Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH Open Access

Efficient resource provisioning for elastic Cloud services based on machine learning techniques

Rafael Moreno-Vozmediano[1*], Rubén S. Montero[1], Eduardo Huedo[1] and Ignacio M. Llorente[1,2]

Abstract
Automated resource provisioning techniques enable the implementation of elastic services, by adapting the available resources to the service demand. This is essential for reducing power consumption and guaranteeing QoS and SLA fulfillment, especially for those services with strict QoS requirements in terms of latency or response time, such as web servers with high traffic load, data stream processing, or real-time big data analytics. Elasticity is often implemented in cloud platforms and virtualized data-centers by means of auto-scaling mechanisms. These make automated resource provisioning decisions based on the value of specific infrastructure and/or service performance metrics. This paper presents and evaluates a novel predictive auto-scaling mechanism based on machine learning techniques for time series forecasting and queuing theory. The new mechanism aims to accurately predict the processing load of a distributed server and estimate the appropriate number of resources that must be provisioned in order to optimize the service response time and fulfill the SLA contracted by the user, while attenuating resource over-provisioning in order to reduce energy consumption and infrastructure costs. The results show that the proposed model obtains a better forecasting accuracy than other classical models, and makes a resource allocation closer to the optimal case.

Keywords: Cloud computing, Elasticity, Auto-scaling, Machine learning

Introduction
Service elasticity is a common feature offered by many cloud platforms and virtualized data-centers. This can be defined as the ability to adapt the system to workload changes, by autonomously provisioning and deprovisioning resources, so that at each point in time, the available resources match the current service demand as closely as possible [1]. The advantages of using service elasticity mechanisms are twofold. On the one hand, it provides Quality of Service (QoS) to the users, which can be expressed using different service metrics, such as response time, throughput (e.g., requests/s), service availability, and so on, depending on the service type. The QoS levels agreed between the service provider and user are defined by means of Service Level Agreements (SLAs), in such a way that service level failures can result in cost penalties for the service provider and a potential loss of clients. On the other hand, service elasticity enables power consumption to be reduced, by avoiding resource over-provisioning. Over-provisioning is a typical and simple solution adopted by many service providers to satisfy peak demand periods and guarantee QoS during the service lifetime. However, this results in a waste of resources that remain idle most of the time, with the consequent superfluous power consumption and CO2 emissions. The use of service elasticity mechanisms enables a reduction in the number of resources needed to implement the service and, along with other efficient techniques for server consolidation, virtual machine allocation, and virtual machine migration, it can lead to important energy savings for the datacenter or cloud provider [2–4].

Cloud providers often implement elasticity by using auto-scaling techniques [5]. These make automated scaling decisions based on the value of specific performance metrics, such as hardware metrics (e.g. CPU or memory usage) or service metrics (e.g., queue length, service throughput, response time, etc.). Auto-scaling mechanisms can be classified as reactive and proactive.

*Correspondence: rmoreno@ucm.es
[1]Computer Science School, Complutense University, 28040 Madrid, Spain
Full list of author information is available at the end of the article

Reactive mechanisms are continuously monitoring the system, and trigger a particular scaling action when a specific condition is met (e.g., provisioning or removing a given number of resources when a particular metric is higher or lower than a particular threshold). The main problem with reactive mechanisms is that the reaction time (time elapsed from the detection of the trigger condition until the resources are ready for use) can be insufficient to avoid the overloading of the system; furthermore, these mechanisms can cause system instability due to the continuous fluctuation of allocated resources. In contrast, proactive (or predictive) mechanisms try to predict the amount of resources needed during the next time period, based on statistical or mathematical models of observed workloads and system metrics. Although most existing cloud platforms and providers use reactive models, there is a great deal of research on predictive models based on time series analysis, queuing theory, reinforcement learning, or control theory, among other aspects [6].

Time series analysis has been widely used to implement auto-scaling mechanisms for applications that exhibit some kind of temporal patterns. Most of these proposals (e.g., [7–10]) use linear statistical methods for time-series forecasting, mainly based on Box and Jenkins [11] autoregressive models (e.g., AR, ARMA, ARIMA, or ARMAX) for predicting service metrics (e.g., the server load) from historical observations. However, as these service metrics can exhibit non-linear patterns, some key features of the input data may not be properly captured by these linear models. Some other studies [12, 13] use nonlinear regression models based on neural networks. The main inconvenience of these methods is the difficulty of designing the network topology of the neural network so that it is efficient, in addition to the training of algorithms, which can be slow, and can get stuck in local minima. Regarding the limitations of these methods, in this work we propose a novel predictive auto-scaling mechanism based on Machine Learning (ML) techniques for time series forecasting, in particular, the Support Vector Machine (SVM) regression technique [14, 15], combined with queue theory for modeling the system performance. The main advantage of the SVM regression model is that it fits well to input data with both linear and nonlinear patterns, and always reaches a unique global solution, with reasonable training times.

This auto-scaling mechanism is aimed at achieving an accurate prediction of the load of an elastic cloud service (e.g., a web server cluster or data stream processing server), as shown in Fig. 1. This figure represents a typical elastic web server cluster, consisting on a service front-end, acting as load balancer, and a variable number of backend servers that process users requests. The auto-scaling mechanisms should allow the system to dynamically adapt to workload changes, by autonomously provisioning and de-provisioning resources (i.e., backend servers), so that at each point in time, the available resources match the current service demand as closely as possible. More specifically, we propose the use of SVM regression to predict the server's processing load (requests/s) based on historical observations, and then we model the performance of the system using a M/M/c queue model [16] to determine the optimal number of resources (backend servers) that must be allocated to satisfy the predicted server demand and fulfill the SLAs (e.g., response time), while trying to avoid excessive resource over-provisioning, thereby reducing energy consumption and infrastructure costs.

We have compared the proposed ML-based auto-scaling mechanism with other classical forecasting mechanisms, including prediction based on last value, the moving average model, and the linear regression model. Our results show that the SVM regression model displays better forecasting accuracy than the classical models, and facilitates better resource allocation, closer to the optimal case.

The main contributions of this paper are the following:

- A novel auto-scaling method based on ML techniques aimed at optimizing the service latency (response time) and reducing over-provisioning of elastic cloud services.
- A SVM regression model for predicting the server's processing load.
- Optimal selection of SVM regression model parameters based on an analytical method.
- A queue-based performance model for determining the number of resources that must be provisioned based on the predicted load.
- An evaluation using load data from a real server.

This paper is organized as follows: Related work section includes the related work on dynamic resource provisioning mechanisms for providing service elasticity. The ML-based forecasting techniques and the performance model are described in Time series forecasting using machine learning techniques section and Performance model section. Evaluation section evaluates the accuracy of ML-based forecasting methods and the resulting resource allocation decisions. Finally, Conclusion and Future Work section summarizes the main conclusions of the paper.

Related work

There are many different proposals and implementations of techniques for dynamic resource provisioning for providing service elasticity to different kind of distributed services, such as web servers [7–9, 17, 18], computing clusters and grids [19–23], big-data clusters [24–26], and so on, which are based on different auto-scaling mechanisms. Besides the time series forecasting methods

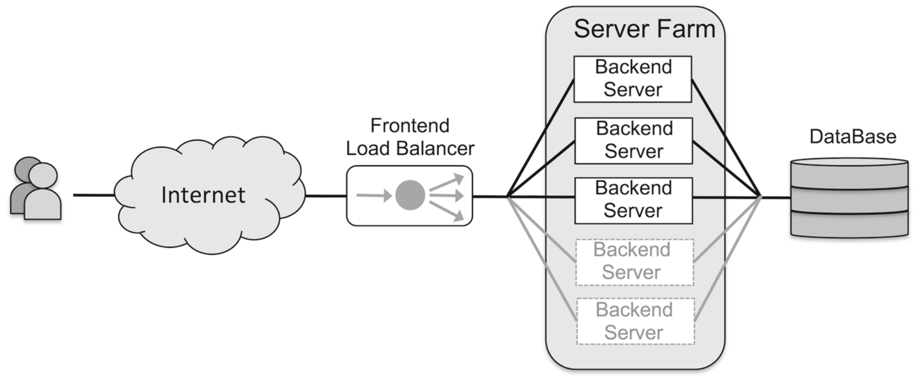

Fig. 1 Elastic cloud service. This figure represents a typical elastic web server cluster, consisting on a service front-end, acting as load balancer, and a variable number of backend servers that process users requests. The cloud auto-scaling mechanisms allow to dynamically adapt the system to workload changes, by autonomously provisioning and deprovisioning resources (i.e., back-end servers), so that at each point in time, the available resources match the current service demand as closely as possible

mentioned above, there are many other proposals and realizations of auto-scaling strategies based on many different mechanisms, such as threshold-based policies, control theory, reinforcement learning, or queuing theory, among others.

Threshold-based mechanisms are reactive auto-scaling algorithms implemented by various cloud providers and platforms (e.g. Amazon EC2, RightScale, and OpenNebula [27]) that enable users to define scaling-up and scaling-down policies or rules based on different metrics. These rules are defined in terms of specific upper and/or lower thresholds for the selected metric, so that if the metric is over (or under) the established threshold for a given time interval, it triggers a scaling action by adding (or removing) a given amount of resources from the infrastructure. Some research works have proposed certain improvements to the basic threshold-based mechanisms, for example Hasan et al. [28] propose the use of auto-scaling policies based on four threshold values, allowing finer autoscaling decisions than if only two thresholds are used; Chieu et al. [29] suggest an extension of the RightScale method based on the number of active sessions, so that to trigger the provisioning of a new instance, the number of active sessions in all instances must exceed a certain threshold.

There are several auto-scaling mechanism proposals based on the concepts of control theory. They usually implement a controller that is responsible for maintaining the output of the system (e.g., the throughput or the latency of the system) at a specific level, by adjusting the control input (e.g., the number of allocated resources). Most control-based systems are reactive mechanisms, for example Lim et al. [30] propose extending the cloud platform with an external feedback controller that enable users to automate the resource provisioning, and introduce the concept of proportional thresholding, a new control policy that takes into account the coarse-grained actuators provided by resource providers; and Padala et al. [31] also use a feedback control system to dynamically allocate resources to applications running on virtualized infrastructure. This is based on a MIMO (multi-input, multi-output) resource controller that determines appropriate allocations of multiple resources to achieve application-level SLOs. However, there are also some proposals related to proactive control-based auto-scaling mechanisms; for example, Roy et al. [9] propose a predictive solution based on a look-ahead optimization controller. This iteratively solves an optimization problem over a predefined horizon, taking into account current and future constraints, by combining the control-based solution with a time series mechanism, based on autoregressive moving average forecasting, to predict the workload of the application. A major drawback of control theory approaches is the difficulty of selecting the correct gain parameters for the model, as these may cause system instability if they are not adequate.

There are also several auto-scaling mechanisms based on Reinforcement Learning (RL) techniques, a type of automatic decision-making approach. The main component of RL systems is a decision-making agent that learns from experience, and decides on the best action to execute (e.g., adding or removing resources) to obtain a maximum reward (e.g., to maximize application throughput, or minimize response time). According to Dutreilh et al. [32], RL approaches are well-suited to autonomic resource allocation in clouds as they do not require the a priori knowledge of the application performance model, but rather learn it as the application run. However, they have to face several problems such as: having good policies in the early phases of learning, time for the learning to converge to an optimal policy, and coping with changes in the application performance behavior over time. These authors propose to deal with these problems using appropriate initialization for the early stages of learning, convergence speedup techniques to reduce the error and improve learning time, and performance model change detection. Other studies have also addressed these problems, for example, Tesauro et al. [33] propose a hybrid approach than combines the strengths of both RL and queuing models, in which RL trains offline on data collected while a queuing model policy controls the system, in order to avoid suffering potentially poor performance in live online training; and Barrett et al. [34] use a RL algorithm known as Q-learning to implement optimal scaling policies in cloud environments, and propose a parallel version of the Q-learning algorithm to reduce the execution time.

Queueing theory can also be used to implement autoscaling mechanisms, by using a queue model of the system and making decisions based on different optimization parameters, such as average queue length, average queue time, or average response time. For example, Salah et al. [35] propose a Markov chain analytical model, based on a finite queueing system, to provide elasticity for cloud-hosted applications; and Kaur and Chana [36] propose a QoS-aware resource elasticity framework, modeled as a closed-form queuing network model, which implements a proactive technique for estimating the elasticity level of machines required at each tier of the application.

Queuing theory has also been combined with other techniques, such as time-series prediction [7, 8], to improve the quality of auto-scaling mechanisms. This combined approach has also been utilized in this work: we first use an efficient time series model, based on SVM regression, to predict the load of a distributed server, and, based on these predictions, we propose a queue performance model to make a near-optimal resource provision for the server. This auto-scaling approach is well-suited to applications that exhibit either linear or non-linear temporal patterns, it is easy to implement as it does not require the development of complex controllers or decision making agents, and it requires a reasonably short execution time to make auto-scaling decisions.

Time series forecasting using machine learning techniques

The auto-scaling method proposed in this work is based on forecasting the load of a distributed server, so that we can estimate the optimal number of resources that must be allocated to satisfy the predicted demand in order to optimize the service response time and reduce overprovisioning. The forecasting method uses time series modeling, i.e., based on past observations of the server load we develop an appropriate model to describe the structure of the series, and we use this model to predict future values.

Time-series analysis is a broad discipline that has been applied to many different fields, such as business, economics, finance, science, and engineering. There are many different methods for time-series modeling and forecasting, although some of the most popular are the statistical methods developed by Box and Jenkins [11], such as the ARMA and ARIMA models. The main advantage of these models is their flexibility and simplicity when representing several varieties of time series, as these characteristics make them quick and easy to use. They do, however, present an important limitation due to their linear behavior; this makes them inadequate in many practical situations. More recently, different methods for time series forecasting based on ML techniques have been proposed [37, 38], including Artificial Neural Networks (ANNs) [39, 40] and Support Vector Machine (SVM) methods [41, 42], which have inherent nonlinear-modeling capabilities. ANN methods are inspired by biological systems, and try to learn from experience to provide generalized results based on their knowledge; ANNs are data-driven and self-adaptive methods that do not make a priori assumptions about the models or data distributions. The main drawback of ANN methods is that they can suffer from multiple local minima, and do not provide a unique global solution. In contrast, SVMs are nonlinear and used for classification, regression, and time-series prediction based on the structural risk minimization principle. SVMs map the input data into a high-dimensional space using nonlinear mapping, and then perform a linear regression of this space. The main advantage of SVM is that the solution obtained is always unique and globally optimal. In this work we use the SVM method for time series forecasting.

Time series training data

A time series is a set of time dependent observations of one or more variables of a system. For example, for a distributed service, such as a web server cluster or a data stream processing server, we take the values of the hourly average system load (measured in requests/s) for N time periods, resulting in a time series $s = \{s_1, s_1, \ldots, s_N\}$, $s_i \in \mathbb{R}, \forall i \in \{1, N\}$. The goal of the forecasting method is to predict the value of this variable for the subsequent time periods, i.e., $s_{N+1}, s_{N+2}, \ldots$; this is known as the *forecasting horizon*.

Before applying the SVM technique for time series forecasting, the observed data must be modeled as input/output pairs, called *training sets*, by splitting the time series into windows of lagged variables of size T, i.e., for each training input window $x_m = \{s_{m-T-1}, \ldots, s_{m-2}, s_{m-1}\}$, the corresponding training output is $y_m = s_m$.

The time series training set is therefore defined by $M = N - T - 1$ input/output pairs $(x_m, y_m)_{m=1\ldots M}$, which can also be expressed in matrix form, as follows:

- The $[M \times T]$ input training matrix:

$$X = \begin{bmatrix} s_{N-1} & s_{N-2} & \cdots & s_{N-T-1} \\ s_{N-2} & s_{N-3} & \cdots & s_{N-T-2} \\ \vdots & \vdots & \ddots & \vdots \\ s_T & s_{T-1} & \cdots & s_1 \end{bmatrix}$$

- The $[M \times 1]$ output training vector:

$$Y = \begin{bmatrix} s_N \\ s_{N-1} \\ \vdots \\ s_{T+1} \end{bmatrix}$$

The support vector regression model

A detailed description of SVM theory and its applications [41, 43–46] is beyond the scope of this paper, so this section highlights the main elements of the SVM model for time series forecasting used in this work, which is based on the sequential minimal optimization algorithm for SVM regression [14, 15].

SVM is a machine learning technique that learns from nonlinear input training data using a linear learner. For this purpose, the SVM regression maps the input data into a high-dimensional feature space via nonlinear mapping using a *kernel function*. Then, a linear regression model is used to regress in the new feature space. In the case of time series forecasting, where we have a training set of M input/output pairs $(x_m, y_m)_{m=1...M}$, the SVM regression can approximate the value of the time series at time t, using the following function:

$$\hat{y}_t = b + \sum_{m=1}^{M} w_m \times K(x_t, x_m) \quad (1)$$

where b is a constant (bias term); w_m are the weight factors ($W = \{w_1, w_2, \ldots, w_M\}$is the weight vector); x_t is the time series data window at time t; and K is the kernel function.

The goal of the SVM algorithm is to find the optimal weight vector, W, that minimizes the regularized risk, R_{reg}, defined as follows:

$$R_{reg} = \frac{1}{2}\sum_{m=1}^{M} w_m^2 + C\sum_{m=1}^{M} L_\epsilon(y_m, \hat{y}_m) \quad (2)$$

The first term of the risk function enforces flatness in the feature space, by penalizing the model complexity. The second term is the Vapnik ϵ-insensitive loss function [46], which measures the empirical error between the model estimation $(\hat{y}_m)$ and the real data (y_m), penalizing those errors larger than $\pm\epsilon$, and is defined as follows:

$$L_\epsilon(y_m, \hat{y}_m) = \begin{cases} |y_m - \hat{y}_m| - \epsilon & \text{if} |y_m - \hat{y}_m| > \epsilon \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

Constant C in Eq. 2 (with $C > 0$) modulates the trade-off between the model flatness and the amount of tolerated deviations larger than ϵ. The optimal values of both ϵ and C are data dependent, and have to be chosen by the user. However, there are some analytical methods that can help to select these parameters [47].

To cope with errors larger than ϵ, the slack variables, ξ and ξ^* can be introduced into the model. These represent the functional distance of two possible, but mutually exclusive samples. So, the expression of regularized risk to be minimized in Eq. 2 can therefore be reformulated as follows:

$$R_{reg} = \frac{1}{2}\sum_{m=1}^{M} w_m^2 + C\sum_{m=1}^{M}(\xi_m + \xi_m^*)$$
$$\text{subject to} \begin{cases} y_m - \hat{y}_m - b \le \epsilon + \xi_m^* \\ \hat{y}_m + b - y_m \le \epsilon + \xi_m \\ \xi_m, \xi_m^* \ge 0 \end{cases} \quad (4)$$

The minimization of Eq. 4 is a standard problem of minimization with constraints, which can be solved by applying Lagrangian theory. Then, the weight vector, W, can be obtained from Lagrange multipliers, α_m and α_m^*, which are associated with a specific training point:

$$w_m = \alpha_m - \alpha_m^*, \forall m \in \{1, M\}$$
$$\text{subject to} \begin{cases} \sum_{m=1}^{M}(\alpha_m - \alpha_m^*) = 0 \\ \alpha_m, \alpha_m^* \in [0, C] \end{cases} \quad (5)$$

Based on the Karush-Kuhn-Tucker conditions [44], only a reduced number of coefficients α_m and α_m^* will be non-zero, and the training points associated with these parameters refer to the model support vectors.

Kernel functions

The kernel function [15, 48] transforms the nonlinear input space into a high-dimensional feature space. In this space, the problem can be solved as a linear problem. Some of the most common kernel functions are the following:

- *Polynomial kernel.* One of the most common polynomial kernels is:

$$K(x, x') = (x \cdot x' + 1)^p \quad (6)$$

  where $x \cdot x'$ is the dot product of feature vectors x and x', and $p \in \mathbb{N}$ is the exponent of the kernel, chosen by the user.
- *Normalized polynomial kernel.* The normalized polynomial kernel is a variant of the polynomial kernel, which can defined as:

$$K(x, x') = \frac{(x \cdot x' + 1)^p}{||x||||x'||} \quad (7)$$

  where $||x||$ and $||x'||$ are the Euclidean norms of vectors x and x', respectively.

- *RBF kernel.* The radial basis function (RBF) or Gaussian kernel is defined as:

$$K\left(x,x'\right) = e^{-\gamma||x-x'||^2} \tag{8}$$

  where $||x - x'||^2$ represent the Euclidean distance between feature vectors x and x', and $\gamma \in \mathbb{R}$ is a user-defined parameter.

In the literature we can find many other kernel functions, such as the Fourier kernel [46], the Pearson VII function-based kernel (PUK) [49], and the multilayer perceptron kernel [50], among others. However, in this work we will use the above-defined basic kernels.

Forecast accuracy measures

Different error measures can be used to evaluate the accuracy of the forecasting models. Some of the most common error measures are the following:

- Mean Absolute Error (MAE):

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{9}$$

- Mean Squared Error (MSE):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2 \tag{10}$$

- Root Mean Squared Error (RMSE):

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2} \tag{11}$$

If the previous accuracy measures are applied to the training data set of the time series, they provide an estimation of how the forecasting model fits these historical data, i.e., they measure the expected or estimated prediction error of the forecasting model. On the other hand, if we apply these accuracy measures to the forecast data within the forecasting horizon, assuming we know the real values of the time series for this period, we obtain the real prediction error made by the forecasting model.

Performance model

The time series forecasting method described in the previous section enables load predictions to be made for an elastic cloud service based on historical observations. The next challenge to address is the development of an accurate performance model, based on these predictions, to decide the optimal number of resources (i.e., backend servers) that must be allocated to the system, in order to fulfill the SLAs contracted with the users. The system performance model proposed in this work is based on queuing theory.

We assume that the server, as shown in Fig. 1, has a single front-end entry point for all the users, and the various client requests are distributed to different parallel backend servers using a load balancer. This distributed server can be modeled as a M/M/c queue [16], as shown in Fig. 2.

The M/M/c queue model is based on the following parameters:

- c is the *number of parallel servers* in the system.
- λ is the *arrival rate*, i.e., the average number of requests that reach the system per time unit, modeled as a Poisson distribution.
- μ is the *service rate*, i.e., the average number of requests that a server can process per time unit.

And the following performance measures:

- ρ is the *system utilization factor*, which is defined as follows:

$$\rho = \frac{\lambda}{c\mu} \tag{12}$$

- W_q is the *average queue time*, i.e., the time a user request is waiting in the system queue before being processed, which can be computed as:

$$W_q = \frac{c^c\rho^{c+1}P_0}{\lambda c!\,(1-\rho)^2} \tag{13}$$

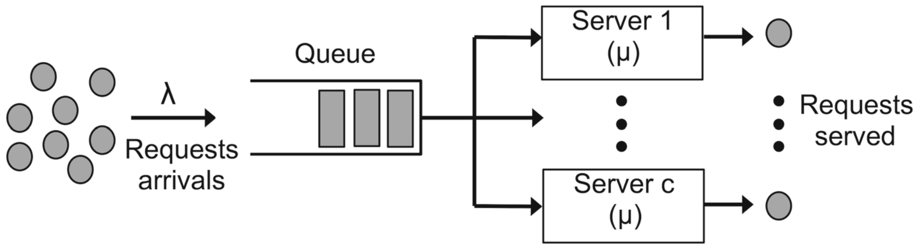

Fig. 2 M/M/c queue model for the distributed server.This is a graphical representation of a M/M/c queue model, with c parallel servers ($c > 1$), each serving user requests. All requests reaching the system join a single queue, and they are served in their order of arrival. λ is the arrival rate, i.e., the average number of requests that reach the system per time unit, modeled as a Poisson distribution. μ is the service rate, i.e., the average number of requests that a server can process per time unit

where P_0 is the probability of the system being idle (i.e., no requests in the queue), which can be computed as:

$$P_0 = \left(\frac{c^c \rho^c}{c!\,(1-\rho)} + \sum_{n=1}^{c-1} \frac{(c\rho)^n}{n!} \right)^{-1} \tag{14}$$

- W is the *average response time*, i.e., the time a user must wait for his request to be processed, which can be computed as:

$$W = W_q + \frac{1}{\mu} \tag{15}$$

For the system to be stable, ρ must be less than 1. So, the goal of the performance model is to find, for each time period, the minimum number of servers (c) that maintain system stability ($\rho < 1$) while satisfying the response time (W) contracted by the user in the SLA.

In this study, we have considered an auto-scaling period of one hour, i.e., the auto-scaling system predicts the average server load for the following hour, and then, based on this prediction and using the M/M/c queuing model, it adjusts the number of resources assigned to the server for this time period. We have chosen this auto-scaling period (one hour) for two main reasons: firstly, the continuous fluctuation of allocated resources can cause system instability, similarly to reactive auto-scaling models; and secondly, many infrastructure providers (e.g. cloud providers) use a minimum charging period of one hour, so if we allocate new resources to the system, it makes no sense to withdraw them within the next one-hour period. However, the proposed auto-scaling mechanisms could work with different auto-scaling periods, according to the system requirements.

Evaluation

In this section, we first present the parameters and results of the SVM-based forecasting model for predicting the load of an elastic cloud service based on historical load observations (input training data set). Then, using these predictions and the M/M/c queue performance model, we show the estimated resource allocation results (i.e., the number of backend servers) for the distributed server. To prove the accuracy of the forecasting models we use a test data set, which allows us to compare the predicted server loads and estimated allocation results with the real server loads and optimal resource allocations for this test interval. In this work we compare the proposed SVM-based forecasting methods with some basic forecasting methods (namely, based on last-value, moving average, and linear regression), but not against other auto-scaling approaches proposed in the literature. This comparison is not feasible in most cases, because most of the proposals lack sufficient information to reproduce the proposed auto-scaling method and its results, such as the parameters of the input model, the workloads used to feed the model or even a detailed description of the model itself. In addition, because different auto-scaling methods use very different metrics and objective functions, it may not always be possible to compare them.

The input data chosen to train the forecasting model and evaluate our proposal were obtained from real web service logs from the Complutense University of Madrid. These data were gathered over a four-week period on an hourly basis. To emulate a server with high data traffic load, the data collected have been extrapolated one order of magnitude. These input training data, summarized in Fig. 3, represent the hourly average load (expressed in requests per second) of the server.

Parameter selection for the SVM forecasting model

The basic parameters of a time-series forecasting model (see "Time series training data" section) are: the size of the training data (N), the lag period (T), and the forecasting horizon. The training data set must be large enough to capture the time series behavior, so in this work we have collected data from a 4-week period, i.e., $N = 672$ hours. To choose an appropriate lag period, we analyzed the seasonal patterns of the time series, by measuring the autocorrelation of the input training data, shown in Fig. 4. As we can see, the input data exhibit a clear autocorrelation for a lag interval of 24 h. For this reason, the chosen lag period was $T = 24$ hours. Finally, the forecasting horizon chosen in our model was one hour, i.e., based on the last N observations, the forecasting model predicts the value of the time series for the next hourly period. We chose this horizon because, in general, the accuracy of the prediction worsens as the forecasting horizon increases, so it is more accurate to apply the forecasting model and make a new prediction each hour. Furthermore, in this work, we obtained these hourly predictions for a test interval of 24 h. For this test interval, the real values of the time series were known, so we were able to compare the predicted values with the real values, allowing us to validate the forecasting model.

Besides these basic parameters, the SVM regression model presented in The support vector regression model section uses some additional configuration parameters that must be adjusted by the user. In particular, the C parameter of the regularized risk in Eq. 2, the ϵ parameter of Vapnik loss function (Eq. 3), and the γ parameter of the RBF kernel function (Eq. 8) must be optimally selected to obtain good estimation accuracy. In the literature there are many different approaches for selecting these parameters [47, 51–53]. In this work we have used the analytical methods proposed by Cherkassky and Ma [47].

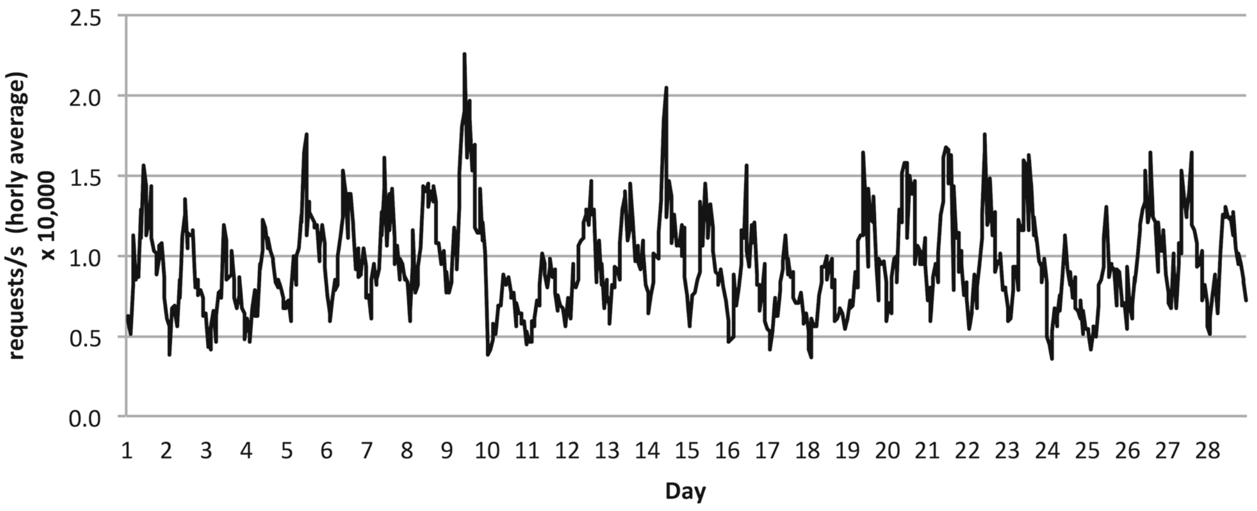

Fig. 3 Summary of input training data. This graph displays the input data used for the training of the SVM-based forecasting model presented in this work. This data were obtained from real web service logs from the Complutense University of Madrid and were gathered over a four-week period on an hourly basis. To emulate a server with high data traffic load, the data collected have been extrapolated one order of magnitude. These input training data represent the hourly average load (expressed in requests per second) of the server for a 4-week period (i.e., $N = 672$ hours)

The value of the regularization parameter, C, can be related to the range of response values in the training data. Therefore, according to [47], it can be chosen as follows:

$$C = max\left(|\bar{y} + 3\sigma_y|, |\bar{y} - 3\sigma_y|\right) \tag{16}$$

where $\bar{y}$ and σ_y are the mean and standard deviation of the y values of training data.

On the other hand, the ϵ parameter should be proportional to the input noise level and should also depend on the number of training samples. According to [47], it can be chosen as follows:

$$\epsilon = 3\sigma\sqrt{\frac{\ln n}{n}} \tag{17}$$

where n is the number of samples in the training input data, and σ is the estimated noise variance observed from the training data, which can be obtained by fitting the input data using a low-bias model, such as a linear

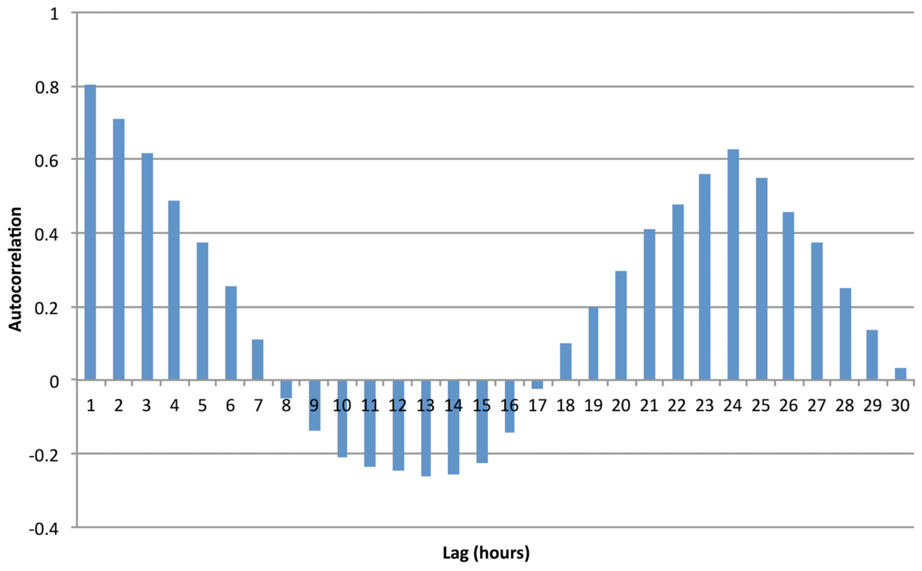

Fig. 4 Autocorrelation analysis of input training data. To choose an appropriate lag period, we analyzed the seasonal patterns of the time series, by measuring the autocorrelation of the input training data for different lag values, as shown in this figure. This was achieved using the Excel autocorrelaction function (r_k, where k is the lag interval), which is computed, as $r_k = s_k/s_0$, where s_k is the autocovariance function at lag k, and s_0 is the variance of the time series. As we can see in this figure, the input data exhibit a clear autocorrelation for a lag interval of 24 h. For this reason, the chosen lag period was $T = 24$ hours

estimator. In our case, we used a first order linear regression model to estimate the y values of the training data, so the estimated noise variance can be computed as follows:

$$\sigma^2 = \frac{1}{n-1} \sum_{i=1}^{n} \left(y_i - \hat{y}_i\right)^2 \tag{18}$$

Finally, the γ parameter of RBF kernel function should be selected to reflect the range of the input training data, and can be chosen as follows:

$$\gamma \sim [0.1 - 0.5] \times range(x) \tag{19}$$

where $range(x) = max(x) - min(x)$ for the input training data.

Using the input data shown in Fig. 3, and applying the previous formulation for the SVM regression model parameters, we obtained the values displayed in Table 1.

Forecasting results

The prediction models presented in this work forecast the average hourly load of a distributed server for a 24 h test interval, based on the historical data shown in Fig. 3, using the parameters specified in the previous section.

The experimental environment used in this work to run the SVM regression models is based on the WEKA tool [54] from Waikato University, with the time series analysis package.

The results of this section are intended to prove that the SVM regression model outperforms other simpler forecasting methods. For this reason, we compared the behavior of the SVM-based models with the following three simple forecasting methods:

- *Forecasting model #1* (based on the last value). The estimated value of the server load in the current time interval is equal to the value in the previous time interval, i.e., $\hat{y}(t) = y(t-1)$.
- *Forecasting model #2* (based on a simple moving average). The estimated value of the server load in the current time interval was computed as the moving average (MA) of order three (i.e., an $MA(p)$ model, with $p = 3$).
- *Forecasting model #3* (based on linear regression). The estimated value of the server load in the current time interval was computed using an autoregressive (AR) model with a lag period of 24 h (i.e., an $AR(p)$ model, with $p = 24$)

In addition to these basic methods, we also evaluated the SVM-based forecasting models using three different kernel functions:

- *Forecasting model #4* (SVM with a polynomial kernel). This is based on SVM regression with a polynomial kernel (see Eq. 6) of order $p = 1$. We empirically tested higher order polynomials kernels, but they offered no better results than order $p = 1$, and took more time to execute.
- *Forecasting model #5* (SVM with a normalized polynomial kernel). This is based on SVM regression with a normalized polynomial kernel (see Eq. 7) of order $p = 2$. As in the previous case, polynomials kernels of higher orders did not outperform order $p = 2$, and took more time to execute.
- *Forecasting models #6 to #8* (SVM with a RBF kernel). This is based on SVM regression with a RBF kernel (see Eq. 8). According to Table 1, the optimal values for the γ parameter are between 0.2 and 1.0, so we executed the forecasting algorithm using three different values of this parameter: low value ($\gamma = 0.2$), medium value ($\gamma = 0.6$), high value ($\gamma = 1.0$), corresponding to forecasting models #6, #7, and #8, respectively.

A summary of all the forecasting methods used in this work is shown in Table 2.

Figure 5 shows the prediction results for the 24 h test interval using the various forecasting methods compared to the real values of the average server load for the same hours. The error bars of the different graphs represent the expected error of the model computed as the RMSE of the training data for each time interval.

Figure 6 shows the accuracy of the different forecasting models for the 24 h test interval, where the MAE, MSE, and RMSE values represent the real prediction error of each forecasting model for this period.

Table 1 Selected values for the SVM parameters

Parameter	Value
C	1.9
ϵ	0.027
γ	$[0.2 - 1.0]$

Table 2 Summary of different forecasting models

Forecasting Model	Method	Parameters
#1	Last Value	
#2	Moving Average	$MA(3)$
#3	Linear Regression	$AR(24)$
#4	SVM - Polynomial Kernel	$C = 1.9, \epsilon = 0.027, p = 1$
#5	SVM - Normal. Polyn. Kernel	$C = 1.9, \epsilon = 0.027, p = 2$
#6	SVM - RBF Kernel	$C = 1.9, \epsilon = 0.027, \gamma = 0.2$
#7	SVM - RBF Kernel	$C = 1.9, \epsilon = 0.027, \gamma = 0.6$
#8	SVM - RBF Kernel	$C = 1.9, \epsilon = 0.027, \gamma = 1.0$

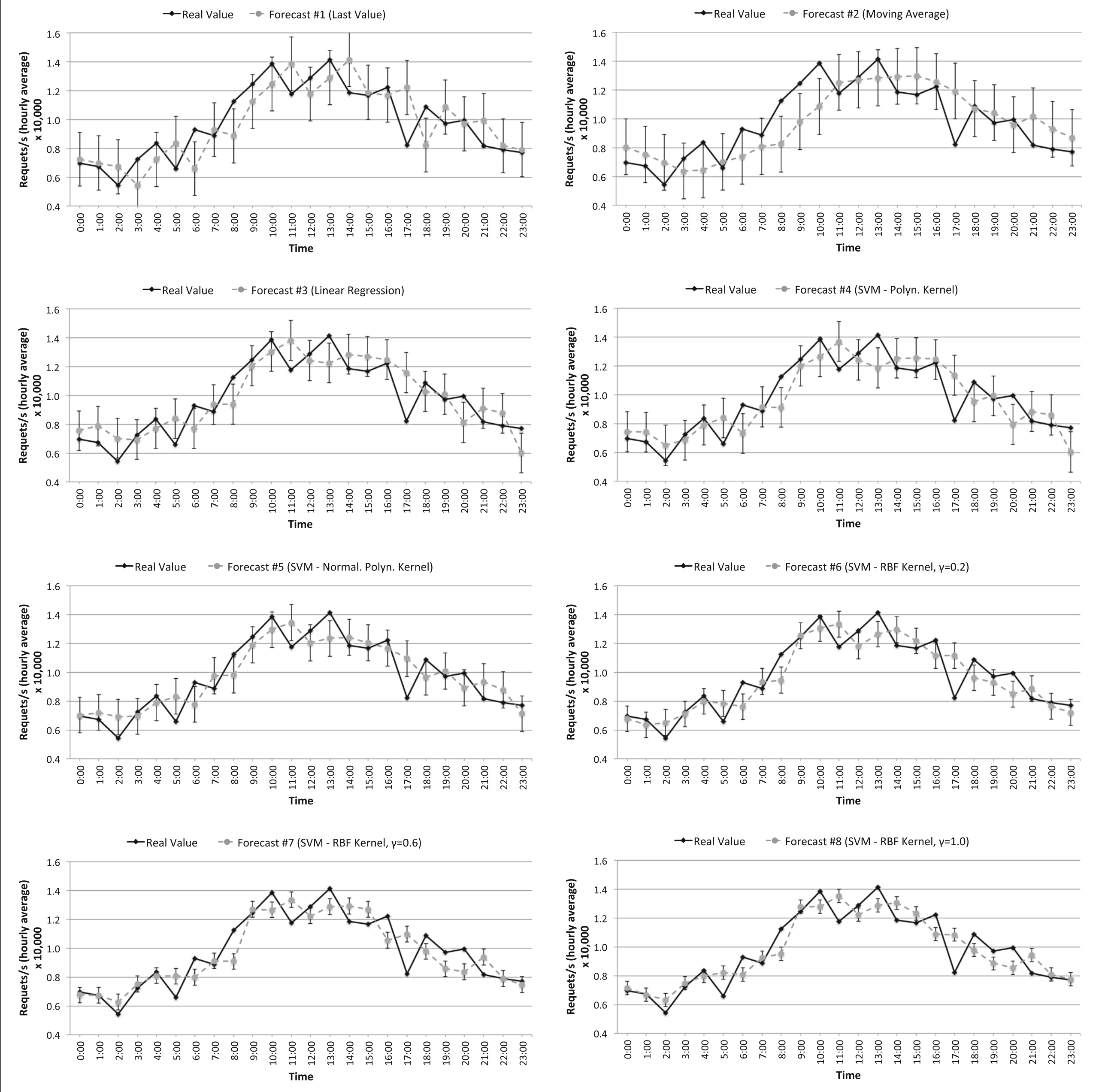

Fig. 5 Predicted server loads obtained with different forecasting models (#1 to #8). This figure shows the prediction results for the 24 h test interval using the various forecasting models (#1 to #8) compared to the real values of the average server load for the same hours. The bar errors of the different graphs represent the expected error (RMSE) of the model. This is computed as the error between the estimated data obtained by the forecasting model for the training period and the real training data of the time series

As we can see, SVM-based forecasting (models #4 to #8) is more accurate than the three basic methods (models #1 to #3), as prediction errors (MAE, MSE, and RMSE) are lower in all cases. In addition, if we compare the SVM-based methods, the RBF kernel with $\gamma = 0.2$ and $\gamma = 1.0$ (models #6 and #8) obtains better results than the polynomial kernels.

Resource allocation results

Once the server load predictions had been obtained, based on the different forecasting models, we were able to apply the M/M/c queue performance model presented in Performance model section to obtain the number of resources (i.e., number of backend servers) that must be provisioned in order to satisfy

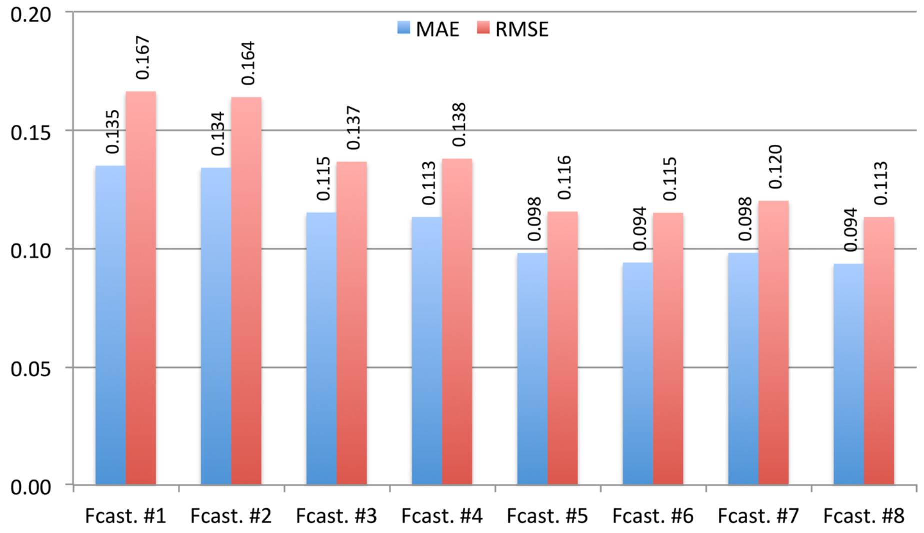

Fig. 6 Accuracy of different forecasting models (real prediction error for the forecasting period) This figure shows the accuracy of the different forecasting models for the 24 h test interval (forecasting horizon), where the MAE, MSE, and RMSE values represent the real prediction error of each forecasting model for this period. They are computed as the error between the data obtained by the forecasting model and the real data of the time series for the forecasting horizon

the expected load and fulfill the SLA contracted by the user, expressed in terms of maximum response time.

In addition, as we know the real hourly load of the server for the 24 h test interval, we were able to compare the optimal number of resources that should be provisioned based on the real load, called *optimal allocation*, with the estimated resource allocation based on the forecasted loads, called *estimated allocation*. We were therefore able to determine, for each forecasting model and each hourly period, whether the server was being overprovisioned (where *estimated allocation* > *optimal allocation*), under-provisioned (where *estimated allocation* < *optimal allocation*)), or correctly provisioned (where *estimated allocation* = *optimal allocation*).

To apply the M/M/c model, we first defined the parameters to be used in the model, summarized in Table 3:

- The arrival rate (λ) is the average load of the server (expressed in requests/s) each hour. We considered the predicted load obtained for each one of the forecasting models for the 24 h test interval, as well as the real load of the server for the same period.
- The service rate (μ) is the number of requests that each backend server can process per time unit. In this work, we assumed a value of $\mu = 200$ requests/s for each backend server. This is a typical throughput value of a mid-range server (e.g., an Amazon EC2 medium instance) serving dynamic content requests (e.g., PHP) [55, 56].
- The system utilization factor (ρ) must be less than 1 to guarantee system stability. The number of provisioned resources must be sufficient to guarantee this condition.
- The average queue time (Wq) and the average response time ($W = 1/\mu + W_q$), which are limited by the SLA contracted by the user. In this work, we assumed that the user SLA established a limit value for Wq that could not exceed 50% of the minimum response time, i.e., $W_q \leq 0.5 \times 1/\mu = 2.5$ ms. Hence, the maximum response time (W) imposed by the SLA was 7.5 ms.

Table 3 M/M/c queueing model parameters

Param.	Meaning	Value
λ	Arrival rate	Real or forecast server load (requests/s)
μ	Service rate	200 requests/s
ρ	System utilization	< 1
W	Average response time	≤ 7.5 ms
W_q	Average queue time	≤ 2.5 ms

Next, using the real and forecast server load values obtained by the different forecasting models as input (i.e. real and forecast λ values), we applied the M/M/c queuing model to determine the number of resources (backend servers) that must be provisioned, in order to guarantee system stability ($\rho < 1$) and fulfill the maximum response time imposed by the SLA ($W \leq 7.5$ ms).

To measure the goodness of the different forecasting models, we will use three different metrics: i) the number of provisioned resources; ii) the number of SLA violations; and iii) the number of unserved requests. We considered that the SLA of a request is violated when its response time is $W > 7.5$ ms. In addition, we also established a maximum limit (time-out) of 1 s for serving a request, so that if the response time exceeds this limit, the request is considered as unserved. Form the point of view of the user, the optimal resource allocation is the one that minimize the number of resources, and hence the cost of the infrastructure, while also minimize the number of SLA violations, and the number of unserved request. When the number of provisioned resources is too low (under-provisioning) the cost of the infrastructure decreases, but the number of SLA violations and unserved requests increases. On the other hand, if the number of provisioned resources is too high (over-provisioning), the number of SLA violations and unserved requests would be negligible, but the cost of the infrastructure shoots up.

Table 4 shows the hourly results of the provisioning for the 24 h test interval. The *Optimal* column shows the optimal allocation results based on the real load of the server. Columns *"#1"* to *"#8"* show the estimated allocation results computed from the predicted server loads (i.e., the central load values displayed in Fig. 5 for the forecasting models #1 to #8, respectively). For these eight columns, this table also relates whether the system is being over-provisioned (↑), under-provisioned (↓), or correctly provisioned (=).

Regarding the number of provisioned resources in Table 4, we can see that, in most cases, the estimated allocation value based on SVM forecasting models (columns #4 to #8) is closer to the optimal value than the simple forecasting models (columns #1 to #3). This fact is most evident in Fig. 7, which shows the total number of over-provisioned and under-provisioned resources over the 24 h test interval.

If we look at the total number of over-provisioned resources, we can see that the SVM-based models (#4 to #8) outperform the simple forecasting methods (#1 to #3); forecasting model #6 (SVM - RBF Kernel, $\lambda = 0.2$) is

Table 4 Allocation 1: Optimal resource allocation (based on real load) vs. estimated resource allocation (based on predicted loads, forecasting models #1 to #8)

Time	Optimal	#1	#2	#3	#4	#5	#6	#7	#8
00:00	37	38(↑)	42(↑)	40(↑)	39(↑)	37(=)	36(↓)	36(↓)	38(↑)
01:00	36	37(↑)	40(↑)	41(↑)	39(↑)	38(↑)	34(↓)	36(=)	35(↓)
02:00	29	36(↑)	37(↑)	37(↑)	34(↑)	36(↑)	35(↑)	33(↑)	34(↑)
03:00	38	29(↓)	34(↓)	37(↓)	36(↓)	37(↓)	37(↓)	40(↑)	39(↑)
04:00	44	38(↓)	34(↓)	41(↓)	42(↓)	41(↓)	42(↓)	43(↓)	42(↓)
05:00	35	44(↑)	37(↑)	44(↑)	44(↑)	44(↑)	41(↑)	42(↑)	43(↑)
06:00	49	35(↓)	40(↓)	41(↓)	39(↓)	41(↓)	40(↓)	42(↓)	42(↓)
07:00	46	49(↑)	42(↓)	49(↑)	48(↑)	51(↑)	49(↑)	48(↑)	48(↑)
08:00	58	46(↓)	43(↓)	49(↓)	48(↓)	51(↓)	49(↓)	47(↓)	50(↓)
09:00	64	58(↓)	51(↓)	62(↓)	62(↓)	62(↓)	65(↑)	66(↑)	66(↑)
10:00	71	64(↓)	56(↓)	67(↓)	65(↓)	67(↓)	67(↓)	65(↓)	66(↓)
11:00	61	71(↑)	65(↑)	71(↑)	71(↑)	69(↑)	69(↑)	69(↑)	70(↑)
12:00	67	61(↓)	66(↓)	64(↓)	64(↓)	62(↓)	61(↓)	63(↓)	63(↓)
13:00	73	67(↓)	66(↓)	63(↓)	61(↓)	64(↓)	65(↓)	67(↓)	66(↓)
14:00	61	73(↑)	67(↑)	66(↑)	65(↑)	64(↑)	67(↑)	67(↑)	67(↑)
15:00	61	61(=)	67(↑)	66(↑)	65(↑)	62(↑)	63(↑)	66(↑)	64(↑)
16:00	63	61(↓)	65(↑)	65(↑)	64(↑)	60(↓)	58(↓)	55(↓)	57(↓)
17:00	43	63(↑)	62(↑)	60(↑)	59(↑)	57(↑)	58(↑)	57(↑)	56(↑)
18:00	56	43(↓)	56(=)	53(↓)	50(↓)	50(↓)	50(↓)	51(↓)	51(↓)
19:00	51	56(↑)	54(↑)	52(↑)	52(↑)	52(↑)	49(↓)	45(↓)	46(↓)
20:00	52	51(↓)	50(↓)	43(↓)	42(↓)	47(↓)	44(↓)	44(↓)	45(↓)
21:00	43	52(↑)	53(↑)	48(↑)	46(↑)	49(↑)	46(↑)	49(↑)	49(↑)
22:00	42	43(↑)	48(↑)	46(↑)	45(↑)	46(↑)	40(↓)	42(=)	42(=)
23:00	41	42(↑)	45(↑)	32(↓)	32(↓)	38(↓)	38(↓)	39(↓)	41(=)

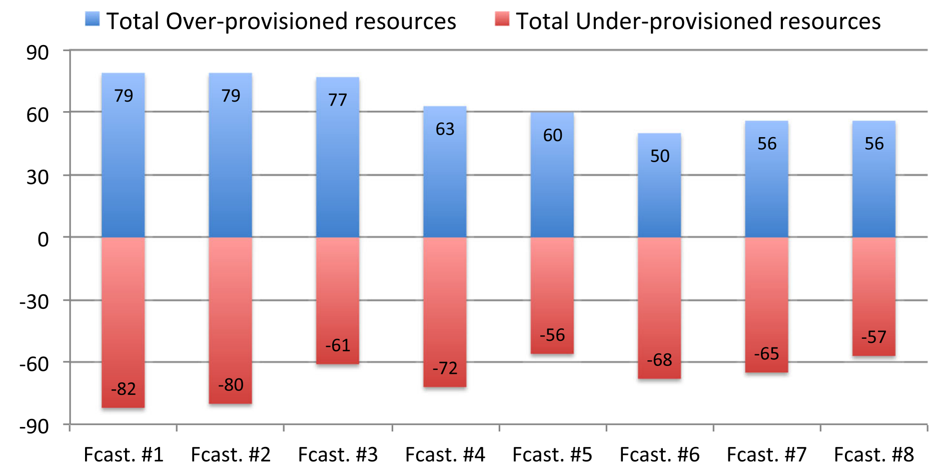

Fig. 7 Over- and under-provisioning of resources using different forecasting models, based on the resource allocation of Table 4 This figure shows the total number of over-provisioned and under-provisioned resources over the 24 h test interval for the different forecasting models (#1 to #8), based on the resource allocation of Table 4. This is computed as the difference between the number of resources that should be provisioned according to the predicted load of the different forecasting models, and the optimal number of resources that should be provisioned according to the real load of the server for the 24 h test interval

the best case. If we look at the total number of under-provisioned resources, the best forecasting methods are, once again, the SVM-based forecasting models #5 (SVM - Normal. Polynomial Kernel) and #8 (SVM - RBF Kernel, $\lambda = 1.0$).

Figures 8 and 9 show, respectively, the percentage of SLA violations and unserved requests on each hourly period for forecasting models #3, #5, and #8 (in order to avoid a mesh of points in the graph, we have chosen these three cases in representation of the basic models, the polynomial SVM-based models, and the RBF SVM-based models, respectively). In addition, Table 5 shows the total number of SLA violations and unserved requests over the 24 h test interval, expressed as a percentage with respect the total number of requests.

Regarding the SLA violations results, we can see that all the forecasting models produce a high number of SLA violations: between 40% and 50% of the total number of requests, as shown in Table 5. This is because all the forecasting models cause under-provisioning of resources in several hourly periods (about half of the periods in most cases). This under-provisioning results in a high number of SLA violations, which can reach, in some cases, almost the 100% of requests, as shown in Fig. 8. If we

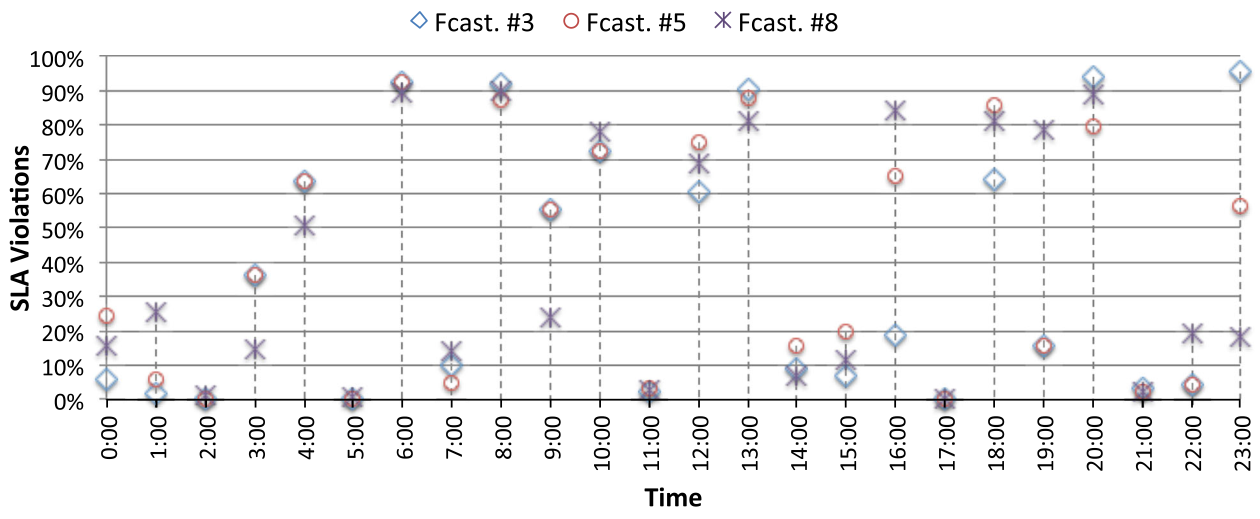

Fig. 8 Percentage of SLA violations for forecasting models #3, #5, and #8, based on the resource allocation of Table 4 This figure shows the percentage of SLA violations on each hourly period for forecasting models #3, #5, and #8 (in order to avoid a mesh of points in the graph, we have chosen these three cases in representation of the basic models, the polynomial SVM-based models, and the RBF SVM-based models, respectively)

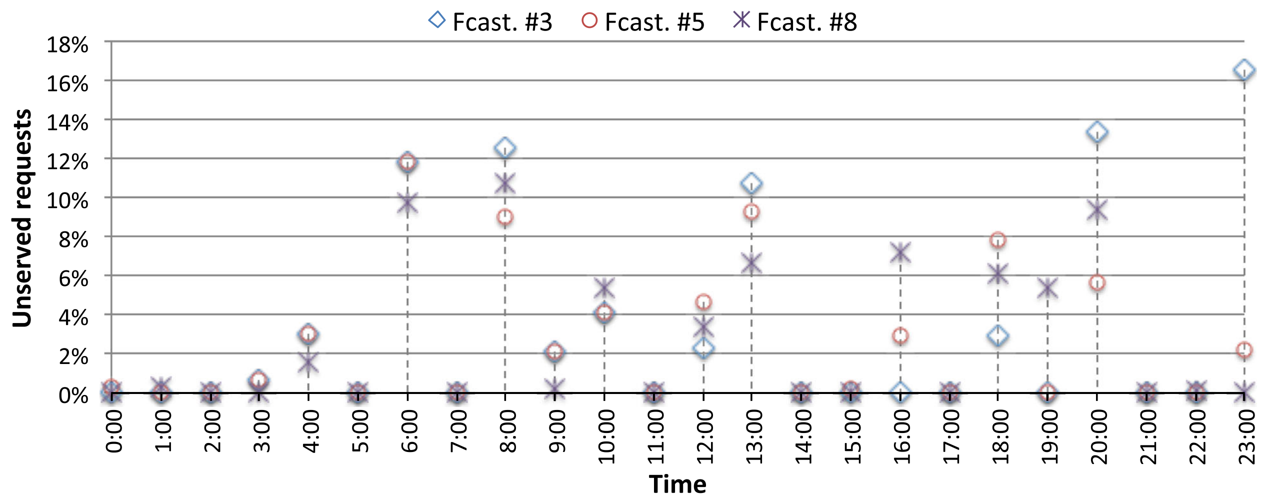

Fig. 9 Percentage of unserved requests for forecasting models #3, #5, and #8, based on the resource allocation of Table 4 This figure shows the percentage of unserved requests on each hourly period for forecasting models #3, #5, and #8 (in order to avoid a mesh of points in the graph, we have chosen these three cases in representation of the basic models, the polynomial SVM-based models, and the RBF SVM-based models, respectively)

compare the percentage of total SLA violations of the different forecasting models in Table 5, we can see that two of the basic forecasting models (specifically, models #2 and #3) behave slightly better than the SVM-based models. This is because the number of hourly periods with a high shortage of resources for these two forecasting models is lower than for other models.

On the other hand, regarding the unserved requests results in Fig. 9, we can see that in most hourly periods the percentage of unserved requests is negligible, and only in a few periods this percentage exceeds 10%. Regarding the percentage of total unserved requests in Table 5, we can see that most SVM-based models outperforms the basic models, being forecasting models #5 and #8 those that present the best behavior.

Table 5 Percentage of SLA violations and unserved requests over total number of requests for the 24 h test interval, based on the resource allocation of Table 4

Resource allocation	SLA Violations (%)	Unserved Requests (%)
Optimal	0.4%	0.4%
Fcast. #1	46.0%	5.4%
Fcast. #2	41.9%	5.4%
Fcast. #3	41.1%	3.6%
Fcast. #4	44.3%	4.5%
Fcast. #5	44.5%	3.1%
Fcast. #6	47.9%	3.7%
Fcast. #7	45.3%	3.8%
Fcast. #8	44.1%	3.2%

We can conclude that the models that perform better are those that minimize the number of under-provisioning periods and under-provisioned resources (so reducing the number of SLA violations, and unserved requests), but at the same time they do not exceed too much the number of over-provisioned resources (so avoiding a significant infrastructure cost increasing). Therefore, the SVM-based forecasting models #5 and #8 exhibit the best trade-off for the three considered metrics (number of resources, number of SLA violations, and number of unserved requests). However, it is important to notice that them basic model #3 also present a good trade-off of the three metrics and a better behavior regarding SLA violations.

In order to reduce the number of SLA violations, we achieved a second resource allocation based on the predicted load values displayed on Fig. 5, but instead of using the central load values of the graphs, we used the central load values plus half the expected error (represented by the error bars in Fig. 5). Table 6 shows the hourly results of this new provisioning, and Fig. 10 shows the total number of over-provisioned and under-provisioned resources over the 24 h test interval.

Regarding the number of over-provisioned resources, we can see that SVM-based forecasting models outperforms the basic models, being forecasting models based on RBF Kernel (models #6, #7, and #8) the ones that behave better. However, these models do not offer the best results regarding the number of under-provisioned resources, in fact, models #7 and #8 are the worst cases, while forecasting models #3 and #5 are the ones with less number of under-provisioned resources.

If we analyze the number of SLA violations and unserved requests for this new resource allocation, as

Table 6 Allocation 2: Optimal resource allocation (based on real load) vs. estimated resource allocation (based on predicted loads plus half the expected error, forecasting models #1 to #8)

Time	Optimal	#1	#2	#3	#4	#5	#6	#7	#8
00:00	37	43(↑)	47(↑)	44(↑)	43(↑)	40(↑)	38(↑)	38(↑)	39(↑)
01:00	36	42(↑)	45(↑)	45(↑)	43(↑)	41(↑)	36(=)	37(↑)	37(↑)
02:00	29	41(↑)	42(↑)	41(↑)	38(↑)	40(↑)	37(↑)	35(↑)	35(↑)
03:00	38	39(↑)	39(↑)	41(↑)	40(↑)	40(↑)	40(↑)	41(↑)	41(↑)
04:00	44	43(↓)	39(↓)	45(↑)	46(↑)	45(↑)	45(↑)	44(=)	43(↓)
05:00	35	49(↑)	42(↑)	48(↑)	48(↑)	47(↑)	44(↑)	44(↑)	44(↑)
06:00	49	40(↓)	45(↓)	45(↓)	43(↓)	44(↓)	43(↓)	44(↓)	44(↓)
07:00	46	54(↑)	47(↑)	53(↑)	52(↑)	54(↑)	51(↑)	49(↑)	50(↑)
08:00	58	51(↓)	48(↓)	53(↓)	52(↓)	54(↓)	52(↓)	49(↓)	51(↓)
09:00	64	63(↓)	56(↓)	66(↑)	66(↑)	65(↑)	67(↑)	67(↑)	67(↑)
10:00	71	69(↓)	61(↓)	71(=)	69(↓)	70(↓)	70(↓)	67(↓)	67(↓)
11:00	61	76(↑)	70(↑)	75(↑)	75(↑)	73(↑)	71(↑)	71(↑)	71(↑)
12:00	67	66(↓)	71(↑)	68(↑)	68(↑)	66(↓)	64(↓)	65(↓)	65(↓)
13:00	73	72(↓)	71(↓)	67(↓)	65(↓)	67(↓)	68(↓)	68(↓)	68(↓)
14:00	61	78(↑)	72(↑)	70(↑)	69(↑)	67(↑)	69(↑)	68(↑)	69(↑)
15:00	61	66(↑)	72(↑)	70(↑)	69(↑)	66(↑)	66(↑)	67(↑)	65(↑)
16:00	63	65(↑)	70(↑)	68(↑)	68(↑)	64(↑)	61(↓)	57(↓)	58(↓)
17:00	43	68(↑)	67(↑)	64(↑)	63(↑)	60(↑)	61(↑)	59(↑)	58(↑)
18:00	56	48(↓)	61(↑)	57(↑)	54(↓)	54(↓)	53(↓)	53(↓)	52(↓)
19:00	51	61(↑)	59(↑)	56(↑)	56(↑)	56(↑)	51(=)	47(↓)	48(↓)
20:00	52	56(↑)	55(↑)	47(↓)	46(↓)	50(↓)	47(↓)	46(↓)	46(↓)
21:00	43	57(↑)	58(↑)	52(↑)	50(↑)	52(↑)	49(↑)	51(↑)	50(↑)
22:00	42	48(↑)	53(↑)	50(↑)	49(↑)	49(↑)	43(↑)	43(↑)	44(↑)
23:00	41	47(↑)	50(↑)	36(↓)	36(↓)	41(=)	41(=)	41(=)	42(↑)

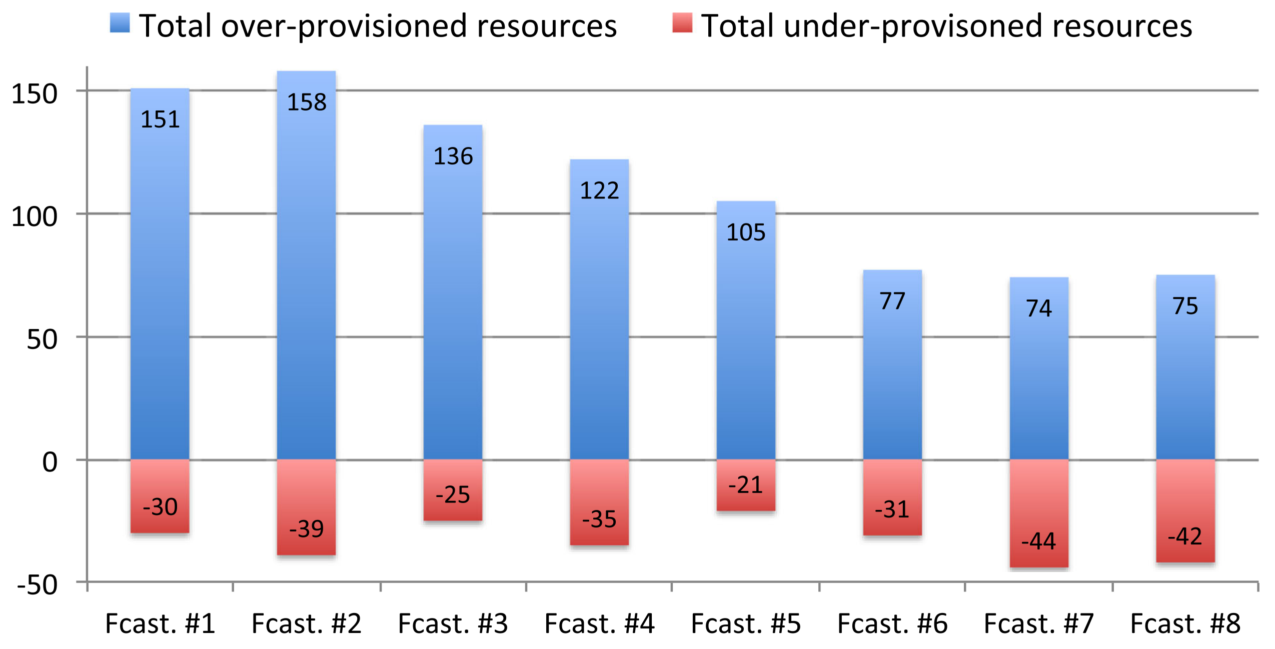

Fig. 10 Over- and under-provisioning of resources using different forecasting models, based on the resource allocation of Table 6 This figure shows the total number of over-provisioned and under-provisioned resources over the 24 h test interval for the different forecasting models (#1 to #8), based on the resource allocation of Table 6. This is computed as the difference between the number of resources that should be provisioned according to the predicted load of the different forecasting models, and the optimal number of resources that should be provisioned according to the real load of the server for the 24 h test interval

shown in Table 7, we can see that the percentage of total SLA violations in all the cases is considerably lower than in the first resource allocation: between 8% for the best case (forecasting model #3) and about 34% for the two worst cases (forecasting models #7 and #8). Similarly, unserved requests have been also significantly reduced with regard to the first resource allocation, being forecasting models #3 and #5 the two best cases, with a negligible number of unserved requests.

In conclusion, we can assert that, in general, SVM-based forecasting models outperform basic forecasting models regarding the number of over-provisioned resources. However, regarding the number of SLA violations and unserved requests, some of the SVM-based models have worse results than basic models. According to the results of the second allocation, based on the predicted load values plus half the expected error, the model that offers the best tradeoff of the three considered metrics (number of resources, number of SLA violations, and number of unserved requests) is the forecasting model #5 (SVM - normalized polynomial model).

Finally, it is important to remark that the execution of the proposed auto-scaling mechanism, including both the server load forecast (using basic or SVM-based models), and the resource estimation, takes only a few seconds (less than one minute in the worst case). Furthermore, using the appropriate techniques, cloud resource instances can also be provisioned or de-provisioned in a matter of seconds [57–59]. Therefore, auto-scaling actions (including startup/shutdown of resource instances) can be done with a minimum delay, typically between 1 and 5 min. To deal with this delay, and taking into account that we use an auto-scaling period of an hour, we can call the auto-scaler a few minutes before the next auto-scaling period, so that the required resources are ready when this period begins.

Table 7 Percentage of SLA violations and unserved requests over total number of requests for the 24 h test interval, based on the resource allocation of Table 6

Resource allocation	SLA Violations (%)	Unserved Requests (%)
Optimal	0.4%	0.4%
Fcast. #1	11.2%	0.4%
Fcast. #2	15.2%	0.7%
Fcast. #3	10.8%	0.1%
Fcast. #4	17.6%	0.5%
Fcast. #5	8.0%	0.1%
Fcast. #6	19.3%	0.5%
Fcast. #7	34.6%	1.5%
Fcast. #8	33.6%	1.3%

Conclusion and future work

In this paper, we have presented an auto-scaling method for adaptive provisioning of elastic cloud services, based on ML time-series forecasting and queuing theory, aimed at optimizing the latency (response time) of the service, and reducing over-provisioning. The auto-scaling system uses a SVM regression to predict the processing load of a web server, based on historical observations. Before applying the SVM regression model, we fine-tuned its parameters, allowing us to capture the nonlinear and temporal patterns of the input data, and achieve an accurate prediction. Using the historical load values of a real web service as input data, we applied the SVM forecasting model using various kernel functions (polynomial kernel, normalized polynomial kernel, and RBF kernel) and different configuration parameters. We compared the accuracy of these predictions with those obtained from other simple methods (last value, MA, and AR models), by computing the MAE and RMSE error measurements for the 24 h test interval. Our results show that the SVM-based regression model has better prediction accuracy than the simple methods.

In addition, the proposed auto-scaling mechanism combines the SVM forecasting method with a M/M/c queue-based performance model. This allowed us to estimate the appropriate number of resources that must be provisioned, according to the predicted load, in order to reduce the service time, and fulfill the SLA contracted by the user. The experimental results also show that, in general, resource allocations based on SVM forecasting are closer to the optimal allocation (based on real load observations) than those based on simple forecasting methods. In particular, SVM forecasting models based on normalized polynomial kernels give the best allocation results with regard to the number of over-provisioned resources, the number of SLA violations, and the number of unserved requests.

As future work, we plan to extend both the forecasting and performance models to other distributed services, such as big data clusters (e.g., Hadoop or Spark clusters), with the goal of implementing efficient auto-scaling mechanisms for these architectures. The forecasting and performance models should be adapted to the particularities and functionality of the different components in these kinds of clusters, such as Mapreduce, HDFS, YARN, and Spark components.

Abbreviations

ANN: Artificial Neural Network; AR: Autoregressive; ARIMA: Autoregressive Integrated Moving Average; ARMA: Autoregressive Moving Average; ARMAX: Autoregressive Moving Average with eXternal input; HDFS: Hadoop Distributed File System; MA: Moving Average; MAE: Mean Absolute Error; MIMO: Multiple Input, Multiple Output; ML: Machine Learning; MSE: Mean Squared Error; QoS: Quality of Service; RBF: Radial Basis Function; RL: Reinforcement Learning; RMSE: Root Mean Squared Error; SLA: Service Level

Agreement; SLO: Service Level Objective; SVM: Support Vector Machine; YARN: Yet Another Resource Negotiator

Acknowledgements
Not applicable.

Funding
This research was supported by the Comunidad de Madrid (Spain), through research grant P2018/TCS4499. This funding allowed us to acquire the computing resources needed for completing this research.

Availability of data and materials
The input data chosen to train and evaluate the forecasting model proposed in this work were obtained from real web service logs from the Complutense University of Madrid. These input and test data sets are available as supplementary material to this manuscript and also at https://goo.gl/Jez9Kg.

Authors' contributions
RMV conceived the study, carried out its design, conducted the experimental section, and drafted the manuscript. RSM participated in the definition and implementation of the experimental section, and helped to refine the manuscript. EH participated in the definition and implementation of the queue-based performance model, and also helped to refine the manuscript. IML coordinated the research, participated in the analysis of different SVM-based methods, and helped to draft and refine the manuscript. All authors read and approved the final manuscript.

Authors information
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Publisher's Note

Author details
[1]Computer Science School, Complutense University, 28040 Madrid, Spain.
[2]IACS/SEAS, Harvard University, 02138 MA Cambridge, USA.

Received: 9 December 2018 Accepted: 29 March 2019
Published online: 16 April 2019

References

1. Herbst N, Kounev S, Reussner R (2013) Elasticity in cloud computing: What it is, and what it is not. In: Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13). USENIX, San Jose. pp 23–27
2. Beloglazov A, Abawajy J, Buyya R (2012) Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Futur Gener Comput Syst 28(5):755–768. Special Section: Energy efficiency in large-scale distributed systems
3. Dougherty B, White J, Schmidt D (2012) Model-driven auto-scaling of green cloud computing infrastructure. Futur Gener Comput Syst 28(2):371–378
4. Ye K, Huang D, Jiang X, Chen H, Wu S (2010) Virtual machine based energy-efficient data center architecture for cloud computing: A performance perspective. In: Green Computing and Communications (GreenCom) 2010 IEEE/ACM Int'l Conference on Int'l Conference on Cyber, Physical and Social Computing (CPSCom). pp 171–178
5. Papadopoulos A, Ali-Eldin A, Årzén K, Tordsson J, Elmroth E (2016) Peas: A performance evaluation framework for auto-scaling strategies in cloud applications. ACM Trans Model Perform Eval Comput Syst 1(4):15:1–15:31
6. Lorido-Botran T, Miguel-Alonso J, Lozano JA (2014) A review of auto-scaling techniques for elastic applications in cloud environments. J Grid Comput 12(4):559–592
7. Jiang J, Lu J, Zhang G, Long G (2013) Optimal cloud resource auto-scaling for web applications. In: 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing. ACM, New York. pp 58–65
8. Messias V, Estrella JC, Ehlers R, Santana MJ, Santana RC, Reiff-Marganiec S (2016) Combining time series prediction models using genetic algorithm to autoscaling web applications hosted in the cloud infrastructure. Neural Computing and Applications 27(8):2383–2406
9. Roy N, Dubey A, Gokhale A (2011) Efficient autoscaling in the cloud using predictive models for workload forecasting. In: Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing, CLOUD '11. IEEE Computer Society, Washington, DC. pp 500–507
10. Verma M, Gangadharan G, Narendra N, Vadlamani R, Inamdar V, Ramachandran L, Calheiros R, Buyya R (2016) Dynamic resource demand prediction and allocation in multi-tenant service clouds. Concurrency and Computation: Practice and Experience 28(17):4429–4442. CPE-15-0088.R1
11. Box G, Jenkins G (1990) Time Series Analysis, Forecasting and Control. Holden-Day, Incorporated, San Francisco
12. Islam S, Keung J, Lee K, Liu A (2012) Empirical prediction models for adaptive resource provisioning in the cloud. Future Generation Computer Systems 28(1):155–162
13. Jiang Y, Perng C, Li T, Chang R (2012) Self-adaptive cloud capacity planning. In: 2012 IEEE Ninth International Conference on Services Computing. pp 73–80
14. Shevade S, Keerthi S, Bhattacharyya C, Murthy K (2000) Improvements to the smo algorithm for svm regression. IEEE Transactions on Neural Networks 11(5):1188–1193
15. Schölkopf B, Smola A (2002) Learning with Kernels. MIT Press, Cambridge
16. Gross D, Shortle J, Thompson JM, Harris C (2008) Fundamentals of queueing theory. Wiley, Hoboken
17. Moreno-Vozmediano R, Montero R, Llorente I (2011) Elastic management of web server clusters on distributed virtual infrastructures. Concurrency and Computation: Practice and Experience 23(13):1474–1490
18. Qu C, Calheiros R, Buyya R (2016) A reliable and cost-efficient auto-scaling system for web applications using heterogeneous spot instances. Journal of Network and Computer Applications 65:167–180
19. Cocaña-Fernández A, Sánchez L, Ranilla J (2016) Leveraging a predictive model of the workload for intelligent slot allocation schemes in energy-efficient {HPC} clusters. Engineering Applications of Artificial Intelligence 48:95–105
20. Marosi A, Kovács J, Kacsuk P (2013) Towards a volunteer cloud system. Future Generation Computer Systems 29(6):1442–1451
21. Montero R, Moreno-Vozmediano R, Llorente I (2011) An elasticity model for high throughput computing clusters. Journal of Parallel and Distributed Computing 71(6):750–757
22. Rodero-Merino L, Vaquero LM, Gil V, Galán F, Fontán J, Montero RS, Llorente IM (2010) From infrastructure delivery to service management in clouds. Future Generation Computer Systems 26(8):1226–1240
23. San-Aniceto I, Moreno-Vozmediano R, Montero R, Llorente I (2011) Cloud capacity reservation for optimal service deployment. In: Proceedings of the The Second International Conference on Cloud Computing, GRIDs, and Virtualization, Rome. IARIA Conference. pp 52–59
24. Gandhi A, Thota S, Dube P, Kochut A, Zhang L (2016) Autoscaling for hadoop clusters. In: 2016 IEEE International Conference on Cloud Engineering (IC2E). MDPI, Basel. pp 109–118
25. Li Z, Yang C, Liu K, Hu F, Jin B (2016) Automatic scaling hadoop in the cloud for efficient process of big geospatial data. ISPRS International Journal of Geo-Information 5(10):173
26. Smowton C, Balla A, Antoniades D, Miller C, Pallis G, Dikaiakos MD, Xing W (2017) A cost-effective approach to improving performance of big genomic data analyses in clouds. Future Generation Computer Systems 67:368–381
27. Moreno-Vozmediano R, Montero R, Llorente I (2012) Iaas cloud architecture: From virtualized data centers to federated cloud infrastructures. Computer 45(12):65–72
28. Hasan M, Magana E, Clemm A, Tucker L, Gudreddi S (2012) Integrated and autonomic cloud resource scaling. In: 2012 IEEE Network Operations and Management Symposium. pp 1327–1334
29. Chieu T, Mohindra A, Karve A (2011) Scalability and performance of web applications in a compute cloud. In: Proceedings of the 2011 IEEE 8th International Conference on e-Business Engineering, ICEBE '11. IEEE Computer Society, Washington, DC. pp 317–323
30. Lim H, Babu S, Chase J, Parekh S (2009) Automated control in cloud computing: Challenges and opportunities. In: Proceedings of the 1st Workshop on Automated Control for Datacenters and Clouds, ACDC '09. ACM, New York. pp 13–18
31. Padala P, Hou K, Shin K, Zhu X, Uysal M, Wang Z, Singhal S, Merchant A (2009) Automated control of multiple virtualized resources. In:

Proceedings of the 4th ACM European Conference on Computer Systems, EuroSys '09. ACM, New York. pp 13–26
32. Dutreilh X, Kirgizov S, Melekhova O, Malenfant J, Rivierre N, Truck I (2011) Using Reinforcement Learning for Autonomic Resource Allocation in Clouds: towards a fully automated workflow. In: 7th International Conference on Autonomic and Autonomous Systems (ICAS'2011). IARIA, Venice. pp 67–74
33. Tesauro G, Jong NK, Das R, Bennani MN (2006) A Hybrid Reinforcement Learning Approach to Autonomic Resource Allocation. In: Proceedings of the 2006 IEEE International Conference on Autonomic Computing (ICAC). IEEE Computer Society, Washington, DC. pp 65–73
34. Barrett E, Howley E, Duggan J (2013) Applying reinforcement learning towards automating resource allocation and application scalability in the cloud. Concurrency and Computation: Practice and Experience 25(12):1656–1674
35. Salah K, Elbadawi K, Boutaba R (2016) An analytical model for estimating cloud resources of elastic services. Journal of Network and Systems Management 24(2):285–308
36. Kaur PD, Chana I (2014) A resource elasticity framework for qos-aware execution of cloud applications. Future Generation Computer Systems 37:14–25. Special Section: Innovative Methods and Algorithms for Advanced Data-Intensive ComputingSpecial Section: Semantics, Intelligent processing and services for big dataSpecial Section: Advances in Data-Intensive Modelling and SimulationSpecial Section: Hybrid Intelligence for Growing Internet and its Applications
37. Ahmed N, Atiya A, Gayar NE, El-Shishiny H (2010) An empirical comparison of machine learning models for time series forecasting. Econometric Reviews 29(5-6):594–621
38. Bontempi G, Taieb S, Borgne YL (2013) Business Intelligence: Second European Summer School, eBISS 2012, Brussels, Belgium, July 15-21, 2012,. Springer Berlin Heidelberg, Berlin
39. Allende H, Moraga C, Salas R (2002) Artificial neural networks in time series forecasting: a comparative analysis. Kybernetika 38(6):685–707
40. Zhang G (2003) Time series forecasting using a hybrid {ARIMA} and neural network model. Neurocomputing 50:159–175
41. Thissen U, van Brakel R, de Weijer A, Melssen W, Buydens L (2003) Using support vector machines for time series prediction. Chemometrics and Intelligent Laboratory Systems 69(1–2):35–49
42. Müller K, Smola A, Rätsch G, Schölkopf B, Kohlmorgen J, Vapnik V (1997) Artificial Neural Networks – ICANN'97: 7th International Conference Lausanne, Switzerland, October 8–10, 1997. In: Proceedings chap. Predicting time series with support vector machines. Springer Berlin Heidelberg. pp 999–1004
43. Burges C (1998) A tutorial on support vector machines for pattern recognition. Data Mining and Knowledge Discovery 2(2):121–167
44. Smola A, Schölkopf B (2004) A tutorial on support vector regression. Statistics and Computing 14:199–222
45. Suykens J, Vandewalle J (1999) Least squares support vector machine classifiers. Neural Processing Letters 9(3):293–300
46. Vapnik V (1998) Statistical Learning Theory. Wiley-Interscience, New York
47. Cherkassky V, Ma Y (2004) Practical selection of svm parameters and noise estimation for svm regression. Neural Netw. 17(1):113–126
48. Shawe-Taylor J, Cristianini N (2004) Kernel Methods for Pattern Analysis. Cambridge University Press, New York
49. Üstün B, Melssen W, Buydens L (2006) Facilitating the application of support vector regression by using a universal pearson {VII} function based kernel. Chemometrics and Intelligent Laboratory Systems 81(1):29–40
50. Rauber T, Berns K (2011) Kernel multilayer perceptron. IEEE Computer Society, Washington
51. Boardman M, Trappenberg T (2006) A heuristic for free parameter optimization with support vector machines. In: The 2006 IEEE International Joint Conference on Neural Network Proceedings. pp 610–617
52. Cassabaum M, Waagen D, Rodriguez J, Schmitt H (2004) Unsupervised optimization of support vector machine parameters. In: Proceedings of SPIE 5426, Automatic Target Recognition XIV. SPIE, Bellingham. pp 316–325
53. Chapelle O, Vapnik V, Bousquet O, Mukherjee S (2002) Choosing multiple parameters for support vector machines. Machine Learning 46(1):131–159
54. Frank E, Hall M, Holmes G, Kirkby R, Pfahringer B (2005) WEKA–A machine learning workbench for data mining. Springer, Boston
55. Iqbal W, Dailey M, Carrera D, Janecek P (2011) Adaptive resource provisioning for read intensive multi-tier applications in the cloud. Future Gener. Comput. Syst. 27(6):871–879
56. Kujawa L (2013) Performance benchmark of popular php frameworks. https://systemsarchitectdotnet.wordpress.com/2013/04/23/performance-benchmark-of-popular-php-frameworks/. Posted on April 23
57. Razavi K, Kolk GVD, Kielmann T (2015) Prebaked μVMs: scalable, instant VM startup for IaaS clouds. In: Proceedings of the 35th IEEE International Conference on Distributed Computing Systems. pp 245–255
58. Razavi K, Costache S, Gardiman A, Verstoep K, Kielmann T (2015) Scaling VM deployment in an open source cloud stack. In: Proceedings of the 6th Workshop on Scientific Cloud Computing
59. Manco F, Lupu C, Schmidt F, Mendes J, Kuenzer S, Sati S, Yasukata K, Raiciu C, Huici F (2017) My VM is lighter (and safer) than your container. In: Proceedings of the 26th Symposium on Operating Systems Principles. pp 218–233