudscale: Elastic resource scaling for multi-tenant cloud systems. In: Proceedings of the 2Nd ACM Symposium on Cloud Computing. pp. 5:1–5:14. SOCC '11, ACM, New York, NY, USA (2011)
30. Tesauro, G., Jong, N.K., Das, R., Bennani, M.N.: A hybrid reinforcement learning approach to autonomic resource allocation. In: Proceedings of the 2006 IEEE International Conference on Autonomic Computing. pp. 65–73. ICAC '06, IEEE Computer Society, Washington, DC, USA (2006)
31. Urgaonkar, B., Shenoy, P., Chandra, A., Goyal, P., Wood, T.: Agile dynamic provisioning of multi-tier internet applications. ACM Trans. Auton. Adapt. Syst. 3(1), 1:1–1:39 (Mar 2008)
32. Villela, D., Pradhan, P., Rubenstein, D.: Provisioning servers in the application tier for e-commerce systems. In: Quality of Service, 2004. IWQOS 2004. Twelfth IEEE International Workshop on. pp. 57–66 (2004)
33. Williams, A.W., Pertet, S.M., Narasimhan, P.: Tiresias: Black-box failure prediction in distributed systems. In: IPDPS. pp. 1–8 (2007)
34. Xu, J., Zhao, M., Fortes, J., Carpenter, R., Yousif, M.: On the use of fuzzy modeling in virtualized data center management. In: Autonomic Computing, 2007. ICAC '07. Fourth International Conference on. pp. 25–25 (2007)
35. Zhang, G., Patuwo, B.E., Hu, M.Y.: Forecasting With Artificial Neural Networks: the State of the Art. International Journal of Forecasting 14(1), 35 – 62 (1998)
36. Zhang, Q., Cherkasova, L., Smirni, E.: A Regression-Based Analytic Model for Dynamic Resource Provisioning of Multi-Tier Applications. In: Proceedings of the Fourth International Conference on Autonomic Computing. pp. 27–. ICAC '07, IEEE Computer Society, Washington, DC, USA (2007)