2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)
Bari, Italy. October 6-9, 2019

A Hybrid Mechanism of Horizontal Auto-scaling Based on Thresholds and Time Series

Paulo Pereira[1], Jean Araujo[2] and Paulo Maciel[3]

***Abstract*— Demand for performance, availability, and reliability in computational systems has increased lately. Improving these aspects is an important research challenge due to the applications and users diversity. In this paper, we propose a hybrid auto-scaling approach that uses reactive and proactive solutions. We focus on the operation of a Web server, where we applied the characteristic of CPU-bound in the application. The approach proposed is a CPU usage monitor and auto-scaling management that is used to improve the system's throughput and reduce the CPU idle time. Our proposal is implemented using triggering thresholds and five forecasting models: Drift, Simple Exponential Smoothing, Holt, Holt-Winters and ARIMA. The main goal of this research is to achieve a better QoS related to the cloud computing environment. One of the obtained results shows that our method represents a throughput improvement of 12.11% by using our proposal, instead of only using a threshold-based technique.**

I. INTRODUCTION

Cloud computing has emerged as a prosperous computing paradigm, allowing customers and companies to count on external providers to process and store their data. Cloud computing encourages an agile service-based computing market by providing easy access to computing resources, it helps to rapidly deploy new services with full access to the World-Wide Web market [1]. Elasticity has an important role in cloud computing environments where resources may be allocated and deallocated according to the demands of the user [2].

The auto-scaling process can be grouped into two classes: reactive and proactive [1]. Reactive algorithms scale (out/in) the environment based on the current state of the system. An important drawback of the reactive approach is neglecting the time of booting up a virtual machine, which may take from 5 to 15 minutes [3]. Neglecting the time of booting up a virtual machine may cause QoS degradation, once the cloud service fails to comply with the QoS requirements of users, which may cause an increase of the cost for the cloud service provider [4]. Proactive scaling approaches are suitable for cloud environments, because they generally have predictable load characteristics, even though they may have unplanned load spikes. Predictive auto-scaling algorithms forecast future demands, so the auto-scaler is able to provide resources with enough anticipation to meet the future needs [3].

[1]Paulo Pereira is with the Centro de Informatica da Universidade Federal de Pernambuco, Recife, Brazil prps@cin.ufpe.br

[2]Jean Araujo with the Unidade Academica de Garanhuns, Universidade Federal Rural de Pernambuco, Garanhuns, Brazil jean.teixeira@ufrpe.br

[3]Paulo Maciel is with the Centro de Informática da Universidade Federal de Pernambuco, Recife, Brazil prmm@cin.ufpe.br

In this paper, we propose a hybrid auto-scaling approach that is based on thresholds and time series forecasting methods to achieve a better system's throughput. Our proposal enables a better CPU utilization because it reduces CPU idle time while increases the system's throughput. It is implemented using triggering thresholds and five well-known forecasting models: Drift, Simple Exponential Smoothing, Holt, Holt-Winters and ARIMA. The objective of this work is to propose a hybrid method of auto-scaling, which can be trained and tested in a real environment to capture the latest trends, in such a way that it will be able to provide sufficiently accurate results for drastically different CPU consumption patterns.

This paper is structured as follows. Section II depicts some related works. The strategy is presented in detail in Section III, comprising the architecture, features and a general view. Section IV presents some experiments to evaluate the proposed strategy and the results that were achieved. Finally, Section V draws some conclusions of the paper and discusses some directions for future works.

II. RELATED WORKS

Lately, several studies are related to forecasting in cloud computing environment. Most of the studies related to forecasting in the cloud computing context are about the auto-scaling mechanism. Concerning related work, we have defined two distinct ways to compare our study with other works in this field of research. First of all, we compare which group the proposals belong (e.g. reactive or proactive). Finally, how many techniques were used in the related work proposals. In this section, we basically describe some of the proposals in this area while we compare to our study.

Ali-Eldin et al. [5] introduced two adaptive hybrid controllers, where one is reactive and the other one is proactive. Their proposal detects the workload growth and also does not deallocate resources prematurely. Their results show that using a reactive controller with a proactive one decreases the probability of the SLA violations ten times compared to a totally reactive auto-scaling mechanism. However, the authors do not consider the delay required for a virtual machine to start up and shut down.

Iqbal et al. [6] proposed a methodology and described a prototype system for automatic identification of over-provisioning in multi-tier applications on cloud computing environments. Their proposal is a hybrid auto-scaling mechanism, where it uses reactive and proactive approach. The authors used response time as their metric. For the proactive approach, the authors chose the regression technique to

978-1-7281-4569-3/19/$31.00 ©2019 IEEE

forecast the future needs of a Web application. According to the authors, it is really hard to capture a resource intensive configuration of a multi-tier Web application that meets a response time requirements for a type of workload.

We also propose a hybrid strategy, but our reactive and predictive approach are both capable of scaling (out/in) the system. Auto-scaling decisions are coordinated between reactive and proactive policies, where the reactive approach is used in case of the proactive action does not forecast accurately.

III. A HYBRID MECHANISM OF HORIZONTAL AUTO-SCALING

The proposed hybrid strategy for virtual machine provisioning consists of two main components: resource utilization monitoring module, and auto-scaling module. The resource utilization monitoring module collects resource usage information of virtual machines in a regular time interval. The auto-scaling module uses the CPU usage history stored in a text file inside the monitoring module to forecast the consumption behavior. The goal of the auto-scaler is to dynamically adjust the assigned resources to elastic applications, depending on the resource usage. Anticipation is a key feature because there will be always a delay from the time when an auto-scaling action is executed until it is effective. In other words, it takes a time to deploy a virtual machine and boot the operating system and application [1].

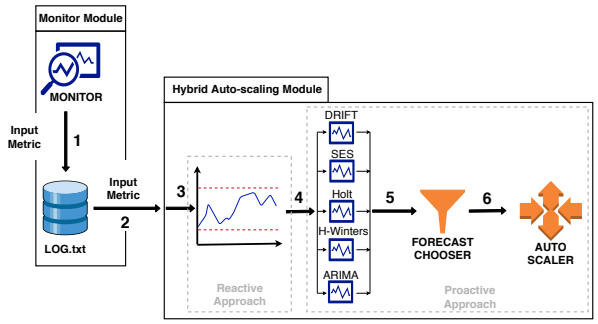


Fig. 1: Information flow diagram.

Figure 1 shows the two main components of the method proposed in this study: the monitor module and the hybrid auto-scaling module. The monitor module consists of a monitor and a file where the input metric for the auto-scaling mechanism is stored. Every time interval, the monitor reads the resource utilization and saves it in the text file (1), which will be read later for the other main module (2). The hybrid auto-scaling module consists of two parts, the reactive and the proactive approach. First the data collected for the monitor module is transformed in a time series, then it is tested by the reactive approach (3). If the resource utilization reaches the up or down threshold, the system is scaled right away by the reactive approach, whereas, the proactive approach starts (4). The proactive approach is composed of five prediction methods, a forecasting methods chooser, and the auto-scaling unit. The five prediction methods are trained and tested (4), and the most suitable one is chosen to predict further resource utilization (5). If necessary, the auto-scaler will scale out or in a virtual machine (6). There are only two situations that will scale (out/in) the system, whether the real resource consumption or the forecast reaches the thresholds.

The auto-scaling mechanism we propose in this paper is completely independent of the service in a scaling scenario. Thus, the type of load balancer and application running in the virtual machines do not matter for our proposal.

A. Monitoring module

The resource monitor component is responsible for monitoring and collecting data about the metrics of the computational, storage, and network resources such as CPU utilization, memory usage, and network traffic.

The auto-scaler needs to monitor performance metrics in order to analyze and determine whether scaling operations need to be taken and how they should be performed. Selection of the right performance metrics is crucial to the success of an auto-scaler. Many factors affects the decision to scale the system or not such as application characteristics, SLA, monitoring cost and the control algorithm itself. The time interval of the monitor determines the sensitivity of an auto-scaler. Nevertheless, if a monitoring interval is too short, it may result computing resources cost and financial cost. It is also likely to cause instability in the auto-scaler. For that reason, it is vital to tune this parameter to reach a balanced performance [1].

It is worth pointing out that every time when the system is scaled out or in, the file text is erased in order to avoid having a biased prediction in the next forecast. Therefore, there is a small cool-down period that helps to ensure that the auto-scaler does not start or finish additional instances before the previous scaling activity takes effect, so the system will have time to stabilize. In other words, the cool-down period avoids the auto-scaler making any changes to the cloud environment for a certain amount of time. The reason behind this procedure is to prevent multiple allocations and releasing of instances while the resource consumption presents high variability, as this would increase the cost of the cloud provider. For this reason, the auto-scaler is only able to change the deployment if sufficient time between scaling process has passed and if the new instances are full-operational. In this study, the cool-down period is as large as the time to collect the number of samples needed as input window for the forecasting techniques.

B. Auto-scaling module

The auto-scaling module processes and analyzes the gathered data about the resource utilization. Our auto-scaling module consists of thresholds and five forecasting methods, which are: Drift, Simple Exponential Smoothing, Holt, Holt-Winters, and ARIMA [2], [4], [7], [8].

The proposed prediction approach works by receiving resource utilization history from virtual machine layer and accumulates all virtual machines' historic usage. We interface the forecasting script with a shell script for real-time prediction of resources.

The forecasting script reads the text file that contains the data about the resource utilization and splits it into training and test sets. The training set is composed of 80%, and the test set is composed of 20% of the total sample [9]. After that, all prediction methods are trained and tested, and the most suitable is chosen as the best method for that prediction, based on MAPE error metric. Later, a new prediction is made with the most suitable method using all samples, specifically, using the whole input window. Then, it is verified whether it is necessary or not to scale (out/in). In other words, the five forecasting techniques are trained and tested, the mean absolute percentage error (MAPE) is calculated for each technique, so the one that has the lowest error is chosen as the best forecasting technique for that specific moment. Figure 2 summarizes the two-step process performed by the forecasting phase. It shows that the input data is split in training and test set.

Usually, the forecast horizon size has the same amount of samples that the test set. Δt is the time interval size collection. It also shows the forecast performed using a chosen method, which is the one defined as the most suitable forecasting model for the moment. The cool-down period is as long as the input window length. Once the system is scaled, the samples collected are discarded. After the cool-down period, there are enough samples to perform the forecasting phase again.To forecast, each forecasting model decomposes the time series in trend, seasonality, and white noise. The Drift model is a simple technique that is based on Naive model, and it captures the trend of the time series by using the average change seen in the past data. The Simple Exponential Smoothing model tries to capture the time series behavior and is suitable for forecasting data with no trend or seasonal pattern. The Holt model is an extension of Simple Exponential Smoothing to allow gathering the trend in the data. The Holt-Winters model extends the Holt method to capture also the seasonality. The ARIMA model tries to capture the auto-correlation among the data. So each forecast technique was chosen to capture several types of workload, which increases the probability to choose a suitable technique for the forecast [10].

The approach proposed in this paper aims to solve the cost-performance trade-off by automatically adjusting application's resources based on its resource utilization. To demonstrate that our method works, we have conducted experiments to compare the system's throughput in several scenarios. Then, based on the experiment results, we are able to confirm that our approach can improve the provided service.

The Algorithm 1 depicts how the forecast is performed in the auto-scaling module. The input of the Algorithm 1 is a time series of CPU consumption that was created by the monitoring module, and the output is the auto-scaling decision. First of all, the auto-scaling module reads the text file that contains the time series, and then it uses the last samples (e.g., the last 90 samples) for training and testing the forecast techniques, in line 2 and 3. The number of samples used to train and test the models is the input window length.

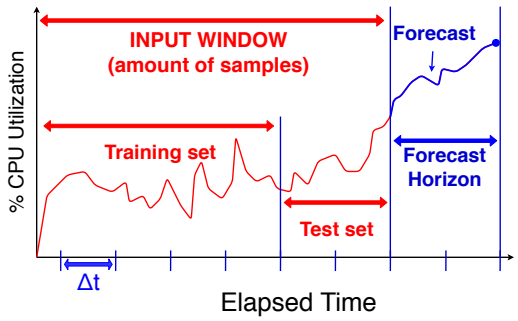


Fig. 2: Splitting into training/test sets and forecasting.

Input: A time series of CPU utilization
Output: Auto-scaling decision

```
1  initialization;
2  timeSeries ← readFile(log.txt);
3  samples ← tail(timeSeries, inputWindowLength);
4  if last(samples) > upThreshold then
5  |  scale(out);
6  else if last(samples) < downThreshold then
7  |  scale(in);
8  else
9  |  traningSet ← split(samples, 80);
10 |  testSet ← split(samples, 20);
11 |  mDrift ← drift(trainingSet);
12 |  mSes ← ses(trainingSet);
13 |  mHolt ← holt(trainingSet);
14 |  mHW ← holtWinter(trainingSet);
15 |  mArima ← arima(trainingSet);
16 |  errorDrift ← mape(mDrift, testSet);
17 |  errorSes ← mape(mSes, testSet);
18 |  errorHolt ← mape(mHolt, testSet);
19 |  errorHW ← mape(mHW, testSet);
20 |  errorArima ← mape(mArima, testSet);
21 |  bestModel ← min(errorDrift, errorSes,
   |    errorHolt, errorHW, errorArima)
22 |  result ← forecast(samples, bestModel,
   |    UpTreshold, DownThreshold);
23 |  scale(result);
24 end
```

Algorithm 1: Auto-scaling forecast

Before starting the forecasting phase, it is tested if the resource consumption already achieved the upper or lower threshold, if it does the algorithm returns the decision to scale (out/in) the system right away, from line 4 to 7. In other words, if the last sample (*last(samples)*) reached any threshold the algorithm returns a result to the auto-scaling controller. The *upThreshold* and *downThreshold* are set by the system administrators according to their necessity.

If it does not reach any threshold, the forecasting phase starts, line 8. First of all, in the forecasting phase, the samples are split into training and test set, where the training set is composed by 80% of the samples and test set composed by

20% [9], in line 9 and 10.

So the 5 forecasting models (Drift, Simple Exponential Smoothing, Holt, Holt-Winters and ARIMA) are trained, where the parameters of each method are chosen according to the time series behavior, all parameters are chosen during the training process of each model, from line 11 to 15. After training all techniques, from line 11 to 15, the models are tested against the test set and the error is calculated, from line 16 to 20. The mean absolute percentage error (MAPE) values are compared to each other, in line 21. All the forecasting techniques used in this paper are statistical ones, which means we are able to disconsider the time to train and test them because it takes less than 0.3 seconds, as we measured.

The model that has the lowest mean absolute percentage error is chosen as the best fit for the workload moment, and then the chosen model is stored in *bestModel* variable. The chosen technique is used to forecast using all samples, in line 22. Finally, *result* if the system needs to be scaled (out/in) is returned, line 23. The *result* can assume three values, which are scale out, scale in or NOP (no operation).

It is worth pointing out that each forecasting technique was chosen according to the related works, and each technique captures some specific feature of the time series. For instance, the Drift model works better with data that have a trend. On the other hand, Simple Exponential Smoothing works better with stationary time series. Holt model captures better the linear trend of the time series. Holt-Winters model captures the seasonality of the data [11]. ARIMA model is more general than exponential models and may capture better patterns in stationary data [9].

The auto-scaling mechanism that we propose in this work does not depend of the service provided. In other words, our proposal is independent of application that is being provided. Besides, it is a hybrid (proactive/reactive) operation, where the proactive behavior analyzes multiple forecasting techniques and elects one technique from time to time to provide estimation about future service demand. Therefore, it allows the auto-scaling to capture the changes in resource consumption.

IV. EXPERIMENTAL RESULTS

In this case study, we aim to verify that our strategy works better than the only reactive one by comparing them to each other. We compared three different workloads and several configurations in our strategy to support that our method presents an improvement in the system throughput. The capacity of a web server is measured in terms of the rate of requests/second that it can fulfill.

There are many available performance metrics that can be used for scaling decisions. In order to reduce the complexity of the monitoring module, we used the CPU utilization as the performance metric. In fact, in provisioning web applications, CPU utilization metric is commonly used to decide whether to trigger scaling actions or not [7], [12], [13].

To evaluate the proposed method, we built an experimental environment and conducted several case studies by generating workload using Apache JMeter™[1] 4.0 benchmark. We also used a switch to connect the physical machines (SWITCH HPE 1420-24G 24P GIGA - JG708B). In this experiment, the target IT system consists of virtual machines managed by XenServer[2] 7.2.0. In our virtual environment, we prepared following types of VM: **Load Balancer VM:** 1 core CPU, 1GB RAM Memory, Linux (Ubuntu 16.04) OS, Nginx 1.12.2; **Web/AP server VM:** 1 core CPU, 1GB RAM Memory, Linux (Ubuntu 16.04) OS, Apache 2.24. These two types of virtual machines are kept in the internal storage of XenServer as templates. When the system scales out, a Web/AP server VM is created from the template, and when the system scales in, a virtual machine is destroyed.

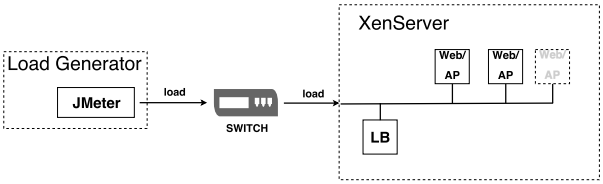


Fig. 3: Experimental environment.

The load balancing mechanism adopted was round-robin, which assigns to each server an equal amount of requests in circular order, handling all servers without priority. In other words, if a system has three web servers and one request comes, it will be sent to the first web server; when the second request comes, it will be sent to the second web server; when the third request comes, it will be sent to the third web server; when the fourth request comes, it will be sent to the first web server, and so on. Figure 3 shows how the experimental environment was set.

TABLE I: Scenarios analyzed in this study.

Scenario	Input Window Length	Forecast Horizon Size
1	30	10
2	30	20
3	30	30
4	60	10
5	60	20
6	60	30
7	90	10
8	90	20
9	90	30
10	reactive threshold-based	

According to our experiments, it takes about 5 minutes to create virtual machines from our templates and make them be recognized by the load balancer. This value is an average of trials measured in our environment. Ten scenarios are assessed in this study, and for each one, we collected a throughput value. Nine scenarios are variations of our

[1]http://jmeter.apache.org/
[2]https://xenserver.org/

proposal (combinations of input window length and forecast horizon size to observe how they affect the system's throughput). A 10th scenario was also generated, representing the purely reactive threshold-based technique. All scenarios can be observed in Table I. Input window length is the number of samples that are analyzed to forecast future demand. On the other hand, forecast horizon size is the number of samples into the future for which forecasts are to be prepared. The amounts of samples chosen for the input window lengths were 30, 60, and 90, and for the forecast, horizon sizes were 10, 20, and 30 [14].

Each scenario represents a combination of input window length and forecast horizon size. For instance: **scenario 1** has an input window length of 30 and forecast horizon size of 10; **scenario 2** has an input window length of 30 and forecast horizon size of 20; **scenario 3** has an input window length of 30 and forecast horizon size of 30; **scenario 4** has an input window length of 60 and forecast horizon size of 10, and so on.

We analyzed each scenario under three distinct workload types, which are shown in the Figure 4. By using these three workloads, we tried to represent a variation between moderate/peak load conditions, peaks that occur periodically, and a more random workload situation. A variation between peaks and moderate load condition represents a constantly increasing and decreasing over time, and by using it, we tried to represent a workload in an e-commerce website, where more requests are made during a period of a day than the night [3], [15]. By using peaks that occur periodically, we tried to represent a load that increases and decreases very quickly. Random workload situation is a special case of workload where there are periodic peaks in a very long timeframe, such as e-commerce websites on Black Friday [3]. These workloads are generated by the JMeter, so they are threads that simulate users. Each user generates HTTP requests fiercely. These three workloads go up to 500 threads (users) that are making requests for 24 hours.

Figure 5 corresponds to the results of our experiments for this threshold settings. It is also worth pointing out that the bars represent the scenarios related to our proposal. The 10th scenario does not have an input window length because it represents the reactive threshold-based technique. So, we used a red line to represent the throughput result for this scenario. Our results suggest that our proposal performs better than the reactive threshold-based technique, which is the most popular auto-scaling method used by commercial cloud providers [1]. Even though we tested several scenarios using settings that may compromise the prediction accuracy, our proposal performed better than the reactive threshold-based technique. Figure 5 shows the results of our ten scenarios under three distinct workloads.

On the other hand, the Workload III has a more random behavior. For this situation, a small window input and forecast horizon are preferable to get a better prediction accuracy when forecasting random behavior of the demand [9]. With that in mind, we are able to observe that our results confirm that. In Figure 5, scenario 9 had the highest throughput for the Workload I and II, and scenario 1 had the highest throughput for the Workload III.

The Table II also summarizes our throughput results (request/second). It is worth pointing out again that the *I* column represents the input window length values and the *H* column represents the horizon size ones. The values in these columns indicate the number of samples used for *I* and *H*. Therefore, each row in the Table II indicates a combination of input window length and forecast horizon size. The bold values in the Table II are the highest throughput results for each workload type, and the underlined values are the lowest ones. As we can observe, there is a considerable difference between our proposal and the purely reactive technique for this threshold setting. Our proposal improved 12.11% the system's throughput for Workload I, 6.46% for Workload II, and 4.93% for Workload III, which represents a better resource usage.

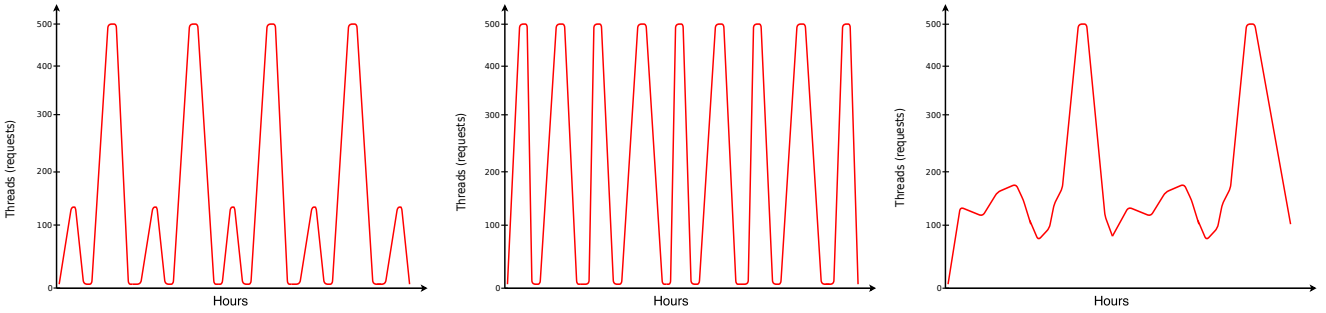
TABLE II: Throughput results (request/s) for 20% and 80% thresholds.

I	H	Workload I	Workload II	Workload III
30	10	2521.74	3382.97	**3484.12**
	20	2463.26	3367.69	3423.76
	30	2531.73	3364.71	3361.19
60	10	2555.04	3419.05	3483.26
	20	2487.95	3382.48	3444.60
	30	2624.50	3329.07	3331.84
90	10	2521.15	3351.58	3466.16
	20	2470.48	3428.18	3464.03
	30	**2754.27**	**3436.00**	3459.01
Reactive	-	<u>2456.95</u>	<u>3227.30</u>	<u>3320.21</u>

According to the result obtained, the reactive threshold-based technique has the lowest throughput values. Otherwise, the hybrid strategy for auto-scaling of virtual machines based on proactive and reactive techniques can considerably improve the throughput of the system, which decreases the probability of QoS violations. Our proposed method provides the best throughput results for several types of workload. Even the combinations of input window length and forecast horizon size do not perform worse than the reactive threshold-based technique, which is the most used method in the commercial auto-scaling systems. Therefore, our results suggest that our proposed method can increase the throughput, which represents a better QoS.

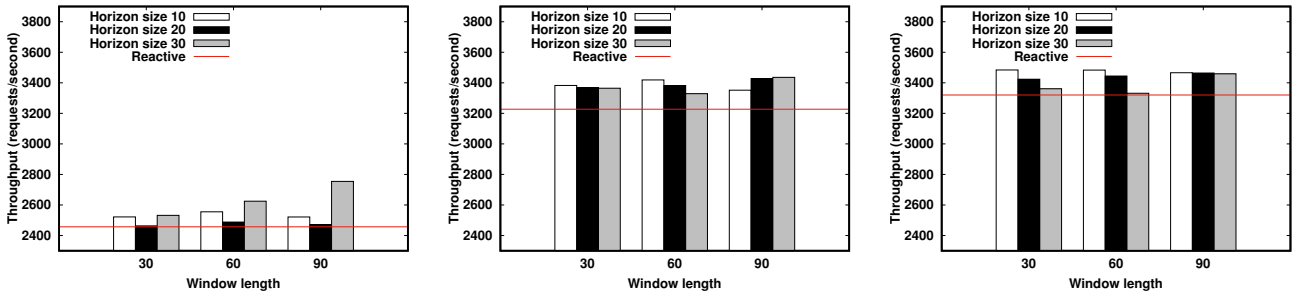
V. CONCLUSIONS

Our approach shows to be a better option than the reactive threshold-based technique, and it can improve the throughput considerably by optimizing the resource utilization. In one of our results, we had a throughput improvement of 12.11% by using our proposal compared to a purely reactive threshold-based technique in our environment. Our proposal works for multi-tenant and multi-application environments because the administrator sets the virtual machines that our proposal will monitor, and if a new virtual machine is instantiated our



(a) Workload I (b) Workload II (c) Workload III

Fig. 4: Types of workload.



(a) Workload I - 10 scenarios (b) Workload II - 10 scenarios (c) Workload III - 10 scenarios

Fig. 5: Throughput results for 20% and 80% thresholds (request/second).

proposal adds it in the list of monitored virtual machines. By using our proposal, the system administrators are able to squeeze extra capacity of their systems.

As future work, we intend to test different upper and lower threshold values to see how it effects the system's throughput. We also intend to use others metrics of resource consumption, such as memory usage, bandwidth, and so on.

ACKNOWLEDGMENT

We would like to thank the Coordination of Improvement of Higher Education Personnel – CAPES, National Council for Scientific and Technological Development – CNPq, and MoDCS Research Group for their support.

REFERENCES

[1] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, "A review of auto-scaling techniques for elastic applications in cloud environments," *Journal of Grid Computing*, vol. 12, no. 4, pp. 559–592, 2014.

[2] R. Shariffdeen, D. Munasinghe, H. Bhathiya, U. Bandara, and H. D. Bandara, "Adaptive workload prediction for proactive auto scaling in paas systems," in *Cloud Computing Technologies and Applications (CloudTech), 2016 2nd Int. Conf. on*. IEEE, 2016, pp. 22–29.

[3] A. Y. Nikravesh, S. A. Ajila, and C.-H. Lung, "An autonomic prediction suite for cloud resource provisioning," *Journal of Cloud Computing*, vol. 6, no. 1, p. 3, 2017.

[4] W. Chen, J. Chen, J. Tang, and L. Wang, "A qos guarantee framework for cloud services based on bayesian prediction," in *Advanced Cloud and Big Data, 2015 Third Int. Conf. on*. IEEE, 2015, pp. 117–124.

[5] A. Ali-Eldin, J. Tordsson, and E. Elmroth, "An adaptive hybrid elasticity controller for cloud infrastructures," in *Network Operations and Management Symposium (NOMS), 2012 IEEE*. IEEE, 2012, pp. 204–212.

[6] W. Iqbal, M. N. Dailey, D. Carrera, and P. Janecek, "Adaptive resource provisioning for read intensive multi-tier applications in the cloud," *Future Generation Computer Systems*, vol. 27, no. 6, pp. 871–879, 2011.

[7] H. Fernandez, G. Pierre, and T. Kielmann, "Autoscaling web applications in heterogeneous cloud infrastructures," in *Cloud Engineering (IC2E), 2014 IEEE International Conference on*. IEEE, 2014, pp. 195–204.

[8] V. A. Farias, F. R. Sousa, and J. C. Machado, "Auto escalonamento proativo para banco de dados em nuvem," *Simpósio Brasileiro de Banco De Dados*, pp. 281–287, 2014.

[9] R. J. Hyndman and G. Athanasopoulos, *Forecasting: principles and practice*. OTexts, 2014.

[10] T. W. Anderson, *The statistical analysis of time series*. John Wiley & Sons, 2011, vol. 19.

[11] P. S. Kalekar, "Time series forecasting using holt-winters exponential smoothing," *Kanwal Rekhi School of Information Technology*, vol. 4329008, pp. 1–13, 2004.

[12] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, and T. Wood, "Agile dynamic provisioning of multi-tier internet applications," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 3, no. 1, p. 1, 2008.

[13] S. Dutta, S. Gera, A. Verma, and B. Viswanathan, "Smartscale: Automatic application scaling in enterprise clouds," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. IEEE, 2012, pp. 221–228.

[14] Y. Shynkevich, T. McGinnity, S. A. Coleman, A. Belatreche, and Y. Li, "Forecasting price movements using technical indicators: Investigating the impact of varying input window length," *Neurocomputing*, vol. 264, pp. 71–88, 2017.

[15] M. Mao and M. Humphrey, "Scaling and scheduling to maximize application performance within budget constraints in cloud workflows," in *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*. IEEE, 2013, pp. 67–78.