Pós-Graduação em Ciência da Computação

Paulo Roberto Pereira da Silva

A Hybrid Strategy for Auto-scaling of VMs: An Approach Based on Time Series and Thresholds

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
http://cin.ufpe.br/~posgraduacao

Recife
2019

Paulo Roberto Pereira da Silva

A Hybrid Strategy for Auto-scaling of VMs: An Approach Based on Time Series and Thresholds

A M.Sc. Dissertation presented to the Center for Informatics of Federal University of Pernambuco in partial fulfillment of the requirements for the degree of Master of Science in Computer Science.

Concentration Area: Performance and Dependability Evaluation
Advisor: Paulo Romero Martins Maciel
Co-Advisor: Jean Carlos Teixeira de Araujo

Recife
2019

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

S586h Silva, Paulo Roberto Pereira da
A hybrid strategy for auto-scaling of VMs: an approach based on time series and thresholds / Paulo Roberto Pereira da Silva. – 2019.
87 f.: il., fig., tab.

Orientador: Paulo Romero Martins Maciel.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2019.
Inclui referências e apêndices.

1. Ciência da computação. 2. Computação em nuvem. I. Maciel, Paulo Romero Martins (orientador). II. Título.

004 CDD (23. ed.) UFPE- MEI 2019-042

PAULO ROBERTO PEREIRA DA SILVA

A Hybrid Strategy for Auto-scaling of VMs: An Approach Based on Time Series and Threshold

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de mestre em Ciência da Computação.

Aprovado em: 13/04/2019.

BANCA EXAMINADORA

_____

Prof. Dr. Paulo Romero Martins Maciel (Orientador)
Centro de Informática /UFPE

_____

Prof. Dr. Kelvin Lopes Dias (Examinador Interno)
Centro de Informática /UFPE

_____

Prof. Dr. Rubens de Souza Matos Júnior (Examinador Externo)
Instituto Federal de Sergipe /IFS

I dedicate this work to my mother who always bent over backward to make me get to where I am today, and to my father for all the support he always gave me during the difficulties.

ACKNOWLEDGEMENTS

Thanks God for allowing me to reach this moment, giving me opportunities, health and strength to overcome adversities. I would like to thank my mother that bent over backward to make me a successful person, she is my inspiration. **Carla Regina Pereira da Silva** is and will always be my hero. She is the greatest warrior that I ever met and I really proud of having her as my mother. Today if I see further it is because I stand on the shoulders of a giant, and this giant it is my mother for sure. I would like to thank my father as well, great man that give me all psychological and financial help to accomplish it. I am so grateful for having these two people as parents.

I would also like to thank my lab friends (MoDCS group), which always tried to help me and responded to all my questions without hesitation or delays. They also built a really nice learning environment, where everyone is more than welcomed.

My roommate Tullyo for being such a nice person. I would also like to thank my Professor, co-advisor, great person and friend Jean Teixeira for all support, all partnership, all psychological and financial help.

I am really grateful to Paulo Maciel for opening the doors of the CIn-UFPE to me. If someone asked me three years ago if I would be qualified to study there, I would totally say no. Therefore, I am really grateful for this opportunity. Paulo Maciel is also a hardworking Professor and a great inspiration for all his students.

To finish, I would like to thank the financial support from the funding agency CAPES, which was essential for concluding this stage of my life.

"The world is a book and those who do not travel read only one page."
(BONNER, 1963, p. 47)

ABSTRACT

Demand for performance, availability, and reliability in computational systems has increased lately. Improving these aspects is an important research challenge due to the wide diversity of applications and customers. The elasticity of cloud computing applications plays an important role, and its implementation is based on auto-scaling techniques, which allow to increase and decrease cloud-based application's capacity. Therefore, it is possible to deal with the workload variation and not interrupt the service. The auto-scaling process can be grouped into two classes: reactive and proactive. In this dissertation, we propose a hybrid auto-scaling approach that uses reactive and proactive solutions. Our proposal is implemented using triggering thresholds and five forecasting models: Drift, Simple Exponential Smoothing, Holt, Holt-Winters and ARIMA. It checks if the CPU utilization has achieved the threshold, if it has, the system is scaled (out/in). On the other hand, if the threshold was not achieved the forecasting phase starts. In this phase, the five forecasting models are trained and tested, so the most suitable one is used to forecast resource utilization. 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.

Keywords: Cloud Computing. Auto-scaling. Elasticity. Time Series. Forecast.

RESUMO

A demanda por desempenho, disponibilidade e confiabilidade em sistemas computacionais tem aumentado bastante ultimamente. Melhorar esses aspectos é um importante desafio de pesquisa devido à grande diversidade de aplicações e usuários. A elasticidade dos aplicativos de computação em nuvem desempenha um papel importante e sua implementação é baseada em técnicas de autoescalonamento, as quais permitem aumentar e diminuir a capacidade da aplicação baseada em nuvem. Portanto, é possível lidar com a variação da carga de trabalho sem interromper o serviço. O processo de autoescalonamento pode ser agrupado em duas classes: reativo e proativo. Nesta dissertação, propomos uma abordagem híbrida de autoescalonamento que utiliza soluções reativas e proativas. Nossa proposta é implementada usando modelo reativo baseado em limiares e cinco modelos de previsão: Drift, Simple Exponential Smoothing, Holt, Holt-Winters e ARIMA. De forma que as técnicas verificam se o consumo de CPU atingiu um limiar, em caso positivo o sistema é escalonado. Por outro lado, se o limiar não foi alcançado, a fase de previsão é iniciada. Nesta fase, os cinco modelos de previsão são treinados e testados, então o mais adequado para o momento é usado para prever o consumo do recurso computacional. O principal objetivo desta pesquisa é alcançar uma melhor QoS relacionada ao ambiente de computação em nuvem. Em um dos resultados obtidos é possível notar que nosso método híbrido representa uma melhoria na vazão do sistema de 12,11% , em relação à técnica baseada nas técnicas puramente reativas.

Palavras-chaves: Computação na Nuvem. Autoescalonamento. Elasticidade. Séries Temporais. Predição.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

The management of computing resources is important for any system, mostly because of the huge demand for performance, availability, and reliability. For personal computers, bad customer service just causes dissatisfaction and annoyance. However, for systems that are provided as services in cloud computing environments, bad customer service may lead to monetary and credibility losses for companies. Good management has huge importance in the service effectiveness, mostly due to volatile demand of those systems.

The demand for high-performance computing has transformed the shape of today's computer industry. The computing is no more limited to personal computers and workstations. It has now become a public grid, where users (personal computers, cell phones, workstations, and servers) can have access to storage and computing resources through the internet. The environment where users can have access to a distant infrastructure, platform, or software over the internet is termed as cloud computing (ROY et al., 2011). Cloud computing requires high performance and efficient underlying microarchitectures so as millions of customers (users) simultaneously can access the available resources (storage, computing, etc.) on the cloud.

Cloud computing provides on-demand access to shared computing resources and services, such as network infrastructure, storage, operating systems, and applications. Such resources and mechanisms can be easily acquired and released with minimal management effort (MELL; GRANCE et al., 2011). These features enable administrators to focus only on the business model, without worrying about infrastructure details (MELL; GRANCE et al., 2011; BANKOLE; AJILA et al., 2013). The experience in acquiring cloud services is often compared to the consumption of public utilities, such as electricity, so the user just pays for that service without worrying or knowing about the infrastructure that provides it (BAUER; ADAMS et al., 2012).

Cloud computing also allows organizations and users to rely on external providers to store and process their data. Cloud computing also has emerged as a successful computing paradigm because it encourages an agile service-based computing market by providing easy access to computing resources (ROY et al., 2011). Cloud computing helps to rapidly deploy new services with full access to the Internet (DUTREILH et al., 2010). The resources available in cloud computing environments seem to be unlimited for the user, and it can be purchased at any time and in any amount. Automatic elasticity is an essential requirement of current cloud platforms, and it is achieved by using auto-scaling techniques. To take advantage of this, companies are increasingly using the cloud computing environment to host their services (MESSIAS et al., 2016).

For some time now, more and more companies are adopting the infrastructure-as-a-service (IaaS) deployment model of cloud computing as their primary way of provisioning

computing capacity. This can be either based on a private IaaS deployment model, where the cloud paradigm is used to reduce the costs of running the IT infrastructure by exploiting economy of scales in the in-house infrastructure, or based on a public IaaS deployment model, where capacity is acquired on-demand from external IaaS providers, which normally charge the usage of their infrastructure following a pay-as-you-go pricing model. Many of the services that need to be migrated to the cloud are critical to the business of the companies that run them and present time-varying demands. Indeed, elasticity is one of the main advantages offered by IaaS providers to their users. An optimal strategy to run services with time-varying demands over IaaS resources would be one that could allocate, at any moment in time, exactly the capacity that is needed to keep the services running with the required performance, i.e. satisfying their service level agreements (SLAs). Such approach would automatically scale the capacity provisioned, allocating more resources when demand increases, and releasing resources as soon as they are no longer needed.

Usually, administrators of web servers deal with the challenge of dynamically adapting to workload fluctuations, which may require more or fewer computational resources. For instance, social networking and e-commerce might face a burst of high load, which may compromise the performance of the service. On the other hand, low loads may under use the system, which causes a waste of resources (CAMPOS et al., 2015). Allocating resources efficiently is a very complex task, which is generally performed by an auto-scaler mechanism.

Resource scaling can be either horizontal or vertical. In cloud computing environments, horizontal scaling means that the system is scaled by adding more virtual machines into the pool of resources, whereas vertical scaling means that you scale by adding more power (e.g., CPU, network bandwidth and memory) to an existing virtual machine (LORIDO-BOTRAN et al., 2014). In the most common operating systems, it is not possible to change the power features of the system on-the-fly. Consequently, most cloud providers only offer horizontal scaling (LORIDO-BOTRAN et al., 2014).

1.1 MOTIVATION AND JUSTIFICATION

One of the biggest challenges in web service is to dynamically adapt to workload fluctuations, which sometimes require more or fewer infrastructure resources such as storage, processing, memory, and bandwidth. For instance, banking, social networking, e-commerce, and access to online games are examples of applications that might receive sudden surges in traffic. This sudden workload may compromise the performance of the service. The inability to deal with workload peaks might also cause service interruption in some cases, resulting in customer dissatisfaction, and financial damage. On the other hand, very low traffic may underuse resources. It is complex to predict the exact moment that these sudden events happen and reallocates resources efficiently (MARTIN; FREI et al., 2003; SHEN; HUANG et al., 2008; BISWAS et al., 2015).

Usually, the auto-scaling mechanism monitors the application and compares thresholds for instantiation and destruction of virtual machines (VMs). However, the definition of these thresholds is a very complex task, since its definition is related to the configuration of several other parameters, contracts, and workload (BISWAS et al., 2015). Therefore, one of the motivations of this study is due to the difficulty of identifying which values should be used in the configuration parameters of the system in the cloud with auto-scaling. Another characteristic that was considered in this study is: the thresholds should take into account that a VM does not become operational at the moment that it is requested, as it will take time to create a VM and start the operating system and the application. Therefore, anticipating the creation of a virtual machine is another motivation.

An auto-scaling process is usually grouped into two classes: reactive and proactive (LORIDO-BOTRAN et al., 2014). Reactive technique scales the system based on the current situation of a monitored metric (e.g., CPU usage, network bandwidth, memory, and workload), and it chooses to allocate or release resources when the monitored metric exceeds a threshold. The most significant drawback of reactive approach is neglecting the time of booting up a virtual machine, which may take from 5 to 15 minutes (NIKRAVESH et al., 2017). In other words, the time to react of reactive approach is generally insufficient, which may overload the system (MESSIAS et al., 2016). It may cause QoS degradation since a virtual machine will not be ready to receive requests when the system needs. On the other hand, the proactive scaling approach is suitable for cloud environments, mostly because these environments usually have predictable load characteristics.

Unplanned load spikes can also be fitted by proactive auto-scaling techniques (NIKRAVESH et al., 2017). Proactive auto-scaling algorithms are used to forecast future demands in order to arrange resources provisioning with enough anticipation, which decreases the probability of QoS violations (NIKRAVESH et al., 2017).

1.2 OBJECTIVES

An efficient instantiation of VMs is an important requirement for cloud-based service providers. We propose a hybrid auto-scaling strategy that is based on thresholds and time series forecasting methods. As none of these forecasting methods is the best in all cases, our proposal is implemented using triggering thresholds and five well-known forecasting models: Drift, Simple Exponential Smoothing, Holt, Holt-Winters and ARIMA, where the most suitable method for the situation is used to forecast the future demand. Our proposal aims to increase the throughput of cloud providers by improving the auto-scaling process. The focus of this work is CPU-bound characteristic, so we used a Web application because it presents this characteristic. This study aims to enable a better CPU utilization by reducing CPU idle time. Our proposal checks if the CPU utilization has achieved the threshold if the threshold was achieved the system is scaled. On the other hand, if the

threshold was not achieved the forecasting phase starts. In this phase, the five forecasting models are trained and tested, so the most suitable one is used to forecast the demand.

The goal of this work is to capture the latest trends to provide accurate forecasts for different CPU consumption patterns. We propose a hybrid method of auto-scaling that uses reactive and proactive techniques and can be trained and tested in runtime providing forecasts to an auto-scaling mechanism. This study also evaluates the performance of the auto-scaling process for cloud environment hosting a web service application that must deal with changes in workload intensity.

In other words, there are some specific objectives, described as follows:

- Proposing an auto-scaling strategy that is based on thresholds and time series forecasting methods;
- Investigating the effects of configuration factors on the performance of the auto-scaling mechanism;

1.3 STRUCTURE OF THE DISSERTATION

This work is structured as follows. Chapter 2 introduces basic concepts of cloud computing and forecasting that are fundamental to understand this dissertation. Chapter 3 depicts some related works. The strategy is presented in detail in Chapter 4, comprising the architecture, features and a general view. Chapter 5 presents some experiments to evaluate the proposed strategy and the results that were achieved. Finally, Chapter 6 draws some conclusions of the dissertation and discusses some directions for future works.

2 BACKGROUND

Forecasting is used for several fields of knowledge, such as automobilism, meteorology, stock exchange, agronomy, healthy, and more. Cloud computing has various aspects of research, such as middleware design, security, architecture, and so forth. This Chapter presents a background to the main components that we used to build our strategy. Our study is mainly about resource management optimization based on a hybrid approach using threshold and prediction methods, so in this Chapter, we give a brief explanation of cloud computing and time series. We also explain briefly the paired difference test, which we used to statistically verify if our strategy represents an improvement in the auto-scaling mechanism.

2.1 CLOUD COMPUTING: AN OVERVIEW

Cloud computing is a combination of several different technologies, and it is basically a distributed system consisting of a collection of virtualized computers that are automatically provisioned (PETER; TIMOTHY et al., 2011). Cloud computing also provides processing power, storage and network as services. This model allows users to obtain resources on-demand, so the users are able to pay only for what and when they need (ARMBRUST et al., 2010). It has several features that make cloud computing environments very attractive for many companies. New companies are able to rent several virtual machines and provide their web-based service, without needing to keep their very own infrastructure (ZHANG et al., 2010). This model allows resource allocation automatically or manually, by adopting the elasticity concept. Elasticity is one of the most important features in cloud computing environments, mostly because it allocates and releases resources according to user's demands (ARMBRUST et al., 2010).

According to The National of Standards and Technology (NIST) of USA, cloud computing has five essential traits (MELL; GRANCE et al., 2011):

- On-demand self-service allows the users to increase or decrease resources via a Web Browser. In other words, users are able to supply their cloud-based application with infrastructure, development tools, software, and other resources, without going through a service provider.

- Broadband access allows users to access their cloud-based application resources (e.g., storage, processing, memory, and network bandwidth) through different platforms, even through the Internet.

- Resource pooling provides a high-level abstraction of several physical servers. These servers are combined to be in such a way that the customer does not have control or knowledge over the exact location of the resources.
- Elasticity allows the cloud resources to be dynamically allocated or released in response to sudden changes in the cloud demand.
- Measured service provides control over resources by tracking their consumption. It also allows cloud providers to offer a pay-as-you-go model.

Basically, cloud computing is a paradigm that computational resources are available as services on Internet, and they can be remotely accessed through the World Wide Web or a private network (ARAUJO et al., 2011). Cloud computing offers several types of transparency, such as access transparency, location transparency, performance transparency, and migration transparency. Access transparency occurs when the users do not know how the files are distributed in terms of specific servers or physical location, which means that the files are available and accessible whenever the users need. Location transparency means that the users do not know where the files are in terms of physical location, the files could be anywhere in the globe. Performance transparency means that the system can be reconfigured to improve performance, but the user is not able to notice it. Migration transparency occurs when the information or service provided is migrated or moved from one physical server to another, and the user is unaware that is happening some change in the information or service location (ZHAO et al., 2009).

Cloud computing has an important role to spread access to computational resources, offering several advantages such as elasticity, security, availability, and reliability. Cloud computing environments can be divided into four groups: private clouds, community clouds, public clouds, and hybrid clouds (ZHAO et al., 2009), each group has its advantages and drawbacks.

- Private clouds are internal clouds that are only accessible by a single company, therefore providing that company with greater control and privacy. In other words, a private cloud is created and maintained by an individual company, which may use resources and infrastructure already present in an organization or a new separate infrastructure (MELL; GRANCE et al., 2011).
- Community clouds are provided for exclusive use of a specific community of consumers, where the resources are shared with more than one organization that have shared concerns (e.g., mission, security requirements, policy, and compliance considerations). Community clouds are managed by one or more organization in the community, a third party or even some combination of them (MELL; GRANCE et al., 2011).

- Public clouds are independent of the company that uses them, and they are third-party provided. Public clouds provide services to multiple clients using the same shared infrastructure. Hybrid clouds are a mix of private and public clouds (MELL; GRANCE et al., 2011).

- Hybrid cloud gives businesses greater flexibility and more data deployment options by allowing workloads to move between private and public clouds (KIRAN et al., 2015).

Cloud computing can be divided into three services models, according to The National of Standards and Technology (NIST) of USA: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) (PETER; TIMOTHY et al., 2011). Figure 1 depicts the cloud computing stack.

- Software as a Service is the idea that the applications are hosted in a platform and infrastructure, where they are accessible from various user devices through the Internet using some user interface, such as web browser. The user just controls limited user-specific application settings, and the users usually only pay for what they use. SaaS provider is the one responsible to host and manage the applications in their own data center. The user does not have access to the underlying hardware, operating system, and any other software that supports the application.

- Platform as a Service is the idea that the frameworks for developers are hosted in the cloud environment, where the developers can build or customize cloud-based applications. These frameworks may be development tools, middleware, database management systems, programming libraries, and so on. PaaS supports the development in all phases of cloud-based applications. For instance, it supports the building, testing, deploying, managing and updating phases (PETER; TIMOTHY et al., 2011).

- Infrastructure as a Service is a form of delivering of hardware (storage, CPU, memory, network) and software (operating system, hypervisor, file system) as a service. Using IaaS, the client can allocate storage, networks, and many other fundamental computing resources. The clients are able to use any operating system that they want to and build their own environment.

Scaling mechanisms are generally divided into two groups, vertical and horizontal (LORIDO-BOTRAN et al., 2014). By using vertical scaling mechanism, the system is scaled up or down by increasing or decreasing the virtual machine capacity, such as changing CPU or memory amount in a single virtual machine of the system. According to Dutta et al. (DUTTA et al., 2012), vertical scaling can increase or decrease the CPU and memory on the physical server, however, where a general Operating System is running, the server

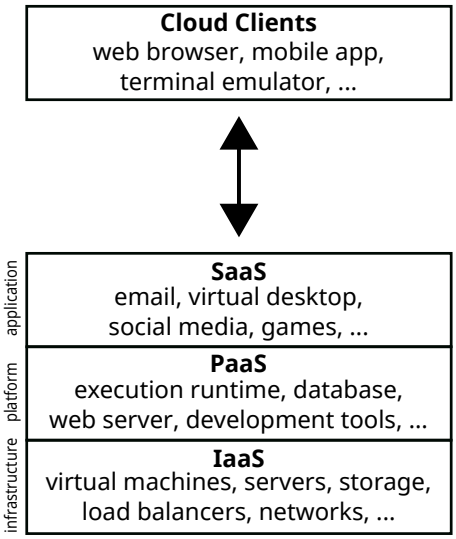


Figure 1 – Cloud computing stack.

has to be stopped. By using horizontal scaling mechanism, the system is scaled out or in by adding or removing virtual machines. A large-scale Web service generally uses several virtual machines running Web servers and load balancer to distributing the requests among the virtual machines. This kind of system is able to scale by adding or removing virtual machines without stopping the system (HIRASHIMA et al., 2016).

Auto-scalers are grouped into two classes: reactive and proactive (LORIDO-BOTRAN et al., 2014). These two classes are related to the auto-scalers anticipation capabilities. Reactive implies that the system reacts only when a change has been detected. This approach uses the last values obtained from the system's monitor. However, a resource provisioning takes some time to be effective, which may cause some issues because the desired effects may arrive too late. The time to react is often insufficient, which may overload the system, and that is the biggest problem in reactive techniques (MESSIAS et al., 2016). On the other hand, the proactive approach tries to forecast future demand. It takes the predictions into consideration to anticipate the resource provisioning, which may accelerate the desired effects in the cloud computing environment. The proactive approach is basically based on time series techniques. The drawback of the proactive technique is that it relies totally on how accurate the forecast is.

Some auto-scalers just respond to current system status, they are purely reactive. Threshold-based rules are a purely reactive auto-scaling technique, and it is the most used by commercial cloud providers (NIKRAVESH et al., 2017). The auto-scaler is triggered based on performance metrics and predefined thresholds.

2.2 TIME SERIES

Basically, there are three types of forecasting models, qualitative techniques, time series, and causal models (CHAMBERS et al., 1971). The qualitative techniques use qualitative data, as the name suggests, and they are very useful when there are no data or there are just a few. The time series, which is the model used by our proposal, try to forecast using patterns and the changes that occur in the pattern. The causal models focus on elements relationship (CHAMBERS et al., 1971).

Time series is generally used to represent the change of a measurement over time (LORIDO-BOTRAN et al., 2014). A time series is a collection of observations obtained over time intervals equally spaced (SHUMWAY; STOFFER et al., 2010). Time series analysis involves developing or using models that best describe an observed data behavior. It happens in order to understand the underlying causes. This process also involves making assumptions about the form of the data and decomposing the time series into constitution components. Predictions about the future are also called extrapolation, which involves taking models to fit on historical data and using them to predict future observations (SHUMWAY; STOFFER et al., 2010). Time series methods are used to detect patterns and anticipate the future values, and their forecast accuracy relies on selecting the right technique and setting the parameters correctly, such as the input window length and forecast horizon size (LORIDO-BOTRAN et al., 2014).

Time series techniques are mostly based on statistical models. None of time series techniques is the best one in all situations. So it is usually necessary to test more than one time series technique (LORIDO-BOTRAN et al., 2014).

A time series may be composed of three components, namely trend, seasonality, and white noise (ANDERSON et al., 2011). A trend is related to an existence of an increase or decrease in data behavior, while seasonality is related to the existence of patterns in seasonal factors, such as the month, a day of the week, or year (HYNDMAN; ATHANASOPOULOS et al., 2014). White noise is an element that is neither a trend nor seasonality because it has a random behavior (WEI et al., 1994). A time series with trend or seasonality is non-stationary because trend and seasonality will always affect the time series in different moments (HYNDMAN; ATHANASOPOULOS et al., 2014; WEI et al., 1994). In this dissertation,, we use a set of time series techniques: Drift, Simple Exponential Smoothing, Holt, Holt-Winters, and ARIMA.

The *Drift* method allows predictions to increase and decrease over time, and the variation is defined as the average change seen in the past data. It is equivalent to drawing a line from the first to last observation and extending it to the future (HYNDMAN; ATHANASOPOULOS et al., 2014). Thus, the forecast value for the moment $T + h$ is described in the following equation.

$$y_{T+h} = y_T + h\left(\frac{y_T - y_1}{T - 1}\right) \tag{2.1}$$

where, T is the size of time series, y_T is the last observation value, y_1 is the first observation value, and h is the forecast horizon.

Exponential smoothing is a big class of forecast models using time series (GARDNER et al., 1993). Naturally, the simplest technique in this class is the simple exponential smoothing (SES), which searches for patterns in the historical data and tries to map the data behavior, after that the SES builds a forecast (KALEKAR et al., 2004). The Simple Exponential Smoothing is suitable for forecasting data with no clear trend or seasonal pattern. Simple Exponential Smoothing needs to select the values of α and ℓ_0. The α is used to decrease the random variation (white noise) from historical data, it also represents the weighting applied to the most recent samples in the time series. This allows to better identify patterns and levels that can be used to estimate future demand. On the other hand, ℓ_0 is the level (or the smoothed value) of the series at time 1, which means that is the first sample of the time series after applied the weight calculated using α. All forecasts can be computed from the data once we know those values. For the methods that follow there is usually more than one smoothing parameter and more than one initial component to be chosen. In some cases, the smoothing parameters may be chosen in a subjective manner, the forecaster specifies the value of the smoothing parameters based on previous experience. However, a more reliable and objective way to obtain values for the unknown parameters is to estimate them from the observed data. We estimated the unknown parameters and the initial values for any exponential smoothing method by minimizing the sum of squared errors (SSE) (HYNDMAN; ATHANASOPOULOS et al., 2014). The Equation 2.2 describes this model.

$$\hat{y}_{T+1|T} = \sum_{j=0}^{T-1} \alpha(1-\alpha)^j y_{T-j} + (1-\alpha)^T \ell_0 \tag{2.2}$$

where, α is the smoothing coefficient, y_{T-j} is the observed smoothed value at the moment T - j, ℓ_0 is the level value, $\hat{y}_{T+1|T}$ is the forecast value.

Holt method was created by Charles C. Holt, and it extends the simple exponential smoothing to allow gathering the trend in the data by inserting a new component (HYNDMAN; ATHANASOPOULOS et al., 2014). The Holt method is based on an exponentially weighted moving average, which is a mean of smoothing random fluctuations that has the following desirable properties: declining weight is put on older data, it is extremely easy to compute, and minimum data is required. The Holt method is clearly a sensible mode of behavior in dealing with simple forecasting problem. The unknown parameters of Holt methods are also estimated by the sum of squared errors. The Equation 2.3 describes the Holt's method, which is composed of two components. The first one is responsible for smoothing the time series, and the second one is responsible for detecting the trend.

$$\hat{y}_{T+1|T} = \alpha y_t + (1-\alpha)(\ell_{t-1} + \beta) + h[\beta(\ell_t - \ell_{t-1}) + (1-\beta)] \tag{2.3}$$

where, ℓ_{t-1} is the smoothing component at the moment *t - 1*, h is the forecast horizon, β is the smoothing parameter for trend, $0 \leq \beta \leq 1$, α is the smoothing parameter, $0 \leq \alpha \leq 1$, $\hat{y}_{T+1|T}$ is the forecast value (HYNDMAN; ATHANASOPOULOS et al., 2014).

Holt-Winters model is an extension of Holt's method to capture the seasonality by adding a new component. This method was created by Holt and Winter (KALEKAR et al., 2004). The following equation describes the Holt-Winters model.

$$\hat{y}_{t+h|t} = \ell_t + hb_t + s_{t+h-m(k+1)} \tag{2.4}$$

where, k is the integer part of *(h-1)/m*, which ensures the estimates of the seasonal indices used for forecasting come from the final sample. The ℓ_t is the level factor, and it shows the weighted average between the seasonality and the the non-seasonal forecast in the moment *t*. α is the smoothing coefficient, β is the trend coefficient, s is the seasonality coefficient, and the seasonal component will sum up to approximately m, which means the amount of seasonal pattern is in the time series (HOLT et al., 2004).

Simple Exponential Smoothing, Holt, and Holt-Winters belong to the same family of exponential models, and their parameters can be either chosen by estimating the parameters by minimizing the sum of squared errors or by maximizing the likelihood. The likelihood is the probability of the data arising from the specified model, consequently, a large likelihood is associated with an accurate model (HYNDMAN; ATHANASOPOULOS et al., 2014).

ARIMA model provides a different approach to time series forecasting. The exponential smoothing methods and the ARIMA model are the most used approaches for time series forecasting (HYNDMAN; ATHANASOPOULOS et al., 2014). While the exponential smoothing methods are based on data trend and seasonality descriptions, the ARIMA model tries to capture the auto-correlation among the data (CORTEZ et al., 2012). The ARIMA is an auto-regressive and moving average combination (PEARCE et al., 1987). The following equation describes the ARIMA model.

$$y'_t = c + \phi_1 y'_{t-1} + ... + \phi_p y'_{t-p} + \theta_1 e_{t-1} + ... + \theta_q e_{t-q} + e_t \tag{2.5}$$

where, y'_t is a differential series that may be integrated more than once, ϕ is the regression coefficient, θ is the moving average coefficient and e is the error. The ARIMA is composed by three values *(p, d, q)*, where p is the auto-regressive degree, d means how many times it will be integrated, q is the moving average parameter (PEARCE et al., 1987).

It is necessary for a time series to be transformed into a stationary time series. Let y_t represent the data sample at time t and then after a time interval τ let the next data sample be $y_{t+\tau}$. Thus the mean and variance of y_t and $y_{t+\tau}$ must be constant and independent of t and the autocovariance between y_t and $y_{t+\tau}$ should only be influenced by τ for the series to be stationary. For this purpose, ARIMA differentiates the original series until the stationary time series is achieved and constitutes d parameter of ARIMA(p, d, q). The

p and q values of ARIMA(p, d, q) can be determined by analyzing partial autocorrelation and autocorrelation plots of the time series data, respectively. During the model fitting process, selecting a high order of p and q will result in a very small amount of white noise variance. The prediction errors of such a model will be large due to overfitting (HYNDMAN; ATHANASOPOULOS et al., 2014).

The parameters estimation errors will be large for high-order model, so it is necessary to introduce some penalty factor to avoid fitting sample data to high-order models. Based on penalty factor, many criteria have been proposed in literature; widely used criteria are the combination of AIC and BIC (PEARCE et al., 1987). The AIC statistic is defined as:

$$AIC(\beta) = -2lnL_x(\hat{\beta}, \hat{\sigma}^2) + 2(p + q + 1) \tag{2.6}$$

where β is the coefficient vector and σ^2 is the white noise variance; those values of p and q for the fitted model are selected, which minimize AIC(β). Also $L_x(\hat{\beta}, \hat{\sigma}^2)$ is the maximum likelihood function, where $\hat{\sigma}$ and $\hat{\beta}$ are likelihood estimators of parameters β and σ which maximize L for given data set X.

The AIC statistic has tendency towards overfitting the model order, which can be corrected by using the BIC statistic (PEARCE et al., 1987). BIC statistic is defined as:

$$BIC = (n - p - q)ln\left(\frac{n\hat{\sigma}^2}{n - p - q}\right) + n(1 + ln\sqrt{2\pi}) + (p + q)ln\left(\frac{\sum_{t=1}^{n} X_i^2 - n\hat{\sigma}^2}{p + q}\right) \tag{2.7}$$

where n represents the number of data samples, X is the data set, and $\hat{\sigma}^2$ is the maximum likelihood estimate of the white noise variance.

The time series is first converted into stationary time series by differentiation, which constitutes parameter d of ARIMA process. Then, from Equation 2.6 and Equation 2.7, those values of p and q are selected, which minimize AIC and BIC statistics.

It is used one error metric to choose the most suitable technique, which is the mean absolute percentage error. This metric was chosen for its efficiency (LAWRENCE et al., 2000), and it is described in the following equation:

$$MAPE = \frac{100}{n}\sum_{i=1}^{n}\left|\frac{A_i - F_i}{A_i}\right| \tag{2.8}$$

where A_i is the actual value, F_i is the forecast value, and n is the amount of fitted points.

2.3 PAIRED DIFFERENCE TEST

A paired difference test is used to compare two population means where you have two samples in which observations in one sample can be paired with observations in the other sample. In other words, this test consists of carrying out more than one measurement in the same sample unit and verifying if there was a difference between these measurements,

where the first information will be paired with the second information, with the third information and so on. For instance, before-and-after observations on the same subjects (e.g. students' diagnostic test results before and after a particular module or course), or comparison of two different methods of measurement or two different treatments where the measurements/treatments are applied to the same subjects. Therefore, it is expected that the measurements of the same individual will be similar, while the measurements of different individuals will be different.

Suppose a sample of n computers with the same configurations, where each one runs a load balancer. We have two load balancing mechanisms (e.g. round-robin and least-connected). Each load balancer is configured to perform the request distribution using one type of load balancing mechanism. We want to find out if, in general, which algorithm leads to improvements the response time of the system. We can use the results from our sample of computers to draw conclusions about the impact of the algorithms in general.

Let x be system's response time using round-robin load balancing mechanism and y be system's response time using least-connected mechanism. To test the null hypothesis that the true mean difference is zero, the procedure is as follows:

- Calculate the difference ($d_i = y_i - x_i$) between the two observations on each pair, making sure you distinguish between positive and negative differences.
- Calculate the mean difference, $\bar{d}$.
- Calculate the standard deviation of the differences, S_d, and use this to calculate the standard error of the mean difference $SE(\bar{d}) = \frac{S_d}{\sqrt{n}}$.
- Calculate the t-statistic, which is given by $T = \frac{\bar{d}}{SE(\bar{d})}$. Under the null hypothesis, this statistic follows a t-distribution with $n - 1$ degrees of freedom.
- Use tables of the t-distribution to compare your value for T to the $t_n - 1$ distribution. This will give the p-value for the paired t-test.

These differences give us a new distribution of values that has its own sample statistics. We took a 95% confidence interval for mean, then we compared against the null hypothesis.

The null hypothesis states that the mean is equal to zero, in other words, it states that there is no difference between some population and other population. On the other hand, the alternative hypothesis states that there is a difference between two populations. The population mean difference is represented by μ_d. So the following confidence interval formula for μ_d is derived assuming that the population of differences has a normal distribution, Equation 2.9.

$$\bar{d} - t_{\frac{\alpha}{2}} \frac{S_d}{\sqrt{n}} < \mu_d < \bar{d} + t_{\frac{\alpha}{2}} \frac{S_d}{\sqrt{n}} \tag{2.9}$$

If there are n pairs, there are $n-1$ degrees of freedom, and the area between $-t_{\frac{\alpha}{2}}\frac{S_d}{\sqrt{n}}$ and $t_{\frac{\alpha}{2}}\frac{S_d}{\sqrt{n}}$ is the confidence level $1-\alpha$, that we chose to be 95%. The most important feature of the paired difference test is that the assumption of normality refers to the distribution of the differences, not the original data themselves. Therefore, when we run or calculate our descriptive statistics we are only concerned with the 1 sample of d_i's (HIGGINS et al., 2003).

It is worth pointing out that for this test to be valid the differences only need to be approximately normally distributed. Therefore, it would not be advisable to use a paired t-test where there were any extreme outliers. Otherwise, it may be necessary to perform the experiments again to try to approximate of the normal distribution.

2.4 FINAL REMARKS

This chapter provided theoretical foundations that are not exhaustive but essential for the reader awareness regarding the building blocks that compose this dissertation. The background on cloud computing, auto-scaling, as well as time series allows understanding the proposal of this dissertation. We also provided an explanation in the paired difference test, which will be used further in a case study to support that our proposal represents an improvement in the auto-scaling mechanism.

3 RELATED WORKS

This chapter describes the work found during the literature review. The present work is part of the planning and evaluation of cloud computing systems, performance evaluation of auto-scaling mechanism and financial costs in public and/or private cloud deployment. Lately, several studies are related to forecasting in the 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 determined four different ways to compare our work with other achievements in this area. First of all, we compare which group the proposals belong (e.g. reactive or proactive). Second of all, how many techniques were used in the related work proposals. Third, we compared which input metric the authors used. Finally, we compared which performance metric the authors were interested in. In this section, we basically describe some of the proposals in this area while we compare to our study. This section aims to provide an overview of the published works on the topic in question, so at the end of this section, it is possible to find an effective conclusion showing the relevance of the work proposed here, considering the others presented.

Akioka et al. (AKIOKA; MURAOKA et al., 2004) described the use of time series to forecast CPU and network load in a computational grid. According to (AKIOKA; MURAOKA et al., 2004), to make a highly useful grid, a scheduling system is essential since a computational grid is usually a heterogeneous environment. Then the scheduling system periodically reaps the load information of the available resources and decides how it will allocate the applications. However, it is quite complicated to keep this information up to date due to the size of these environments, so this implies that a good prediction of CPU and network load is very useful for good scheduling (AKIOKA; MURAOKA et al., 2004).

Roy et al. (ROY et al., 2011) aimed to develop a model-predictive algorithm for workload forecasting, which is used for resource auto-scaling. The authors proposed a look-ahead resource allocation algorithm based on a statistical forecasting technique called autoregressive moving average method (ARMA). Although the authors used ARMA, which has a low consumption of resources and can perform the prediction very quickly, it may not be the most suitable for several types of workload. Our proposal aims to increase forecasting accuracy by using several forecasting methods.

Chen et al. (CHEN et al., 2015) applied a Bayesian prediction method to predict and presented a QoS guarantee strategy for cloud service. In their research during the execution of services, a system monitors and predicts the QoS during the system's run-time, and based on monitored and predicted values, the system selects a cloud service to finish tasks. According to (CHEN et al., 2015), their strategy can decrease the chances of QoS violations. Their results show that the Bayesian prediction approach is a good forecasting method,

however, it may not be the most suitable for several types of workload. Our proposal aims to decrease even more the chances of QoS violations by using several forecasting methods, which may increase the probability of achieving a more accurate prediction.

Nikravesh et al. (NIKRAVESH et al., 2017) proposed a high level design of a self-adaptive prediction suite, which chooses the most suitable prediction algorithm based on the incoming workload pattern. The authors also analyzed the impact of sliding window size on time series techniques to observe how the prediction accuracy was affected. The authors only considered machine learning techniques, which have drawbacks due to the time that they spend to train, test and validate.

Jiang et al. (JIANG et al., 2013) proposed an auto-scaling scheme that instantiates virtual machines out or in by time-unit based on predicted workloads for Web Applications. In other words, the proposed scheme is used to predict the average number of web requests in a future time period, where the time-unit used was of one hour. The web request data used for the authors was a time series, and the forecasting technique used by them was linear regression. Their main goal was to propose an optimal VM-level auto-scaling scheme with cost-latency trade-off, and according to their results, their scheme presented an optimal cost-latency for resource auto-scaling. The authors only used a machine learning technique, which may not be highly recommended because it takes a considerable period of time to train, test and validate. It is also more complex than statistical forecasting methods, which can be performed faster.

Biswas et al. (BISWAS et al., 2015) presented an auto-scaling algorithm that uses linear regression and support vector machine (SVM) to forecast future requests to the cloud environment. Their study also includes a discussion of system design and implementation experience for a prototype system that implements their technique. The authors only considered machine learning techniques, which have drawbacks because they consume resources for calculation and may take a long time to train, test and validate.

Yang et al. (YANG et al., 2013) proposed a method that uses a linear regression model to forecast future loads of cloud-based services. Their auto-scaling mechanism scales up or down the system according to the predicted workload. It is worth pointing out that their auto-scaling mechanism uses both horizontal and vertical scaling. Vertical scaling is implemented by changing the partition of resources (e.g. CPU, memory, storage, etc.) inside a virtual machine. The horizontal scaling adjusts the number of virtual machine instances. Experimental results are provided to demonstrate the efficiency of their approach. However, the authors also used only machine learning technique, which may not be highly recommended because it takes a considerable period of time to train, test and validate.

Shariffdeen et al. (SHARIFFDEEN et al., 2016) proposed a method that can be trained during the system's run-time to capture the latest trends and provide sufficiently accurate results for drastically different workload patterns. The authors also proposed an ensemble

technique, which combines results from ARIMA and exponential models, such as Holt, Holt-winters, and so on. The authors only used the prediction approach of auto-scaling, which may not be highly recommended in workloads that have peaks oscillation. To overcome that, our proposal uses predictive and reactive techniques, which support that a workload oscillation can be handled, and the QoS violation probability will be decreased.

In a more recent work, Augustyn et al. (AUGUSTYN; WARCHAL et al., 2017) suggested a reactive approach using two types of metrics, which are resource metrics (e.g., CPU utilization) and application metrics (e.g., response time). The authors claim that the user does not know the optimal values for the lower and upper threshold, considering CPU utilization as the metric to scale the system. So, they suggest using other metrics. However, their proposal uses a threshold-based method, which neglects the time to boot up a virtual machine. We test several threshold scenarios to observe which one results in a better system's throughput, and we also use the proactive approach to overcome the time to boot up a virtual machine.

Niu et al. (NIU et al., 2012) proposed a bandwidth auto-scaling facility that dynamically reserves resources from multiple data centers for VoD providers. It tracks the history of bandwidth demand in each channel of video using cloud monitoring services, and it estimates periodically the expectation, volatility, and correlations of demands for the near future using time series approach. They also formulate a bandwidth balance and a reduction of storage costs. However, the authors did not propose a system monitor, which is an important feature for auto-scaling methods. Our proposal aims to monitor during the system's run-time to ensure that the system can be scaled at any time.

Fernandez et al. (FERNANDEZ et al., 2014) presented an auto-scaling system that benefits from the heterogeneity of cloud infrastructures to better enforce customer requirements, even under large and temporary workload variations. The authors proposed a predictor that prevents SLA violations in advance by forecasting the workload to estimate the incoming traffic. Their predictor takes the monitoring data as input and uses different time-series analysis techniques, such as Exponential Smoothing, Holt-Winters, and ARIMA. However, the authors did not test several input window lengths and forecast horizon sizes, which could improve the accuracy of such methods. Similar to Fernandez et al., Calheiros et al. (CALHEIROS et al., 2015) proposed a workload prediction module using the ARIMA model, where it applies feedback from the latest observed loads to update the model on the run. The predicted load is used to dynamically provide VMs in an elastic cloud environment taking into consideration QoS parameters, such as response time and rejection rate. It is important to not imply that one technique is the most suitable for different kinds of workload, so we proposed a hybrid strategy composed of several forecasting techniques.

Farias et al. (FARIAS et al., 2014) proposed a proactive auto-scaling strategy based on workload prediction of cloud database using time series analysis. Similar Shariffdeen

et al., Fenandez et al., and Calheiros et al, the authors used ARIMA model to predict the incoming workload. The main goal of their research was to improve the QoS and cost reduction. However, only using predictive strategy may not be highly recommended because there are many types of workload with peaks, which may be unpredictable.

Suleiman et al. (SULEIMAN; VENUGOPAL et al., 2013) proposed an analytical model to study the effects of some metrics and their respective thresholds, such as the CPU utilization, the response time, monitoring time windows, and the number of requests to a Web service application. They used this analytical model to determine when to add or remove instances of servers in cloud environment. The authors used the proposed elasticity models and algorithms to conduct simulation experiments with a number of elasticity rules, where it was used different CPU utilization thresholds. They have validated the resulting metrics against the results from the experiments have conducted using the same rules and workload on Amazon EC2. The simulation results demonstrated reasonable accuracy of their elasticity models and algorithms in approximating CPU utilization, application's response time, and number of servers. The models have been able to capture the trends and relationship between changing CPU utilization thresholds and these metrics with acceptable variations. Their proposal uses threshold-based method, which neglects the time to boot up a virtual machine, as already mentioned. We tests several threshold scenarios to observe which one results a better system's throughput, and we also use the proactive approach to overcome the time to boot up a virtual machine.

Moore et al. (MOORE et al., 2013) proposed a hybrid elasticity management framework that takes as input commonly used reactive rule-based scaling strategies and proactive auto-scaling techniques. They also presented a case study, based on real datasets, to demonstrate that their proposal is able to make appropriate auto-scaling decisions and improve the resource utilization when compared to only threshold-based technique. They proposed to use two controllers, one reactive and one proactive. The predictive controller is composed of 3 models, a time series technique, and two incrementally updatable Naive Bayes models. Although the Naive Bayes works very well with strongly random-pattern data, it does not work very well with time series that present strong tendence and seasonality behaviors.e

Ali-Eldin et al. (ALI-ELDIN et al., 2012) 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. They only used regression as proactive technique, which means that they do not consider other forecasting techniques, such as exponential smoothing, ARIMA, and so forth.

Iqbal et al. (IQBAL et al., 2011) 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 forecast the future needs of a Web application. According to the authors, it is very difficult to identify a minimally resource intensive configuration of a multi-tier Web application that satisfies given response time requirements for a given workload.

Urgaonkar et al. (URGAONKAR et al., 2008) proposed a technique that employs two methods that operate at two different time scales, predictive and reactive. The predictive approach allocates capacity at the time-scale of hours or days. The reactive approach operates at time-scale of minutes to respond to peaks of the workload. The authors also proposed an analytical model based on queuing theory to capture the behavior of applications with an arbitrary number of tiers. Their experiments were executed on a forty machine Xen/Linux-based hosting platform.

Analyzing the related works, we are able to observe that forecast is extensively studied in the auto-scaling literature. Several prediction techniques are proposed to perform auto-scaling. Comparing the related works to our proposal, we can observe that our proposal provides more prediction techniques in order to increase the forecast accuracy while increasing the system throughput and decreasing the QoS violation probability. Table 1 shows a comparison among the most relevant related works and our proposal. We compare which group the proposals belong (e.g. reactive or proactive), how many techniques were used in the related work proposals, which input metric was used to scale do system, and which performance metric their proposal improved.

Hybrid approaches to scale the system automatically were proposed in (MOORE et al., 2013; ALI-ELDIN et al., 2012; IQBAL et al., 2011; URGAONKAR et al., 2008), where (MOORE et al., 2013) used Naive models, (ALI-ELDIN et al., 2012) used regression, (IQBAL et al., 2011) also used regression, and (URGAONKAR et al., 2008) only considered using neural network. Regressions and neural networks are costly, and they may not be the most suitable forecasting technique for the moment. So our work proposes to use five forecasting techniques, which are based on statistical methods. Therefore, they are not costly and the probability of choosing the most suitable technique for the moment is increased considerably.

This study is relevant because it uses several features of the related works, whereas there is an improvement of approaches proposed by them. This dissertation approaches the weakness of related works in order to improve the auto-scaling mechanism for cloud computing environments. This research makes use of several forecasting techniques to increase the accuracy of the predictions and to support a better auto-scaling process. This study presents a hybrid strategy for virtual machine provisioning. Five different forecasting models are presented as the base of our proposal: Drift, Simple Exponential Smoothing, Holt, Holt-Winters, and ARIMA. The models used by this strategy were

Table 1 – The most relevant related works comparison.

	Approach	#	Input	Output
(AKIOKA; MURAOKA et al., 2004)	proactive	1	CPU/network	throughput
(ROY et al., 2011)	proactive	1	workload	SLA violation
(CHEN et al., 2015)	proactive	1	network	SLA violation
(NIKRAVESH et al., 2017)	proactive	3	workload	-
(JIANG et al., 2013)	proactive	1	workload	cost
(BISWAS et al., 2015)	proactive	2	workload	SLA violation
(YANG et al., 2013)	proactive	1	workload	SLA violation
(SHARIFFDEEN et al., 2016)	proactive	3	workload	response time
(AUGUSTYN; WARCHAL et al., 2017)	reactive	1	CPU	response time
(NIU et al., 2012)	proactive	3	network	throughput
(FERNANDEZ et al., 2014)	proactive	3	workload	SLA violation
(CALHEIROS et al., 2015)	proactive	1	1	response time
(FARIAS et al., 2014)	proactive	1	workload	cost
(SULEIMAN; VENUGOPAL et al., 2013)	reactive	1	CPU	response time
(MOORE et al., 2013)	hybrid	4	workload	response time
(ALI-ELDIN et al., 2012)	hybrid	2	workload	SLA violation
(IQBAL et al., 2011)	hybrid	2	response time	SLA violation
(URGAONKAR et al., 2008)	hybrid	2	CPU	response time
Our Proposal	hybrid	6	CPU	throughput

chosen based on related works. The hybrid strategy presented in this dissertation aims to reduce the latency of provisioning cloud resource, and improve the cloud service quality through improving the system throughput. 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.

3.1 FINAL REMARKS

In this chapter, it was presented the most relevant related works for our proposal. Although there are many studies in the literature about auto-scaling mechanism using time series approach, most of them only use either proactive approach with only few forecasting techniques or reactive approach. It may be a problem because none of the few forecasting techniques may be the best fit for the moment, and using more forecasting techniques increases the probability of having a better fit for the scenario. Another problem is that using only a reactive approach, the time to boot up the virtual machine is neglected, which may increase the probability of QoS violations. So using a hybrid strategy of auto-scaling virtual machines using time series and threshold may be a better solution because it will decrease the probability of QoS violation, while it will not neglect the time to boot

up virtual machines. The impact of improving the auto-scaling mechanism is huge for companies because it reduces the request rejection rates, which consequently improves the system's throughput.

4 A HYBRID STRATEGY FOR AUTO-SCALING OF VMS

The popularity of on-demand cloud service has spurred the migration of increasing numbers of web applications to the cloud. One of the most attractive features for cloud web application providers is the ability to access computing resource elastically (by scaling out or in) according to dynamic resource demands. In this scenario, providers only pay for resources that are consumed at a specific point in time, which if operated correctly, will result in less cost and higher quality service than is achievable by hosting on standard hardware.

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 (ANDERSON et al., 2011).

This chapter has two section, where the first one (Section 4.1) describes in a detailed manner our hybrid strategy for auto-scaling of virtual machines.The last section refers to the final remarks of our proposal.

4.1 STRATEGY DESCRIPTION

Currently, cloud providers and third-party cloud services offer schedule-based and rule-based mechanisms to help users automatically scale (out/in). Many of these are based on resource utilization, such as “Add small instances when the average CPU is above 70%.” The mechanisms are simple and convenient. However, it is not always straightforward for the users to select the “right” scaling indicators, especially when the application models are complex, and the resource utilization indicators are limited and very low-level. In other words, these trigger mechanisms do not really solve the performance-resource mapping problem, and sometimes the resource utilization indicators are not expressive enough to address user performance requirements directly.

IaaS cloud resources can be efficiently managed and utilized by predicting either the future workload or the future resource utilization pattern. The nature and type of workload at a public cloud are not deterministic, so some cognitive techniques are required to predict the type and nature of workload along with size and rate. Also, the workload

does not provide realistic information about required memory and CPU before subjecting to a virtual machine. Therefore, a better way for efficient resource management is to predict the resource utilization of all virtual machines within the cloud and then allocate resources that fulfill the required predicted memory, CPU, and storage. This approach has more realistic information about virtual machines than the workload prediction. In this approach, we predict the resource utilization of a virtual machine. Then, based on predicted utilization, the resources are allocated by the hypervisor. The predicted resources utilization tells that the future workload will require less or more CPU power.

Resource estimation lies in the core of auto-scaling as it determines the efficiency of resource provisioning. It aims to identify the minimum amount of computing resources required to process a workload to determine whether and how to perform scaling operations. Accurate resource estimation allows the auto-scaler to quickly converge to the optimal resource provisioning. On the other hand, estimation errors either result in insufficient provisioning, which leads to a prolonged provisioning process and increased SLA violations, or over-provisioning, which causes more cost.

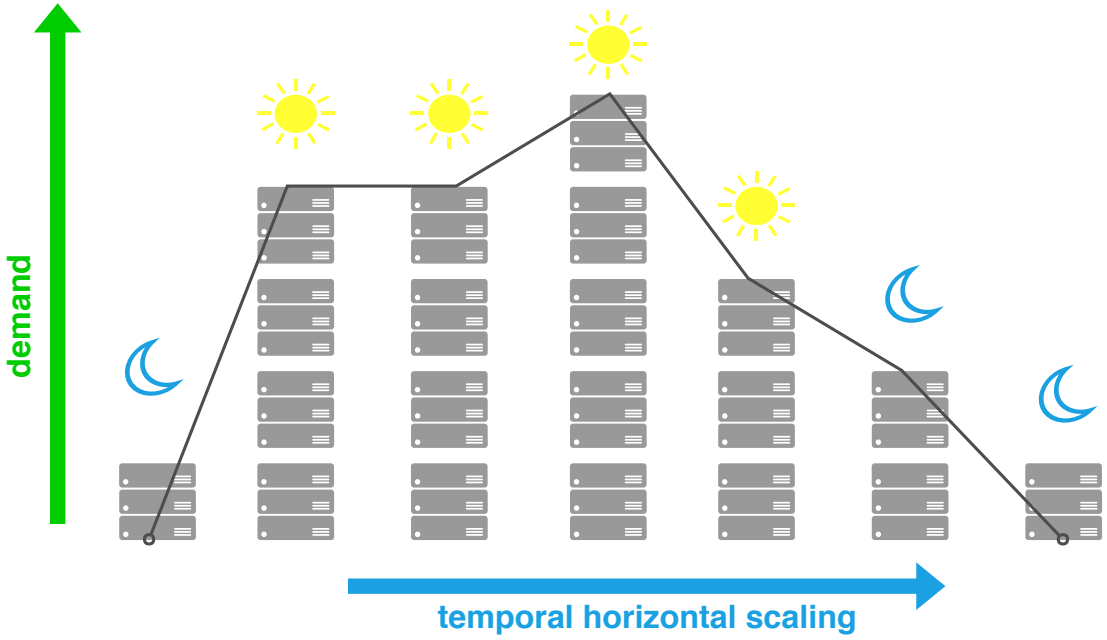


Figure 2 – Auto-scaling x demand.

For our proposal, we consider that a data center that supports an IaaS deployment model is comprised of a number of clusters that use some virtualization technology. Those clusters host VMs that run a web service. The data center defines a set of instance types that can be instantiated and started on top of its clusters. Each instance type characterizes a VM in terms of resource capacity (CPU, RAM memory, network bandwidth, and so on).

We assume that all VMs are running a website, for simplicity. This website may present time-varying workloads, which means that the amount of VM instances needed to appropriately run this service may vary over time. Thus, the website is seen as a

component to be dynamically provisioned with resources. We assume that the VMs are horizontally scalable, and a load balancing service is able to appropriately balance the demand of the users to each VM that has been allocated for that website. For the sake of simplicity, we assume that each VM is of an instance type. Different VMs may be executed, but each service may only run at instances of the same type. If different VM types were used it would be possible to tune the infrastructure in a more accurate way, even when large instances are in use. However, it would bring a complexity not only to the scaling out service but also to the load balancer, as both would have to deal with heterogeneous resources accordingly. We believe the gain achieved by using heterogeneous resources in a website does not compensate the extra complexity (LORIDO-BOTRAN et al., 2014).

Assuming that the website is associated with an SLA. We assume that there is a mapping that relates the satisfaction of the SLA and the resources utilization of the allocated VMs. If the resource utilization is kept sufficiently often between some range, then the SLA of the website is always satisfied. Clearly, keeping the resource utilization between a range will result in needing fewer VMs and reduce cost. Therefore, the goal of our proposal is to perform auto-scaling, in such a way that we minimize the idle CPU time and increases the system throughput.

Our scheme scales the cloud resource out, in or NOP (no operation) by time-unit re-allocation based on predicted optimal resource demands. Web application providers can specify their budget constraints and SLA with respect to latency for their applications. In each time-unit, web application providers can be notified of the total cost, SLA violations and re-allocation state (e.g. scaling out or in or NOP) by using the auto-scaling scheme. Notice that in practical applications, an unpredictable burst of requests will happen. To tackle this unpredictable scenario, this scheme monitors the CPU usage in real-time. Once the resource utilization is bigger than a threshold, the scheme could dynamically instantiate a new VM to process the exceeding number of requests.

To achieve the main goal of this study, which is to improve the system's throughput while reducing the CPU idle time, we propose a hybrid approach to auto-scaling of virtual machines. Our proposal is a mix of reactive and proactive techniques. The reactive approach is threshold-based, and it is only used in case of unpredictable CPU usage change. In other words, we prioritize the proactive approach and use the reactive method to ensure that the system will be scaled if it is necessary.

To predict the resource utilization, the data is generally denoted as a time series (ZHAO; SCHULZRINNE et al., 2003): $X(t); t \in T$, where T is an index of the time fragment, and $X(t)$ is the random variable, representing the total CPU that is being utilizated in the t time fragment. The prediction problem can be defined as follows: given the current and past observed values $X(t-k), X(t-k+1), ..., X(t-1), X(t)$, predict the future value $X(t+p)$, where k is the length of the history data used for prediction (input window

length) and p is the predictive time fragment (forecast horizon size).

For instance, a mail server usually experiences the highest CPU usage every Monday morning, while at midnight, the resource utilization drop to a low level; also, the usage will be much higher on weekdays than on weekends. Many more users may request a mail service on festivals and holidays than on other days. Therefore, the resource utilization behavior pattern can be established and key features such as hourly, daily, weekly, monthly, seasonally, and so on, can be identified by analyzing the history data. Considering a web requests time series $X(t-k), X(t-k+1), ..., X(t-1), X(t)$, where $X(t)$ represents the CPU usage during this time fragment t, the consumption in the next time fragment t is related to the resource utilization in previous time fragments.

The focus of this work is related to the operation of a Web application where it was applied the characteristic of CPU-bound, so we could perform huge variations in resource utilization. The metric analyzed to increase the system's throughput was the resource utilization in percentage. We chose this resource because if it is too high, it means that the server is really busy and may start denying requests. The thresholds are used as limits of resource utilization, they also define moments that some actions need to be executed, otherwise it may occur QoS violations.

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 aim of the auto-scaler is to dynamically adapt the resources assigned to the elastic applications, depending on the resource usage. Anticipation is important because there is 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 (LORIDO-BOTRAN et al., 2014).

Figure 3 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,

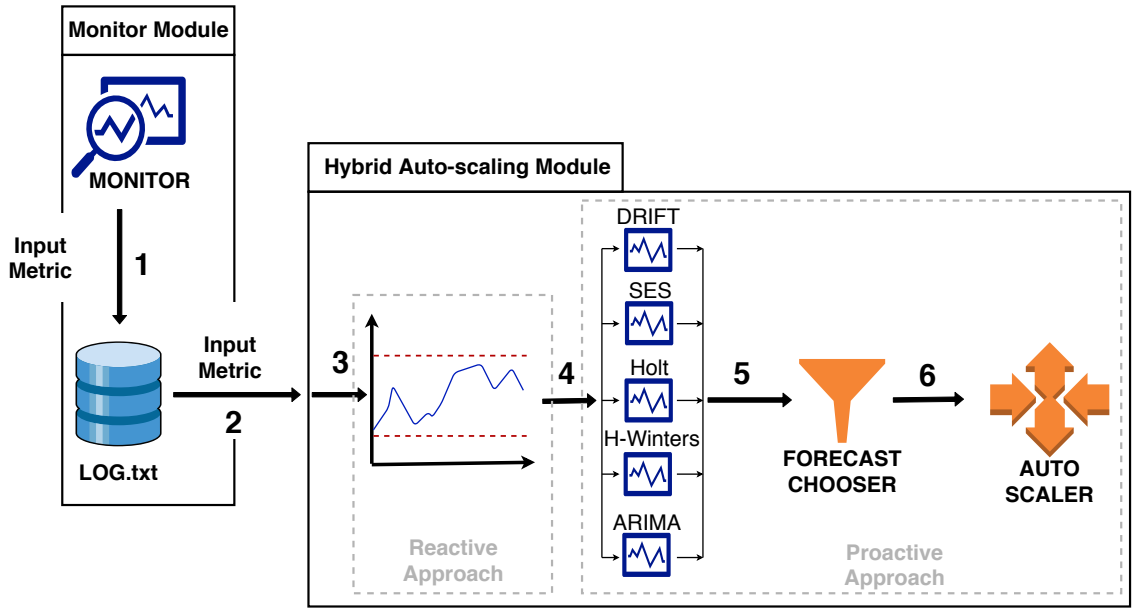


Figure 3 – Information flow diagram.

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.

4.1.1 The information flow

It is worth pointing out that we chose the time interval of 30 seconds based on related works and also because we assumed that the workload is increasing and decreasing quickly. For other applications, the workload could increase and decreases slower, so the time interval for collecting samples could be increased. In other words, choosing the time interval depends on the necessity of each application (KAUR et al., 2019). To each sample collected, the main script verifies if it has achieved the upper or lower threshold. If the resource utilization has achieved the threshold the reactive auto-scaling process begins, which means that our approach invokes the hypervisor to start performing the scaling process. If the threshold has not to be achieved yet, the main script invokes the forecasting script to calculate the forecasting techniques, which is represented by the auto-scaling module (2). It gets the latest samples collected based on the input size length and starts the analysis.

First, our proposal splits the data into two sets, training and test set, then it starts the model fitting. It is worth pointing out that the hybrid approach trains all five techniques to fit them as best as possible to the resource utilization time series. Later in this chapter, we explain how we choose the parameters of each method. By that, our proposal calculates which method is the most suitable for the current scenario based on accuracy metric (MAPE). After choosing the most suitable forecasting technique, our hybrid approach finally performs the forecast using the training and the test set as input for the chosen

method. The parameters are calculated again for the most suitable technique, and our approach generates a final result for that moment. The forecasting process returns the result indicating if it is necessary to scale (out/in) the system or do nothing. If there is the need of scaling out or in the system, our proposal invokes the hypervisor to scale it.

The auto-scaling mechanism we propose in this dissertation 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. It works very well as long as some requirements are met, which will be discussed later in this chapter.

4.1.2 Monitoring module

All cloud computing environments need to support monitoring of their resources, providing measurements about user demands (LORIDO-BOTRAN et al., 2014). The resource monitor component is responsible for collecting information about the metrics of the computational, storage, and network resources (e.g., CPU utilization, memory usage, and network traffic). There are many useful tools that access information about cloud computing infrastructure utilization. Auto-scaling decisions are based on the information provided by monitoring modules, and the performance of the auto-scaler relies on the quality of the resource usage data. A bad monitoring system will compromise the quality of the auto-scaler performance, which will increase the QoS violation probability.

The auto-scaler needs to monitor some performance indicators to determine whether scaling operations are necessary and how they should be performed. Selection of the right performance indicators is vital to the success of an auto-scaler. The decision is often affected by many factors, such as application characteristics, monitoring cost, SLA, and the control algorithm itself. The monitoring interval determines the sensitivity of an auto-scaler. However, very short monitoring intervals result in high monitoring cost both regarding computing resources and financial cost, and it is likely to cause oscillations in the auto-scaler. Therefore, it is important to tune this parameter to achieve balanced performance (LORIDO-BOTRAN et al., 2014).

The Algorithm 1 shows the resource utilization monitoring process of VMs running a Web Application. It is worth taking into consideration there may be several virtual machines running at the same time in the environment providing different services. Thus, we have written a shell script to capture only the CPU usage of the Web application. In other words, it may have many virtual machines running many other applications in the cloud, but our monitor is set to get only the resource consumption of our web-service. So it does not matter if there are many different applications running in other virtual machines, our monitor only captures the resource consumption of our service and saves the values in a text file, in line 6.

As we are able to notice in Algorithm 1, we start a variable to store the sum of the virtual machines' resource utilization. The *while* loop from line 3 to line 5 repre-

Algoritmo 1: VM CPU usage monitor

Input: A list of VMs running the Web Application
Output: Average consumption of VMs in a text file

```
1 values ← 0;
2 while listOfVMS ≠ END do
3 |   values ← values + vmCpuUsage;
4 end
5 output ← values/listOfVMS.Size;
6 append(output, file);
```

sents the resource utilization sum of all virtual machines in the system that is running our application. After this loop, we are able to get the resource consumption average of our application. In line 5, the *listOfVMS.Size* represents the number of virtual machines running our web-service in the system. Line 6, we append the result in the end of the file.

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 (Appendix B). In other words, the cool-down period prevents the resource allocator of making any changes to the system deployment on the cloud for a certain amount of time. The motivation behind this is to avoid frequent creation and releasing of instances when the resource consumption exhibits high variability, as this would have a negative impact on the cost (JAMSHIDI et al., 2014). As a result, the auto-scaler is only able to change the deployment if sufficient time between scaling action has passed and if the new instances' state is 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.

4.1.3 Auto-scaling module

Auto-scaling is getting some attention from the research community, as the limitations of the present solutions become more and more clear. One of such limitations is the lack of flexibility. The auto-scaling configuration is set using a GUI with limited choices regarding the metrics to be monitored and the conditions that will trigger the corresponding scaling actions.

The auto-scaling module processes and analyzes the gathered data about the system utilization. Our auto-scaling module consists of thresholds and five forecasting methods, which are: Drift, Simple Exponential Smoothing, Holt, Holt-Winters, and ARIMA (SHARIFFDEEN et al., 2016; CHEN et al., 2015; FERNANDEZ et al., 2014; FARIAS et al., 2014). The prediction happens during the system's run-time, so training and testing each prediction method cannot take too long.

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 selected model among the five forecasting methods fits the data and then predicts the next resource utilization values.

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 (HYNDMAN; ATHANASOPOULOS et al., 2014). 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. It is worth pointing out that every time a sample is collected, the whole auto-scaling verification process is performed again, which means that every 30 seconds our approach is performed to verify if there is the need to scale the system (KHAN et al., 2015). Figure 4 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 (ANDERSON et al., 2011).

The approach proposed in this dissertation 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.

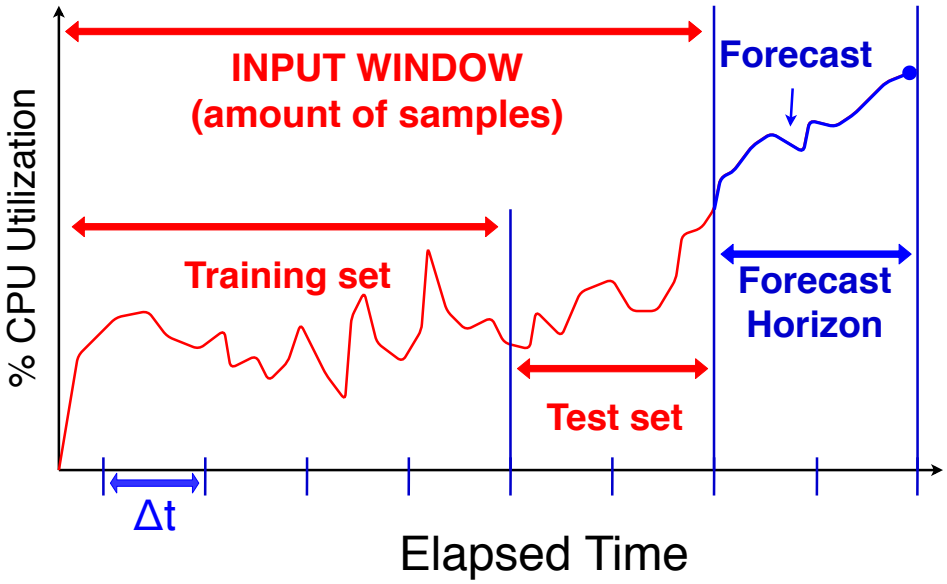
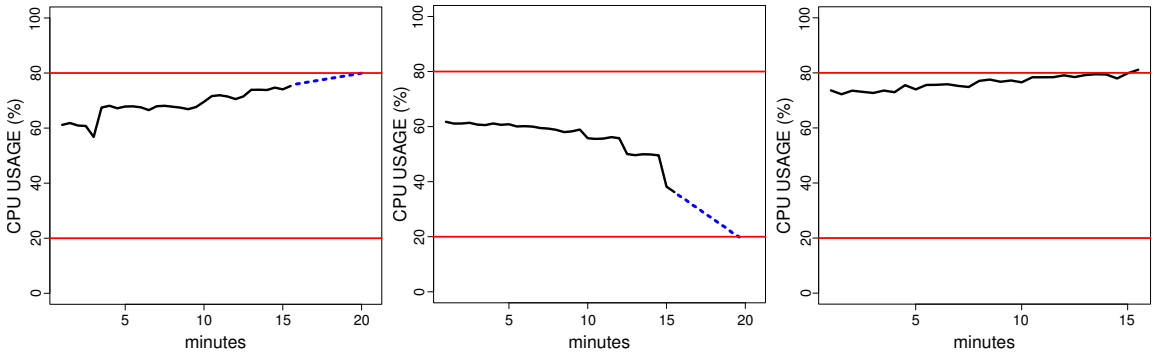


Figure 4 – Splitting into training/test sets and forecasting.

The Figure 5(a) was generated during our case study and shows the forecast exceeding the upper threshold, which performs the scaling out process. According to this prediction, the system needs a new virtual machine to keep the CPU utilization between the upper and lower threshold. The Figure 5(b) was also generated during our experiment. It shows the forecast going past the lower threshold, which performs the scaling in process. According to this prediction, the system needs to destroy one virtual machine. Otherwise, the system will have more virtual machines than necessary, which is a waste of resource. So, the system starts the scaling-in process to destroy the last virtual machine created. In the Figure 5(a) and Figure 5(b), the forecast is the dotted line. The Figure 5(c) shows the CPU utilization, measured in real time, exceeding the upper threshold, it happens when previous forecasts did not predict that the threshold would be achieved. So, this reactive approach is used to ensure the system stabilization.

The Algorithm 2 depicts how the forecast is performed in the auto-scaling module. The input of the Algorithm 2 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.

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



(a) System scaling out by prediction.

(b) System scaling in by prediction.

(c) Scaling out by achieving the upper threshold.

Figure 5 – System scaling types.

(*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% (HYNDMAN; ATHANASOPOULOS et al., 2014), 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 dissertation 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 (JR et al., 1985). Holt model captures better the linear trend of the time series (HYNDMAN et al., 2005). Holt-Winters model captures the seasonality of the data (KALEKAR et al., 2004). ARIMA model

Algoritmo 2: Auto-scaling forecast

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

initialization;
$timeSeries \leftarrow readFile(log.txt)$;
$samples \leftarrow tail(timeSeries, inputWindowLength)$;
if $last(samples) > upThreshold$ **then**
    $scale(out)$;
else if $last(samples) < downThreshold$ **then**
    $scale(in)$;
else
    $traningSet \leftarrow split(samples, 80)$;
    $testSet \leftarrow split(samples, 20)$;
    $mDrift \leftarrow drift(trainingSet)$;
    $mSes \leftarrow ses(trainingSet)$;
    $mHolt \leftarrow holt(trainingSet)$;
    $mHW \leftarrow holtWinter(trainingSet)$;
    $mArima \leftarrow arima(trainingSet)$;
    $errorDrift \leftarrow mape(mDrift, testSet)$;
    $errorSes \leftarrow mape(mSes, testSet)$;
    $errorHolt \leftarrow mape(mHolt, testSet)$;
    $errorHW \leftarrow mape(mHW, testSet)$;
    $errorArima \leftarrow mape(mArima, testSet)$;
    $bestModel \leftarrow min(errorDrift, errorSes, errorHolt, errorHW, errorArima)$
    $result \leftarrow forecast(samples, bestModel, UpTreshold, DownThreshold)$;
    $scale(result)$;
end

is more general than exponential models and may capture better patterns in stationary data (HYNDMAN; ATHANASOPOULOS et al., 2014).

Load-balancing mechanisms should be ready to update the set of available server replicas, as it can change at any moment due to the creation and removal of VM instances. Our auto-scaling process takes around 5 minutes to instantiate a new virtual machine and make the load balancer to recognize it.

The auto-scaling mechanism we propose in this dissertation is completely independent of the service in a scale scenario. Besides, it follows a proactive-reactive operation mode and the proactive behavior considers the use of multiple forecasting techniques, from which one is judiciously selected from time to time to provide estimation about future service demand, allowing auto-scaling to be adaptive to changes on resource consumption patterns.

4.2 FINAL REMARKS

In summary, to deploy our hybrid auto-scaling method, it is essential that the hypervisor is running on a Linux environment, and it must have the R statistical language also installed, otherwise our approach will not be able to predict the resource consumption. It is worth pointing out that we chose the paravirtualized environment for simplicity. However, it would also run in a full-virtualized environment, making some minor adjustments.

It is also important to say that other hypervisor may be used to replace the XenServer. However, it would be necessary to change the monitor module, which is not that complicated. Our hybrid approach only needs to be running in the master node (cloud controller), the other nodes just need to be using the XenServer hypervisor in order to be monitored accurately. The auto-scaling mechanism we propose in this dissertation is completely independent of the service in a scale-out scenario, which means that the type of load balancer and the type of application running in the virtual machines do not matter for our proposal, it works very well as long as the previous requirements are met. In order to use the monitor module presented in this chapter, all physical nodes must run XenServer as the hypervisor.

5 CASE STUDIES

This chapter presents three case studies, which show the proposed method applied in a real system. This chapter aims to evaluate the performance of our hybrid auto-scaling approach. In the first case study (Section 5.1), we compared our proposal against the purely reactive auto-scaling technique. We compared three different workloads and several configurations in our strategy to support that our method presents an improvement in the system throughput. In the second case study (Section 5.2), we performed a statistical analysis to support that our strategy differs from the purely reactive technique, which means that our strategy is statistically different from the purely reactive one. Forecasting methods are very sensitive to the input window length and the horizon forecasting size. Therefore, in the third case study (Section 5.3), we performed an analysis to see which of these two components has the highest impact on the forecasting models of our proposal, which affect directly the performance of the system. The results achieved in those case studies provide evidence on the usefulness and efficacy of this approach.

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 (FERNANDEZ et al., 2014; URGAONKAR et al., 2008; DUTTA et al., 2012).

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.

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

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

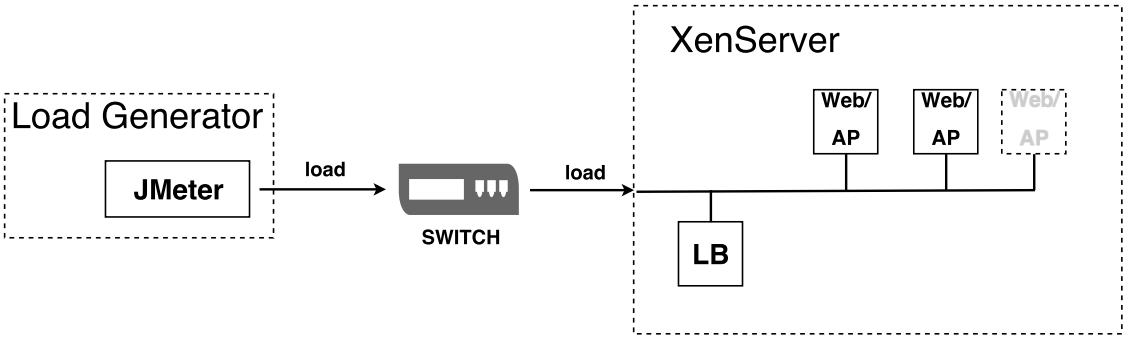


Figure 6 – Experimental environment.

the fourth request comes, it will be sent to the first web server, and so on. Therefore, the requests are distributed equally to the application servers. Figure 6 shows how the experimental environment was set.

5.1 CASE STUDY 1: EXPERIMENTAL HYBRID APPROACH

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.

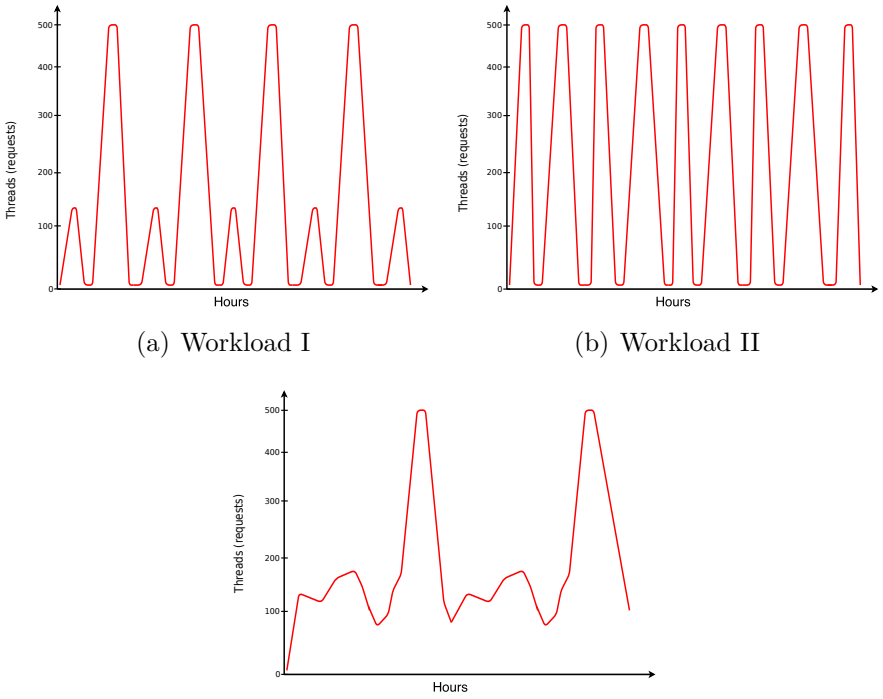
Table 2 – Scenarios analyzed in this study.

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 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 2. 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 (SHYNKEVICH et al., 2017). The time interval between samples collection is 30 seconds as already mentioned (KHAN et al., 2015).

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.



(a) Workload I

(b) Workload II

(c) Workload III

Figure 7 – Types of workload.

We analyzed each scenario under three distinct workload types, which are shown in the Figure 7. 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 (NIKRAVESH et al., 2017; MAO; HUMPHREY et al., 2013). By using peaks that occur periodically, we tried to represent a load that increases and decreases very quickly (BANKOLE; AJILA et al., 2013). 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 (NIKRAVESH et al., 2017). 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.

We ran each type of workload for all 10 scenarios, where nine of them are using a set of input window length and forecast horizon, and the 10th is a purely reactive threshold-based technique. Each workload has a duration of 24 hours.

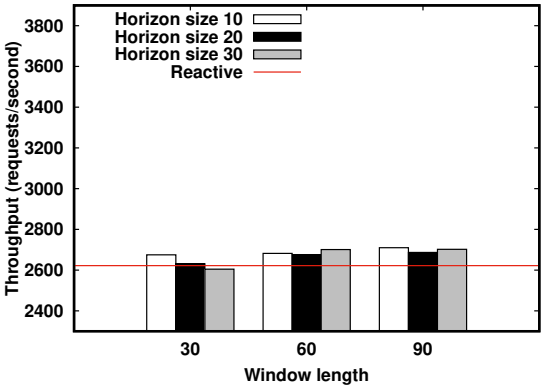
5.1.1 Lower threshold 30% and upper threshold 70%

In the first threshold settings, we set the lower threshold as 30% and the upper threshold as 70%. Figure 8 is a bar graph, and it shows the results of our experiments. 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. Proactive techniques are dependent on how accurate forecasting methods are (LORIDO-BOTRAN et al., 2014). So, when we test several combinations of input window length and forecast horizon size, the prediction accuracy may be highly affected. Figure 8 shows the results of our ten scenarios under three distinct workloads.

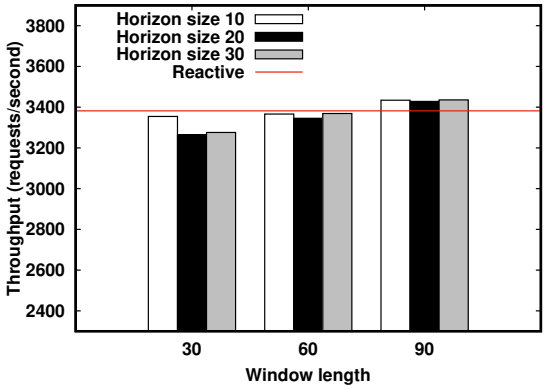
Table 3 – Throughput results (request/s) for 30% and 70% thresholds.

I	H	Workload I	Workload II	Workload III
30	10	2675.07	3354.41	3259.55
	20	2630.78	<u>3265.24</u>	3170.01
	30	<u>2604.63</u>	3275.82	3249.38
60	10	2681.88	3366.42	3298.78
	20	2676.07	3345.60	3151.76
	30	2700.76	3369.17	<u>3107.44</u>
90	10	**2710.08**	3434.16	3335.24
	20	2686.60	3428.56	**3428.56**
	30	2702.11	**3435.86**	3220.87
Reactive	-	2621.76	3381.89	3333.73

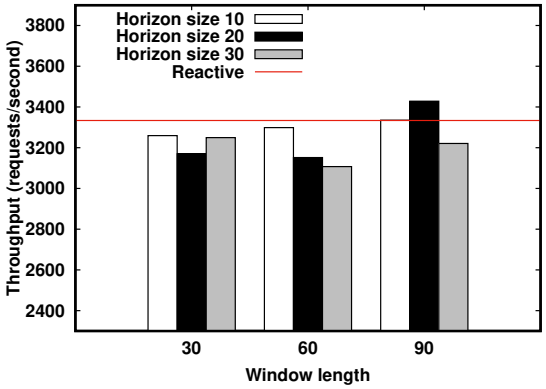
In Figure 7, we can observe that there is a strong seasonal pattern in Workload I and II. So, it is preferable to use an input window length and forecast horizon size large enough to capture this seasonal pattern (HYNDMAN; ATHANASOPOULOS et al., 2014). If the prediction model is able to capture this pattern, the data behavior will be forecasted more precisely. In Figure 8, the input window length of 90 samples had the highest throughput for the Workload I, II and III. That occurs because the area where the CPU usage can vary is very small, which means that many small variations may scale the system. So



(a) Workload I - 10 scenarios



(b) Workload II - 10 scenarios



(c) Workload III - 10 scenarios

Figure 8 – Throughput results for 30% and 70% thresholds (request/second).

using more samples to forecast the CPU utilization, the auto-scaling module can capture better the workload variation, which will scale fewer times the system.

The Table 3 also summarizes our throughput results (request/second), where 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 3 indicates a combination of input window length and forecast horizon size. The bold values in the Table 3 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 3.36% the system's throughput for Workload I, 1.59% for Workload II, and 2.84% for Workload III, which represents a better resource usage.

Our hybrid method provides the best throughput results for several types of workload. Even trying different combinations of input window length and forecast horizon size, some of them do not perform worse than the reactive threshold-based technique. Therefore, our results suggest that our proposed method can increase the throughput.

Using this set of thresholds, considering the whole experiment, the system was scaled

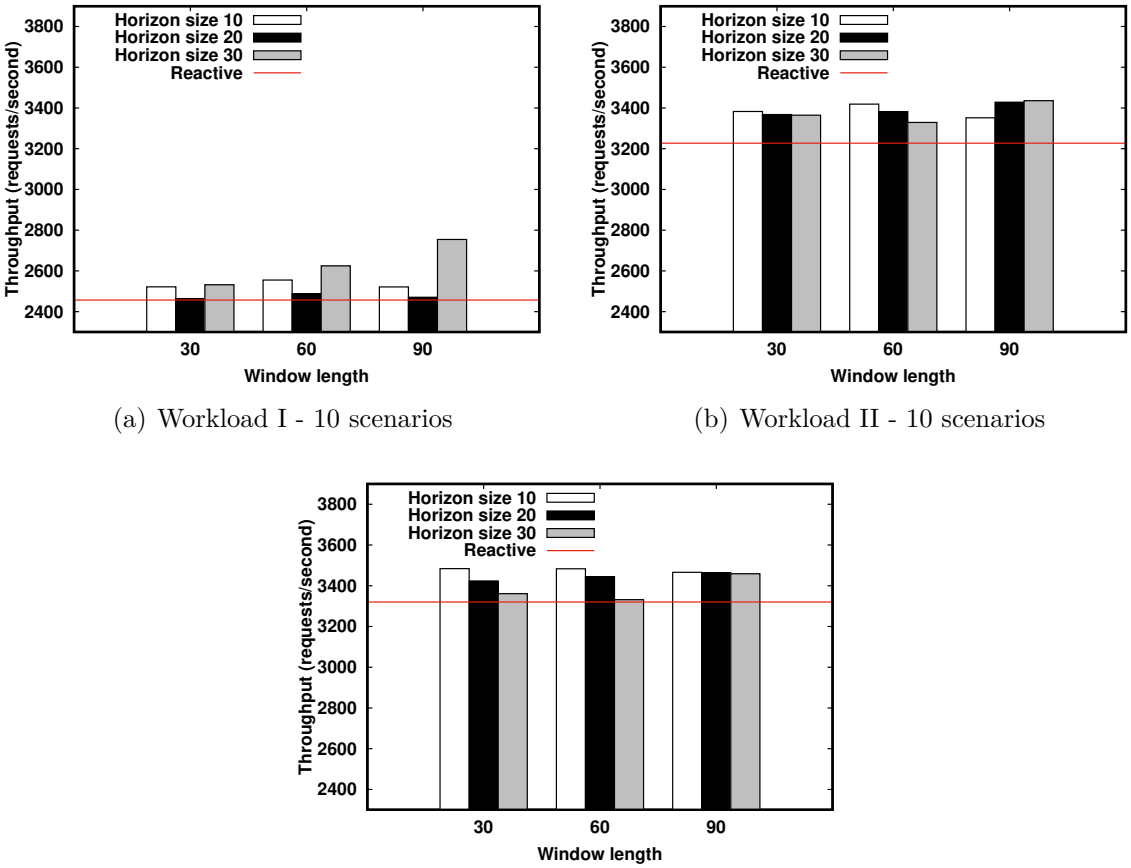
Table 4 – Times that each technique was used to scale the system.

Scenario	Workload	Reactive	Drift	SES	Holt	H.W.	ARIMA	Total
1	1	0	3	0	6	5	4	18
	2	0	0	0	4	4	8	16
	3	0	1	0	7	6	4	18
2	1	0	3	0	6	6	5	20
	2	0	4	0	10	5	3	22
	3	0	5	0	9	6	8	28
3	1	0	4	2	12	3	1	22
	2	0	2	0	8	7	3	20
	3	0	1	0	9	5	9	24
4	1	0	1	0	2	3	10	16
	2	0	1	0	7	4	4	16
	3	2	2	0	3	0	3	10
5	1	0	1	0	7	6	4	18
	2	0	2	0	7	4	5	18
	3	0	1	1	4	4	2	12
6	1	0	3	0	5	4	4	16
	2	0	1	0	5	4	6	16
	3	0	1	0	6	4	5	16
7	1	0	3	0	5	2	6	16
	2	0	0	0	5	5	6	16
	3	0	0	1	6	3	2	12
8	1	0	4	0	8	2	4	18
	2	0	3	0	7	5	1	16
	3	0	0	0	7	4	5	16
9	1	1	4	2	5	1	3	16
	2	0	2	0	5	6	5	18
	3	0	2	0	5	6	1	14
Total	-	3	54	6	170	114	121	468

468 times. The most technique used to scale the system was the Holt method, the second one was the ARIMA method, the third was the Holt-Winters, the forth was Drift, and finally the fifth was the reactive approach. The times that the Holt method was used represent 36.32% of the total times that the system was scaled against the ARIMA that represents 25.85% of the total times.

5.1.2 Lower threshold 20% and upper threshold 80%

Setting the lower threshold as 20% and the upper threshold as 80%, we could improve our results. Figure 9 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 (LORIDO-BOTRAN et al., 2014). 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 9 shows the results of our ten scenarios under three distinct workloads.



(a) Workload I - 10 scenarios

(b) Workload II - 10 scenarios

(c) Workload III - 10 scenarios

Figure 9 – Throughput results for 20% and 80% thresholds (request/second).

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 (HYNDMAN; ATHANASOPOULOS et al., 2014). With that in mind, we are able to observe that our results confirm that. In Figure 9,

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 5 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 5 indicates a combination of input window length and forecast horizon size. The bold values in the Table 5 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 5 – 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.

Using this set of thresholds, considering the whole experiment, the system was scaled 413 times. The most technique used to scale the system was the Holt method again, which represents 32.92% of the total scaling times of the system. The second one was the Holt-Winters method that represents 28.81%, and the third was the ARIMA that represents 27.11%. As we can notice, once again the Holt method was the technique more used to

Table 6 – Times that each technique was used to scale the system.

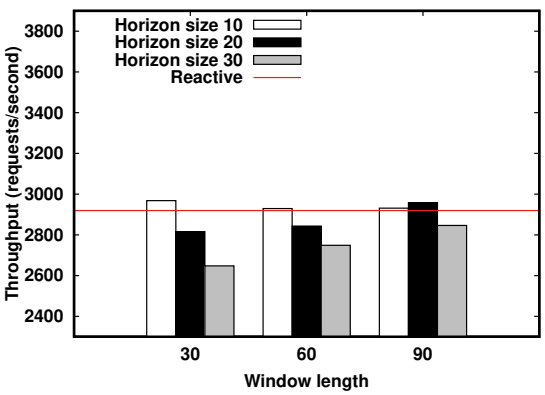
Scenario	Workload	Reactive	Drift	SES	Holt	H.W.	ARIMA	Total
1	1	0	0	0	3	2	10	15
	2	0	0	0	5	3	3	11
	3	0	1	0	4	6	4	15
2	1	0	1	0	5	12	1	19
	2	0	3	0	10	5	3	21
	3	0	2	1	9	7	7	26
3	1	0	2	2	7	3	4	18
	2	0	0	0	7	7	4	18
	3	0	5	0	5	4	7	21
4	1	0	1	0	5	5	3	14
	2	0	1	0	6	5	2	14
	3	0	2	0	2	3	2	9
5	1	0	0	0	8	4	2	14
	2	0	2	0	4	3	5	14
	3	0	0	0	5	5	1	11
6	1	0	2	0	6	2	3	13
	2	0	0	0	5	3	7	15
	3	0	3	0	3	5	5	16
7	1	0	4	0	4	3	3	14
	2	0	0	0	4	6	5	15
	3	0	0	2	4	3	1	10
8	1	0	2	0	7	3	3	15
	2	3	0	0	5	4	5	17
	3	0	2	0	5	3	6	16
9	1	1	3	0	3	3	5	14
	2	0	0	0	2	7	7	17
	3	0	1	0	3	3	4	11
Total	-	4	37	5	136	119	112	413

scale the system. However, the ARIMA and Holt-Winters method were also used many times.

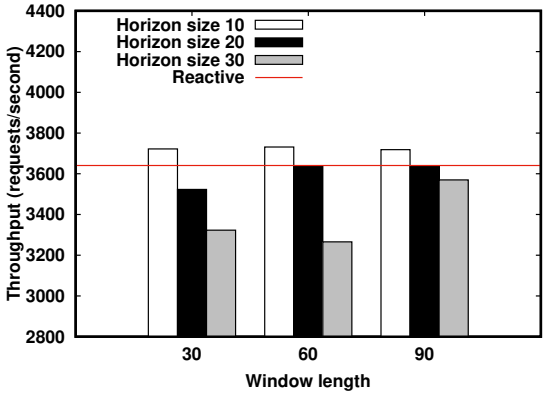
5.1.3 Lower threshold 10% and upper threshold 90%

The throughput results in requests per second are shown in the Figure 10. Our results suggest that our proposal performed again better than the reactive threshold-based technique. However, differently, from the settings that used 20% and 80% as lower and upper threshold respectively, some combinations of input window length and forecast horizon

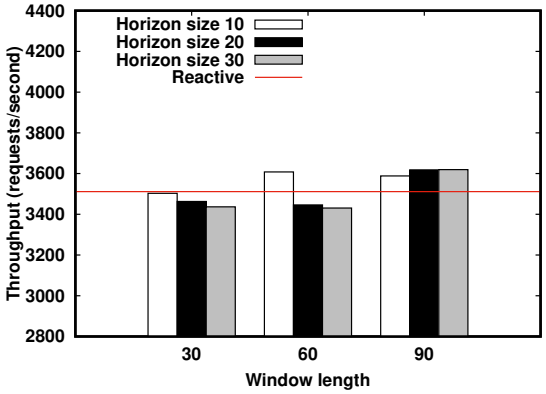
size perform worse than the threshold-based technique. It happens because proactive techniques are dependent on how accurate prediction mechanisms are (LORIDO-BOTRAN et al., 2014). So when we tested several combinations of input window length and forecast horizon size, the prediction accuracy was highly affected. Figure 10 shows the results of our ten scenarios under three distinct workloads.



(a) Workload I - 10 scenarios



(b) Workload II - 10 scenarios



(c) Workload III - 10 scenarios

Figure 10 – Throughput results for 10% and 90% thresholds (request/second).

There is a strong seasonal pattern in Workload I and II. Although it was preferable to use an input window length and forecast horizon size large enough to capture this seasonal pattern in this two types of workloads, the Figure 10 shows the contrary for this threshold settings. The best throughput results for Workload I and II were the forecast horizon size of 10 samples, and the worst ones were when the forecast horizon size was 30. When there are few samples as input window, the forecast horizon cannot be large, otherwise, it may compromise the prediction accuracy (HYNDMAN; ATHANASOPOULOS et al., 2014). We can observe that happens for Workload I and II, the worst throughput results are when the input window uses few samples and the horizon size is large.

Each row in the Table 7 indicates a combination of input window length and forecast horizon size. The bold values in the Table 7 are the highest throughput results for each workload type, and the underlined values are the lowest ones. Even though some forecast

scenarios were worse than the threshold-based technique, our proposal improved 1.67% the system's throughput for Workload I, 2.48% for Workload II, and 3.08% for Workload III, which represents a better resource usage.

Table 7 – Throughput results (request/s) for 10% and 90% thresholds.

I	H	Workload I	Workload II	Workload III
30	10	**2968.26**	3722.17	3503.19
	20	2815.50	3522.96	3462.95
	30	2647.92	3323.21	3436.87
60	10	2929.24	**3731.27**	3608.06
	20	2843.05	3636.02	3446.32
	30	2748.64	3265.70	3430.82
90	10	2931.01	3718.45	3588.52
	20	2958.49	3635.96	3618.21
	30	2846.36	3569.79	**3619.56**
Reactive	-	2919.45	3640.84	3511.27

Our hybrid method provides again the best throughput results for several types of workload. Even trying different combinations of input window length and forecast horizon size, some of them do not perform worse than the reactive threshold-based technique. Therefore, our results suggest that our proposed method can also increase the throughput of this threshold settings.

Using this set of thresholds, considering the whole experiment, the system was scaled 330 times. The most technique used to scale the system was the Holt method again, which represents 41.21% of the total scaling times of the system. The second one was the Holt-Winters method that represents 26.36%, and the third was the ARIMA that represents 24.84%. As we can observe, the times that the system was scaled decreased as we changed the set of threshold. Therefore, the set of thresholds impacts directly in how many times the system is scaled, which also impacts directly in the system performance.

It is worth pointing out that from the 30%-70% to 20%-80% threshold, there was a slight increase in the throughput. However, setting the lower threshold as 10% and the upper as 90% made the system throughput grow considerably. The 30%-70% threshold configuration does not allow the resource usage increase or decrease heavily along the execution. This configuration is very sensitive to changes, which makes the system scale more times. Consequently, it causes resource waste due to the number of times that the system scales and the resource that is not fully used. On the other hand, 20%-80% threshold setting allows a slight oscillation of CPU usage, which decreases the resource waste. It permits the resource to be more utilized before the system needs to scale out, and it makes the system scale less, which decreases the resource waste. The 10%-90% threshold takes the resource usage to the limits. Although the throughput increased considerably,

Table 8 – Times that each technique was used to scale the system.

Scenario	Workload	Reactive	Drift	SES	Holt	H.W.	ARIMA	Total
1	1	0	0	0	5	3	4	12
	2	0	0	0	3	1	4	8
	3	0	1	0	2	4	5	12
2	1	0	1	0	8	3	0	12
	2	0	0	0	2	6	4	12
	3	0	1	0	2	6	5	14
3	1	0	1	0	11	3	3	18
	2	0	1	0	8	4	3	16
	3	0	3	0	10	3	6	22
4	1	0	1	0	3	4	4	12
	2	0	0	0	5	2	1	8
	3	0	1	0	3	1	5	10
5	1	0	2	0	5	2	1	10
	2	0	1	0	3	1	5	10
	3	0	1	0	5	4	4	14
6	1	0	1	0	4	5	2	12
	2	0	0	0	10	4	6	20
	3	0	1	0	5	6	4	16
7	1	1	1	0	4	3	3	12
	2	0	1	0	3	3	1	8
	3	0	1	0	7	2	0	10
8	1	0	1	0	1	3	3	8
	2	0	1	0	4	2	3	10
	3	0	1	0	3	4	2	10
9	1	0	1	0	6	2	3	12
	2	0	0	0	11	3	0	14
	3	0	1	0	3	3	1	8
Total	-	1	24	0	136	87	82	330

it also increases the QoS violation probability because the resource usage is achieving its maximum.

Our experiments also imply that if the goal is to decrease the QoS violation probability while increasing the throughput, our hybrid method should be used with 20%-80% threshold settings. Our results support that the throughput can be considerably improved by using the thresholds as 20%-80% while decreasing the QoS violation probabilities. On the other hand, if the main goal is to improve the system throughput by increasing the resource usage, our hybrid method should be set to use the 10%-90% thresholds. To summarize, our experiments supported that our proposal improves the system throughput

while decreasing the resource waste when compared to the purely reactive threshold-based method. It is worth pointing out that the size of input window and foreccast horizon for our approach should be chosen carefully. In the Section 5.3, we suggest which would be a good choice for the size of these parameters.

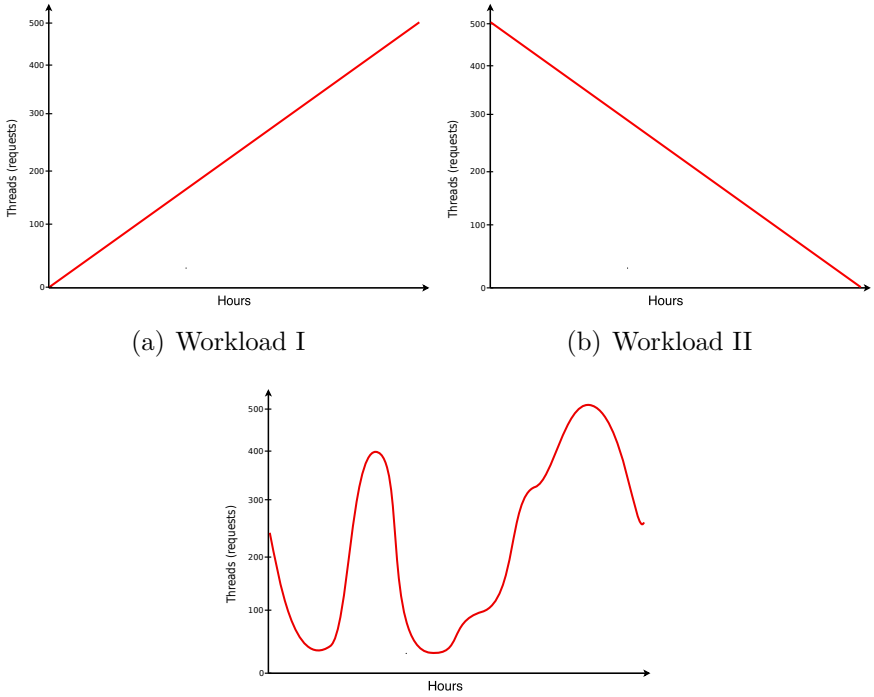
5.2 CASE STUDY 2: STATISTICAL EVALUATION

In this case study, we performed a statistical analysis to support that our strategy differs from the purely reactive technique, which means that our approach can improve the system throughput indeed. In other words, this case study helps to support that our proposal increases the system throughput. We performed a paired test between our hybrid method and the reactive threshold-based model. The experiment is composed of three scenarios, where each scenario runs two times, first time to measure our proposal, and the second time to measure the purely reactive approach. Ten samples were collected in each run, where it was one sample by an hour. We chose this interval of collection because it is enough time to stabilizated the system according the time of the workload that we tested, which were observed empiracally.

We chose the paired sample design and used paired difference test to compare if our proposal and the purely reactive approach differ from each other. In a paired design, the method of analysis takes individual differences for each pair of samples and treat the differences as a sample from a single population. The paired test is interested in the difference between our strategy and purely reactive at each plot. So we treat each sample plot as a pair of results (Y_i^p, Y_i^r), where Y_i^p corresponds to the ith sample plot of our proposal, and Y_i^r corresponds to the ith sample plot of the reactive model, then we calculate the difference (d_i), that is $d_i = Y_i^p - Y_i^r$. These differences give us a new distribution of values that has its own sample statistics. We took a 95% confidence interval for mean, then we compared against the null hypothesis.

The null hypothesis states that the mean is equal to zero, in other words, it states that there is no difference between our proposal and the reactive model. On the other hand, the alternative hypothesis states that there is a difference between our strategy and the purely threshold-based one. The population mean difference is represented by μ_d. So the following confidence interval formula for μ_d is derived assuming that the population of differences has a normal distribution

Each scenario corresponds to a type of workload. Workload I represents the increase of requests arriving in the system as the time goes by, Figure 11(a). On the other hand, the workload II represents the decreasing of requests, Figure 11(b). The Figure 11(c) depicts the workload III, which represents the variation of requests arriving over time. In other words, the workload III tries to represent a random load behavior. We chose workloads different from the Section 5.1 in order to see how the system behaves under diverse workloads. These workloads are generated by the JMeter, so they are threads that



(a) Workload I

(b) Workload II

(c) Workload III

Figure 11 – Types of workload - case study 3.

simulate users. Each user generates HTTP requests fiercely. These three workloads go up to 500 threads (users) that are making requests for 24 hours.

Table 9 shows the throughput values collected during the experiment. The columns **Proposal** correspond to the measurements of our hybrid auto-scaling approach, and the columns **Reactive** represent the measurements of the reactive threshold-based technique. The input window length and the forecast horizon size of our proposal were set as 30 and 10 respectively. As already mentioned, the lower threshold was set as 20% and the upper threshold was set as 80%.

Figure 12 depicts the behavior of our proposal against the purely reactive method. As we can observe, our proposal tends to move away from the reactive model as the time goes by. Figure 12(a) depicts it. It occurs because the average throughput increases every time the system scales using our hybrid strategy. As the resource consumption increases, our auto-scaling mechanism prepares a new virtual machine earlier than the threshold-based strategy. That impacts directly in the system throughput, which decreases the probability of QoS violations.

5.2.1 Paired difference test

The p-value gives the probability of observing the test results under the null hypothesis. The lower the p-value, the lower the probability of obtaining a result like the one that was observed if the null hypothesis was true. Thus, a low p-value indicates decreased support

Table 9 – Throughput measurements (request/s).

Workload I		Workload II		Workload III	
Proposal	**Reactive**	**Proposal**	**Reactive**	**Proposal**	**Reactive**
1325.63	1316.54	5958.62	5994.10	3080.10	3100.21
2001.27	1841.05	5568.80	5124.79	1706.63	1553.47
2427.46	2159.29	4604.51	4282.53	5335.34	4938.22
2972.31	2616.10	3605.79	3583.69	2801.17	2577.74
3332.89	2802.41	3445.48	3348.07	1783.04	1397.30
3527.67	3151.32	3169.11	3034.48	2045.27	1849.19
3824.78	3442.91	2871.31	2772.06	2197.99	1909.50
4275.12	3882.54	2654.37	2105.49	4360.78	4125.16
4911.62	4505.47	1966.19	1837.22	5834.02	5199.31
5149.26	4823.65	1532.58	1277.54	3924.15	3517.83

for the null hypothesis. The cutoff value for determining statistical significance is usually a value of 0.05. This corresponds to a 5% chance of obtaining a result like the one that was observed if the null hypothesis was true (GARDNER; ALTMAN et al., 1995). In other words, we intend to support that using a paired difference test to compare our hybrid auto-scaling technique and the purely reactive approach, we are able to reach p-values lower than 0.05 for different scenarios.

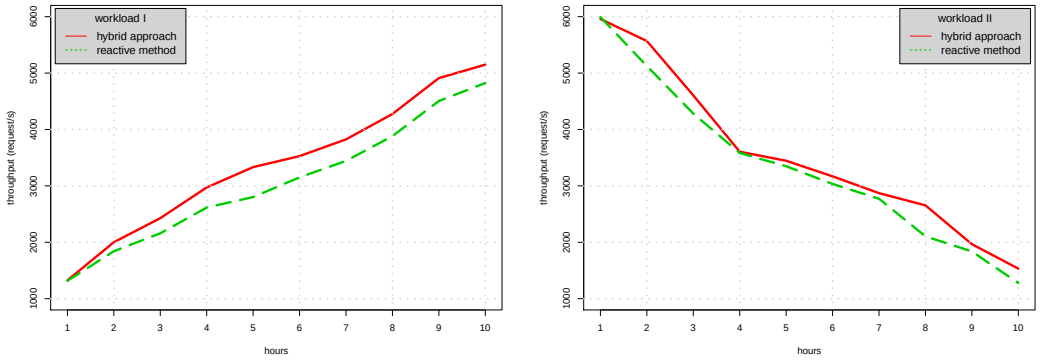
5.2.1.1 Scenario 1

Performing the paired difference test, we can compare these sets of measurements to assess whether their means differ. So using the paired test to compare our proposal's throughput against the reactive's throughput, we have the difference results in Table 10.

Calculating the confidence interval of the differences, we have the following result $CI : [216.67, 424.67]$. Therefore, we are 95% confident that the average of the throughput difference between our proposal and the reactive solution would be between 216.67 and 424.67. Zero is not in the confidence interval, then we have evidence to refute the null hypothesis for the throughput in scenario 1.

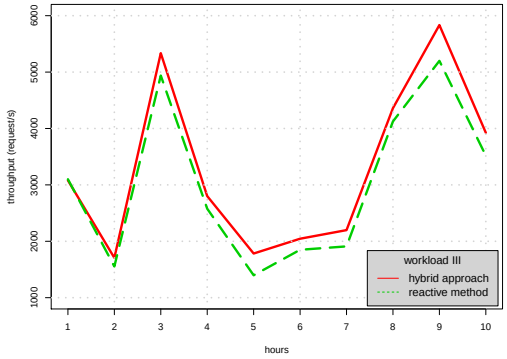
For scenario 1, we have a p-value of 0.000065, which is far too lower than 0.05. Therefore, the null hypothesis that there is no improvement using our proposal may be rejected at the 5 percent level based on the paired test.

When the workload gradually increases, we are able to observe in Table 10 in column three that the differences also increase, which suggests that as the time goes by, our proposal tends to get even better than the reactive approach. That occurs because our proposal prepares a new virtual machine to be fully operational earlier than the threshold-based technique. It impacts directly on the QoS metrics. It can also be observed in Figure 12(a), where it is visual that the difference increases over time. Anticipation



(a) Workload I

(b) Workload II



(c) Workload III

Figure 12 – Behavior of the hybrid and reactive approach.

is very important because there is always a delay from the time when an auto-scaling action is executed until it is effective. For instance, it takes several minutes to assign a physical server to deploy a VM, move the VM image to it, boot the operating system and application, have the server fully operational, and make the load balancer recognize it (LORIDO-BOTRAN et al., 2014).

Reactive threshold-based techniques might not be able to scale in case of sudden traffic bursts, such as when an E-commerce website has special offers. Therefore, proactivity might be required in order to deal with fluctuating demands and being able to scale in advance.

Figure 13 is a boxplot of the difference between the throughputs. This boxplot does not show the medians, it shows the means instead. It is important pointing out that they look similar, it occurs because there are only 10 samples. However, as the p-value is 0.000065, which is far too lower than 0.05, we are able to refute the null hypothesis.

Table 10 – Throughput (request/s) paired test (scenario 1)

	proposal	**reactive**	$d_i = Y_i^p - Y_i^r$
	1325.63	1316.54	9.09
	2001.27	1841.05	160.22
	2427.46	2159.29	268.17
	2972.31	2616.10	356.21
	3332.89	2802.41	530.48
	3527.67	3151.32	376.35
	3824.78	3442.91	381.87
	4275.12	3882.54	392.58
	4911.62	4505.47	406.15
	5149.26	4823.65	325.61
Mean	3374.80	3054.12	320.67
SD	1233.50	1136.81	145.37

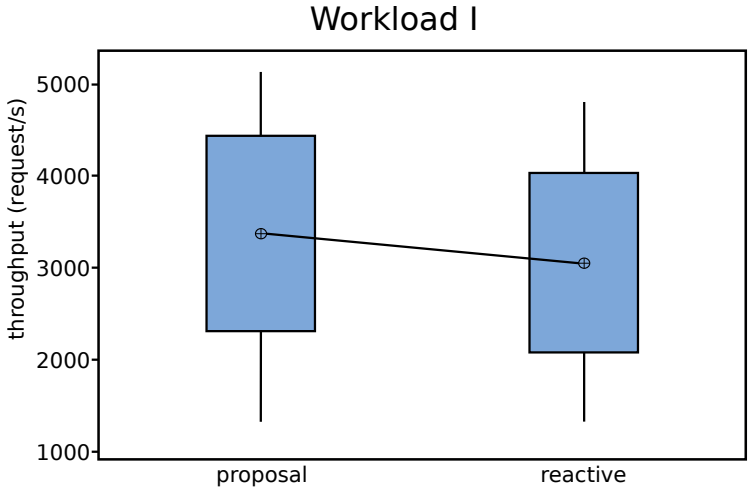


Figure 13 – Boxplot scenario 1.

5.2.1.2 Scenario 2

For scenario 2, we also used the paired test to compare our proposal's throughput against purely reactive's throughput. So we have the difference results in Table 11.

Calculating the confidence interval of the differences, we have the following result CI : $[67.58, 335.77]$. Therefore, we are 95% confident that the average difference of throughput between our proposal and the reactive threshold-based method would be between 67.58 and 335.77. Zero is not in the confidence interval, then we have evidence to refute the null hypothesis for the throughput in scenario 2.

For scenario 2, we have a p-value of 0.00784, which is lower than 0.05. Therefore, the null hypothesis that there is no improvement using our proposal may be rejected at the 5 percent level based on the paired test.

Table 11 – Throughput (request/s) paired test (scenario 2)

	proposal	**reactive**	$d_i = Y_i^p - Y_i^r$
	5958.62	5994.10	-35.48
	5568.80	5124.79	444.01
	4604.51	4282.53	321.98
	3605.79	3583.69	22.1
	3445.48	3348.07	97.41
	3169.11	3034.48	134.63
	2871.31	2772.06	99.25
	2654.37	2105.49	548.88
	1966.19	1837.22	128.97
	1532.58	1277.54	255.04
Mean	3537.67	3335.99	201.67
SD	1452.49	1476.83	187.44

Figure 14 is a boxplot of the difference between the throughputs. This boxplot does not show the medians, it shows the means instead. It is important pointing out that they look similar, it occurs because there are only 10 samples. However, as the p-value is 0.00784, which is lower than 0.05, we are able to refute the null hypothesis.

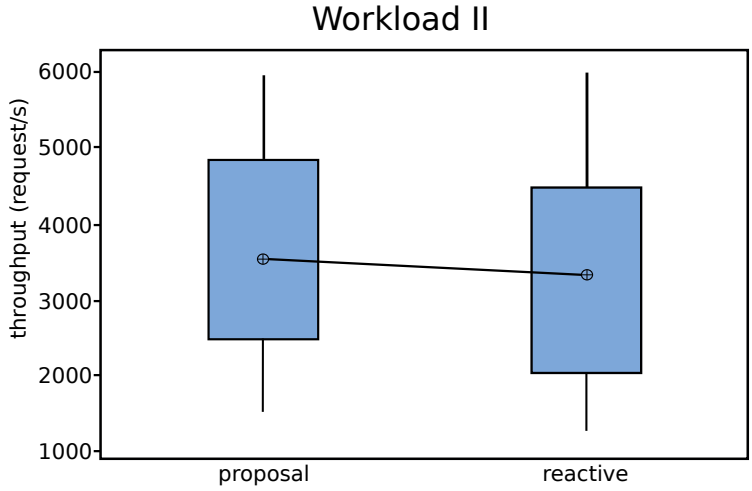


Figure 14 – Boxplot scenario 2.

5.2.1.3 Scenario 3

For scenario 3, we also used the paired test to compare our strategy of auto-scaling and the most used auto-scaling technique used by cloud providers, which is the reactive one. So we have the difference results in Table 12.

Calculating the confidence interval of the differences, we have the following result $CI : [162.95, 417.15]$. Therefore, we are 95% confident that the average difference of

Table 12 – Throughput (request/s) paired test (scenario 3)

	proposal	**reactive**	$d_i = Y_i^p - Y_i^r$
	3080.10	3100.21	-20.11
	1706.63	1553.47	153.16
	5335.34	4938.22	397.12
	2801.17	2577.74	223.43
	1783.04	1397.30	385.74
	2045.27	1849.19	196.08
	2197.99	1909.50	288.49
	4360.78	4125.16	235.62
	5834.02	5199.31	634.71
	3924.15	3517.83	406.32
Mean	3306.84	3016.79	290.056
SD	1490.86	1393.95	177.67

throughput between our proposal and the reactive model would be between 162.95 and 417.15. Zero is not in the confidence interval, then we also have evidence to refute the null hypothesis for the throughput in scenario 3.

For scenario 3, we have a p-value of 0.00059304, which is far too lower than 0.05. Therefore, the null hypothesis that there is no improvement using our proposal may be rejected at the 5 percent level based on the paired test.

Figure 15 is a boxplot of the difference between the throughputs. This boxplot does not show the medians, it shows the means instead. It is important pointing out that they look similar, it occurs because there are only 10 samples. However, as the p-value is 0.00059304, which is lower than 0.05, we are able to refute the null hypothesis.

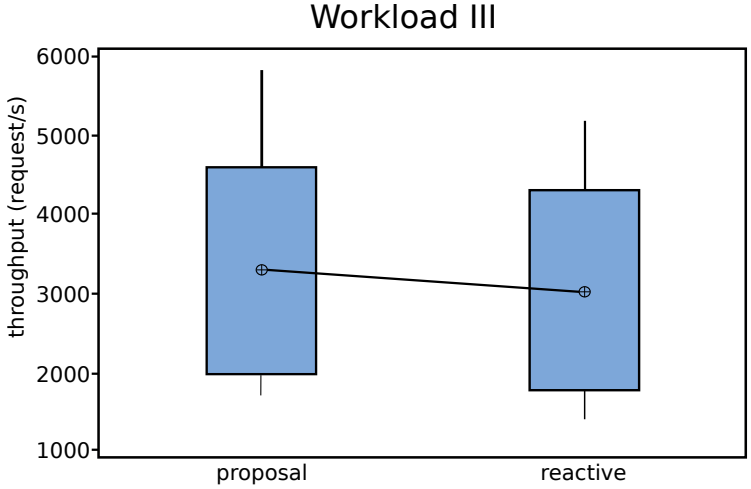


Figure 15 – Boxplot scenario 3.

After we performed and analyzed all these scenarios, we could confirm that zero is in

none of the confidence intervals that we tested, which means that we have evidence to refute the null hypothesis. In other words, there is a significant difference between the reactive and hybrid approaches, considering the specified performance metric. Then we can conclude that our proposal really increases the system throughput.

5.3 CASE STUDY 3: INPUT WINDOW LENGTH X FORECAST HORIZON SIZE

Forecasting methods are very sensitive to the input window length and the horizon forecasting size. Thus, in this case study, we performed an analysis to see which of these two components has the most impact on the forecasting models, which affect directly the throughput of the system. To investigate how the performance of a predictive system depends on the selection of an input window length and the forecast horizon size, we used the Pareto chart. The Pareto chart is used to show the influence of the independent variables in the response variable. The Pareto chart shows from the largest influence to the smallest one. These influences are based on t-statistics that test the null hypothesis that the influence is 0 (MINITAB et al., 2017).

The t-statistics allow to use data to perform a hypotheses test about an unknown population (SAMUELS et al., 2012). Each bar in the Pareto chart represents factors that influence statistically the response variable. In other words, the bars in our Pareto chart represent the influence of the input window length, the forecast horizon size and the influence of both terms combined. The values in the X axis are results from the t-statistics, and they represent the influence of each factor in the response variable. Figure 16 shows the influence of the input window length and forecast horizon size in the throughput of the system. As we can observe, the main influence in the throughput is the input window length, which suggests that the number of samples that are used to perform the predictions is the main responsible for the improvement of the system throughput.

The Figure 17 shows how the variation of input window length and forecast horizon size individually affect the throughput results. The results shown in Figure 17 are averages of the whole experiment. As we can observe, there is an increase in the throughput when the input window grows from 30 to 60 samples. However, when the input window grows to 90, there is a huge improvement in the throughput. According to these results, we may assume that setting the input window length to be 90 is the better choice, among the scenarios analyzed, for better system throughput. On the other hand, according to the Figure 17, if we increase the forecast horizon size from 10 to 20, there is a huge decline in the throughput, and if we increase the forecast horizon size to 30, the throughput keeps getting worse. So these results suggest that if we keep increasing the forecast horizon size we will not get any better results. This behavior is expected because the bigger is the forecast horizon size, the worse is the prediction accuracy, which will affect directly the system throughput.

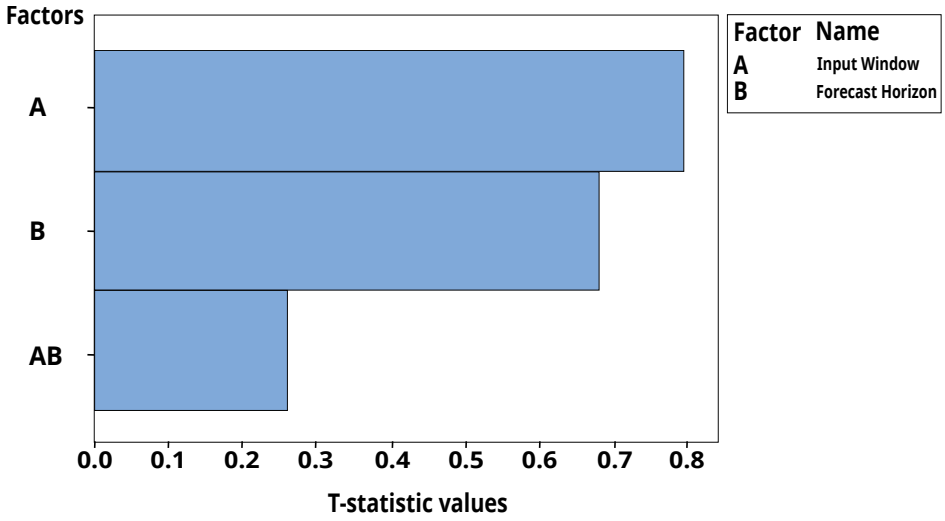


Figure 16 – The influence of the factors.

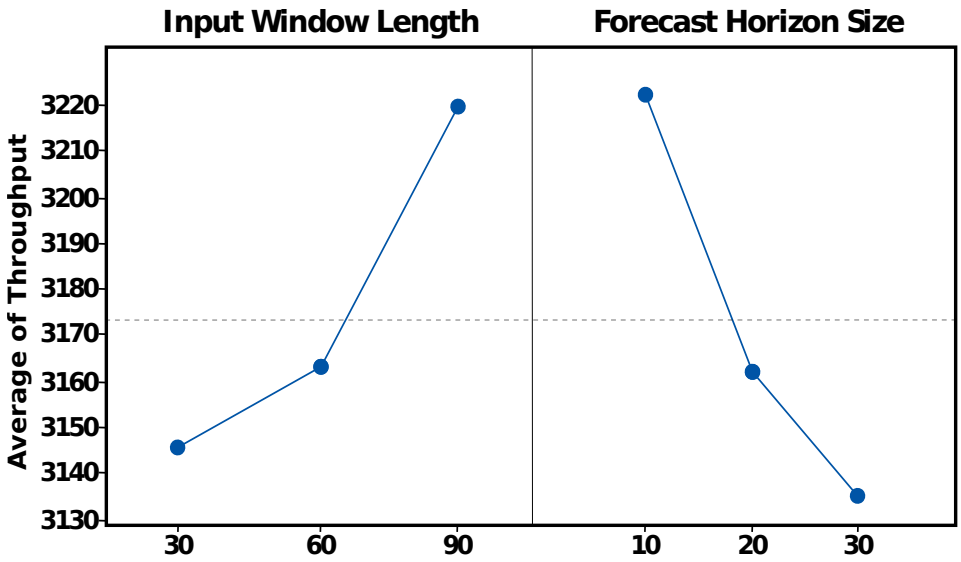


Figure 17 – Main effects of the factors.

The Figure 18 depicts the interaction effect between the input window length and the forecast horizon size. As we can observe, the interaction between these two factors is significant because the lines are not parallel (MINITAB et al., 2017). Figure 18 shows that when the input window has only 30 samples and the forecast horizon is of 30 steps ahead, the throughput results are the worst ones. That occurs because the forecast model is not able to capture the system behavior using only 30 samples and it tries to build a long-term forecast, which compromises the forecast accuracy, consequently it affects the system throughput. On the other hand, we are able to observe that using the forecast size of 10, which means a short-term forecast, the throughput increases significantly. We are also able to observe that using the forecast horizon size of 30, the input window length also needs to be increased in order to capture precisely the data behavior and finally be able

to build an accurate forecast. Figure 18 also implies that there is not a huge improvement in the system throughput when increasing the input window length while using a forecast horizon size of 10. However, when using a forecast horizon size of 30, there is a huge improvement in the system throughput when the input window length is increased.

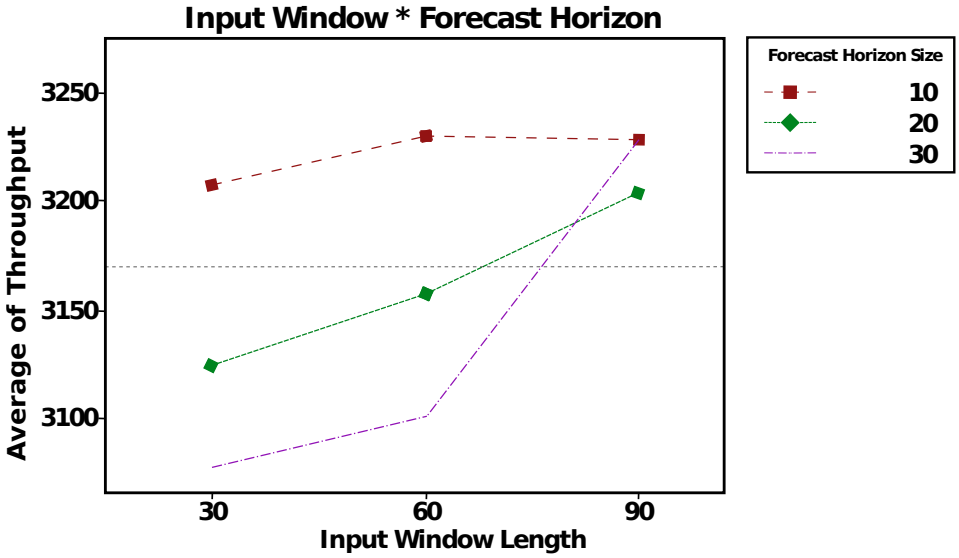


Figure 18 – Interaction effect of the factors.

The main contribution of this case study is the detailed investigation of the dependency of the system's throughput on the choice of a forecast horizon size and an input window length. The experiments discover a dependency of the system performance on the combination of the input window length and the forecast horizon. However, the input window length is even more critical for our strategy. The following pattern is observed: the highest system's throughput is achieved when the input window length is approximately equal to the 90 samples and the forecast horizon size is 10 samples. The presence of the pattern depends on the ability of forecast techniques to infer relevant information from the input data. It gives a simple solution for setting initial values for the input window length parameter depending on the forecast horizon selected. In summary, we are able to conclude that our strategy usually works better when the input window length is equal to 90 samples and the forecast horizon is equal to 10 samples.

5.4 FINAL REMARKS

This chapter presented three case studies that show the usefulness and efficacy of our strategy. The first case study proved that our strategy improves considerably the system throughput.The second case study proved that our strategy differs statically from the purely reactive approach improving the system throughput significantly. The last case study is an investigation to know which component in the forecasting methods is more important when setting up our strategy.

6 CONCLUSIONS AND FUTURE WORK

A large proportion of worldwide IT companies have adopted cloud computing for various purposes. Providers of public cloud services, as well as owners of private clouds, need to design and manage their systems to keep up with users expectations regarding performance and dependability of their infrastructures. Time series forecasting models were developed based on statistical tools and systematic methodologies. As already mentioned, there is not one best technique for all situations. In other words, one technique may be the best fit for a database and be the worst for another one, even the amount of data can influence the choice of technique. Providing better QoS while maintaining lower resource lease costs in cloud computing environments is very challenging for a reactive auto-scaling solution. Even though proactive solutions can overcome these challenges, they are dependent on highly accurate prediction mechanisms. Therefore, it is important to test some techniques to verify which one is the best in a given case.

This research achieved a number of results in the areas that it has explored, and the major contributions are the methods for improving the throughput of the system. We were able to observe that when the workload gradually increases the differences between our proposal and the reactive technique also increase, which suggests that as the time goes by, our approach tends to get even better than the reactive model. That occurs because our proposal prepares a new virtual machine to be fully operational earlier than the threshold-based technique. It impacts directly on the QoS metrics. The contributions of this study are summarized as follows.

6.1 CONTRIBUTIONS

This study presented a hybrid strategy for virtual machine provisioning because the reactive techniques are not able to cope with variations in demand without causing some instability. Five different forecasting models are presented as the basis of our proposal: Drift, Simple Exponential Smoothing, Holt, Holt-Winters, and ARIMA. The methods used by this strategy were chosen based on related works. Thus, our method is generic enough to be used by any cloud service provider. Our experiment demonstrates that the hybrid strategy can reduce the latency of provisioning cloud resource, and improve the cloud service quality through improving the system throughput. 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.

The main contribution of this study corresponds to the hybrid strategy to auto-scale virtual machines using a reactive and proactive approach, where the proactive approach has used a set of time series forecasting techniques. The proposed hybrid strategy takes

advantage of both reactive and proactive auto-scaling techniques. For instance, the system is monitored and the auto-scaling model verifies if a threshold was achieved. If it was not achieved, a forecasting process starts. The proposed hybrid strategy for virtual machine provisioning consists of two main components: CPU utilization monitoring module, and auto-scaling module. The CPU utilization monitoring module collects resource usage information of virtual machines in a fixed 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 aim of the auto-scaler is to dynamically adapt the resources assigned to the elastic applications, depending on the resource usage. Anticipation is important because there is 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. Reactive threshold-based techniques might not be able to scale in case of sudden traffic bursts, such as when an E-commerce website has special offers. Therefore, proactivity might be required in order to deal with fluctuating demands and being able to scale in advance.

In summary, the main contributions of this dissertation are:

- An auto-scaling strategy based on thresholds and time series forecasting methods;
- An algorithm for choosing the most suitable forecasting technique for a time series;
- An improvement of system's throughput and CPU utilization by using our strategy;

6.2 SUMMARY OF RESULTS AND CONSTRAINTS

In one of our experiments, we had a throughput improvement of 12.11% by using our proposal. Our experiment also demonstrates that varying the threshold values can influence the throughput of the system. Varying the threshold from the 30%-70% to 20%-80% threshold, there was a slight increase in the throughput. Setting 10%-90% threshold made the system throughput grow considerably. However, changing these thresholds may affect directly in our approach, and it may also increase the probability of SLA violation. To summarize, our experiments supported that our proposal improves the system throughput while decreasing the resource waste when compared to a threshold-based method. Our study also shows that the input window and forecast horizon size cannot be randomly chosen, because they affect directly the forecast accuracy, which consequently affects the system throughput.

We also performed a statistical evaluation in order to support that our proposal represents an improvement in the system performance. We used the paired difference test to confirm if there is a difference between our strategy and the reactive one. We were also able to observe that according to our last case study analysis, the main factor that affects

the system throughput is the input window size, however, the forecast horizon size is also very significant.

Although this work proposes an improvement in the system's throughput and our results show that is better to use our strategy, it still has several constraints. For instance, we used only one performance metric, and we did not use a public cloud to test our strategy (e.g. Amazon EC2). Another huge constraint is that the thresholds still need to be set, which is not an easy task. In other words, it is very difficult to choose optimal thresholds, and our strategy does not cover it, which means that the user still has to have previous knowledge in how to set the thresholds up. Using only CPU utilization as input for the forecasting techniques may not consider exactly what is happening in the system. Therefore, more metrics could be used, such as memory usage, network bandwidth, and so on. We could also propose a monitor in application level, where we could measure the response time, for instance.

6.3 FUTURE WORK

In Section 5.1 and Section 5.2, we chose to use different workloads in order to see how the system behaves. In Section 5.1, the workloads present a cyclic pattern, on the other hand, in Section 5.2, they do not. In future works, we intend to perform the case study in Section 5.2 using cyclic-pattern workloads. We also intend to use others metrics of resource consumption, such as memory usage, bandwidth, and so on. It is worth pointing out that we intend to use some machine learning techniques with statistical techniques in order to increase even more the forecast accuracy for auto-scaling in future works. Another future work that is very interesting is to integrate our approach to some private cloud platform, such as CloudStack and OpenStack.

REFERENCES

ADAMS, K.; AGESEN, O. et al. A comparison of software and hardware techniques for x86 virtualization. *ACM SIGARCH Computer Architecture News*, ACM, v. 34, n. 5, p. 2–13, 2006.

AKIOKA, S.; MURAOKA, Y. et al. Extended forecast of cpu and network load on computational grid. In: IEEE. *Cluster Computing and the Grid, 2004. CCGrid 2004. IEEE International Symposium on*. [S.l.], 2004. p. 765–772.

ALI-ELDIN, A.; TORDSSON, J.; ELMROTH, E. et al. An adaptive hybrid elasticity controller for cloud infrastructures. In: IEEE. *Network Operations and Management Symposium (NOMS), 2012 IEEE*. [S.l.], 2012. p. 204–212.

ANDERSON, T. W. et al. *The statistical analysis of time series*. [S.l.]: John Wiley & Sons, 2011. v. 19.

ARAUJO, J.; MATOS, R.; MACIEL, P.; MATIAS, R.; BEICKER, I. et al. Experimental evaluation of software aging effects on the eucalyptus cloud computing infrastructure. In: ACM. *Proceedings of the Middleware 2011 Industry Track Workshop*. [S.l.], 2011. p. 4.

ARMBRUST, M.; FOX, A.; GRIFFITH, R.; JOSEPH, A. D.; KATZ, R.; KONWINSKI, A.; LEE, G.; PATTERSON, D.; RABKIN, A.; STOICA, I. et al. A view of cloud computing. *Communications*, ACM, v. 53, n. 4, p. 50–58, 2010.

AUGUSTYN, D. R.; WARCHAL, L. et al. Metrics-based auto scaling module for amazon web services cloud platform. In: SPRINGER. *International Conference: Beyond Databases, Architectures and Structures*. [S.l.], 2017. p. 42–52.

BANKOLE, A. A.; AJILA, S. A. et al. Cloud client prediction models for cloud resource provisioning in a multitier web application environment. In: IEEE. *Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on*. [S.l.], 2013. p. 156–161.

BAUER, E.; ADAMS, R. et al. *Reliability and availability of cloud computing*. [S.l.]: John Wiley & Sons, 2012.

BISWAS, A.; MAJUMDAR, S.; NANDY, B.; EL-HARAKI, A. et al. Predictive auto-scaling techniques for clouds subjected to requests with service level agreements. In: IEEE. *Services (SERVICES), 2015 IEEE World Congress on*. [S.l.], 2015. p. 311–318.

BONNER, G. I. *St. Augustine of Hippo. Life and Controversies*. [S.l.]: London, 1963.

CALHEIROS, R. N.; MASOUMI, E.; RANJAN, R.; BUYYA, R. et al. Workload prediction using arima model and its impact on cloud applications' qos. *IEEE Transactions on Cloud Computing*, IEEE, v. 3, n. 4, p. 449–458, 2015.

CAMPOS, E. G. et al. Performance evaluation of auto scaling mechanisms in private clouds for supporting a web service application. *Thesis*, Universidade Federal de Pernambuco, 2015.

CHAMBERS, J. C.; MULLICK, S. K.; SMITH, D. D. et al. How to choose right forecasting technique. *Harvard business review*, HARVARD BUSINESS SCHOOL PUBLISHING CORPORATION 60 HARVARD WAY, BOSTON, MA 02163, v. 49, n. 4, p. 45, 1971.

CHE, J.; HE, Q.; GAO, Q.; HUANG, D. et al. Performance measuring and comparing of virtual machine monitors. In: IEEE. *Embedded and Ubiquitous Computing, 2008. EUC'08. IEEE/IFIP International Conference on*. [S.l.], 2008. v. 2, p. 381–386.

CHEN, W.; CHEN, J.; TANG, J.; WANG, L. et al. A qos guarantee framework for cloud services based on bayesian prediction. In: IEEE. *Advanced Cloud and Big Data, 2015 Third Int. Conf. on*. [S.l.], 2015. p. 117–124.

CHISNALL, D. et al. *The definitive guide to the xen hypervisor*. [S.l.]: Pearson Education, 2008.

CORTEZ, P.; RIO, M.; ROCHA, M.; SOUSA, P. et al. Multi-scale internet traffic forecasting using neural networks and time series methods. *Expert Systems*, Wiley Online Library, v. 29, n. 2, p. 143–155, 2012.

DUTREILH, X.; MOREAU, A.; MALENFANT, J.; RIVIERRE, N.; TRUCK, I. et al. From data center resource allocation to control theory and back. In: IEEE. *Cloud Computing (CLOUD), 2010 IEEE 3rd Int. Conf. on*. [S.l.], 2010. p. 410–417.

DUTTA, S.; GERA, S.; VERMA, A.; VISWANATHAN, B. et al. Smartscale: Automatic application scaling in enterprise clouds. In: IEEE. *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. [S.l.], 2012. p. 221–228.

FARIAS, V. A.; SOUSA, F. R.; MACHADO, J. C. et al. Auto escalonamento proativo para banco de dados em nuvem. *Simpósio Brasileiro de Banco De Dados*, p. 281–287, 2014.

FERNANDEZ, H.; PIERRE, G.; KIELMANN, T. et al. Autoscaling web applications in heterogeneous cloud infrastructures. In: IEEE. *Cloud Engineering (IC2E), 2014 IEEE International Conference on*. [S.l.], 2014. p. 195–204.

GARDNER, E. S. et al. Forecasting the failure of component parts in computer systems: A case study. *International Journal of Forecasting*, Elsevier, v. 9, n. 2, p. 245–253, 1993.

GARDNER, M. J.; ALTMAN, D. G. et al. *Statistics with confidence: confidence intervals and statistical guidelines*. [S.l.]: Brithis Medical Journal, 1995.

HIGGINS, J. J. et al. Introduction to modern nonparametric statistics. Cengage Learning, 2003.

HIRASHIMA, Y.; YAMASAKI, K.; NAGURA, M. et al. Proactive-reactive auto-scaling mechanism for unpredictable load change. In: IEEE. *Advanced Applied Informatics (IIAI-AAI), 2016 5th IIAI International Congress on*. [S.l.], 2016. p. 861–866.

HOLT, C. C. et al. Forecasting seasonals and trends by exponentially weighted moving averages. *International journal of forecasting*, Elsevier, v. 20, n. 1, p. 5–10, 2004.

HYNDMAN, R. J.; ATHANASOPOULOS, G. et al. *Forecasting: principles and practice*. [S.l.]: OTexts, 2014.

HYNDMAN, R. J.; KHANDAKAR, Y. et al. *Automatic time series for forecasting: the forecast package for R*. [S.l.]: Monash University, Department of Econometrics and Business Statistics, 2007.

HYNDMAN, R. J.; KOEHLER, A. B.; ORD, J. K.; SNYDER, R. D. et al. Prediction intervals for exponential smoothing using two new classes of state space models. *Journal of Forecasting*, Wiley Online Library, v. 24, n. 1, p. 17–37, 2005.

IQBAL, W.; DAILEY, M. N.; CARRERA, D.; JANECEK, P. et al. Adaptive resource provisioning for read intensive multi-tier applications in the cloud. *Future Generation Computer Systems*, Elsevier, v. 27, n. 6, p. 871–879, 2011.

JAIN, S.; KUMAR, R.; ANAMIKA, S. K. J. et al. A comparative study for cloud computing platform on open source software. *ABHIYANTRIKI: An International Journal of Engineering & Technology (AIJET)*, v. 1, n. 2, p. 28–35, 2014.

JAMSHIDI, P.; AHMAD, A.; PAHL, C. et al. Autonomic resource provisioning for cloud-based software. In: ACM. *Proceedings of the 9th international symposium on software engineering for adaptive and self-managing systems*. [S.l.], 2014. p. 95–104.

JIANG, J.; LU, J.; ZHANG, G.; LONG, G. et al. Optimal cloud resource auto-scaling for web applications. In: IEEE. *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*. [S.l.], 2013. p. 58–65.

JR, E. S. G. et al. Exponential smoothing: The state of the art. *Journal of forecasting*, Wiley Online Library, v. 4, n. 1, p. 1–28, 1985.

KALEKAR, P. S. et al. Time series forecasting using holt-winters exponential smoothing. *Kanwal Rekhi School of Information Technology*, v. 4329008, p. 1–13, 2004.

KAUR, G.; BALA, A.; CHANA, I. et al. An intelligent regressive ensemble approach for predicting resource usage in cloud computing. *Journal of Parallel and Distributed Computing*, Elsevier, v. 123, p. 1–12, 2019.

KHAN, M. N.; LIU, Y.; ALIPOUR, H.; SINGH, S. et al. Modeling the autoscaling operations in cloud with time series data. In: IEEE. *Reliable Distributed Systems Workshop (SRDSW), 2015 IEEE 34th Symposium on*. [S.l.], 2015. p. 7–12.

KIRAN, M.; MAIYAMA, K.; MIR, H.; MOHAMMAD, B.; OUN, A. A. et al. Agent-based modelling as a service on amazon ec2: Opportunities and challenges. In: IEEE. *Utility and Cloud Computing (UCC), 2015 IEEE/ACM 8th International Conference on*. [S.l.], 2015. p. 251–255.

LAWRENCE, M.; O'CONNOR, M.; EDMUNDSON, B. et al. A field study of sales forecasting accuracy and processes. *European Journal of Operational Research*, Elsevier, v. 122, n. 1, p. 151–160, 2000.

LORIDO-BOTRAN, T.; MIGUEL-ALONSO, J.; LOZANO, J. A. et al. A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of Grid Computing*, Springer, v. 12, n. 4, p. 559–592, 2014.

MAO, M.; HUMPHREY, M. et al. Scaling and scheduling to maximize application performance within budget constraints in cloud workflows. In: IEEE. *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*. [S.l.], 2013. p. 67–78.

MARSHALL, D. et al. Understanding full virtualization, paravirtualization, and hardware assist. *VMWare White Paper*, p. 17, 2007.

MARTIN, L.; FREI, J. et al. Forecasting supply chain components with time series analysis. In: IEEE. *Electronic Components and Technology Conference, 2003. Proceedings. 53rd*. [S.l.], 2003. p. 269–278.

MELL, P.; GRANCE, T. et al. The nist definition of cloud computing. Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology Gaithersburg, 2011.

MESSIAS, V. R.; ESTRELLA, J. C.; EHLERS, R.; SANTANA, M. J.; SANTANA, R. C.; REIFF-MARGANIEC, S. et al. Combining time series prediction models using genetic algorithm to autoscaling web applications hosted in the cloud infrastructure. *Neural Computing and Applications*, Springer, v. 27, n. 8, p. 2383–2406, 2016.

MINITAB et al. *Effects plots for Analyze Definitive Screening Design*. 2017. Disponível em: <https://support.minitab.com/en-us/minitab/18/help-and-how-to/modeling-statistics/doe/how-to/screening/analyze-screening-design/interpret-the-results/all-statistics-and-graphs/effects-plots/>.

MOORE, L. R.; BEAN, K.; ELLAHI, T. et al. Transforming reactive auto-scaling into proactive auto-scaling. In: ACM. *Proceedings of the 3rd International Workshop on Cloud Data and Platforms*. [S.l.], 2013. p. 7–12.

NIKRAVESH, A. Y.; AJILA, S. A.; LUNG, C.-H. et al. An autonomic prediction suite for cloud resource provisioning. *Journal of Cloud Computing*, Springer, v. 6, n. 1, p. 3, 2017.

NIU, D.; XU, H.; LI, B.; ZHAO, S. et al. Quality-assured cloud bandwidth auto-scaling for video-on-demand applications. In: IEEE. *INFOCOM, 2012 Proceedings IEEE*. [S.l.], 2012. p. 460–468.

PACKAGE, R. S. et al. R: A language and environment for statistical computing. *Vienna, Austria: R Foundation for Statistical Computing*, 2009.

PEARCE, D. K. et al. Short-term inflation expectations: Evidence from a monthly survey: note. *Journal of Money, Credit and Banking*, JSTOR, v. 19, n. 3, p. 388–395, 1987.

PETER, M.; TIMOTHY, G. et al. The nist definition of cloud computing. *NIST special publication*, v. 800, p. 800–14, 2011.

PORTNOY, M. et al. *Virtualization essentials*. [S.l.]: John Wiley & Sons, 2012. v. 19.

REDDY, P. V. V.; RAJAMANI, L. et al. Evaluation of different hypervisors performance in the private cloud with sigar framework. *International Journal of Advanced Computer Science and Applications*, Citeseer, v. 5, n. 2, 2014.

ROY, N.; DUBEY, A.; GOKHALE, A. et al. Efficient autoscaling in the cloud using predictive models for workload forecasting. In: IEEE. *Cloud Computing (CLOUD), 2011 IEEE Int. Conf. on*. [S.l.], 2011. p. 500–507.

SAMUELS, M. L.; WITMER, J. A.; SCHAFFNER, A. A. et al. *Statistics for the life sciences*. [S.l.]: Pearson education London, 2012.

SHARIFFDEEN, R.; MUNASINGHE, D.; BHATHIYA, H.; BANDARA, U.; BANDARA, H. D. et al. Adaptive workload prediction for proactive auto scaling in paas systems. In: IEEE. *Cloud Computing Technologies and Applications (CloudTech), 2016 2nd Int. Conf. on*. [S.l.], 2016. p. 22–29.

SHEN, H.; HUANG, J. Z. et al. Forecasting time series of inhomogeneous poisson processes with application to call center workforce management. *The Annals of Applied Statistics*, JSTOR, p. 601–623, 2008.

SHUMWAY, R. H.; STOFFER, D. S. et al. *Time series analysis and its applications: with R examples*. [S.l.]: Springer Science & Business Media, 2010.

SHYNKEVICH, Y.; MCGINNITY, T.; COLEMAN, S. A.; BELATRECHE, A.; LI, Y. et al. Forecasting price movements using technical indicators: Investigating the impact of varying input window length. *Neurocomputing*, Elsevier, v. 264, p. 71–88, 2017.

SULEIMAN, B.; VENUGOPAL, S. et al. Modeling performance of elasticity rules for cloud-based applications. In: IEEE. *Enterprise Distributed Object Computing Conference (EDOC), 2013 17th IEEE International*. [S.l.], 2013. p. 201–206.

TANENBAUM, A. S. et al. *Modern operating system*. [S.l.]: Pearson Education, Inc, 2009.

TEAM, R. C. et al. R: A language and environment for statistical computing. Vienna, Austria, 2014.

URGAONKAR, B.; SHENOY, P.; CHANDRA, A.; GOYAL, P.; WOOD, T. et al. Agile dynamic provisioning of multi-tier internet applications. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, ACM, v. 3, n. 1, p. 1, 2008.

WEI, W. W.-S. et al. *Time series analysis*. [S.l.]: Addison-Wesley publ Reading, 1994.

YANG, J.; LIU, C.; SHANG, Y.; MAO, Z.; CHEN, J. et al. Workload predicting-based automatic scaling in service clouds. In: IEEE. *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*. [S.l.], 2013. p. 810–815.

ZHANG, Q.; CHENG, L.; BOUTABA, R. et al. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, Springer, v. 1, n. 1, p. 7–18, 2010.

ZHAO, G.; LIU, J.; TANG, Y.; SUN, W.; ZHANG, F.; YE, X.; TANG, N. et al. Cloud computing: A statistics aspect of users. In: SPRINGER. *IEEE International Conference on Cloud Computing*. [S.l.], 2009. p. 347–358.

ZHAO, W.; SCHULZRINNE, H. et al. Predicting the upper bound of web traffic volume using a multiple time scale approach. In: CITESEER. *WWW (Posters)*. [S.l.], 2003.

APPENDIX A – DEPLOYMENT

To deploy our strategy, some adjustments in the environment are necessary. Understanding the hypervisor is very important to deploy our hybrid approach because each hypervisor has its own technical features. X86 operating systems are designed to run directly on the bare-metal hardware, so they naturally assume they fully own the computer hardware. The x86 architecture offers four levels of privilege known as Ring 0, 1, 2 and 3 to operating systems and applications to manage access to the computer hardware (TANENBAUM et al., 2009). While user level applications typically run in Ring 3, the operating system needs to have direct access to the memory and hardware and must execute its privileged instructions in Ring 0.

A.1 FULL-VIRTUALIZATION X PARAVIRTUALIZATION

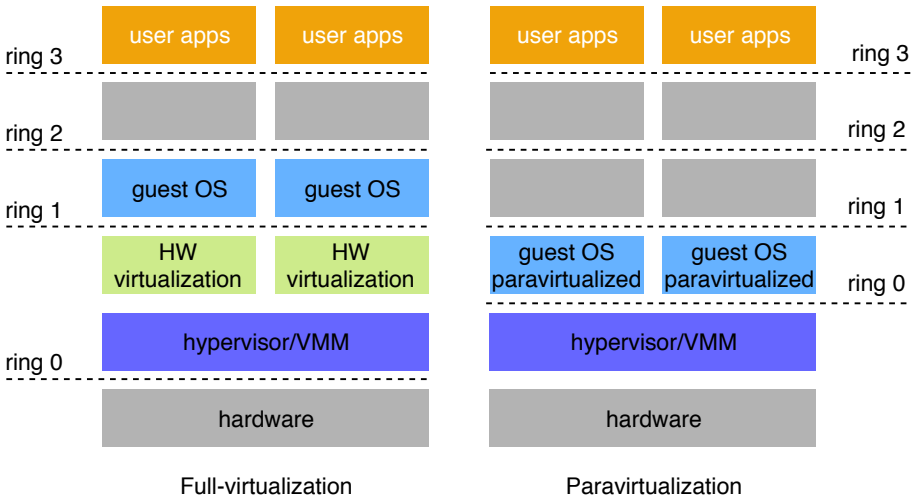


Figure 19 – Virtualization.

Virtualizing the x86 architecture requires placing a virtualization layer under the operating system, which expects to be in the most privileged Ring 0, to create and manage the virtual machines that deliver shared resources. Basically, two alternative techniques now exist for handling sensitive and privileged instructions to virtualize the x86 Architecture. Full-virtualization approach translates kernel code to replace non-virtualizable instructions with new sequences of instructions that have the intended effect on the virtual hardware (PORTNOY et al., 2012). This combination of binary translation and direct execution provides Full-virtualization as the guest OS is fully abstracted (completely decoupled) from the underlying hardware by the virtualization layer. The full-virtualization approach allows data-centers to run an unmodified guest operating system, thus maintaining the existing investments in operating systems and applications and providing a

non-disruptive migration to virtualized environments. VMware ESXi server uses a combination of direct execution and binary translation techniques to achieve full-virtualization of an x86 system (ADAMS; AGESEN et al., 2006).

Paravirtualization involves modifying the OS kernel to replace non-virtualizable instructions with hyper-calls that communicate directly with the virtualization layer hypervisor (MARSHALL et al., 2007). The hypervisor also provides hyper-call interfaces for other critical kernel operations such as memory management, interrupt handling and time keeping. The paravirtualization approach modifies the guest operating system to eliminate the need for binary translation. Therefore it offers potential performance advantages for certain workloads but requires using specially modified operating system kernels (ADAMS; AGESEN et al., 2006).

A.2 WHY *XENTOP*?

The biggest challenge involved in our monitoring module is being able to handle the CPU usage from up to thousands of several VMs running our service. To get the CPU usage in our proposal, we used a hypervisor default tool called *Xentop*, which displays real-time information about all virtual machines running in the virtualized environment. The *Xentop* is not only simple to use but also formally tested and validated, that is our motivation for adopting a default tool to monitor our system. The *Xentop* utility is included in all versions of XenServer. It displays all details about the environment. The proposed method uses a shell script to collect the resource usage and saves it in a text file. The CPU usage is saved in percentage values, which is achieved by monitoring the fourth column of the information displayed by the *Xentop* utility. Figure 20 depicts an example of the *Xentop* output.

```
xentop - 12:47:36   Xen 3.4.2
18 domains: 3 running, 14 blocked, 0 paused, 0 crashed, 0 dying, 0 shutdown
Mem: 6710304k total, 16188104k used, 50914944k free    CPUs: 16 @ 1595MHz
      NAME  STATE   CPU(sec) CPU(%)     MEM(k) MEM(%)  MAXMEM(k) MAXMEM(%) VCPUS NETS NETTX(k) NETRX(k) VBDS   VBD_OO   VBD_RD   VBD_WR SSID
ANG Exch R --b---       85    0.1    1048492    1.6    1088512      1.6     1    1        0        0    2        0        0        0    0
BUILD - Wi --b---     1035    0.8    1048492    1.6    1049600      1.6     1    1      289   586610    2        0    15404    16920    0
BUILD - Wi -----r     2027    0.8    1048492    1.6    1049600      1.6     1    1      348  3469577    2       30    12908    33521    0
BUILD - Wi --b---     2105    0.9    1048492    1.6    1049600      1.6     1    1      348  3449999    2       29    18209    47006    0
BUILD - Xe --b---     2630    2.8    1048492    1.6    1049600      1.6     1    1      367  1354336    2        0    89673    72500    0
Mer        --b---       72    0.5    1048576    1.6    1058816      1.6     1    1        0  1810252    1        2    10618     6880    0
  Domain-0 -----r    86368   23.8     771328    1.1     771328      1.1    16    0        0        0    0        0        0        0    0
DUBSLABLIC --b---     4710    3.4    1048492    1.6    1049600      1.6     2    1     2263  3927904    2     1574    49604   145602    0
TEMPLATE - --b---     6066    2.1     524204    0.8     525312      0.8     1    1    30827  2091757    2        6   130929   363509    0
TEMPLATE - --b---    10027    3.1     524204    0.8     525312      0.8     1    1    53217   358851    2      987   170706   569657    0
TEMPLATE - --b---     5182    2.6     524204    0.8     525312      0.8     1    1    27579   801035    2      130   103198   273097    0
TEMPLATE - --b---      930    4.8    1048492    1.6    1049600      1.6     2    1     4114  3010148    2        0    24577    42548    0
TEMPLATE - --b---     4745    2.7    1048492    1.6    1049600      1.6     2    1    28402   682272    2      150    47117   212051    0
TEMPLATE - --b---      624    2.6     524204    0.8     525312      0.8     1    1    95460  1614654    2        2    45185    43163    0
TEMPLATE - --b---    10214    3.8     524204    0.8     525312      0.8     1    1    44702  3245797    2        1   111662   320987    0
TEMPLATE - -----r    19873    8.2    1048492    1.6    1049600      1.6     2    1    53260   790678    2     1055   140332   474010    0
XA501-eID_ ------    16120   28.1     524204    0.8     525312      0.8     1    1    90898  2576731    2       17  1779202  1312402    0
Xen        --b---     7607    0.5     520448    0.8     524288      0.8     1    1       61   415458    1        4     3803    36798    0
```

Figure 20 – Xentop view.

A.3 WHY XENSERVER?

The Xen open source project was designed initially to support paravirtualized operating systems. While it is possible to modify open source operating systems, such as Linux and OpenBSD, it is not possible to modify closed source operating systems such as Microsoft Windows. Hardware vendors are rapidly embracing virtualization and developing new features to simplify virtualization techniques. First generation enhancements include Intel Virtualization Technology (VT-x) and AMD's AMD-V, which both target privileged instructions with a new CPU execution mode feature that allows the virtual machine manager (VMM) to run in a new root mode below ring 0. The hardware virtualization (MARSHALL et al., 2007) support enabled by AMD-V and Intel VT technologies introduces virtualization in the x86 processor architecture itself.

For this study, we chose to deploy our hybrid auto-scaling approach in a paravirtualized environment, where we chose to work with XenServer. As our hybrid approach makes intensely use of shell script, it is important that our hypervisor runs on a Linux platform. That is the main reason for choosing this hypervisor. We also chose the XenServer because it is an open-source, complete, managed server virtualization platform built on the powerful Xen Hypervisor (CHISNALL et al., 2008). Paravirtualization modifies the guest operating system so that it is aware of being virtualized on a single physical machine with less performance loss. XenServer is a complete virtual infrastructure solution that includes a 64-bit Hypervisor with live migration, full management console, and the tools needed to move applications, desktops, and servers from a physical to a virtual environment (REDDY; RAJAMANI et al., 2014).

Based on the open source design of Xen, XenServer is a highly reliable, available, and secure virtualization platform that provides near-native application performance (REDDY; RAJAMANI et al., 2014). Xen usually runs in higher privilege level than the kernels of guest operating systems. It is guaranteed by running Xen in ring 0 and migrating guest operating systems to ring 1. When a guest operating system tries to execute a sensitive privilege instruction (e.g., installing a new page table), the processor will stop and trap it into Xen (CHE et al., 2008). In Xen, guest operating systems are responsible for allocating the hardware page table, but they only have the privilege of direct read, and Xen (CHE et al., 2008) must validate updating the hardware page table. Additionally, guest operating systems can access hardware memory with only non-continuous way because Xen occupies the top 64MB section of every address space to avoid a TLB flush when entering and leaving the Hypervisor (CHE et al., 2008). Another advantage of choosing XenServer is that a single machine can act as both the Management Server and the hypervisor host, in smallest deployment. The management server controls cloud resources, as well as the administrator, can manage and interact with the management server by using a UI or APIs. Also using the XenServer hypervisor, we are able to build a huge cloud infrastructure. For instance, Amazon EC2 makes use of the XenServer hypervisor (JAIN et al., 2014).

A.4 WHY R STATISTICAL LANGUAGE?

Our hybrid approach also makes intensely use of statistical calculations, for this reason, we chose to use the R statistical language, which is a programming language, mainly dealing with the statistical computation of data and graphical representations. The various offerings of this tool include linear and non-linear modeling, classical statistical tests, time-series analysis, clustering, and graphical representation (PACKAGE et al., 2009). It can be referred to as a more integrated suite of software facilities, for the purpose of data manipulation, calculation and data visualization (TEAM et al., 2014).

There are a number of advantages of this data analytics tool, which make it so very popular among Data Scientists. First, the fact that it is by far the most comprehensive statistical analysis package available totally works in its favor (TEAM et al., 2014). This tool strives to incorporate all of the standard statistical tests, models, and analyses as well as provides for an effective language so as to manage and manipulate data. R environment can run in a Linux command-line interface, which makes it a lot of easier to integrate shell scripts with R statistical language.

The R is used to train and test the forecasting techniques in order to capture as much as possible the CPU utilization time series behavior. We use the *forecast* package, which is the most used package in R for time series forecasting. It contains functions for performing decomposition and forecasting with Exponential Smoothing, ARIMA, Moving Average models, and so forth. For aggregated data that is fairly low dimensional, as CPU utilization time series, this package should provide an adequate forecasting model (HYNDMAN; KHANDAKAR et al., 2007).

APPENDIX B – SHELL SCRIPT CODE

This is the main script in the hybrid approach. It is responsible for monitoring the environment, storing the monitor results in a file, calling the R script (Appendix C), and finally scaling or not the system. In this main script is where the user set the thresholds and the interval between collecting the samples. As this main script is a shell script, our approach is only able to run on Linux environments.

Listing B.1 – Shell script

```
1
#!/bin/bash
3 echo "Running the Autoscaler"
LB=192.168.0.205
5 PASSWORD=soeusei
NAMES[1]=vm
7 LASTIP[1]=192

9 AUX=30
LOGS=0
11 while true
    do
13      LINES=$(xentop -b -i 1 | grep vm | wc -l)
        xentop -b -i 2 | grep vm | awk '{print $4 "\t" $6}' | tail -$LINES | awk '{t
            +=1}{c+=$1}{m+=$2} END {print c/t "\t" m "\t" m/t}' >> log.txt
15
        if [ "$AUX" -gt 0 ]
17          then
                sleep 10s
19              AUX=$((AUX-1))
                continue
21
            else
23              PREDICAO=$(Rscript rscript.R $LINES)

25          if [ "$PREDICAO" == "NULL" ]
            then
27              echo "$PREDICAO NULL"

29          else

31              read VALOR LIMITE <<<"$(echo $PREDICAO)"
                #se o limite superior foi atingido cria vm, else mata vm
33              if [ "$LIMITE" == "s" ]
                then
35
                    NAMES[$((LINES+1))]=vm$LINES
37
                    xe vm-install template=vm1_ new-name-label=${NAMES[$((LINES+1))
                        ]}
39                  xe vm-start vm=${NAMES[$((LINES+1))]}
                    echo "VM: ${NAMES[$((LINES+1))]}"
41
```

```
                #ssh-keygen -R 192.168.0.205 -y
43              #LASTIP[$((LINES+1))]=$(sshpass -p $PASSWORD ssh -o
                    StrictHostKeyChecking=no $LB -l root "cat /var/lib/dhcp/
                    dhcpd.leases | grep -B 8 webserver | tail -9 | head -1 | awk
                     '{ print \$2 }'")

45              AUXIPGREP=""
                while [ -z "$AUXIPGREP" ]
47              do
                    AUXIPGREP=$(xe vm-list name-label=${NAMES[$((LINES+1))]}
                        params=networks | head -n 1 | grep "ip:")
49                  AUXIP=$(xe vm-list name-label=${NAMES[$((LINES+1))]} params=
                        networks | head -n 1 | awk '{print $5}')
                done
51
                LASTIP[$((LINES+1))]=${AUXIP:0:-1}
53              echo "NOME VM: ${NAMES[$((LINES+1))]} IP VM: ${LASTIP[$((LINES
                    +1))]}"

55              #atualizar apache ports.conf e /var/www/html/index.html
                ssh-keygen -R ${LASTIP[$((LINES+1))]} -y
57              sshpass -p $PASSWORD ssh -o StrictHostKeyChecking=no ${LASTIP[$
                    ((LINES+1))]} -l root sed -i "5c\Listen\ ${LASTIP[$((LINES
                    +1))]}\:80" /etc/apache2/ports.conf

59              sshpass -p $PASSWORD ssh -o StrictHostKeyChecking=no ${LASTIP[$
                    ((LINES+1))]} -l root "echo 'vm$LINES' > /var/www/html/index
                    .html"

61              sshpass -p $PASSWORD ssh -o StrictHostKeyChecking=no ${LASTIP[$
                    ((LINES+1))]} -l root sudo service apache2 start
                sshpass -p $PASSWORD ssh -o StrictHostKeyChecking=no ${LASTIP[$
                    ((LINES+1))]} -l root sudo service apache2 restart
63

65              #atualizar loadbalancer
                sshpass -p $PASSWORD ssh -o StrictHostKeyChecking=no $LB -l root
                     sed -i "5i\ server\ ${LASTIP[$((LINES+1))]}\;" /etc/nginx/
                    sites-available/default
67
                sshpass -p $PASSWORD ssh -o StrictHostKeyChecking=no $LB -l root
                     service nginx restart
69
                #apagando logs gigantes
71              sshpass -p $PASSWORD ssh -o StrictHostKeyChecking=no $LB -l root
                     rm -rf /var/log/nginx/access.log*

73              mv log.txt log.txt$LOGS
                LOGS=$((LOGS+1))
75

77              AUX=30

79              echo "LEVANTOU VM: $(date)"

81          elif [ "$LIMITE" == "i" ]
            then
```

```
83                      LINES=$(xentop -b -i 1 | grep vm | wc -l)
                        if [ "$LINES" -gt 2 ]
85                      then
                            echo "NOME VM: ${NAMES[$LINES]}  IP VM: ${LASTIP[$LINES]}"
87                          VMUUID=$(xe vm-list is-control-domain=false name-label=${
                                NAMES[$LINES]} --minimal | cut -d ',' -f1)
                            xe vm-shutdown uuid=$VMUUID force=true
89                          xe vm-uninstall uuid=$VMUUID force=true

91                          sshpass -p $PASSWORD ssh -o StrictHostKeyChecking=no $LB -l
                                root sed -i "/${LASTIP[$LINES]}/d" /etc/nginx/sites-
                                available/default
                            sshpass -p $PASSWORD ssh -o StrictHostKeyChecking=no $LB -l
                                root service nginx restart
93
                            mv log.txt log.txt$LOGS
95                          LOGS=$((LOGS+1))

97                          AUX=30

99                          echo "MATOU VM: $(date)"
                        fi
101                 fi
                fi
103         fi

105         sleep 30s
    done
```

APPENDIX C – R SCRIPT CODE

This is the R script in the hybrid approach. It is responsible for the reactive and the proactive scale decisions. After it calculates if there is a need to scale the system, it returns the decision for the main script (Appendix B).

Listing C.1 – R script

```
library(forecast)
library(tseries)

args <- commandArgs(TRUE)

forecasting <- function(){
  dados <- read.table("/home/paulinho/Development/C/log_segundos.txt", header=FALSE,
       sep = "", dec=".", skipNul = TRUE, blank.lines.skip = TRUE)
  if(nrow(dados) >= 30){


    base <- ts(tail(dados[,1], 30), frequency = 2)

    min_time <- min(time(base))
    max_time <- max(time(base))

    treino <- window(base, start = min_time, end = as.numeric(min_time + (((max_time
         - min_time)/100)*80)))
    teste <- window(base, start = as.numeric(min_time + (((max_time - min_time)/100)
        *80)), end = max_time)


    m_arima <- forecast(auto.arima(treino), h = 10)
    m_drift <- rwf(treino, drift = TRUE, h = 10)
    m_ses <- forecast(ets(treino, model = "ANN", damped = FALSE), h = 10)
    m_holt <- forecast(ets(treino, model = "AAN", damped = FALSE), h = 10)
    m_hw <- forecast(ets(treino, model = "AAA", damped = FALSE), h = 10)


    acuracia_mape <- c(0,0,0,0,0)
    acuracia_mape[1] <- accuracy(m_arima, teste)[5]
    acuracia_mape[2] <- accuracy(m_drift, teste)[5]
    acuracia_mape[3] <- accuracy(m_ses, teste)[5]
    acuracia_mape[3] <- accuracy(m_holt, teste)[5]
    acuracia_mape[4] <- accuracy(m_hw, teste)[5]

    #caso o threshold nao tenha sido cruzado pela predicao, entao testa se foi pelas
         amostras
    aux = 0
    if(which.min(acuracia_mape) == 1){
      aux2 = 0
      aux3 = as.numeric(args[1])
      m_arima <- forecast(auto.arima(base), h = 10)

      if(m_arima$mean[length(m_arima$mean)] >= 80){
```

```
      cat(paste(m_arima$mean[length(m_arima$mean)], "s"))
      aux = 1
      aux2 = 1
      aux3 = 3
    }else if(m_arima$mean[length(m_arima$mean)] <= 20){
      cat(paste(m_arima$mean[length(m_arima$mean)], "i"))
      aux = 1
      aux2 = 1
    }

    if(aux2 > 0 && as.numeric(aux3) > 2){
      invisible(setwd("/home"))

      invisible(write.table(acuracia_mape, "/home/accuracy_value.txt", sep="\t",
          append = TRUE, col.names = FALSE))
      invisible(pdf(file=paste(Sys.time(),"_arima", ".pdf", sep="")))
      invisible(plot(m_arima, type = "l", ylim = c(0,100), xlab = "minutes", ylab
          = "CPU USAGE"))
      invisible(lines(base, type = "l"))
      invisible(abline(h=80, col="red"))
      invisible(abline(h=20, col="red"))
      invisible(dev.off())
    }

  }else if(which.min(acuracia_mape) == 2){
    aux2 = 0
    aux3 = as.numeric(args[1])
    m_drift <- rwf(base, drift = TRUE, h = 10)

    if(m_drift$mean[length(m_drift$mean)] >= 80){
      cat(paste(m_drift$mean[length(m_drift$mean)], "s"))
      aux = 1
      aux2 = 1
      aux3 = 3
    }else if(m_drift$mean[length(m_drift$mean)] <= 20){
      cat(paste(m_drift$mean[length(m_drift$mean)], "i"))
      aux = 1
      aux2 = 1
    }

    if(aux2 > 0 && as.numeric(aux3) > 2){
      invisible(setwd("/home"))

      invisible(write.table(acuracia_mape, "/home/accuracy_value.txt", sep="\t",
          append = TRUE, col.names = FALSE))
      invisible(pdf(file=paste(Sys.time(), "_drift", ".pdf", sep="")))
      invisible(plot(m_drift, type = "l", ylim = c(0,100), xlab = "minutes", ylab
          = "CPU USAGE"))
      invisible(lines(base, type = "l"))
      invisible(abline(h=80, col="red"))
      invisible(abline(h=20, col="red"))
      invisible(dev.off())
    }

  }
   else if(which.min(acuracia_mape) == 3){
     m_ses <- forecast(ets(base, model = "ANN", damped = FALSE), h = 10)
```

```
96
        s <- m_ses$mean <= 80
98      i <- m_ses$mean <= 20

100     intersect_s <- which(diff(s)!=0)[1]
        intersect_i <- which(diff(i)!=0)[1]
102     if(!is.na(intersect_s)){
          limiar_s <- time(m_ses$mean)[(intersect_s + 1)]
104       cat(paste(limiar_s, "s"))
        }else if(!is.na(intersect_i)){
106       limiar_i <- time(m_ses$mean)[(intersect_i + 1)]
          cat(paste(limiar_i, "i"))
108     }

110     if(!is.na(intersect_s) || !is.na(intersect_i)){
          invisible(write.table(acuracia_mape, "/home/accuracy_value.txt", sep="\t",
              append = TRUE, col.names = FALSE))
112
          invisible(setwd("/home"))
114
          invisible(pdf(file=paste("ses_",Sys.time(), ".pdf", sep="")))
116       invisible(plot(m_hw, type = "l", ylim = c(0,100), xlab = "minutes", ylab =
              "CPU USAGE"))
          invisible(lines(base, type = "l"))
118       invisible(abline(h=80, col="red"))
          invisible(abline(h=20, col="red"))
120       invisible(dev.off())
        }
122
    }
124 else if(which.min(acuracia_mape) == 4){
      aux2 = 0
126   aux3 = as.numeric(args[1])
      m_holt <- forecast(ets(base, model = "AAN", damped = FALSE), h = 10)
128
      if(m_holt$mean[length(m_holt$mean)] >= 80){
130     cat(paste(m_holt$mean[length(m_holt$mean)], "s"))
        aux = 1
132     aux2 = 1
        aux3 = 3
134   }else if(m_holt$mean[length(m_holt$mean)] <= 20){
        cat(paste(m_holt$mean[length(m_holt$mean)], "i"))
136     aux = 1
        aux2 = 1
138   }

140   if(aux2 > 0 && as.numeric(aux3) > 2){
        invisible(setwd("/home/paulinho/Development/resultados"))
142
        invisible(write.table(acuracia_mape, "/home/accuracy_value.txt", sep="\t",
            append = TRUE, col.names = FALSE))
144     invisible(pdf(file=paste(Sys.time(), "_holt", ".pdf", sep="")))
        invisible(plot(m_holt, type = "l", ylim = c(0,100), xlab = "minutes", ylab =
            "CPU USAGE"))
146     invisible(lines(base, type = "l"))
        invisible(abline(h=80, col="red"))
148     invisible(abline(h=20, col="red"))
```

```
      invisible(dev.off())
    }

  }else if(which.min(acuracia_mape) == 5){
    aux2 = 0
    aux3 = as.numeric(args[1])
    m_hw <- forecast(ets(base, model = "AAA", damped = FALSE), h = 10)

    if(m_hw$mean[length(m_hw$mean)] >= 80){
      cat(paste(m_hw$mean[length(m_hw$mean)], "s"))
      aux = 1
      aux2 = 1
      aux3 = 3
    }else if(m_hw$mean[length(m_hw$mean)] <= 20){
      cat(paste(m_hw$mean[length(m_hw$mean)], "i"))
      aux = 1
      aux2 = 1
    }

    if(aux2 > 0 && as.numeric(aux3) > 2){
      invisible(setwd("/home"))

      invisible(write.table(acuracia_mape, "/home/accuracy_value.txt", sep="\t",
          append = TRUE, col.names = FALSE))
      invisible(pdf(file=paste(Sys.time(), "_hw", ".pdf", sep="")))
      invisible(plot(m_hw, type = "l", ylim = c(0,100), xlab = "minutes", ylab = "
          CPU USAGE"))
      invisible(lines(base, type = "l"))
      invisible(abline(h=80, col="red"))
      invisible(abline(h=20, col="red"))
      invisible(dev.off())
    }

  }

  if(aux == 0){
    current_sample <- tail(base, n=1)
    aux2 = 0
    aux3 = as.numeric(args[1])
    if(current_sample >= 80){
      cat(paste(current_sample, "s"))
      aux2 = 1
      aux3 = 3
    }else if(current_sample <= 20){
      cat(paste(current_sample, "i"))
      aux2 = 1
    }

    if(aux2 > 0 && as.numeric(aux3) > 2){
      invisible(setwd("/home"))

      invisible(pdf(file=paste(Sys.time(), "_base", ".pdf", sep="")))
      invisible(plot(base, type = "l", ylim = c(0,100), xlab = "minutes", ylab = "
          CPU USAGE"))
      invisible(abline(h=80, col="red"))
      invisible(abline(h=20, col="red"))
      invisible(dev.off())
```

```
        }
204
206
      }
208
    }else{
210   print("dados insuficientes")
212 }
}
214
forecasting()
```